Design and Implementation of a Media Access Component at Picsearch Using a Rigorous Software Engineering Approach

Diego Núñez Silva

This thesis is presented as part of Degree of European Master in Software Engineering

School of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona
Sweden
This thesis is submitted to the School of Computing at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Software Engineering. The thesis is equivalent to 30 weeks of full time studies.

Contact Information:
Author:
Diego Núñez Silva 801207-T059
E-mail BTH: dina09@student.bth.se
E-mail UPM: dmnez@alumnos.fi.upm.es

University advisor(s):
Prof. Lars - Åke Fredlund
Universidad Politécnica de Madrid

Prof. Richard Torkar
Blekinge Institute of Technology

Industrial Collaboration:
Picsearch Services AB
Liljeholmsvägen 30
SE-117 61 Stockholm, Sweden

School of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona
Sweden

Internet : www.bth.se/com
Phone : +46 455 38 50 00
Fax : +46 455 38 50 57
Abstract

Context. With the arrival of a new generation of sophisticated smartphones, possibilities for mobile video usage are presenting exciting new opportunities. This Master’s thesis is based on a collaboration with Picsearch to design and implement a software component that enables the company’s video services in Android smartphones.

Objectives. Utilize a SE approach to perform a requirements analysis, architecture design and implementation of a software component that enables rapid development of Android applications that are built upon Picsearch video services.

Methods. Three methods support this work: 1) a case study that involved several interviews with Picsearch’s personnel, and a study of their video platform’s documentation; 2) a systematic literature review (SLR) that reflects the latest SE literature about architecture design methods; 3) prototyping as means to perform the component’s progressive evaluation-improvement process.

Results. The concepts applied in the SE approach utilized in this collaboration made possible to satisfy the expectations of Picsearch for this Master’s thesis, and provided three deliveries: 1) a media software component, 2) its API’s documentation, and 3) a proof of concept application that demonstrates the component’s capabilities.

Conclusions. The media software component delivered with this Master’s thesis is result of applying a rigorous SE approach that started with a RE phase, required a SLR, software architecture design, implementation, and ended up with a quality evaluation. This approach proved to be effective to achieve the goal of this thesis, as well as the satisfaction of Picsearch.

Keywords: architecture design, media component, market analysis, Picsearch.
Acknowledgments

A very special acknowledgement to my supervisor Dr. Lars-Ake Fredlund for his help, valuable feedback and patience until the completion of this work. Tusen tack!

To Dr. Richard Torkar for backing up the supervision of this thesis.

To Picsearch’s personnel, and in special to Sean Atkinson, Robert Risberg, Johan Rönnblom and Magnus Ahlberg for the time and support dedicated to this work.

To the Erasmus Mundus organization that made my dream come true when allowed me to study this Master’s program. Many thanks to Nella Aveledo, Oscar Dieste, Natalia Juristo, Richard Torkar, Tony Gorschek for their effort, great job and continuous improvement of the EMSE program.

Mi más especial agradecimiento a mis padres por sus enseñanzas y apoyo durante todos estos años. La mayor parte de mis logros se los debo a ustedes, y siempre estaré en deuda. Los quiero mucho, y de nuevo, muchas gracias!

A Lore, Ricardo, FG, y toda mi familia por el cariño y buenos deseos que siempre he recibido. Comparto con ustedes este logro.

Un agradecimiento muy especial a Rafael Rojano y William Uzuriaga por su amistad y ayuda en mi proceso de adaptación a Suecia. Muchas gracias!

Another very special acknowledgement to Johannes Pelto-Piri for his friendship and help at explaining the working culture in Sweden. Tack för hjälpen!
To my classmates: Miao, Nandana, Slavisa, Diego, Ahmad, José for their friendship and teachings during this experience.

Finally, my gratitude to all professors, colleagues, classmates and friends that have influenced me to become a better professional and the most important, a better person. My sincere appreciation to this immense list of people that luckily grows every day.
# List of Figures

1.1 Component Overview ........................................... 2
2.1 Planning Process for Market Research \[55\] .................... 7
2.2 Video Streaming Scenario - Use Cases ........................ 13
2.3 Browse Video Categories ....................................... 14
2.4 Search Videos .................................................. 14
2.5 Play Video ..................................................... 15
2.6 Rate Video ..................................................... 15
2.7 Dynamic Video Uploading to Multiple Accounts - Use Cases ... 16
2.8 Select Video Account ........................................... 16
2.9 Upload Video ................................................... 17
2.10 Publish Video .................................................. 17
3.1 SLR Execution Process ......................................... 25
3.2 Architecture Design Activities[39, 38] ........................ 34
4.1 Initial Functional Conceptualization ............................. 47
4.2 Component’s Internal Entities - Conceptual View 1 ............ 49
4.3 Component’s Internal Entities - Conceptual View 2 ............ 50
4.4 Interaction View - Applying Synchronous Processing .......... 51
4.5 Interaction View - Applying Asynchronous Processing ........ 54
4.6 Component Internal View ....................................... 57
4.7 Component Internal View Refined ............................... 64
4.8 API Class Diagram ............................................. 67
4.9 Applications Developed as Post-deployment Evaluation ....... 73
5.1 Usefulness ...................................................... 77
5.2 Understandability ............................................... 79
5.3 Consistency ..................................................... 80
6.1 Our Software Engineering Approach and its Phases ............ 84
6.2 Time Distribution - Time Table ................................ 85
6.3 Time Distribution - Gantt Chart ................................ 85
6.4 Proof of concept application ................................... 93
List of Tables

3.1 Search terms construction process[69] .......................... 22
3.2 Search terms ..................................................... 22
3.3 Literature Databases ............................................. 23
3.4 Detailed InclusionCriteria ....................................... 23
3.5 Detailed Exclusion Criteria ..................................... 24
3.6 Quality Criteria ................................................... 24
3.7 Results Found per Database .................................... 25
4.1 Significant Quality Attributes ................................. 41
4.2 Tactics ............................................................. 43
6.1 Component’s Metrics ............................................ 92
A.1 Data Extraction Form ........................................... 95
A.2 Articles Selected 1 ................................................. 96
A.3 Articles Selected 2 ............................................... 97
A.4 Customized Queries According to the Selected Databases . 98
A.5 Customized Queries According to the Selected Databases (Cont.) 99
B.1 MediaFieldsEnum ............................................... 125
# Contents

Abstract i

Acknowledgments ii

1 Introduction 1
1.1 Background 1
1.1.1 About Picsearch 1
1.1.2 Problem Definition 1
1.2 Aims and Objectives 3
1.3 Research Questions 3
1.4 Outcomes 4
1.5 Research Methodology 4
1.6 Structure 5

2 Market Analysis 6
2.1 Execution 6
2.1.1 STAGE 1 - Identify and Articulate the Decision Problem 7
2.1.2 STAGE 2 - Identify Key Questions 8
2.1.3 STAGE 3 - Identify Research Techniques 8
2.1.4 STAGE 4 - Design and Results of the Market Analysis 10

3 A Systematic Review on Architecture Design Methods 20
3.1 Introduction 20
3.1.1 Research Questions 21
3.2 Planning 21
3.2.1 Search Terms 21
3.2.2 Study Selection and Quality Criteria 23
3.2.3 Quality Assessment 24
3.2.4 Data Extraction 24
3.3 Execution and Results 25
3.4 Analysis 26
3.4.1 Available Software Architecture Design Methods 26
3.4.2 Principles and Core Elements 26
3.5 Discussion 33
3.5.1 Method Selection .......................... 35
3.5.2 SLR Strengths and Weaknesses ............. 36

4 Architecture Design .......................... 37
4.1 Software Architecture Sketch Design .......... 37
4.1.1 Identify Software Requirements .......... 37
4.1.2 Identify & Select Architecture Tactics ....... 41
4.1.3 Create Software Architecture Sketch ....... 46
4.2 Software Architecture Early Evaluation ....... 55
4.3 Software Architecture Refinement ............. 56
4.3.1 Interfaces Definition ..................... 57
4.4 Software Architecture Middle Evaluation ....... 65
4.5 Architecture Transformation Into Implementation ...... 65
4.5.1 Introducing Open Source Software(OSS) .... 65
4.5.2 Component Implementation Details ......... 66
4.5.3 Thread Safety in Multithreading Environments .... 71
4.6 Software Architecture Post-deployment Evaluation ...... 72

5 Evaluation .................................. 74
5.1 Questionnaire .............................. 75
5.2 Results ................................... 77
5.2.1 Usefulness, Understandability & Consistency: .... 77
5.2.2 Reusability .............................. 80
5.2.3 Modifiability ............................ 80
5.2.4 Security ................................ 80
5.2.5 Documentation ......................... 81
5.2.6 Other Comments & Suggestions ............ 81
5.2.7 Contribution of our Work to Picsearch ....... 82
5.3 Future of the API ............................ 82

6 Thesis Results & Conclusions .................. 83
6.1 Market Analysis ............................ 85
6.1.1 Results ................................ 86
6.2 Systematic Literature Review .................. 88
6.2.1 Results ................................ 88
6.3 Architecture Design & Implementation ......... 90
6.3.1 Results ................................ 90
6.4 Evaluation ................................ 92
6.5 Overall Impressions ......................... 94

A Tables SLR ................................ 95
A.1 Data Extraction Form ......................... 95
A.2 Articles Selected ............................ 95
A.3 Queries .................................................. 98

B API Documentation ........................................ 100
B.1 Background ............................................. 100
B.2 Specifications .......................................... 100
  B.2.1 Android Version - API Level ...................... 100
  B.2.2 Thread Safety & Programming Model .............. 100
B.3 API Methods ........................................... 105
  B.3.1 assignRating ........................................ 105
  B.3.2 getMediaContent .................................. 106
  B.3.3 getUploadURL ...................................... 107
  B.3.4 getUserProfiles ................................... 108
  B.3.5 listAllCategories .................................. 110
  B.3.6 listAllMedia ....................................... 111
  B.3.7 listAllTags ......................................... 112
  B.3.8 listMedia ........................................... 113
  B.3.9 listFilteredMedia .................................. 115
  B.3.10 searchMedia ....................................... 117
  B.3.11 searchMedia ....................................... 119
  B.3.12 setModeration ..................................... 121
  B.3.13 uploadMedia ...................................... 123
B.4 MediaFieldsEnum List ................................. 125
Chapter 1

Introduction

1.1 Background

1.1.1 About Picsearch
Picsearch is a Swedish company that develops and provides image search services for large websites. The services developed and provided by Picsearch power several large internet companies, such as Ask and Lycos. Picsearch also develops and provides a video streaming service under the brand name Screen9, which is used for video communities, vlogging (video blog), video reviews on e-commerce sites, corporate video presentations, news videos and virtual showings on real estate portals. Picsearch video streaming service includes a flash player, upload, transcoding, hosting and streaming.

1.1.2 Problem Definition
On-line video usage is continuing to rise as user expectations for rich media content grow. With the arrival of a new generation of sophisticated smartphones, possibilities for mobile video usage are presenting exciting new opportunities. As part of a market strategy Picsearch decided to expand its video services by making them available in Android smartphones. Part of this expansion includes the creation of a software artifact capable to support rapid development of mobile video applications. This Master’s thesis is based on our collaboration with Picsearch. Its main goal is the design and implementation of a software component that enables the company’s video services in Android smartphones.

Picsearch’s intended goal for this thesis was rapid development. Thus our main challenge was related to the need of converting the idea of rapid development into a functional piece of software in a period of 30 weeks. Purposefully Picsearch defined a high level goal with no concrete requirements in order to invite us to get involved into the company’s atmosphere. Consequently they wanted us to define our own requirements according to our understanding of the company and
its market demands. Hence, this work required a period of seven months for completion, in which five of them elapsed with our presence at Picsearch’s offices by means of a daily attendance. Only by experienced this close involvement we could ground Picsearch’s goal to create the software component that is delivered along with this thesis.

![Figure 1.1: Component Overview](image)

This Master’s thesis required us to tackle several challenges that required the application of diverse software engineering (SE) concepts. Therefore, to attain our final outcome we did apply rigorously an entire SE approach. Our approach included a market analysis where requirements engineering techniques were used in order to: 1) learn about the company, its market demands and competitors, and 2) identify and elicit the right functionality that needed to be delivered along with our work. Further, to cope with the architecture design and implementation issues we took support from the Gradually Proceeded Software Architecture Design Process (GADesign), an architecture design method found in the SE literature. GADesign was selected after performing a systematic literature review (SLR) that revealed the latest scientific research about architecture design. One of our main challenges was to experience the method and its results by applying it rigorously in a case study that involved not only software design but also implementation. Finally, we assessed the opinion of the personnel at Picsearch about the quality of our work, and the extent in which the results satisfied their expectations for this Master’s thesis.

Clearly, the main aim of this thesis was to take an industrial need and solve it by applying diverse SE techniques acquired through the course of our Master studies. By means of this thesis we contributed simultaneously to academia and industry in different aspects. Our contribution to academia shaped into 1) a SLR that collects the latest scientific literature related to architecture design methods, and 2) a practical experience about the usefulness of the selected design method in our case study. On the other hand, Picsearch benefited from this collaboration by obtaining a solution that satisfied to a high extent the company’s expectations.
1.2 Aims and Objectives

Aim: Design and implement a software component following a SE architecture design method to enable rapid development of Android applications that are built upon Picsearch video services.

- **Goal 1:** Conduct a market analysis in order to:
  - Understand the market demands behind our software component.
  - Select the right functionality to deliver in the software component.

- **Goal 2:** Conduct a systematic literature review in order to:
  - Identify SE methods specialized on providing guidance for designing a software architecture.
  - Select a method that provides a comprehensive guidance, and that is contextually compatible with the characteristics of our case study.

- **Goal 3:** Use the selected SE method to design and implement a software component capable to offer rapid development of Android applications that rely on Picsearch video services.

- **Goal 4:** Evaluate Picsearch’s personnel opinion about the extent in which our component fulfills its functional requirements and main quality attributes.

1.3 Research Questions

- **RQ1:** What are the current state-of-the-art methods that provide guidance for designing a software architecture?
  - **RQ1.1:** What are the core elements of each method?
  - **RQ1.2:** What are the steps needed to execute the method?

- **RQ2:** Which method presents a higher suitability for supporting our practical architecture design?

- **RQ3:** To what extent did the rigorous SE approach succeed in developing a quality component?
  - **RQ3.1:** To what extent did the market study provide a useful contribution to the entire SE approach adapted in this work?
  - **RQ3.2:** How did the use of GADesign enhance the architecture design process?
Chapter 1. Introduction

1.4 Outcomes

- An analysis that explains the current market demands to justify the design and implementation of the software component delivered as part of this thesis.

- A systematic literature review (SLR) that includes the software architecture design methods available in the SE literature.

- An empirical evaluation of GADesign based on our experience designing and implementing our software component.

- A fully functional implementation of our software component.

- An Android application built as proof-of-concept to demonstrate the capabilities of the software component.

- A developer’s guide that explains the component’s usage and main design decisions.

- An evaluation performed with personnel at Picsearch about the ability of the component to satisfy its functional requirements and most significant quality attributes.

1.5 Research Methodology

- Case study: Our case study involved several interviews with Picsearch’s personnel, as well as a study of their video platform’s documentation. This method was the base to understand the context of Picsearch, and their expectations for this Master’s thesis in terms of the current market of video providers. Further, the understanding achieved by this case study supported us to define the requirements that were finally implemented in the software component delivered along with this Master’s thesis.

- Systematic Literature Review: A SLR was conducted in order to identify those architecture design methods available in the current SE literature. Moreover, this SLR revealed the method that we selected to design our component’s architecture.

- Prototyping: Prototyping was selected as a way to provide continuous feedback about the evolution of the implementation of our software component. The goal was to develop by increments a prototype application that gradually included the features being implemented in the component. By using prototyping in this manner we could experience the usage of our software component, as well as its results in a real Android application.
1.6 Structure

In Chapter 2 we describe our market analysis, its planning phase and the results obtained after executed it. Chapter 3 presents a systematic literature review (SLR) that shows the architecture design methods that we identified in the SE literature. Moreover, Chapter 3 also discusses the selection of GADesign as the method that we utilized in Chapter 4. Chapter 4 explains step by step the application of GADesign as means to design and implement our software component. Chapter 5 summarizes the results of three interviews that we perform to the personnel at Picsearch for evaluating the quality of our component’s design and implementation. Finally, in Chapter 6 we explore our opinion about the usefulness of: 1) the results provided by our market analysis, 2) GADesign as architecture design method, and 3) all the phases of this thesis assembled to form an entire SE approach.
Chapter 2

Market Analysis

The popularity of Android devices is currently on the raise and diverse companies as Picsearch, are aligning their business goals with the market tendencies. According to Garner\cite{28} the Android market share grew 888.8% in 2010 by selling 67 million units, and moved to the 2\textsuperscript{nd} place after Nokia’s Symbian. Moreover, IMSresearch\cite{40} forecasts that 140 million devices including smartphones and tablets will be shipped by the end of 2011. Further, the popularity of Android devices in the last months is derived mainly by two factors: 1) The recent release of Google’s Android 3.0 (Honeycomb) operating system for tablets, and 2) the improvements to the Android Market website\cite{40}, where all android native applications are officially published. These are causing a reassessment in terms of business goals for many companies, \textit{e.g.}, Picsearch. The increasing consumer demand implies that more and more native applications will be developed, and as \cite{70} explains, mobile video is becoming a very desirable feature to include. These facts fundamentally expands the business opportunities for Picsearch, and motivated us to create a software component that can simplify the use of Picsearch’s video services in Android applications.

This chapter describes the execution of our market analysis, and it is organized in four stages. Stage 1 describes the problem addressed by this market analysis. Stage 2 presents the key questions identified in the planning process. Stage 3 describes the research techniques used to cope with this analysis. Finally, Stage 4 explains how these research techniques were utilized to answer our key questions, and presents the results of our analysis.

2.1 Execution

Market research in a generic context is defined as a business activity that collects and examines information about people’s preferences of products that they buy or might buy and their feelings about products that they already bought\cite{17,53}. Moreover, marketing research often involves understanding the customer (\textit{e.g.}, consumers, influencers), the company (\textit{e.g.}, product design, service, pricing), and if needed it can be extended to include competitors\cite{66}. 

6
Chapter 2. Market Analysis

To plan our market study we followed the process described in [55], which is depicted in Figure 2.1. Sections 2.1.1 to 2.1.4 describe the process applied to our context.

![Figure 2.1: Planning Process for Market Research](55)

### 2.1.1 STAGE 1 - Identify and Articulate the Decision Problem

The stage 1 of the planning process aims at defining what is the problem or issue that raises the need of this market research, and who is the decision maker, *i.e.*, the person that will act with the research results. Thus in our context the necessity of a market research is justified for the need of knowing what video-related features are important in the current market of Android applications. As we explained during the introduction of this chapter, Android as a platform has acquired high priority for Picsearch; nevertheless, since Picsearch lacked in any similar software component for either Android, or any other mobile platform, there was also a lack of a customers base that could express their preferences or requirements when using video in mobile applications. Being aware of this limitation we decided to perform a market study to overcome this issue.

The results of our market analysis did reveal potential functional requirements. Because Picsearch delegated the responsibility of decision making to us, we were in position to decide what functionality to include/exclude in our software component. Nevertheless, during the requirements selection process we were careful to consider Picsearch’s opinion and we tried to make them participants of the process. Their feedback became highly valuable due to their experience in the domain.
2.1.2 STAGE 2 - Identify Key Questions

The second stage of the planning process consisted in identifying important questions to answer with this market research. Contextualizing this stage to Picsearch’s scenario, the key questions were:

1. What are the current capabilities of Picsearch’s Online Video Platform API?
   By analyzing the capabilities offered by Picsearch’s API we were able to identify technical and functional boundaries, i.e., what was feasible to implement in our component taking into account that it relies on the capabilities of the video platform.

2. How do potential customers intend to use video in their mobile applications?
   This question was formulated from the business and technical standpoints. We wanted to understand in which scenarios potential customers desire to use mobile video, and what functionalities they might need.

3. What are the strategies taken by Picsearch’s competitors to support the Android platform? We tried to study companies providing similar services in order to understand how they were expanding to Android. By doing this we could compare our contribution with this library and ensure such value was not misdirected.

2.1.3 STAGE 3 - Identify Research Techniques

Stage 3 involved the identification of appropriate research techniques for answering the key questions presented in Stage 2. Our market study was fairly small-scale and certainly the scope was very specific and limited to a single objective: the identification of the component’s significant functional requirements.

In the requirements engineering (RE) literature there are diverse requirements elicitation techniques available such as: interviews, observation, apprenticing, document studies, focus groups, study of similar companies, prototyping, stakeholder analysis, among some others[52, 4]. Each technique has different characteristics, which make it more (or less) suitable for eliciting specific information. For example, in a software engineering (SE) scenario like this, interviews are ideal for eliciting present work, problems and key issues; however, its characteristics make it unsuitable for evaluating realistic possibilities of how a simplified version (prototype) of a system will work in real life. On the other hand, prototyping is perfectly suitable to elicit this realistic possibilities but it is unable to elicit present work, problems and key issues as interviews do. Thus the selection of the correct technique(s) depends on what type of information is to be elicited.
To execute our market analysis we chose interviews, apprenticing and document studies as the most suitable techniques to answer our research questions. These techniques were selected based on the characteristics of this market analysis. As follows we explain each technique and the rationale behind its selection:

**Interviews**

Interviews are an elicitation technique where the requirements engineer enquiries stakeholders about the current state of a software system and its possible faults or improvements. Hence, requirements are elicited from the answers to those questions. Interviews were found suitable to our market analysis because the facilities that Picsearch granted us to enquiry their personnel in relation to their understanding about the company, the market, future market opportunities, and competitors. The personnel’s opinion and knowledge helped to clarify our understanding about this analysis.

**Document Studies**

Document studies are a technique that involves the examination of existing documents such as forms documentation of the current computer system, computer logs or any other document that could provide information. Document studies were selected to cover the examination of Picsearch’s technical documentation about the online video platform. As well, we applied the technique in a more informal manner to review any documentation, website or information that could be related to Picsearch competitors, and video in mobile applications.

**Apprenticing**

Apprenticing consists in involving the analyst to actually learn a particular task under the instruction and supervision of an experienced user. Hence, the analyst learns operations and business processes by observing, asking questions, and practically doing, rather than being informed of them. Apprenticing offers very good results where the analyst is inexperienced with the domain, and when users experience difficulties explaining their actions. Before starting this thesis we were very inexperienced in the domain of video streaming, thus apprenticing was selected to support our domain learning process through the revision and analysis of popular Android video applications.
2.1.4 STAGE 4 - Design and Results of the Market Analysis

As we stated previously the objective of this market analysis is the identification of our component’s significant functional requirements. To achieve our objective we defined a set of research question, which were answered by means of a set of RE techniques. Below we explained the design and results of our study.

What are the current capabilities of Picsearch’s online video platform API?

To answer this question we decided to make use of two techniques: document studies and interviews. As first step we reviewed technical documentation available for the video platform. We wanted to understand the video platform’s application programming interface (API). An API is an interface that describes the functionality, services or methods implemented by a software entity, which can be accessed by developers to perform various tasks [65]. So once we examined the documentation we had a clearer idea of which were the capabilities of the API. Our component relies to a large extent on the services provided by the API, so this step was very important because later on it would influence its design.

For the second step we interviewed: 1) the chief technical officer (CTO) and 2) developers of the video platform. The obvious reason to select these stakeholders was their deep involvement with the video platform’s inner workings and its future improvements. The interviews were designed as semi-structured and made of a combination of open and closed questions regarding the video platform. The closed questions were focused on unclear concepts and doubts derived from our documentation review. The open questions were focused on improvements and ideas for extending the API’s functionality. As well, we questioned them about ideas, requirements or their conceptualization about this thesis. One interesting fact that we observed was that within the company different stakeholders had a different understanding and expectations about this thesis. Thus derived from this set of interviews and informal talks with Picsearch’s personnel, we sat with the CTO, discussed the ambiguities detected during our interviews and discussions, and defined exactly the scope of our work.

Results

The company currently has developed three APIs with different purposes and characteristics: 1) the XML-RPC (eXtensible Markup Language-Remote Procedure Call) API, 2) the AJAX (Asynchronous JavaScript and XML) API, and the REST (Representational State Transfer) API.
Each API uses a different protocol and experiences a different degree of completeness, though all of them are specialized in video management and streaming. From the available APIs the XML-RPC is the most complete and mature. It was the first API developed and therefore it contains the largest number of methods. The disadvantage of this API considering our purposes is that it uses a special IP blocking. Only pre-registered IPs can access the functionality delivered by the XML-RPC API. Considering that applications that utilize our component will be installed in random smartphones, it would be practically impossible to register their IPs in the video platform. So for this reason the XML-RPC was discarded.

The AJAX API on the other hand is an API specially designed to be called from web browsers utilizing JavaScript code. The responses provided by the video platform are web browser-dependent, i.e., embedded HTML and Javascript code is included as part of the response. The focus of our component is to use Android native code to perform the communication with the video platform, then, a web browser is not involved. Consequently, the AJAX API was also discarded.

The REST API is specialized in providing access to the video platform from mobile devices, however it can be accessed from web application as well. The decision for choosing the REST API was: 1) the API does not experience IP blocking restrictions, and 2) it provides web browser-independent responses. Currently, the HTTP REST API offers the functionality listed below.

- **Media uploading**: Allows to upload videos through the network via HTTP and stores them in Picsearch’s server.
- **Media access**: This functionality provides access to videos and their properties. As well, it enables the videos to be streamed over the network.
- **Media listing**: Retrieves a list of video items of interest. This functionality can be tailored by defining the information that needs to be included with the video items that conform the resulted list. Furthermore, the feature supports the definition of filters to reduce the number of results.
- **Media searching**: Enables to perform video searching based on a provided criterion.
- **Media rating**: Adds a rating to a specific video and then calculates the average value considering previous ratings.
- **Category listing**: Offers the possibility to retrieve the diverse video categories associated to a Picsearch account.
How potential customers intend to use video in their mobile applications?

To address this question we made use of three research techniques: interviews and apprenticing. We started interviewing the CEO (chief executive officer) and the sales manager at Picsearch. Being Picsearch a small company both stakeholders are highly knowledgeable about customers’ preferences and market trends.

To the best of our knowledge Picsearch currently lacks in customers that currently has developed any Android video application; therefore, customer interviews were unfeasible. Nevertheless, we took advantage of the expertise and skills of the CEO and the sales manager about understanding the market of video providers and foreseeing potential customers. Our interviews enquired our stakeholders about: 1) potential customers that could use our software component, 2) important features to include in it.

As a next step we applied apprenticing to learn more about the domain of mobile video. We reviewed a set of popular applications as a way of learning through examples about video usage in Android. The applications reviewed were chosen based on our previous knowledge and the information provided by our interviews. Our stakeholders highlighted journals and newspapers as potential users of our component. Therefore, we decided to browse the Android Market to find highly rated applications related to the journalism domain. The goal was to get inspiration from the applications reviewed and convert it into requirements for our component.

Results

After analyzing video usage in various Android video applications, e.g., Youtube, TED Mobile, Fox News, Msnbc, BBC News, CBS News, we observed that most applications make use of very similar video-related features. We compared the features observed in our revision versus the capabilities currently offered by Picsearch’s REST API, and not surprisingly they are very consistent. This is very reasonable if we considered that the REST API was designed to include the most essential video features. The video features observed in our revision helped us to model a first scenario where mobile video is utilized in generic Android applications.

Furthermore, our interviewees expressed that currently Picserch has customers in diverse domains, e.g. real-state, travel agencies, services, media and more. However, when it comes to video in mobile devices they targeted journals as potential customers. At present most of the important events are video recorded, thus journals’ websites are constantly updating their news to reflect sudden
events. Further, the latest smartphones are offering high-quality recording capabilities, situation that opens the door to new possibilities for reporting sudden events. Therefore, our interviewees described an scenario where our software component enables journalists for recording and reporting events directly from their smartphones. This second scenario stands on the functionality represented by the first scenario, thus it extends functionality on top. Both scenarios are explored in the following sections.

**Scenario 1 - Video Streaming**

This scenario (see Figure 2.2) included features implemented in most mobile video applications independently of the domain where they belong to. On it, a user scrolls through a list of videos associated to a Picsearch account in order to select one of interest. Once selected, the streaming starts and the video is played in the smartphone. This scenario as well includes searching capabilities and video rating. One important consideration is that video commenting is not included. When the feature was discussed with Picsearch, they explained that in their experience it is rarely used. Therefore we decided to discard the feature and to focus on the most essential functionality.

![Figure 2.2: Video Streaming Scenario - Use Cases](image)

Before going into detail with this scenario we consider important to describe the concept of account in the Picsearch’s video platform. An account is the basic mechanism to manage storage and access to videos located in the platform. A video is uploaded and registered to an account, thus the relationship account-video is 1:n. Further, each account can give access to multiple users, and in turn, a user can have access to multiple accounts by using the same username and password. A user must have a particular role (e.g., administrator, editor, user) per account. The role represents a particular level of rights that allow or constrain the user to perform certain operations (e.g., streaming, uploading, rating). Each account is identified by an encrypted string token that contains account’s information as the account’s identifier, the role and the token’s validity period.
Use Case - Browse Categories: The use case (UC) enables a user to get a list of all the available video categories associated to an account. Each category is composed by an identifier and a description. A video can be assigned to only one category, thus the relationship category-video is 1:n. In this UC the application communicates with Picsarch’s server and requests all categories. All categories are displayed in the application and the user selects one of interest. The application then requests the server all the videos related to such category and displays the response (see Figure 2.3).

![Figure 2.3: Browse Video Categories](image)

Use Case - Search Video: The application enables the user to perform a search and find a particular video. The searching process covers the following criteria: 1) video title, 2) video description and 3) video tags, and is not exclusive to cover only one criterion. It accepts one, two or the three criteria if needed. The user types the term of interest and executes the searching operation. The application responds with a list of videos that satisfy the search criteria (see Figure 2.4).

![Figure 2.4: Search Videos](image)
Use Case - Play Video: The user selects a video either by browsing diverse categories or by a searching operation. The application then retrieves detailed information to access videos stored in Picsearch’s video platform. This information includes the video’s identifier, name, description, URL, resolution and encoding format among the most important fields. The application then uses the video’s URL to start the streaming process and play the video in the smartphone’s screen (see Figure 2.5).

Use Case - Rate video: The user provides to the application a rating value for a video of interest. The application sends this value to Picsearch’s server. The server calculates the average value after considering previous users’ ratings. The server returns the global rating to the application. Finally, the application displays the resulted value (see Figure 2.6).
Chapter 2. Market Analysis

Scenario 2 - Dynamic Video Uploading to Multiple Accounts

This scenario (see Figure 2.7) was inspired on a journalism domain, where Picsearch counts with important customers. This functionality relies on Scenario 1, but it intends to turn the perspective to cover the interests of a journalist. The scenario provides journalists a flexible way to upload mobile video after recording it in sudden events where a professional camera is not available. The scenario also allows a journalist to moderate a video. The moderation enables or disables the video from being publicly accessed. Furthermore, in order to organize their videos it is common that journals own several Picserach storage accounts. Therefore, this scenario also considers this situation and enables a journalist to choose dynamically the account that he/she needs to operate from their mobile devices.

![Figure 2.7: Dynamic Video Uploading to Multiple Accounts - Use Cases](image)

**Use Case - Select Video Account:** The credentials required to access a Picsearch account are a valid username and password. A user can be allowed to access one or more accounts with the same credentials. Consequently, this use case enables a user to retrieve a list of accounts, where each account’s object is composed by its token, account’s description and role. Next, the user selects the account of interest. The string token for the respective account is kept as means to access the video platform and execute subsequent operations (see Figure 2.8). This use case also allows the user to change the account on demand at any moment.

![Figure 2.8: Select Video Account](image)
Chapter 2. Market Analysis

Use Case - Upload Video: Assuming that a video account has been selected (see Select Video Account UC), the user requests the application to upload a video. The uploading process requires the application to submit the video title, description, category, related tags and file to upload. The data is sent through the network and stored in Picsearch’s server. Once the process is completed the server responds the application with the result of the operation, same that is shown to the user (see Figure 2.9).

![Figure 2.9: Upload Video](image)

Use Case - Moderate Video: Video moderation enables or disables the video from being publicly accessed, i.e., even though the public URL of a video is known, Picsearch’s server can block the streaming process if the moderation status indicates so. Thus in this use case the user identifies a video and provides to the application a moderation status. The application sends this status to Picsearch’s server, where online access to that video will be enabled or disabled. Finally, the server returns to the mobile application the result of the operation, same that is displayed for the journalist’s acknowledgement (see Figure 2.10). The assumption that justifies this use case is that important events will require a journalist to enable the video for immediate access.

![Figure 2.10: Publish Video](image)
What are the strategies taken by Picsearch’s competitors to support the Android platform?

Competitors are often sources of inspiration for coming up with new ideas or improving the current level of services. To answer this question we enquired Picsearch about the companies that compete in the market of video streaming. Once the information was available we started reviewing every competitor’s website in order to find out the services they offered specifically for the Android platform. The strategy consisted in finding how other companies were approaching Android, or in general mobile platforms.

Results

The reviewed companies were: Apegroup, Streamingbolaget, Brightcove, JayCut, Mpsbroadband, Ooyala.com, Qbrick, Streamio and The Plaform. Our findings were that all companies optimize video transcoding in order to offer a correct visualization in small or medium size screens. Nevertheless, with the exception of Brightcove and Picsearch, the rest of the companies do not offer any mechanism to facilitate the usage of their services in mobile platforms. One of the outcomes of this thesis was the delivery of a software component capable of reducing the developer’s effort to integrate Picsearch streaming services in Android applications. Time savings can be translated to economic benefits. Therefore, we believe that this work is backed up by an interesting strategy that seems to be neglected by most competitors. Our analysis of this issue derives into two possible causes: 1) competitors in general assume that most mobile applications are developed for mobile browsers, or 2) they minimize the effort required to build the communication to their services.

We think that the first possible cause may represent a valid assumption. However, the current market seems to encourage both: mobile web applications and native applications. Each type experiences benefits and drawbacks, and the decision about using one or another depends on the characteristics of the application that is to be built.

Concerning the second possible cause, the infrastructure required to communicate a mobile application with the video platform may not be too high in terms of code complexity; however, it requires time and expertise to do it in a proper manner. Reliability on the other hand becomes another advantage. By providing a common infrastructure that is massively used by multiple customers, the degree of reliability on the software component increases because any possible problem will be easily detected and fixed. Therefore Picsearch considers that the component delivered with this thesis will represent a competitive advantage that could pay off with the current boom of Android devices.
From all the reviewed companies there is one must be highlighted: BrightCove. Brightcove is an American company with an extensive market share. To the best of our knowledge they are the only company already providing an SDK (software development kit) for Android. The SDK includes an API that provides read-only methods to access their media platform. Furthermore, they provide a customized media player that extends the functionality provide by Android’s built-in video player. By observing one of the leading companies taking this step we realized this is the way to proceed for promoting mobile video, and this is as well a very positive step taken by Picsearch to increase its market share.
Chapter 3

A Systematic Review on Architecture Design Methods

3.1 Introduction

Software architecture plays a critical role for attaining intellectual control over a sophisticated system with enormous complexity [50], so the analysis of a software architecture is a central activity in software development [14].

Quality software is such that is fit for its aimed purpose, thus in that sense, the quality of a system emanates in a large extent from the software architecture. High quality software comprises the right features and the right attributes required to satisfy business goals and user needs [61]. This is the level where the system’s quality attributes are verified and described [32]. A software architecture enables a fundamental way for communicating design decisions and establishing effective work breakdown structures [61]. Besides, the use of explicit descriptions of an architecture improves system comprehension and promotes software reuse [32].

The process of building quality software requires a discipline in all its phases, emphasizing a careful design of the software architecture [61]. Nevertheless, software architecture has remained ignored and frequently less important when compared to other activities like coding [27]. Further, even in academical environments, software architecture has been often neglected in software engineering education to teach only low-level code design [61]. This fact is contradictory with the general perception that considers a software architecture as an enormous risk activity in software projects, where the wrong architecture may lead to poor quality software and even worse, to a project failure [61].

Architectures are influenced by the system requirements specification (SRS), and other factors (organizational, technological) that directly or indirectly affect a software project. A proper analysis needs to be employed for transforming a set of influential factors into a software architecture. This analysis will act as an indicator of the degree in which an architecture fulfills its significant requirements.
Currently, software architects can choose among several methods for designing a software architecture. Such methods are called Software Architecture Design Methods\cite{22}. Motivated for applying our findings in a practical case at Picsearch, we intend to find out the availability of those architecture design methods and their applicability and usefulness towards our Master’s thesis. Thus this chapter presents a systematic literature review (SLR) that covers the architecture design methods available in the current software engineering (SE) literature.

3.1.1 Research Questions

For this review, we were interested in researching about the availability of architecture design methods, as well as their characteristics and the details about their application. As a result, the following research questions were formulated:

- **RQ1**: What are the current state-of-the-art SE methods that provide guidance for designing a software architecture?
  - **RQ1.1**: What are the core elements of each method?
  - **RQ1.2**: What are the steps needed to execute the method?

- **RQ2**: Which method presents a higher suitability for supporting our practical architecture design?

3.2 Planning

The planning phase discusses the search strategy, the literature sources, the inclusion and exclusion criteria, and the overall methodology of this SLR.

3.2.1 Search Terms

A proper register of the utilized terms facilitates the transparency and replicability of the searching process\cite{47}. As advised in \cite{33, 69}, we followed the strategies explained in Table 3.1 to identify the search terms presented in Table 3.2.
Chapter 3. A Systematic Review on Architecture Design Methods

Strategy

1. Major terms are obtained from the research questions by identifying the context and outcome.
2. By identifying alternative terms and synonyms of search terms.
3. By extracting the keywords from some relevant papers already identified.
4. Boolean OR is utilized for including search terms or alternative spellings and synonyms.
5. Boolean AND is utilized to link the major terms with other terms.

<table>
<thead>
<tr>
<th>Search terms</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Software architecture design</td>
</tr>
<tr>
<td>2. Global analysis</td>
</tr>
<tr>
<td>3. Method</td>
</tr>
<tr>
<td>4. Technique</td>
</tr>
<tr>
<td>5. Approach</td>
</tr>
<tr>
<td>6. Guide</td>
</tr>
<tr>
<td>7. Analysis</td>
</tr>
<tr>
<td>8. Software requirements specification</td>
</tr>
<tr>
<td>9. Requirements</td>
</tr>
<tr>
<td>10. SRS</td>
</tr>
</tbody>
</table>

Table 3.1: Search terms construction process

A primary revision utilized as sources the databases presented in Table 3.3. These prestigious databases were selected due to their high quality content of SE articles. Furthermore, we executed a snowball process in order to ensure that important articles were not missed. A snowball process is a secondary revision based on references encountered in the articles selected during the primary revision. Its purpose is to identify an extra set of relevant studies. The snowball search was powered by Google Scholar in addition to the databases described in Table 3.3.
Chapter 3. A Systematic Review on Architecture Design Methods

3.2.2 Study Selection and Quality Criteria

The basic and the detailed inclusion/exclusion are guidelines that support the reviewer to decide if an article focuses on the SLR’s object of study. In this SLR the basic inclusion criterion consisted in including all those studies centred on software architecture design methods. This criterion was applied by reading titles, keywords and abstracts for all the studies identified in the primary revision, as well as in the snowball search. Only articles that satisfied the basic criterion were subjected to the detailed inclusion/exclusion criteria. Otherwise, the articles were dismissed from the SLR.

Our detailed inclusion/exclusion criteria are presented in Tables 3.4 and 3.5. We applied them by reading abstracts, introductions and conclusions of the studies. One important criterion was the inclusion of comparative studies and literature reviews. These type of studies specialize on collecting and summarizing state-of-the-art literature about a specific topic. Thus we analyzed them in order to identify references to interesting articles, and consequently extend our list of selected studies. This strategy resulted very fruitful because we realized that important work had been done previously in this area.

<table>
<thead>
<tr>
<th>Study Inclusion Criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 The article is peer reviewed.</td>
</tr>
<tr>
<td>2 The article is available in full text.</td>
</tr>
<tr>
<td>3 The article discusses a method, approach or framework that provides guidance about designing a software architecture.</td>
</tr>
<tr>
<td>4 The article includes a comprehensive design methodology.</td>
</tr>
<tr>
<td>5 The article can be a comparative study, literature review or a systematic review.</td>
</tr>
<tr>
<td>6 The article provides experiences or additional information about a particular method.</td>
</tr>
</tbody>
</table>

Table 3.3: Literature Databases

<table>
<thead>
<tr>
<th>Databases</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 IEEE Xplore</td>
</tr>
<tr>
<td>2 ACM Digital Library</td>
</tr>
<tr>
<td>3 Springer Link</td>
</tr>
<tr>
<td>4 Engineering Village (Inspec, Compendex)</td>
</tr>
<tr>
<td>5 Scopus</td>
</tr>
</tbody>
</table>

Table 3.4: Detailed Inclusion Criteria
Chapter 3. A Systematic Review on Architecture Design Methods

24

Study Exclusion Criteria

1. Articles that do not match the previous inclusion criteria will be excluded.
2. The article discusses an automated architecture design method.
3. The article is mostly focused on presenting a CASE tool for supporting architecture design.
4. The article discusses only architecture evaluation or analysis without discussing design aspects.

Table 3.5: Detailed Exclusion Criteria

3.2.3 Quality Assessment

In addition to the inclusion/exclusion criteria it is also important to ensure quality through an assessment of the selected studies [47, 69]. A quality assessment allows to understand the limitations of each individual study [69], and to facilitate the selection process by comparing the targeted studies against a quality criteria. We defined our quality criteria as suggested in [69, 47, 48, 13, 33], same that is presented in Table 3.6.

<table>
<thead>
<tr>
<th>Quality Criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Is the design methodology clearly defined and appropriate for the problem under consideration?</td>
</tr>
<tr>
<td>2. Is the study explaining its contribution in relation to other design methods previously available?</td>
</tr>
<tr>
<td>3. Is the method considering functional and non-functional requirements analysis as part of the methodology?</td>
</tr>
<tr>
<td>4. Are there any restrictions or limitations reported?</td>
</tr>
</tbody>
</table>

Table 3.6: Quality Criteria

3.2.4 Data Extraction

The data extraction form helps to identify important articles’ information that try to answer the SLR’s research questions [48]. Table A.1 in Appendix A presents our data extraction form.
3.3 Execution and Results

As shown in Figure 3.1, the queries presented in Tables A.4 and A.5 (see Appendix A) served to extract our primary studies. From 475 identified studies, 398 studies were excluded by applying our basic criterion. From the 77 remaining studies 48 were removed as they were duplicates. Our detailed inclusion/exclusion criteria and quality assessment were applied over the remaining set of studies, resulting in a total of 12 articles. After executing the snowball search 8 additional studies were added to our inclusion set. The final set consisted of 20 studies, which is presented in Tables A.2 and A.3 (see Appendix A).

NB: As part of the snowball search we found that some studies reference books as support for understanding some design methods. These books were not included in our inclusion set, nevertheless they are referenced during the explanation of the methods in Section 3.4.

<table>
<thead>
<tr>
<th>Data source</th>
<th>Total Found</th>
<th>Total Selected</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEEE Xplore</td>
<td>54</td>
<td>13</td>
</tr>
<tr>
<td>ACM Digital Library</td>
<td>53</td>
<td>8</td>
</tr>
<tr>
<td>Springer Link</td>
<td>39</td>
<td>4</td>
</tr>
<tr>
<td>Engineering Village (Inspec, Compendex)</td>
<td>191</td>
<td>23</td>
</tr>
<tr>
<td>Scopus</td>
<td>138</td>
<td>29</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>475</strong></td>
<td><strong>77</strong></td>
</tr>
</tbody>
</table>

*Table 3.7: Results Found per Database*

*Figure 3.1: SLR Execution Process*
3.4 Analysis

3.4.1 Available Software Architecture Design Methods

What are the current state-of-the-art SE methods that provide guidance for designing a software architecture? 20 studies related to software architecture design methods have been found in this SLR. These studies discuss 9 different design methods. In [39] and [38] Hofmeister et al. present a discussion and comparison of five different design methods: Attribute-Driven Design (ADD) method [8, 9, 5], developed at the SEI; Siemens’ 4 Views (S4V) method [68, 36, 37, 58, 59], developed at Siemens Corporate Research; the Rational Unified Process or 4 + 1 views (RUP 4 + 1) [49, 51] developed by Rational Software (now IBM); Business Architecture Process and Organization (BAPO) [1, 34, 74] developed at Philips Research, and Architectural Separation of Concerns (ASC) [63] developed at Nokia Research.

Furthermore, in [22], Falessi et al. extend the work previously presented by Hofmeister et al. [39, 38] performing a comparison of nine different design methods in order to evaluate to which extent such methods cover architects’ needs. Five of these methods are already discussed in Hofmeister’s work. From the rest, based on our quality criteria we decided to include: Goal & Scenario (G & S) [75], and Component-Bus System and Properties [31].

Although not covered by the previous methods comparisons, two more design methods were found during this SLR: Gradually Proceeded Architecture Design (GADesign) [71] method; and the Business And Software Architecture co-Design (BASAD) [18] method.

3.4.2 Principles and Core Elements

This section describes the core elements of each design method, as well as an overview of how the authors explain the method’s execution. By doing this we intend to answer the following sub-research questions:

\textbf{RQ1.1:} What are the core elements of each method?

\textbf{RQ1.2:} What are the steps needed to execute the method?
Chapter 3. A Systematic Review on Architecture Design Methods

Attribute-Driven Design (ADD)

The Attribute-Driven Design (ADD)[8, 9, 5] method was created at the Software Engineering Institute(SEI). The ADD\(^1\) method focuses on determining the architectural drivers for a system. "The architectural drivers are the combination of business, quality and functional requirements that shape the architecture"[8].

The ADD method recursively decomposes the system to be designed. The first decomposition is performed on the overall system, while subsequent decompositions are refinements of the preceding decomposition. At each decomposition step the functional requirements are met by assigning responsibilities or tasks to the divisions of the element being decomposed. The quality and business requirements for the given element are met by selecting an architecture style[8].

The decomposition is analyzed using three separate views: the logical view, the concurrency view and the deployment view. Each view describes different responsibilities for the element being decomposed. The logical view is used to capture the responsibilities and conceptual interfaces for the design elements. The concurrency view is used to examine the system in terms of instances of components, resource contention, synchronization points and other parallel activities. The deployment view is only used when systems execute on multiple processors, and it supports the architect to reason about allocation to physical hardware. [5, 8, 39, 38].

The final architecture shows the decomposition of the system’s functionalities into units, the identification of possible concurrency and physical network configurations, and the allocation of the functional units to processors [5]. The architectural drivers(significant functional, quality and business requirements) are utilized to verify that decisions made during the decomposition are correct. Further, design constraints are also checked to verify that none of them has been violated[8].

In ADD architects perform the following activities for each module being decomposed: 1) select the architectural drivers, 2) select an architectural pattern that satisfies the drivers, 3) create modules, assign them functionality from use cases and model the results using multiple views, 4) define interfaces for the modules, and 5) verify the modules satisfy use cases and quality scenarios [38, 39].

\(^1\)Initially called the Architecture Based Design method, but the name is a trademark of Lockheed Martin, Inc.
Siemens’4 Views

The Siemens Four-Views (S4V) method was developed at Siemens Corporate Research. The method utilizes four different views: conceptual, execution, module and code architecture view, in order to separate different engineering concerns. Thus this reduces the complexity of the architecture design.

The conceptual view is focused on the application domain, where the functionality of the system is translated and mapped to conceptual components and connectors. The module view maps components and connectors obtained from the conceptual view to subsystems and modules. The module view helps to address how the conceptual solution can be attained with today’s software platforms and technologies. The execution view describes how modules are mapped to runtime platform elements (e.g., OS tasks, processes, threads). The code view supports the architect to determine how runtime entities from the execution view are mapped to deployment components (e.g., executables), how modules from the module view are mapped to source components, and how the deployment components are produced from the source components.

These views are strongly supported by a recurring activity called Global Analysis. Global Analysis aims at identifying the organizational, technological, and product factors that influence the architecture design (e.g., requirements, desired system qualities, organizational constraints, existing technology, etc.). Based on these factors, architectural issues or challenges are identified. As next step, design strategies are proposed to overcome these issues or challenges. The chosen design strategies are applied to one or more views. The method expects the architect to interleave Global Analysis with the views design process, and iterate among the design tasks of the four views until the architecture fulfills its significant requirements.

RUP’s 4 + 1 Views

The Rational Unified Process (RUP) is a software development process created by Rational Software, nowadays IBM. RUP defines a software architectural design method that uses the concept of 4 + 1 views (RUP 4 + 1). Four views to describe the design: logical view, process view, development view and physical view. An extra use case view (+1) is utilized to associate the architecture design to the system’s context and goals.

The logical view primarily addresses functional requirements by decomposing the system into objects or classes abstracted from the problem domain. The process view works with concurrency and processes distribution, system integrity
and fault tolerance. The development view describes the static organization of the software in its development environment, *i.e.*, how software is packaged and deployed. The physical view describes how software is mapped to hardware. The fifth or +1 view includes a set of selected use cases or scenarios that supports the understanding of the architecture design; same that is organized around the prior four views\[49, 51\].

One of the main characteristics of RUP is its iterative nature. The architectural design is carried over several iterations, gradually populating the 4 views and driven by architecturally significant use cases, nonfunctional requirements and risks. Each iteration produces an executable architectural prototype, which is used to validate the architectural design\[38, 39\].

**Business Architecture Process and Organization**

The Business Architecture Process and Organization (BAPO)\[74, 1, 34\] method was developed by Philips Research, and aims at developing an Architecture (the A in BAPO) for software-intensive systems that fits optimally in the context of Business (B), development Process (P), and Organization (O)\[34, 74, 38, 39\].

In BAPO the purpose of architecture is to bridge the gap that exists between the needs, objectives, and wishes of the customer and their realization by available technology. In this approach five views(customer view, application view, functional view, conceptual view, realization view) or CAFCR views, support the structure of the architecture description that intends to bridge the existing gap between the customer and technology. The customer view captures customer-related knowledge. The application view studies how the system that is being architected can be used to fulfill the customer’s needs. The functional view intends to describe the functional and non-functional requirements of the system under development in a precise way. The conceptual view describes the different parts that comprise a system, how they cooperate, and the essential concepts that govern how the system works. The realization view describes how the system is developed using available technologies\[1, 34\].

In this method the architect iteratively perform the following steps: 1) collects information in one of the CAFCR views; 2) examines a particular quality attribute across the views to set up a link among the views and in accordance to the surrounding business, processes and organization. Finally, the architecture is considered complete when there is sufficient information to build the system and the quality attribute analysis shows no discrepancies 38, 39].
Architectural Separation of Concerns

Architectural Separation of Concerns (ASC) or ARES System of Concepts\textsuperscript{[63]} was developed by Nokia. It is a conceptual framework that aims at achieving separation of concerns to reduce the complexity of an architecture design. ASC is based on the premise that concerns related to different segments in the architectural process (typically including design, implementation, deployment, execution) are often separable\textsuperscript{[39, 38]}\textsuperscript{[39, 38, 63]}. In ASC the architect examines a set of design inputs such as functional and non-functional requirements and technology platforms. Then, using a palette of techniques produces and prioritizes ASRs (architecturally significant requirements), and groups these ASRs in segments based on the architectural process that they address\textsuperscript{[39, 38, 63]}. ASC considers three design categories: 1) implementation, 2) performance, and 3) delivery/installation/upgrade.

Implementation (write-time) design addresses the ASRs related to the write-time segment. Some of its design decisions involve\textsuperscript{[39, 38, 63]}:

- The selection of the implementation technology,
- The partition of functional requirements among different architectural scopes of a product portfolio, product family, or single product,
- The establishment of portability layers for multiplatform products,
- The organization of classes of functional requirements in different subsystems,
- The development of API descriptions to facilitate work division and outsourcing.

Performance (runtime) design addresses runtime-related ASRs such as concurrency and performance models. Moreover, it defines task and process partitions, scheduling policies and resource allocation\textsuperscript{[39, 38, 63]}.

Delivery/installation/upgrade design decisions deal with ASRs of the corresponding segments such as partitioning into separately loadable/executable units, installation support, configuration data, upgrade/downgrade policies and mechanisms, management of shared components, external dependencies and compatibility among requirements\textsuperscript{[39, 38, 63]}.

Goal & Scenario

The Goal & Scenario(G&S) approach proposes to combine the use of goal-oriented and scenario-based models to perform an incremental architectural design process. Goals are used to refine functional and non-functional requirements, explore alternatives, and their application to an architectural design. Goals are based on
the use of the Goal-oriented Requirement Language (GRL). The scenario notation is used to illustrate the incremental inclusion of requirements into the architectural design. This illustration is carried by the use of Use Case Maps (UCM). Even though goal-orientation is very suitable for dealing with requirements, it is possible that sometimes goals are too abstract to capture at once. In these cases, operational scenarios can be useful to facilitate the understandability of those goals.[75]

G&S aims at developing GRL models and refining business goals and non-functional requirements until design decisions are taken. These decisions are then further elaborated into UCM scenarios. UCM scenarios are used to describe the diverse functionalities considered by the architecture design. Further, UCM scenarios are also useful to verify these functionalities satisfy the system’s goals. The G & S approach iterates this process in an incremental manner until the architecture design is considered complete.[75]

**Component Bus System Property (CBSP)**

The CBSP (Component, Bus, System, Property)[31] approach aims at associating software requirements and architectures using intermediate models. CBSP intends to extract and identify architectural information from the system’s requirements, and then use this information to explicate the system with a requirements-architecture intermediate model[31].

CBSP introduces the concept of dimensions(C,B,S,CP,BP,SP). Dimensions are a set of general architectural concerns that can be used for: 1) to perform a systematic classification and refinement of requirements, and 2) to capture architectural trade-off issues, and alternatives available. C represents model elements that relate to an individual architecture component. B are model elements that involve a bus (connector). S applies for system-wide features or features related to a large subset of components and connectors within the system. Further, CP implies data or processing component properties, BP bus properties, and SP system (or subsystem) properties. Each requirement is assessed for its relevance to the system’s dimensions(C,B,S,CP,BP,SP). Thus, each derived CBSP artifact explains a different architectural concern, and represents an early design decision[31].

CBSP involves the following steps: 1) requirements selection for next iteration, 2) architectural classification of requirements, 3) identification and resolution of classification mismatches, 4) architectural refinement of requirements, 5) trade-off choices of architectural elements and styles with CBSP[31].
CBSP allows developers to systematically explore and identify: 1) architectural elements (components and connectors, 2) architectural properties (CP, BP, and SP), 3) architectural dependencies, and 4) suitable architectural styles. Nevertheless, the application of CBSP does not produce an architecture directly. CBSP supports the design of the building blocks required to architect a given system. Later on, the architect will take these blocks and use them to build an effective architecture that fulfills requirements[31].

A Gradually Proceeded Software Architecture Design Process (GADesign)

The Gradually Proceeded Software Architecture Design Process (GADesign)[71] divides the architecture design process into several phases and provides a sequence of well-defined steps that facilitate such process. The input to GADesign is a list of functional and non-functional requirements. GADesign divides the architecture design into three phases: 1) Software architecture sketch design (SAS Design), 2) Software architecture refinement, and 3) Architecture transformation to implementation[71].

SAS design provides system stakeholders with an initial sketch (SAS model) of the system’s architecture. This sketch explains how the architecture fulfills the system’s requirements. The software architecture refinement phase intends to refine the SAS design model in order to attain a more detailed architecture specification. This phase involves the definition of interfaces and behavior details of each component, and their interaction protocols. Further, runtime elements such as processes/threads and their interaction are also defined. The architecture transformation to implementation phase uses the architecture specification obtained from the prior phase, and maps it into a component implementation platform[71].

One important characteristic of GADesign is the flexibility of its phases. Depending on the characteristics and complexity of a given project GADesign allows phases to be merged into one, or one phase can be split into more smaller phases to simplify the architecture design. The architecture is considered complete once the requirements have been addressed in the design[71].

Business And Software Architecture co-Design (BASAD)

The Business And Software Architecture co-Design (BASAD)[18] process involves business architecture design, software architecture design, and automated transformation between the business and software architecture models. Business architecture design is driven by requirements and business goals, and it is carried by business architects in collaboration with stakeholders. Software architecture design is conducted by software architects, and utilizes as base the previous busi-
ness architecture. Both, business architects and software architects execute the architecture modeling design and refinement process in an interactively and iteratively manner\cite{18}.

Business architecture models (BAMs) describe an architecture from the business perspective. BAMs are composed by business building blocks, workflows, process logics, and system’s constraints. Furthermore, BAMs express requirements that need to be met by the architecture design. Software architecture models (SAMs) describe the structure of the system, the components that constitute such structure, and the components’ inter-relationships. Hence, BASAD transforms BAMs into SAMs\cite{18}.

The transformation process is executed in two steps: 1) model mapping and 2) model merging. In the former step, the BASAD model transformer converts BAMs into the Business Computing Models (BCMs). The model transformer consists of a set of predefined mapping rules that allow to convert a BAM into a BCM. In the model merging step, the model transformer merges BCMs and Support Computing Models (SCMs). SCMs are business-independent computing models that provide infrastructures to support the system’s computing. The result of the model merging step is the final Software Architecture Model (SAM) that represents the complete system’s architecture\cite{18}.

3.5 Discussion

Nine different software architecture design methods were reviewed and identified as part of this SLR. Because their systematicity to separate concerns it can be understood that all methods are prepared to target architectures for large software systems. In \cite{39, 38} it is mentioned that architecture design methods experience some essential differences derived in a large extent from the domain where the methods were created, e.g., architecture designs for the information systems domain emphasize data modeling, while designs for the telecommunication domain emphasize continuous operation, live upgrade, and interoperability\cite{39, 38}. Nevertheless, as \cite{39, 38} also mentions, "all software architecture design methods must have much in common as they deal with the same basic problem", i.e., design a software architecture.

During our review we found the following several commonalities among the design methods. Here we present them:

- **Requirements**: Functional and non-functional requirements are utilized as inputs for starting the architecture design process.
• **Transformation:** Implicitly or explicitly the majority of the methods intend to transform the requirements into design decisions, making this task systematic and traceable along the design process.

• **Separation of concerns:** Either by explicit views or conceptually, all the reviewed methods consider the separation of concerns as means to reduce the complexity of the design process.

• **Evaluation:** An architectural evaluation is performed either, at the end of a design iteration or at the end of the design process. This evaluation aims at validating that the resulted architecture fulfills the system’s requirements.

Moreover, Hofmeister et al. 39, 38 propose a general model that includes commonalities among architecture design methods in addition to desirable properties that an ideal method should experience.

![Figure 3.2: Architecture Design Activities][39, 38]

As can be observed from Figure 3.2, the model consists of three activities: 1) Architectural analysis, which express architecturally significant requirements (ASRs) based on the architectural concerns and context of the system; 2) Architecture synthesis, that provides candidate architectural solutions that addresses ASRs; 3) Architectural evaluation, which validates that the design decisions taken are correct to fulfill the ASRs. As can be seen, these tasks are aligned with the commonalities we found through our SLR.

Once explained our findings in this SLR, it is important to retake **RQ2:** Which method presents a higher suitability for supporting our practical architecture design? To answer this question it is necessary to emphasize that all methods would be able to support our design because fundamentally they strive for achieving the same goal by following different procedures. However, even though all the review methods share a common goal, they have different characteristics that make them more or less suitable for a certain software system. Thus at this point we need to consider the context of Picsearch and our component, and evaluate our set of methods to decide which one fits best in our case study.
3.5.1 Method Selection

RQ2: Which method presents a higher suitability for supporting our practical architecture design? In order to answer RQ2 we need to consider the context of Picsearch and the product to be designed. Our product is a clear market-driven requirements engineering example, where the requirements are set on products rather than directed towards projects\[29\]. In market-driven scenarios, the flow of requirements is not limited to one project, and the requirements are elicited from internal (e.g., designers, analysts) and external (e.g., customers) sources\[29, 10\]. In our case, the flow of requirements were elicited after considering the following sources:

- **Internal sources**
  - **Picsearch services**: Technical capabilities and specifications of their video platform. This includes the characteristics of the REST API.
  - **Current customers base**: Even though Picsearch still is not offering services tailored for the Android platform, the CEO and the sales manager sketched a scenario where they explain potential customer’s needs and expectations.

- **External sources**
  - **Popular video mobile applications**: Popular applications that make use of mobile video were an excellent source of inspiration to analyze their inclusion of mobile video features.
  - **Similar companies**: At this point most of the competitors were still not giving high priority to the Android platform, but at least one in addition to Picsearch was already providing support to facilitate the use of their services in the development of Android applications.

In our study, the requirements were not completely defined in an initial stage. This was in a large extent because this product was not focused on a specific customer, but on the market available for online video providers. Hence, requirements were elicited gradually and iteratively in order to set a base of significant requirements that covered the most essential features required by the market of mobile video applications. Time is limited for a Master’s thesis, therefore the design phase was soon started in order to get early feedback from the incremental deliverables.

For projects with the previous characteristics, Madison\[54\] suggests the use of an agile-architecture approach. This approach consists in "pragmatically balancing business and architectural priorities while delivering both with agility"\[54\]. Furthermore, Nord et al.\[60\], support that "architecture-centric methods are built
on concepts and techniques that practitioners can tailor to an agile approach”. Moreover, they state that “architecture-centric methods can add value to agile methods by emphasizing quality attributes and their role in shaping the architecture’s design”[60]. Thus the suitable design method for our architecture required: 1) to support an iterative design process, and 2) be flexible and open to tailoring to be used in small software projects where agility to deliver is critical.

The tailoring process for the reviewed design methods was not explicit in all cases, nevertheless it is over-understood that this can be done and that it is subject to the experience and needs of the architect, and size and complexity of the system to be built. GADesign describes tailoring in a very clear manner, fact that caused a deep interest on us.

In a general comparison, we went from exhaustive architecture-centered approaches (ADD, S4V, RUP 4+1, BAPO, ASC, CBSP, BASAD) to more light-weight design methods(G&S, GADesign). Our requirements for a method required a right balance among exhaustiveness, understandability in terms of method specification and use, tailoring, and applicability to agile projects like ours. Thus after the review, our general feeling was that GADesign successfully covered our requirement criteria. It is not stated but clear that GADesign is to some extent inspired on other more mature and exhaustive methods, probably ADD, RUP 4+1 and S4V. It does not take the concept of views, but instead define a clear gradual process to design an architecture. GADesign is less exhaustive to some extent but simpler and clearer to implement. Our general feeling after reviewing GADesign’s article was that it provided: 1) the right degree of exhaustiveness that our component required, 2) a very clear explanation about its structure and use, 3) encouragement to tailor it to the needs of the architect, and 4) a prefect match to iterative software projects. Thus despite of having reviewed other interesting design methods, we believe that in this specific situation GADesign was the best option. Therefore, we decided to use it to overcome our architecture design process, same that is presented in Chapter 4.

3.5.2 SLR Strengths and Weaknesses

The most important software engineering databases have been covered in this SLR in order to extract the articles presented in Tables A.2 and A.3(see Appendix A). Hence, we are highly confident that our review covers the most relevant design methods published in the current SE literature. A possible weakness is that unintentionally we could have missed some search terms that could have returned other important results. Furthermore, English is not our native language, so it always exists the risk of misinterpretation during the review process. The reduce the impact of these issues we tried to double check the review process, specially the search terms and the studies when something seemed to be unclear.
Chapter 4

Architecture Design

This chapter explores the execution of our component’s architecture design process. In order to find a suitable design method we performed the systematic literature review (SLR) presented in Chapter 3, and selected GADesign[71] as the method that provided the guidelines required to support the architecture design.

GADesign originally divides the design process into three phases: 1) Software architecture sketch design (SAS Design), 2) Software architecture refinement, and 3) Architecture transformation to implementation. However, the method specifies adaptability depending on the characteristics and complexity of a given project.

The experience executing the method set the basis to answer RQ3-2: How did the use of GADesign enhance the architecture design process? Our discussion nevertheless is not presented here but in Chapter 6, were we describe in detail the benefits experienced. Hence, Chapter 4 specifically discusses the systematic design process according to the GADesign’s guidelines.

4.1 Software Architecture Sketch Design

The SAS Design phase provides system stakeholders with an initial sketch of the system’s architecture. This sketch enables early understanding and evaluation about its ability to fulfill the specified requirements[71]. We accomplished the SAS Design phase in three steps: 1) identify software requirements, 2) identify and select architecture tactics, and 3) create architecture sketch. Each step is described in the following sections.

4.1.1 Identify Software Requirements

This step is aimed at identifying those significant functional requirements or quality attributes that impose strong influence over the architecture design. Succeeding steps utilize these requirements to shape the architecture.
Functional Requirements

The component being designed is required to communicate with Picsearch’s REST[23, 24] API. The REST API is the access point to the functionality delivered by the online video platform. Our component abstracts the issues involved in the communication between an Android application and the REST API. This in order to enable the video platform’s functionality in Android, and ease to developers the access to this functionality. In terms of functional requirements, we decided to include the functionality required to support the scenarios presented in Section 2.1.4. This functionality is summarized as follows:

- **Get profiles data**: The component must retrieve the accounts associated to a valid username and password. An account is the basic mechanism utilized by Picsearch to manage storage and access to videos located in the platform. A video is uploaded directly to an account, thus the relationship account-video is 1:n. Further, each account can give access to multiple users, and in turn, a user can have access to multiple accounts by using the same username and password. A user must have a particular role (e.g., administrator, editor, user) per account. The role represents a particular level of rights that allow or constrain the user to perform certain operations (e.g., streaming, uploading, rating). Each account is identified by an encrypted string token that contains account’s information as the account’s identifier, the role and the token’s validity period. The string token for the respective account is used as means to access the video platform and execute subsequent operations.

- **Get media information for streaming purposes**: The component must provide a method to retrieve detailed information to access videos that are stored in Picsearch’s video platform. This information includes mainly the video’s identifier, name, description, URL, resolution and encoding format. The URL is specially important because it enables Android’s media player to access the video in order to start the streaming process.

- **List media information**: The component must provide functionality to retrieve a list of videos and video categories associated to an account. The listing functionality must allow to paginate or filter data in order to obtain more specific results.

- **Search media content**: The component must enable a method to perform video searching. The method must allow the caller to define a search criteria based on: 1) video title, 2) video description and 3) video tags. The searching process must not be exclusive to cover only one criterion. It must cover one, two or the three criteria if needed.
- **Upload media content**: The component must implement functionality to upload videos to Picsearch’s platform. Along with the file to upload, the method must receive as arguments the video’s name, description, category and associated tag.

- **Moderate videos**: Video moderation enables or disables a video from being publicly accessed. In other words, even though the public URL of a video is known, Picsearch’s server can block the streaming process if the moderation status indicates so. Thus the component must offer functionality to modify the moderation status for a given video.

- **Assign rating**: The component must deliver functionality to rate a video. When a rating request arrives, Picsearch’s platform calculates an average value after considering previous ratings. Consequently, the method must return this average rating value.

An important aspect in regard to the REST API is its response type. This is made of JSON(JavaScript Object Notation) content. All responses provided by the REST API require to be converted into Java plain objects. Android utilizes Java as its default programming language. Therefore, the goal of providing Java objects is to facilitate the manipulation of data. Additionally, passwords sent to the REST API must travel encrypted using a SHA-1 algorithm.

**Quality Attributes**

To select our significant quality attributes we reasoned about the benefits we wanted to provide with our software component. Our analysis derived into the idea of designing with two perspectives in mind: 1) customers, and 2) Picsearch. The former is the end user, and as well the main beneficiary of our work. The latter is the responsible to keep our component’s future evolution. Thus based on this consideration we decided to choose a set of quality attributes appropriate to benefice both stakeholders.

Regarding the customer’s perspective, our first goal was to provide a ready-to-use piece of software that could power multiple customers’ applications. Thus we identified reusability as our first quality attribute. Reusability in our context is the ability of include our component’s functionality in multiple Android applications without any minimal effort. Additionally we realized that it was paramount to facilitate the usage of our component. In our opinion the level of satisfaction accomplished by our component would depend on the API’s usage experience. Therefore, we identified understandability, consistency & usefulness as means to design a quality API able to truly minimize the effort of any developer to utilize it.
On the other hand, this thesis would produce just a first release of a product that Picsearch was expecting to evolve. Consequently we identified modifiability as the quality attribute that would help us to address an architecture capable to evolve and simple to maintain. Furthermore, security was a Picsearch’s concern that raised when the Get profiles data functionality (see Section 4.1.1) was suggested. This functionality requires the sending of passwords over the network, so, if security was not granted we would risk the possibility to compromise sensitive information.

Configurability was another potential quality attribute that was considered but finally dismissed. As we explained, the interaction with Picsearch’s platform is based on the usage of accounts. Every request to the platform requires to send the account’s token to indicate the account to target. In an initial stage of this work, when the Get profiles data functionality (see Section 4.1.1) was still not identified, we were concerned about developers hardcoding the accounts’ string tokens in their applications. This represented a problem because in case of a device loss, the account would be in risk of misuse by an alien person. Cancelling the account to open a new one would not be an option because it would impact other users accessing the same account. Because the unsuitability of this solution we decided to explore other mechanism to read the account’s token. A potential solution was to integrate our component to work with QR Codes (Quick Response code). A QR Code is a type of barcode specially designed to be read by smartphones cameras. The idea was to provide the account’s token as a QR Code image, so the customer’s applications would need to read this image to setup the account. We created a prototype for this idea but we found several disadvantages:

1. The extra effort added to customers to pre-configure their accounts.
2. The dependency to a third party library to read QR Codes.
3. Possible failures when accessing the smartphone’s camera to read QR Codes. When it comes to hardware access, it is not rare to experience unexpected behaviours in Android smartphones. This would risk the reliability of our component.

Based on the previous reasons we dismissed the use of QR Codes. Later on, still having the configurability issue, one of our interviews with the CEO derived into the scenario presented in Section 2.1.4. This scenario includes the Get profiles data functionality. This functionality tries to address a different goal in terms of functionality, but accidentally offered an elegant solution to overcome the configurability problem. Once the issue was solved we decided that our architecture design did not require extra efforts in terms of configurability, and the quality attribute was dismissed. Table 4.1 summarizes our component’s significant quality attributes:
Chapter 4. Architecture Design

4.1.2 Identify & Select Architecture Tactics

Once the architecture significant requirements were identified, the next step consisted in identifying and selecting architecture tactics that could be used to fulfill these requirements. GADesign assumes that in complex software systems there may exist a large number of candidate tactics available, and some of them may not be totally compatible in terms of the quality attribute they pursue (e.g., performance vs modifiability). Therefore, GADesign originally splits this step into two parts: 1) Identify candidate architecture tactics and 2) Trade-off analysis and tactic selection. Afterwards, during the trade-off analysis, the architect can prioritize which tactics and quality attributes are paramount. Our rationale to join these two steps in one was simply the manageable number of architectural tactics, and the absence of incompatibilities among them.

Architecture patterns, architecture tactics and design patterns are important elements that are strongly related to the architectural process. As follows we explain each concept:

- An architectural pattern, also known as architecture style, describes a fundamental structural organization schema for software systems[16]. Further, Fielding[23] defines an architecture pattern as a set of constraints that restricts the roles/features of architectural elements, and the allowed relationships among those elements within any architecture that comply to that pattern. A clear example is the client-server architectural pattern. The pattern is conformed by two element types: client and server, and their coordination is expressed in terms of the protocol that the server uses to communicate with each of its clients. The term client-server implies only that a server and multiple clients exist, but it does not discuss the specific

<table>
<thead>
<tr>
<th>Quality Attribute</th>
<th>Rationale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reusability</td>
<td>The component must provide specific functionality that can be reused in any Android generic application.</td>
</tr>
<tr>
<td>Modifiability</td>
<td>The architecture must be easily maintainable and capable to evolve to support new features.</td>
</tr>
<tr>
<td>Understandability, Consistency &amp; Usefulness</td>
<td>The API must be useful and easy to learn and understand in order to encourage developers to use it.</td>
</tr>
<tr>
<td>Security</td>
<td>Data sent over the network without proper encryption is prone to be sniffed and misused.</td>
</tr>
</tbody>
</table>

Table 4.1: Significant Quality Attributes
functionality assigned to any of the elements. An important characteristic
of architectural patterns is that they exhibit known quality attributes\cite{9}. For instance, the client-server architectural pattern enhances modifiability
because it separates the client’s responsibilities from the server’s. Modifications
applied to the implementation of the services or the number of servers
providing services is invisible (at least in principle) to the clients\cite{73}.

- An architectural tactic is a design decision that helps to control the result of
an important quality attribute for a given software system\cite{6, 9}. An example
of a tactic is the use of redundancy with the purpose of increasing the
availability of a system. Thus an architect counts with tactics as a palette
design options. Further, any architectural pattern implements several
tactics, often related to different quality attributes\cite{9}.

- A design pattern expresses a commonly recurring structure of communicat-
ing components that solves a general design problem\cite{43}. Design patterns
also support an architecture design but presents a difference if compared
with architectural patterns. Design patterns are smaller in scale than ar-
chitectural patterns. Design patterns are medium-scale patterns, while ar-
chitectural patterns are large-scale patterns. Design patterns tend to be
programming language independent, and they aim at influencing the archi-
tecture of components or subsystems, more than the structure of the entire
software system as architectural patterns do\cite{7}.

If available, GADesign advises architects to choose an architecture pattern as
a reference model to help constructing the initial functional decomposition. In
case that an architecture pattern is not available, still tactics can be identified
based on the quality attribute they address.

Our case study did not identify any exact architectural pattern applicable
to this situation. Architectural patterns work in architectures at a large scale,
where the idea of a complete software system is well defined. Our issue is that
we designed a component instead of a complete software system. Our component
in future will be a part of a system; however, it is difficult to identify an archi-
tectural pattern for a system that is unknown in advance. Although our design
does not fully comply the client-server pattern (because we are not designing the
server side), our work can be categorized as part of a client-server architecture;
where our component represents a client that requests and receives services from
Picsearch’s video platform, same that represents the server. Nevertheless, still
architectural tactics and design patterns were available to support our design;
therefore, we focused on identifying those that could support us to influence the
result of our quality attributes.
Table 4.2 presents the initial set of tactics that we found suitable to attain our significant quality attributes. As can be observed, some of these tactics have an abstract nature. Their selection was intentional in order to fit our case study in an initial stage when details were scarce. Nevertheless, once we started getting inside about the component’s architecture, more concrete tactics were introduced. To keep the order of appearance we decided not including this second set of tactics in Table 4.2, but they are described along with our explanation. For all cases, in Section 4.1.3 we explain in detail the function of each tactic.

<table>
<thead>
<tr>
<th>Tactic</th>
<th>Quality Attributes¹</th>
</tr>
</thead>
<tbody>
<tr>
<td>Provide common services</td>
<td>R X M S U</td>
</tr>
<tr>
<td>Separate interface from implementation</td>
<td>X</td>
</tr>
<tr>
<td>Hide information</td>
<td>X X</td>
</tr>
<tr>
<td>Anticipate expected changes</td>
<td>X</td>
</tr>
<tr>
<td>Develop product-specific interfaces to external components</td>
<td>X</td>
</tr>
<tr>
<td>Maintain data confidentiality</td>
<td>X</td>
</tr>
<tr>
<td>Use OSS rather than build(Reduces design &amp; development time)</td>
<td>X</td>
</tr>
<tr>
<td>Follow API design guidelines</td>
<td>X</td>
</tr>
</tbody>
</table>

Table 4.2: Tactics

¹ R=Reusability, M=Modifiability, S=Security, U=Understandability, Usefulness, Consistency.

As follows we describe the tactics shown in Table 4.2:

- **Provide common services**: A common service is a strategy to achieve semantic coherence. Semantic coherence concerns about the relationships that exist among responsibilities in a software system. The goal is to ensure that every component have a single purpose without excessive dependencies on separate components. Maintaining semantic coherence means achieving loosely coupled and highly cohesive components, but it goes further. It considers the context of change 9. Semantic coherence intends to keep highly cohesive and loosely coupled modules not only to support initial requirements, but it also encourages the architect to evaluate possible changes, and prepare the architecture design to support them. Further, retaking the concept of common services, their use promotes reusability. A loosely coupled component with a single responsibility has high chances to be reused because its functionality is very well defined and its low level of dependencies isolates it to some extent from the rest of the system. Thus another system
requiring this functionality can just reuse this piece of software. Moreover, common services also supports modifiability. Modifications performed to a service needs to be made only once, rather than in each module where the service is used. This also prevents the propagation of ripple effects because modifications related to one service must be contained inside the module that provides that service[9].

- **Separate interface from implementation:** Separating an interface from its implementation enables modifiability and testability. Firstly, clients calling an interface are kept unaware of a specific implementation. Consequently, the implementation can be modified or substituted without causing any impact to the clients of a given interface[30]. Secondly, interfaces enable test-driven development (TDD). Test can be defined based uniquely on the interface and not on the functionality that implements it. Having different implementations with the same interface allows substitution for various testing purposes[9, 72, 12].

- **Hide information:** Information hiding aims at decomposing the responsibilities of a module into smaller pieces, and choosing which information to make public or private. Interfaces are used to make the public responsibilities available. Elements made public experience very low chances of changing, while private elements exhibits the component’s inner workings and possibly they may suffer modifications. This tactic allows to isolate changes within a software component and prevent changes from propagating to others. Separate components relying directly on a module’s implementation details could cause disastrous ripple effects because at some point these implementation details may suffer modifications. This tactic is highly related to *anticipate expected changes*(see next tactic) because it considers those changes as the basis for decomposition[9, 12, 72, 42].

- **Anticipate expected changes:** Trying to anticipate to potential changes provides a way to evaluate a particular decomposition of responsibilities. The tactic intends to minimize the effects of changes, *i.e.*, for each change the proposed decomposition should limit the set of modules that need to be modified. This tactic is different from semantic coherence because it tries to minimize the impact of the changes rather than trying to introduce coherence within the component’s responsibility. This tactic in reality is difficult to implement due to it is very unlikely to anticipate to all possible changes in a complex software system[9].

- **Develop product-specific interfaces to external components:** Develop product-specific interfaces aims at reducing dependencies on external components[36]. In case that such external component changes, the modifications required would be contained by the system-specific interface.
• **Maintain data confidentiality:** Sensitive data should be protected from unauthorized access. Confidentiality is commonly achieved by encrypting data and communication links. By using encryption we prevent unauthorized access of confidential data[9].

• **Use OSS rather than build:** The use of open source software(OSS) can reduce considerably development time[36]. Further, popular OSS is widely tested by a large community of users. This supposes an advantage in terms of reliability. On the other hand, one of the most important disadvantages is the common lack of proper and clear documentation. Nevertheless, OSS still represents benefits in terms of productivity when building software systems.

• **Follow API design guidelines:** Bloch[12] describes good APIs with the following characteristics: 1) easy to learn and use even without documentation, 2) hard to misuse, 3) sufficiently powerful to satisfy requirements, 4) easy to extend and appropriate to audience. Thus in terms of quality attributes, these characteristics can be interpreted as understandability, consistency and usefulness. The following guidelines were introduced by expert software designers as means to design good APIs.

  – **An API should do one thing and do it well:** There must be a single responsibility associated to the API, and this responsibility must be clear and easy to explain[12].

  – **An API must provide sufficient functionality for the caller to achieve its task:** An API that provides insufficient functionality to complete certain task is considered incomplete[35].

  – **An API should be minimal, without imposing undue inconvenience on the caller:** The smaller an API is, the better it is to learn. The fewer types, methods and parameters an API includes, the easier it becomes to understand, remember and use correctly[35, 12].

  – **APIs should be designed from the perspective of the caller:** When designing an API it is important to involve stakeholders that are familiar with the expectations of the caller, because is the caller the API's end user. If possible, it is ideal to let the stakeholders get involved writing the methods’ signature[35].

  – **Make interfaces easy to use correctly and hard to use incorrectly:** The more that is known about the context of an API, the more strict in terms of policies the API should be. A careful design of types and parameters can lead to catch errors at compile time instead of being delayed until runtime[35, 56, 42].
Chapter 4. Architecture Design

Avoid using optional method parameters: When an optional parameter is not present in a call, the method must assign a default value that is defined as part of the method’s internals. This default behavior is not visible when reading the code and it may lead to confuse the caller when using the API. It is advised to make one method per behavior, instead of using optional parameters [42, 35].

Provide detailed debugging information: Debugging information permits the user to receive detailed programming feedback. Debugging information requires the API to provide descriptive messages and optionally codes when either a method executed has succeeded or failed [42].

Good APIs are ergonomic: Ergonomics is related to the study and creation of style guides that define naming conventions, code layout, documentation style and so on. A major part of ergonomics relates to consistency, e.g., an API is easier to learn and memorize when its methods always place parameters of a particular type in the same order. Consistency enables transference of learning, i.e., once a caller has learned part of an API, she/he has also learned much of the remainder [35, 12].

4.1.3 Create Software Architecture Sketch

The goal here is to apply the selected tactics to build a software architecture. To simplify this step GADesign introduces the principles of One Pattern At a Time and Design Integration Precedes Implementation proposed by Buschmann[15]. One Pattern At a Time aims at integrating the selected tactics one by one, with descending order of importance. This eases the design process since aspects of other, not yet applied tactics do not need to be considered.

Design Integration Precedes Implementation explains that architects should focus primarily on the architecture design and not on the interfaces details. Thus it is paramount to complete the architecture design and ensure that the constituent components cooperate harmoniously. Once this has been achieved, the architecture can focus on refining the architecture and define its respective interfaces. We followed this recommendation and organized this work to reflect that sequence of events. This section is uniquely dedicated to architecture design and not to implementation. Implementation details are presented in Section 4.5.

It is important to start describing the initial functional conceptualization we had for this component. Then gradually introduce more details about the procedure followed to refine our architecture. Figure 4.1 shows this initial conceptualization. It can be observed the interaction existing between the Android ap-
Application, our media access component, and Picsearch’s server. The component’s main goal is to abstract the interaction existing whenever an Android application requires to access the media services provided by Picsearch’s server.

**Figure 4.1:** Initial Functional Conceptualization

**Designing for Reusability**

Designing for reuse requires a previous analysis to identify that functionality that ideally should be contained in a common software entity. Reusable code become expensive to build considering the analysis and design it requires. Therefore, only selected functionality should be target of reuse. The functionality we implemented was a result of the analysis presented in Chapter 2.

Bass et al.\[9\] introduce *Provide common services* as a tactic that supports reusability. The goal is to contain frequently used services into a single unit that can be used by multiple callers. In our case study the common service is the access to the media capabilities provided by Picsearch. Diverse customers may require to access video from Android applications, thus it is important to offer them a component that will simplify the development of their applications.

Gamme 25] categorizes reuse in three levels. Classes are positioned at the lowest level of reuse. Classes are used by composition or inheritance. Frameworks present the highest level of reuse. They create abstractions to solve problems. Frameworks represent abstractions through the use of classes and the definition of relationships between them. Frameworks typically incorporates more than a single class, and their use is commonly by subclassing. A difference of class reuse, frameworks normally utilize inversion of control (IOC). In IOC user defined methods that customize the framework will often be called from within the framework itself, rather than from the user’s application code\[44\]. Thus the application activity is coordinated by the framework instead of the user’s application code. Hence,
IOC powers frameworks to serve as extensible skeletons. Moreover, in the middle of the scale, between classes and frameworks we find design patterns. Design patterns are smaller than frameworks, but they represent a higher abstraction level because they are design ideas and concepts rather than concrete code.

Based on the previous categorization, our work offers reuse at class level. This component delivers an application programming interface (API) that is actually the subject of reuse. The API provides access to a component’s functionality, and it describes with methods the services it enables. By providing a well-defined API it becomes clear to callers what functionality is public. Everything else remains private as part of the component’s internals. Hence, by using an interface to prevent the access to the component’s internals the architect can modify the component’s internal behavior without affecting external callers. Indeed, a change in the internal behavior implies that the component’s interface remains intact.

One of our tasks involved the API definition; however in this an early stage of the design process, GADesign advises to work with abstract entities instead of trying to define classes and methods. The goal is to simplify the design process by thinking in a sketch, i.e., a high level view. The concern here is to ensure that important functionality and quality attributes are already addressed in each of the abstract entities that conform the sketch. Therefore, at this early stage we only focused on ensuring that reusability was addressed in the initial sketch (see Figure 4.2). For this purpose we modeled the API as the entity that achieved reusability and published the component’s functionality.

**Defining Component’s Internal Entities**

Based on our functional requirements (see Section 4.1.1) and an analysis of Figure 4.1, we analyzed the internal responsibilities required to support the API. The identified responsibilities resulted into an initial definition of our component’s internal entities (see Figure 4.2). The following list presents the results:

1. **Encryption client**: The interaction with Picsearch’s server requires the client to authenticate with a valid username and password. Passwords are sensitive information that must be encrypted when sent via HTTP. The encryption algorithm must be SHA-1. The rationale for creating this entity relies on a functional requirement, but it also follows our tactic: *Maintain data confidentiality*.

2. **HTTP client**: A client is needed to handle the HTTP requests and responses that take part in the communication with Picsearch’s video platform.
3. **Response parser**: This parser makes a binding between JSON (JavaScript Object Notation) content and Java plain objects. The server’s response is made of JSON content, while our component’s output consists of Java objects. Thus the parser’s responsibility is to interpret that JSON content and convert it into Java objects.

4. **Error analyzer**: Analyzes the server’s response in order to find problems or incompatibilities with mobile devices. In some occasions, videos stored in the server might not be transcoded to formats compatible with mobile devices; therefore, it is needed to analyze the response and ensure that its content is complete and correct.

![Figure 4.2: Component’s Internal Entities - Conceptual View 1](image)

When the previous entities were identified and modeled we found a potential design fault. In Figure 4.2 it can be observed that the API communicates directly with the rest of the entities. This implies that in the API should exist some knowledge and business logic to coordinate the execution of the remaining entities. The problem with this situation is that it violates high cohesion and loosely coupling (semantic coherence) because 1) its unique responsibility is to publish functionality throughout methods, but not to handle a complex business logic, and 2) the more knowledge it has about other entities the more dependencies it experiences, and the more difficult it becomes to maintain and modify. Thus after analyzing this issue we realized that in order to keep the API’s semantic coherence we required to introduce an intermediate entity that would coordinate the interactions among the rest of the entities. More specifically, this intermediate entity was required to abstract the API from the remaining component’s entities.
Consequently, we searched in the literature for any design tactic that could help us to overcome this issue, giving this search as result: *Introduce a mediator.*

This tactic relies on the mediator design pattern’s goal. This pattern defines an object that abstracts how a separate set of objects interacts. The benefits of a mediator are: 1) achieve a loosely coupled design that keeps objects from referring to each other explicitly, and 2) ease modifications by containing the interaction or moderated logic inside a single object [26]. The only disadvantage of this tactic is that it breaks the principle of semantic coherence in the mediator. The mediator has a well defined responsibility, but it contains many references and dependencies to the entities it is moderating, i.e., it is highly coupled. This is a trade-off that had to be taken to keep a clean design for the API and the rest of our entities. Thus Figure 4.3 shows our architectural sketch after adding a mediator entity.

Bass *et al.* [9] define *Maintain semantic coherence* as a tactic that supports modifiability by creating highly cohesive and loosely coupled entities where modifications are localized within an entity. We applied *Maintain semantic coherence* when modeling and implementing each one of the entities described in the previous list. From 4.3 it is observed that there are five internal entities that correspond to the responsibilities previously defined, plus the API. Furthermore, Figure 4.4 is a sequence diagram that depicts the interaction existing among the component’s entities. Note well how the interaction functions both, internally and externally in a synchronous manner. We emphasize this first approach because later on we introduced some improvements to such execution paradigm.

![Figure 4.3: Component’s Internal Entities - Conceptual View 2](image-url)
Figure 4.4: Interaction View - Applying Synchronous Processing
Chapter 4. Architecture Design

Introducing Asynchronous Processing

Two important decisions when designing a component’s architecture are to decide the communication paradigm it uses 1) internally to coordinate the component’s internal execution, and 2) externally to handle the way the API responses to calls received from separate components. This communication paradigm can be either synchronous or asynchronous depending on the situation. From the execution sequence shown in Figure 4.4 it is perceived that both internally and externally our design relies on a synchronous paradigm. Externally the API method’s receive calls and return responses to the Android application in a synchronous way. Internally the component follows a synchronous execution by waiting every entity to complete before the next is called.

After having started with a internal and external synchronous approach we detected a potential improvement. This enhancement came up after finding that Android’s programming model advises developers to avoid involving the main thread, also called graphic user interface(GUI) thread, in the execution of long-running operations such as network access or database queries. The reason is that the main thread manages the interaction with the graphic user interface(GUI), and if it becomes busy for a long time, the GUI gets unresponsive until the thread completes its task. Even worse, in case the GUI thread gets blocked for more than a few seconds(about 5 seconds currently)[2], Android assumes the application is unresponsive due to some problem, and shows automatically a dialog indicating ”application not responding”. So for sake of GUI performance it is advised to spawn a background thread to deal with long-running operations. At this point we noticed that performance needed to be added to our list of significant quality attributes.

Our component pulls information from Picsearch’s server, so it involves continuous network access. Hence, if developers does not follow Android’s programming model advice, there is a potential possibility of getting unresponsive applications. Consequently we realized that designing an asynchronous external behavior would help to cope with these performance issues. By providing an asynchronous API the methods receive calls and immediately return the control of the execution to the caller, avoiding any unresponsive GUI. In terms of implementation, an asynchronous API would save developers effort by avoiding them spawning a background thread to call our synchronous API(see Figure 4.4). Thus motivated on simplifying developer’s work we decided to introduce Apply asynchronous processing as a tactic to support external asynchronous communication. The tactic talks about using a background thread to process long-running operations, dismissing the main thread(GUI thread) from the job, and allowing it to attend GUI user events[30, 9]. This tactic does not apply totally to the context of our work because we do not deal directly with a GUI, however we are trying to help developers by
reducing their efforts focused on providing a better user experience. So in order to use this tactic to provide an asynchronous external behavior we needed to create modifications to the component’s internal behavior to make it asynchronous as well. The use of the tactic requires the introduction of multithreading to our component’s inner workings. The inclusion of this design decision does not alter the conceptual view presented in Figure 4.3, however it does change Figure 4.4 in the way callers interact with our component. Further, Figure 4.5 shows a second interaction view that introduces asynchronous processing.

Figure 4.5 shows in step 2 that the API calls Mediator. This call is made by the main thread and it is here when the main thread creates a background thread to continue the component’s execution. Step 3 and 4 are executed by the main thread, where the control is returned to the Android application to keep the GUI responsive. On the other hand, the background thread that was previously created waits until it receives a CPU time slice to execute its task. From step 5 to step 15 all tasks are executed by the background thread. In step 15, still using a background thread, Mediator requests to a callback interface to take control of the execution, and also provides the data obtained from Picsearch’s server. This request needs to be handled in such a way where the main thread pauses its current task to receive the result of our component’s execution. In other words, the background thread queues the call to the callback interface as a job for the main thread. The reasoning behind this is the design of the Android GUI toolkit. The Android GUI works under a single-thread model, i.e., it is not thread-safe and must always be manipulated by the main thread[2]. Thus a violation to this policy results in an exception thrown by Android. Moreover, the data returned by our component is used to update the GUI in some manner, thus in order to ease developers’ job, it is a good idea for our component to report the resulted data directly to the main thread.

The asynchronous alternative presents advantages to developers in terms of performance and simplicity, however it also has drawbacks. Some developers may not feel comfortable at coding under inversion of control; therefore, the fact of using a callback interface may discourage them to use our library. Another possibility is that a developer may prefer to handle the multithreading issues manually, thus our component could be unsuitable in that scenario. Because of these possibilities, and in order to provide flexibility to developers we decided to provide APIs to work under both paradigms: synchronous and asynchronous. We expect that by doing this everyone takes advantage of our component in the most convenient manner.
Figure 4.5: Interaction View - Applying Asynchronous Processing
In terms of design, to this point our conceptualization adjusted fairly well to support both paradigms. Nevertheless, we realized that our component’s implementation would require special considerations in terms of the API methods’ signatures and the mechanism in which the mediator would handle the multithreading execution. Being this section part of the design phase, the functionality was only modeled as shown in Figure 4.5 but we did not go into detail until the implementation phase (see Section 4.5).

4.2 Software Architecture Early Evaluation

GADesign establishes an early evaluation aimed at reasoning about the architecture’s ability to satisfy functional requirements and quality attributes. The method does not provide an explicit guideline to perform the evaluation; nevertheless, it provides a very useful checklist to go through in order to review the completeness of an architecture. The checklist contains the following points [71]:

1. The key components and their connections have been identified and well defined.

2. The architecture satisfies all influential functional requirements.

3. One interaction diagram exists at least for each architecturally significant use case.

4. Each significant quality attribute is addressed in the SAS (software architecture sketch) model.

The key components and their connections have been identified and well defined. Our architecture is based on just one component, and internally it is composed by five entities and an API. Each composing element was defined and described in Section 4.1.3. The connections and interactions among components were depicted in Figures 4.4 and 4.5 by our interaction diagrams. Our API and all the entities that take part within our component were modeled by applying semantic coherence; therefore, they experience a single, well-defined responsibility. Thus we believe our architecture successfully covers this checkpoint.

The architecture satisfies all influential functional requirements. Our component’s significant functional requirements were defined in Section 4.1.1. Not all of these requirements are observable at design level. Some of the functionality explained must be delivered throughout the API’s methods, thus at this stage of the design process the API had not been defined yet. Therefore that functionality was not evaluated but until the end of the implementation process. On the other hand, functionality as the HTTP client used to access the REST API, the
password encrypting client, and the parser to convert JSON content into Java objects are explained in Section 4.1.3, and can be clearly identified in Figure 4.3. Excepting the functionality provided by the API’s methods, the rest of the functional requirements were identified in our design as satisfied.

**One interaction diagram exists at least for each architecturally significant use case.** Our component requires the evaluation of two scenarios: when the API is used 1) synchronously, and 2) asynchronously. The sequence diagrams for those two scenarios are presented in Figures 4.4 and 4.5. In terms of interaction, the differences in each diagram rely on 1) how the API is called externally, and 2) how the API interacts with Mediator. More than those differences, the internal flow of each sequence diagram follows a very similar behavior regardless which API’s method is executed. Thus, although we had not implemented the API, we verified that our component’s internal entities were interacting in the proper manner as to support the execution of any API’s methods.

**Each significant quality attribute is addressed in the SAS(software architecture sketch) model.** Table 4.1 in Section 4.1.1 summarizes our component’s most significant quality attributes. Reusability, modifiability and security were described in Section 4.1.3. Later on we realized performance was another significant quality attribute, therefore the component’s asynchronous mode was introduced.

Because these quality attributes are evaluated based on an API’s implementation, understandability, usefulness and consistency were obvious until the API was defined (see Section 4.3.1). Therefore, despite of these quality attributes being so critical for our component, we could not address them on architecture blocks. Hence, we retake understandability, usefulness and consistency in Section 4.3.1 with the definition of the API.

### 4.3 Software Architecture Refinement

A SAS(software architecture sketch) model is mainly focused on coarse-grained design, and the goal of the software architecture refinement phase is to ground that coarse-grained design into technology details such as a precise definition of the component communication protocol and the interfaces of each component. The result of the refinement phase is a more detail model called SAD (Software Architecture Definition).

The software architecture refinement phase involves three forms of refinement: 1) component refinement, 2) connector refinement and 3) interface refinement.
Component refinement strives for decomposing high-level components into concrete and smaller ones, in order to ease their implementation. Connector refinement’s focus is about refining how components are interconnected. Lastly, interface refinement intends to define the component’s interface with strict operation signatures.

This case study presents only a single component, thus the component refinement process follows a 1 to 1 relationship from SAS to SAD. Each of the internal entities that conform our component were mapped to one sub-component (see Figure 4.6). Nevertheless we could apply interface refinement to define our API, and the interfaces of our sub-components. Moreover, we applied connector refinement in order to design how our sub-components were interconnecting. In the end of the refinement phase we had a better understanding of our component including its API, the interfaces that constituted each of its sub-components and how the sub-components were interconnected through their interfaces. More detail is presented in the following subsections.

Figure 4.6: Component Internal View

4.3.1 Interfaces Definition

API Definition

Listing 4.1 presents the asynchronous API designed for our component. To simplify our explanation we decided only to include one API mode because we consider it is enough to make clear our point. Nevertheless, Appendix B presents the complete API documentation provided to Picsearch. This documentation provides details about both APIs, their methods, and the parameters that each method receives.
Listing 4.1: Asynchronous Application Programming Interface

```java
public abstract class AsynchronousAPI {

    public void getMediaContent(S9Commons commons, String mediaId);

    public void listAllMedia(S9Commons commons, MediaFieldsEnum[] fields, APIOrderEnum order);

    public void listMedia(S9Commons commons, MediaFieldsEnum[] fields, APIOrderEnum order, int offset, int maxResults);

    public void listFilteredMedia(S9Commons commons, MediaFieldsEnum[] fields, APIOrderEnum order, int offset, int maxResults, HashMap<MediaFieldsEnum, String> filters);

    public void listAllCategories(S9Commons commons);

    public void searchMedia(S9Commons commons, String searchText);

    public void searchMedia(S9Commons commons, String searchText, MediaFieldsEnum[] fields, APIOrderEnum order, int offset, int maxResults, HashMap<MediaFieldsEnum, String> filters, MediaFieldsEnum[] constraints);

    public void getUploadURL(S9Commons commons);

    public void uploadMedia(S9Commons commons, String uploadURL, String filePath, S9UploadParameters mediaParams);

    public void assignRating(S9Commons commons, String mediaId, int rating);

    public void getUserProfiles(String username, String password, S9CallbackHandler callbackHandler);

    public void setModeration(S9Commons commons, String mediaId, ModerationEnum moderationValue);

}
```

Designing for Understandability, Usefulness & Consistency

To design our API we followed the guidelines presented in Section 4.1.2. As follows we describe our considerations for each guideline:
An API should do one thing and do it well: As we explained previously, our two APIs have a single and well defined responsibility: enable Picsearch media services for the Android platform. Listing 4.1 shows how the methods defined are consistent with the functional requirements described in Section 4.1.1.

An API must provide sufficient functionality for the caller to achieve its task: Complete functionality is essential for the success of our component. Our API offers a caller the essential operations required for handling videos: uploading, streaming, listing, searching, rating and moderating. Thus being confident that the study presented in Chapter 2 produced positive results, we believe that our component addresses usefulness by allowing the caller to perform a full interaction with Picsearch’s video platform.

An API should be minimal, without imposing undue inconvenience on the caller: One of our main goals when we elicited our functional requirements was to choose that functionality that was really needed and useful for a caller. To a high extent our API was driven by the functionality that Picsearch’s REST API was able to delivered. Fortunately Picsearch already counted with a highly refined API that focused on the essential operations required to handle media content. However, our market study also provided important functionality that was required and not yet supported by the REST API, so it needed to be approved by the company and included to the video platform. The result (see Listing 4.1) is a set of methods that enable a developer a full interaction with the video platform.

It is arguable that some of our methods provide the same functionality, e.g., listAllMedia(), listMedia() and listFilteredMedia() so the API can be potentially reduced. However, our rationale to do this was to facilitate the job of developers for using the appropriate method for their needs. For instance, a caller that simply needs a full list of existing media can just call listAllMedia and provide three arguments instead of calling listFilteredMedia where six parameters are required and filtering is not needed. For the latter case the caller would be assigning null values for these three non-required parameters, which goes against to one of our guidelines: Avoid using optional method parameters. Thus we believe we included in our API an easily manageable number of methods that provides complete functionality, and tailors the methods according to developers needs.

APIs should be designed from the perspective of the caller: During the requirements elicitation we interviewed Picsearch’s personnel in order to get inside about the perspective of a possible component’s user. Furthermore, with Picsearch’s CEO support we contacted a customer in order get their opinion about our API’s understandability, usefulness & consistency. Unfortunately, at the time of writing this document we did not received any feedback. The bottom
line is that to the best of our possibilities we tried to take the caller’s stand point when eliciting our requirements and designing the component’s API.

**Make interfaces easy to use correctly and hard to use incorrectly:** When writing the methods’ signatures we were careful to define strict parameters types to reduce the possibility of confusion or unintentional misuse. If the reader revise 4.1 there is no ambiguity for the parameters definition. Enum types as MediaFieldsEnum or ModerationEnum were specially created to handle constants, and with the purpose of forcing the caller to use exactly one of the expected values. The rest of the parameters with the exception of S9Commons (see Section 4.5.2) are common concrete classes and primitives, e.g., String, HashMap or int. Furthermore, the parameters of HashMap type were defined as to include enum types as key values, thus the risk of ambiguity for selecting a value is eliminated. Finally, default values for optional parameters were well defined in the documentation presented in Appendix B. With all these elements included with our component we believe that the risks of confusion or misuse are reduced to a high extent.

**Avoid using optional method parameters:** As we explain previously, some of our methods as the case of listAllMedia(), listMedia() and listFilteredMedia() present the same functionality: retrieving a list of video items. The goal was to provide the caller three different ways of listing videos depending on the needs, instead of forcing them to use a single method with optional parameters that enable or disable optional functionality, e.g., filtering media. In listAllMedia, the method’s signature does not include a filter, so the method brings everything it finds in the video platform. For listFilteredMedia a filter is required to retrieve a more specific list of results. By providing tailored methods with a well defined purpose the designer avoid utilizing optional parameters to cover additional functionality. Moreover, the fact of avoiding optional parameters reduces the ambiguity to understand of functionality of the method. Although we tried to avoid optional parameters by providing different versions of a method, sometimes it is impossible to completely dismiss their use. Therefore, to reduce any possible impact we tried to provide a clear documentation about the behavior of a method when either optional parameters are provided or omitted.

**Provide detailed debugging information:** Both API versions, the synchronous and the asynchronous API provide as a result an object of the class S9ResultData. S9ResultData contains the result obtained from the video platform. Internally S9ResultData holds an object of the class S9ErrorInfo. This object, as can be noticed in Listing 4.2 keeps track of the error message, error code, and exception derived from the server response, or the component’s possible exceptions. Additionally we created ErrorsEnum in order to define common possible errors an
messages. The goal was to notice the caller about an unsuccessful execution, and try to provide as detailed information as possible.

**Listing 4.2:** Component’s Debug Information - S9ErrorInfo

```java
public class S9ErrorInfo {
    private int errorCode;
    private String errorMessage;
    private Exception exception;
    private ErrorsEnum errorEnum;

    public S9ErrorInfo(ErrorsEnum error, Exception exception) {
        // Internal implementation
    }

    public S9ErrorInfo(int errorCode, String errorMessage, Exception exception) {
        // Internal implementation
    }

    // Getters and setters
}
```

**Good APIs are ergonomic:** When we wrote the methods’ signatures we took care about the consistency in the naming, parameters type and ordering. Listing 4.1 shows how every method sharing the same or similar parameters types, also shares the same ordering, naming and type of such parameters. As well, the methods’ naming try to describe the functionality delivered, e.g., uploadMedia or assignRating. Further, in the case of methods experiencing similar functionality, e.g., listAllMedia, listMedia, listFilteredMedia, the naming was consistently kept to reflect the similarity.

**Mediator Interface Definition**

Listing 4.3 presents the Mediator interface. The purpose of this interface is to abstract the component’s internal flow when an API’s method is executed. The API assigns the mediator the responsibility of handling the execution of a given method. The goal is to keep the API and the rest of the internal sub-components unaware of the existence of other sub-components in order to eliminate dependencies. Thus the API simply calls Mediator’s executeMethod() and the *magic* is done behind the scenes. It can be appreciated that our Mediator’s interface shows a single responsibility: orchestrate the interaction with the rest of the sub-components in order to execute an API method’s functionality. Thus it can be say that it experiences cohesion; nevertheless, it is not loosely coupled because it holds dependencies to the rest of the sub-components.
**Listing 4.3:** Mediator Interface

```java
public interface Mediator {
    public S9ResultData executeMethod(SdkAPIMethodsEnum methodSdk, 
                                       RestAPIMethodsEnum methodRest, 
                                       HashMap<APIParamsEnum, Object> parameters, 
                                       HashMap<String, Object> extras);
}
```

**Security Client Interface Definition**

Listing 4.4 presents the SecurityClient interface. Note well that as interface we refer to the access point or mechanism to interact with the sub-component and not to the interface Java type. Then, our interface provides a single responsibility: to encrypt a String value with the SHA-1 algorithm and return it as response. Further, it does not keep dependencies to the mediator or any other sub-component. Hence, SecurityClient attains semantic coherence.

```java
public class SecurityClient {
    public String encryptToSHA1(String strToEncrypt) {
        // internal implementation
    }
}
```

**HTTP Client Interface Definition**

Listing 4.5 shows the HttpClient interface. The responsibility of this interface is to handle the HTTP connection to Picsearch’s video platform. The HTTP connection involves sending a request and obtaining a response from the server. All methods excepting uploadMedia() utilize HTTP GET as means to connect to the video platform. Only uploadMedia() uses HTTP POST in order to send the media file as `multipart/form-data`. 
Chapter 4.  Architecture Design

Listing 4.5: HttpClient Interface

```java
public class HttpClient {
    public String executeGETRequest(RestAPIMethodsEnum methodRestAPI,
                                     HashMap<APIParamsEnum, Object> parameters,
                                     HashMap<String, Object> extras)
    {
        // Internal implementation
    }
    public String executePOSTRequest(RestAPIMethodsEnum methodRestAPI,
                                       HashMap<APIParamsEnum, Object> parameters,
                                       HashMap<String, Object> extras)
    {
        // Internal implementation
    }
}
```

Response Parser Interface Definition

The interface presented in Listing 4.6 has as unique responsibility converting the JSON content obtained from the video platform into plain Java objects. This conversion involves a parsing process from JSON to Java. The JSON content can variate depending on the API’s method that has been executed. Therefore, parseJsonResponse() receives as parameter responseClass, which indicates the class type of the object that must be returned. Note well that responseClass must be a sub-class of S9Response (extends S9Response). This is a constraint that helps us to force that the returned object belongs to a S9Response sub-class.

```
Listing 4.6: Response Parser Interface

```java
public class JsonResponseParser {
    public S9Response parseJsonResponse(String jsonContent,
                                          Class<? extends S9Response> responseClass)
    {
        // Internal implementation
    }
}
```

Error Analyzer Interface Definition

Listing 4.7 shows the Error Analyzer’s interface. Its responsibility consists in analyzing the plain Java objects generated by Response Parser in order to find errors or missing values in the object’s state. Note that Error Analyzer ignores the existence of Response Parser; there is no direct communication among these two sub-components. The entire coordination among sub-components is performed by
Mediator, thus the dependencies within the component are eliminated. As a result of `analyzeErrors()`, the method creates a `S9ErrorInfo` object. This object provides debug information about a problem that may have occurred. The structure of `S9ErrorInfo` was already presented in Listing 4.2.

### Listing 4.7: Error Analyzer Interface

```java
public class ErrorAnalyzer {
    public S9ErrorInfo analyzeErrors(S9Response response, SdkAPIMethodsEnum methodSdk) {
        // Internal implementation
    }
}
```

### Component’s Internal View Refined

Figure 4.7 presents a refined view of the component that summarizes the methods defined in each sub-component’s interface for supporting the execution of an API’s method.

![Component Internal View Refined](image)

*Figure 4.7: Component Internal View Refined*
Chapter 4. Architecture Design

4.4 Software Architecture Middle Evaluation

The software architecture middle evaluation intends to ensure consistency between SAS (Software Architecture Sketch) and SAD (Software Architecture Definition) models. The goal is to check that the information presented in the SAS model is preserved and consistent in the SAD model.

Our architecture refinement phase (see Section 4.3) was relatively simple because our architecture only has one component. Hence, the component mapping process SAS to SAD was 1:1. Furthermore, the internal entities described in Section 4.1.3 also followed a 1:1 mapping relationship to become sub-components in our SAD model. Thus through this evaluation we could verify that the elements considered in our SAS model were still present and refined in our SAD model. Moreover, we validated that our SAD model kept satisfying our functional requirements and quality attributes.

4.5 Architecture Transformation Into Implementation

The goal of this phase is to transform SAD models into platform specific models, grounding the higher-level specification into a lower-level implementation.

4.5.1 Introducing Open Source Software (OSS)

One of the tactics we introduced in Section 4.1.2 was Use OSS rather than build. The goal of this strategy is to increase productivity by reducing development time. Further, popular OSS tends to be reliable because it has behind a large amount of people using it and testing it; therefore, critical bugs are quickly found and also fixed almost immediately.

During the implementation of our component we decided to use two open source components: 1) Apache HTTP-MIME[3], and 2) Jackson JSON-Processor[41]. The first OSS component provides support for MIME multipart encoded entities. Android already includes an HTTP client, however it does not handle the multipart mime type. This type is needed in order to upload binary data by HTTP, as is the case of our upload media functionality. This component was called within our HTTP Client sub-component. The second OSS component provides functionality to process JSON content and use it to create Java objects. This second component was used in our Response Parser sub-component for parsing the JSON server response into plain Java objects.
Changeability in these OSS components is not expected in the short term, \textit{i.e.}, they are not expected to be replaced for a new version, different provider or changed in any form. Nevertheless, it is always important to try to reduce dependencies to external components in order to avoid any risk of change. To achieve this we used: \textit{Develop product-specific interfaces to external components}. This tactic explains that when an external component is introduced, its use should be abstracted through an interface in order to reduce dependencies and allow future changeability. We followed this recommendation by setting our own interfaces for both, our HTTP Client and Response Parser sub-components. They internally instantiate the already mentioned OSS components, so by setting our own interfaces we protect Mediator from ripple effects that could be originated from changes in those external OSS components. The designed interfaces are shown in Section 4.3.1. By using these interfaces changeability is more easily attained because modifications must be performed only within the sub-component, while the rest of the sub-components remain intact.

### 4.5.2 Component Implementation Details

Figure 4.8 shows a class diagram that depicts our component’s base implementation. As follows we explore some design decisions made during the implementation process.

**Implementing the Application Programming Interfaces\(^{(APIs)}\)**

Starting from the top of Figure 4.8, we can observe the hierarchy of our APIs. Both APIs, the SynchronousAPI and AsynchronousAPI mostly mirror each other with some small but important differences:

1. **Callback interface**: All the asynchronous methods receive as parameter an object of S9CallbackHandler. With the exception of getUserProfiles(), the callback interface instance is contained inside of a S9Commons (see Listing 4.8) parameter. The structure of the S9CallbackHandler class is shown in Listing 4.9.

2. **Return type**: The asynchronous version declare methods where their signature is \texttt{void}, \textit{i.e.}, they do not return anything. On the other hand, the synchronous version’s methods return an object that extends S9ResultData. The asynchronous version utilizes the callback interface as mechanism to provide the S9ResultData to the caller. In Listing B.3 can be noticed how the method processCallback receives as parameter the S9ResultData object.
Figure 4.8: API Class Diagram
**Listing 4.8:** S9Commons Implementation

```java
public class S9Commons {
  private String authenticationToken;
  private S9CallbackHandler callbackHandler;

  public S9Commons(String authenticationToken, S9CallbackHandler callbackHandler) {
    // Internal implementation
  }
  // Getters and setters
}
```

**Listing 4.9:** Callback Interface Implementation

```java
public interface S9CallbackHandler {
  public abstract void processCallback(S9ResultData data);
}
```

Moreover, the APIs class structure was justified on three tactics: 1) *Separate interface from implementation*, 2) *Hide information*, and 3) *Anticipate to expected changes*.

*Separate interface from implementation* talks about defining a super type as interface, and extending this super type as implementation. In other words, it is about creating a polymorphic relation between two entities. The super type uniquely defines the signature of the methods, and the sub type implements the behavior. Ideally the super type is an abstract type as a Java interface\(^1\) or an abstract class. An abstract class in Java is as its name indicates an abstract type, and one of its characteristics is that is allows to include implementation if required, though it cannot be instantiated. On the other hand, and interface is a pure abstract type and does not allow to include any implementation on it. Then, we decided to utilize an abstract class because we wanted to use our API as a factory of its implementation, *i.e.*, enable a getInstance() static method in the API to return an instance of its sub-class. The reasoning behind is to avoid that developers could directly instantiate the sub-class. By avoiding this we ease the future component’s maintenance because the implementation class can be changed, and as long as the API remains intact, callers will not notice the difference or experience any ripple effect. Listing 4.10 shows the concept.

---

\(^1\)Note well that in our context the term *"interface"* can express two ideas: 1) An interface as the facade that is used to access the functionality of a software component, *e.g.*, an API, and 2) an interface as the abstract type defined in the Java language. At some point those definitions may get related, but in reality they express two different ideas.
Chapter 4. Architecture Design

Listing 4.10: Abstracting Concrete Instantiation within the API

```java
public abstract class SynchronousAPI {
    public static SynchronousAPI getInstance() {
        return new SynchronousAPIImpl();
    }
    // Rest of the API
}
```

From Listing 4.10 it can be observed that our API abstracts the instantiation of its concrete class, SynchronousAPIImpl in this case. Hence, the caller of getInstance() does not know to which class belongs the returned instance. Only one thing is known, the returned instance extends SynchronousAPI, and this can be seen in the method’s returned type. This concept is also included in the asynchronous API.

The previous concept was motivated by other two tactics: *Hide information* and *Anticipate expected changes*. *Hide information* advises to decompose the functionality of a software entity in order to decide which information can be made public and which should be kept private. The goal is to hide from callers those pieces of information where changeability is expected. *Anticipate expected changes* supports precisely on helping to detect those pieces prone to change. The latter tactic advises to anticipate to potential changes in order to minimize their effects.

In our case study we see highly possible that changes will be performed to the component’s implementation once Picsearch becomes in charge of its maintenance. Therefore, for us it is very important that callers do not access neither the API’s implementation classes(SynchronousAPIImpl and AsynchronousAPIImpl), nor any other class taking part in the component’s internal implementation. Thus our solution is to access the components implementation indirectly through the getInstance() method declared in each API. Then, by combining the advice of the previous three tactics we attain a good degree of isolation to hold those modifications, and also we protect callers from suffering ripple effects.

Lastly, SynchronousAPIImpl and AsynchronousAPIImpl behave in a highly similar manner, hence, it becomes logic to share common functionality. Therefore, in order to avoid code duplication, we decided to place that common functionality in the class APICommonCode.
Mediator Implementation Details

A mediator was introduced to isolate the component’s inner workings from the API. A mediator is responsible of organizing the way internal sub-components are called to deliver the expected functionality. The Mediator’s interface is shown in Figure 4.8.

Because the synchronous and the asynchronous variants, the Mediator’s interface is implemented twice. AsynchronousMediator does not present any implementation details important to highlight. Nevertheless, in AsynchronousMediator there are some hidden details that reveal the asynchronous behavior.

The Android SDK offers several mechanisms\cite{21} to deal with asynchronous behavior in order to avoid unresponsive GUIs. One of these mechanisms is AsyncTask\cite{19}. AsyncTask is an Android class that handles automatically asynchronous executions where long-running operations require to be executed in a background thread instead of in the main thread. When the long-running operation is done, AsyncTask queues a new job in the main thread’s list of tasks. This job requires the main thread to process the result obtained by the background thread when the latter has finished processing that long-running operation. Thus AsyncTask abstracts the complexity existing when the main thread and a background thread require to interact, simplifying the job of developers. Consequently this class reduces practically all the complexity related to an asynchronous design. The functionality delivered by AsyncTask is accessed by inheritance, and by design AsyncTask obligates callers to use the main thread to start the asynchronous execution.

Our asynchronous execution relies on AsyncTask to handle the network communication. Our AsynchronousMediator class implements the Mediator interface and extends AsyncTask. Similarly to our APIs implementations, AsynchronousMediator and SynchronousMediator experience some similar behavior. Therefore, to avoid duplication and ease maintenance we placed the common behavior inside the MediatorCommonCode class. As can be seen in Figure 4.8, the MediatorCommonCode class is the responsible of coordinating the component’s execution by managing the calls to the classes that deliver the functionality required to execute each API’s method.
4.5.3 Thread Safety in Multithreading Environments

Android is by nature a multithreading operating system, and consequently it offers built-in support to develop multithreading applications. Therefore, it was critical to define the component’s expected behavior under a multithreading execution.

Our component by design is stateless, i.e., it does not hold any value or state along its execution. This characteristic makes it thread-safe, i.e., thread-safeness allows an object to be used by multiple threads at the same time without experiencing problems or strange behaviors. Nevertheless, there is a limitation associated when the component is used in an asynchronous mode. Section 4.5.2 explains that when the asynchronous mode is used, AsyncTask is internally used to handle the execution. AsyncTask’s asynchronous behavior relies on the execution of its doInBackground() method, and by design it only allows the main thread to call this method. Any omission of this rule results in a runtime exception. Thus this limitation is then transferred to our component. Therefore, if the asynchronous API is used (see Listing 4.11), callers must be careful to use the main thread to execute our API’s asynchronous methods.

Listing 4.11: Asynchronous API Use

```java
AsynchronousAPI api = AsynchronousAPI.getInstance();
api.getUserProfiles(username, password, new S9CallbackHandler() {
    public void processCallback(S9ResultData data) {
        Screen9UserProfileResponse response =
            (Screen9UserProfileResponse) data.getResponse();
        // Process response
    }
});
```

On the other hand, when the synchronous API is used (see Listing 4.12) the previous restriction does not apply. Any thread can call the method without causing negative effects. The only advice we give is that when calling the synchronous API, a manual asynchronous mechanism should be used in order to avoid GUI unresponsiveness. We exemplify this advice in Listing 4.12. The call to the API is performed by a background thread and scheduled through a Handler[20]. Handler is another mechanism offered by Android to deal with asynchronous behavior.
Listing 4.12: Synchronous API Use

```java
final SynchronousAPI api = SynchronousAPI.getInstance();
Handler handler = new Handler();
handler.post(new Runnable()
{
    public void run()
    {
        S9ResultData data = api.getUserProfiles(username, password);
        Screen9UserProfileResponse response =
            (Screen9UserProfileResponse) data.getResponse();
        // Process response
    }
});
```

We believe that the limitation associated to our asynchronous API does not represent a problem because the component is perfectly capable to function well under the normal expected usage. With normal expected usage we visualize two scenarios: 1) a developer uses the asynchronous API through a call initiated by the main thread, and 2) a developer creates a background thread to call our synchronous API. If a problem raised, it would be most probably due to a developer trying to call the asynchronous API from a different thread other than the main thread. This would mean that the developer intends to handle concurrency manually, and if this is true, then the best option would be to use the synchronous API that does not impose any restriction.

4.6 Software Architecture Post-deployment Evaluation

This evaluation occurs once the architecture has been implemented and deployed. The purpose of it is to assess significant quality attributes directly when running the software.

In our case study we evaluated our component through two Android applications. Figure 4.9 shows the screenshots of our applications. The right screenshot corresponds to a prototype that we used along the implementation process. By testing our component with this prototype we could 1) experience the ”feeling” of using the API and 2) verify that the component’s functionality was correct. This prototype was the base of an iterative process for finding component’s weaknesses and fixing them. Such iterations were the base to refine our component until the point that it satisfied its functional and non-functional requirements.

On the other hand, the left screenshot shown in Figure 4.9 is a fully functional application inspired on the journalism scenario presented in Section 2.1.4. Picsearch asked us to create a proof of concept where our component was used. The
goal was to evaluate the component’s capabilities from a functional perspective. Moreover, this proof of concept is expected to be used as demo to show how the video platform is suitable to interact with video mobile applications.

Finally, our component as much as our applications satisfied their respective goals. Nevertheless, there is always room for improvement, and our component is not an exception. We believed that with more iterations the class structure could be more elegant. However, a Master’s thesis has a limited time, and close to finalize the collaboration with Picsearch our results were satisfactory for both parts, Picsearch and us, which can be considered as a measure of success. Chapter 5 presents the results of several interviews that we performed internally at Picsearch to evaluate our work. The goal was to survey the personnel involved in the video platform about: 1) their opinions about our results, and 2) possible improvements.

![Figure 4.9: Applications Developed as Post-deployment Evaluation](image)
Chapter 5

Evaluation

Chapter 5 presents a summary of a set of interviews performed to the personnel of Picsearch. The interviews aimed at expressing the opinion of Picsearch’s personnel about the extent in which our results satisfy our functional requirements and quality attributes. The interviews followed a semi-structured approach and used as guide the questionnaire presented in Section 5.1. Further, our results are presented in Section 5.2. The personnel interviewed were:

1. Robert Risberg: Chief Executive Officer (CEO) at Picsearch. Mr. Risberg is highly involved on customer relationships and has a deep understanding about the functionality that needs to be delivered to keep the company in constant competition. We interviewed him to know his opinion about the functionality delivered by our API. Because his low involvement with technical matters, Mr. Risberg was not enquired with questions that required a deep technical knowledge.

2. Sean Atkinson: Chief Technical Officer (CTO). Mr. Atkinson is the leader of the team that supports the video platform. His opinion was quite important because the deep knowledge he possesses about the functional specifications of the video platform, as well as the technical elements that support it.

3. Johan Rönnblom: Software Engineer. Mr. Rönnblom is the responsible for designing, developing and maintaining the REST API. He expressed as a concern his participation as interviewee due to his deep involvement with the design of the REST API and the fact that our API mirrors its functionality. In his words this could bias him about giving a positive opinion. We agreed that this is a valid threat to this evaluation. Nevertheless, we decided to go on with the interview because his experience could offer advantages in identifying important inconsistencies or problems within our API.

Note: To prepare our interviewees we sent them: 1) the documentation of our API, 2) the UML class diagram of our component, and 3) the component’s source code.
5.1 Questionnaire

This section presents the questionnaire that was utilized to guide our interviews. The questionnaire covers those important quality attributes that were identified in our architecture design. Furthermore, the questionnaire discusses the opinion of the interviewees about the documentation delivered with the API, as well as future ideas to extend this Master’s thesis.

1. **Usefulness, Understandability & Consistency:**

   - What is your opinion about the usefulness of the methods included in the APIs?
     - Is there some critical functionality missing?
     - Is there unimportant functionality included?
   
   - Using the following scale, how would you rate the usefulness of the APIs?
     1 extremely useless
     2 very useless
     3 useless
     4 fair
     5 useful
     6 very useful
     7 extremely useful
   
   - What is your opinion about the completeness of the APIs considering the needs of an average Picsearch customer?
   
   - Considering the information provided by the callback interface (server response, called method and error occurred):
     - Is this enough information to control the callback?
     - Is there some missing data?
   
   - Do you consider it an advantage that the APIs offer both a synchronous and an asynchronous mode to satisfy diverse developer preferences?
   
   - Except for the additional maintenance involved, do you see any other disadvantage with having these two APIs?
   
   - Using the following scale, how would you rate how easy (intuitive, understandable) are the APIs to use? Please explain:
     1 extremely hard
     2 very hard
     3 hard
Chapter 5. Evaluation

4 fair
5 easy
6 very easy
7 extremely easy

- What are the conceptual difficulties in understanding the API in your opinion?
- How would you describe the consistency of the API in terms of naming, arguments used, arguments ordering and returned values?
- Using the following scale, how would you rate the consistency of the API?
  1 extremely inconsistent
  2 very inconsistent
  3 inconsistent
  4 fair
  5 consistent
  6 very consistent
  7 extremely consistent

2. Documentation:

- Is it clear and useful?
- What is missing (more/less examples, explanation etc.)?

3. Reusability:

- To which extent does our component permit reusability in Android applications? Explain any limitation you perceive.

4. Modifiability:

- Considering the current design of the API and the internal structure, how easy or complex could be to modify it or maintain it?
- Do you encounter/foresee any potential problem with respect to extending or modifying the current API?

5. Security:

- Do you consider that a more refined approach to handle security in the API is needed?
- Is there some other data in addition to passwords that should be encrypted and is not currently done?
• Would you use SSL or do you consider it necessary at some point?

6. Additional comments not covered in the previous questions:

• Overall impressions of the API
• Any new ideas for additional REST API functionality inspired by this library/the promise of smart phones in general

5.2 Results

5.2.1 Usefulness, Understandability & Consistency:

Usefulness

The interviewees considered that our API was primarily useful because: 1) it saves development time, 2) it was inspired on functionality required to fulfil the market scenarios presented in Section 2.1.4, and 3) because the application proof-of-concept demonstrates its capabilities.

Mr. Atkinson opined that in terms of functionality "this is the right balance of what is needed for a specific application", thus the list of included methods offer the adequate functionality that an average customer needs. It does not miss important functionality, but neither include unimportant one. Mr. Atkinson exemplified with his experience when he participated in the design of the XML-RPC API. The XML-RPC API is the most extensive API that Picsearch offers. He mentioned that many assumptions about customers needs were made when the methods were defined, but unfortunately many of those assumptions were wrong. This resulted in a long API with several methods that experience a low usage.

Figure 5.1 presents the usefulness ratings given by our interviewees. The measurement scale goes from 1 that indicates the API is extremely useless, to 7 which denotes an extremely useful approach.
The previous values were provided by our interviewees with base on their experience working at Picsearch. They also emphasized that their opinion may not be the optimal considering that they did not use our component to develop a practical example. This was a validity threat to our evaluation that we foresaw during the design process. Therefore we decided to distribute a stable version of our component with its documentation to a Picsearch’s customer. Mr. Riseberg acted as mediator in this process. Unfortunately we did not obtain any response from the contacted company. Hence, despite of the fact that Picsearch lacks in Android developers, we believed that based on their experience in the company our interviewees could perform an interesting evaluation of our work.

**Callback Interface Data**

Our interviewees considered that the data object provided as part of the callback interface (server response, source and error occurred) offers the right information to allow a proper handling of the callback flow. Further, it also enables the developer for keeping track of the origin of the call. The interviewees did not identify any other element that could be missing, or that in some moment could offer more advantages to handle the callback.

**Synchronous & Asynchronous Programming Models**

Mr. Atkinson expressed his preference to only include the asynchronous API. He justified his judgement based on the recommendations given by Android (see Section 4.1.3). Further, he mentioned that in this early stage of the component he would not have included the synchronous API in order to reduce the extra overhead associated to the maintenance of the two APIs. Only once the asynchronous API had acquired a more mature status and its advantages were obvious, then he would consider introducing the synchronous version.

For Mr. Rönnblom on the other hand, it was a good idea to offer both, the synchronous and asynchronous APIs to provide more flexibility to developers. Based on his previous experience, he mentioned the need of using both approaches depending on the situation you face as developer. Despite of lacking in any experience using our API, he believed it is positive to offer both API versions.

Besides maintainability, our interviewees could not identify any other disadvantage related to having both API versions.

**Understandability**

The overall impressions about the understandability of our APIs were very positive (see Figure 5.2). The general opinion was that the structure of the API can
be understood fairly easily; however, the fact of having a callback interface may reduce understandability to some extent because it uses a programming model that is somehow unusual for an average developer. Another possible source of confusion may be related to having both APIs, this is because despite of being extremely similar they experience differences in terms of arguments and return types.

A validity threat related to this section is the close involvement of our interviewees to the design and maintenance of the APIs offered by Picsearch (REST, XML-RPC, AJAX). This fact could facilitate their understanding because they knew in advance the functionality that needs to be delivered in this type of APIs.

Figure 5.2 shows the understandability ratings given to our API. The scale goes from 1 (extremely hard), to 7 (extremely easy).

Figure 5.2: Understandability

Consistency

In terms of functionality, our interviewees commented that our APIs are consistent with the services offered by Picsearch. Nevertheless they could point out inconsistencies regarding to naming. Mr. Atkinson mentioned a silly-caps inconsistency in the naming of one of the enums utilized in the APIs (i.e., SdkAPIMethodsEnum). The name was corrected to respect the naming conventions utilized in the API (i.e., SdkApiMethodsEnum). Mr. Rönblom pointed out that all our public classes utilize S9 as prefix to indicate that they belong to the Screen9 API. However, the actual API interfaces (S9SynchronousAPI and S9AsynchronousAPI) did not follow this naming convention. After his valuable observation the inconsistency was corrected.

Figure 5.3 presents the consistency ratings given by our interviewees. The scale goes from 1 (extremely inconsistent), to 7 (extremely consistent).
5.2.2 Reusability

No comments or concerns were mentioned about possible design flaws that could affect the reusability of our component. The fact that our component was used in two different Android applications and tested in five different Android devices without experiencing problems caused a positive effect in their evaluation of reusability.

5.2.3 Modifiability

Our interviewees agreed that the class design was appropriate for the functionality we were striving for. No potential improvements were made towards the class structure. Mr. Atkinson emphasized his support for the separation of concerns we presented in our design. He liked the idea of using a mediator as responsible for coordinating the inner workings of our component. Further, he added that our design eases the maintenance of the synchronous and asynchronous APIs by abstracting the common elements shared by both interfaces.

5.2.4 Security

At this point the SHA-1 encryption algorithm offers the adequate level to secure the transmission of passwords. Nevertheless, our interviewers commented the possibility of extending the REST API to use SSL as a way to provide more security to those customers that require a higher privacy to access their media accounts. In case this step is taken our component would need some modifications to adapt SSL to the request/response processes.
5.2.5 Documentation

In general, our interviewees gave very positive comments about the content, organization and presentation of the documentation delivered. In words of Mr. Risberg our API "is very solid and well documented". However, we are aware that there are always aspects to be improved. Through the interviews we experienced encountered arguments related to documentation. Mr. Atkinson suggested structuring the document in such a way that arguments that are frequently used in the API's methods may be explained in one section in order to avoid repetition. This would reduce the length of the document. Then, in each method's section we could try to explain only the particular details applicable for that method. On the other hand, Robert Risberg commented that repetition was correct because the reader goes specifically to a section where the method of interest is explained. Thus all the relevant information must be available in that specific section in order to avoid forcing the reader to browse the whole document.

As possible areas of improvement, Mr. Atkinson mentioned that it would be preferable if the documentation could provide the public methods of every response class provided by each of the API's methods. In the documentation we decided mentioning only the class names and not internal details because their structure are very intuitive, and we assume that a developer with an IDE (Integrated Development Environment) will easily review the public methods and attributes of the class of interest.

Mr. Atkinson, trusting that the asynchronous API has higher probabilities of being used, explored the possibility of focusing the documentation examples on just the asynchronous cases. Following this approach both APIs would be explained, but only the asynchronous API would show examples; getting in this way a reduction in the length of the document.

One point that wakens the evaluation of our documentation is that our interviewees had not experienced the actual need of using the documentation with the intention of developing with our APIs. Mr. Rönblom pointed out that the documentation may be clear at first glance, but it is very difficult to spot missing sections or possible improvements before it is actually used for practical purposes.

5.2.6 Other Comments & Suggestions

Mr. Atkinson suggested merging the synchronous and asynchronous APIs as one unique API in order to simplify its usage and the user’s documentation. This is definitely an option that we considered during the design process; however it was discarded. We believed that although the API would get simplified by following this approach, it would be prone to confuse the caller. The most obvious problem
is the return type of each method. If the reader observes the signature of the methods included in the synchronous API, all of them return an instance that extends S9Response, while the asynchronous methods return void. Consequently, by merging both interfaces we would force the merged signature to return a S9Response despite of the fact that the asynchronous methods always return a null value. Although it would be optimal to have only one API for simplicity and maintenance reasons, we believe that understandability would be negatively impacted. Thus in this trade-off understandability has a higher priority in our design.

### 5.2.7 Contribution of our Work to Picsearch

Besides of the component and documentation delivered, our API includes the method getUserProfile(), which is currently not supported by Picsearch’s production REST API. The functionality was added only to the testing environment as a proof of concept; however, it has proven to be a promising feature that will probably be included as part of the production REST API. Mr. Rönnblom commented that by request of Mr. Risberg he had been working on this feature to improve the functionality that was initially introduced as part of this thesis.

### 5.3 Future of the API

In words of Mr. Atkinson, the next step for the component would be to subject it to testing under a very extended set of Android smartphones. During this work we tested our component with four different smartphones and one Android tablet. In all of them the component behaved as it was expected. Nevertheless, before offering our component as part of their commercial services, Picsearch would like to ensure that it is tested in a more proper manner. Mr. Risberg also expressed openly his interest for making our work available for Picsearch’s customers.

Regarding the relation between the continuous growth of the smartphones market and its influence on our work, Mr. Risberg mentioned his interest in adding a feature that would allow Picsearch to gather statistical data about the way Android users interact with video in their applications. The goal is to allow the component to send extra information to the video platform, where later on it could be post-processed to generate statistical analyses. This type of statistical analyses have become very popular in web applications, and they have proven to be highly useful and profitable because they help to get a better understanding about users’ preferences.
Chapter 6

Thesis Results & Conclusions

On-line video usage in mobile devices is continuing to rise as user expectations for rich media content grow. As part of a market strategy Picsearch has decided to expand its video services by making them available in Android smartphones. This industrial Master’s thesis contributes with this expansion by providing a software component that enables rapid development of Android applications that rely on Picsearch video services.

The software component delivered with this thesis is the result of applying a rigorous software engineering approach that started with a requirements engineering phase, passed by a systematic literature review (SLR), software design and implementation, and ended up with an evaluation. In fact this was the main reason that motivated us to collaborate with Picsearch: to apply diverse SE areas in order to solve an industrial need. It is important to highlight our intention to continue a career in the software industry, therefore, this was the perfect opportunity to ground in a practical scenario all the knowledge acquired during our Master studies.

This chapter presents our discussion about the SE approach followed in this thesis, the results obtained, and our perception about each of the phases that took part in this work. It is intended to answer \textbf{RQ3: To what extent did the rigorous SE approach succeed in developing a quality component?} Hence, as starting point we consider important to give an overview of the phases involved in this SE approach, and the time that each one required. Thus Figure 6.1 describes our approach and its phases, and Figures 6.2 and 6.3 its distribution of time.

Figure 6.2 provide more details about the tasks that were performed in each phase. The color of each row in the table links the task with the phase it belongs to. The only task not associated to a phase was \textit{Thesis content refinement}; which refers to the refinement and correction process of this thesis document. Everything else supported one of the phases presented in Figure 6.1.
Figure 6.1: Our Software Engineering Approach and its Phases

Note well from Figures 6.2 and 6.3 that the order of the tasks does not follow exactly a waterfall behavior as Figure 6.1 shows. Figure 6.1 intends to simplify how our phases supported each other in order to deliver the final outcome of this thesis: the software component. However, rather than a waterfall model, our work followed an incremental approach, where requirements, design and implementation were introduced in two increments. The time pressure and the uncertainty for targeting the right functionality pushed us to follow this approach. The goal was to start delivering soon in order to ensure that our work was aligned with Picsearch expectations. Figure 6.2 shows the two increments experienced in our approach. Each increment started with a market analysis task, and ended up with architecture design and implementation related tasks (including documentation, prototyping).

The SLR took part only in the first iteration because no more SLR-related work was required once the review was completed and the design method had been selected. Figure 6.3 marks the tasks executed inside each iteration. Moreover, the evaluation task took part only after the final increment was completed. The intention was to perform an evaluation of the entire outcome (i.e., the software component, the documentation and our performance) rather than by increment. Further information about the tasks is given in the following sections, where we summarize every phase of our approach, and we discuss their contribution.
6.1 Market Analysis

One of the main difficulties we experienced in a starting point of this thesis was the lack of knowledge in the domain of video providers. Picsearch required as outcome of this collaboration a software artifact capable of providing rapid development of Android applications that made use of their services. The goal was perfectly clear, however purposefully it was set to an abstract level, thus not specific requirements were defined. Therefore, part of this thesis required: 1) understanding the company’s vision for this thesis, 2) learning the domain of video providers, and 3) defining the requirements for our software component. Hence, this section aims at addressing **RQ3.1: To what extent did the market study provide a useful contribution to the entire SE approach adapted in this work?** This research question intends to explore the results of this phase and whether these results were useful to the overall SE approach or not.
6.1.1 Results

From Figures 6.2 and 6.3 we observe that our Market Analysis phase was executed in two tasks. *Market analysis 1* was the longest and the one that set us into the context of this thesis. This task fundamentally helped us to define where to start and how to address this thesis. Our beginning at Picsearch was a little uncertain due to the lack of concrete software requirements for this thesis. Picsearch wanted as part of this collaboration to analyze the market and find out by ourselves what functional requirements needed to be included. Thus *Market analysis 1* helped us to overcome this challenge. Concretely this task served to define the Scenario 1 presented in Section 2.1.4. Same that provided the first set of functional requirements for our software component.

*Market analysis 2* on the other hand had a much shorter duration. This task took part in the second increment of our approach, and it helped us to model the Scenario 2 presented as well in Section 2.1.4. Scenario 2 was the base that supported us to define a second set of functional requirements. Further, to be more precise about the explanation of our results, we consider important to highlight the individual outcomes provided by each of our selected requirements engineering (RE) techniques:

- **Interviews**: Interviews represented the main source of information due to our physical presence at Picsearch. By interviewing the personnel we did not get specific indications of what to do, but little by little we came to understand their expectations for this thesis. Moreover, by listening to their answers we got situated in the context of the domain and their vision as a company, so at some point we got immersed in the company’s atmosphere. It is important to highlight that being physically in the company we had the opportunity to make questions anytime, and have informal discussions about any aspect relevant for this thesis. Moreover we got access to Picsearch’s weekly meetings, where their strategies were discussed. The latter could not be considered as an interview, however it supported the interviewing process over the time. By having a better idea about Picsearch’s business goals our questions became gradually more specific. The initial interviews had very general discussions and they mostly helped us to define where to start and what to do. Later on, once we had clear the goal to pursue, the interviews were more specific and focused on specific functionality to deliver in our software component. These latter interviews influenced to a high extent the Scenarios presented in Section 2.1.4. Thus the results of this technique were: 1) an understanding of the context of this thesis, *i.e.*, Picsearch’s thesis focus, company’s vision, the video providers domain, and Picsearch competitors; and 2) two scenarios (see Section 2.1.4) that served as base to define the functional requirements that were finally implemented in our component.
• **Document studies:** This technique was applied to review the online video platform (OVP) technical documentation provided by Picsearch. By studying OVP’s documentation we learned about their capabilities. This was a very important process because our component was constrained by these capabilities, thus the functionality we were expected to deliver needed to be aligned with the OVP. Furthermore, the OVP’s documentation served as a source of inspiration to our requirements selection process. Moreover, document studies were also applied in a more informal manner to review any documentation available on the web that targeted mobile video services, *e.g.*, technical websites about video in the Android OS, Picsearch competitors’ websites, and sites rating mobile video applications. Thus the study of all of these documents complemented the information that we gathered from our interviews. By doing both, document studies and interviews we could cross information to refine the knowledge acquired during the market analysis. Hence, the outcomes of applying document studies were: 1) an understanding of the capabilities of the OVP, 2) support to learn more about the domain of video providers, 3) the identification of popular mobile video applications, 4) technical information about handling video in the Android OS, 5) a source to identify features that were commonly used in mobile video; features that were compared with the OVP capabilities, and that served to model Scenario 1 (see Section 2.1.4).

• **Apprenticing:** Apprenticing was applied in order to fill the gap that existed due to our lack of experience in Picsearch’s domain. The technique helped us to get familiar with the functionality delivered by the OVP and by popular mobile video applications. Basically what we did was to play with the OVP and mobile video applications to get a feeling of important functionality and also identify any element that either could be missing or improved. Based on our findings we made questions to personnel at Picsearch in order to clarify ideas and complement the information that we acquired as result of the application of the other RE techniques (*i.e.*, interviews and document studies). The outcome of using this technique was 1) a valuable experience about understanding how video is used in mobile applications, and 2) identification of features commonly implemented in popular mobile video applications. Note well that this technique complemented interviews and document studies because it attacked the market analysis from a different perspective. The fact of using the actual mobile applications provided a very valuable knowledge than could not be acquired by reading documentation or making questions. By applying apprenticing we realized what was the role of our component when developing a mobile video applications, and how it was intended to support Android developers.
In summary, this market analysis precisely served as an understanding process of the domain of video providers. It included a revision of documents, competitors and mobile video applications that provided us important knowledge about how video is generally used on the web and specifically in mobile devices. Once we were familiar with the domain, we interviewed personnel at Picsearch in order to get information about their customer’s needs. The results of these interviews plus our previous revision allowed to define the functional requirements that were finally implemented in our software component.

6.2 Systematic Literature Review

In Chapter 3 we presented a systematic literature review (SLR) focused on identifying the software architecture design methods available in the SE literature. By means of this SLR we wanted to show the research that has been done in the area of software architecture. Furthermore, by identifying the available design methods we could analyze their characteristics. This analysis supported us to select the most appropriate method to suit our case study. Thus this section highlights the results of our SLR, and discusses our opinion about the research in the area.

6.2.1 Results

Nine different architecture design methods were identified as result of this SLR. Based on their characteristics to separate architectural concerns it can be observed that most methods target architectures for large software systems. Based on the publication dates of the selected articles it is clear that the most important research in the area was done from the second half of 1990s and until the first half of this century. In these 10 years we have observed the introduction of very recognized methods in academia as RUP’s 4 + 1 Views\cite{49, 51}, the Attribute-Driven Design (ADD)\cite{8, 9, 5}, and the Siemens Four-Views (S4V)\cite{68, 36, 37, 58, 59}. These design methods are pioneers in the area of software architecture, count with several literature sources describing them, and also seem to influence newer methods. On the other hand, with the rest of the methods we experienced a less availability of literature. Some methods were identified only on its publishing article. Some other, as the case of Architectural Separation of Concerns (ASC)\cite{63}, or the Business Architecture Process and Organization (BAPO)\cite{74, 1, 34} were developed by Nokia and Philips Research respectively between 2000 and 2004, but right now there is practically no information available.

One of our concerns after having performed this SLR is the lack of literature reporting the use of the methods in practical studies. We understand that in some cases as Siemens, Rational Software (IBM), Nokia and Philips Research the design methods were created as part of the internal research to improve the qual-
ity of their software systems. Thus this implies that these methods have been used in industrial settings. Nevertheless, with the exception of RUP that became commercial to some extent, based on the current literature it is not evident any practical evaluation of each of these methods. This makes difficult the method selection because the process becomes a bit superficial, and based only on the apparent characteristics described for each method. The issue is that until it is not used, it is difficult to realize each method’s potentialities and disadvantages. In our case study, having clear that there was a high degree of uncertainty in the requirements for our software component we decided to select the method that expressed explicit tailoring and understandability in their application. This in order to adapt the method to our needs and reduce the learning curve as possible. Thus tailoring and understandability performed as our selection criteria and as well the reason for what we finally chose GADesign.

Another issue to reflect about the available architecture design methods is the nature of the software methodologies they seem to target. Most of the methods describe a very exhaustive and systematic architecture design process, that is why they are known as architecture-centric methods. Our perception of this SLR is that these methods assume a good degree of certainty when it comes to requirements definition. However, with the current trend of agile methodologies on the raise, which main intention is to deliver fast and reducing the time spent on activities that may not remunerate economically to the software project, it becomes unclear where these architecture-centric methods fit on the current situation. Architecture-centric methods imply a considerable up-front design, normally effort that increases the quality of a software product, but that normally does not add economic benefits to the project, so in some sense it contradicts the premise of agile methodologies. Madison[54] and Nord et al.[60] presents interesting articles that claim that architecture-centric methods can be tailored in order to be introduced in agile approaches for improving their quality, and still delivering with agility. Personally we believe this is an interesting claim because our experience using GADesign basically intended to tailor an architecture-centric method in a software project that was characterized by uncertainty in its requirements and required agility to deliver. However being this project developed by one person it is a bit difficult to take significant conclusions. Therefore, we believe that the concept of using architecture-centric methods in agile projects is an important area that could provide valuable improvements to the current projects, and that unfortunately right now experiences scarce literature.
6.3 Architecture Design & Implementation

The architecture design and implementation phase of our work utilized the guidelines provided by GADesign. GADesign’s easy understandability and tailoring capabilities made it the best candidate from a set of nine design methods identified during our SLR. Thus this section intends to answer **RQ3.2: How did the use of GADesign enhance the architecture design process?** The section discusses our opinion about the benefits experienced by utilizing GADesign, and its potentialities as a design method.

6.3.1 Results

GADesign divides the architecture design into three phases: 1) Software architecture sketch design (SAS Design), 2) Software architecture refinement, and 3) Architecture transformation to implementation [71]. Each phase focuses on a different level of abstraction. The architect starts working with very coarse-grained components, but gradually the architecture experiences continuous refinements until the point where implementation takes place. This is one characteristic observed in most of the reviews methods. They strive for decomposing the architecture in order to reduce its complexity and simplify the design process. Personally we had a previous experience with the Siemens Four-Views (S4V) [68, 36, 37, 58, 59] method, and it presents a view-based separation of concerns. Thus it starts focusing on functional-conceptual components and ends up with actual implementation. Our experience has been that this separation of concerns truly facilitates the design process. Whenever architecture design is performed in an ad-hoc manner there is a common tendency to start thinking about classes or in the system’s component structure. This may not be wrong, but the issue is that while functional requirements may be covered because they are normally obvious, quality attributes may be in danger of being neglected. As we know, quality attributes sometimes are not explicitly stated in a requirements specification, thus in such cases it is the architect’s responsibility to use common sense to identify which quality attributes are important for the system. We consider this aspect as the main benefit of using SE design methods, because they make a strict emphasis on both, functional requirements and quality attributes as the basis of the architecture design process. In an initial stage of our design we defined our functional requirements and quality attributes, and they functioned as a mental guide to model the architecture and direct the design process. This is in our opinion what truly enhanced the architecture of our component, and in turn the main value of GADesign as an architecture-centric method.

In terms of clarity and understandability GADesign is very explicit about the purpose of each step required to design an architecture. Nevertheless, if compared with S4V for example, GADesign is not very detailed about how every step should
be executed. Thus the level of exhaustiveness to reach during the design process relies to a high extent on the architect’s experience. Hence, experience becomes an important factor when applying GADesign. The quality vs. time trade-off is easily encountered in architecture design; therefore experience is always crucial to know when the good-enough point has been reached. Not being experienced architects, our component’s design was ideal to experiment with GADesign because it was complex enough as to involve functional requirements, quality-attributes, architectural tactics and design patterns; but at the same time simple enough as to allow mistakes and correct them without requiring an excessive extra-effort.

Retaking the level of exhaustiveness, based on a previous experience using S4V we observed that it gives more strict guidelines in order to avoid missing the most minimal aspect that could influence the architecture. Consequently, we believe that the method protects the architect from his/her lack of experience because it encourages to cover everything that apparently influences the architecture. However, this also can become extremely time consuming and costly. GADesign on the other hand is clear when explaining its guidelines, however it relies to a high extent on the architect’s experience to know the good-enough point of exhaustiveness. In Section 6.2.1 we mentioned the possibility of including architecture-centric methods in agile methodologies. Based on the essence of GADesign, we believe that it would be suitable to work with agile methodologies because it can be easily tailored. Moreover, by not imposing any strict documentation guidelines the design effort can be adapted to the project’s time availability. In that sense, in the case of S4V we believe that due to its exhaustiveness it requires more effort and experience to know how to adapt it to the conditions of an agile project.

Evaluations is another aspect to highlight in our experience using GADesign. The method defines an evaluation after the completion of each phase in order to ensure that the architecture is progressively validated. This avoids future re-design in case the architecture deviates from satisfying its significant requirements. This is interesting and useful because every evaluation gradually focuses on a more fine-grained architecture. This fact simplifies the validation process because every evaluation acts as a filter; consequently, if an architecture passes an evaluation, it means that at that specific abstraction level the architecture satisfies its significant requirements. Thus the next evaluation gets simplified because coarser aspects were already validated in a previous phase. Therefore, this approach protects the architect of getting overwhelmed by a single final evaluation that intends to validate every architectural aspect. In general we think that separation of concerns is advantageous for both, design and evaluation. Based on our SLR we could observe that no all methods include evaluations as part of their processes. This does not mean that they neglect evaluations, but we understand that they conceptualize design and evaluation as two separate process. In that
sense we think that GADesign presents an advantage by integrating both tasks within a single approach. Finally, although it is the architect’s responsibility to decide when and how to do the evaluation, we believe it is very useful to know how authors suggest including evaluations within their method’s execution in order to take the best method’s advantage.

In summary, as response to **RQ3.2: How did the use of GADesign enhance the architecture design process?** We emphasize three main aspects:

1. **Requirements-driven process:** The whole design process was driven by its functional requirements and quality attributes, thus by doing this we ensured their achievement.

2. **Architecture breakdown:** The architecture design is considerably simplified by breaking the process in the three abstraction levels suggested by GADesign. By doing this, in each phase we could focus in a smaller group of concerns at once, thus we reduced complexity and achieved results with a higher quality.

3. **Evaluations:** The progressive evaluations proposed by the method helped us to identify and correct problems in the earliest stage, thus the risk of re-work in a late stage were minimized.

### 6.4 Evaluation

Three deliveries resulted from our collaboration at Picsearch: 1) a software component, 2) the API’s documentation, and 3) a proof of concept application that demonstrates the capabilities offered by our component. Taking into account that the software component is the main deliverable from this thesis, we considered important to present some metrics related to their implementation (see Table 6.1). Further, the API’s documentation is presented in Appendix B, and our application proof of concept is depicted in Figure 6.4.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Total</th>
<th>Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total lines of code</td>
<td>2017</td>
<td>-</td>
</tr>
<tr>
<td>Number of concrete classes</td>
<td>44</td>
<td>-</td>
</tr>
<tr>
<td>Number of interfaces</td>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td>Number of abstract classes</td>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td>Number of methods</td>
<td>251</td>
<td>-</td>
</tr>
<tr>
<td>Depth of inheritance</td>
<td>-</td>
<td>1.435</td>
</tr>
</tbody>
</table>

*Table 6.1: Component’s Metrics*
Based on the these deliverables we conducted a set of interviews to the personnel at Picsearch, same that provided the results presented in Chapter 5. During the interviews we focused on formulating questions about the usefulness of the functionality delivered with the component, its quality attributes and its API documentation. The opinion of our interviewees were very positive for every aspect, although they also expressed about some possible improvements. Our main interest for doing this evaluation was to know Picsearch’s opinion about how well our SE approach achieved its intended goal. However, the evaluation also had a secondary goal: to know Picsearch’s satisfaction in relation to our results and performance. The latter might no be related to SE, but to some extent their satisfaction reflected the quality of our work, which fortunately resulted to be highly positive.

On the other hand, it is fair to mention that a more proper SE evaluation was considered but not carried. From the strict SE stand point, a proper evaluation would have involved actual Android developers, and required them to experiment with our component to provide some feedback about its usage. The reason for discarding this experiment were: 1) it was unfeasible for a Master’s thesis to carry an experiment of this type considering the effort it would require plus the effort demanded by the previous phases of our approach; 2) it required subjects with an Android programming background, which right now is not as extended as other commercial languages as C# or Java; and 3) it would be a real challenge to find participants willing to provide a considerable time to actually try to code
something that truly tested the component’s capabilities. Therefore, we decided that an experiment of this type was out of scope, and we decided to focus only on the personnel at Picsearch.

In summary we believe that we evaluated with Picsearch the main aspects of our work, and the company demonstrated to be highly satisfied with our results. However, we also recognize that our current evaluation experiences some validity threats, and that a more proper SE evaluation was possible, however unfeasible due to the time limitation for this thesis.

6.5 Overall Impressions

Having a developer’s background, before starting this SE Master’s degree our perspective about software development was too narrow and centered on coding. This program has widen our perspective to consider that not only technical but also business factors impose a high influence in the engineering of a software system. Hand by hand with this new perspective this Master’s has helped us to understand different concerns that involve the software engineering, e.g., requirements engineering, software architecture, project management, verification and validation. Thus this thesis as a final assignment of our studies has allowed us to experience those concerns in a practical study. In this sense it has been very exciting to take a real industrial case, and apply our knowledge in order to create a solution that is backed up by the resources acquired during our studies. Personally this was our main motivation to work with this thesis, and fortunately it ended with a very fructiferous outcome.

With regard to our SE approach it is very rewarding to observe that every phase accomplished its expected outcome, and as a whole, the entire approach proved to be practical, useful and suitable to deliver a quality component that in turn solved Picsearch’s needs.
Appendix A

Tables SLR

A.1 Data Extraction Form

<table>
<thead>
<tr>
<th>Id</th>
<th>Fundamental Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Data Extractor</td>
</tr>
<tr>
<td>2</td>
<td>Data Checker</td>
</tr>
<tr>
<td>3</td>
<td>Date of Data Extraction</td>
</tr>
<tr>
<td>4</td>
<td>Article Title</td>
</tr>
<tr>
<td>5</td>
<td>Authors’ Name</td>
</tr>
<tr>
<td>6</td>
<td>Application Domain</td>
</tr>
<tr>
<td>7</td>
<td>Journal/Conference/Conference proceedings</td>
</tr>
<tr>
<td>8</td>
<td>Retrieval Search Query</td>
</tr>
<tr>
<td>9</td>
<td>Year of publication</td>
</tr>
</tbody>
</table>

**RQ1:** What are the current state-of-the-art SE methods that provide guidance for designing a software architecture?

<table>
<thead>
<tr>
<th>Id</th>
<th>Name of software architecture design method</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Applicability domain of software</td>
</tr>
<tr>
<td>12</td>
<td>Industrial validation</td>
</tr>
<tr>
<td>13</td>
<td>Core elements of each method</td>
</tr>
<tr>
<td>14</td>
<td>Steps needed to execute the method</td>
</tr>
</tbody>
</table>

*Table A.1: Data Extraction Form*

A.2 Articles Selected
<table>
<thead>
<tr>
<th>Id</th>
<th>Ref</th>
<th>Year</th>
<th>Study Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>[49]</td>
<td>1995</td>
<td>Architectural Blueprints - The &quot;4+1&quot; View Model of Software Architecture</td>
</tr>
<tr>
<td>2</td>
<td>[68]</td>
<td>1995</td>
<td>Software Architecture in Industrial Applications</td>
</tr>
<tr>
<td>3</td>
<td>[8]</td>
<td>2000</td>
<td>The Architecture Based Design Method</td>
</tr>
<tr>
<td>4</td>
<td>[58]</td>
<td>2001</td>
<td>Effective software architecture design: From global analysis to UML descriptions</td>
</tr>
<tr>
<td>5</td>
<td>[5]</td>
<td>2001</td>
<td>Introduction to the attribute driven design method</td>
</tr>
<tr>
<td>6</td>
<td>[57]</td>
<td>2001</td>
<td>Software architecture in a changing world: developing design strategies that anticipate change</td>
</tr>
<tr>
<td>7</td>
<td>[51]</td>
<td>2002</td>
<td>The 4 + 1 View Model of Architecture</td>
</tr>
<tr>
<td>9</td>
<td>[59]</td>
<td>2003</td>
<td>Experience with global analysis: a practical method for analyzing factors that influence software architectures</td>
</tr>
<tr>
<td>10</td>
<td>[1]</td>
<td>2004</td>
<td>Multi-view Variation Modeling for Scenario Analysis</td>
</tr>
<tr>
<td>11</td>
<td>[31]</td>
<td>2004</td>
<td>Reconciling Software Requirements and Architectures with Intermediate Models</td>
</tr>
<tr>
<td>12</td>
<td>[37]</td>
<td>2005</td>
<td>Global Analysis: moving from software requirements specification to structural views of the software architecture</td>
</tr>
<tr>
<td>13</td>
<td>[71]</td>
<td>2006</td>
<td>A gradually proceeded software architecture design process</td>
</tr>
<tr>
<td>14</td>
<td>[38]</td>
<td>2006</td>
<td>Generalizing a Model of Software Architecture Design from Five Industrial Approaches</td>
</tr>
</tbody>
</table>

*Table A.2: Articles Selected 1*
<table>
<thead>
<tr>
<th>No.</th>
<th>Year</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>2006</td>
<td>The essential components of software architecture design and analysis</td>
</tr>
<tr>
<td>16</td>
<td>2007</td>
<td>A general model of software architecture design derived from five industrial approaches</td>
</tr>
<tr>
<td>17</td>
<td>2007</td>
<td>A Pattern-Driven Process Model for Quality-Centered Software Architecture Design - A Case Study on Usability-Centered Design</td>
</tr>
<tr>
<td>18</td>
<td>2007</td>
<td>Do architecture design methods meet architects’ needs?</td>
</tr>
<tr>
<td>19</td>
<td>2010</td>
<td>Co-design of the Business and Software Architectures: A Systems Engineering and Model-Driven Method From Requirements to Architectural Design - Using Goals and Scenarios</td>
</tr>
</tbody>
</table>

*Table A.3: Articles Selected 2*
# A.3 Queries

<table>
<thead>
<tr>
<th>Data source</th>
<th>Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEEE Xplore</td>
<td>(&quot;Abstract&quot;:&quot;software architecture design&quot; AND (&quot;Abstract&quot;:&quot;global analysis&quot; OR &quot;Abstract&quot;:method OR &quot;Abstract&quot;:technique OR &quot;Abstract&quot;:approach OR &quot;Abstract&quot;:guide OR &quot;Abstract&quot;:analysis OR &quot;Abstract&quot;:&quot;software requirements specification&quot; OR &quot;Abstract&quot;:requirements OR &quot;Abstract&quot;:SRS )) OR (&quot;Publication Title&quot;:&quot;software architecture design&quot; AND (&quot;Publication Title&quot;:&quot;global analysis&quot; OR &quot;Publication Title&quot;:method OR &quot;Publication Title&quot;:technique OR &quot;Publication Title&quot;:approach OR &quot;Publication Title&quot;:guide OR &quot;Publication Title&quot;:analysis OR &quot;Publication Title&quot;:&quot;software requirements specification&quot; OR &quot;Publication Title&quot;:requirements OR &quot;Publication Title&quot;:SRS )) AND (FtFlag:yes) AND (AbstractFlag:yes)</td>
</tr>
<tr>
<td>ACM</td>
<td>(&quot;Abstract&quot;:&quot;software architecture design&quot; AND (&quot;Abstract&quot;:&quot;global analysis&quot; OR &quot;Abstract&quot;:method OR &quot;Abstract&quot;:technique OR &quot;Abstract&quot;:approach OR &quot;Abstract&quot;:guide OR &quot;Abstract&quot;:analysis OR &quot;Abstract&quot;:&quot;software requirements specification&quot; OR &quot;Abstract&quot;:requirements OR &quot;Abstract&quot;:SRS ))</td>
</tr>
<tr>
<td>Springer Link</td>
<td>(&quot;software architecture design&quot; AND (&quot;global analysis&quot; OR method OR approach OR requirements))</td>
</tr>
</tbody>
</table>

*Table A.4:* Customized Queries According to the Selected Databases
<table>
<thead>
<tr>
<th>Database</th>
<th>Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eng. Village</td>
<td>(software architecture design wn AB AND (global analysis OR method OR technique OR approach OR guide OR software requirements specification OR requirements OR SRS) wn AB ) OR (software architecture design wn TI AND (global analysis OR method OR technique OR approach OR guide OR software requirements specification OR requirements OR SRS) wn TI )</td>
</tr>
<tr>
<td>Scopus</td>
<td>TITLE-ABS-KEY(&quot;software architecture design&quot; AND (&quot;global analysis&quot; OR method OR technique OR approach OR guide OR &quot;software requirements specification&quot; OR requirements OR SRS))</td>
</tr>
</tbody>
</table>

*Table A.5:* Customized Queries According to the Selected Databases (Cont.)
Appendix B

API Documentation

B.1 Background

On-line video usage is continuing to rise as user expectations for rich media content grow. With the arrival of a new generation of sophisticated smart mobile phones, possibilities for mobile video usage are presenting exciting new opportunities. At present Screen9 is planning to expand its services to platforms supported by smart phones. Android OS is one of the platforms targeted by Screen9. Part of this expansion includes the design of a component that is able to support rapid development of applications involving video streaming services.

Currently, in order to extend its video streaming services to mobile devices, Screen9 offers an SDK capable to support rapid development of applications using diverse video features including interactive on-demand streaming and video file uploads and management of associated meta data such as commenting and ratings. Thus this document explains the API offered by such SDK and some design issues to consider when using it.

B.2 Specifications

This SDK provides an API that abstracts the communication with Screen9’s server, therefore internet access is required. The classes S9SynchronousAPI and S9AsynchronousAPI implements the functionality of the API. The following sections explains basic concepts of this SDK.

B.2.1 Android Version - API Level

The SDK requires Android 1.5 (API Level 3) or above.

B.2.2 Thread Safety & Programming Model

Our component by design is stateless, *i.e.*, it does not hold any value or state along its use. This characteristic makes it thread-safe, *i.e.*, thread-safeness allows
an object to be used by multiple threads at the same time without experiencing problems or strange behaviors.

To adapt to developer needs this SDK offers two programming models: 1) synchronous and 2) asynchronous. The former uses a single thread programming model, while the latter relies on the use of a background thread to perform its operations. The relevant issues to each model are explained below:

**Synchronous Programming Model**

Android’s programming model advises developers to avoid involving the main thread in the execution of long-running operations, *e.g.*, network access or database queries. The reason is that the main thread manages the interaction with the graphic user interface(GUI), and if it becomes busy accessing a network resource, the GUI gets unresponsive until the thread completes its task. Therefore, considering our SDK performs network communication we recommend to spawn a background thread to call our synchronous API. Android’s relevant information can be found in [2].

We exemplify this advice in Listing B.1. The call to the API is performed by a background thread and scheduled through a Handler[20]. Handler is another mechanism offered by android to deal with asynchronous behavior.

**Listing B.1: Synchronous API Use**

```java
final S9SynchronousAPI api = S9SynchronousAPI.getInstance();
Handler handler = new Handler();
handler.post(new Runnable(){
    public void run(){
        S9ResultData data = api.getUserProfiles(username, password);
        S9UserProfileResponse response =
            (S9UserProfileResponse) data.getResponse();
        // Process response
    }
});
```

**Asynchronous Programming Model**

The asynchronous API was introduced with the goal of saving effort of developers for creating a background thread to call our API. Thus the synchronous API incorporates internally the use of Android’s AsyncTask[19] to create a background
thread that communicates with our server.

This programming model requires the caller to implement a callback interface (see Sections Common Elements and Callback Interface & Callback Data) that will be called when the operation of a method has been completed. Further, there is a limitation associated when the component is used in an asynchronous mode. Previously we explained that AsyncTask is internally used to handle the asynchronous execution. AsyncTask by design it only allows the main thread to start its execution. Any omission of this rule results in a runtime exception. Thus this limitation is then transferred to our component. Therefore, if the asynchronous API is used (see Listing B.2), callers must be careful to use the main thread to execute our API’s asynchronous methods.

Listing B.2: Asynchronous API Use

```java
S9AsynchronousAPI api = S9AsynchronousAPI.getInstance();
api.getUserProfiles(username, password, new S9CallbackHandler()
{
    public void processCallback(S9ResultData data)
    {
        S9UserProfileResponse response = (S9UserProfileResponse) data.getResponse();
        // Process response
    }
});
```

The previous restriction does not apply for the synchronous API. Any thread can call the method without causing negative effects.

Common Elements:

The inner working of every API method requires a set of common values. These values are contained as an instance of the class `S9Commons`. An common object requires non-null values for the following attributes:

- **Authentication token**: The token associated to a Screen9 account. For example: mhhWUZypR2HLZVtEBQ7bqwWXeHUZQR2JI
- **Callback handler**: An object of a class implementing the interface `S9CallbackHandler`. See Section B.2.2 for more detail.

**NOTE**: These two values must not be null. When detecting a null value the API will throw an `IllegalArgumentException`.
Callback Interface & Callback Data:

As explained in Section Asynchronous Programming Model, the API uses the interface `S9CallbackHandler` to perform asynchronous calls when the result of a method execution is ready. The interface `S9CallbackHandler` is a type used by the API to identify objects than can be called once a method has completed its task. Thus objects required to be called by the API must implement this interface. The interface defines the method `processCallback` (see Listing B.3) which receives an instance of `S9ResultData` as argument.

Listing B.3: S9CallbackHandler Interface

```java
public interface S9CallbackHandler {
    public void processCallback(S9ResultData callbackData);
}
```

An instance of the class `S9CallbackHandler` provides information about result of an API method. Internally `S9ResultData` contains the following attributes with its respective getters.

- **Response**: Instance that inherits the class `S9Response`. Not all methods provide as response an instance of the same class. For example, the method `getMediaContent` provides an instance of the class `S9MediaContentResponse` (see Listing B.4), while the method `listAllMedia` return an instance of `S9ListMediaResponse`. Review the documentation of every method to find what they provide as a response.

- **Source**: Provides an instance of `SdkAPIMethodsEnum` (see Listing B.4) that identifies the API method that has called the callback interface. Additionally, every instance of `SdkAPIMethodsEnum` implements a `toString` method that returns their String representation.

- **Error**: Provides an instance of `S9ErrorInfo`. This object provides information about any error occurred. Internally, `S9ErrorInfo` contains the attributes listed below. See Listing B.4 to exemplify its usage.

  - **Error code**: Code of the error occurred. Some errors do not have a code associated.
  - **Error message**: Description of the error occurred.
  - **Exception**: If the problem was caused by an exception, such exception will be caught and provided as part of the callback data. Note well that not all errors contain exceptions, so this value may be null.
public void processCallback(S9ResultData callbackData) {
    if (callbackData.isError() == false) {
        if (callbackData.getSourceMethod() == SdkApiMethodsEnum.getMediaContent) {
            S9MediaContentResponse response = (S9MediaContentResponse) callbackData.getResponse();
            // Do something with the answer
        }
    } else {
        S9ErrorInfo error = callbackData.getError();
        String errorMessage = error.getErrorMessage();
        Exception exception = error.getException();
        // Do something with the error
    }
}
Appendix B. API Documentation

B.3 API Methods

B.3.1 assignRating

This method allows to set a rating to a video that goes from 0 to 100. The user inputs the rating and the platform calculates the average value based on previous ratings. The result of the method is a `S9ResultData` object that internally contains an instance of the class `S9RatingResponse`. The response object holds internally the average or global rating of the video.

**Synchronous Version**

*Listing B.5: Synchronous assignRating*

```java
public S9ResultData assignRating(String authToken, String mediaId, int rating)
```

**Arguments:**

1. `authToken`: It contains internally the authorization value to access a media account.
2. `mediaId`: The identifier of the video or media to rate.
3. `rating`: The rating value that goes from 0 to 100. Where 0 is the lowest value and 100 the highest value.

*Listing B.6: Synchronous assignRating Example*

```java
S9SynchronousAPI api=S9SynchronousAPI.getInstance();
S9ResultData result=api.assignRating("auth-token","media-id", 85);
S9RatingResponse response=(S9RatingResponse) result.getResponse();
```

**Asynchronous Version**

*Listing B.7: Asynchronous assignRating*

```java
public void assignRating(S9 Commons commons, String mediaId, int rating)
```
Appendix B. API Documentation

Arguments:

1. **commons**: It contains internally the authorization value and an instance of the callback interface to call. See Section B.2.2 for more detail.

2. **mediaId**: The identifier of the video or media to rate.

3. **rating**: The rating value that goes from 0 to 100. Where 0 is the lowest value and 100 the highest value.

Listing B.8: Asynchronous assignRating Example

```java
S9AsynchronousAPI api = S9AsynchronousAPI.getInstance();
S9Commons commons = new S9Commons("auth-token", new
    S9CallbackHandler() {
        public void processCallback(S9ResultData callbackData) {
            S9RatingResponse response =
                (S9RatingResponse)callbackData.getResponse();
        }
    });
api.assignRating(commons,"media-id",85);
```

B.3.2 getMediaContent

This method allows to access the specifics of a video (e.g., URL, title, description and more). The result of the method is a S9ResultData object that internally contains an instance of the class S9MediaContentResponse. The response object gives detailed information about the media resource.

Synchronous Version

Listing B.9: Synchronous getMediaContent

```java
public S9ResultData getMediaContent(String authToken, String mediaId)
```

Arguments:

1. **authToken**: It contains internally the authorization value to access a media account.

2. **mediaId**: The identifier of the video or media to rate.
Appendix B. API Documentation

Listing B.10: Synchronous getMediaContent Example

```java
S9SynchronousAPI api = S9SynchronousAPI.getInstance();
S9ResultData result = api.getMediaContent("auth-token", "media-id");
S9MediaContentResponse response = (S9MediaContentResponse) result.getResponse();
```

Asynchronous Version

Listing B.11: Asynchronous getMediaContent

```java
public void getMediaContent(S9Commons commons, String mediaId)
```

Arguments:

1. commons: It contains internally the authorization value and an instance of the callback interface to call. See Section B.2.2 for more detail.

2. mediaId: The identifier of the video or media to rate.

Listing B.12: Asynchronous getMediaContent Example

```java
S9AsynchronousAPI api = S9AsynchronousAPI.getInstance();
S9Commons commons = new S9Commons("auth-token", new S9CallbackHandler() {
    public void processCallback(S9ResultData callbackData) {
        S9MediaContentResponse response = (S9MediaContentResponse) callbackData.getResponse();
        String videoURL = response.getMediaUrl();
    }
});
api.getMediaContent(commons, "media-id");
```

B.3.3 getUploadURL

To be able to upload videos to Screen9’s video platform it is required to request a dedicated URL for this purpose. This method allows to get such URL. The result of the method is a S9ResultData object that internally contains an instance of the class S9UploadURLResponse. This method returns an authorization token with rights to upload a media item and also the URL where the video will be uploaded.
Appendix B. API Documentation

Synchronous Version

Listing B.13: Synchronous getUploadURL

```java
public S9ResultData getUploadURL(String authenticationToken)
```

Arguments:

1. **authToken**: It contains internally the authorization value to access a media account.

Listing B.14: Synchronous getUploadURL Example

```java
S9SynchronousAPI api = S9SynchronousAPI.getInstance();
S9ResultData result = api.getUploadURL("auth-token");
S9UploadURLResponse resp = (S9UploadURLResponse)result.getResponse();
```

Asynchronous Version

Listing B.15: Asynchronous getUploadURL

```java
public void getUploadURL(S9Commons commons)
```

Arguments:

1. **commons**: It contains internally the authorization value and an instance of the callback interface to call. See Section B.2.2 for more detail.

Listing B.16: Asynchronous getUploadURL Example

```java
S9AsynchronousAPI api = S9AsynchronousAPI.getInstance();
S9Commons commons = new S9Commons("auth-token", new
S9CallbackHandler() {

    public void processCallback(S9ResultData callbackData) {
        S9UploadURLResponse response =
            (S9UploadURLResponse)callbackData.getResponse();
        String uploadURL = response.getUrl();
    }
});
api.getUploadURL(commons);
```

B.3.4 **getUserProfiles**

This method retrieves the different Screen9 accounts associated to a given username and password. The result of the method is a **S9ResultData** object that internally contains an instance of the class **S9UserProfileResponse**.
Appendix B. API Documentation

Synchronous Version

**Listing B.17:** Synchronous getUserProfiles

```java
public S9ResultData getUserProfiles(String username, String password)
```

Arguments:

1. **username & password:** The login information(username & password) registered in the platform for a user.

**Listing B.18:** Synchronous getUserProfiles Example

```java
S9SynchronousAPI api = S9SynchronousAPI.getInstance();
S9ResultData result = api.getUserProfiles("username", "password");
S9UserProfileResponse response = (S9UserProfileResponse) result.getResponse();
```

Asynchronous Version

**Listing B.19:** Asynchronous getUserProfiles

```java
public void getUserProfiles(String username, String password,
S9CallbackHandler callbackhandler)
```

Arguments:

1. **username & password:** The login information(username & password) registered in the platform for a user.

2. **callbackhandler:** An instance of the callback interface to call. See Section B.2.2 for more detail.

**Listing B.20:** Asynchronous getUserProfiles Example

```java
S9AsynchronousAPI api = S9AsynchronousAPI.getInstance();
api.getUserProfiles(username, password, new S9CallbackHandler() {

    public void processCallback(S9ResultData callbackData) {
        S9UserProfileResponse response =
        (S9UserProfileResponse) callbackData.getResponse();
        List<Account> screen9Accounts = response.getAccounts();
    }
});
```
B.3.5 listAllCategories

This method retrieves all the categories created for a specific Screen9 account. The result of the method is a **S9ResultData** object that internally contains an instance of the class **S9ListCategoriesResponse**.

**Synchronous Version**

**Listing B.21:** Synchronous listAllCategories

```java
public S9ResultData listAllCategories(String authToken)
```

**Arguments:**

1. **authToken**: It contains internally the authorization value to access a media account.

**Listing B.22:** Synchronous listAllCategories Example

```java
S9SynchronousAPI api = S9SynchronousAPI.getInstance();
S9ResultData result = api.listAllCategories("auth-token");
S9ListCategoriesResponse response =
    (S9ListCategoriesResponse) result.getResponse();
List<Category> categoriesList = response.getCategories();
```

**Asynchronous Version**

**Listing B.23:** Asynchronous listAllCategories

```java
public void listAllCategories(S9Commons commons)
```

**Arguments:**

1. **commons**: It contains internally the authorization value and an instance of the callback interface to call. See Section B.2.2 for more detail.

**Listing B.24:** Asynchronous listAllCategories Example

```java
S9AsynchronousAPI api = S9AsynchronousAPI.getInstance();
S9Commons commons = new S9Commons("auth-token", new
    S9CallbackHandler()
        {
            public void processCallback(S9ResultData callbackData) {
                S9ListCategoriesResponse response =
                    (S9ListCategoriesResponse) callbackData.getResponse();
                List<Category> categoriesList = response.getCategories();
            }
        });
api.listAllCategories(commons);
```
B.3.6 listAllMedia

This method retrieves all the videos uploaded to a specific Screen9 account, and which their moderation status has been set as APPROVED (see Section B.3.12 for method setModeration). The result of the method is a S9ResultData object that internally contains an instance of the class S9ListMediaResponse. The method returns a maximum of 100 media items.

Synchronous Version

Listing B.25: Synchronous listAllMedia

```java
public S9ResultData listAllMedia(String authToken, 
      MediaFieldsEnum[] fields, APIOrderEnum order)
```

Arguments:

1. `authToken`: It contains internally the authorization value to access a media account.

2. `fields`: It is an array of elements of the enum type MediaFieldsEnum. This array specifies the fields that need to be returned for every media element contained in the resulted list. Table B.1 in Appendix B.4 shows the list of available fields. If this parameter is null, the list will only include the MEDIAID field.

3. `order`: Specifies the sorting criterion for the list. The sorting criterion is represented by one of the following values: POSTED, RATING, TITLE. Thus the list will be ordered upwardly based on the value specified. If null is provided then by default the list is sorted by TITLE.

Listing B.26: Synchronous listAllMedia Example

```java
S9SynchronousAPI api = S9SynchronousAPI.getInstance();
MediaFieldsEnum[] mediaFields = new MediaFieldsEnum[] { 
      MediaFieldsEnum.MEDIAID, MediaFieldsEnum.TITLE, 
      MediaFieldsEnum.DESCRIPTION, MediaFieldsEnum.IMAGE };

S9ResultData result = api.listAllMedia("auth−token", 
      mediaFields, APIOrderEnum.RATING);
S9ListMediaResponse resp = 
      (S9ListMediaResponse) result.getResponse();
List<Media> mediaLst = response.getMedia();
```
Appendix B. API Documentation

Asynchronous Version

Listing B.27: Asynchronous listAllMedia

```java
public void listAllMedia(S9Commons commons,
                         MediaFieldsEnum[] fields,
                         APIOrderEnum order)
```

Arguments:

1. `commons`: It contains internally the authorization value and an instance of the callback interface to call. See Section B.2.2 for more detail.

2. `rest of the arguments`: Please review the synchronous version described above.

Listing B.28: Asynchronous listAllMedia Example

```java
S9AsynchronousAPI api = S9AsynchronousAPI.getInstance();
S9Commons commons = new S9Commons("auth-token", new
        S9CallbackHandler()
        {
            public void processCallback(S9ResultData callbackData)
            {
                S9ListMediaResponse response =
                    (S9ListMediaResponse)callbackData.getResponse();
                List<Media> mediaLst = response.getMedia();
            }
        });

MediaFieldsEnum[] mediaFields = new MediaFieldsEnum[]{
        MediaFieldsEnum.MEDIAID, MediaFieldsEnum.TITLE,
        MediaFieldsEnum.DESCRIPTION, MediaFieldsEnum.IMAGE};

api.listAllMedia(commons, mediaFields, APIOrderEnum.RATING);
```

B.3.7 listAllTags

This method retrieves all the tags registered for a given Screen9 account. The result of the method is a `S9ResultData` object that internally contains an instance of the class `S9ListTagsResponse`.

Synchronous Version

Listing B.29: Synchronous listAllTags

```java
public S9ResultData listAllTags(String authToken)
```
Arguments:

1. **authToken**: It contains internally the authorization value to access a media account.

Listing B.30: Synchronous listAllTags Example

```java
S9SynchronousAPI api = S9SynchronousAPI.getInstance();
S9ResultData result = api.listAllTags("auth-token");
S9ListTagsResponse response = (S9ListTagsResponse) result.getResponse();
```

Asynchronous Version

Listing B.31: Asynchronous listAllTags

```java
public void listAllTags(S9Commons commons)
```

Arguments:

1. **commons**: It contains internally the authorization value and an instance of the callback interface to call. See Section B.2.2 for more detail.

Listing B.32: Asynchronous listAllTags Example

```java
S9AsynchronousAPI api = S9AsynchronousAPI.getInstance();
S9Commons commons = new S9Commons("auth-token",
   new S9CallbackHandler() {
      public void processCallback(S9ResultData callbackData) {
         S9ListTagsResponse response = (S9ListTagsResponse) callbackData.getResponse();
      }
   });
api.listAllTags(commons);
```

B.3.8 listMedia

This method presents the same functionality than listAllMedia (see B.3.6), but introduces two additional parameters: offset and maxResults. Detail of these two parameters are explained below. The result of the method is a S9ResultData object that internally contains an instance of the class S9ListMediaResponse.
Appendix B. API Documentation

Synchronous Version

Listing B.33: Synchronous listMedia

```java
public S9ResultData listMedia(String authToken,
                              MediaFieldsEnum[] fields, APIOrderEnum order,
                              int offset, int maxResults)
```

Arguments:

1. **authToken**: It contains internally the authorization value to access a media account.

2. **fields**: It is an array of elements of the enum type `MediaFieldsEnum`. This array specifies the fields that need to be returned for every media element contained in the resulted list. Table B.1 in Section B.4 shows the list of available fields. If this parameter is null, the list will only include the `MEDIAID` field.

3. **order**: Specifies the sorting criterion for the list. The sorting criterion is represented by one of the following values: `POSTED`, `RATING` and `TITLE`. Thus the list will be ordered upwardly based on the value specified. If null is provided then by default the list is sorted by `TITLE`.

4. **offset**: From the complete list of videos associated to an account, this parameter specifies the video to start from. For example, if an offset of 4 is set, the returned list will include as first element the 4th video of the complete list existing for an account(3 elements are skipped in the list).

5. **maxResults**: Specifies the maximum number of media items to retrieve in the resulted list. The maximum value is 100.

Listing B.34: Synchronous listMedia Example

```java
S9SynchronousAPI api = S9SynchronousAPI.getInstance();
MediaFieldsEnum[] mediaFields = new MediaFieldsEnum[] {
    MediaFieldsEnum.MEDIAID, MediaFieldsEnum.TITLE,
    MediaFieldsEnum.DESCRIPTION, MediaFieldsEnum.IMAGE
};

int offset = 4;
int maxResults = 50;
S9ResultData result = api.listMedia("auth-token", mediaFields,
                                    APIOrderEnum.RATING, offset, maxResults);
S9ListMediaResponse resp = (S9ListMediaResponse) result.getResponse();
List<Media> mediaLst = response.getMedia();
```
### Asynchronous Version

**Listing B.35: Asynchronous listMedia**

```java
public void listMedia(S9Commons commons,
    MediaFieldsEnum[] fields, APIOrderEnum order,
    int offset, int maxResults)
```

**Arguments:**

1. **commons:** It contains internally the authorization value and an instance of the callback interface to call. See Section B.2.2 for more detail.

2. **rest of the arguments:** Please review the synchronous version described above.

**Listing B.36: Asynchronous listMedia Example**

```java
S9AsynchronousAPI api = S9AsynchronousAPI.getInstance();
S9Commons commons = new S9Commons("auth-token",
    new S9CallbackHandler()
    {
        public void processCallback(S9ResultData callbackData)
        {
            S9ListMediaResponse response =
                (S9ListMediaResponse) callbackData.getResponse();
            List<Media> mediaLst = response.getMedia();
        }
    });
MediaFieldsEnum[] mediaFields = new MediaFieldsEnum[] {
    MediaFieldsEnum.MEDIAID, MediaFieldsEnum.TITLE,
    MediaFieldsEnum.DESCRIPTION, MediaFieldsEnum.IMAGE};
api.listMedia(commons, mediaFields, APIOrderEnum.RATING, 4, 50);
```

### B.3.9 listFilteredMedia

This method presents the same functionality than listMedia (see B.3.8), but introduces one additional parameter: filters. Detail of this parameter is explained below. The result of the method is a `S9ResultData` object that internally contains an instance of the class `S9ListMediaResponse`.

**Synchronous Version**

**Listing B.37: Synchronous listFilteredMedia**

```java
public S9ResultData listFilteredMedia(String authenticationToken,
    MediaFieldsEnum[] fields, APIOrderEnum order,
    int offset, int maxResults, FilterSet filters)
```
Arguments:

1. **authToken**: It contains internally the authorization value to access a media account.

2. **fields**: It is an array of elements of the enum type `MediaFieldsEnum`. This array specifies the fields that need to be returned for every media element contained in the resulted list. Table B.1 in Section B.4 shows the list of available fields. If this parameter is null the list will only include the `MEDIAID` field.

3. **order**: Specifies the sorting criterion for the list. The sorting criterion is represented by one of the following values: `POSTED`, `RATING`, and `TITLE`. Thus the list will be ordered upwardly based on the value specified. If null is provided then by default the list is sorted by `TITLE`.

4. **offset**: From the complete list of videos associated to an account, this parameter specifies the video to start from. For example, if an offset of 4 is set, the returned list will include as first element the 4th video of the complete list existing for an account (3 elements are skipped in the list).

5. **maxResults**: Specifies the maximum number of media items to retrieve in the resulted list. The maximum value is 100.

6. **filters**: Filters offer a solution to restrict the elements returned by the method to those that match certain criteria. Multiple filters will further restrict results using boolean AND logic. See the example below to get a better understanding of its usage.

### Listing B.38: Synchronous listFilteredMedia Example

```java
S9SynchronousAPI api = S9SynchronousAPI.getInstance();
MediaFieldsEnum[] mediaFields = new MediaFieldsEnum[] {
  MediaFieldsEnum.MEDIAID, MediaFieldsEnum.TITLE,
  MediaFieldsEnum.DESCRIPTION, MediaFieldsEnum.IMAGE
};

int offset = 4;
int maxResults = 50;

FilterSet filters = new FilterSet();
filters.addTagsFilter(new String[] {"tag1", "tag2"});
filters.addModerationFilter(ModerationEnum.APPROVED);

S9ResultData result = api.listFilteredMedia("auth-token",
  mediaFields, APIOrderEnum.RATING,
  offset, maxResults, filters);
S9ListMediaResponse resp = (S9ListMediaResponse) result.getResponse();
List<Media> mediaLst = response.getMedia();
```
Asynchronous Version

Listing B.39: Asynchronous listFilteredMedia

```java
public void listFilteredMedia(S9Commons commons,
    MediaFieldsEnum[] fields, APIOrderEnum order,
    int offset, int maxResults, FilterSet filters);
```

Arguments:

1. **commons**: It contains internally the authorization value and an instance of the callback interface to call. See Section B.2.2 for more detail.

2. **rest of the arguments**: Please review the synchronous version described above.

Listing B.40: Asynchronous listFilteredMedia Example

```java
S9AsynchronousAPI api = S9AsynchronousAPI.getInstance();
S9Commons commons = new S9Commons("auth-token", new
    S9CallbackHandler()

    public void processCallback(S9ResultData callbackData) {
        S9ListMediaResponse response =
            (S9ListMediaResponse) callbackData.getResponse();
        List<Media> mediaLst = response.getMedia();
    }
});

MediaFieldsEnum[] mediaFields = new MediaFieldsEnum[] {
    MediaFieldsEnum.MEDIAID, MediaFieldsEnum.TITLE,
    MediaFieldsEnum.DESCRIPTION, MediaFieldsEnum.IMAGE
};

FilterSet filters = new FilterSet();
filters.addCategoryFilter("category-id");
api.listFilteredMedia(commons, mediaFields,
    APIOrderEnum.TITLE, 1, 20, filters);
```

B.3.10 searchMedia

This method introduces basic search functionality to find videos in a Screen9 account. The given search term will be compared against the **TITLE**, **DESCRIPTION** and **TAGS** of every media item. The method returns a maximum of 100 results. The results are ordered by **TITLE**. Every media item contains the fields: **TITLE**, **DESCRIPTION** and **MEDIAID**. The result of the method is a **S9ResultData** object that internally contains an instance of the class **S9ListMediaResponse**.
Synchronous Version

Listing B.41: Synchronous searchMedia

```java
public S9ResultData searchMedia(String authToken, String searchText)
```

Arguments:

1. `authToken`: It contains internally the authorization value to access a media account.

2. `searchText`: The term used to search media items.

Listing B.42: Synchronous searchMedia Example

```java
S9SynchronousAPI api = S9SynchronousAPI.getInstance();
S9ResultData result = api.searchMedia("auth-token", "search term");
S9ListMediaResponse resp = (S9ListMediaResponse) result.getResponse();
List<Media> mediaLst = resp.getMedia();
```

Asynchronous Version

Listing B.43: Asynchronous searchMedia

```java
public void searchMedia(S9Commons commons, String searchText);
```

Arguments:

1. `commons`: It contains internally the authorization value and an instance of the callback interface to call. See Section B.2.2 for more detail.

2. `searchText`: The term used to search media items.

Listing B.44: Asynchronous searchMedia Example

```java
S9AsynchronousAPI api = S9AsynchronousAPI.getInstance();
S9Commons commons = new S9Commons("auth-token", new S9CallbackHandler() {

    public void processCallback(S9ResultData callbackData) {
        S9ListMediaResponse response = (S9ListMediaResponse) callbackData.getResponse();
        List<Media> mediaLst = response.getMedia();
    }

});

api.searchMedia(commons, "search term");
```
B.3.11 searchMedia

This method offers a very similar functionality than the method presented in section B.3.10. It also offers search functionality, however, this method introduces more arguments to configure a search. The result of the method is a S9ResultData object that internally contains an instance of the class S9ListMediaResponse.

Synchronous Version

Listing B.45: Synchronous searchMedia

```java
public S9ResultData searchMedia(String authToken, String searchText, MediaFieldsEnum[] fields, APIOrderEnum order, int offset, int maxResults, FilterSet filters, MediaFieldsEnum[] constraints)
```

Arguments:

1. **authToken**: It contains internally the authorization value to access a media account.
2. **searchText**: The term used to search media items.
3. **fields**: It is an array of elements of the enum type MediaFieldsEnum. This array specifies the fields that need to be returned for every media element contained in the resulted list. Table B.1 in Section B.4 shows the list of available fields. If this parameter is null, the list will only include the MEDIAID field.
4. **order**: Specifies the sorting criterion for the list. The sorting criterion is represented by one of the following values: POSTED, RATING and TITLE. Thus the list will be ordered upwardly based on the value specified. If null is provided then by default the list is sorted by TITLE.
5. **offset**: From the complete list of videos associated to an account, this parameter specifies the video to start from. For example, if an offset of 4 is set, the returned list will include as first element the 4th video of the complete list existing for an account(3 elements are skipped in the list).
6. **maxResults**: Specifies the maximum number of media items to retrieve in the resulted list. The maximum value is 100.
7. **filters**: Filters offer a solution to restrict the elements returned by the method to those that match certain criteria. Multiple filters will further restrict results using boolean AND logic.
8. **constraints:** An array of `MediaFieldsEnum` to search within. Only `TITLE`, `DESCRIPTION` and `TAGS` are valid values.

**Listing B.46:** Synchronous searchMedia Example

```java
S9SynchronousAPI api = S9SynchronousAPI.getInstance();
MediaFieldsEnum[] mediaFields = new MediaFieldsEnum[] {
    MediaFieldsEnum.MEDIAID, MediaFieldsEnum.TITLE,
    MediaFieldsEnum.DESCRIPTION, MediaFieldsEnum.IMAGE
};

int offset = 4;
int maxResults = 50;

FilterSet filters = new FilterSet();
filters.addCategoryFilter("category-id");
filters.addTagsFilter(new String[] {["tag1", "tag2"]});
filters.addModerationFilter(ModerationEnum.APPROVED);

MediaFieldsEnum[] constraints = new MediaFieldsEnum[] {
    MediaFieldsEnum.TITLE, MediaFieldsEnum.DESCRIPTION,
    MediaFieldsEnum.TAGS
};

S9ResultData result = api.searchMedia("auth-token", "search-term",
    mediaFields, APIOrderEnum.TITLE, offset, maxResults, filters,
    constraints);
S9ListMediaResponse resp =
    (S9ListMediaResponse) result.getResponse();
List<Media> mediaLst = resp.getMedia();
```

**Asynchronous Version**

**Listing B.47:** Asynchronous searchMedia

```java
public void searchMedia(S9Commons commons, String searchText,
    MediaFieldsEnum[] fields, APIOrderEnum order,
    int offset, int maxResults, FilterSet filters,
    MediaFieldsEnum[] constraints)
```

**Arguments:**

1. **commons:** It contains internally the authorization value and an instance of the callback interface to call. See Section B.2.2 for more detail.

2. **rest of the arguments:** Please review the synchronous version described above.
Listing B.48: Asynchronous searchMedia Example

```java
S9AsynchronousAPI api = S9AsynchronousAPI.getInstance();
S9Commons commons = new S9Commons("auth-token", new
S9CallbackHandler()
{
    public void processCallback(S9ResultData callbackData) {
        S9ListMediaResponse response = (S9ListMediaResponse)
        callbackData.getResponse();
        List<Media> mediaLst = response.getMedia();
    }
});
MediaFieldsEnum[] mediaFields = new MediaFieldsEnum[] {
    MediaFieldsEnum.MEDIAID, MediaFieldsEnum.TITLE,
    MediaFieldsEnum.DESCRIPTION, MediaFieldsEnum.IMAGE
};
int offset = 4;
int maxResults = 50;
FilterSet filters = new FilterSet();
filters.addCategoryFilter("category-id");
filters.addTagsFilter(new String[]{"tag1", "tag2"});
filters.addModerationFilter(ModerationEnum.APPROVED);
MediaFieldsEnum[] constraints = new MediaFieldsEnum[] {
    MediaFieldsEnum.TITLE, MediaFieldsEnum.DESCRIPTION,
    MediaFieldsEnum.TAGS
};
api.searchMedia(commons, "search-term", mediaFields, APIOrderEnum.
    TITLE, offset, maxResults, filters, constraints);
```

B.3.12 setModeration

This method sets the moderation value to a media item. The moderation field
indicates whether a media item is approved or unapproved to be published. The
result of the method is a S9ResultData object that internally contains an in-
stance of the class S9Response. The response object only shows the status of
the operation, no method-specific value is returned from the server.

Synchronous Version

Listing B.49: Synchronous setModeration

```java
public S9ResultData setModeration(String authToken, String mediaId,
    ModerationEnum moderationValue)
```
Appendix B. API Documentation

Arguments:

1. **authToken**: It contains internally the authorization value to access a media account.

2. **mediaId**: The identifier of the video or media to rate.

3. **moderationValue**: It is an enum value of the `ModerationEnum` class. It specifies the moderation status to set in the server. Only users with Administrator rights are allowed to execute the method.

**Listing B.50**: Synchronous setModeration Example

```java
S9SynchronousAPI api = S9SynchronousAPI.getInstance();
S9ResultData result = api.setModeration("auth-token", "media-id", ModerationEnum.APPROVED);
```

Asynchronous Version

**Listing B.51**: Asynchronous setModeration

```java
public void setModeration(S9Commons commons, String mediaId, ModerationEnum moderationValue)
```

Arguments:

1. **commons**: It contains internally the authorization value and an instance of the callback interface to call. See Section B.2.2 for more detail.

2. **mediaId**: The identifier of the video or media to rate.

3. **moderationValue**: It is an enum value of the `ModerationEnum` class. It specifies the moderation status to set in the server. Only users with Administrator rights are allowed to execute the method.

**Listing B.52**: Asynchronous setModeration Example

```java
S9AsynchronousAPI api = S9AsynchronousAPI.getInstance();
S9Commons commons = new S9Commons("auth-token", new S9CallbackHandler()
{
    public void processCallback(S9ResultData callbackData)
    {
        S9Response response = callbackData.getResponse();
    }
});
api.setModeration(commons, mediaId, ModerationEnum.APPROVED);
```
Appendix B.  API Documentation

B.3.13  uploadMedia

This method enables to upload media to the Screen9 server and associate it to a specific Screen9 account. When the media is uploaded by default the moderation status is set to UNMODERATED and it is not published (see method set-Moderation in Section B.3.12). This return value of this method only indicates an error in case it occurred, but no specific response from the server exists. This method depends directly on getUploadURL (see Section B.3.3) to obtain the authorization token and the url where the item will be uploaded.

Synchronous Version

Listing B.53: Synchronous uploadMedia

```java
public S9ResultData uploadMedia(String authToken, String uploadURL,
                                String filePath, S9UploadParameters mediaParams)
```

Arguments:

1. **authToken**: It contains internally the authorization value to upload a video. **NOTE**: The token used for this method is the token returned by getUploadURL (see Section B.3.3).

2. **uploadURL**: The URL that specifies where the video is going to be uploaded. This URL is obtained by the method getUploadURL (see Section B.3.3).

3. **filePath**: The absolute filesystem path where is the file to be uploaded (for example /sdcard/videos/demo.3gp).

4. **mediaParams**: This is a plain object that contains values of required fields for each media item uploaded: TITLE, DESCRIPTION, CATEGORYID and a TAG.

Listing B.54: Synchronous uploadMedia Example

```java
S9SynchronousAPI api = S9SynchronousAPI.getInstance();
S9UploadParameters mediaInfo = new S9UploadParameters("title",
            "cat_id", "description", "tag");
S9ResultData result = api.uploadMedia("auth−upload−token",
            "http://upload_uri", "/sdcard/demo.3gp", mediaInfo);
```
Appendix B. API Documentation

Asynchronous Version

**Listing B.55:** Asynchronous uploadMedia

```java
public void uploadMedia(S9Commons commons, String uploadURL,
                        String filePath, Screen9UploadParameters mediaParams)
```

Arguments:

1. **commons:** It contains internally the authorization value and an instance of the callback interface to call. **NOTE:** The token contained internally must be taken from the result obtained after execute `getUploadURL` (see Section B.3.3).

2. **uploadURL:** The URL that specifies where the video is going to be uploaded. This URL is obtained by the method `getUploadURL` (see Section B.3.3).

3. **filePath:** The absolute filesystem path where is the file to be uploaded (for example `/sdcard/videos/demo.3gp`).

4. **mediaParams:** This is a plain object that contains values of required fields for each media item uploaded: `TITLE`, `DESCRIPTION`, `CATEGORYID` and a `TAG`.

**Listing B.56:** Asynchronous uploadMedia Example

```java
S9AsynchronousAPI api = S9AsynchronousAPI.getInstance();
S9Commons commons = new S9Commons("auth−upload−token",
                                   new S9CallbackHandler()
                                   {
                                       public void processCallback(S9ResultData callbackData)
                                       {
                                           // Process callback data
                                       }
                                   }
                                   )
                                   );
S9UploadParameters mediaInfo =
                                   new S9UploadParameters("title", "cat.id", "description", "tag");
api.uploadMedia(commons, "http://upload_url",
                "/sdcard/demo.3gp", mediaInfo);
```
B.4 MediaFieldsEnum List

<table>
<thead>
<tr>
<th>MediaFieldsEnum</th>
<th>CATEGORYID</th>
<th>CATEGORYNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUDIOCODEC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>DOWNLOADS_COMPLETED</td>
<td>DOWNLOADS_STARTED</td>
</tr>
<tr>
<td>DURATION</td>
<td>FILENAME</td>
<td>FORMATS</td>
</tr>
<tr>
<td>IMAGE</td>
<td>IMAGES</td>
<td>MEDIAID</td>
</tr>
<tr>
<td>NUMRATINGS</td>
<td>ORIGINALSIZE</td>
<td>POSTED</td>
</tr>
<tr>
<td>PROCESSED</td>
<td>PROCESSING_PROGRESS</td>
<td>PROPERTIES</td>
</tr>
<tr>
<td>RATING</td>
<td>REFERENCE</td>
<td>STATUS</td>
</tr>
<tr>
<td>TAGS</td>
<td>TITLE</td>
<td>TRANSCODEDSIZE</td>
</tr>
<tr>
<td>USERID</td>
<td>USERNAME</td>
<td>VIDEOCODEC</td>
</tr>
</tbody>
</table>

*Table B.1: MediaFieldsEnum*


[75] L.L.E. Yu. “From requirements to architectural design-using goals and scenarios”. In: *First International Workshop From Software Requirements to Architectures-STRAW’01*. Citeseer, p. 22.