Designing a surrounding environment for a mobile application creation editor

David Eriksson

January 17, 2013
Master’s Thesis in Computing Science, 30 credits
Supervisor at CS-UmU: Anders Broberg
Examiner: Håkan Gulliksson

UMEÅ UNIVERSITY
DEPARTMENT OF COMPUTING SCIENCE
SE-901 87 UMEÅ
SWEDEN
Abstract

The SATIN project has the aim to enable end users to create their own mobile applications without prior programming experience, and in this way assist non programmers in the transition to the digital society. In this thesis an environment has been designed and developed, that surrounds the existing SATIN mobile application creation editor, with the purpose of supporting the developers and facilitate collaboration.

Based on the theory of End User Programming, a web-based environment built in Django, a Python framework, was developed and the existing editor was adapted to function together with this environment. The resulting environment is to be seen as a proof of concept for the future progress of the SATIN project.
Contents

1 Introduction 1

2 Problem Description 3
  2.1 Methods .................................................. 4

3 End User Programming 7
  3.1 An introduction to EUP .................................. 7
  3.2 Understanding the users in EUP systems ............... 8
  3.3 Design guidelines for EUP environments ............... 8
  3.4 Supporting the user in EUP .............................. 9
  3.5 Potential pitfalls when building EUP systems .......... 9
  3.6 Summary .................................................. 10

4 Accomplishment 11
  4.1 System structure .......................................... 11
  4.2 System design ............................................ 12
  4.3 System implementation ................................ 12

5 Results 15
  5.1 System structure .......................................... 15
  5.2 System design ............................................ 22
  5.3 System implementation ................................ 34
  5.4 Reflection regarding the result and EUP ............... 43

6 Conclusions 45
  6.1 Limitations and future work ............................ 45

7 Acknowledgments 49

References 51
## List of Figures

2.1 The SATIN editor, with several components already dragged to the canvas . 3
2.2 The Component library, the building blocks for the SATIN editor . . . . . 4

5.1 A screenshot from the Red Foundry editor (taken 2012-11-06) . . . . . . 16
5.2 An overview of the different pages in the SATIN environment . . . . . . 22
5.3 Mockup: The SATIN landing page when users are not logged in . . . . . 23
5.4 Mockup: The SATIN start page when logged in (version 1) . . . . . . . . 24
5.5 Mockup: The SATIN start page when logged in (version 2) . . . . . . . . 25
5.6 Mockup: The SATIN start page when logged in (version 3) . . . . . . . . 26
5.7 Mockup: The SATIN Help request page . . . . . . . . . . . . . . . . . . . 27
5.8 Mockup: A single SATIN Help request . . . . . . . . . . . . . . . . . . . . 28
5.9 Mockup: The SATIN Component request page . . . . . . . . . . . . . . . 29
5.10 Mockup: A single SATIN Component request . . . . . . . . . . . . . . . . 30
5.11 Mockup: Creating a Component request from the editor . . . . . . . . . . 31
5.12 Mockup: A single Tutorial viewed in the SATIN editor . . . . . . . . . . 31
5.13 Mockup: The SATIN Tutorial page . . . . . . . . . . . . . . . . . . . . . . . 32
5.14 Mockup: The Component page listing all developed components . . . . . 33
5.15 Overview of the preexisting Portal database . . . . . . . . . . . . . . . . . 35
5.16 Overview of the implemented Django models in the SATIN environment . 35
5.17 Screenshot: The SATIN environment start page . . . . . . . . . . . . . . 36
5.18 Screenshot: The development start page, listing the developer’s components . 36
5.19 Screenshot: The development start page, listing the developer’s components . 36
5.20 Screenshot: The Component request overview page . . . . . . . . . . . . 39
5.21 Screenshot: A single Component request . . . . . . . . . . . . . . . . . . . 40
5.22 Screenshot: The menu list from within the editor . . . . . . . . . . . . . . 40
5.23 Screenshot: A single Help request . . . . . . . . . . . . . . . . . . . . . . . 41
5.24 Screenshot: The notification page, with links to the relevant notified content 42
5.25 Screenshot: Excerpt from the SATIN splash page . . . . . . . . . . . . . 42
Chapter 1

Introduction

We live in a digital age. The mobile market is steadily advancing. In 2010, 10.9 billion mobile applications were downloaded globally [13]. Even though the amount of available applications and download numbers are truly impressive, the fact remains that the end users have not yet had their application quota filled. To fully become a member of this new digital and mobile society, individuals should be able to build applications by themselves, based on their individual needs. Building mobile applications is no easy task however, and often requires substantial programming knowledge.

The SATIN project\(^1\) began in 2009 with the aim to enable end users to create their own mobile applications without prior programming competence or experience. By allowing end users to create mobile services, and share them with other end users, the SATIN project can assist in the transition to the digital society. This is done with an editor where users can create their own applications by combining different components, such as buttons, textfields, maps and so on. The entire process requires no programming knowledge, and components can be built and added by experienced programmers to increase the potential of applications made through the system. The goal of this thesis is to focus on the environment surrounding this editor: a system that should offer support and tools during all phases of the application building workflow, as well as facilitating collaboration between users with different level of experience.

This Master thesis was conducted at CodeMill\(^2\), an Umeå-based company that offers high-quality system- and software development. CodeMill have taken part in the SATIN project through development work regarding the graphical editor and the editor API.

Initially, an introduction to the problem is given. Then the theory of End User Programming is discussed in relation to the problem. The chapter after describes the methods and tools used for the three phases of work: System structure, System design and System implementation. The result is presented following a similar structure. Finally, some conclusions are made and a collection of the future work is presented.

\(^1\)http://www.satinproject.eu
\(^2\)http://www.codemill.se
Chapter 2

Problem Description

SATIN currently offers an editor for using direct manipulation on graphical components to modify and connect them to each other, in the quest to create mobile web applications. Technical information regarding component development and how the editor works can be found in the SATIN wiki [14].

![Figure 2.1: The SATIN editor, with several components already dragged to the canvas and connected](image)

This graphical programming language offers programming novices the possibility to build their own applications, but an editor alone cannot fully help the end users to become application developers. A supporting environment is needed that beyond keeping track of and saving all the user’s applications offers help and tutorials. This environment is the goal of the master thesis.
2.1 Methods

The first step will be to conduct a scientific in-depth study of the theory of End User Programming, a notion that is closely related to the SATIN project. With this theory as a ground, the practical work phase will begin.

Initially, a functionality specification will be made, structuring the possible supporting features that such a system could offer. To achieve this, information and inspiration will be gathered through many relevant channels such as users, the SATIN design team and so on. The resulting information will be structured as the functionality base for the next part: the graphical phase.

A graphical structure will follow the functionality specification, which will combine the functionality in a non-intrusive support system surrounding the editor, since the editor is the most important part of the SATIN environment. The new functionality should integrate seamlessly with the editor functionality.

Finally, an initial version of the SATIN environment will be implemented, possibly with limited functionality due to time limitations. This will combine the results from the func-
tionality specification phase and the graphical choices made in the second phase into a functioning system with the support from the End User Programming theory from the ground up. The task of choosing suitable implementation frameworks and methods is part of this implementation phase.
Chapter 3

End User Programming

To establish a theory that the thesis work could be based on, End User Programming (from now on abbreviated EUP) was chosen as the scientific in-depth study. This section is the result of this study part of the thesis and introduces the notion of EUP. Based on the literature, an understanding of the concept has been established. Different topics within EUP have been analyzed regarding the SATIN environment, such as understanding the EUP users, potential pitfalls when building EUP systems, and how to offer the support that EUP users need: all relevant factors for creating a surrounding environment for the SATIN editor.

3.1 An introduction to EUP

The notion of EUP covers much ground and can be viewed as specific branch of End User Development, which can be defined as follows:

"End-User Development can be defined as a set of methods, techniques, and tools that allow users of software systems, who are acting as non-professional software developers, at some point to create, modify or extend a software artefact." [12]

Moving from the more general definition of EUD, EUP can be defined as "programming to achieve the result of a program primarily for personal, rather public use." [9]. The important distinction between ordinary programming and EUP would in this case be the programmers intention regarding the users of the application or program built: EUP is more focused on the specific situation and needs of the End user programmer. Both of these definitions fits into the SATIN attitude that users should be able to create "their own" applications.

The spectrum of EUP actions ranges from small tasks such as creating custom addition functions in spreadsheets, creating procedures in programs like Photoshop, setting up email filtering rules and creating custom applications. The art of EUP is often abstracted into three levels of complexity: Modify, Build, Code, commonly called the gentle slope [12, 6]. This is relatable to the SATIN components: At initial level, a user can modify a default application simply by changing parameters. The next step contains integrating components with each other and creating something possibly unique and new. The third level illustrates the coding of actual components and submitting them to the SATIN system, an action that in SATIN will be handled by the actual programmers and will not be directly applicable to the main target audience. However, since it will be possible for experienced programmers to use the SATIN system for application creating as well, the border is fairly imprecise.
Since the goal of EUP could be formulated as blurring the limits between authors and consumers [1], removing the distinction between "expert" users and "ordinary" users in the SATIN environment could be a step towards a label-less environment with flexible role changing between being the student and being the expert.

3.2 Understanding the users in EUP systems

To be able to create a functional and supporting EUP environment, there is a need to fully understand the users that the system will support. There are many human factors in play when a system should be customized to support development of applications. People have cognitive biases that affect their program creating ability [10], and these need to be considered.

For instance, people tend to collect just enough information for making a decision. This will not necessarily be the best one. As an example, this is of relevance when a suitable component should be chosen: if the first component works, the second and third component will not be considered better choices.

Another factor is that short term goals always gets prioritization over long time goals: i.e a quick way to make a program work would be preferred to a more generic solution that later could be used in other applications as well.

It is also important to understand that in most cases, the end users do not have software engineering experience, nor do they have a particular interest in this area [3]. If they have a goal (i.e an application they want to build) and feel that it requires skills or abilities too far outside of their knowledge domain, they might not accept the environment as a suitable tool.

3.3 Design guidelines for EUP environments

When designing an EUP environment, there are many relevant elements to consider. The user satisfaction with the environment can be seen as inversely proportional to the domain scope and variability in the user population [16]: the wider the target group, the harder it will be to motivate them to use the system, since it will not be specifically targeted to their individual needs.

Considering that the SATIN target group is not limited to a specific domain, this will be an important challenge. The one big shared trait is their inexperience of programming, and this should be utilized. Programming concepts such as for-loops, boolean statements and so on should be based on beginners' perception of programming, not on how programming languages usually function [5].

Most communities do not value time spent learning. Just as with the gentle slope, the effort to enter and start living in the community should not be large. Functionality could be displayed and hidden, depending on the need for it, to keep the overall experience light [6] - since shown information is visible at the cost of visibility of other information [8]. The workflow should be determined by the user, not the environment, and the functionality should match this.

Communication possibilities could be valued higher than tools, especially communication between different levels of users [7]. If a user needs help with a simple application error, a quick look from a more experienced application builder could beat a debugging tool by far. It would thus be valuable to support and enhance possibilities for novices to get in contact
3.4 Supporting the user in EUP

The guidelines for designing EUP systems contained many relevant features and thoughts that support the users in a more general way. There are also some aspects to consider when focus is placed on actual application creation. The environment could help the user overcome developing barriers by suggesting solutions or pointers in the correct direction [4]: if the user is stuck on how to connect a social media-component with another component, the system could present a suitable bridge component if that would be an appropriate solution. Design is inherently hard, so facilitating creativity by supplying examples is an important task for environment to handle [11].

Support can also be given through the use of todo-lists containing important tasks that are required to complete before an application could be considered "working”. By highlighting the components that still need an input and/or output before they can function normally, the user can easily identify what needs to be done [10]. This could also be taken further by automated error checking throughout the application [5]. When considering that an important part of the SATIN concept is to create an easy-to-use system for programming novices, this method could seem to be the wrong approach. The goal should be to design an environment where the development process is so user-friendly, no errors will be made during development. In practice, a compromise will likely be most suitable.

3.5 Potential pitfalls when building EUP systems

It is easy to assume that helping out the user in any way would be a positive behavior from the system. By using default values if an input is not submitted, by automatically setting input and output connections and through automatic handling of errors, the development phase could flow remarkably smoother. This could however backfire: if the system automatically makes an incorrect adaption, the trustworthiness of the system is damaged and the user might lose confidence in his or her skills [16]. The error handling and adaption will have to avoid taking too much responsibility at the risk of introducing errors and likewise.

A typical problem with EUP is that applications turn out too low quality for the purpose they were created [3] and this has to be addressed through limiting the possible error rate, while still maintaining the feeling that the user is in control.

As mentioned in Section 3.2, users usually do not search further than the first found working solution. This could present a quality and usability issue in a system such as the SATIN environment. The quality of the applications could be reduced if the most suitable components are not found when searching, since similar but less suitable components might be in the way. It all comes down to the tradeoff between an open environment where a large number of components are user generated (which is the SATIN approach), and the less open environment where the number of components are kept at bay, to ensure quality and reduce redundant overflow in the building material. With a growing component library, the problem of choice arises: if the user wants to choose between the different choices of components, this could result in huge time losses for basic simple parts of application building [10].

It can be concluded that many problems and challenges with EUP systems comes down to not drowning users in information and choices, and neither in automatic functionality that is not simple to follow. This common *featuritis* [7], the art of adding too much functionality...
and features to a system, is common among IDEs and likewise when they adopt to a wider user base. Since SATIN has a very broad target group domain, the possible functionality range is huge. To keep the system simple to use, a key factor will be to keep this range within the borders of acceptable amounts of functionality.

3.6 Summary

Through this chapter many aspects of EUP have been covered, ranging from an introductory part discussing EUP in general to discussing risks with EUP systems. Focus has also been dedicated to understanding and supporting the EUP users, in that the systems built with EUP as a theory ground should suit their target audience regarding knowledge requirements and remove the feeling that technology is something complex and advanced. With the increased knowledge about EUP, the thesis will now enter the practical part starting with the system structure, described in the next chapter.
Chapter 4

Accomplishment

With End User Programming as a theoretical base for the thesis work, the process to create a SATIN environment could begin. This process was divided into three main phases: System structure and content, system design, and system implementation. The different phases are described in separate sections below, with descriptions of the methods and tools that were used during the process. It should also be noted that all phases have been supported through continuous collaboration together with the SATIN design team.

4.1 System structure

To compile an all inclusive list of meaningful and possible functionality in the SATIN environment, input was gathered from many different sources:

- Work regarding the SATIN environment design was done during a university course in the spring 2012. The resulting material was a useful source for inspiration, including notes from a workshop with members of the SATIN project group
- Input from meetings with the SATIN project group
- Studying competitors in the simple mobile app creation field
- Studying the existing SATIN editor, and the basic site that currently functions as a Component uploading portal for developers
- Ordinary brainstorming regarding desired functionality

Using the combined result of this input, the functionality was structured into a number of items formulated according to a part of Scrum methodology called userstories: “As a [end user role], I want [the desire] so that [the rationale]”. This collection of stories was to be used as a priority list for both the design and implementation phases: since the magnitude of the feature collection implied that the required implementation time would collide with the project time restrictions, this restriction of which features makes it to the first version was crucial.

The Feature Story list was sent to the SATIN design team and the team members were to order the different stories after priority. This feedback resulted in a base for the selection of features to bring into the first prototype. It is hard to strictly grade individual features in a complex system since they often depend on each other and their usefulness varies depending
on what other features the system supports, but the results were used as guidelines for making this selection.

4.2 System design

Looking initially at the top tiers of the functionality derived from the System Structure phase, connections between functionality parts could be made. This resulted in a summary of all relevant pages and their internal connections. Based on this listing, graphical mock-uping could take place. The focus during the mockup process was to combine the system functionality into a graphical layout that neither intrudes nor seems to complex for beginners to handle. It is relevant to note that this phase was not to be hindered much from what was possible to implement based on time and editor modification restraints, but instead focus on how the implementation would be under perfect circumstances. The implementation phase would be better off with too many mockups rather than too few.

Sketching were initially done by hand, in an iterative process to quickly produce different versions of the basic layout. Once the groundworks were done, the mockup tool Balsamiq Mockups\(^1\) was chosen to create more changeable and clear sketches, with focus still on functionality. Balsamiq Mockups also offers possibilities for interactive prototypes, such as generating clickable PDFs. This was practical when the system design was tried out by members of the SATIN design team and the supervisors. Due to time restraints, the mockups were not tested on users from the target group and is thus not user verified.

4.3 System implementation

To understand the scope of the existing relevant applications for the SATIN environment, those applications were thoroughly examined. The EditorGUI communicates with the EditorCore and the EditorCore communicates with the existing SATIN Portal. This simple Portal, together with database and REST-API, was a key point to inspect when planning to build a SATIN environment that would work with the existing EditorCore and EditorGUI with as little configuration workload as possible. A Component Portal with component administration possibilities was developed by CodeMill, closely integrated with the EditorCore, whose functionality also would be represented in the SATIN environment.

A server was setup at CodeMill for running a branch of the EditorCore, built in Java. This java implementation was to be modified in order to work with and towards the new SATIN environment. This included modifications regarding how data was exchanged with the database through the REST-API. XML was used in the earlier Portal and because of simplicity reasons as well as time savings when implementing the new REST-API, JSON was now preferred. Since CodeMill had no implementation hours left for the EditorCore, this was made as a part of the implementation phase. Focus would be placed on getting the environment up and running together with the EditorCore without spending too much implementation time.

To implement the actual SATIN environment, a Python framework called Django was used for the task: Django is "a high-level Python Web framework that encourages rapid development and clean, pragmatic design"\(^2\) and was chosen particularly because CodeMill had experience with this framework and could offer support if needed. To facilitate the

\(^1\)http://balsamiq.com/products/mockups
\(^2\)http://djangoproject.com/
4.3. System implementation

front-end development process, *Twitter Bootstrap*\(^3\) was used as a front-end framework. The simplicity of Bootstrap would also work well with the desired simplicity of the environment design. Since most of the design was described through mockups, and the Bootstrap framework already got fonts, colors and likewise covered, the front-end phase went very smooth. A Django module called *Tastypie*\(^4\) was used to enable a REST-API connected to the Django database, that the EditorCore could communicate with.

The EditorGUI is built in *Backbone.js*\(^5\) and this was integrated into the Django application to benefit from the Django authorization, sessions and likewise. Modification to the EditorGUI was kept small since the editor was not the primary focus: only needed adjustments were done and no work was put into making the look of the editor coherent with the Bootstrap-styled environment. The goal, as earlier stated, was to create a prototype with application building functionality in place to fully show how the flow of the environment could be, without getting stuck on any of the individual parts of the implementation.

\(^3\)http://twitter.github.com/bootstrap/
\(^4\)http://tastypieapi.org/
\(^5\)http://backbonejs.org/
Chapter 5

Results

This section describes the results in detail from the three main phases, System structure and content, System design, and System implementation.

5.1 System structure

It should be mentioned that the SATIN editor was stand-alone in the beginning of this thesis: there was no surrounding environment and all created applications belonged to a single test user. Therefore the Feature list is built from scratch, beginning with such a basic and self-explanatory item as Login functionality.

It is worth noting that there are some competitors that similarly to SATIN target a non-programming audience. Because of this, some functionality will of course seem similar. The big difference between the SATIN project and other similar systems such as iGenApps\(^1\) and RedFoundry\(^2\) (see Figure 5.1) is basically cost and required technical knowledge. SATIN is completely free and the required technical knowledge is low, not only considering programming ability. Other systems may not require you to know programming, but might still assume a fairly high technical knowledge. SATIN is unique in this regard. By analyzing what feels too complicated with other systems, it is easier to make these features simpler in the SATIN system.

Feature story list

The System structure phase resulted in a comprehensive list of 27 items. They are individually described below, in the Scrum Story format. The features are approximately ordered by importance. Note that items marked with an asterisk in the end requires significant changes in EditorCore and/or EditorGUI and their possibility of implementation is therefore limited (more about this in Chapter 5, Section 3: System implementation).

As a user I want to log into SATIN to access all my previous work

The system should provide basic user functionality, with register possibilities and so on. This is a cornerstone for the entire system and fairly self-explanatory why it is highly prioritized.

---

\(^1\)http://www.igenapps.com

\(^2\)http://redfoundry.com
As a component developer I want to upload and view my developed components directly in the SATIN environment, so I don’t have to be logged into several systems.

The goal of End User Programming could as mentioned be formulated as blurring the limits between authors and consumers [1]. One step in the direction of removing the distinction between expert users and "ordinary" users in the SATIN environment is to keep a tight integration between the ordinary environment and the developer environment. A developer should simply be an application builder with some extra possibilities regarding component development, the remaining functionality should stay the same.

As a user I want to share my applications through social media to reach my target group.

The overall mission of SATIN is to empower end-users to create mobile services, and to share these services with other end-users. Therefore, sharing functionality and easy access to the applications created through SATIN is very important. Facebook and Twitter sharing seems as the most suitable choices.
As a user I want to send a Component request in case I don’t find the component I need, to fill the gap the missing component would cause for me and other application builders

The concept of a community with developers and “ordinary” users revolves around collaboration. Not everything can be built with a limited set of components. The unexperienced people who want to build applications can help each other, learn from others, and combine the available building blocks in many ways, but if some important part is missing, such as a component for social media for instance, this domain cannot really be reached. This is where the developers come into play. By constantly adding new parts, the potential gaps can be covered and with time, the building potential will be close to infinite. However, to make sure developers focus on the most needed parts first, there is need for a way to collect this information. When users need a component they cannot find, they can create a request or upvote an existing request, to indicate to developers that this is a needed part for the SATIN editor.

As a new visitor I want to test the SATIN editor without having to register first, to easily be able to try out SATIN*

Since people in the target audience do not necessarily possess technical knowledge, it is important to allow them to try out the SATIN editor early so that the actual simplicity of the editor can be felt and tried. By offering the possibility to play around in the editor without requiring any kind of signup, the barrier for testing can be kept low.

As a new visitor I want to see a describing Tutorial video on the start page so I understand how SATIN works

It is important to clearly describe the entire process of SATIN to the potential users. By showing a tutorial video that briefly covers the simplicity of the editor (showing some drag and drop and a few connections) as well as the nontechnical process of publishing an application (build, test, publish, share), the users will be well armed to try out the editor themselves and feel confident that they will manage.

As a user I want to create a help thread so I can get help from the community with my application building

The possibility of receiving help is a key factor in a functioning environment around the SATIN editor. The users can ask for help when needed, and help out others when they can. An interesting functionality would be to include the current application in the help request so that the other users can view the application directly and thus easier find and give instructions how to correct the error(s). This application transparency should however be optional.

As a user I want to receive notifications to find out if someone have commented on my tutorial, replied to a help thread and if a component request has been marked as "in progress" et cetera

Notifications are a common part of many modern systems. Users need a way to get information about relevant happenings such as that someone commented their help request or that a component they need now is in development. This kind of information is optimal to distribute through a simple and subtle notification system.
As a user I want new versions of components to not simply replace earlier versions but instead offer me an update option, to prevent that a new version breaks my application*

In the current form of the editor, there is only one version of a component available: the newest one. This could cause problems in case a component goes through a major change, such as adding required inputs: an application that was in the building phase will suddenly consist of new errors. To prevent this, there would be need for a subtle possibility to let the user choose if a component upgrade should be done or if the old version should be kept since it is working.

As a user I want to make my application available through a static URL such as "/username/app/”, to differ between preview versions and a "working" live version*

In the current editor, the editor generate a randomized URL whenever you build the application. If you have to update and republish your application, the path changes and old links become obsolete. This is unpractical for many reasons: Sharing becomes a problem since you have to make sure everyone has the correct link, and bookmarks break. By enabling the functionality that the published application always is reachable through the same path, independent from how many times the user change and update the application, this problem is avoided.

As a new visitor I want to see a selection of applications on the start page to get a feeling for what I can build with SATIN

Examples are a useful tool for grasping a new concept. By displaying some of the simple applications built by SATIN users, and letting new visitors see that these applications are published recently, it both shows some of the potential for the SATIN editor as well as indicates that the environment got active users.

As a component developer I want to tag a Component request as "in progress" to indicate that such a component is in the works

As described earlier, Component requests serve an important purpose. But for them to reach their maximum effect, we need to avoid one problem: that several component developers work on the same component. Users also want feedback that there are components being developed based on the requests they create. By letting the component developers tag requests to indicate that they will build this component, both of these things are solved. It also enables a developer to notify everyone who have requested a component about when it is released into the system, directly from the component adding phase.

As a user I want to organize components in categories such as "Favorites", "My components" et cetera to easier keep them organized*

A constantly growing component library means that the components a user once used successfully in an application could be hard to find the next time they are needed. Some kind of structuring where the most useful components could be stored would be an efficient way of accessing these components.
As a user I want to create tutorials/examples to help and facilitate for other users*

Enabling possibilities for help is central in the system as mentioned. By creating tutorials and examples, users can help other users in a structured way to inform about a regular reoccurring question such as how a specific component should be used. A component developer could for example create and attach a tutorial for a newly created component.

As a user I want to let other users see how my application is built so they can help me finding errors or learn from how I have built*

Accessing another user’s application to see the components and connections would be really efficient when trying to find an error, but also when learning for yourself. Allowing other users to view an application, possibly by turning it into a tutorial application with a viewable inside, are effective and helpful.

As a user I want to test my application in a controlled manner such as a sandbox testing environment to be assured it is working as intended*

As with all development, testing is important to ensure quality and that everything functions as planned. When it comes to mobile applications, it could be hard to test it simply by opening the application since other factors often come into play (such as location, accelerometer et cetera). A sandbox testing environment with for instance simulated GPS location and likewise would be a solution to many of these problems.

As a user I want to search for components in an Application Store-like manner as well as browse a selection of the most used components, to get a feeling for which components are the most interesting for me

The SATIN environment should not only offer help, it should also offer inspiration and facilitate creativity. By informing which components are popular and highlighting new groundbreaking components, users can get inspired and build applications they had not thought of otherwise.

As a component developer I want to connect a tutorial directly to a component to make it easier for the users to understand it

Understandability is important when building components. Every individual block should be so simple to use that no tutorial is needed. Nevertheless, some components might have interesting uses in different scenarios and this can be well described though a tutorial. By connecting a tutorial directly to the component, users looking for a tutorial relevant to this particular component will know that it is a suitable choice.

As a user I want to add additional tags on other users’ components to increase searchability in the system*

SATIN is a largely crowdsourced system, with users supplying the building blocks for other users to build with. As components are used in new ways, they might be incorrectly described. For instance, a component with data-retrieving functionality might originally be built for Facebook data, but later on the same component is used to retrieve data from Twitter. To offer the best possible searchability for users, tagging of components is a way of
keeping the material updated without pressuring all developers to keep track on how their components are used.

As a user I want to build applications from Application Templates to get an easier start*

Applications can be categorized. Many applications have the same basic functionality and only some things are changed, such as the involved accounts (social media) and locations. By enabling applications to be build from templates, the development process will go much faster and avoid errors to be made in the groundwork since the Templates could be well assembled. Similar to tutorials, these can also be very effective to study when users are learning the system.

As a user I want to rate components and tutorials, to indicate to other users regarding the quality and usefulness*

Rating is a common part of the web, where everything from products to sites can be rated. By enabling rating of content the users will handle the separation between good and bad content in a true crowdsourced fashion and thus the visible material quality will stay high.

As a user I want to charge money for my applications to earn money on my application building

The possibility of making money is often a motivator for users. If the possibility to sell SATIN built applications existed, the interest could even reach the commercial market. There are however consequences to consider, such as how the payment possibilities would affect the ecosystem, and whether it would be possible to charge money for using components as well.

As a user I want to build on an application simultaneously with another user, to benefit from collaboration*

Just as with whiteboards and the multi-editing of Google Docs, simultaneous collaboration is an effective way to combine the ideas of several users into one result. By tinkering with components together and testing out different solutions, application building can be fun and effective. This could of course be done by sharing a computer, but this is not always a plausible solution.

As a user I want to create Personas to facilitate the development process for an application

One discussed concept for the SATIN environment is a so called Toolbox, where different tools for enhancing the application development process would be available for the users. These tools have great potential and can function similar to plugins to the SATIN environment or editor. A tool to create Personas to identify the target group for an application could be seen as a Proof of Concept of this toolbox.
As end user I want a list of my chosen applications available in text file format, to be able to use this in a background execution application for Android

As a side project in SATIN, an application was built for running SATIN applications in the background since they are web applications and thus cannot do this natively. This project could be worked into the environment by offering the possibility of listing applications that users want to run on their phone.

As a user I want to receive achievements for different actions in the environment (creating applications and tutorials et cetera) to receive credibility and recognition for my contribution to the ecosystem

Gamification is a new and powerful technique for motivation in any kind of system [2]. It applies the mechanics of gaming to non-gaming activities: earning points for answering questions, receiving levels for creating tutorials and components, and getting achievements for simple helpful tasks are a few examples of gamification. By adding this concept to the SATIN environment, collaboration could be enhanced further and participation in the system could increase.

As a user I want to subscribe to other users to receive notifications if they create tutorials, components et cetera

Some users might be producers of exceptionally good tutorials and components. Just with YouTube-channels and likewise, it would be beneficial to follow or subscribe to such a user to be informed if new content is added to the system.
5.2 System design

Graphical mockups of the SATIN environment was created with the assistance of Balsamiq Mockups. Focus was placed on creating a simple and clean structural design which would invite even non-technical users to click around, and not be intimidated by pages cluttered with information. The content to mockup was based on the prioritized Feature story list. This section will present most of these mockups and describe them in detail, including interesting design choices. It should be noted that even though all mockups went through several iterations, most of them will only be shown in the version chosen for implementation.

In Figure 5.2 a summary of the relevant pages is presented. Since the editor is the core functionality of the SATIN, connections between the editor and other pages is in focus. Three main features are identified and can be seen to the top right: Help requests, Component requests and Tutorials. Developers have a fourth main feature in Component development.

Figure 5.2: An overview of the different pages in the SATIN environment
The SATIN environment start page has two important tasks: First of all to convince visitors in the target group to go ahead and test the SATIN editor: it should appear easy to use and this page should quickly explain how simple and nontechnical this editor is. Secondly it should present login functionality for the already registered users. Figure 5.3 displays the planned layout. The first thing to notice is that the textual information is very limited: since the editor is so simple to use, its explanation should not require more than a few lines to text. At the bottom of the page, mobile applications built with SATIN is on display: this both indicates that the SATIN ecosystem is alive as well as gives inspiration and motivation regarding what can be built with the editor. This splash screen would also be a good place for displaying a tutorial video that covers the basics of the editor.

Figure 5.3: Mockup: The SATIN landing page when users are not logged in
Once a user registers or logs in, the start page will shift focus. Since the mobile applications are central in the SATIN environment, these are displayed here with easy possibilities for editing, viewing and sharing. An application should be viewed as a standalone but simple part of the system, and it was decided to represent applications as stand-alone boxes with a clean appearance because of this. An application should not have to be much more than just an icon and a title, similar to how they appear in modern smartphones.

Figure 5.4: Mockup: The SATIN start page when logged in, user applications listed vertically

Apart from applications, the start page also displays excerpts of recent Help requests, Tutorials and Component requests to give the users (both developers and regularly users) an overview of recent content and quick access to this content. Figure 5.4 and 5.5 display two different versions of how to layout the relevant components. Vertically aligning the Application boxes allows for better space usage, for instance every content box to the right...
can display longer titles. Horizontal alignment does put more focus on the applications however and was the preferred choice in the end.

For a new user, there will naturally be no applications on this page. To improve the flow towards the first application, a placeholder application box encourages the new user to begin creating one. This can be viewed in Figure 5.6.

Figure 5.5: Mockup: The SATIN start page when logged in, user applications listed horizontally
The first of the main features is Help requests: users can create a simple post, directly from the editor or through the environment menu, where they describe an application building related problem, and other users in the environment can post answers. The layout for this kind of feature is common on the web (for instance pages like stackoverflow.com) and the SATIN environment does not deviate much from the typical layout. Figure 5.7 displays a list of the recent Help requests, with search possibilities and a filtering option for the user’s own posts only. A single Help request (shown in Figure 5.8) is displayed in a similar traditional fashion with the question in top and user answers listed at the bottom. The user could also connect an application to the request. This would make questions more easily answerable, since the others can find out the answer by looking at this particular application (without possibility to modify it, of course).
Figure 5.7: Mockup: The SATIN Help request page listing all the available Help requests, possibly filtered by a search query.
Figure 5.8: Mockup: A single SATIN Help request, along with comments and information regarding the connected application

Component requests follows the design choices that were done for Help requests, since they share many common traits such as displaying a question and a number of comments. Figure 5.9 displays a list of Component requests similar to the one for Help requests, and Figure 5.10 shows the single Component request view. Developers can tag the request as "in development" when they add a comment, to indicate that this will soon be developed. A natural place for creating a Component requests is when a search in the editor Component library does not return any results, as described in Figure 5.11. This creates a natural flow from the need of a component to the request of such a component. If an existing request seems to be suitable for this search, it will be displayed to prevent duplicate requests.
5.2. System design

Figure 5.9: Mockup: The SATIN Component request page listing all the available Component requests, possibly filtered by a search query
Figure 5.10: Mockup: A single SATIN Component request, along with comments and possibly a developer tag
5.2. System design

Figure 5.11: Mockup: The context for creating a Component request directly from the Component library search in the editor, based on the search terms that did not return any suitable components.

*Tutorials* are the third of the three major environment features. Figure 5.13 displays available tutorials in the by now recognizable fashion. When entering a tutorial, the user gets to see a view-only-application in the editor (see Figure 5.12), with the added functionality of notes from the creator describing the different parts of the creation. Since the Component library is of no use here, this space is instead used for the tutorial comments and questions.

Figure 5.12: Mockup: A single Tutorial viewed in the SATIN editor, together with the additional describing notes and the tutorial comments.
Component development is the cornerstone of the SATIN editor and if a user is a component developer, these pages will be available in the SATIN environment menu - along with the earlier mentioned possibility to mark Component requests as "in development". Figure 5.14 displays a developer's created Components, and the possibility to edit such a component. A component should be possible to connect both to a tagged Component request as well as to an example tutorial for this component. Under the Development menu, there is also a page for listing Component requests currently tagged by the user.
Figure 5.14: Mockup: The Component page listing all developed components for the currently logged in component developer, as well as editorial possibilities for the currently chosen component.
5.3 System implementation

The preexisting REST-API was quite complex and used XML for sending and receiving data. A database overview for the preexisting Portal can be seen in Figure 5.15. Based on this data, two Django models were extracted as the base for the editor functionality: Composition and ComponentInfo. These two, complemented with the default Authorization user model in Django, allowed the new SATIN environment to now go past what was already done and look at new functionality. Figure 5.16 displays a simple overview of the built Django models. The EditorCore java source code was modified to handle JSON instead of XML, both when reading from and sending data to the SATIN environment. By using the Django Tastypie module, the SATIN environment now offers a complete REST-API for receiving and saving compositions and components. A typical JSON output for a single composition and a single component could look as follows, where in the composition case blockFile contains the base64-encoded string for the editor composition metadata, and in the component case blockFile and codeFile are encoded Component XML and Javascript, respectively.

Sample Composition JSON:
```
{
    blockFile: "PD94bWwgdm [...]",
    compositionId: "11",
    createdDate: 1357123491000,
    description: "Application description text here",
    modifiedDate: 1357123491000,
    moduleName: "App_c8328247b89c4553b81139b0bcc6e47c",
    name: "Application name here",
    resource_uri: "/api/v1/composition/11/",
    status: "BETA",
    user: "/api/v1/user/1/",
    version: 1,
    versionDescription: ""
}
```

Sample Component JSON:
```
{
    blockFile: "PeRpmerg [...]",
    codeFile: "LyogDQog [...]",
    createdDate: 1354709004000,
    description: "Displays a message on the screen",
    email: "test@test.se",
    example: "",
    id: "3",
    metaFile: "",
    modifiedDate: 1354781917000,
    name: "Alert",
    rating: 6,
    resource_uri: "/api/v1/componentinfo/3/",
    status: "BETA",
    version: 2,
    versionDescription: "Alert component version 2"
}
```
5.3. System implementation

Time and technical limitations due to not focusing on the EditorCore and the editor itself resulted in a somewhat limited implementation with some functionality not yet implemented. This is further described in Chapter 6, Section 1: Limitations and Future work. Here follows a description of the functionality that made it to the implementation stage.

As a user I want to log into SATIN to access all my previous work

Figure 5.15: Overview of the preexisting Portal database

Figure 5.16: Overview of the implemented Django models in the SATIN environment

Figure 5.17 displays the start page of the SATIN environment. In this case, the user satin is logged in and can browse its created applications.
Chapter 5. Results

Figure 5.17: Screenshot: The SATIN environment start page, with the user "satin" logged in. Applications are displayed horizontally at the top and recent Help and Component requests are displayed at the bottom. Since the user is a developer, the Development menu is visible.

As a component developer I want to upload and view my developed components directly in the SATIN environment

The development section of the environment is in place and offers uploading and editing functionality for the components (as seen in Figure 5.18 and 5.19). The developer can connect a component directly to a Component request, as well as save different versions of a component to avoid breaking applications if major changes are implemented.

Figure 5.18: Screenshot: The development start page, listing the developer’s components, with possibility to list all versions of them.
As a user I want to share my applications through social media to reach my target group

Application sharing was made possible through Facebook and Twitter (see Figure 5.20 for details).

As a user I want to send a Component request in case I don’t find the component I need, to fill the gap the missing component would cause for me and other application builders

Component requests were implemented and Figure 5.21 displays the creation form of such a request. Figure 5.22 shows the listing of all Component requests and Figure 5.23 contains
the view of a single request. Users can upvote a request written by someone else instead of creating a duplicate request. Filtering options exists for "only untagged requests" as well as sorting on highest upvoted request first, to facilitate for developers. The original author of the request can edit or delete the request if desired. Developers can also tag a request as "in development", as asked for in another Feature story.

Figure 5.21: Screenshot: Excerpt from the Add new Component request page
Figure 5.22: Screenshot: The Component request overview page, with checkboxes for sorting functionality
As a new visitor I want to test the SATIN editor without having to register first, to easily be able to try out SATIN

The SATIN editor is accessible without registering, directly from the start page. The visitor can connect and customize components, and build the application to view the result, but environment specific functionality like saving and asking for help is disabled (as can be seen in Figure 5.24).
As a user I want to create a help thread so I can get help from the community with my application building.

Help requests have as mentioned much in common with Component requests and their look and feel reflect this. Figure 5.25 displays a single Help request. Functionality for marking a specific answer as the solution to the problem is in place to make it easier for users with similar problems to quickly find out which answer best answered the question.

As a user I want to receive notifications to find out if someone have commented on my tutorial, replied to a help thread and if a component request has been marked as "in progress" et cetera.

A simple notification functionality is in place, using `django-notifications` (see Figure 5.26). Notifications can be accessed through the circular badge next to the username.

---

3https://github.com/brantyoung/django-notifications
Figure 5.26: Screenshot: The notification page, with links to the relevant notified content

As a new visitor I want to see a selection of applications on the start page to get a feeling for what I can build with SATIN

Figure 5.27 displays the bottom of the satin start page when a user is not logged in: this displays the latest built applications through the SATIN editor and offers the possibility to view them.

Figure 5.27: Screenshot: Excerpt from the SATIN splash page, containing the latest built applications though the SATIN editor and links to try out these applications

Django sessions are used to authorize the EditorCore towards the SATIN environment REST-API: when a user logs in, a session key is generated and this is used to connect to the EditorCore through the EditorGUI. When the EditorCore calls the REST-API, this key is required to extract the JSON data. This prevents unauthorized access to the environment data.
5.4 Reflection regarding the result and EUP

To summarize this chapter, a reflection regarding the resulting parts and the theory from End User Programming is in place. Initially in Chapter 3 an introductory view was given. The statement that EUP would be more focused on programming things with a very small target group, possibly the user alone, plays well into how the SATIN environment lets users customize their applications and thus often creates applications for very specific use cases. The three levels of complexity mentioned in the same section is not entirely present as long as the editor examples or templates remain to be implemented: simply modifying an already built application would represent the first level of complexity and as it is in its current state, this level is not really implemented. By integrating the users and developers into the same environment, with almost identical possibilities regarding for instance Help and Component requests, the distinction between different skill levels is blurred. Unexperienced users can feel like "one of the developers" even though they lack programming experience whatsoever.

A simple, clean and minimalistic graphical environment design was chosen to avoid impressions that the learning curve would be too steep or the required knowledge would be too high. The risk of drowning users in information and features is avoided in this simple and light graphical design with only a few features focused at a time. Since there are a few things to grasp when first time users learn to use the editor, the environment should not add any complex things to this list and risk overloading the non-technical new users.

Functionality such as upvoting Component requests and search tags for finding earlier similar posts are useful when trying to avoid duplicated content and overflow in information. By keeping the content sharp and focused, the problem of choice can be avoided and less time can be spent sifting through the material searching for something suitable. It should also be noted that most of the implemented functionality (along with the remaining items on the Feature story list) is focusing on collaboration and users helping each other, rather than individual features.

As conclusion it can be said that even though the notion of EUP is radiating through the result as a whole, there is still much to be done and considered. Primarily, much work can be done with the editor. Solution suggestion, integrated application building todo-lists and assistive input/output-handling is a few SATIN editor specific EUP concepts that could be considered in upcoming versions of the graphical editor.
Chapter 6

Conclusions

A workflow for developing mobile applications is now in place. The registered user can enter the editor and build on its mobile application, test and save it, and later share or further modify this application. If help is needed, the first tools for getting help are in place. If important building blocks are missing, these can be requested. Mobile application builders and experienced component developers are integrated into the same environment. With a theoretical in-depth study of End User Programming as a starting point and working from several angles to produce desired functionality and look for the surrounding ecosystem for the SATIN editor, it was possible to implement a functioning SATIN environment. The two big parts of the SATIN project (editor and environment) are finally combined.

With that said, the SATIN project is an ongoing process. The environment is far from finished and should be viewed as a proof of concept rather than a complete implementation. By refining and further improving this result, a complete and active SATIN environment could be reached. In the everlasting battle for non programmers to catch up with the mobile world and be a part of digital society, this work is a step in the right direction.

6.1 Limitations and future work

Due to the time limitations and the choice not to spend too much time on the existing parts (EditorCore, EditorGUI), a number of tasks from the Feature story list remains undone and will remain as future work. Highly prioritized but still not implemented functions will be discussed briefly.

As a user I want new versions of components to not simply replace earlier versions but instead offer me an update option, to prevent that a new version breaks my application

Component versions are supported, but update possibilities are not supported in the current editor implementation. As it is now, applications simply keep using the older version unless manually switched out.
As a user I want to make my application available through a static URL such as "/username/app/", to differ between preview versions and a "working" live version

Built applications are made available through /username/[app-id], but the ability to create separate preview and live builds of applications remains to be built.

As a user I want to create tutorials/examples to help and facilitate for other users

Tutorials are an important part in the SATIN environment but the ability to create special tutorial applications is yet to be implemented in the EditorCore and EditorGUI.

As a user I want to let other users see how my application is built so they can help me finding errors or learn from how I’ve built

This is tightly connected with the ability to create tutorials. Enabling other users to view applications without modification possibilities remains to be implemented.

Below follows a list of remaining Feature story list items that were not implemented, mainly because of low priority and time limitations.

- As a new visitor I want to see a describing Tutorial video on the start page so I understand how SATIN works
- As a user I want to organize components in categories such as "Favorites", "My components" et cetera to easier keep them organized
- As a user I want to test my application in a controlled manner such as a sandbox testing environment to be assured it is working as intended
- As a user I want to search for component in an Application Store-like manner as well as browse a selection of the most used components, to get a feeling for which components are most interesting for me
- As a component developer I want to connect a tutorial directly to a component to make it easier for the users to understand it
- As a user I want to add additional tags on other users’ components to increase search-ability in the system
- As a user I want to build applications from Application Templates to get an easier start
- As a user I want to rate components and tutorials, to indicate to other users regarding the quality and usefulness
- As a user I want to charge money for my applications to earn money on my application building
- As a user I want to build on an application simultaneously with another user, to benefit from collaboration
As a user I want to create Personas to facilitate the development process for an application

As end user I want a list of my chosen applications available in text file format, to be able to use this in a background execution application for Android

As a user I want to receive achievements for different actions in the environment (creating applications and tutorials et cetera) to receive credibility and recognition for my contribution to the ecosystem

As a user I want to subscribe to other users to receive notifications if they create tutorials, components et cetera

Apart from the Scrum story list, there are a few other items worth mentioning in the future work section. First out is some legacy features from the earlier Portal implementation: The Type upload section from the earlier Component Portal is not implemented in this proof of concept. Neither is the component icon upload possibility (to let the SATIN server host the component icon image specified in the XML). The component voting functionality seemed to be incomplete in the original and is not yet entirely fixed.

Some mockuped features are left to be implemented later. For instance, the functionality that Component requests and request possibility shows up directly in the Component library search, described in Figure 5.11.

The user handling has room for improvement. In its current implementation, there is no way for a user to change from ordinary user to developer once registered. There is also no form of admin users in this current state - content can only be deleted by its original creator. This of course requires some thought before a system like this would go live.

The editor itself has potential to better suit the new SATIN environment. For instance, the test mode when a user is building in the editor without registering is useful, but needs to be complemented with the possibility to register directly in the editor and save the application they just built to reach its full potential. Some of the functionality in the editor (such as sharing) are also obsolete and does not work like sharing does from the SATIN environment. The visual look of the editor could also benefit from adapting to the SATIN environment look, at least regarding things such as buttons and fonts.

There is also the concern of privacy. In its current form, there are no such thing as private components or private web-applications. The ownership of the created material is not specified. These things are connected to potential open source models and likewise, and those factors have not gone past the discussion phase. What terms users have to agree with to use the system remains to be decided.
Chapter 7

Acknowledgments

Special thanks goes out to the following:

Supervisor Andreas Jakobsson at CodeMill
Supervisor Anders Broberg at Umeå University
CodeMill employee Daniel Hedlund for support regarding the EditorCore
The SATIN project group with the SATIN design team in particular
CodeMill for support and providing a lovely office
References


