

Institutionen för datavetenskap
Department of Computer and Information Science

Final thesis

**A platform for third-party applications
on the web**

by

Marcus Abrahamsson

LIU-IDA/LITH-EX-G--13/038-SE

2013-08-20



Linköpings universitet

Final Thesis

**A platform for third-party applications
on the web**

by

Marcus Abrahamsson

LIU-IDA/LITH-EX-G--13/038-SE

2013-08-20

Supervisor: Jesper Strige, Ipendo Systems AB

Examiner: Patrick Lambrix, Linköping University

Abstract

A trend seen on the web today is to create a platform where externally developed applications can run inside some kind of main application. This is often done by providing an API to access data and business logic of your service and a sandbox environment in which third-party applications can run. By providing this, it is made possible for external developers to come up with new ideas based on your service. Some good examples on this are Spotify Apps, Apps on Facebook and Salesforce.com.

Ipendo Systems AB is a company that develops a web platform for intellectual properties. Currently most things on this platform are developed by developers at Ipendo Systems AB. Some interest has though risen to enable external developers to create applications that will in some way run inside the main platform.

In this thesis an analysis of already existing solutions has been done. These solutions were Spotify Apps and Apps on Facebook. The two have different approaches on how to enable third-party applications to run inside their own service. Facebook's solution builds mainly on iframe embedded web pages where data access is provided through a web API. Spotify on the other hand hosts the third-party applications themselves but the applications may only consist of HTML5, CSS3 and JavaScript.

In addition to the analysis a prototype was developed. The purpose of the prototype was to show possible ways to enable third-party applications to run inside your own service. Two solutions showing this were developed. The first one was based on Facebook's approach with iframing of external web pages. The second was a slightly modified version of Spotify's solution with only client-side code hosted by the main application. To safely embed the client side code in the main application a sandboxing tool for JavaScript called Caja was used.

Of the two versions implemented in the prototype was the Iframe solution considered more ready to be utilized in a production environment than Caja. Caja could be seen as an interesting technique for the future but might not be ready to use today. The reason behind this conclusion was that Caja decreased the performance of the written JavaScript as well as adding complexity while developing the third-party applications.

Acknowledgement

I would like to thank the people at Ipendo Systems AB for letting me do my thesis work there. It was exciting to see how the work at a medium sized IT company is at a daily basis.

I would also like to thank the developers at the Caja project at Google for answering many of my questions.

Lastly I would like to thank my family and friends for supporting me under some rough weeks at the beginning of this thesis.

Contents

1 Introduction	1
1.2 Background	1
1.3 Purpose	1
1.4 Limitations.....	1
1.5 Method.....	2
1.6 Literature studies	3
1.7 Outline.....	3
2 Background	5
2.1 Google Caja	5
2.1.1 How Caja is used	6
2.2 Iframe	8
2.3 Internet Security.....	8
2.3.1 Cross-Site Scripting (XSS).....	8
2.3.2 Redirecting the window	8
3 Study of platforms.....	9
3.1 What is an Application Platform?	9
3.2 Requirements on the platforms to analyze	10
3.3 List of criteria to investigate.....	10
3.4 Existing solutions.....	10
3.4.1 Apps on Facebook	11
3.4.2 Spotify Apps	12
3.4.3 Comparison	14
4 Prototype.....	15
4.1 What the prototype should do	15
4.2 Implementation.....	15
4.2.1 Canvas	18
4.2.2 Caja	18
4.2.3 Web API.....	20
4.3 Create applications for the platform.....	20
5 Discussion.....	23
6 Conclusion	25
6.1 Future work.....	25
References.....	26
Figures	27
Appendix 1	28

Glossary	28
Appendix 2	29
Appendix 3	30

Chapter 1

Introduction

This thesis was the last part of the studies of a bachelor's degree in computer engineering at Linköping University, Sweden. The thesis was written at Ipendo Systems AB in Linköping and should embrace 16 ECTS-points (European Credit Transfer System). The work was about investigating different ways to open up a web platform for third-party applications. This was done by first studying existing solutions on the market. To show how this could be done a prototype was developed.

1.2 Background

Ipendo Systems AB is an IT-company that mostly develops web portal solutions for medium sized enterprises. The company has two different branches. The first branch is a consulting part where systems for other companies are developed. The other branch consists of a product they develop themselves. This is a platform for intellectual properties that is used by companies to keep track of things like copyrights, trademarks and patents. As the moment most things on this platform are being developed by developers at Ipendo Systems. Interest has however lately risen about extending the platform to make it possible for external developers to contribute to it. Inspiration to this possible solution has been taken from e.g. Facebook Apps and Spotify Apps. Cloud platforms like Google App Engine have also been a source of inspiration.

1.3 Purpose

The purpose of this thesis was to look into how a web platform could be opened for external developers. The thesis should include an analysis of already existing solutions on the market. The analysis should look at companies that are well-known in the computer industry and that in some way point out the future of this area. Things that were of interest to look at were which techniques that had been used.

In addition to the analysis, a simple prototype should be developed. This prototype should include a web application opened for third-party applications. The purpose with the prototype is not to be a production ready web application but instead show possible solutions. The prototype should therefore be more of a proof of concept on some of the techniques that could be used for this task.

1.4 Limitations

Limitations were made to be able to concentrate fully on some specific parts. The analysis was limited to examining only two existing solutions on the market. It would probably have been good to look at more solutions to get a better picture of alternative solutions but the time was not considered enough. Limitations were also made early to not look at solutions that included a sandbox for server-side code. This was considered too complex by the author of the thesis.

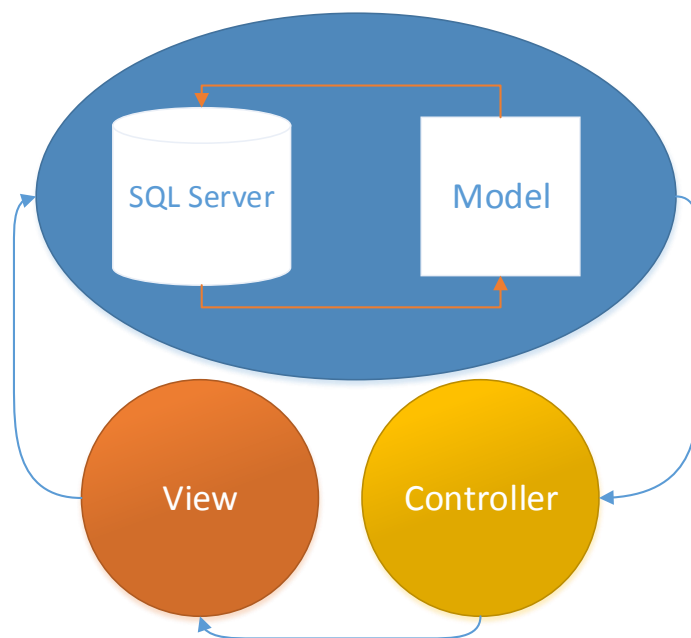
The prototype was also limited to not look at authorization of the web API. Because of the limited time span it was considered more interesting to look at a solution which includes a sandbox for JavaScript.

1.5 Method

Since the purpose of the thesis was to analyze existing solutions and then develop a prototype using some of the approaches found, the requirements of the implementation part were not clear at the beginning. The work started with a lot of information searching about how different web sites had enabled third-party applications. The decision of what solutions to analyze more in depth was built on the type of implementation and the size of the company. A decision was made to focus on two kinds of solutions and see if it was possible to implement one or both of them in the prototype.

When the analysis had been done a design specification was made for the prototype. The development of the prototype was mainly made in ASP.NET MVC 4 with C# as programming language. This development environment was chosen because it is used a lot at Ipendo Systems AB. In addition to ASP.NET MVC JavaScript was also heavily used for some parts of the system.

The architecture of the MVC pattern made it naturally to divide the development into different subgroups. Each subgroup was then developed independently. The development of each subgroup was also divided into smaller areas in accordance to the MVC pattern. These areas are Model, Controller and View. The Model is tightly connected with the database and can therefore be considered as one subgroup. The work of each subgroup started with the Model and database subgroup. After that was the business logic developed in the Controller. Lastly the user interface was developed in the View. These steps were then made several times for each subgroup to add more functionality over time.



Halfway in the implementation a decision was made to skip the authorization part for the web API. Instead another way to publish apps was implemented. The decision was made since Ipendo Systems AB was more interested to see what possibilities sandboxing of JavaScript had.

1.6 Literature studies

Almost all the literature used in this thesis can be found on the internet. The analysis part was mainly done by reading the developer documentation provided by the platforms. A problem with these documentations can be that they are not entirely correct all the time. This was especially true for the documentation on Spotify which had not been updated for a while. Because of this the source code was also an important source to get knowledge about the implementations.

The documentation on the platforms' official pages was complemented by forum posts on the Q&A web site StackOverflow.com. How trustworthy these sources are can be discussed and you should try to back them up with more resources. However, pure coding examples is the big strength with StackOverflow and these are easy to try out yourself.

Under the development of the model that used Google Caja the developers on the Caja project at Google were a great help. They answered many of my questions sent by either email or on their discussion group. This was almost necessary if I wanted to use Caja since the documentation was not the best. The best document that really described how Caja worked was from 2008 which made it accurate at some parts and not so accurate on some at the time of this thesis.

1.7 Outline

The thesis is divided into six chapters. They are in an order that hopefully will give the reader the ability to follow the work from beginning to end. Here is a brief summary of the six chapters.

Chapter 1 – An introduction of the thesis. Contains information about the background and the purpose of the thesis. It also gives information about methods and sources that were used.

Chapter 2 – This chapter contains some theory that could be useful for the reader to understand the remaining chapters. The theory mostly concerns a sandbox environment named Caja which is developed by Google.

Chapter 3 – This chapter contains a study of what an application platform could be. It also contains a study of two existing platforms. The two studied platforms were Facebook and Spotify.

Chapter 4 – This chapter should give the reader knowledge about the prototype that was developed.

Chapter 5 – This chapter discuss some of the things that was found during the studies of the different application platforms. It also discusses the benefits and disadvantages with the two models for third-party integration that were developed in the prototype.

Chapter 6 – This chapter contains a short conclusion of the thesis work and also gives some proposal about future work that could be done in the subject.

Appendix 1 – This is a glossary that explains many of the terms used in this report.

Chapter 2

Background

The purpose of this chapter is to give some basic knowledge about the techniques used later in the prototype. I especially think that the part about Google Caja could be good to read even to more experienced web developers since it explains a technique that is not widely used on the web today. A brief explanation of security problems on the web has also been written to show some of the security problems that can arise with untrusted code on a web page.

2.1 Google Caja

Caja is a tool developed by Google to make third-party HTML, CSS and JavaScript safe to embed in a website. This makes it interesting to use when trying to develop a platform where users can upload applications made of HTML, CSS and JavaScript. If these applications were not controlled in some way it would open up for malicious developers to upload dangerous code. The same thing would be the case if you want to include untrusted advertisement in your page.

Caja makes JavaScript safe by rewriting the code to a subset of JavaScript that embraces the rules of an object-capability language [1]. An object-capability language is a memory-safe object language with encapsulation plus additional restrictions. The restrictions are there to protect the outside world from the objects. A memory-safe object language is a language where an object only can invoke another object if it holds a reference to it. JavaScript is an example on this. JavaScript does however not embrace the rules of an object-*capability* language. C# is an example of a memory-safe object language *with encapsulation*. In these languages the clients of an object can make requests to the object only by its public interface. It is then up to the object itself to decide how to react on the request. The advantage with these languages is that an encapsulated object can ensure that the only way to access its code or change any variables is through the public interface.

In an object-capability language an object can only affect other objects if it holds a reference to it. Objects do not have any references by default. The only way to get a reference to another object is by normal message passing rules. Object references therefore become the same as access authorization. An object can be denied access just by not being given the references that would give them access to the object.

Caja forces the external JavaScript code to follow the rules of an object-capability language. It does so by compiling the JavaScript with a static verifier and inserts runtime checks in the code. Following are some examples on what Caja does with the code.

Forbidden names: Some names are forbidden to use in Caja. No names can for example end with a double underscore “__”. The reason behind this is that in some browsers there is a property named __proto__. Access to this property would give authority to create more objects like it. This violates the principle of least authority and names ending with “__” are therefore not allowed.

No shared global environment: Caja compiles JavaScript code into units of isolation called modules. These modules are then loaded into a container on a host webpage. When the

code is compiled it is given the references to the objects it has authority to use. A webpage can contain several such modules. Each module will however have its own global environment and therefore not be able to communicate if they are not given a reference to the same object.

Removal of dangerous functions: Caja does not contain functions like “with” and “eval”. The with function could be used to change private members of a JSON object. This would break the encapsulation rules of an object-capability language and is therefore not allowed.

2.1.1 How Caja is used

Caja has two parts. The first one is a JavaScript part called the Caja subsystem which is included in the HTML document of a host page. The second part is the compiler, called cajoler, which runs on a server and is responsible for rewriting the guest code.

Figure 1 provides an illustration showing the architecture of a web page with Caja.

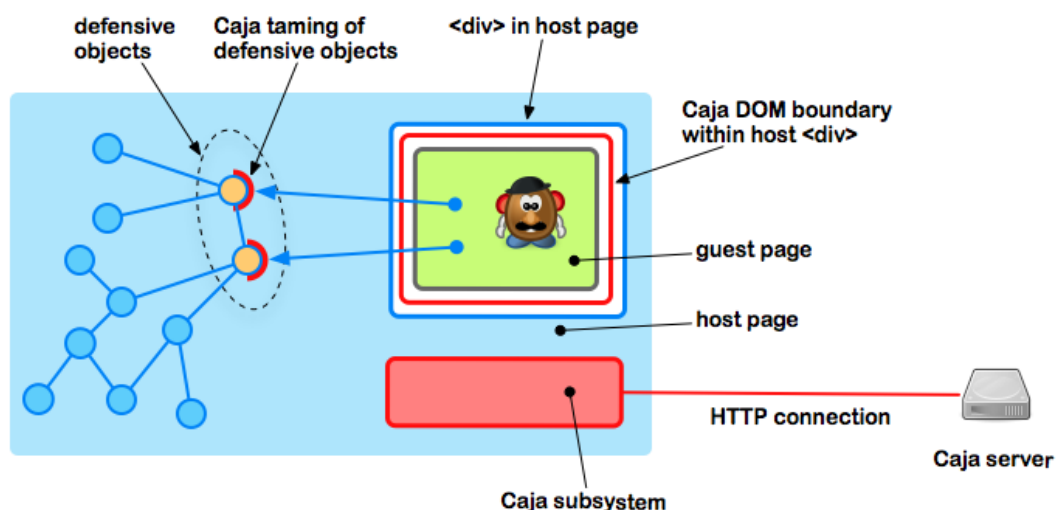


Figure 1 Explanation of the parts in a Caja page

What follows is a more detailed explanation of some of the parts used in Caja.

Host page

This is a regular web page that acts as a container for third-party code. In figure 1 the light-blue background can be viewed as the host page. The host page has the Caja subsystem loaded into it. This is a JavaScript library that is responsible for setting up the environment which the guest code will run in. The host page embeds the untrusted code inside a div element, also called container, which separates the host page from the guest page.

Guest page

This is the third-party application that will be embedded into the provided container in the host page. The guest page is a regular web page written in HTML, CSS and JavaScript.

Policy

This is an object where the rules deciding what the guest code will be able to do are set. One example is the Uri policy. This policy controls the guest codes network access by whitelisting the domains that are allowed.

Defensive object

The objects that you want the guest code to have authority to use are called defensive objects. These are objects created with caution to provide only limited authority to the guest code. These objects implement the policy mentioned before.

Taming

Taming is the process of making the defensive objects safe in the meaning that guest code cannot modify them in ways that were not intended.

The Cajoling process

When a host page wants to load the third-party application, the Caja subsystem first asks a Caja server to make it safe. The Caja server is provided with the policy, the tamed defensive objects and the URL where the app can be found at. The Caja server then gets the guest page by sending a GET request to the provided URL. It then transforms the loaded web page so that its code only has access to the tamed defensive objects. The Caja server then sends the compiled (cajoled) code back to the Caja subsystem in the host page. The Caja subsystems then embed the untrusted but now safe web page into a predefined div element. Figure 2 illustrates this procedure.

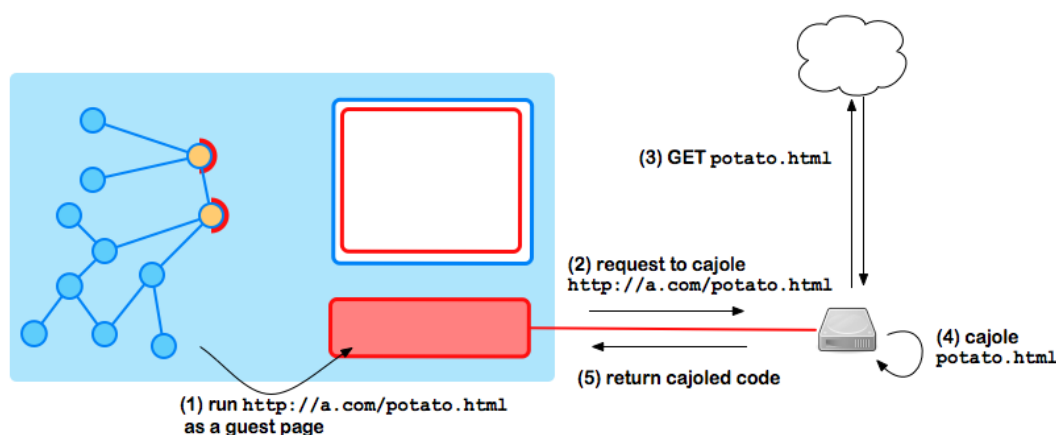


Figure 2 The Cajoling process

From the guest page point of view it will look like it runs within a W3C DOM compliant document object. The document object is limited to the things that the Caja subsystem allows it to do. JavaScript objects in the guest page will not affect the host page. The only way the guest page has to communicate and affect the host page is by calling the tamed objects provided by the Caja subsystem. [2-3]

A code example on how to set up Google Caja can be found in Appendix 2.

2.2 Iframe

An IFrame (Inline Frame) is an HTML element that embeds an HTML document inside another HTML document. This makes it possible to show several HTML documents inside the same page in the web-browser. A good example of when IFrames are used is advertisement on websites. A website that has advertisement from an external source can put an IFrame element in the HTML document and in that way embed an HTML document provided by an for example advertising agency. This way the advertisement agency can update their advertisement themselves without interacting with the developers of the website the advertisement is shown at.

Following is an example of the syntax for embedding an html document inside an IFrame element. The src-property is the address to the HTML document that should be embedded.

```
<iframe src="http://example.com/ad.html "></iframe>.
```

2.3 Internet Security

When you create a platform where external developers can upload code you get a lot of upsides. Unfortunately you also get some downsides, especially when it comes to security. If users can upload JavaScript that will run on your domain you also open up for hackers to use this with bad intentions. To illustrate what this could mean will I describe a couple of attacks that should be taken under consideration when embedding untrusted material on your website.

2.3.1 Cross-Site Scripting (XSS)

This might be the most common of all attacks in modern web browsers. Cross-Site Scripting attacks are a kind of injection attacks which often occur on sites which allow user submitted content. There are mainly two categories of XSS, reflected and stored [4]. In stored XSS, malicious scripts get saved at the server e.g. in a database. The following example illustrates how this can be done.

Example 1(stored XSS): An attacker uploads a new forum post containing malicious JavaScript code. The post will be saved in the server's database. When another user loads the page with the malicious post it will run the JavaScript code of that post. Since this JavaScript is on the trusted domain it has access to things like session variables and cookies. The attacker could read all the cookies and send them to a server controlled by the attacker using an element where the cookies are appended to the src url.

In the case where the users have the possibility to upload web applications with JavaScript stored XSS attacks are something you need to deal with some way.

2.3.2 Redirecting the window

If user submitted code have access to the top window it can redirect the page to whatever URL it likes. A malicious user could use this to redirect the browser window directly on page load. The page being redirected to could mimic the login screen for the original site, tricking the user to enter their credentials. The user tries to login in but the request goes to the attackers server which now has the users credentials.

Chapter 3

Study of platforms

This chapter starts with a brief explanation of application platforms. It is intended to give the reader an overview about what an application platform is and what it can be used for. The rest of the chapter is regarding already existing solutions. Two of these solutions have been analyzed more in depth. These are Apps on Facebook and Spotify Apps.

3.1 What is an Application Platform?

A trend seen on the web today is to create a platform where externally developed applications can run inside your own application. This is often done by providing an API to access data or business logic of your service and a sandbox environment in which the third-party application can run in. By providing this it is made possible for external developers to come up with new innovative ideas based on your service. The platform could also be used internally within an organization to enable reuse of business logic to speed up the development process [5].

Some well-known implementations of this are Facebook Apps and Spotify Apps but they are not alone. A good example of a more business oriented application that also has made it possible for third-party applications is Salesforce.com.

Here is an example when an application platform open for third-party applications could be useful. Let's say that you have created an application for streaming music. You have a large database with music and a good way to make it available for people with a streaming technique. Suppose that you would like to extend this application with more services. This could for example be a service that recommends good new music or maybe a quiz game based on the music in your database. You could probably do this by yourself but might feel that other people would do it better. In that situation would it be a great idea to make it possible for external developers to create new services that run inside your application.

It can be hard to separate the differences between an application platform and a cloud service like Platform as a Service (PaaS) provided by for example Google App Engine. The main difference between them is that Google App Engine provides a platform to create standalone applications on. They provide an environment where the application can run. The platform does however not have any business specific data that the developed application can use. On an application platform however focus is laid on how to integrate third-party apps to an already existing application. This can be done in several ways. Some remind very much of Google App Engine and some have taken a different approach.

The common denominator for most application platforms is the possibility for external developers to extend a main application. This makes it easier for companies to create customized software that runs inside an already good platform. The result of this is that the third-party apps can reuse data and business logic of the platform and bring new functionality to it. If the integration part was not possible, a standalone application would have been needed. This can lead to users that must have several stand-alone applications which can be frustrating since the user needs to have several programs open at the same time and switch between them. With an application platform you can bring all these functionalities into one application.

3.2 Requirements on the platforms to analyze

The systems that were to be analyzed had to meet some basic requirements. Some requirements were hard, their presence were needed to make the system considered as an alternative. Other requirements were soft which means that it would have been good if they were found but not completely necessary. The hard requirements, also called “should” requirements, were the following.

- It should be possible to run third-party applications within the main application. This was the most important requirement that the system should fulfill.
- The solution should not consist of any third-party server-side code executing on the application platform. This requirement was a consequence of the narrow timespan.
- The main application should provide some ability for the third-party application to integrate with data or business logic on the main application.
- Some kind of authentication and authorization for the users of the main application should exist.

The soft requirements, also called “nice to have” requirements, which were considered nice to have but not necessary were the following.

- Some kind of style sheet to make it easier for developers to make third-party applications look like the main application could be good to have.
- It could be good if the platform come from a well-known company whose platform is used by many developers. A well-known company’s solution could in some way stake out the trend and future of the subject and could therefore be interesting to look at.

3.3 List of criteria to investigate

A list of areas to investigate more closely was created based on some of the solutions that had the requirements stated in section 3.2.

- How is the third-party application brought into the main application’s user interface?
- How does the main application create a sandbox environment for the third-party application to execute within?
- How does the main application provide a way for the third-party app to integrate with the data and logic on the main application?
- How are the apps authenticated and authorized?
- Does the main application provide any style sheet that the third-party app can use to look more like the main application?
- How do you register a third-party application?
- Does the main application host the application on their own servers?

3.4 Existing solutions

The analyzing part of the thesis was about looking at already existing implementations of application platforms. In the searches it was found that many companies had taken a similar approach like Facebook which builds on an iframe solution. Among these Salesforce.com and yahoo.com could be found [6]. As a business model Salesforce.com is

more similar to what Ipendo Systems AB would like to have. Since Salesforce's solution was launched in pilot the winter 2012[7] which is after Facebook's launched their solution and the two had so much in common it was fair to believe that Salesforce had taken inspiration from Facebook. Because of this it was it considered interesting to look at Facebook.

Since many companies used the same approach as Facebook it was not easy to find big companies that used another approach that didn't include too much server-side code. The decision was at the end to see how Spotify had done with their application platform. The fact that Spotify's apps run their own browser within the Spotify client makes it a little bit special [8]. The development process is however very similar to regular web development and was therefore chosen to be analyzed.

3.4.1 Apps on Facebook

Facebook's platform fulfills all the "should" requirements that were stated in section 3.2. The platform makes it possible to run third-party applications within the main application and they do not provide any possibility to run third-party code on their own servers. They provide the ability to let the third-party applications integrate with the data on their own application. To use the third-party apps they must first be authenticated. Facebook can also be considered to be a well-known company whose platform is used by many developers and therefore also fulfills one of the "should" requirements.

Facebook is a social networking web site where you can connect with your friends and family. It was founded 2004 by Mark Zuckerberg and has rapidly grown to be one of the biggest web sites of the world. The fact that lots of people spend parts of their lives at Facebook has made it interesting for companies to be there too.

In 2007 Facebook launched the Facebook platform [9]. The platform is a framework that enables software developers to create applications that integrate and interact with the Facebook core. The platform consists of several parts.

To interact with the data on Facebook is something called the Graph API used. This is a web API with REST architecture that makes it possible to use Facebook's data outside Facebook itself. Using the Graph API, a third-party application can read and write data on behalf of the users to the Facebook core [10].

Before the third-party app can access any data through the Graph API the user must authorize the application [11]. When a user accesses an application for the first time it must first give the application some permissions. This is done by stating all the permissions the application wants and then letting the user decide whether to accept it or not. If the user chose not to accept them the application cannot make requests to any private data the user may have.

To integrate an application inside the Facebook user interface is a technique where Facebook does not need any external code at their servers used. The technique they use is to embed an external web site in an iframe element inside the Facebook application. They use the word canvas to describe this technique [12]. One important aspect of iframes is that they go under the same-origin policy. This policy restricts how a document or script from one origin can interact with a resource from another domain [13]. Because of this

JavaScript in an iframe from another domain cannot interact with its parent and vice versa. This is what Facebook utilizes to include third-party apps in a secure way. The technique is not perfect however. The location property of the top window is writeable which means that an iframe can redirect the parent window to another web page.

To register an app on Facebook is a special developer account on Facebook needed. People with this developer account can create new applications. Developers can then choose between several different application types. To integrate an application into the user interface of Facebook is the type “Application on Facebook” chosen. This application type takes one important parameter which is the URL to the web application you want to run inside Facebook. The web page on this URL is then what is loaded in to the iframe element when a user accesses the application.

To make it easier to develop for the Facebook platform there are a PHP and JavaScript SDK. These SDKs are not needed for development but they do simplify lots of it. The JavaScript SDK does e.g. abstract away some of the problems with the same-origin policy which makes it easy to do AJAX calls from the browser [14].

Facebook does not provide any style sheet that external developers can use.

Facebook’s way of enabling third-party applications gives developers the ability to choose the development environment they prefer. This means that the apps can be written in any server side language and be hosted on any server in the world. This makes the development process easy since developers do not have to learn a new language before they can start.

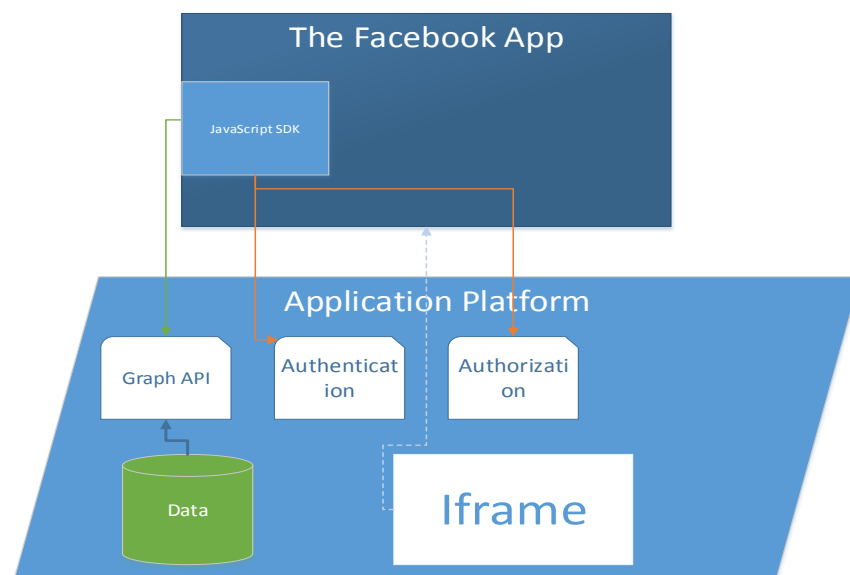


Figure 3 showing the architecture of a Facebook canvas App

3.4.2 Spotify Apps

Spotify fulfills all the “should” and “could” requirements stated in section 3.2. Their platform makes it possible to run third-party inside their own application. The applications consist purely of client-side code. The platform makes it possible for the third-party applications to integrate with the main application. Users need to be authenticated before they can use the applications. Spotify also provides style sheets that can be used to make

the third-party applications look like the main application. Spotify can also be considered as a well-known company.

Spotify is a music streaming service which provides millions of songs online to their customers. They have around 6 million paying subscribers and have been said to be the future of the music industry. As with Facebook it is interesting for companies to be on Spotify since that's where a lot of their customers are. This is especially true for companies involved in the music market like music magazines.

In 2011 Spotify released their platform that gives third-party developers the ability to create applications that will run inside the Spotify client [15]. Even though Spotify seems like a native client on your desktop it has a web browser inside it. Because of this the apps on Spotify are developed with regular web technologies like HTML5/CSS3 and JavaScript. The applications are then loaded into the Spotify client through the web browser.

The web browser in the Spotify client is based on chromium. Some of the standards of HTML5 have though been disabled to meet up with Spotify's guidelines for third-party apps. It is for example not possible to play any audio or video with HTML5 inside the Spotify client. It is also not possible for the user to change the location by manually typing an URL. This could be viewed as a way to in some way sandboxing the third-party applications.

Spotify does not accept that the third-party application contains any server-side code. Therefore the applications can only consist of client-side code written in HTML, CSS and JavaScript. To interact with the Spotify core is a JavaScript API provided. With this API developers get access to things like the music player, playlists and songs.

Spotify provides the ability to use the same style like the rest of the client. This is done by providing CSS and JavaScript files. It is strongly recommended to use these files to create the style of the application. The reason behind this is to make the app feel like it is an integral part of Spotify. If the app is too much unlike the rest of the UI the app risk not being accepted.

Spotify is very clear about what they think an app in Spotify should look like and apps that do not follow these requirements are not accepted. This means that the apps made by third-party developers are quite restricted in what they can do and how they look [16]. Before an app is published it needs to go through several approval stages. This is done by sending the code to Spotify. If the app does not follow the guidelines or is unsafe in any way it will be rejected. After approval the apps are published at the Spotify App finder where they can be found by the users.

Users on Spotify do not have to authorize any permission for the third-party applications. Since users on Spotify do not have much private data there is not much need for setting individual permissions. With the extensive approval process that each application goes through it could however be seen as that Spotify authorize the applications for the users.

The third-party applications on Spotify are hosted by Spotify so the developers do not have to think about maintaining their own servers. If developers would like to update their apps they need to send the updated code to Spotify where it has to go through the approval process before it is published again.

The fact that the apps in Spotify are written in HTML5/CSS3 and JavaScript makes them easy to develop for anyone with knowledge in web programming. One disadvantage is however the fact that you cannot write any backend code. This means that it is hard to save user specific data.

3.4.3 Comparison

The paths Facebook and Spotify have taken to enable third-party apps within their product have both their advantages and disadvantages. Facebook does not have an approval stage where “bad” apps are denied in the same way as Spotify does. This can undermine the product if it has too many bad apps, giving Facebook a bad reputation. Spotify on the other hand is much more concerned about what they accept inside their product. Their guidelines force developers to create apps that will fit in to Spotify and enrich their product.

Facebook’s iframe solution makes it very easy to develop an app to it. Since it only embeds an external web page the developer can work with the web technologies he or she prefers. This makes it very easy to develop apps for the Facebook platform

Spotify hosts the third-party apps themselves which means that they stand for the servers that provides the apps. This could be a problem if they provided lots of heavy applications but since the apps only are made of front-end code the workload on the server is not that high. The fact that you cannot have any server-side code also simplifies the need for sandboxing untrusted code at your own server which can be a very tough job. Facebook on the other hand does not host any of their apps since they only embed external web pages. This means no extra workload on their datacenters. Neither do they have to concern about server side security. It also means that the apps can run server side code on the external server hosting the web page. This can be good if you e.g. want to save user specific data.

Chapter 4

Prototype

This chapter describes the implementation of the prototype. It gives a short scenario of what the prototype should be able to do. A description of the main application is then given. The two different types for third-party applications are then described more in depth.

4.1 What the prototype should do

There should be a main web application that provides some kind of service. This main application should also have some base functionality like the ability for users to register and log in to the application.

The main application should then be extended so that applications developed by external developers can run inside the original main application. It should provide a way for external developers to use the existing data in the main applications database and also contribute with new data to it. The system should make it possible for external developers to register and upload their applications through some kind of form on the main application. A user should then be able to install the applications they are interested in. The installed applications should then be accessible through a home screen where all of them are listed.

4.2 Implementation

The first thing designed and implemented was the main application in which the third-party applications would later be accessed from. The data that is provided by the main application is information about different exercises people can do during a workout session. The main application has one application from the beginning that uses this data. To be able to access this application the users must be registered and logged in.

The main application is built with ASP.NET MVC 4 which forces the developer to use the MVC pattern. The MVC pattern separates the code into different areas based on their purpose. These parts are the Model, Controller and the View. The Model consists of objects that describe some real-world entity. In the main application these are Users and Workout exercises.

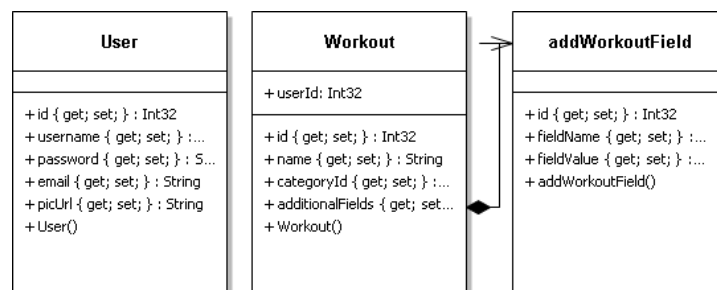


Figure 4 Class diagram of Workout and User

The model objects are persisted in a SQL Server database. A complete view of the database scheme can be found in Appendix 3. To simplify data access a Data Access Layer (DAL) was developed. Each model has its own corresponding interface for a DAL that puts up an

abstraction of how the persisting logic is implemented. This interface is then implemented by a concrete class that contains the implementation of the persisting logic. This makes it a lot easier to retrieve and persist data since you don't have to think about how it is implemented after you have done it the first time. This pattern also makes it easy to change the persisting logic if you want to. One example of when you would like to do so is if you need to change the database provider. A new concrete class that implements the original interface could then be developed that has logic that adapts to the new database provider. Since the new implementation has the same interface as before it can easily be switched out in the code.

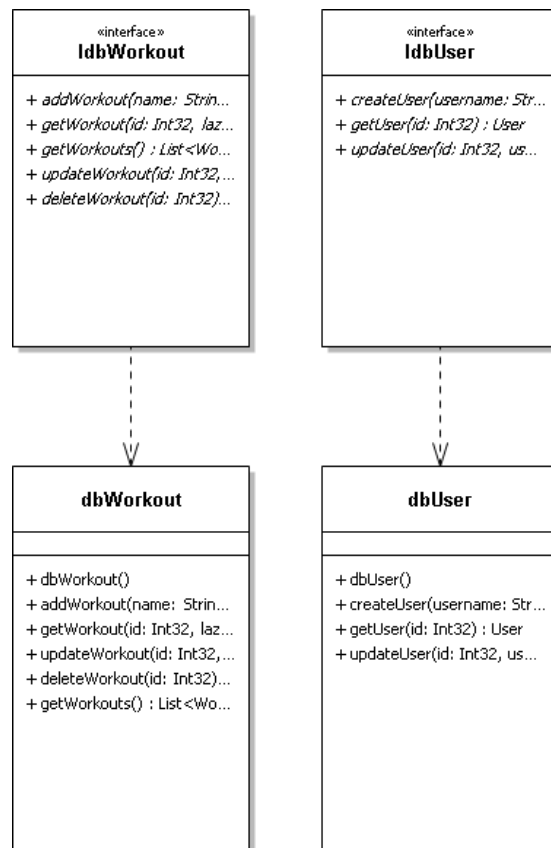


Figure 5 Class diagram of the Workout and User DAL

After the main application had been created focus was laid on creating a way to integrate third-party applications to it. The first solution was to try to almost mimic the Facebook approach with an iframe solution combined with a web API. This was chosen since many companies use this approach and it would therefore be interesting to look into how this could be done.

As the implementation of the Facebook approach progressed it became clear that the work had come to a crossroad. The authorization part which can be very important to an application platform was weighted against trying to create another way to integrate third-party applications. The authorization method intended to be developed was an OAuth service provider. The OAuth model is becoming more and more well-known and is a quite common authorization model on the internet today [17]. To set up an own OAuth service provider is however not a trivial task and would have taken some time. It was therefore

decided to lay the remaining time on trying a different way to integrate third-party applications.

Since Spotify's solution had been studied, an attempt to create a system using some of those parts was made. The part of Spotify's implementation that was chosen was to let the third-party application contain only HTML5, CSS3 and JavaScript. A problem found with this was that Spotify run their apps in their own special web browser. This browser has some capabilities turned off which makes the third-party applications safer to use. There were some security problems that needed to be fixed since the application platform should be able to be accessed from a regular web browser. To accomplish this it was decided to use a sandboxing tool for JavaScript. There are some different JavaScript sandboxes to be found on internet today but many of them force the developers to write code in ways they are not used to. The decision to use Google Caja as sandboxing tool was made on that it let the developers write code in manners they are used to. This should make the development of third-party applications for the platform easier compared with other JavaScript sandboxes.

All this lead to that there are two different kind of ways to develop and integrate third-party applications to the main application in the prototype. Some things are in common for both application types and this was put in a base class. These are things like the name of the application and a picture associated with the application. This base class was then inherited by two child classes that extended it with data specific for that application type.

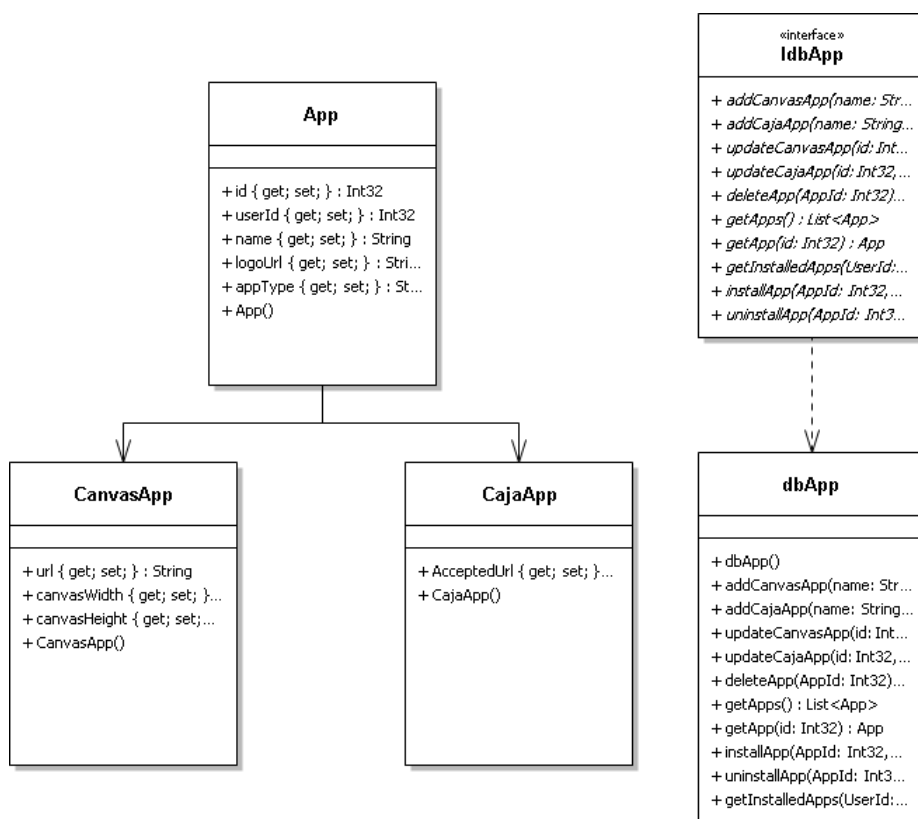


Figure 6 Class diagrams of the App Model and the corresponding DAL

The following subchapters will explain the two applications types that were called Canvas and Caja more in depth.

4.2.1 Canvas

The Canvas app type uses the same architectural approach like Facebook does for their apps. That means that the apps are not hosted by the application platform. Instead the main application embeds an external web page inside an iframe.

To register a canvas application the developer needs to fill out a form. The most important input field is the URL that points to the location where the third-party application can be found. The ability to choose the size of the iframe in which the app will be embedded in can also be chosen.

To develop a canvas application is just like regular web development. The app is developed as a web page and deployed to a server of the developers' choice. This external web page is then embedded into the main application inside an IFrame element.

To interact with the data on the main application a web API is used. The web API embraces a RESTful architectural style which makes it easy to use from any web page. This makes it possible for the canvas application to interact with the data on the main platform. It also makes it possible for the canvas app to feel like an integral part of the main platform, even though it is hosted externally.

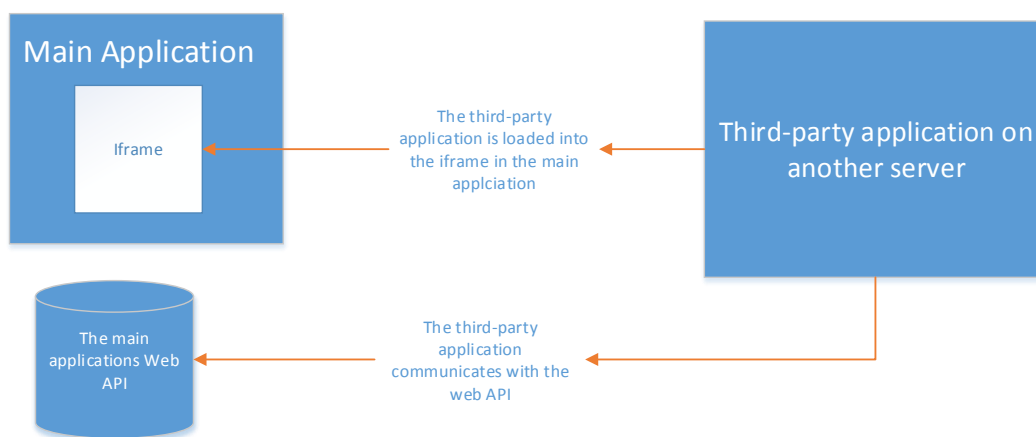


Figure 7 Showing the embedding of an external web application

To make it possible for the canvas application to see which user that is currently logged in, the main application sends a POST request with the currently logged in users userId when loading the application. The canvas app can then use this userId to access user specific data. The security of this implementation is not very good because there is almost no authorization. Authorization was however not included in the scope of this thesis.

4.2.2 Caja

The Caja applications use a different approach to integrate third-party applications, yet with some similarities. In contrast to the canvas apps, these apps are hosted by the application platform. That will say, the files that the third-party application uses are hosted on servers provided by the application platform. The applications can only contain HTML, CSS and JavaScript. Developers who wish to register and upload an app create a zip-file

which at minimum includes a file named index.html. The supported file formats are .html, .css and .js.

Google provides a Caja-server at caja.appspot.com which has been used. This Caja-server fetches the web page to be cajoled through a GET request. This enforces public availability for the web pages that will be used in the third-party application. This can however be a security problem if the files are hosted on the same domain as the main platform. The Caja solution depends on that all untrusted code runs inside the Caja subsystem. If the files could be accessed outside Caja, the app would have free access to all the session variables and cookies owned by the main application. To prevent this are the files sent to another, safe, domain on the same server.

The first thing done when a Caja app is started is that the Caja subsystem is initialized. The most code used by the Caja subsystem can be found in the Caja.js file provided by Google. The second thing to do is to create the secure objects that the Caja application will have access to. These objects will build up an API for the third-party application to use. To enable a safe use of these objects are they tamed with the taming function provided by the Caja subsystem. This is done so that the third-party application will not be able to change the objects in any unintended way.

The Caja subsystem also controls all the network access the third-party app has. Only calls to domains that have been approved by an admin are accepted. The domains that should be allowed are set in an object called uriPolicy. Each network access made by the third-party application is filtered through this object. If the requested domain can't be found in the uriPolicy the request is aborted. This prevents the app to get images and scripts from domains that the platform provider doesn't trust. It does also prevent AJAX calls from being made even to the web API provided by the platform. To solve this, tamed objects that communicate with the web API server has been set up. With these objects the third-party app does not have to communicate with the server through AJAX request sent from its own code. Instead it asks the host page to do it for them through the tamed objects provided by the Caja subsystem. The host page can then decide whether the request should be allowed or not.

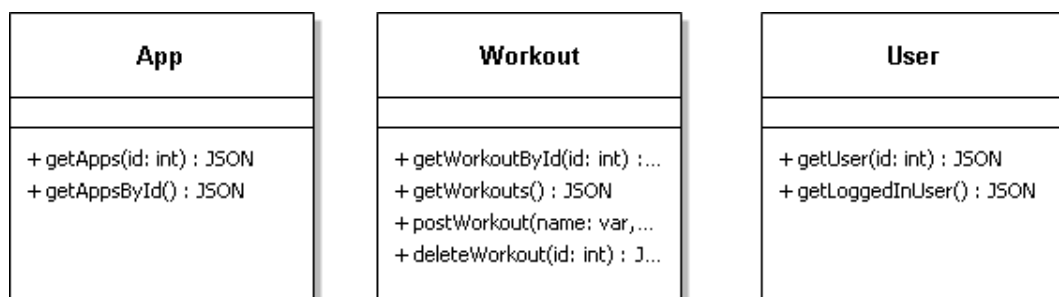


Figure 8 Class diagram of the tamed functions that is provided to the third-party application

When the Caja subsystem has been initialized and the tamed objects and uriPolicy has been created it is time to ask the Caja server to cajole the third-party application. The Caja server is being provided with the tamed objects, the Uri policy and the URL to the web

page that should be cajoled. When the Caja server responds with the cajoled code is it embedded in the div element that was created earlier.

To develop applications that run inside the Caja subsystem is almost like develop a regular web page consisting of just HTML, CSS and JavaScript. There are however some things that are not allowed. Some of these can be found in the theory chapter earlier in this thesis. To access data on the platform the third-party application uses the tamed objects provided by the Caja subsystem. These functions makes AJAX requests to the web API and then return the result to a callback function in the third-party application. Access to the currently logged in member is also made through the API provided by the Caja subsystem.

4.2.3 Web API

A web API was developed to enable third-party apps to interact with the main platform. This was implemented in ASP.NET MVC 4 Web API which made it very easy to create a web API from an already existing data model. This API is of the architectural style REST. The API supports CRUD operations by using different HTTP verbs in accordance to the REST style, see table.

Http verb	CRUD operation
GET	Read
POST	Create
PUT	Update
DELETE	Delete

To for example get information about a specific workout exercise can the following URL be typed in the browser: <http://workoutapps.azurewebsites.net/api/workout?id=1020>

This will return the following string:

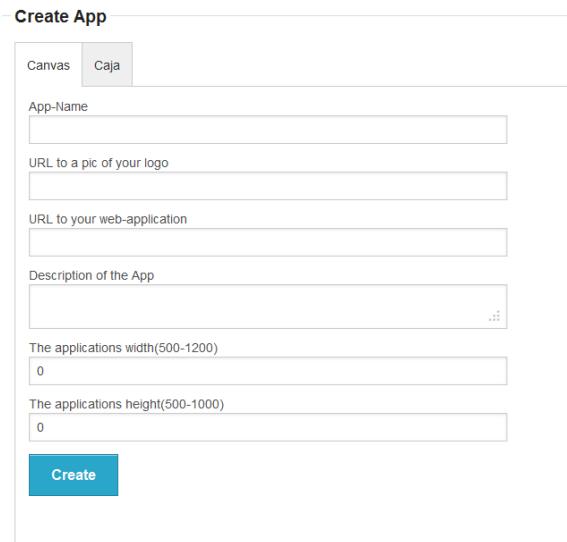
```
{"id":1020,"name":"Spinning","categoryId":1,"additionalFields":[{"id":14,"fieldName":"25","fieldValue":"Minutes"}, {"id":15,"fieldName":"Place","Indoor":"Place"}], "userId":1}
```

This string is of JSON type which makes it easy to handle in JavaScript code.

The web API makes it possible for third-party apps to do almost all the things the main platform can. This is essential to make the third-party applications feel like an integral part of the main system.

4.3 Create applications for the platform

In the prototype anyone who is registered on the main application can register a third-party application. Since there are two application types are there two possible ways to register an application. The most important difference between them is that the canvas apps requires an URL to where the application can be found while the Caja apps requires a zip file containing the application files.



Create App

Canvas **Caja**

App-Name

URL to a pic of your logo

URL to your web-application

Description of the App

The applications width(500-1200)

The applications height(500-1000)

Create

Figure 9 Registration form for canvas apps

When an application has been registered, a user with administrator permissions needs to approve it before it is made accessible for other users. After it has been approved any user on the main application has the ability to install it. To install an application the user goes to an install page which lists all the available applications. The user can click on an application and get a description of it. If the user decides to install it the user clicks on the install button which connects the user and the app to each other.

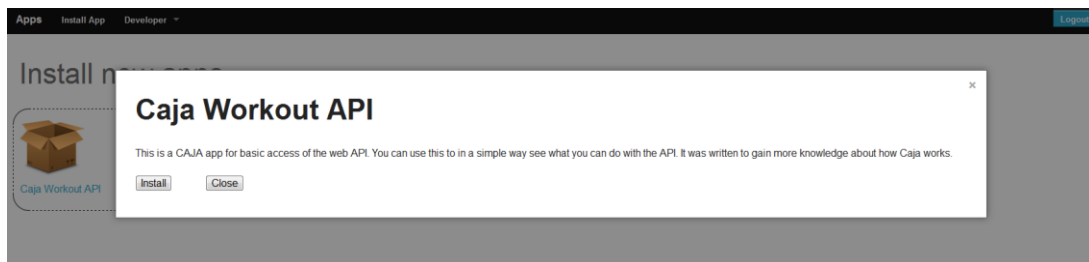


Figure 10 Example of how to install an application

The installed apps will then be available from the user's home screen.



Figure 11 Example of the home screen for a user.

Chapter 5

Discussion

This chapter includes a discussion based on the analysis and implementation of the prototype.

The iframe solution described in the implementation has proven to be quite efficient. It makes the development very easy since you can use the web development technology of your choice. This advantage could also be turned around to be a disadvantage since the developers of the third-party application must host the application themselves. This gives some cost for the third-party app to find a server to run the application on.

Another disadvantage with the iframe solution is that the provider of the application platform loses control over what the applications contains. It is almost impossible to control what the apps contain since they are embedded web pages from another domain. This makes it very hard to set strict guidelines on what third-party applications should be like and then force them to follow these guidelines.

Another disadvantage with iframes is that they do not provide a complete protection from the third-party application. For example the location of the parent window can be changed. Another disadvantage is that the iframe can add an extra scrollbar if the content doesn't fit into the provided area which doesn't look very nice.

Whether to use Google Caja or not for an application platform can be discussed. The sandbox environment does add more complexity to the development process. It was quite hard to debug the code when problems occurred which lead to some frustrating errors. The system was also not very stable at the moment of the implementation and had some browser incompatibles.

Another problem with Caja was the performance. For this thesis the Caja server at caja.appspot.com was used. The loading times for this could be up to ten seconds. This is of course not acceptable. One solution to this would probably have been to install your own Caja-server. A problem with this is that Caja is not compiled for Windows which was the operating system used during the implementation.

Whether these disadvantages are acceptable or not is up to the provider of the application platform. The Caja sandbox is very strict and gives very good protection. This protection might however be overkill in some situations. If you only have a few third-party applications developed by people you trust, Caja might not be the best solution.

An alternative to Caja could be to manually read the code and check that it does not do any harm. This would make it very easy for developers since it would let them develop the applications just like they are used to. One drawback with this solution would be that it adds more work for the providers of the application platform. To make it a little bit easier could it be combined with a JavaScript sanitizer less strict than Caja.

The advantage with solutions like the one Spotify have and the Caja one implemented in the prototype is that the provider gets more control of what the third-party applications

contain. This makes it much easier to assure that the application maintains a decent quality.

Chapter 6

Conclusion

When I look back at the work of this thesis I think the scope was too wide at the beginning. It would probably been better to concentrate on a smaller area from the start. If I can see any advantage with this would it be that I learnt very much and it gave me a broad knowledge about different techniques used in modern web development. Coming from a background with very little experience in JavaScript and web browser security were these parts probably the most interesting to work with. The work also deepened my knowledge about ASP.NET MVC 4 and Microsoft SQL.

What the prototype that was developed will lead to is hard to say. Ipendo Systems AB found the solutions developed in the prototype not really suitable for what they wanted to do. It was however considered interesting to look at these techniques since they in some way show some interesting areas of modern web technologies.

The use of Google Caja was a fun and interesting part of the thesis. JavaScript sandboxes are very hard to create but the developers of Caja have come far. A problem with Caja is however its lack of good documentation and use cases. My hope is therefore that this thesis will help other developers interested in Caja to see how and to what it can be used.

6.1 Future work

The original description of this thesis was to look at frameworks for third-party applications. It was intended to include both server-side and client-side implementations. The time frame was however too small to orderly look into both solutions and focus was therefore laid on more client-side oriented solutions. One interesting subject for the future would therefore be to look at how to enable more server-side code executed on your hosting server. The most challenging part here would probably be to put up a sandbox for the code.

Another subject for more studying would be to compare different JavaScript sanitizers and sandboxes. There exists some on the market today and it could therefore be interesting to compare them to each other.

Another interesting subject for further investigation is the authentication and authorization of the web API. There are many different solutions for this today and even though OAuth might be the most popular today is the future on this topic is not totally predictable. There exist some alternatives at the moment of this thesis, among these can OpenId and SAML be found. An analysis on which of them best suited would probably be an interesting topic.

References

- [1] Mark S. Miller, Mike Samuel, Ben Laurie, Ihab Awad, Mike Stay, *Caja Safe active content sanitized JavaScript*, Available from: <https://google-caja.googlecode.com/files/caja-spec-2008-06-07.pdf>, 2013-05-12
- [2] Google, *About Caja*, Available from: <https://developers.google.com/caja/docs/about/>, 2013-05-12
- [3] Google, *Introduction*, Available from: <https://developers.google.com/caja/>, 2013-05-12
- [4] OWASP, *Cross-Site Scripting (XSS)*, Available from: https://www.owasp.org/index.php/Cross-site_Scripting_%28XSS%29, 2013-05-23
- [5] Jeff Coble, *Building an application platform, Goals and Strategies*, Available from: <http://engineeringnotebook.org/building-an-application-platform/>, 2013-05-19
- [6] Salesforce, *Force.com Canvas Developer's Guide*, Available from: http://www.salesforce.com/us/developer/docs/platform_connectpre/canvas_framework.pdf, 2013-05-16
- [7] Salesforce, *Salesforce Platform Introduces New Salesforce Identity and Salesforce Touch Platform Services, Igniting a New Era of Social and Mobile Enterprise Apps*, Available from: <http://www.salesforce.com/company/news-press/press-releases/2012/09/120919-7.jsp>, 2013-07-20
- [8] Paul Lamere, *Building a Spotify App*, Available from: <http://musicmachinery.com/2011/12/02/building-a-spotify-app/>, 2013-05-19
- [9] Facebook, *Facebook platform launches*, Available from: <http://web.archive.org/web/20110522075406/http://developers.facebook.com/blog/post/21>, 2013-05-17
- [10] Facebook, *Graph API*, Available from: <https://developers.facebook.com/docs/reference/api/>, 2013-05-15
- [11] Facebook, *Data access, Login, Privacy and Permissions*, Available from: <https://developers.facebook.com/docs/reference/api/data-access/>, 2013-05-15
- [12] Facebook, *Canvas tutorial*, Available from: <https://developers.facebook.com/docs/appsonfacebook/tutorial/>, 2013-05-15
- [13] Mozilla, *Same-Origin Policy*, Available from: https://developer.mozilla.org/en-US/docs/JavaScript/Same_origin_policy_for_JavaScript?redirectlocale=en-US&redirectslug=Same_origin_policy_for_JavaScript, 2013-05-19
- [14] Facebook, *JavaScript SDK*, Available from: <https://developers.facebook.com/docs/reference/javascript/>, 2013-05-22

- [15] Spotify, Spotify, a perfect platform for apps, Available from: <http://press.spotify.com/se/2011/11/30/spotify-a-perfect-platform-for-apps/>, 2013-05-18
- [16] Spotify, Developer guidelines, Available from: <http://developer.spotify.com/technologies/apps/guidelines/developer/>, 2013-05-18
- [17] Wikipedia, OAuth, Available from: <http://en.wikipedia.org/wiki/OAuth>, 2013-05-19

Figures

Figure 1. Google, guestInsideHost.png, Available from: <https://developers.google.com/caja/docs/about/guestInsideHost.png>, 2013-05-12

Figure 2. Google, embeddingStep2.png, Available from: <https://developers.google.com/caja/docs/about/embeddingStep2.png>, 2013-05-12

Appendix 1

Glossary

API – Application Programming Interface, a set of rules defining how different programs should interact with each other.

REST – Representational State Transfer, a software architectural style for distributed systems, often used in web APIs.

App – short term for application.

DIV – is a HTML element that is used to define a division or section within an HTML document.

HTML – Hypertext Markup Language, a markup language mostly used when creating web pages.

CSS – Cascading Style Sheets, a style sheet markup language used to describe the presentation semantics of the elements in a markup language like HTML.

JavaScript – A script language that can be used in the web browser.

W3C DOM – Stands for Document Object Model. It is a cross-platform and language independent convention that makes it possible to access and update a document's content, structure and formatting. The document could for example be an HTML page.

C# - An object-oriented language developed by Microsoft and that is a part of the .NET platform.

URI – Uniform Resource Identifier. A string used to identify or to name a resource.

URL – Uniform Resource Locator. A string that identifies a webpage.

Sandbox – A term which is used to describe the separation of two programs.

Iframe – An HTML element that embeds a web page inside another web page.

Chromium – An open source web browser.

PHP – A script language mostly used on web servers to create web pages with dynamic content.

SDK – Software Development Kit. A collection of tools that makes it easy to develop against specific software. Could in the most basic cases exist of an API.

AJAX – Asynchronous JavaScript and XML. A collection of several different techniques that can be used when building interactive web pages.

Appendix 2

```
<html>
<head></head>
<body>
<script type="text/javascript"
  src="//caja.appspot.com/caja.js">
</script>

<div id="guest"></div>

<script type="text/javascript">
  function helloWorld(text) {
    alert(text);
  };

  caja.initialize({
    cajaServer: 'https://caja.appspot.com/',
    debug: true
  });

  var uriPloicy = {
    rewrite: function (uri) {
      if (/http:\/\/www\.example\.com/.test(uri))
      {
        return uri;
      }
      else {
        console.log("Net Access denied: The requested uri( " + uri + "
)is not allowed. Check the uri policy to see what uri that is allowed.");
        return undefined;
      }
    }
  };

  caja.markFunction(helloWorld);

  tamedHelloWorld = caja.tame(helloWorld);
  caja.load(document.getElementById('guest'), uriPloicy, function (frame) {
    frame.code('http://pageToCajole.com')
      .api({
        helloWorld: tamedHelloWorld
      })
      .run();
  });
</script>

</body>
</html>
```

Appendix 3

The database scheme used in the prototype

