Customization of Software for Implementation on Third Party Platforms

Fredrik Zettergren
Patrik Åkerstrand
Abstract

Customization of Software for Implementation on Third Party Platforms

Fredrik Zettergren and Patrik Åkerstrand

An increasingly common scenario in software development, and certainly in web technologies, is that companies customize parts of the developed software to run on third party platforms. The purpose of this is often to increase brand recognition or to increase traffic to their websites. Quite often the target platforms are numerous and are far from homogenous while the software itself is required to have preserved functionality in its original environment. This creates numerous difficulties both within the software structure, maintenance and in development routines. This thesis considers some of those difficulties and their potential solutions. Focus is put on an online real estate search company, Booli Search Technologies AB, that would like to offer its partners parts of its platform for implementation on their websites. Booli states both general and specific demands on the customization regarding functionality, preserved performance, security and updates.

The report begins with a survey of literature and related work within the area. Interesting solutions are then discussed and evaluated. Finally, a prototype implementation is done and evaluated together with Booli and some of its partners from different aspects. The results are presented together with a discussion and possible future improvements. Overall the prototype implementation satisfies the demands stated by Booli, even though some improvements are suggested.
## Thesaurus

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>The Booli platform</strong></td>
<td>The code and data collected by Booli Search Technologies AB intended for the website booli.se.</td>
</tr>
<tr>
<td><strong>Module</strong></td>
<td>A subset of the Booli platform centered around a GUI. The elements referred to as modules by Booli are sometimes referred to as widgets in other contexts.</td>
</tr>
<tr>
<td><strong>Widget</strong></td>
<td>A visual element on a webpage which a user can interact with. A widget contains both state and the ability to update itself to reflect changes of its state as the user interacts with it.</td>
</tr>
<tr>
<td><strong>Spoof</strong></td>
<td>We use the definition as defined in [12]: “To make a message appear as though it originated from a user or computer other than the one that actually generated the message.”</td>
</tr>
<tr>
<td><strong>Hooks</strong></td>
<td>A programming technique that allows functions or methods to be run after specific events occur. We define hooks both as template methods that can be extended by subclasses in a class hierarchy to provide specifics actions when an event occurs, as well as observable events that can be registered and listened for.</td>
</tr>
</tbody>
</table>
# Contents

1  Introduction  
   1.1  Background  
   1.2  Assumptions and limitations  
   1.3  The Booli platform  
      1.3.1  The model view controller pattern  
      1.3.2  Technical structure  
      1.3.3  Components  
   1.4  Task specification  
      1.4.1  General requirements  
      1.4.2  Modules  

2  Method  
   2.1  Literature studies  
   2.2  Early evaluation  
   2.3  Prototype implementation  
   2.4  Final evaluation  

3  Related work  
   3.1  Google  

6
7
8
8
9
9
13
13
16
21
21
22
22
22
23
3.1.1 API key structure ................................. 24
3.2 Nestoria .............................................. 24
  3.2.1 Widgets ......................................... 25
  3.2.2 API ................................................ 25
3.3 Trulia .................................................. 26
  3.3.1 API ................................................ 26
  3.3.2 Widgets ......................................... 26
3.4 Zillow .................................................. 27
  3.4.1 API ................................................ 27
  3.4.2 Widgets ......................................... 27
  3.4.3 Co-branding .................................... 27

4 Design decisions ........................................ 29
  4.1 Easy implementation and platform independence ................. 29
  4.2 Preserved functionality of booli.se ............................... 30
    4.2.1 Duplication of code ................................ 30
    4.2.2 Integrated customization ............................... 31
    4.2.3 Overriding customization ............................... 31
    4.2.4 Conclusion ........................................ 31
  4.3 Preserved performance .................................... 33
    4.3.1 Cross domain requests ............................... 33
  4.4 Software update system ................................... 37

5 Redesigning the Booli platform ................................ 39
  5.1 Modularization of code ................................... 39
  5.2 Authentication system .................................. 40
    5.2.1 Server to server authentication ...................... 40
## CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.2.2</td>
<td>Client to server authentication</td>
<td>43</td>
</tr>
<tr>
<td>5.3</td>
<td>Authorization system</td>
<td>44</td>
</tr>
<tr>
<td>5.4</td>
<td>API for external requests</td>
<td>45</td>
</tr>
<tr>
<td>5.4.1</td>
<td>Calling the API</td>
<td>45</td>
</tr>
<tr>
<td>5.4.2</td>
<td>Requesting code</td>
<td>46</td>
</tr>
<tr>
<td>5.4.3</td>
<td>Requesting data</td>
<td>47</td>
</tr>
<tr>
<td>5.5</td>
<td>Support systems</td>
<td>47</td>
</tr>
<tr>
<td>5.5.1</td>
<td>Testing frameworks</td>
<td>48</td>
</tr>
<tr>
<td>5.5.2</td>
<td>Availability monitoring</td>
<td>49</td>
</tr>
<tr>
<td>5.5.3</td>
<td>Utilization monitoring</td>
<td>49</td>
</tr>
<tr>
<td>5.5.4</td>
<td>Documentation generator</td>
<td>50</td>
</tr>
<tr>
<td>5.5.5</td>
<td>Technical discussion forum</td>
<td>50</td>
</tr>
<tr>
<td>6</td>
<td>Evaluation</td>
<td>52</td>
</tr>
<tr>
<td>6.1</td>
<td>Platform independence</td>
<td>52</td>
</tr>
<tr>
<td>6.2</td>
<td>Ease of implementation</td>
<td>54</td>
</tr>
<tr>
<td>6.3</td>
<td>Preserved performance of Booli.se</td>
<td>55</td>
</tr>
<tr>
<td>6.4</td>
<td>Preserved functionality of Booli.se</td>
<td>56</td>
</tr>
<tr>
<td>6.5</td>
<td>Authentication</td>
<td>56</td>
</tr>
<tr>
<td>6.6</td>
<td>Authorization</td>
<td>57</td>
</tr>
<tr>
<td>6.7</td>
<td>Software update system</td>
<td>58</td>
</tr>
<tr>
<td>7</td>
<td>Future work</td>
<td>59</td>
</tr>
<tr>
<td>7.1</td>
<td>Implementation of software modules</td>
<td>59</td>
</tr>
<tr>
<td>7.2</td>
<td>Authentication system</td>
<td>60</td>
</tr>
<tr>
<td>7.3</td>
<td>Authorization system</td>
<td>60</td>
</tr>
<tr>
<td>7.4</td>
<td>Support systems</td>
<td>61</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>8    Acknowledgements</td>
<td>62</td>
<td></td>
</tr>
<tr>
<td>A    Google API key - empiric study material</td>
<td>65</td>
<td></td>
</tr>
<tr>
<td>B    Cross domain performance tests</td>
<td>66</td>
<td></td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Background

Booli Search Technologies AB has developed a platform for use in online real estate search. The platform was developed in an effort to launch a fast, complete and intuitive website for Swedish property listings. The website, booli.se, was launched during late 2007 and has since then had a steady growth in popularity.

Even though the platform was intended for use on the website booli.se, the company has started investigating the possibility to license parts of the platform to other companies. Except for licensing, Booli would also like to be able to offer its partners modules, sometimes referred to as widgets, from the platform for implementation on their websites with the purpose of strengthening the company brand and to increase traffic volumes to booli.se. Both of these wishes require extensive customization of the platform and its support systems since the platform is not currently intended for anything else but the website booli.se. The purpose of this master thesis is to identify which customizations are necessary and how they are best performed. To evaluate the result of the modifications, a prototype solution will be developed and tested.

Before the start of this master thesis project Booli stated a number of requirements for the customization which are stated in 1.4 Task specification.
1.2 Assumptions and limitations

We assume that the reader of this master thesis has general knowledge of IT and web technology in general. A familiarity with web 2.0 terms and its usage throughout the Internet will greatly simplify the understanding of the content of this master thesis report.

The scope of this master thesis covers an evaluation of the most popular methods used when customizing software for implementation on third party platforms. No extensive work is done regarding development of brand new solutions; instead methods already proposed by others are used as an inspiration and customized for Booli’s particular needs. Nor are any other systems considered to any greater extent, but the solutions are evaluated solely for the needs stated by Booli in the purpose of collaboration with their partners.

The scope is also limited to a survey of solutions we found relevant on an early stage. In cases such as cross domain requests, several techniques that we found inappropriate are omitted.

Since this master thesis is time boxed to 20 calendar weeks, only a prototype is implemented which makes no attempt at being an exhaustive solution for production. Because of this, many of the proposed and implemented solutions are only tested briefly in their prototype implementation. The implemented solution to a specific problem, however, has been found through identifying and evaluating multiple solutions to the problem against each other.

The term platform independence in this master thesis report does not imply total platform independence but rather that functionality, on systems common in web development and especially those used by Booli and its partners, is asserted.

In this report we will not discuss or mention issues and technical solutions for gathering and aggregating the data that the user can access on booli.se. This is a whole subject in itself and we will instead treat only issues related to displaying mentioned data in contexts differing from the one originally intended when the feature was built for booli.se.
1.3 The Booli platform

The “Booli platform” refers to the code and data developed and collected by Booli Search Technologies AB for use on the website booli.se. The code is a large set of files written in PHP, JavaScript, HTML and XML connected to each other by different types of interfaces. The data is a set of listings from a large portion of the Swedish real estate agencies. Currently approximately 260 000 listings of different types are available via the platform.

1.3.1 The model view controller pattern

Booli’s platform is built utilizing the model view controller, MVC, design pattern. MVC is a well-known and extensively used pattern [8]. It can be applied both as a design pattern as well as an architectural pattern. The idea behind the pattern is to split the application logic from the presentation and the user interface response to a user’s interaction. This decouples the application, increasing the flexibility and possibility of code reuse. Almost all graphical frameworks utilize this pattern one way or another [8], as well as entire frameworks, such as Ruby on Rails [13].

The model captures the application logic and is responsible for modifying the state of the application. The view is a presentation of the state of the system, pulling the appropriate data from the model and presenting it to the user of the system. Finally, the controller is responsible to react to user interaction with the system interface, communicating to the model what action the user wants.

Using the same logic it is possible to present the result of an event and/or state of the system, in different formats by exchanging the view object, or changing how the user interface reacts to the event by changing the controller.
1.3. THE BOOLI PLATFORM

1.3.2 Technical structure

The website, booli.se, for which the Booli platform was developed, uses AJAX technology to give the visitor a responsive graphical user interface. To accomplish this, the developers have built the platform with a variety of languages and techniques. The main languages used in the platform are:

- SQL for database queries
- PHP for server side scripts
- JavaScript for client side browser scripts
- HTML and CSS for browser presentation

1.3.3 Components

The core of the Booli platform is the property search. To aide users in the search process, a number of components have been added.
Figure 1.2: The components of Booli affected by the API. The figure also shows how the storage of data is abstracted from the views displaying the data.
1.3. THE BOOLI PLATFORM

1.3.3.1 The Suggest

The purpose of the Suggest system is to aide a user in defining a specific geographical area. This is done by allowing a user to input fragments of a name and return an ordered set of areas which names are or starts with the given string. The ordering of the set is based on the amount of real estate listings in each area in descending order. The input ”Sto” could for example return the following suggestions

- Stockholm, Stockholms län
- Storfors, Värmlands län
- Storuman, Västerbotten
- ...

The Suggest consists of three major components; the JavaScript object which together with the HTML page acts as the view, the PHP controller and the PHP model. A user interacts with the suggest via a HTML page with an input field. When the contents of the input field is changed, the JavaScript Suggest object retrieves the contents from the field and sends an Ajax request to the suggest controller with the contents as a parameter. The controller forwards the request to the Suggest model which retrieves suggestions from the database.

1.3.3.2 The Search

The search takes as input a search string or an id of a geographical area. Many times, a user has found a geographical area with the help of the Suggest, and therefore sends the id of that area as input. Even though this is true for almost all users, every now and then, a user performs a search without selecting a suggestion. In these cases the search uses the Suggest model to retrieve a suggestion for the string inputted, and simply chooses the suggestion with the most listings for that search string. Once a geographical area has been uniquely selected, the search queries the database for relevant listings. These listings are returned to the client in JavaScript Object Notation (JSON)
Figure 1.3: The structure of the Suggest module
form. Once returned to the client, the JSON object is evaluated and presented to the user with the help of the Hitlist and Map modules.

1.4 Task specification

Booli has prior to the start of this master thesis stated some general requirements for the customizations and a list of modules that it would like to offer partners of the company. For each module, a short specification has also been written regarding functionality, user interface and security.

1.4.1 General requirements

These are the general requirements Booli has stated for all modules that are to be implemented on third party platforms.

1.4.1.1 Easy implementation

A person with general knowledge of IT systems, but not necessarily any special knowledge about the Booli platform or the technologies used by it, should be able to implement Booli’s modules on their system with the help of only a written guide. This requirement will be evaluated by the amount of time spent on support by the Booli staff during each implementation.

1.4.1.2 Platform independence

All of the modules should be, to the largest possible extent, platform independent. In this context this means that the functionality of the modules offered by Booli should not be severely affected by hardware architecture, operating system, server type or any other installed software.

1.4.1.3 Preserved performance

The performance, that is speed and stability, is an essential part of Booli’s trademark. It is therefore very important for the company that the performance of the software, either original or the one
Figure 1.4: The structure of the Search module. Notice how the MVC pattern enables two different views of the same data. The data is displayed both in a map and in a hit list.
running on third party platforms, is not noticeably affected by the customization.

1.4.1.4 Preserved functionality of booli.se

The changes made to the Booli platform and its support systems should not affect the functionality of the original platform in a negative manner. This means that functionality of booli.se should be unaffected. This requirement also includes preserved development possibilities for the code serving the website. Therefore customization of the platform should not increase the original platform code complexity in an unproportional manner.

1.4.1.5 Software update system

Updates made to the Booli platform should be reflected in the modules implemented on third party platforms. Furthermore, updates intended specifically for the external modules should be able to be deployed automatically from Booli’s servers, requiring no manual updates to individual implementations in most cases.

1.4.1.6 Authentication

Booli would like to maintain control over who uses the API and implement an authentication system that allows only registered and approved third parties to use the API. The authentication system should not affect performance of the modules and be as transparent as possible to the implementing party.

1.4.1.7 Authorization

Booli would like to maintain control over which parts of the API that are available to a partner and offer access to different parts of the API to different partners.
1.4.1.8 Extensibility

Booli would like a solution that can be easily extended to include future features of the API. They would also like a way to request only data from the API, so more technically advanced partners may request data and present it in a way different from how Booli presents it. Additionally, Booli has plans to give brokers the opportunity to add, edit and remove their own listings on Booli through the API. This requirement demands a higher level of security than what is available today.

1.4.2 Modules

The modules from the platforms that Booli would like to offer its partners for implementation on their websites are the Searchbox, Hitlist, Searchbar and the Map. Each module specification is derived from an already existing feature on booli.se. Overall, the specifications are focused on mimicking the exact behaviour of the module presented on the company’s own website.

1.4.2.1 Searchbox

The Searchbox module consist of a text field for geographical searches. When a user enters text into the field suggestions from the Booli property database is displayed in real time. The suggestions presented are names of geographical areas where there are properties for sale in the Booli database, and part of the geographical area name is the text input from the user. Each suggestion is associated with a resource identifier on booli.se that lets the user see all property listings within that area. Selection of one of the suggestions can either lead to results being displayed on booli.se or on the site where the bundle is implemented depending on which software bundles that are available to the client. The size of text, colours of fonts and images in the graphical user interface of the module should be customizable by the implementing part.
1.4. TASK SPECIFICATION

1.4.2.2 Hitlist

The Hitlist module presents results from a search query. The module also has built in pagination and functionality for sorting hits according to their different attributes. The module integrates with the Searchbar and Map modules. Each row in the hitlist presents some basic details about a listing. Each row is also associated with an URL to a page where more extensive information is displayed for the listing. The size of text, colours of fonts and images in the graphical user interface of the module is customizable by the implementing part.

Figure 1.5: The search box as it appears on booli.se. As part of the Searchbox module the appearance can be customized to fit the design of the implementing part.
1.4. TASK SPECIFICATION

CHAPTER 1. INTRODUCTION

Figure 1.6: The hitlist as it appears on booli.se. As part of the Hitlist module the appearance can be tweaked to fit the design of the implementing part.

1.4.2.3 Searchbar

The Searchbar requires and incorporates the Searchbox along with different filtering controls. When the user interacts with the filtering controls the search results are dynamically updated to reflect the new state. Property searches can be made with filters on both geographical placement and property attributes for an object. The size of text, colours of fonts and images in the graphical user interface of the module is customizable by the implementing part.
1.4. TASK SPECIFICATION

CHAPTER 1. INTRODUCTION

Figure 1.7: The search bar as it appears on booli.se. As part of the Searchbar module the appearance can be customized to conform with the design of the implementing part’s platform.

1.4.2.4 Map

The Map is used for presentation of property listings with a geographical perspective. Each listing passed to the Map is presented as a marker. If the number of markers passed at once exceeds 50, clusters of markers are presented. Each cluster can expand and display its markers for the user. Each marker is associated with an URL to a page with more extensive information about the listing. These pages are located on booli.se. The module can integrate with the hit list module. The size of text, colours of fonts and images in the graphical user interface of the module is customizable by the implementing part.
Figure 1.8: The map as it appears on booli.se. As part of the Map module, some of the appearance such as popup style and colors can be tweaked to fit the design of the implementing part.
Chapter 2

Method

We began our master thesis project by working out a project plan for the best way to investigate and evaluate different approaches. This plan was made in cooperation with Booli to synchronize with their partner program and therefore to be able to implement prototype solutions as soon as possible after their completion. The plan consisted of four major steps; literature studies, early evaluation, prototype implementation and final evaluation.

2.1 Literature studies

Since our master thesis project revolves around web technologies, most relevant literature is found in online documents. We therefore began our studies by reading documents published by the major web institutions such as Google, W3C and Yahoo!. With the help of the findings from these institutions we moved on to less general literature, such as text published by other online Real Estate search companies. The results of the literature studies were a number of possible approaches to the problems associated with the customization.
2.2 Early evaluation

As soon as feasible approaches were identified during the literature studies we made early evaluation to be able to select the ones that suited our needs. Depending on the problem and solution, evaluation was done either by discussion and verification of assumptions or by technical evaluation. The results of this evaluation process together with details on each process can be found in section 4 Design decisions.

2.3 Prototype implementation

To assert that our conclusions from the early evaluation process were correct we implemented a prototype solution of our customization. The prototype solution was thoroughly tested on Boolis development servers. Once the testing showed satisfying results the modules were deployed to one of Boolis partners servers. Once deployed, the modules were evaluated and tested in production.

2.4 Final evaluation

The final evaluation was performed together with Booli and its partners. During this phase we evaluated the prototype solution and compared the results to the requirements stated in section 1.4 Task specifications. The results were then analyzed and discussed with all the involved parts. The conclusions can be found in section 6 Evaluation and section 7 Future work.
Chapter 3

Related work

The idea of customizing parts of a developed platform for implementation on third party systems has blossomed on the web during the last 5 years, to the extent that it has become a ubiquitous part of the Internet. Companies have discovered that the result of time being put into development can be greatly improved by letting partners implement the same software on their systems. This benefit comes either as direct payment from the implementing part or as an increase of awareness of the company brand. Many of the successful companies in different fields provide web services or public APIs for others to use. Below we will discuss a number of companies that are all, with the exception of Google, in the real estate field.

3.1 Google

Google hardly needs any formal introduction, having virtually revolutionized online search in the last decade, and has grown rapidly since its founding ten years ago. It now features a lot of publicly available APIs, where the most well-known one might be Google Maps. Google has a well-developed system for authentication, requiring users of the API to sign up for API keys bound to domains. The key needs to be included in all requests to the Google APIs.
3.1.1 API key structure

The Google API key is 86 characters long and is, from what can be observed, made up of four parts. The first eight characters are the same for all keys, and this part can probably be used as a quick and inexpensive first check to verify if the attached key of the request has any chance of being a valid Google key. From empiric investigation we have found that the next 22 characters of the API key somehow encodes the Google Account of the user who requests the key for a domain. Tying each API key to an account gives Google quite a few interesting possibilities. They can quickly see which users who most frequently use their API, but perhaps even more importantly, it enables them to keep a blacklist of user accounts that have violated their terms of service. Upon incoming requests they can check the key against the blacklist and deny any requests with API keys created by the blacklisted account. The next 28 characters of the API key seem to encode the domain that the key is valid for. This part remains the same for a given domain regardless of the account being used. Since the key is validated on Google’s servers and the algorithm for doing so is secret and proprietary it is very hard to determine if they can extract the domain from the key or if this part of the key uniquely identifies a domain. Given the domain the server then generates a hash that is included in the JavaScript code that is sent back to the requester. On the client side a check is made against window.location to see if the same hash can be generated, and if so the request is considered valid. If the check fails, the user is presented with the information that the key is invalid and provided a link to Google’s key generator. The material for the empiric investigation is presented in Appendix A.

3.2 Nestoria

Nestoria is a British vertical search engine for property listings within the UK, Germany, Italy and Spain[10]. Nestoria offers a portfolio of modules free of charge for third party websites under the name ”tools for webmasters”. They also provide a public API that can be queried for data in a more powerful and interactive way, from the point of view of a developer, than the webmaster tools.
3.2. NESTORIA CHAPTER 3. RELATED WORK

3.2.1 Widgets

The web master tools use a variety of different client side solutions to fetch and display the data from Nestoria[11]. All of them have in common that all the implementer of the widgets need to do is copy some HTML and, in some cases JavaScript, into her own site to make it work. The widget then internally takes care of everything else that is needed. The house pricing widget is implemented using HTML and JavaScript. The HTML creates a reference element that the widget uses as the container all fetched data will be displayed within. The widget asynchronously inserts a script element into the document to fetch data from Nestoria, and a call-back is applied to display the data once it is loaded. The Property Lists widget uses another approach to achieve the same effect. Instead of requesting the data asynchronously and displaying the data once it is loaded, this widget instead directly writes to the DOM using JavaScript, temporarily halting rendering of the page until the data is loaded and displayed. This conveys the feeling that the data resides on the implementing part’s site, rather than at Nestoria, but it also delays the rendering of the rest of the content of the implementing part until all data has been downloaded from Nestoria. The last method used by Nestoria widgets is the method used by the DropIn Maps widget. This widget utilizes an iFrame to link in content from Nestoria into the target page. In this approach all data and functionality still resides on Nestoria’s servers and the webmaster only needs to insert the iFrame element into her page.

3.2.2 API

The public API of Nestoria is implemented as a web service and requests are made by HTTP GET requests. The desired API action, response format and additional action specific parameters are specified as HTTP GET query parameters. The response to a request is returned in the body of the HTTP response, and can be returned as either JSON or XML. For the JSON format a call-back parameter can be specified. This enables a client side implementation of the API to take appropriate actions once a response to a client side request has been returned from the Nestoria API[9].
3.3 Trulia

Trulia is a US based vertical real estate search engine in the USA[20]. They have both a public API as well as widgets that pull data from that API. Trulia does not at the moment offer an API that provides information about specific listings. Instead they provide access to statistics about areas and the ability to retrieve geographical data. Trulia’s API is completely free to use and open for everyone, although some of the widgets require the end-user to be a real-estate broker with a valid domain registered in Trulia’s search index.

3.3.1 API

The Trulia API is built using a RESTful architecture[21]. All requests to the API should be directed to the same resource, and the request must at least provide two mandatory HTTP GET parameters named library and function. The library parameter specifies, as the name suggests, the name of the library in which to invoke a function. The function parameter specifies which function to call in the selected library. If either library or function is unknown or spelled incorrectly, an error response is sent back with details about the error. If the API key is an unknown key, an HTTP header with code 403 (Forbidden) is sent as response. There seems to be no additional assertion that the supplied key is used by the user who signed up for a key, so it would be possible to monitor incoming requests to the Trulia API and steal any of the keys used in the requests for later use.

3.3.2 Widgets

Trulia offers, as previously mentioned, a range of free widgets that can be included on any website. They pull their data directly from Trulia, either as requests to the API when they need to update themselves, or the data is sent together with the widget upon first load. The techniques used to display the widgets differ a bit, but most of them use a snippet of HTML for placement on the web page and JavaScript for pulling data from the API. The only widget not taking this approach is TruliaStats, a widget for displaying statistics over time as graphs, which instead use Flash. [19]
3.4 Zillow

Zillow is a vertical search engine for properties in the USA. It was founded in 2005 and launched in 2006. Zillow claims to cover 83,000,000 listings and provide services such as home valuation based on trends in their databases [6]. Zillow has an extensive public API divided into four distinct parts. Access is granted to the API by signing up for a key. This key can be used to gain access to each API part, and access is granted upon request.

3.4.1 API

The Zillow API is built with a RESTful architecture, and requests are made using HTTP GET [5]. Each part of the API has its own URL, and method parameters are specified as HTTP GET parameters. The API key, called zws-id, is mandatory to include with each request. The authentication system seems to check only that the zws-id is a valid id. This means that anyone can use a valid zws-id and use it like it was their own. Zillow does state that they log ip-numbers in conjunction with the zws-id, but it seems to use it only to ensure that the request limit of 1000 calls a day is not exceeded.

3.4.2 Widgets

In addition to the API, Zillow also offers two widgets for calculating mortgage and displaying mortgage rates. These widgets are implemented using an iFrame approach, keeping JavaScript source code on Zillow’s own site. This also makes it very simple to implement at another page, requiring nothing more than that the implementing part knows how to copy and paste. [4]

3.4.3 Co-branding

If another site has a Zillow profile, i.e. is registered as a user on Zillow, they offer to co-brand their site whenever a user arrives to Zillow through a link on the first site. This link can be either in a widget or a regular link. The co-branded Zillow site shows a profile picture, contact details and
logo of the originating website on the top-row of its own site. [3]
Chapter 4

Design decisions

Parallel to the survey we performed, to find the best practices for solving the problems associated with the customization, we started to develop prototype solutions for the modules. During this process we performed a number of tests and evaluations on different techniques and approaches. Some of these evaluations consisted solely of discussions while other consisted of technical tests and performance measuring. Usually these discussions or evaluations would lead to a design decision for the customization. These decisions, together with the motivations that led to each decision, are explained here.

4.1 Easy implementation and platform independence

Since server side software and configuration varies a lot between different websites, designing software that is easy to implement and compatible with the different platforms is not trivial. Among Booli’s current partners alone we have identified the following operating systems and software setups in different combinations:

- Operating systems
  - Windows 2000 Server
  - Ubuntu Linux
4.2. PRESERVED FUNCTIONALITY OF BOOLI.SE

4.2. PRESERVED FUNCTIONALITY OF BOOLI.SE

- Debian Linux

- Web servers
  - Apache
  - Apache 2
  - Microsoft IIS

- Server side languages
  - PHP 4
  - PHP 5
  - ASP
  - CGI

Since developing versions of the software bundles for each platform would increase the complexity of the development a great deal, we decided it was not a reasonable solution. Instead we set our focus on making the code 100% browser executable.

4.2 Preserved functionality of booli.se

To be able to customize the Booli platform and at the same time preserve the functionality of booli.se, we identified three possible solutions: duplication of code, integrated customization and overriding customization.

4.2.1 Duplication of code

To fully avoid affecting the performance of booli.se, a duplicate source file would be created for each of the files needed for the module. These files, instead of the originals, would then be rewritten for implementation on third party platforms.
This approach satisfies the requirement for preserved functionality of booli.se, but it also creates a lot of duplicated code. This duplication complicates maintenance and updates to the code to such an extent that it is not a reasonable, nor feasible, long term solution.

### 4.2.2 Integrated customization

With integrated customization, instead of duplication of code, all customization is done within the original code. Functions, procedures and classes get flags to perform in different ways depending on the context the code is running in. While this approach does not give any duplication of code, it does increase the complexity in the original code. This slows down development and increases risks for bugs in the original code, thus not fulfilling the requirement of preserved functionality of booli.se

### 4.2.3 Overriding customization

With overriding customization, the files containing code that needs to be customized in a module gets an override-file where methods and variables of the original code is overridden. This approach enables us to change the behaviour of some of the functions to work properly even though the code is now used in a different context than the one originally intended at booli.se For example, when implementing a suggest that needs to retrieve suggest data from another domain, an override file is loaded that redefines how this request for suggest data is created and how the data is received. Only the part of the code that differs from the original code needs to be rewritten, thus minimizing the amount of new code produced.

### 4.2.4 Conclusion

After evaluation the time consumption and complexity of the three solutions, we decided to implement an overriding customization. This approach does not increase the complexity of the original code, nor does it create duplicate code and was therefore more suitable for our needs.
Figure 4.1: The files containing code that needs to be customized in a module gets a override-file where methods and variables of the original code is overridden.
4.3 Preserved performance

Preserved performance, i.e. speed and stability, of the platform is a key requirement from Booli for this survey and implementation. To maintain the stability of the platform, one important part of the solution is the “overriding customization” mentioned earlier. When it comes to speed, the performance of the Booli platform and its modules is affected mainly by the client side JavaScript engine and the client-server request delays. A short rundown of the JavaScript performance of different browsers will be made, but most focus will be put on the client-server request delays, since not much can be done about the user’s choice of browser.

4.3.1 Cross domain requests

The client-server communication is even more complex when integrating modules on third party platforms than in regular web 2.0 Ajax application. Usually, the browser communicates with the server via an XMLHttpRequest [24]. When it comes to implementations on third party platforms, usually situated within another domain, a security measure in JavaScript gets in the way. The measure is called the ”same origin” policy [17], and it prevents documents loaded from one origin to change data in documents loaded from another origin.

Fortunately there are several solutions and workarounds for this limitation. Two of these workarounds are evaluated and tested in this report: Dynamic Script Tag and server side proxy.

4.3.1.1 Dynamic Script Tag

The implication of the same origin policy is that a JavaScript served from domain A, cannot load data from any other domain than A. However, the limitation does not cover loading scripts from other JavaScript files from other domains. This is usually done when the webpage is served from the server to the client, but can also be done asynchronously. To include a script during runtime, a script element can be created and added to the DOM.

Even though the tag is designed to load a JavaScript source file, the server handling the request can serve any type of data. By letting the source url contain parameters, and the answer contain
4.3. PRESERVED PERFORMANCE

CHAPTER 4. DESIGN DECISIONS

The data response together with a call-back function call, cross domain JavaScript requests can be achieved.

![Diagram](domain_a_domain_b.png)

Figure 4.3: The client simulates and Ajax request by inserting a script tag into the DOM that loads data from domain B. The response contains both data and a callback so that the client may resume processing once all data has been retrieved from domain B.

An advantage of this solution is that no server side code has to be implemented on the third party platform for the data request. All communication is done from client to the data delivery server. On the other hand, this is a workaround, and the code is therefore somewhat awkward. There is also an ongoing discussion in various forums about blocking cross domain requests due to its widespread use in malicious code. This is currently only being discussed and will most likely not be implemented in widespread browsers for another few years.

4.3.1.2 Server side proxy

Since there is no same origin policy in any of the widely used server side scripting languages, a server side proxy can be used to achieve cross domain requests. The proxy is implemented on the
same server that serves the JavaScript source, server A. The client then performs requests for data to server A1 which forwards the request to server B1. Once the response is retrieved from server B1, server A1 forwards the data to the client through the original XMLHttpRequest.

Figure 4.4: In the proxy solution the server receives all the Ajax-requests made by the client, but instead of doing any work it forwards the request to the server in domain B and echoes B’s response back to the client.

This solution has the convenience of unchanged JavaScript source code. Since the solution is done directly to a server within the same domain as the server serving the JavaScript source, no workarounds are necessary. On the other hand, a server side proxy has to be implemented on the third party system. This can be more daunting that it sounds, since operating systems, languages and other environmental variables change among Booli’s partners.

4.3.1.3 Performance tests

Both of the above proposed solutions require some extra execution time compared to the corresponding functionality provided on boolise. Dynamic Script Tags come with extra code to run. The amount of time this takes depends on both the speed of client machine as well as which browser, and thus JavaScript engine, the client uses. The server side proxy requires an extra HTTP request to forward the client’s request to Booli’s servers. To assert that none of the identified solutions were inferior to the other speedwise, we performed a series of tests comparing the two using the Searchbox Module.
The environment in which the tests were performed was as follows. We set up one of our computers to act as a client. Another computer was setup to act as the host and proxy. Finally, a third computer was set up to act as Booli’s web service, providing suggestions when requested. These tests were performed by a code snippet that rotated between requests for ten different input strings. A total of 100 requests were made by each solution prototype. The code snippet first ran a test using the cross domain Dynamic Script Tag solution. For each request the test code inspected the returned result to assert that it was valid and also recorded the response time. The results were accumulated in an array for later analysis. Once the first test round was complete, another test round was started; now using the server side proxy solution approach. Using the same method of inspecting the result and record the response time, these results were also saved.

The results from the tests are displayed in figure 4.5, where the y-axis represents request time, and the x-axis represents the request number. We also made a moving average for the requests to be able to distinguish trends in the request response times more easily. Finally, we calculated an average request response time for the solution prototypes.

![Figure 4.5](image)

Figure 4.5: The y-axis represents request time, and the x-axis represents the request number. The dotted lines are moving averages of the last 10 request response times with the same coloured solution

During the test, both solutions showed a 100% success rate, i.e. correctly formed data was
returned on each request. The test also showed that the server side proxy solution was slightly faster, 130ms on average, than the Dynamic Script Tag, 139ms on average, in this test environment.

4.3.1.4 Discussion and conclusion

The results produced by our tests are probably not representative for the different environments that exist out in the real world. One has to take into account that our entire test environment was setup inside the same network, which heavily marginalizes the effects of latency and bandwidth. This also means that the time it takes to create and insert a script tag, which is usually small compared to transfer times, becomes an important deciding factor. For a more realistic setup it is likely that the Dynamic Script Tag solution would be more efficient since it communicates directly with the supplier of the data. However, this is not guaranteed either since it is possible that the client has a good connection to its host, and that the host has a good connection with booli.se, but the client has a bad direct connection to booli.se. In this case it is again possible that the server side proxy is faster than the Dynamic Script Tag approach. To be completely confident that either solution is consistently more efficient than the other we could have executed more test rounds and varied the testing environment more. We feel, however, that these results which represent the ideal environment for the server-side proxy solution were enough to make a well founded decision.

The difference in speed between the two possible solutions for cross domain requests in our set up is small, roughly 7%. As mentioned earlier, there is wide spread in operating systems and scripting languages among Booli’s partners, which makes server side implementations of proxies time consuming and many times complex. The ease of implementation together with a performance comparable to server side proxies, even under circumstances that were ideal for the server side proxy solution, made the Dynamic Script Tag the most natural solution.

4.4 Software update system

Since Booli wants as many partners as possible to implement its platform modules; there is reason to believe that the number of implementations soon will be too large for manual maintenance and
updates from the company’s staff. The fact that the company has already accumulated a number of partners during the short period the company has been active is proof of this alone.

For the solution to be scalable, it is therefore crucial that most or all implementations can be maintained or updated through a central code repository. We therefore chose a solution where only a few lines of code were implemented on the third party platforms. The code works as a loader for modules of the Booli platform by implementing a few control parameters, and then fetches the modules from Booli’s main code repository.

![Diagram of central code repository](image)

Figure 4.6: For the solution to be scalable, it is crucial that most or all implementations can be maintained or updated through a central code repository.
Chapter 5

Redesigning the Booli platform

5.1 Modularization of code

When we started to look at what needed to be done to incorporate the API functionality in the already existing code base, we had our minds set to try and find good and flexible solutions without them being overly complex. We decided to refactor existing code where possible to remove any assumptions of the context in which it would be used. This meant trying even harder to decouple models from controllers and views, placing most of the logic solely in models. The biggest advantage of decoupling in this manner is that it gives the ability to work with data alone, and only at the last step convert it into a displayable form. The displayable form differs greatly between Booli.se and the needs for the API, since much of the output on Booli.se is HTML, but the API needs to output XML or JSON.

After this work was done we looked at the models and tried to refactor overly complex methods into more easily understandable parts. We found great use of different refactoring patterns [1], but we found the patterns Extract method and Replace Conditional with Polymorphism especially useful. In these patterns complex methods is split up so the code becomes easier to read and follow, or such that complexity is reduced by using the virtues of object-oriented programming. The end result was a much more flexible system that is easier to work with and can be more easily extended as the needs of the API grows.
5.2 Authentication system

One of the requirements from Booli is that they want to be able to control who is allowed to use their API, what parts of the API and ensure that only approved partners can retrieve data from the system. Furthermore, Booli wishes to be able to track when and how the API is used. This establishes the need for an authentication system that efficiently and securely can differentiate between valid and invalid requests. Additionally, the authentication procedure must be simple enough that it does not add unwanted and unnecessary complexity to the implementation of the Booli API at the third parties. For this reason, two different authentication systems are required; one for server to server requests and one for client to server requests. Below we give a detailed explanation of why and how the authentication system works and is implemented. If a request to the API fails to authenticate, regardless of the authentication method used, it is rejected any further processing and a response detailing the reason why the authentication failed is sent back to the requester.

Once a request has been authenticated it is allowed to continue into the authorization system.

5.2.1 Server to server authentication

Since requests server to server are granted much greater freedom and possibility to interact with the Booli platform, than their client counter-parts, the requirements on the authentication system grows. Given that a particular third party user has been given access to all parts of the platform, she is able to add, edit and delete her listings on Booli, as well as extract statistics and other more generic data. This type of powerful access requires an authentication system setup in such a way that it is easy for clients to generate valid requests to the API, but hard for an attacker. For this purpose Booli and the third party establishes a shared secret key that will be used in future requests. This shared key must be able to be established securely and renewed if needed. One way of achieving this is by using https during key generation, taking advantage of an already secure system to avoid eavesdropping. Additional measures have to be taken to avoid unauthorized third parties from generating keys or obtaining a valid key belonging to a third party.
5.2. AUTHENTICATION SYSTEM  CHAPTER 5. REDESIGNING THE BOOLI PLATFORM

Given that a common shared secret has been established between Booli and a third party, the authentication procedure looks as follows. The third party (A) wants to make a request to Booli (B) for some desired data. He then generates a random, unique, string to be used for the request. This unique string is needed to ensure that requests can not be repeated by an attacker[18]. Therefore B will only accept requests where the unique string has not been used by A before in a long enough time. The string is hashed together with the common secret key and the current timestamp into a request token. The request is then made to B together with the request token, the unique string, the timestamp and the public key of A. The timestamp is used to ensure that this request token is only valid from the moment it was created and in a fixed interval of time thereafter. It is thus important that A’s and B’s internal clocks are synchronized properly. This issue can be resolved by agreeing to synchronize the clocks on a trusted third party time server.

Once B receives the request he looks at the unique string and the timestamp included in the request. If the unique string has been used before by A the request is rejected since it might be a repeating attempt by an attacker. If the timestamp provided in the request is not within a certain interval of the current time at B, the request is rejected since the request token is only valid for this fixed amount of time. This is again to protect against attacks, and especially repeating attacks. If these two checks are passed, B looks up the secret shared between A and B by using the public key provided in the request. After the secret key has been retrieved B makes a hash of the secret, the unique string and the timestamp provided in the request and compares the computed value to the request token. If they match the request is considered authenticated. If the provided token does not match the value computed at B, the request is allowed no further processing, and a response is sent back to A with details of the authentication failure. For a visual representation of the above procedure, see figure 5.1

The security of this authentication scheme relies on the confidentiality of the shared secret key. If this key is compromised an attacker can generate valid requests as long as he is familiar with the rest of the authentication scheme.
5.2. AUTHENTICATION SYSTEM  CHAPTER 5. REDESIGNING THE BOOLI PLATFORM

![Visual Chart](image)

Figure 5.1: A visual chart of the server to server authentication procedure. A and B have established a secret common key $k$. For each request from A to B, A generates a random unique string $s$. This string is used together with the current time $ts$ and $k$ to create a request authentication token $t$. A then makes the request to B and attach $s$, $t$, $ts$ and the public API key $A$ in the request to B. When B receives the request it goes through a number of validation checks involving both authentication and authorization. If all checks succeed, the request is processed by the system. If any check fail, the system responds with an error code and details about the error.
5.2. AUTHENTICATION SYSTEM  CHAPTER 5. REDESIGNING THE BOOLI PLATFORM

5.2.2  Client to server authentication

The authentication requirements for the client to server requests are such that Booli wants to be able to distinguish between valid and invalid requests. This system cannot use the same authentication mechanisms as the server to server authentication system, since the client must be aware of the key that is attached to every request to Booli’s servers. Since this key can be read in plain text from the script source it must be such that knowing the key is not enough to generate a valid request from a random host. This means that the key must provide enough information to uniquely identify a domain. This can be implemented in a number of different ways; either by a method of encryption or by letting the key reference the domain for which it was generated. Given the domain a hash can be generated which is compared to the location of the client. Additionally, on the server side, the HTTP REFERER can be compared against the domain if provided. Not all clients provide this information though, so it can only be used if it is set. It is also easy to spoof the referrer, so it should not have a central role in the authentication process.

For Booli, the implementation of the client to server authentication system is described below. For each request to the server, the client attaches his API-key, together with the current location of the client from window.location. When the request is processed at the server, it first looks at the key and ensures that it is a known API key. If the server cannot find the key in its database, the request is rejected. Once the key has been located the server checks which domain it is valid for against the origin of the request in the domain parameter. This information is stored together with the key so that it may be updated and extended as needed. If the origin of the request is not valid for the supplied key, the request is again rejected. Finally, as a last security measure the server checks the HTTP REFERER. This parameter is not mandatory to set, and it is also quite easy to spoof, but together with the other security measures and an authorization system, we are confident that it is secure enough. If this parameter is set and differs from the supplied location, the request is rejected. If all of these checks pass, the request is considered authenticated and is allowed further processing. It should be noted that this authentication procedure is pretty weak, at least when compared to the server to server authentication procedure. An attacker could go to one of Booli’s partner sites, copy the JavaScript code that is downloaded there and paste it into his own
site. Then he would only need to change a couple of parameters in how the code requests data from Booli’s API to be pretty close to obtaining this data. The last obstacle to overcome is to spoof the HTTP referrer to either correspond to the site he originally copied the source code from, or unset it completely. We therefore recommend that only non-vital information can be retrieved through the client authentication system. It would, however, be a completely different story to take the source code and be able to serve it on one’s own site to other users without Booli’s approval. While it is easy to spoof one’s own referrer, it is much harder to spoof the referrer for your users. This means that it would make no sense to steal the code for the software modules outlined in this thesis, since you would not be able to serve it as your own anyway.

5.3 Authorization system

Booli wants different parts of their API to be available to different partners, with an easy way to change the permissions to the different parts on a partner to partner basis. This demand requires the API to run an authorization system on each incoming request. Once a request has been properly authenticated, it also needs to be authorized before the request can continue to be processed. Since the request is already authenticated it is a simple task to look up the permissions held by the requester. These permissions are then compared to the permissions needed to perform the request. Only if the requester holds all the permissions needed, to perform the request, is the request allowed to continue to process. If one or more permissions are missing, the request is stopped and a response sent back to the requester with details about the failed authorization.

The necessary granularity of the authorization system varies from module to module in the API. For instance, a broker is only authorized to access and modify his own listings and content, and any request trying to modify a listing that is not his own will be rejected. This kind of authorization requires a quite fine-grained authorization procedure that inspects the request and permissions thoroughly. On the other end of the spectrum are requests to download code to display different modules. The authorization system here only needs to check whether the requester is permitted to download the requested code or not without any deeper analysis. To accommodate these quite
different needs of request analysis, the authorization system consists of a core that accepts the request. After inspecting what kind of request it is processing it loads an authorization module that performs any further analysis of the request and its parameters. This modularized authorization system makes it easier to add deeper analysis where needed and keep other types of authorizations simple. The downside of this approach is that some analysis and authorization code might be duplicated between the different authorization modules, but it can be solved with some planning and proper class hierarchy architecture.

5.4 API for external requests

The Booli API is a web interface for retrieving code and data from external platforms and systems. One typical usage would be a partner of Booli that runs a website and wants the SearchBox module (1.2.1.1) implemented on their site. Typically Booli would supply the partner with a JavaScript code snippet to put on their site to handle all requests to the Booli API, but any software able to perform HTTP-get and HTTP-post requests and parse a response in JSON or XML format would be able to interface with the API. A key design guideline when constructing the API has been language and system independence.

5.4.1 Calling the API

A call to the Booli API, whether to retrieve code or data, is done via HTTP-get. The HTTP-post request is reserved for insertion of data into the database. The API is structured with a RESTful (Representational state transfer) architecture which means the controllers and actions are structured with a focus on resources [16]. To retrieve anything from the API a key has to be retrieved in beforehand from Booli which is then incorporated into each request.
5.4.2 Requesting code

A request for code from the Booli API is currently restricted to JavaScript code only. There will possibly, in the future, be code offered in other languages. A typical request for JavaScript code is the example mentioned above with the SearchBox module. To retrieve the code for a module, the following would be mandatory in the call:

- Repository base URL (http://api.booli.se/source-code/)
- Module name (SearchBox)
- Module version (1.0)
- Key (4934e4ce9a99e41ce68717fe0e972bf3a671187a)
- Domain (current client URL)

The repository base URL is the same for all source codes delivered by Booli. The module name on the other hand represents one of the modules described in 1.4.2 above. Since all Booli code that is implemented on third party platforms is retrieved from the Booli servers, updates are made to the code quite often. To avoid being forced to test the implementations on all partner sites on each update, a versioning system is implemented. This adds the requirement that the module version is added to each request. If version 1.0 is stated, future updates to the module will not be considered. The last parameter required is the domain. This is used in the authentication system to verify that the request is made from a domain which has an agreement with Booli and therefore a valid key. In the case of client-to-server requests, as in the case of JavaScript modules, the domain is set by the JavaScript keyword window.location.host which returns the host of which the client got the code. With all these parameters set, a typical request for the SearchBox module would look something like this:

http://api.booli.se/source-code/searchBox?version=1.0&key=4934e4ce9a99e41ce68717fe0e972bf3a671187a&domain=www.partnerwithkey.se
5.5. SUPPORT SYSTEMS

CHAPTER 5. REDESIGNING THE BOOLI PLATFORM

5.4.3 Requesting data

Even though a request for data is usually on the same format when it comes from an external implementation is when coming from within the Booli platform itself, we decided to route all requests from external sources through our API. This enables us to separate all requests from external partners when analyzing load on servers and also require key validation on all requests.

A request for data is done in pretty much the same manner as the request for code. The following parameters would be required in the case of retrieval of suggests:

- Data base URL (http://api.booli.se/suggest)
- Action to be taken (getSuggests)
- Request specific parameters
  - suggestString
  - nrOfAreas
  - nrOfStreets
- Key (c0cc732ae7148eb2e56ca1870d3d99212bebcb83)
- Domain (http://www.partnerwithkey.se/)

The request is done by a JavaScript code snippet and would look like this:

http://api.booli.se/suggest/getSuggests?suggestString=Stockholm&nrOfAreas=6&nrOfStreets=3&key=c0cc732ae7148eb2e56ca1870d3d99212bebcb83&domain=http%3A//www.partnerwithkey.se/

5.5 Support systems

Since the Booli API will deliver content and functionality to other business partners in their daily work and business model, it is essential that any issues related to the API are discovered quickly. The most preferable result is that issues, bugs and other problems can be found and resolved before
they reach the production stage, but this might not always be possible. Below we describe some support systems that try to achieve the above mentioned objectives.

5.5.1 Testing frameworks

To avoid severe bugs from reaching production code a couple of testing frameworks are used. In these frameworks tests can be automated and run both separately and as part of bigger suits of tests. Untested code is not allowed to be checked into the code repository and made available for deploy to the production servers. The code is tested on two different levels, namely unit tests and integration tests.

The unit tests test the public interface of each class in the code base and ensure that they return the expected result, or perform the expected side-effect, given specific input. This approach enables the developers to catch errors in the implementation before they reach production code, but this benefit relies much upon the quality of the tests themselves. Unit tests can only show the presence of errors, so a deficient unit test or a unit test that is not kept in synchronization as the code base evolves might very well miss bugs and errors. It is thus important to have a continuous and well-functioning process in place that updates unit tests as code is changed and vice-versa.

The integration tests test the system, or parts of the system, as they interact together. This can be done automatically, by testing some of the more central parts of the code without mocking their dependencies and ensuring that the generated result is the expected result, but usually it also requires some manual supervision, especially when it comes to evaluating the result visually. This means ensuring that elements of the result are placed correctly, have the correct font size and colour and so on.

The frameworks utilized in the Booli API are PHPUnit and JsUnit. Both frameworks are part of the xUnit test framework family originating from the work of Kent Beck on the SUnit test framework for SmallTalk [1]. They are used in conjunction with PhpUnderControl [14], which is a plug-in for the continuous integration tool CruiseControl [2]. PhpUnderControl can be used to run tests and test suites as well as visualize test results, test code coverage and enforce coding standards.
5.5.2 Availability monitoring

To ensure that the services provided by Booli’s API are available to partners at all times, it is a sound idea to have some kind of automatic availability monitoring system. There are numerous of these services available from various companies. The solutions vary from online distributed solutions, to desktop applications. The different approaches have different merits. The distributed solution can discover availability problems that are not visible by an application running from within the company’s intranet, for instance if the server becomes unreachable from some locations in the world, but is still available at others. This enables the company to update a service status page with information about the external availability problem and thus receive goodwill for good service and alertness. A desktop availability monitoring application, however, often contains more sophisticated tools for surveillance of the service and the system, and can often provide a more detailed analysis of the reason why the service is down than a remote surveillance system can. A local application could for instance monitor both CPU, memory and network utilization and use this information in its analysis.

Booli has decided to use both methods of surveillance to achieve the best possible monitoring. It uses an online distributed service to alert administrators whenever the service becomes unavailable from any location, and a local desktop application to monitor the health of the servers and alert administrators when critical levels are reached.

5.5.3 Utilization monitoring

To keep track of the utilization of the API and usage patterns of the users, a utilization monitoring system can be implemented. This was not done in the prototype implementation, since it is outside of the scope of the thesis, but we have tried to prepare the prototype for later integration with such a system by providing hooks that can be tapped into when needed. These hooks provide details about the user, what parameters were supplied in the request, which part of the API that was accessed and the result returned to the user.

It would also be possible to connect this support system to the authorization system, since Booli
might want to limit the allowed number of requests per day and user to the API sometime in the future. In this case, the authorization system could query the utilization monitor of the number of requests made thus far, and then decide whether or not to allow this request depending on the number of requests remaining.

### 5.5.4 Documentation generator

For any software system it is very important to have documentation, both a low level class- and method reference, but also at a higher level with examples and diagrams. This documentation should be readily available and up to date. A documentation generator can greatly help to achieve this goal, since it takes a lot of time and work to keep it at a satisfactory state manually. A documentation generator is a programming tool that generates documentation, often in various different output formats, from a set of files, often source code files [22]. This has numerous advantages both for the programmers and the end users. For the programmers, it is easy to update the documentation since all documentation about the method, class or module she is working on is available in the same file as the source code. This makes it a simple task to update the documentation simultaneously as the code. As long as this is done consistently, the API documentation will always be up to date. Keeping examples of how to use the system in nearly the same format, but in the same files also makes it much simpler to remember to update the examples when the interface changes. For the end users it is nice to have consistently up to date documentation, and this can give a very noticeable impact on how quickly and painless a system can be integrated to make use of another, in this case the Booli API. In Booli the documentation generator of choice is Doxygen[23]. Much of the code base is already well documented, but more can be done to keep it in good and current state. We propose that a routine for documenting and usage examples is established.

### 5.5.5 Technical discussion forum

Whenever an API, or other type of technical platform, is offered for use in other places, problems inevitably pop up. These can take a number of different shapes and often revolve around issues the
development team had never foreseen to be a problem. This has nothing to do with the competence of the development team itself, but rather that coming from the outside it is hard to understand how to properly use the different parts of the system together. For the development team, however, this is very basic and intuitive, since they have built it and know about their design choices. Apart from this clash of viewpoints, it is also likely that bugs will surface when more people start to use the API, perhaps in ways not intended by the development team. The reports that, hopefully, start coming in as bugs are discovered are invaluable for the development team, but the reporting user also likes to know that his discovery was important and appreciated.

A common, and simple to setup, solution to how to handle these problems is to setup a technical discussion forum. This is an approach taken by all of the companies benchmarked in the related work section of this report, and is also common among most of the big companies offering APIs, such as Yahoo and Microsoft. The discussion forum can be either public or private, but the important part is that every end user with privileges to use the API should also be able to post new discussion threads, reply to other problems and generally feel like he is part of a team solving implementation problems together. A forum can also alleviate some load off of the development team as the community starts to help each other out without the involvement of a company developer. It will, of course, take some time to get to this point, especially in a private forum.

Booli has not yet setup a technical discussion forum for issues regarding their API, but it is something we recommend that they do in the future.
Chapter 6

Evaluation

To assert that the design decisions we have taken throughout our thesis project fulfills the requirements as stated in 1.4 Task specification; we evaluated the prototype implementation in a number of ways. The evaluations were performed in different ways depending on the type of requirement. For purely technical requirements the evaluation typically consisted of more formal machine test, such as a unit test suite, while more loosely stated requirements, such as ease of implementation, where evaluated by the number of support requests from partners.

6.1 Platform independence

Since the scope of this report only covers common web development platforms, and especially those used by Booli and its partners, the evaluation of this requirement will be done considering those platforms. As mentioned earlier, these are the operating systems, servers and languages used by Booli and its partners today:

- Operating systems
  - Windows 2000 Server
  - Ubuntu Linux
  - Debian Linux
6.1. PLATFORM INDEPENDENCE

- Web servers
  - Apache
  - Apache 2
  - Microsoft IIS

- Server side languages
  - PHP 4
  - PHP 5
  - ASP
  - CGI

During our evaluation we implemented the Booli platform modules on systems with the following configurations:

- System 1
  - Operating system: Ubuntu Linux
  - Web server: Apache 2
  - Server side language: PHP 5

- System 2
  - Operating system: Windows 2000 Server
  - Web server: Microsoft IIS
  - Server side language: ASP.NET

- System 3
  - Operating system: Debian Linux
  - Web server: Apache 2
6.2. EASE OF IMPLEMENTATION

- Server side language: CGI

- System 4
  - Operating system: Mac OS X
  - Web server: Apache 2
  - Server side language: PHP 5

System 1 is one of Booli development servers while System 4 is one of our development laptops. These systems are the ones where most of the laboratory practical work was done and were therefore more exhaustively tested than the other systems. System 2 and 3 are the servers of two of Booli partners where the prototype modules were implemented.

The selected solution with cross domain JavaScript and minimal server side scripting for the partner modules gave very good results for the platform independence tests. All implementations were made without any problems due to environment differentiation except for system 2. This system uses a content management system which placed a HTML form tag as the outer most element. Since the HTML specification does not allow nested forms [15], our SearchBox was not rendered properly. This was easily solved by simply moving the SearchBox form outside of the CMS form.

Even though the problem was easily solved, it is none the less a shortcoming in our design. The use of an outer form on web pages is not uncommon, and the problem will therefore probably surface again. A more thorough discussion regarding this issue can be found under 7 Future Work.

6.2 Ease of implementation

Since the ease with which a solution is implemented is not easily tested formally, the evaluation of this requirement was done by communication with the partners who implemented the modules. Except for the problem mentioned with nested forms, all partners implemented the solutions with an average of two technical questions per implementation and an average implementation time of four hours.
This evaluation is not to be considered a final measurement for how easy the solution was implemented since not enough partners have implemented it yet, but should rather be viewed as a hint of how future implementations might go. We therefore considered this a positive result for the ease of implementation requirement.

6.3 Preserved performance of Booli.se

The preserved performance of Booli.se was evaluated in two ways; the amount of bugs inflicted upon the platform by the customization and the average response time on the software modules on Booli.se.

The number of bugs introduced in the platform, by the customizations required to implement a prototype API solution, during the two months the solution were first implemented, tested, updated and deployed was one. This bug was due to a mix-up where one of the developers mistook a JavaScript override file for its original counterpart in the Booli platform when working on an update for one of the modules. This was mainly a human error and could not be attributed to the choice of solution. It could, however, act as a sign of warning that additional care has to be taken to properly document the override files so they can not be mistaken for their original counterparts.

Since we chose a solution where no original code, that is sent out to the client, is altered but rather extended by overriding code, it would be meaningless to measure possible increase of response time on booli.se. Any change in response time on booli.se is due to refactoring, and since this is an ever ongoing process that could not be isolated from our prototype, it would not give any conclusive measurements of whether the response time had increased due to the introduction of the API functionality. In fact, during our time at Booli, the average response time for a request was lowered drastically due to optimizations in the back end structure and algorithms.

With one bug, which was mainly introduced due to the human factor, and with no possibility to measure how the API has affected the response time of booli.se isolated from other concurrent optimizations of the code, the performance requirement can be considered fulfilled.
6.4 Preserved functionality of Booli.se

As stated in 6.3 Preserved performance of Booli.se, none of the code sent out and executed in a client’s browser was altered during the customization. Therefore, no functionality at booli.se was affected. The requirement, however, also included preserved development possibilities for the platform. This part of the requirement is achieved by the use of a platform JavaScript file version history. However, these JavaScript files make certain assumptions about the interface of the backend and where the resources to fetch specific data is located. This means that even if the interface of the platform changes, these old resources must still be backward compatible, to ensure that all versions of the modules remain functional, at least until all implementing partners have had a reasonable amount of time to upgrade to a newer version. To support this backward compatibility the controllers, i.e. resources, used by the partner modules are separated from the controllers used within the rest of the platform by residing on another sub domain, api.booli.se. This lets Booli alter the regular controllers without affecting the partner modules; however, there is a limit to how much this can be done until the spread between the two different controller sets is so big that maintaining both interfaces becomes unmanagable.

Due to these reservations of having to support backward compatibility, since partners may have old versions of the modules implemented, the requirement of preserved functionality of booli.se should be considered only partially fulfilled.

6.5 Authentication

We feel that we have designed an authentication system that is secure enough to fulfil the requirements outlined in section 1.4 Task specification. However, a couple of issues exist, that might need further investigation and work depending on how severe Booli finds them.

The first issue is that the client authentication is pretty weak, at least compared to the server-server authentication procedure. An attacker could go to one of Booli’s partner sites, copy the JavaScript code that is downloaded there and paste it into his own site. The he would only need
6.6. AUTHORIZATION

The authorization system we have outlined is both flexible and powerful. It will be no problem to fulfill the requirements as outlined in section 1.4. As the API grows and the number of different services offered increases, Booli will need to ensure that modules for authorization are added as well. This should not be a big problem, but it should be noted that we have not solved the authorization problem for all future applications of the API.
6.7 Software update system

The performance of the update system was evaluated by pushing updated code to the partners of Booli which implemented the solution on three different occasions. This is done by simply changing the code in the code repository stored on Booli’s servers. The changes are then propagated to partners automatically.

This evaluation was done by performing three individual upgrades. The updates were:

1. Minor layout update to the SearchBox module
2. Update for the authentication system
3. Update of the JavaScript Suggest algorithm

Updates one and three were made without any problems whatsoever on either the implementation or the API side of the solution. Update number two on the other hand required an adjustment in the authentication system to which the partners had configuration strings on their servers. This forced us to contact the partners manually and ask them to update the configuration by hand. Even though this is suboptimal, the small amount of code implemented on the partner platforms should not be updated very often. The update to the authentication system was done simply because the partner modules are in an early stage of development severe problems could still be detected.

Since the only complication with the software update system was due to the early nature of the solution, the software update system requirement can be considered fulfilled.
Chapter 7

Future work

While we feel that this project has been largely successful and yielded a lot of positive results, we have identified a number of areas that need further work once this master thesis project is finished.

7.1 Implementation of software modules

As previously stated in the limitations section of the report, the aim of the thesis was not to implement production ready software modules. While prototypes have been developed for each of the modules listed in section 1.4.2 Modules, they still need to undergo more extensive testing and in some cases redesign to fit into a production environment in a reliable and scalable manner. We propose that Booli use our prototypes as an inspiration and proof of concept, but practice throw-away prototyping to ensure that all constraints are fulfilled for their future partner deals. One exception to this recommendation is the SearchBox module which already has been put through a number of iterations of throw-away prototyping during our work on this thesis, and it is close to being production ready. This module would be well suited for evolutionary prototyping for the rest of its development cycle. Some additional development could be added to the SearchBox to avoid problems with CMS systems that use HTML form tags that wrap whole pages. This could be solved by decreasing the dependence on the form structure of the SearchBox, and if available, make the input field a part of the wrapping form. Another solution would reimplementing the SearchBox as
a Flash based module.

Finally, additional modules that are not outlined in this document might need to modify some of the existing code to be able to be included into the API. We have tried to have an extensible and flexible solution in mind while redesigning the Booli platform, but we did not take every possible future scenario into account since this would have made the solution unnecessarily complex and hard to maintain.

### 7.2 Authentication system

The server to server authentication system requires Booli to implement some kind of key sharing scheme as the number of partners, that requires the kind of security this authentication system provides, grow. This might be difficult and time consuming work, depending on just how secure and automatic Booli wants the key sharing scheme to be. In its simplest form Booli can just settle for a SSL-encrypted communication channel to distribute the keys. We recommend, however, that a little more thought is put into it to ensure that the system is not easy to exploit, since this might very well generate badwill for the company.

### 7.3 Authorization system

The authorization system implemented during this project is solid and flexible enough to accommodate any of the authorization constraints we could identify by looking at the requirements put forward by Booli. It should be noted, however, that we only implemented the framework for the authorization system, and authorization modules needs to be created to implement and enforce specific authorization constraints and requirements as they emerge. Details of which parts of the authorization system that are complete and which remains to be implemented has been turned in to Booli.
7.4 Support systems

Some of the support systems we identified as essential for a well-functioning web service system have not yet at the time of writing been implemented. Namely, it is the utilization monitoring system and the technical discussion forum. The first system can provide BooLi with information about how, when and by whom the system is used. This information can be invaluable in analysis of work on the API. The technical discussion forum has manifold merits. Firstly, it provides partners with a constantly available channel of communication with both BooLi’s developers, but also other partners implementing the BooLi API. This will probably generate an impression of reliability, transparency and service-mindedness. Secondly, it allows partners who have similar problems to help each other on the forum and thus create a community around the BooLi API. Thirdly, the forum enables BooLi to create a knowledgebase of common problems and their solutions. This can help save time since partners with a specific problem that has already been solved can be directed to the knowledgebase. We recommend that these systems are setup as soon as possible.
Chapter 8

Acknowledgements

During this master thesis, and especially during the writing of this report, we have had a lot of help from the people around us and would like to extend our appreciation for the work they have put in. First and foremost we would like to thank Booli Search Technologies AB. The company has showed a lot of interest in our work from day one, and have never let us wait more than a few minutes when we needed technical guidance. They have also been very helpful in providing hardware and office space for us to work in.

When it comes to more formal questions about the format and content of this report, the Department of Information Technology at Uppsala University for whom we wrote the report, has been of much help. Our reviewer Mikael Pettersson and examiner Anders Jansson at the university have been especially helpful.

We would also like to extend a general thank you to all the people on the internet forums around the web that have participated in discussions and answered questions on their spare time. Their help has definitely given us interesting input and different angles to approach difficult problems.

Last but not least, we would like to thank the people around us, friends, family and girlfriends for being so understanding about us working late nights on this project.

Thank you!
Bibliography


[8] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. Design Patterns - Elements of Reusable Object-Oriented Software. Addison-Wesley, 1994.


# Appendix A

## Google API key - empiric study material

<table>
<thead>
<tr>
<th>Domain</th>
<th>Google User Account</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.test.com">http://www.test.com</a></td>
<td><a href="mailto:patrik.akerstrand@gmail.com">patrik.akerstrand@gmail.com</a></td>
</tr>
<tr>
<td>ABQIAAAAL85MK8VaXwruE0y2eLb72RRQ2moUQsuJWyAJRdaRYWIWzZyo4hTF58nGZ-rV3ygahVtzhqItIawHIQ</td>
<td></td>
</tr>
<tr>
<td><a href="http://www.test.com">http://www.test.com</a></td>
<td><a href="mailto:fredrik.zettergren@gmail.com">fredrik.zettergren@gmail.com</a></td>
</tr>
<tr>
<td>ABQIAAAAOnZSLdy_nVFySSM-uFl3ARQ2moUQsuJWyAJRdaRYWIWzZyo4hTSvywMnDhwAodlZSS3-u7crbFZNw</td>
<td></td>
</tr>
<tr>
<td><a href="https://www.test.com">https://www.test.com</a></td>
<td><a href="mailto:booli.se@gmail.com">booli.se@gmail.com</a></td>
</tr>
<tr>
<td>ABQIAAAAADqahHKW-libzsd6kt-6p4RRAP4Q5dMxfgbkzUXrzlXPui4lkBhRlkIEC4zatCU767Bx-DDZP2HHJEw</td>
<td></td>
</tr>
<tr>
<td><a href="https://www.test.com">https://www.test.com</a></td>
<td><a href="mailto:patrik.akerstrands@gmail.com">patrik.akerstrands@gmail.com</a></td>
</tr>
<tr>
<td>ABQIAAAAAOnZSld_y_nVFySSM-uFl3ARRAP4Q5dMxfgbkzUXrzlXPui4lkBhQmBgyU14iNMSPhbEzG1Ll2u3pcKg</td>
<td></td>
</tr>
<tr>
<td><a href="https://www.test.com">https://www.test.com</a></td>
<td><a href="mailto:booli.se@gmail.com">booli.se@gmail.com</a></td>
</tr>
<tr>
<td>ABQIAAAAADqahHKW-libzsd6kt-6p4RRAP4Q5dMxfgbkzUXrzlXPui4lkSx8agYjJ5o3Of21KLwDmgObHh10A</td>
<td></td>
</tr>
<tr>
<td><a href="http://www.anotherpage.com">http://www.anotherpage.com</a></td>
<td><a href="mailto:patrik.akerstrand@gmail.com">patrik.akerstrand@gmail.com</a></td>
</tr>
<tr>
<td>ABQIAAAAAL85MK8VaXwruE0y2eLb72RRHe-oXyZpG7WDgcvBIVfwOgPAQehQoOEEHaU0E1Pzt4zwBFyc5ifyyCA</td>
<td></td>
</tr>
<tr>
<td><a href="http://www.anotherpage.com">http://www.anotherpage.com</a></td>
<td><a href="mailto:fredrik.zettergren@gmail.com">fredrik.zettergren@gmail.com</a></td>
</tr>
<tr>
<td>ABQIAAAAAOnZSld_y_nVFySSM-uFl3ARRHe-oXyZpG7WDgcvBIVfwOgPAQehRLXmdJyEDZtyrceF8MKNsMMo9jMw</td>
<td></td>
</tr>
<tr>
<td><a href="http://www.anotherpage.com">http://www.anotherpage.com</a></td>
<td><a href="mailto:booli.se@gmail.com">booli.se@gmail.com</a></td>
</tr>
<tr>
<td>ABQIAAAAADqahHKW-libzsd6kt-6p4RRHe-oXyZpG7WDgcvBIVfwOgPAQehTh74fKb44-8KDn-4EJuT0yC4s0WQ</td>
<td></td>
</tr>
</tbody>
</table>

65
# Appendix B

## Cross domain performance tests

### Test results

Table B.1: Test results of server side proxy versus Dynamic Script Tag

<table>
<thead>
<tr>
<th>Test nr</th>
<th>Response time (ms)</th>
<th>Successful</th>
<th>Test nr</th>
<th>Response time (ms)</th>
<th>Successful</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>124</td>
<td>Yes</td>
<td>1</td>
<td>219</td>
<td>Yes</td>
</tr>
<tr>
<td>2</td>
<td>125</td>
<td>Yes</td>
<td>2</td>
<td>112</td>
<td>Yes</td>
</tr>
<tr>
<td>3</td>
<td>109</td>
<td>Yes</td>
<td>3</td>
<td>106</td>
<td>Yes</td>
</tr>
<tr>
<td>4</td>
<td>105</td>
<td>Yes</td>
<td>4</td>
<td>106</td>
<td>Yes</td>
</tr>
<tr>
<td>5</td>
<td>102</td>
<td>Yes</td>
<td>5</td>
<td>108</td>
<td>Yes</td>
</tr>
<tr>
<td>6</td>
<td>102</td>
<td>Yes</td>
<td>6</td>
<td>104</td>
<td>Yes</td>
</tr>
<tr>
<td>7</td>
<td>102</td>
<td>Yes</td>
<td>7</td>
<td>103</td>
<td>Yes</td>
</tr>
<tr>
<td>8</td>
<td>101</td>
<td>Yes</td>
<td>8</td>
<td>98</td>
<td>Yes</td>
</tr>
<tr>
<td>9</td>
<td>100</td>
<td>Yes</td>
<td>9</td>
<td>103</td>
<td>Yes</td>
</tr>
<tr>
<td>10</td>
<td>102</td>
<td>Yes</td>
<td>10</td>
<td>110</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Continued on next page
### Table B.1 – continued from previous page

<table>
<thead>
<tr>
<th>Server side proxy</th>
<th>Dynamic script tag</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Test nr</strong></td>
<td><strong>Response time (ms)</strong></td>
</tr>
<tr>
<td>11</td>
<td>104</td>
</tr>
<tr>
<td>12</td>
<td>106</td>
</tr>
<tr>
<td>13</td>
<td>105</td>
</tr>
<tr>
<td>14</td>
<td>119</td>
</tr>
<tr>
<td>15</td>
<td>322</td>
</tr>
<tr>
<td>16</td>
<td>120</td>
</tr>
<tr>
<td>17</td>
<td>115</td>
</tr>
<tr>
<td>18</td>
<td>121</td>
</tr>
<tr>
<td>19</td>
<td>122</td>
</tr>
<tr>
<td>20</td>
<td>122</td>
</tr>
<tr>
<td>21</td>
<td>116</td>
</tr>
<tr>
<td>22</td>
<td>121</td>
</tr>
<tr>
<td>23</td>
<td>119</td>
</tr>
<tr>
<td>24</td>
<td>131</td>
</tr>
<tr>
<td>25</td>
<td>109</td>
</tr>
<tr>
<td>26</td>
<td>99</td>
</tr>
<tr>
<td>27</td>
<td>109</td>
</tr>
<tr>
<td>28</td>
<td>109</td>
</tr>
<tr>
<td>29</td>
<td>107</td>
</tr>
<tr>
<td>30</td>
<td>110</td>
</tr>
<tr>
<td>31</td>
<td>136</td>
</tr>
<tr>
<td>32</td>
<td>114</td>
</tr>
<tr>
<td>33</td>
<td>119</td>
</tr>
<tr>
<td>34</td>
<td>121</td>
</tr>
</tbody>
</table>

Continued on next page
## APPENDIX B. CROSS DOMAIN PERFORMANCE TESTS

Table B.1 – continued from previous page

<table>
<thead>
<tr>
<th>Server side proxy</th>
<th>Dynamic script tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test nr</td>
<td>Response time (ms)</td>
</tr>
<tr>
<td>35</td>
<td>117</td>
</tr>
<tr>
<td>36</td>
<td>122</td>
</tr>
<tr>
<td>37</td>
<td>110</td>
</tr>
<tr>
<td>38</td>
<td>103</td>
</tr>
<tr>
<td>39</td>
<td>101</td>
</tr>
<tr>
<td>40</td>
<td>102</td>
</tr>
<tr>
<td>41</td>
<td>102</td>
</tr>
<tr>
<td>42</td>
<td>103</td>
</tr>
<tr>
<td>43</td>
<td>105</td>
</tr>
<tr>
<td>44</td>
<td>101</td>
</tr>
<tr>
<td>45</td>
<td>104</td>
</tr>
<tr>
<td>46</td>
<td>110</td>
</tr>
<tr>
<td>47</td>
<td>113</td>
</tr>
<tr>
<td>48</td>
<td>108</td>
</tr>
<tr>
<td>49</td>
<td>112</td>
</tr>
<tr>
<td>50</td>
<td>111</td>
</tr>
<tr>
<td>51</td>
<td>105</td>
</tr>
<tr>
<td>52</td>
<td>102</td>
</tr>
<tr>
<td>53</td>
<td>204</td>
</tr>
<tr>
<td>54</td>
<td>188</td>
</tr>
<tr>
<td>55</td>
<td>174</td>
</tr>
<tr>
<td>56</td>
<td>255</td>
</tr>
<tr>
<td>57</td>
<td>362</td>
</tr>
<tr>
<td>58</td>
<td>207</td>
</tr>
</tbody>
</table>

Continued on next page
Table B.1 – continued from previous page

<table>
<thead>
<tr>
<th>Server side proxy</th>
<th>Dynamic script tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test nr</td>
<td>Response time (ms)</td>
</tr>
<tr>
<td>59</td>
<td>117</td>
</tr>
<tr>
<td>60</td>
<td>121</td>
</tr>
<tr>
<td>61</td>
<td>103</td>
</tr>
<tr>
<td>62</td>
<td>99</td>
</tr>
<tr>
<td>63</td>
<td>144</td>
</tr>
<tr>
<td>64</td>
<td>108</td>
</tr>
<tr>
<td>65</td>
<td>116</td>
</tr>
<tr>
<td>66</td>
<td>169</td>
</tr>
<tr>
<td>67</td>
<td>164</td>
</tr>
<tr>
<td>68</td>
<td>123</td>
</tr>
<tr>
<td>69</td>
<td>133</td>
</tr>
<tr>
<td>70</td>
<td>129</td>
</tr>
<tr>
<td>71</td>
<td>117</td>
</tr>
<tr>
<td>72</td>
<td>110</td>
</tr>
<tr>
<td>73</td>
<td>102</td>
</tr>
<tr>
<td>74</td>
<td>100</td>
</tr>
<tr>
<td>75</td>
<td>99</td>
</tr>
<tr>
<td>76</td>
<td>108</td>
</tr>
<tr>
<td>77</td>
<td>102</td>
</tr>
<tr>
<td>78</td>
<td>98</td>
</tr>
<tr>
<td>79</td>
<td>103</td>
</tr>
<tr>
<td>80</td>
<td>104</td>
</tr>
<tr>
<td>81</td>
<td>105</td>
</tr>
<tr>
<td>82</td>
<td>115</td>
</tr>
</tbody>
</table>

Continued on next page
Table B.1 – continued from previous page

<table>
<thead>
<tr>
<th>Test nr</th>
<th>Response time (ms)</th>
<th>Successful</th>
<th>Test nr</th>
<th>Response time (ms)</th>
<th>Successful</th>
</tr>
</thead>
<tbody>
<tr>
<td>83</td>
<td>310</td>
<td>Yes</td>
<td>83</td>
<td>249</td>
<td>Yes</td>
</tr>
<tr>
<td>84</td>
<td>119</td>
<td>Yes</td>
<td>84</td>
<td>119</td>
<td>Yes</td>
</tr>
<tr>
<td>85</td>
<td>119</td>
<td>Yes</td>
<td>85</td>
<td>108</td>
<td>Yes</td>
</tr>
<tr>
<td>86</td>
<td>164</td>
<td>Yes</td>
<td>86</td>
<td>504</td>
<td>Yes</td>
</tr>
<tr>
<td>87</td>
<td>167</td>
<td>Yes</td>
<td>87</td>
<td>105</td>
<td>Yes</td>
</tr>
<tr>
<td>88</td>
<td>159</td>
<td>Yes</td>
<td>88</td>
<td>100</td>
<td>Yes</td>
</tr>
<tr>
<td>89</td>
<td>132</td>
<td>Yes</td>
<td>89</td>
<td>103</td>
<td>Yes</td>
</tr>
<tr>
<td>90</td>
<td>128</td>
<td>Yes</td>
<td>90</td>
<td>102</td>
<td>Yes</td>
</tr>
<tr>
<td>91</td>
<td>134</td>
<td>Yes</td>
<td>91</td>
<td>102</td>
<td>Yes</td>
</tr>
<tr>
<td>92</td>
<td>150</td>
<td>Yes</td>
<td>92</td>
<td>109</td>
<td>Yes</td>
</tr>
<tr>
<td>93</td>
<td>128</td>
<td>Yes</td>
<td>93</td>
<td>107</td>
<td>Yes</td>
</tr>
<tr>
<td>94</td>
<td>174</td>
<td>Yes</td>
<td>94</td>
<td>112</td>
<td>Yes</td>
</tr>
<tr>
<td>95</td>
<td>443</td>
<td>Yes</td>
<td>95</td>
<td>119</td>
<td>Yes</td>
</tr>
<tr>
<td>96</td>
<td>104</td>
<td>Yes</td>
<td>96</td>
<td>107</td>
<td>Yes</td>
</tr>
<tr>
<td>97</td>
<td>102</td>
<td>Yes</td>
<td>97</td>
<td>107</td>
<td>Yes</td>
</tr>
<tr>
<td>98</td>
<td>104</td>
<td>Yes</td>
<td>98</td>
<td>108</td>
<td>Yes</td>
</tr>
<tr>
<td>99</td>
<td>104</td>
<td>Yes</td>
<td>99</td>
<td>109</td>
<td>Yes</td>
</tr>
<tr>
<td>100</td>
<td>104</td>
<td>Yes</td>
<td>100</td>
<td>118</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table B.2: Statistics

<table>
<thead>
<tr>
<th>Method</th>
<th>Server side proxy</th>
<th>Dynamic script tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average response time (ms)</td>
<td>130.03</td>
<td>139.41</td>
</tr>
<tr>
<td>Successful attempts (%)</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>