Methodologies and Architecture for the Implementation of a Web Application

Metoder och arkitektur för utveckling av en webbapplikation

Lisan Chen
Gabriel Banfalvi

Examensarbete inom information- och programvarusystem
Kandidat
Degree Project in Information and Software Systems

Stockholm, Sweden 2012
Kurs II121X, 15hp
TRITA-ICT-EX-2012:328
Methodologies and Architecture for the Implementation of a Web Application

Bachelor’s Thesis

Lisan Chen & Gabriel Banfalvi
For Kungliga Tekniska Högskolan
Spring – Summer 2011

This is a bachelor’s project for KTH researching the usability of web applications and the usefulness of a number of new web-related technologies. To do this, it intends to rewrite a collaboration desktop application as a web application and evaluate the process. This report discusses the project and provides valuable information to developers and researchers seeking to write their own web applications.

Detta är ett examensarbete vid Kungliga Tekniska Högskolan, som undersöker användbarheten av webbapplikationer och ett antal nya webbteknologier. Som del av arbetet kommer ett traditionellt datorprogram skrivas om som en webbapplikation, och denna process kommer att utvärderas och dokumenteras skriftligt. Denna rapport syftar till att ge värdefull information för utvecklare och forskare som är intresserade av webbapplikationer i en djupare nivå.
Contents

1. Introduction.............................................................................................................................................. 1
   1.1 Thesis Background ................................................................................................................................. 1
   1.2 Problem and Objectives ......................................................................................................................... 2
   1.3 Report Disposition ................................................................................................................................. 2
   1.4 Report Language .................................................................................................................................... 3
2. Extended Background and Related Work.................................................................................................... 3
   2.1 FISTY, the Student Sharing Application ............................................................................................... 4
      2.1.1 Problems with FISTY ...................................................................................................................... 4
   2.2 Current cross-platform technologies ..................................................................................................... 5
      2.2.1 Oracle’s Java ................................................................................................................................... 5
      2.2.2 Microsoft’s .NET Framework ......................................................................................................... 6
      2.2.3 Adobe Flash ..................................................................................................................................... 6
      2.2.4 Adobe Air ......................................................................................................................................... 7
      2.2.4.1 Conclusions about alternative platforms .................................................................................... 7
   2.3 Web browser applications ...................................................................................................................... 7
   2.4 Technologies related to web development .............................................................................................. 8
      2.4.1 WebSocket ....................................................................................................................................... 9
      2.4.2 Web Storage / DOM Storage ........................................................................................................... 9
      2.4.3 HTML5 Cache manifests ................................................................................................................ 10
      2.4.4 HTML5 Canvas element ................................................................................................................ 11
      2.4.5 HTML5 drag and drop .................................................................................................................... 11
3. Project Criteria and Process ...................................................................................................................... 12
   3.1 Project criteria ....................................................................................................................................... 12
   3.2 Development Process .......................................................................................................................... 13
      3.2.1 Methodology ................................................................................................................................... 13
4. Project Design .......................................................................................................................................... 15
   4.1 Software architecture ............................................................................................................................ 15
      4.1.1 Extending the client-server model .................................................................................................. 16
   4.2 The application’s database .................................................................................................................. 18
   4.3 In-application user interface ................................................................................................................ 19
      4.3.1 Single document interface ............................................................................................................. 19
5. Implementation .................................................................................................................. 22
   5.1 First steps into web application development ............................................................. 22
   5.2 Implementing the site with jQuery and JavaScript ...................................................... 23
   5.3 Struggles with WebSockets ......................................................................................... 25
       5.3.1 JWebSocket ........................................................................................................... 25
       5.3.2 PHP-WebSocket ................................................................................................... 25
       5.3.3 Node.js and Socket.io ......................................................................................... 25
   5.4 Implementing all the features ...................................................................................... 26
       5.4.1 Contacts ................................................................................................................ 26
       5.4.2 Whiteboard .......................................................................................................... 27
       5.4.3 Messages .............................................................................................................. 28
       5.4.4 Chat ...................................................................................................................... 29
   5.5 Remainder of features .................................................................................................. 30
6. Results .................................................................................................................................. 30
   6.1 Evaluation of the project results .................................................................................. 30
7. Discussion ............................................................................................................................ 31
   7.1 Mistakes and setbacks during development ................................................................. 31
   7.2 What could have been done differently ....................................................................... 32
   7.3 On the viability of web applications ............................................................................. 32
8. Conclusion ............................................................................................................................ 34
9. References ........................................................................................................................... 34
Appendix A – Term Glossary .................................................................................................. 35
1. Introduction
(By Gabriel Banfalvi)

This chapter provides the reader with a short overview of the project. It starts by giving a general description of the state of software development today. It then lists a set of issues developers face and discusses how this project intends to deal with these problems. The last two sections of this chapter present the layout for the rest of the report and explain what language is going to be used.

1.1 Thesis Background

In the last few decades, computer technologies have evolved at a very high pace. As computer hardware continually evolves and targets greater and more challenging problems, so does the software that runs on it. One of the latest challenges software developers have is related to the interconnection of vast numbers of computers and finding ways users can take advantage of this networking.

One of the ways this was achieved by a group of students, was by developing a collaboration application, written in Java, that students can use to work together. It is described in greater detail in Chapter 2.

During and after its development a number of development issues arose:

- **Resource usage for software updates and distribution:** As it was being developed, this application was updated almost on a daily basis. Due to the software’s nature, it was necessary for users to always have the latest version. Developing a system to deal with this problem can be expensive and a waste of resources.

- **Cross-platform development compromises:** Java is successful at ensuring that the application runs on several operating systems, but at the same time, it forces developers to program in a way that caters to the lowest common denominator of all platforms to maintain consistency.

- **Data management:** As an increasingly larger set of users starts using an application like the aforementioned one, with a high number of clients (the application follows a client-server model, as the one seen in Figure 1), bandwidth consumption increases drastically.
1.2 Problem and Objectives
The aforementioned issues present themselves to a large number of developers and threaten the long-term survival of numerous projects as their user-base expands. How can a developer deal with these obstacles? By changing software platforms and allocating more hardware resources, developers can try to circumvent them (these alternatives will be explored in Chapter 2), but there can be other ways of dealing with these problems.

This thesis researches the development of programs that can run in the web browser (known as web applications) and new technologies associated with web applications as an alternative to traditional cross-platform applications. This is done by re-implementing a realistically-sized desktop program as a web application.

The questions this project seeks to answer are these:

- Can a web application replace a desktop application?
- Can web applications resolve issues other operating system-agnostic platforms cannot?
- What limitations do web applications face today?

This research will hopefully give other developers, through the experience of the development process, better awareness when faced with their own choices. The result of this implementation and the answer to these questions is presented in Chapters 6 and 7 respectively.

1.3 Report Disposition
Chapter two provides an in-depth background to this project. It describes the Java application mentioned in Section 1.1; it gives the reader an overview of the state cross-platform development today; it finally introduces web applications and technologies related to them.

Chapter three describes the criteria that will be used in this thesis to evaluate web applications. These criteria will be later used to answer the preceding questions. It also gives
a description of the project’s preparation stage, what software and other resources the team would use and what the kind of development process was chosen.

Chapter four explores the actual design stage of the project. It talks about what kind of architecture the software is designed to have, how the database is modeled and the project’s preliminary user interface designs.

Chapter five discusses the implementation process along with its setbacks, discoveries and shows the final model of the implementation. It also explains very briefly why some parts have not been implemented.

Chapter six evaluates the implementation according to the criteria in chapter three and discusses what it was like to implement a web application.

Chapter seven reviews the results of the project and discusses the mistakes that were made, and what should have been done differently. It then uses the results to answer the questions in the introduction and provide a more experienced view on the software development today. It also gives prospective developers recommendations for their future projects and suggests possible paths researchers interested in following this project can follow.

Chapter eight summarizes the contents of this project as a conclusion.

Chapter nine is a list of all the works cited throughout this report.

Appendix A is a glossary of all the technical terms used throughout this document and Appendix B is a copy of all the source-code written throughout this project.

1.4 Report Language
The report uses a moderately technical language and the reader is expected to have a basic understanding of how a computer as well as how computer networking works. The reader should also be familiar with a small part of the relevant terminology.

All terms and acronyms are explained when they’re first mentioned, but there’s also a glossary of frequently recurring terms in Appendix A.

A small amount of programming knowledge is needed to understand the contents of the report. A more in-depth understanding is required if the reader wishes to see how specific parts of our application work for himself. Source-code is attached in Appendix B.

Mentions of “the team”, “the development team” or “the project team” refer to the working members of this thesis project, Gabriel Banfalvi and Lisan Chen.

2. Extended Background and Related Work
(By Gabriel Banfalvi & Lisan Chen)
This chapter serves as context to the remainder of the report. It provides a background of the team's previous experience developing a networked Java application – which is the basis for the application to be developed in this project’s implementation stage. It also describes in depth the issues that arose during development and how they are usually dealt with. It finally explores web applications and software related to them.

2.1 FISTY, the Student Sharing Application
(By Lisan Chen)

As a second year project, a number of students at KTH (including the members of this thesis) developed a project collaboration application in Java.

The program would present an interface where a user can create an account with a username and a password. When he/she logs in, the user is shown a contact list. To it, he/she may add other users in the system and send instant messages to them. The focus of the program, however, lies in the project management.

If two or more users wishes to work together, a “project” can be created. A project is special type of group in the system. Within the project, members can share files, post “tasks” to be done in the project, and message or chat with other members. An additional feature that was added to the project was an electronic whiteboard. All members of a group could simultaneously draw onto this whiteboard and the image would update for everybody.

The purpose was to let students work and study together, despite not being in the same location.

The program was developed in Java 2 Standard Edition, a software platform the development team believed every student had. The platform’s main advantage was that it did not matter what operating system users were running. It connected to the project’s own custom server application which in turn stored all data MySQL Database server that managed accounts and other information (e.g. tasks and messages).

2.1.1 Problems with FISTY

This program worked well and the group succeeded at achieving its objectives with the final version of the product. Unfortunately, there were a number of issues that could not be solved and had to be simply ignored, as they lied outside of the scope of the project. To avoid redundancy, the reader is directed to Section 1.1 of this report, where they are listed. The following were possible ways to circumvent these issues:

*Resource usage for software updates and distribution*: To deal with this issue an update server would have had to be set up with extra bandwidth to not interfere with the rest of the application’s functions. If this is done, the client would either have to check every time it is initialized or the server would have to find a way to notify clients after each release.
Cross-platform development compromises: Catering to common standards is understandable – it would have been wrong of the developer to include features that would only be available to one set of users – but it worsened the user’s experience when he was forced to use an application that cannot take advantage of any modern functionality his operating system provides. The clearest solution would be to develop an application for each platform, but that nullifies the purpose of using Java in the first place.

Data management: The program did not need all the information it downloaded every time (it would, for example, download all the messages). An option would be to selectively send only the files and resources a user needs at a specific moment, which might incur in a high number of requests. Alternatively, if this becomes difficult to implement, one can extend the host’s bandwidth and hardware. Finally, one may store the downloaded on the local computer and then attempt to synchronize the data when a connection is established.

The conclusion the development team came to was that if it would have been forced to turn to these solutions it would have been better to find other, better-suited tools to accomplish the job with instead.

2.2 Current cross-platform technologies
(By Gabriel Banfalvi)

There are a large number of platforms that have been developed with the aim of simplifying cross-platform development. The reader is encouraged to research them further, as they might provide solutions to the problems he might be dealing with as a developer.

2.2.1 Oracle’s Java

Java is a collection of software products originally developed by Sun Microsystems. It offers a complete solution that allows developers to create applications and deploy them in a large number of environments. Java is used in an incredibly wide variety of computing platforms – from small embedded devices like phones and Blu-Ray players, to large networks of supercomputers – and its different “editions”¹ allow developers to specify which audience they wish to target.

Source-code written in Java is not specific to any processor or operating system. It instead works by being compiled to an intermediate programming language called byte-code that engines on different operating systems (called virtual machines) can execute. A developer simply writes his program and distributes the byte-code. Virtual machines will be able to run it anywhere. The virtual machines also often take advantage of their respective operating system’s libraries and offer the developer the possibility to give their application a native “look-and-feel” and better performance.

¹ The different editions refer to “packages” with different sets of functionality and software libraries.
On the client’s side, the Java Runtime Environment (JRE) consists of: the compiler that transforms the byte-code to native code, a set of standard libraries that all JRE’s have standardized and the virtual machine that executes the native code.

Unfortunately, from previous experience, a large compromise has to be made. Java gives developers the chance to develop their own GUI (graphical user interface) components and interfaces, but they risk making them stand out and make the application seem inconsistent unless they are all further customized for every platform the application might run on.

2.2.2 Microsoft’s .NET Framework
The .NET Framework is a software platform that mainly runs on Microsoft Windows and works in a similar way to Java. Source-code is compiled to an intermediate language (known as Common Intermediate Language) which is then interpreted by a virtual machine. Just like Java, the client must have a runtime environment on his computer/device.

The .NET framework is mainly intended to run on Windows. This offers advantages in terms of user interface consistency and performance, but might not be what a number of developers are looking for when looking for cross-platform software. There are other implementations available on Linux and Mac OS thanks to Novell’s Mono project\(^2\), but they have varying levels of completeness and there can occasionally be compatibility problems.

2.2.3 Adobe Flash
Adobe Flash is a multimedia platform that also allows developers to write “rich internet applications” (RIAs). It was originally built to add animation, video and interactivity to web pages, but due to its extensive functionality, it is now also used to write more complex software.

Flash can both embed itself in web pages and run directly from the web browser or Adobe’s Flash Player. It works by manipulating bitmaps and vectors to animate drawings, still images and text. It can stream audio and video and take input from the mouse, keyboard, microphone and camera.

Flash content can be displayed on a large number of operating systems and devices and it is estimated that approximately 95% of desktop PCs have it installed (Stat Owl, 2011).

Because of its extensive set of features, Flash has enhanced considerably the functionality of web sites. It was not originally built to develop applications though and performance issues arise as soon as it has to perform computationally intensive tasks. It also lacks a number of libraries Java and .NET have to interact with the operating system. Finally the “feature gap” between standard web sites and “Flash-enhanced sites” is quickly closing; new features

\(^2\) Employees working on the Mono project have been let go from Novell’s owner company, Attachmate, and the project has been closed down (Kerner, 2011). Xamarin, a company focused on Mono development, continues as its main developer (Icaza, 2011).
(such as video and animation) are added to the standards (e.g. HTML and CSS) that web sites are developed by.

2.2.4 Adobe Air
Adobe Integrated Runtime is a cross-platform environment that combines Adobe Flash, HTML and AJAX to create Rich Internet Applications (RIA) that have some characteristics of traditional desktop applications.

Air applications can be deployed in two ways: As an in-browser application and onto its runtime environment. Both solutions have advantages and disadvantages.

If developers deploy AIR applications as in-browser apps, they do not require installation, but miss some of the additional functionality desktop AIR apps have. On the other hand, installing RIAs on the users’ computers give applications additional functions like access to the underlying system’s file system, but it requires more work for the users and developers, as strict security measures are in place.

Because Adobe attached the AIR installer to Adobe Reader, a widely-used PDF reader, it already has 100 million installations worldwide (Ludwig, 2009).

2.2.4.1 Conclusions about alternative platforms
Even though these software platforms serve their own purposes and have a large install base, they do not solve the issues presented in in Section 1.1 directly. The .NET Framework is not commonly available on computers that don’t run Windows, Flash has performance issues and Adobe AIR has to deal with tradeoffs at the application deployment stage.

Fortunately for developers, there is a platform that is available on almost all desktop operating systems which offers a very different approach to software development: the web browser.

2.3 Web browser applications
(By Lisan Chen)

A web browser application (more commonly known as a web application) is a program that is accessed through a computer network using a web browser. It usually consists of two parts: a server component that provides and stores data for the user and a client component that is executed in the users’ web browsers as HTML.

Web applications have several advantages over applications written in the aforementioned platforms:

- Because web browsers are practically ubiquitous, web applications are considered to be one of the most expansive software platforms. Every modern desktop and laptop computer has a browser.
Another key advantage is the ability to update and maintain web applications without having to distribute or install any software. The developer only has to update the content once on the server for all clients to get the updated version.

Since all web browsers conform to similar standards, software running on them is essentially platform-agnostic.

Web applications took a relatively long time to gain popularity (Google Trends) and still lag behind in popularity compared to other technologies; however, there are two defining events that popularized web applications; these also became its two most defining elements:

The first event was in 1995, when Netscape introduced JavaScript, a client-side scripting language. It was the first time programming code could be downloaded along with HTML and executed on the user’s computer. This is not only useful for processing or to provide feedback, but it also allowed developers to alter the document the user viewed (called Document Object Model) dynamically.

The second was in 1999, when Microsoft shipped Internet Explorer 5.0 with a function called XMLHttpRequest. It allowed scripts to request information (as XML or plain text) from the server without having to reload the page. Clients could finally interact with the server without the need to reload content in the web browser. This technology would later be called Ajax (Asynchronous JavaScript and XML).

An additional component of the web that aided the adoption of web applications is Cascading Style Sheets (CSS). Thanks to CSS web pages can be styled and laid out without affecting the content. This allows developers to present sites that look and behave like applications.

2.4 Technologies related to web development
(By Lisan Chen & Gabriel Banfalvi)

Part of the project is to research new and upcoming technologies. These are all related to web development, but they are to be evaluated less from a traditional design and digital publishing standpoint and more as tools to produce software. Some of the technologies are individual standards web browsers decide to adhere to; others belong to the HTML5 specification.

The HTML 5 specification is a standard created in response to users’ and developers’ observed needs:

- Text in web pages is placed inside “elements” – special text tags that give text meaning. A number of unused or misused elements are removed (for example, style tags are removed in favor of using CSS to style text) and complex or ambiguous ones are simplified (e.g. tags to change the DOM)
New APIs are added (e.g. support for web application features, support for mobile devices).

2.4.1 WebSocket
(By Gabriel Banfalvi)

WebSocket is a web browser technology available to developers as a JavaScript API that provides developers with bi-directional, full-duplex communication between a browser and a server.

Connecting to a Web Socket server is considered to be very simple and because of its non-blocking event-based behavior, very convenient to developers too. An example is seen in Listing 1.

```javascript
var wsUri = "ws://echo.websocket.org/";
var websocket = new WebSocket(wsUri);
websocket.onopen = function(evt) { onOpen(evt) }; //when connection opens
websocket.onclose = function(evt) { onClose(evt) }; // when conn. closes
websocket.onmessage = function(evt) { onMessage(evt) }; //when a msg. is recvd.
websocket.onerror = function(evt) { onError(evt) }; // when an error occurs
websocket.send("Hello World"); // sending a message
```

Listing 1 – Connecting to a server with a WebSocket.

The WebSocket standard simplifies the issue of bi-directional web communications greatly. Before this, web applications had to struggle with HTTP’s standard behavior. Because HTTP is a request-response protocol, the server can only send information to the client when the client requests it. The server can’t send data to the client “by its own initiative”. There were a number of alternative solutions presented to the developer:

- **Long Polling** – The client could continually poll the server for updates (through Ajax, for example) and the server would only reply when something “reply-worthy” actually occurred. Every single request would require that the connection was reestablished and kept open for long periods of time. This technique often leads to large amounts of wasted bandwidth.
- **Adobe Flash Socket** – Adobe Flash has built-in functionality to establish socket connections. If the developer wishes to use a Flash Socket-based solution, he will have to write a Flash component that offers a similar API on the client’s side and an application to accept connections on the server’s side.

2.4.2 Web Storage / DOM Storage
(By Lisan Chen)

Web Storage is an API that allows developers to store content within the users’ web browsers. If a web browser is disconnected or if the user wishes to work offline, it can still take advantage of the Web Storage to store and retrieve data until its later synchronized again.
Unlike developers’ other choice for local storage, Cookies, Web Storage gives them up to about 5 – 10 megabytes of storage capacity and the client can directly store and retrieve data. Cookies only provide enough storage for a single string of text and clients have no way to alter their content.

There are two types of Web Storage available to developers: Local and session storage.

- Local storage is shared across all pages of a web domain (e.g. google.com) and persists even after the browser is closed. This allows the client to use the data throughout multiple browsing sessions.
- Session storage, on the other hand only exists during the “lifetime” of individual pages and windows. As soon as the user leaves the page or closes the window, the information is lost. This is useful if the user wants to handle different sets of data within several browsing sessions.

Data is accessed through a JavaScript API and uses an associative array. It can also be accessed through “getter” and “setter” methods, as displayed in Listing 2.

```javascript
// Save data to a the current session's store
sessionStorage.setItem("username", "John");

// Access some stored data
alert("username = " + sessionStorage.getItem("username"));
```

Listing 2 - Storing and recalling information from Session Storage

2.4.3 HTML5 Cache manifests

(By Lisan Chen)

Because web applications consist of web pages (with their respective resources) that have to be downloaded through a network, they can become unusable when there is no network connection. A cache manifest file gives developers the ability to create a list of files that the web browser will fetch and store on the computer until the cache manifest is updated. This way, web applications can be useful even if the web browser is not connected to the internet.

The system works by attaching a link to the list of files at the beginning of each HTML file. A line similar to the one in Listing 3 must be added to the html element for the cache manifest file to work.

```html
<!DOCTYPE HTML>
<html manifest="/cache.manifest">
<body>
...
</body>
</html>
```

Listing 3 - Sample HTML file with cache manifest attached.

Cache manifest files always have the string “CACHE MANIFEST” on the first line. After that, files can be listed in three sections:
- **CACHE** – These files will be stored locally.
- **NETWORK** – These files will be excluded from local storage.
- **FALLBACK** – These files will be downloaded and replace anything within the “NETWORK” section that is inaccessible.

Listing 4 is an example Cache Manifest file. It specifies that four files will be stored locally and one can only be downloaded from the network. If it is not available, it will be replaced by a different one.

```
CACHE MANIFEST
CACHE:
/ index.html
/test.css
/test.js
/test.png
NETWORK:
/mESSAGES.html
FALLBACK:
/mESSAGES.html /offline.html
```

**Listing 4 - Sample cache manifest file.**

2.4.4 **HTML5 Canvas element**
(By Gabriel Banfalvi)

The canvas element is a web browser component that allows rendering of 2D shapes and bitmap images.

The Canvas element is presented as a paintable region defined in HTML with a width and height attribute. Once it is created, a “drawing context” is available in JavaScript. The “drawing context” is a JavaScript object with predefined functions. The developer can render any number of 2D shapes and bitmaps on to it. The contents of the canvas can also be exported as bitmaps and reimported if necessary.

All this can be combined with JavaScript events, for example, to render custom user interface components. Because the canvas element is standardized, rendering will be consistent across web browsers and operating systems.

2.4.5 **HTML5 drag and drop**
(By Gabriel Banfalvi)

Drag and Drop provides a new API in HTML5 that allows content to be dragged from the user’s computer and the web browser to each other. Compared to Web Storage or the Canvas element, it is a relatively minor new feature, but one that can dramatically improve the user’s experience.

To make an element within the browser “draggable” all that developers have to do is to give them a “draggable” attribute in HTML, as shown in Listing 5.
“Source elements” (the “draggable” component or file) and “destinations” (the target, where files are dropped) can handle these through a new set of JavaScript events. Event listeners can be used to handle these events as they occur. An example of how to deal with this sort of events is show in Listing 6.

```javascript
function handleDragStart(element) {
    this.style.opacity = '0.4';
    /* element to be dragged, set to be semi-transparent */
}
draggableElement.addEventListener('dragStart', handleDragStart, false);
/* We take the draggable element, wait until a "dragStart" element is fired and call the predefined handleDragStart function */
```

Listing 6 - A function to handle dragging is defined and then attached to an element.

3. Project Criteria and Process
(By Gabriel Banfalvi & Lisan Chen)

The first part of this chapter discusses the criteria that would be used to evaluate the success of the project. The criteria are useful because they can be used to answer the questions presented in Chapter 1. The second part of this chapter describes the project team’s intended development process.

3.1 Project criteria
(By Gabriel Banfalvi)

There is a very large difference between web applications and desktop applications both from the technical side and the user’s perspective. Readers wanting to assess whether to use web applications or not must focus on the aspects that seem most challenging and relevant to development. The three aspects deemed to be most relevant are:

- **Performance** – One of the biggest drawbacks web applications have had until recent years has been slow performance. Web browsers have since early 2008 raced to develop faster and faster JavaScript engines. This has not only improved computational speed drastically, but also opened up possibilities to developers seeking to develop larger and more complex products.

- **Adaptability** – Web Applications have to be developed very differently to regular software (both architecturally and in relation to user experience). If one wishes to “adapt” a desktop application, there are a number of advantages and pitfalls to web application development which have to be taken into consideration if a developer wants to “convert” a desktop application.
Ease of development – Developers wishing to write web applications have to know HTML, JavaScript, CSS and all their respective APIs, libraries and design patterns; they have to make sure their applications deal with every web browsers’ quirks and figure out how to use certain technologies that are only in their infancy.

In comparison, a platform like Java offers a complete solution to programmers. It has established methodologies and standards from its initial development stages all the way to distribution and it has well-documented libraries.

3.2 Development Process
(By Lisan Chen)

This chapter describes in depth how it was decided that the development process would proceed. The first section describes the chosen development methodology and the second describes the team’s technical choices.

3.2.1 Methodology
The team started the project with limited understanding of what would be needed to complete the entire project, so a requirements analysis was carried out. The following summary of the functional requirements was made:

The application is a program that allows users to share a variety or resources over the internet. The application uses user accounts to store profile information, personal files and contact relationships between users.

The collaboration application will allow the user to create projects and invite other users, or alternatively to accept project invitations from other users and join their projects.

Through the default view, the user can view his contact list, add contacts or remove them and send messages to them. This view leads to a contact list.

When a user creates a project or opens an existing one, he will have the option to invite other users (if he is the creator), upload files, send messages to the other members, create, edit and remove project tasks, chat in real time and draw on an electronic whiteboard.

The whiteboard will run in real time, for everyone that has the project view open. The users may paint and draw shapes on it, and others will see the changes as they happen. Because it requires that several users draw concurrently, there is no way to perform undo operations.

Following that, a research phase was begun. In it, the most common web languages were studied. The team’s members had some previous experience with web design, but it was never really studied thoroughly and focused on the presentational aspect. A large amount of online documentation was downloaded and a textbook (Stobart & Parsons, 2008) was purchased to learn the basics of PHP, JavaScript and MySQL. KTH provided the project team a virtual server with root access. There, testing could be performed and the previously learned material could be experimented with and practiced on.
After the project team’s level of confidence in programming grew, research was started to learn about software platforms that provide the type of solutions the team can base its work on. A number of meetings were held to discuss how development should proceed.

3.2.1.1 Software platforms
The platforms the team started researching ranged from the operating system the server would run on to specific server modules that would provide specialized services. The following table lists the software the team settled for.

| Operating System               | Ubuntu 10.10 Server – Chosen for familiarity. It is easy to manage and has both a good package manager and up-to-date packages. The environment lets developers prototype software very quickly. |
| Web Server                    | Apache 2 – Chosen for being the most popular Web Server on the Linux operating system. Offers great amounts of documentation. |
| Database Management System (DBMS) | MySQL – Chosen for being the most popular free DBMS on Linux. It has extensive documentation and support. It is the most likely to have software packages to interface with other software solutions. |
| Web Socket Server             | jWebSocket Server – Chosen for the team’s greater experience with Java. |

Table 1 - Development Environment

The runtime for the scripting language PHP was installed to write the server-side software that would also interact with the DBMS. PhpMyAdmin, an open source database manager, was also installed.

3.2.1.2 Process Planning
The development team decided that the implementation stage should follow a Feature Driven Development (FDD) process. This would be ideal, because the process is both iterative and incremental. There are aspects of FDD that cannot be used due to the small number of members (e.g. code ownership, dealing with all the stages individually), but it was considered that following FDD would help the team maintain good practices. Figure 2 displays a simplified model of the development process.

The following would be the team’s interpretation of FDD:

- **Develop overall model & build feature list** – The overall model would be developed as a list of features, database schemas, interface mockups and other process diagrams. A list of interdependencies was made to sort the features by priority.
- **Plan by feature** – The interface mockups would be detailed further, a mock-usability study would be performed through hallway testing (people would be brought in and would test mock-ups and discuss the component that was being developed) and meetings would be held about relevant aspects of the implementation. These features would then be divided into smaller, more manageable tasks (resembling user stories). This way, team members would be able to split up work according their speed.
• **Design by feature & build by feature** – The team member in charge of a specific task would design it then implement the feature. Programming would happen individually, but if there are any stages where either member can get stuck, the team can turn to pair programming instead and solve the problem together.

![Figure 2 - Simplified diagram of iterative development process](image)

Using a source code revision system was also discussed. It was decided that because both members had to work on a small number of common files and because they have to be repeatedly uploaded to the server for testing, the best way to avoid inconsistencies is to not use any form of revision system. Another reason not to use a revision system was that the development team would not benefit from any of the features they offer (e.g. file locking, atomic operations and version merging) due to the project’s pace.

Source code and resources were to be uploaded through an FTP service and backed up whenever the team believed it crossed a development milestone.

It was decided that because the development team consisted of only two people, instead of scheduling meetings according to standard agile software development methodologies, they would be held on the spot for greater efficiency.

4. **Project Design**  
(By Gabriel Banfalvi & Lisan Chen)

Before the product implementation begun, a number of designs describing how the software would be developed were made. These covered the product’s architecture, its database, how the program would handle its users and its user interface. Only after the initial decisions were made, would development team proceed with the implementation.

4.1 **Software architecture**  
(By Gabriel Banfalvi)

The product’s architecture was without a doubt the hardest part to define from the beginning. The team had never written software of this scale. Designing an application like this without previous experience seemed like an overwhelming task. The first section discusses how the team designed the application with regard to a standard client-server
4.1.1   Extending the client-server model

Applications are usually organized by logical parts called “tiers”. Each tier is assigned a role. Desktop applications usually have a single tier and reside on the client’s computer. Client-server applications on the other hand usually have two or more tiers:

- **The client** – It acts as a presentation tier which interacts with the user
- **The server** – It represents the application tier that manages data and business logic.
- **The database** – Acting as the data tier, storing and retrieving information.

For this project, the application’s tiers would have to be expanded, both in functionality and number.

4.1.1.1   Layering application tiers

When the application was first planned, one of the main objectives of the team was to modularize the code. It was the team’s intention to allow developers to remove and add “modules” (the main features) without threatening the integrity of the rest of the program. Because the application is divided into several tiers, this new modular system would have to extend over both. Figure 3 shows what it would look like conceptually.

![Figure 3 - Client-server model supporting modules](image)

This gave the team the idea to create a basic framework that would:

- Load each module, isolate it and initialize it when the entire program is loaded.
- Handle messaging between the server and the client. It would forward messages appropriately between modules (e.g. the chat module on the client would contact the chat module on the server).

4.1.1.2   Adding an interactive tier

The team realized, as it researched WebSockets, that the client would need to connect through two channels whereas the original Java client only needed one. This was due to two reasons:

- The application needs to be on the client’s computer. Because the Java client is the application, it only needs to download the user’s data and use it to populate the
application with it. In the web application’s case, the application itself needs to be downloaded, and a web server is needed to do that.

- HTTP connections and socket connections are different. Even in software solutions that merge both functions and can act as both an HTTP Server and a WebSocket Server, two connections have to be established.

Even if the application is fully loaded on the client’s computer with a cache manifest (meaning that the application does not have to connect to the web server again), an HTTP connection has to be made occasionally to check if the manifest is updated.

![Figure 4](attachment:standard_client-server_model.png)

**Figure 4 - Standard client-server model with three tiers**

In Figure 4, the original client connected to the server and it, in turn, connects to the database. In Figure 5 though, the client connects to two services and they each have to separately connect to the database.

![Figure 5](attachment:two_connections_client-server_model.png)

**Figure 5 - Client-server model with two connections from the client**

The team came to the conclusion that if two connections were used, the application could take advantage of their different behaviors. It could use the HTTP connection to request static pages like forms and read-only content from the database with Ajax and interact with the server through the WebSocket connection.

Because the WebSocket server cannot be changed while it is running, it has to be restarted whenever source code needs to be changed on it. The biggest advantage is that this way, static content is not bound to the WebSocket server and it can be changed on the fly. Figure 6 shows how the application’s modular architecture can be merged with a four-tier model.
4.2 The application’s database
(By Gabriel Banfalvi)

Since the application data would be stored in a database, it can be stored according to its features and updated through the development process incrementally. Each application feature would have at least one table, and different columns could maintain relationships across tables. Figure 7 displays these relationships and all the fields in detail.

The *users* table would hold a list of user IDs, usernames, e-mails, full names, privilege levels and passwords. The passwords would be stored hashed as insurance if the database is hacked. This way, even if someone recovers the passwords, they would be unreadable.

There would also be a *userToUser* table that links users as contacts and gives their relationship a unique ID. The dependence on the user ID field from *users* allows the entry in *userToUser* to be deleted if an entry in *users* is deleted effectively making a relationship disappear if one of its members is removed. These kinds of dependencies are present throughout the program.

The *projects* table would assign each project an ID, a title, a description, a link to the owner’s ID, a Boolean labeling it as active/inactive and a creation date. An additional table, *projectToUser*, links projects to their members, to make searching for projects a user belongs to easier after login. It also contains a field to assign a color intended to allow users to color-code their projects.

Messaging would be handled by three tables. The table *userMessages* handles messages from one user to the other, *projectMessages* handles messages a user wants to send to all members’ inboxes and *posts* handles messages a user wants to leave on the message board. They all link to relevant user and project IDs and have a title, a body and a posting date. They can also be assigned a priority to allow users to highlight important messages.

Two types of calendars would be available: a personal one (*userCalendar*) and one dedicated to projects (*projectCalendar*). A user can then organize his personal events and share them with colleagues if he wishes to. It also allows users to set deadlines on projects and share important dates. They both have a title, a description, a start date and duration. The user’s
calendar had a field to define it as public if the user wishes to share it and the project calendar has a reference to the poster’s ID.

![Relational database model of project](image)

Figure 7 - Relational database model of project

### 4.3 In-application user interface

(By Lisan Chen)

Web application developers have to take into consideration that the application will run within the web browser. While it provides plenty of advantages to developers, it also forces certain constraints.

#### 4.3.1 Single document interface

The original Java program used several windows to allow users to see several tasks at once. Since the user is running the program within the web browser, he will most likely handle navigation differently.

Though web browsers can have several windows or tabs open at the same time, it is uncommon for users to navigate between them continuously. When users change windows/tabs, they usually change their work context entirely (e.g., switching from their mail client to their news reader) and seldom need to go back and forth between tabs quickly – unless it is to deal with momentary actions (looking something up quickly, for example). It
is also difficult for different browser windows to communicate, effectively making each window its own program.

A Multiple Document Interface would be ideal to deal with several tasks (as features of this program) simultaneously, but in this case, because the program is inside the web browser, a Single Document Interface is chosen. A diagram of the two solutions is displayed in Figure 9.

![Diagram of Multiple and Single Document Interfaces](image)

**Figure 8 - Multiple and single document interfaces**

To make up for the lack of visual responsiveness, the user can be notified of events other ways. If any component needs the user’s attention, a small popup can fade in and out somewhere to the edge of the user’s field of view where it will not interfere with his work. If the user does not wish to deal with it immediately, a small badge with a counter can be placed above the icon relevant to the notification. Figure 10 shows how this would work.

![Diagram of Popup Notifications and Badges](image)

**Figure 9 - Popup notifications and badges**
4.3.2 Program layout

The program is designed to offer a simple and straightforward interface. The first view the user is displayed is the login screen. If the user is not registered, there is a link to a separate page where he can create a new account.

An option to register on the front-page has been considered, but because it would be a one-time event for the user, it can be placed on a separate page, to make the interface cleaner. Figure 11 displays what it would look like.

Once logged in, the main interface to the program is displayed. A consistent way to display all the features/functions had to be found. It was decided that the program would have a toolbar displaying all the application’s options. Its layout can be seen in figure 12.

The toolbar would first display buttons to change between functions on the left. The chosen function would have its title presented in the middle of the toolbar to make the user aware of where he is and the functions content would be beneath the toolbar. On the right-hand side of the toolbar the user’s full name is displayed. This gives the current viewer immediate information of what is logged in. If the user’s name is clicked, two options are displayed:

- **User settings**: This option takes the user to personal settings that allow the user to change his display name, e-mail, and password and provide a profile picture.
- **Log out**: This option allows the user to end his session and log out of the application.
It was decided rest of the user interface elements for each function would be determined during the implementation stage. The project team was uncertain of what challenges it would have to deal with throughout the project and it believed that following FDD would allow it to adjust to the schedule more efficiently to an already ambitious set of goals.

5. Implementation
(By Gabriel Banfalvi & Lisan Chen)

This chapter describes the implementation process of the application.

5.1 First steps into web application development
(By Gabriel Banfalvi)

The implementation stage of this project started as the team learned how to work with PHP and MySQL. At this stage the team worked together to establish a common framework of functions to develop on. The team started by learning about managing session variables and handling database queries. At the same time, it started designing a basic login and registration page following the design in Figure 11. It is displayed in Figure 13. The program is temporarily titled “Nakama” which in Japanese means colleague or friend.

![Welcome to Nakama!](image)

Figure 12 - Login Screen

Once this stage complete, the members proceeded to work on the program’s layout as displayed in Section 4.3. The first version simply loaded different functions as if they were regular pages. When this was complete, the team realized that even if it used session storage (described in Section 2.4.2) and session variables, it would be difficult to keep the user updated of events happening in different parts of the program if they weren’t permanently

---

3 HTTP requests are usually stateless. To keep persistent information about a client’s requests, the server can identify the client with a unique ID and attach session variables to it. The variables can store information about the client as long as he doesn’t finish his browsing session. For example, after a user has logged in, the server can keep his login information attached to his session ID so the client doesn’t have to keep logging in.
loaded. Additionally, transitions where the page would go entirely blank while it loaded the next page made the experience of using the program jarring.

5.2 Implementing the site with jQuery and JavaScript
(By Lisan Chen)

The first step to correct this issue was to change how pages loaded. This is the stage at which the team learned about the relevance of Ajax. As mentioned in Section 2.3, Ajax lets web browsers load content without reloading the entire page. Listing 7 shows how this works.

```javascript
var ajaxRequest = new XMLHttpRequest(); // Request object

/* The request will change states when sending, sent, waiting for another request and complete. The last state is the relevant one. An anonymous function to handle state changes is created. */
ajaxRequest.onreadystatechange = function() {
  if(ajaxRequest.readyState == 4){
    /* The following is the returned information. */
    var response = ajaxRequest.responseText;
  }
}

/* This is where a connection is opened. The arguments are comprised of the HTTP method, the destination of the request and whether it should be asynchronous */
ajaxRequest.open("GET", "getAnswer.php", true);

ajaxRequest.send(); // The request is sent.
```

Listing 7 - How to make an Ajax request

It was decided that the program would load every part of the program at once in the background. The problem was that the JavaScript code on the browser side started to become somewhat complex and minor compatibility issues started arising between web browsers. The team started using jQuery, a small JavaScript library that makes it easier to write code that manage the DOM, events and Ajax requests. It also makes sure all functions are compatible with all browsers. A request comparable to the one in Table 8 is shown in Table 9.

```javascript
$.ajax({
  url: "example.php",
  success: function(msg){
    var response = msg;
  }
});
```

Listing 8 - How to make an Ajax request with jQuery

The second stage was developing the modular system described in Section 4.1.1.1. To do that, an “abstract” JavaScript class was designed with functions to register elements as program modules, display themselves on the screen and parse messages from the server. A

JavaScript doesn’t have any built-in features to make abstract classes. A regular class was designed with predefined functions that would do nothing. Other objects inheriting from this base class would have to re-implement these features.
sample UML diagram with showing how this inheritance works with two of the classes is in Figure 14.

Each module has a unique name. When it wishes to send a message to the server, the send function in the application prefaces the messages with the module name. When the application receives a message, it is also prefaced with the module name. This allows application to route messages from client module to server module efficiently.

![UML Diagram]

**Figure 13 - Two of the modules inheriting functions from their superclass**

Source-code for each module can be written in separate JavaScript files to be interpreted independently. If any of the files contains an error, the rest of the program can keep on working.

Functions were also created to load the static content mentioned in Section 4.1.1.2. When called, the function places a div element over the entire screen to block input, as well as to fade out distracting elements of the interface. It then loads the contents of the popup through Ajax. It also places a small “x” in the top right corner to allow users to close the popup at any time. An image with the site’s registration form is displayed in Figure 15.

![User Registration Form]

**Figure 14 – Screenshot of popup for user registration**
5.3 Struggles with WebSockets
(By Gabriel Banfalvi)

Once the main application’s features were done and implemented, it was time to set up the WebSocket server. Unfortunately, the team wasn’t as successful as expected.

5.3.1 JWebSocket
Our first attempt was with JWebSocket, a solution implemented in Java. The team chose this software package to start with because both members had familiarity with Java and thought it would have a gentler learning curve.

The first issue that the team faced was during the program’s initialization. Despite having both the Java SDK and JRE installed and having configured the file paths according to instructions, the program wouldn’t start and would return numerous errors about files not being found.

Once that issue was dealt with, the second problem presented itself with networking. The server came with a pre-installed plugin that would simply echo messages from the client, but it wouldn’t allow connections to pass through. Having spent several days trying to get simply that plugin on the server to work, the team decided that it would instead be better to simply find another WebSocket server.

The application also consumed a large amount of memory and processing power. The manager of the virtual machines even requested the team to upgrade the system and reboot it because it seemed to take up a large amount of resources.

5.3.2 PHP-WebSocket
PHP-WebSocket is a server implementation designed in PHP, another language the development team is familiar with. Despite its apparent simplicity, there were issues making it work and getting it to connect with the web browser. After performing an extensive number of tests to check if the issue lied with the server, the team moved on to a software platform of a different nature.

5.3.3 Node.js and Socket.io
One of the most popular server-side development platforms to emerge in the last few years is Node.js. Programs that run on it are written in JavaScript and it runs on Google’s V8 JavaScript engine. Node.js can be used to build any kind of network program.

A module that can run atop Node.js is Socket.io. It emulates a socket connection between a server and a web browser though a number of means: WebSocket, Flash socket, and Long polling (as described in Section 2.4.1). It is comprised of two components, a module to be installed alongside Node.js and a set of scripts that have to be attached to the client.

The team decided to install Node.js and Socket.io following the instructions available online (Lung, 2011) and attempt to run a WebSocket server with it. This turned out to be successful
and the team proceeded to download another module to communicate with the database. Once the system described in Section 4.1.1.2 was complete, the development team decided to work on implementing all the features.

5.4 Implementing all the features
Each feature was broken down into a set of user stories which were subsequently divided into tasks to be worked on either on the client or the server.

5.4.1 Contacts
The first function the team started with was the contact list as this was deemed to be the simplest and it would be needed by other parts of the program. It was decided that contacts would be sorted into lists depending on whether the user added them personally or if they are project acquaintances. The personal contacts are also sorted as being online or offline.

![Figure 15 - A computer sketch of the contacts module](image)

To implement the online/offline functionality, users were placed into an “online” list on the server as they logged in. At the same time, the client would request the user’s contacts and at that moment the server would use the list to filter out the contacts that are online and notify them of the user’s new status. The server filters through the user’s online contacts when the user logs off to notify them too. Project contacts would be merged into this module from the project member lists later when the projects module is implemented.

Another part of contact lists was having user profile images. This was an idea that proved itself harder to implement than expected. Form data can be submitted through Ajax, but files such as profile images can’t. A traditional HTTP upload would be necessary and that would force the whole application to reload. To circumvent this problem, a hidden iframe would be the upload target and refresh invisibly inside the upload form. What this means is that the form that uploads images, sends the image to the server and asks the results to be rendered inside a hidden page. Figure 17 shows the final design of the contacts module.

Contacts are added through an “Add contact”-button that displays a popup. The server is then queried for users with names that have the letters that have been searched for. The

---

5 “Iframe” stands for “inline frame”. A web page embedded inside another page.
client application then lists the search results inside the popup and allows users to contact the people on the list or to add them directly. Figure 18 shows the final design for the popup.

---

5.4.2 Whiteboard

Implementing the whiteboard presented its own challenges. The first was learning how to use the canvas element. Though its API isn’t very different to other drawing libraries, writing the application in JavaScript and dealing with the standard behavior HTML elements was arduous.

One of the issues was that the team wanted the canvas to resize and adapt to the user’s different window sizes. Unfortunately, resizing elements in the web browser also redraws them. In the case of the canvas, this means that the web browser erases it. As a compromise, the development team decided to give it fixed dimensions. It started by finding what the most common sizes are (Blume & Bowden, 2009) and then determining that a size of 950*520 pixels would satisfy 90% of people browsing the web. A layout of the application is visible in Figure 19.

---

The second issue was displaying some form of feedback. The mouse doesn’t react when hovering over the canvas and when the mouse is dragged it can try to start selecting (nonexistent) text. To deal with it, the default text selection event was overridden and a second canvas element was placed over the first. This second layer was used to display a
cursor at the mouse’s location showing what area would be painted. A model of how it works is shown in Figure 20.

Another problem that the team had to deal with designing the interface was creating all the components that would control a drawing tool’s options – for example a pen tool would have a stroke width and color, therefore it would need a slider for the width and a color-picker to choose colors. Because these could be later re-used, a unique class for all these components was created. This class used the MVC design pattern to generate predesigned views (the components). The feedback from the views would then be returned to the class, acting as a controller, which would update the model in memory.

The outcome of this work was a well-designed platform for the whiteboard that could be used to develop the whiteboard further if there was time. The final design for the whiteboard is shown in Figure 21.

![Figure 20 – Screenshot of whiteboard design](image)

5.4.3 Messages

The messages module was also designed with consideration to the possibility of merging in some features from the projects module. The interface would consist of: a toolbar with options like writing new messages, a sidebar that displays different message lists and a message viewing pane that sorts and shows the actual messages. A preliminary design with several message viewing panes is visible in Figure 22.

There are different display modes depending on what view is chosen:

- The conversation view can sort messages according to users’ conversations. A conversation is created when a user selects the option to write a new message. Successive replies to that message will remain within that conversation.
- The inbox and outbox view have a more traditional view of messages. A list contains all the sent/received messages and a pane beneath displays the message content.
To implement the functionality used in the conversation view. What the server does is assign a conversation ID to each new message. When the message is created through the “New message”-button, the ID is completely new, and when the message is a reply, it takes the conversation ID from the previous message. This way, messages and their replies share the same conversation ID in the database and are easy to fetch for the server. Figure 23 displays a screenshot of the final implementation of the messaging module.

An additional feature was added to the contacts module that allows developers to handle a “user search”-component. It is a text field that starts searching through the user’s contacts as soon as he starts typing and then allows the user to select a contact. This allows the user to choose one of his contacts as the recipient of a message. An example of the component in use is displayed in the new message popup.

#### 5.4.4 Chat

The chat module was designed with the purpose of allowing users to communicate instantaneously and messages aren’t stored on the server. All the server does is to broadcast the messages from the sending user back to all the members of a conversation. The largest struggle with the chat module was to arrange the HTML elements according to the designs. The model and the final design of the chat module are displayed as Figure 25 and 26.
5.5 Remainder of features
Time ran out before the chat function was implemented. The project team asked the project supervisor for extra time and a two month extension was granted. During the remaining time the chat function was implemented, the rest of the modules were refined and the project module was begun.

6. Results
(By Lisan Chen)

The team only managed to complete a large part of the work it set out to do, but not all of it. Creating a framework to run the application on, setting up and managing a WebSocket server and learning to manipulate the DOM took more time out of the project than what the team expected. The parts of the program that were completed present future developers with a stable platform that can be used for a myriad of uses.

6.1 Evaluation of the project results
Section 3.1 presented a list of criteria that the development team would evaluate the process by. The results of that evaluation are the following:

Performance – Performance was deemed to be satisfying. The only moments the application seemed to slow down at were when the application loaded a large number of resources and when fading animations for popups covered the screen. The former can be resolved by using cache manifests and the latter can be dealt with by using these animations selectively.

The team believes that complex applications can be created inside the web browser. On the server’s side, both Node.js and Apache performed flawlessly. Extrapolating from the practically non-existent memory and processor requirements the servers had during testing it seems like developers can expand their programs to serve large numbers of users.

Adaptability – The biggest issue during development was redesigning the application to work successfully inside the web browser. Unfortunately, not all functions could be used and time only allowed the development team to use WebSockets and the canvas element, but
they have shown a great amount of potential. This experience shows that there are a large number of applications that can be recreated as web applications.

**Ease of development** – This criterion is unfortunately not successfully met. Though most of the difficulties the team faced can be blamed on lack of experience developing in this environment, they were not the only reasons development was difficult. The following are some of the reasons:

- **Debugging**: Though PHP does provide stack traces and error reporting on regular pages, Ajax requests occur in the background. If a script fails, the developer has to scan error logs to find where and at what time the error took place and check to see if that could be the reason the application became unresponsive. At the same time, JavaScript’s flexible nature allows it to ignore null pointers and other errors and continue working until other functions much later return an error. This also makes debugging JavaScript also very difficult.

- **Variety of languages and styles**: PHP is written imperatively, JavaScript is event-based, MySQL is a query language, HTML is a markup language and CSS is a stylesheet language – all merged to develop one application. In comparison all the former development team had to know to develop FISTY was Java and MySQL. It makes the learning curve steeper and the ability to circumvent steps prone to failure more difficult.

- **Unstable platforms**: A number of software solutions used in this project were relatively young and some of them performed inconsistently.

7. **Discussion**  
(by Gabriel Banfalvi & Lisan Chen)

This chapter first explores the some of the things that went wrong during the project. It then discusses the changes that could have been made to avert these mistakes. The final part talks about the overall conclusions the team reached in relation to web applications.

7.1 **Mistakes and setbacks during development**

There were few errors during development, mostly related to planning and the project approach. There were several minor hacking attempts, but they were more of a time-consuming nuisance than anything that would seriously affect the project.

The project was overly ambitious from the start. Though the current development team had twice as much time for development (the new team had seven months to work on half time versus the old team, which had two months at full time), the former team had five times as many members (ten versus two people). The breadth of knowledge that was required for this
The project was also surprising in hindsight. A much better schedule would have been needed with stricter deadlines and proper time estimations.

Though the implementation process did realistically emulate the development process for a web application and did successfully fulfill its purpose of researching how viable their implementation is, the team’s choice of aiming to “deliver a product” as opposed to simply “do research” into new technologies limited how much new “research material” they were exposed to.

After the team set up the web server a flood of unusual requests started coming in. Hackers were attempting to perform port scans with software like DFind (DFind Port Scanner) to find vulnerabilities in any software running with a listening port. In response, their IP address was blocked and the system was searched for any signs of entry. All software was updated to the latest version for security.

7.2 What could have been done differently
(By Gabriel Banfalvi)

There are a small number of choices which would have been approached differently these are related to the development process and the choice of software.

Apache and PHP would not have been chosen to serve static files and instead, Node.js would have been used. This way the HTTP server and the WebSocket servers could have shared data (e.g. user session information) and a large amount of redundancy would have been avoided.

Instead of choosing MySQL, a document-oriented database like MongoDB could have been chosen. It uses a JavaScript interface too, making interfacing with it even easier.

The decision to not use version control was misguided, a solution like Git which has strong support for code branching and distributed development would have made the development process considerably faster.

7.3 On the viability of web applications
(By Gabriel Banfalvi)

Three questions were first presented in the introduction to this report. They embody the entire purpose of this report. The following are the conclusions the development team has reached after evaluating the development stage.

Can a web application replace a desktop application?

It can, with certain limitations. A web application can have the interface, the performance and most of the functions a desktop application has. It is still missing certain features, but
they will likely be introduced in the near future. They are discussed further in the third question.

A developer planning on making a web application has to insure libraries and APIs are available and are supported by his users’ browsers.

**Can web applications resolve issues other operating system-agnostic platforms cannot?**

Yes, the browser acts as a consistent software platform across operating systems, but with certain tradeoffs. Web applications aren’t yet mature enough and though they *do* solve the issues presented in Section 2.1.1 they also present new issues:

- Web browsers don’t render all content consistently, this means that libraries have to be used that can bridge the gap between different web browsers. CSS support is different depending on what web browser is used, so the application has to be tested on every web browser.
- Not all web browsers support all the technologies. It is important that the program has fallback solutions for instances when users use older web browsers that can’t cope with the latest features.
- Most browsers have fast JavaScript engines that run as fast as native scripting languages. Older browsers are not that efficient. Developers have to stress-test applications on every browser to insure there are no bottlenecks.

**What limitations do web applications face today?**

Web applications still have a long way to go before they can replace a large number of desktop applications. The following are a few of the areas that should be developed further or are considered to be “work in progress”.

- **User interface components** – There are a number of form elements that can be re-used as components, but there is no support for the creation of custom elements. Some components aren’t even supported across all web browsers yet.
- **Local file access** – There is no established platform today to allow the web browser to read and write to the user’s file system. The World Wide Web Consortium has a proposed API (Uhrhane & Google, 2011) but it’s not supported yet by most web browsers (Deveria).
- **Streaming media upload support** – Support for streaming camera images and sound has existed *to* the web browser, but not *from*. This feature would allow users to stream their own content which can be used for a myriad of uses, from content creation and online video and audio editing, to new forms of voice chat and teleconferencing. There are promising experiments (Web Real-time Communication, 2011) that the team hopes will be explored further in the future.
8. Conclusion
(By Lisan Chen)

This thesis intended to research how viable web applications are as a replacement to desktop applications. To do this an attempt was made to convert a desktop application into a web application and research new potential technologies. Though performance is satisfactory and most programs can be rewritten for the web browser, developers must be research the software platforms they are going to use and gain significant experience.

9. References


Appendix A – Term Glossary

API
API stands for Application programming interface. It is a set of specifications that programs can use to communicate with each other.

Bi-directional
A communication channel that can move in both directions at once.

Byte-code
Programming code that can be interpreted by different machines.

Client
Software that accesses a service on another computer.

Client-server model
The client server model is a relationship between systems where one or several servers provide resources to multiple clients.

Compiler
A compiler is a program that transforms programming code into

Cross-platform software
Cross-platform (or multi-platform) software refers to software that is built with the intention of running on several platforms.

Div element
An HTML element that can be placed inside a web page.

Full-duplex
A communication channel that allows communication in both directions at the same time.

Multi-platform software
See cross-platform software.

MVC
Model-view-controller

Operating System
An operating system is software consisting of several programs and data that manages computer hardware resources and allows other applications to run.

Platform
A platform is a combination of software (and sometimes hardware) that software applications can run on. It could be an operating system, a combination of operating system and hardware architecture, or simply a web browser.

Request-response protocol
A communications protocol where one computer makes a request and a second replies to that request.

Runtime Environment
Software designed to allow programs to run on it.

Server
A computer that serves out resources.

Socket
An endpoint for a computer connection.

Software library
A collection of resources that can be used to develop software.
| Virtual Machine | A software implementation of a computer that executes programs as if it was a real one. |