Institutionen för datavetenskap
Department of Computer and Information Science

Final thesis

Meteor framework, a new approach to web development: an experimental analysis

by

Sahand Sadjadee

LIU-IDA/LITH-EX-A--13/067—SE

2013-12-18
Final Thesis

Meteor framework, a new approach to web development: an experimental analysis

by

Sahand Sadjadee

LIU-IDA/LITH-EX-A--13/067—SE

2013-12-18

Supervisor: Erik Berglund
Examiner: Anders Fröberg
Abstract

The traditional definition of a dynamic web application is about a collection of programs executed at server-side to provide content for clients. These types of web applications produce content at server-side and deliver it to their clients via multiple pages. In result, the client-side has the responsibility to render the content and perform a limited amount of calculations to increase the performance and user experience.

Meteor is a web framework designed for developing Single Page Applications and compared with traditional web frameworks, it takes a new approach in which most of the computations are done at the client-side. This leads to having the server-side primarily used for data storage and secondarily performing a limited amount of computations based on the Model View View-Model pattern.

This thesis tries to examine how web development is affected by Meteor framework from different angles by performing an experimental analysis on Meteor framework. It will investigate different attributes of Meteor framework used for developing a real-world application and finally concludes by presenting the advantages and disadvantages of using it.
## Contents

1. Introduction ........................................................................................................... 5  
2. SkillBind: A social network for exchanging skills ........................................... 7  
   2.1 User authentication ...................................................................................... 7  
   2.2 Top menu and tab switching ..................................................................... 7  
   2.3 Search tab .................................................................................................. 7  
   2.4 Browser tab ............................................................................................... 8  
   2.5 Contacts tab ............................................................................................. 8  
   2.6 Profile tab ................................................................................................. 8  
   2.7 Activities tab ............................................................................................ 9  
   2.8 Message box tab ........................................................................................ 9  
   2.9 Notifications tab ....................................................................................... 9  
3. Software development and maintenance .............................................................. 11  
   3.1 Code structure ............................................................................................ 11  
   3.2 Third party libraries .................................................................................. 12  
   3.3 GUI ............................................................................................................. 13  
   3.4 Account management ................................................................................ 15  
   3.5 Session handling ...................................................................................... 16  
   3.6 Monolingual software development .......................................................... 17  
   3.7 Development environment ....................................................................... 18  
4. Data handling ...................................................................................................... 19  
   4.1 Data storage ............................................................................................... 19  
   4.2 Data access using RPC .............................................................................. 20  
   4.3 Data access using synchronized collections ............................................ 21  
5. Deployment ......................................................................................................... 23  
   Conclusion ....................................................................................................... 25  
   References ....................................................................................................... 27  
   Appendixes ...................................................................................................... 29
Chapter 1

Introduction

A dynamic web application is traditionally about a collection of programs running on one or multiple servers producing content for their clients around the world. In this form of web applications, most of computations are done at server-side and clients are primarily responsible for rendering the received content and secondarily performing a limited amount of computations. The main reason for having client-side computations for this type of web is to reduce the work load at the server-side and the data flow between the server-side and the client-side as much as possible in order to increase performance and improve user experience. For example, verifying the input data format inside of a registration form, could lead to less computation at the server-side and more responsiveness from the web application specially in case of having incorrect information being entered into the fields.

Unlike the traditional form of web frameworks, Meteor[1] focuses on developing a new type of web applications based on Model View Model-View pattern[2] in which most of computations are done at the client-side and at the same time the server-side is primarily responsible for data storage and delivery and secondarily is used for performing a set of computations. This thesis studies the advantages and disadvantages of using Meteor framework for developing a real world web application by doing a subjective comparison between Meteor and Google Web Toolkit[3] as a Java[4] based client-side framework combined with Java Server Pages[5]/Servlet[6] technologies at server-side.

As GWT framework replaces Javascript[7] with Java at the client-side and having Java at the server-side, it leads to a monolingual Java based development environment. At the same time Meteor framework uses Javascript at the server-side and having Javascript at the client-side, it provides a monolingual Javascript based development environment. In fact in this setting, both frameworks share one common attribute which is providing a monolingual development environment with the aim to reduce both the development and maintenance costs.

To perform a proper subjective comparison, this thesis relies on Skillbind[8] application based on Meteor framework and Javandad[9] application based on GWT framework both developed by the author. This thesis also relies on long experience of the author in web development area. The reason for performing a subjective comparison in this thesis is to examine different attributes and behaviors of Meteor as a web application development framework and investigate how they are different compared with currently available frameworks. The subjects are chosen based on the author’s experience in web development area and also the development experience of the SkillBind project as a real-world web application.

The first section describes SkillBind application mainly from the users’ point of view by examining different functionalities provided by it. It also talks a little bit about the implementation details where required. The second section describes the technical details of using Meteor framework for implementing the mentioned functionalities. It also describes in what way Meteor is different compared with GWT framework combined with JSP/Servelt technologies. Finally, it concludes the advantages and disadvantages of using Meteor in context and concludes if Meteor has been a right choice for developing SkillBind project or not.
Chapter 2

SkillBind: A social network for exchanging skills

SkillBind is a social network via which users have the possibility to exchange skills with each other. It allows them to declare to others what they can teach and what they are willing to learn. The reason for choosing this subject was mainly about it having common functionalities with many other real world applications as today. Real-time messaging system, search engine, account management, custom data storage and presentation to name a few. Another reason was not being able to find any website with the same functionality and perspective at the time which made this topic more interesting.

Figure 1 shows the Sign up and Log in form located at the welcome page.

2.1 User Authentication

Each user is identified by its own email address which must be unique in the system and unchangeable for each created account. After doing the account registration, the user is required to verify the provided email address. The whole verification process is performed based on a generated key assigned to the just registered email account. As registration is completed, a link containing the key is sent to the registered email address. This key is valid for 7 days and after that it expires and signup process must be redone. The user has the possibility to change or recover its password. In password recovery process, there is also a generated key involved. Figure 2 shows a verification email sent to the user’s registered email address right after submitting the account registration form.

2.2 Top Menu and Tab Switching

The application content is presented inside of seven different panels which are navigable through a set of top located menu items. The top menu bar is permanent and always viewable by the user as it stays on top of the page even in case of scrolling down or up. The top menu is implemented using Bootstrap[10] as a template and is styled using Cascade Styling Sheets[11].

Each tab is structured inside of the client folder as a package, a folder, including a template and a Javascript source file. Templates are used to separate the GUI from business logic[12] which in result leads to less coupling between software components and then decreases the development and maintenance cost

2.3 Search Tab

The search tab provides a search engine which enables the logged in user to search for other users based on the five following criteria.

- What the user is willing to learn
- what the user can teach in return
- what city and country the user lives in
- what languages the user speaks with
- what gender the user has
The result is ranked based on the grades of each person in the result set. The default value for each criterion is loaded from the profile. Users can change them temporarily but for a permanent change, they need to go to the profile section. Because all the users’ records are only available at server-side, the search engine works based on calling remote procedures [13] to deliver the searching functionality and in result, the search result is not live. If the displayed information of the found match is changed by the owner, the shown information in the search result will not change unless the logged in user does the search again.

Figure 3 shows the search engine with one found match in the search result.

2.4 Browser Tab

The logged in user can view other users’ profile inside of the browser tab. There are two ways to access other users’ profile page:

1. Using the email address bar located inside of the Browser tab.
2. Using the search engine.

Via the Browser tab, the logged in user can send messages to and start activity with the viewed user. It is also possible to save the viewed user as a contact and refer back to it later.

The browser tab fetches the required information from the server and waits until the retrieval process is complete. After that the retrieved information is shown inside of generated components. Depending on what pieces of information is available, Graphical User Interface components are generated and in result no empty component will be displayed to the user. As tested in low bandwidth connections, the process may take too long, deceiving the user that no information is available to be displayed. In this scenario, an explicit progress bar is required as the progress bar of the browser does not work in case of RPC calls.

Figure 4 shows the browser tab displaying some other user’s information.

2.5 Contacts Tab

Via the Contacts tab, the logged in user can view the added contacts and filter them based different criteria. Only users’ emails are stored as contacts and the rest of information for every contact is retrieved from the server via making RPC calls. In result the shown information is not reactive. The logged in user has the option to remove contacts from the contact list.

The reason for not storing the whole contact information is because of the condition in which the information is changed by the owner but not in the contact list of every other user having the owner as a contact. This results in inconsistency and is avoided by this approach.

Figure 5 shows a contact list containing three contacts.

2.6 Profile Tab

In the profile tab, users can add or change their personal information which is viewable by other users. The information is organized into three categories “Profile”, “Knowledge” and “Work Experience” and is stored using MongoDB Collections[14] at server-side.

Users can select a profile image from their hard disk. As the user selects an image along with other information, they are stored temporarily at client-side. As the user clicks on the save button, they are written to a synchronized collection at client-side which in result will be uploaded to the server-side by Meteor.

Bootstrap modal dialogs have been used to collect the user’s information about teaching and learning subjects, education and work experience.

Figure 6 shows how users can add or change their general information via the Profile tab.
2.7 Activities Tab

The logged in user can view, finish and evaluate activities started with other users. The whole information about activities is stored and accessed using synchronized collections. This causes any changes can be viewed instantly by the other party.

If two users agree to exchange skills, they need to start an activity. Having an activity has the following advantages:

1. Other users can see what a user is doing at the moment.
2. Other users can see what a user has done so far.
3. The user participating in the activity can be evaluated by the other party, if positive can higher the user’s position in search results done by other users.

Figure 7 shows the Activities tab containing one activity.

2.8 Message Box Tab

In the message box tab, users can view their received messages and reply to each message in the form of a conversation. As conversations are stored using synchronized collections, each conversation is reactive therefore users do not need to do any page refresh to see the new received messages.

The conversations are ranked based on the date and time of the latest message of each conversation. They are displayed as ranked in a panel located at left side of the conversation panel. The user can see the number of new messages as a number located at the right side of “MessageBox” menu item in the top menu. As the user clicks on the “MessageBox” menu item, this number disappears meaning that the system assumes that the user has viewed them all. The same approach is applied in many of today social networking and communication platforms like Facebook[15].

Figure 8 shows the MessageBox tab containing six conversations.

2.9 Notifications Tab

In the notifications tab, users can observe their received and sent activity requests through two separate panels. As requests are stored using synchronized collections, they operate reactively and no page refresh is required to see the new ones. Users can accept or deny the received ones or cancel the sent ones before the other party accepts them. Like the message box, the user can view the number of newly received requests without viewing the notifications panel itself. Once the user accepts a request, an activity is added to his/her records instantly and viewable in Activities panel without any page refresh.

Figure 9 shows the Notifications tab containing the received and request tabs inside.
Chapter 3

Software development and maintenance

3.1 Code Structure

Every Meteor based web application is divided into two client-side and server-side tiers which deliver the specified requirements by working together. Table 3-1 shows the amount of work done at client-side and server-side in line of code and hour metrics in the case of the Skillbind project.

<table>
<thead>
<tr>
<th></th>
<th>Client-side</th>
<th>Server-side</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hours of coding</td>
<td>400(84.5%)</td>
<td>73(15.5%)</td>
<td>473</td>
</tr>
<tr>
<td>Lines of code</td>
<td>4132 (87%)</td>
<td>635 (13%)</td>
<td>4767</td>
</tr>
</tbody>
</table>

As shown in table 3-1, most of work has been done at client-side which has a direct relationship with the application being fat at client-side. Table 3-2 shows the work distribution in Javandad project which has been developed by using Java language, JSP/Servlets and GWT, at server-side and client-side.

<table>
<thead>
<tr>
<th></th>
<th>Client-side</th>
<th>Server-side</th>
<th>Shared</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lines of code</td>
<td>3641(62%)</td>
<td>1277(22%)</td>
<td>9966(16%)</td>
<td>5884</td>
</tr>
</tbody>
</table>

As shown in table 3-2, the code distribution differs significantly compared with the Skillbind project. The reason is that all main computations in the Javandad application are done at the server-side which leads to having a larger amount of code at the server-side compared with the Skillbind application.

Another interesting figure shown in table 3-2 is the total number of lines of code written for implementing the Javandad application. Compared with the Skillbind application, the Javandad application is delivered with smaller amount of functionalities but it has taken more effort in terms of lines of code written. This can verify that, implementing every requirement using Meteor framework needs fewer lines of code compared with GWT framework as experienced in both projects. Figures in table 3-2, do not include the lines of code written in any configuration file. If added the difference is more significant.

As estimated, the Javandad application has taken more than 900 hours to be completed. This is because of more lines of code to be written, more development complexities and larger amount of unknown errors encountered during development process. For example, combining JSP and GWT modules caused an unknown error during development process which took significant amount of hours to be fixed.

Every Meteor based application consists of HTML, Javascript and CSS source files which are integrated by the framework at execution time. One of the issues which may rise during development time is software complexity due to large amount of code at client-side[16]. To address this issue, as the software is modularized
as much as possible, the complexity decreases and leads to less development and maintenance costs [17]. As Meteor integrates all separate source files, the program can be structured by developer anyhow. Like GWT, the program can be modularized by using folders separate folders for different modules. Each folder should contain one HTML file and one or more Javascript source files which are not coupled to other modules.

As applied in SkillBind project, every tab is implemented as an independent module. The TopMenu module is composed of all other modules representing different pages. Each HTML file only includes one template which has a graphical representation in the GUI and is manipulated by Javascript modules during execution. This approach gives less coupling between different software components which can lead to less software development and maintenance costs.

Unlike GWT framework, Meteor follows only one global name space for HTML elements at client-side which requires the developers to use an identifier for only one HTML element in the whole project. To overcome this problem, developers can use long names which start with the containing module name. This will as well increase the code readability during the maintenance process. For example, the id “profileGeneralGender” was chosen for the field “gender” located at the general tab inside of the profile tab pane. At the same time using long identifiers consisting of multiple individual words can become quite annoying.

Using Meteor framework, Unlike the client-side, the server-side includes smaller amount of code which is primarily about data storage causing less code complexity compared with the client-side. As implemented in SkillBind project, the whole server-side implementation resides in only one Javascript source file. Both Meteor and GWT frameworks provide client-side and server-side code separation, as organized in two different folders. As will be discussed in chapter 5, the percentage of procedures which can only be declared at server-side has a direct relationship with the type of the application being developed by using Meteor framework.

### 3.2 Third Party Libraries

Meteor is delivered with a set of built-in libraries called packages designed for different purposes. This gives the developers a complete handy toolbox which helps them implement their applications in a more rapid pace with less need for any other third party library to be added manually. Having well tested built-in libraries with enough documentation and supporting communities, can decrease the development cost with enforcing less reuse costs [18]. As a drawback, if some error appears in any of these libraries a wide range of Meteor based web applications are affected.

<table>
<thead>
<tr>
<th>API name</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accounts-UI</td>
<td>User Management</td>
</tr>
<tr>
<td>AmplifyJS</td>
<td>Local Data Storage</td>
</tr>
<tr>
<td>BackboneJS</td>
<td>Multipage Simulation</td>
</tr>
<tr>
<td>Bootstrap</td>
<td>GUI</td>
</tr>
<tr>
<td>jQuery</td>
<td>Client-Side Scripting</td>
</tr>
</tbody>
</table>

As shown in the table 3-3, AmplifyJS[19], BackboneJS[20], Bootstrap, jQuery[21] and Accounts-UI built-in libraries are used in the SkillBind project and they all operate at client-side. At the same time, it is easy to add custom third party libraries by putting them in the client and public folders to be used consequently at client-side and server-side. In contrast, GWT is only a front-end toolkit framework and is delivered with no other built-in library. For any other functionality at server-side or client-side, 3rd party libraries must be provided separately.
Apart from the backend libraries provided by the application server, the following 3rd party libraries are used in the case of Javandad project at server-side:

<table>
<thead>
<tr>
<th>API Name</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>JDO</td>
<td>Data Storage</td>
</tr>
<tr>
<td>Apache FileUpload</td>
<td>File Uploading</td>
</tr>
</tbody>
</table>

Along with GWT itself, 3rd party libraries can be used at client-side for different functionalities. As a drawback, not every Java library can be used at client-side and it must be specifically designed to be used with GWT framework. This indeed decreases the number of available options when it comes to choosing a third party library. On the other side, it is possible to use Javascript libraries along with GWT at client-side. This can always be considered as plan B when there is no available GWT based third party library for a specific purpose. As a drawback, it increases code complexity and changes the development environment into a bilingual one.

### 3.3 GUI

As Meteor has been designed for developing Single Page Applications, Javascript plays an important role for having a fully functional GUI inside the application. Using built-in libraries available in Meteor framework, it is possible to deliver multiple page web applications as well. The main difference compared with traditional frameworks is that developing multiple page applications in Meteor takes more time and effort. On the other hand, a multiple page application is created at client-side programmatically as it is still a single page from infrastructural point of view.

Despite the fact that SkillBind is a single page web application, it delivers some content inside of pages other than the main page. For example, changing password and account activation are two of the cases. As in Meteor, the data access also happens locally by using synchronized collections and also only raw data excluding GUI elements is downloaded from the server, in both cases of single page and multiple page, the application is responsive and operates smoothly.

Other factors like data exchange overhead, server-side and client-side processing speed affect the overall performance of the application. As an advantage, Meteor can access data locally and respond to the users’ commands instantly. The data synchronization between the client-side and the server-side is done before local reading and after local writing which in fact happens in an idle time period.

In result, Meteor based applications are expected to be more responsive in both forms of single page and multipage compared with traditional web applications. As a sample, Skillbind application is more responsive and operates faster compared with Javandad application as tested in different scenarios.

#### 3.3.1 Single page

A single page Meteor based web application resembles an applet embedded inside of the browser as there is no page refreshing required.

As there is no explicit concept of page in single page applications[22], sharing certain part of content delivered by the application is troublesome. As a main menu is used for navigating through the application, hash marks in the address bar can be used to mark each part of the content. In case of page refreshing or external redirection, these hash marks can be used to show the requested part of application.

Because there is only one main page in SkillBind application, one main top menu based on Bootstrap has the responsibility to show a pane at a time and hide the rest using Javascript and CSS.
Code 3-1. SkillBind top menu implemented using Bootstrap and combination of templates.

```html
<div id="my-tab-content" class="tab-content">
  <div class="tab-pane" id="search" style="margin-top: 43px;">
    {{>Home}}  //handlebar
  </div>
  <div class="tab-pane" id="browser" style="margin-top: 43px;">
    {{>Browser}}
  </div>
  ...
</div>
```

Unlike Meteor, for developing a single page application by using GWT, Only one HTML file should be used and different GWT components are embedded inside of it. One main GWT component used to organize components are panels. As there is only one page, for providing different views components are switched from visible to invisible and vice versa. CSS is also used for styling GWT components as well as in Meteor.

As client-side Java code is entirely separated from the HTML code, for each HTML file there can be unlimited number of Java files and HTML elements only work as place holders for GWT components.

3.3.2 Multiple page

Unlike the traditional server-side frameworks which are designed for multiple page web development, developing a multi page application in Meteor must be done at client-side. As implemented in SkillBind, Using the Backbone built-in library, more pages other than the main page can be generated to deliver different types of content. In the following cases, it is required to create multiple pages at client-side:

1. A fully multiple page application in which content is delivered though different pages.
2. A single page application in which some of the content is delivered though one or multiple pages other than the main page.


```javascript
var Router = Backbone.Router.extend({
  routes: {
    "activation?*querystring": "activation",
    "activationc?*querystring": "activationc",
    "passwordreset?*querystring": "passwordreset"
  },
  activation: function (querystring) {
    // Code
  },
  activationc: function (querystring) {
    // Code
  },
  passwordreset: function (querystring) {
    // Code
  }
});
```

In case one, the result application will resemble like traditional web applications with the difference that the application will more likely operate faster as mentioned earlier. Unlike Meteor, GWT based applications can be delivered as real multipage applications. The simplest approach is to have multiple HTML files at client-
side which are not integrated unlike the Meteor and each handle a separate presentation. These HTML files can be bundled via hyperlinks.

As applied in Javandad application, instead of having multiple html files, JSP files were used at backend. This approach gives less dependability on GWT components to deliver dynamic contents to the client. The result application has a combination of synchronous and asynchronous server-side connectivity.

Unlike Meteor, GWT has only been designed for client-side and at the same time does not enforce any obligations at server-side. In result, the backend can be any technology but as mentioned before having java at server-side leads to a monolingual development environment. As applied in Javandad project, the backend was a combination of JSP and Servlets.

In both GWT and Meteor frameworks, the waiting time for having the container elements ready and the pure data being downloaded from the server-side can be displayed by an embedded progress bar. The browser progress bar is useless in most of cases and it is only usable for synchronous content download from the server-side.

3.4 Account Management

As needed in most of today web applications, a web application needs to distinguish its users and allow them store certain types of information to customize their received services based on them.

By default, Meteor is delivered with login interface/provider packages which allow easier and with lower cost account management by providing all main functionalities including logging in, logging out, password recovery, email verification. In fact by having these packages as standard built-in libraries, not much coding is required to have a fully functional authentication mechanism inside of the final application.

One point of concern would be, if any security problem exists in the functionalities provided by any of these packages, a wide range of applications empowered by Meteor are affected. This of course can be a strong motivation for a large community of hackers to try to breach the security perimeters applied by these built-in libraries.

**Code 3.3.** custom login GUI implemented at Welcome page by calling Meteor.loginWithPassword.

```
Template.Welcome.login = function()
    var email = document.getElementById("loginEmail");
    var password = document.getElementById("loginPassword");
    Meteor.loginWithPassword(email.value,password.value, function(err){
        //code inside
    });
```

As applied in SkillBind application, some functionalities including email verification and password recovery are implemented from scratch. At the same time, the login and signup GUI at the welcome page has been customized.

Unlike Meteor being delivered with user authentication components, Most of web frameworks do not come with a default authentication mechanism. At this stage one of the following decisions must be made:

1. Implementing an independent authentication procedure.
2. Using a third party stand alone authentication framework like Apache Shino or OpenID/OAuth based libraries like JOpenID and Paouth to perform the authentication process with.

As decided in Javandad project, a 3rd party library was chosen to perform the authentication process based on OpenID standard and Google was chosen as the login provider. Despite of using a third party library as the
case of Skillbind project, it took more time in terms of configuration and debugging compared with its counterpart in Skillbind project.

A good point about Meteor is that it also provides login provider packages for using external authentication providers including Facebook, Twitter, Weibo and Google which eliminates any need for a third party library for using any of these four providers. Meteor separates the authentication GUI from the provided authentication services by providing Account-UI package designed only for handling the user interface. In fact using the Account-UI library is optional and applications can still benefit from the provided authentication services by using the login provider packages despite of having customized authentication GUI.

3.5 Session Handling

One of different attributes of Meteor framework compared with traditional frameworks is the way it handles sessions which in fact returns to the different architecture of Meteor framework. As there are no multiple page requests to the server as done in traditional frameworks like JSP/Servlet based applications there is no need to track the user among different requests and therefore no shared session object is required at server-side like in the traditional form of it[23]. As the user logs in the login information including a permanent user id, loaded from the server-side, is stored at client-side which is used for distinguishing different users from each other. This id is used in publish/subscribe mechanism, Remote procedures and also applying rules for synchronized collections.

At the client-side, there is an object called “Session object” which is designed to hold the user’s session related data at client-side. As the session between the client-side and server-side is renewed after each page refresh, the data inside of the session object gets also reset. The “Session object” is a data container suitable for storing intermediate data at client-side which are not needed after each page refresh. If the application needs to preserve data locally at client-side even after each page refresh as the case of Skillbind application, a more persistent data container like Amplify is required.

As GWT is a stand-alone client-side framework only, the co-working server-side framework must have the session handling capability. At the same time GWT has the possibility to access the session data built-in or using RPC calls.

A major issue about Meteor framework is about session timeout handling. By default the session never expires as tested in Skillbind application and there is no functionality provided at server-side to do the job. Once the user is logged in, he stays logged in for an unlimited period of time. As the result the session timeout mechanism must be implemented from scratch or deployed using a 3rd party library.

In case of any need for storing the users’ temporary data from the moment the user logs in until the user logs out, one of the following approaches can be taken:

1. Using synchronized collections.
2. Using Amplify API instead of the session object.

3.5.1 Using synchronized collections

All synchronized collections used in the SkillBind application can hold data permanently and thus they can be used for this purpose. For example “Meteor.users” collection holds a document for each user permanently. But for making the storage temporarily, an extra procedure is required to remove the data after the user logs out.

In this approach a special collection, so called the Session collection, is created at server-side. As the user logs in or requests temporary data storage for the first time, a new document for the user is created inside of the collection. The users’ temporary data is stored at server-side and then synchronized with client-side or vice versa. If the application gets reset, the data is untouched and then synchronized with client-side again. The
client-side version of data could be different than the server-side if the publish method filters the data in a way that some part of the session information is cut out.

For every user, there can be a document in the so called Session collection. A unique identifier like the user id must be used to retrieve the related document to the logged in user as needed. As Meteor based applications perform number of the major functionalities at client-side, the need for sharing some data between client-side and server-side increases. That’s where a subscription from the client to the Session collection located at server-side, allows access to the Session document from the client-side. This access can be controlled by applying some rules from the server-side. For example writing access can be limited or some part of session can be hidden from the client-side.

As the user logs out, the session data for that specific user is not deleted automatically. An extra procedure is required to clean up the session data from the collection at server-side.

3.5.2 Using Amplify API instead of the session object

In this approach data is only stored at client-side. Because Meteor Session object does not survive a page refresh, the temporary data is stored locally using the built-in Amplify library. This approach is useful when the server-side does not need to access such data and the data is only used for client-side purposes.

As the user logs out, the data needs to be deleted from Amplify object manually, otherwise it will be available even though the user is logged out which in return can cause a major security breach.

In the Skillbind application, the Amplify library is used for storing GUI current settings and data to be restored back in case of a page refresh. For example all the text fields’ current values are copied into Amplify object when a page refresh happens and then restored after the GUI is created back again. In result, the user can continue his work after page refresh without any data loss. A page refresh can happen in case of application update by the end of a maintenance process.

3.6 Monolingual Software Development

As Meteor relies on only one language at both server-side and client-side, the development process is expected to become easier compared with other bilingual or multilingual frameworks. The key advantage between using Javascript and any other language is that, any language other than Javascript needs to be compiled to Javascript to become executable by the browser. This enforces an extra step which can increase the development cost.

In case of GWT combined with a Java-based backend framework, software development is done using Java language at both sides. As Java is not executable by the browser, then it must be compiled to Javascript. Because of long compilation time from Java to Javascript specially in bigger projects and not being able to debug the Java code being executed by the browser after compilation, GWT development process happens in a mode so called “development mode” in which a plug-in is used to execute the Java code directly inside of the browser. Once the development process is complete, the client-side Java code is compiled into Javascript which is ready to be executed by any browser with no need for the mentioned plug-in.

Having GWT framework at client-side along with a Java based framework at server-side leads to an environment with two versions of Java language at each side. As the code at each side is compiled with a different compiler, it can cause some confusion and lead to programming errors as experienced during the development of Javandad application. For example, using certain classes from J2SE standard library or none GWT-based third party libraries at client-side causes compile-time errors, as they are not compile-able to Javascript.
3.7 Development Environment

Unlike traditional frameworks, Meteor framework simplifies the development process by requiring no complex configuration for performing deployment, remote procedure calls and data handling. In Meteor framework, any configuration is done inside of the code by using Javascript language.

In contrast, GWT based applications like Javandad, require a large amount of configuration done in different XML[24] files for different purposes including deployment, remote procedure calls and data handling. In such cases, using an IDE equipped with required editing tools can increase time efficiency.

Meteor based applications can be easily deployed locally by using a single command line during the development process. As what the developer writes at client-side is exactly what is executed inside of the browser, the browser becomes part of the development environment in which the developer can observe the output and perform client-side debugging using the browser tools. This is not true about any other monolingual web framework which is not based on Javascript language. In case of GWT, client-side debugging is done by using the Google developer plug-in and Google Eclipse IDE during development process.

On the other side, performing server-side debugging for Meteor based applications is a bit of issue. During development time, a complete log of exceptions and errors triggered from the server-side is available to developers. But as Meteor provides no built-in functionality for performing server-side debugging, it is required to use a third party tool called Node Inspector ,designed and developed for debugging server-side Javascript code executed by Nodejs.

As Meteor framework is available as a stand-alone kit containing required tools and libraries for developing applications and does not involve much development complications, a combination of a Javascript editor, browser and the tools available in the framework standard kit form an efficient development environment.

As the case of Skillbind project, the development process was done by using a multi-language source code editor called Komodo providing a certain set of basic functionalities including smart language detection, code highlighting, basic text editing features and a set of keyboard shortcuts.

Like Meteor framework, GWT is also delivered as a kit but not containing all required tools. As mentioned above, to reduce costs and improve time efficiency, GWT developer plugin, GWT IDE plugin and an IDE are required which do not exist in the standard kit and must be provided separately.
Chapter 4

Data handling

Meteor by default relies on MongoDB, a NoSQL database[25] which is not used in most of web applications as today[26]. Compared with relational databases, working with MongoDB in Meteor comes with no extra work on database initialization and DBMS connection setup. Data collections are created literally inside the program itself which are stored permanently on hard disk by MongoDB at server-side.

MongoDB has a flexible schema and there is no fixed data model for a specific document which means data can be inserted into the document without having the schema determined beforehand[27]. At the same time different documents inside of a specific collection can accommodate different fields. In result, a language like Javascript becomes a perfect match for it as Javascript language itself does not have a fixed schema for its objects or on the other hand it’s fair to say that MongoDB has been designed for Javascript language or alike languages as it almost uses the same format for data storage as Javascript does for object storage.

As mentioned, entity objects are implicitly created by MonogoDB when a query is run on a collection of documents and query result is returned in the form of an array of Javascript entity objects each representing a matched document. These objects are created on fly and may contain different fields depending on the document they are representing. If each document’s schema changes during the application’s lifetime then the returned object will adapt to the change and will accommodate the new pieces of data as new fields. Consequently, it is no possible to declare an entity class initially with a fixed schema with such high level of flexibility in the database schema. In the case of Java language as a language which requires fixed schema for its objects, using MongoDB, the query results are returned as objects of one general container class named BSON[28] instead of specific entity objects with different schemas relevant to each document.

On the other side, Both Java and relational databases have fixed schema which means both database and corresponding entity objects must have a schema before any data manipulation. In such system, relations belonging to the same table have the same data model against their counterpart documents existing in the same MongoDB collection. Entity objects are not either coupled to the query system so entity objects must be initialized out of the query results. By the help of a persistent library like JDO as an abstraction layer between the program and the underlying relational database, objects can be stored to or retrieved from the database directly.

As experienced during development of both projects, having fixed schema requires more coding and designing efforts. At the other side, semi-structured databases cause more programming errors and they require more time and effort being spent on debugging.

In SkillBind Project, different types of data are stored apart in different collections.

Code 4-1. creating collections as data containers at server-side.

```javascript
conversations = new Meteor.Collection('conversations');
notifications = new Meteor.Collection('notifications');
activities = new Meteor.Collection('activities');
activation = new Meteor.Collection('activation');
preset = new Meteor.Collection('preset');
```
In this approach, instead of having one big collection for all possible data, collections work as different data containers which help to separate different types of data. Another benefit that comes with this approach is the possibility of applying different insert, remove and update rules for different collections at client-side.

In contrast, MySQL a SQL based DBMS is used in Javandad project. The MySQL DBMS and a suitable JDBC[29] driver for it are provided separately at server-side. There is also an extra class named “DatabaseManager” accompanied with required Entity classes working as an abstraction layer between the business layer and the database layer to manage the access to the database. As an example it has the responsibility to open a connection to the database and reopen it if closed before executing a transaction.

Using Meteor framework, there are two ways available to access data located at server-side from the client-side

1. Using RPC Calls
2. Using Synchronized Collections

Both solutions are facilitated by Meteor framework. In both cases, most of work is done by Meteor framework and it causes faster development compared with many other frameworks. Depending on the context, any of two could be decided. The key difference between the two approaches is that synchronized collections are live and are updated instantly at client-side. For instance, synchronized collections were used for implementing the messaging system in the SkillBind project so the user can see the received messages instantly without a need to refresh the page.

**RPC Calls**

Compared with many other frameworks like GWT in which the configuration for implementing RPC calls are done partially through XML files, Meteor requires no out of code configuration and less amount of work is done inside the program using Javascript language itself.

**Code 4-2.** calling remote procedure “getMatches” from client-side which is executed every time a user searches for other members based certain criteria.

```javascript
var output = Meteor.call("getMatches", teachv, learnv, countryv,
languagesv, genderv, function(err, data){
});
```

**Code 4-3.** defining the remote procedure “getMatches” at server-side which sends the search results back to client.

```javascript
Meteor.methods(

getMatches: function(tcv, lcv, countryv, languagesv, genderv){
}

);
```

Due to a little amount of extra coding and having no need for any external configuration files, implementing RPC based communication becomes easier with having less difficulties compared with other frameworks. Unlike Meteor, GWT framework involves more coding both at server-side and client-side and some configurations must be done via Java annotations and XML files.
Synchronized Collections

In this new feature of Meteor, for every collection at server-side there could be a counterpart collection declared at client-side which is synchronized with it automatically by Meteor. The synchronization process is entirely controlled by the developer with no need for any extra configuration file. To bundle two collections one at server-side and the other one at client-side, the client-side collection needs to subscribe to the published server-side collection.

At server-side via the method “Meteor.publish”, developers can decide what is going to reside in the counterpart collection at client-side. Via “Meteor.allow” method at server-side, the developer can decide what operations are allowed to be done on the counterpart collection at client-side.

**Code 4-4.** defining insert, remove and update permissions for activities collection.

```javascript
activities.allow({
  'insert': function (userId, upd) {
    /* user and doc checks ,
    return true to allow insert */
    return true;
  },
  'remove': function (userId, upd) {
    /* user and doc checks ,
    return true to allow insert */
    return true;
  },
  'update': function (userId, upd) {
    /* user and doc checks ,
    return true to allow insert */
    return true;
  }
});
```

In fact, synchronized collections create a full duplex persistent connection between the client-side and the server-side by which the client can also get updated by the server without requesting it. For example, this capability is used for implementing the Skillbind messaging system.

For security and performance reasons, synchronized collections cannot be applied to the whole data set and they can only be applied to a certain sub section of the available data at server-side. As the case of SkillBind project, they are used for making the users profile data accessible at client-side. For applications in which there is no custom user data, this feature can become redundant or less used.
Chapter 5

Deployment

The easiest way of deploying Meteor based web applications is using “meteor deploy” command. This command directly deploys the application on Meteor default host servers. Meteor offers hosting every application created by using Meteor framework on its own host servers. The default domain address is the project name as a sub domain of “meteor.com”. For example in the case of Skillbind application, the default domain address is “skillbind.meteor.com”.

If one decides not to use Meteor default host servers because of any reasons like performance or security issues, it is also possible to deploy the application on custom servers. To do that, the command “meteor bundle” is used to export the application to a compressed file deployable on any server having NodeJS and MongoDB installed.

As a good point, providing a custom domain address for the web application is not related to the application being deployed on Meteor default servers or custom ones. For example, Skillbind application is hosted on Meteor default servers but at the same time has the www.skillbind.com custom domain address. First of all, the custom domain address needs to be mentioned in the deployment command, then it must be redirected to “origin.meteor.com” domain address. The rest is done by Meteor default host servers and the customized domain address is redirected to the right uploaded application.

Like Meteor, GWT applications having Java/Python at server-side can be deployed on Google public host servers called Google App Engine[30]. The default domain address is the project name as a subset of “.appspot.com”. For example, the default domain address for Javandad application is “javandad.appspot.com”. Compared with Meteor, deploying GWT applications takes a larger amount of time and appears to be more troublesome. Before each deployment the client-side needs to be recompiled to Javascript.

As Google Appspot provides several UI based utilities and services for administrating and monitoring the uploaded application, the development team has a good control over the application once it is uploaded. Google Appspot provides database, server logs and server instances monitoring capability. In contrast, there is less monitoring and administration capability over Meteor based applications once they are deployed on default host servers. Using the command “meteor mongo”, developers can access the server-side Mongo database through a terminal.

It is also possible to deploy GWT applications on custom application servers. For doing that, the war folder inside of GWT compiled projects can be exported to a zip file and then to a war file. The project compilation can be done by writing ANT scripts from scratch or using an IDE.

As most often, web applications temporarily stop delivering content and become unavailable during the update process. As Meteor is equipped with one feature named Hot Code Pushes[32], web applications can be updated without interrupting users anyhow. The only influence on the users’ work is a page refresh after the update process is completed. Unlike Meteor, GWT applications become unavailable for under 10 seconds while the update process is being completed as experienced in case of Javandad project.
As mentioned before, Meteor based web applications must be ready to handle any page refresh. If the application is not designed for such situation, the result may be damaging and a wide range of users may lose their current unsaved data after a page refresh.

In case of Skillbind project, the www.skillbind.com address is purchased and then redirected to http://origin.meteor.com. The following command is also used to deploy the application on Meteor default host servers:

`meteor deploy www.skillbind.com`

If the application already exists, it works as an application update and the whole existing data at server-side remains untouched.

As discussed above, the deployment process for Meteor applications seems to be easier, more time efficient and less problematic. In contrast, once the application is uploaded, the development team has more control over the GWT applications in terms of administration and monitoring.
CONCLUSION

Meteor as a Single Page Application development framework takes a new approach in developing web applications as it is today. In this new approach, most of computations are done at client-side as much as possible and server-side has mainly the responsibility of data storage. This thesis tried to analyze Meteor framework based on developing a real world web application. This helps to engage with the framework experimentally and find the advantages and disadvantages of using it compared with other traditional frameworks. This thesis did not discuss some controversial topics regarding Meteor including security and performance which are out of scope of this study.

Web development using Meteor has the following advantages and disadvantages as experienced during the development of SkillBind project.

Advantages

**Built-in account management**
Most of the work is done by the framework itself. There is the possibility to customize the provided services and use them partially.

**Easy data handling**
No out of code configuration is required as it is done literally inside of the code. Data containers are explicitly declared using Javascript and implicitly handled by the framework. As facilitated by Meteor, small amount of coding is required to implement RPC calls and synchronized collections. Automatic data synchronization at some level unifies the client-side and server-side and converts them into one tier.

**Responsiveness**
As GUI elements are generated locally and only raw data is downloaded from the server-side, it improves the responsiveness of the application. Instant local read/write access to the data for client-side computations also has a direct effect on the performance of the application.

**Monolingual development**
Using one language both at client-side and server-side without the need to do any extra task or use any extra tool during development.

**Effortless inline configuration**
All configurations are done using Javascript. This results in no configuration files needed to be bundled at execution time. Meteor framework also requires a small amount of configuration in total.

**Effortless deployment**
As Meteor provides default host servers and by default no export file is required to deploy an application, the deployment becomes fast and effortless. Meteor also does not require an application to stop operating while redeployment.
Disadvantages

**Complicated GUI development for multipage applications**

As Meteor is designed for developing Single Page Applications, developing multi page applications involves using third party client-side libraries which consequently results in spending more time and effort. This would be the case for even having one extra page than the main page which is accessible via a unique URL.

**Problematic session handling**

As there is no built-in container to store session related information as the user logs in until the user logs out neither at client-side and server-side, extra procedures are required to implement the functionality using existing data containers. Another problem is about the session never getting expired which requires from scratch implementation or using a third party library.

In my opinion Meteor framework in overall makes web development more convenient and faster by facilitating different web development aspects including data handling and transfer, account management and application deployment. Compared with GWT framework, the developer faces fewer problems during development process, fewer extra development tools and configurations are required and all of the configurations are done using the Javascript language itself.

If Meteor is used for developing conventional multi-page alike applications, the GUI implementation and session handling implementation can cause more development cost. As designed, Meteor framework shows to be a good choice for developing single page applications and at the same time it combines the experience of web application development with the experience of desktop application development by unifying the server and client tiers into one.

In fact Meteor has managed to make web application development simpler and less costly and at the same time it can be considered as a leap forward toward moving business logic from the server-side to the client-side.
REFERENCES


Appendixes
Figure 1. SkillBind welcome page containing the sign up and log in form.

Figure 2. welcome email from SkillBind containing a unique key.
Figure 3. The Search tab content pane.
Figure 4. the Browser tab content pane.
**Figure 5.** the Contacts tab content pane.
Figure 6. the Profile tab content pane.
**Figure 7.** The Activities tab content pane.
Figure 8. The MessageBox tab content pane.
Figure 9. the Notifications tab content pane.
Figure 10. SkillBind project, source file packaging at client-side.

Figure 11. Password recovery dialog box accessible from the welcome page.
Figure 12. Komodo, a web source code editor used in SkillBind project.
The publishers will keep this document online on the Internet - or its possible replacement - for a considerable time from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: http://www.ep.liu.se/

© Sahand Sadjadee