

E-BOOK USER ACTIVITY ASSESSMENT

PROJECT REPORT

CSC 899

by

Selman Kahya

selmanhalid@gmail.com - (908) 910 - 1938

San Francisco State University, California, USA

December, 2013

TABLE OF CONTENTS

1. Introduction	4
2. Background	5
2.1 E-Books.....	5
2.2 Problem Statement.....	6
2.3 Related work	7
3. System Design	9
3.1 Technologies.....	9
3.2 Roadmap, Specific Aims & Methods.....	10
3.2.1 Needs (or Usability) Assessment.....	10
3.2.2 Technology Study.....	10
3.2.3 App Functionality and Interface Design	10
3.2.4 Creating the Back-end	10
3.2.5 Developing the Application.....	11
3.2.6 Deployment and Final Evaluation.....	11
3.3 Deciding How to Track Activities and Present Data	11
3.4 Radium: Open Source Technology	13
3.5 Use Cases.....	14
3.6 System Architecture	16
3.6.1 Radium App.....	16
3.6.2 Tracking Script.....	17
3.6.3 Back-end - Handling Requests.....	17
3.6.4 Main Application	18
3.6.5 Database.....	19
3.6.6 Deployment	20
4. Implementation	21
4.1 Back-end.....	21
4.1.1 Technologies and libraries used	24
4.2 Implementing Google Chrome App.....	26
4.2.1 Technologies and libraries used:.....	27
4.2.1 Radium Integration.....	27

4.2.2 Tracking Library.....	28
4.2.3 Main Application	31
4.2.4 Screenshots.....	35
5. Conclusion.....	41
5.1 Next Semester Plan - Future Directions.....	41
6. References.....	42

Chapter 1

1. INTRODUCTION

This document describes the background, design and implementation of the E-Book User Activity Assessment Project developed by Selman Kahya under the supervision of Ilmi Yoon. Project scope covers the design and implementation of a Chrome Application.

The second chapter of this report explains the background of the project and also introduces related work in the field.

The third chapter describes the overall system design of the application. This includes technologies that are being used, roadmap, use cases, and the system architecture including front-end, back-end and database components. Design decisions have been made here will be used for the implementation later on. This chapter aims to help reader to understand the development process better.

The fourth chapter deals with the implementation of the application. It gives more detail about the technologies and languages used to implement the application including front-end, back-end and database. The reasoning behind these choices will be discussed, and finally this chapter will describe several interesting technical challenges and their solutions.

In the last chapter, conclusion has been drawn and the results are argued with respect to the initial project goal. Further work is also discussed, including new features and potential new directions that might be incorporated in the future.

Chapter 2

2. BACKGROUND

This chapter covers the background of the project, problem statement and related work in e-book user activity assessment. It gives an insight into the evaluation of people's reading behavior and explains the overall goal of the project.

2.1 E-BOOKS

An electronic book (variously: e-book, eBook, e-Book, ebook, digital book, or even e-edition) is a book-length publication in digital form, consisting of text, images, or both, readable on computers or other electronic devices. E-books are getting more popular each day. There are several advantages of e-Books that people prefer it over traditional books. Historically, to buy a book, you had to go to a bookstore, and if they don't have it, you had to try other bookstores until you find it. However, with digitization of the books, now it is possible find books online by searching for the book through e-stores such as Kindle Store, Google Play or App Store. Then, it would take couple of clicks to buy the e-Book. Another reason could be that, instead of carrying the books or having storage to keep them all, it is relatively easier to have a device with couple of gigs of memory. You can store tens of, hundreds of books in your device without worrying about the physical presence of books. It is even possible to read books online without saving them in your device.

In order to read e-Book, reader must have an e-Book reader device and/or an e-book reader application. Many e-Book readers come with a built-in reader application such as Amazon Kindle Cloud Reader, while some other devices would require you to have e-

Book reader application installed on your device. Fortunately, there are tons of free e-Book reader apps in the stores, so reader doesn't have to pay for it.

All these advances in reading technology has brought greater range of opportunities for publishers and authors to offer and create better products, and for users to see their reading behavior, get suggestions based on the reading activity and increase reading comprehension by reading smarter and more conscientiously. By tracking user activity, companies can offer useful services to users including what book to read next, word-lookups that user did and helping user to memorize these words by playing a game, or getting visualized analysis of their reading behavior etc.

2.2 PROBLEM STATEMENT

E-Book User Activity Assessment is a project that implements a possible and plausible solution to create a showcase of tracking and presenting user reading activity by creating new services based on the assessment. This requires Internet and Mobile device communication, ePub, JavaScript, AJAX, PHP, MySQL, and mobile app development.

In the past, publishers and authors had no way of knowing what happens when a reader starts reading a book. It wasn't possible to track their activity at all. Does reader quit after reading just one or two pages, or finish it in a single sitting? How many minutes do they spend on each page? Do they read introduction or just skip it? Which parts does get their attention so that they underline passages and taking notes in the sides?

Now, with the emerging e-Book market and advanced e-Book readers, it is becoming possible to track and analyze not only how many people buy or read a particular book, but how intensely they read it. Unlike traditional books, E-Book Reader (e.g. Kindle, iBook) can

produce very valuable assessment and information of the book readers from the natural reading activity.

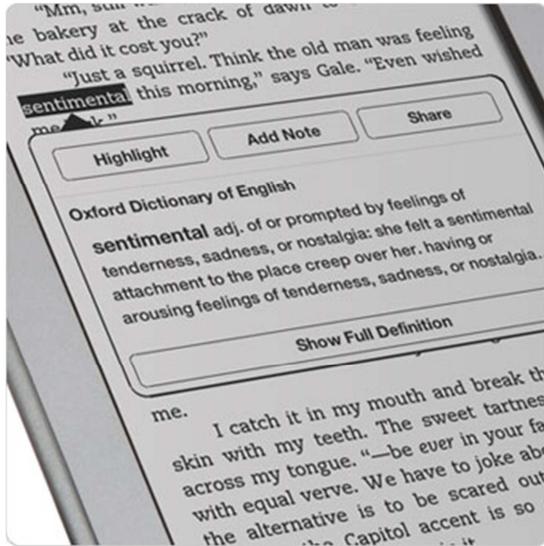
E-Book Assessment Project will allow us to track minutes per page to assess reading comprehension level for individual reader and to find the most popular pages among books across many readers as well. Through this app, users of similar reading level and interest can share their thoughts and participate to play various word games to challenge each other.

Not just for only grown-ups, e-Books are also recognized by the usefulness for kids. Features that our system will offer can also be used by parents to help their kids to grow reading skill in a systematic and fun way. Kids can take word quizzes and challenge with other kids in a similar reading level. Parents can be provided with the average minutes per page for specific book, statistics from kids with comparable reading level and the words that their child look up while reading.

The goal of this project is to study the way people interact with e-books by tracking their reading behavior and activities, and looking for the services that can be offered using the tracking data.

2.3 RELATED WORK

After doing a research on the field, it is found that Kindle and iBook usually provide text description while our application will be showing visual images for all the words that they look up while reading.



Amazon Kindle Word-Lookup



iPad E-book Reader Word-Lookup

Word look ups can be collected, organized with visual/audio representation that Kindle or iBook don't provide, and then made accessible to the readers with their favorite presentation styles. Such knowledge can be effectively used for assessment of reading level as well as seeds for many possible educational and fun activities. For example, users can challenge with other users of similar reading level by playing various quizzes and games which will help them to increase their reading comprehension.

It is also found that nearly all popular E-book readers either don't track user activity or don't have an interface that allows users to see their reading activity. It is in our plan to have a module that visualizes user activity, and provides functionality for user to interact with other readers.

Chapter 3

3. SYSTEM DESIGN

This chapter covers the conceptual design of the project including front-end, back-end and database components. The implementation of the application will be based on the architecture presented in this chapter.

3.1 TECHNOLOGIES

In order to create a showcase, we will be designing an environment that users can read e-Books (in .epub format) while we are tracking their activity, and they can do word look-ups using the e-Book reader application. This will require knowledge of HTML, CSS, JavaScript, AJAX, MySQL and ePub.

We will develop a Chrome application that will run within the Chrome browser (www.google.com/chrome). There are two options:

Chrome Packaged Application: Applications in this category deliver a great experience as native apps, and also provides a sandbox to ensure the safety of users. Just like web apps, Chrome Apps are written in HTML, JavaScript, and CSS.

Chrome Extension: Extensions are small software programs that can modify and enhance the functionality of the Chrome browser. Extensions are also written in HTML, JavaScript, and CSS.

3.2 ROADMAP, SPECIFIC AIMS & METHODS

3.2.1 NEEDS (OR USABILITY) ASSESSMENT

We will conduct a short survey to determine which type of Chrome application we should develop: Chrome Packaged Application or Chrome Extension.

3.2.2 TECHNOLOGY STUDY

After determining the type of the application, we will make our research to find the best way to develop it. One option could be that our application might run top of another e-Book reader chrome application or extension. So, we won't have to spend time on developing an e-Book reader, instead we can spend more time on thinking how to analyze and present tracking data. Since many technologies will be used in this project, we also need to do some research to understand how to connect and integrate components of the system.

3.2.3 APP FUNCTIONALITY AND INTERFACE DESIGN

Based upon the result from needs assessment and the technology study, we will decide the conceptual design of the application.

3.2.4 CREATING THE BACK-END

We will create a RESTful API that handles incoming requests from front-end. Tracking data will be stored in MySQL database. Back-end also will be responsible for complex tracking analysis operations while the application will be responsible just for presenting the data, nothing more.

3.2.5 DEVELOPING THE APPLICATION

We will start developing front-end and back-end in parallel. Application will be responsible for tracking user activity while reading and also be responsible for providing word lookup functionality which requires us to use Google Dictionary API and Google Image Search API.

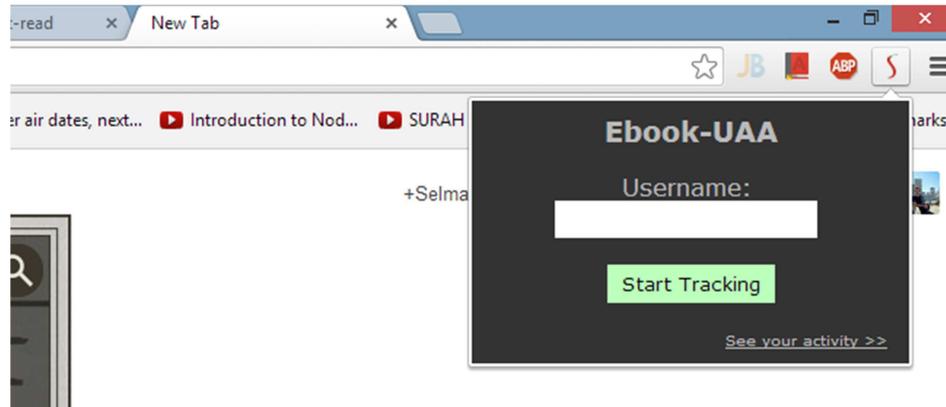
3.2.6 DEPLOYMENT AND FINAL EVALUATION

We will document our findings and draft an executive summary that includes what could be done next. If developed successfully, chrome application or extension can also be published in the Chrome App Store.

3.3 DECIDING HOW TO TRACK ACTIVITIES AND PRESENT DATA

Our initial plan was to develop a Chrome Extension which enables developers to manipulate the web page that user is currently viewing. If we could inject our tracking code to current webpage, we would be able to track user activity while reading content.

After developing a prototype, we realized that we couldn't do what we want with the functionality that Chrome Extension provides to track user activity as well as visualize the data.



Ebook-UAA Chrome Extension Prototype

One of the problems was deciding how to start tracking of user activity. Obviously, users won't be reading e-books all the time, since Chrome is a web browser. To solve this problem, we put "Start Tracking" button as you can see in the picture above. Then, there was another problem. Users should've registered to our system, so that, we can process each incoming request to server based on the user information. And extensions don't allow having UI that is more complex than a small box shown in the picture.

We discovered that if we develop a Chrome extension, we will have to create a web page that users can register and see their activity. That would increase the scope of our project as well as the complexity of the overall design.

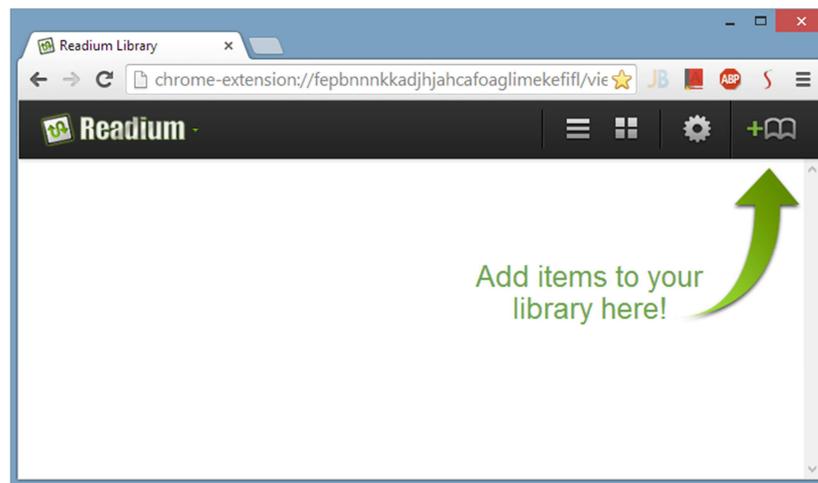
As a result, we decided to create a Chrome Packaged Application by using AngularJS - JavaScript MVC Framework - which allows us to have all the required features as separate modules and have both a tracking script and user interface for registration and viewing activity in a single platform.

3.4 READIUM: OPEN SOURCE TECHNOLOGY

After deciding to go with Chrome Packaged Application, we started looking for an open-source web application that provides e-book reading and uploading to include into our application. Since creating a new e-book reader was out of scope of this project, this seemed to be the best solution.

Radium is an open source technology for handling EPUB documents. It has an API that makes it really easy to upload, view and develop EPUB content. They also have an open source E-Book Reader Chrome Application “Radium” that is developed for Chrome Browsers and can be found on Chrome App Store.

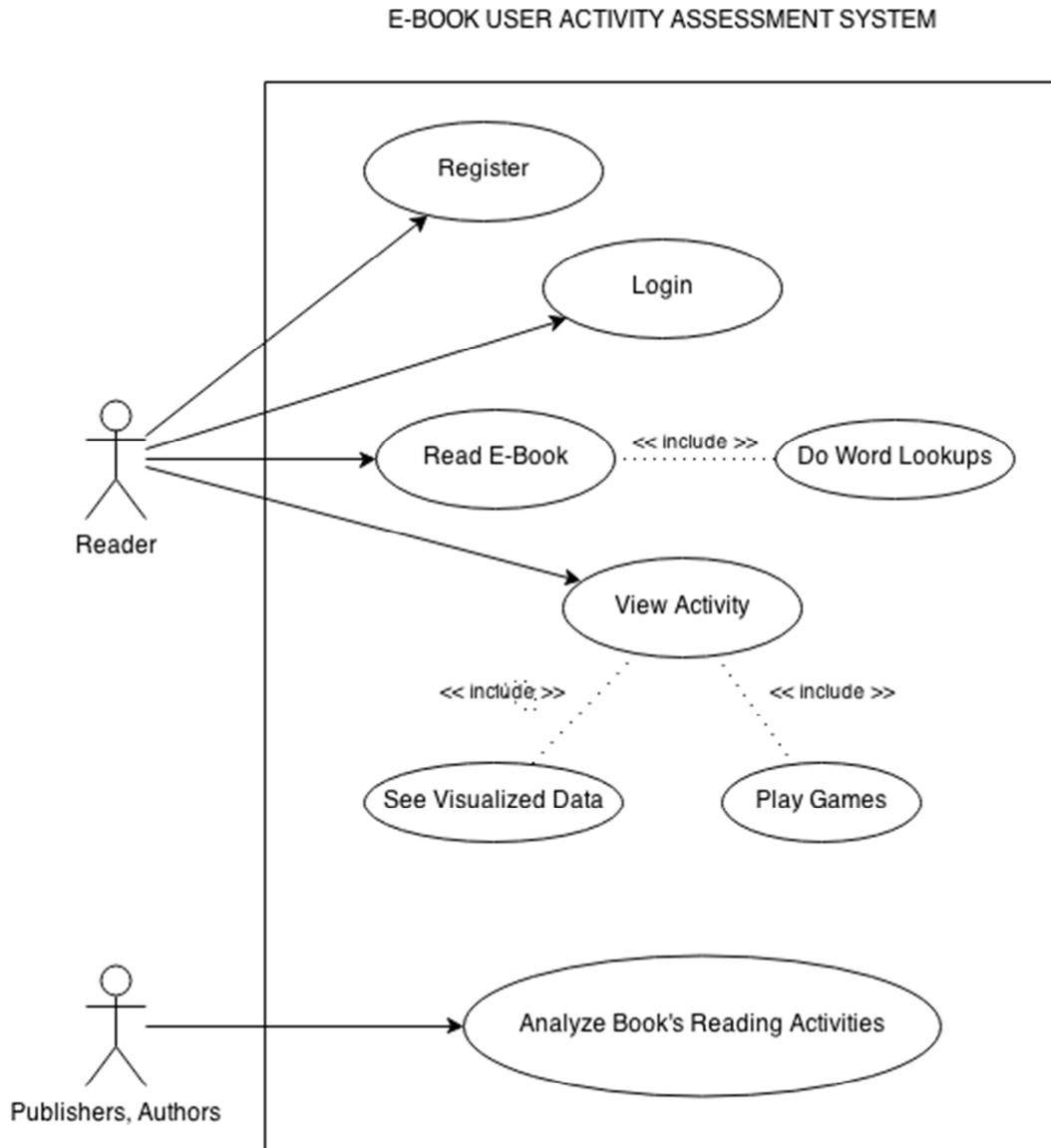
After carefully analyzing the Radium E-Book Reader Chrome Application, since it is open source project, we decided to include it in our application and use it as our e-book reader module. In that way, we won’t have to spend time with creating an e-book reader application from scratch. For details, you can visit www.readium.org/.



Radium E-Book Reader Chrome Application

3.5 USE CASES

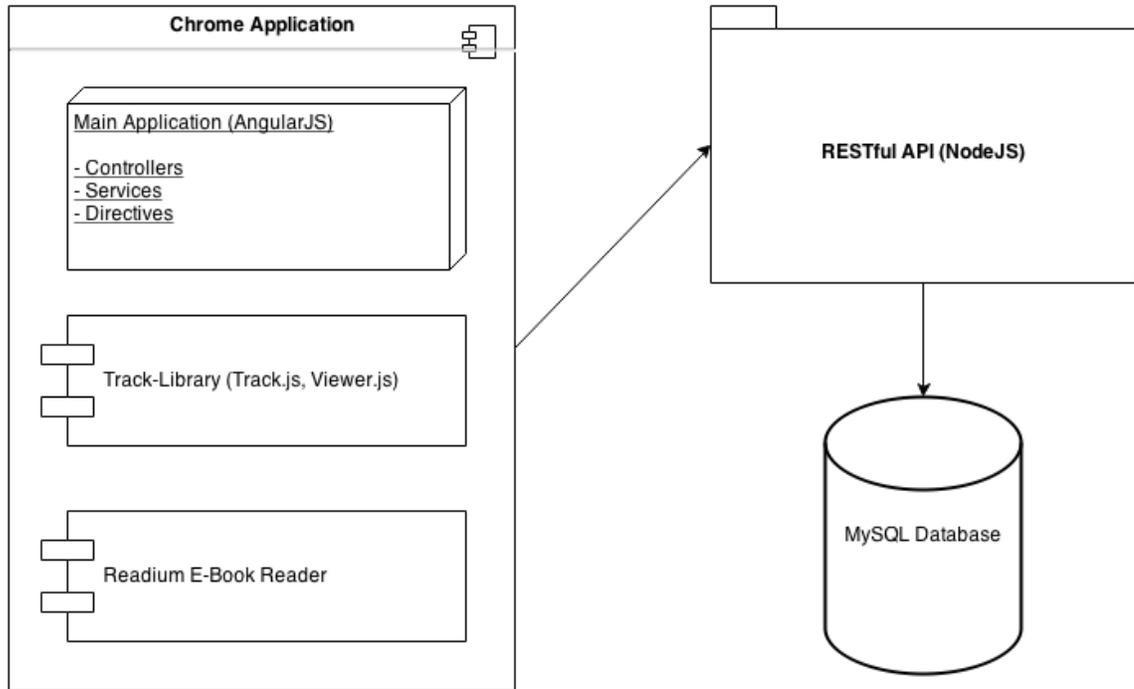
Possible use cases of our system are described below.



1. **Register:** User can register to the system by filling a simple signup form. Only registered users will be able to use the application.
2. **Login:** User can use the application only after logging in successfully which requires an e-mail address and password.
3. **Read E-Book:** User can read e-books using the application. This may include upload, and read already uploaded e-books.
 - a. **Do Word Lookups:** While reading an e-book, user can do word lookups to see the definition of the word by double clicking on them.
4. **View Activity:** Users can view their activity after logging into the system. This may include how much time spent on each book, word lookups etc.
 - a. **See Visualized Data:** User can see his/her activities that are presented by using charts, graphs and so on.
 - b. **Play Games:** User can play single and/or multiplayer games with other registered users that are in similar reading level. This aims to increase reading comprehension level for individual reader.
5. **Analyze Book Reading Activities:** Publishers or authors can view collected tracking data for their content to analyze readers' reading behavior while they are reading their book such as how much time spent on a particular page, on which pages readers did more word lookups etc.

3.6 SYSTEM ARCHITECTURE

Each component of the system will be discussed in this section.



3.6.1 READIUM APP

As mentioned before, we are planning to integrate Radium E-Book Reader – Chrome Application into our application. At the end of this process, instead of two separate Chrome applications, there will be just one main application which is explained in detail below.

When user opens our application, first, we will ask user to login. After a successful login, user will be able to start reading e-books by clicking Start Reading button on main application homepage. This will redirect user to the Radium home page while user is still inside our Chrome Application without user noticing it.

3.6.2 TRACKING SCRIPT

Since we are using Radium to have e-book upload and read functionality in our application, we have to write a script, and import it on the Radium e-book viewer page in order to track user activity. We do that by including Track.js on viewer.html. More details about this can be found in implementation chapter.

3.6.3 BACK-END - HANDLING REQUESTS

For handling tracking activity requests, and storing them in the database, we are planning to create a RESTful NODE.JS API that our application will leverage. One of the main reasons that we decided to implement our back-end using NodeJS is having prior knowledge and experience with NodeJS. Other reason is being able to easily create a communication layer between NodeJS API and AngularJS using Angular \$resources.

There are four kinds of requests will be made to the server and we are going to create four classes to provide the following functionality:

1. **Activity:** New activity, Get Activities.
2. **User:** Register user, Check login credentials, Get user's activity.
3. **Word Lookup:** Create new word-lookup DB record, Get user's word-lookups.
4. **Socket:** Currently, some games require sockets to have multi-player option. Socket functions will handle those requests.

3.6.4 MAIN APPLICATION

As we decided to develop Chrome Packaged Application, instead of Chrome Extension, we needed to decide which JavaScript framework to use to create such application. It is important to mention that only HTML, CSS, and JavaScript are allowed for Chrome Application and Extension development.

Because of having prior experience with AngularJS (MVC Framework written for JavaScript) and it works perfectly within the Chrome browser, we decided to use AngularJS to create our main application.

Our application structure is planned as follows:

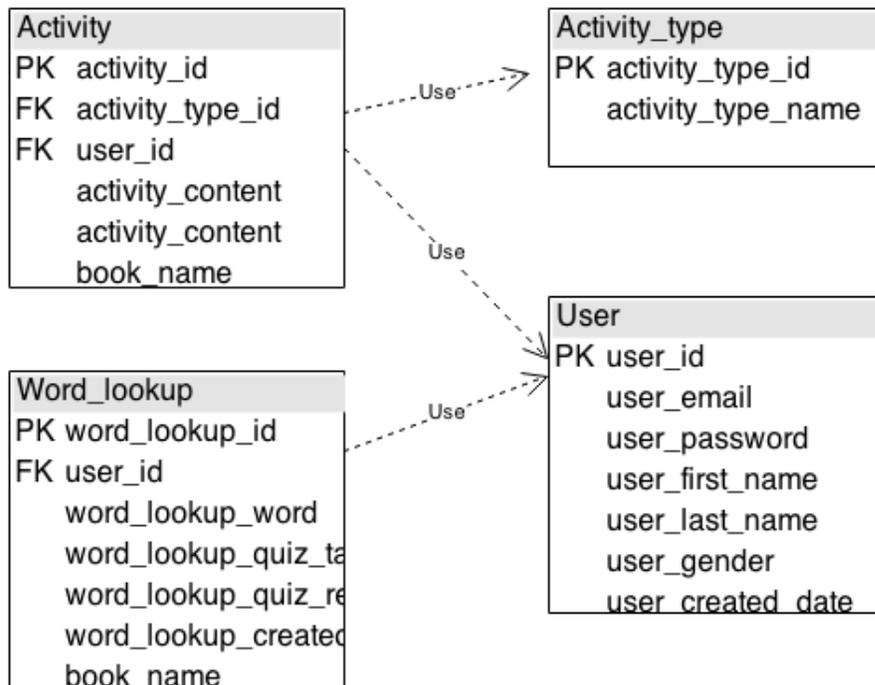
- Home
- Register & Login
- Dashboard (visualizes user activity)
- Word-Lookups (show word-lookups that user did filtered by books)
- Games
 - o Word-lookup Quiz (single player)
 - o Word Game (multi-player)

This initial structure will be extended during the second semester.

AngularJS is a Model-View-Controller framework which means we are required to have at least six controllers and six views in order to provide the functionality introduced above. Application structure will be explained in more detail in implementation section of this report.

3.6.5 DATABASE

We are going to use one of the most popular relational databases: MySQL. It is easy to execute queries in the API using node-mysql NodeJS library. Our database schema is shown below.



Activity: Stores user activity information which can be user started reading, finished reading and changed the page.

Activity_type: Possible activity types are defined here.

User: Holds personal information of all users.

Word_lookup: User word-lookups are stored in this table.

3.6.6 DEPLOYMENT

We will use AppFog, which is a popular PAAS cloud provider and offers free membership, to deploy our API.

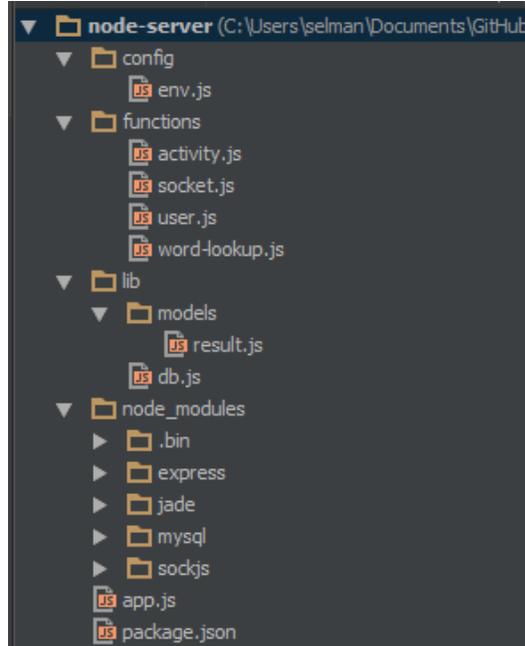
Chapter 4

4. IMPLEMENTATION

This chapter deals with the implementation of the application. It gives more detail about the technologies and languages used to implement the application including front-end, back-end and the database. The reasoning behind the architectural decisions is discussed, and finally this chapter will introduce several interesting technical challenges and their solutions.

4.1 BACK-END

NodeJS is used to create a back-end server that Chrome application will leverage.



Folder Organization

App.js: Server instance is being created here. It also has endpoint definitions.

```
var app = express();

// SET environment configuration
var config = require('./config/env');

// Endpoint handlers
var user = require('./functions/user');
var activity = require('./functions/activity');
var wordLookup = require('./functions/word-lookup');

// ACTIVITY ROUTES
app.post(  '/activity', activity.create);
app.get(   '/activity/:id', activity.read);
app.post(  '/activity/:id', activity.update);
app.delete('/activity/:id', activity.delete);

// WORD LOOKUP
app.post(  '/word-lookup', wordLookup.create);
app.get(   '/word-lookup/:id', wordLookup.read);
app.post(  '/word-lookup/:id', wordLookup.update);
app.delete('/word-lookup/:id', wordLookup.delete);

// MEMBERSHIP Routes
app.post(  '/user/register', user.register);
app.post(  '/user/login', user.login);

// USER Routes
app.get(   '/user/:id/activity', user.activity);
app.get(   '/user/:id/word-lookup', user.lookup);
app.get(   '/user/:id/word-lookup/books', user.books);

// Server is starting, sit tight!
var server = http.createServer(app);

server.listen(process.env.VCAP_APP_PORT || 3000, function(){
  // start socket
  socket.init(server);
});
```

All the endpoints are defined here. For example, if there is a GET request made to `localhost:3000/user/1/activity`, server returns all the activities of user with `id = 1`. Handler functions are defined under `/functions`.

Package.json: Project information and dependencies are stored here.

```
{
  "name": "application-name",
  "version": "0.0.1",
  "private": true,
  "scripts": {
    "start": "node app.js"
  },
  "dependencies": {
    "express": "3.4.0",
    "jade": "*",
    "mysql" : "2.0.0-alpha9",
    "sockjs": "0.3.8"
  }
}
```

Env.js: It is used to define environment variables such as MySQL server credentials. Here is how they can be defined:

```
app.configure('development', function(){
  console.log('Running on development mode..');
  mysqlConfiguration = {
    host : 'localhost',
    user : 'root',
    password : '',
    ...
  }
});

app.configure('server', function(){
  ..
});
```

As it is shown above, environment variables will be set based on the running environment. If API is running on local host, development mode configuration will be applied; otherwise, if it is running on the cloud, server configuration will be applied instead.

Functions/*.js: Request handler functions are defined under “functions” folder. Here is an example request handler function:

```
// returns user activities (e.g. /user/1/activity)
exports.activity = function(req, res){

    var user_id = req.params.id;
    db.execute('    SELECT * FROM activity
                WHERE user_id = ?
                ORDER BY activity_created_time ASC',
                [user_id], function(err, result){

        if(err)
            res.send(500,
                new resultModel.result(false, {},
                ['Error while getting word lookup data!']));

        else{
            res.send(new resultModel.result(true, result));
        }
    });
};
```

Lib/db.js: Currently, it has just one database operation function: `executeQuery()` which takes SQL string as a parameter and execute the query.

Lib/models/*.js: Models are defined under “models” folder.

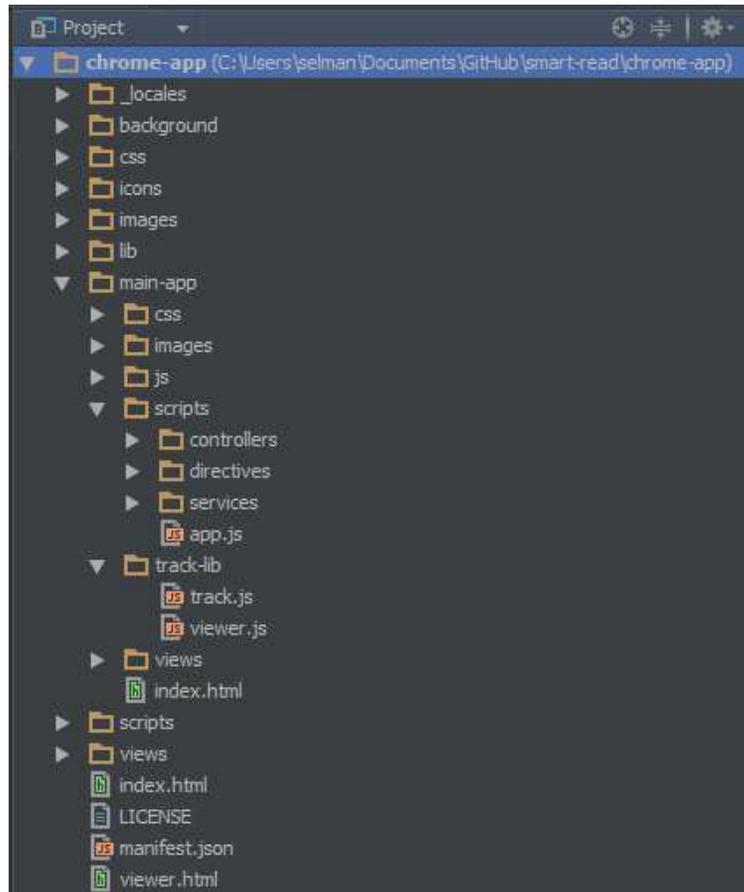
4.1.1 TECHNOLOGIES AND LIBRARIES USED

- **NodeJS:** It is a platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices. [1]
- **Express Framework for NodeJS:** Express is a minimal and flexible node.js web application framework, providing a robust set of features for building web applications. [2]

- **node-mysql:** A pure node.js JavaScript Client implementing the MySQL protocol. It helps to execute MySQL queries using defined MySQL server.
- **SockJS-node:** It is a Node.js server side counterpart of SockJS-client browser library written in CoffeeScript. The API design is based on the common Node API's like Streams API or Http.Server API. [3] We use SockJS-client library to create sockets on NodeJS server.

4.2 IMPLEMENTING GOOGLE CHROME APP

In this section, implementation of the Google Chrome Application will be discussed in detail.



Chrome Application Folder Organization

4.2.1 TECHNOLOGIES AND LIBRARIES USED

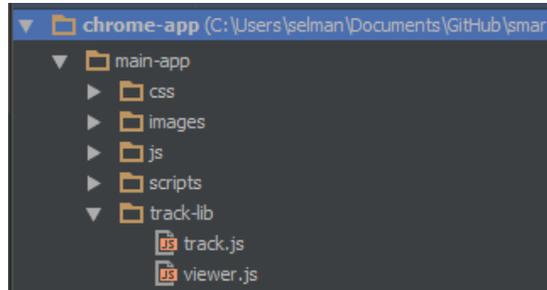
- **HTML, CSS, JavaScript**
- **jQuery:** jQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers. [4]
- **AngularJS:** It is an open-source JavaScript MVC framework, maintained by Google, which assists with running single-page applications.
- **SocketJS-client:** SockJS is a browser JavaScript library that provides a WebSocket-like object. Multi-player games are supported with the API that this library provides. [5]
- **Radium API:** It is a JavaScript library for browser-based EPUB 3 reading, which was initially deployed as a Google Chrome browser extension. More detail will be given on this in Radium Integration section.

4.2.2 READIUM INTEGRATION

As mentioned before, we decided to include Chrome Radium E-Book Reader Application into our project, so that, we don't have to deal with the process of uploading, and reading e-books. Radium does all these for us, while we just put our tracking code to viewer.html. Radium folders are _locales, background, css, icons, images, lib, scripts, views, index.html, and viewer.html.

4.2.3 TRACKING LIBRARY

Tracking scripts are located at /track-lib under main-app folder.



Track-lib Folder Organization

Track.js: This file is imported in viewer.html. It tracks user activity while user is reading an e-book.

```
<!-- viewer.html tracking script -->
<script src="../../main-app/track-lib/track.js" type="text/javascript">
</script>
<script src="../../main-app/track-lib/viewer.js" type="text/javascript">
</script>
```

Init function in track.js initializes the required variables and binds page changing events to the buttons. So, when user clicks on next page or previous page button, server is being notified.

```
// attach a function to page change event
$( "#next-page-button" ).click(function() {
    pageChanged();
});
```

In pageChanged function, a call is made to the server as follows:

```
// activity_type_id = 3 ---> page_changed
SMARTREAD.services.ActivityService.newActivity({
    activity_type_id: '3',
    activity_content:
        "{old: '" + previousPage +
```

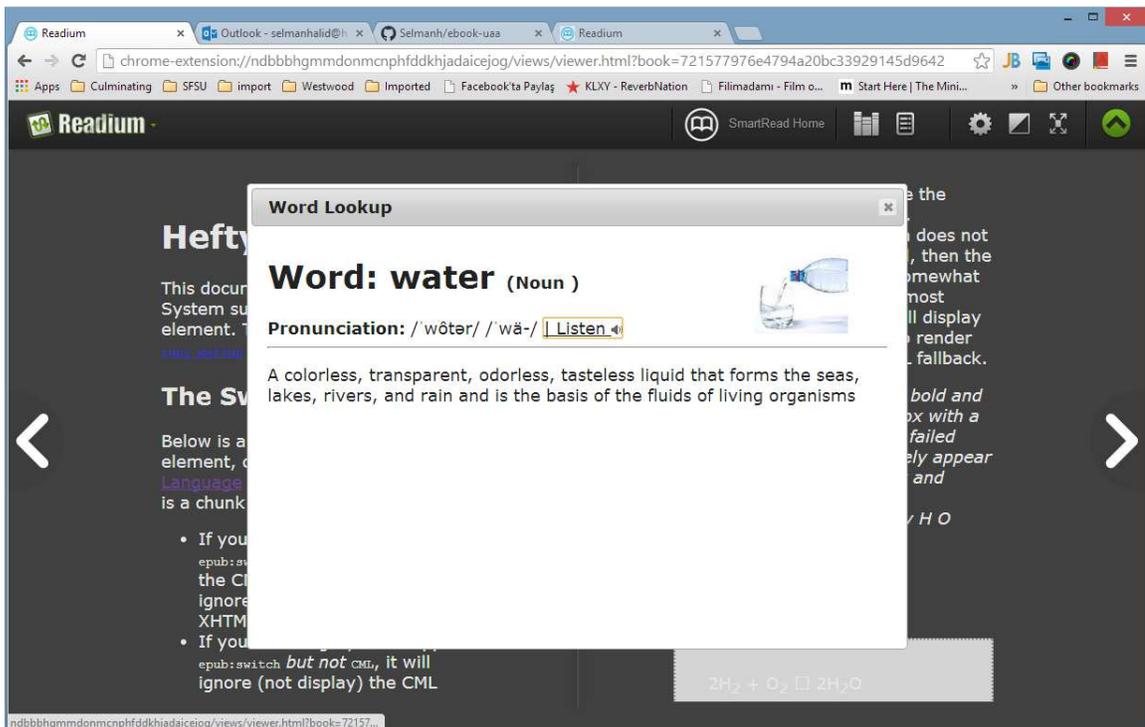
```
        "", new: "" + currentPage + ""}"}  
    }, function() {})
```

`newActivity` function is defined under **SMARTREAD.services**. The other functions (services) that make calls: `LookupService` (using Google Dictionary API), `ImageService` (using Google Image Search API).

Viewer.js: Features that we want to add to E-Book reader module should be defined here. Currently we have word-lookup function that provides word-lookup when user double clicks on a word.

When page is fully loaded, we bind double click event to all words between body tags:

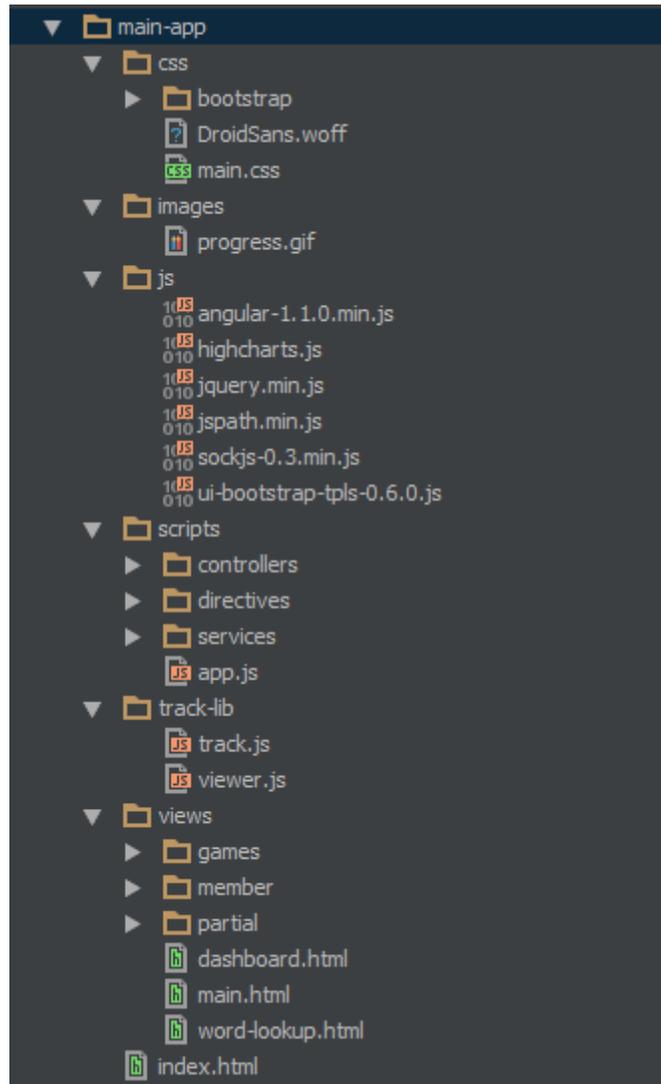
```
// iframe is loaded, it is time to bind events to epub reader!
var bindEvents = function(iframe){
    $(iframe).contents().find("body").unbind( "dblclick" );
    $(iframe).contents().find("body").on(
        'dblclick', viewerWordLookup
    );
}
```



Word Lookup (After double click on a word)

4.2.4 MAIN APPLICATION

This is the main application that user always interacts with - except reading an e-book (we use Radium for reading).



Main Application Folder Organization

CSS, images, js: All external libraries, css framework files, and images are stored under those folders.

Scripts/app.js: This file is the starting point for Angular application. Routes and application configuration is defined here.

AngularJS is a MVC framework, so it should have its controllers and views.

Views/*.html: Views are stored here in html format. There is a view for each route defined in AngularJS.

Scripts/*.js: AngularJS Controllers, Directives, and Services are defined here.

- **Controllers:** In Angular, for each view, there is a controller that sends commands to its associated view to change the view's presentation of the model.

Example controller:

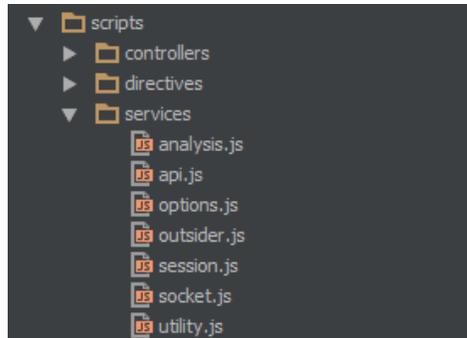
```
mainApp.controller('LogoutCtrl',
    function ($scope, $http, $timeout,
              $location, member, session) {

        // first logout the current user
        // even if he already logged in
        session.end(function() {
            $location.path('/login');
        });

    });
```

Logout controller is using session service to terminate current user's session.

- **Services:** Angular services are singletons objects or functions that carry out specific tasks. Most of our services are responsible for making calls to the server.



Services Folder Organization

Example AngularJS service:

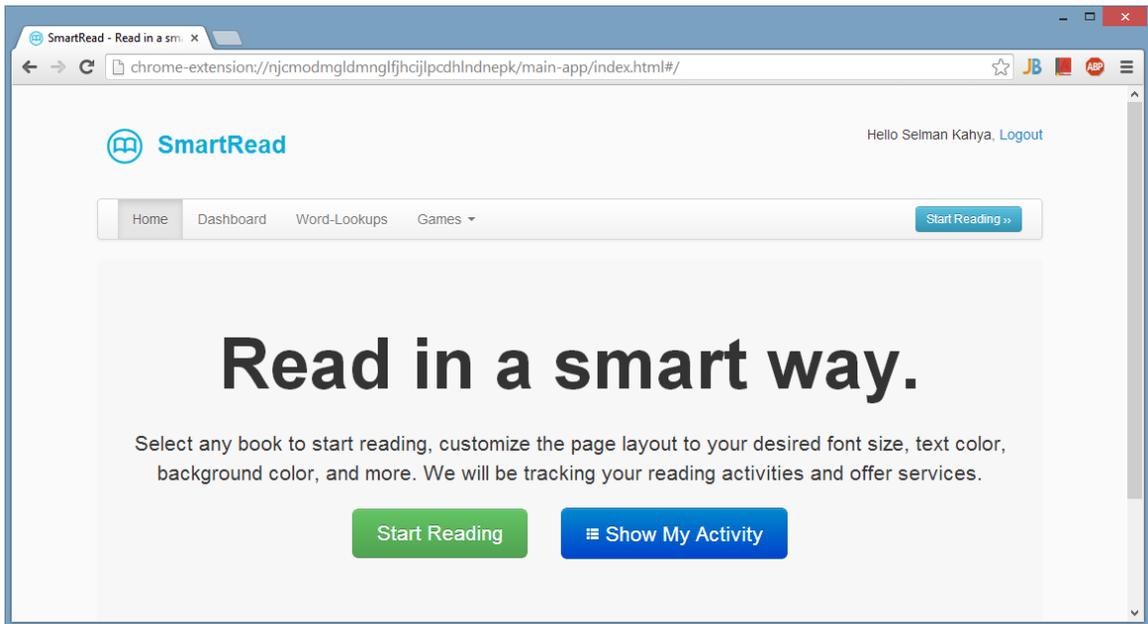
```
api.factory('userService', function($http, serverOptions) {
  return {
    activity : function(user_id, callback){
      $http({
        method: 'GET',
        url: '/user/' + user_id + '/activity'
      })
      .success(function(response) {
        callback(response.result);
      });
    },
    ...
  }
});
```

- **Directives:** Directives in AngularJS are used to make custom HTML elements and simplify DOM manipulation. They can modify the behavior of new and existing DOM elements, by adding custom functionality, like a datepicker or an autocomplete widget. Currently, only directive we have is highcharts-ng. It draws graphs with the given data.

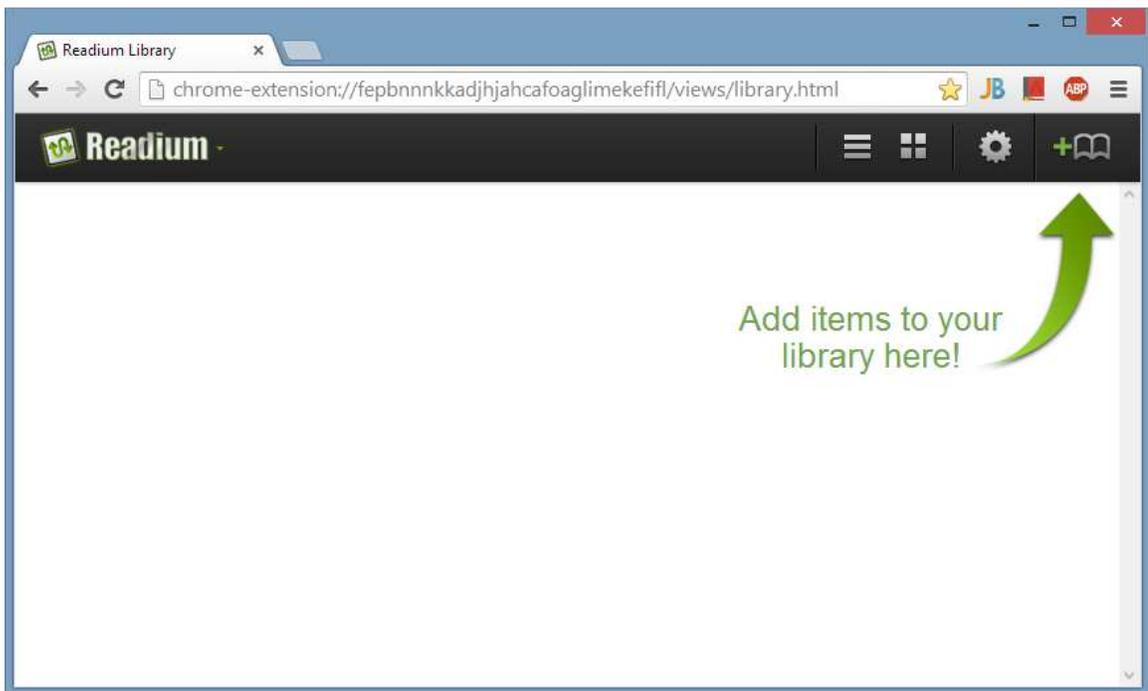


HighCharts Directive in use

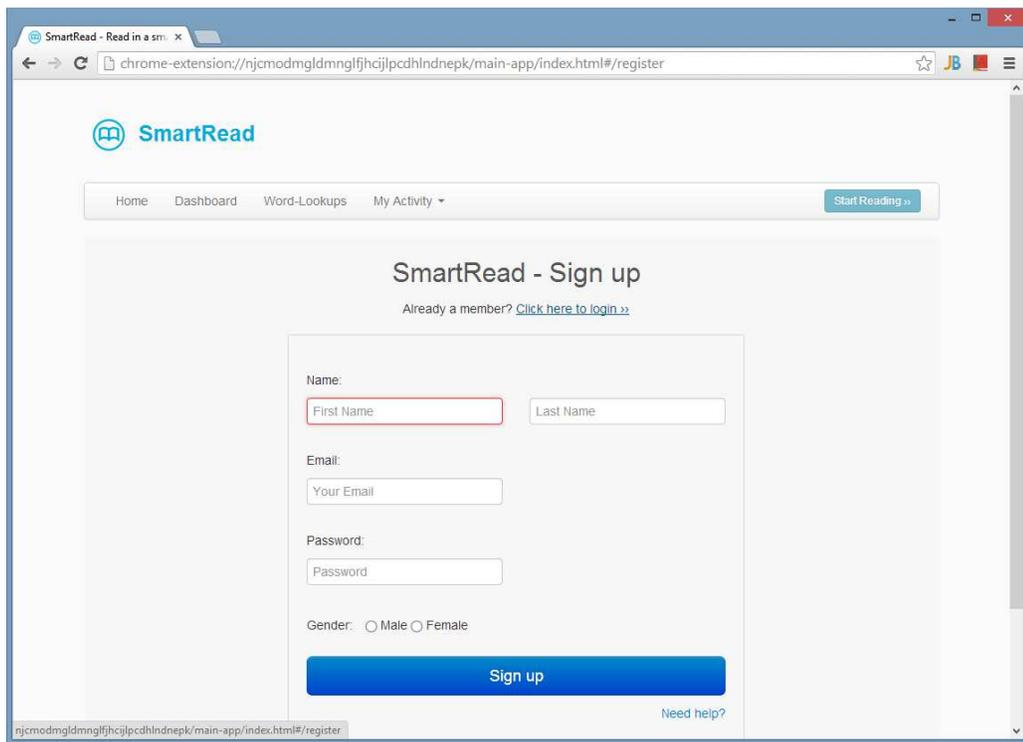
4.2.5 SCREENSHOTS



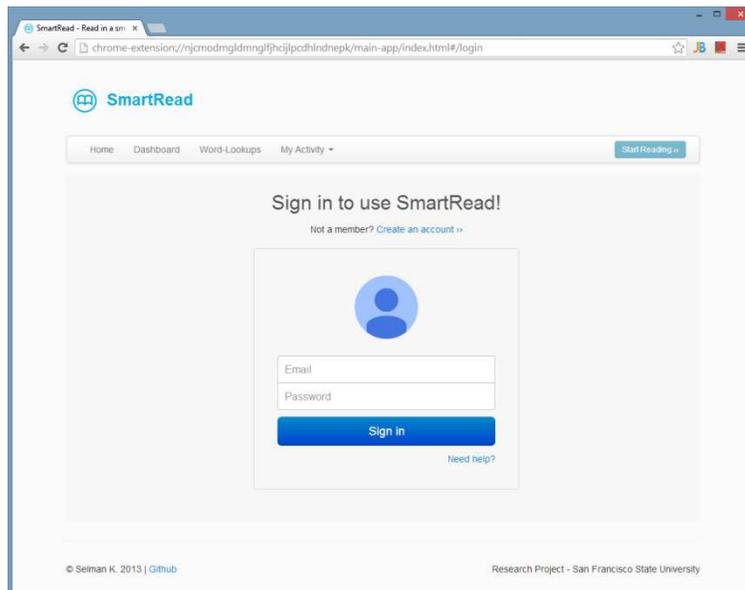
Chrome Application Home Page



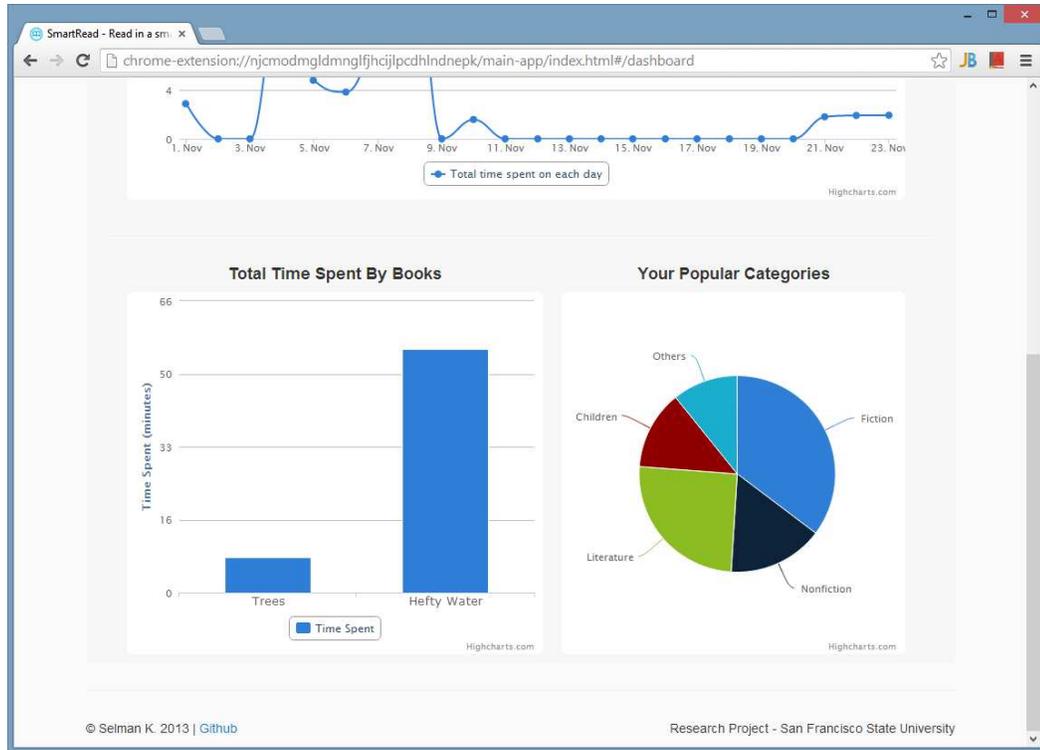
E-Book Reading Module



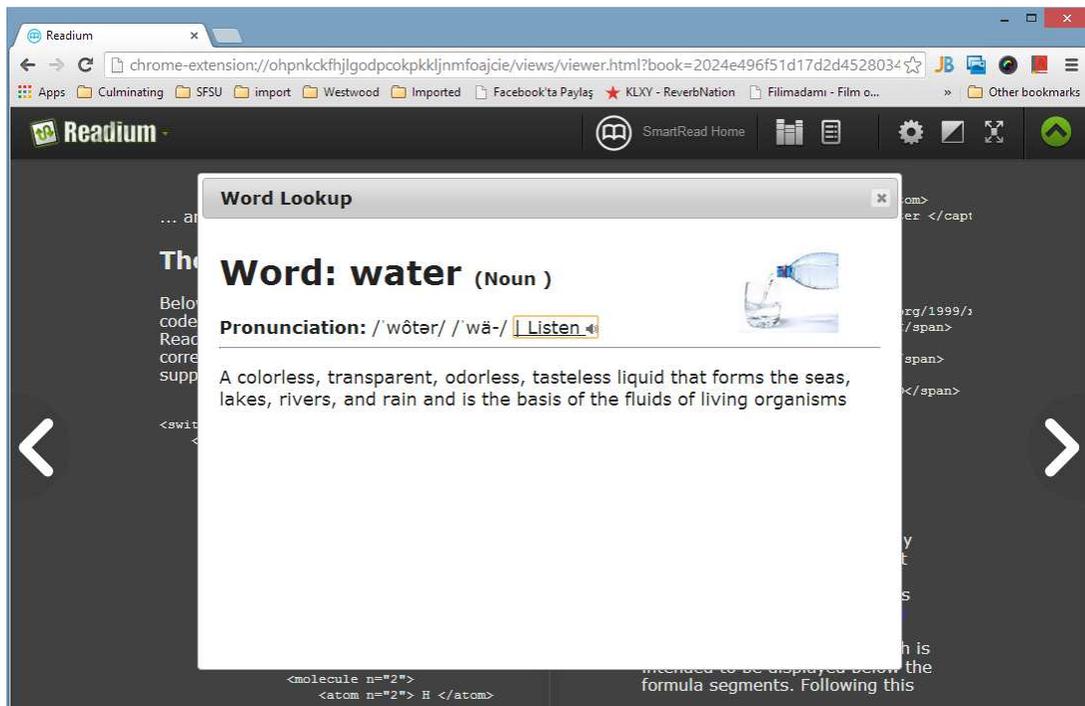
Signup Page



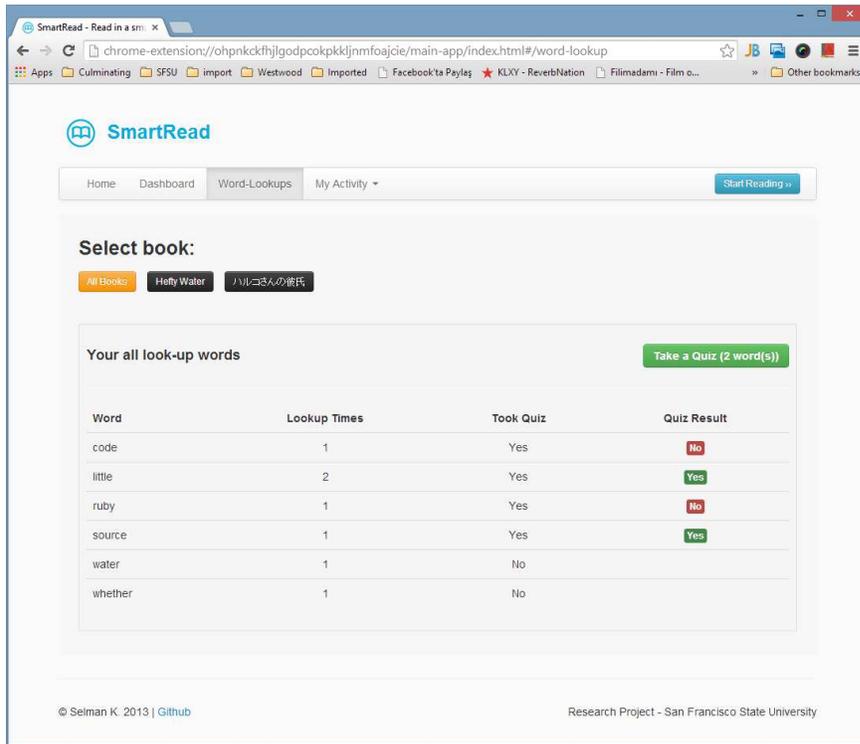
Login Page



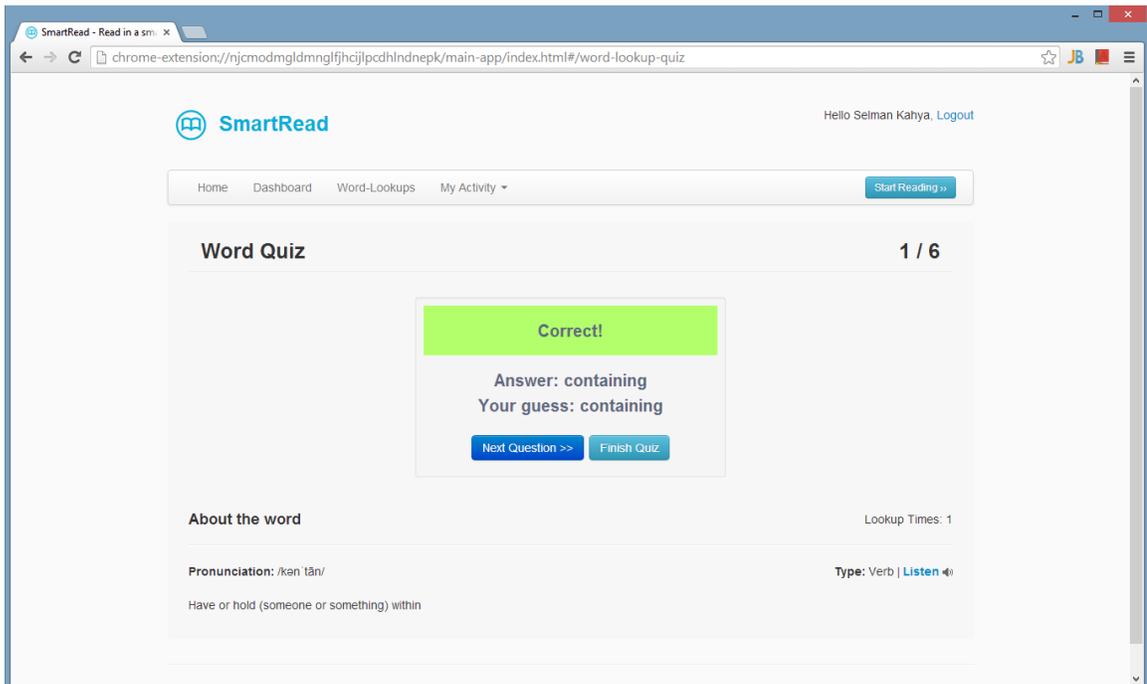
User Dashboard



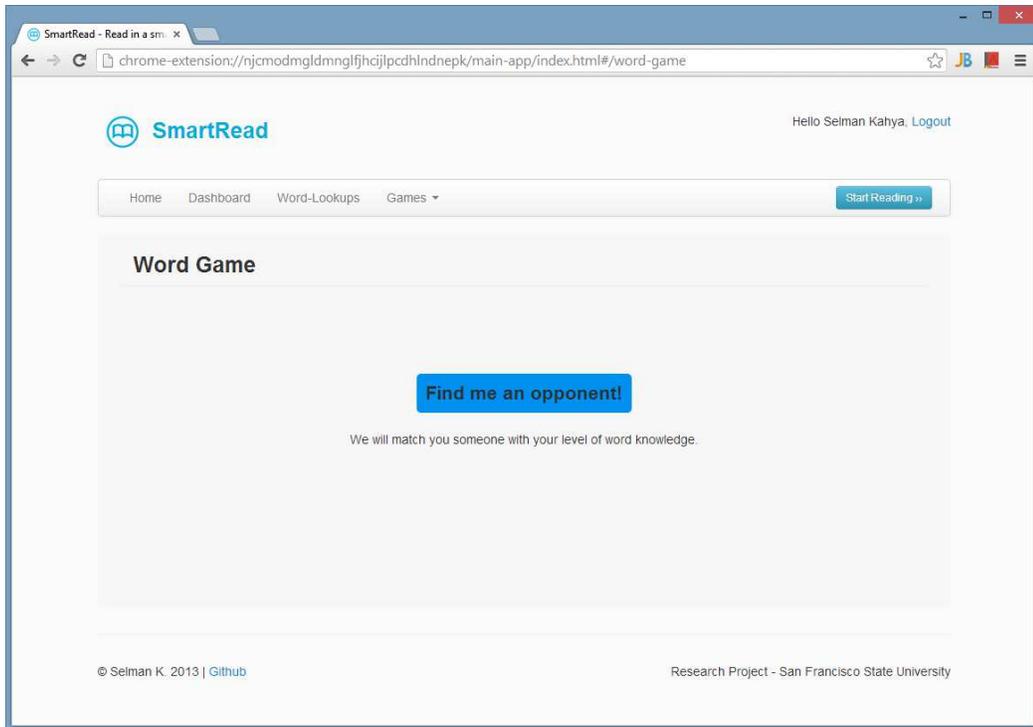
Word-lookup while reading (after a double click on a word)



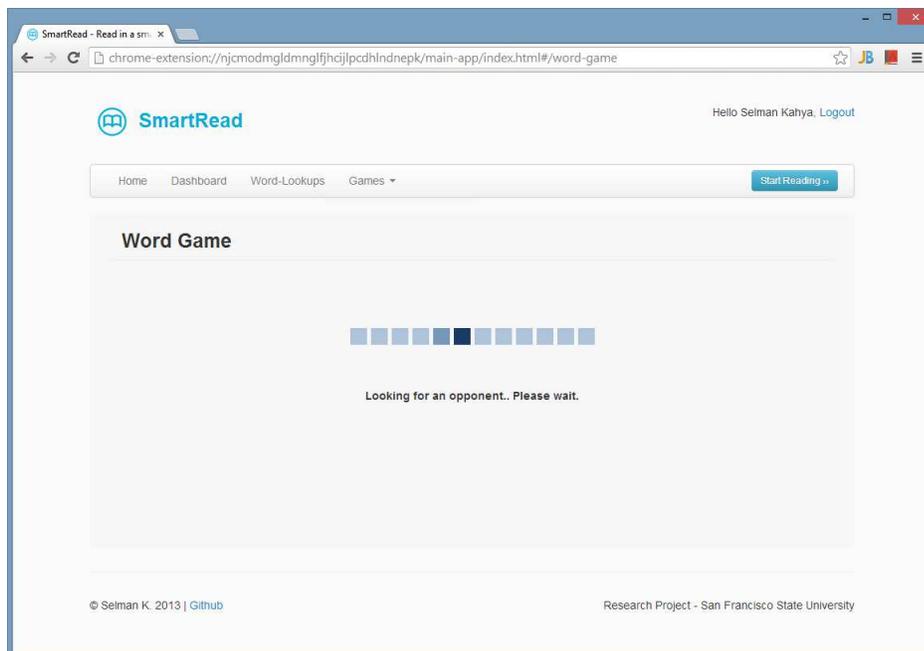
Word Lookup Details Page



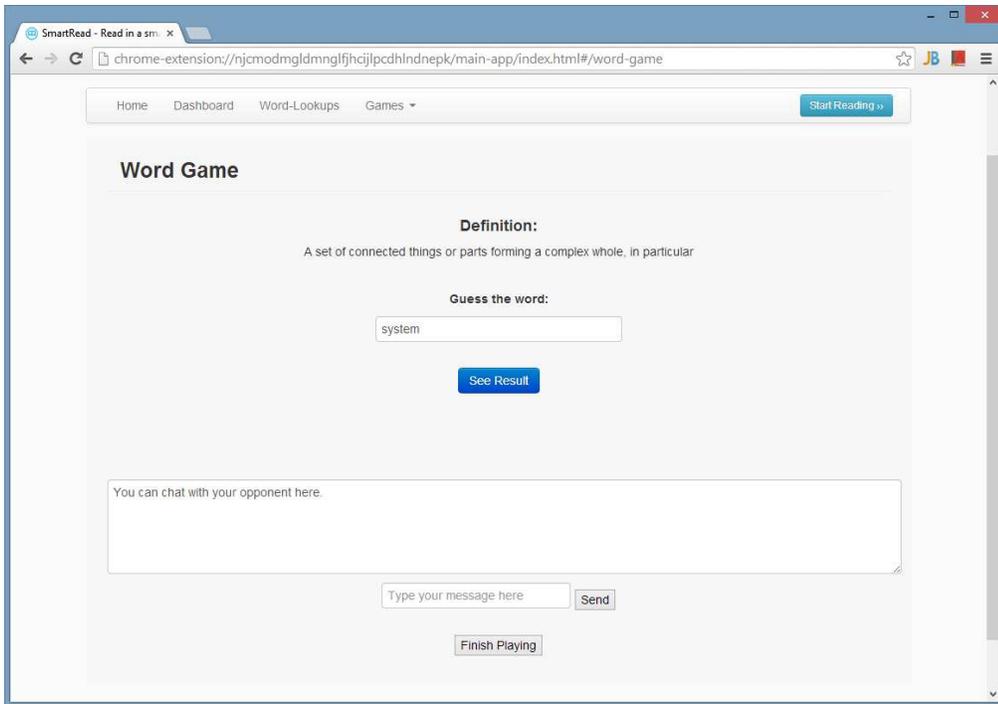
Playing Word Quiz (single player)



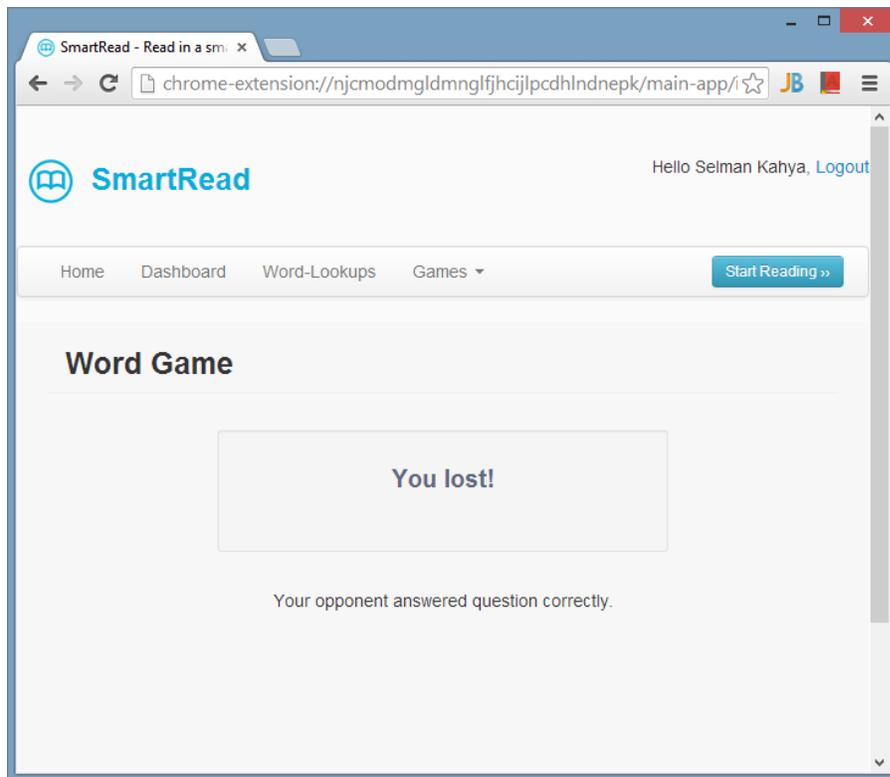
Playing Word Game (multi-player)



Looking for an opponent to play word game (multi-player)



Game started (multi-player)



Opponent won! (multi-player)

Chapter 5

5. CONCLUSION

Our plan for this semester was to create a platform/system that will allow us to present a showcase of tracking and presenting user reading activity by creating new services based on the assessment.

Our major goals were:

- Find or create a platform to track users' e-book reading activity.
- Provide word-lookup functionality.
- Offer games using user's reading level and activity to increase reading comprehension.
- Present activity data to the user in a proper way (using graphs etc.)

At the end of the semester, our application has had major functionality added to create a simple showcase as planned.

5.1 NEXT SEMESTER PLAN - FUTURE DIRECTIONS

After this point, we are planning to focus more on the services that we can offer using the data we collect while user is reading an e-book. Our concentration will be on high school students. Adding interactive features such as multi-player word games is also being considered.

6. REFERENCES

- [1] NodeJS - <http://nodejs.org/>
- [2] ExpressJS - <http://expressjs.com/>
- [3] SockJS-node - <https://github.com/sockjs/sockjs-node>
- [4] jQuery - <http://jquery.com/>
- [5] SockJS-client - <https://github.com/sockjs/sockjs-client>