VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Klim Skuridin

SPACED REPETITION CLOUD WEBAPP IN JAVA EE AND GOOGLE MATERIAL DESIGN

Information Technology
2016

VAASAN AMMATTIKORKEAKOULU
VAASA UNIVERSITY OF APPLIED SCIENCES
Degree Programme in Information Technology

ABSTRACT

Author	Klim Skuridin
Title	Spaced Repetition Cloud WebApp in Java EE and Google Material Design
Year	2016
Language	English
Pages	42
Name of Supervisor	Pirjo Prosi

The aim of this thesis project was to develop a web application using Java with deployment on Amazon Web Services cloud platform within an Elastic Computing 2 VM instance. The developer aims to help users learn new information using the Spaced Repetition principle. The main tools of the project include Java/JSP for back-end development, Twitter Bootstrap framework for front-end development, GitHub Version Control System for code management, MySQL for app data persistence, Tomcat server for deployment and Eclipse IDE for coding the back-end.

Keywords: Java, WebApp, AWS, Spaced Repetition, Bootstrap, GMD

CONTENTS

LIST OF FIGURES AND TABLES

1. INTRODUCTION

The choice of the thesis paper topic stems from a personal need for an application that would serve as a personal ‘knowledge vault’, holding all of the bits and pieces of information such as vocabulary and terms, as well as providing a mechanism that would help in remembering them long-term. No ready-made solution that could fulfill all of these requirements has been found, with some apps being quite similar, yet not exactly what is needed.

The application is based on a Spaced Repetition principle that is widely used in education and has proven to be rather effective. The information is stored and presented in the form of flashcards and displayed to the user in a number of predefined and exponentially-increasing time intervals, allowing for an optimal chance of long-term recall.

The application’s main technology stack is based on Java EE (Enterprise Edition), involving the use of Servlets and JSP (Java Server Pages). The program is deployed on an Apache Tomcat server within an AWS (Amazon Web Services) EC2 (Elastic Computing) VM (Virtual Machine) instance. The UI (User Interface) part is implemented using the Twitter Bootstrap framework and jQuery/AJAX for communicating with the back-end. The data is stored using a MySQL relational database, running on the same EC2 VM instance as the app server.

The thesis paper is based on three main sections. The first section explains the idea and the background of the project, including the main algorithm and the choice of technologies. The second section deals with the analysis stage, such as analyzing requirements, creating UI sketches, designing a database, a back-end structure and drawing all of the relevant UML diagrams. The third section describes the actual implementation, including the UI and server-side coding, as well as creating a database, setting up a server and manual testing.

The thesis software project is open-source, freely available in GitHub and licensed under the GNU General Public License Version 3, allowing commercial use, distribution, modification, patent use and private use; the program is required to have its source code disclosed, contain a copyright and license notice, any derivatives must be provided using the same license and provide information regarding state changes; the program author cannot be held liable for any and all issues that might occur from its use. The full text of the license is available in the project repository under the name LICENSE.md and on the gnu.org website.

2. PROJECT BACKGROUND & DESCRIPTION

2.1 Project Description

Spaced Repetition App is a program which makes remembering things easy.

The Spaced Repetition System (SRS) is a presentation method that gives the user the information before (s)he would forget it and makes sure that it stays constantly fresh in his/her mind. SRS utilized the Spaced Repetition Technique that incorporates increasing intervals of time between subsequent review of previously learned material in order to exploit the psychological spacing effect /3/.

Anyone who needs to remember things in their daily life can benefit from such a program. Because it is a lot more efficient than traditional study methods, the user can either greatly decrease time spent studying, or greatly increase the amount that has been learnt.

Possible applications include:

- Serving as a ‘knowledge vault’
- Learning new languages
- Memorizing new terms and concepts
- Expanding vocabulary

The biggest problem with learning new things is the retention rate: most of the things that people learn, they end up forgetting soon after they come across them.

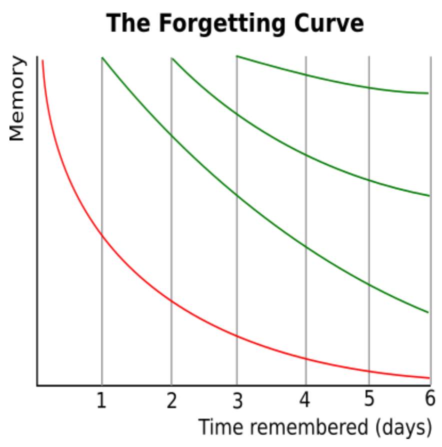


Figure 1: The Forgetting Curve

The Spaced Repetition System solves this problem by presenting the users with a possibility to periodically revise every bit of information that they would like to remember long-term.

The SRS idea is not new and there are similar solutions already available with the most popular ones being Mnemosyne and Anki. The problem with most of them is the over-abundance of functionality that makes them not very practical to use, lack of a well-designed web interface and a rather complex and versatile approach that is supposed to address the needs of many users instead of focusing on a number of basic key functions.

The key principal that allows the SRS to achieve high material retention rates, is the consistency of repetition. And the biggest factor that helps in achieving consistency, is how practical the program is to use. Current solutions, unfortunately, are not practical enough to be of much use due to their complexity. The software being developed is supposed to address the issues observed with other similar programs.

One of the key purposes of the application is to serve as a 'vault' for all of the bits and pieces of information one might come across during life, in order to keep all of the data in a single place and have a mechanism of retaining it in one's own memory. Current solutions appear to be designed more for a specific narrower use case, e.g. preparing for an exam and studying a single topic of interest.

2.2 Application Algorithm

The basic idea of the application is to let the user input new bits of information, in the form of cards, that need to be revised later, present those cards periodically using an algorithm and let the user grade the level of retention of each presented piece of information, which, in turn, determines when this particular bit of data will be presented next.

The above-mentioned algorithm is best described by Leitner System, which the program's main concept is largely based upon.

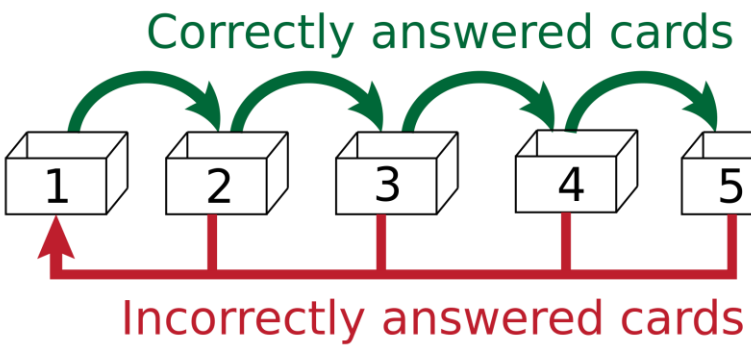


Figure 2: The Leitner System

The Leitner System is one of the simplest implementations of the SRS. The idea of this system is to sort the cards into several categories (or boxes). Each of these categories represents a time period according to which the included card appears to the user. Each card starts with the lowest group and gets promoted to the higher one based on how well the user is able to recall its contents /4/.

For example, a system with five boxes would mean that the cards are sorted into five groups, with each of these groups meaning: 1 – one day, 2 – three days, 3 – one week, 4 – one month, 5 – six months. This means that when a user adds a new card, it gets added to box #1 and is displayed to the user the next day. If a user is able to recall the card’s contents correctly the following day, then the card gets promoted to box #2 and is then displayed to the user again three days later. Then the process repeats and the card can either be promoted to box #3 or demoted. All of the cards get demoted to box #1.

The current implementation of this system is designed to have eight boxes that should correspond to the following time intervals: 1 – one day, 2 – two days, 3 – one week, 4 – two weeks, 5 – one month, 6 – two months, 7 – six months, 8 – twelve months.

2.3 Relevant Technologies

The primary version of the program is supposed to be a Web App. This means that it should be used through a browser and not as a native OS app. The reason for this is the need to be able to use the program on any device. Creating native apps for each OS would significantly increase both the complexity and the time required for the app to be developed.

Java EE & JSP are the back-bone technology behind the application's logic. Java EE is a server-side technology responsible for processing requests that are sent by front-end AJAX scripts, working with the database and forwarding the data back to AJAX in JSON format with the help of the Google Gson library. Java EE works as a combination of Java Servlets and Java Server Pages with the latter being used on the front-end side, as part of the .jsp pages.

AJAX & jQuery. The necessity for adding these technologies comes from the fact of Java EE being a server-side technology. As such, Java EE is not capable of changing the page's contents dynamically, without page refreshes, once they have already been sent to the user from the server. Having to refresh the page would be a significant flaw from a UI/UX perspective. AJAX allows for asynchronous communications between the front-end pages and servlets. jQuery is the most widely used technology for utilizing AJAX.

Twitter Bootstrap is the most widely used front-end framework that is currently available. Other alternatives include, for example, Zurb Foundation, Semantic UI, Skeleton etc.

Apache Tomcat. Used as a main application server. Runs within an EC2 VM instance. Tomcat was the most obvious choice due to its popularity and previous experience of working with it. Other possible options are JBoss, Jetty, Simperium etc.

MySQL is used for storing all of the app's data such as cards and users information, except for the session data, which is stored as a HttpSession Java object in Tomcat's memory. MySQL server runs within the EC2 VM instance. MySQL is the most widely used industry solution for running a relational database.

Amazon Web Services are used for deploying the app within an EC2 VM instance. The EC2 instance is bound to a personal domain name through Route 53. The main reasons for choosing this platform is its broad community support due to its high popularity, as well as the availability of the Free Tier option that lets users use the platform for one year at a discounted price. Possible other alternatives include Windows Azure and Google Cloud Apps.

GitHub is used as a Version Control System for storing all of the program code. The app repository is connected directly to the project in Eclipse and all of the changes can be committed to GitHub directly from IDE.

Eclipse, the Integrated Development Environment, is used for developing as well as testing during the development process.

3. ANALYSIS & DESIGN

3.1 Requirements

The scope of the project is to design a web application that allows a user to create, modify, browse, delete and grade cards based on the SRS algorithm.

Based on the project scope, the Use Cases Diagram shown in Figure 3 has been designed.

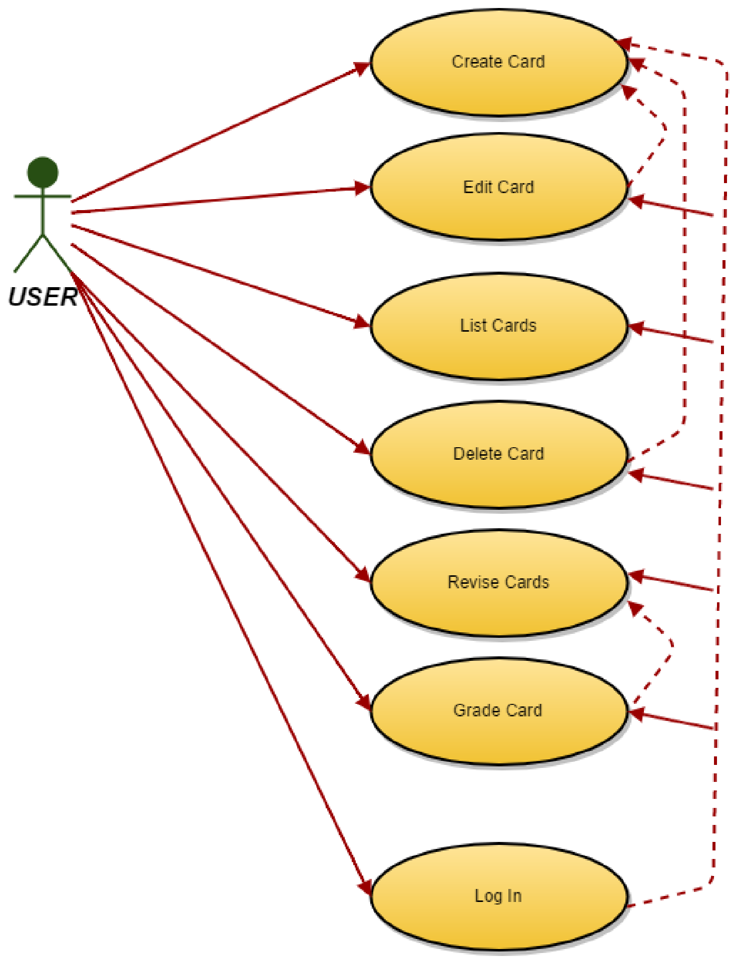


Figure 3: Use Cases Diagram

The ‘Log In’ use case aims to address the need for having more than one user logged in at the same time.

Figure 4 analyzes each of the use cases in detail.

‘Create Card’ Use Case	
Pre-conditions	The user is logged in
Input	Card front side and back side text
Description	Lets user create a new card
Exceptions and errors	Cannot connect to the database
Result or output	The card information is saved to the database
‘Edit Card’ Use Case	
Pre-conditions	The user is logged in The card already exists in the database
Input	Card front side and back side text
Description	Lets user modify existing card’s contents
Exceptions and errors	Cannot connect to the database
Result or output	The card information is saved to the database
‘List Cards’ Use Case	
Pre-conditions	The user is logged in
Input (Optional)	Keyword to search for
Description	Lets user list all cards or search for specific one(s) based on a keyword
Exceptions and errors	There are no cards in the database Cannot connect to the database
Result or output	Displays the list of cards as table that match the searching criteria
‘Delete Card’ Use Case	
Pre-conditions	The user is logged in The card must exist in the database
Input	-
Description	Lets user delete an existing card
Exceptions and errors	Cannot connect to the database
Result or output	The card is deleted from the database
‘Revise Cards’ Use Case	
Pre-conditions	The user is logged in At least one card is due for revision
Input	-
Description	Lets user start the revision process
Exceptions and errors	There are no cards to revise Cannot connect to the database
Result or output	Displays one of the cards that is due for revision
‘Grade Card’ Use Case	
Pre-conditions	The user is logged in
Input	Grading decision: either promote or demote
Description	Lets user grade one of the cards during the revision
Exceptions and errors	Cannot connect to the database
Result or output	The card’s level is modified and the next card is displayed

‘Log In’ Use Case	
Pre-conditions	-
Input	Username and password
Description	Lets user log in to the main app interface
Exceptions and errors	Username and password do not match Cannot connect to the database
Result or output	The user is forwarded to the main.jsp page.

Figure 4: Use Cases Description

3.2 Front-End

The first stage of the front-end design is to create a set of mockups with each of them representing a UI window, or a view, within a program.

A mockup is an image that shows what kind of UI elements each view will contain and how they will be placed. However, mockups are not supposed to represent the style and the aesthetic appearance of the app in every detail and are meant to show the overall abstract structure of the page only. The mockups are designed using Balsamiq, a software tool for sketching user interfaces.

The UI displayed on Figures 5-8 are designed for the desktop version, however, since the program uses a mobile-first Bootstrap front-end framework, the UI is automatically made compatible to work on devices with any screen size. Therefore, designing a separate set of sketches for mobile UI is not necessary.

The UI design prioritizes usability and is done with the Good UI principles is mind.

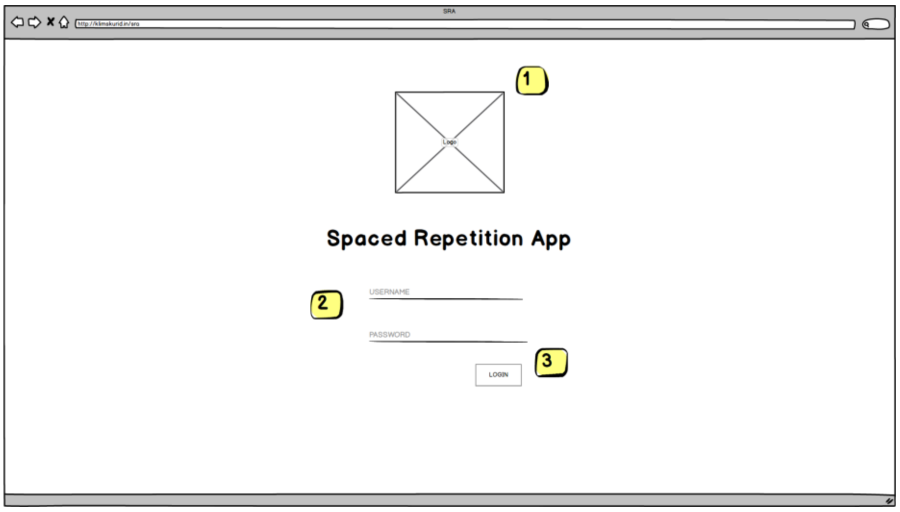
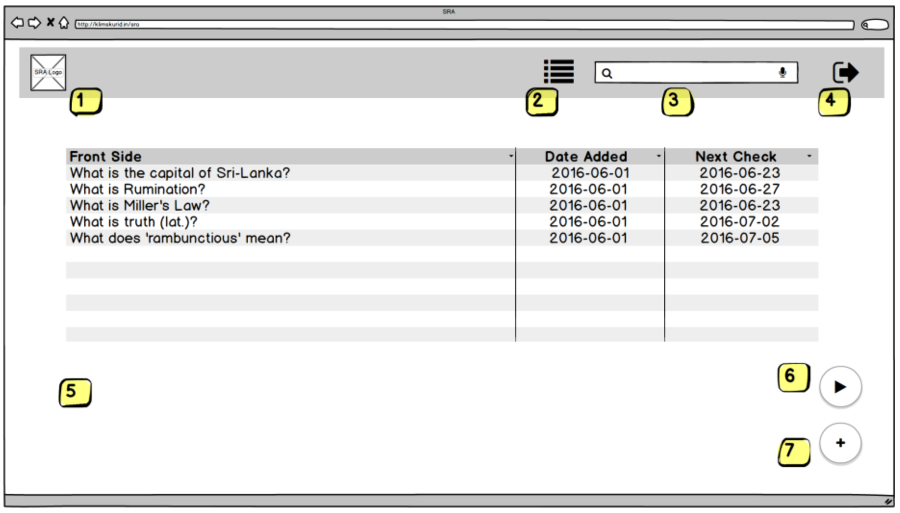


Figure 5: Login UI Mockup

1. Software Logo. The mockup is using a generic placeholder for this purpose. The actual app should contain a real logo.
2. Credentials Input Form. Form for inputting user credentials: username and password.
3. Login Button.



Front Side	Date Added	Next Check
What is the capital of Sri-Lanka?	2016-06-01	2016-06-23
What is Rumination?	2016-06-01	2016-06-27
What is Miller's Law?	2016-06-01	2016-06-23
What is truth (lat.)?	2016-06-01	2016-07-02
What does 'rambunctious' mean?	2016-06-01	2016-07-05

Figure 6: List Cards UI Mockup

1. Small Software Logo.
2. List Cards Icon. By pressing this icon, a user can list all of the cards from the database as a table.
3. Search Bar. The search bar, essentially, does the function as the previous icon, but it lets user to list the cards pre-filtered based on the search keyword.
4. Log Out Icon. Logs user out of the main UI and displays the Login UI.
5. Cards List. A table/list for displaying cards information.
6. Play Floating Action Button (FAB). Starts the revision process by displaying the Revise Cards UI.
7. New Card FAB. Adds a new card by displaying the Create/Edit Card UI.

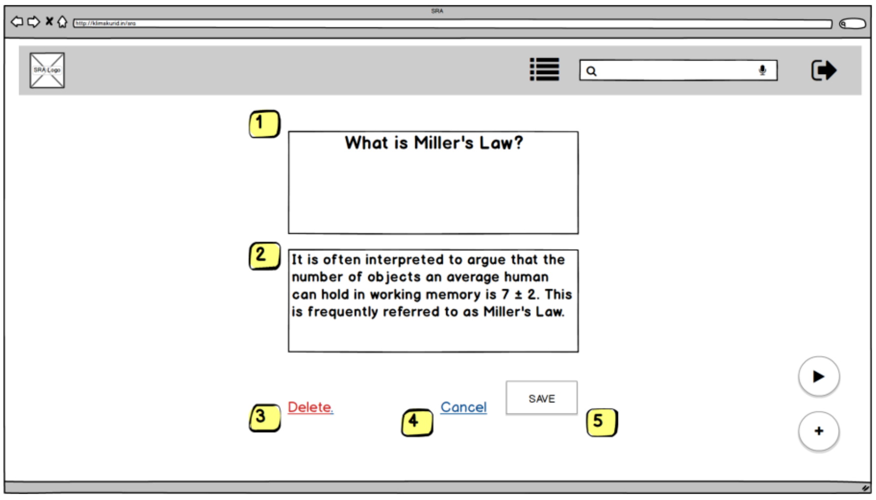

SRA
SRA Logo
1
What is Miller's Law?
2
It is often interpreted to argue that the number of objects an average human can hold in working memory is 7 ± 2. This is frequently referred to as Miller's Law.
3
Delete
4
Cancel
SAVE
5

Figure 7: Create/Edit Card UI Mockup

1. Card Front Side. Displays the text from the front side of the card, e.g. a question or a term to be recalled.
2. Card Back Side. Displays the text from the back side, e.g. an answer or a definition of a term.
3. Delete Link. Deletes the card from the database. The link is displayed on the left side with some distance from other links and buttons.
4. Cancel Link. Cancels the editing or creating the card process and forwards to the List UI View.
5. Save Button. Saves the changes to the database.

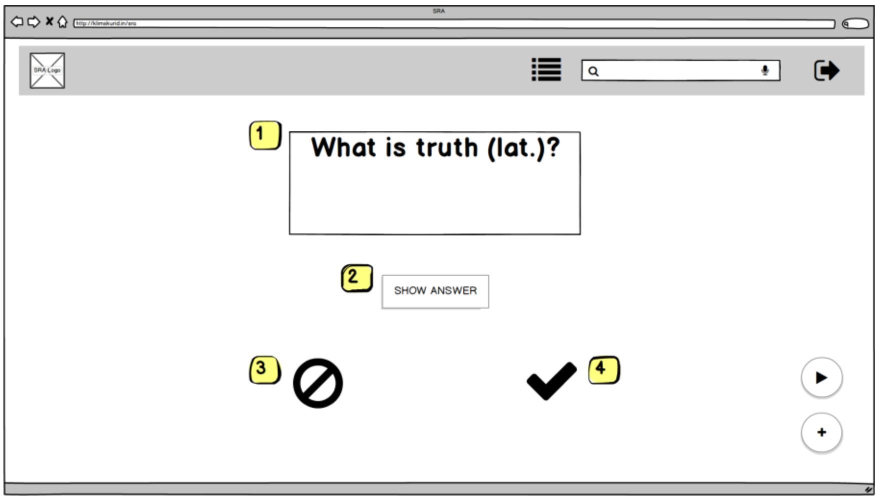


Figure 8: Revise Cards UI Mockup

1. Card Front Side. Displays the text from the front side of the card, e.g. a question or a term to be recalled.
2. Show Answer Button. When pressed, the button is hidden and the Card Back Side is displayed instead.
3. Demote Card Button. When pressed, the card's level decreases. Automatically switches to the next card after being pressed. The button only appears after the Show Answer Button has been pressed.
4. Promote Card Button. When pressed, the card's level increases. Automatically switches to the next card after being pressed. The button only appears after the Show Answer Button has been pressed.

Based on the above-mentioned design, the final list of front-end related files should be as follows. The list does not include third-party scripts, CSS and libraries, e.g. jQuery, besides Bootstrap.

- login.jsp
- main.jsp
- list.jsp
- revise.jsp
- card.jsp
- main.css
- main.js
- bootstrap.css
- bootstrap.js

3.3 Database

Based on the requirements of the application, it has been decided to use a single database with two tables to store all of the cards and user information for all users. According to the MySQL Development Reference, a single table can store up to one billion rows and up to 4TB of data on a Linux ext3 filesystem /1/. This means that there should not be any issues when storing a large amount of cards, for multiple users, in a single table.

Another possible approach would be to use a separate table for each user's cards, however, that would mean having a high number of tables, with each table involving one to three files on the filesystem. This would be a poorer choice for performance reasons /2/.

According to the design, the passwords of all users are stored in the database as plain text. For obvious reasons, such approach does not offer much security in case the database is compromised. A better approach would be to implement hash functions. However, doing so would require extensive work that goes well beyond the scope and the educational nature of this project. Therefore, it has been decided to opt-out for a more straightforward and less secure option of storing passwords as plain text. A commercial application being used by multiple users would be strongly advised to implement the more secure approach.

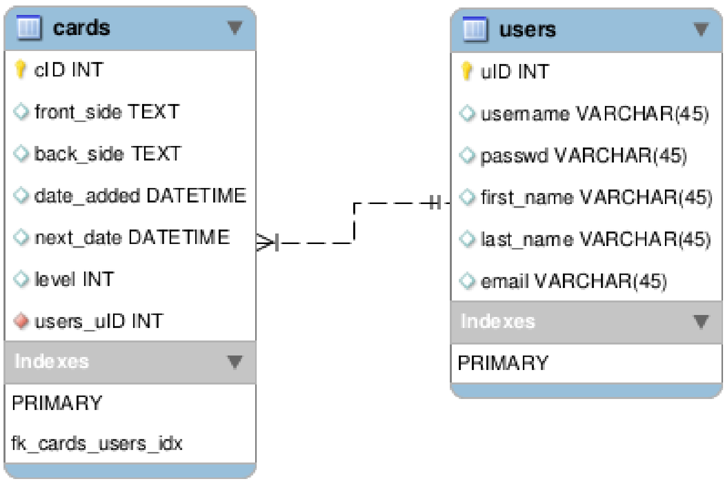


Figure 9: EER Diagram

The Cards table contains six columns and is meant to store all of the information related to cards and the User table has six columns and stores the user info. There is a one-to-many connection between the User and the Card tables, which means that the Cards table also contains a Foreign Key which determines the owner of each particular card using the User ID.

3.4 Back-End

The main working principle of the app's back-end is to use AJAX to make calls from JSP pages to servlets, which connect to the database and retrieve or send the data. The data is then sent back to AJAX from the Servlet in the form of JSON using the Google Gson library. The following diagram in Figure 10 illustrates the main idea.

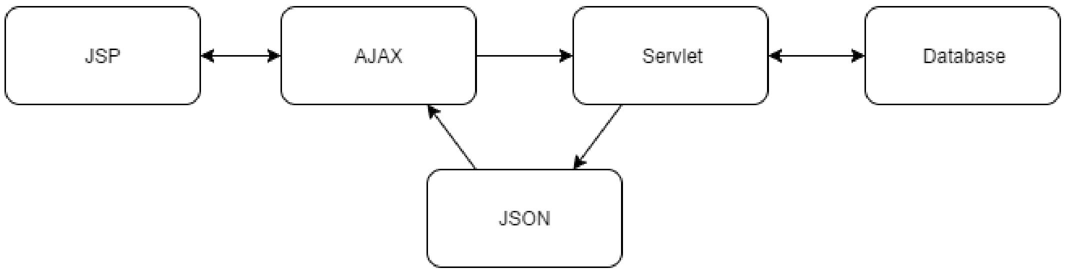


Figure 10: Back-End Algorithm Diagram

The structure of the back-end code is shown in Figure 11:

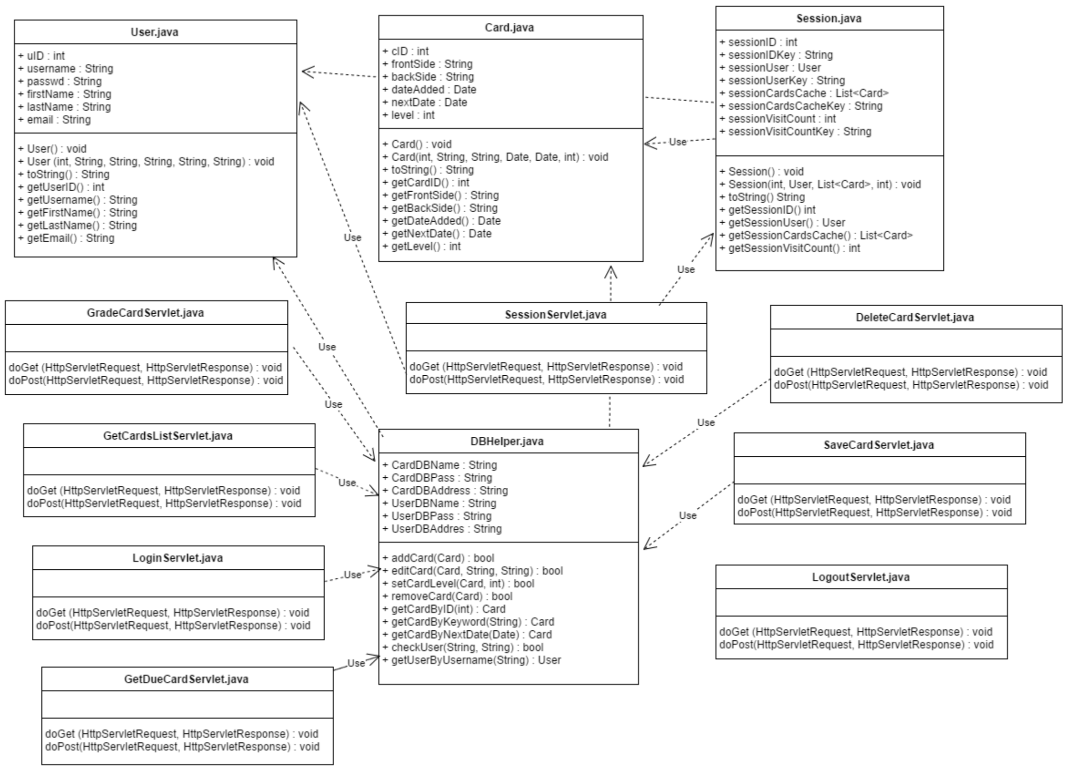

User.java
+ uID : int
+ username : String
+ passwd : String
+ firstName : String
+ lastName : String
+ email : String
+ User() : void
+ User (int, String, String, String, String, String) : void
+ toString() : String
+ getUserID() : int
+ getUsername() : String
+ getFirstName() : String
+ getLastName() : String
+ getEmail() : String
Card.java
+ cID : int
+ frontSide : String
+ backSide : String
+ dateAdded : Date
+ nextDate : Date
+ level : int
+ Card() : void
+ Card(int, String, String, Date, Date, int) : void
+ toString() : String
+ getCardID() : int
+ getFrontSide() : String
+ getBackSide() : String
+ getDateAdded() : Date
+ getNextDate() : Date
+ getLevel() : int
Session.java
+ sessionID : int
+ sessionIDKey : String
+ sessionUser : User
+ sessionUserKey : String
+ sessionCardsCache : List<Card>
+ sessionCardsCacheKey : String
+ sessionVisitCount : int
+ sessionVisitCountKey : String
+ Session() : void
+ Session(int, User, List<Card>, int) : void
+ toString() String
+ getSessionID() int
+ getSessionUser() : User
+ getSessionCardsCache() : List<Card>
+ getSessionVisitCount() : int
GradeCardServlet.java
doGet (HttpServletRequest, HttpServletResponse) : void
doPost(HttpServletRequest, HttpServletResponse) : void
SessionServlet.java
doGet (HttpServletRequest, HttpServletResponse) : void
doPost(HttpServletRequest, HttpServletResponse) : void
DeleteCardServlet.java
doGet (HttpServletRequest, HttpServletResponse) : void
doPost(HttpServletRequest, HttpServletResponse) : void
GetCardsListServlet.java
doGet (HttpServletRequest, HttpServletResponse) : void
doPost(HttpServletRequest, HttpServletResponse) : void
DBHelper.java
+ CardDBName : String
+ CardDBPass : String
+ CardDBAddress : String
+ UserDBName : String
+ UserDBPass : String
+ UserDBAddres : String
+ addCard(Card) : bool
+ editCard(Card, String, String) : bool
+ setCardLevel(Card, int) : bool
+ removeCard(Card) : bool
+ getCardByID(int) : Card
+ getCardByKeyword(String) : Card
+ getCardByNextDate(Date) : Card
+ checkUser(String, String) : bool
+ getUserByUsername(String) : User
SaveCardServlet.java
doGet (HttpServletRequest, HttpServletResponse) : void
doPost(HttpServletRequest, HttpServletResponse) : void
LoginServlet.java
doGet (HttpServletRequest, HttpServletResponse) : void
doPost(HttpServletRequest, HttpServletResponse) : void
LogoutServlet.java
doGet (HttpServletRequest, HttpServletResponse) : void
doPost(HttpServletRequest, HttpServletResponse) : void
GetDueCardServlet.java
doGet (HttpServletRequest, HttpServletResponse) : void
doPost(HttpServletRequest, HttpServletResponse) : void
Use

Figure 11: Java Classes Diagram

Except for the servlets, there are a total of four Java classes. The Card and User classes represent the same structure as the corresponding tables in the database, in order to make development easier.

The DBHelper class contains all of the methods for working with the database, for both the User and the Card classes and tables.

The SessionServlet.java class is for tracking the user sessions using the HttpSession object.

Card.java, User.java & Session.java files contain all of the attributes, basic constructors, getters and toString() methods for their respective classes.

Based on all of the above, the final list of back-end related files should be as follows:

- Card.java
- User.java
- Session.java
- DBHelper.java
- SessionServlet.java
- GradeCardServlet.java
- GetCardsListServlet.java
- LoginServlet.java
- GetDueCardServlet.java
- DeleteCardServlet.java
- SaveCardServlet.java
- LogoutServlet.java
- web.xml (Deployment Descriptor)

3.5 Server & Deployment

The application is deployed on the Amazon Web Services cloud platform, within an Elastic Computing 2 Virtual Machine running Linux Ubuntu x64 14.04 LTS version.

The VM contains the App Server, Tomcat 7, as well as the MySQL Server for data persistence. The clients connect to the VM through an internet connection from the PC or mobile devices.

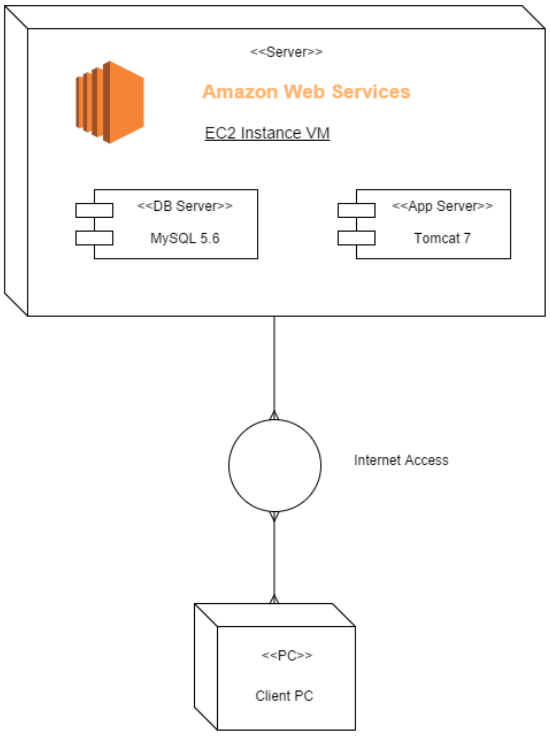


Figure 12: Deployment Diagram

The application is developed using Java, which means that the code is executed on the server and not on the client machine(s).

4. IMPLEMENTATION & DEPLOYMENT

The project is developed using a local VirtualBox Virtual Machine running Ubuntu Linux 14.04 LTS. All of the server versions, including Tomcat and MySQL are set up to be precisely the same as on the deployment VM. Therefore, the development and the production environments are configured to be as similar as possible, in order to avoid any compatibility and configuration-related issues during the development process.

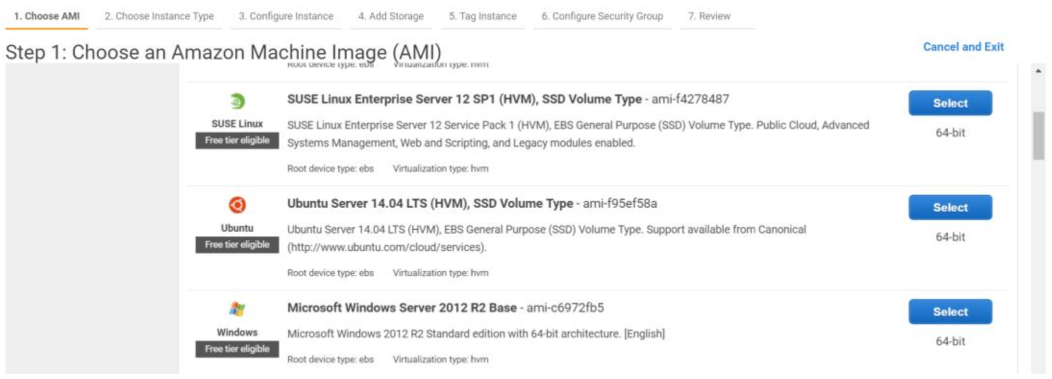
4.1 Server Environment

There are two AWS services that are planned to be used within this project. The first one is Route 53, which acts as a DNS server and stores the records that point to an internal EC2 IP address. The second one is EC2 itself, which acts as the main server for the app.

The first step in setting up the server environment is to sign up for an Amazon Web Services account. All of the newly-created accounts are eligible for a Free Tier program, allowing new users to use many of the AWS services at a discounted price during the first year. EC2 service is included in the program, therefore, the costs of setting and running the VM are very low.

After signing up, it is considered a good practice to set up billing alarms and Identity and Access management (IAM) account(s). The billing alarms are necessary so that there would be no surprise charges, since AWS bills users on a per-hour basis, depending on how much processing power they are using. The IAM accounts act as a security mechanism that allows for a fine-tuned system of users, roles and access policies. It is considered insecure to use the main root account for accessing the AWS resources and the use of IAM accounts is strongly encouraged by the AWS development guidelines /5/.

The creation of an EC2 VM is done through an AWS web console. The process is rather straightforward with the console wizard guiding the user through the whole process. It is also necessary to create a Security Group and have an Elastic IP assigned during the process. The Elastic IP is then used in order to bound the developer's personal domain name to an EC2 instance in Route 53 /8/. After creating the instance, a key-pair is generated and downloaded locally. It is then possible to connect to the VM through SSH using the given key. For the needs of this project, the smallest possible 'micro' server instance has been chosen. The server is located in the eu-west-1 region, Ireland /6/.



Step 1: Choose an Amazon Machine Image (AMI)

SUSE Linux Enterprise Server 12 SP1 (HVM), SSD Volume Type - ami-f4278487
SUSE Linux Enterprise Server 12 Service Pack 1 (HVM), EBS General Purpose (SSD) Volume Type. Public Cloud, Advanced Systems Management, Web and Scripting, and Legacy modules enabled.
Root device type: ebs Virtualization type: hvm

Ubuntu Server 14.04 LTS (HVM), SSD Volume Type - ami-f95ef58a
Ubuntu Server 14.04 LTS (HVM), EBS General Purpose (SSD) Volume Type. Support available from Canonical (http://www.ubuntu.com/cloud/services).
Root device type: ebs Virtualization type: hvm

Microsoft Windows Server 2012 R2 Base - ami-c6972fb5
Microsoft Windows 2012 R2 Standard edition with 64-bit architecture. [English]
Root device type: ebs Virtualization type: hvm

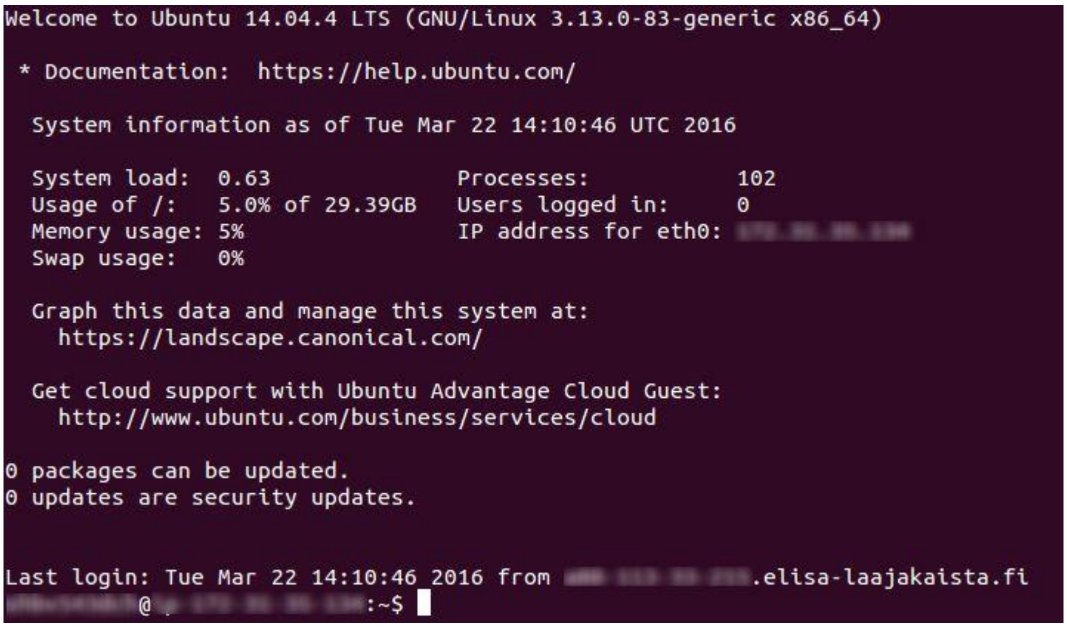
Figure 13: Creating EC2 Instance

After connecting to the VM, it is considered good practice to close all of the most-known security loopholes, such as the ability to connect as 'root' and having a 'root' password. It is also a good idea to change the default SSH port to something other than #22 /9/. Changing the port requires adjusting the Security Group rules to allow the incoming TCP traffic on that port number instead of #22. One other good security measure would be to change the username from the default 'ubuntu' to something else.

It is also advised to install and configure the AWS Command Line Tools for working with all of the AWS services remotely, in addition to the console. This allows for a greater degree of flexibility and options /7/.

After the SSH connection is made and secured, the server needs to be updated through the ‘apt-get update’, ‘apt-get upgrade’ and ‘apt-get dist-upgrade’ commands. This updates all of the packages, as well as the OS kernel.

Files can be transferred between the server and the local machine using any of the FTP clients, such as FileZilla. The connection is made with using SFTP and the same key as with the SSH connection.



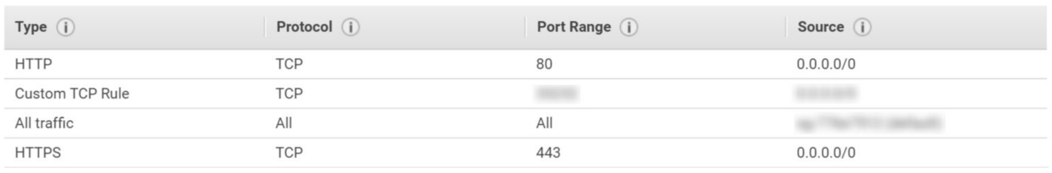
```
Welcome to Ubuntu 14.04.4 LTS (GNU/Linux 3.13.0-83-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

  System information as of Tue Mar 22 14:10:46 UTC 2016

  System load:  0.63              Processes:           102
  Usage of /:   5.0% of 29.39GB   Users logged in:     0
  Memory usage: 5%                IP address for eth0: [redacted]
  Swap usage:   0%

  Graph this data and manage this system at:
    https://landscape.canonical.com/

  Get cloud support with Ubuntu Advantage Cloud Guest:
    http://www.ubuntu.com/business/services/cloud

0 packages can be updated.
0 updates are security updates.


Last login: Tue Mar 22 14:10:46 2016 from [redacted].elisa-laajakaista.fi
[redacted]@[redacted]:~$ 
```

Figure 14: The Server is Updated

After updating the server, the first step is to setup the LEMP server stack. LEMP stands for Linux/nginx/MySQL/PHP. This is necessary in order to have access to the server through a web browser and be able to use web-based configuration tools, such as phpMyAdmin for MySQL DB configuration. The nginx is chosen over Apache due its superior performance on small servers such as the one used during this project /10/.

Here is a list of steps that are required in order to set up and configure all of the server components.

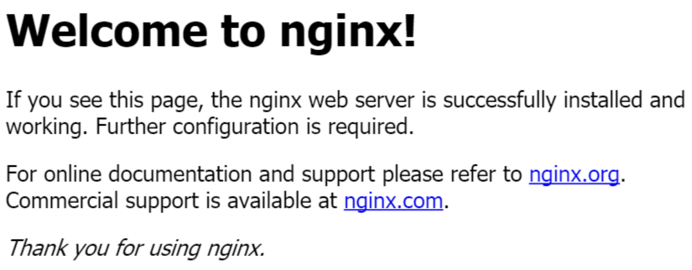
1. Installing nginx and modifying the '/etc/nginx/sites-available/default' server configuration /11/.
2. Installing MySQL server and running the 'mysql_secure_installation' script. It is advised to change the default MySQL port number for security reasons /11/.
3. Installing php and testing it with an info.php file. Configuring php.ini in order to enable the opcache module /11/.
4. Installing phpMyAdmin in order to administer the databases and users through the web interface. The 'root' account of phpMyAdmin should not be allowed to login from remote hosts. A separate user account should be created for this purpose /11/.
5. Configuring EC2 Security Group in order to allow inbound traffic on ports 80, 443, 3306 and 8080 for Tomcat /6/.



Type	Protocol	Port Range	Source
HTTP	TCP	80	0.0.0.0/0
Custom TCP Rule	TCP	[redacted]	[redacted]
All traffic	All	All	[redacted]
HTTPS	TCP	443	0.0.0.0/0

Figure 15: Configuring EC2 Security Group

The server can now be tested and should be working fine.



Welcome to nginx!

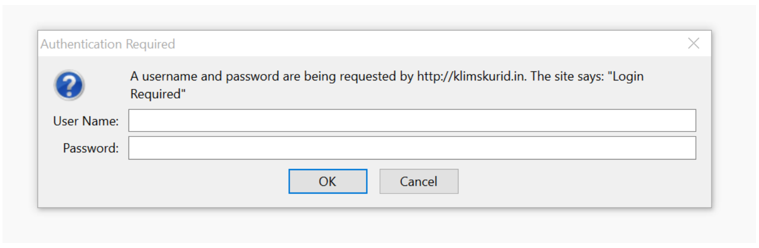
If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

Figure 16: Nginx is Ready

6. Improving security. For security reasons, it is best to rename all of the administration-related files and directories on the server, such as info.php, phpMyAdmin and any other there might be. It is also advised to protect these files and folders with a password through nginx basic auth /14/.



Authentication Required

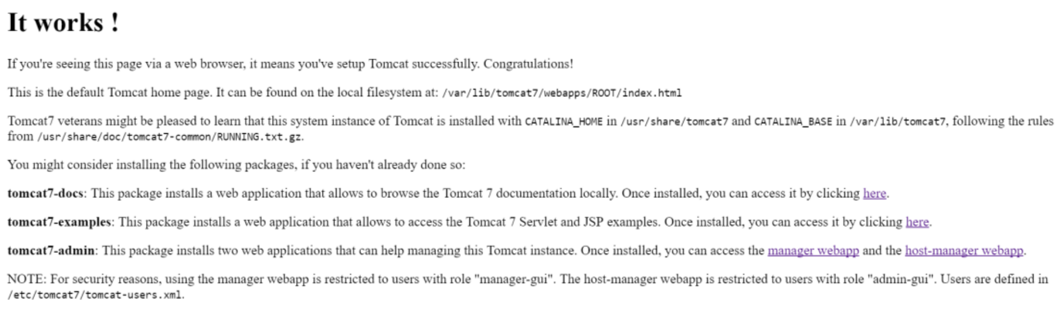
A username and password are being requested by http://klimskurid.in. The site says: "Login Required"

User Name:

Password:

Figure 17: Nginx Basic Auth

7. Installing Java and Tomcat. There are two Java installations available for Linux OS: Oracle Java and OpenJDK. The first one was used in this instance, for compatibility reasons since that is the one used on the development machine and in Eclipse IDE. Tomcat docs, examples and admin web apps need to be installed separately and can them be viewed using a web browser. The default port number is 8080 /12/.



It works !

If you're seeing this page via a web browser, it means you've setup Tomcat successfully. Congratulations!

This is the default Tomcat home page. It can be found on the local filesystem at: `/var/lib/tomcat7/webapps/ROOT/index.html`

Tomcat7 veterans might be pleased to learn that this system instance of Tomcat is installed with `CATALINA_HOME` in `/usr/share/tomcat7` and `CATALINA_BASE` in `/var/lib/tomcat7`, following the rules from `/usr/share/doc/tomcat7-common/RUNNING.txt.gz`.

You might consider installing the following packages, if you haven't already done so:

tomcat7-docs: This package installs a web application that allows to browse the Tomcat 7 documentation locally. Once installed, you can access it by clicking here.

tomcat7-examples: This package installs a web application that allows to access the Tomcat 7 Servlet and JSP examples. Once installed, you can access it by clicking here.

tomcat7-admin: This package installs two web applications that can help managing this Tomcat instance. Once installed, you can access the manager webapp and the host-manager webapp.

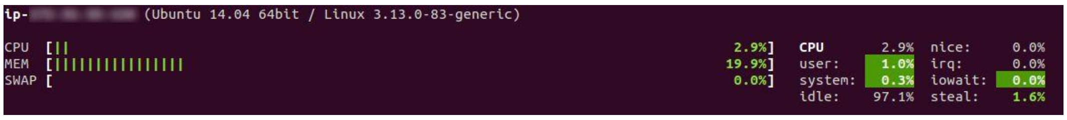
NOTE: For security reasons, using the manager webapp is restricted to users with role "manager-gui". The host-manager webapp is restricted to users with role "admin-gui". Users are defined in `/etc/tomcat7/tomcat-users.xml`.

Figure 18: Tomcat is Ready

8. Securing Tomcat. An admin username and password needs to configured in ‘/etc/tomcat7/tomcat-users.xml’ for accessing the management web apps. It is also advised to either delete or at least rename the docs and examples web apps /13/.

9. In order to access the MySQL server remotely, an admin user account needs to be created on the server. The account can then be tested from a MySQL Workbench on a local development machine. The connection is established through SSH /15/.
10. The last thing to do is to reboot the server and to create an EC2 volume snapshot for backup purposes /6/.

The whole configuration takes up rather small amounts of the server's processing power, both CPU and memory-wise.



```
ip-[redacted] (Ubuntu 14.04 64bit / Linux 3.13.0-83-generic)

CPU  [||                                                                                   2.9%]   CPU       2.9%  nice:     0.0%
MEM  [||||||||||||||||                                                                    19.9%]   user:     1.0%  irq:      0.0%
SWAP [                                                                                     0.0%]   system:   0.3%  iowait:   0.0%
                                                                                                   idle:    97.1%  steal:    1.6%
```

Figure 19: Server Resources Usage

The server is now configured and ready to be used.

4.2 Front-End

The key goal of designing the User Interface for this application is to offer a better alternative than a number of similar, already existing apps. This is achieved by following the Material Design (MD) concept developed by Google, as well as by making the UI as simplistic as possible, leaving out all of the non-essential functionality /21/.

The default version of Twitter Bootstrap /16/ is not based on MD, therefore a third-party framework, Bootstrap Material Design (BMD), is used. BMD is built on top of the default Bootstrap version and offers all of the same components and functionality, except with the improvement of making them MD-ready /17/.

The first thing to do is to create a file structure and a basic HTML template file that includes all of the required scripts, stylesheets and other auxiliary data. The scripts are put at the end of the file so that they would be loaded last and not break the loading of the page. The stylesheets are placed at the beginning. The file also includes a font and an icon set. The font is Roboto and the icons set is Material Icons. Both are provided by Google and are considered to be an industry-standard choice for many apps that are developed these days /21/.

The HTML template includes a <noscript> tag for detecting when JavaScript (JS) is turned off and, if it is, displays an appropriate message to the user, while hiding the rest of the app contents by adding a display:none CSS property to the <body> tag. The app relies heavily on jQuery /17/ for all of the UI-related scripting and cannot work properly without JS. The head section also detects if the browser is Internet Explorer version less than 10, and shows an appropriate message in that case. Optimizing the website for old browser takes a significant amount of time and it has been decided to simply drop their support since their user base accounts for a rather small percentage of all users.

The main HTML file contains a number of <div> containers that wrap each of the app’s views sections, making it easy to toggle these sections with JS, using button clicks and other event triggers /19/.

The first part to be developed is the login UI which is displayed to the user by default when loading the app. It contains two input fields, the login button and the app's logo and name.

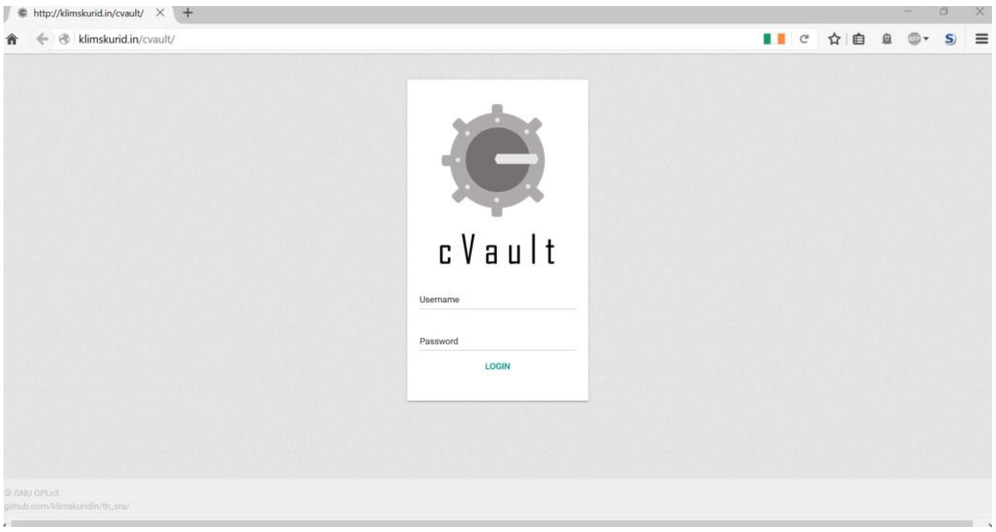
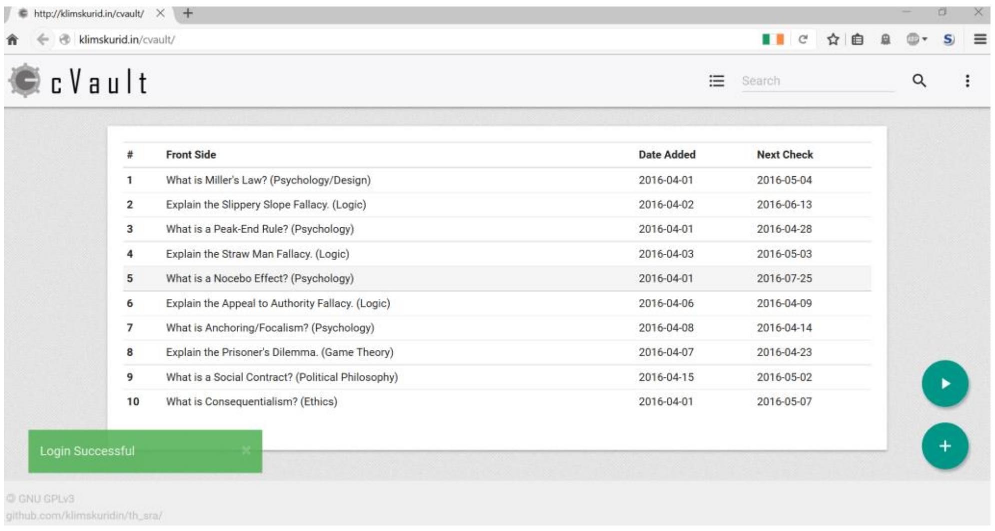


Figure 20: Login UI

The three main UI views consist of the navigation bar, FAB buttons and a footer. The default one is the List View, which displays the list of all the cards as a table, the second one is the Edit View that is meant for adding new cards and editing existing ones, and the third one is the Revise View, for revising and grading cards. The List View consists of a single card, while the other two contain two cards each.



#	Front Side	Date Added	Next Check
1	What is Miller's Law? (Psychology/Design)	2016-04-01	2016-05-04
2	Explain the Slippery Slope Fallacy. (Logic)	2016-04-02	2016-06-13
3	What is a Peak-End Rule? (Psychology)	2016-04-01	2016-04-28
4	Explain the Straw Man Fallacy. (Logic)	2016-04-03	2016-05-03
5	What is a Nocebo Effect? (Psychology)	2016-04-01	2016-07-25
6	Explain the Appeal to Authority Fallacy. (Logic)	2016-04-06	2016-04-09
7	What is Anchoring/Focalism? (Psychology)	2016-04-08	2016-04-14
8	Explain the Prisoner's Dilemma. (Game Theory)	2016-04-07	2016-04-23
9	What is a Social Contract? (Political Philosophy)	2016-04-15	2016-05-02
10	What is Consequentialism? (Ethics)	2016-04-01	2016-05-07

Figure 21: Main UI / List View

The FAB part needed to be created separately since the BMD, for some reason, does not contain this element /17/. The FAB buttons are created through CSS based on an example found on CodePen /18/. A BMD btn class is further added to the buttons to make them behave similarly to all other buttons.

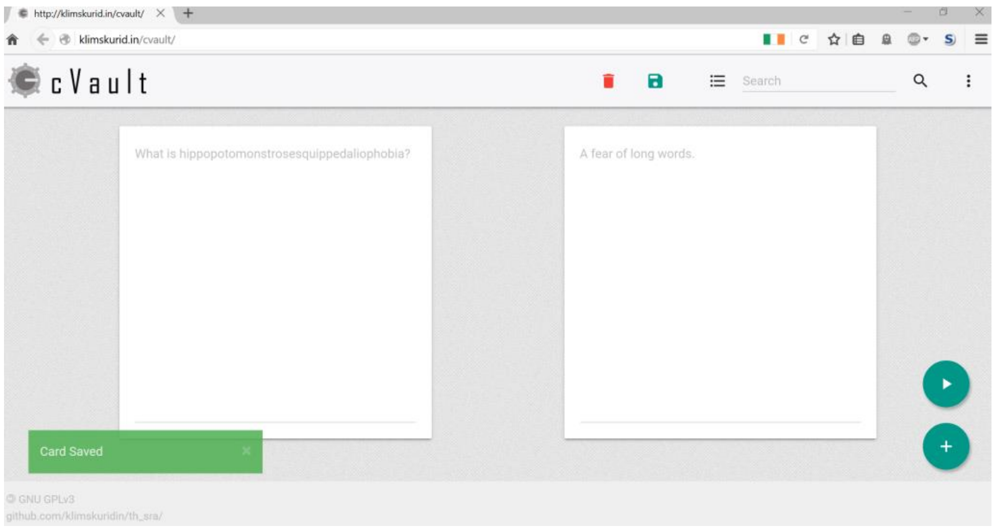

http://klimskurid.in/cvault/
klimskurid.in/cvault/
cVault
Search
What is hippopotomonstrosesquippedaliophobia?
A fear of long words.
Card Saved
© GNU GPLv3
github.com/klimskuridin/th_sra/

Figure 22: Main UI / Edit View

UI animation and transitional effects are created with the help of the Animate.css framework. The animations are called using jQuery functions bound to button click events /19/. Animate.css relies solely on CSS3 for creating all of the transitional effects /20/.

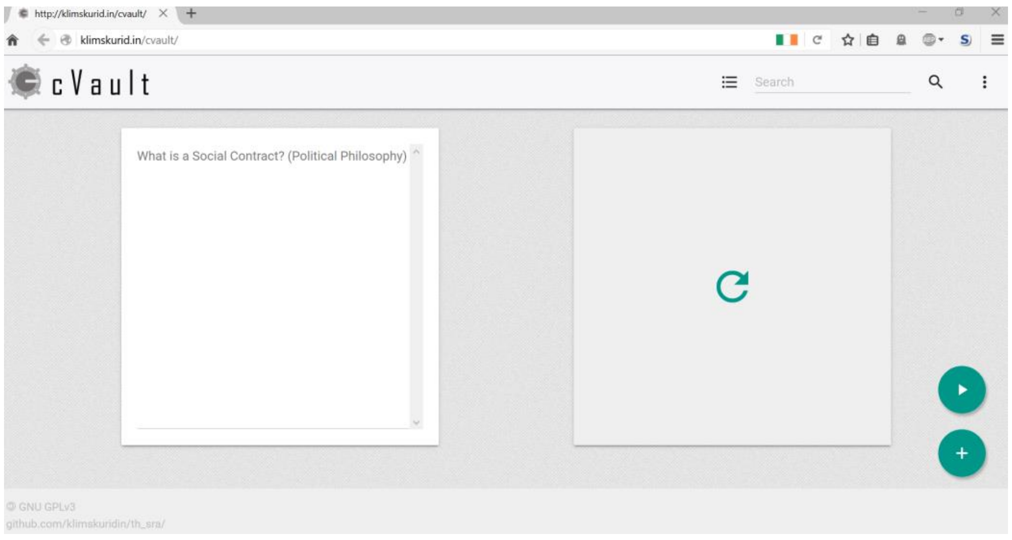
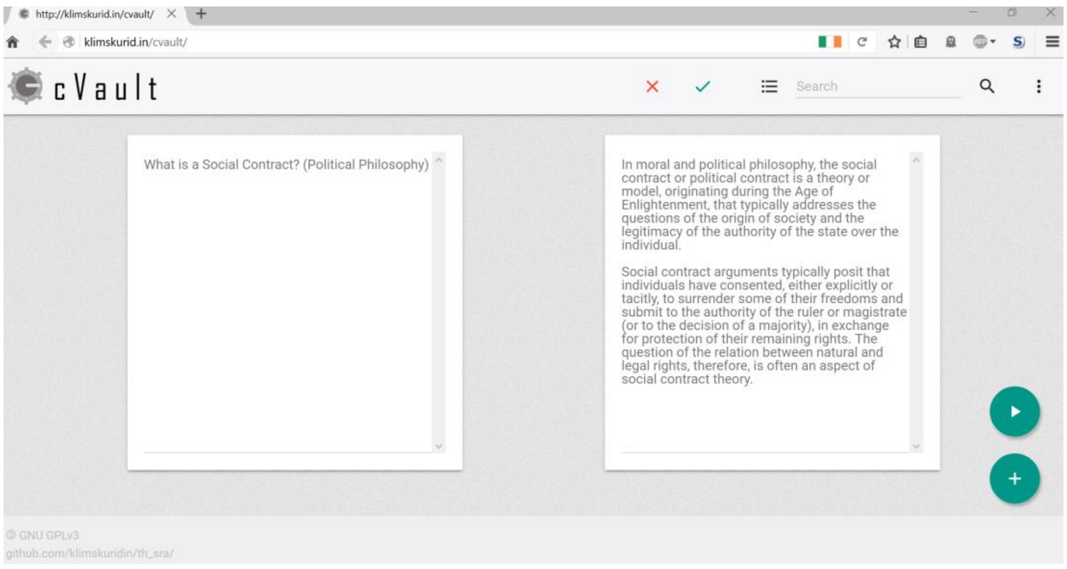

http://klimskurid.in/cvault/
klimskurid.in/cvault/
cVault
Search
What is a Social Contract? (Political Philosophy)
© GNU GPLv3
github.com/klimskuridin/th_sra/

Figure 23: Main UI / Revise View - Hidden

In order to provide action feedback to the user, popup alarms have been added to some of the triggers, such as logging in and out, grading a card and when saving and deleting one. The alarm element is found in the original Bootstrap component library /16/.



What is a Social Contract? (Political Philosophy)

In moral and political philosophy, the social contract or political contract is a theory or model, originating during the Age of Enlightenment, that typically addresses the questions of the origin of society and the legitimacy of the authority of the state over the individual.

Social contract arguments typically posit that individuals have consented, either explicitly or tacitly, to surrender some of their freedoms and submit to the authority of the ruler or magistrate (or to the decision of a majority), in exchange for protection of their remaining rights. The question of the relation between natural and legal rights, therefore, is often an aspect of social contract theory.

Figure 24: Main UI / Revise View - Shown

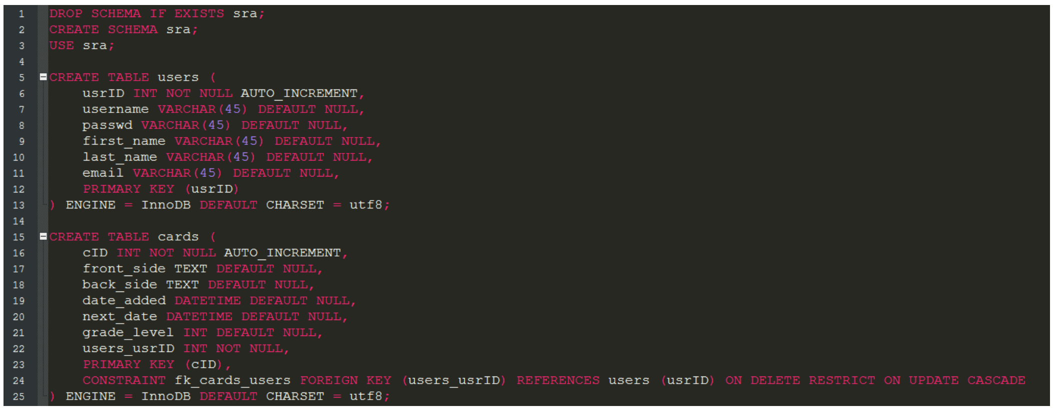
At the moment, the UI relies on static data since the back-end part has not been developed yet. The static data has been placed in the JS file variables. Java will not allow jQuery access to external files, therefore, storing this data outside of the JS file will not possible. The search button is bound to the same function as the List button, since there is no search function available yet.

UI responsiveness and mobile views are implemented by using the Bootstrap's columns grid /16/, as well as by using the @media CSS property that changes the elements properties when the viewport size is smaller than a certain amount of pixels. The mobile UI optimization is not yet complete at this point and will be further developed after the back-end is ready.

Other things that are worth noticing are the logo and the app name. These things are currently meant to be used temporarily and are not final. It is considered that using some kind of real-like logo and name is much preferred over generic placeholders which diminish the aesthetical appearance of the app's UI.

4.3 Database

Creating a database in a rather straightforward process that requires only one script with 25 lines of code. The script is written in SQL.



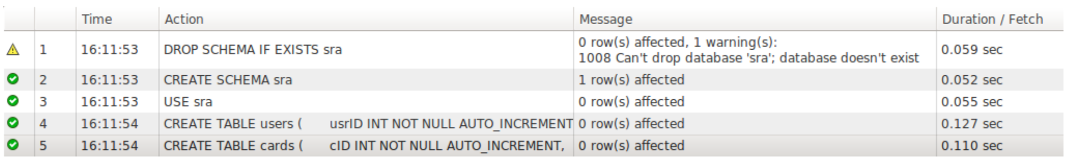
```sql
DROP SCHEMA IF EXISTS sra;
CREATE SCHEMA sra;
USE sra;

CREATE TABLE users (
    usrID INT NOT NULL AUTO_INCREMENT,
    username VARCHAR(45) DEFAULT NULL,
    passwd VARCHAR(45) DEFAULT NULL,
    first_name VARCHAR(45) DEFAULT NULL,
    last_name VARCHAR(45) DEFAULT NULL,
    email VARCHAR(45) DEFAULT NULL,
    PRIMARY KEY (usrID)
) ENGINE = InnoDB DEFAULT CHARSET = utf8;

CREATE TABLE cards (
    cID INT NOT NULL AUTO_INCREMENT,
    front_side TEXT DEFAULT NULL,
    back_side TEXT DEFAULT NULL,
    date_added DATETIME DEFAULT NULL,
    next_date DATETIME DEFAULT NULL,
    grade_level INT DEFAULT NULL,
    users_usrID INT NOT NULL,
    PRIMARY KEY (cID),
    CONSTRAINT fk_cards_users FOREIGN KEY (users_usrID) REFERENCES users (usrID) ON DELETE RESTRICT ON UPDATE CASCADE
) ENGINE = InnoDB DEFAULT CHARSET = utf8;
```

Figure 25: Database Creation Query

The first three lines create the database if hasn't been created yet. It is convenient to add this condition for testing purposes, since if there is a problem with the query and it does not run completely, it would otherwise be necessary to drop the schema manually /22/.

The other two parts are responsible for creating the tables. The last part of the query that creates the 'cards' table, sets a User ID from the 'users' table as a Foreign Key constraint. This is necessary in order to differentiate between cards that belong to different users. Without this constraint it would not be possible /22/.

After the script is ready, MySQL Workbench is used to connect to a local database setup within the development environment. The query runs successfully, which means that now it is possible to connect to the AWS VM and create the database on the main server.



		Time	Action	Message	Duration / Fetch
⚠	1	16:11:53	DROP SCHEMA IF EXISTS sra	0 row(s) affected, 1 warning(s): 1008 Can't drop database 'sra'; database doesn't exist	0.059 sec
✓	2	16:11:53	CREATE SCHEMA sra	1 row(s) affected	0.052 sec
✓	3	16:11:53	USE sra	0 row(s) affected	0.055 sec
✓	4	16:11:54	CREATE TABLE users (usrID INT NOT NULL AUTO_INCREMENT	0 row(s) affected	0.127 sec
✓	5	16:11:54	CREATE TABLE cards (cID INT NOT NULL AUTO_INCREMENT,	0 row(s) affected	0.110 sec

Figure 26: Database Query Results

After it is created, the database is populated by a second script that creates ten cards to serve as sample data. The query needs to be executed for each of the created users. MySQL NOW() function is used in order to set the creation date, as well as the card due dates when adding the cards.

All of the scripts run successfully without any errors. Now the database has been created and it is possible to move on to creating the back-end part of the application.

4.4 Back-End & Deployment

The first thing to do in order to start developing the back-end, is to convert the HTML website into the JSP format. Converting is a rather straightforward process that requires changing only the file resolution from HTML to JSP and the local external CSS and JS resources paths using the ${pageContext.request.contextPath} function.

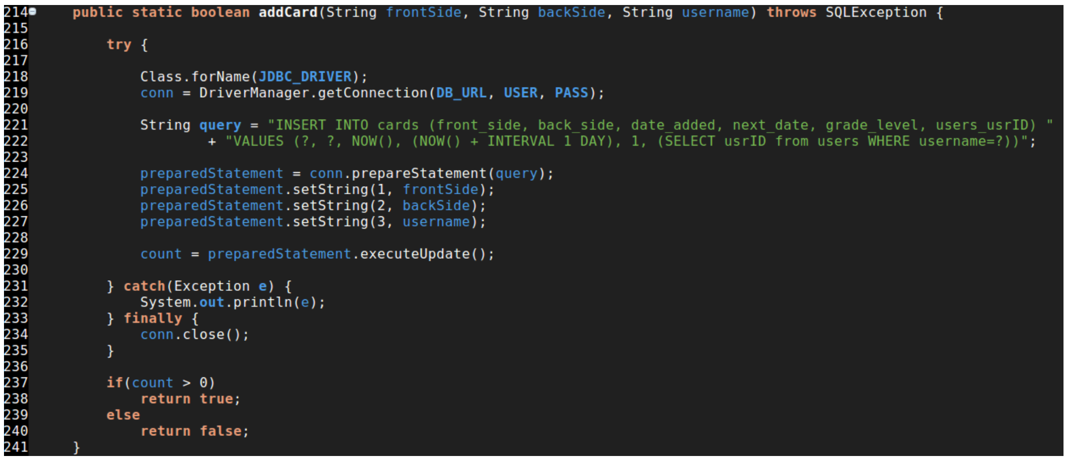
The project can then be accessed through Eclipse which can generate the WAR file for deployment. The WAR file is then uploaded to the AWS Tomcat server using the Tomcat Web Application Manager GUI App. No further modifications are necessary and the previously-developed part of the app runs fine.

After the app has been converted to Java, it is necessary to import the libraries and add them to the project build path. The project is dependent on four libraries only, two of which (mysql-connector and ojdbc) are related to working with MySQL, one provides the servlet functionality and the last one is the Gson which converts responses to and from the JSON format /24/.

After adding the libraries, all of the program's JAVA files need to be created and mapped in web.xml deployment descriptor. The mapping is necessary only for the servlets, which will be accessed by the AJAX calls. The app currently contains two basic classes, two DB-related classes and seven servlets. The basic classes are Card and User which contain all of the constructors, getters and setters for working with these basic app entities.

The DBHelper class contains all of the methods for accessing the database, adding, updating, retrieving, removing and otherwise modifying the data. The DBDebug class is used for testing the DB-related methods through the console, in order to ensure that they work fine, before they are integrated within the app.

All of the DB-related methods are built using the PreparedStatement Java class, which allows specifying variables within the SQL query in a protected manner, ensuring that no SQL Injection type attack could occur. This kind of protection is necessary since the app user has access to a search bar, which forwards the searching keyword as an SQL query argument to the database /23/.

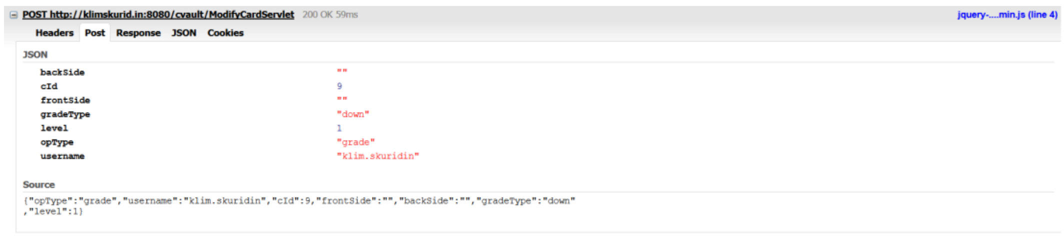


```java
public static boolean addCard(String frontSide, String backSide, String username) throws SQLException {

    try {

        Class.forName(JDBC_DRIVER);
        conn = DriverManager.getConnection(DB_URL, USER, PASS);

        String query = "INSERT INTO cards (front_side, back_side, date_added, next_date, grade_level, users_usrID) "
                + "VALUES (?, ?, NOW(), (NOW() + INTERVAL 1 DAY), 1, (SELECT usrID from users WHERE username=?))";

        preparedStatement = conn.prepareStatement(query);
        preparedStatement.setString(1, frontSide);
        preparedStatement.setString(2, backSide);
        preparedStatement.setString(3, username);

        count = preparedStatement.executeUpdate();

    } catch(Exception e) {
        System.out.println(e);
    } finally {
        conn.close();
    }

    if(count > 0)
        return true;
    else
        return false;
}
```

Figure 27: DBHelper addCard Method

The most complicated phase of the development process is to create a working and reliable connection between the front-end, the back-end and the database. As described in the Analysis section, the data is retrieved through jQuery, which then sends it to a corresponding servlet using a AJAX POST request in JSON format. The servlet then parses the request, using a Gson method, and depending on the received data and the requested operation type, calls the DBHelper method(s) that are responsible for retrieving the data from the database. If the data has been retrieved successfully, a JSON response is composed by the servlet and sent back to jQuery AJAX method. The response is then parsed and a corresponding action is performed based on its contents.

Connecting all of the application's parts for the first time creates many opportunities for errors and issues to occur. These would need to be tracked and debugged. This is mainly done by using the FireBug Firefox browser plugin that displays all of the requests, responses and possible issues that greatly assist in pinpointing the source of the problem (Figure 28).

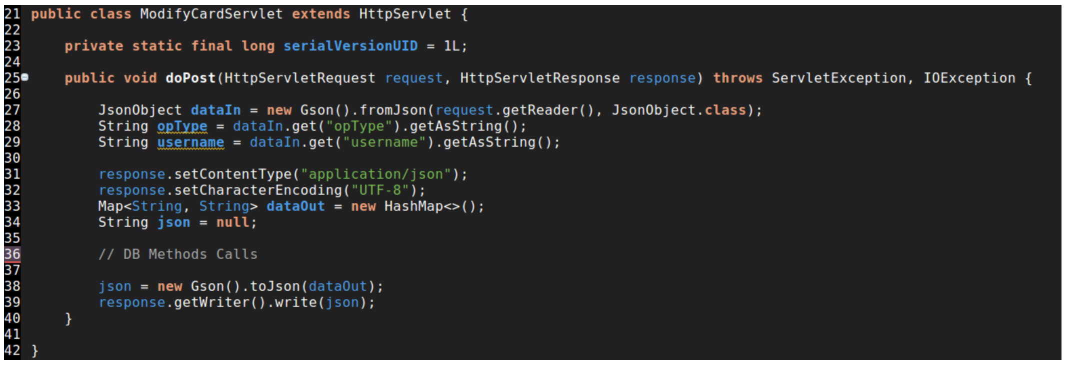

POST http://klimskurid.in:8080/cvault/ModifyCardServlet 200 OK 59ms
jquery-....min.js (line 4)
Headers
Post
Response
JSON
Cookies
JSON

```
backSide      ""
cId           9
frontSide     ""
gradeType     "down"
level         1
opType        "grade"
username      "klim.skuridin"
```

```
{"opType":"grade","username":"klim.skuridin","cId":9,"frontSide":"","backSide":"","gradeType":"down"
,"level":1}
```

Figure 28: JSON Request in FireBug

Each of the servlets is responsible for a specific action. For example, the LoginServlet receives the data from the Login UI username and password inputs, forwards it to the database method, which checks if such username/password combination exists, and sends a true/false response. The user is then either logged in or not. The ModifyCardServlet is responsible for most of the card-related actions, such as adding, updating, and grading cards. The operation type is determined based on the opType JSON parameter.



```java
public class ModifyCardServlet extends HttpServlet {

    private static final long serialVersionUID = 1L;

    public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

        JsonObject dataIn = new Gson().fromJson(request.getReader(), JsonObject.class);
        String opType = dataIn.get("opType").getAsString();
        String username = dataIn.get("username").getAsString();

        response.setContentType("application/json");
        response.setCharacterEncoding("UTF-8");
        Map<String, String> dataOut = new HashMap<>();
        String json = null;

        // DB Methods Calls

        json = new Gson().toJson(dataOut);
        response.getWriter().write(json);
    }

}
```

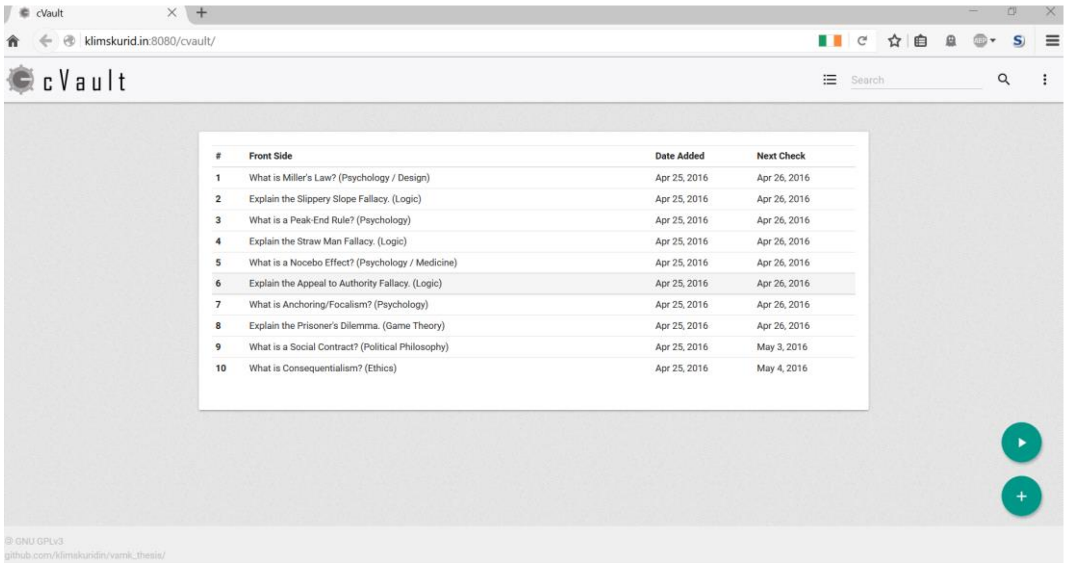
Figure 29: ModifyCardServlet Method

Retrieving the next card, that is due for revision, is done using the GetDueCardServlet, which selects one of the cards with a next_date parameter less than of a current timestamp. Only one card is returned, even if more than one is due for revision. A user is not allowed to skip the revision of the card and select a new one randomly. The same card will be returned multiple times until it has been graded.

The cards DB table stores all of the dates in the DATETIME format, which contains both date and time. The time part is not currently utilized by the app and has been added for possible future use cases allowing repetition of cards in time intervals of less than 24 hours.

The original design also included the SessionServlet, which, unfortunately, has not yet been implemented. It has been decided to leave this section out of the thesis paper and implement it at a later time.

After all of the application functions have been developed locally, they are tested manually with each function and each request/response monitored for possible issues and inconsistencies. A WAR file is then created and deployed on the AWS Tomcat server (Figure 30).


cVault
klimskurid.in:8080/cvault/
cVault
Search
© GNU GPLv3
github.com/klimskuridin/vamk_thesis/

#	Front Side	Date Added	Next Check
1	What is Miller's Law? (Psychology / Design)	Apr 25, 2016	Apr 26, 2016
2	Explain the Slippery Slope Fallacy. (Logic)	Apr 25, 2016	Apr 26, 2016
3	What is a Peak-End Rule? (Psychology)	Apr 25, 2016	Apr 26, 2016
4	Explain the Straw Man Fallacy. (Logic)	Apr 25, 2016	Apr 26, 2016
5	What is a Nocebo Effect? (Psychology / Medicine)	Apr 25, 2016	Apr 26, 2016
6	Explain the Appeal to Authority Fallacy. (Logic)	Apr 25, 2016	Apr 26, 2016
7	What is Anchoring/Focalism? (Psychology)	Apr 25, 2016	Apr 26, 2016
8	Explain the Prisoner's Dilemma. (Game Theory)	Apr 25, 2016	Apr 26, 2016
9	What is a Social Contract? (Political Philosophy)	Apr 25, 2016	May 3, 2016
10	What is Consequentialism? (Ethics)	Apr 25, 2016	May 4, 2016

Figure 30: Application Deployed on AWS

The application has been tested to run without any major issues and all of the created functions appear to work as expected.

5. CONCLUSION & DISCUSSION

The main objective of this project was to develop an application that would utilize the Spaced Repetition principle, serve as a reliable tool for revising bits of knowledge and provide an arguably better user experience than existing solutions while doing so. The above-mentioned goals have been achieved, however, there is still enough potential for improving the program.

Most importantly, the application currently does not offer a way to track sessions and store local cache data, there is no hashing when storing passwords, there is no user registration and account management system and no due card reminder mechanism.

One of the biggest development challenges was to design every aspect of the application prior to starting the implementation process, and to develop each one, while adhering to the previously created plan, and making sure that each component of the program is compatible with every other one.

The development process has presented a lot of opportunities for learning and gaining new experience, particularly in organizing the development environment and workflow, as well as in using specific tools to test, pinpoint any possible issues and software bugs during the development process.

It has been decided to continue developing the project and use it as a learning platform for testing and integrating new features before moving on to developing other market-ready commercial applications.

6. REFERENCES

/1/ MySQL Reference C.10.3 Limits on Table Size. Accessed 09.05.2016. https://dev.mysql.com/doc/refman/5.7/en/table-size-limit.html

/2/ MySQL Reference C.10.2 Limits on Number of Databases and Tables. Accessed 09.05.2016. https://dev.mysql.com/doc/refman/5.5/en/database-count-limit.html

/3/ Wikipedia – Spaced Repetition. Accessed 09.05.2016. https://en.wikipedia.org/wiki/Spaced_repetition

/4/ Wikipedia – Leitner System. Accessed 09.05.2016. https://en.wikipedia.org/wiki/Leitner_system

/5/ Amazon AWS Documentation – IAM. Accessed 09.05.2016. https://aws.amazon.com/documentation/iam/

/6/ Amazon AWS Documentation – EC2. Accessed 09.05.2016. https://aws.amazon.com/documentation/ec2/

/7/ Amazon AWS Documentation – CLI. Accessed 09.05.2016. https://aws.amazon.com/documentation/cli/

/8/ Amazon AWS Documentation – Route 53. Accessed 09.05.2016. https://aws.amazon.com/documentation/route53/

/9/ Ubuntu Community Documentation – OpenSSH. Accessed 09.05.2016. https://help.ubuntu.com/community/SSH/OpenSSH/Configuring

/10/ Nginx vs Apache Overview. Accessed 09.05.2016. https://www.nginx.com/blog/nginx-vs-apache-our-view/

/11/ DigitalOcean LEMP Setup Guide. Accessed 09.05.2016. https://www.digitalocean.com/community/tutorials/how-to-install-linux-nginx-mysql-php-lemp-stack-on-ubuntu-14-04

/12/ Apache Tomcat Documentation – Manager App. Accessed 09.05.2016. https://tomcat.apache.org/tomcat-7.0-doc/manager-howto.html

/13/ Apache Tomcat Documentation – Security. Accessed 09.05.2016. https://tomcat.apache.org/tomcat-7.0-doc/security-howto.html

/14/ Nginx Documentation – Restricting Access. Accessed 09.05.2016. https://www.nginx.com/resources/admin-guide/restricting-access/

/15/ MySQL Reference – Adding Users. Accessed 09.05.2016. https://dev.mysql.com/doc/refman/5.7/en/adding-users.html

/16/ Twitter Bootstrap Documentation – CSS. Accessed 09.05.2016. https://getbootstrap.com/css/

/17/ Bootstrap Material Design Documentation. Accessed 09.05.2016. https://rosskevin.github.io/bootstrap-material-design/

/18/ Material Design FAB Icon. Accessed 09.05.2016. http://codepen.io/ad-dyosmani/pen/xcLCz

/19/ jQuery API Documentation. Accessed 09.05.2016. https://api.jquery.com/

/20/ Animate.css Documentation. Accessed 09.05.2016. https://github.com/daneden/animate.css

/21/ Google Material Design Specifications. Accessed 09.05.2016. https://www.google.com/design/spec/material-design/

/22/ MySQL Reference – Database Use. Accessed 09.05.2016. https://dev.mysql.com/doc/refman/5.7/en/database-use.html

/23/ Wikipedia – Prepared Statement. Accessed 09.05.2016. https://en.wikipe-dia.org/wiki/Prepared_statement

/24/ Google Gson User Guide. Accessed 09.05.2016. https://sites.google.com/site/gson/gson-user-guide