*Florida International University*

*School of Computing and Information Sciences*

Software Engineering Focus

Final Deliverable

Project Title: Vocabulary in Reading Study

VIRS 3.0

**Team Members:** Christian Hidalgo, Juan Carlos Valladares

**Product Owner(s)**: Eric Dwyer, Seyedjafar Ehsanzadehsorati

**Mentor(s)**: Leila Zahedi

**Instructor**: Masoud Sadjadi

The MIT License (MIT)

Copyright (c) *2017 Florida International University*

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

***Abstract***

*This document presents the information necessary to gain a good understanding of Vocabulary in Reading Study (VIRS 3.0). VIRS is a web app which facilitates learning of new languages through analysis of text and gathering of data which is then displayed to the user. This data is important to the user as it categorizes words in the text and gives priority to more popular words. The user then knows which words to use more and makes it easier to use these in conversation. The user can now test their knowledge of vocabulary, a user can register, a user can translate, and new administrative tools for the product owner.*

[**Introduction**](#_ba70ox9edxb5) **6**

[Current System](#_mbx0cy8ybnjd) 6

[Purpose of New Version](#_1fob9te) 7

[**User Stories**](#_3znysh7) **8**

[Implemented User Stories](#_2et92p0) 9

[Pending User Stories](#_fdvuzpgpueoc) 9

[**Project Plan**](#_afik6w6hgbad) **10**

[Hardware and Software Resources](#_2s8eyo1) 10

[Sprints Plan](#_2jxsxqh) 12

[**System Design**](#_c5lo7ufcs9k8) **18**

[Architectural Patterns](#_1y810tw) 18

[**System Validation**](#_qsh70q) **23**

[Test Suite](#_33v1mmyjx1b8) 23

[Test Cases](#_gt8j2is1kke7) 23

[**Appendix**](#_3as4poj) **24**

[Appendix A - UML Diagrams](#_1pxezwc) 24

[Appendix B - User Interface Design](#_49x2ik5) 24

[Appendix C - Sprint Review Reports](#_2p2csry) 24

[Appendix D - User Manuals, Installation/Maintenance Document, Shortcomings/Wishlist Document and other documents](#_147n2zr) 24

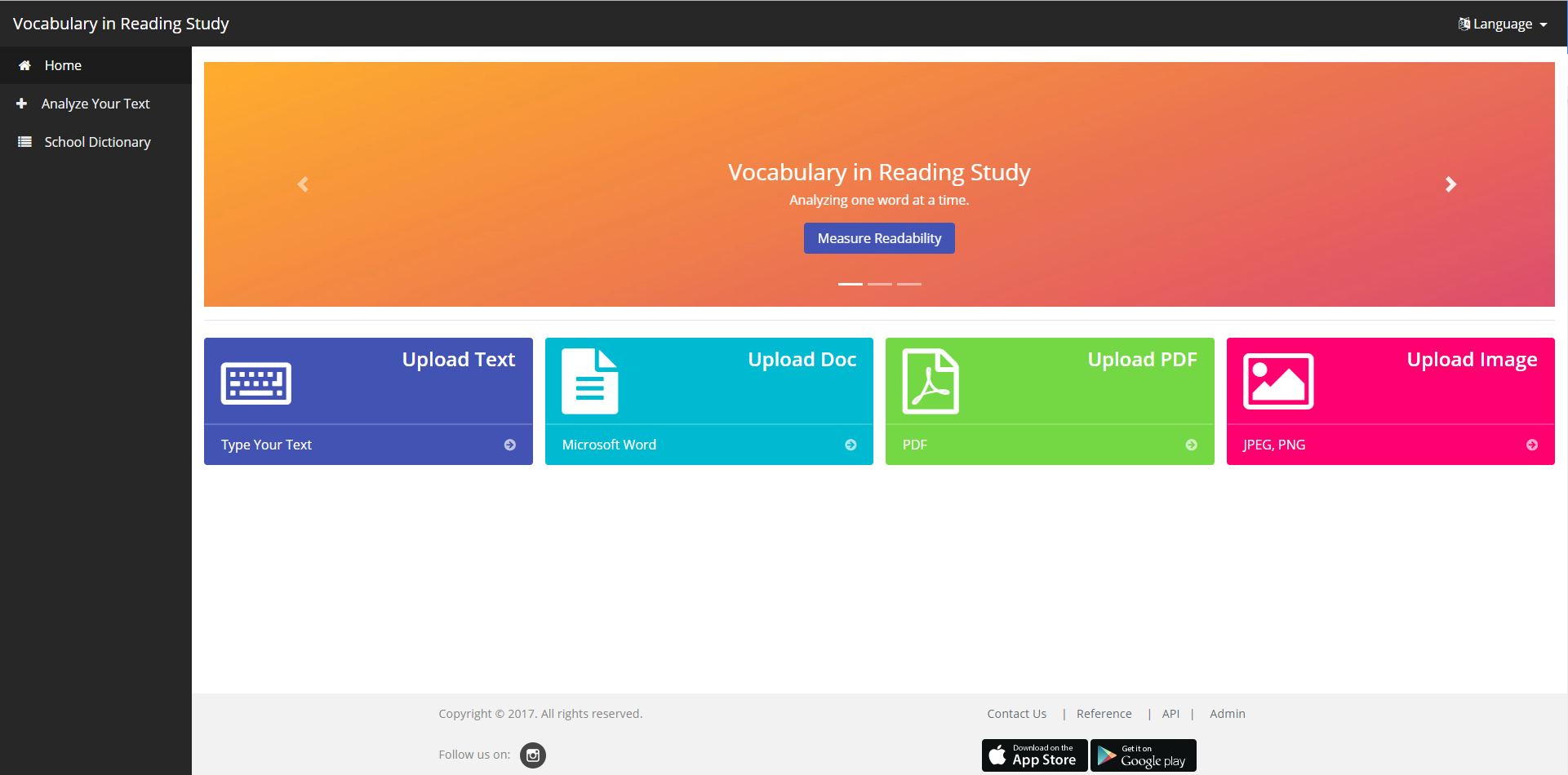
# 

# **Introduction**

One of the major challenges that English Language learners (ELL) as well as mainstream students face is the lack of a reliable source to improve their academic words. There is no easy way to validate the easiness of text, which allows professors to select the appropriate materials for class. A challenging text is a wonderful way to propel students forward, yet something too hard to read can cause the opposite effect. Vocabulary in Reading Study(VIRS) is the solution to all these problems.

## Current System

This is the third iteration of the application. The current system consists of using MySQL, Spring MVC and Angular hosted in Amazon web servers. The system can currently analyze a text, word document, pdf and even images to give a readability score. The web application currently also displays a dictionary that is clickable for definitions. The application includes administrative tools for the product owners to add and remove words from the dictionary. The text analyzer also has a text.The application has a cohesive theme that is present throughout the site.



*Figure 1-VIRS 2.0 dashboard*

## Purpose of New Version

The imperative task of the new system is to add features to make this a well rounded website, that achieves the purpose of helping those seeking to learn English as their new language. The updates must maintain the theme currently in the system and be in the vein of the purpose of this project.

Features must be accessible on both desktop and mobile platform, the features must enrich the website so as to ensure user satisfaction with not only the main feature of text analysis, but also the new features added in this version.

The features must be created to maintain the current system, while adding unto it. Therefore, maintaining the current backend API. The features to be included are a testing page for users to develop their vocabulary, a translation tool for user to better understand new words in their home language, a registration page so users can keep information on their use of the site and the product owners know who is using their site, and an administrative tool that allows for better management of the current database of words and their categories.

# **User Stories**

The following section provides the detailed user stories that were implemented in this iteration of the VIRS project. These user stories served as the basis for the implementation of the project’s features. This section also shows the user stories that are to be considered for future development.

## Implemented User Stories

User Story: [#259] Add Facebook link

User Story: [#257] Fix PDF upload

User Story: [#258] Fix .Docs upload

User Story: [#265] [Frontend] Add Different Tests Levels

User Story: [#266] [Frontend] Skip Questions

User Story: [#267] [Frontend] Add external link to ck12 (School Library)

User Story: [#268] [Frontend] Add Tests section to main page

User Story: [#264] [Frontend] Translator

User Story: [#273] [Frontend] Finish Test

User Story: [#274] [Frontend] School Library

User Story: [#276] [Backend] Translation Endpoint to connect with Frontend

User Story: [#277] Add multiple words through dictionary to CSV

User Story: [#278] [Backend] Register User

User Story: [#279] [Backend] Login User

User Story: [#280] [Frontend] Account Page

User Story: [#283] Itranslate limitations for different users

User Story: [#284] User Registration API

User Story: [#285] [Frontend] - Limit Test Attempts (Unregistered)

User Story: [#286] [Backend] - Fix Wikipedia API

User Story: [#288] - [Frontend] - Paypal Donate Button

## 

**User Story Name:** Add Different Test Levels

* Description: As a user, I would like to be able to select a difficulty level for the tests I am going to take, so I can challenge myself intellectually.

Acceptance Criteria

* The user selects the difficulty desired
* The quiz/tests loads the questions from the right batch of questions.

**Use Case**

* Name: Select Test Difficulty
* Actor: User
* Preconditions: User must be currently using the application and navigate to the test page.
* Description <Flow of events>:

1. System displays 6 different categories of tests.
2. User selects one category.

System loads test.

**User Story Name:** Add Vocabulary Tests Section

* Description: As a user i would like to be able to navigate to a test page so i can test my vocabulary knowledge.

Acceptance Criteria

* The user clicks on the “Tests” link
* The system displays the test section of the application
* The system displays different categories to the user.

**Use Case**

* Name: Display Tests Page
* Actor: User
* Preconditions: User must be currently using the application.
* Description <Flow of events>:

1. User goes to the sidebar to locate the “Tests” link
2. Once clicked, the system will load a page displaying different test categories.

**User Story Name:** Finish\_and\_display\_results

* Description: As a user, I would like to be able to press ‘finish’ in order to end the current tst session and be displayed with my performance

Acceptance Criteria

* The user hits the “finish” button when he/she desired to stop.
* The system first checks if the user has attempted more than 10 questions
* The system calculates and displays the results to the user.

**Use Case**

* Name: Finish\_And\_Display\_Results
* Actor: User
* Preconditions: User must be currently in the process of taking a test and has attempted more than 10 questions
* Description <Flow of events>:

1. Question is currently loaded/displayed. Test in progress.
2. User hits the “Finish” button
3. System verifies that the user has done more than 10 questions
4. If user hasn’t done more than 10, display warning and test proceeds.
5. If user has done more than 10 questions, test stops and a table displaying the performance is populated.

**User Story Name:** Skip Questions

* Description: As a user, I would like to be able to press a skip button so I can skip the current question and get another one

Acceptance Criteria

* The user hits the “skip” button when the question is not submitted
* The user gets notified he/she presses the skip button.

**Use Case**

* Name: Skip Question
* Actor: User
* Preconditions: User must be currently in the tests page and taking one of the tests and have a question loaded
* Description <Flow of events>:

1. Question is currently loaded/displayed
2. User hits the “skip” button
3. System notifies user that he is skipping the question
4. User either accepts or denies the skip.
5. If user accepts the skip, system loads next question, otherwise stays in current question

**User Story Name:** [Frontend] Translator

* Description: **As a** User **I would like** to input text  **so that** the text can be translated to selected language

Acceptance Criteria

* Text can be inputted and target language can be selected
* Translated text can be displayed

**Use Case**

* Name: Translate text
* Actor: User
* Preconditions: User must be in the itranslate page of the site
* Description :
  + - * + System displays text area for user to input text and allows user to select target language
        + System then displays translated text

**User Story Name:** Login User

* As a user I would like to be able to log in to the site in order to access my personalized content.

**Acceptance Criteria**

* The user navigates to the login page.
* The user enters required information.
* The system validates information entered.
* The system provides feedback of success/failed login attempt

**Use Case**

* Name: Login\_User\_Success
* Actor: Registered user
* Preconditions: User must be currently in the home page
* Description <Flow of events>:

1. Actor clicks ‘login’ on the sidebar menu
2. Modal appears asking for user information
3. System validates user once user submits information
4. System responds with a validation result

**User Story Name:** User API

* As an ADMIN user, I would like to have access to an API that will let me modify the users in the database

**Acceptance Criteria**

* Admin user logs in.
* Admin user selects the desired user to modify
* The modification is done correctly

**Use Case**

* Name: Delete\_User
* Actor: Admin user
* Preconditions: Actor must be logged in into the system
* Description <Flow of events>:

1. Actor navigates to the User API
2. Actor selects a desired user to delete from the database
3. System verifies if user exists and deletes from database.

**User Story Name:** Register User

* Description: As a user, I would like to have the option to register so I can access the features of a registered user.

Acceptance Criteria

* User navigates to the ‘Register’ page
* User enters required information in form and submits it
* Information is stored in a database.
* Page provides feedback of success/failed registration

**Use Case**

* Name: User\_Registration\_Success
* Actor: Unregistered user
* Preconditions: Application is loaded and accessible
* Description <Flow of events>:

1. Actor clicks on ‘Register’ from the sidebar menu.
2. System loads the registration form
3. Actor fills out the required information.
4. Actor hits ‘Submit’
5. System notifies actor od a successful registration

**User Story Name:** [Backend] iTranslator endpoint

* Description: **As a** Frontend **I would like** to send requests to the backend **so that** I can have the backend translate words

Acceptance Criteria

* The backend returns a json that includes the translation

**Use Case**

* Name: Itranslate\_Backend
* Actor: Frontend
* Preconditions: Frontend is able to send requests to backend
* Description :
  + Frontend sends post request to backend
  + Backend takes request and prompts google for translation
  + Backend receives translation from google
  + Backend sends json with translation to frontend

## 

**User Story Name:** Profile Page

* As a user I would like to be able to log in to the site in order to access my personalized content.

**Acceptance Criteria**

* The user navigates to the login page.
* The user enters required information.
* The system validates information entered.
* The system provides feedback of success/failed login attempt

**Use Case**

* Name: View\_Profile\_Page
* Actor: Registered user
* Preconditions: User must be currently in the home page
* Description <Flow of events>:

1. Actor clicks ‘Profile’ on the sidebar menu
2. System navigates to the Profile page
3. System loads Actor’s info from the database into page

**User Story Name:** Limit Test Attempts (Unregistered User)

* As an unregistered user, I would like to have a limited number of attempts for each section so i could then be lead to register for the extended services

**Acceptance Criteria**

* A user (unregistered) presses on the ‘Start’ button
* The attempt for that test section goes down by one
* The test gets locked out once the counter reached 0

**Use Case**

* Name: Decrease\_test\_attempt\_by\_1
* Actor: Unregistered User
* Preconditions: Actor must be in the main test page
* Description <Flow of events>:

1. Actor selects a desired test category.
2. Actor presses the ‘start’ button for test to begin.
3. System reduces the number of attempts remaining by 1.

**User Story Name:** Add multiple words

* Description: **As a**n administrator **I would like** to be able to upload a csv **so that** I can add multiple words to database at once

Acceptance Criteria

* Words from csv are added to database

**Use Case**

* Name: Add\_Multiple\_Words
* Actor: Administrator
* Preconditions: Administrator has logged in to admin portal and has a csv file ready to upload with content
* Description :
  + Admin uploads csv
  + System processes csv
  + System adds words to database
  + System displays to user words that were added.

**User Story Name:** Itranslate Limit

* Description: **As a** User **I would like** to know the limits of characters I can translate **so that** I know how much left I can translate

Acceptance Criteria

* User can see how many characters they have left to translate
* User can’t translate any more characters once they’ve reached zero

**Use Case**

* Name: Translation\_Limit\_001
* Actor: User
* Preconditions: User must be in translation page
* Description :
  + User translates
  + Counter with Characters left to translate updates
  + User can’t translate once they reach 0 characters

**User Story Name:** Paypal Donate Button

* Description: **As a** User **I would like** to be able to click a donate button **so that** I can donate money to support the website.

Acceptance Criteria

* A user can click a donate button and they can enter an amount for donation.
* Money donated is in paypal account

**Use Case**

* Name: Paypal\_Donate\_Button
* Actor: User
* Preconditions: Be on the site
* Description :
  + User clicks on donate button
  + User goes through paypal donation process
  + User is returned to site with confirmation

## 

## 

## 

## 

## 

## Pending User Stories

User Story: [#289] - [Frontend] - Paid Subscription

User Story: [#281] [Frontend] Edit Profile

## User Story: [#287] [Frontend] - Add WAT tests

**User Story Name:** Paid Subscription

● As a user, I would like to have a paid subscription so that I can have more access to website features

Acceptance Criteria

● The user can submit payment information

● The user has more access to limited features in the website

## 

**User Story Name:** Edit Profile

* As an registered user, I would access an edit profile page so i can edit my information stored in the system

**Acceptance Criteria**

* A user (registered) navigates to the profile page
* User modifies selected fields
* System correctly updated the database

**User Story Name:** Add WAT Tests

* As a user, i would like to have a WAT test section so i can take the WAT related tests on this system

**Acceptance Criteria**

* A user navigates to the Tests page
* User selects the WAT category
* System loads WAT category questions

# **Project Plan**

This section describes the planning that went into the realization of this project. This project incorporated the agile development techniques and as such required the sprints to be planned. These sprint planning’s are detailed in the section. This section also describes the components, both software and hardware, chosen for this project.

## Hardware and Software Resources

The following is a list of all hardware and software resources that were used in this project:

***Hardware:***

* Computer running Linux, Mac OS or windows.
* For the deployed application make sure you stay within the AWS server constraints
  + 64bit Amazon Linux 2017.03 v2.5.5 running Java 8

***Software:***

The following list is the software used in the application. Note that it is quite extensive and includes all the development layers of the stack.

**Front end:**

* Angular 4.3.1
* Ng-Bootstrap 1.0.0
* Ng-Translate 7.0.0
* Chat.js 2.7.1
* Font-Awesome 4.7.0
* Ng2-Charts 1.6.0
* Rxjs 5.1.0
* Jasmine 2.5.45
* Karma 1.7
* Typescript 2.3.3

**Back end:**

* Maven 3.5
  + Commons-lang3 3.4
  + HikariCP 1.5.6.RELEASE
  + Jai-imageio-core 1.3.1
  + Jai-imageio-jpeg2000 1.3.0
  + Jasypt-spring-boot-starter 1.16
  + Levigo-jbig2-imageio 2.0
  + Mysql-connector-java 1.5.6.RELEASE
  + Opencsv 3.3
  + Opencv 3.2.0-1
  + Spring-boot-starter-data-jpa 1.5.6.RELEASE
  + Spring-boot-starter-security 1.5.6.RELEASE
  + Spring-boot-starter-test 1.5.6.RELEASE
  + Spring-boot-starter-web 1.5.6.RELEASE
  + Springfox-swagger-ui 2.7.0
  + Springfox-swagger2 2.7.0
  + Sqlite-jdbc 1.5.6.RELEASE
  + Tess4j 3.4.1
  + Thucydides-core 0.9.275
  + Tika-parsers 1.16
* Tesseract 3.05.01
* Leptonica 1.74.4
* Mysql 14.14
* Java 1.8.0
* Spring Developer Suite 3.9

**Android:**

* Android Studio 3.0
* Gradle 3.0
* Android sdk 23

**iOS:**

* Xcode 9.1
* Command line tools 9.2

**Other:**

* Git 2.14.1
* Bash 3.2.57

## Sprints Plan

*Sprint 1*

Planning Meeting Minutes: 01/12/2018

Attendees: Christian Hidalgo, Juan Carlos Valladares, S.J. Ehsanzadeh

Start time: 5.30pm

End time: 6.50pm

After discussion, the velocity of the team were estimated to be 50pts.

The product owner chose the following user stories to be done during the next sprint. They are ordered based on their priority.

User Story: [#260] [Frontend] Add STEM word category

User Story: [#268] [Frontend] Add Tests section to main page

User Story: [#269] [Frontend] Add iTranslate Section

The team members indicated their willingness to work on the following user stories.

<Christian Hidalgo>

User Story: [#260] [Frontend] Add STEM word category

User Story: [#268] [Frontend] Add Tests section to main page

User Story: [#269] [Frontend] Add iTranslate Section

*Sprint 2*

Planning Meeting Minutes: 01/29/2018

Attendees: Christian Hidalgo, Juan Carlos Valladares, S.J. Ehsanzadeh

Start time: 5.30pm

End time: 6.50pm

After discussion, the velocity of the team were estimated to be 100pts.

The product owner chose the following user stories to be done during the next sprint. They are ordered based on their priority.

User Story: [#259] Add Facebook link

User Story: [#257] Fix PDF upload

User Story: [#258] Fix .Docs upload

User Story: [#265] [Frontend] Add Different Tests Levels

User Story: [#266] [Frontend] Skip Questions

User Story: [#267] [Frontend] Add external link to ck12 (School Library)

User Story: [#264] [Frontend] Translator

User Story: [#270] [Backend] Request to Google translation API

User Story: [#270] [Backend] Request to Google translation API

User Story: [#271] [Frontend] Create Itranslate page

The team members indicated their willingness to work on the following user stories.

<Christian Hidalgo>

User Story: [#259] Add Facebook link

User Story: [#257] Fix PDF upload

User Story: [#258] Fix .Docs upload

User Story: [#265] [Frontend] Add Different Tests Levels

User Story: [#266] [Frontend] Skip Questions

User Story: [#267] [Frontend] Add external link to ck12 (School Library)

<Juan Carlos Valladares>

User Story: [#264] [Frontend] Translator

User Story: [#270] [Backend] Request to Google translation API

User Story: [#271] [Frontend] Create Itranslate page

*Sprint 3*

Planning Meeting Minutes: 02/9/2018

Attendees: Christian Hidalgo, Juan Carlos Valladares, S.J. Ehsanzadeh

Start time: 6.40pm

End time: 6.50pm

After discussion, the velocity of the team were estimated to be 100 pts.

The product owner chose the following user stories to be done during the next sprint. They are ordered based on their priority.

User Story: [#152] [Frontend] Add Remaining Tests

User Story: [#273] [Frontend] Finish Test

User Story: [#274] [Frontend] School Library

User Story: [#275] [Frontend] Update Credits page

User Story: [#276] [Backend] Translation Endpoint to connect with Frontend.

The team members indicated their willingness to work on the following user stories.

<Christian Hidalgo>

User Story: [#152] [Frontend] Add Remaining Tests

User Story: [#273] [Frontend] Finish Test

User Story: [#274] [Frontend] School Library

<Juan Carlos Valladares>

User Story: [#275] [Frontend] Update Credits page

User Story: [#276] [Backend] Translation Endpoint to connect with Frontend

*Sprint 4*

Planning Meeting Minutes: 02/23/2018

Attendees: Christian Hidalgo, Juan Carlos Valladares, S.J. Ehsanzadeh

Start time: 6.40pm

End time: 6.50pm

After discussion, the velocity of the team were estimated to be 100pts.

The product owner chose the following user stories to be done during the next sprint. They are ordered based on their priority.

User Story: [#277] Add multiple words through dictionary to CSV

User Story [#278] [Backend] Register User

User Story [#279] [Backend] Login User

User Story [#280] [Frontend] Account Page

User Story [#281] [Frontend] Edit Profile

The team members indicated their willingness to work on the following user stories.

<Christian Hidalgo>

User Story [#278] [Backend] Register User

User Story [#279] [Backend] Login User

User Story [#280] [Frontend] Account Page

User Story [#281] [Frontend] Edit Profile

<Juan Carlos Valladares>

User Story: [#277] Add multiple words

*Sprint 5*

Planning Meeting Minutes: 03/09/2018

Attendees: Christian Hidalgo, Juan Carlos Valladares, S.J. Ehsanzadeh

Start time: 6.40pm

End time: 6.50pm

After discussion, the velocity of the team were estimated to be 100pts.

The product owner chose the following user stories to be done during the next sprint. They are ordered based on their priority.

User Story: [#282] [Frontend] Graph colors match with display

User Story [#283] Itranslate limitations for different users

User Story: [#284] User Registration API

The team members indicated their willingness to work on the following user stories.

<Christian Hidalgo>

[#284] User Registration API

<Juan Carlos Valladares>

User Story: [#282] [Frontend] Graph colors match with display

User Story [#283] Itranslate limitations

*Sprint 6*

Planning Meeting Minutes: 04/13/2018

Attendees: Christian Hidalgo, Juan Carlos Valladares, S.J. Ehsanzadeh

Start time: 6.40pm

End time: 6.50pm

After discussion, the velocity of the team were estimated to be 100pts.

The product owner chose the following user stories to be done during the next sprint. They are ordered based on their priority.

User Story: [#285] [Frontend] - Limit Test Attempts (Unregistered)

User Story: [#286] [Backend] - Fix Wikipedia API

User Story: [#287] [Frontend] - Add WAT tests

User Story: [#288] - [Frontend] - Paypal Donate Button

The team members indicated their willingness to work on the following user stories.

<Christian Hidalgo>

User Story: [#285] [Frontend] - Limit Test Attempts (Unregistered)

User Story: [#286] [Backend] - Fix Wikipedia API

User Story: [#287] [Frontend] - Add WAT tests

<Juan Carlos Valladares>

User Story: [#288] - [Frontend] – Paypal Donate Button

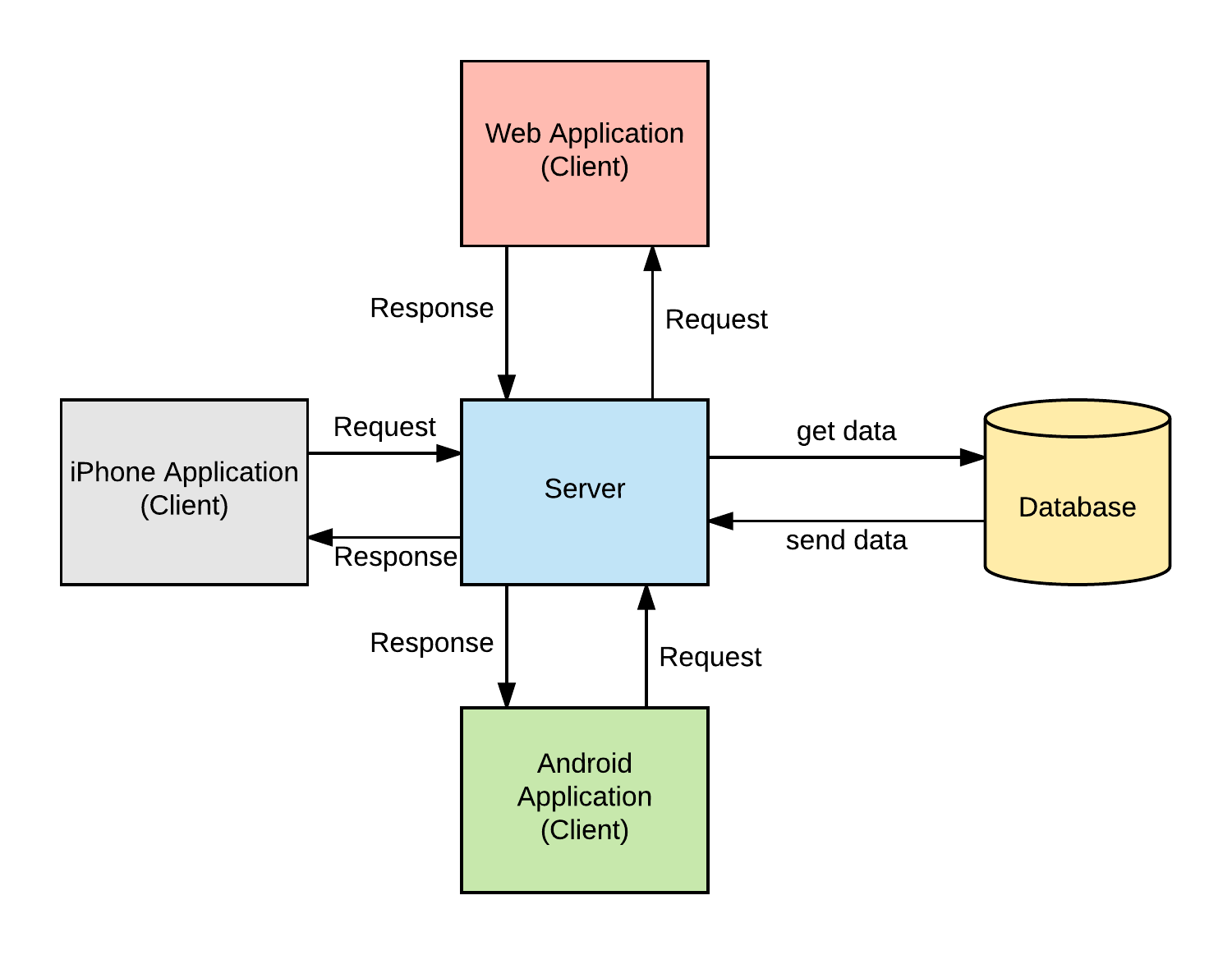
# **System Design**

This section contains information on the design decisions that went into this project. The architecture patterns are outlined and explained. The entire system is shown in a package diagram and the subsystems are explained. Finally, the design patterns used in the project are discussed.

## Architectural Patterns

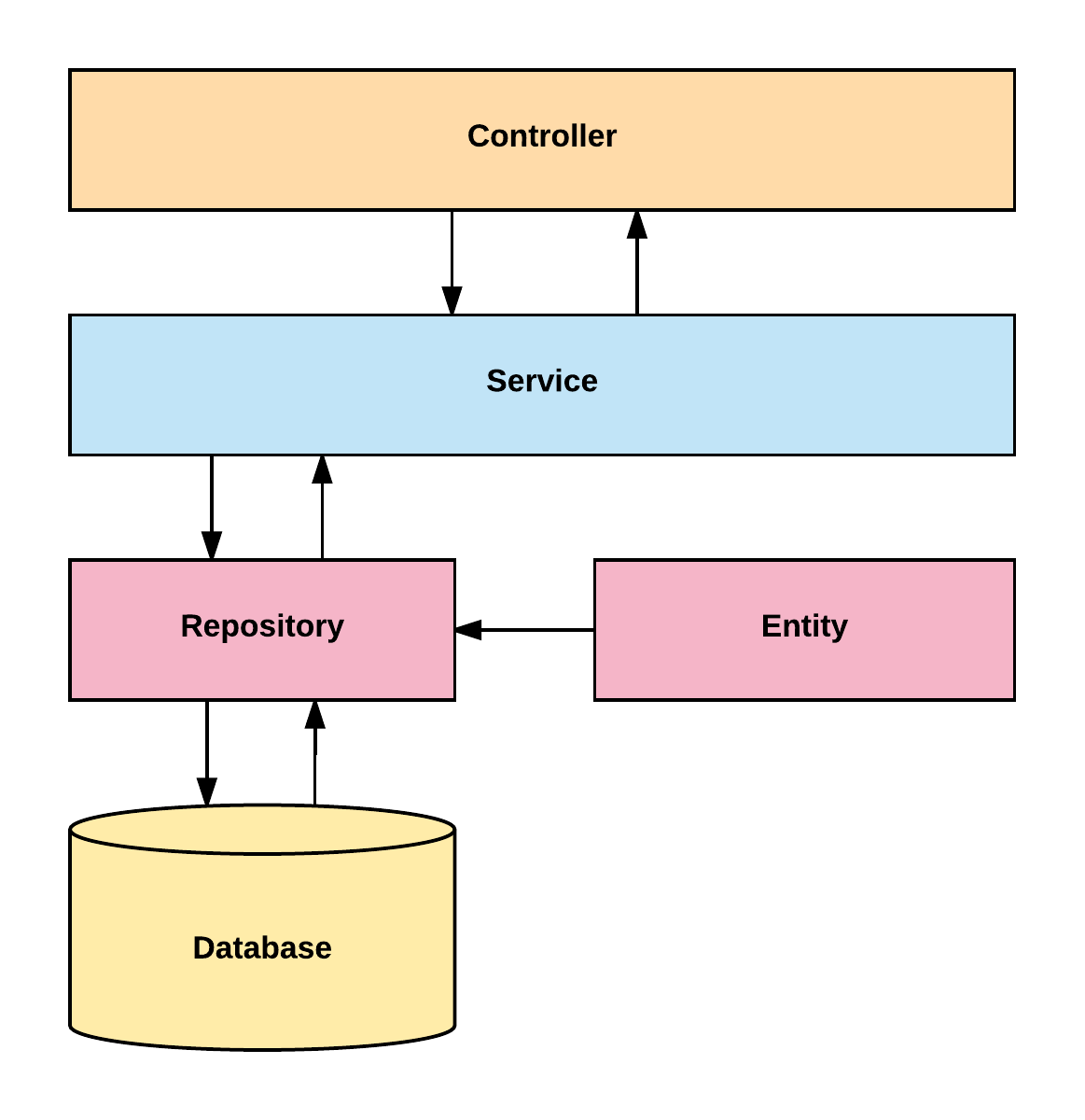
Model View Controller is the main design for the architecture. We are separating the main three parts of the application: user interaction, processing of information and storage. The segregation of this concerns favors production since each of the developers work in a separate section. This separation of components makes the system easily modifiable in the future. With this design, we ensure that we have multiple views for a controller; the system produces APIs that can be consumed by any other application.

Client-Server is used in the system to deliver the application. A highly available system that can be consumed from several parties called for this design. It allows the centralization of the code logic and database. We had to access the application from the web, iOS and Android devices so this was a very good choice.

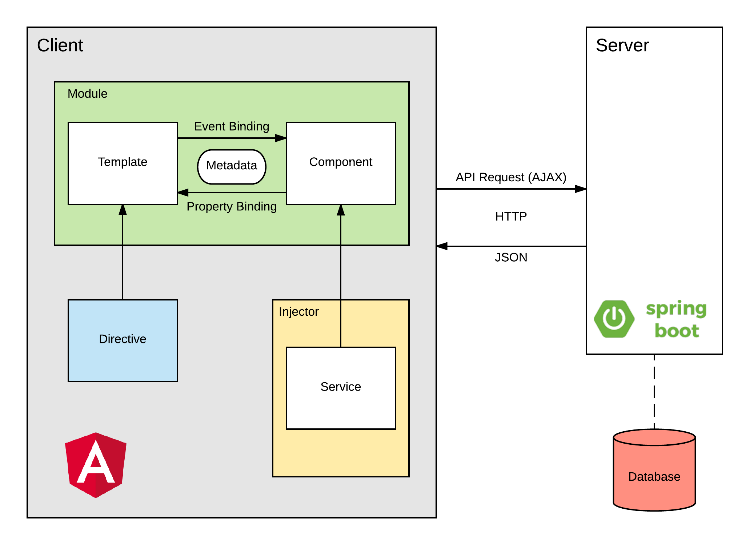


*Figure 2 Model View Controller*

Repository pattern is another design used in our system. It minimizes the amount of duplicate code in the system by abstracting the basic CRUD operations. It also ties the data entities to the domain model which favors development. The code would have to comply with the entity restrictions in order to even comply. It also helps maintain data integrity.



*Figure 3- Server System Design*



*Figure 4- Client System Design*

**System and Subsystem Decomposition**

The system is made out of two major subsystems and two minor ones. They are broken down as follows according to its tasks and interactions.

Server Subsystem:

* Interacts with the database.
* Serves all the requests from the web
* Handles server side security.
* Analyzes the statistics of the text
* Performs OCR
* Optimizes images.

Client Subsystem:

* Displays the application
* Routes server-side API calls
* Contains the main boundaries for user interaction.
* Client side data validation.

Android Subsystem:

* Interacts with the web application to display in an Android device.
* Handles device data storage needed for the web.

iPhone Subsystem:

* Interacts with the web application to display it in an iOS device.

**Deployment Diagram**

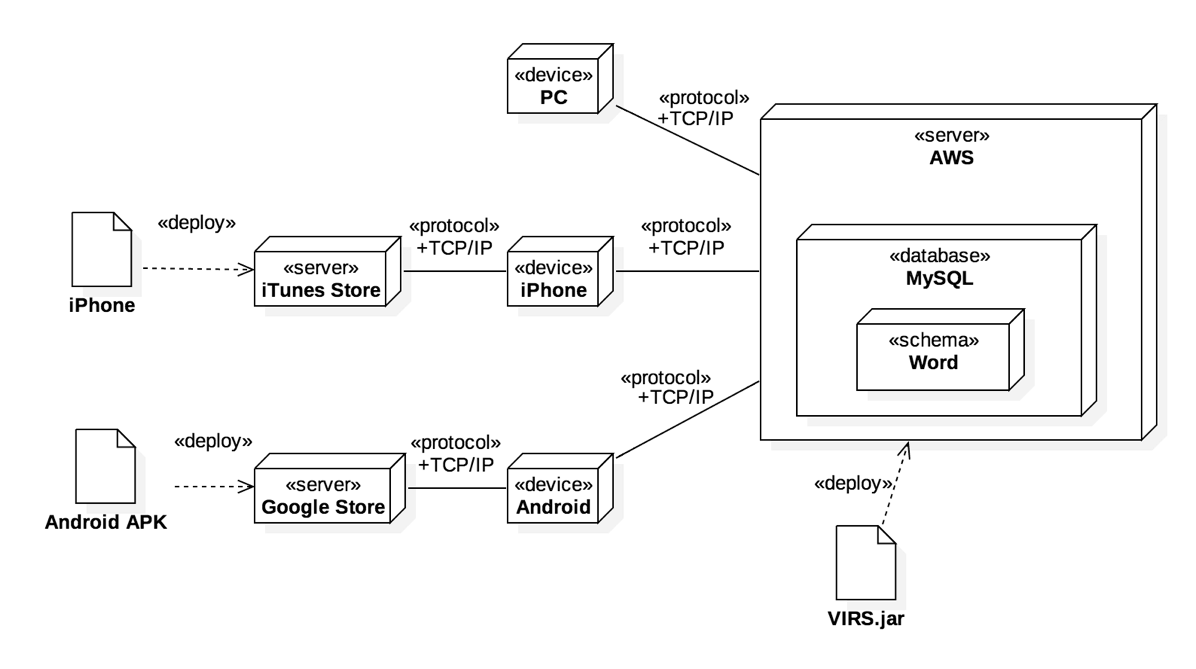
Deployment of the application consists in several steps run with a bash script. It starts in the front end and it propagates to the backend and eventually to the final product. We are building with ng for Angular products and Maven for the backend.

This is detailed explanation for the development pipeline.

* Angular
  + Clean and build the application
  + Run karma tests
  + Run e2e
* Copy resources to the backend
* Spring
  + Clean and build
  + Run Unit tests
  + Create jar executable

This is detailed explanation for the production pipeline.

* Angular
  + Clean and build the application
  + Run karma tests
  + Run e2e
* Copy resources to the backend
* Spring
  + Increase version number
  + Clean and build
  + Run Unit tests
  + Run Integration tests
  + Create jar executable
* Bundle the application for AWS
* Upload to AWS servers.



*Figure 5 - Deployment Diagram*

**Design Patterns**

The following design patterns were used in the application.

**Dependency Injection:** This is a core design pattern for Spring and Angular 4. It allows the objects to be readily available and injected when needed rather than having to create them.

**Bridge:** This design pattern is used by the application to interact with libraries that are not native. Ex: tesseract and OpenCV

**Front controller:** Controllers are one of the main Components in the MVC design.

**Marker:** All components in Spring are annotated with markers to further specify the correct stereotype. This allows the initialization of the correct optimized components when the application runs.

**Module:** The application is divided in several modules according to functionality. This separation promotes organization and grouping of features.

# **System Validation**

## Test Suite

* Test case ID: TestsComponents\_ShouldCreate\_001
* Test case ID: iTranslator\_Hello\_toSpanish\_001
* Test case ID: BeginnerComponents\_ShouldCreate\_001
* Test case ID: Itranslator\_request\_001
* Test case ID: BeginnerComponents\_ShouldCreate\_001
* Test case ID: TestComponents\_ShouldStopTest\_001
* Test case ID: Add\_words\_multiple\_categories001
* Test case ID: User\_Registration\_Success
* Test case ID: User\_Login\_001
* Test case ID: Load\_User\_Info
* Test case ID: Translation\_Limit\_001
* Test case ID: Delete\_User\_Info
* Test case ID: Delete\_User\_Info
* Test case ID: Paypal\_Donate\_001

## Test Cases

* Test case ID: TestsComponents\_ShouldCreate\_001
* Description/Summary of Test: The Tests component should be created
* Pre-condition: The server must be running
* Expected Results: Pass
* Actual Result: Pass
* Status (Fail/Pass): Pass
* Test case ID: iTranslator\_Hello\_toSpanish\_001
* Description/Summary of Test: Testing whether the string ‘hello’ is translated to Spanish
* Pre-condition: Available backend server to send requests to. User to click on iTranslate option in sidebar.
* Expected Results: ‘hola’
* Actual Result: ‘hola’
* Status (Fail/Pass):PASS
* Test case ID: BeginnerComponents\_ShouldCreate\_001
* Description/Summary of Test: The Tests component should be created
* Pre-condition: The server must be running
* Expected Results: Pass
* Actual Result: Pass
* Status (Fail/Pass): Pass
* Test case ID: BeginnerComponents\_ShouldCreate\_001
* Description/Summary of Test: The Tests component should be created
* Pre-condition: The server must be running
* Expected Results: Pass
* Actual Result: Pass
* Status (Fail/Pass): Pass
* Test case ID: TestComponents\_ShouldStopTest\_001
* Description/Summary of Test: The test should stop
* Pre-condition: The server must be running
* Expected Results: Pass
* Actual Result: Pass
* Status (Fail/Pass): Pass
* Test case ID: Itranslator\_request\_001
* Description/Summary of Test: Frontend requests ‘hello’ translation to backend to Spanish
* Pre-condition: Frontend to be functional for request, and google translation servers to be available
* Expected Results: return a json file:
  + {original: hello

target: es

translated: hola}

* Actual Result:
  + {original: hello

target: es

translated: hola}

* Status (Fail/Pass): Pass
* Test case ID: Add\_words\_multiple\_categories001
* Description/Summary of Test:Admin uploads csv file with 20 words and different categories is uploaded to be added to database
* Pre-condition: Admin is logged on and csv is created with 20 words and different categories
* Expected Results: System displays the words added to database
* Actual Result: System displays the words added to database
* Status (Fail/Pass): PASS
* Test case ID: User\_Registration\_Success
* Description/Summary of Test: User must be registered with provided information
* Pre-condition: The server must be running, user must be in register page
* Expected Results: Pass
* Actual Result: Pass
* Status (Fail/Pass):Pass
* Test case ID: User\_Login\_001
* Description/Summary of Test: User should login successfully
* Pre-condition: User must have an account registered with the system
* Expected Results: Pass
* Actual Result: Pass
* Status (Fail/Pass): Pass
* Test case ID: Load\_User\_Info
* Description/Summary of Test: User Info should be loaded into the profile page
* Pre-condition: User must have an account registered with the system and be logged in
* Expected Results: Pass
* Actual Result: Pass
* Status (Fail/Pass): Pass
* Test case ID: Translation\_Limit\_001
* Description/Summary of Test: Make a translation with the string ‘hello’ and see if the chars left goes down by 5 because ‘hello’ has 5 characters.
* Pre-condition: Be in the iTranslate page for the first time
* Expected Results: Characters left to update should be 995
* Actual Result: Characters left to update should be 995
* Status (Fail/Pass): Pass
* Test case ID: Delete\_User\_Info
* Description/Summary of Test: User Info should be deleted from the DB
* Pre-condition: Admin mist be logged on and on the user API
* Expected Results: Pass
* Actual Result: Pass
* Status (Fail/Pass): Pass
* Test case ID: Paypal\_Donate\_001
* Description/Summary of Test: 1. Click donate button

2. Go donate $1 in paypal process

3. User is taken back to site

* Pre-condition: Be in dashboard of the site
* Expected Results: Return back to dashboard of the site
* Actual Result: Return back to dashboard of the site
* Status (Fail/Pass): Pass

# **Glossary**

**Academic Word List (AWL):** List of words used in the natural English language with frequency high enough but that does not make it to the high frequency list.

**High Frequency List:** List of words used in natural English language with a high frequency.

**Medium Frequency List:** List of words used in natural English language with a medium frequency.

**Low Frequency List:** List of words used in natural English language with a low frequency.

**Flesch Reading Ease Score:** A test designed to calculate how hard a text is to understand in English.

**Word Definition:** Meaning of a word as per Wiki Dictionary. It contains etymology, meaning, and usage information.

**Category:** The assigned value to a word from one of the above lists. A word category can be: AWL, High Frequency, Medium Frequency, and Low Frequency.

**Inflection:** A modification if a word to express additional meanings: plural and conjugations.

**School Dictionary:** A collection of all the lists. It can be used as reference for the word categories.

# 

# **Appendix**

## Appendix A - UML Diagrams

## 

Figure - Test Class Diagram

## 

Figure - Tests use cse diagram

## 

Figure - Delete user sequence diagram

## 

Figure - Layout Class Diagram

## 

Figure - Delete user use case diagram

## 

Figure - Display tests sequence diagram

## 

Figure - Display tests use case diagram.

## 

Figure - Display test class duagram

## 

Figure - Finish test sequence diagram

## 

Figure - Decrease attempt use case diagram

## 

Figure - Decrease attempt sequence diagram

## 

Figure - Login use case diagram

## 

Figure - Login sequence diagram

## 

Figure - User registration class diagram

## 

Figure - User profile class diagram

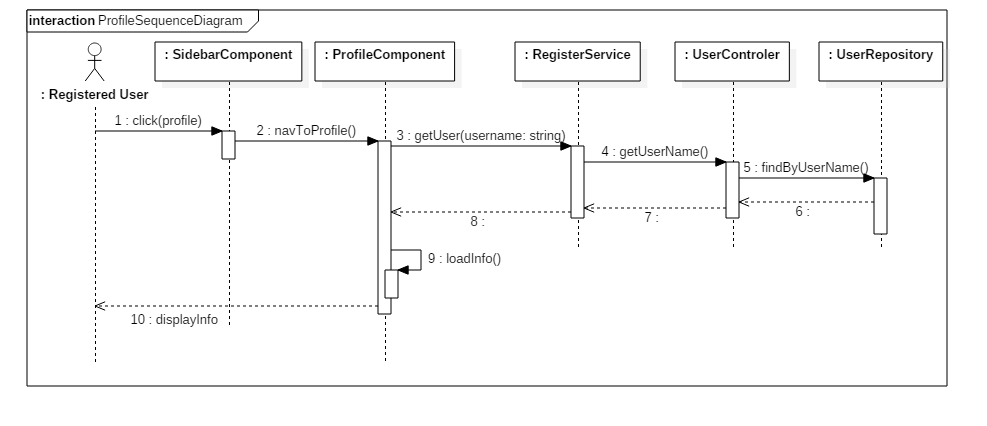


Figure - User profile sequence diagram

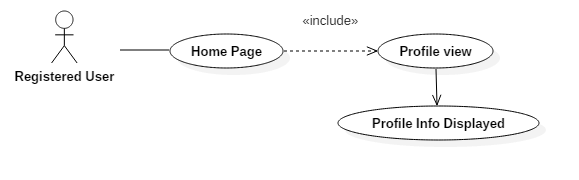


Figure User profile use case diagram

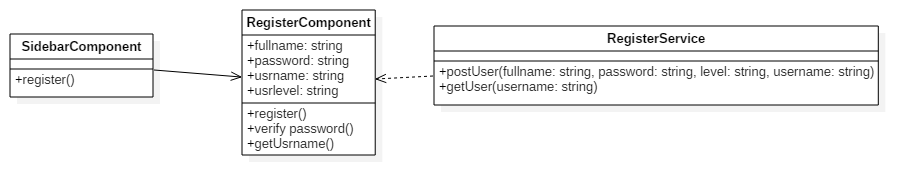


Figure - User Registration class diagram

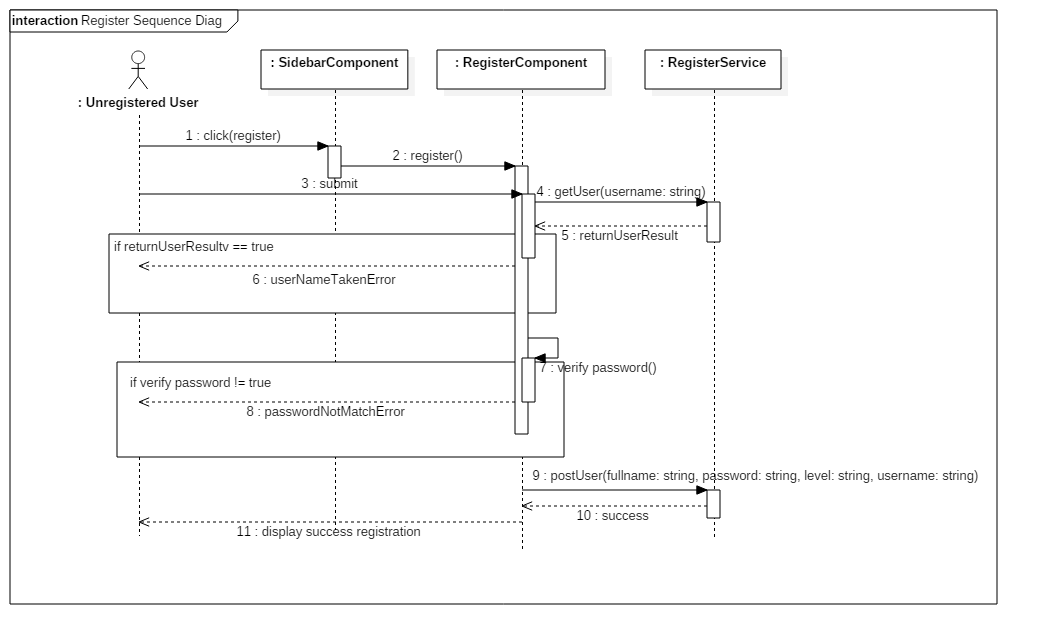


Figure - User registration sequence diagram

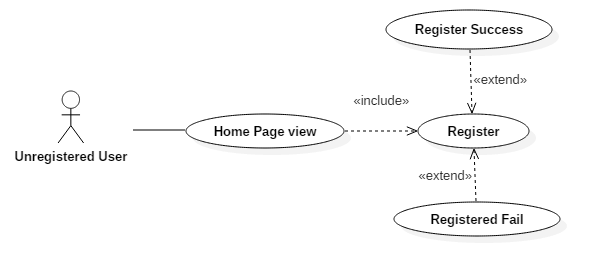


Figure - User registration use case diagram

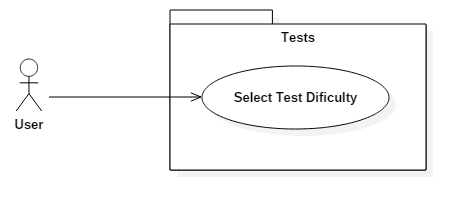


Figure - Display tests categories use case diagram

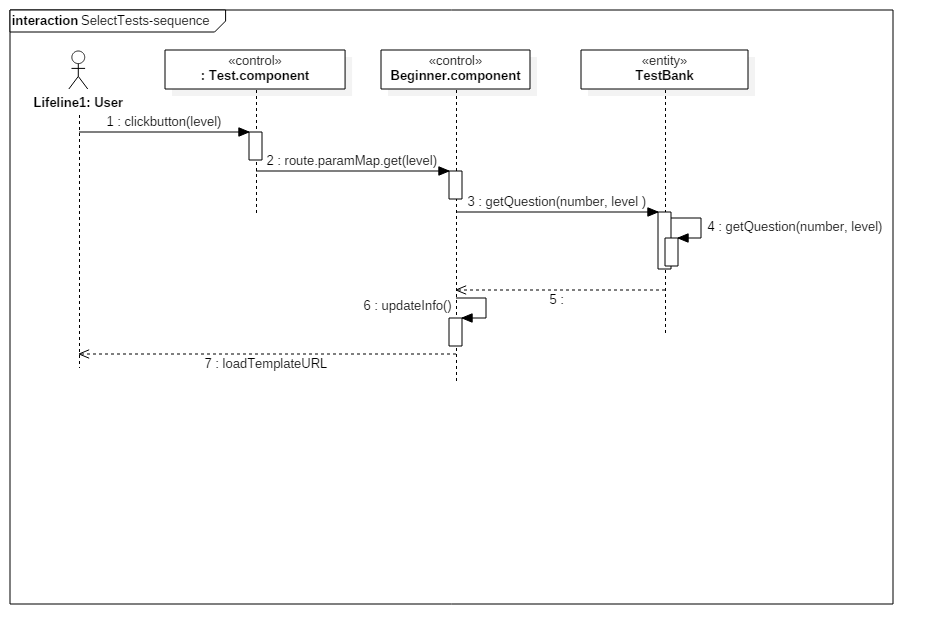


Figure - Test selection sequence diagram

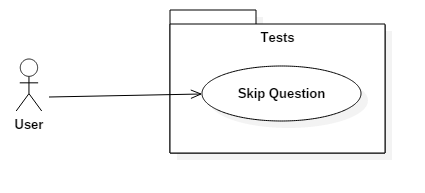


Figure - Skip question use case diagram

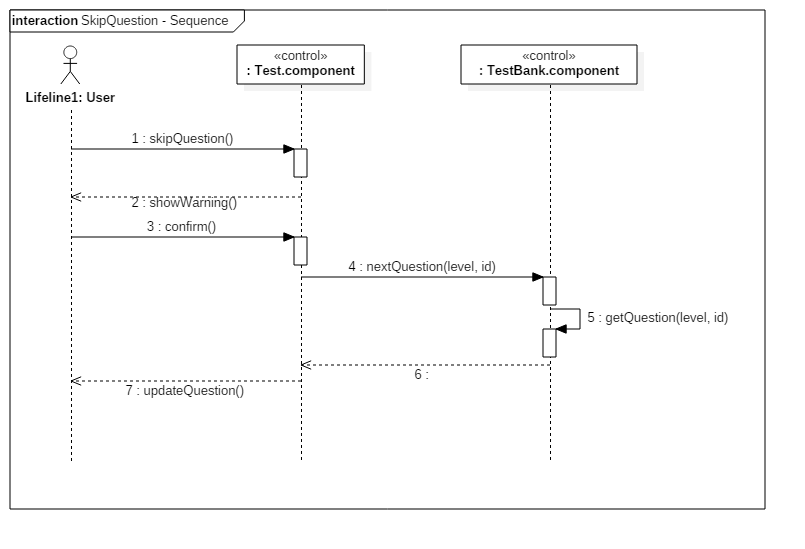


Figure - Skip question sequence diagram

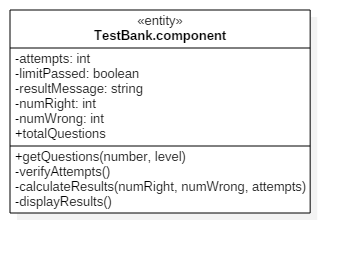


Figure - Test bank class diagram

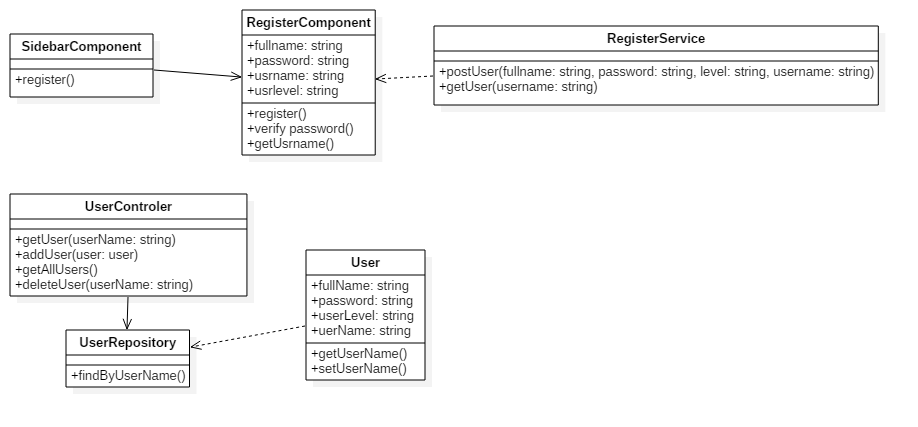


Figure - User Registration class diagram

## 

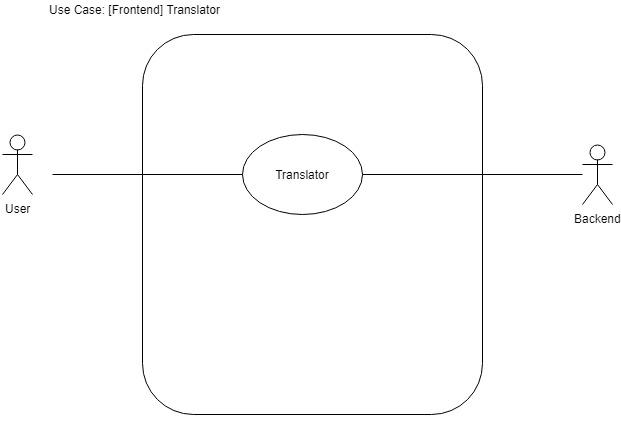


Figure - Translator use case diagram

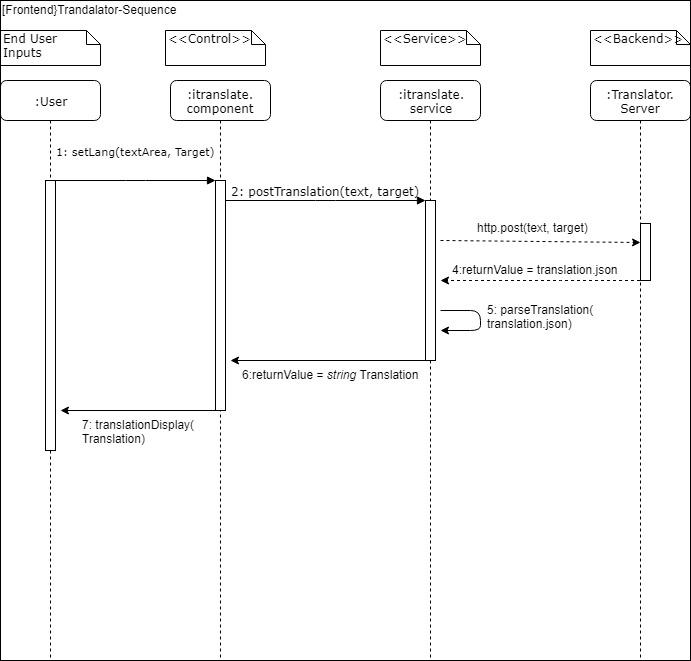


Figure - Translator sequence diagram

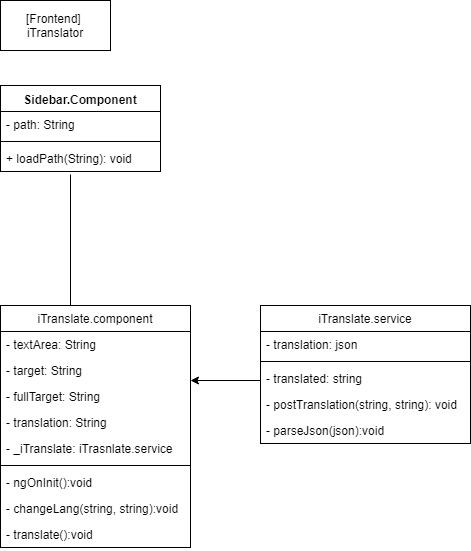


Figure - Translator class diagram

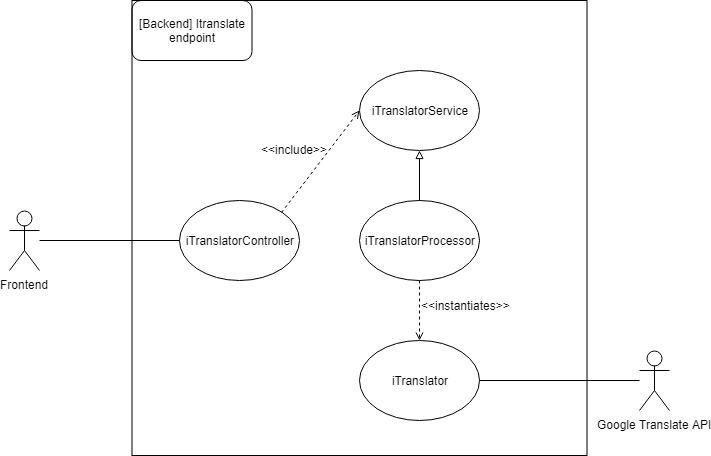
****

Figure - Translate endpoint use case diagram

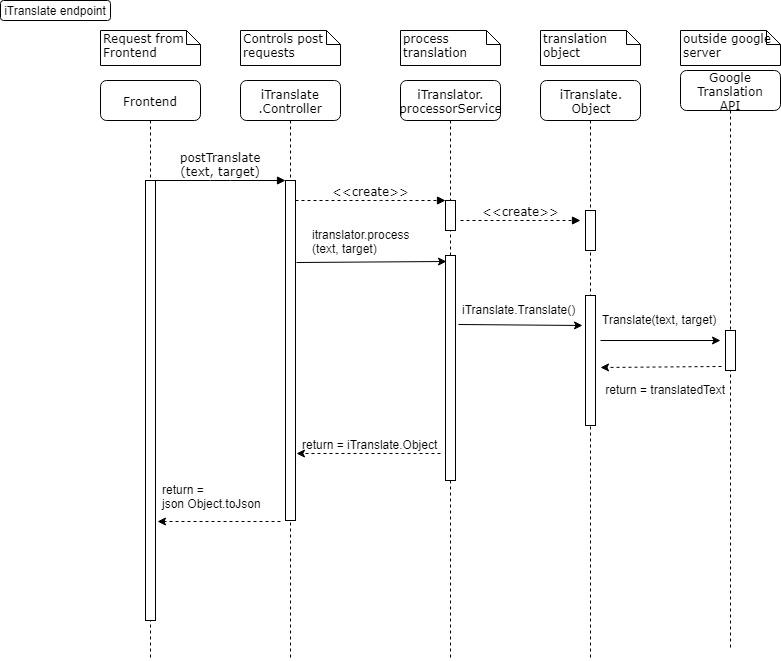


Figure - Translation endpoint sequence diagram

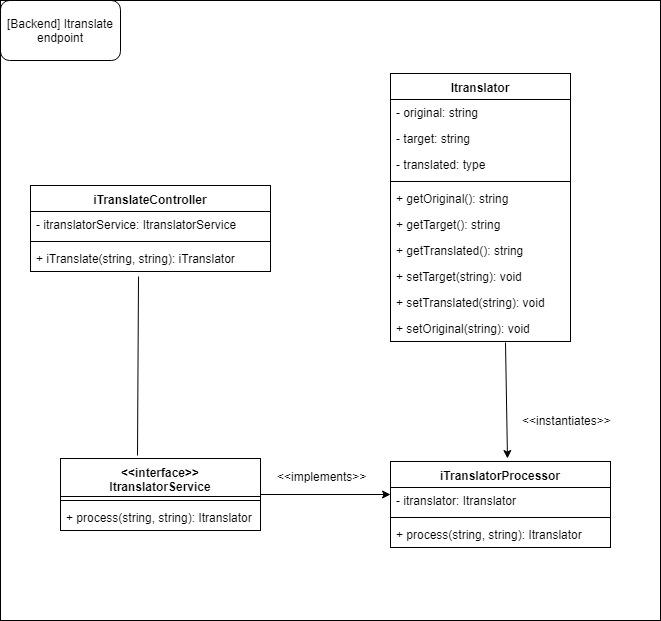
****

Figure - Translate endpoint class diagram

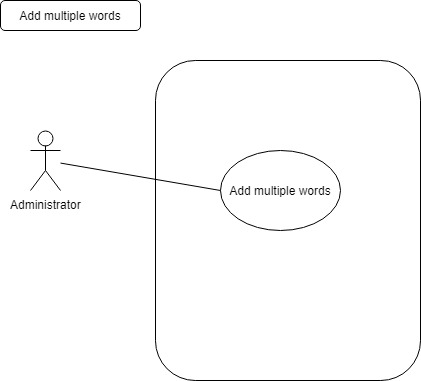


Figure - Add miltiple words use case diagram

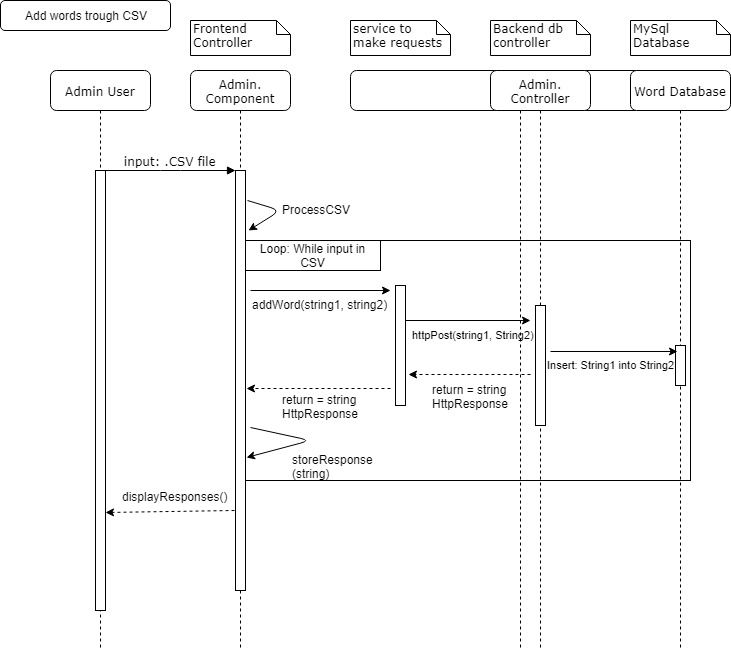


Figure - Add words thru CSV sequence diagram

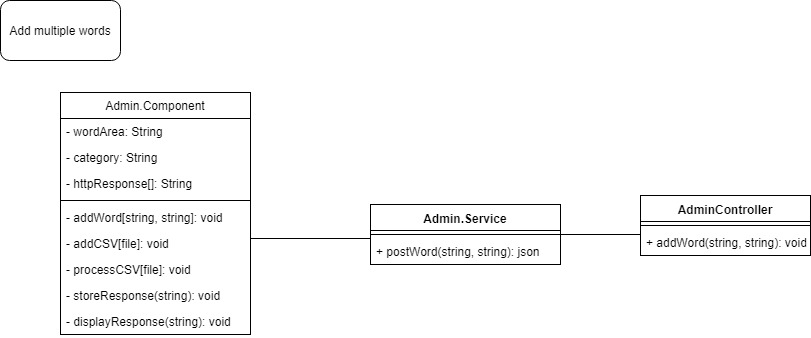


Figure - Add multiple words class diagram

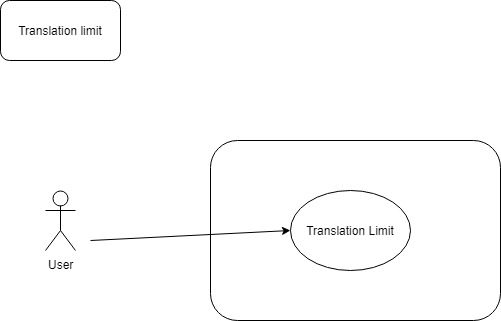


Figure - Translation limit use case diagram

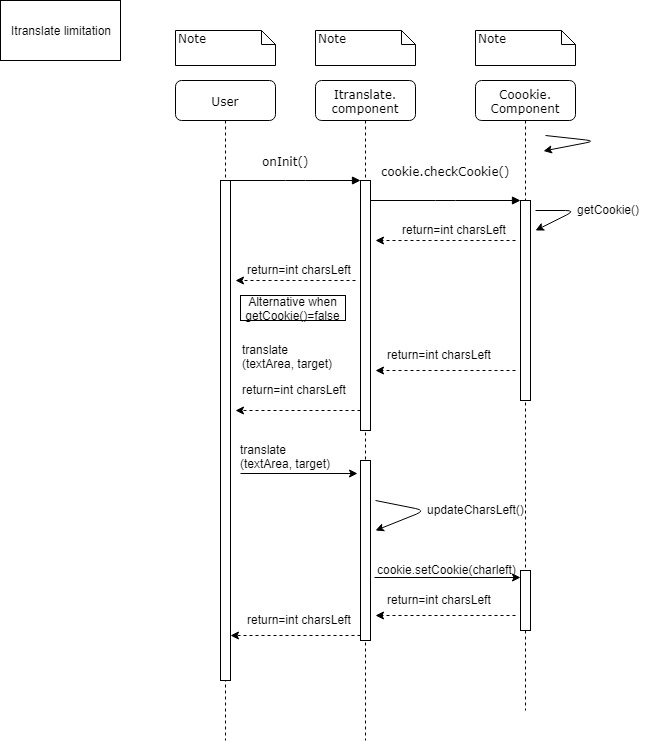


Figure - Translate limitation sequence diagram

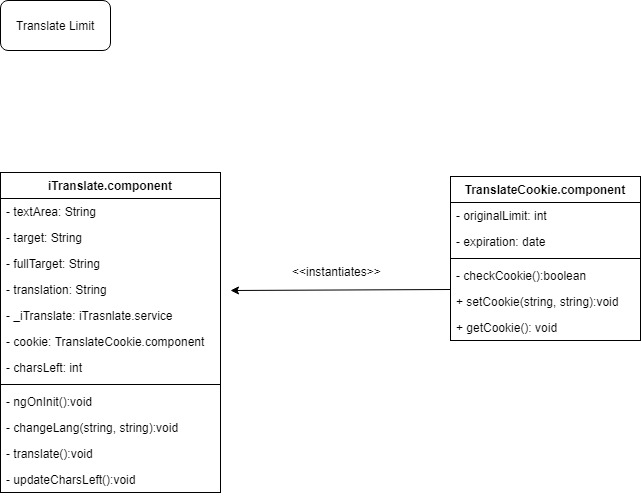
****

Figure - Translate limit class diagram

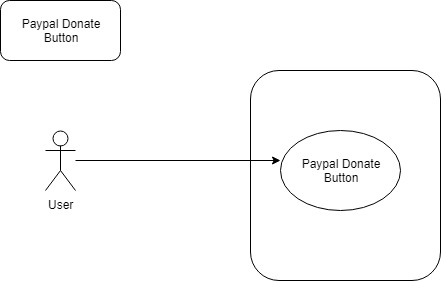


Figure - Paypal donate button use case diagram

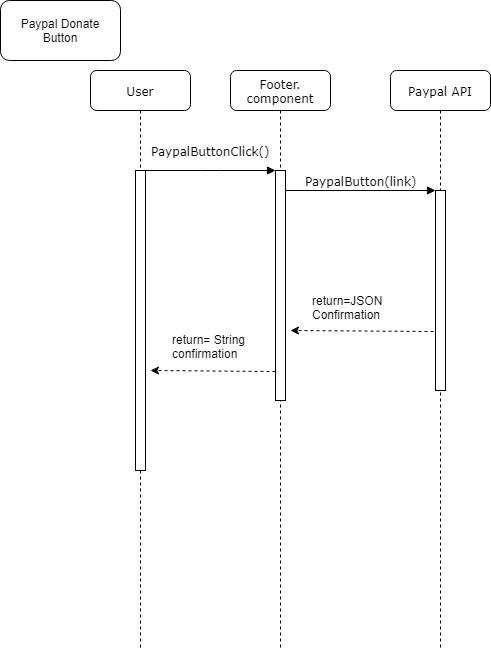


Figure - Paypal donate button sequence diagram

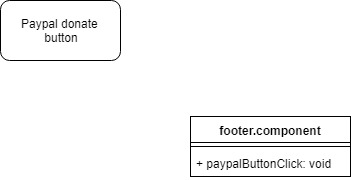


Figure - Paypal donate button class diagram

## Appendix B - User Interface Design

The frontend for this application was developed using ng-Bootstrap, HTML5, Bootstrap, Typescript and CSS

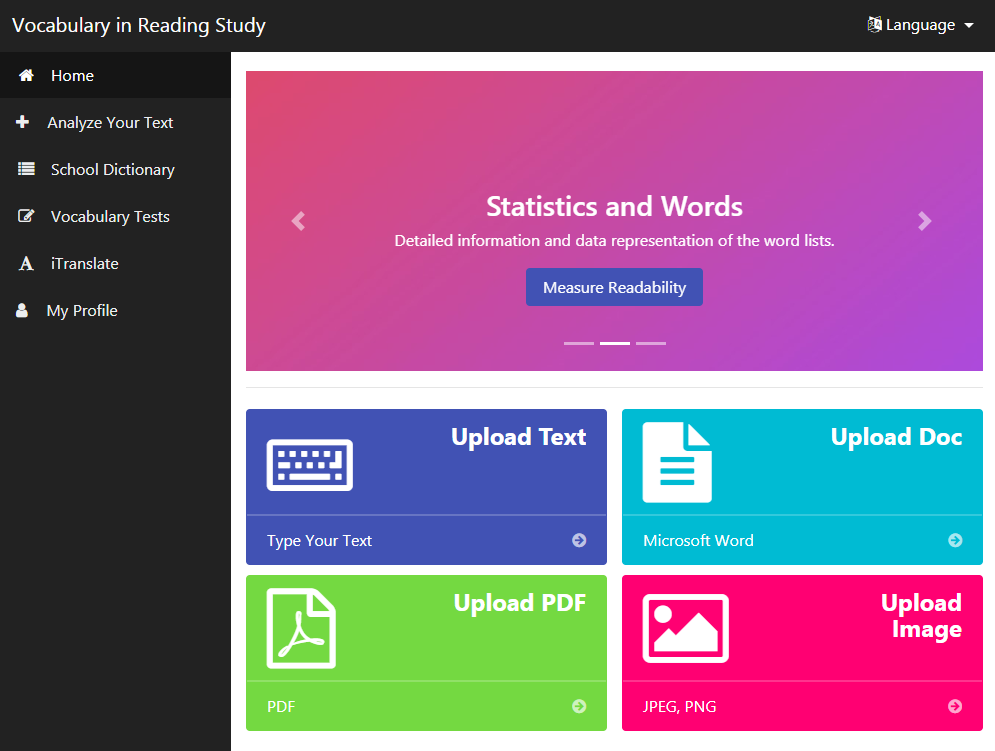


Figure - VIRS 3.0 home page

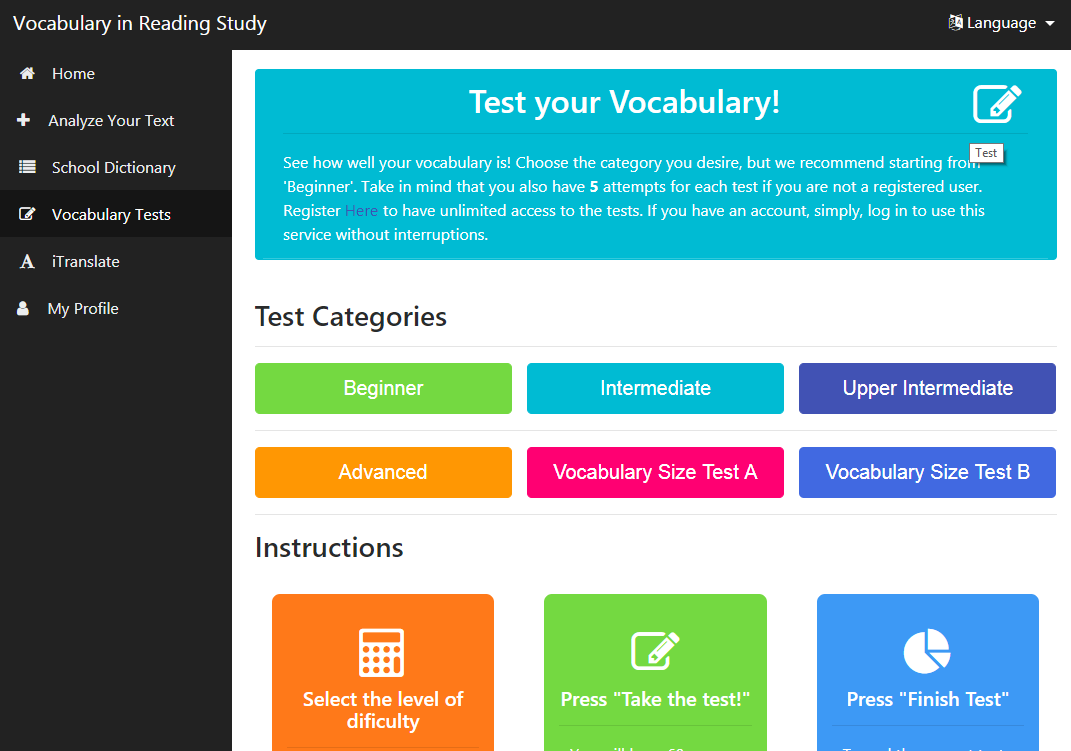


Figure - Test section displaying test categories

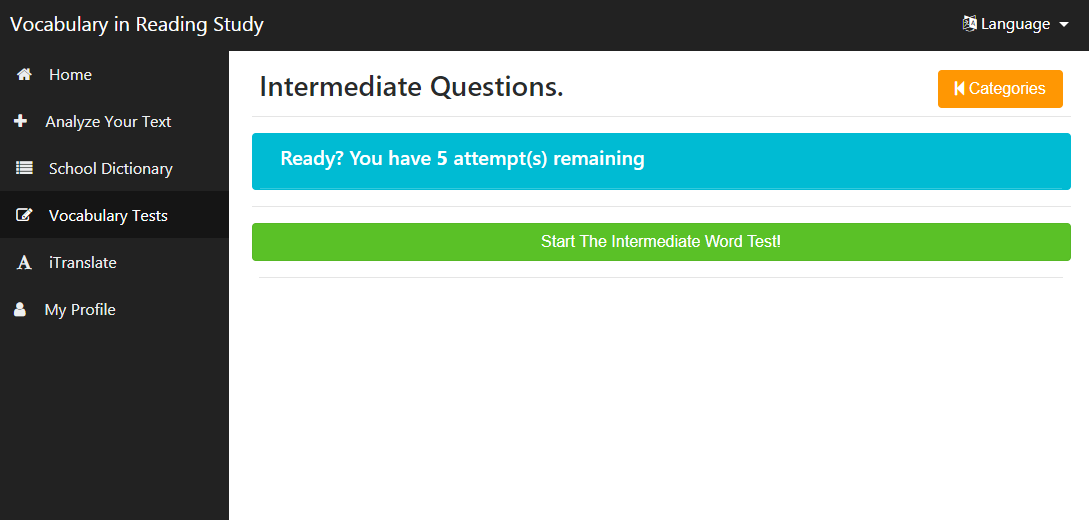


Figure - Ready state for a test category

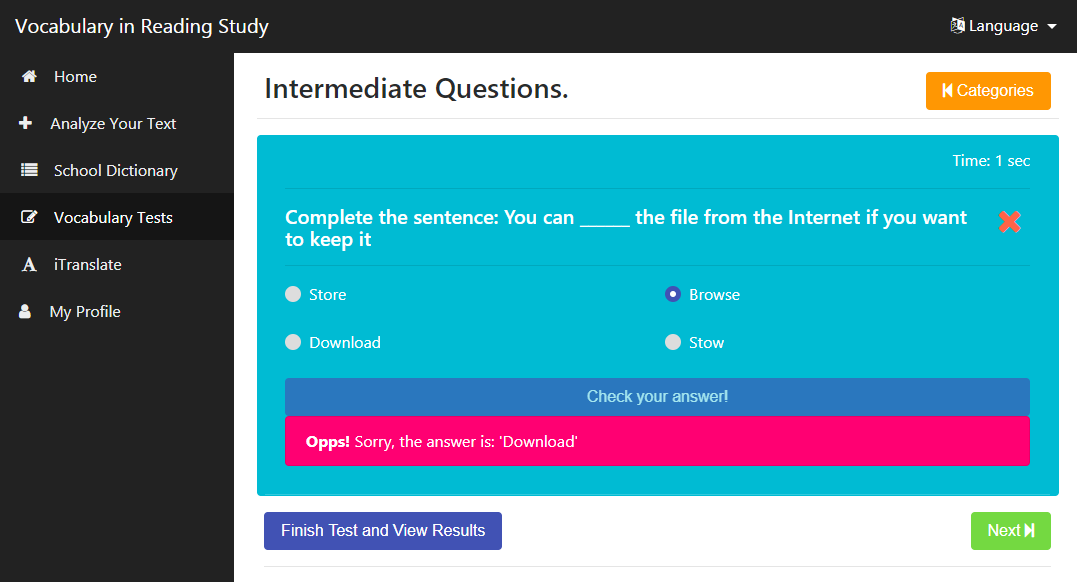


Figure - Test providing feedback on wrong answer

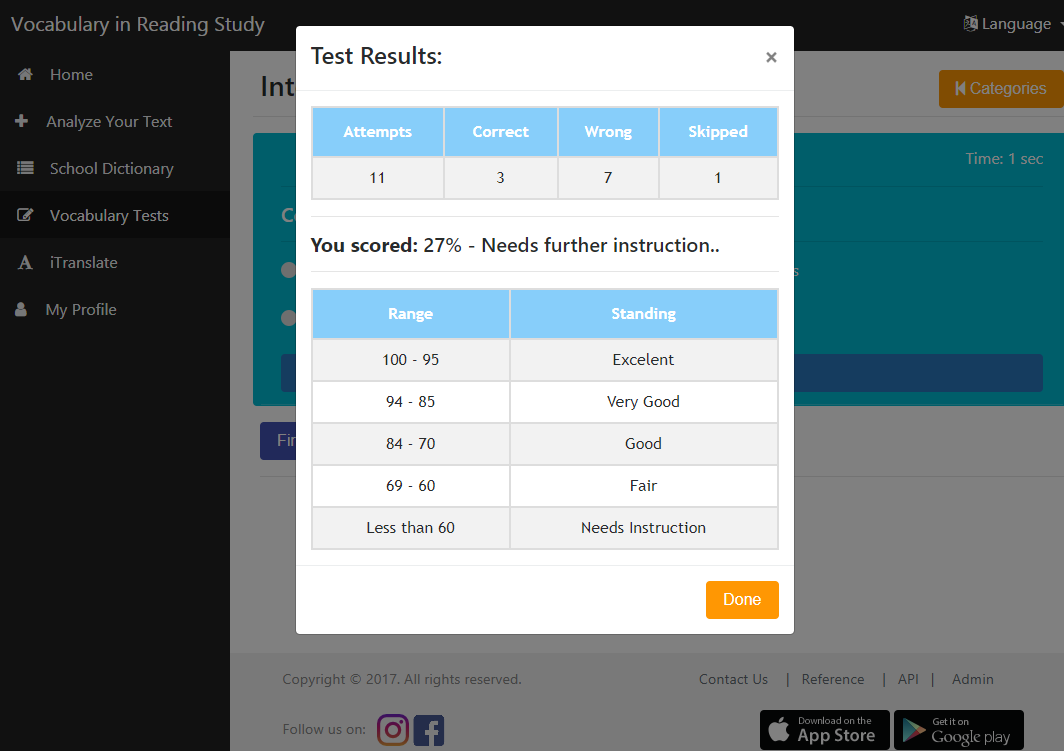


Figure - System providing overall performance on test session

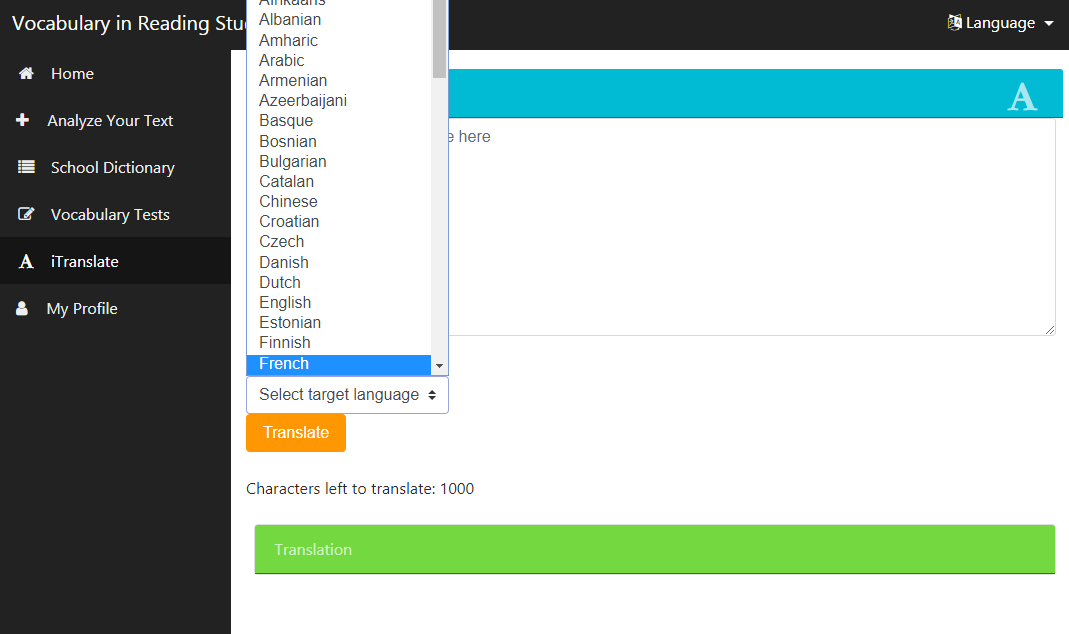


Figure - ITranslator with a menu of available languages

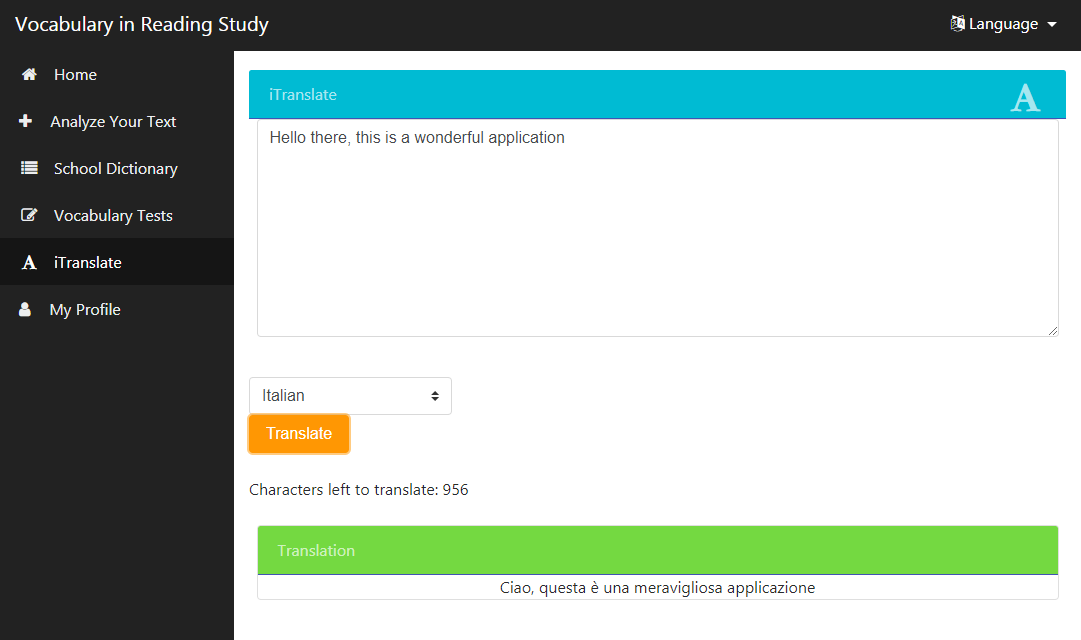


Figure - Sample text translated using the ITranslator

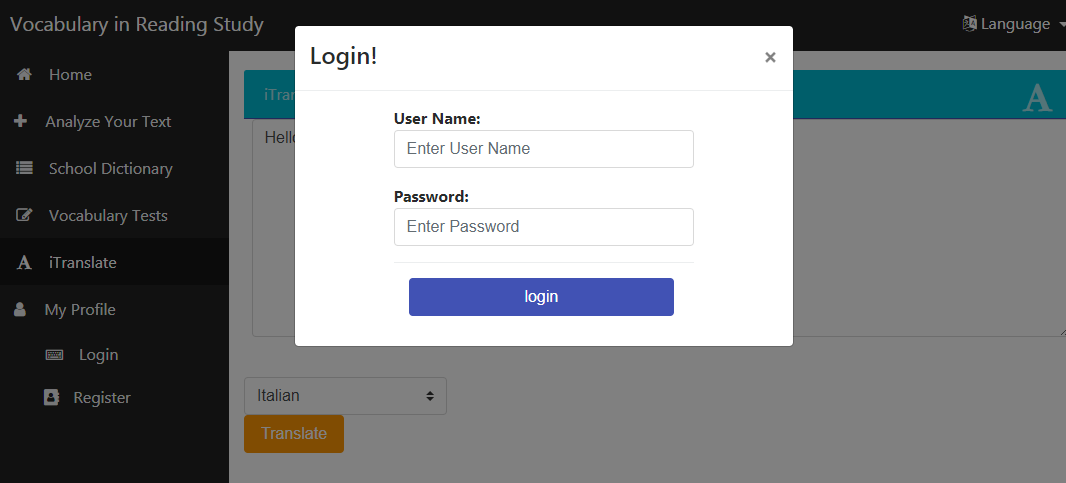


Figure - Login page

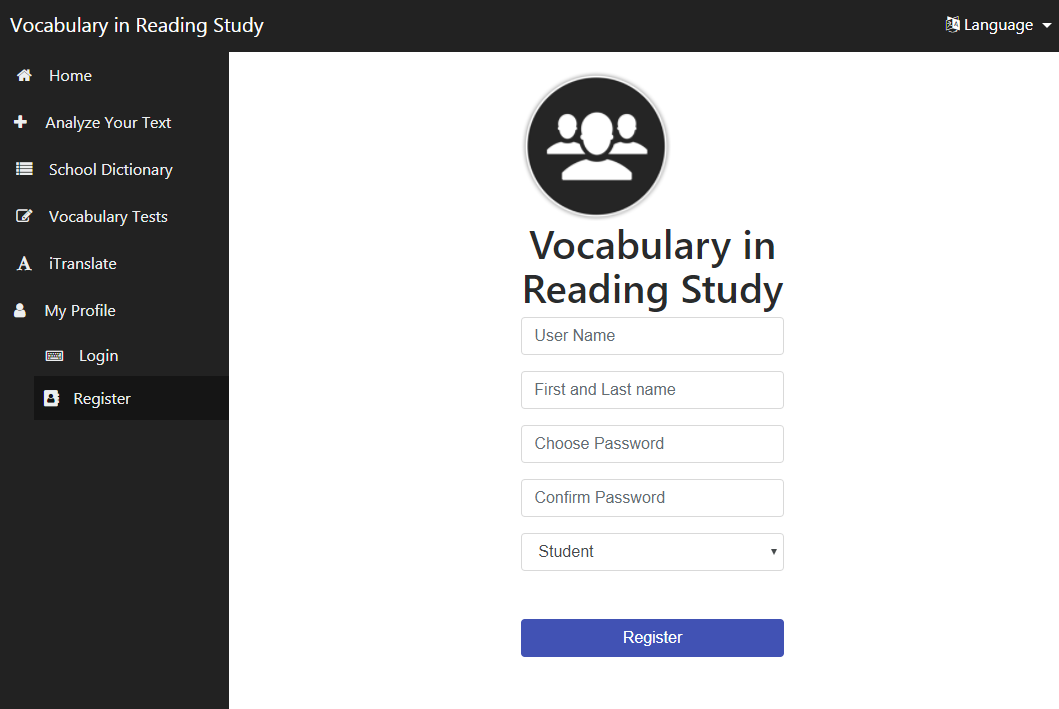


Figure - Registration page

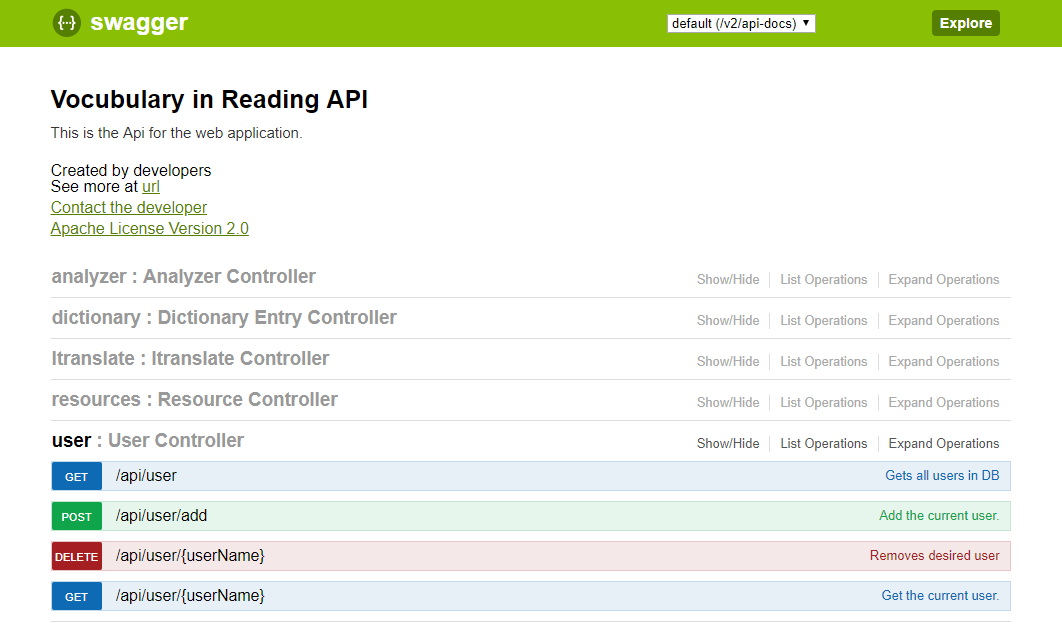


Figure - API list showing the added user controller API

## Appendix C - Sprint Review Reports

**Sprint 1 Review Meeting Minutes**

Attendees: <Christian Hidalgo, Juan Carlos Valladares>

Start time: <6.30pm>

End time: <7.00pm>

After a show and tell presentation, the implementation of the following user stories were accepted by the product owners: All.

* User Story: [#260] [Frontend] Add STEM word category
* User Story: [#268] [Frontend] Add Tests section to main page
* User Story: [#269] [Frontend] Add iTranslate Section

The following ones were rejected and moved back to the product backlog to be assigned to a future sprint at a future Sprint Planning meeting.

* None

**Sprint 2 Review Meeting Minutes**

Attendees: <Christian Hidalgo, Juan Carlos Valladares>

Start time: <6.00pm>

End time: <7.00pm>

After a show and tell presentation, the implementation of the following user stories were accepted by the product owners: All.

* [#259] [Frontend] Add Facebook link
* [#153] [Frontend] Add Vocabulary Tests Section
* [#258] Fix .Docs upload
* [#257] Fix PDF upload
* [#265] [Frontend] Add Different Tests Levels
* [#266] [Frontend] Skip Questions
* [#267] [Frontend] Add external link to ck12 (School Library)
* [#267] [Frontend] Translator

The following ones were rejected and moved back to the product backlog to be assigned to a future sprint at a future Sprint Planning meeting.

* None

**Sprint 3 Review Meeting Minutes**

Attendees: <Christian Hidalgo, Juan Carlos Valladares>

Start time: <6.00pm>

End time: <7.00pm>

After a show and tell presentation, the implementation of the following user stories were accepted by the product owners: All.

* [#275] [Frontend] Update Credit Page
* [#276] [Backend] iTranslate endpoint
* [#152] [Frontend] Add Remaining Tests
* [#273] [Frontend] Finish Test
* [#274] [Frontend] School Library

The following ones were rejected and moved back to the product backlog to be assigned to a future sprint at a future Sprint Planning meeting.

* None

**Sprint 4 Review Meeting Minutes**

Attendees: <Christian Hidalgo, Juan Carlos Valladares>

Start time: <6.00pm>

End time: <7.00pm>

After a show and tell presentation, the implementation of the following user stories were accepted by the product owners: All.

* [#275] Add multiple words
* [#278] [Backend] Register User
* [#279] [Backend] Login User
* [#280] [Frontend] Account Page

The following ones were rejected and moved back to the product backlog to be assigned to a future sprint at a future Sprint Planning meeting.

* [#281] [Frontend] Edit Profile

**Sprint 5 Review Meeting Minutes**

Attendees: <Christian Hidalgo, Juan Carlos Valladares>

Start time: <6.00pm>

End time: <7.00pm>

After a show and tell presentation, the implementation of the following user stories were accepted by the product owners: All.

* User Story: [#282] [Frontend] Graph colors match with display
* User Story [#283] Itranslate limitations
* User Story: [#284] User Registration API

The following ones were rejected and moved back to the product backlog to be assigned to a future sprint at a future Sprint Planning meeting.

* none

**Sprint 6 Review Meeting Minutes**

Attendees: <Christian Hidalgo, Juan Carlos Valladares>

Start time: <6.00pm>

End time: <7.00pm>

After a show and tell presentation, the implementation of the following user stories were accepted by the product owners: All.

* User Story: [#285] [Frontend] - Limit Test Attempts (Unregistered)
* User Story: [#286] [Backend] - Fix Wikipedia API

The following ones were rejected and moved back to the product backlog to be assigned to a future sprint at a future Sprint Planning meeting.

* User Story: [#288] - [Frontend] - Donation button through paypal
* User Story: [#287] [Frontend] - Add WAT tests

## Appendix D - User Manuals, Installation/Maintenance Document, Shortcomings/Wishlist Document and other documents

### Manuals

These are all the manuals for the system.

#### Creating AWS Environment From Scratch

Elastic Beanstalk Dashboard

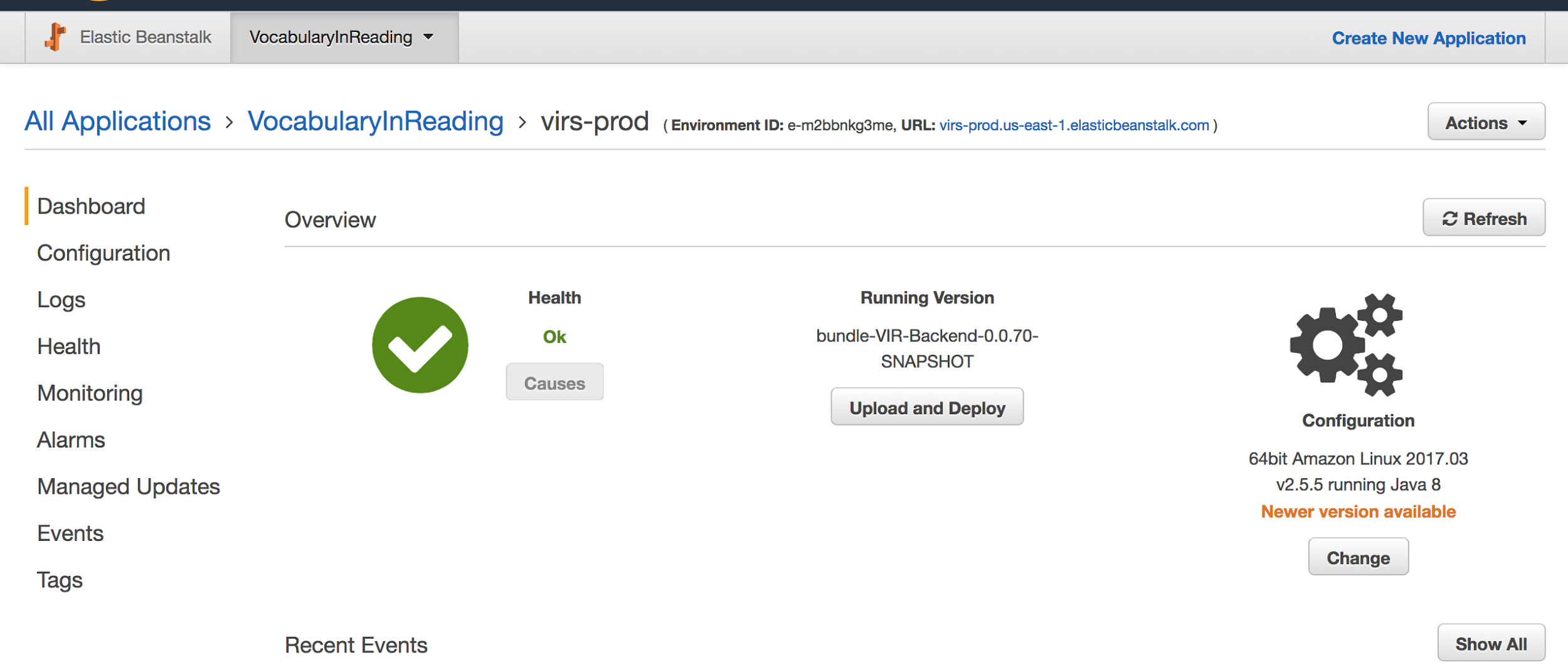


Figure 52 - AWS Deployment Dashboard

* If needed create a new Application.
* Create a new environment with all the default configurations and the sample application.
* Under Config got ahead and create a database with the "username" and "password" (Note the password has to be long, keep in mind that this password will be encrypted later on)
* Under Config/Software Configuration add a new Property name with the name used in the java EncryptorConfig.java for the name and the value.

RDS Dashboard

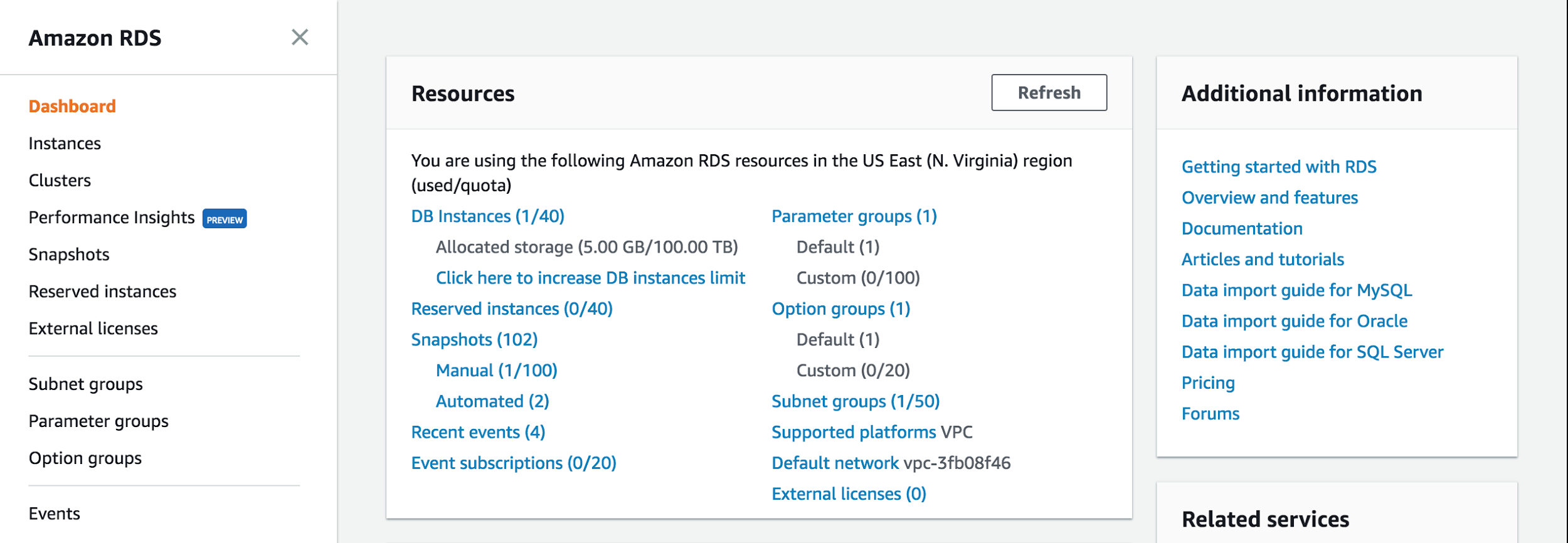


Figure 53- RDS Dashboard

* Go to the database associated with the beanstalk above.
* Select it and click Instance Details.
* Click on the security groups: the one that starts with "rds-aws...."
* There go to the "InBound" tab at the bottom and add a new TCP protocol with port 3306 and pick your ip address. (This will allow to connect using workbench)

Workbench

* Login with the credentials and the connection string listed on the RDS dashboard.
* Create a new database.
* Populate any data needed.

Java IDE

* Add properties for the database, encryptor place holder, and port 5000 (needed for production)
* Compile and generate JAR.

Elastic Beanstalk Dashboard

* Upload the new version of the application.

EC2 Dashboard

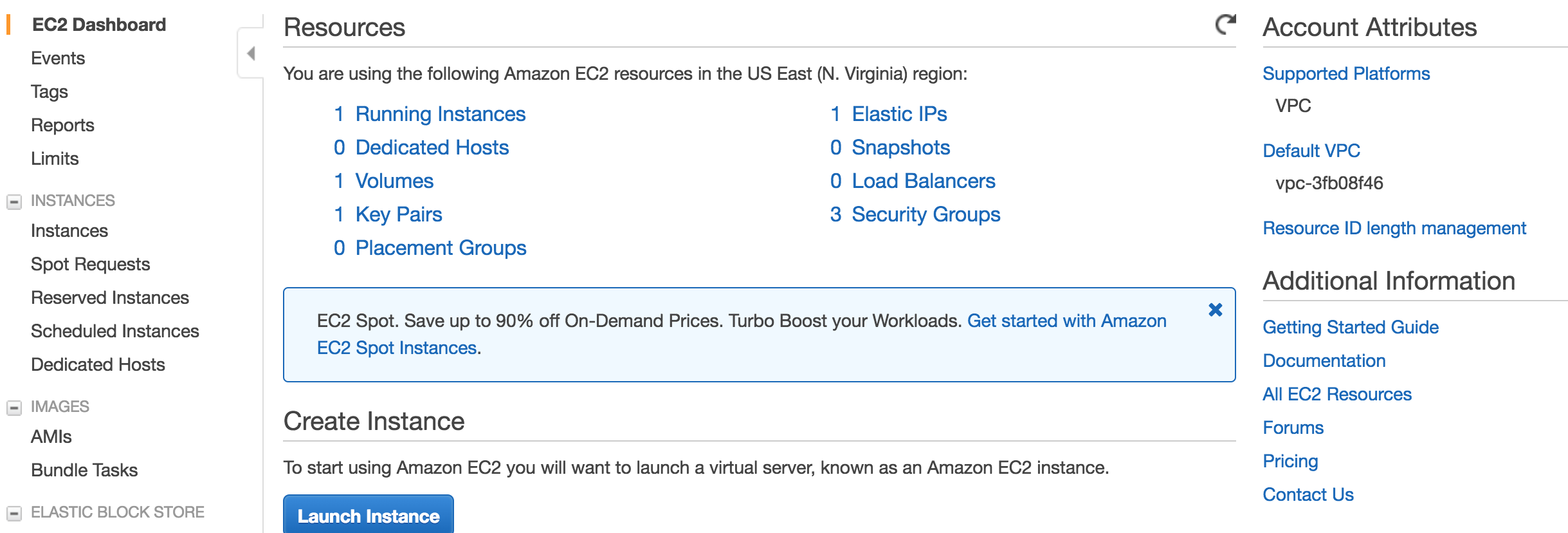


Figure 54- EC2 Dashboard

* If you do not have a .pem file. Create a key to log in via ssh.
* Click 'Key pairs' to go to the section of the keys. (from here is self explanatory...)
* Locate the EC2 instance and right click on it to get the ssh connection information. (for the key to work it has to be associate with the BeanStalk instance, this should be already done, but you can do this in the 'Instance settings' of the Beanstalk)

#### 

#### Building and Deploying the application

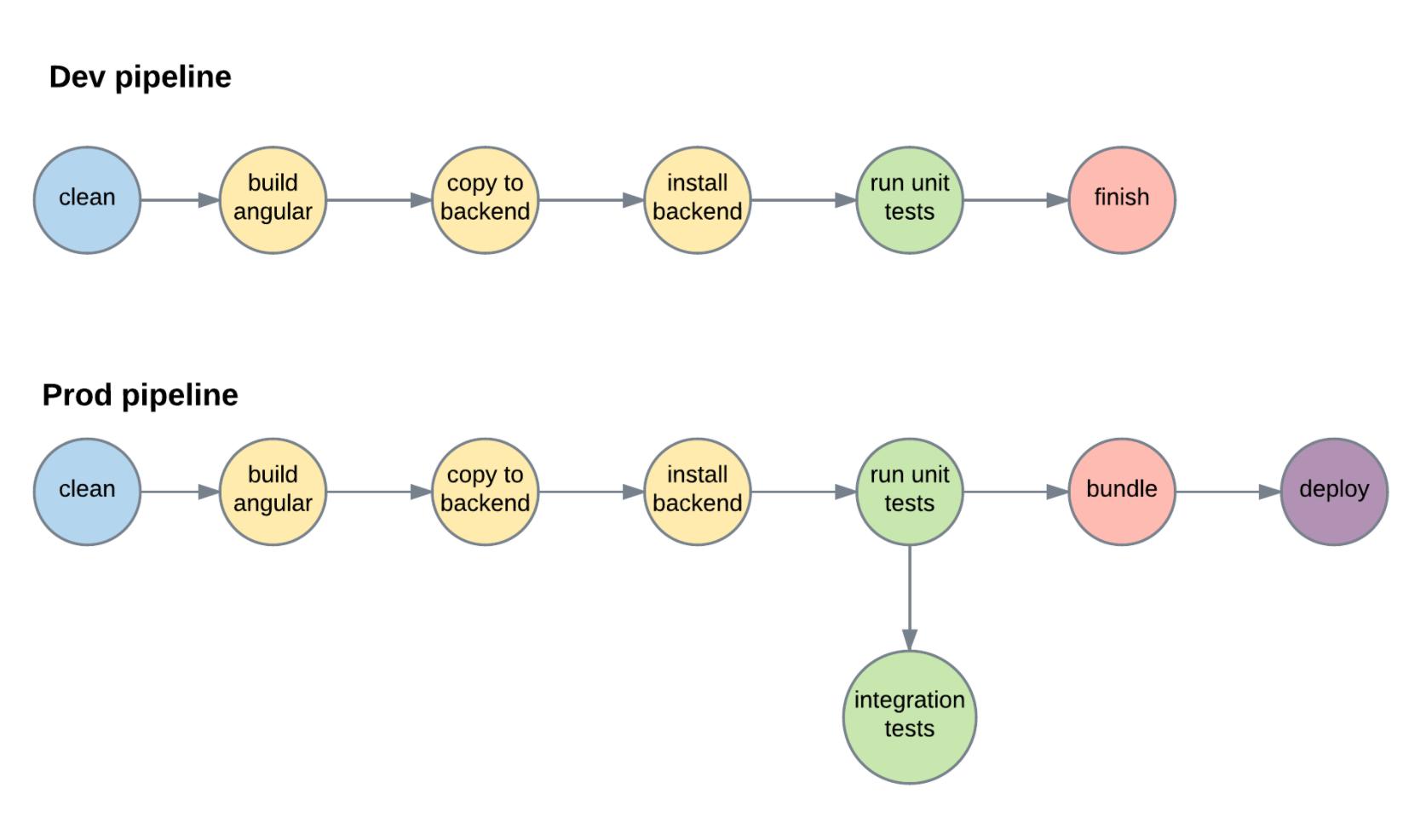


Figure 55- Development pipeline - Production pipeline

* Run ./build.sh
* If everything passes run ./integration-tests.sh
* If everything passes then run ./build.sh prod
* Upload the generated zip bundle file located /vir-3.0/Code/VIR-Backend/release

### 

### Installation/Maintenance

* Install Tesseract for your system. <https://github.com/tesseract-ocr/tesseract/wiki>
* Create an environment variable TESSERACT\_PATH pointing to your installation directory. Where the executable is.
* Create an environment variable TESSDATA\_PREFIX pointing to the directory above the 'tessdata' one; should be in your installation directory but it can be downloaded from the web.
* Clone the repo
* Create a local MySQL account with credentials
  + Username: root
  + Password: root
* Create schema called: vir
* Run:
  + ./build.sh
  + ./start
* Access the application at: localhost:8080

Note: If you are using a Unix like system make sure you give execution permissions to the files. chmod +x build.sh

### **Shortcomings/Wishlist**

* Add security to application (http to https)
* Add a library for faculty material
* Add a complete WAT test section
* Increase the tests banks