Florida International University
School of Computing and Information Sciences

Software Engineering Focus

Final Deliverable

NEAT

Team Members: Fernando Campo, Gabriel Fernandez, Pierre Jimenez, Giselle Pacheco, Justin Lopez, and Nelson Cruz

Product Owner(s): Vladan Lavolic

Mentor(s): Mohsen Taheri, Ranjeet Deshmukh

Instructor: Masoud Sadjadi

Abstract

Students take multiple courses that often have many concurrent assignments. This large amount of homework, projects and tests that they have to go through can create a situation where students finish non-critical tasks early, critical ones too late or even feel so inundated that they do nothing. Neat takes care to organize assignments for the student so that they only have to worry about getting the tasks done. Unlike other task management software, Neat uses an Agile approach so that not only deadlines are taken into account, but so are the amount of tasks per assignment and the velocity of the student. Neat was created by users of Agile development trying to show students the power of Agile without having to go through a course.

Table of Contents

Introduction 6
Current System 6
Purpose of New System 6
User Stories
Implemented User Stories 8
Pending User Stories 10
Project Plan
Hardware and Software Resources 12
Sprints Plan 14
Sprint 1 14
Sprint 2 15
Sprint 3 15
Sprint 4 17
Sprint 5 21
Sprint 6 23

Sprint 7 …… …... 23

System Design

Architectural Patterns …….. 25

System and Subsystem Decomposition …….…. 26

Deployment Diagram …….…... 28

Design Patterns ……..…….…... 29

System Validation ……. 30

Glossary …… …. 73

Appendix …… …. 74

Appendix A - UML Diagrams ……. 74

Static UML Diagrams ……. 74

Dynamic UML Diagrams ……..…….. 74

Appendix B - User Interface Design …….…... 106

Appendix C - Sprint Review Reports ……..…. 112

Appendix D - User Manuals, Installation/Maintenance Document, Shortcomings/Wishlist Document and other documents ……..……..…… 116

References ……...……... 125

INTRODUCTION

Students are tasked with a gamut of school assignments: homework, tests and projects. With so many goals to accomplish it can be daunting knowing where to start. Neat is a task management app that is focused for these users. What differentiates Neat from other similar software is the algorithm that decides what assignments need to be worked on first. The app was built using Agile principles and the algorithm is implemented using Agile principles.

The Current System section covers what our users have to do now to plan their tasks. Purpose of New System describe the problem and the solution presented by the Neat team. The User Stories section covers all the parts that the development team implemented. It also contains those user stories that were not finalized but the team still wishes were accomplished. Project Plan details the strategy and thought processes for the project, as well as the major pieces of development software chosen.

System Design illustrates the holistic view of the project. It describes the architecture of the system and subsystems as well as the reasons behind behind those choices. The System Validation section describes and details all the tests that were run against the system. The Glossary defines the vocabulary of the project. Finally, the Appendix contains the diagrams from the user stories and screenshots of the system in action. It also includes what we did in each sprint and where we see the app going in the future.

Current System

Most students do not have any system of organization of their assignments. If they do have one, they tend to use a planner or a calendar app. If a student downloads a task management app they can have something that will keep track of upcoming deadlines, but they cannot know at a glance which assignments are more pressing. If there is a homework assignment due in one week versus a project that is due in two, how does the student know which one they should start first? This an issue that all students face and that no other app can inform which assignment has higher priority.

Students are also curious about where they are in relation to other students. How many people are already halfway done? How far behind am I in relation to the class? Normally, the student has to go to class and take an informal poll of their classmates.This can lead to complacency if

the few that have been asked say they have not started. Another issue that occurs is that a student can get stuck on a problem and have no one to turn to immediately. What if they want to chat with someone that can help them with that subject? If they are not physically close to a location where they can get help it will cause stagnation in their momentum.

Purpose of New System

The Neat app creates a system where a student can input their assignment and tasks with its due date. From those pieces of information it decides which assignment task should be completed in order. Because of this the student can just focus on the task at hand instead of organizing their schedule. This also frees up the student from the anxiety associated with knowing which assignment they need to start with. In addition, it alerts the student which assignment they are falling behind on and if they need to pick up the slack.

The new system is also prepared to share assignments amongst a class. By doing this, the students can compare where they are in relation to the other students in the class. Another part that the app is prepared for is the chatroom. Here the student has a channel for help through tutors. The entire app is geared towards enabling students to finish their assignments with less stress and greater help.

User Stories

The following section provides the detailed user stories that were implemented in this iteration of the Neat 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 #118-Create loading screen
As a user, when I open the Neat app on my phone, I want to see a short "loading screen" before I'm directed to the proper page.

User Story #126-Register a new account
As a user, I want to be able to easily register a new account after selecting the "Register" option. In order to be able to use the Neat system

User Story #129-Create assignment dashboard
As a user, after I login to Neat, I want to be taken to a dashboard which shows all of my assignments so I can see all my assignments

User Story #130-Update progress
As a user, I want to be able to update my progress in completing the study steps for an assignment

User Story #132-Save login details
As a user, I want to be able to keep my login persistent between app restarts so i don't have to type my username and password every time I open Neat.

User Story #134-Reset account password
As a user, if i forgot my password, I want to be able to reset it, so that i can login and continue to use Neat system.

User Story #137-Setup backend on AWS

As a developer, I want to make sure that the backend is fully setup and working, so that Neat can function on any remote device.

User Story #138-Create database schema
As a database modeler, I want to create a database schema so that I can save all pertinent information about a user and their assignments

User Story #146-Create Login screen
As a user, I want to be able login to Neat so that I can access the functionality.

User Story #155-Add a new assignment
As a user I would like to be able to add an assignment so I can track my progress

User Story #156-Create an add task button
As a user I would like to have an add task Button so that I can add a task to an assignment

User Story #165-Continuous integration on the backend
As a developer, I would like for main commits to master branch to trigger a rebuilding process of the application as well as test all the main features

User Story #185-Website:Create Simple & Beautiful Homepage
As a user, when I visit the Neat website, I want to be presented with a simple and beautiful page. User Stories #184: About & #186: Contact and Services were concatenated into #185.

User Story #188-Create footer for website
As a user I would like to see a footer on the website so that i have easy access to website information.

User Story #191-Create Gmail login token
As a developer I want the users of my app to be able to login using Gmail tokens.

User Story #194-Create infrastructure for email validation
As a developer, I want to have an infrastructure on the backend to be able to validate a user's e-mail, so that the user can change passwords and verify their account.

User Story #199-Create login & register backend endpoints
As a developer, I want to have endpoints for the backend to create new users and to retrieve user authentication credentials, so that the user can log in to their account.

User Story #213-Create “Class Ranking" Page for Assignments (Backend)
As a front-end developer, I want to be able to fetch student statistics about tasks completed per assignment, so that I can populate the class ranking page.

User Story #214-Create chatBot UI
As a User, when the ChatBot sends me a message, I want to have a nice interface for viewing and replying to the message, so that I can potentially use it to schedule an appointment with online tutoring.

User Story #221-Create backend points for database
As a programmer, I want to hit API endpoints so that I can access database information

User Story #228-Create assignment view page
As a user, I want view assignment details and the task associated to that assignment, so that I can track my progress and mark tasks as completed.

User Story #229-Create endpoints for returning data to frontend
As a front-end developer, I want to be able to fetch specific information from the database, so that I can provide a demo of our connected application

User Story #237-Refactor the flow of the application
As a developer pieced the app a little better so that the flow would be what it should be.

User Story #270-Create object-level permissions in the backend
As a user, I want to make sure I only have access to my personal information, so that my data cannot be accessed by others.

User Story #271-Implement new progress algorithm
As a front-end developer, I want to receive the smart progress status for all of the logged in user’s assignments, so that the user knows what assignments need work.

User Story #274-Refactor frontend
Refactor mobile app code using modern programming techniques like separation of concern.

Pending User Stories

User Story #178-Set up teacher administrative UI & login page
As a teacher, I want to be able to log into my dashboard on a browser so that i can be able to view my classes and the students enrolled in them.

User Story #212-Create new user type "Mentor" in database
As a Mentor, I want to be able to approve every 5 tasks that students in a classroom environment have completed, to verify that they're not lying.

User Story #234-Create class ranking page
As a user I would like to see my assignments progress compared to other students in the class, so that i can evaluate my progress

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 plannings 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:

MySQL
We decided to use MySQL as our persistent data storage system because of the inherent benefits of a structure query language, security and speed that the Neat project requires, in addition to its large support and user-base.

Django
We decided to use Django in our backend implementation doe to its flexibility to customize pieces of the web site to suit the needs of the project; in addition, documents developed in Django are not directly exposed to the internet, offering an additional security layer.

Python
We had to use Python in our backend implementation because of our decision to use Django, which is a high-level Python Web framework; therefore, it enforces the use of Python as the underlined development language.

XCode
We had to use XCode because although our front end application is written mainly in JavaScript, all code developed for IOS, has to be compiled through XCode first before been able to be executed.

Atom
We choose Atom as our primary development environment (IDE) because of its simple interface, its ability to load custom modules that help developers in organizing, formatting, and debugging.

In addition, Atom's painless integration with Git makes it a great tool within a team environment.

React Native

We decided to use React Native to develop the frontend App because it allows us to learn one framework and implement the acquired knowledge in multiple platforms; in addition, Although we still write JavaScript with React Native, the components we define will end up rendering as native platform widgets providing the high performance that a native application would.

Sprints Plan

Sprint 1

User Story #146- Create login screen

Task

- Create login screen
- Create text inputs for username and password
- Login information should be validated in database
- Login should check if username already exists (front end)
- Design Logo Page

Acceptance Criteria

- A user with a username and password should be able to login
- A user should have the option to save username by itself, or username and password together
- There should be a link to the New User Account menu

User Story #137- Setup backend on AWS

Task

- Set up database documentation
- Test backend
- Create and set up database
- Set up AWS with project dependencies
- Set up basic django REST environment

Acceptance Criteria

- A user with a username and password should be able to login
- A user should have the option to save username by itself, or username and password together
- There should be a link to the New User Account menu

Sprint 2

User Story #199 Create login & register backend endpoints

Task

- Test the endpoints
- Code the login and register endpoint
- Research REST API and Password Storage
- Create documentation

Acceptance Criteria

- Endpoint URLs are accessible and take json requests
- The endpoints return useful information or error messages
- A token is returned when an user is successfully logged in
- Passwords are securely stored

Sprint 3

User Story #185 (Website) Create Simple & Beautiful Homepage

Task

- Research and develop a Modern Webpage designs
- Create Navigation Bar
- Create multiple sections (landing page, about, services, and contact) and related images
- Ensure web page is scalable depending on device resolution

Acceptance Criteria

- Create Simple Landing/Homepage and structure (16pts)
    - Image links (use relevant Google Play/Apple store images) to our Neat Application in the Apple Store/Google Play Store
- The navigation bar should match the colors/theme of the website. (8pts)
    - Link to Online Tutoring Page
    - Link to About Us/About the Company Page
    - Link to “Neat Intelligence” page
- Create About Us section including relevant information (2pts)
- Create Footer (2pts)
    - Contains copyright statement for Neat

- Email and Phone number for contact

User Story #194 Create infrastructure for e-mail validation

Task

- Create views for sending e-mail and validating code
- Generate random code
- Develop the code in django backend
- Testing
- documentation

Acceptance Criteria

- User's e-mail is verified in the database
- There exist endpoints that the front-end can call
- The backend is able to send emails to the user
- A new code is generated each time a validation e-mail is sent

User Story #221 Create backend endpoints for database

Task

- Test the endpoints
- Combine User & UserInfo
- Create serializers for models
- Create loggers for tests
- Create test for database
- Research database testing
- Research logger
- Create Django Models

Acceptance Criteria

- Must be able to request data from all database tables
- Must be able to post a new user
- Must be able to post to all model tables
- Must follow the database schema

Sprint 4

User Story #118 Create Loading Screen

Task

- Documentation for Create Loading Screen
- Testing for Create Loading Screen
- Create View that represents the splash screen
- App checks if new user

Acceptance Criteria

- The Neat logo should be displayed in the center of the screen. Consider using a background color that complements the Neat logo.
- At the end of the loading screen, the user should be taken to the appropriate page. (For example, if this is the user's first time ever opening Neat, they should be taken to the page that asks if they are a new user so that they can register an account. If the user has a saved username and password, automatically attempt to log them in. And so on. Consider all possibilities.)
- It should take less than 3 seconds to show the Neat logo and execute the logic that determines which page to direct the user to.

User Story #129 Create Assignment dashboard

Task

- Hit the dashboard endpoint
- How many days until due date
- Create progress bar
- Display list of assignments

Acceptance Criteria

- View a list of assignments
- View the progress and the number of days until assignment is due
- Have a button to add assignment

User Story #130 Update progress

Task

- Test the function is correct
- Have data go to backend
- Create toggle

Acceptance Criteria

- Refer to mockup to make sure the design is correct
- When a user is done with a task they click a checkbox/toggle and the progress is updated.
- When going back to the assignments view the toggles have the same state as when the page was left

User Story #131 Display completion Progress Bar

Task

- Create function that will change color depending on progress

Acceptance Criteria

- Check mockup for an example progress bar visual.
- Below 33% use red color
- Above 33% and below 67% use yellow. Above 66% is green

User Story #132 Save login details

Task

- Store token to local storage
- Documentation for save login Details
- Testing of save login details

Acceptance Criteria

- When a new user opens the app, they are taken to the login screen where they can login, register, or reset password (forgot password).
- When an existing user who has previously successful logged in, they are taken to the Dashboard instead of the login screen.
- Token information is saved to Local Storage of device.

User Story #155 Add new Assignment

Task

- Hit the assignment endpoint
- Create done button
- Create a form
- Create add assignment button

Acceptance Criteria

- Have input for assignment name, and due date
- Have a done button that once pressed adds assignment to a list

User Story #156 Create add task button

Task

- Create button that signifies "add"
- Have the button navigate to a different screen

Acceptance Criteria

- can tap the + button
- can save the task
- screen goes to the list of tasks after adding new task

User Story #213 Create "Class ranking" page for Assignments (Backend)

Task

- Create endpoint for collab view
- Create view that fetches proper data from models
- Documentation
- Testing

Acceptance Criteria

- Endpoint is created that returns all students' task completion percentage for an assignment, given an assignment id.
- Proper messages are returned if the assignment doesn't exist
- Only data about students who are enrolled in the assignment should return

- If a student has no tasks added to the assignment, they should not be part of the data.

User Story #228 Create Assignment View Page

Task

- Documentation
- Testing for create
- Add collaboration page component to assignment view
- Fetch task data from endpoint to populate view- assignment view
- Create View UI for assignment View page

Acceptance Criteria

- Displays a ListView of all of the tasks associated with an assignment.
- All tasks have a checkbox next to them (check mockups) indicating whether or not they're marked as "completed".
- Has a back button of some kind.

User Story #229 Create endpoints for returning data to frontend

Task

- Create endpoints for coupling users to assignments
- Create endpoints for getting user assignment completion
- Create endpoint for getting user info from token
- Documentation
- Testing

Acceptance Criteria

- All models in the database are created and updated to mirror the information to be stored and fetched
- All required endpoints are provided to fetch and update database models through http requests

User Story #269 Make Team Page

Task

- Download an edited image of each member

- Created Page and stylized
- Add page to navbar
- Add images and bio for each image
- Update social media links

Acceptance Criteria

- Must be displayed neatly and organized.
- Each member has their own image, name and bio.
- About section links out to it.
- Connected to the Navigation Bar.

User Story #220 Add new Class

Task

- Hit the class endpoint
- Create done button
- Create a form
- Create add class button

Acceptance Criteria

- Have input for class name
- Have a done button that once pressed adds class to a list

Sprint 5

User Story #126 Register new account

Task

- Test components
- Create controller and routing
- Create new account registration view
- Create backend for new account

Acceptance Criteria

- To be able to specify the type of account; (student or teacher).

- If creating a student's account, obtain proper information from student
- If creating a teacher's account, obtain proper information from teacher

User Story #134 Reset password

Task

- Test reset password
- Reset password view

Acceptance Criteria

- Never send plaintext passwords in email, it's bad security practice. Send the user a link which allows them to set a new password
- Confirmation emails

User Story #214 Create Chatbot UI

Task

- Create offset for tabbar
- Set avatar as Neat penguin
- Create API gets and puts
- Create UI

Acceptance Criteria

- Gets user input from the UI form.
- Sends user's input to AI Team's ChatBot API.
- Gets chat response data from the AI Team's ChatBot API.
- Sends chat response data to the user.

User Story #270 Create object-Level permissions in the backend

Task

- Add filtering to views
- Edit endpoints to include permissions
- Testing
- Turn on authentic action in the database

Acceptance Criteria

- Authentication must be turned on for the database
- The database must only return information relevant to the logged in user
- A user must not be able to edit or delete other users' database models

Sprint 6

User Story #178 Create teacher administration UI and login

Task

- Test login page
- Create login website frontend
- Create Documents and Diagrams
- Develop Angular 2 skeleton
- Research django frontend vs dedicated separate frontend

Acceptance Criteria

- ome to a conclusion on the best frontend for this UI
- Set up the basic UI framework
- Set up basic login functionality
- Allow professors to view data in a concise manner
- Connect to the backend to keep task and student records updated
- Update Courses and Task

User Story #234 Create "Class ranking" page for Assignments (frontend)

Task

- Test class ranking page
- Create class ranking view

Acceptance Criteria

- User progress is displayed against the rest of the class

User Story #271 Implement new progress algorithm

Task

- Change and add new endpoints with new algorithm
- Code the new algorithm
- Update scripts and endpoints due to new fields
- Change models to accommodate new algorithm
- Testing and documentation

Acceptance Criteria

- There exist a backend endpoint that returns the smart status of the logged in user's assignment
- The endpoints return extra relevant data so that the front end doesn't have to send multiple requests
- The collaboration endpoint is updated with the new algorithm
- The algorithm is implemented according to the business rules
- Assignments should now have weights

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

- **Presentational and Container Components**

A simple pattern used to separate *how things looks* and *how things work*. Presentational component are usually written as *stateless functional components* and receive data and callbacks via *props* (parameters). Container components are usually written as *stateful classes* and provide the data and behavior to presentational or other container components. This provides better separation of concerns and better reusability.

- **Client-Server**

We decided to use the client-server architectural pattern for many reasons: Our application relies heavily on data storage and retrieval, and less on mobile hardware utilization or heavy computation. As such, it is imperative to have large amounts of storage, something not feasible on mobile devices alone. Furthermore, the same data has to be accessed by many different clients. A peer-to-peer model could cause privacy issues and was not considered. This naturally leads to the separation of client and server; The client deals with rendering the user interface, taking input from the user, and communicating the user's action to the server; While the server deals with any heavy computation, data storage and retrieval, and providing endpoints for communication

- **Model-View-Template**

The backend systems we chose to use, Django with Django Rest Framework, use the MVT architectural pattern. In this system, the view is in charge of selecting which data is represented: describing the queryset. The template is in charge of how the data is shown, which, with the Rest Framework, deals with serialization/deserialization into JSON. The model creates, updates, or retrieves the data. The traditional controller component becomes the framework itself, and most of the logic is attributed to the view.

System and Subsystem Decomposition

Our system is divided into two parts, frontend and backend, as shown by the figure. In the backend, the urls determine the endpoints. Once the http request hits an endpoint, control is transferred to the view. The view performs any computation and business logic required, as well as determine the queryset and instructions for the models. Control is transferred to the serializer, which translates the json in the http request to an object representation that the model controller will understand. The serializer also takes care of database integrity validation. The model controller will then update the database based on the queryset and instructions provided.

On the frontend, use used React Native to build the mobile app. React Native make use of components to render the view of the mobile app. The Login View is made from a LoginContainer component that contains the LoginForm component. Many of the views within the app are composition of component for better reusability. Once logged in, the Dashboard view shows current assignments. We make use of a subsystem class called AuthServices, which provide a simplified interface to the backend API endpoints. All calls that are made to the backend services are made through the AuthServices class. Subsystems components are separated by features, for example, Login, Add Assignment, View Assignment, and Dashboard.

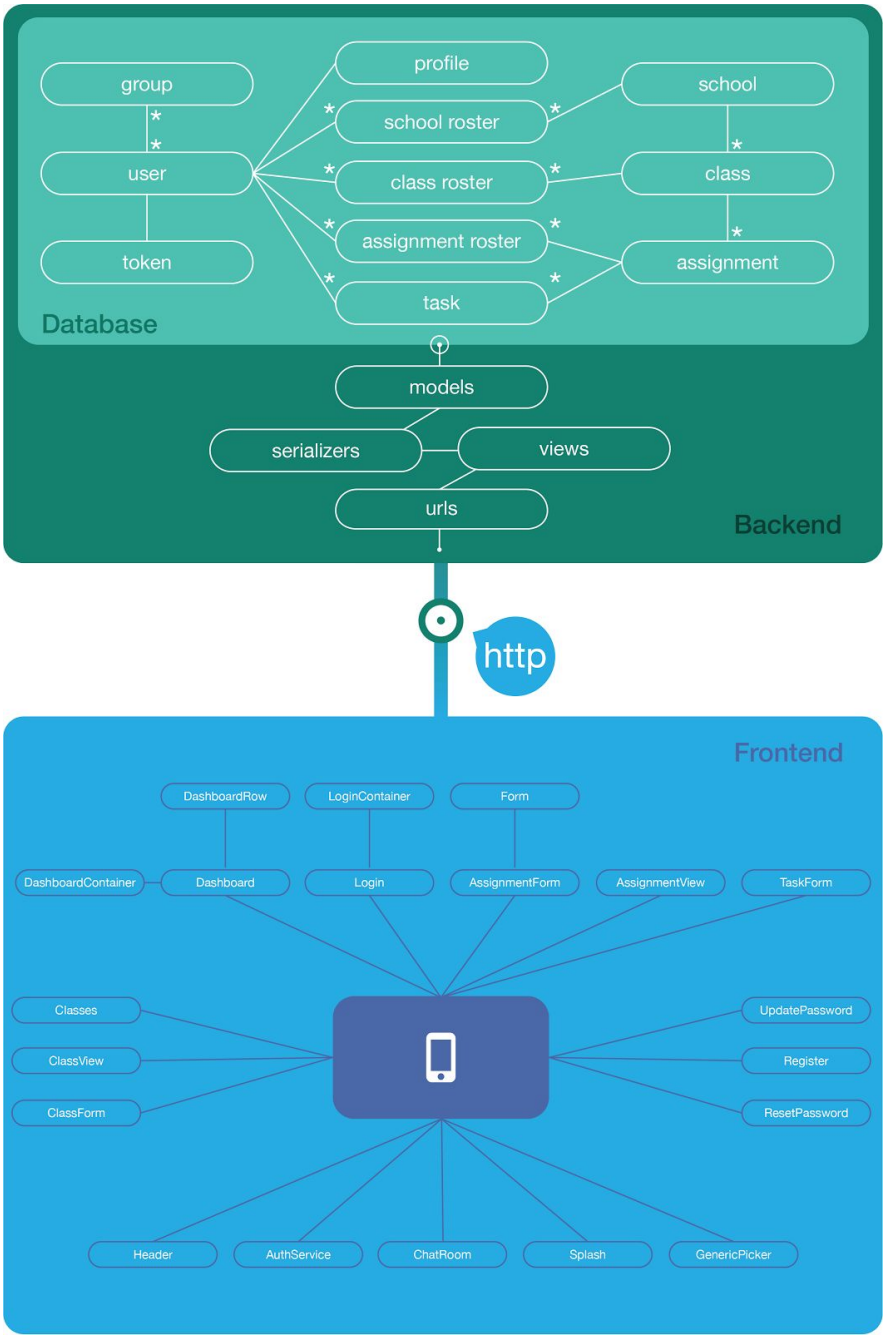


Figure: System & Subsystem Diagram

Deployment Diagram

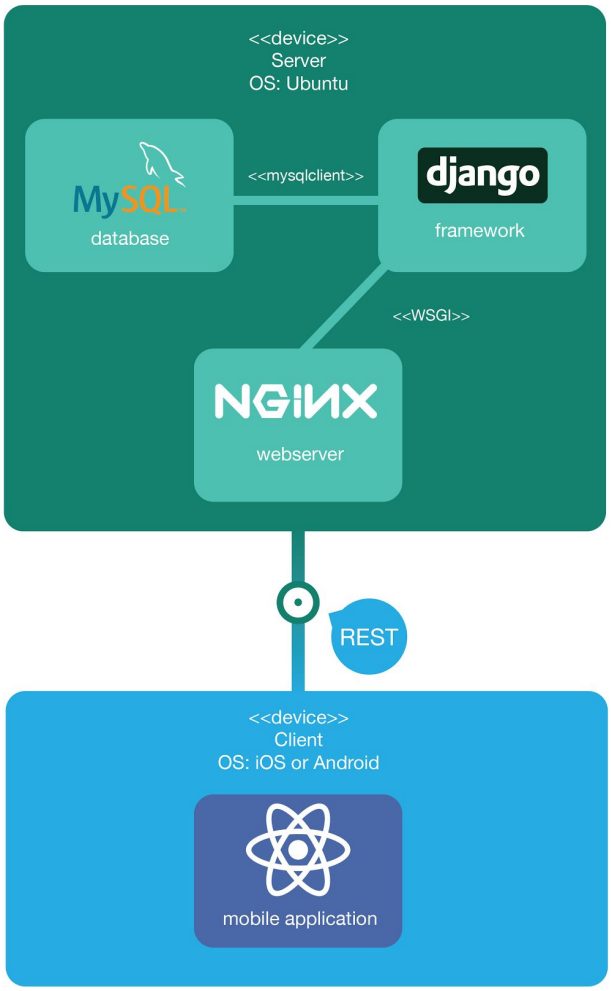


Figure: Deployment Diagram

Design Patterns

- **Adapter**
  - *Definition*: Convert existing classes or models into a more specific instance that fits our application better.
  - *Use*: We extended the default Django user to include a profile. This allowed us to store more information about the user, but at the same time reuse existing authentication and permissions modules that require the default user.
- **Decorator**
  - *Definition*: Attach extra functionality to an existing interface
  - *Use*: Many of our views in the REST framework have decorators that will affect the models in different ways, depending on the http request method. The user interface itself does not change, but the functionality does.
- **Module**
  - *Definition*: Used to further emulate the concept of classes in such a way that we're able to include both public/private methods and variables inside a single object, thus shielding particular parts from the global scope.
  - *Use*: Most of the React components are exported classes with modules.
- **Components**
  - *Definition*: Components let you split the UI into independent, reusable pieces, and think about each piece in isolation.
  - *Use*: All the view are composed from components.
- **Facade**
  - *Definition*: Provides a convenient higher-level interface to a larger body of code, hiding its true underlying complexity.
  - *Use*: Used in AuthServices.

System Validation

In this section, we present our system test and subsystem tests for each user story. We tested all of our user stories using manual testing in order to ensure that our system does not contain any faults or defects.

User Story #118 - Create Loading Screen

Integration Tests:

Purpose: To test the functionality of use case: #118 Loading Screen

Test ID: LoadScreen_001 (Sunny Day):
ID: NEAT-SD-118-01

Purpose

- Test if user is taken to the login screen.

Precondition

- User has not logged into the app before.

Expected Result

- User is taken to the login screen.

Actual Result

- User is taken to the login screen.

Test ID: LoadScreen_002 (Sunny Day):
ID: NEAT-SD-118-02

Purpose

- Test if user is taken to the dashboard screen.

Precondition

- User has logged into the app before.

Expected Result

- User is taken to the dashboard screen.

Actual Result

- User is taken to the dashboard screen.

Test ID: LoadScreen_003 (Rainy Day):
ID: NEAT-RD-118-01

Purpose

- Test if user is taken back to the login screen if username and password is wrong.

Precondition

- User login is wrong.
- User is at the login screen.

Expected Result

- User is taken back to the login screen.

Actual Result

- User is taken back to the login screen.

User Story #126 - User Registration

System Tests:

Purpose: To test the functionality of use case: #126 User Registration

Test ID: Registration_001 (Sunny Day):

Purpose

- Ensures that valid information is accepted by the system.

Precondition

- All fields are filled with valid information

Input

- Username, password, e-mail address, user-info

Expected Result

- A new user is registered, and user is taken to the dashboard

Actual Result

- A new user is added to the system

Test ID: Registration_002 (Rainy Day):

Purpose

- Ensure the system validates the information before requesting registration.

Precondition

- Register button is pressed without any information filled.

Input

- Null on all fields.

Expected Result

- All empty fields are highlighted in red

Actual Result

- No request is made to backend

User Story #130 - Create Assignment Dashboard

Unit Tests:

Purpose: To test the functionality of use case: #130 Assignment Dashboard

Test ID: AssignDash_001 (Sunny Day)

Purpose

- Ensure that a user can toggle switch

Precondition

- User is logged in
- User has added a class
- User is part of a school
- User has an assignment
- User is in the Assignment View

Input

- Click on switch

Expected Result

- Switch turns from False to True, or from True to False. In iOS this would be pale to green, or green to pale, respectively.

Actual Result

- Switch toggles correctly

User Story #132 - Implement Persistent Login

Unit Tests:

Purpose: To test the functionality of use case: #132 Persistent Login

Test ID: PersistentLogin_001 (Sunny Day):
ID: NEAT-SD-132-01

Purpose

- Test if user token is saved to local storage.

Precondition

- User has successfully authenticated.
- Is at the login screen.

Expected Result

- User token is saved to local storage.

Actual Result

- User token is saved to local storage.

Test ID: PersistentLogin_002 (Rainy Day):
ID: NEAT-RD-132-01

Purpose

- Test if user token is not saved if username and password is wrong.

Precondition

- User login is wrong.
- User is at the login screen.

Expected Result

- User token is not saved.

Actual Result

- User token is not saved.

User Story #134 - Password Reset

System Tests:

Purpose: To test the functionality of use case: #134 Password Reset

Test ID: Validate_001 (Sunny Day):

Purpose

- Ensure the system validates a current user.

Precondition

- A valid user name is entered by user, the user exists in the system.

Input

- Valid user name.

Expected Result

- An e-mail is sent to the user's primary e-mail address.

Actual Result

- An e-mail Is received by user with validation code.

Test ID: Validate_002 (Rainy Day):

Purpose

- Ensure the system rejects an invalid user.

Precondition

- An invalid user name is entered by user, the user does not exists in the system.

Input

- Invalid user name.

Expected Result

- Prompt the user with incorrect username message.

Actual Result

- No e-mail Is received by user.

User Story #137 - Setup Backend on AWS

System Tests:

Purpose: To test the functionality of use case: #137 Retrieve Information From the Backend Remotely

Test ID: Backend_001 (Sunny Day):

Purpose

- Ensure the backend is reachable from the given URL

Precondition

- Browser is open

Input

- Visit http://127.0.0.1:8000/api/

Expected Result

- List of all available models

Actual Result

- List of all available models

Test ID: Backend_002 (Rainy Day):

Purpose

- Ensure the backend is only reachable from the predefined URL

Precondition
- Browser is open

Input
- Visit http://127.0.0.1:8000/api/

Expected Result
- Http 404

Actual Result
- Http 404

User Story #138 - Create Database Schema

System Tests:

Purpose: To test the functionality of use case: #185 Database Schema

Test ID: Database_001 (Sunny Day):

Purpose

- Check that the default startdate cannot be before today. startDate has a default value of today if none is placed.

Precondition

- All fields are filled with valid information.

Input

- startDate

Expected Result

- startDate is greater than or equal today's date

Actual Result

- Assignment gets created.

User Story #146 - Log In

System Tests:

Purpose: To test the functionality of use case: #146 Log In

Test ID: Login_001 (Sunny Day):

Purpose

- To test the functionality when a user logs in with a valid username and password.

Precondition

- The user should have access to the application
- The database needs to properly bind to the application and be accessible
- The user must have a registered account
- The user must be on the login page

Input

- Username: admin
- Password: admin123

Expected Result

- The user is redirected to their homepage

Actual Result

- The user is redirected to their homepage

Test ID: Login_002 (Rainy Day):

Purpose

- To test the functionality when a user attempts to login with an invalid username.

Precondition

- The user should have access to the application
- The database needs to properly bind to the application and be accessible
- The user must be on the login page

Input

- Username: test
- Password: 123

Expected Result

- The message “invalid username” should appear

Actual Result

- The message “invalid username” should appear

User Story #155 - Add Assignment

Unit Tests:

Purpose: To test the functionality of use case: #155 Add Assignment

Test ID: AddAssign_001 (Sunny Day)

Purpose

- Ensure that a user can add an assignment

Precondition

- User is logged in
- User has added a class
- User is part of a school

Input

- Class is selected
- Assignment Name
- Due date of the assignment

Expected Result

- User can see an added assignment on the dashboard

Actual Result

- Added assignment on the dashboard

Test ID: AddAssign_002 (Sunny Day)

Purpose

- Ensure that there is a default due date

Precondition

- User is logged in
- User has added a class
- User is part of a school

Input

- User inputs an assignment name

Expected Result

- Assignment is added with the default due date

Actual Result

- Assignment is added

Test ID: AddAssign_003 (Rainy Day)

Purpose

- Ensure that if a user does not enter an assignment name, assignment is not added

Precondition

- User is logged in
- User has Selected a manual class
- User is part of a school

Input

- Due date of the assignment

Expected Result

- App stays on the Assignment form page

Actual Result

App stays on assignment page

User Story #156 - Add Task

Unit Tests:

Purpose: To test the functionality of use case: #156 Add Task

Test ID: AddTask_001 (Sunny Day)

Purpose

- Ensure that a user can add an task

Precondition

- User is logged in
- User has added a class
- User is part of a school
- User has at least one assignment on dashboard

Input

- task Name
- Due date of the task

Expected Result

- User can see an added task on the assignment view

Actual Result

- Added task on the on the assignment view

Test ID: AddTask_002 (Sunny Day)

Purpose

- Ensure that there is a default due date

Precondition

- User is logged in
- User has added a class
- User is part of a school
- User has added at least 1 assignment

Input

- User inputs an task name

Expected Result

- Assignment is added with the default due date

Actual Result

- Assignment is added

Test ID: AddTask_003 (Rainy Day)

Purpose

- Ensure that if a user does not enter an assignment name, assignment is not added

Precondition

- User is logged in
- User has added a class
- User is part of a school

Input

- Due date of the assignment

Expected Result

- App stays on the Assignment form page

Actual Result

- App stays on assignment page

User Story #185 - Create Simple & Modern Webpage

System Tests:

Purpose: To test the functionality of use case: #185 Modern Web Page

Test ID: Webpage_001 (Sunny Day):

Purpose

- Ensure all anchors work, directing to individual sections

Precondition

- NeatStudy.com is live and running.

Input

- Click “Home”, “About”, “Services”, or “Contact”

Expected Result

- Direct to those specific sections on the homepage.

Actual Result

- Direct to those specific sections on the homepage.

Test ID: Webpage_002 (Sunny Day):

Purpose

- Ensure hyperlink to Team Page redirects

Precondition

- NeatStudy.com is live and running.

Input

- Click Team Page

Expected Result

- Direct to the Team Page.

Actual Result

- Direct to the Team Page.

Test ID: Webpage_003 (Sunny Day):

Purpose

- Ensure that emails are sending to the correct email addresses, with the correct subject and body text.

Precondition

- The user is on the Contact Section and Neatstudy is running. The user has filled out all of the forms on the messaging page.

Input

- The User fills in the user name, receiver addresses, message subject, and message body for the message that is to be sent. Then user presses the “Send Message” button.

Expected Result

- The message should be sent to neatstudy@outlook.com with the correct subject and message body text.

Actual Result

- The message was sent to neatstudy@outlook.com with the correct information.

Test ID: Webpage_004 (Rainy Day):

Purpose

- Ensure that emails are not being sent with erroneous receiver addresses/subjects/body text from the form.

Precondition

- The User is on the Contact Section.

Input

- User has not filled out a receiver email address, a subject, a message body, or all 3. Then user presses the “Send Message” button.

Expected Result

- The email will not be sent due to a lack of the required parameters for message.

Actual Result

- Server replied asking for necessary information to send email.

User Story #194 - Validate a user's email

System Tests:

Purpose: To test the functionality of use case: #194 Validate email

Test ID: Validate_001 (Sunny Day):

Purpose

- Ensure backend is able to create codes and send emails to users.

Precondition

- Http request is made to the server, using the send_email URL.

Input

- User token

Expected Result

- Backend sends an email with a code to the user's email address.

Actual Result

- Email with code received

Test ID: Validate_002 (Sunny Day):

Purpose

- Ensure backend verified a user given a correct code.

Precondition

- Http request is made to the server, using the verify URL & user's code.

Input

- User token

Expected Result

- User verified.

Actual Result

- User verified.

Test ID: Validate_003 (Rainy Day):

Purpose

- Ensure backend does not verify incorrect codes.

Precondition

- Http request is made to the server, using the verify URL & user's incorrect code.

Input

- User token

Expected Result

- User could not be verified.

Actual Result

- User could not be verified.

User Story #199 - Authenticate Users

System Tests:

Purpose: To test the functionality of use case: #199 Authenticate Users

Test ID: Authenticate_001 (Sunny Day):

Purpose

- Ensure login backend endpoint works correctly.

Precondition

- All fields are filled with valid information Http request is made to the server, using the login URL.

Input

- '{"username":"admin","password":"password123"}'

Expected Result

- Token

Actual Result

- Token

Test ID: Authenticate_002 (Sunny Day):

Purpose

- Ensure register backend endpoint works correctly.

Precondition

- Http request is made to the server, using the register URL.

Input

- '{"username":"newuser","email":"email@gmail.com","password":"password123", "profile":{"grade":"12","age":"23","gender":"male"}}'

Expected Result

- User created.

Actual Result

- User created.

Test ID: Authenticate_003 (Rainy Day):

Purpose

- Ensure login backend endpoint errors work correctly when given incorrect information

Precondition

- Http request is made to the server, using the login URL

Input

- '{"username":"admin2","password":"password123"}'

Expected Result

- Error; User does not exist.

Actual Result

- Error; User does not exist.

Test ID: Authenticate_004 (Rainy Day):

Purpose

- Ensure register backend endpoint errors work correctly when given incorrect information

Precondition

- Http request is made to the server, using the login URL

Input

- '{"username":"newuser","email":"email@gmail.com","password":"password123", "userInfo":{"grade":"12","age":"23","gender":"male"}}'

Expected Result

- Error; User does not exist.

Actual Result

- Error; User does not exist.

User Story #213 - Fetch Assignment Collaboration Data

Unit Tests:

Purpose: To test the functionality of use case: #213 Assignment Collab

Test ID: AssignCollab_001 (Sunny Day)

Purpose

- Ensure backend returns an array of user percentages per assignment.

Precondition

- Http request is made to the server at the collab endpoint.

Input

- User token, Assignment PK.

Expected Result

- Array of user names and percentages for the assignment given.

Actual Result

- Array of user names and percentages for the assignment given.

Test ID: AssignCollab_002 (Rainy Day)

Purpose

- Ensure backend returns a proper error message when the assignment doesn't exist.

Precondition

- Http request is made to the server at the collab endpoint.

Input

- User token, Assignment PK.

Expected Result

- Error message saying the assignment given doesn't exist.

Actual Result

- Error message saying the assignment given doesn't exist.

User Story #214 - Create Chatbot UI

Unit Tests:

Purpose: To test the functionality of use case: #214 Chatbot UI

Test ID: ChatbotUI_001 (Sunny Day)

Purpose

- Ensure user can open the chatroom

Precondition

- User logged in

Input

- Chatroom icon pressed on TabBar

Expected Result

- Chatroom UI appears

Actual Result

- Chatroom UI appears

Test ID: ChatbotUI_002 (Sunny Day)

Purpose

- Ensure user can type input and have it appear in chat

Precondition

- User logged in
- User in chatroom UI

Input

- Keyboard presses

Expected Result

- Keyboard strokes appear on the screen after pressing enter

Actual Result

- Keyboard strokes appear on the screen after pressing enter

Test ID: ChatbotUI_003 (Rainy Day)

Purpose

- Ensure that if a user presses the chatroom icon while in the chatroom, the chat is not reset

Precondition

- In chatroom

Input

- Click chatroom icon

Expected Result

- No change in view

Actual Result

- No change in view

User Story #221 - Create Backend Endpoints For Database

System Tests:

Purpose: To test the functionality of use case: #221 Backend Endpoints

Test ID: Endpoints_001 (Sunny Day):

Purpose

- Login is OK with superuser

Precondition

- User in system

Input

- Hit login endpoint

Expected Result

- HTTP_200_OK

Actual Result

- HTTP_200_OK

Test ID: Endpoints_002 (Sunny Day):

Purpose

- Login is not-OK with non existent credentials

Precondition

- User not in system

Input

- Hit login endpoint

Expected Result

- HTTP_400_BAD_REQUEST

Actual Result

- HTTP_400_BAD_REQUEST

Test ID: Endpoints_003 (Sunny Day):

Purpose

- Test user can be inserted

Precondition

- None

Input

- Hit register-list endpoint

Expected Result

- HTTP_200_OK

Actual Result

- HTTP_200_OK

Test ID: Endpoints_004 (Sunny Day):

Purpose

- Test that user data can be fetched

Precondition

- User logged in

Input

- Hit user-detail endpoint

Expected Result

- HTTP_200_OK

Actual Result

- HTTP_200_OK

Test ID: Endpoints_005 (Sunny Day):

Purpose

- Test that fetched user data is correct

Precondition

- User logged in
- Data located in repository

Input

- Chatroom icon pressed on TabBar

Expected Result

- testData equals repository data

Actual Result

- testData equals repository data

Test ID: Endpoints_006 (Sunny Day):

Purpose

- Test that username and email can be updated

Precondition

- User logged in

Input

- PUT to user list endpoint

Expected Result

- testData equals repository data

Actual Result

- testData equals repository data

Test ID: Endpoints_007 (Sunny Day):

Purpose

- UserInfo can be created during register

Precondition

- User logged in

Input

- POST userInfo endpoint

Expected Result

- HTTP_200_OK

Actual Result

- HTTP_200_OK

Test ID: Endpoints_008 (Sunny Day):

Purpose

- userInfo can be gotten and is correct

Precondition

- User logged in
- userInfo data in repository

Input

- Hit GET userInfo endpoint

Expected Result

- testData equals repository data

Actual Result

- testData equals repository data

Test ID: Endpoints_009 (Sunny Day):

Purpose

- Test user info can be changed

Precondition

- User logged in
- userInfo data in repository

Input

- Hit PUT userInfo endpoint

Expected Result

- getData equals repository data

Actual Result

- getData equals repository data

Test ID: Endpoints_010 (Sunny Day):

Purpose

- Test School created successfully

Precondition

- User in system

Input

- Hit POST School endpoint

Expected Result

- HTTP_201_CREATED

Actual Result

- HTTP_201_CREATED

Test ID: Endpoints_011 (Sunny Day):

Purpose

- Test School gotten successfully

Precondition

- User in system
- School data in repository

Input

- Hit GET school endpoint

Expected Result

- School data equal to repository data

Actual Result

- School data equal to repository data

Test ID: Endpoints_012 (Sunny Day):

Purpose

- Test School can be updated successfully

Precondition

- User in system
- School data in repository

Input

- Hit PUT school endpoint

Expected Result

- Test data equal to repository data

Actual Result

- Test data equal to repository data

Test ID: Endpoints_013 (Sunny Day):

Purpose

- Test school roster can be created

Precondition

- User in system

Input

- Hit POST schoolRoster endpoint

Expected Result

- HTTP_200_OK

Actual Result

- HTTP_200_OK

Test ID: Endpoints_014 (Sunny Day):

Purpose

- Test that school roster data can be fetched

Precondition

- User logged in
- SchoolRoster in repository

Input

- Hit GET schoolRoster endpoint

Expected Result

- SchoolRoster data equal to repository data

Actual Result

- SchoolRoster data equal to repository data

Test ID: Endpoints_015 (Sunny Day):

Purpose

- Test SchoolRoster can be updated successfully

Precondition

- User in system
- SchoolRoster data in repository

Input

- Hit PUT school endpoint

Expected Result

- Test data equal to repository data

Actual Result

- Test data equal to repository data

Test ID: Endpoints_016 (Sunny Day):

Purpose

- Test class can be created

Precondition

- User in system

Input

- Hit POST class endpoint

Expected Result

- HTTP_200_OK

Actual Result

- HTTP_200_OK

Test ID: Endpoints_017 (Sunny Day):

Purpose

- Test that class data can be fetched

Precondition

- User logged in
- Class in repository

Input

- Hit GET class endpoint

Expected Result

- Class data equal to repository data

Actual Result

- Class data equal to repository data

Test ID: Endpoints_018 (Sunny Day):

Purpose

- Test class can be updated successfully

Precondition

- User in system
- Class data in repository

Input

- Hit PUT class endpoint

Expected Result

- Test data equal to repository data

Actual Result

- Test data equal to repository data

Test ID: Endpoints_019 (Sunny Day):

Purpose

- Test classRoster can be created

Precondition

- User in system

Input

- Hit POST classRoster endpoint

Expected Result

- HTTP_200_OK

Actual Result

- HTTP_200_OK

Test ID: Endpoints_020 (Sunny Day):

Purpose

- Test that classRoster data can be fetched

Precondition

- User logged in
- ClassRoster in repository

Input

- Hit GET classRoster endpoint

Expected Result

- ClassRoster data equal to repository data

Actual Result

- ClassRoster data equal to repository data

Test ID: Endpoints_021 (Sunny Day):

Purpose

- Test classRoster can be updated successfully

Precondition

- User in system
- ClassRoster data in repository

Input

- Hit PUT classRoster endpoint

Expected Result

- Test data equal to repository data

Actual Result

- Test data equal to repository data

Test ID: Endpoints_022 (Sunny Day):

Purpose

- Test assignments can be created

Precondition

- User in system

Input

- Hit POST assignments endpoint

Expected Result

- HTTP_200_OK

Actual Result

- HTTP_200_OK

Test ID: Endpoints_023 (Sunny Day):

Purpose

- Test that assignments data can be fetched

Precondition

- User logged in
- Assignments in repository

Input

- Hit GET assignments endpoint

Expected Result

- Assignments data equal to repository data

Actual Result

- Assignments data equal to repository data

Test ID: Endpoints_024 (Sunny Day):

Purpose

- Test assignments can be updated successfully

Precondition

- User in system
- Assignments data in repository

Input

- Hit PUT assignments endpoint

Expected Result

- Test data equal to repository data

Actual Result

- Test data equal to repository data

Test ID: Endpoints_025 (Sunny Day):

Purpose

- Test tasks can be created

Precondition

- User in system

Input

- Hit POST tasks endpoint

Expected Result

- HTTP_200_OK

Actual Result

- HTTP_200_OK

Test ID: Endpoints_026 (Sunny Day):

Purpose

- Test that tasks data can be fetched

Precondition

- User logged in
- Tasks in repository

Input

- Hit GET tasks endpoint

Expected Result

- Tasks data equal to repository data

Actual Result

- Tasks data equal to repository data

Test ID: Endpoints_027 (Sunny Day):

Purpose

- Test tasks can be updated successfully

Precondition

- User in system
- Tasks data in repository

Input

- Hit PUT tasks endpoint

Expected Result

- Test data equal to repository data

Actual Result

- Test data equal to repository data

User Story #228 - Create Assignment View Page

Integration Tests:

Purpose: To test the functionality of use case: #228 Assignment View

Test ID: AssignView_001 (Sunny Day):

ID: NEAT-SD-228-01

Purpose

- Test if assignment detail page is shown after clicking on assignment from a list of assignments.

Precondition

- User has successfully authenticated
- User is at the assignment dashboard

Expected Result
- User is at the assignment detail page screen.

Actual Result
- User is at the assignment detail page screen.

Test ID: AssignView_002 (Rainy Day):
ID: NEAT-RD-228-01

Purpose
- Test if assignment detail page is not shown after clicking on the add button from a list of assignments.

Precondition
- User has successfully authenticated
- User is at the assignment dashboard

Expected Result
- User is not at the assignment detail page screen.

Actual Result
- User is not at the assignment detail page screen.

User Story #229 - Fetch User Assignment Progress

Unit Tests:

Purpose: To test the functionality of use case: #229 Assignment Progress

Test ID: AssignProgress_a001 (Sunny Day)

Purpose

- Ensure backend returns the user's percentage per assignment

Precondition

- Http request is made to the server at the assignment progress endpoint

Input

- User token, assignment pk, user pk

Expected Result

- The user's completion percentage for the assignment

Actual Result

- The user's completion percentage for the assignment

Test ID: AssignProgress_a002 (Rainy Day)

Purpose

- Ensure backend returns a proper error message when the assignment doesn't exist

Precondition

- Http request is made to the server at the assignment progress endpoint

Input

- User token, assignment pk, user pk

Expected Result

- Error message saying the assignment given doesn't exist

Actual Result

- Error message saying the assignment given doesn't exist

Test ID: AssignProgress_b001 (Sunny Day)

Purpose

- Ensure backend returns the user info by token

Precondition

- Http request is made to the server at the user endpoint

Input

- User token

Expected Result

- The token's user information

Actual Result

- The token's user information

Test ID: AssignProgress_b002 (Rainy Day)

Purpose

- Ensure backend returns no information with the wrong token

Precondition

- Http request is made to the server at the user endpoint

Input

- Invalid user token

Expected Result

- An empty array

Actual Result

- An empty array

Test ID: AssignProgress_c001 (Sunny Day)

Purpose

- Ensure backend can enroll a user in an assignment

Precondition

- Http request is made to the server at the assignment roster endpoint

Input

- User token, user url, assignment, url

Expected Result

- User enrolled & assignment roster data

Actual Result

- User enrolled & assignment roster data

Test ID: AssignProgress_c002 (Rainy Day)

Purpose

- Ensure backend returns a proper error when the user is already enrolled

Precondition

- Http request is made to the server at the assignment roster endpoint

Input

- User token, user url, assignment, url

Expected Result

- The fields assignment, user must make a unique set

Actual Result

- The fields assignment, user must make a unique set

User Story #234 - Create Class Ranking page

Integration Tests:

Purpose: To test the functionality of use case: #234 Class Ranking Page

Test ID: ClassRanking_001 (Sunny Day):

Purpose

- Ensure the system reports a correct ranking.

Precondition

- All assignments for a user are selected as finished.

Input

- N/A

Expected Result

- User is ranked as first or tied for first.

Actual Result

- User progress bar is displayed in green percentage is shown as 100%.

Test ID: ClassRanking_002 (Rainy Day):

Purpose

- Ensures system displays a corrected ranking by completion date/time when users are tied.

Precondition

- Multiple user with 100% completion.

Input

- For multiple users all assignments are marked as completed.

Expected Result

- Only one user is marked as first based on time of completion.

Actual Result

- Actual ranking number is displayed on the page.

User Story #269 - Create Team Page

Unit Tests:

Purpose: To test the functionality of use case: #269 Team Page

Test ID: Team_001 (Sunny Day):

Purpose

- Ensure the filtered options only display individuals belonging in each section.

Precondition

- Neatstudy.com is live and running, each image has a fliter code.

Input

- “Lead Members”, “Developers” or “Mentors”.

Expected Result

- Bios and images will be filtered per result.

Actual Result

- Bios and images will be filtered per result.

Test ID: Team_002 (Sunny Day):

Purpose

- Ensure the everyone option displays all individuals working on the project.

Precondition

- Neatstudy.com is live and running, each image has a fliter code.

Input

- “Everyone”.

Expected Result

- All bios and images will be shown.

Actual Result

- All bios and images will be shown.

User Story #270 - Fetch Logged-in User Data

Unit Tests:

Purpose: To test the functionality of use case: #270 User Data

Test ID: UserData_001 (Sunny Day)

Purpose

- Ensure backend returns only the data that belongs to the logged in user

Precondition

- Http request is made to the server at the dashboard endpoint

Input

- User token

Expected Result

- Array of assignments & tasks that belong to the user

Actual Result

- Array of assignments & tasks that belong to the user

Test ID: UserData_002 (Rainy Day)

Purpose

- Ensure that a user cannot delete an object that they did not create

Precondition

- Http request is made to the server at the class endpoint, with DELETE method

Input

- User token, Class PK

Expected Result

- Error message saying that the user doesn't have permissions to perform that action

Actual Result

- Error message saying that the user doesn't have permissions to perform that action

---

User Story #271 - Fetch Assignment Smart Status

Unit Tests:

Purpose: To test the functionality of use case: #271 Smart Status

Test ID: SmartStatus_001 (Sunny Day)

Purpose

- Ensure backend returns smart status for the given user and assignment

Precondition

- Http request is made to the server at the assignment progress endpoint

Input

- User token, assignment pk

Expected Result

- Smart status for the user at the given assignment & extra information about the algorithm

Actual Result

- Smart status for the user at the given assignment & extra information about the algorithm

---

Test ID: SmartStatus_002 (Rainy Day)

Purpose

- Ensure that the server sends a proper error message when the smart status isn't available

Precondition

- Http request is made to the server at the assignment progress endpoint

Input

- User token, assignment pk for which the user doesn't belong to

Expected Result

- Error message saying that the user doesn't have any tasks in this assignment

Actual Result

- Error message saying that the user doesn't have any tasks in this assignment

---

Glossary

Agile - A methodology of designing software that emphasizes iteration, lightweight modeling and fast development.

Django - A backend web service based on the Python programming language designed to quickly come online.

MySQL - A relational database solution.

React-Native - A frontend mobile service based on React and Javascript which is geared for ease of development on both iOS and Android.

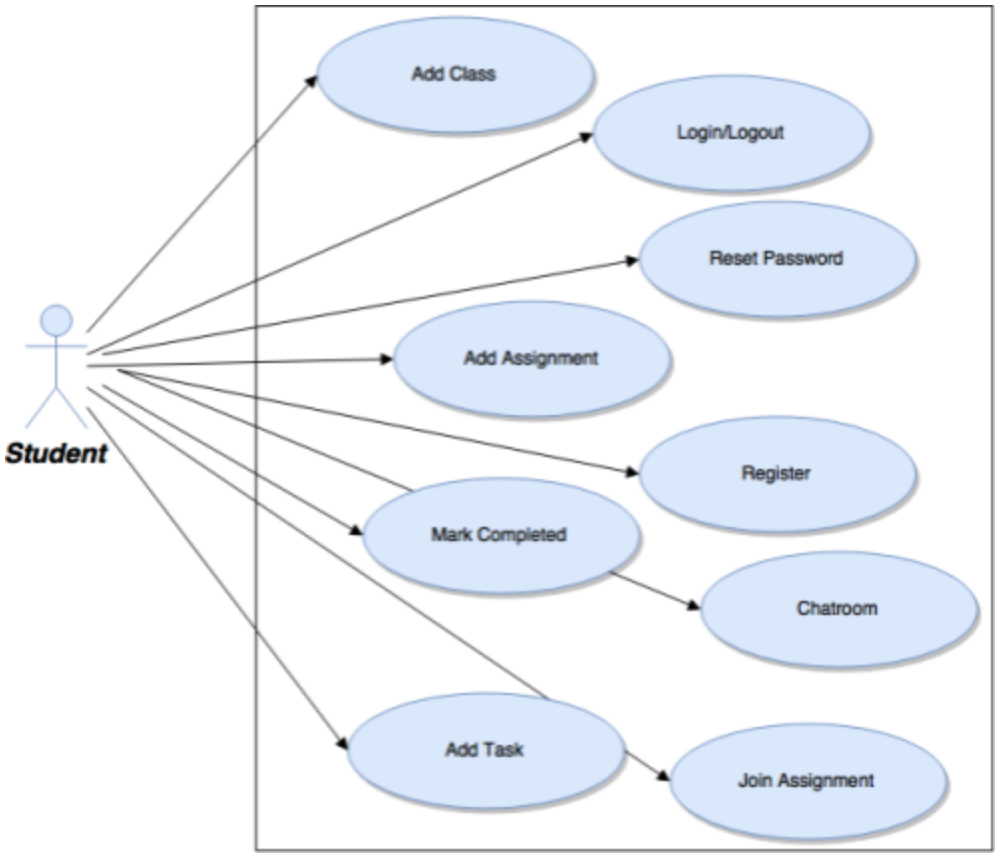
Sprint - A set development cycle (typically two weeks) were selected features are worked on.

UI - User Interface

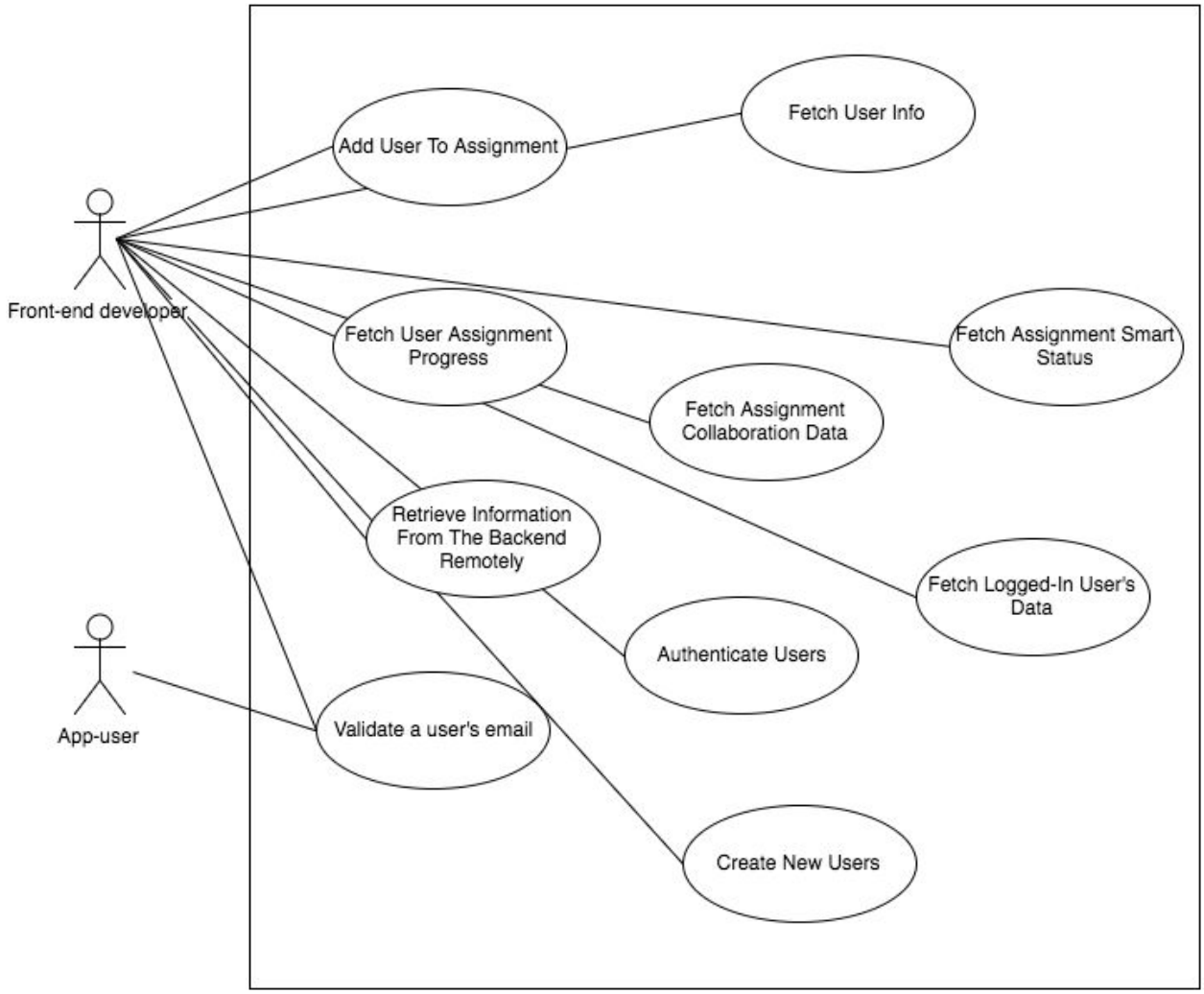
APPENDIX

Appendix A - UML Diagrams

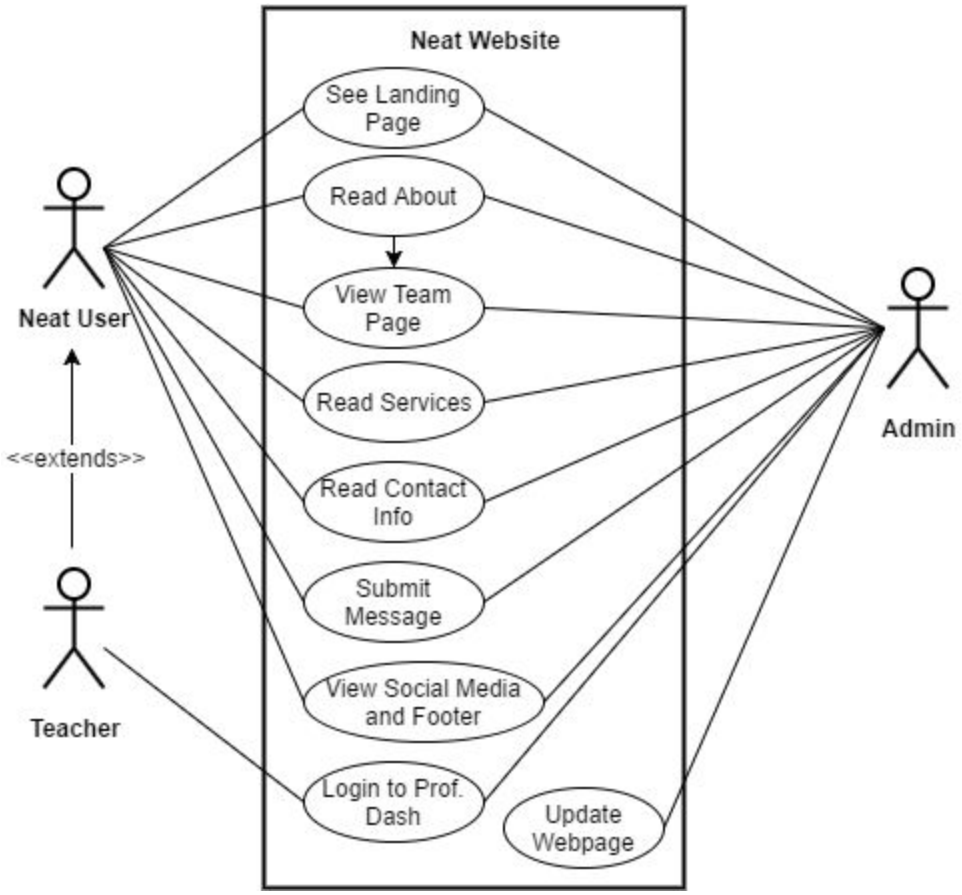
Use Case Diagrams



^ *Figure 1: Frontend Centric Use Case Diagram*

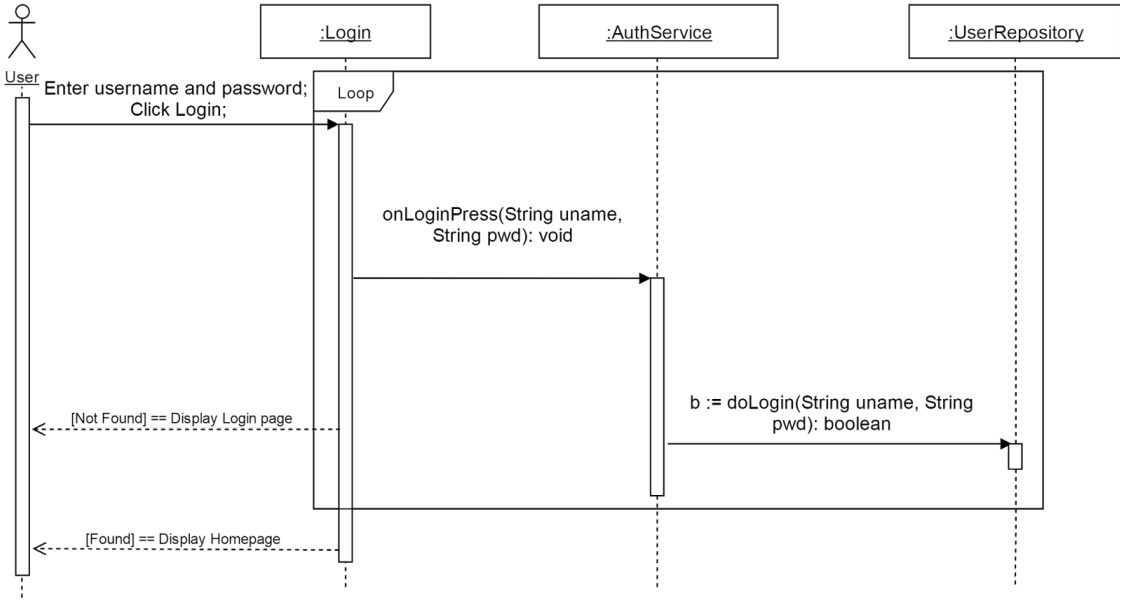


^ *Figure 2: Backend Centric Use Case Diagram*

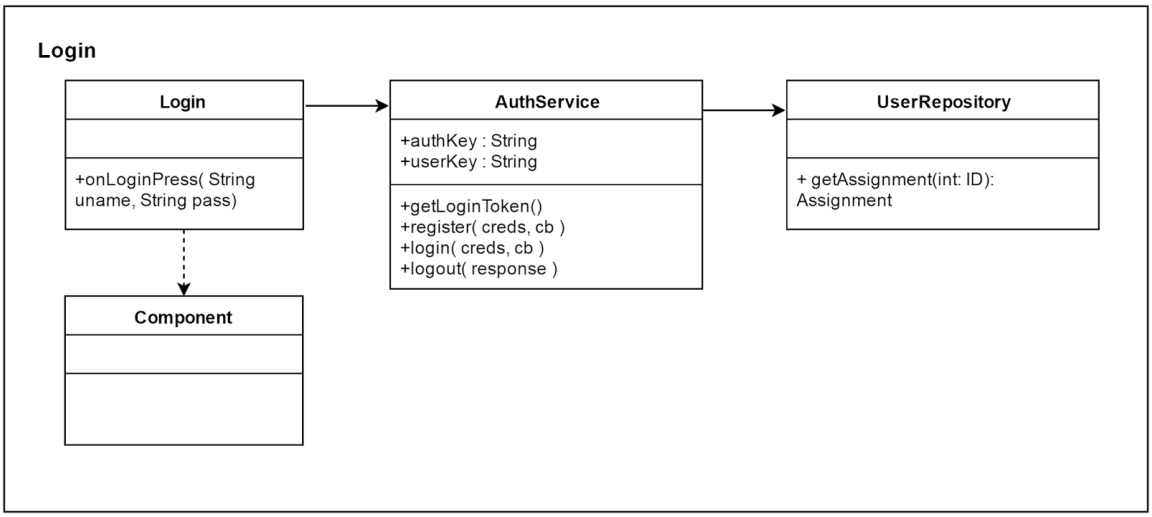


^ *Figure 3: Neat Webpage Use Case Diagram*

User Story #146-Create Login Screen

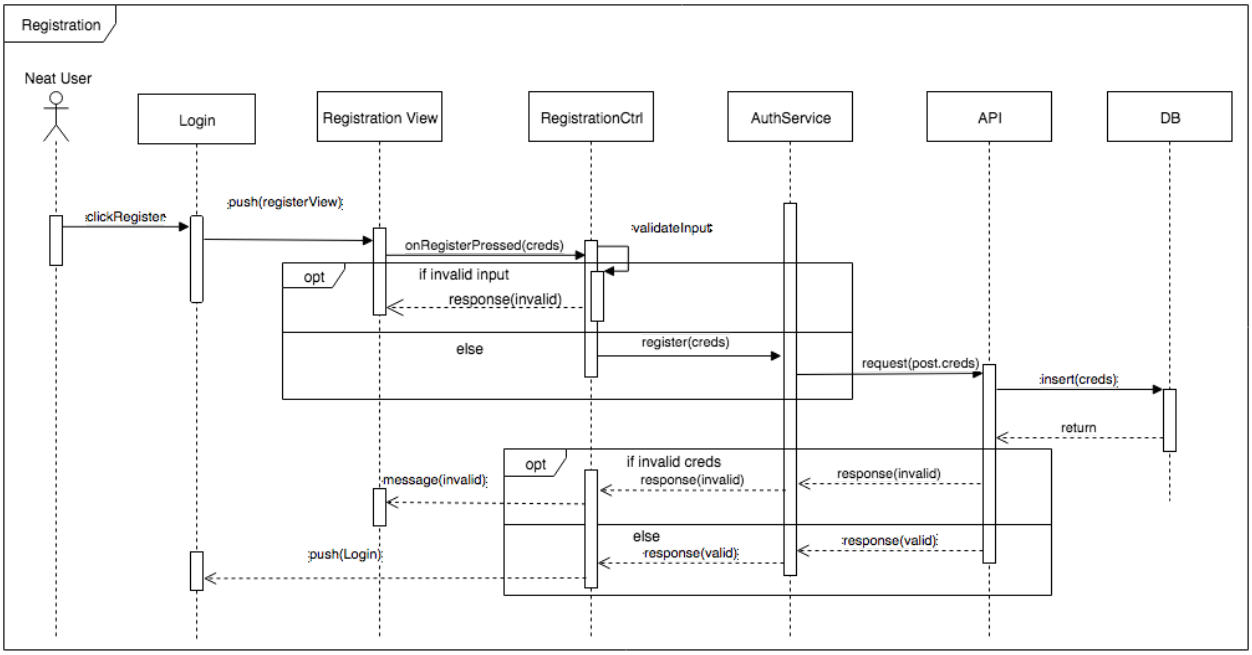


^ *Figure 4: "Login" Sequence Diagram*

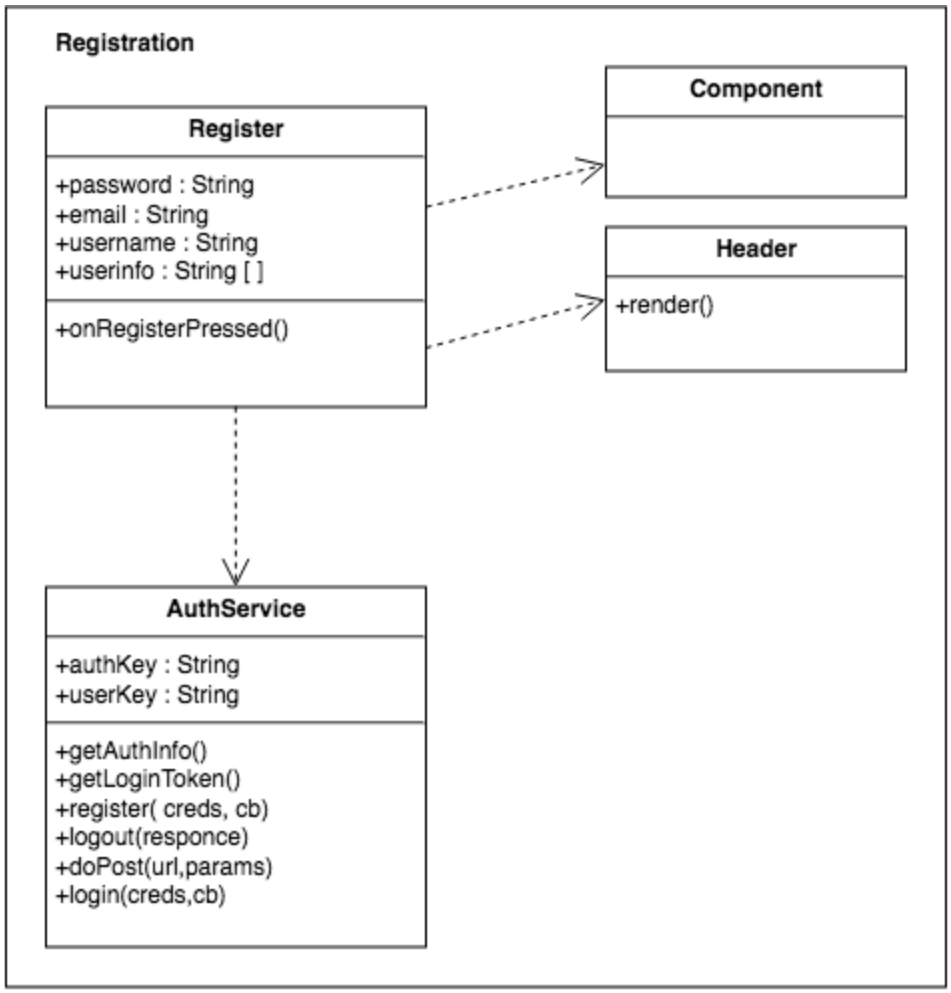


^ Figure 5: "Login" Class Diagram

User Story #126-Register a new account

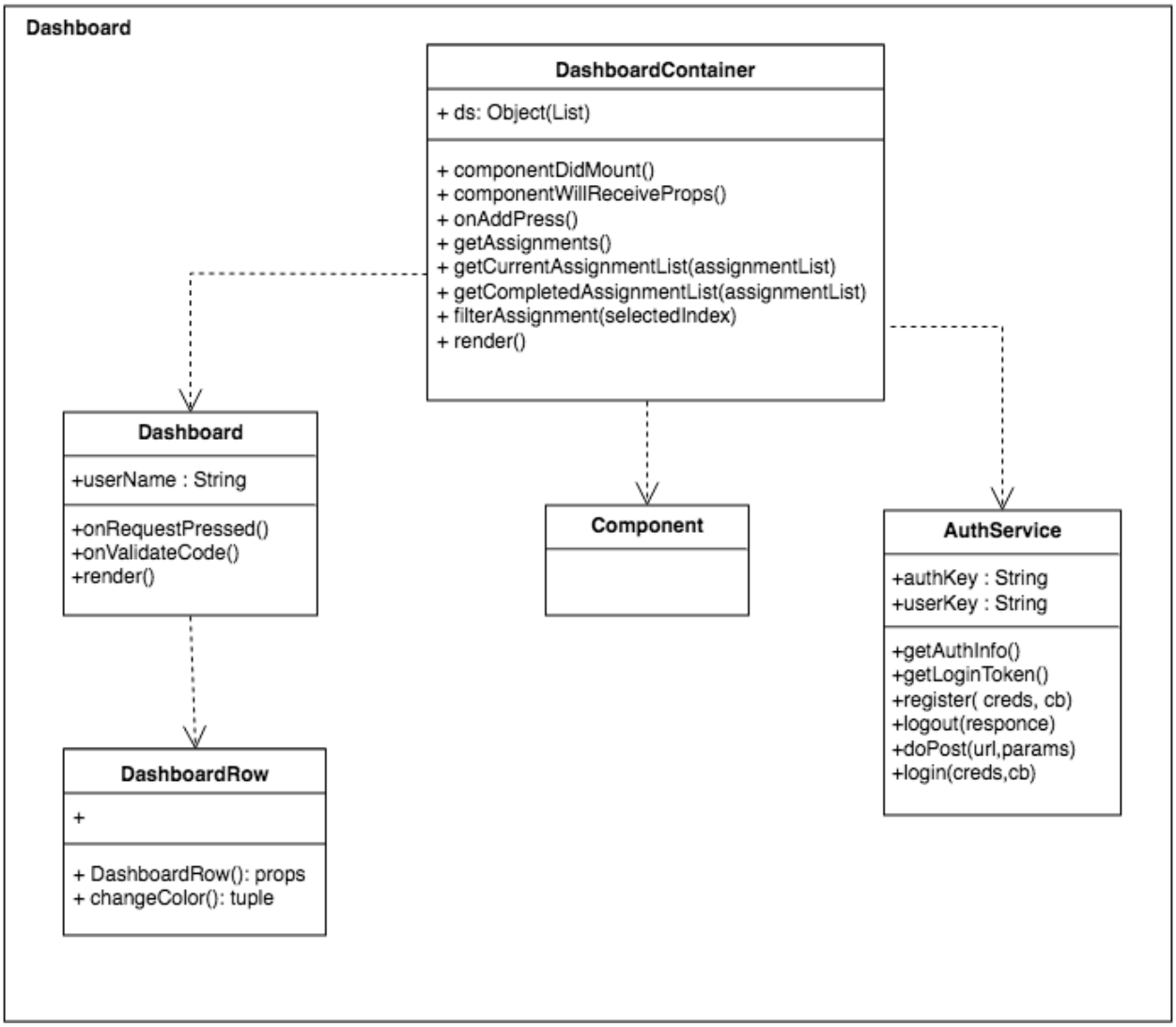


^ Figure 6: "Register a new account" Sequence Diagram



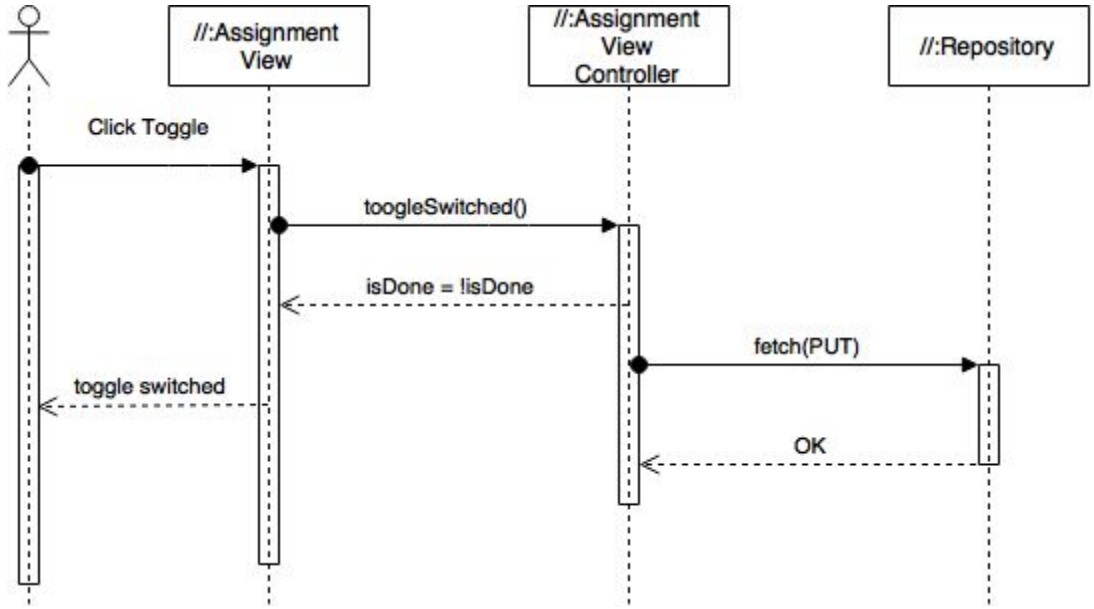
^ *Figure 7: User Story #126 Class Diagram*

User Story #129-Create assignment dashboard

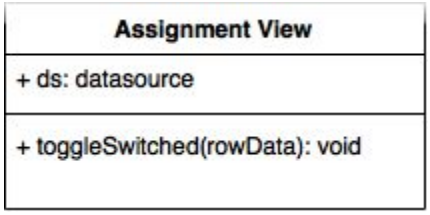


^ *Figure 8: User Story #129 Class Diagram*

User Story #130-Update progress

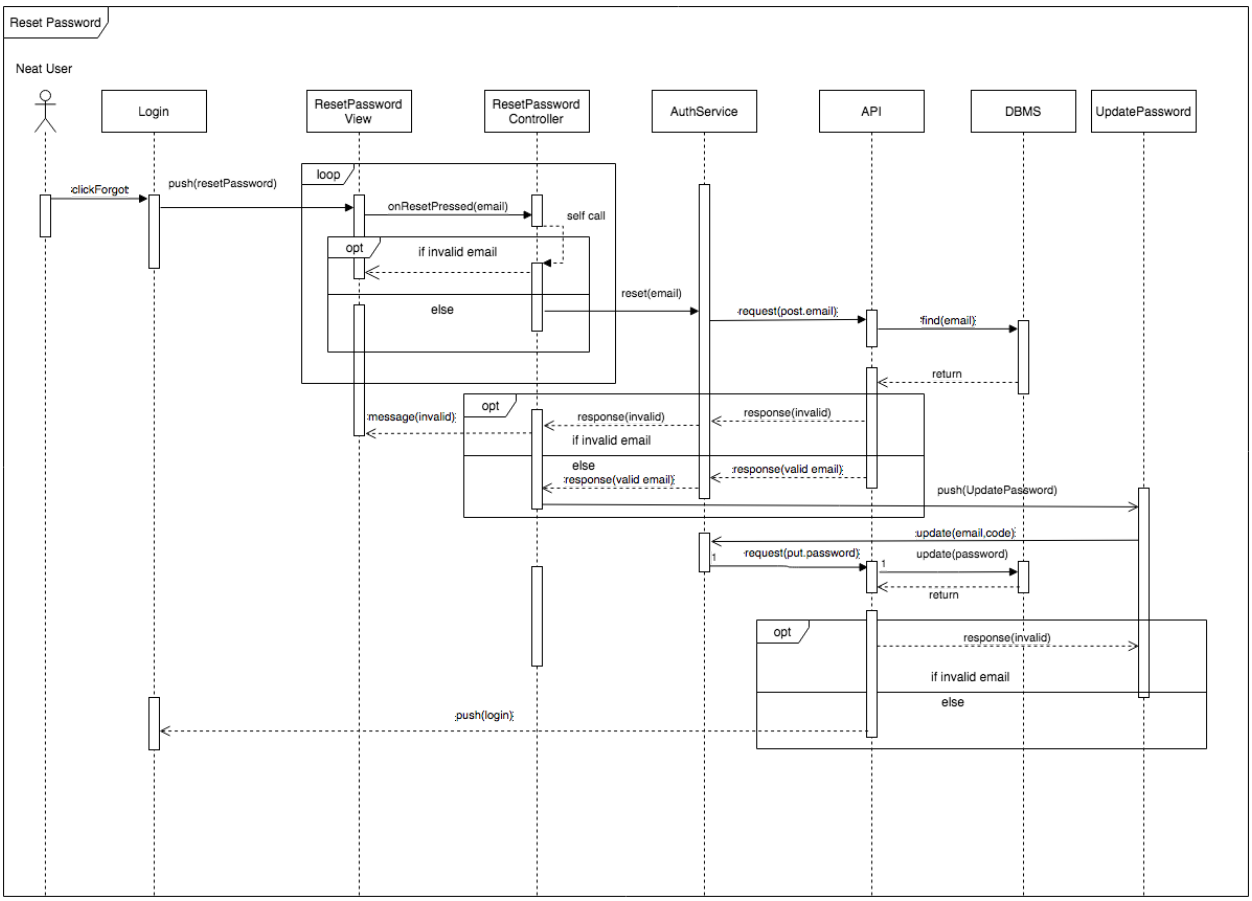


^ Figure 9: "Update progress" Sequence Diagram

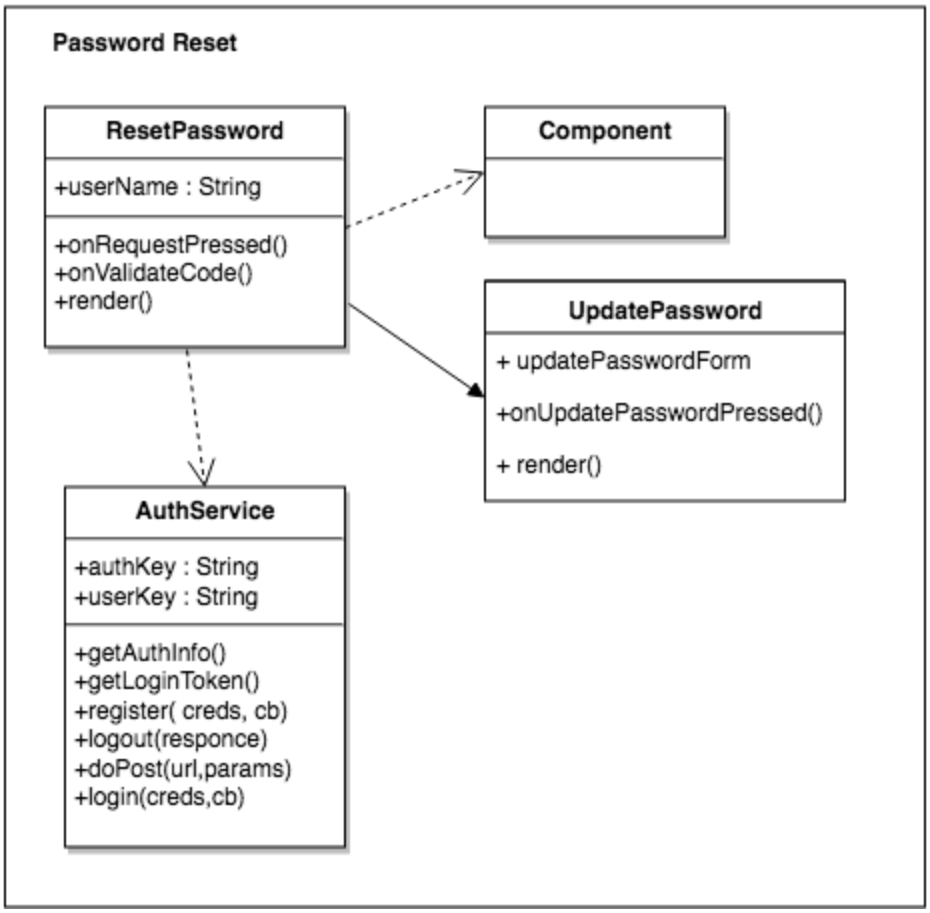


^ Figure 10: User Story #130 Class Diagram

User Story #134-Reset account password

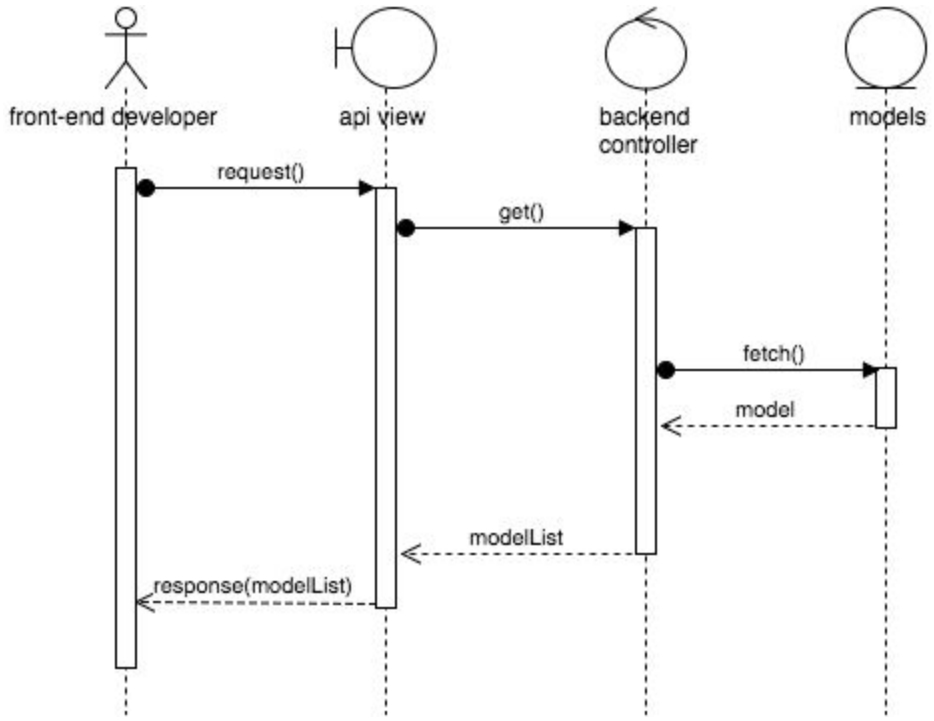


^ Figure 11: "Reset account password" Sequence Diagram

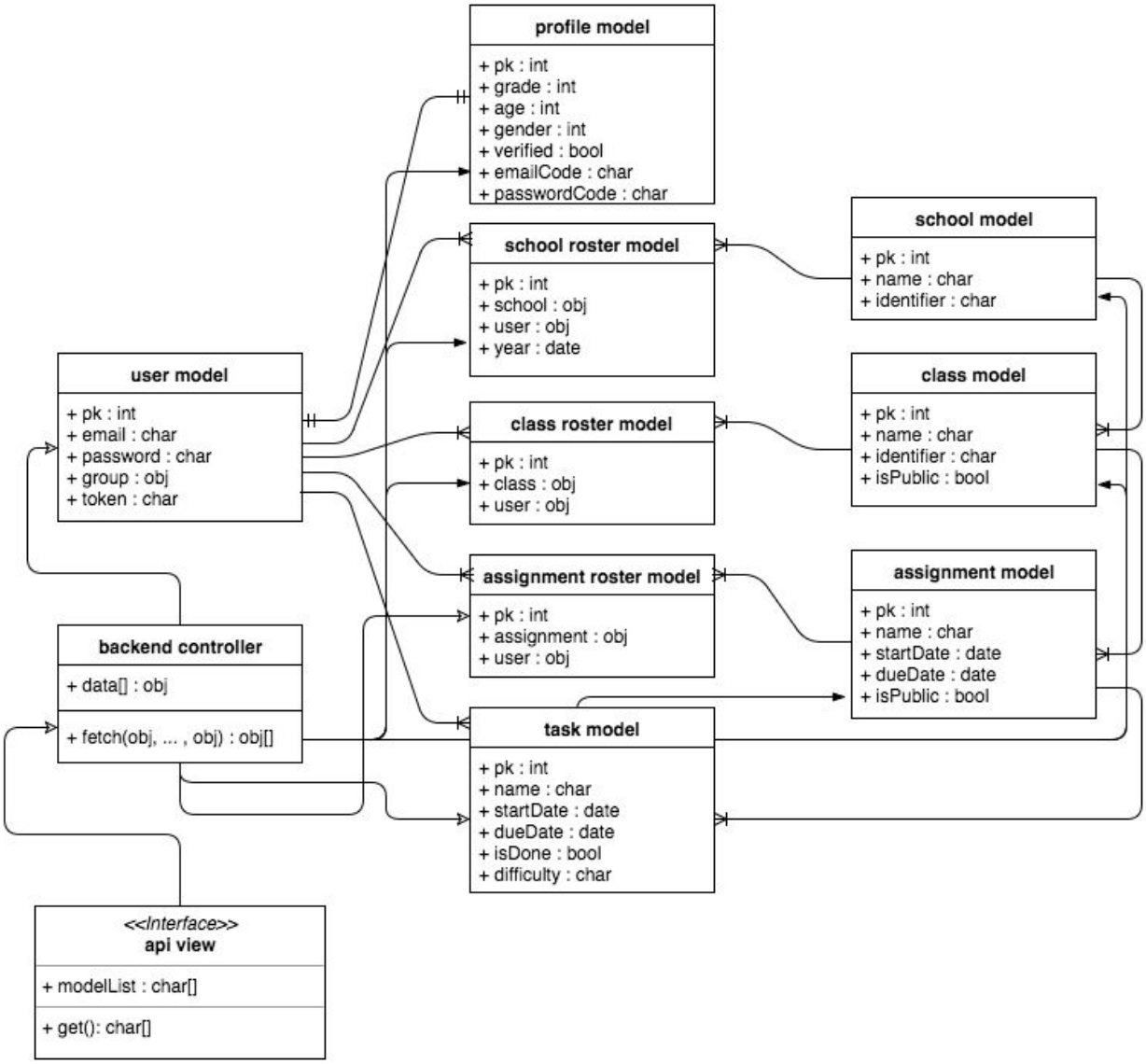


^ Figure 12: User Story #134 Class Diagram

User Story #137-Setup backend on AWS

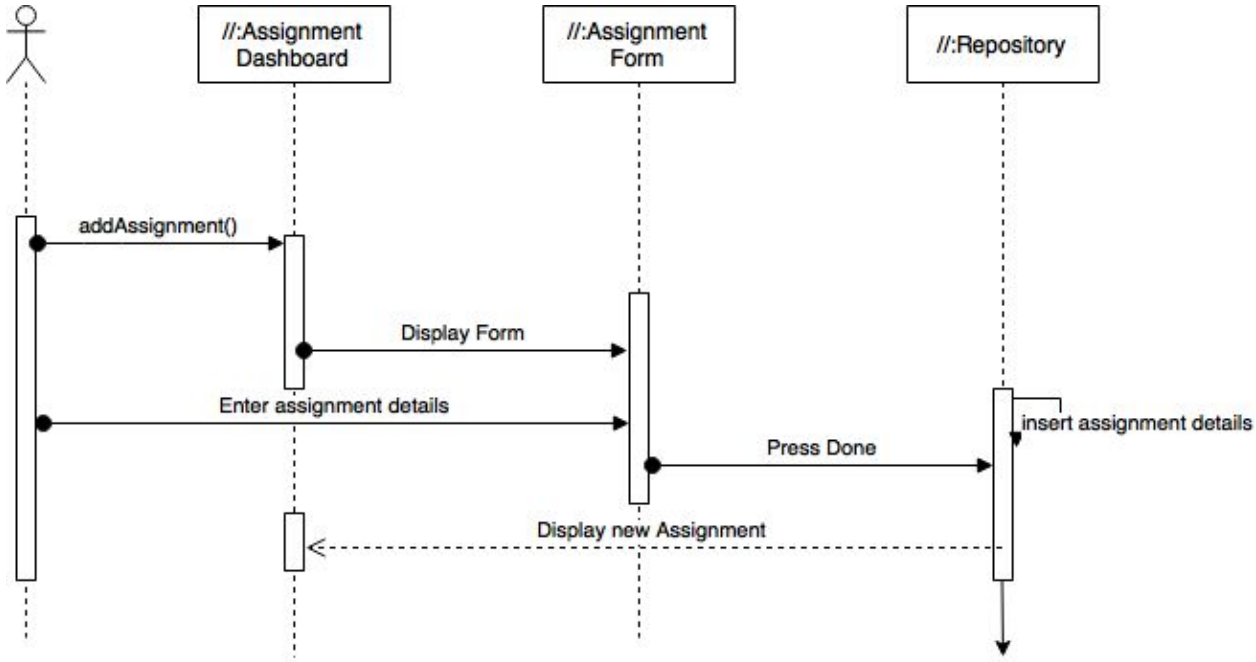


^ Figure 13: "Retrieve Information From The Backend Remotely" Sequence Diagram

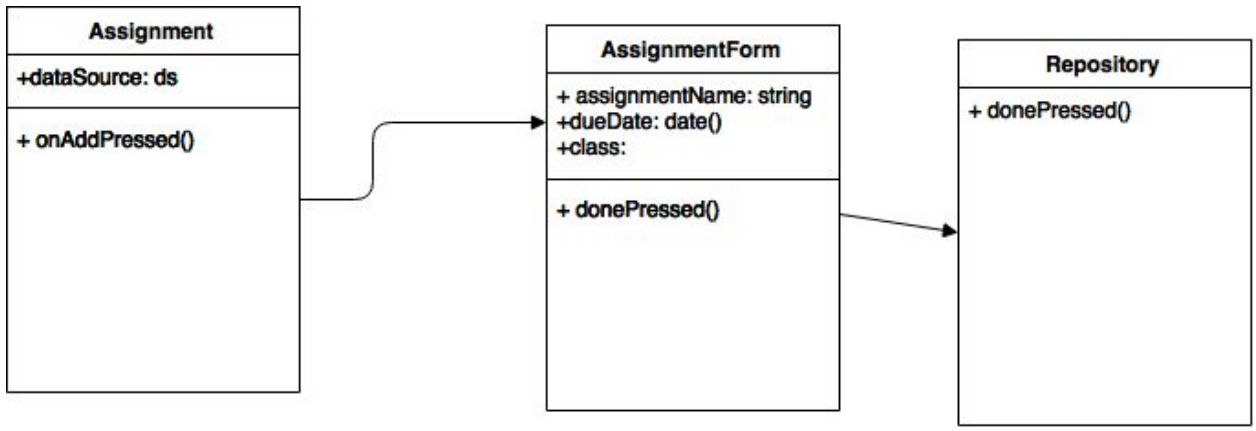


^ Figure 14: User Story #137 Class Diagram

User Story #155-Add a new assignment



^ Figure 15: "Add a new assignment" Sequence Diagram



^ Figure 16: User Story #155 Class Diagram

User Story #156-Add Task

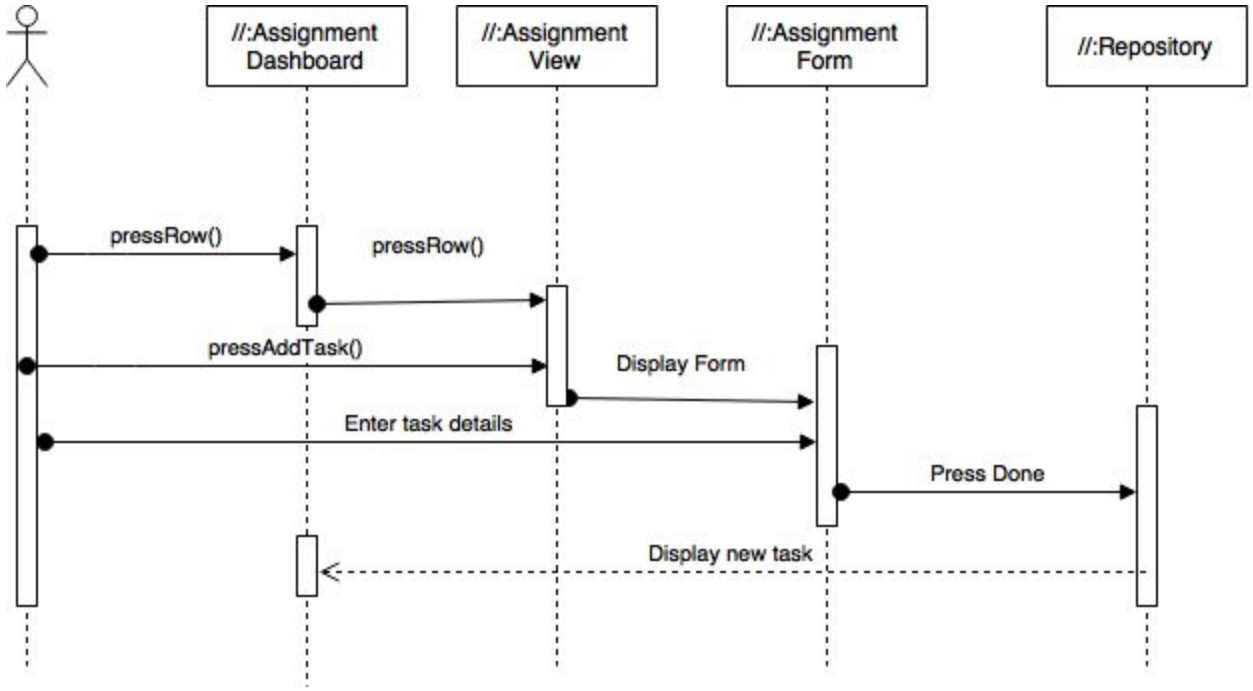
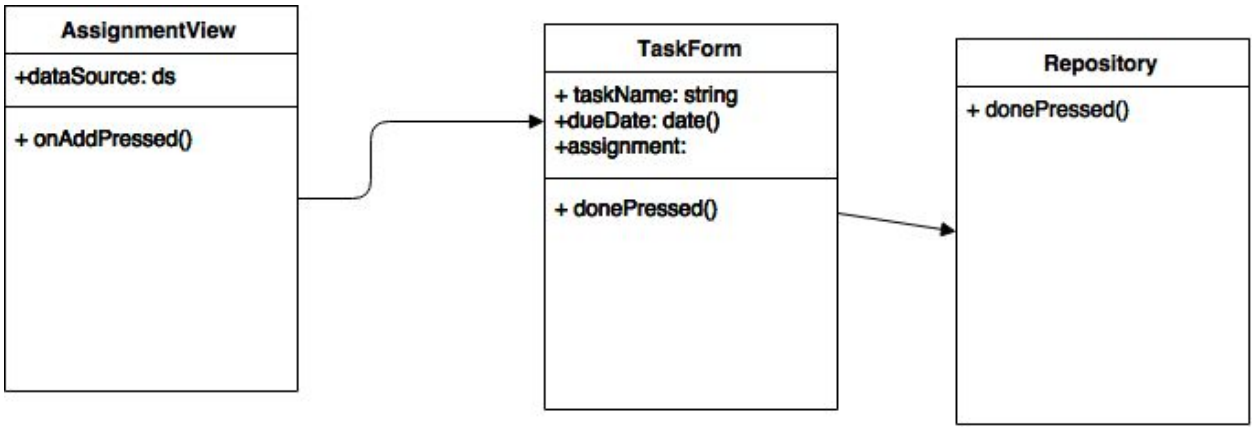
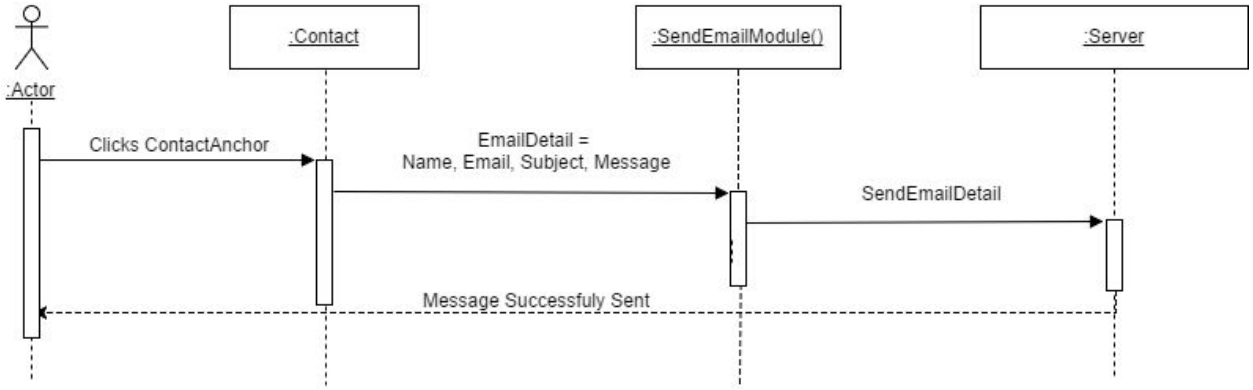


Figure 17: "Add Task" Sequence Diagram

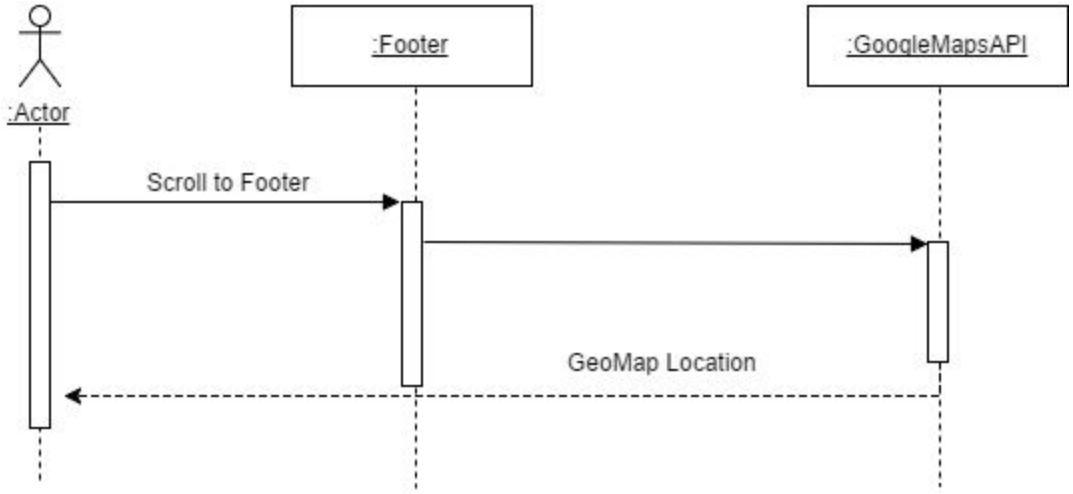


^ Figure 18: User Story #156 Class Diagram

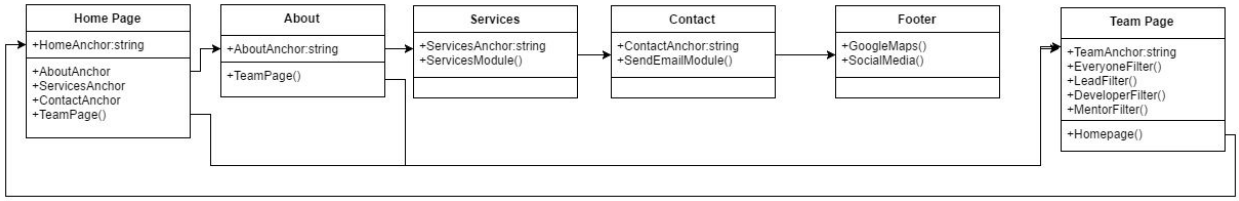
User Story #185 - Website: Create Simple & Beautiful Homepage



^ Figure 19: "SendEmail" Sequence Diagram

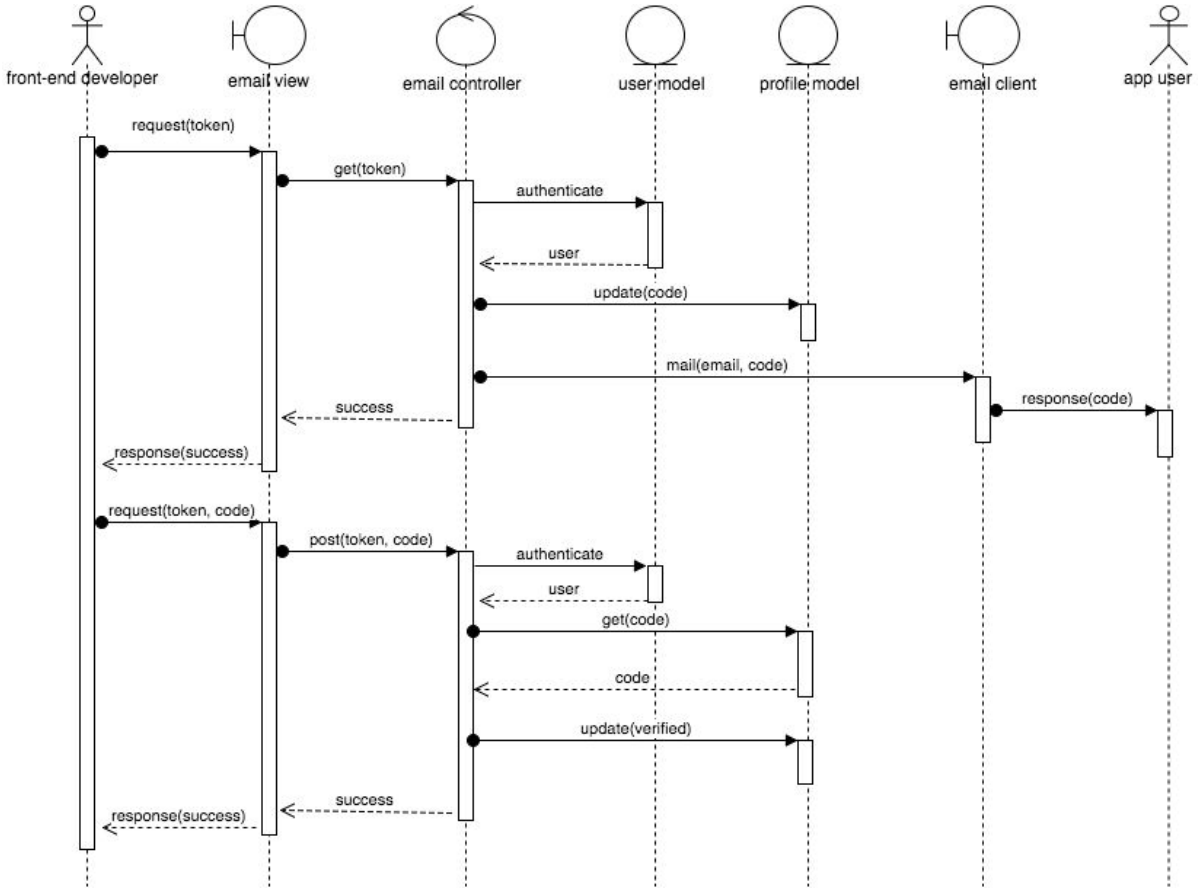


^ Figure 20: "GoogleMaps" Sequence Diagram

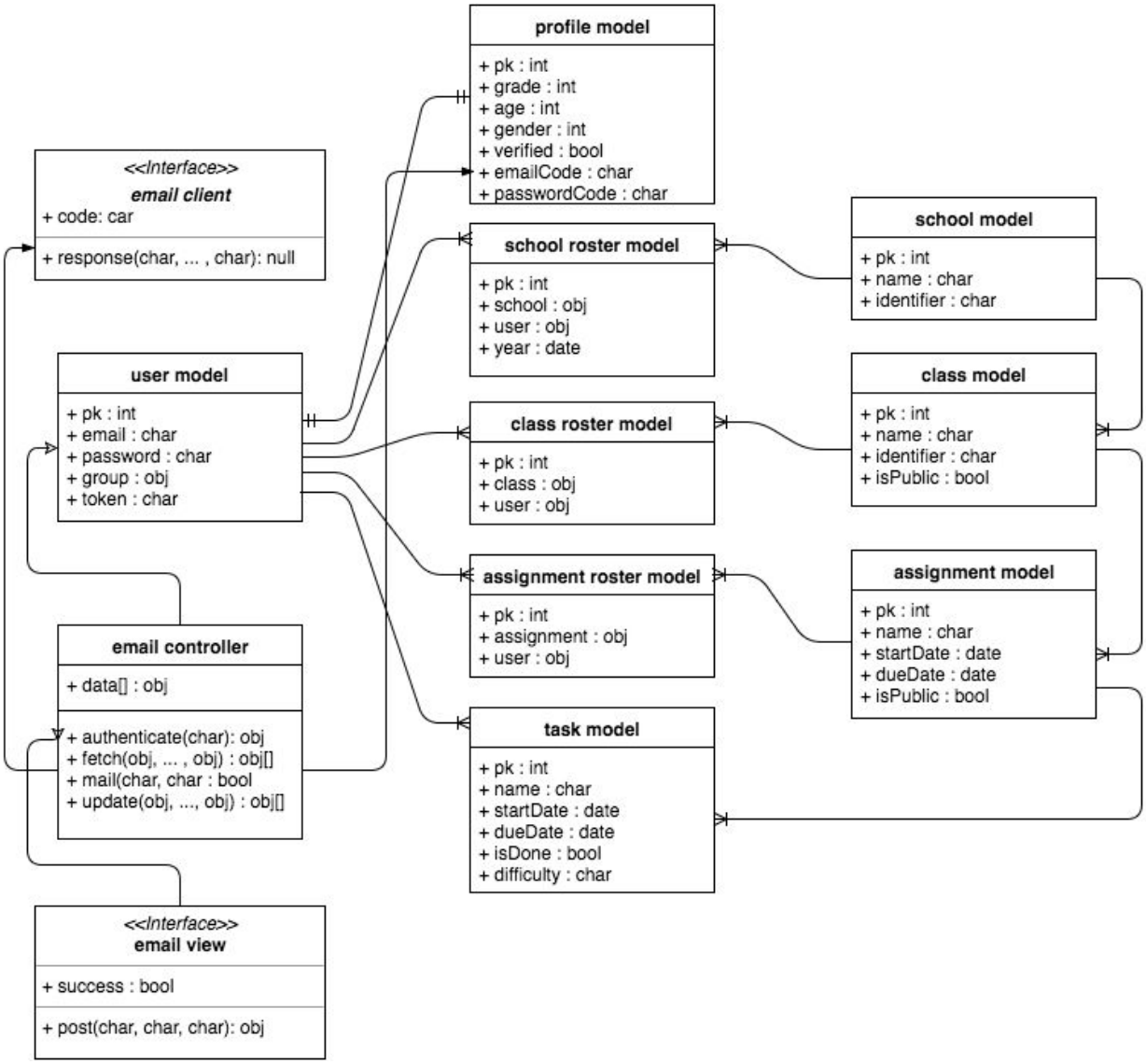


^ *Figure 21: User Story # 185 Class Diagram*

User Story #194-Create infrastructure for email validation

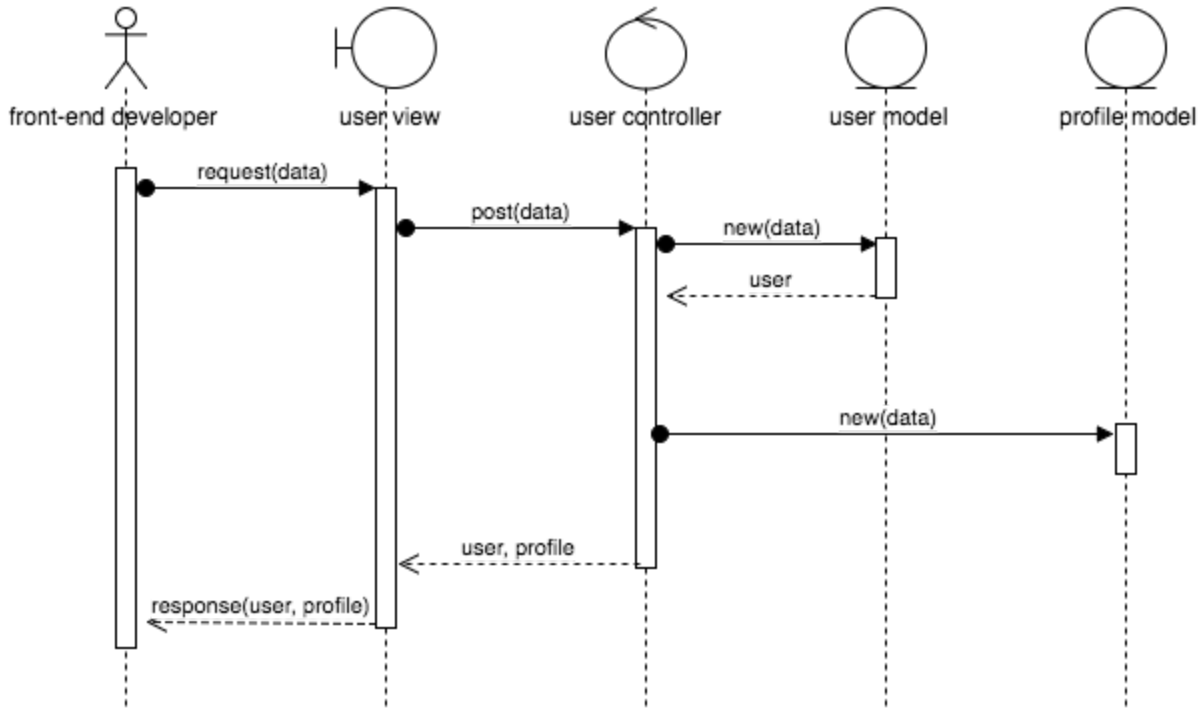


^ *Figure 22: "Validate a User's E-mail" Sequence Diagram*

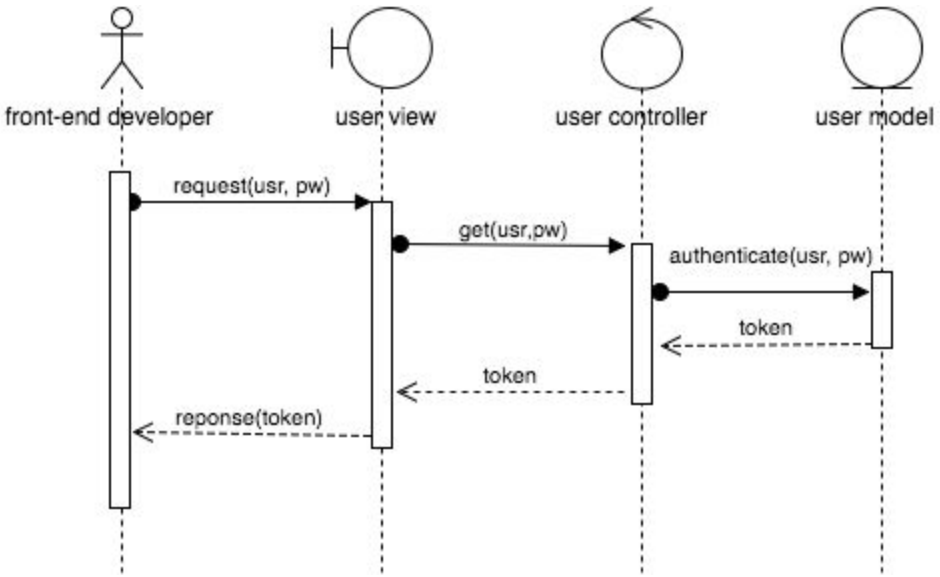


^ Figure 23: User Story #194 Class Diagram

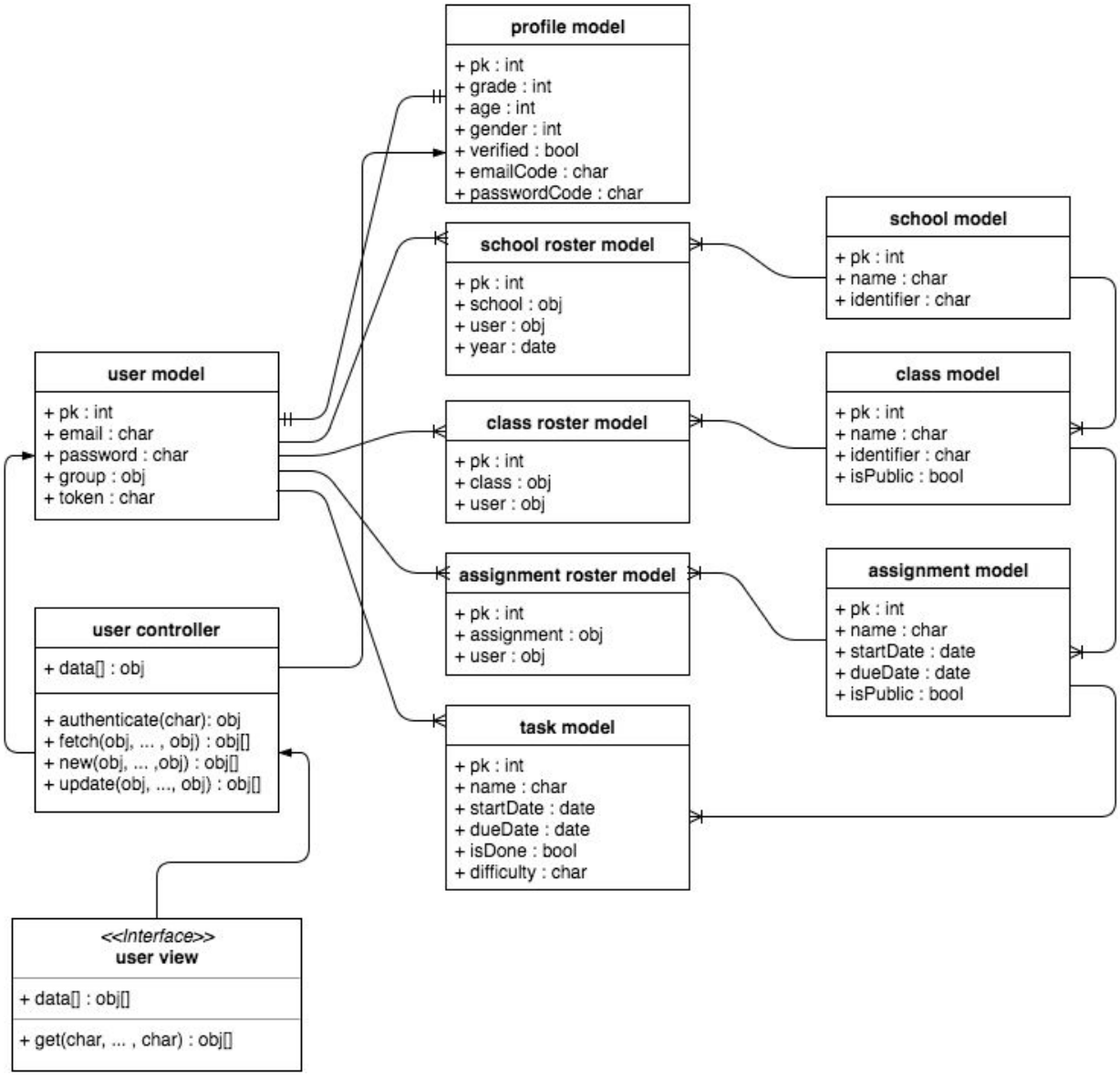
User Story #199-Create login & register backend endpoints



^ *Figure 24: "Create New Users" Sequence Diagram*

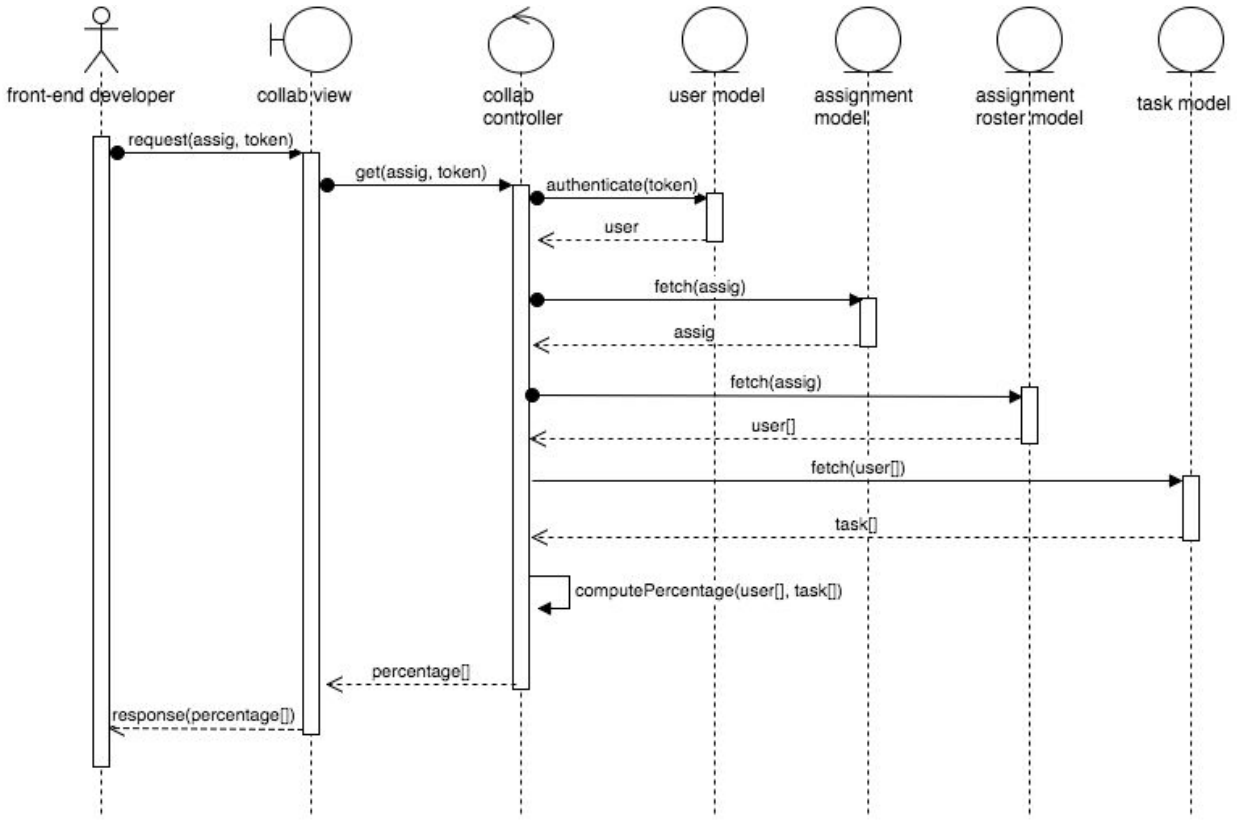


^ Figure 25: "Authenticate Users" Sequence Diagram

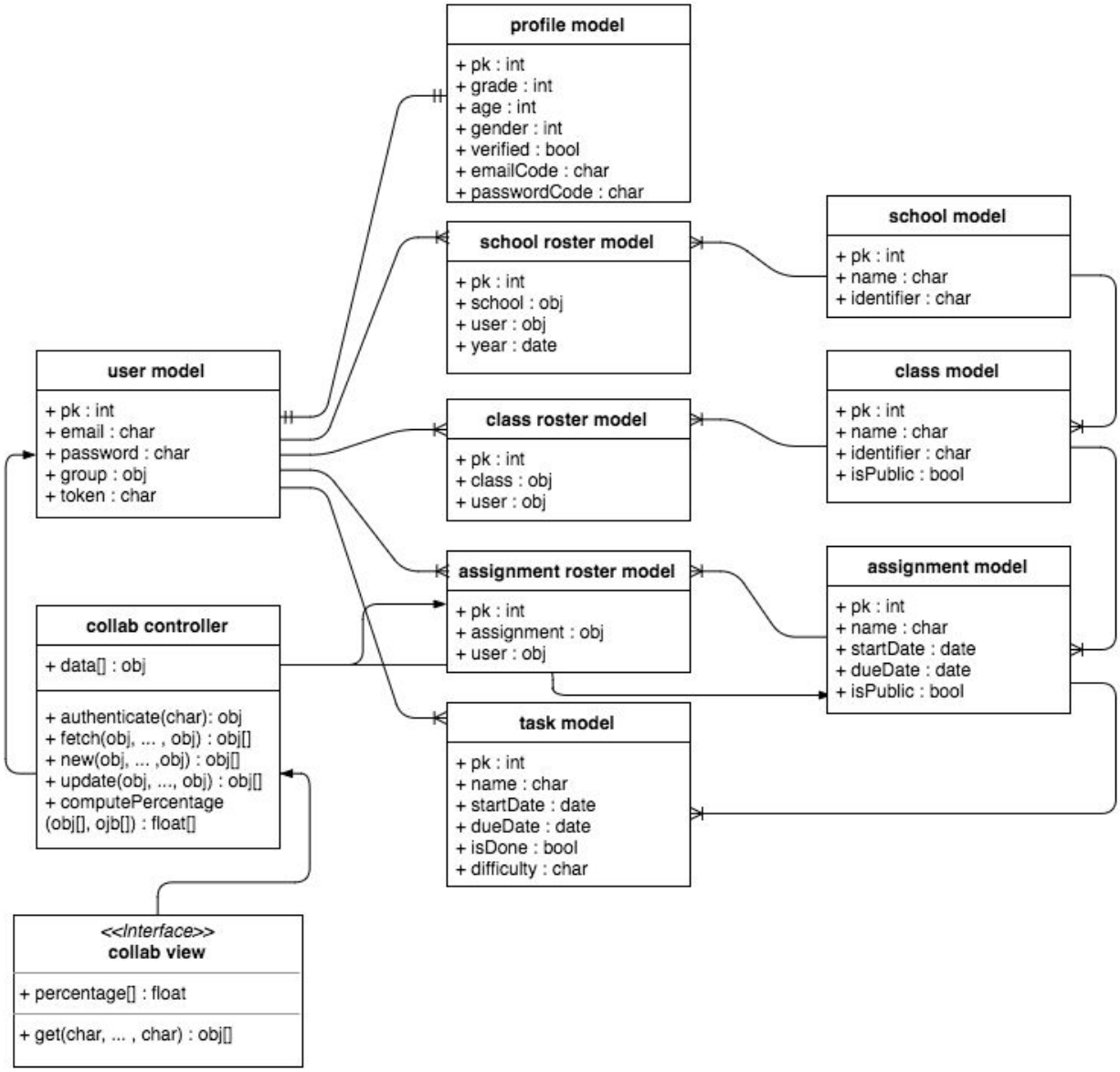


^ Figure 26: User Story #199 Class Diagram

User Story #213-Create “Class Ranking" Page for Assignments (Backend)

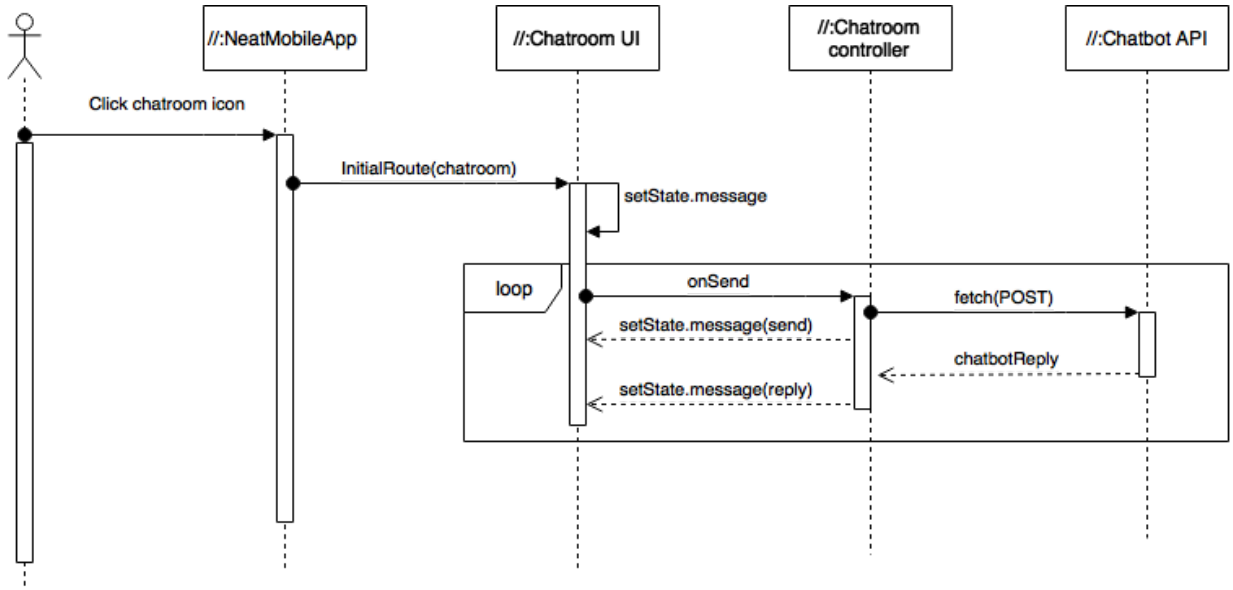


^ Figure 27: “Fetch Assignment Collaboration” Sequence Diagram

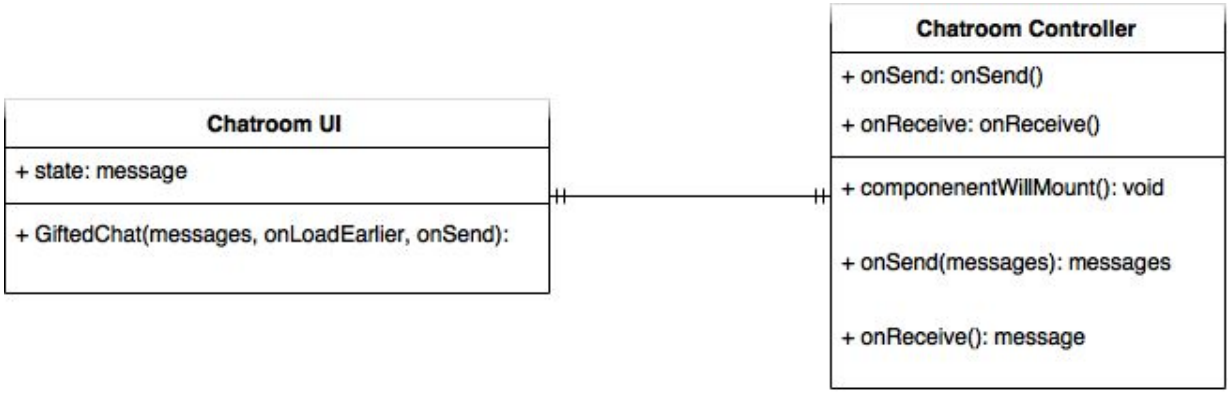


^ Figure 28: User Story #213 Class Diagram

User Story #214-Create chatBot UI

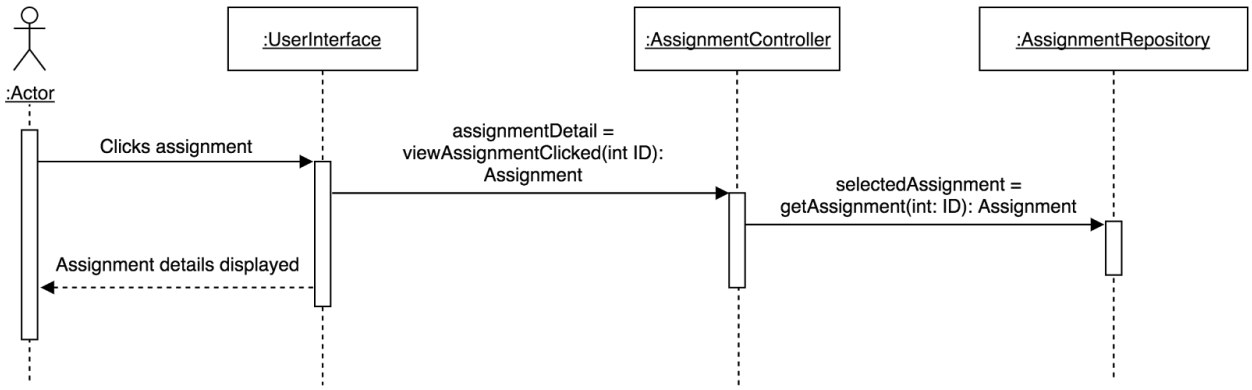


^ Figure 29: "Create Chatbot UI" Sequence Diagram

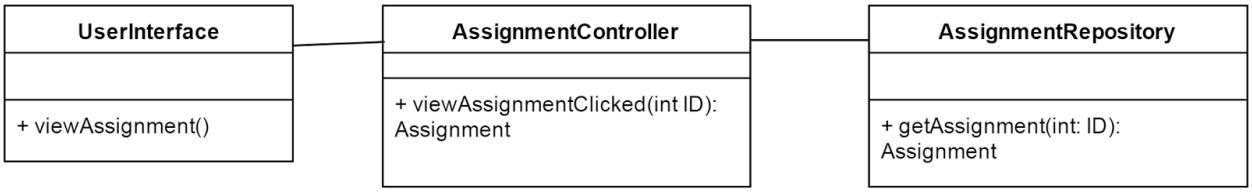


^ Figure 30: User Story #214 Class Diagram

User Story #228-Create Assignment View Page

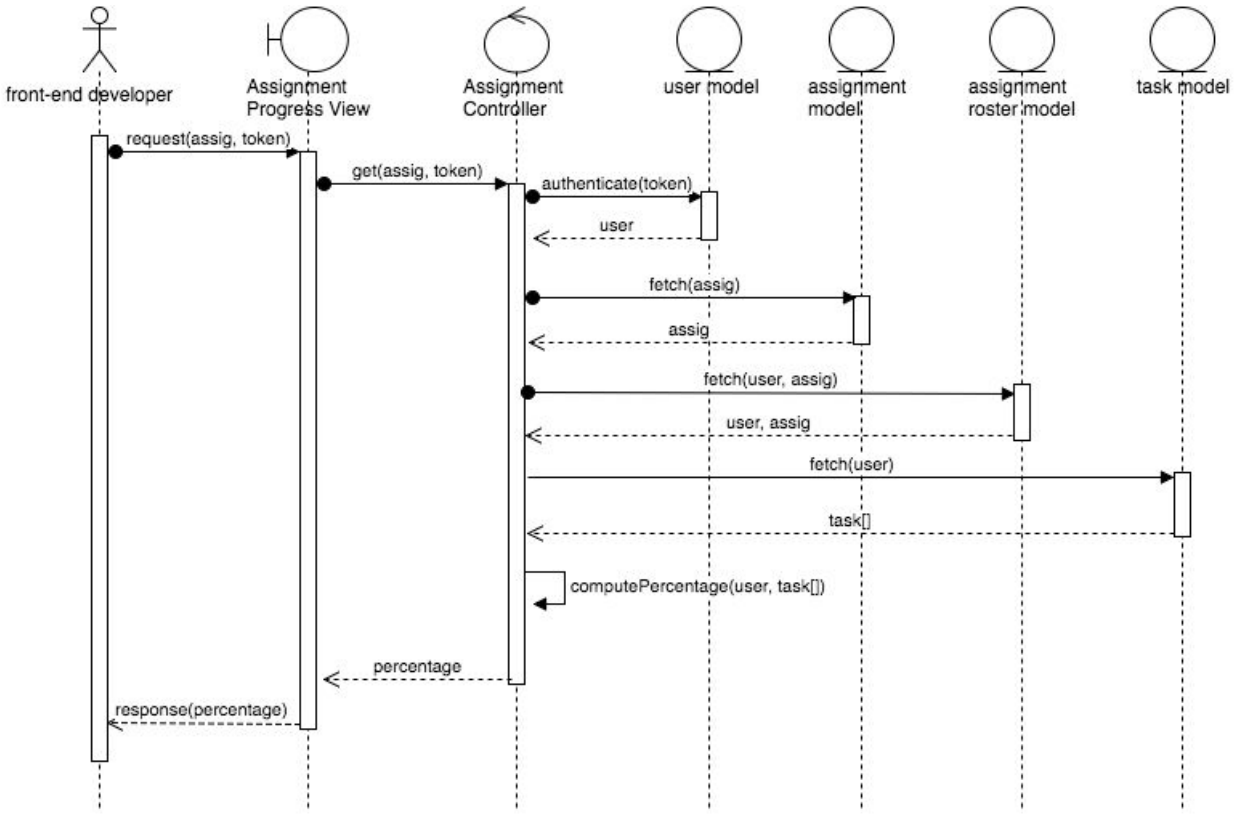


^ *Figure 31: "Create Assignment View Page" Sequence Diagram*

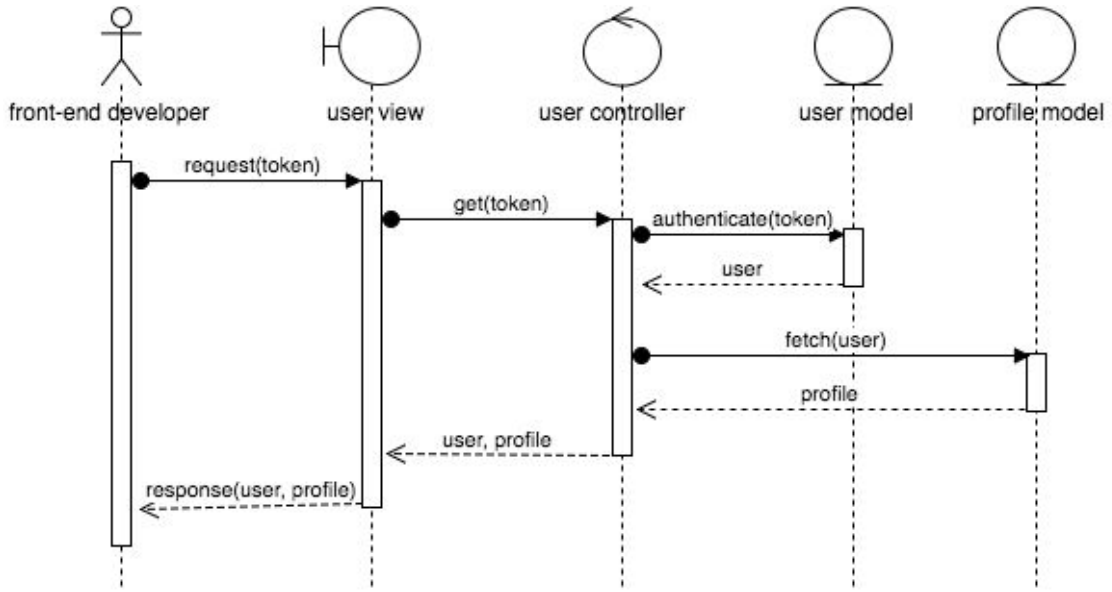


^ *Figure 32: "Create Assignment View Page" Sequence Diagram*

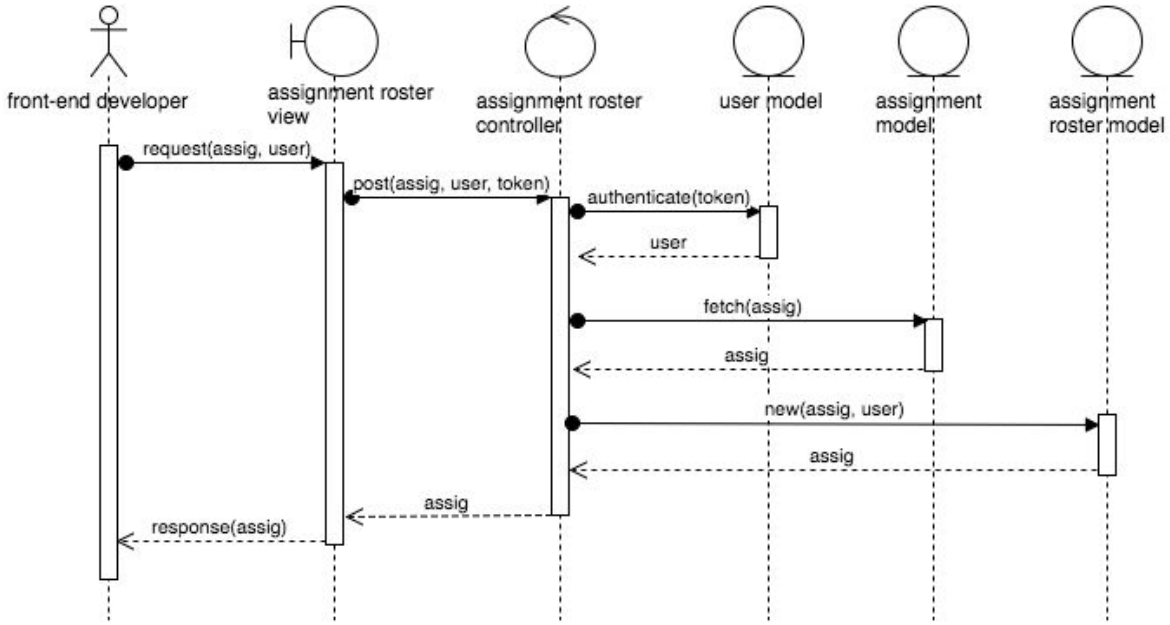
User Story #229-Create endpoints for returning data to frontend



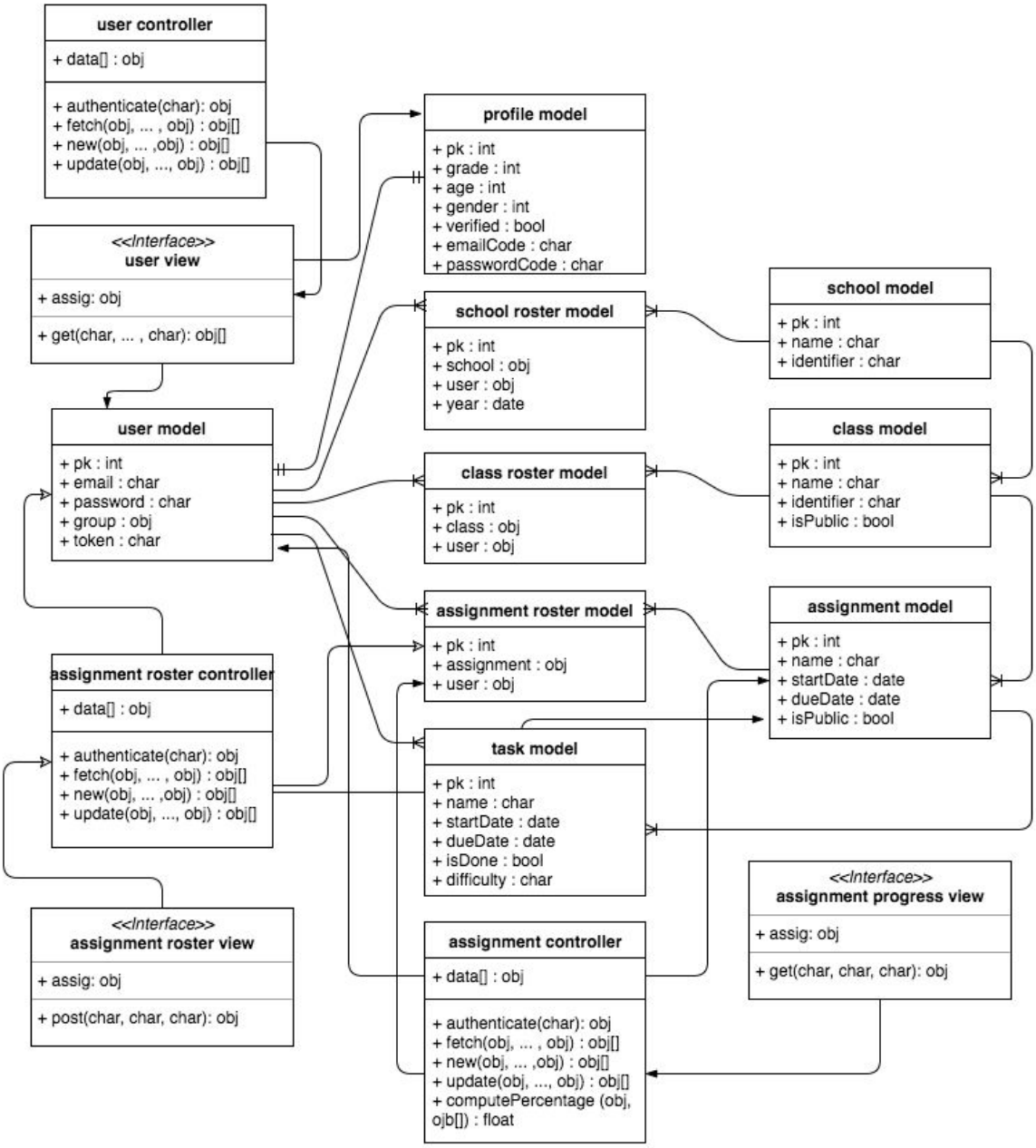
^ Figure 33: "Fetch User Assignment Progress" Sequence Diagram



^ Figure 34: "Fetch User Info" Sequence Diagram

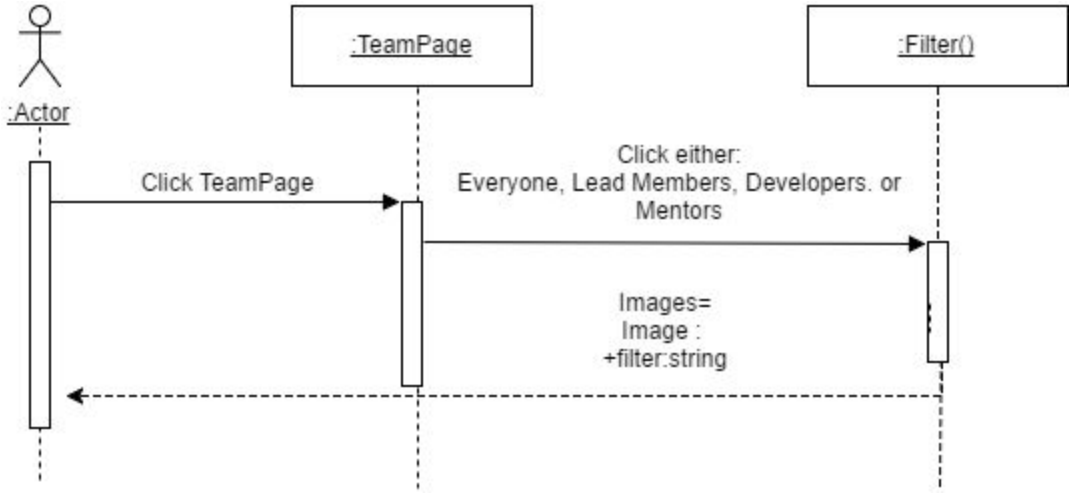


^ Figure 35: "Add User to Assignment" Sequence Diagram

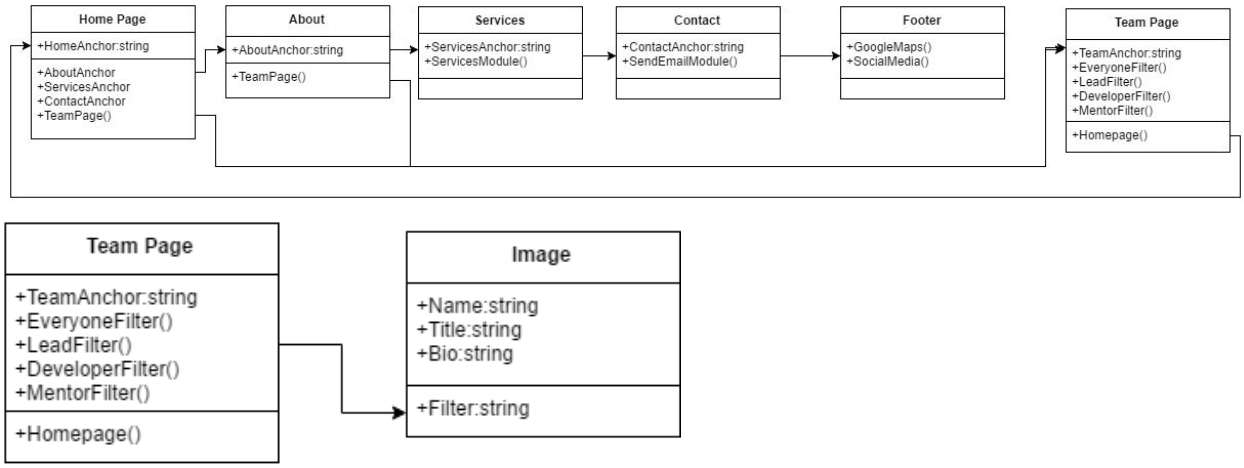


^ Figure 36: User Story #229 Class Diagram

User Story #269 - Neat Study Team Page

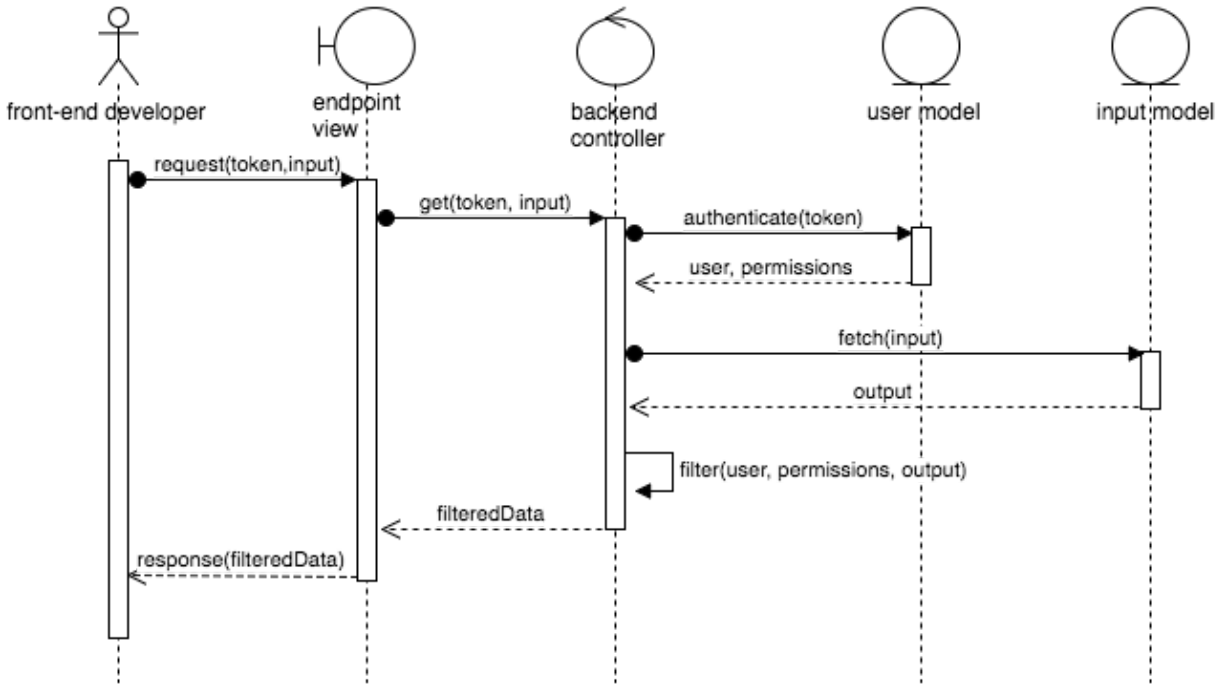


^ Figure 37: "Filter Team Page Bios" Sequence Diagram

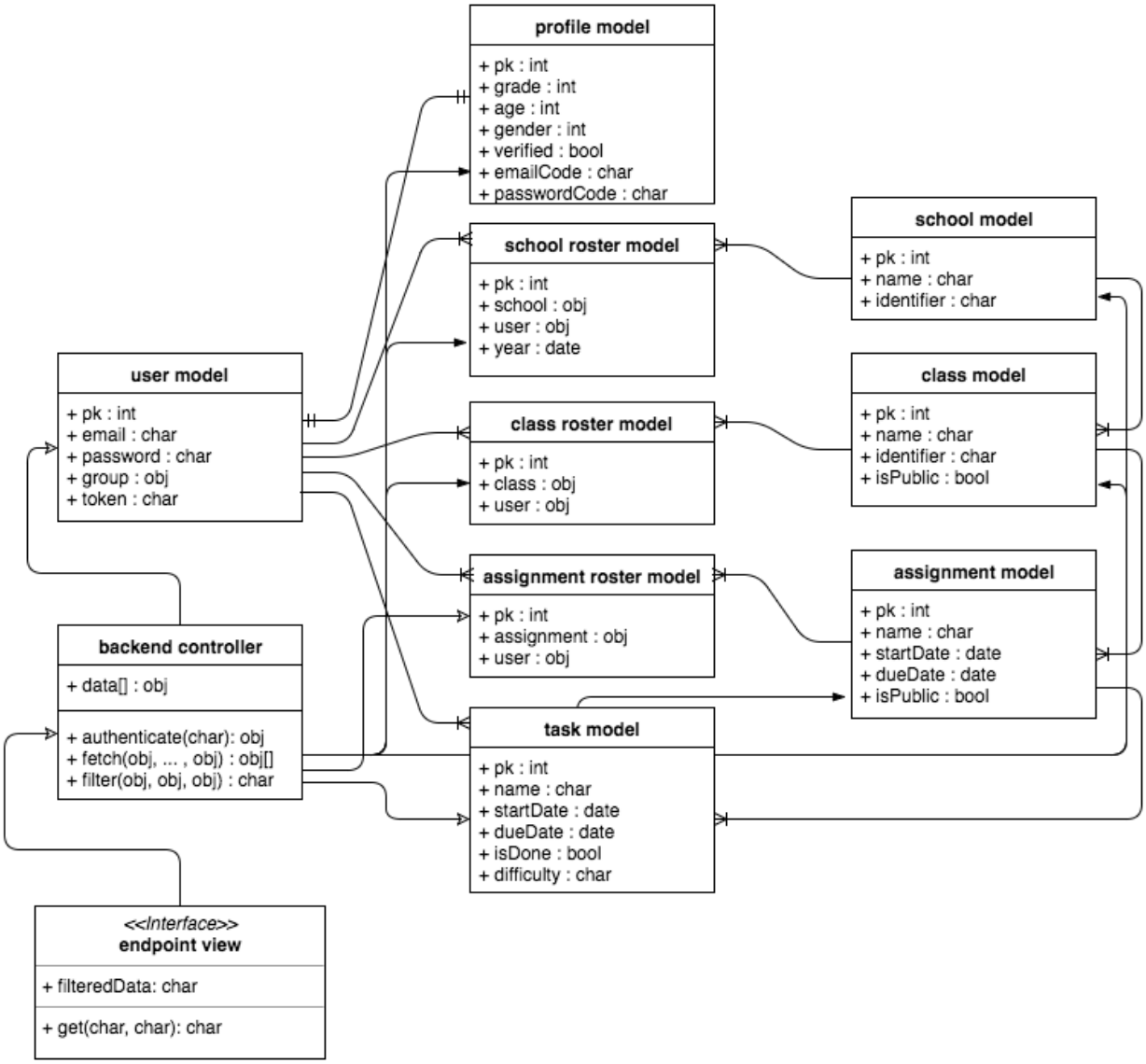


^ Figure 38: User Story #269 Class Diagram

User Story #270-Create object-level permissions in the backend

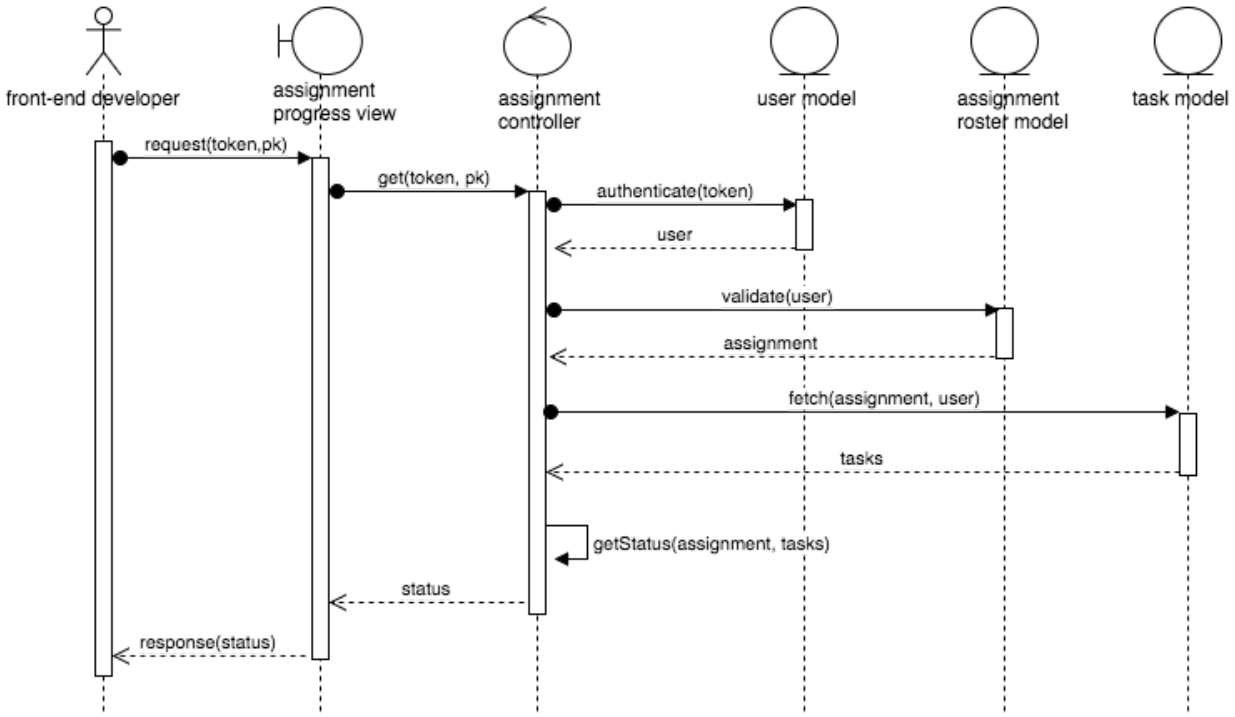


^ *Figure 39: "Fetch Logged-In User's Data" Sequence Diagram*

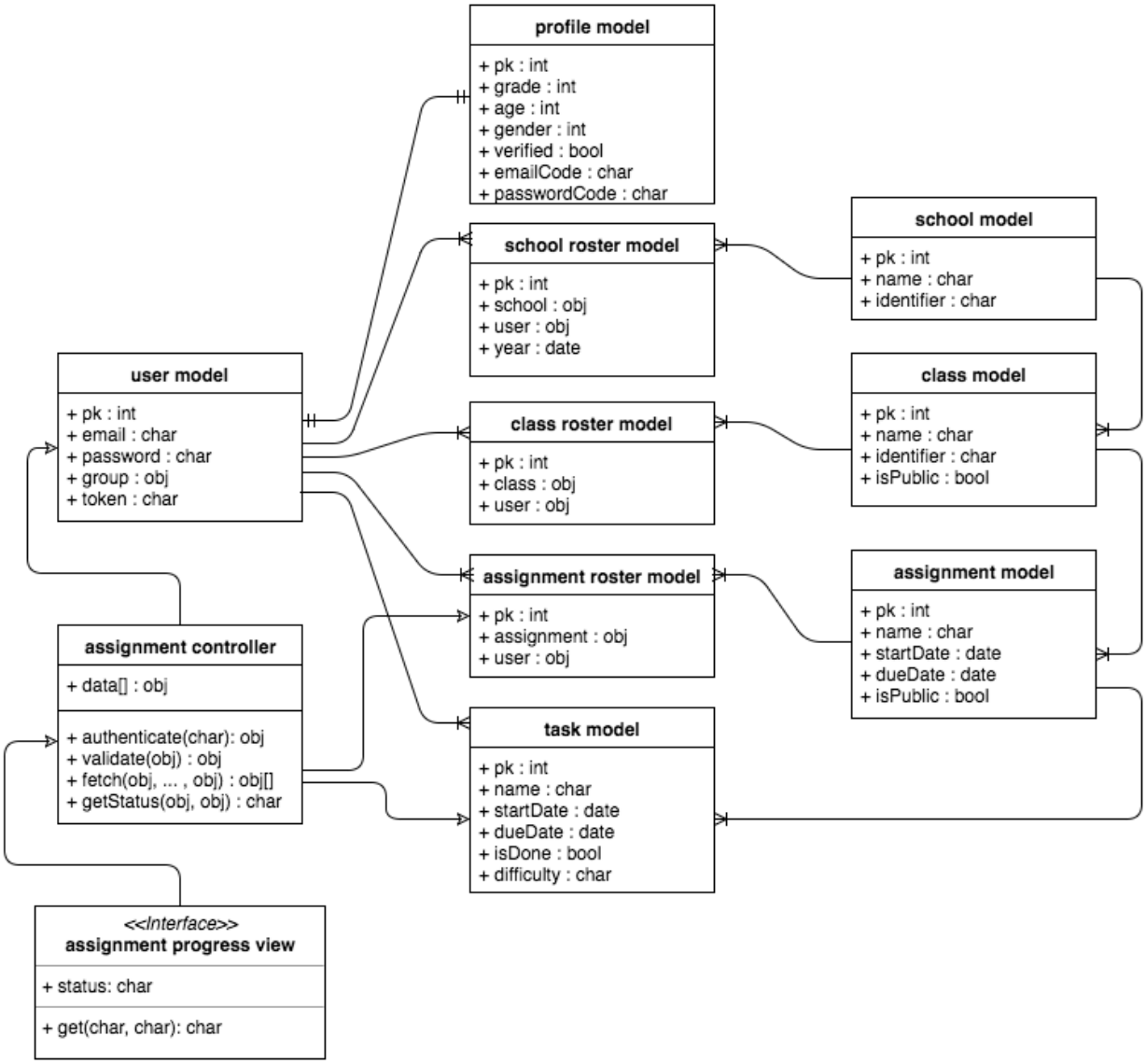


^ Figure 40: User Story #270 Class Diagram

User Story #271-Implement new progress algorithm

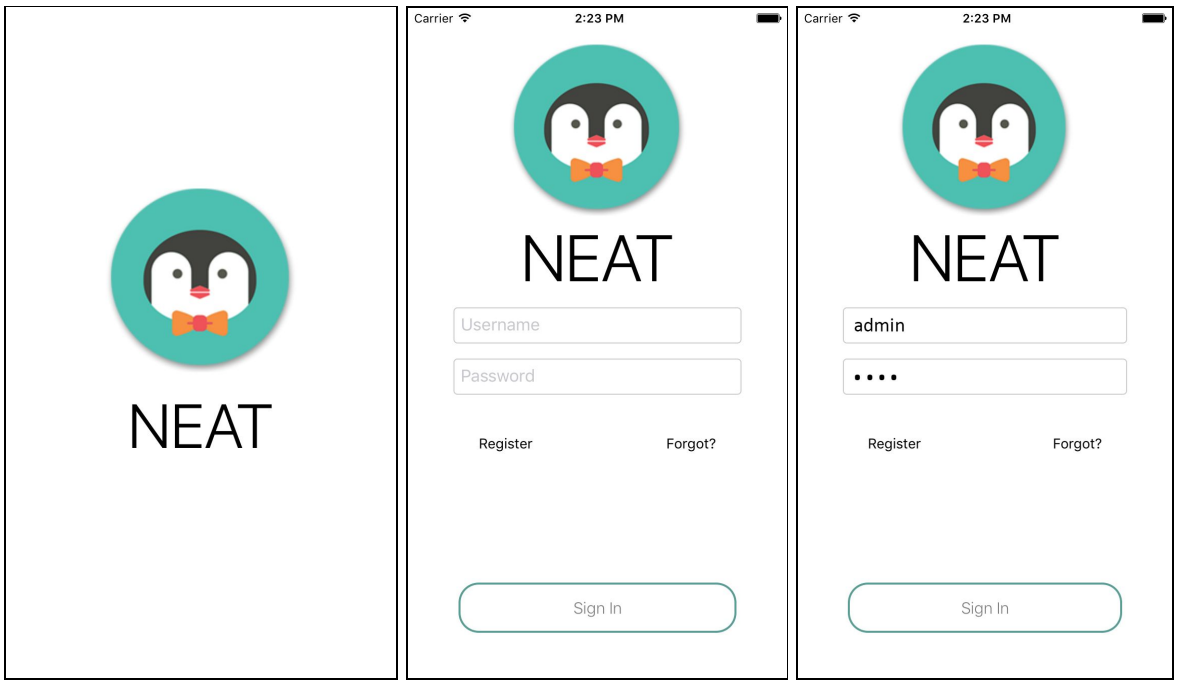


^ Figure 41: "Fetch Assignment Smart Status" Sequence Diagram

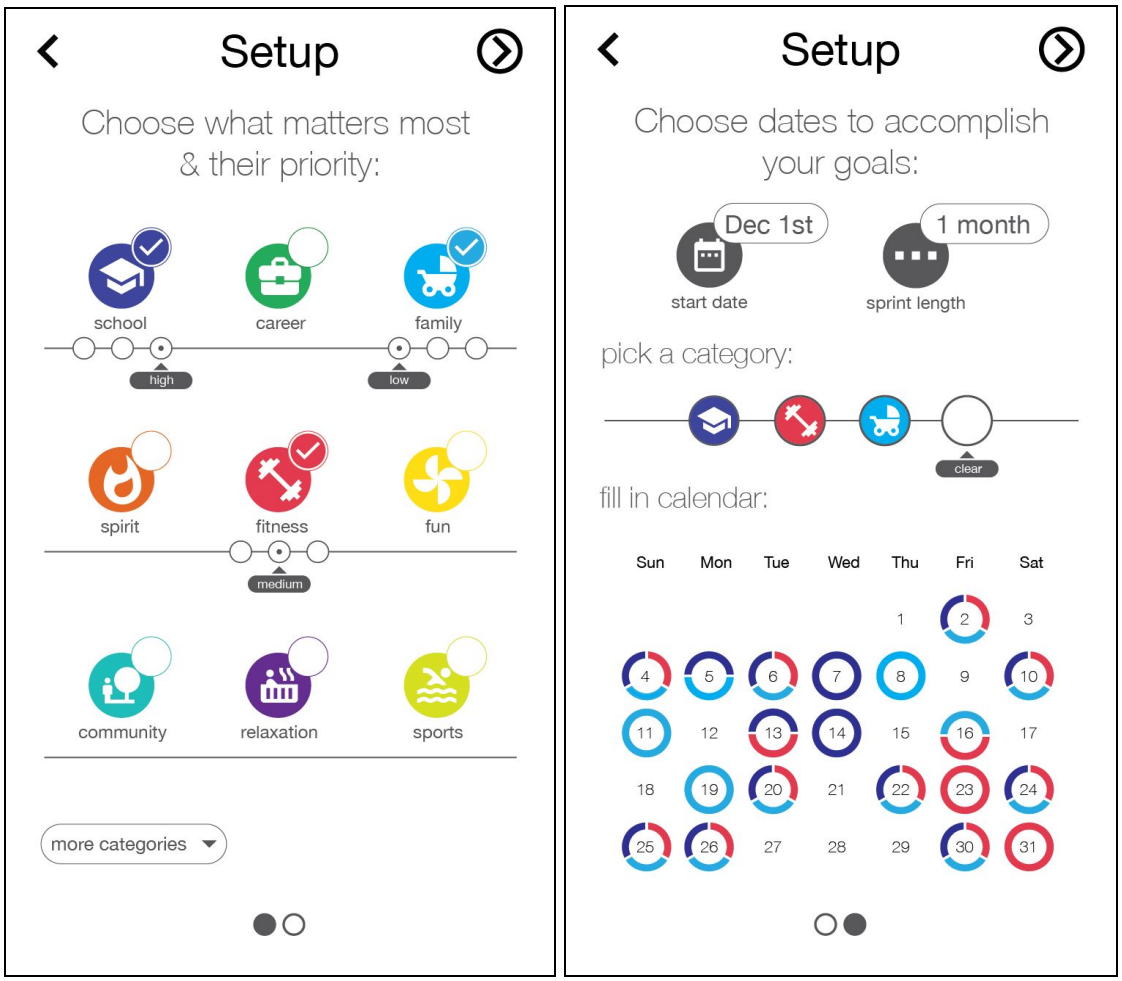


^ *Figure 42: User Story #271 Class Diagram*

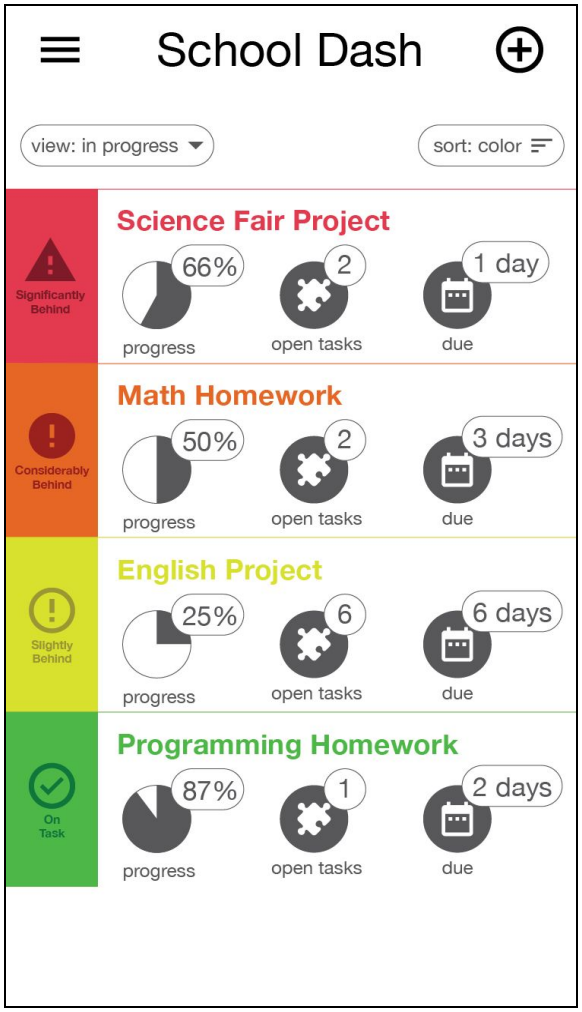
Appendix B - User Interface Design



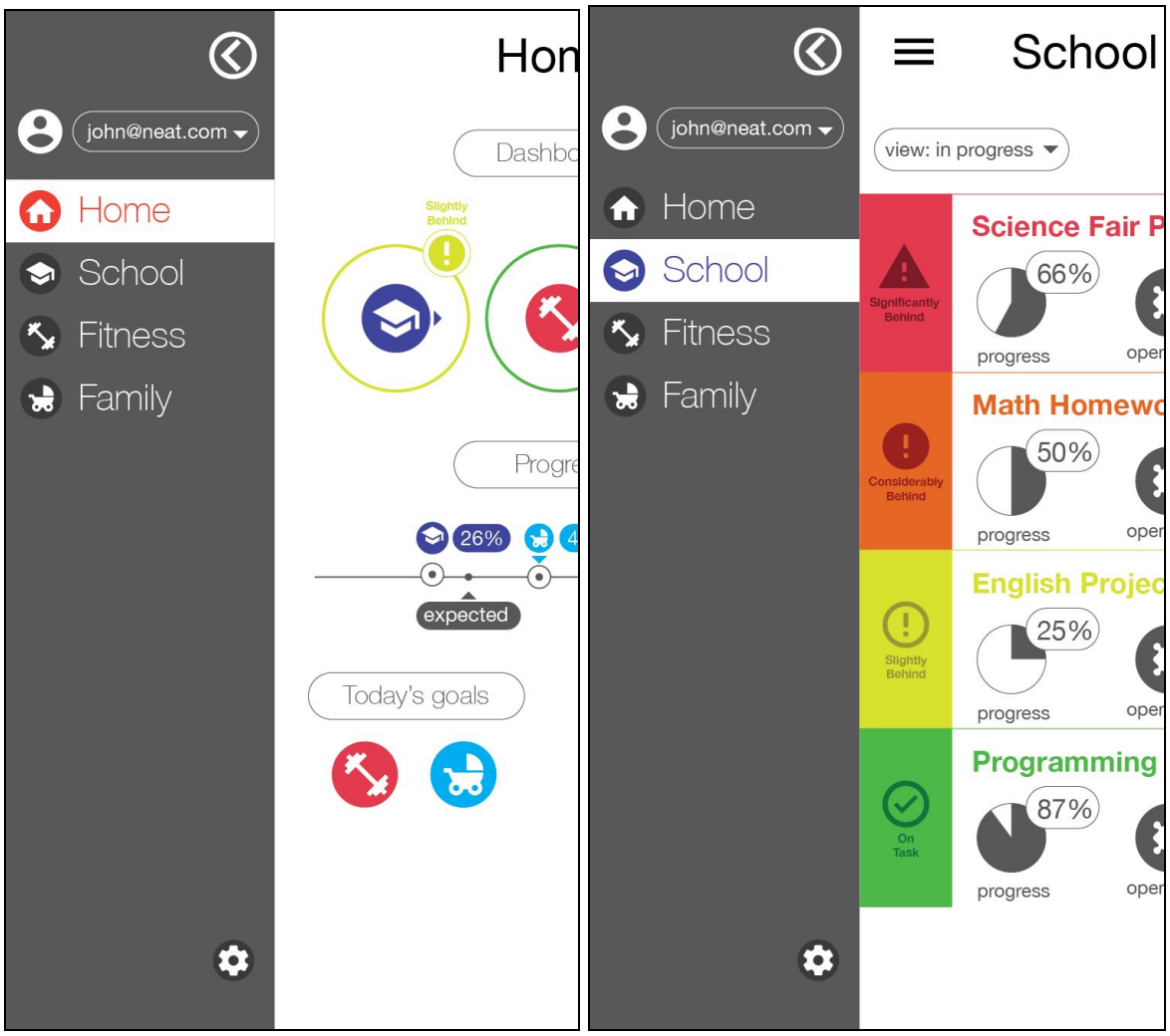
^ *Figures: Loading Screen + First time Logging In*



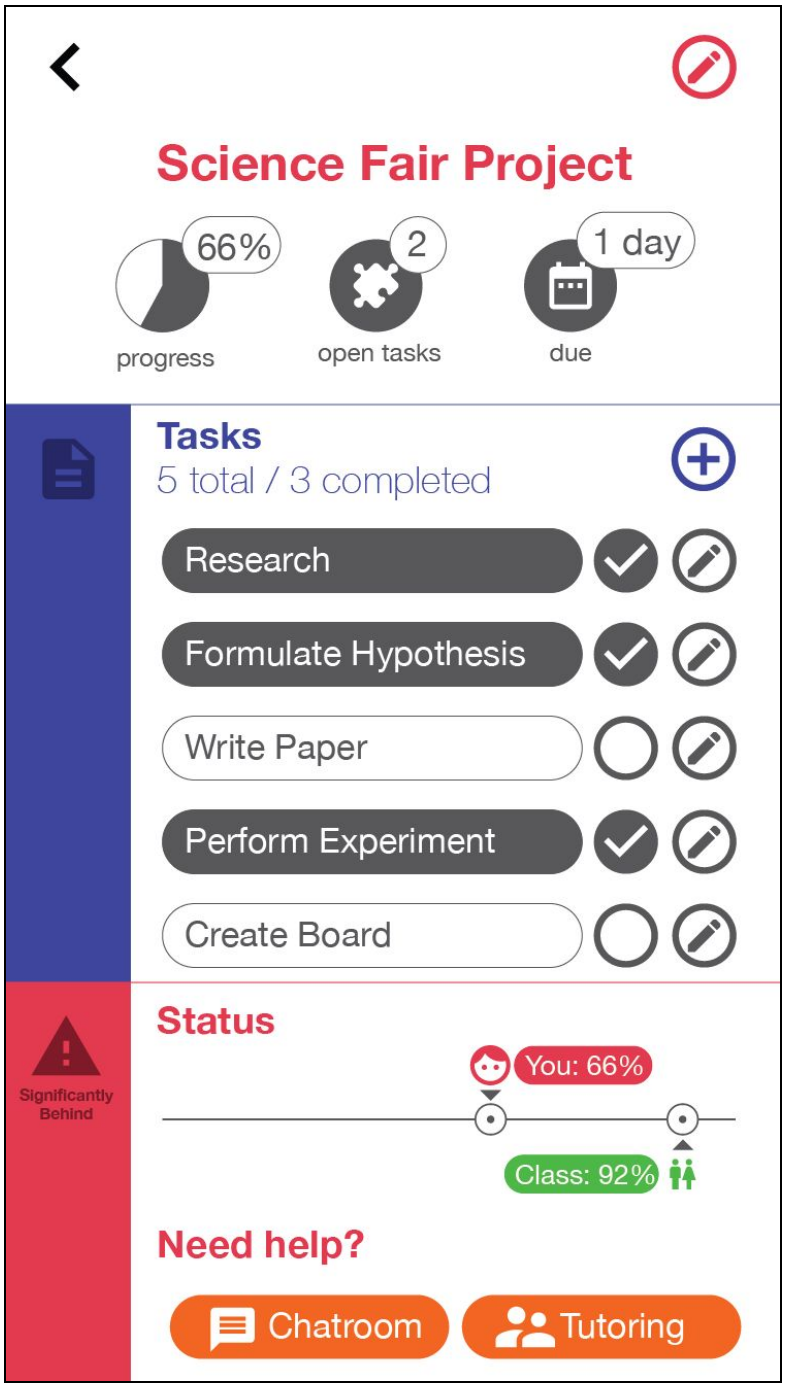
^ *Figures: First time user setup*



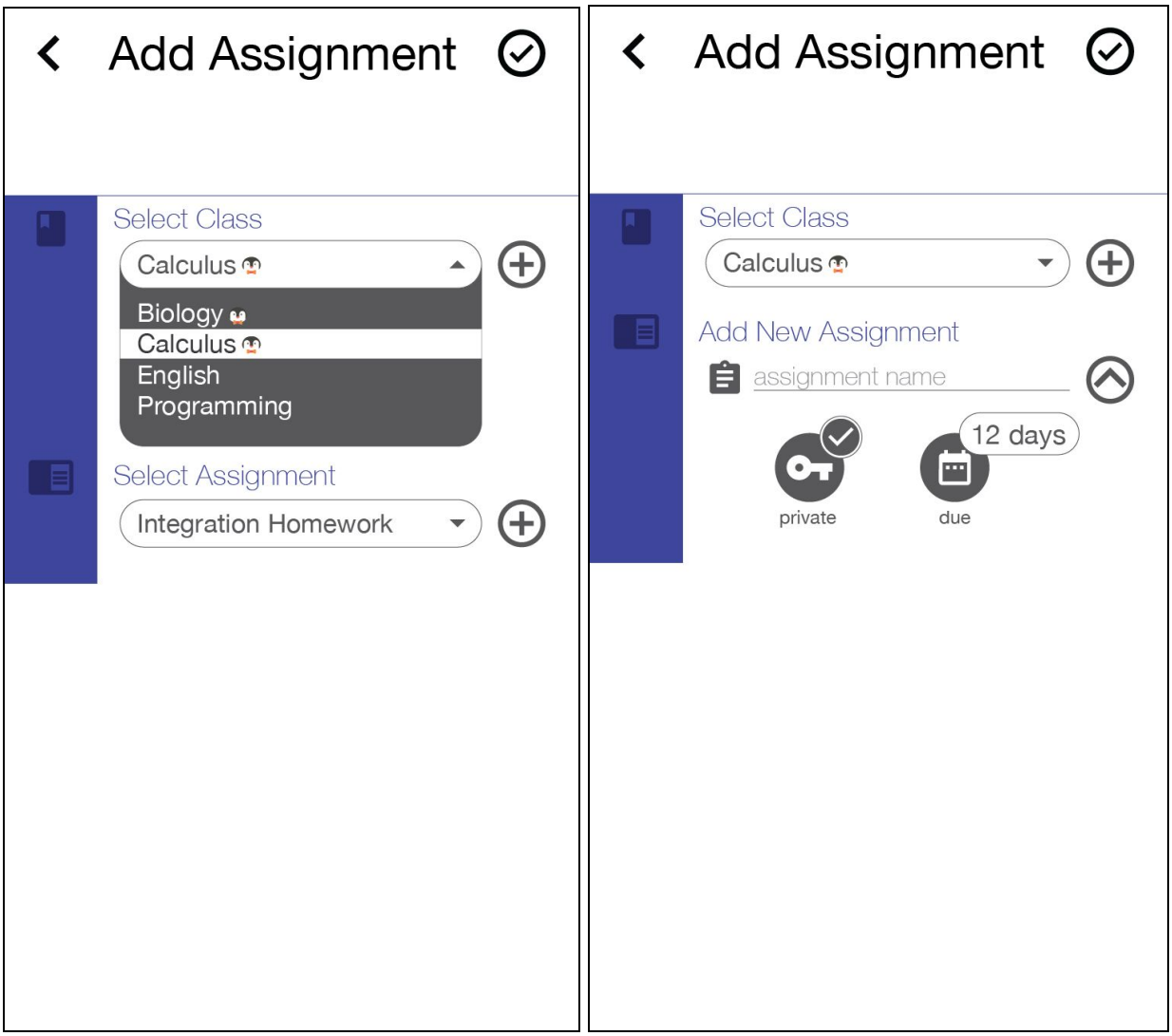
^ *Figures: Dashboards*



^ *Figures: Sidebars*



^ *Figure: Assignment Detail*



^ *Figures: Add Assignment*

Appendix C - Sprint Review Reports

Sprint 1

Date: September 9, 2016
Attendees: Giselle, Pierre, Fernando, Justin, Pachev, Gabriel
Start time: 1:30PM
End time: 2:05PM

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

- 119
- 120
- 123
- 124
- 137
- 146
- 138

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

- 199

Sprint 2

Date: September 26, 2016
Attendees: Giselle, Pierre, Fernando, Justin, Pachev, Gabriel
Start time: 2:25PM
End time: 3:11PM

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

- 199
- 165

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

- 191
- 221

Spring 3

Date: October 10, 2016
Attendees: Giselle, Pierre, Fernando, Justin, Pachev, Gabriel
Start time: 2:40PM
End time: 3:13PM

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

- 194
- 191
- 221
- 185

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

- None

Sprint 4

Date: October 25, 2016
Attendees: Giselle, Pierre, Fernando,Justin, Pachev, Gabriel
Start time: 1:14PM
End time: 1:52PM

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

- 213
- 229
- 220
- 132

- 118
- 228
- 130
- 155
- 156
- 129
- 131
- 237

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

- 212

Sprint 5

Date: November 7, 2016
Attendees: Giselle, Pierre, Fernando, Justin, Pachev, Gabriel
Start time: 3:00PM
End time: 3:42PM

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

- 126
- 134
- 117

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

- 214
- 274
- 275

Sprint 6

Date: November 21, 2016
Attendees: Giselle, Pierre, Fernando, Justin, Pachev, Gabriel
Start time: 1:20PM

End time: 1:40PM

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

- 231
- 282
- 275
- 274
- 214

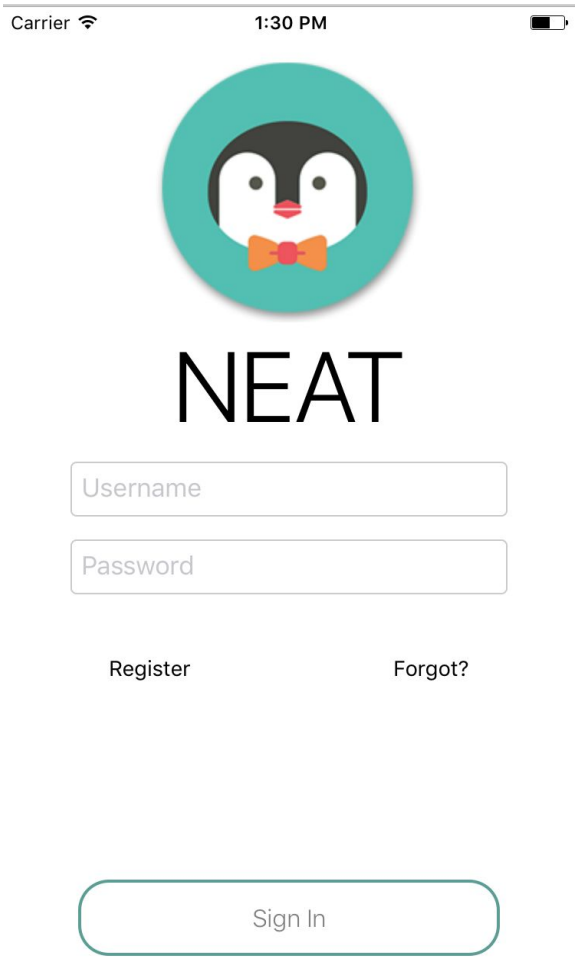
The following ones were rejected and moved back to the product backlog to be assigned to a future sprint at a future Spring Planning meeting.

- 234
- 178

Appendix D - User Manuals, Installation/Maintenance Document, Shortcomings/Wishlist Document and Sprint Retrospective

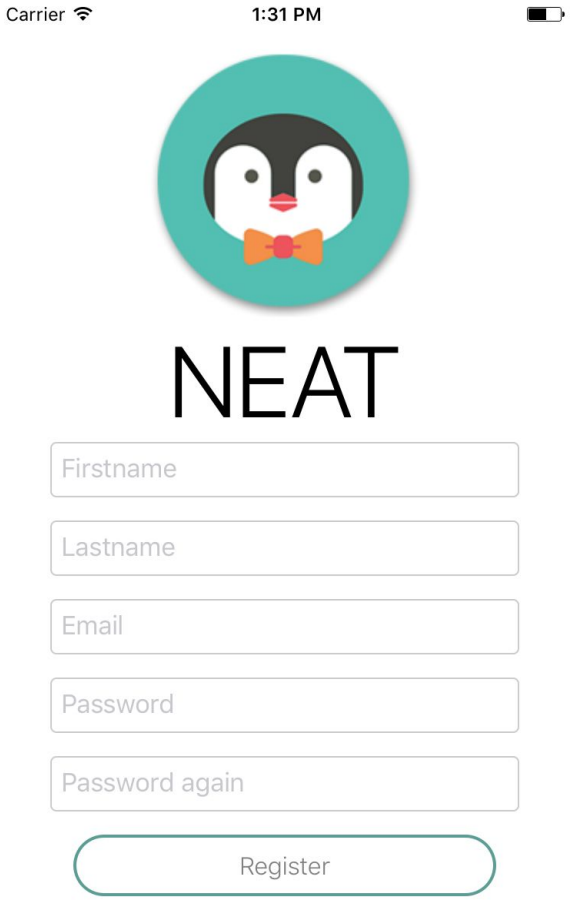
User Manual

- *Login*
  - Type your username (e-mail) and password, and click on “Sign In” button



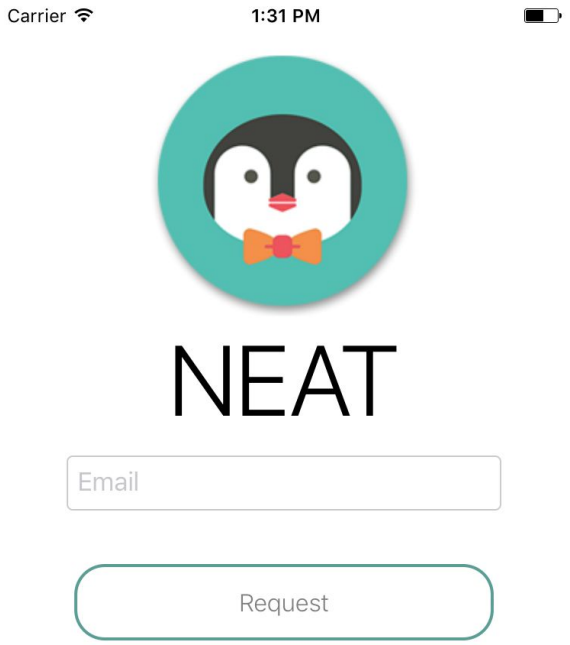
  -
- *Register*

- Click on “Register” to create a new account and fill in the information

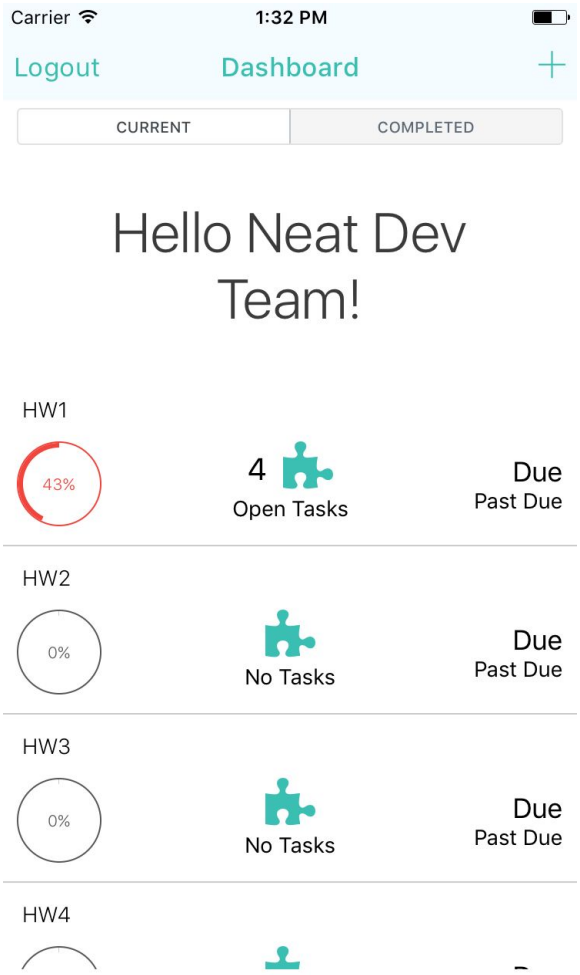


-

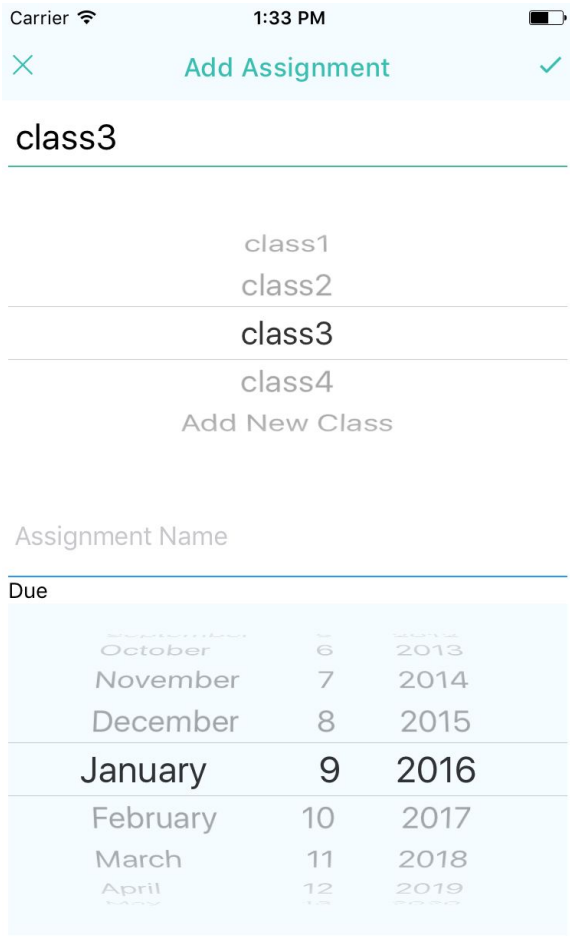
- *Forgot Password*
  - If you forgot your password, hit “forgot?” button and enter your e-mail. You will get a code to enter in the app that allows you to reset your password.



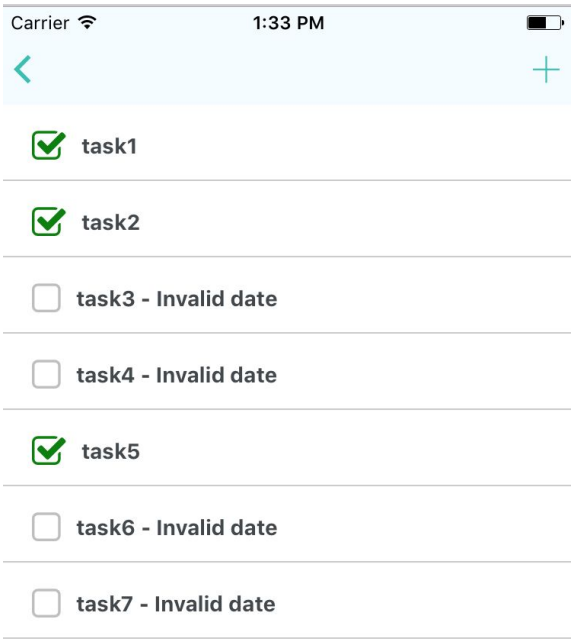
- *Dashboard*
    - Once logged in, you will see your dashboard. This is a list of all your assignments with information about them.



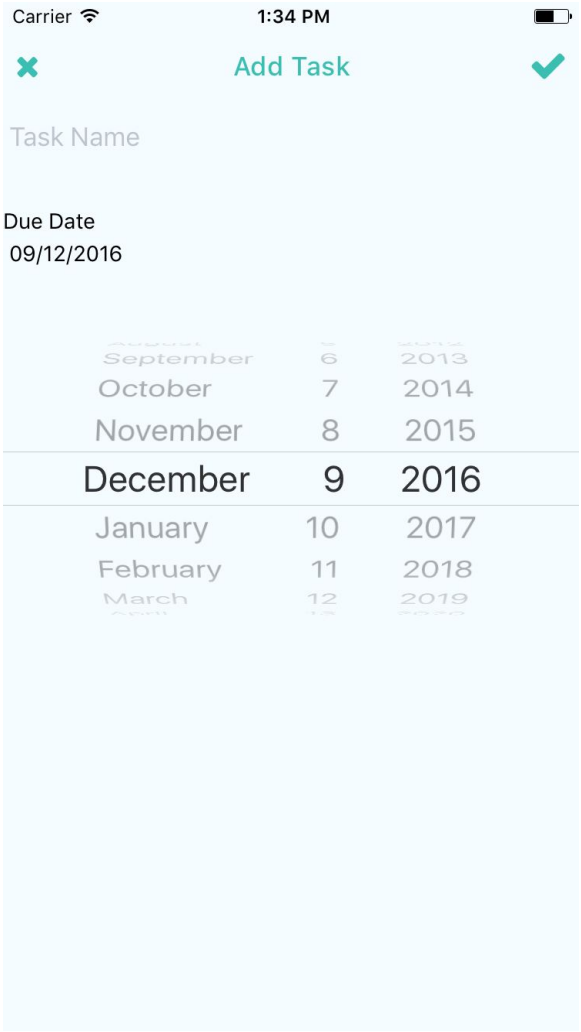
- *Add Assignment*
  - To add an assignment, click on the “+” button in your dashboard. You can select a class you have already joined, or create a new one. Afterwards, you can select to join an assignment (if the class is a neat class), or create a new assignment.



- *View Assignment Detail*
  - Back at the dashboard, click on any assignment to view more information. You will see a list of tasks that belong to the assignment.



- *Add Task*
  - Click on the “+” button in the assignment view to add a new task. Fill out the task name and the due date.



-

Installation & Maintenance Document

Backend Installation

Django and Django REST Framework For Neat

Rest API Documentation

For an in depth documentation go to http://localhost/docs or the official docs at http://52.87.176.128/docs

Installation

Python Virtual Environment

This project is best ran in a virtual environment. You can use pyvenv, which comes with python 3 and greater. The virtual environment lets you run different versions of python and packages from other projects.

Installation (Unix)

First install python3+ on your machine and then download and install pip. Then from the root of the project run:

1. `pyvenv venv` - Create a virtual environment in the venv folder
2. `source venv/bin/activate` - Load the environment
3. `pip install -r neatBackend/Requirements.txt` - Install dependencies
4. `deactivate` - Unloads the environment

Installation (Windows)

Note - Most documentation is for unix systems. Differences between windows and unix are: `env\Scripts\` instead of `env/bin/` and libraries go in `env\Lib\` rather than `env/lib/`)

First install python3+ on your machine and then download and install pip. Then from the root of the project run:

1. `pip install virtualenv` - Install virtualenv if not already done so. Create a virtual environment in the venv folder
2. `virtualenv venv` - This creates will create a series of directories and scripts
3. `venv/Scripts/activate` - Load the enviroment (There should be a (venv) before the current directory path name
4. `pip install -r neatBackend/Requirements.txt` - Install dependencies

5. `deactivate` - Unloads the environment

Mysql

Download the latest version of Mysql, and follow the instructions for installing on your system.

Setup

Once it has been installed, you need to start the mysql server and create a database and a user:

```
CREATE DATABASE neatdb;
CREATE USER 'admin'@'localhost' IDENTIFIED BY 'admin';
GRANT ALL ON *.* TO 'admin'@'localhost';
exit;
```

Deletion

If restarting the database from scratch:

```
DROP DATABASE neatdb;
exit;
```

Usage

Python Virtual Machine (Unix)

1. `source venv/bin/activate` - Load the environment, run scripts and develop under environment
2. `deactivate` - Unload the environment when you're done with python

Python Virtual Machine (Windows)

1. `venv/Scripts/activate` - Load the environment, run scripts and develop under environment
2. `deactivate` - Unload the environment when you're done with python

Django Quickstart

- `python manage.py createsuperuser` - add yourself to the database as an admin so that you can login to the REST API
- `python manage.py migrate` - apply model changes
- `python manage.py runserver` - run test server default is http://localhost:8000

Django Useful Commands

- `python manage.py runserver 0.0.0.0:8000` - run test server accessible remotely
- `rm -r restAPI/migrations` - delete database (follow MYSQL procedures as well)
- `python manage.py makemigrations restAPI` - start new database

Optional- Running with docker

This project can also be ran using docker. Docker is a container system meant to run an application with the same environment it was built in. This ensures dependencies remain the same on every system. To get started, install the docker toolbox with your favorite package manager. vist docker's website for installation instructions.

To run this using docker, make sure that your docker machine is running if you are on Mac/Windows. To create a virtualbox using `docker-machine` and then load it use the following commands:

1. `docker-machine create --driver virtualbox default`
2. `docker-machine start default`
3. `eval $(docker-machine env default)`
4. `docker-machine ls` to get the ip address of your docker-machine

Once the previous commands are done you should have a working VM running and loaded. Use `docker ps` to get a list of docker containers (should be empty at this time).

Finally, to start the app: `docker-compose up -d --build`

Possible Database error

After the previous step, let the app build in the container and start. Once the app has started, if you get a cannot connect error "Host is not allowed", then the database did not get created correctly(mysql issue). To fix this, I've included a script inside the container itself. run the command `docker exec -it neatbackend_db_1 bash -l` to enter the database container(this must be done while the container is running). From that point execute the script with the following command `./docker-entrypoint-initdb.d/script.sh` and it should create all the files you need. Stop the running instance docker-compose and return the command `docker-compose up -d --build`.

This wills start the application with all dependencies installed, as well as a small webserver to serve the static files. You can then reach the running application by visiting the ip address of your docker-machine

Frontend Installation

Neat Mobile App built with React Native

Installation

- Install React-Native

Install NeatMobileApp

- Clone Neat-Ver-1.0: `git clone`

  ```
  https://github.com/FIU-SCIS-Senior-Projects/Neat-Ver-1.0.git
  ```

- install dependencies

  ```
  cd Neat-Ver-1.0\NeatMobileApp\

  npm install
  ```

Usage

- For iOS, from the command line, run via command: `react-native run-ios` or open XCode and load project, Run `Product -> Run (⌘+R)`
- For android, from the command line, run via the command: `react-native run-android` assuming you have an emulator or device running and attached
- To run Jest, `npm test` not implemented yet
- To debug Jest unit cases, install node_inspector and run `npm run test-chrome` not implemented yet

Shortcomings/Wishlist Document

One of the items that we wished to complete was the chatroom. The UI was completed, but the chatbot that will be replying and helping students is based upon another project. Once the other project is complete then the chatroom would hit the endpoints of the API specified by the chatbot project.

The collaboration page is another aspect of the project that we wanted to complete to make it into the first version of Neat. The backend of the collaboration is finalised, but the frontend still needs to be integrated with the current system.

The website in future versions is expected to be fleshed out with a Teacher Dashboard. This dashboard will enable a teacher to organize her classes and create assignments which can then be shared to all students. The teacher will also be able to keep track of what tasks are causing the biggest bottleneck and tailor class activities around that information.

Sprint Retrospectives

Date: September 9, 2016
Attendees: Gabriel, Fernando, Justin, Giselle, Pachev, Pierre, Nelson
Start time: 1:45 PM
End time: 2:00 PM

What went wrong?

- Did we do a good job estimating our team's velocity?
    - Overestimated due to problems dealing with setting up environments
- Did we do a good job estimating the points (time required) for each user story?
    - No, we overestimated them.
- Did each team member work as scheduled?
    - Yes, but had hurdles.

What went right?

- Everyone had a part that they were interested in.

How to address the issues in the next sprint?

- How to improve the process?
    - Give more time to setup
- How to improve the product?
    - Learn the development software as much as possible

Date: September 26, 2016
Attendees: Gabriel, Fernando, Justin, Giselle, Pachev, Pierre, Nelson
Start time: 1:45 PM
End time: 1:55 PM

What went wrong?
- Did we do a good job estimating our team's velocity?
    - Underestimated, did not give out enough points; we worked more than the amount of points allotted due to the time taken in learning the technology.
- Did we do a good job estimating the points (time required) for each user story?
    - No, we underestimated them.
- Did each team member work as scheduled?
    - It was mostly good, but some issues on the login.

What went right?
- Fast set up of back end and testing, getting a code base for the front end

How to address the issues in the next sprint?
- How to improve the process?
    - Stick to user stories and dive the work well, Communicate the goals better, visually and functionally
- How to improve the product?
    - Standardize development conventions, document everything and comment

Date: October 10, 2016
Attendees: Gabriel, Fernando, Justin, Giselle, Pachev, Pierre
Start time: 1:45 PM
End time: 1:55pm

What went wrong?
- Did we do a good job estimating our team's velocity?

    - Better than last time, smaller detail the last task for testing are still needed to complete.
- Did we do a good job estimating the points (time required) for each user story?
    - We still need to get better assigning task points.
- Did each team member work as scheduled?
    - It was mostly good. It happened again with the web app.

What went right?
- Coding features.

How to address the issues in the next sprint?
- How to improve the process?
    - Testing before sprint review and document and that user stories do not confict with others.
- How to improve the product?
    - Standardize development conventions, document everything and comment. Code for one week, testing and documentation other week. Get better with mingle and create task for everything.

Date: October 25, 2016
Attendees: Gabriel, Fernando, Justin, Giselle, Pachev, Pierre
Start time: 2:00 PM
End time: 2:10pm

What went wrong?
- Did we do a good job estimating our team's velocity?
    - Yes, most work done in a single sprint, demo and website mostly complete.
- Did we do a good job estimating the points (time required) for each user story?
    - Yes, for the new stories. Past stories had to be completed this sprint though.
- Did each team member work as scheduled?
    - Yeah no story conflicts

What went right?
- Good demo, finished a lot of user stories, no scheduling conflict, website looks really good

How to address the issues in the next sprint?

- How to improve the process?
    - Talk more about what endpoints we need.
- How to improve the product?
    - Just keep working on stories, need a design on the front end before coding.

Date: November 7, 2016
Attendees: Gabriel, Fernando, Justin, Giselle, Pachev, Pierre
Start time: 1:30 PM
End time:

What went wrong?

- Did we do a good job estimating our team's velocity?
    - It was a bit slow, due to confusion
- Did we do a good job estimating the points (time required) for each user story?
    - No, refactoring is taking longer than expected as well as the teacher dash
- Did each team member work as scheduled?
    - No due to unclear stories

What went right?

- Had a meeting to clarify UI and what to work on
- Database is done
- Didn’t have to work as much

How to address the issues in the next sprint?

- How to improve the process?
    - Have clear stories to work on
- How to improve the product?
    - Know exactly what is wanted and needed

References

Christie, T. (n.d.). Quickstart. Retrieved November 30, 2016, from http://www.django-rest-framework.org/tutorial/quickstart/

Tutorial. (n.d.). Retrieved November 30, 2016, from https://facebook.github.io/react-native/docs/tutorial.html

Documentation. (n.d.). Retrieved November 30, 2016, from https://docs.djangoproject.com/en/1.10/