

**A Web-based Project**

**Management System**

**Submitted by: Joelle Kok Jia Xin**

**Matriculation Number: U1622741D**

**Supervisor: Chua Hock Chuan**

School of Electrical & Electronic Engineering

A final year project report presented to the Nanyang Technological University

in partial fulfilment of the requirements of the degree of

Bachelor of Engineering

**2020**

Table of Contents

[Abstract i](#_Toc39519405)

[Acknowledgements ii](#_Toc39519406)

[Acronyms iii](#_Toc39519407)

[List of Figures iv](#_Toc39519408)

[List of Tables vi](#_Toc39519409)

[Chapter 1 Introduction 1](#_Toc39519410)

[1.1 Background 1](#_Toc39519411)

[1.2 Motivation 1](#_Toc39519412)

[1.3 Objective & Scope 3](#_Toc39519413)

[1.4 Accomplishment 3](#_Toc39519414)

[1.5 Organization of Thesis 4](#_Toc39519415)

[Chapter 2 Review 5](#_Toc39519416)

[2.1 Project Overview 5](#_Toc39519417)

[2.1.1 Database 5](#_Toc39519418)

[2.1.2 Django Framework 6](#_Toc39519419)

[2.1.3 ReactJS Library 6](#_Toc39519420)

[2.2 Google Calendar 6](#_Toc39519421)

[Chapter 3 System Overview 8](#_Toc39519422)

[3.1 System Requirements 8](#_Toc39519423)

[3.1.1 Hardware 8](#_Toc39519424)

[3.1.2 Software 8](#_Toc39519425)

[3.2 Functional Requirements 10](#_Toc39519426)

[3.2.1 Website 10](#_Toc39519427)

[3.2.2 Admin Page 10](#_Toc39519428)

[3.3 Non-Functional Requirements 11](#_Toc39519429)

[3.3.1 Security 11](#_Toc39519430)

[3.3.2 Usability 11](#_Toc39519431)

[3.3.3 Portability 11](#_Toc39519432)

[3.4 Data Requirements 12](#_Toc39519433)

[3.4.1 Auth user 12](#_Toc39519434)

[3.4.2 Project 13](#_Toc39519435)

[3.4.3 Task 13](#_Toc39519436)

[3.4.4 Comment 14](#_Toc39519437)

[3.4.5 Task Attached Document 15](#_Toc39519438)

[3.4.6 Semester Start Date 16](#_Toc39519439)

[Chapter 4 Design 17](#_Toc39519440)

[4.1 Database Design 17](#_Toc39519441)

[4.1.1 Schema Tables 17](#_Toc39519442)

[4.1.2 Entity-Relationship Diagram 19](#_Toc39519443)

[4.2 Web Interface Design 22](#_Toc39519444)

[4.2.1 Login Page 22](#_Toc39519445)

[4.2.2 Calendar Page 22](#_Toc39519446)

[4.2.3 Tasks List Page 22](#_Toc39519447)

[Chapter 5 Implementation 23](#_Toc39519448)

[5.1 MySQL Workbench 23](#_Toc39519449)

[5.2 Django 23](#_Toc39519450)

[5.2.1 settings.py 23](#_Toc39519451)

[5.2.2 models.py 25](#_Toc39519452)

[5.2.3 serializer.py 25](#_Toc39519453)

[5.2.4 view.py 27](#_Toc39519454)

[5.2.5 urls.py 28](#_Toc39519455)

[5.3 ReactJS 29](#_Toc39519456)

[5.3.1 Material UI 29](#_Toc39519457)

[5.3.2 Calendar Store 29](#_Toc39519458)

[5.3.3 Login Store 29](#_Toc39519459)

[Chapter 6 Results and Discussion 31](#_Toc39519460)

[6.1 Webpage Views 31](#_Toc39519461)

[6.1.1 Login Page 31](#_Toc39519462)

[6.1.2 Calendar Page 32](#_Toc39519463)

[6.1.3 Project Listing 37](#_Toc39519464)

[6.1.4 Task List Page 37](#_Toc39519465)

[6.2 Admin Page 40](#_Toc39519466)

[6.2.1 Django administration login 40](#_Toc39519467)

[6.2.2 Django administration Tasks 42](#_Toc39519468)

[6.2.3 Django administration Projects 45](#_Toc39519469)

[6.2.4 Django administration Semesters 48](#_Toc39519470)

[6.2.5 Django administration Users 50](#_Toc39519471)

[6.3 Use-Case Diagram 53](#_Toc39519472)

[Chapter 7 Conclusion and Recommendation 55](#_Toc39519473)

[7.1 Conclusion 55](#_Toc39519474)

[7.1.1 Overview 55](#_Toc39519475)

[7.1.2 Tools used 55](#_Toc39519476)

[7.2 Recommendation for Future Works 56](#_Toc39519477)

[7.2.1 Password Change 56](#_Toc39519478)

[7.2.2 Auto Change Status of Task 56](#_Toc39519479)

[References 57](#_Toc39519480)

[Appendix A 59](#_Toc39519481)

# ****Abstract****

There are currently no good and customized tools to manage projects such as the Design and Innovation Project (DIP) and Final Year Project (FYP). A DIP and FYP management tool would require features such as a customized calendar suited to the weeks of the academic calendar while also being able to allocate tasks which students can upload their response to.

With these requirements in mind, the objective of this project is to create a customized web application to facilitate the tracking of DIP and FYP progression. In this customized web application, there should be a calendar reflective of the academic calendar, and users should be able to appoint new tasks. These tasks can then be completed by the appointed students.

To accomplish this, we have developed a web-based application using Django for backend and ReactJS for frontend. This application allows users to schedule tasks and meetings on an online web-based platform. Students can now submit their tasks through the website and their professor will be able to view and comment on the said tasks in real-time. The website created in this project is customized to the university academic calendar and the needs of a project management system of the university.

# ****Acknowledgements****

Firstly, the author would like to thank Nanyang Technological University for giving this opportunity to work on the project. Secondly, during this period, family and friends have been an important pillar of support for the author. For that, the author would like to thank her family and friends for their constant support.

The author would also like to thank her partner Celia Ong, without her, the project would not have been possible. Most importantly the author would like to express her deepest gratitude to her supervisor, Associate Professor Chua Hock Chuan for his guidance and patience during the term of her project.

April 2020

Joelle Kok Jia Xin

# ****Acronyms****

|  |  |
| --- | --- |
| PK | Primary Key |
| FK | Foreign Key |
| CORS | Cross-Origin Resource Sharing |
| URL | Uniform Resource Locator |
| FYP | Final Year Project |
| DIP | Design and Innovation Project |
| CRUD | Create, Read, Update, Delete |

# List of Figures

[Figure 4.1 : Django Default Tables 19](#_Toc38771389)

[Figure 4.2 : Django Tables 20](#_Toc38771390)

[Figure 4.3 : Designed Database 21](#_Toc38771391)

[Figure 6.1 : Login Page web view 31](#_Toc38771392)

[Figure 6.2 : Login Page web view on failed login 32](#_Toc38771393)

[Figure 6.3 : Calendar Page for Professor 32](#_Toc38771394)

[Figure 6.4 : Calendar Page for Student 33](#_Toc38771395)

[Figure 6.5 : Pop up for adding new Task 34](#_Toc38771396)

[Figure 6.6 : Pop up for clicking on Calendar Task 35](#_Toc38771397)

[Figure 6.7 : Menu button panel from Calendar for Professor 36](#_Toc38771398)

[Figure 6.8 : Menu button panel from Calendar for Student 36](#_Toc38771399)

[Figure 6.9 : Project Listing Page 37](#_Toc38771400)

[Figure 6.10 : Weekly Report Page opened dropdown 37](#_Toc38771401)

[Figure 6.11 : Weekly Report Page closed dropdown 38](#_Toc38771402)

[Figure 6.12 : Meetings List Page 38](#_Toc38771403)

[Figure 6.13 : Strategy Plan Page 39](#_Toc38771404)

[Figure 6.14 : Django Admin Login Page 40](#_Toc38771405)

[Figure 6.15 : Django Admin Page 41](#_Toc38771406)

[Figure 6.16 : Django Admin Tasks Page 42](#_Toc38771407)

[Figure 6.17 : Django Admin add new Task Page 43](#_Toc38771408)

[Figure 6.18 : Django Admin edit Task Page 44](#_Toc38771409)

[Figure 6.19 : Django Admin Projects Page 45](#_Toc38771410)

[Figure 6.20 : Django Admin add new Project Page 46](#_Toc38771411)

[Figure 6.21 : Django Admin edit Project Page 47](#_Toc38771412)

[Figure 6.22 : Django Admin Semesters Page 48](#_Toc38771413)

[Figure 6.23 : Django Admin add new Semester Page 49](#_Toc38771414)

[Figure 6.24 : Django Admin edit Semester Page 49](#_Toc38771415)

[Figure 6.25 : Django Admin Users Page 50](#_Toc38771416)

[Figure 6.26 : Django Admin add new User Page 51](#_Toc38771417)

[Figure 6.27 : Django Admin edit User Page 52](#_Toc38771418)

[Figure 6.28 : Use-Case Diagram 54](#_Toc38771419)

# List of Tables

[Table 1 : User Requirement Table 12](#_Toc38771427)

[Table 2 : Project Requirement Table 13](#_Toc38771428)

[Table 3 : Task Requirement Table 14](#_Toc38771429)

[Table 4 : Comment Requirement Table 15](#_Toc38771430)

[Table 5 : Task Attached Document Requirement Table 15](#_Toc38771431)

[Table 6 : Semester Start Date Requirement Table 16](#_Toc38771432)

[Table 7 : Schema Tables 18](#_Toc38771433)

# Introduction

## Background

Technology has become an indispensable part of life. From transport to communication, it can be seen how technology has become integrated with the way people live. As such, it is hard to imagine life without technology in the technology-driven world of today. One major invention created is the Internet. As one of the great 10 communication inventions [1], it is estimated that 51.2% of the global population were using it at the end of 2018 [2]. With the prevalence of the internet, many processes that were once only done offline have moved to online platforms. For example, the invention of internet banking, a system that allows customers to conduct transactions from their bank account online. Instead of spending time to visit a branch or an ATM, customers can now complete most transactions online. This has made banking more convenient and easier for customers [3].  Keeping this in mind, the internet has many uses and can also be use to make FYP and DIP management easier.

## Motivation

The internet has made many processes more convenient, one such area that has benefitted from it is the way data can be stored and accessed from. Data can now be stored and retrieved from a database which can be made available online. Although databases have been around since the 1960s, only in the mid-1990s with the advent of the Internet that the use of databases grow exponentially [4]. Traditionally, information is stored on paper and must be physically filed and organized. However, with the invention of databases, information can be stored in a computer. There are benefits in storing information in computer instead. Firstly, by eliminating the need for physical document, the usage of paper can be reduced, thus reducing the chance of a misplaced document or a damaged document. Secondly, a well-designed database allows data to be stored and retrieved easily. Information stored in a database is searchable through the usage of keywords, making searching for data faster and simpler. An example would be the time taken to “google” something versus the time needed to flip through a book or dictionary. It is easier to sieve data out of a database as compared to sieving it out of physical documents. Therefore, to make managing data more convenient, it is logical to move offline processes to online where possible.

One such process that has moved online, is the way people manage their schedules. For example, applications like outlook calendar and google calendar allows users to manage tasks and events. The task created would be stored in a database where it could be retrievable on an online platform. This shift to an online platform makes it convenient for users to keep track of their schedule. However, although these calendar systems exist to help manage tasks, they are not customized to the university academic calendar and the needs of a project management system of a university.

To elaborate, the university academic calendar runs on a 13 weeks schedule and what is needed is a customize calendar to publish task and allow students to upload their responses. Current calendar like outlook and google calendar only allows events to be added to the calendar and there is no such function like allowing participants to upload documents etc.

## Objective & Scope

The objective of this project is to create a web application to facilitate the tracking of Design and Innovation Project (DIP) and Final Year Project (FYP) progression, moving this process from a mixture of offline and online procedure to a full online platform. This would be done using Django for backend and ReactJS for frontend.

As this project is split into backend and frontend, the author’s focus is on the backend of the project, thus this report will focus more on the backend and admin website of the project.

## Accomplishment

A web-based application was created using Django for backend and ReactJS for frontend. This application allows users to schedule tasks and meetings. Students can submit their tasks through the website and upload meeting notes, and their professor can comment on the said tasks in response. The resulted website is customized to the university academic calendar and the needs of a project management system of a university.

## Organization of Thesis

The chapter will be organized as such:

Chapter 1 – Introduction: introducing the background, motivation, object, scope and accomplishment of the project.

Chapter 2 – Review: review of google calendar and the various system used; MySQL, Django and ReactJS

Chapter 3 – System Overview: describes the requirements of the project; functional, non-functional and data requirements

Chapter 4 – Design: elaborates on the database design and web interface design of the project

Chapter 5 – Implementation: an overview of the key components used in the implementation of the project

Chapter 6 – Results and Discussion: illustrations and elaboration of the web-based application and administration application

Chapter 7 – Conclusion and Recommendation: includes suggestion on future work on the project

# Review

This chapter will review on the different components to be used in the development of the application and web-based management system with a similar concept to this project.

## Project Overview

The aim of this project is to create a web-based application for project management by using Django framework and ReactJS. By definition, a web-based application is a program that is accessed over a network connection using HTTP, often running inside a web browser [5]. This project involves full-stack development. A full-stack developer is someone who understands every layer of the full-stack [6]. Thus, to complete this project, an understanding of every layer of the full stack is needed. The key components to understand are the database, backend, and the frontend to be used in the development of the application. This subchapter will be touching upon the key components used for the project.

### Database

A database is defined as an organized set of data that is stored in a computer and can be looked at and used in various ways [7]. There are many kinds of databases, but for this project, a relational database will be used. A relational database being a database that recognizes relationships between different pieces of information [8]. The database will hold the necessary data needed for the web-based application, for example, user details and tasks details.

For the database, MySQL will be used, and the data will be managed through MySQL Workbench application. MySQL is a popular database management system that uses relational database [9], it is designed to process millions of queries while ensuring unique memory caches, full-text indexes, and optimum speed [10].

### Django Framework

The definition of a framework in programming refers to a platform for developing software applications, providing a foundation on which developers can build programs for a specific platform [11]. Frameworks represent the collective investment in a software infrastructure, helping to cut time wasted in re-implementing standard algorithms and data structures [12]. Django provides a high-level framework that enables developers to build web applications with relatively few lines of code. It is simple, robust, and flexible, allowing you to design solutions without much overhead [13].

### ReactJS Library

ReactJS is a JavaScript library which is deployed to develop reusable user interface (UI) components [14]. Coupled with the help of website like Material UI which provides reusable components used by organisations like Netflix and Amazon [15], it would ease the process of creating the outlook of the website.

## Google Calendar

Google Calendar is a free online calendar for personal use or workplace use, it can be used to schedule events and alert users of upcoming events [16]. For the project, it is similar to google calendar in terms of its functionality of scheduling events. The web application created should be able to schedule meetings and task deadlines for the students. These are accessible by both the student and the professor involved. However, the difference is that Google Calendar does not allow users to upload documents or submissions to the event, it is purely a reminder system for the event happening at the targeted timeslot. Nonetheless, the Calendar overview of the events scheduled is something that can be studied for the making of the web application. For the web application, it will be desirable to create a Calendar view to view all the events and allowing users to add new events by clicking on the dates.

# System Overview

This chapter will elaborate on the requirements of the web-based application, expanding on the system, functional, non-functional and database requirements.

## System Requirements

### Hardware

The following device were used in the development of this application:

* Personal Laptop running on Windows 10 Home (Version 1903) (OS build 18362.778)
* Personal Laptop running on macOS Cataline

### Software

The following are tools that were used in the development of this application:

* MySQL Workbench 8.0 (Version 8.0.17)
* Visual Studio Code (Version 1.44.1)
* Google Chrome (Version 80.0.3987.163)
* Application Requirements
  + certifi==2019.11.28
  + chardet==3.0.4
  + defusedxml==0.6.0
  + Django==2.2.6
  + django-allauth==0.41.0
  + django-cors-headers==3.1.1
  + django-rest-auth==0.9.5
  + djangorestframework==3.10.3
  + idna==2.8
  + mysqlclient==1.4.4
  + oauthlib==3.1.0
  + Pillow==7.0.0
  + psycopg2-binary==2.8.4
  + python3-openid==3.1.0
  + pytz==2019.3
  + requests==2.22.0
  + requests-oauthlib==1.3.0
  + six==1.13.0
  + sqlparse==0.3.0
  + urllib3==1.25.7

MySQL Workbench was used for the database, Visual Studio Code was used for editing the codes for the project, Google Chrome was used to view the frontend of the project and the application requirements are the list of application to install for the application to work. Application requirements are installed using the command prompt, this list is in the requirements.txt in the env folder and is installed using pip, a package installer for Python.

## Functional Requirements

### Website

The website should be able to be used by two key users, professors and students. On the website, it should contain a calendar week view where tasks will be displayed and clicking on the tasks should bring the user to a list of tasks for that respective user. The users should be able to add new tasks and comment on submitted tasks. The students should be able to complete tasks by submitting a text input or document input.

### Admin Page

The admin page should be available to the professor where the key process they should be able to complete is to add new users or add new semester start dates.

## Non-Functional Requirements

### Security

The few security feature is to ensure that only relevant personnel can access the administrative page and that data are filtered according to the users. There should not be a case where unauthorized users are able to view another user’s details.

### Usability

For usability, the website should be instinctive, and user should be able to understand how to use the website quickly.

### Portability

The website should be able to be viewed through common web platform; google chrome, safari.

## Data Requirements

The stored information in the database will be in the following formats. This subchapter will illustrate the different database tables and description of each table column.

### Auth user

|  |  |
| --- | --- |
| Field | Requirements |
| id | - Must be Unique  - Integer |
| password | - Use PBKDF2 algorithm with a SHA256 hash  - Alphanumerical |
| username | - Username is used for logging into the account  - Alphanumerical |
| first\_Name | - Alphanumerical |
| last\_Name | - Alphanumerical |
| is\_Staff | - Used to check if account is a professor or student account  - Boolean |
| project\_Id | - Linked to the corresponding project\_Id in the Project table that the student is involved in  - Must be Integer |

Table : User Requirement Table

Table 1 is used to store the user information of each account that is used for login. The username and password would be used for login into the website. As we would like to have two type of users for this project, is\_Staff is used to differentiate the two different kind of accounts. project\_Id is used to specify the project of student users, if it is a professor account, this field can be left blank.

### Project

|  |  |
| --- | --- |
| Field | Datatype |
| project\_Id | - Must be Unique  - Integer |
| tutor\_Id | - Linked to the id in the User table of the tutor in charge of the project  - Integer |
| project\_Name | - Name of the Project  - Alphanumerical |
| project\_Description | - Description of the Project  - Alphanumerical |
| is\_FYP\_Project | - Used to check if the project is a FYP or DIP type  - Boolean |

Table : Project Requirement Table

Table 2 stores the list of FYP and DIP projects in the system. The project will be link to the tutor’s user id.

### Task

|  |  |
| --- | --- |
| Field | Requirements |
| task\_Id | - Must be Unique  - Integer |
| project\_Id | - Linked to the project\_Id in the Project table  - Integer |
| student\_Id | - Linked to the id in the User table  - Integer |
| tutor\_Id | - Linked to the id in the User table  - Integer |
| task\_type | - Type of task  - Enumeration |
| desc | - Description of task  - Alphanumerical |
| task\_Created\_Date | - Date the task was created  - Date time format |
| task\_Due\_Date | - Due date of the task  - Date time format |
| submission\_Date | - Submission deadline for the task  - Date time format |
| content | - Submission content  - Alphanumerical |
| hours\_Spent | - Hours spent on the corresponding task by user  - Integer |
| status | - Current status of the task  - Enumeration |

Table : Task Requirement Table

Table 3 stores the list of tasks, these tasks can be of different kinds, for example, meetings or weekly report. The task table will contain the task details of the task.

### Comment

|  |  |
| --- | --- |
| Field | Requirements |
| comment\_Id | - Must be Unique  - Integer |
| task\_Id | - Linked to the task\_Id from Task table which the comment is posted to  - Integer |
| user\_Id | - id from User table of the user who posted the comment  - Integer |
| content | - The comment contents  - Alphanumerical |
| creation\_Date | - The date time of the comment  - Date time format |

Table : Comment Requirement Table

Table 4 stores the comments for the corresponding task in Table 3. These comments will contain the task\_Id to link to the task table.

### Task Attached Document

|  |  |
| --- | --- |
| Field | Requirements |
| task\_Attach\_Document\_Id | - Must be Unique  - Integer |
| task\_Id | - Linked to the task\_Id from Task table which the document is posted to  - Integer |
| attach\_Document | - The name of the attached document  - Alphanumerical |
| uploaded\_Date | - The date time of the attached document being posted  - Date time format |

Table : Task Attached Document Requirement Table

Table 5 holds the document information of the document uploaded for the tasks.

### Semester Start Date

|  |  |
| --- | --- |
| Field | Requirements |
| semester\_Id | - Must be Unique  - Integer |
| semester | - Semester 1 or Semester 2  - Enumeration |
| start\_Date | - The first day of the corresponding semester  - Date time format |

Table : Semester Start Date Requirement Table

Table 6 stores the semester start dates for the university. This will be reflected on the calendar view of the website.

# Design

This chapter elaborate on the design of database and the web-view for the web-based application. The database design will be elaborating on the various tables used and the web-view elaborating the view design.

## Database Design

The database contains both Django tables (tables which were generated through the Django framework and is not designed by the author) and tables designed by the author.

### Schema Tables

|  |  |
| --- | --- |
| Table Name | Django Table? (Y/N) |
| account\_emailaddress | Y |
| account\_emailconfirmation | Y |
| auth\_group | Y |
| auth\_group\_permission | Y |
| auth\_permissions | Y |
| auth\_user | Y (But edited to suit the project) |
| auth\_user\_groups | Y |
| auth\_user\_user\_permissions | Y |
| authtoken\_token | Y |
| comment | N |
| django\_admin\_log | Y |
| django\_content\_type | Y |
| django\_migrations | Y |
| django\_session | Y |
| django\_site | Y |
| project | N |
| semester\_start\_date | N |
| socialaccount\_socialaccount | Y |
| socialaccount\_socialapp | Y |
| socialaccount\_socialapp\_site | Y |
| socialaccount\_socialtoken | Y |
| task | N |
| task\_attach\_document | N |
| tasklist\_semester | N |

Table : Schema Tables

Table 7 shows the list of tables used for the database. The Django framework has its own base tables, the auth\_user table was edited while the other tables were kept the same. More information about the tables can be found from Annex A.

### Entity-Relationship Diagram

#### Entity-Relationship Diagram for Django tables

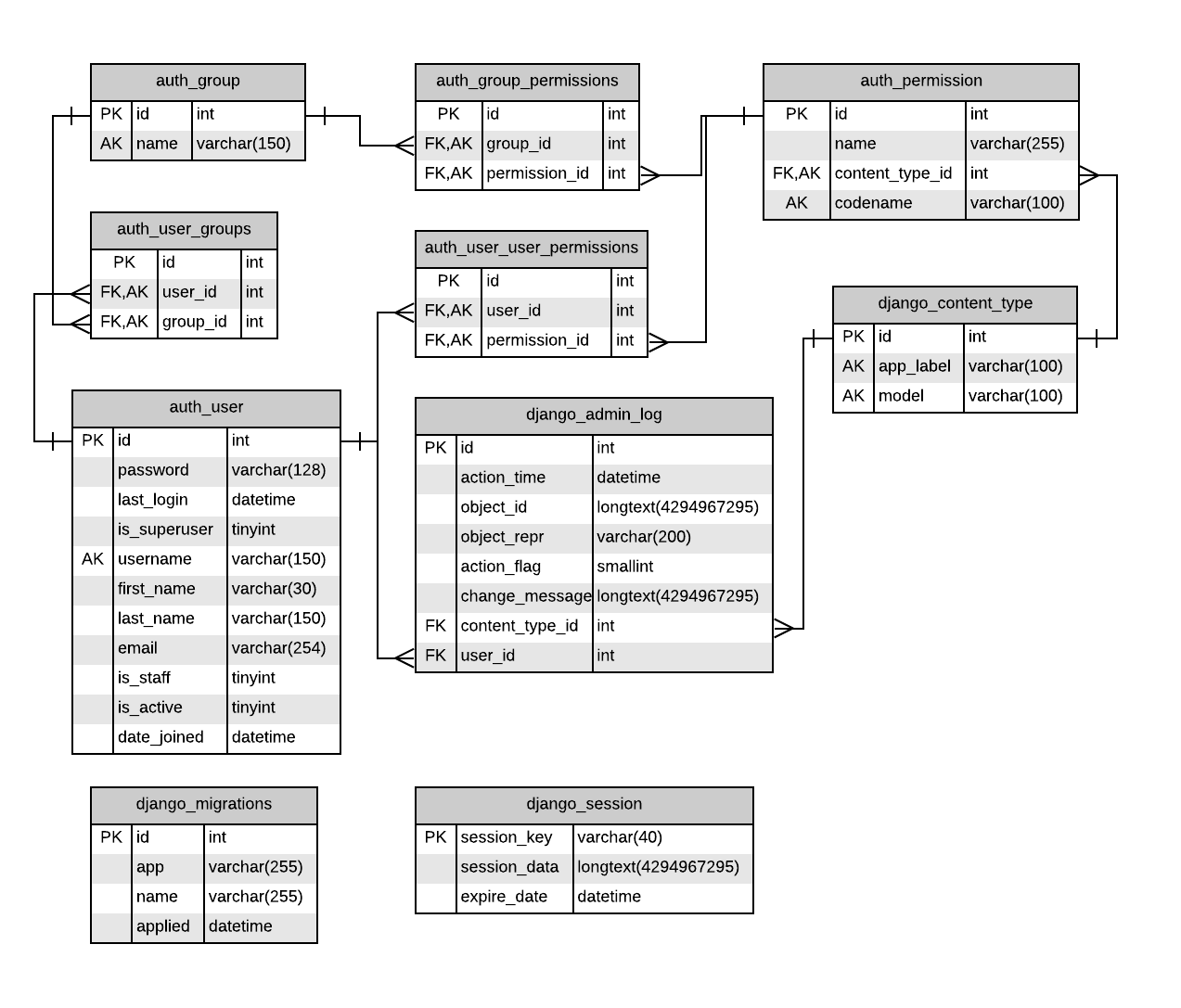


Figure . : Django Default Tables

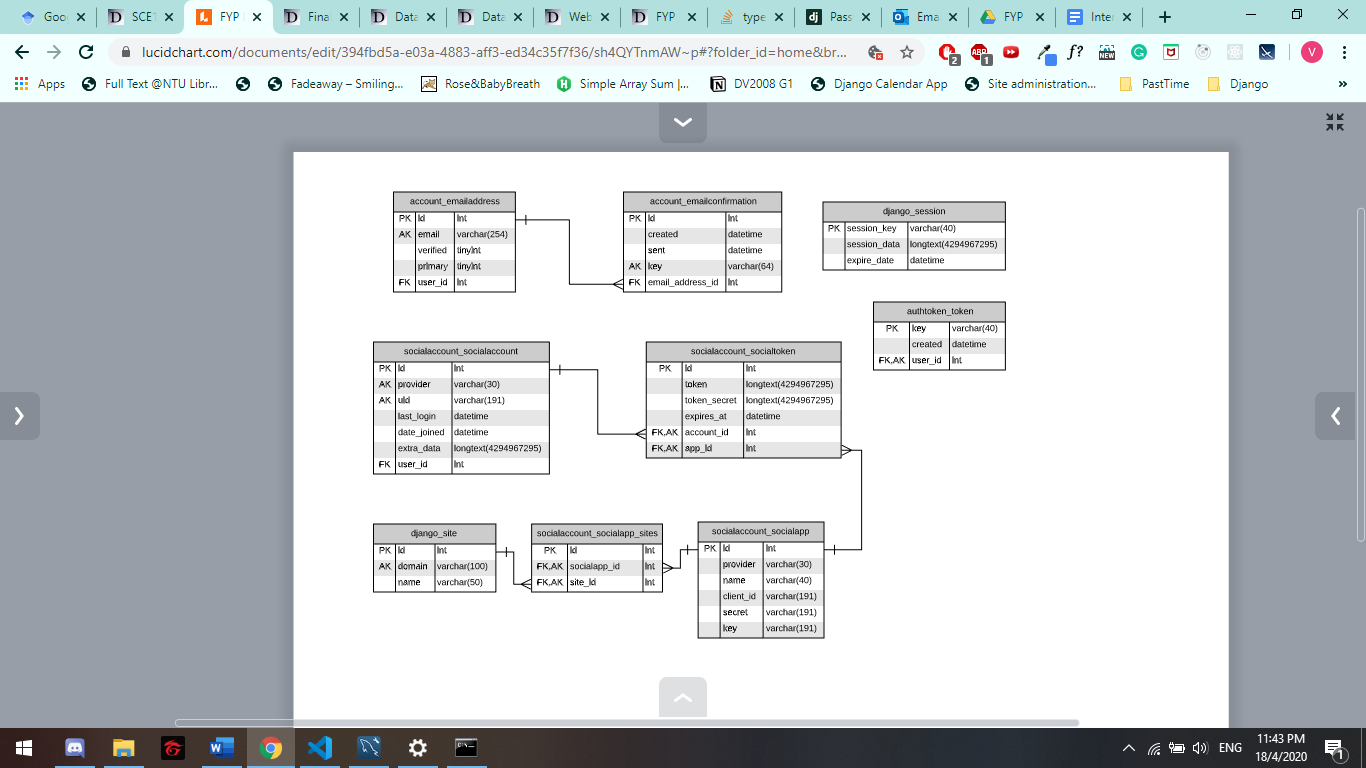


Figure . : Django Tables

Figure 4.1 and Figure 4.2 are tables generated from Django framework. The auth\_user table in Figure 4.1 is the unedited version, for the actual implementation of the project, that table is edited to suit the needs of the project.

#### Entity-Relationship Diagram for auth\_user and new tables

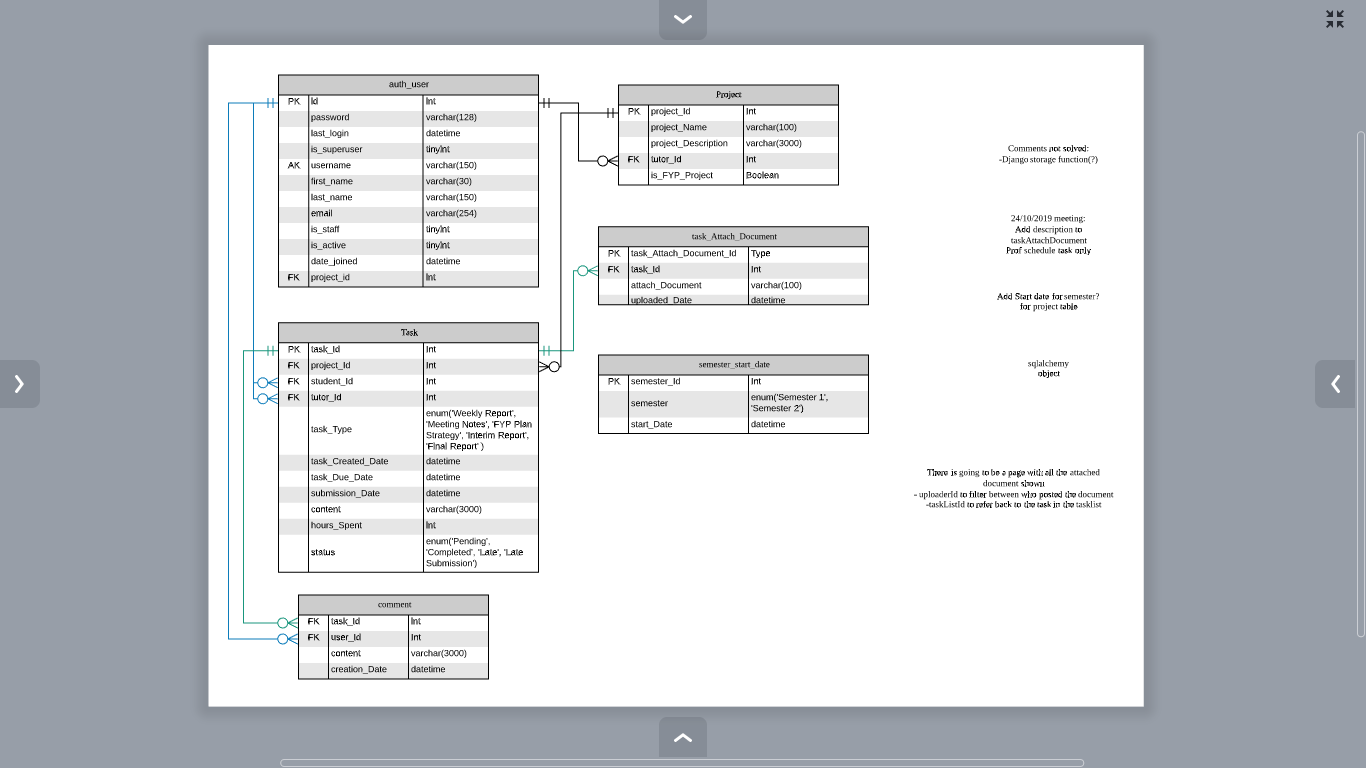


Figure . : Designed Database

Figure 4.3 illustrates the entity-relationship diagram for the tables proposed for the application. The auth\_user table in Figure 4.3 is edited from of Django default auth\_user table in Figure 4.1.

## Web Interface Design

### Login Page

The login page should be a page where user can log in to the website using their username and password.

### Calendar Page

The calendar page will be a calendar view of Monday to Friday with the various tasks and their deadlines annotated. There should be an add task button on the left side of the page, allowing users to add new tasks to the calendar.

### Tasks List Page

The tasks list page will be a list of all the tasks for the given student with task details. The details are as such:

* Week number
* Submission status
* Deadline
* Submitted date
* Number of hours
* Content
* Files
* Comments

There should be a tally of the number of hours clocked at the top of the page.

# Implementation

This chapter will elaborate on the tools used in the implementation of the web-based application. MySQL was used for the database, Django framework for backend processes and ReactJS library for the frontend view.

## MySQL Workbench

For the database, it was implemented with MySQL. Lucidchart (an online platform that can be used to map out entity-relationship diagrams) was used to designed and plan out the database, while MySQL Workbench was used to create and manage the database.

## Django

Django framework was used for implementing the backend services, namely it is used to retrieve data from the MySQL. This sub chapter will elaborate on the key files used for Django implementation.

### settings.py

This file is where settings for the Django framework is specified. The key settings were done under the:

* DATABASES,
* INSTALLED\_APPS,
* CORS\_ORIGIN\_ALLOW\_ALL,
* AUTH\_USER\_MODEL and
* REST\_AUTH

Under DATABSES, the database to connect to is specified.

Codes for the DATABSES settings:

|  |
| --- |
| DATABASES = {      'default': {          # 'ENGINE': 'django.db.backends.sqlite3',          # 'NAME': os.path.join(BASE\_DIR, 'db.sqlite3'),          'ENGINE': 'django.db.backends.mysql',          'NAME': 'fypdb',          'USER': 'fyp',          'PASSWORD': 'xxxx1!',          'HOST': 'localhost',          'PORT': '3306',      }  } |

In this project, we created a database called ‘fypdb’ with a user, ‘fyp’ and a password or ‘xxx1!’, the database is located at localhost port 3306. If the database is to be changed, the name of the database should be changed here.

Under INSTALLED\_APPS, a list of apps was specified. If these apps are not specified, we are not able to use them in the application. The applications specified are the authentication for login, rest framework for data view of the API and CORS header. CORS header is to allow connection between Django and ReactJS frontend. Cross-Origin Resource Sharing ([CORS](https://developer.mozilla.org/en-US/docs/Glossary/CORS)) is a mechanism that uses additional [HTTP](https://developer.mozilla.org/en-US/docs/Glossary/HTTP) headers to tell browsers to give a web application running at one [origin](https://developer.mozilla.org/en-US/docs/Glossary/origin), access to selected resources from a different origin [17].

CORS\_ORIGIN\_ALLOW\_ALL was set to true to allow connection between Django and ReactJS frontend.

At AUTH\_USER\_MODEL the base user class of Django was override with the edited user class specified for the project.

The default token serializer under REST\_AUTH\_SERIALIZERS was replaced with the edited token serializer which was edited to suit the project.

### models.py

A model is the single, definitive source of information about the data. It contains the essential fields and behaviors of the data that is stored [18].

The classes specified in the models.py reflects all the tables in the database, each table will have their corresponding model class. Some of the model classes are used for admin.py. These used models are reflected at the admin page. Changes in the data for these models can thus be done through the Django admin page. The registered tables are:

* Auth users
* Comments
* Projects
* Semesters
* Task Attach Documents
* Tasks

### serializer.py

Serializers allow complex data such as query sets and model instances to be converted to native Python datatypes that can then be easily rendered into JSON, XML or other content types. Serializes also provide deserialization allowing parsed data to be converted back into complex types, after first validating the incoming data [19].

Each serializer can only have one model class. Thus, to combined multiple tables into one serializer, the serializer are combined instead. For example, a serializer using the Comment model can be incorporated into another serializer using the Task model by changing one of the fields to be the Comment serializer.

A basic serializer:

|  |
| --- |
| class userNameSerializer(serializers.ModelSerializer):      class Meta:          model = AuthUser          fields = ('id', 'first\_name', 'last\_name') |

Example of combining serializers:

|  |
| --- |
| class userAndProjectsSerializer(serializers.ModelSerializer):      projects = projectSerializer(source='project\_set', read\_only=True, many=True)      class Meta:          model = AuthUser          fields = ('id', 'username', 'is\_staff', 'projects', 'project\_id') |

### view.py

The serializers are then used in the views.py classes. In the view.py classes, we can change the query set to allow filtering.

A basic view.py class:

|  |
| --- |
| class projectViewSet(viewsets.ModelViewSet):      queryset = Project.objects.all()      serializer\_class = projectSerializer |

A view.py class with filtering:

|  |
| --- |
| class commentViewSet(viewsets.ModelViewSet):      serializer\_class = commentSerializer      def get\_queryset(self):          queryset = Comment.objects.all()          task\_id = self.request.query\_params.get('task\_id', None)          if task\_id is not None:              queryset = queryset.filter(task\_id\_\_task\_id=task\_id)            user\_id = self.request.query\_params.get('user\_id', None)          if user\_id is not None:              queryset = queryset.filter(user\_id\_\_id=user\_id)            return queryset |

### urls.py

The urls.py will use the view sets to create the view for the data retrieved.

List of URLs at localhost:

* Tasks: <http://localhost:8000/api/task/>
* Comments: <http://localhost:8000/api/comment/>
* Projects: <http://localhost:8000/api/project/>
* Task with Comments: <http://localhost:8000/api/taskComment/>
* Semester Start: <http://localhost:8000/api/semesterStart/>
* Documents: <http://localhost:8000/api/document/>
* Users And Projects: <http://localhost:8000/api/usersAndProjects/>
* Login: <http://localhost:8000/rest-auth/login/>

## ReactJS

ReactJS was used for the frontend view of the web-application. This subchapter will highlight the modules used and the store used in the project. A store is used to hold the whole [state tree](https://redux.js.org/glossary#state) of your application [20].

### Material UI

Material UI is a website providing react components, the components used in developing the view of the application is provided by Material UI.

### Calendar Store

This store is used to store the list of tasks for the current user. When the tasks API is called, the returned data is stored in the Calendar Store.

### Login Store

This store is used to store the information returned from the API when the user logins successfully.

The props stored after login are:

* token: login token, the token returned after successfully login in
* error: return error if there is one
* loading: true or false, for changing the page to loading indicator if true
* user: user id
* projects: list of projects under the user, may be empty
* is\_Staff: 1 or 0 (1 for true, 0 for false)
* paramQuery: query line to filter project based on choice for login user (empty for students)

These props are also stored in the local storage, thus when the page reloads, the props can be retrieved from the local storage.

# Results and Discussion

## Webpage Views

This subchapter will show the various webpages and the function of the buttons for the website.

### Login Page

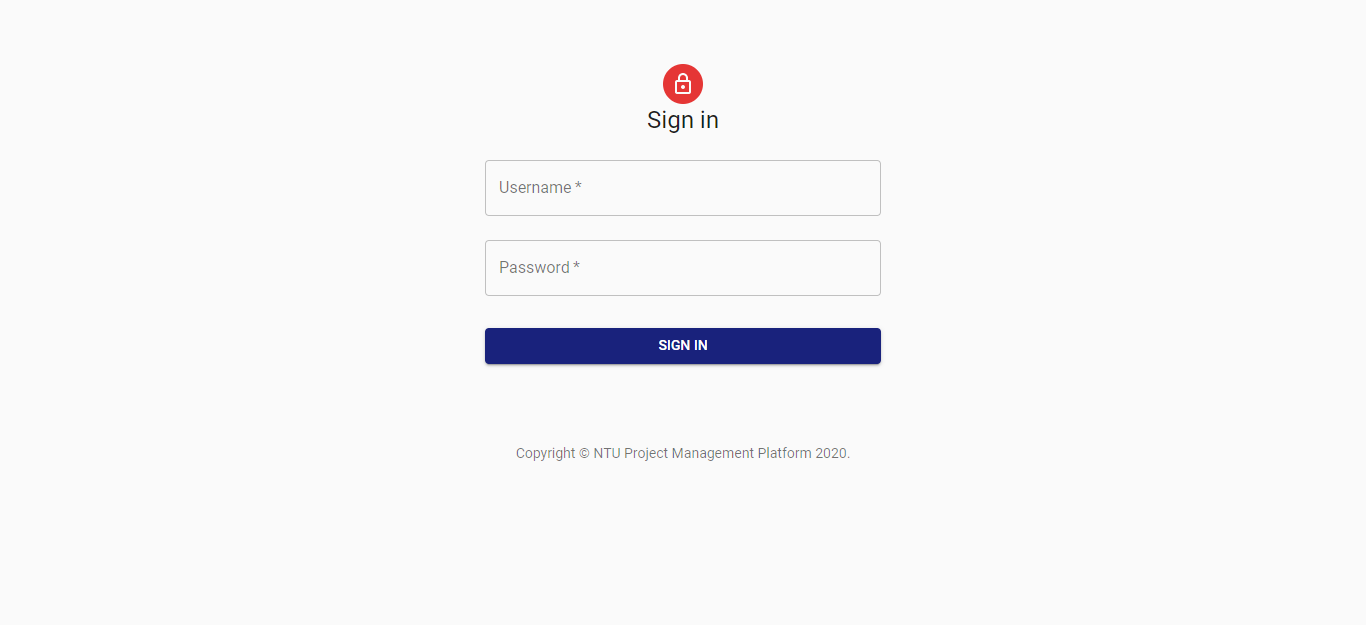


Figure . : Login Page web view

Figure 6.1, the login page is the first page the user comes to. This is a simple login page for the application, in this project, the user will login with their username and password. Clicking the blue sign in button without username or password inputted will results in a warning message, ‘Please fill in your username/password’.

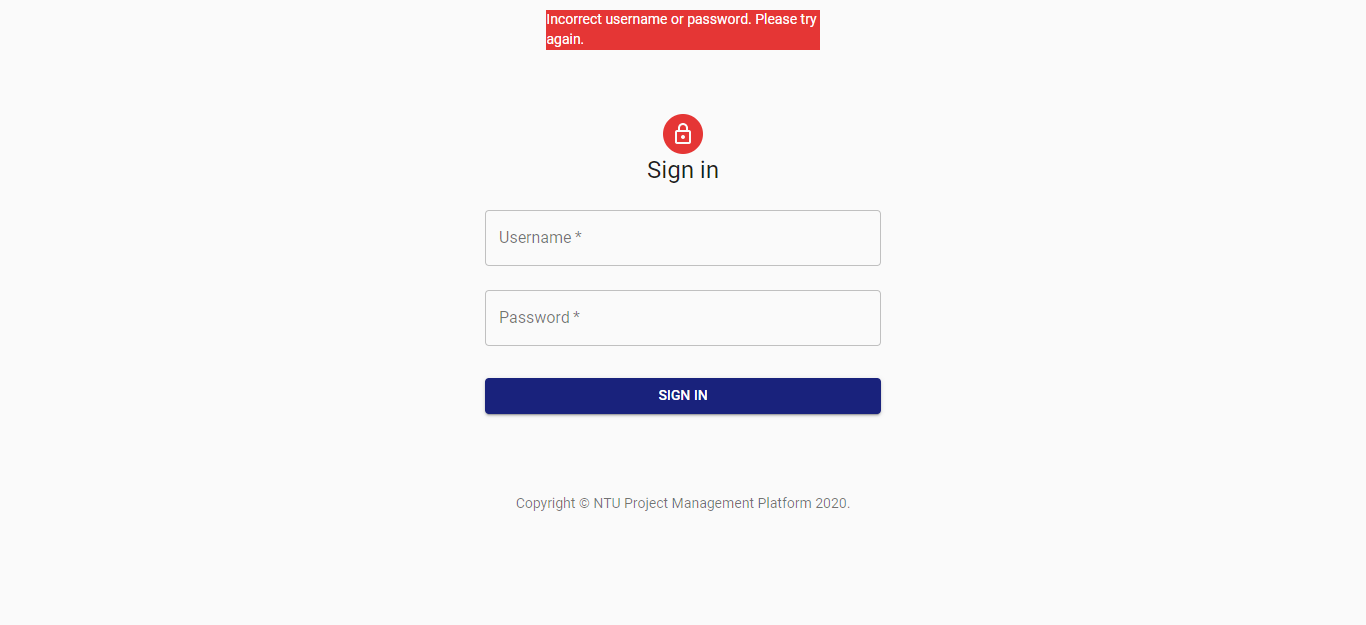


Figure . : Login Page web view on failed login

If username or password is wrong, the top of the page will display the error, ‘Incorrect username or password. Please try again.’ as shown in Figure 6.2.

### Calendar Page

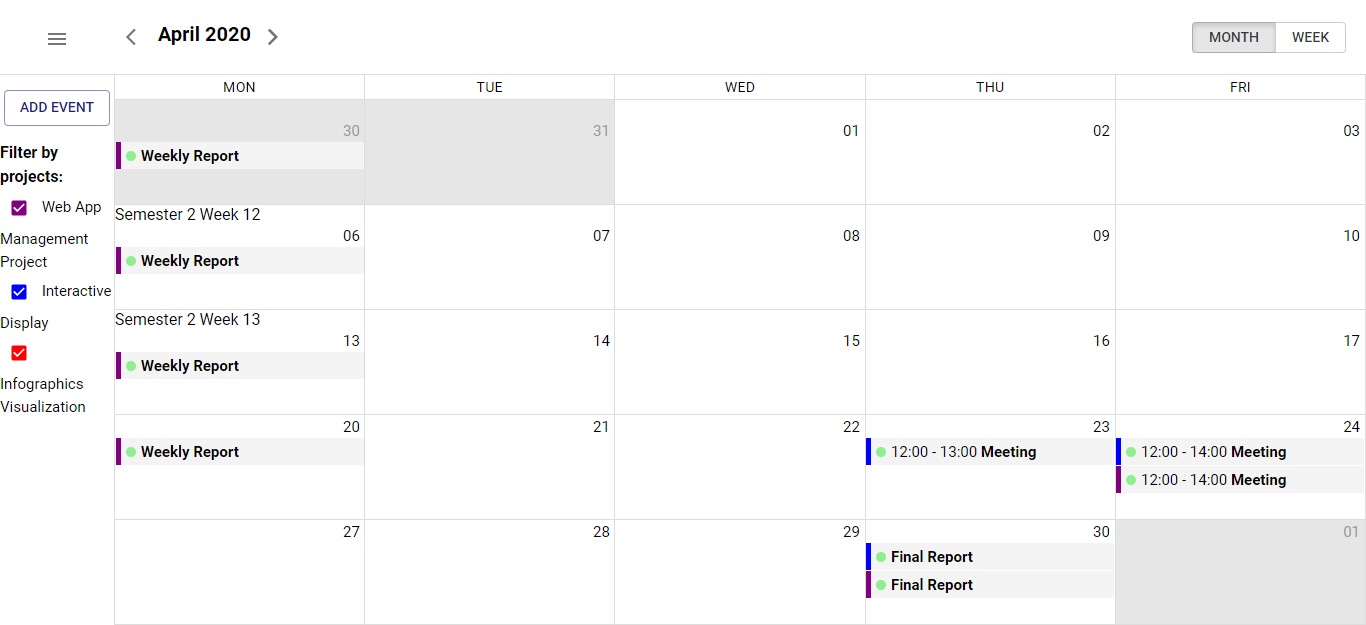


Figure . : Calendar Page for Professor

After successful login, users will be redirected to the calendar page, Figure 6.3 illustrates a professor’s account logged in. The left panel in Figure 6.3 consist of an add event button and a filter panel. The filter panel allows the professor to select which projects task to be reflected on the calendar. A student account will not have the filter options.

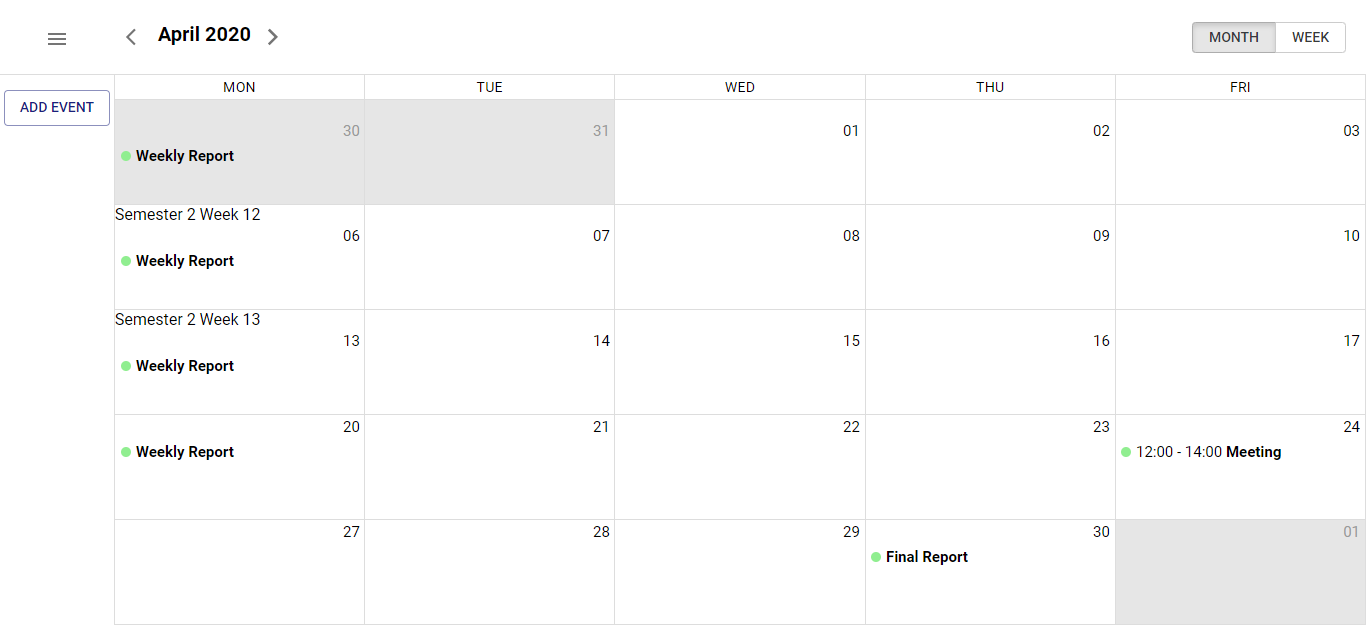


Figure . : Calendar Page for Student

Figure 6.4 illustrates a student’s account logged in. For the student’s account, there will be no filter options on the left side of the screen.

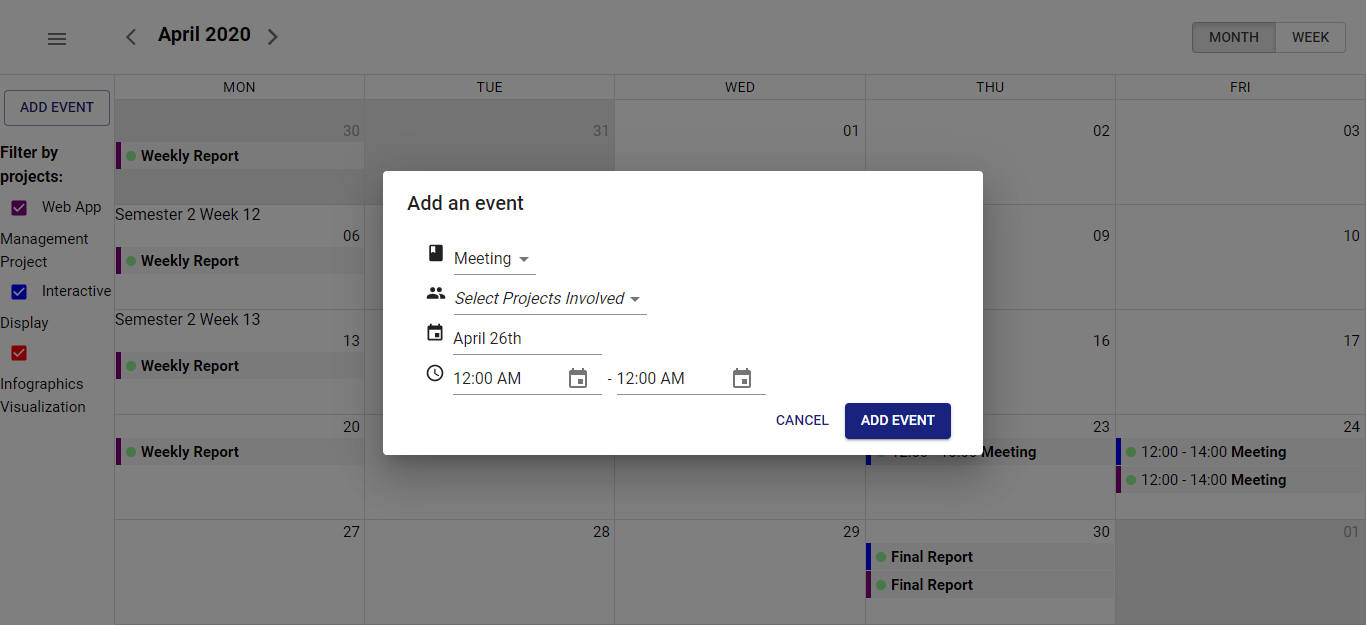


Figure . : Pop up for adding new Task

Clicking on the add event button or a date box on the calendar will display a pop up as shown in Figure 6.5. The user will then be able to choose

* the kind of task to add (Weekly Report, Meeting, FYP Plan Submission, Interim Report Submission or Final Report Submission)
* One or several projects to create the task for
* The due date of the task or start and end time of the event for meeting

After filling up the option, clicking on the add event button on the form will send the new task details to the database through the backend.

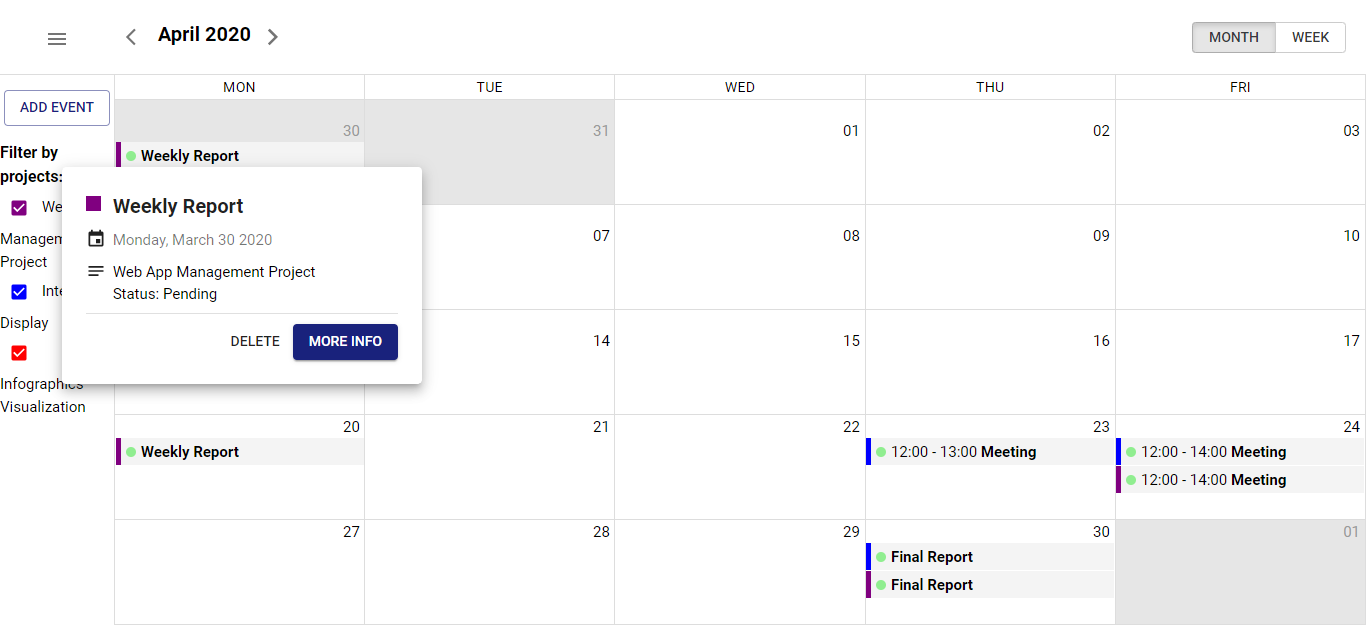


Figure . : Pop up for clicking on Calendar Task

Clicking on a task on the calendar will display a pop up as shown in Figure 6.6. The general details of the task will be displayed with two buttons on the bottom, the delete and more info button. The delete button will remove the task from the database, while the more info button will bring users to the task list page.

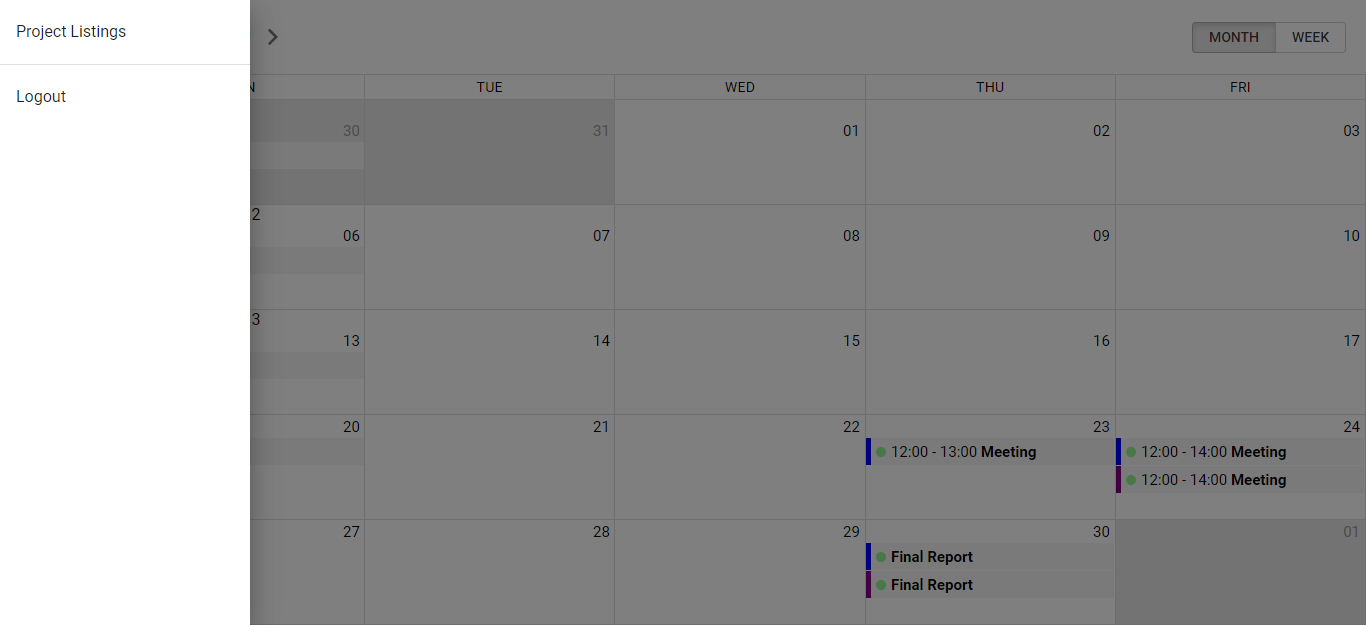


Figure . : Menu button panel from Calendar for Professor

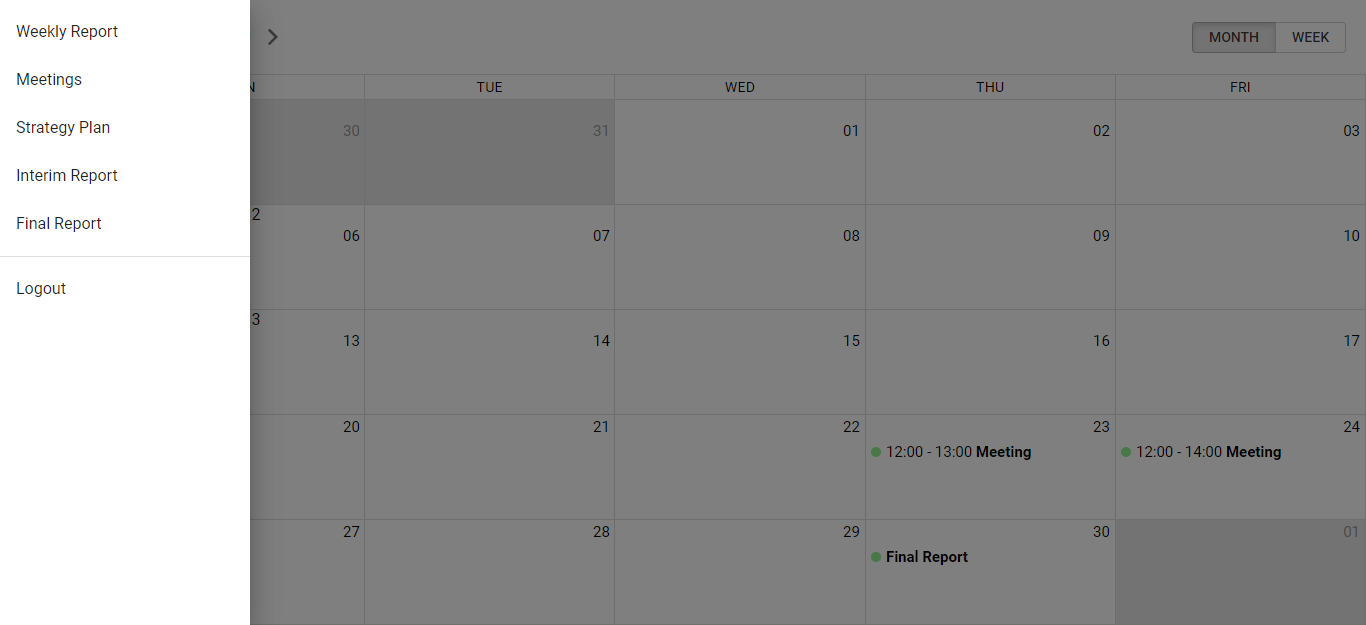


Figure . : Menu button panel from Calendar for Student

The menu button on the top left of the calendar web view will open a menu panel. A professor will open a panel as shown in Figure 6.7 while a student will open a panel as shown in Figure 6.8. The project listing option is only available for professor’s account. The student account will have a list of task type and clicking on the option will route the student to the respective task list page. The logout button will log the user out and route the user back to the login page.

### Project Listing

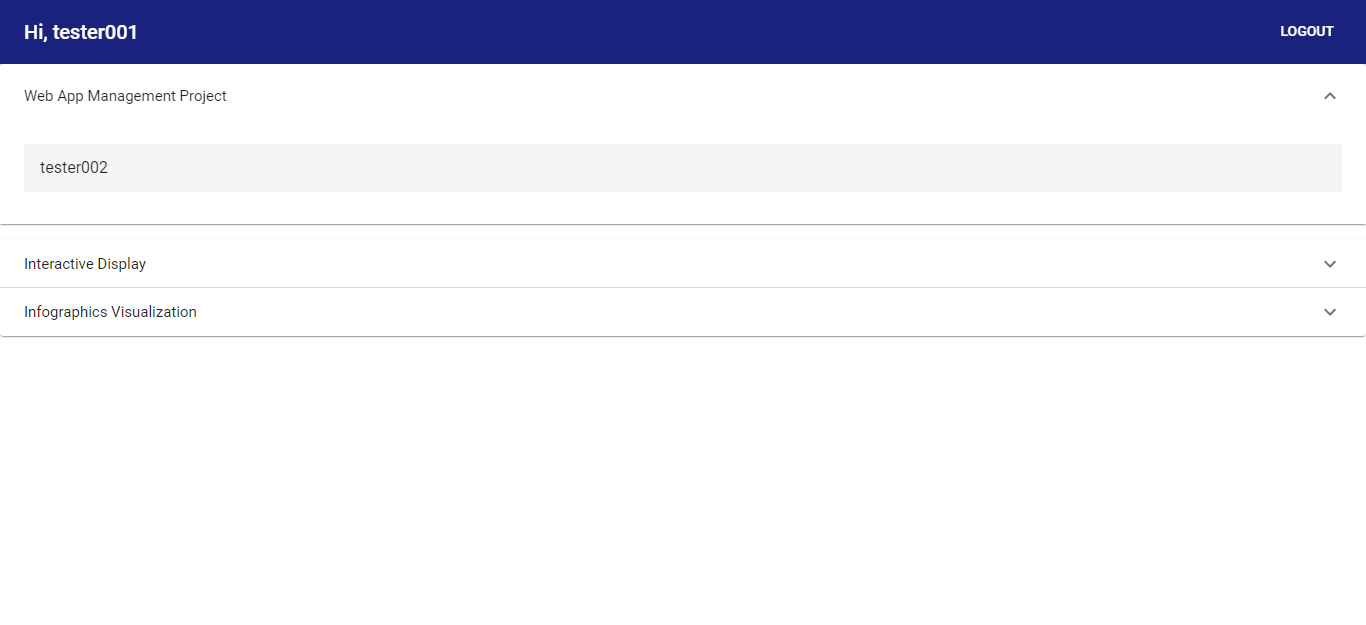


Figure . : Project Listing Page

The project listing page as shown in Figure 6.9 is only available for the professor’s account. The professor can see the list of projects they oversee and the respective student(s) under the project. Clicking the student’s name will route the professor back to the calendar page with the selected student’s respective tasks list.

### Task List Page

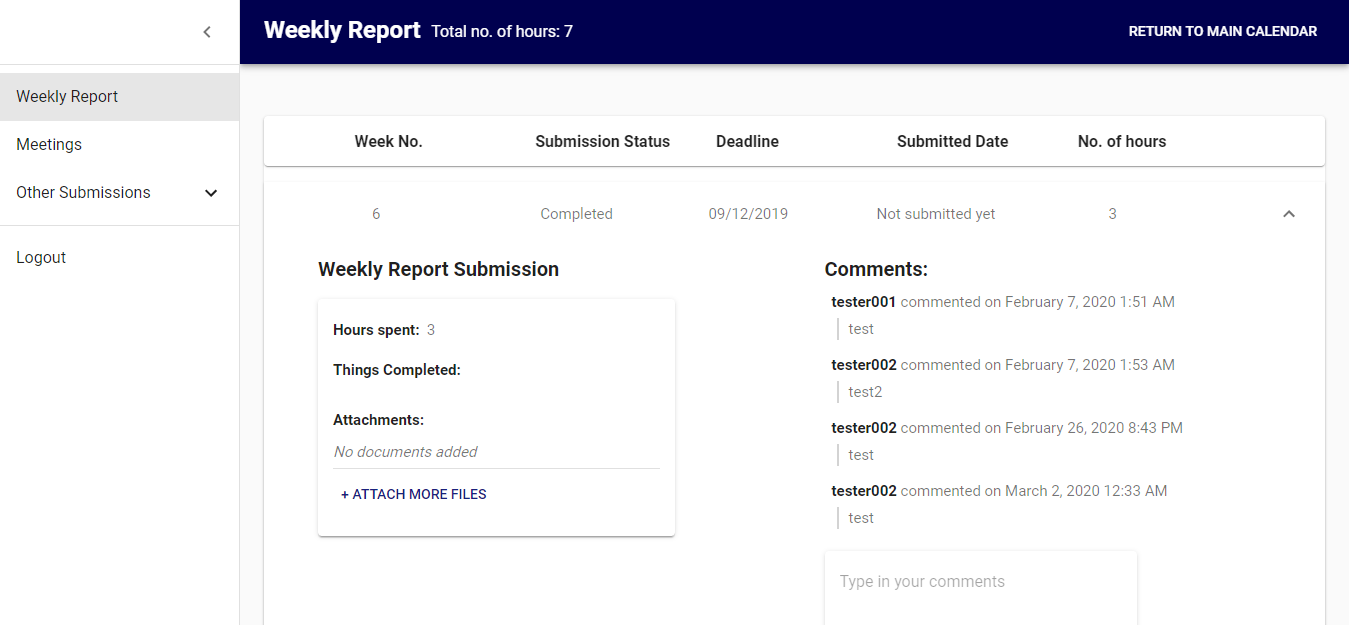


Figure . : Weekly Report Page opened dropdown

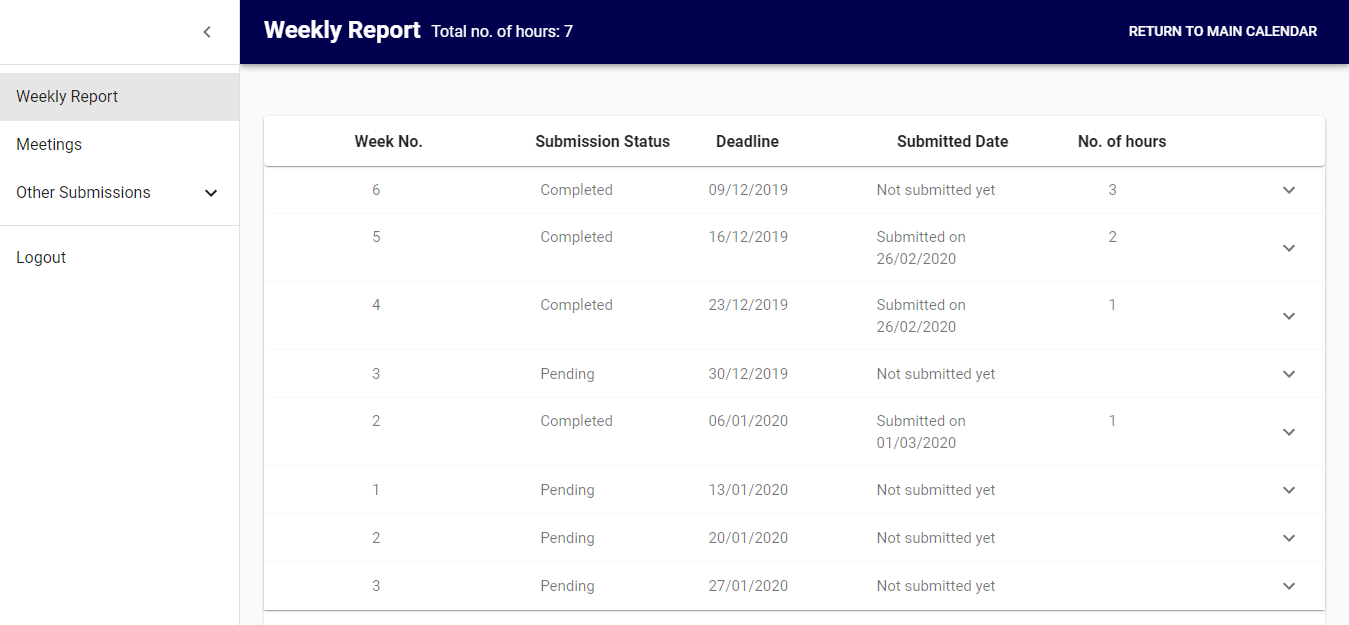


Figure . : Weekly Report Page closed dropdown

The weekly report page is illustrated in Figure 6.10 and Figure 6.11. Figure 6.10 is the default view when routed into the page, the dropdowns will all be opened. Figure 6.11 is when the dropdowns are all closed. Clicking on the option on the left panel will toggle the user between the various task type list.

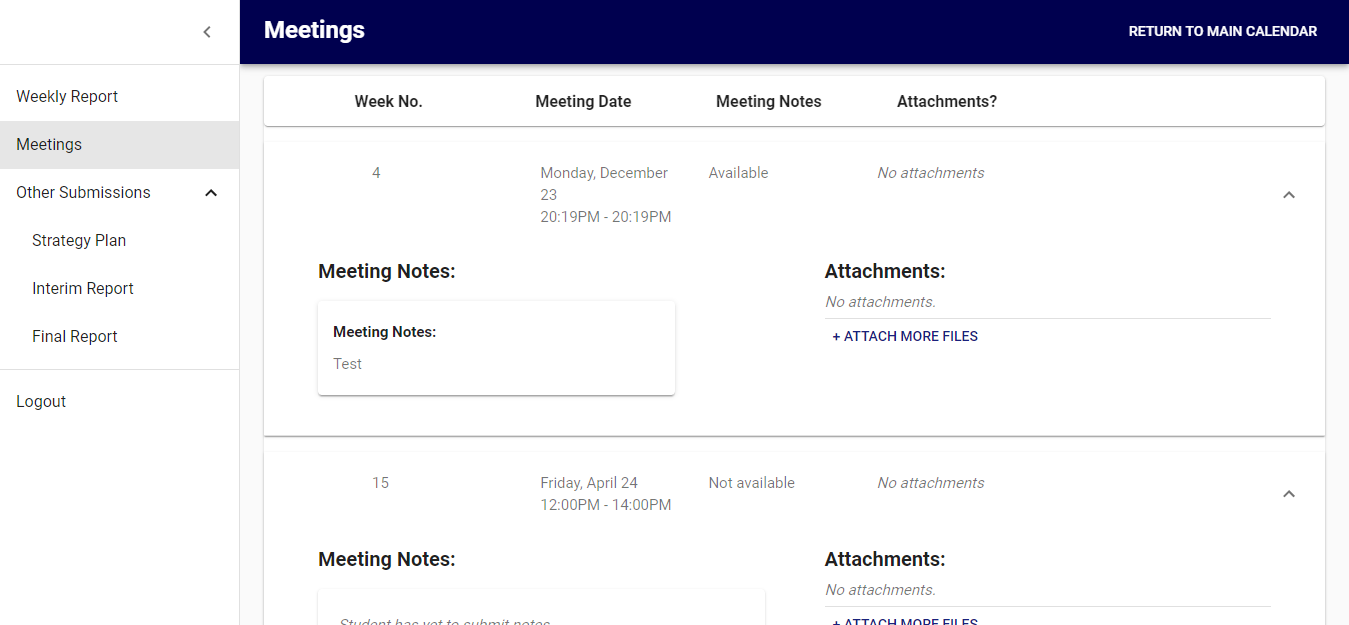


Figure . : Meetings List Page

Figure 6.12 illustrates the meetings list page. The user can upload meetings takeaway and contents to this page.

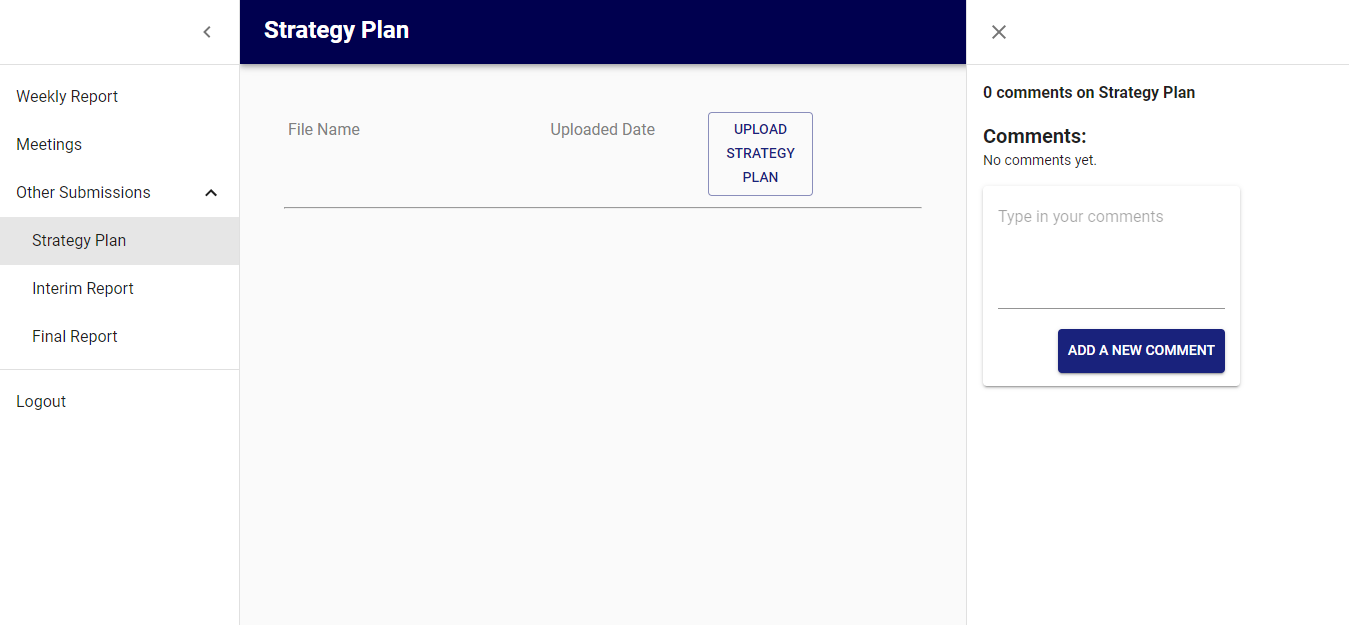


Figure . : Strategy Plan Page

Figure 6.13 illustrates the strategy plan page. The interim report and final report page are similar to Figure 6.13.

## Admin Page

This subchapter will elaborate on the Django Admin page, mainly elaborating on the key function of adding users, semester and editing tasks.

### Django administration login

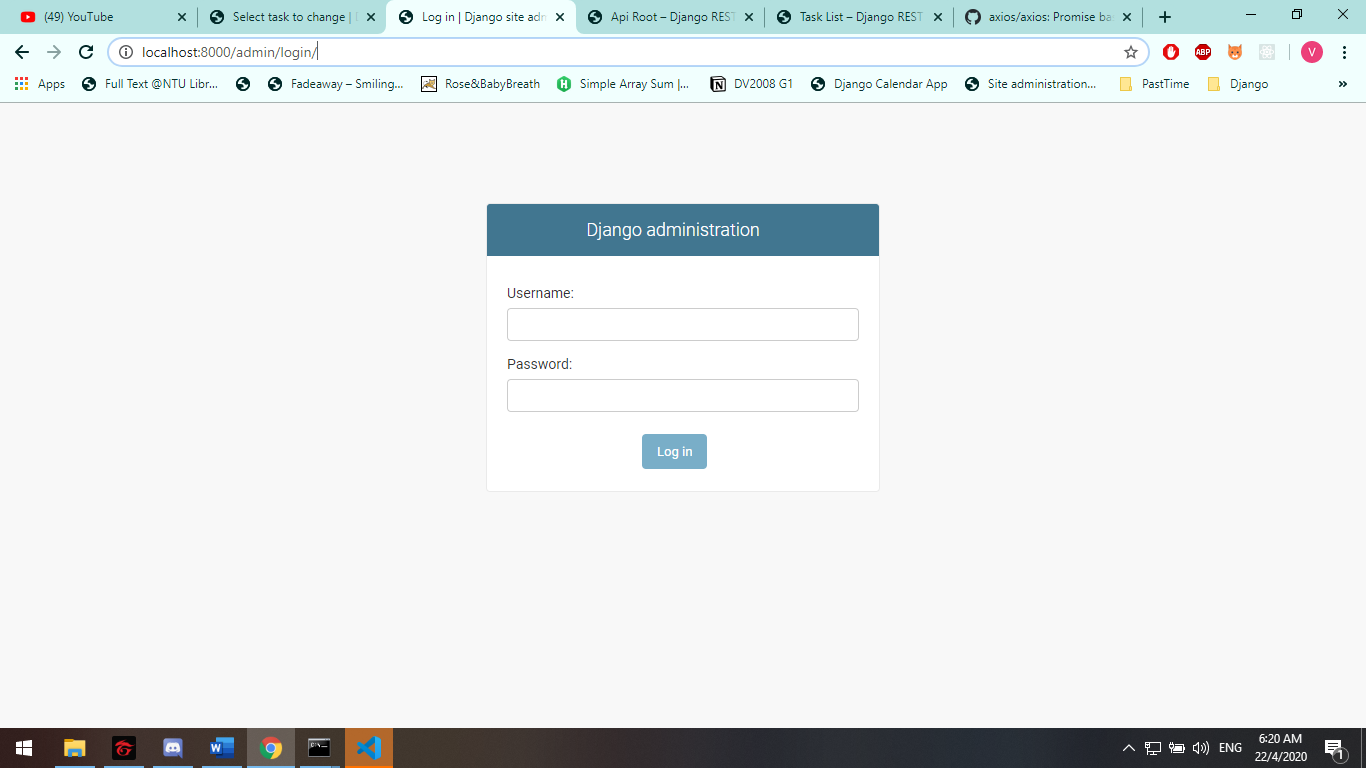


Figure . : Django Admin Login Page

Figure 6.14 is the Django administration login page at <http://localhost:8000/admin/login/>. Django administration is a website that allows the administer to edit the models that are registered to the admin site. The models are registered through the admin.py page (Chapter 5.2.2).

Basic syntax for registering model:

|  |
| --- |
| admin.site.register(<model name>) |

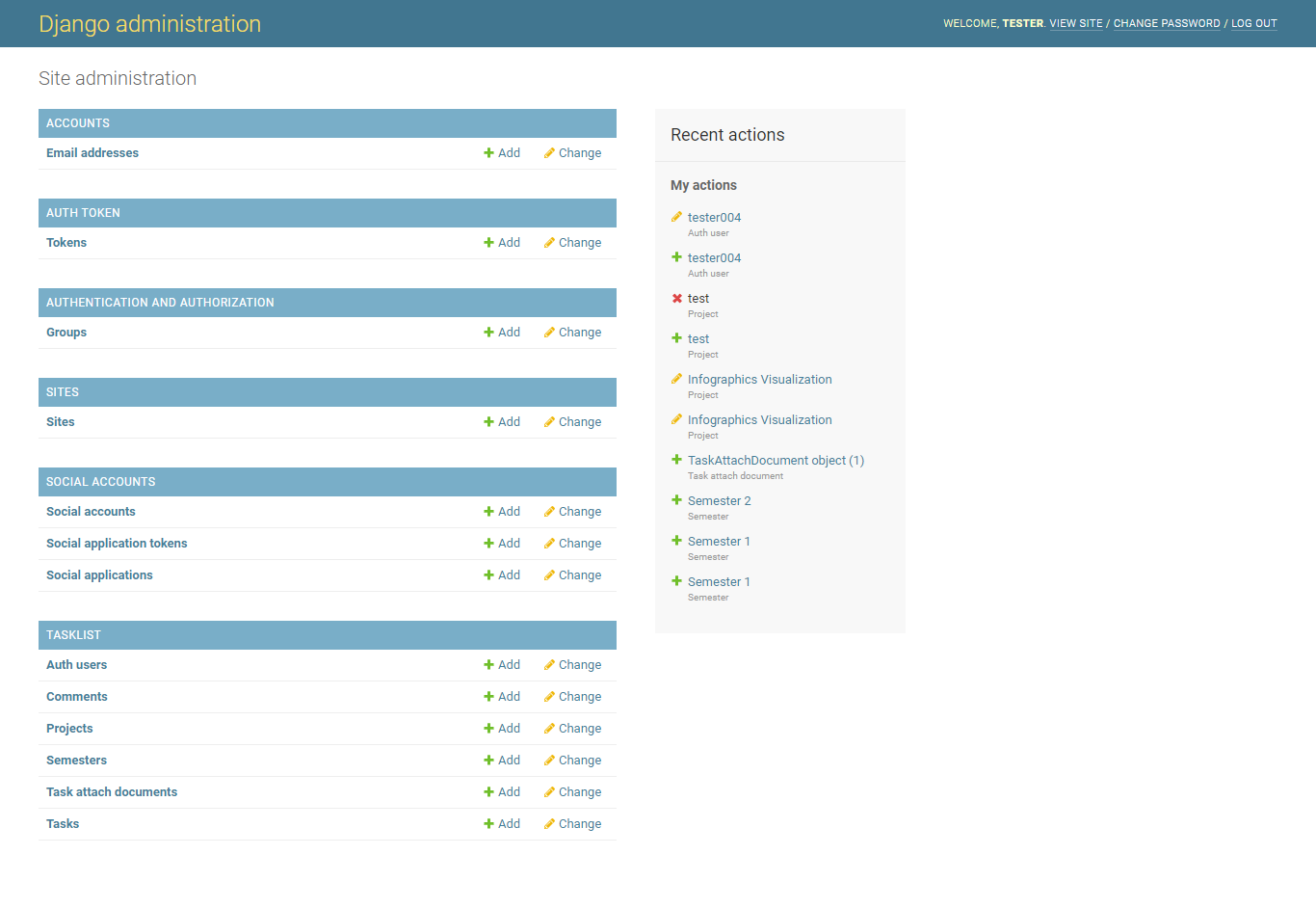


Figure . : Django Admin Page

Figure 6.15 illustrates the Django administration homepage after successful login. Accounts, auth token, authentication an authorization, sites, social accounts are Django registered models. Under the last header, tasklist are the models manually registered for the admin site.

### Django administration Tasks

#### Overview

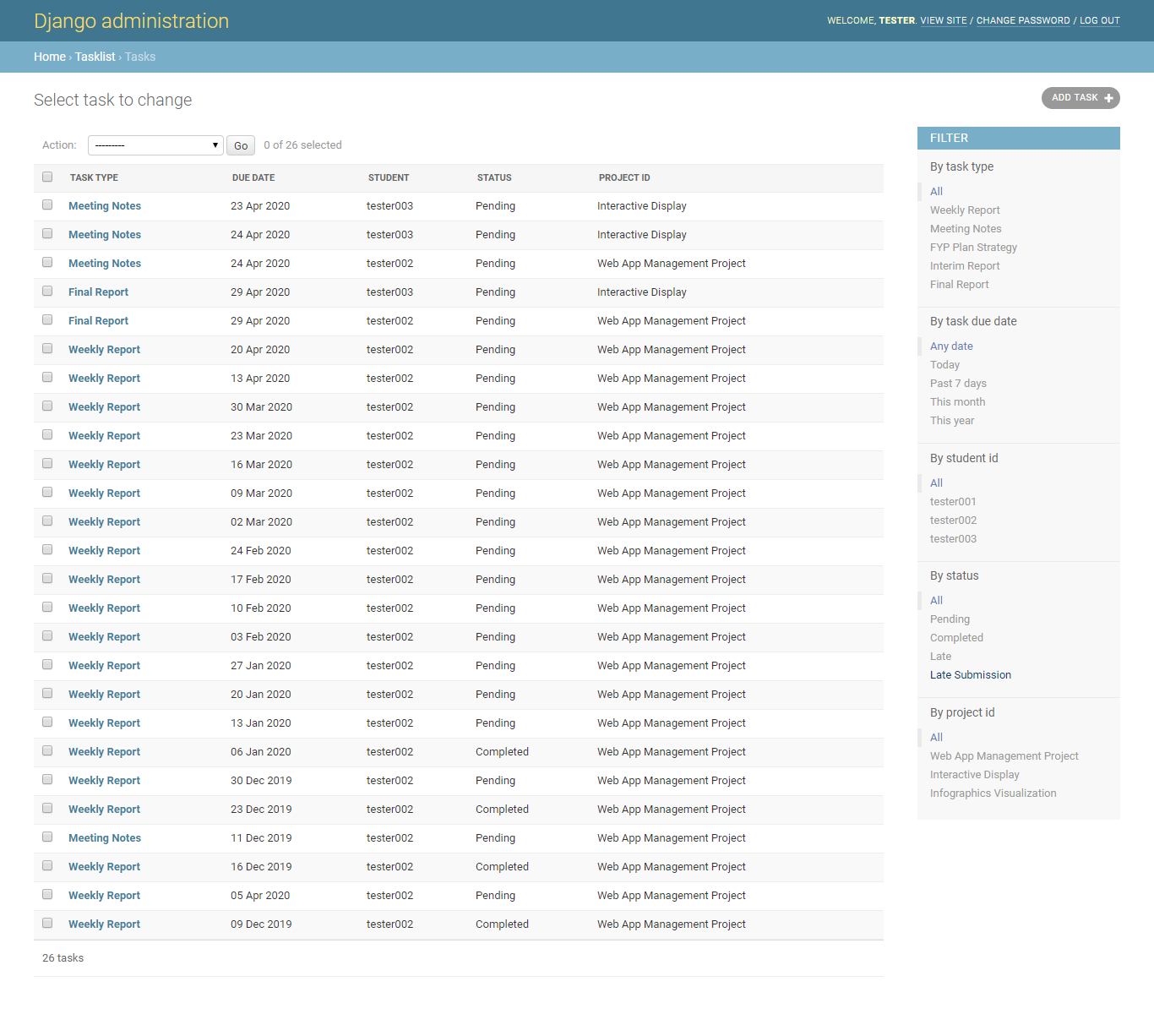


Figure . : Django Admin Tasks Page

Figure 6.16 illustrates the Tasks page, the view after Tasks is clicked from Django administration homepage. The page shows the list of tasks, reflecting the task table date from the database. Each column showing the task type, due date, student name, status, and project of each task item. The right side of the page are filtering option for the administrator. At this page, admin can select several tasks and delete them by selecting ‘Delete selected tasks’ from the dropdown beside action at the top of the list and then clicking Go.

Codes for the filtering and display of the Task table:

|  |
| --- |
| class TaskAdmin(admin.ModelAdmin):      model = Task      list\_display = ('task\_type', 'due\_date', 'student', 'status', 'project\_id')      list\_select\_related = ('student\_id', 'project\_id')  list\_filter = ('task\_type', 'task\_due\_date', 'student\_id', 'status', 'project\_id')  admin.site.register(Task, TaskAdmin) |

#### Add New

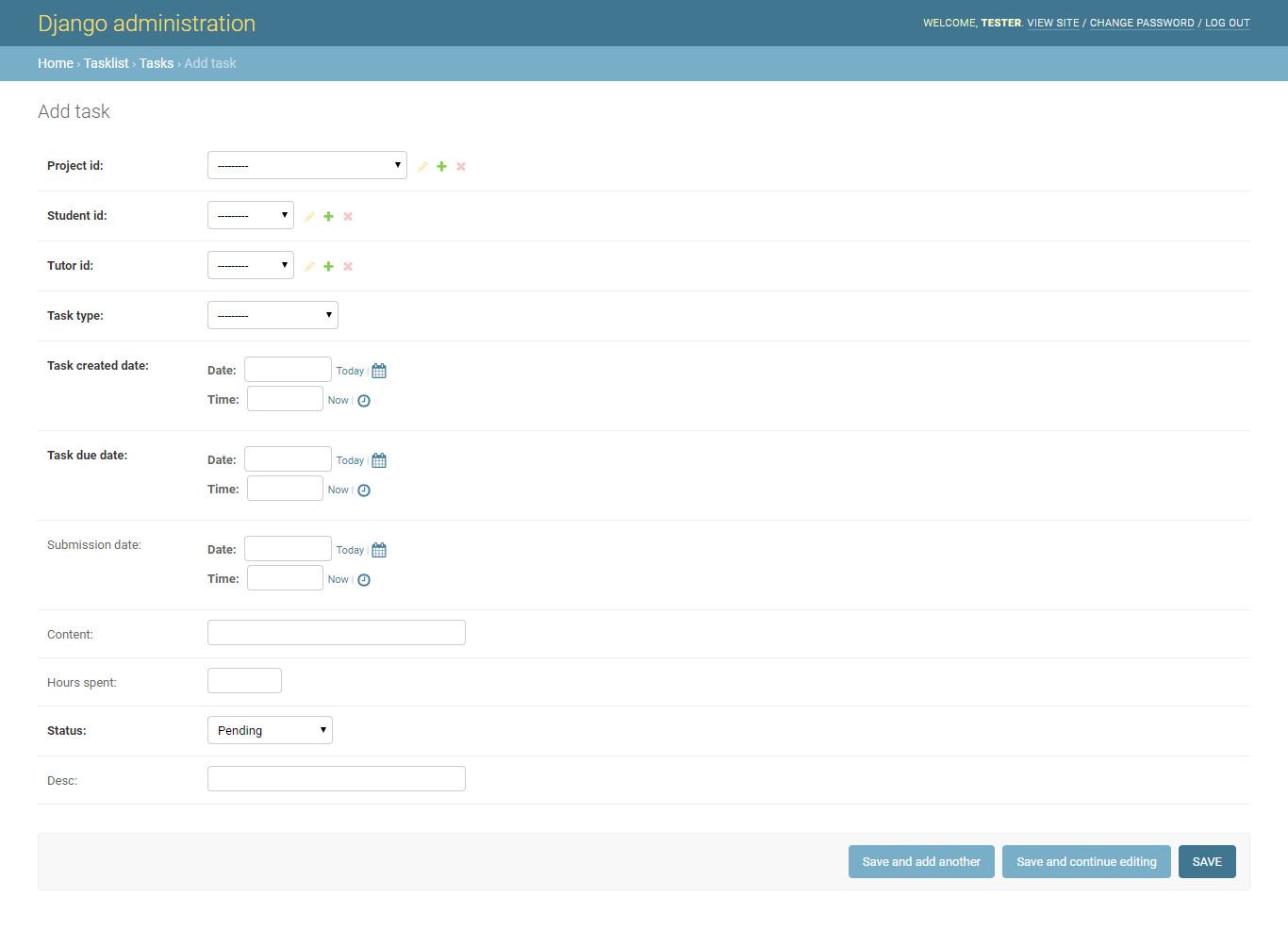


Figure . : Django Admin add new Task Page

Figure 6.17 illustrates the view after clicking add task at the top right side of the Tasks page. This page allows admin to add new task, committing it to the database after clicking on the save button on the bottom right of the page. The first 3 selections will reflect the name of the respective ids instead of id number (for example, name of project and not project id). Task created date for meetings task type should reflect the start time of the meeting.

#### Edit

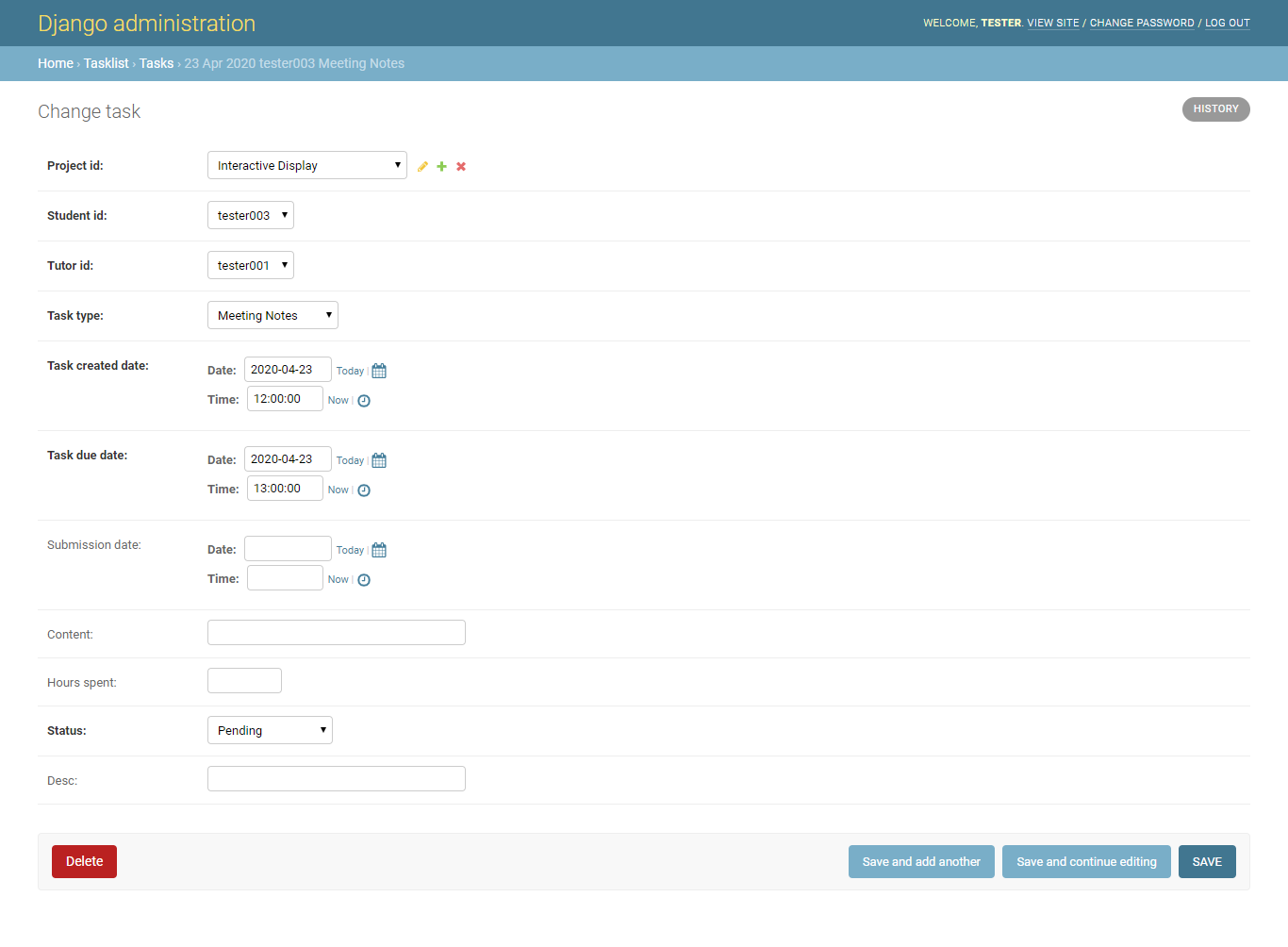


Figure . : Django Admin edit Task Page

Figure 6.18 illustrates clicking on a row on the Tasks page, bringing the admin to the edit page. The admin can then edit the selected Task details. Changes can be committed by clicking the save button on the bottom right of the screen.

### Django administration Projects

#### Overview

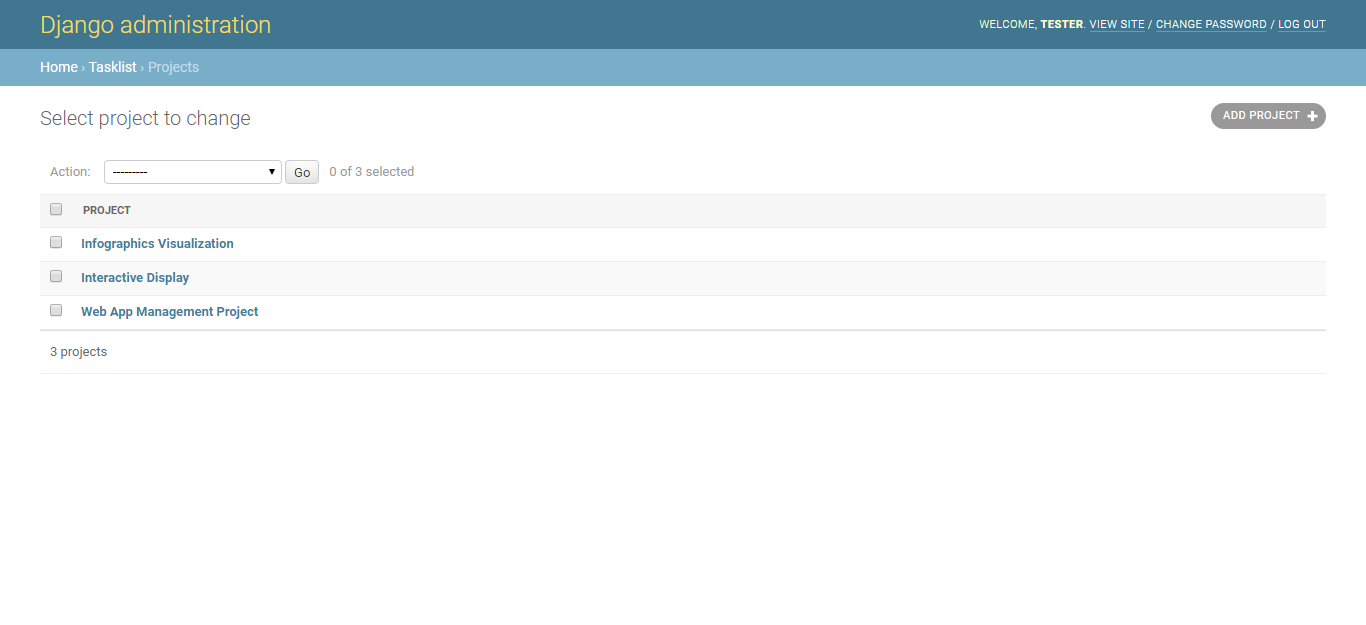


Figure . : Django Admin Projects Page

Figure 6.19 illustrates the Projects page, the view after Projects is clicked from Django administration homepage. This page illustrates the list of projects. At this page, admin can select several projects and delete them by selecting ‘Delete selected projects’ from the dropdown beside action at the top of the list and then clicking Go.

#### Add new

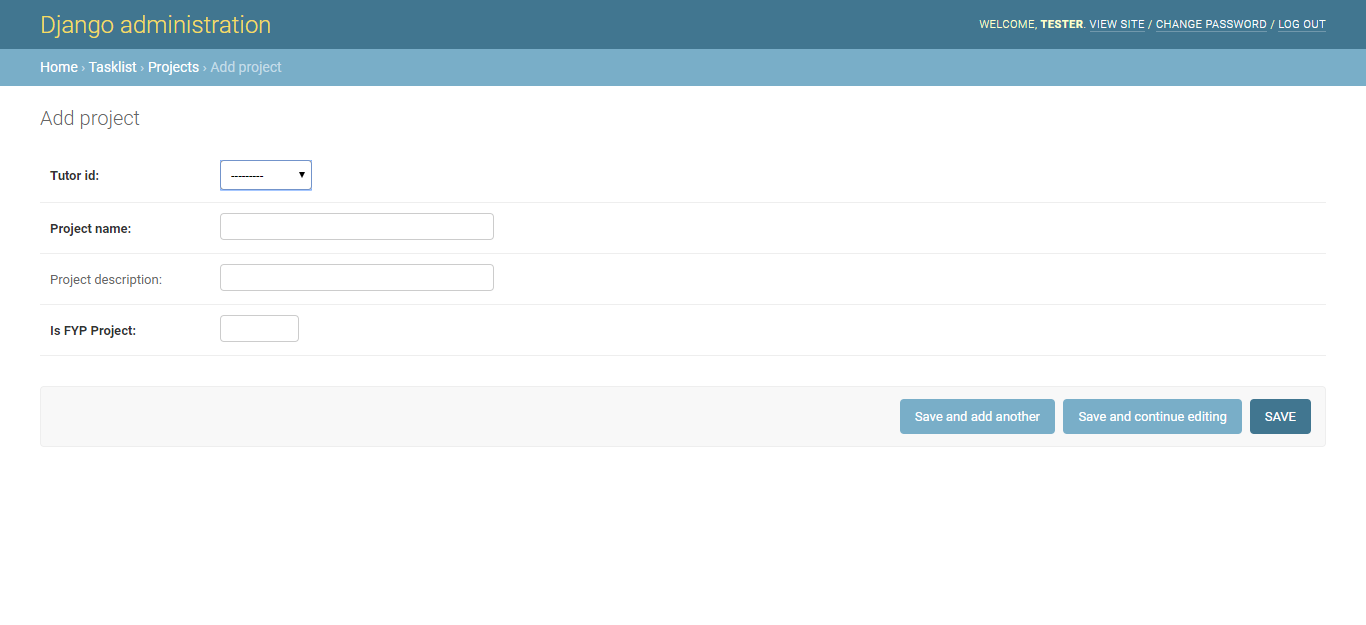


Figure . : Django Admin add new Project Page

Figure 6.20 illustrates the view after clicking add project at the top right side of the Projects page. This page allows admin to add new project, committing it to the database after clicking on the save button on the bottom right of the page. Tutor id will be selecting the tutor name, is FYP Project should be either 1 or 0, 1 for true and 0 for false.

#### Edit

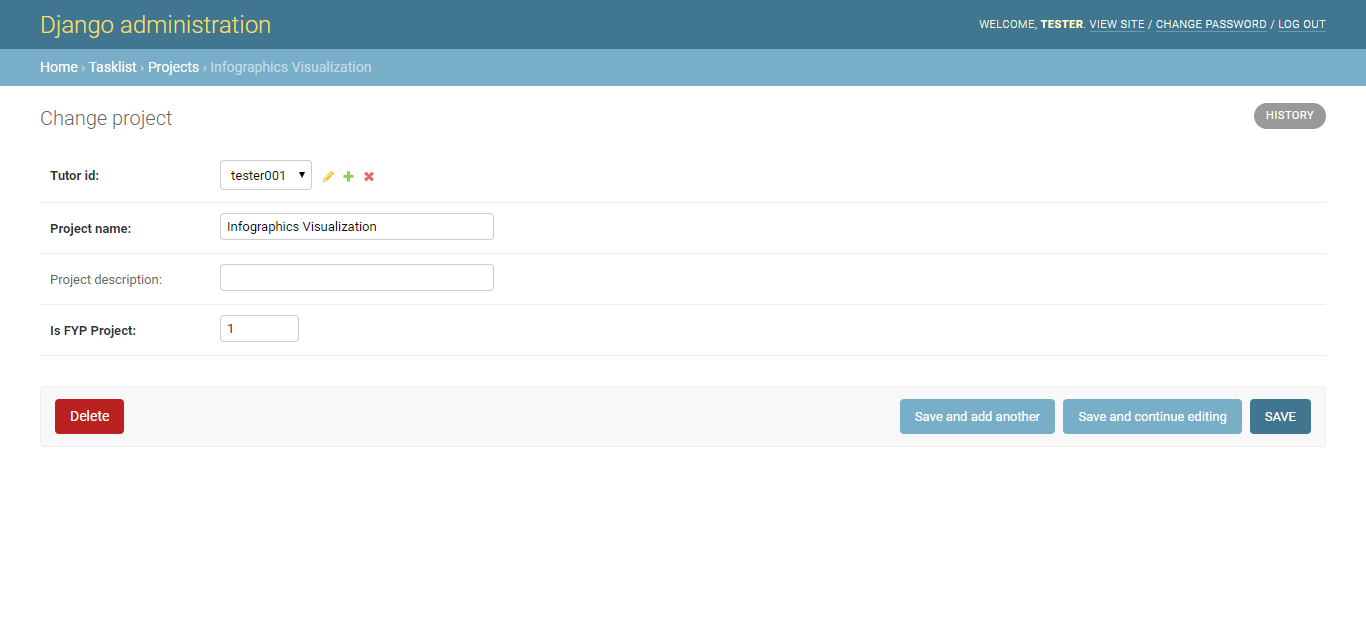


Figure . : Django Admin edit Project Page

Figure 6.21 illustrates clicking on a row on the Projects page, bringing the admin to the edit page. The admin can then edit the selected project details. Changes can be committed by clicking the save button on the bottom right of the screen.

### Django administration Semesters

#### Overview

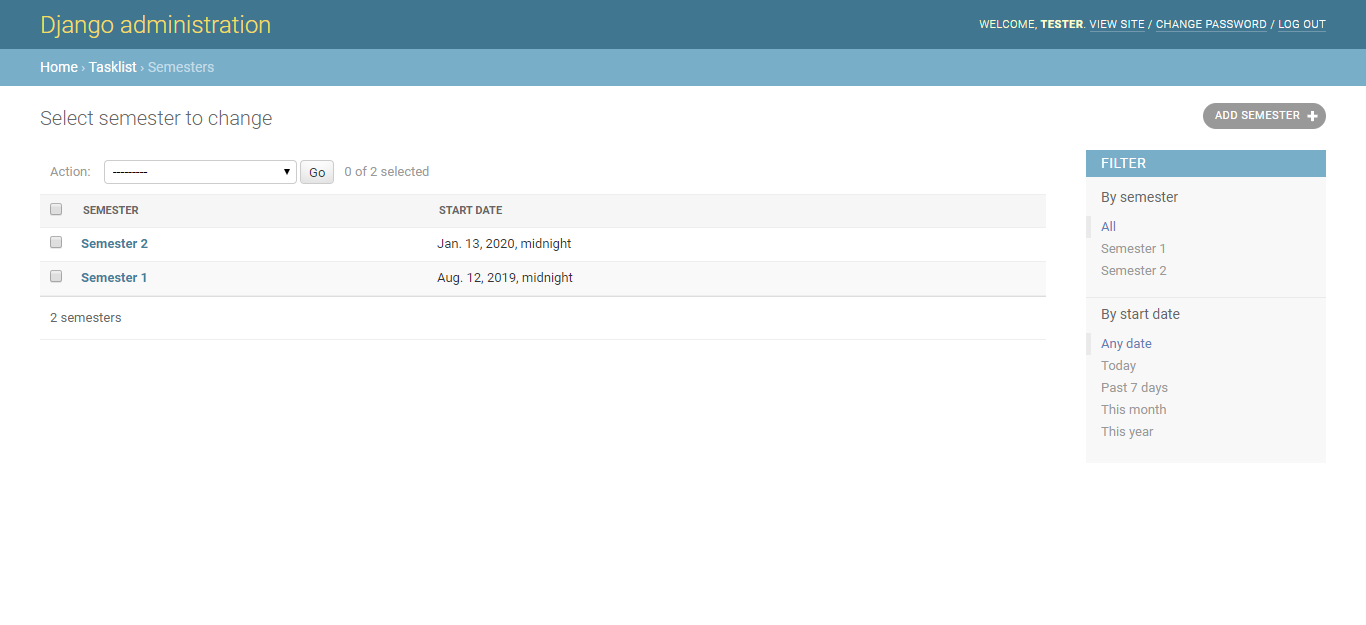


Figure . : Django Admin Semesters Page

Figure 6.22 illustrates the Semesters page, the view after Semesters is clicked from Django administration homepage. This page illustrates the list of semesters. At this page, admin can select several semesters and delete them by selecting ‘Delete selected semesters’ from the dropdown beside action at the top of the list and then clicking Go.

#### Add New

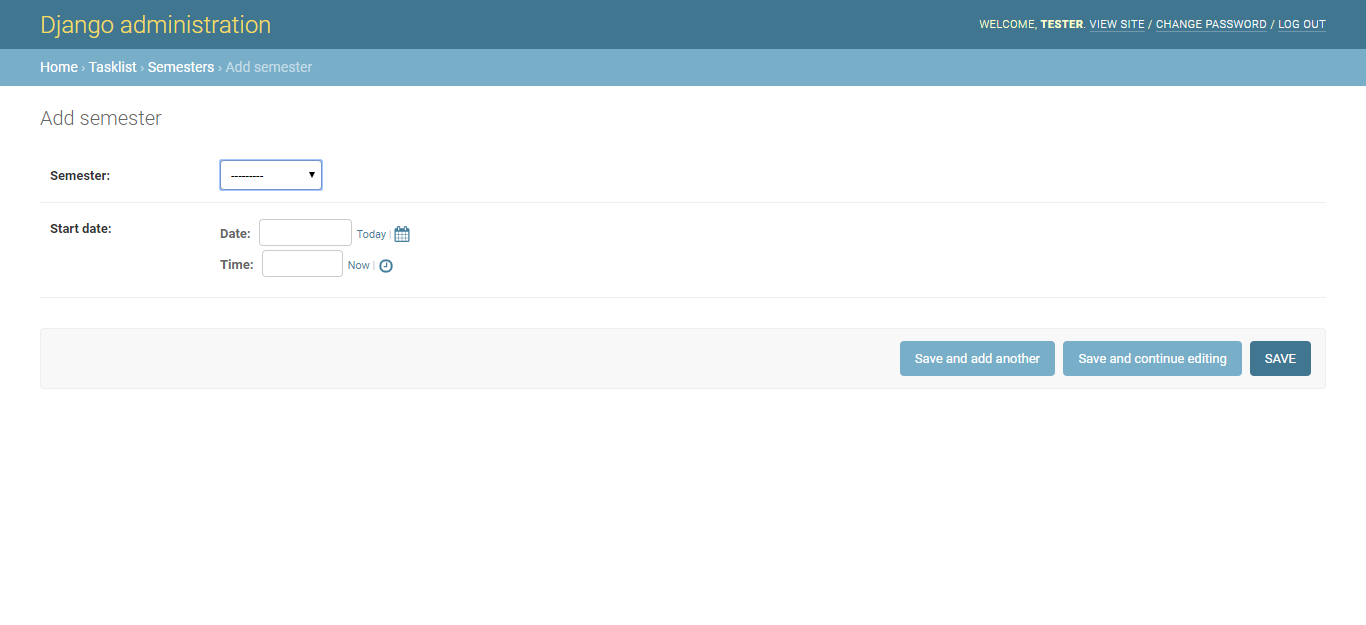


Figure . : Django Admin add new Semester Page

Figure 6.23 illustrates the view after clicking add semester at the top right side of the Semesters page. This page allows admin to add new semester, committing it to the database after clicking on the save button on the bottom right of the page. Semester will be selecting the between Semester 1 and Semester 2.

#### Edit

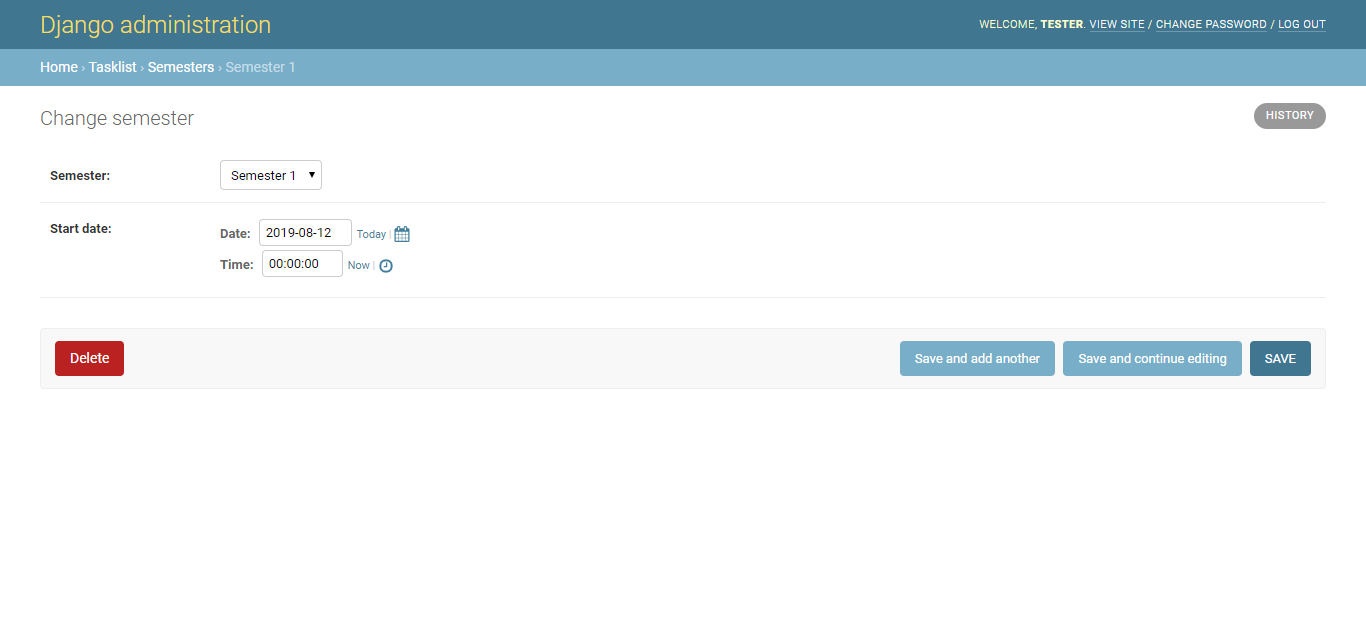


Figure . : Django Admin edit Semester Page

Figure 6.24 illustrates clicking on a row on the Semesters page, bringing the admin to the edit page. The admin can then edit the selected semester details. Changes can be committed by clicking the save button on the bottom right of the screen.

### Django administration Users

#### Overview

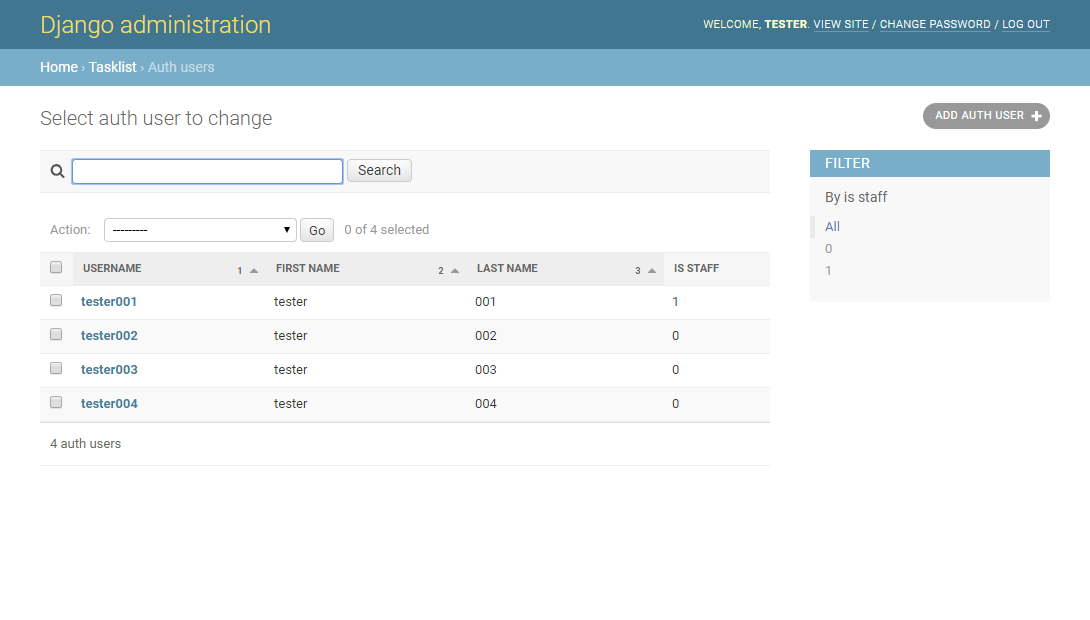


Figure . : Django Admin Users Page

Figure 6.25 illustrates the Users page, the view after Auth Users is clicked from Django administration homepage. Each column showing the username, first name, last name and is staff (0 for false, 1 for true) of each user item. The right side of the page allows filter by is staff. At this page, admin can select several users and delete them by selecting ‘Delete selected users’ from the dropdown beside action at the top of the list and then clicking Go.

#### Add New

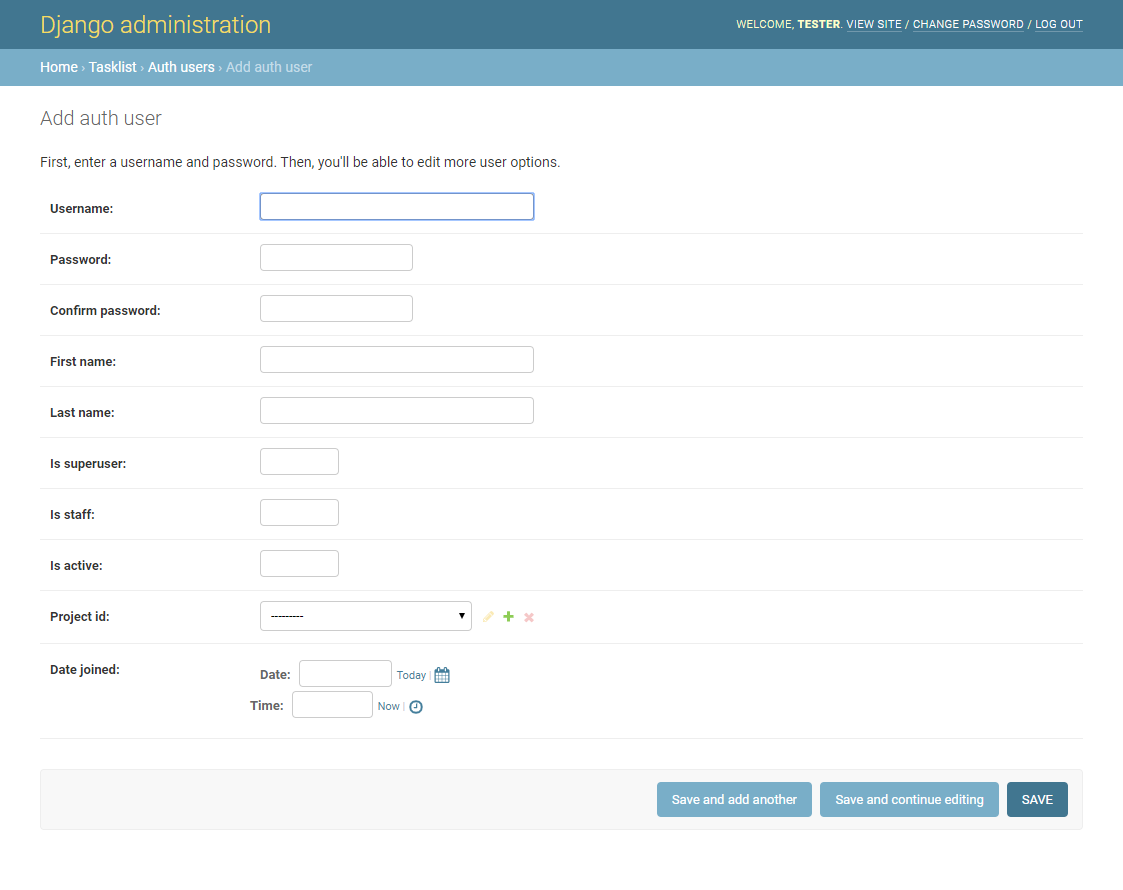


Figure . : Django Admin add new User Page

Figure 6.26 illustrates the view after clicking add auth user at the top right side of the Users page. This page allows admin to add new user, committing it to the database after clicking on the save button on the bottom right of the page. For inputting new user, username must be unique. For is staff and is active option, 0 or 1 should be filled in, 0 for false and 1 for true. Project id will be selecting from a list of projects.

#### Edit

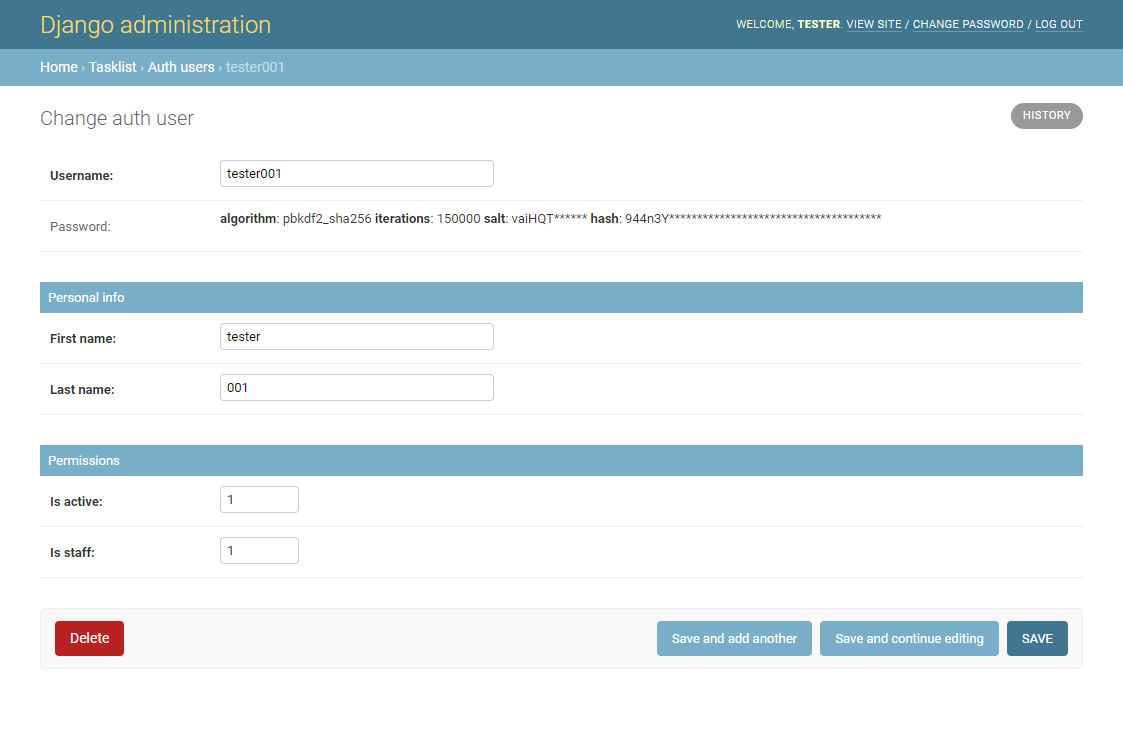


Figure . : Django Admin edit User Page

Figure 6.27 illustrates clicking on a row on the Users page, bringing the admin to the edit page. The admin can then edit the selected user details. Changes can be committed by clicking the save button on the bottom right of the screen.

## Use-Case Diagram

Admin Professor

Student

Admin Website

Project Website

Figure . : Use-Case Diagram

Figure 6.28 illustrates a use-case diagram for the websites. The Django administrator website can only be access by admin professors (Professor users whose is staff is set to true). The project website can be accessed by all users.

# Conclusion and Recommendation

## Conclusion

### Overview

A centralized platform where FYP and DIP submission can be collated will make submissions more organized. Through the website, professor can get all the documents submitted for the projects they oversee. Students can use the website to submit the documents and use the website as a reminder of the various deadlines and meetings. The website thus meets the objective of the project to create a web application to facilitate the tracking of DIP and FYP progression, moving this process from a mixture of offline and online procedure to a full online platform.

### Tools used

Django framework and ReactJS was used in the creation of this website.

Django framework comes with its own set of functions which made creating a website faster and simpler. The challenging part of using Django framework is understanding how the it works and getting to know the various functions and syntax to use them. After getting to know how to use the framework, it will greatly ease the process of website creation. For example, Django comes with its own authentication function, thus cutting the time to create a login function for the application.

ReactJS has many modules are available for installation, these modules have helped in the creation of the application.

## Recommendation for Future Works

This subchapter will elaborate on some recommendation for future works.

### Password Change

Currently in the application created, the only way to change the password of the user is to use command prompt to change it. Ideally, changing of password should be done on the project website or a link through email if the user forgets their password. Therefore, a password change function would be good to implement into the application.

Command Prompt to change password:

|  |
| --- |
| python manage.py changepassword <username> |

### Auto Change Status of Task

A recommendation would be to schedule cronjob to auto update the status of a Task. The cronjob should check the Task table for tasks with status set as ‘pending’ followed by checking the due date against the current time and updating the status of the tasks accordingly. A suggestion will be to set the cronjob to be scheduled at 12 midnight daily.

# References

[1] J. P. Conti, "The 10 greatest communications inventions," *Communications Engineer,* vol. 5, no. 1, pp. 14-21, 2007.

[2] I. W. T. I. I. database. Statistic [Online]. Available: <https://www.itu.int/en/ITU-D/Statistics/Pages/stat/default.aspx>

[3] P. Gerrard and J. B. Cunningham, "The diffusion of internet banking among Singapore consumers," *International journal of bank marketing,* 2003.

[4] QuickBase. *A Timeline of Database History*. Available: <https://www.quickbase.com/articles/timeline-of-database-history>

[5] Technopedia. (2017). *Web-Based Application*. Available: <https://www.techopedia.com/definition/26002/web-based-application>

[6] J. Vainikka, "Full-stack web development using Django REST framework and React," 2018.

[7] O. Dictionary, "Database," ed, 2020.

[8] O. Dictionary, "Relational Database," ed, 2020.

[9] Oracle. (2020). *MySQL 8.0 Reference Manual*. Available: <https://dev.mysql.com/doc/refman/8.0/en/what-is-mysql.html>

[10] T. Branson, "The 5 Best Reasons to Choose MYSQL - And its 5 Biggest Challenges,"Available: <https://dataconomy.com/2017/04/5-reasons-challenges-mysql/>

[11] TechTerms, "Framework," ed, 2013.

[12] P. Wayner, "7 reasons why frameworks are the new programming languages,"Available: <https://www.infoworld.com/article/2902242/7-reasons-why-frameworks-are-the-new-programming-languages.html>

[13] J. Forcier, P. Bissex, and W. J. Chun, *Python web development with Django*. Addison-Wesley Professional, 2008.

[14] S. Aggarwal, "Modern Web-Development using ReactJS," *International Journal of Recent Research Aspects,* vol. 5, pp. 133-137, 2018.

[15] M. UI. (2020). *Material UI*. Available: <https://material-ui.com/>

[16] T. Feddern-Bekcan, "Google calendar," *Journal of the Medical Library Association: JMLA,* vol. 96, no. 4, p. 394, 2008.

[17] M. w. docs. (2020). *Cross-Origin Resource Sharing (CORS)*. Available: <https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS>

[18] Django. *Django Documentation Version 3.0*. Available: <https://docs.djangoproject.com/en/3.0/topics/db/models/>

[19] Django. *Django REST framework - Serializerz*. Available: <https://www.django-rest-framework.org/api-guide/serializers/>

[20] Redux. *Redux Documentation - Store*. Available: <https://redux.js.org/api/store>

# Appendix A

#### A1. Edited Auth User Table

|  |  |  |
| --- | --- | --- |
| Field | Key | Datatype |
| id | PK | INT (11) |
| Password |  | VARCHAR (128) |
| last\_login |  | DATETIME |
| is\_superuser |  | TINYINT (1) |
| username |  | VARCHAR (150) |
| first\_Name |  | VARCHAR (30) |
| last\_Name |  | VARCHAR (150) |
| email |  | VARCHAR (254) |
| is\_Staff |  | TINYINT (1) |
| is\_active |  | TINYINT (1) |
| date\_joined |  | DATETIME |
| project\_Id | FK | INT (11) |

#### A2. Project Table

|  |  |  |
| --- | --- | --- |
| Field | Key | Datatype |
| project\_Id | PK | INT (11) |
| tutor\_Id |  | INT (11) |
| project\_Name |  | VARCHAR (100) |
| project\_Description |  | VARCHAR (3000) |
| is\_FYP\_Project |  | TINYINT (1) |

#### A3. Task Table

|  |  |  |
| --- | --- | --- |
| Field | Key | Requirements |
| task\_Id | PK | INT (11) |
| project\_Id | FK | INT (11) |
| student\_Id | FK | INT (11) |
| tutor\_Id | FK | INT (11) |
| task\_type |  | ENUM ('Weekly Report', 'Meeting Notes', 'FYP Plan Strategy', 'Interim Report', 'Final Report') |
| desc |  | VARCHAR (300) |
| task\_Created\_Date |  | DATETIME |
| task\_Due\_Date |  | DATETIME |
| submission\_Date |  | DATETIME |
| content |  | VARCHAR (3000) |
| hours\_Spent |  | INT (11) |
| status |  | ENUM ('Pending', 'Completed', 'Late', 'Late Submission') |

#### A4. Comment Table

|  |  |  |
| --- | --- | --- |
| Field | Key | Requirements |
| comment\_Id | PK | INT (11) |
| task\_Id | FK | INT (11) |
| user\_Id | FK | INT (11) |
| content |  | VARCHAR (3000) |
| creation\_Date |  | DATETIME |

#### A5. Task Attached Document Table

|  |  |  |
| --- | --- | --- |
| Field | Key | Requirements |
| task\_Attach\_Document\_Id | PK | INT (11) |
| task\_Id | FK | INT (11) |
| attach\_Document |  | VARCHAR (100) |
| uploaded\_Date |  | DATETIME |

#### A5. Semester Start Date

|  |  |  |
| --- | --- | --- |
| Field | Key | Requirements |
| semester\_Id | PK | INT (11) |
| semester | FK | ENUM ('Semester 1', 'Semester 2') |
| start\_Date |  | DATETIME |