FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of bachelor's thesis

Title:	Application for student account management in an organization.
Student:	Mark Awad
Supervisor:	Ing. Michal Valenta, Ph.D.
Study program:	Informatics
Branch / specialization:	Web and Software Engineering
Department:	Department of Software Engineering
Validity:	until the end of summer semester 2021/2022

Instructions

The main idea is that each student is assigned a card. Through this card, they will be able to mark their attendance and use the bank in the application to manage their virtual currency.

The organization has a list of rules and incentives for the student in regards to the currency to increase their attendance. The rules should be reflected in the application.

Follow these steps in the thesis:
1. Collect customer requirements for the system and formalize them.
2. Research and discuss existing solutions.
3. Using standard software engineering methods design your own solution.
4. Implement a functional prototype of the system using Django framework, test and document it.
5. Evaluate your solution.

Electronically approved by Ing. Michal Valenta, Ph.D. on 6 November 2020 in Prague.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor's thesis

Application for student account management in an organization

Mark Awad

Department of Software Engineering
Supervisor: Ing. Michal Valenta, Ph.D.

February 10, 2022

Acknowledgements

I would like to thank my thesis supervisor Ing. Michal Valenta, Ph.D. for his consistent support throughout the writing of the research paper and for his comments that allowed me to further improve the structure of the paper.

A special thanks goes to the St. Paul Sunday School of St. Mark Church in Kuwait, for their constant support and for giving me the opportunity to design for them the application and for having a real world scenario where I took the lead on understanding the business requirements and finding a suitable solution.

Finally, all of this would not have been possible without the support of my parents and how they always motivated me to push myself forward and break new limits, and for that I thank them.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on February 10, 2022 . . . . . .

Citation of this thesis

Awad, Mark. *Application for student account management in an organization.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2022.

Abstrakt

Cílem této práce je návrh a implementace webové aplikace pro malou organizaci spravující studentské účty podle svých požadavků. Každý student má unikátní RFID kartu a prostřednictvím aplikace si může označit svou docházku a vložit/vybrat virtuální měnu organizace. Pro motivaci studentů je za docházku každého studenta přidána pobídka. Tato pobídka přidává na studentský účet nastavitelnou částku virtuální měny. Čím pravidelnější docházka, tím vyšší motivace. Virtuální měnu lze použít při registraci na akce, nákupu oběda nebo nákupu zboží z obchodu organizace. Aplikace má administrátorskou část, která umožní správu studentů v systému. Aplikace je napsána v Pythonu pomocí webového frameworku Django a je nasazena na Google Cloud Platform pomocí principů DevOps.

Klíčová slova správa uživatelů, správa účtů, docházka, virtuální měna, python, django, testy řízený vývoj

Abstract

The goal of this thesis is to provide a web application for a small organization where they have to manage student accounts as per their requirements. Each student has a unique RFID card and through the application, they can mark

their attendance and deposit/withdraw the organization's virtual currency. To motivate the students, an incentive is added for each student's attendance. This incentive adds an adjustable amount of the virtual currency to the student's account. The more regular the attendance, the higher the incentive becomes. The virtual currency can be used when registering for events, buying lunch or purchasing items from the organization's store. The application has a dashboard for admins to able to manage students in the system. The application is written in Python using the Django web framework and is deployed to Google Cloud Platform using DevOps principles.

Keywords user management system, accounts management, attendance, virtual currency, python, django, test driven development

Contents

List of Figures

Chapter 1

Introduction

Smaller Organisations have a difficulty managing their user accounts, especially when the needs of their organisation are constantly expanding. Using tools such as Microsoft Excel could be sufficient in the short term. However, with an increasing number of users, complexity and demands, manual maintenance almost always fails to keep up.

Finding the appropriate tools for organization is a difficult task. Not all organizations share the same needs and abstract solutions tend to be more difficult as they often require more training and a lot of customization. This is exact problem faces my client, they offer free classes to students of different age groups. They aim to manage their students' accounts, provide them with a banking feature that operates on a virtual currency. Having this type of currency in place aims to motivate more students to attend the classes and participate actively, in return, the students can purchase items from the organization's store, buy snacks/meals and pay for future events all through this virtual currency. The client would also like to keep track of the students that are not attending so that they can inquire about them and encourage them to join future classes/events that meet their needs.

The purpose of this project is to provide a fully automated and convenient solution for the organization, this will be done through the following steps:

- Collecting customer requirements for the system and formalizing them.
- Researching existing solutions.
- Finding a solution based on the research.

Chapter 2

Aim of Bachelor Thesis

In order to provide the best solution to the client, a research of the available products in the market were analysed. Most student management solutions available focus on the payment aspect, whether it is allocated to the school or as administrative fees or as instalment management features. However, for a non-profit organization which does not depend or need student fees to operate, this feature would not prove to be so useful.

The client needs three main system components in order to operate;

- Student account management
- Student attendance
- Integrated virtual currency

They would like to be able to keep track and manage current students and graduates accounts. They would also like to have the possibility to add an attendance record to any given service. A key point in the system is that it should be able to have a virtual currency which students are able to obtain through attendance or by depositing it into their personal bank account. The more they attend the services consecutively and regularly, the more virtual money they get.

2.1 Available Solutions

From the research identified, I was able to find two key applications that could be regarded as competitors to my custom solution. These applications are Microsoft Excel or a Customer Relationship Management (CRM) solution. For the purpose of simplicity, I have looked into Microsoft Dynamics as it is one of the leading CRM solutions.

I was also able to find other customized student management solutions that already exist, offering management of student accounts and their attendance.

However, they lacked the option of a virtual currency solution as an incentive for the students. For this reason, I have decided to disregard comparing these solutions.

2.1.1 Microsoft Excel

One of the simplest ways to manage student accounts is to create a spreadsheet with the students' information. Another spreadsheet needs to be created for attendance based on the preferred duration. And finally, one last spreadsheet that would keep a record of the students balance of the virtual currency.

This was how the organization was managing its students' information for years. However, this method of storing and altering data was quite a tedious and error prone process. Maintaining the information between all of the spreadsheets can be quite cumbersome especially as the student count increases. A lot of manual work is required and details can be missed quite easily.

The data is also not relational, it would be quite difficult to relate the data between spreadsheets and gather useful statistics. For example; if the teacher wanted to know how consistent a student is when it comes to their attendance, they would have to check all of their previous attendance spreadsheets and gather the information manually.

However, Microsoft Excel offers a few benefits though. The application is quite easy to use and almost everyone in this day and age has basic to intermediate experience with the tool. This eliminates the need for training and would allow the administrators to get started right away and manage the data.

The tool is also readily available and almost everyone has a subscription to the entire Microsoft Office Suit, so no extra payment is needed.

2.1.2 Microsoft Dynamics

Microsoft Dynamics is the more complicated solution available. Microsoft offers multiple dynamic solutions and for the purposes of this research, I will specifically discuss Dynamics CRM which is a software as a service (SAAS), managed and hosted by Microsoft in the cloud.

While Dynamics is a great solution for managing customer accounts (in this case students), the learning curve is quite slow. For a person that has not worked with any CRM solutions before, they would first need to understand and learn the concepts and applications for such a solution and then learn how to use the tool itself which can be quite difficult and time consuming.

The user interface (UI) of the solution (see Figure 2.1) is quite superb in terms of viewing the information, navigating through the platform and for customization. You do not need to know how to code to be able to customize

account forms for example as you can drag and drop buttons and fields. However, with more complex demands, coding knowledge is necessary.

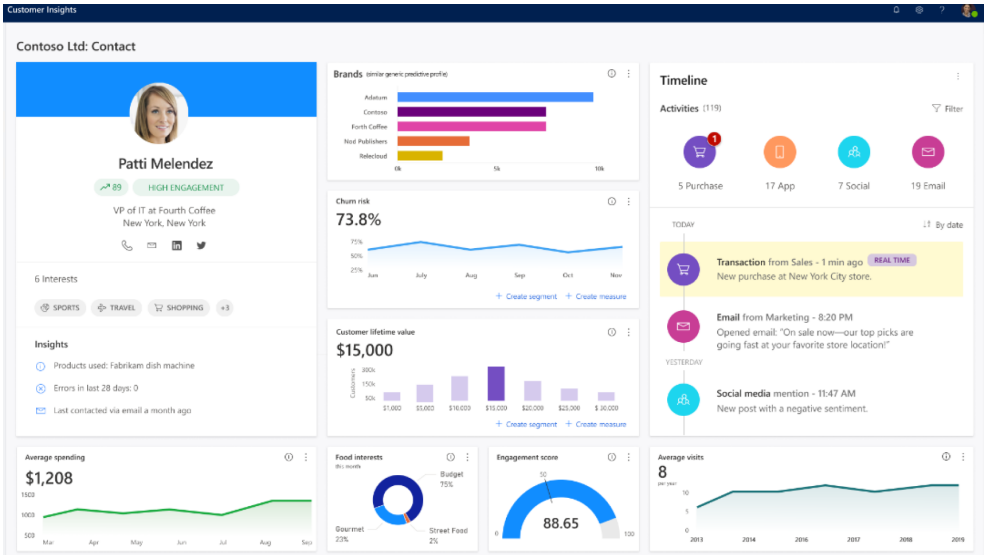

Figure 2.1: Example of UI in Microsoft Dynamics CRM

A feature that could be quite useful for the organization, is that Dynamics is able to send specific and/or personalized emails to customers or admins with any information needed. This could be helpful for the organization if they need to know for example how many people were absent, who was not present for which duration of time, reminders to students of upcoming lessons and much more.

With Dynamics, you also have the ability to limit privileges, which can help the organization limit mistakes and errors. The client can have multiple roles for admins, developers and teachers, each with their required permissions.

Dynamics does not have the ability for users to have an integrated attendance system or the addition of a virtual currency. Adding these features would be quite expensive as you would have to integrate it to the platform and create your own customization. Testing these features would be quite difficult and time consuming as well but still possible.

That being said, Microsoft offers a great customer service experience where the experts would be able to help and guide the user with any issues they might be facing and while it might take more time to get a response, they would provide you with exactly what is needed. The solution is also hosted on the cloud, so there is no need to manage the infrastructure and you can go ahead and start with the customization right away.

Because Dynamics is managed and hosted by Microsoft, the client will be paying a hefty monthly fee to keep the solution running. And while exporting

of data is possible, it is not quite easy and requires the user to have the knowledge and the appropriate to be able to do this without errors. This makes moving to a different solution when needed much more difficult.

2.2 Results

While Microsoft Excel does not require training and is readily available, a lot of manual work is needed to update and maintain the existing data which is quite error prone and takes more time to achieve.

Microsoft Dynamics is managed and hosted on the cloud. However, the learning curve can take some time to be able to modify the solution to the client's needs and is the more expensive solution.

The research of these products was quite useful as it allowed me to better understand the requirements and needs for an organization. With my provided solution, every task would be automated and it would be hosted on a small compute in Google Cloud Platform (GCP) which has a small monthly cost for the client. My solution does not focus much on the UI as it is not a major requirement for the organization only being used by the admins and teachers.

Chapter 3

Analysis

To better understand the needs of the client, an analysis of their requirements and use cases were conducted.

3.1 Requirements

The client would like a solution that fits their requirements precisely and with little to no manual work. The requirements can be split into two parts, functional and non-functional.

3.1.1 Functional Requirements

1. Add any given day to the attendance sheet.
    a) Priority: High
    b) The system should only add the days where the teachers will present lectures. This could be weekly or bi-weekly.
    c) There should exist an option for the admin to add a given day to the attendance.
2. Students mark their attendance through their unique id.
    a) Priority: High
    b) The system should present an easy way of marking the student's attendance to avoid the manual work of actually typing the full student's name.
3. Admins have the ability to mark student's attendance by name.
    a) Priority: Medium

   b) The system should be flexible enough to allow the admin to add student's attendance if the students do not remember their unique id.

4. Keep record of attendance table.

   a) Priority: High

   b) The system keeps track of each student's attendance and admins can view these records.

5. Keep records of students' transactions; deposits and withdrawals of virtual currency to the system.

   a) Priority: High

   b) The students are allowed to deposit or withdraw printed currency to the system.

6. Keep record of students balance.

   a) Priority: High

   b) The admins are able to view each student's balance.

7. Keep track of the total money in circulation in the system.

   a) Priority: Medium

   b) Admins are able to view how much of the virtual currency all of the students have at the current time.

8. Admins can add/change/delete students from the system.

   a) Priority: High

   b) The system needs to have a simple interface for admins to on-board new students, change or delete records completely.

9. Automatic addition of the virtual currency for each attendance entry.

   a) Priority: High

   b) The system needs to automatically add an adjustable amount of currency each time they have marked their attendance.

10. Consecutive Attendance of students results in more added virtual currency.

    a) Priority: High

    b) The system checks for consecutive attendance and adds bonus based on the amount of times they have attended class without skipping attendance.

c) There should be a weekly, monthly and quarterly attendance bonus.

11. Send a report after every attendance event.

    a) Priority: High

    b) The report should include the students who have not attended with how many times they haven't attended previously.

12. The system supports having different types of bank cards.

    a) Priority: Medium

    b) The accepted cards are Basic, Gold and Platinum.

    c) The bonus attendance amount of the currency is affected by a multiplier for different card types.

13. The students get their grade automatically updated after every year.

    a) Priority: Medium

    b) The system updates the student's grade based on the admin's input of the year ending and starting a new one.

14. Users who are no longer students can be saved on the system as Alumni.

    a) Priority: Low

    b) The students are moved to become alumni once they passed grade 13.

    c) The admin is able to choose the students so that the system can make them alumni.

3.1.2 Non-functional Requirements

1. Web Application:

    a) The client side (hosts and users) will have a web application interface accessible on a web browser.

    b) The web application will be running on cloud.

    c) The client side must be running on Chrome or Safari.

2. RFID card scanner:

    a) The unique id of each card is to be represented as the student's id.

    b) Students can scan their cards to quickly add attendance or use banking features.

3. Backup:

a) A monthly backup of the data is stored on the system.

4. Server:

a) The backend of the web application will be stored on a compute instance in GCP.

3.2 Use Cases

It is necessary to understand how the users of the organization will interact with the system. Based on the analysis, the actors of the system and their interactions were identified.

3.2.1 Actors

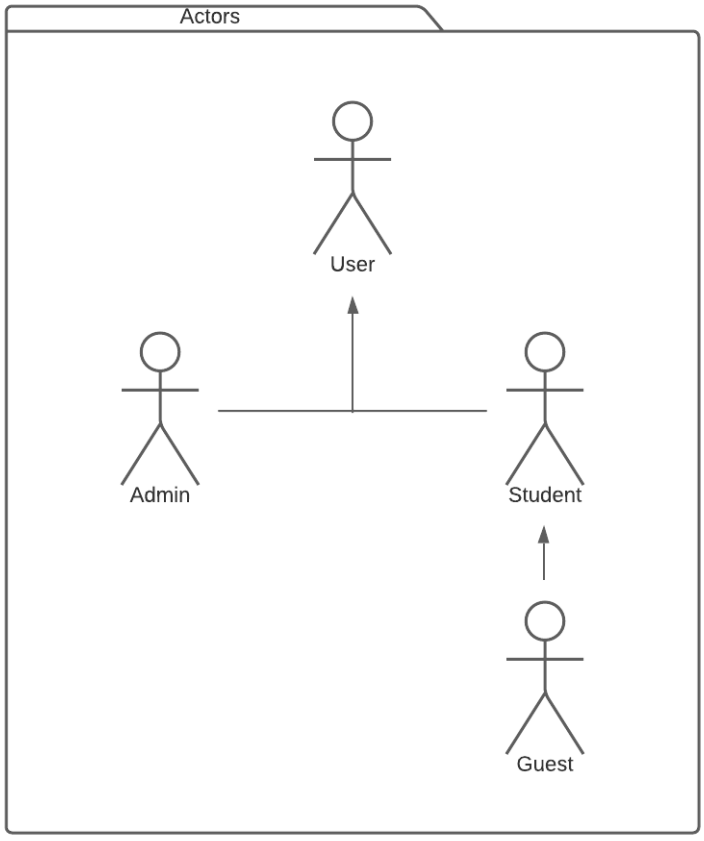

Figure 3.1: Actors involved in use cases.

1. Student

- Participates in Attendance
- Has unique RFID card.

2. Admin
    - Manages the students in the system.
3. Guest
    - Guest is a student with limited privileges to do higher level tasks not accessible to other students in the system.

3.2.2 Student Management

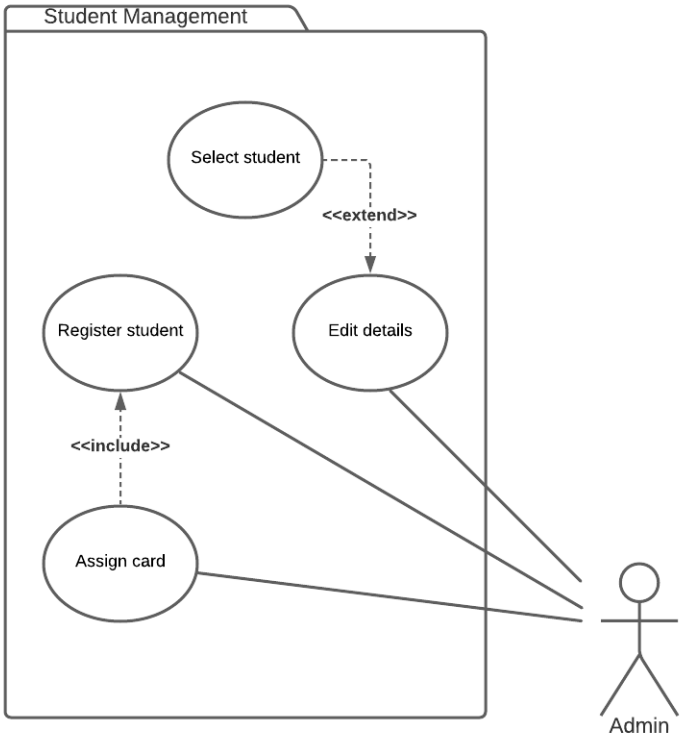

Figure 3.2: Student Management

3.2.2.1 Select Student

Admin can select any student registered in the system and perform actions.

3.2.2.2 Edit Details

Admin can edit student details including:

- Profile information.
- Assign/Change card.
- Change amount of virtual currency.

3.2.2.3 Register Student

Admin can register new students to the system.

3.2.2.4 Assign Card

With each new user a card must be assigned to them.

3.2.3 Attendance

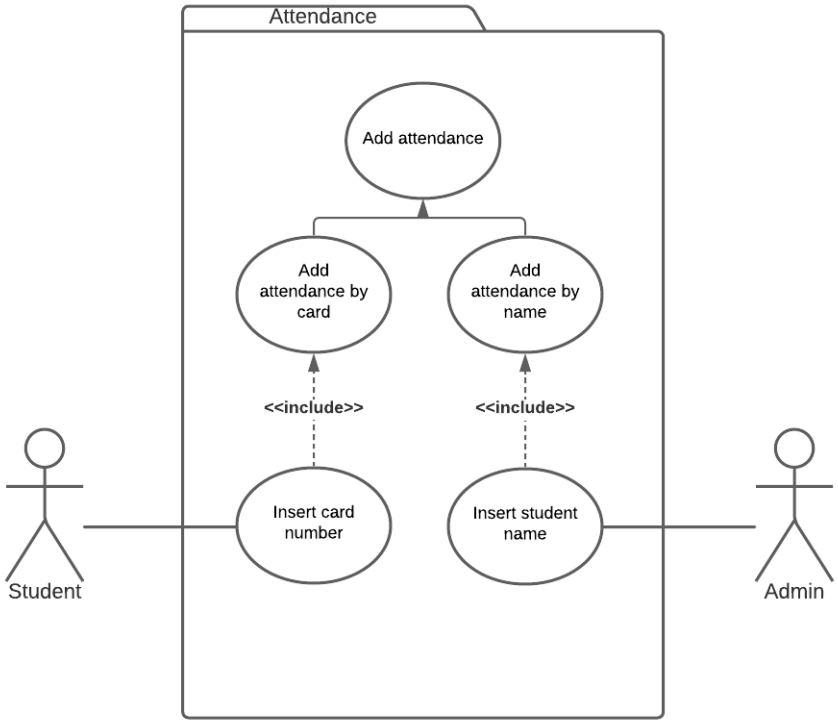

Figure 3.3: Attendance

3.2.3.1 Add Attendance

For each attendance, the system has two options to mark students presence.

- By Card

  Students are able to scan their card through the system to mark their presence.

- By Name

  The admin has the option to mark a student's attendance by inserting their name in the system.

3.2.4 Bank Management

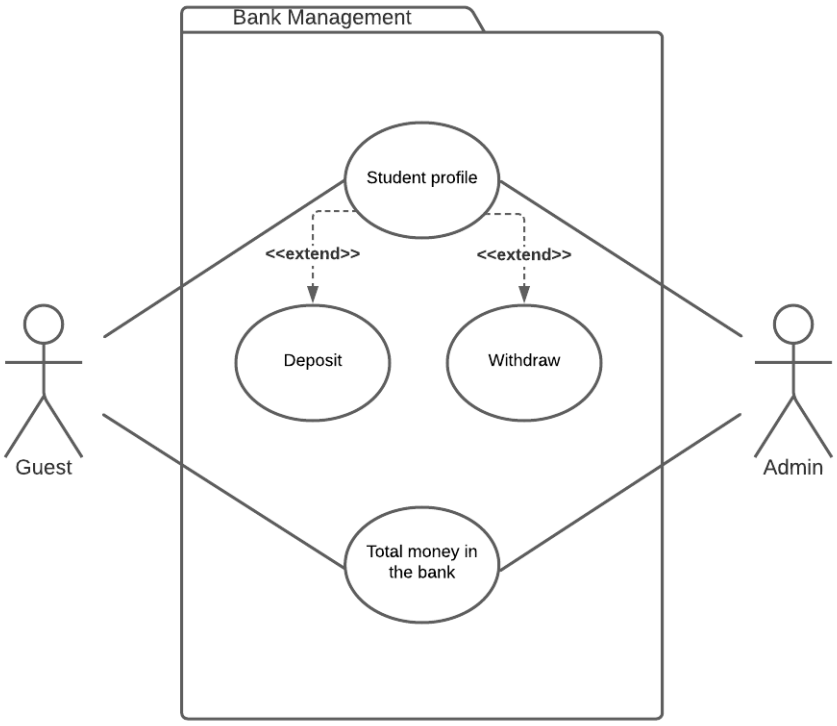

Figure 3.4: Bank Management

3.2.4.1 Student Profile

Both the admin and guest can view a student's bank profile which displays the amount of available currency in the bank and the list of total transactions.

3.2.4.2 Deposit

Actors can deposit currency for a student.

3.2.4.3 Withdraw

Actors can withdraw currency for a student.

3.2.4.4 Total money in the bank

The actors can display the total money in circulation for all students.

Chapter 4

System Architecture & Implementation

The application was built using Django's Model-View-Template (MVT) architecture. Where a model describes what a database table looks like with all of its fields and constraints defined in a class. A Django view, takes in a web request and returns a response. A template on the other hand is a way to generate Hypertext Markup Language (HTML) dynamically. It contains the desired parts of static HTML output, as well as some syntax to help insert data dynamically.

The application was separated into three layers; the data, business and presentation layers. This helped me design and write code more effectively as the separation allowed me to simplify any problem and identify exactly which layer needed modification. Connecting the different layers in the end proved to be an easy task.

One rule I had during development was that I made sure that in order for the presentation layer to reach the data layer, it had to go through the business layer. This allowed for a more consistent code flow.

4.1 Structure

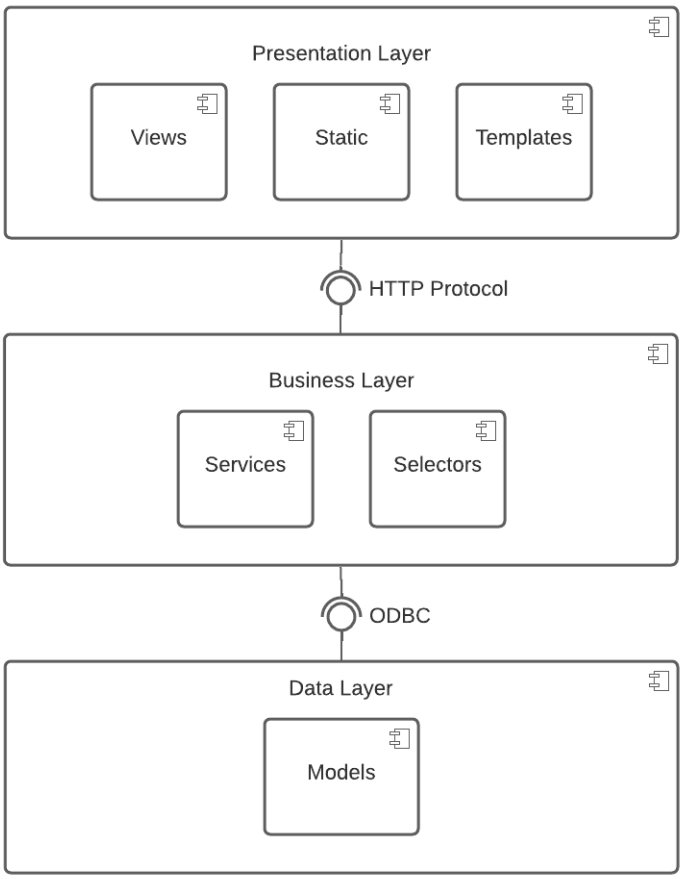

Figure 4.1: Application Structure

4.1.1 Data Layer

The data layer handles the classes and database models. This layer does not include any logic, all it does is persist data or retrieve it from the database.

4.1.2 Business Layer

The business layer includes all of the logic involved with the application. It can only access the data layer.

4.1.3 Presentation Layer

The presentation layer is the first interaction between the client and the server. It is where the client requests a certain page and the server renders this page

with the necessary data. It can only access the business layer.

4.2 Logical Architecture

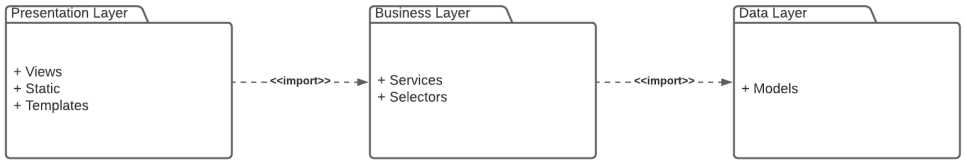

Figure 4.2: Logical Architecture

4.2.1 Data Layer

4.2.1.1 Models

The models are a class representation of database tables. A class mostly includes defined fields as variables and may include some helper functions with little to no logic.

4.2.2 Business Layer

4.2.2.1 Services

The services are classes that transform and send data to the data layer to be persisted.

4.2.2.2 Selectors

The selectors are classes that retrieve data from the data layer. This data can then be transformed if needed.

4.2.3 Presentation Layer

4.2.3.1 Views

It accepts web requests as a parameter and generates the needed response.

4.2.3.2 Static

Cascading Style Sheets (CSS) and JavaScript files.

4.2.3.3 Templates

HTML files with specific Django template syntax.

4.3 Database Model

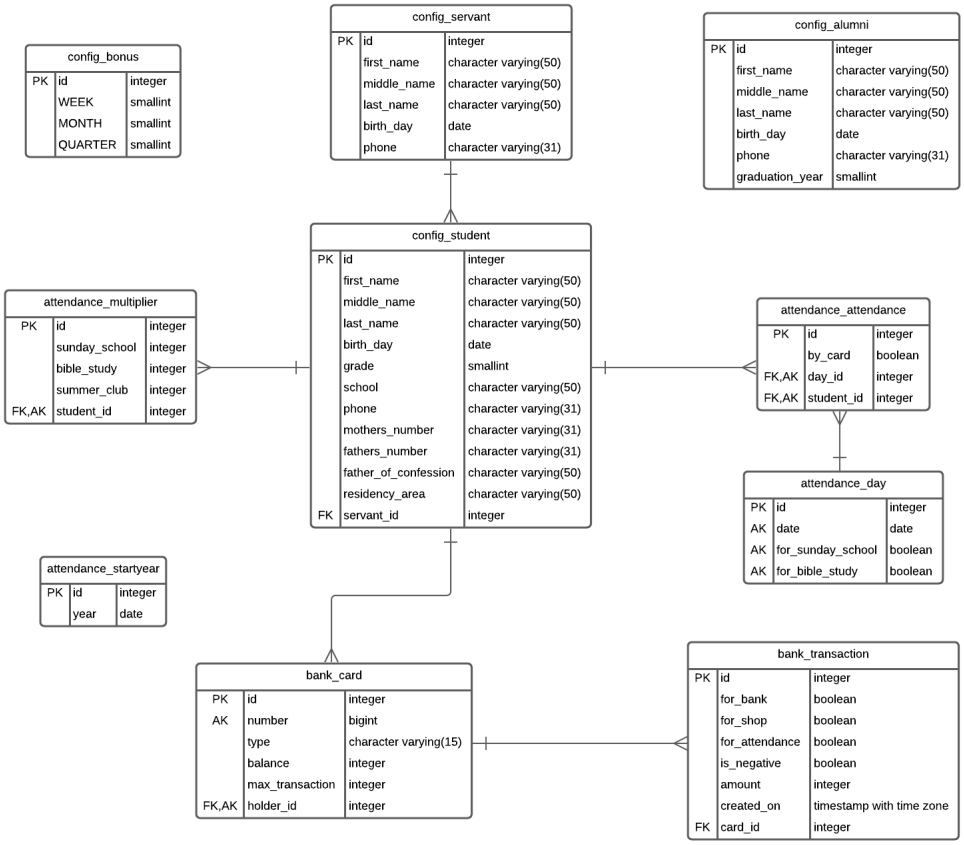

Figure 4.3: Database Model

The main table in the database model is the 'config_student', this is where most of the student records would be added and maintained. Connected to it is the table 'bank_card' where each bank card belongs to one student. The bank card number would act as the student's id. The reason for this is because if a student had lost their card, the admin would be able to change the number on the connected record while still maintaining the previous transactions per user and without having unnecessary duplication.

The 'config_student' table has a many-to-many relation with the 'attendance_day' through the 'attendance_attendance' table. This is where all of the

attendance information is handled for the students. The 'attendance_multiplier' is a table which keeps records of the consecutive attendance per student for each service.

The 'config_servant' table holds the teachers information and which teacher is in charge of which student. While the 'config_alumni' table holds the information of the graduated students from the organization.

The table 'config_bonus' holds just one record which is adjustable which let's the system know how much of the virtual currency should be given to each student for each consecuitive attendance. The table 'attendnace_startyear' holds one record to help the system identify, which date was the beginning of the organization's calendar year.

Chapter 5

Testing & Documentation

With the significant advances in technology over the past decades, we can safely say that we are dependent on software for our daily and professional lives. The impact of a software failing can be catastrophic to a businesses or even an individual. If a system is down, work cannot be achieved.

In a study quantifying the high economic impacts of an inadequate software testing infrastructure [1], the effects noticed were failures in software due to poor code quality, increased software development costs and increased time to market.

Testing is the process of executing a program with the intent of finding errors [2]. In this chapter we focus on the testing strategies used that increased the code quality and the robustness of the application.

5.1 Testing Scenarios

Functional tests were created using both Django and Selenium frameworks [3]. 'StaticLiveServerTestCase' class was used as a base from the module 'django.contrib.staticfiles.testing', this allows the class to launch a live HTTP server in a separate thread, which allows for the integration with selenium to test the website's functionality.

5.1.1 Examples

```
1  def test_users_can_log_attendance_on_the_go(self):
2      # Alice logs in to the website as admin
3      self.browser.get(self.live_server_url)
4
5      user_input: WebElement = self.wait_for(lambda: self.
           browser.find_element_by_id('id_username'))
6      pass_input: WebElement = self.wait_for(lambda: self.
           browser.find_element_by_id('id_password'))
7
8      user_input.send_keys('admin')
9      pass_input.send_keys(os.environ.get('
           DJANGO_SUPERUSER_PASSWORD', 'admin'))
10     pass_input.send_keys(Keys.ENTER)
11
12     # assert Alice is logged in as admin
13     time.sleep(2)
14     self.assertIn('Log Out', self.browser.page_source)
15
16     # She opens the attendance for Sunday School service
           to start logging students
17     self.wait_for(lambda: self.browser.
           find_element_by_link_text('Attendance')).click()
18     self.wait_for(lambda: self.browser.
           find_element_by_id('id_sunday_school')).submit()
19
20     # Mark comes in and places his ID on the RFID card
           reader and his attendance his counted
21     card_input: WebElement = self.wait_for(lambda: self.
           browser.find_element_by_id('card_number'))
22     card_input.send_keys(self.mark.card.number)
23     card_input.send_keys(Keys.ENTER)
24
25     # assert Mark's attendance is recorded
26     time.sleep(2)
27     self.assertIn('Attendance Count: 1', self.browser.
           page_source)
```

Listing 5.1: Functional test to check students attendance on the go.

5.2 Unit Testing

The application was written using Test Driven Development (TDD) [3]. The strategy used was the red-green-refactor cycle. Red refers to the first implementation of a failing test, green indicates the need to write code that would successfully pass the test and refactor when needed. This cycle improves the code quality and productivity [4].

In a Microsoft case study, TDD was found to double the code quality while increasing the time of writing tests by 15% [5].

5.2.1 Framework

A feature with Django is that through 'django.test.TestCase' class, the unit tests were run in their own environment with a specific test database that would be generated when tests run and be destroyed once that tests are complete.

This feature ensures that any database records that would be created, would not alter the current database environment while also aiding with the process of running the tests repeatedly without expecting any database related issues that may arise due to duplication of records.

5.2.2 Test Results

```
1 Creating test database for alias 'default'...
2 System check identified no issues (0 silenced).
3 ..........................................
4 ----------------------------------------------------------------------
5 Ran 42 tests in 15.929s
6
7 OK
8 Destroying test database for alias 'default'...
```

Listing 5.2: Test Results run on the terminal.

In total, 41 unit tests were created for models, services and selectors and 1 functional test was created for having attendance. All of the tests were successful.

5.2.3 Examples

```
1  class AttendanceTestCase(BaseTestCase):
2
3      def test_adding_same_student_to_attendance
4      _returns_integrity_error(self):
5          AttendanceService().add_attendance_by_card(card=
               self.card1, for_sunday_school=True)
6          with self.assertRaises(IntegrityError):
7              AttendanceService().add_attendance_by_card(
                   card=self.card1, for_sunday_school=True)
8
9      def test_adding_attendance_for_student
10     _increases_their_multiplier(self):
11         AttendanceService().add_attendance_by_card(card=
               self.card1, for_sunday_school=True)
12         AttendanceService().add_attendance_by_card(card=
               self.card1, for_bible_study=True)
13         AttendanceService().add_attendance_by_card(card=
               self.card1)
14
15         multiplier = MultiplierSelector().get_multiplier
               (student=self.card1.holder)
16         self.assertEqual(multiplier.sunday_school, 1)
17         self.assertEqual(multiplier.bible_study, 1)
18         self.assertEqual(multiplier.summer_club, 1)
19
20     def test_money_is_added_as_bonus
21     _for_attendance_with_card(self):
22         AttendanceService().add_attendance_by_card(self.
               card1, for_sunday_school=True)
23         bonus = BonusSelector().get_weekly_bonus()
24         self.assertEqual(self.card1.balance, bonus)
25
26     def test_no_money_is_added_when_student
27     _has_attendance_by_name(self):
28         AttendanceService().add_attendance_by_student(
               self.card1.holder, for_sunday_school=True)
29         self.assertEqual(self.card1.balance, 0)
```

Listing 5.3: Attendance service unit tests.

```
1  class CardTestCase(BaseTestCase):
2
3      def test_card_number_is_not_duplicate_in_db(self):
4          CardService.create_card(holder=self.student,
               number=1234)
5          with self.assertRaises(IntegrityError):
6              CardService.create_card(holder=self.student2
                   , number=1234)
7
8      def test_deposit_money_works(self):
9          card = CardService().create_card(holder=self.
               student, number=1234)
10         CardService().add_money(card=card, amount=50)
11         self.assertEqual(card.balance, 50)
12
13         CardService().add_money(card=card, amount=50)
14         self.assertEqual(card.balance, 100)
15
16     def test_withdraw_money_works(self):
17         card = CardService().create_card(holder=self.
               student, number=1234)
18         CardService().add_money(card=card, amount=50)
19         CardService().take_money(card=card, amount=20)
20
21         self.assertEqual(card.balance, 30)
22
23     def test_cannot_deposit_in_decimals(self):
24         card = CardService().create_card(holder=self.
               student, number=1234)
25         with self.assertRaisesMessage(ValueError, '
               Cannot deposit in decimals.'):
26             CardService().deposit_money(card=card,
                   amount=9.7)
27
28     def test_cannot_deposit_negative_values(self):
29         card = CardService().create_card(holder=self.
               student, number=1234)
30         with self.assertRaisesMessage(ValidationError, '
               Negative values are not accepted.'):
31             CardService().deposit_money(card=card,
                   amount=-20)
```

Listing 5.4: Card service unit tests.

```
1  class BonusTestCase(BaseTestCase):
2
3      def test_bonus_is_created(self):
4          self.assertTrue(not BonusService().record_exists
               ())
5          BonusService().add_bonus_values(week=5, month
               =10, quarter=50)
6          self.assertTrue(BonusService().record_exists())
7
8      def test_bonus_is_updated(self):
9          BonusService().add_bonus_values(week=5, month
               =10, quarter=50)
10         BonusService().add_bonus_values(week=10, month
               =20, quarter=50)
11         weekly_bonus = BonusSelector().get_weekly_bonus
               ()
12         self.assertEqual(weekly_bonus, 10)
13
14     def
           test_initial_bonus_gets_added_if_no_record_exists
           (self):
15         BonusService().add_bonus_values_if_not_exists(
               week=5, month=10, quarter=50)
16         weekly_bonus = BonusSelector().get_weekly_bonus
               ()
17         self.assertEqual(weekly_bonus, 5)
18
19     def test_initial_values_do_not_change_existing_ones(
           self):
20         BonusService().add_bonus_values(week=5, month
               =10, quarter=50)
21         BonusService().add_bonus_values_if_not_exists(
               week=10, month=25, quarter=50)
22         bonus = BonusSelector().get_bonus_values()
23         self.assertTrue(bonus.WEEK == 5 and bonus.MONTH
               == 10 and bonus.QUARTER == 50)
```

Listing 5.5: Bonus service unit tests.

5.3 Continuous Integration

The application is run through docker. When running the docker-compose file [1] and building the application image, in the Dockerfile, python will run the unit & functional tests after installing all dependencies. If any of the tests fail, the application will not start.

```
FROM python:3.8-slim

ARG APP

RUN mkdir -p $APP
WORKDIR $APP

ENV PYTHONDONTWRITEBYTECODE 1
ENV PYTHONUNBUFFERED 1

COPY . $APP
RUN pip3 install -r requirements.txt && \
    python3 manage.py collectstatic --noinput && \
    python3 manage.py test

EXPOSE 80
ENTRYPOINT \
    ["/app/deploy/docker/app/docker-entrypoint.sh"]
```

Listing 5.6: Dockerfile for building the application.

The application is then uploaded and installed on a remote server in GCP using docker commands.

[1] I prepared this docker-compose file and is included in the discovery.

Chapter 6

Discussion

The application is currently deployed and running in production since early October of 2021. There are 89 students enrolled to the system with over 500 attendance records by the time of writing this paper.

The feedback I received has been extremely positive and both the client as well as the students are enjoying the interaction with the system. I have managed to fulfil almost every requirement.

For future work, I plan on completing the missing requirements which are:

- Send a report after every attendance event.
- The system supports having different types of bank cards.
- The students get their grade automatically updated after every year.
- Users who are no longer students can be saved on the system as Alumni.

I also plan on enhancing the UI to make it more interactive and user friendly. The installation of the application in the server is a semi-automated step. I plan on adding ansible playbooks that would automate for me the deployment aspect of the application and would automatically ship the code to my compute instance in GCP and start the application.

Chapter 7

Conclusion

The aim of the thesis has been fulfilled successfully. I conducted a research of the available competitors on the market. I analysed the requirements and use cases for the system which helped me gain a deeper understanding of the client's needs.

I was able to build the system using a three layered architecture and have a consistent coding style throughout the application. The application was thoroughly tested using TDD principals and is deployed and running on a compute instance in GCP.

The client has received the application and it is currently being used by the organization where they have over 89 students enrolled.

Bibliography

[1] Planning, S. The economic impacts of inadequate infrastructure for software testing. *National Institute of Standards and Technology*, 2002.

[2] Myers, G. J.; Sandler, C.; et al. *The art of software testing*. John Wiley & Sons, 2011.

[3] Percival, H. *Test-driven development with Python: obey the testing goat: using Django, Selenium, and JavaScript*. "O'Reilly Media, Inc.", 2014.

[4] Kumar, P. R.; Raju, G.; et al. An External Quality Supporting Test-Driven Development of Web Service Choreographies. *International Journal of Computer Applications*, volume 975, 2014: p. 8887.

[5] Bhat, T.; Nagappan, N. Evaluating the efficacy of test-driven development: industrial case studies. In *Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering*, 2006, pp. 356–363.

Appendix A

Acronyms

CRM Customer Relationship Management

SAAS Software as a service

UI User Interface

GCP Google Cloud Platform

TDD Test Driven Development

MVT Model-View-Template

HTML Hypertext Markup Language

CSS Cascading Style Sheets

Appendix B

Application Snapshots

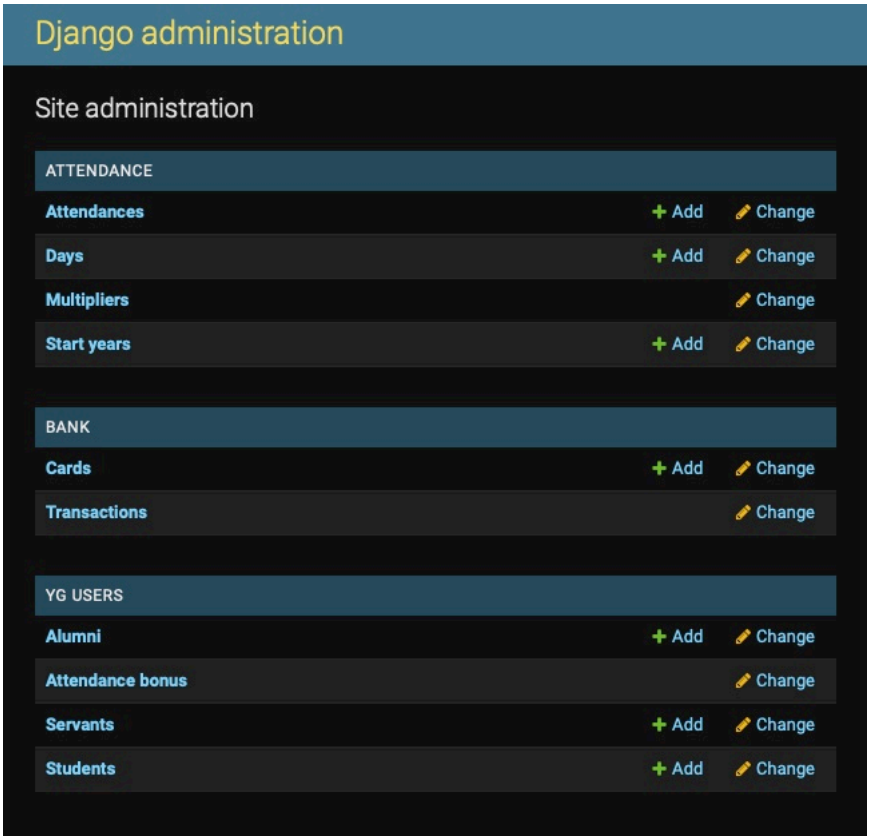

Figure B.1: Admin dashboard where you can add or change any information in the application.

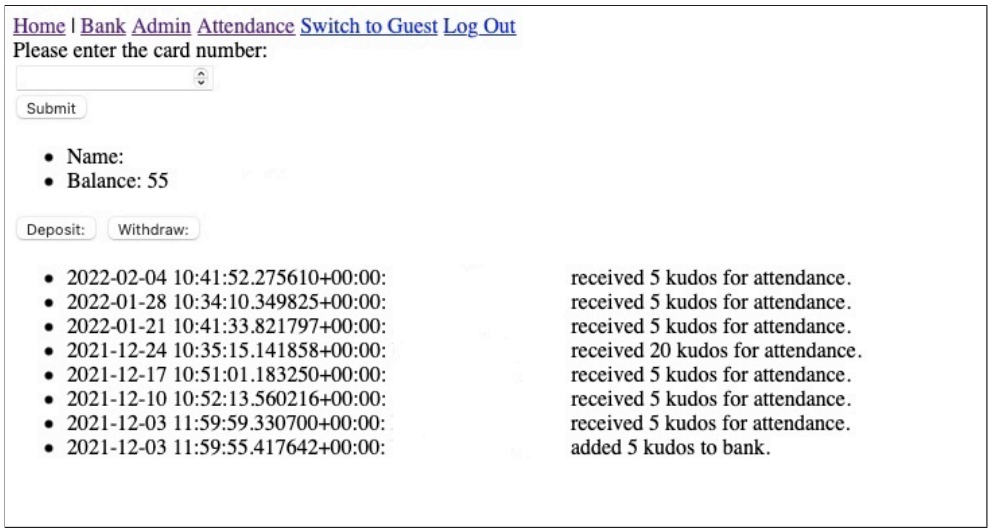

Figure B.2: By placing the card number of a student, the admin is able to view their balance and deposit/withdraw virtual currency.

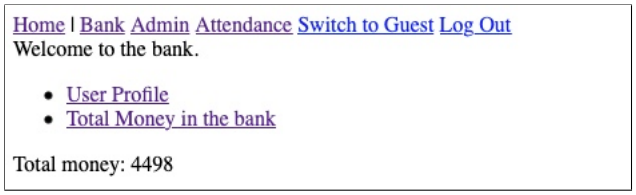

Figure B.3: The admin is able to view the total money which all students have combined.

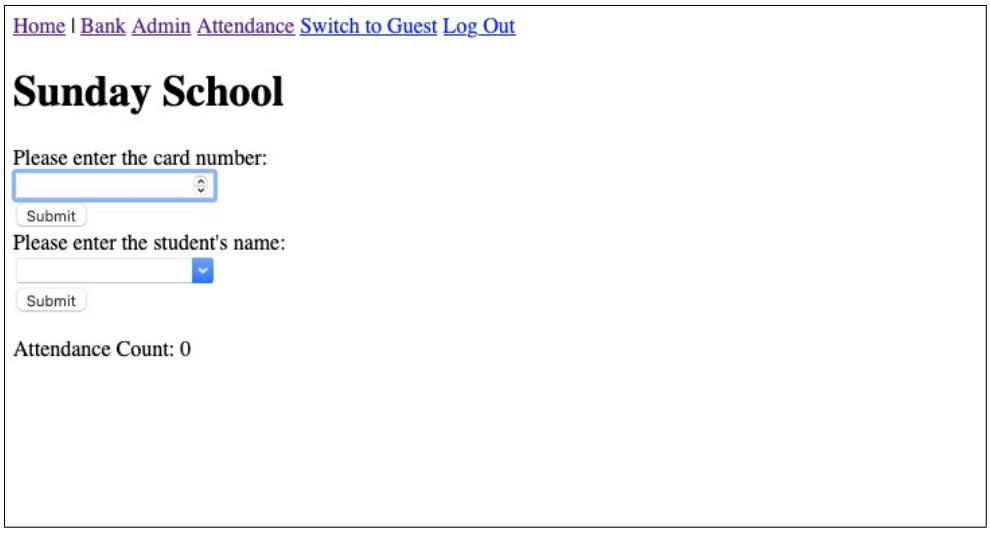

Figure B.4: Attendance available for Sunday School service where the admin can add the student's card number or name.

Appendix C

Contents of enclosed CD

- `README.md` ..... the file with application description
- `config` ..... the config sub-application directory
- `deploy` ..... the directory with all deployment related material
- `requirements.txt` ..... file with all of python3 dependencies
- `yg_system` ..... the yg_system sub-application directory
  - `settings.py` ..... the settings file for the application
- `attendance` ..... the attendance sub-application directory
- `functional_tests` ..... directory with functional tests
- `scripts` ..... directory which includes all scripts used for the application
- `bank` ..... the bank sub-application directory
- `manage.py` ..... django executable file
- `static` ..... directory with all of the static files present
- `thesis-master` ..... the thesis text directory
  - `thesis.pdf` ..... the thesis text in pdf format
  - `latex-source-code` ..... the thesis latex source code