|  |  |  |  |
| --- | --- | --- | --- |
| **CS102** | **Spring 2023/24** | Project Group | 3I |
| Instructor: | **Uğur Güdükbay** |
| Assistant: | **Melih Coşğun** |

|  |  |  |
| --- | --- | --- |
| **Criteria** | **TA/Grader** | **Instructor** |
| Presentation |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| Overall |  |  |

University Sync

Prototype

**Arda Ege Erdoğan, Atilla Akkuş, Berkay Demirçin, Muhammad Musa Qureshi, Zaeem Masood Sheikh**

|  |
| --- |
| **Report Type**  **Detailed Design Report**  **27 April 2024** |

# Introduction

UniversitySync is a desktop app that aims to provide students with a social platform where they can make friends, keep track of the complex campus life, and make their overall university experience easier and better. Our application implements a robust database utilizing modern technologies.

Our app allows students to make posts, message others, and follow events to aid in socializing. In addition, students are able to exchange dormitory rooms, review and see these reviews about dormitories and cafeterias to make their university experience more convenient.

# Details

## The Final Implementation

Starting off, we were able to deliver on almost all the designed features we planned from the beginning. The only change we made was to substitute the admin panel mentioned in the requirements report with the MySQL workbench which allows an administrator to maintain and manipulate the database. More specifically, this is only used for things that cannot be done within the app such as adding new dormitories and cafeterias. Moreover, this implementation allowed us to implement a contact us page for the users to communicate with the administrator by sending messages through the database. As mentioned above, the rest of our features were either all wholly implemented or substituted with a very close alternative, such as the sort/filter menu we planned in the UI reports being replaced with sort/filter buttons in the final implementation. Below are some screenshots from the final implementation of our project.

The login page of our project is pictured below.

A screenshot of a computer login

Description automatically generated

The register page for new users is pictured below.

A screen shot of a computer screen

Description automatically generated

This is the Home page in the final version, giving the user many options to interact with while maintaining visibility. Note that some recents posts are being displayed.

A screenshot of a computer

Description automatically generated

After clicking on “Reply” under a posts, the below screen is shown with all the previous replies and an option to type a new reply.

A screenshot of a computer

Description automatically generated

Below is the pop-up to create a new forum post under the “Forum” tab, which displays the same posts under the Home tab, except this tab displays all the forum posts as opposed to only the recent ones.

A screenshot of a computer

Description automatically generated

The food tab, showing an example of the sort buttons mentioned earlier, note that just like the rest of the content in the program, the cafeterias and the accompanying information are pulled from the database.

A screenshot of a computer

Description automatically generated

An example image of a cafeteria page, in this case Marmara Cafeteria from the above image.

A screenshot of a restaurant

Description automatically generated

Pictured below is the review page of Marmara Cafeteria, note that the page will look the same for any other cafeteria or dormitory, except the image, rating and amount of reviews will change based on which specific instance you’re reviewing.

A screenshot of a computer

Description automatically generated

The dormitory transfer posts page is pictured below.

A screenshot of a computer

Description automatically generated

When the user first clicks the Social tab from the top pane, or when they click on My Friends under the Social tab, the friends list is pictured below.

A screenshot of a computer

Description automatically generated

After the user clicks on the Social tab and then goes to “All Messages”, they will be brought to the page below. Notice that “Deniz Uslu” here is highlighted, as their chat pane is open.

A screenshot of a computer

Description automatically generated

Below is the events tab, where the events that the current user instance is following are displayed.

A screenshot of a computer

Description automatically generated

After the user clicks the “Add” button in the events pane on the bottom right of the screen, the below screen pops up for the user to post a new event.

A screenshot of a computer

Description automatically generated

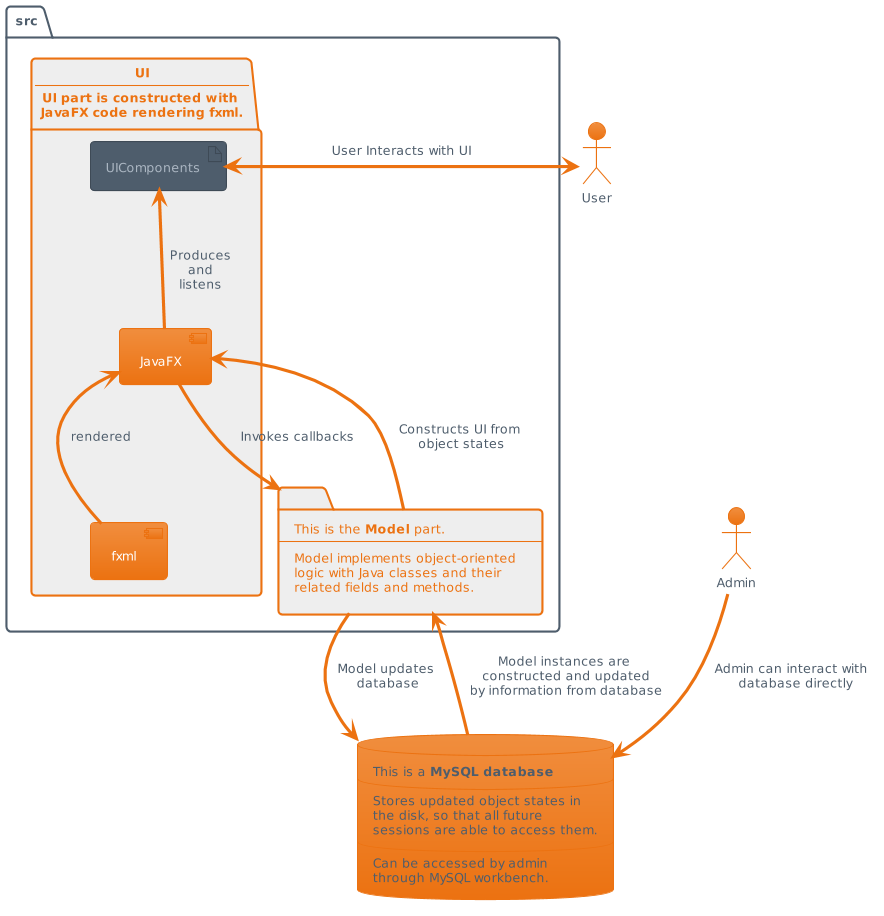
An image of a user profile is shown below, currently in the “Edit Profile” tab. The user info along with the option to edit the profile picture can also be seen.

A screenshot of a computer

Description automatically generated

## System Overview

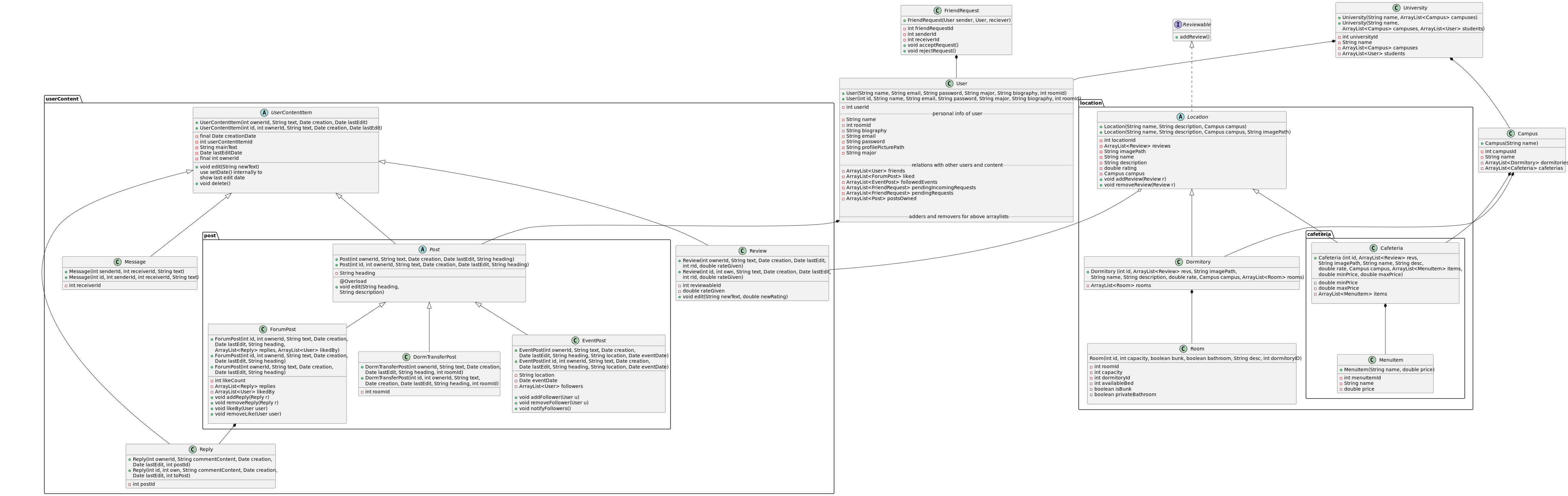
Our application is a desktop app that implements a MySQL relational database, Java classes and Maven to handle dependencies. In addition, we use JavaFX for implementation of the UI, and CSS to improve the overall look of the UI. We use Model View Controller (MVC) pattern in our application. As can be seen in the diagram below, the database provides information for instantiating Java objects.



For the user interface, JavaFX renders fxml structure with object states to construct the view, i.e., UI components with appropriate content. The model part synchronizes object states with database and updates both, with UI callback functions triggered by user. The users can access features that are only visible through the UI. Direct access to the database is only possible for the administrator, allowing them to interact with database using the MySQL workbench.

## Core Design Details

To be better organized in our implementation, our project utilized an extensive UML diagram for our model classes, seen below. To design our UML diagram seen below, we used PlantUML [1]. Note that DAO and controller classes are excluded from the UML diagram as they do not follow conventional OOP relationships.



A group of text on a white background

Description automatically generated

This diagram utilizes universal UML arrow relationships, rectangles encapsulating classes denote packages, and more info about how to interpret the UML can be found in the legend above. Note that some methods and fields are inherited, and some methods (such as getters and setters) are not included to improve readability. All classes are public unless stated otherwise.

Our program utilizes the MVC (Model View Controller) design pattern, which helped with connecting the UI, database, and Java code elements together.

Separating the classes into packages allows UniversitySync to be modular, expandable, and easier to maintain. The location package contains all logic retaining to locations, such as dormitories and cafeterias. The userContent package contains all content that the user of the app is be able to post. Classes not part of a package have more general purposes such as containing user information and providing the framework for expandability.

Each class whose objects can be added or removed during the runtime have two consturctors. One has no id parameter. It adds the object to database by calling the associated Dao method, Dao method handles the insertion and returns the auto-assigned id from the database (primary key). Then the constructor completes the construction by assigning ID to Java object instance. This constructor is used when a new data is to be added to database, for instance adding a review, registering a user. Other constructor is the regular constructor that is used when a existing data in database is to be pulled to a Java object instance.

All setters do the necessary validation internally, and some are also to be utilized by other methods. For example, the edit() method in the UserContentItem abstract class calls setDate() method internally to set the last edited date information to ‘now’, i.e., the computer time when the method is called. The setDate() method uses Java’s Date and TimeStamp classes, as mentioned in the diagram.

In our app, we offer filtering and sorting functionalities for dormitories, cafeterias, posts etc. according to multiple properties. The sorting (according to price, rating, recency) and filtering (according to who posted the ForumPost, following status of EventPost etc.) are done in Dao classes while accessing these data using ORDER BY or WHERE SQL keywords.

As for the Model classes, User class handles user info, such as their password and update these fields with the help of a setter. In addition, void addFriend(User friend) method does the validation and add the parameter User as friend. void removeFriend(User friend) removes the friend from the associated ArrayList if that User exists as friend. Similar functionality is implemented for handling friendRequests. Classes related with User include FriendRequest. FriendRequest contains two sides of the request.

UserContentItem abstract class handles all user-made content including Reply, Review, Message and another abstract class Post. The reason for this class is to ensure that all the common behaviors of user content are stored in one class to reduce duplicate code and create a more maintainable program via the use of inheritance. Some common behaviors include the void edit() method, which allows users to edit the content of their post and updates the lastEditDate according to the system time using setDate(), and void delete() which deletes the content item. Message provides opportunity for one-to-one communication in the app and stores two users involving it and the text of the message. Replies are UserContentItem's that have a ForumPost associated with them. In addition, the Review class, derived from UserContentItem allows users to leave reviews on classes implementing the Reviewable interface. This class is designed in a way to ensure that all reviews follow a general template, and it’s objects will hold the rating given by the user along with the review text, in the inherited mainText variable.

The Post abstract class aims to accomplish the same goals but in a narrower scope for ForumPost, DormTransferPost and EventPost classes. EventPost class, derived from the Post abstract class retains information regarding a specific event, such as the event’s location and time. User class objects are able to follow these events by using the void addFollower(User u) method, removed by void removeFollower(User u). DormTransferPost class holds tha data for the posts that are seeking a new Dormitory to reside in. In addition to functionality in Post, it includes the room type that is associated with the post. ForumPost class holds general purpose posts that are likable and replyable. Who liked the ForumPost and replies to ForumPost are stored in the instance.

University and Campus classes exist mainly to ensure that our app stays expandable in the case that it may be extended to include multiple universities and campuses. As with all other classes, this class will also house getters and setters.

The Reviewable interface is implemented for expandability reasons, as other classes may be changed or added to implement this interface. Currently, Location implements this interface. Location abstract class, similar to the UserContentItem abstract class, aims to reduce duplicate code and help UniversitySync to be more maintainable and expandable. This class implements Reviewable interface. This class holds general information regarding to locations on campus, such as cafeterias and dormitories. In addition addReview(Review r) and removeReview(Review r) adds, removes reviews to these locations respectively. This class also holds reviews given by users. More specifically, the Dormitory and Room classes aim to ensure a more unified design across all dormitories and rooms. Cafeteria and MenuItem classes work in the same manner, making the designs more uniform, thus making the app less error prone.

In controller classes, the SessionManager controller class is one of the most important ones, and it initializes and keeps track of the current user that is logged in to the app. The personalized content is displayed by keeping track of the logged in user through SessionManager.

The LoginController is used to enter what user is passed to SessionManager to keep track of. On the HomePage, we have a controller class HomePageController that initializes and displays the most 3 recent forum posts. Each forum post is assigned a controller class named PostController. This controller gets the data from the database using a ForumPostDao method, and initializes all UI components accordingly. Furthermore, all UI elements, such as the like and comment button, have methods assigned in the controller class that are triggered when the button is pressed. The methods use ForumPostDao methods to verify conditions and update the database if necessary.

The ForumPost tab has a similar implementation for its posts, but has a separate controller class ForumController, which apart from displaying all the forum posts, has methods to facilitate addition of new posts on the app, and ordering the posts according to the current user’s friends. The methods include displayMakePostPopup which loads a template to add a new post and allows user to enter the information and post it. It also has a method switchPosts that allows to switch between all the posts and posts made by friends.

Overall, in most of the other pages as well, there is a system of hierarchy of controller classes. Like in the Food tab, the controller assigned to that page is the CafeteriaController class, which has methods to initialize the page if the tab is clicked, and methods to order the cafeterias according to their ratings and prices. But for each cafeteria displayed on the page, the UI element has a separate controller class CafeteriaNameController. This is implemented to access which cafeteria’s UI elements are accessed by the user, and switch to the next page which displays the information for a particular café. The separate page has a different controller class CafeteriaDetailController, which displays all the details regarding the café and has the option to switch to the reviews page, which again has a separate controller ReviewPageController. It has methods to initialize and display all previous reviews of that location and has methods to facilitate the addition of new reviews. So in total, for each cafeteria, there are 3 controllers assigned to it, and there is one controller, CafeteriaController, that is assigned for the whole Food page.

The implementation is similar for the Dormitory info page as well, where there is DormitoryController class for the whole page, and DormitoryNameController, DormitoryDetailController and, ReviewPageController class for each dormitory being displayed.

In the Socials tab, we have controller classes for each sub tab we have, for example, SocialAllMessagesController. In each such controller, we render the required elements, and assign each element its own controller. Such as in the All Messages tab, the SocialAllMessagesController assigns SocialMessageOneUserController to each user that the current user has a message with. This controller allows displaying of the messages between the two user and assigns a SocialMessageBoxController to the chat when it is displayed, which prints all the messages between the users, and assigns a SocialEachTextController to each text box. The SocialMessageBoxController uses a the DAO methods to get the messages between the users, and uses a for loop to display the messages and assign the SocialEachTextController class. It also considers if the message is from the current user or other user, and for each case makes the necessary adjustment in the UI to display it.

Along with the 3 main components of the MVC pattern, we included a DAO folder of classes to manage database access (CRUD) operations in an organized way. Classes have static methods that generate queries based on the input, send queries to MySQL database, and returns data from the database if desired. These methods prepare statements using PreparedStatement and assign parameters into that statement ensuring secure access. If any data is returned as ResultSet, Dao methods process this data and returns it as more usable ArrayList of desired type of the object. This folder includes several DAO classes for each of the important model classes. Firstly to handle the database connection, DBConnectionManager class has a single method initializeConnection. It accepts connection url and credentials as parameters and creates a connection to database. This connection is stored as the static variable of type Connection. Other Dao classes access this connection by DBConnectionManager.getConnection()

UserDao manages database access related to User type of object. Adding, updating, deleting, accessing users are implemented. Similar functionalities for FriendRequest and friendship are also done through this Dao class. Regarding login and registration process, methods for authentication and ensuring email uniquity are also implemented. Getting Major's available for User and sending a request to admin are also implemented in this Dao class.

CampusDao, along with the Campus model class, this class ensures expandability. It currently has one method for retrieving the data for a campus based on its database ID given as parameter.

CafeteriaDao class manages database access for Cafeteria related data. It allows getting all cafeterias by default order, by price order, or by rating order. Also, a method allowing to get a specific Cafeteria based on its database ID is implemented. Getting MenuItems of a specific Cafeteria is also implemented. Adding or removing or updating a Cafeteria is not implemented since it shouldn't be done during the runtime.

DormitoryDao class manages database access for Dormitory related data. It allows getting all dormitories by default order, or by rating order. Also, a method allowing to get a specific Dormitory based on its database ID is implemented. Getting Room types of a specific Dormitory or all dormitories is also implemented. Adding or removing or updating a Dormitory is not implemented since it shouldn't be done during the runtime.

ReviewDao manages database access for Reviews that are given to a Reviewable (i. e. Dormitory and Cafeteria currently). Adding, removing, deleting, checking existence of Review and accessing all reviews of a particular Reviewable are implemented.

DormTransferPostDao manages database access related to DormTransferPost's, including adding, deleting, updating and accessing (a particular one based on ID or all). Also, it has a method for getting DormTransferPost's that are seeking for a specific room type, which is functional for filtering posts.

EventPostDao manages database access related to EventPost's, including adding, deleting, updating, and accessing (a particular one based on ID or all). This class also manages database access related to followers of events (Adding, removing, checking follower to a particular event, getting events followed by a particular User).

ForumPostDao DAO class manages ForumPost related access, which are adding, deleting, editing, getting (all, by a specific user, by ID, by friends of a specific user or sorted by recency). Further, access related to liking a ForumPost (adding, removing, getting liked posts) and replying to a ForumPost (adding, removing, getting replies) are sustained via this Dao class.

MessageDao manages database access related to Message and messaging functionality. Adding a message, getting messages between two specific User’s, and getting all User's that a particular User has messaged with are implemented.

## Splitting Up the Work

Zaeem Masood Sheikh implemented most of the UI using JavaFX and helped connect these UI elements with the Java code using Controller classes mentioned earlier and has helped with every controller class. Muhammad Musa Qureshi focused on implementing the Controller classes, and also helped figure out how we could utilize the MVC pattern. Berkay Demirçin helped implement DAO classes such as UserDao, ForumPostDao and Controller classes like PostController. Atilla Akkuş helped with DAO classes such as MessageDao and Controller classes while also helping with the MVC design. Arda Ege Erdoğan created the initial database tables, helped with MVC and helped other members with database related problems, while also implementing some DAO classes like ReviewDao and DormitroyDao. The model classes were implemented by Atilla Akkuş and Berkay Demirçin, although other members made changes as necessary (ie. adding new constructors and adding validation).

Splitting up the work allowed each member to play to their strengths and ensures even contribution from each member.

# Summary & Conclusions

The most difficult part of the project work was trying to evenly distribute the work, as every member had different schedules to work and different interests (ie. UI vs database). As a group, we disliked the fact that there was little mention of the expectations of the project. Therefore, if we were to develop another app, we would try to first try to figure out how many features we actually need, and plan accordingly, and start the implementation earlier as this project took over 40 hours of work from each individual member, not counting the time it took to learn new technologies. Overall, due to the immense effort we all put into the project, we are all happy with what we’ve achieved, and this project showed us that even tasks we deemed impossible at the start are plausible with enough effort.

Our app follows best practices to avoid problems and utilize many modern technologies. The relational database implementation with MySQL, and MVC design pattern by using JavaFX minimizes problems that may arise during implementation. An extensive UML design helps with implementation and clearly communicate the role of each class, and how they may interact with each other. Along with this, an even distribution of responsibilities ensured that all members of our team worked most effectively and efficiently. Overall, these robust design choices mean that UniversitySync is easy to maintain and easily expandable.

# References

1. Roques, A. (2009) *PlantUML*. Retrieved from https://plantuml.com/ Last visited: 27.04.2024