CS 319
OBJECT ORIENTED SOFTWARE ENGINEERING
DESIGN REPORT
PEER-REVIEW

GROUP 1D

MEMBERS:

Eylül Çağlar (Group Leader) 21703949

İsmet Alp Eren 21703786

Kaan Ateşel 21703694

Kerem Alemdar 21702133

Muharrem Berk Yıldız 21802492

Instructor: Eray Tüzün

Teaching Assistant(s): Elgun Jabrayilzade, Erdem Tuna

1. Introduction

1.1 Purpose Of The System

Pire that will be designed is a digital peer review application which aims to help Bilkent University students and instructors for their Computer Science courses and projects. Differently from other peer review applications “Pire” offers many additional features that let users to easily access groups, review group tasks, grade other groups projects and at the end of the course see the grades of their projects given by instructors or TAs.

1.2 Design Goals

Following subsections show the design goals that are being considered in the making of the Pire application in the light of nonfunctional requirements.

1.2.1 Nonfunctional Design Goals

Response Time

This Pire (Peer Review) application that was designed for Bilkent University Computer Science courses is a database dependent application. When database factor is considered, response time is important to maintain application efficiently. Therefore, the system is designed to have a maximum response time of less than 2 seconds. This ensures to keep the users’ attention in the application.

Extensibility

Pire application will be implemented by considering the real life cases. It means that application programmers take into account that users may want new

features in the application. Thus, Pire application implementation will give a chance to add new features easily by just adding new things such as subclasses or new methods. This type of implementation will give extensibility quality to the implementation.

Reliability

The application is designed to behave in an expected logical way, data loss and delay is not much possible. The unintended system behaviors are minimized by making the system 99% reliable since a single unintended behavior might disrupt the process of the application and flow of the project.

2. Subsystem Decomposition

Our system is divided into four subsystems. The Application View subsystem handles user interfaces. The Application Database Controller subsystem handles data flows between our Database subsystem and our application. Application Controller subsystem handles in-app functions.

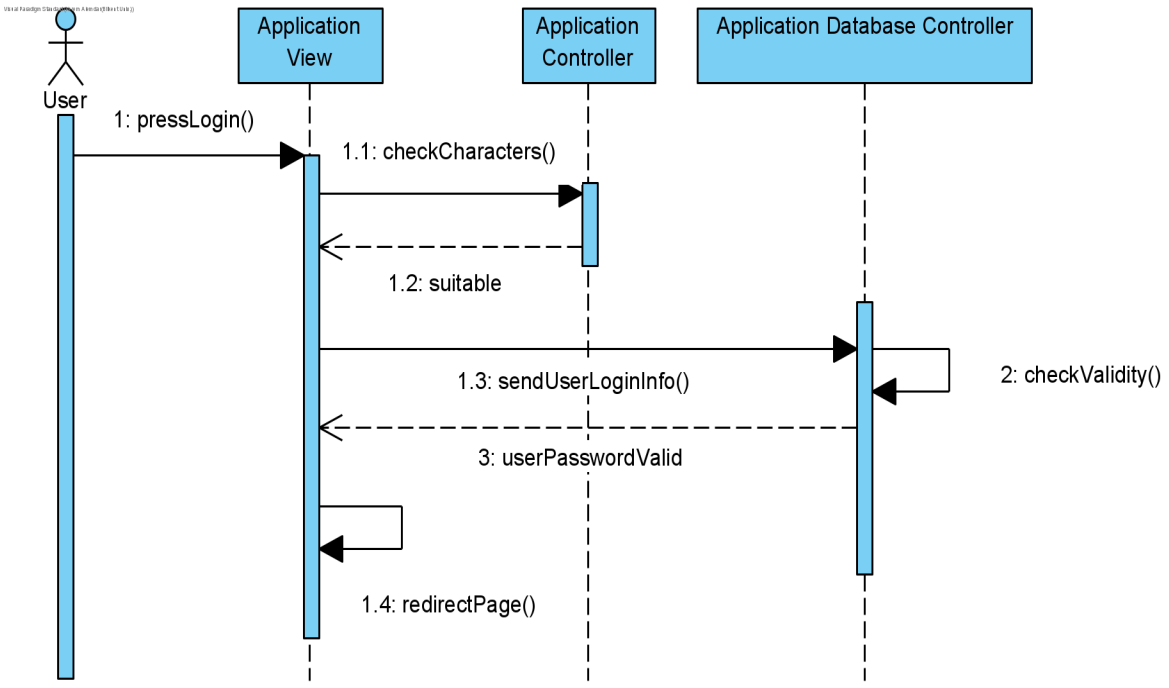

Figure 1 : Sequence diagram for login

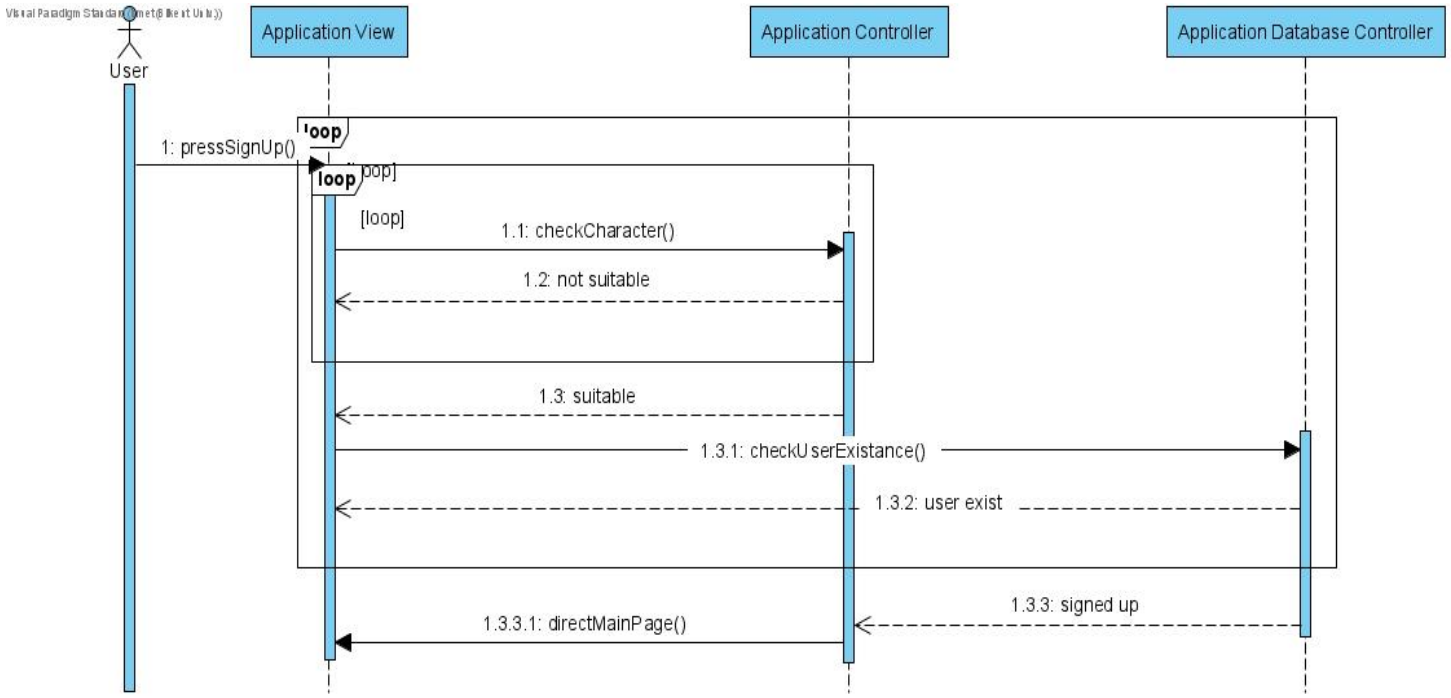

Figure 2: Sequence diagram for Sign Up

2.1 Application Database Controller

Pire application system decomposed into three subsystems. First subsystem is the application View. This system is responsible for handling user user events such as sending messages or clicking a button. This subsystem calls the Application Controller methods which is responsible for sending requests to the backend server. Last subsystem is called the Database Controller subsystem. This is a backend server of the application which communicates with the database server. It saves data to the database and also retview data from the database and sends them to the application controller.

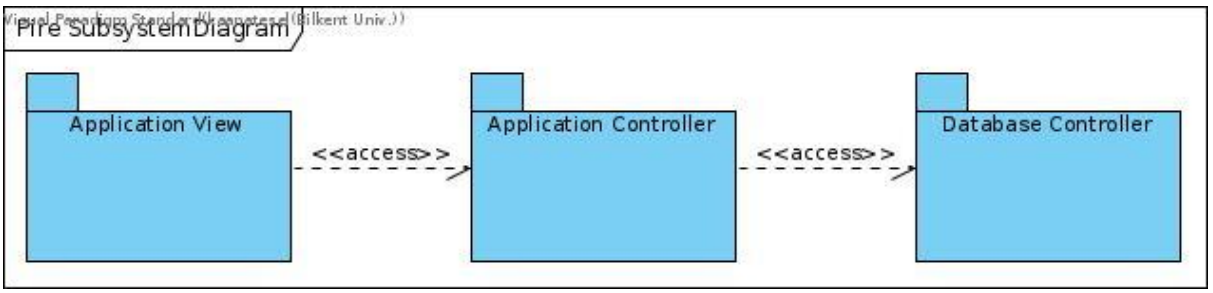

Figure 3: UML representation of the subsystem decomposition

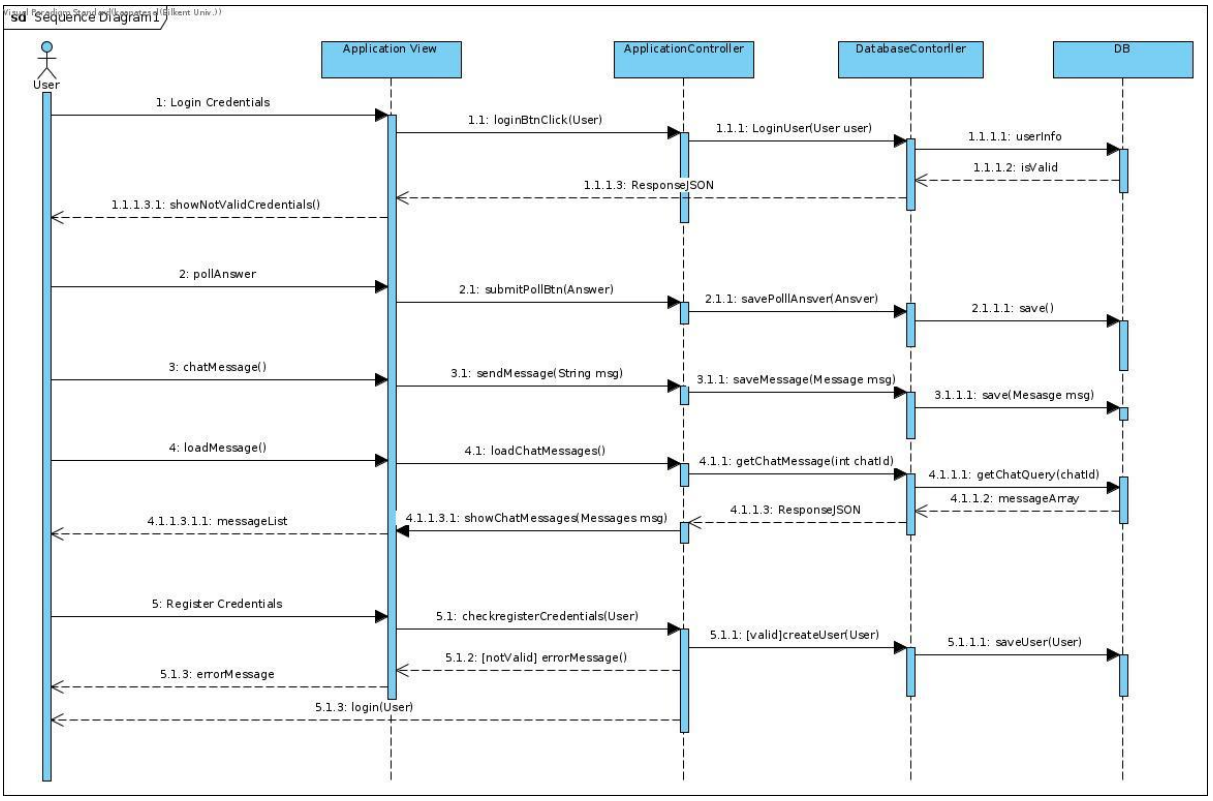

Figure 4: Example Sequence diagram of system.

3. Hardware / Software Mapping

The application is a single Java program that can be used where Java Virtual Machine is installed and where the internet is connected. So, any computer with JVM and internet connection can use this peer review application easily.

4. Access Control

The system has to access the database to get and save data and information. The users have to access the database to login the system and have to keep accessing it while using the application. Program keeps the information such as school id, e-mail, phone number if needed and project details. Since documents and personal information are private and should be kept safe, database security is an important issue for the system.

The system will hashes and encryption all necessary and private data and information to keep users safe and unreachable by both users and non-users.

5. Persistent Data Management

The system needs to store data about:

- User information
- Group information
- Global information

5.1 User information

User information will stored in order to provide following functionalities:

- Authenticate user when tried to login
- Show the name of the user when send a message
- Show the name of the user when post a review.
- Show the user's personal information like email, surname, second name.
- Display user’s friends and incoming friend requests.
- Sending and receiving group invitations.
- Check user type and permissions

- Send and show personal messages among users

Therefore the system will be storing following datas:

- User name (firstname, second name)
- User surname
- User email address
- User password
- User ID
- User's friend list.
- Personal Messages.

5.2 Group information

Group information will stored in order to provide following functionalities:

- Display pdf documents of the group members.
- Show the group members.
- Allocate and display group tasks, deadline and assigned person.
- Check prerequisites of the group for new members.
- Display group polls and its results.
- Display Group schedules assigned by the group leader.
- Provide document information like file url, contributors, artifact type and score.
- Show the reviews of other students.
- Display reviews and grades of instructors and TAs.

Therefore the system will be storing following datas:

- Group name.
- Member IDs and Group Leader.
- Group Tasks

- Group Task Templates
- Prerequisites, its type, condition, and strictness.
- Document url, grade and reviews.

5.3 Global information

Global information will stored in order to provide following functionalities:

- Show global chat where messages and their names are shown.
- Show global polls started by the instructor or TA including its template and answers.
- Show schedules date and their message.

Therefore the system will be storing following datas:

- Chat messages.
- Global polls templates and result values
- Schedule date, content.

6. Subsystem Services

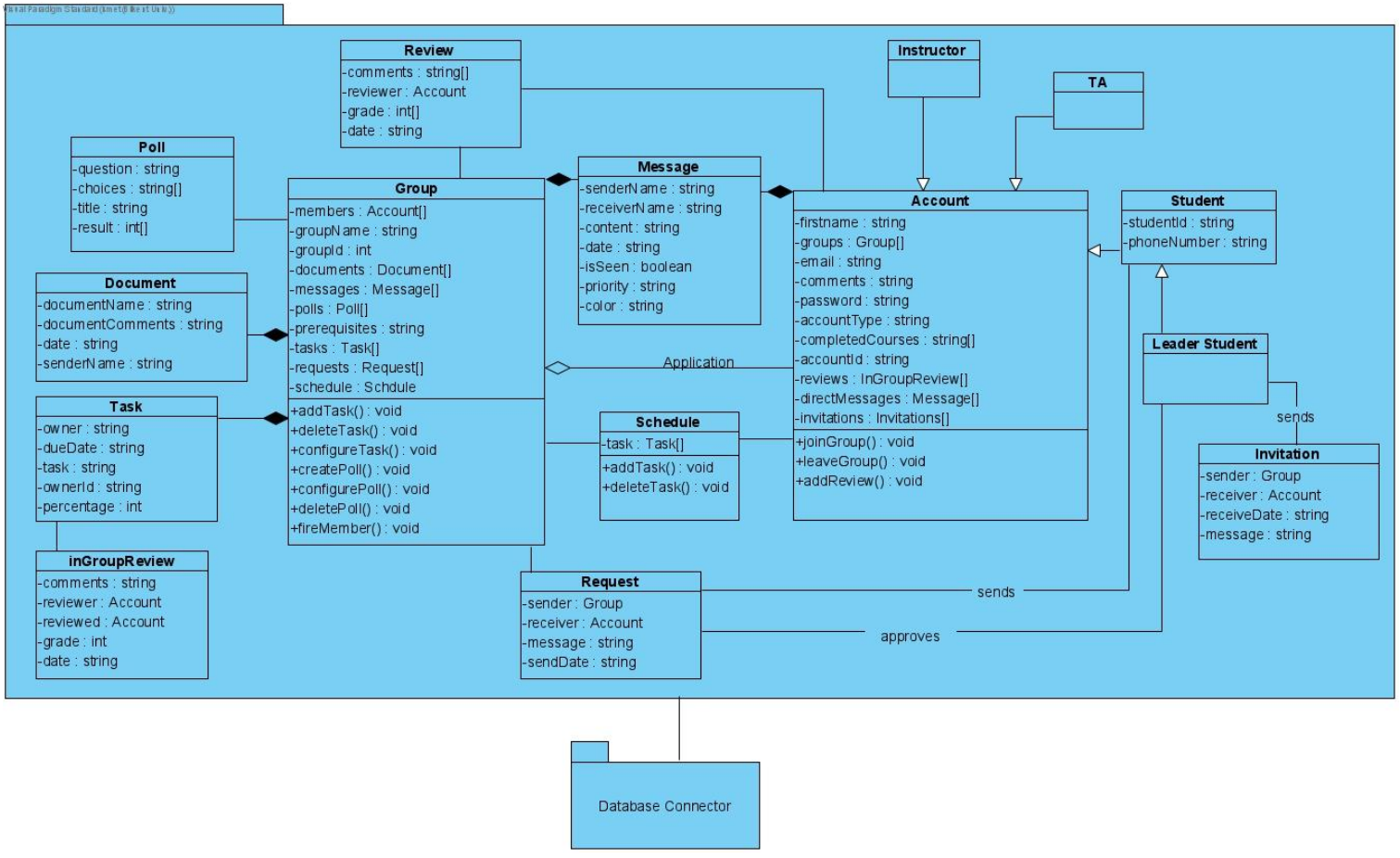

Figure 5: Example Sequence diagram of system.

Group Class

Explanation: This class is a representation of a group.

Attributes:

- **private members:** This attribute contains the members of the group as an Account array.
- **private groupName:** This attribute contains the name of the group.
- **private groupId:** This attribute contains a unique id of the group.
- **private documents:** This attribute contains sent documents to the group by group members as a Document array.
- **private messages:** This attribute contains messages sent to groups by group members as a Message array.
- **private polls:** This attribute contains the polls created by that group as a Poll array.
- **private tasks:** This attribute contains tasks for that group as a Task array.
- **private requests:** This attribute contains the join request sent to that group as a Request array.
- **private schedule:** This attribute contains schulade of the group as a Schedule object.

Methods:

- **public addMember():** This method lets users add new members to the current group with the permission of the group leader.
- **public addDocument():** This method lets users share their documents with their groups.
- **public deleteDocument():** This method lets users delete their documents from group documents list.
- **public showReviews():** This method lets users see reviews that are made for group projects.
- **public addTask():** This method lets users add new tasks to groups.
- **public deleteTask():** This method lets users delete the task belonging to a group.
- **public configureTask():** This method lets users edit tasks belonging to a group.
- **public createPoll():** This method lets users create new polls.
- **public configurePoll():** This method lets users edit polls belonging to a group.
- **public deletePoll():** This method lets users delete polls belonging to a group.
- **public fireMember():** This method lets users to delete their group member from their groups if it is needed. Such a need may occur when a member needs to change a group or a member wants to withdraw from the current course.

Review Class

Explanation: This class includes all reviews to the finished project. Both students,TA's and instructors could give reviews to other projects and give grades. In addition to giving grades students could give a suggestion and TA's & Instructors may include feedback in their review.

Attributes:

- **private comments:** This array includes comments and suggestions/feedback related to the project.
- **private reviewer:** It stores who the reviewer is.
- **private grade:** Integer array of grades, each index represents different categories of grades. Ex: Reliability, Security, Overall, etc..
- **private date:** It contains review date as string.

inGroupReview Class

Explanation: This class includes all in group review that is given by project members to their own teammates at the end of the project.

Attributes:

- **private reviewer:** It stores who the reviewer is.
- **private reviewed:** This attribute stores who has been evaluated.
- **private comments:** This string array includes comments of the reviewer to the reviewed teammate related to his/her performance and attitudes.

- **private grade:** Integer array of grades, each index represents different categories of grades. Ex: Studiousness, Team work, Does the given tasks fulfilled, etc…
- **private date:** It contains review date as string.

Poll Class

Explanation: Poll class is for enabling users to create polls where they can create their own poll questions.

Attributes:

- **private question:** Question is a string for where users enter their poll questions.
- **private choices:** This is a private string array for users to indicate their choices for their poll questions.
- **private title:** Title is a string for users to indicate their polls subject, thus other users of the system can understand the aim of that poll.
- **private result:** Result is an integer array for storing the number of votes for every individual choice.

Task Class

Explanation: Task class is for arranging schedules.

Attributes:

- **private owner:** Owner is a string for indicating who will do the task.
- **private dueDate:** Due date string shows the date which is the last date of the task to be completed.

- **private task:** Task string is for showing what should be done before the deadline.
- **private ownerId:** Owner id is a string which is specific for every individual user. In Task class, the owner id shows the id of the user who should complete the task.
- **private percentage:** Percentage is an integer in range 0-100. It shows how much the task has been completed by the owner of the task.

Request Class

Explanation: Invitation request from group to student in order to attract their attention.

Attributes:

- **private sender:** This attribute contains which group sends the request.
- **private receiver:** This attribute contains who will receive the invitation request.
- **private message:** This attribute contains messages related to the request.
- **private sendDate:** This attribute contains request creation date.

Message Class

Explanation: This class includes all attributes related to messages.

Attributes:

- **private senderName:** This attribute contains who sent the message.
- **private receiverName:** This attribute contains who will receive the message, this could also contain group name.
- **private content:** This attribute contains messages to send.

- **private date:** This attribute contains the date which the message was sent.
- **private isSeen:** This attribute contains whether the message is seen by the receiver or not.
- **private priority:** This attribute contains priority of the message. Depends on the priority, even if the user closed notifications off, they will still receive notification.
- **private color:** The message sent may have different colors depending on priority of the message, or how often users go back to a specified message.

Document Class

Explanation: Document class is for documents which are uploaded to the system by users.

Attributes:

- **private documentName:** Document name is a string for showing the name of the uploaded document.
- **private documentComments:** Document comments string includes the users', who uploaded the document, explanation for the document.
- **private date:** Date string shows the upload date of the document.
- **private senderName:** Sender name is a string which shows the user name who uploaded the particular document.

Schedule Class

Explanation: Both users and groups have schedules which contain which tasks to be done and when to do them.

Attributes:

- **private tasks:** This attribute contains all tasks related to that schedule in the task array.

Account

Explanation: This class represents the functionalities of the account.

Attributes:

- **private firstname:** The attribute that contains the user's first name.
- **private groups:** This is an array attribute which contains the user groups.
- **private email:** The attribute that contains the user's email address.
- **private comments:** This is an array attribute which contains the user comments about other projects and reviews..
- **private password:** The attribute that contains the user's hashed password.
- **private accountType:** The attribute that contains the user's type student or instructor or TA.
- **private completedCourses:** This is a list of user completed courses from old semesters.
- **private accountId:** The attribute that contains the user's ID.
- **private reviews:** This is an array of user reviews which users have given to other groups' projects.

- **private directMessages:** This is an array of Message objects. Users private or group chat messages keep in this list.
- **private invitations:** This attribute is a list of Invitation objects. Which keeps the current invitation of the users.

Methods:

- **public joinGroup():** This method adds the user to a selected group.
- **public leaveGroup():** This method removes the user from his current group.
- **public publishReview():** A new review is created and written to the database.

Instructor Class

Explanation: This class is extended from account class. Then it is specialized in order to meet the instructor users.

TA Class

Explanation: This class is extended from account class. Then it is specialized in order to meet the TA users

Student Class

Explanation: This class is extended from account class. Then it is specialized in order to meet the student users.

Attributes:

- **private studentId:** The attribute that contains the user's id.

- **private phonenumber:** The attribute that contains the user's phonenumber.

LeaderStudent Class

Explanation: This class is extended from student class but also has the ability to manage the group.

Methods:

- **public sendInvetation():** Send invitation objects to other students which are not in the group.
- **public acceptStudent():** Add a new student to the group.

Invitation Class

Explanation: This class is responsible for notifying students that they are invited to a group.

Attributes:

- **private sender:** This is the LeaderStudent object who sended the invitation.
- **private receiver:** This is the student who will receive the invitation.
- **private receiveDate:** This attribute represents the delivery time of invitation.
- **private message:** The invitation message kept in this attribute.

6.2 Database Controller Subsystem

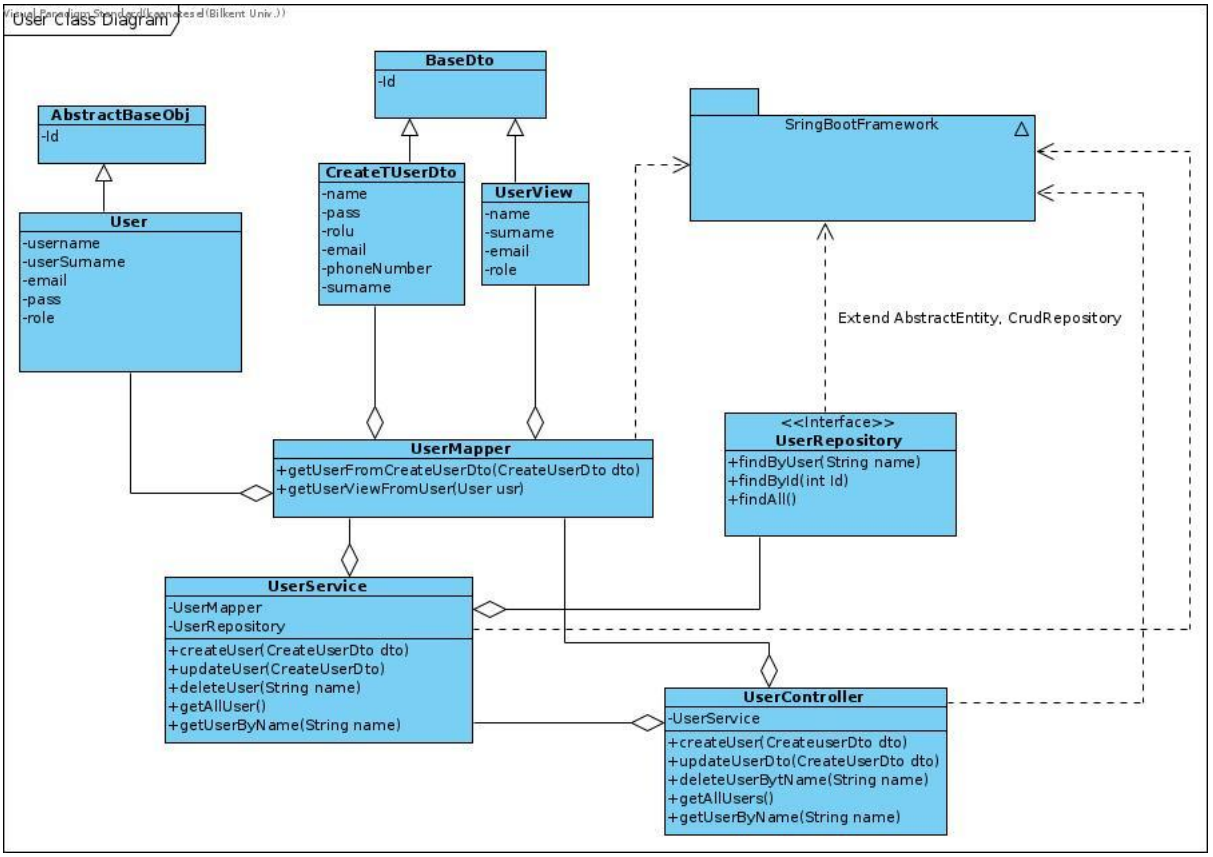

Figure 6: Database Class Diagram.

AbstractBaseObj Class:

This class is a base class of any database object. Thus all database objects will extend from this object.

Attributes:

- int Id

Methods:

- Getters and Setters

User Class:

Contains properties of User class. It also has getters and setters.

Attributes:

- String name
- String Surname
- String Role
- String phoneNumber
- String password

Methods:

- Getters and Setters

UserView Class:

This class is a return type object. It is the same as the user but doesn't have any private attributes such as passwords. Therefore it is safe to show in application.

Attributes:

- String name
- String Surname
- String Role
- String phoneNumber

Methods:

- Getters and Setters

UserRepository Interface:

This class is responsible for searching the object in the database. It returns the object if found else gives error.

Methods:

- **public User findByUser(String name):** return an user object from database.
- **public User findById(int Id):** find User by name
- **public List<User> findAll():** return the all user from the database.

UserService Class:

This class is responsible for managing user database objects. It handles api services store, retrieve, update and delete of the user.

Attributes:

- UserMapper
- UserRepository

Methods:

- **public UserView createUser(CreateUserDto dto):** create user from createUserDto and save it to database.
- **public UserView updateUser(CreateUserDto dto):** find the user and update the variables according to CreateUserDto and then save it to the database.
- **public UserView deleteUser(String name):** Delete the user with the given name form database.
- **public List<UserView> getAllUser():** retrieve all user from database
- **public getUserByName(String name):** retrieve the user with given name

UserController Class:

This class handles the http request with sended from the application. It uses get post and delete requests. It works when there is a request to the given url.

Attributes:

- UserServise
- UserRepository

Methods:

- **public ResponseEntity<UserView> createUser(CreateUserDto dto):** create user from createUserDto and save it to database and return the object in request body.
- **public ResponseEntity<UserView> updateUser(CreateUserDto dto):** find the user and update the variables according to CreateUserDto and then save it to the database. Then the object in the request body.
- **public ResponseEntity deleteUser(String name):** Delete the user with the given name form database. Return Http code 200 always.
- **public ResponseEntity<List<UserView>> getAllUser():** retrieve all users from database. Return a list of user objects in the request body.
- **public ResponseEntity<List<UserView>> getUserByName(String name):** retrieve the user with given name and return it in the request body.

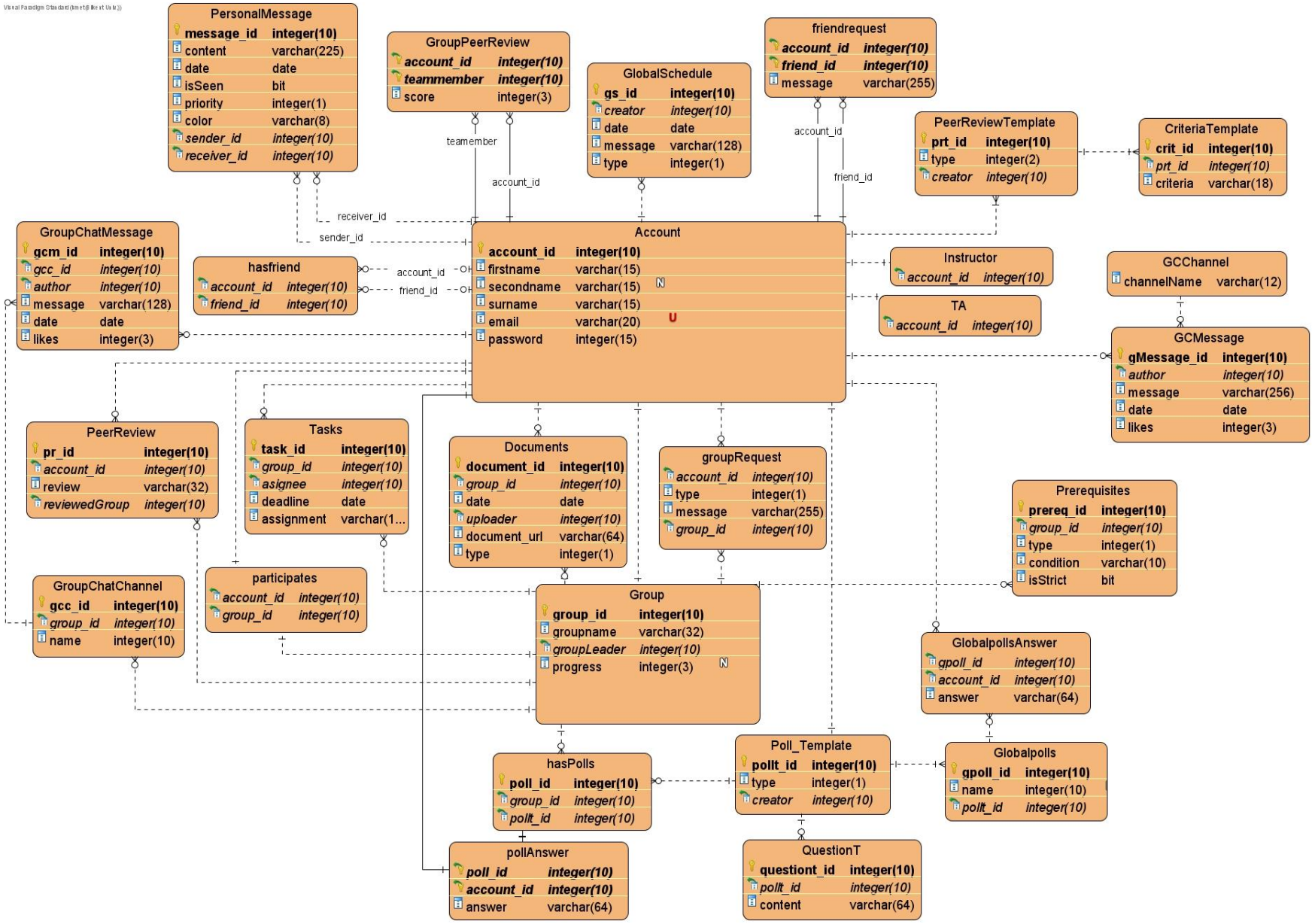

Figure 7: ER diagram