**MARMARA UNIVERSITY**

**FACULTY OF ENGINEERING**

ELECTRICAL&ELECTRONICS ENGINEERING

CSE2062

**OBJECT-ORIENTED PROGRAMMING**

**Project 1**



NAME STUDENT NUMBER

Efe Emir PEKMEZ 150718040

Batuhan Berkay ÖZTÜRK 150717031

Mehmet Mücahit DÖNMEZ 150717059

Hüseyin MENTEŞE 150717028

Mehmet Ali ÖZÇELİK 150717010

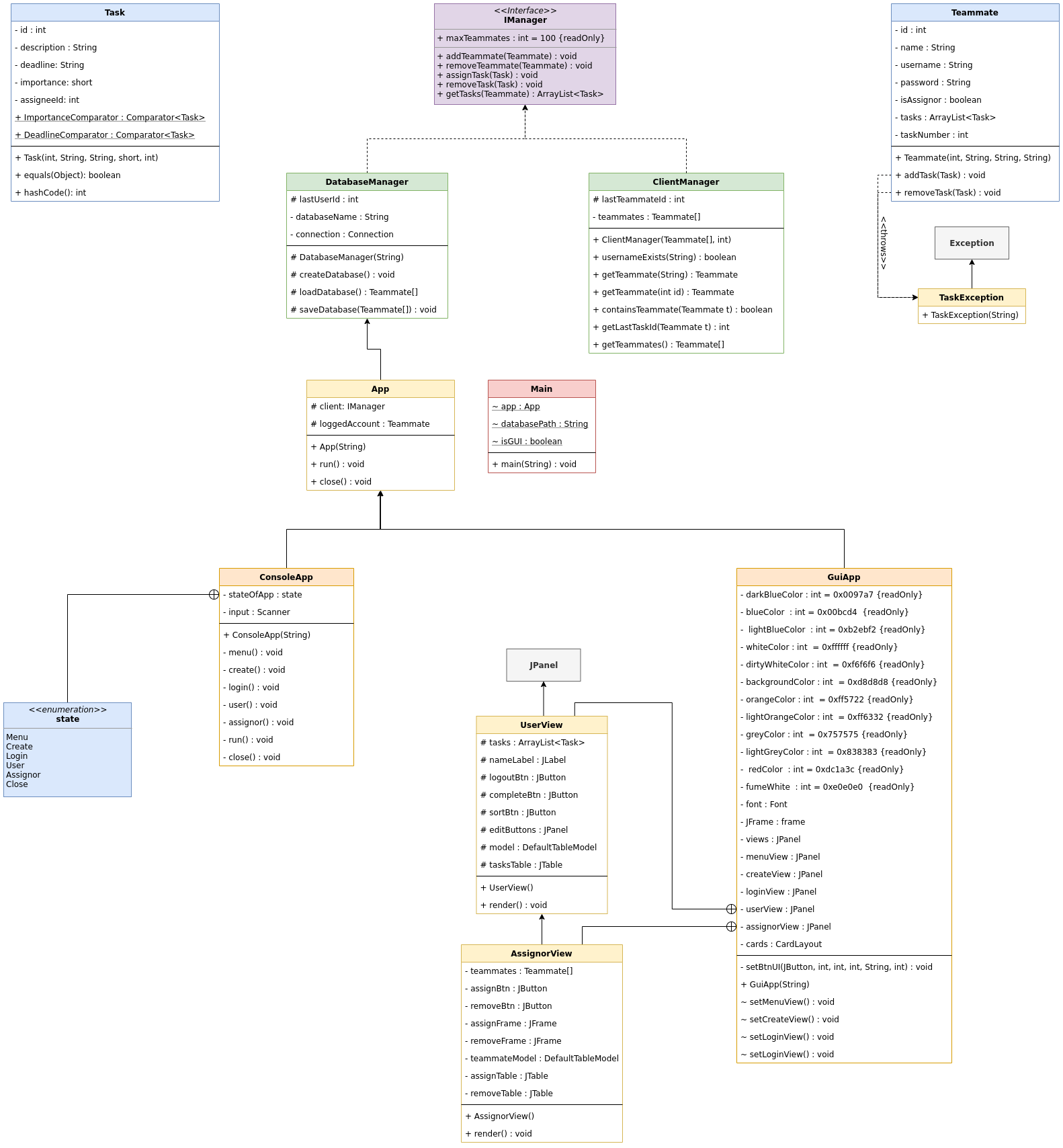
Submitted to: Şakir BİNGÖL

Due Date

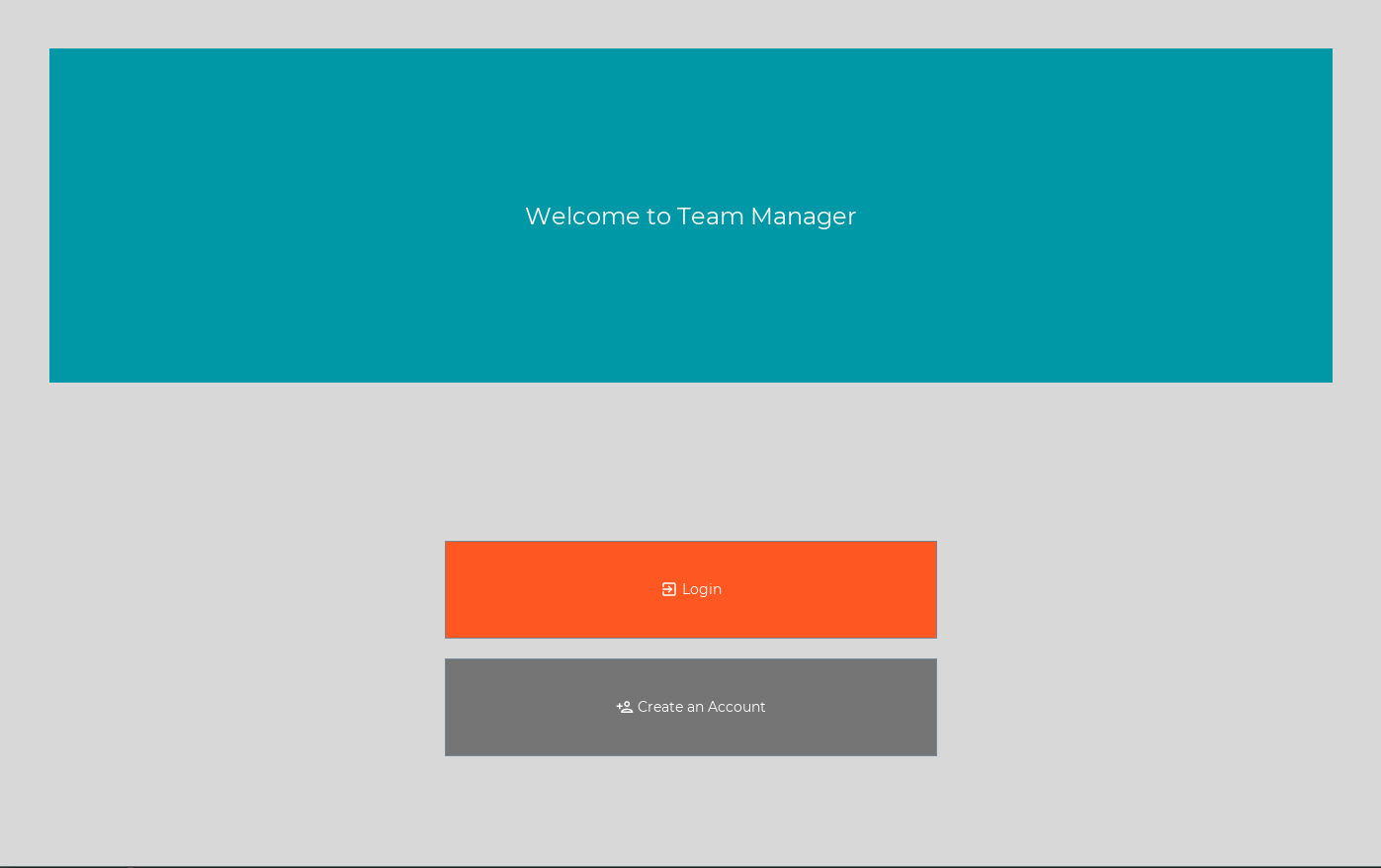
17.04.2020

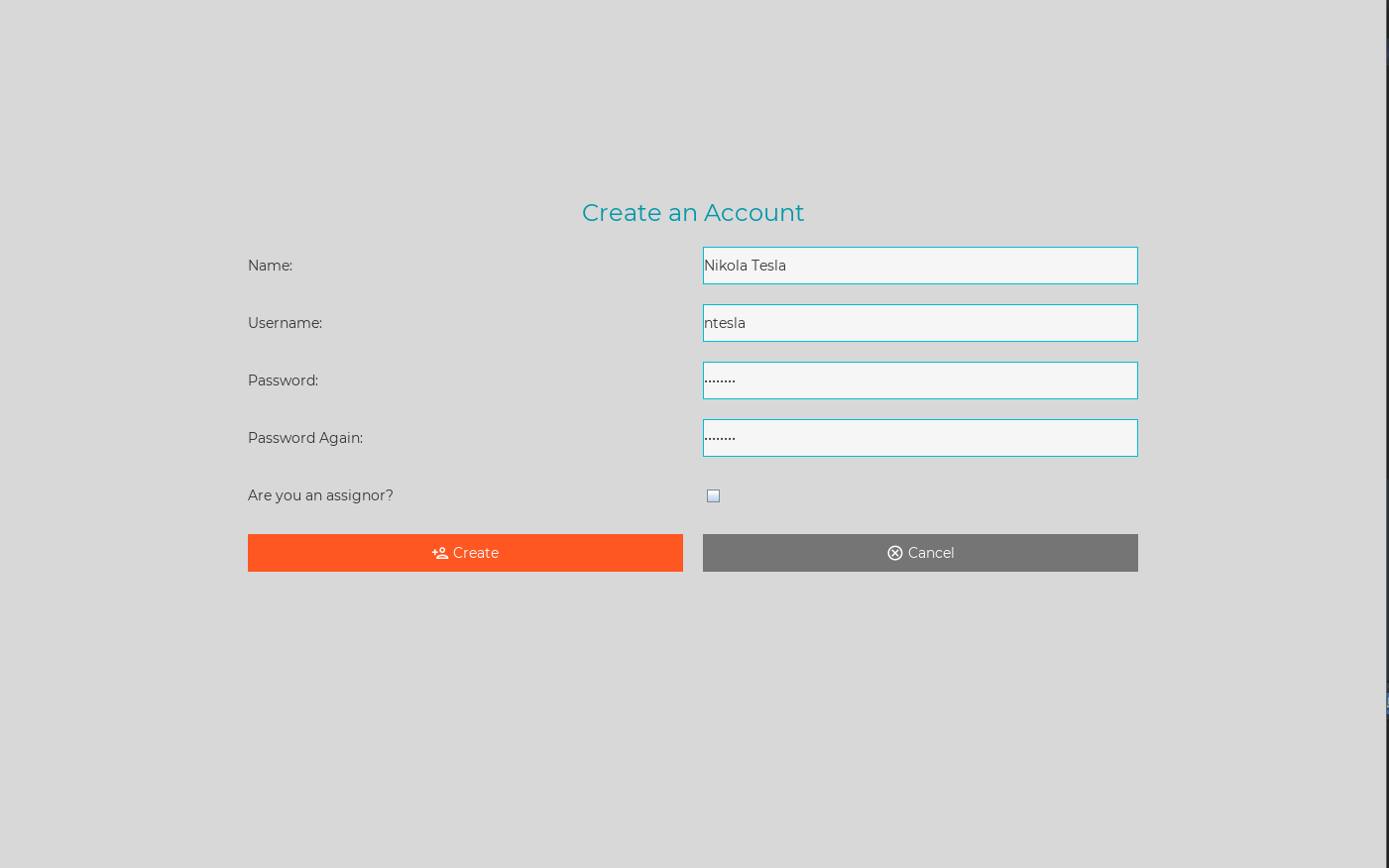
**Problem Definition:**

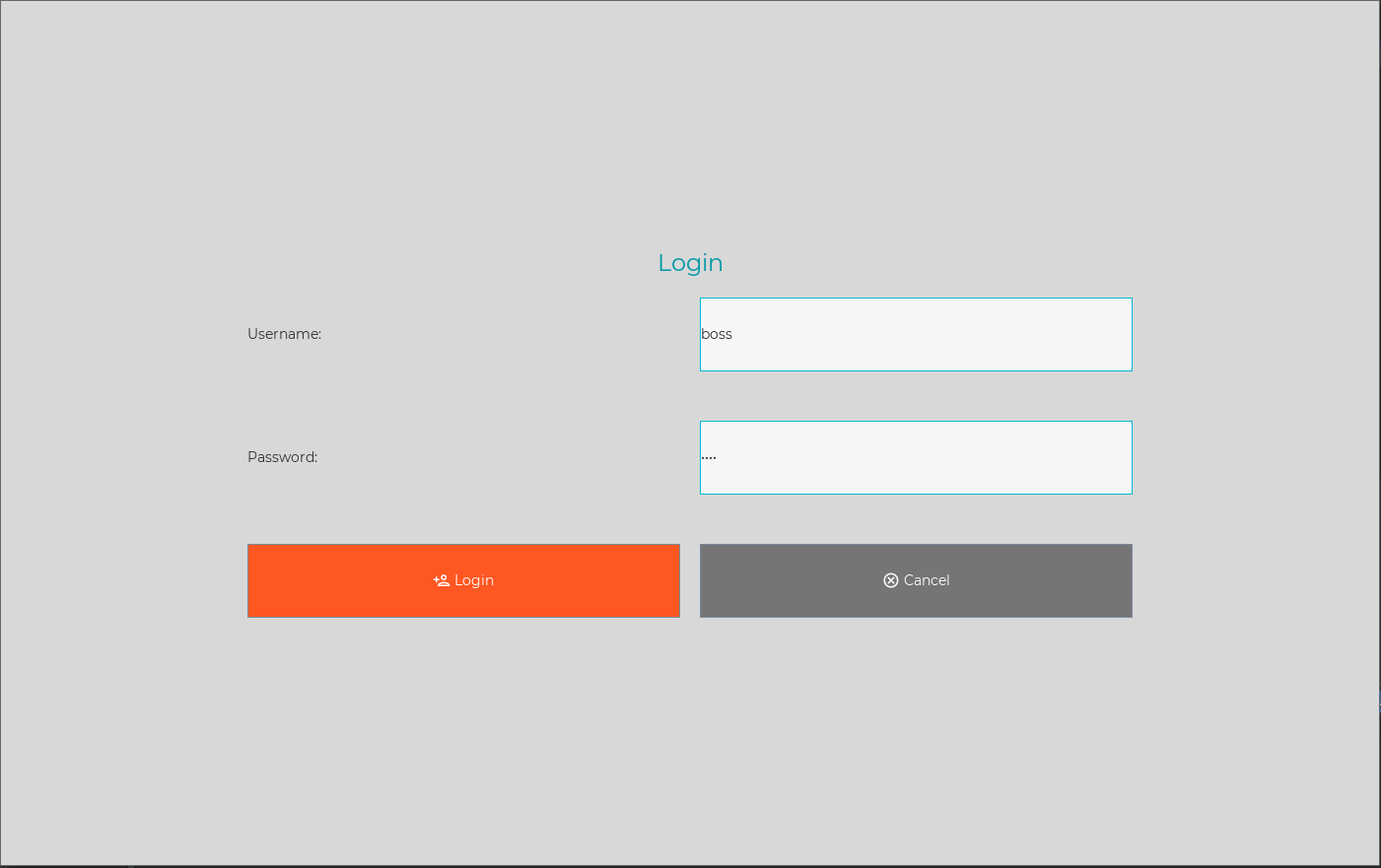
Managing a team of maximum 100 people; giving people assignments with deadlines and importance values, managing how a work is shared.

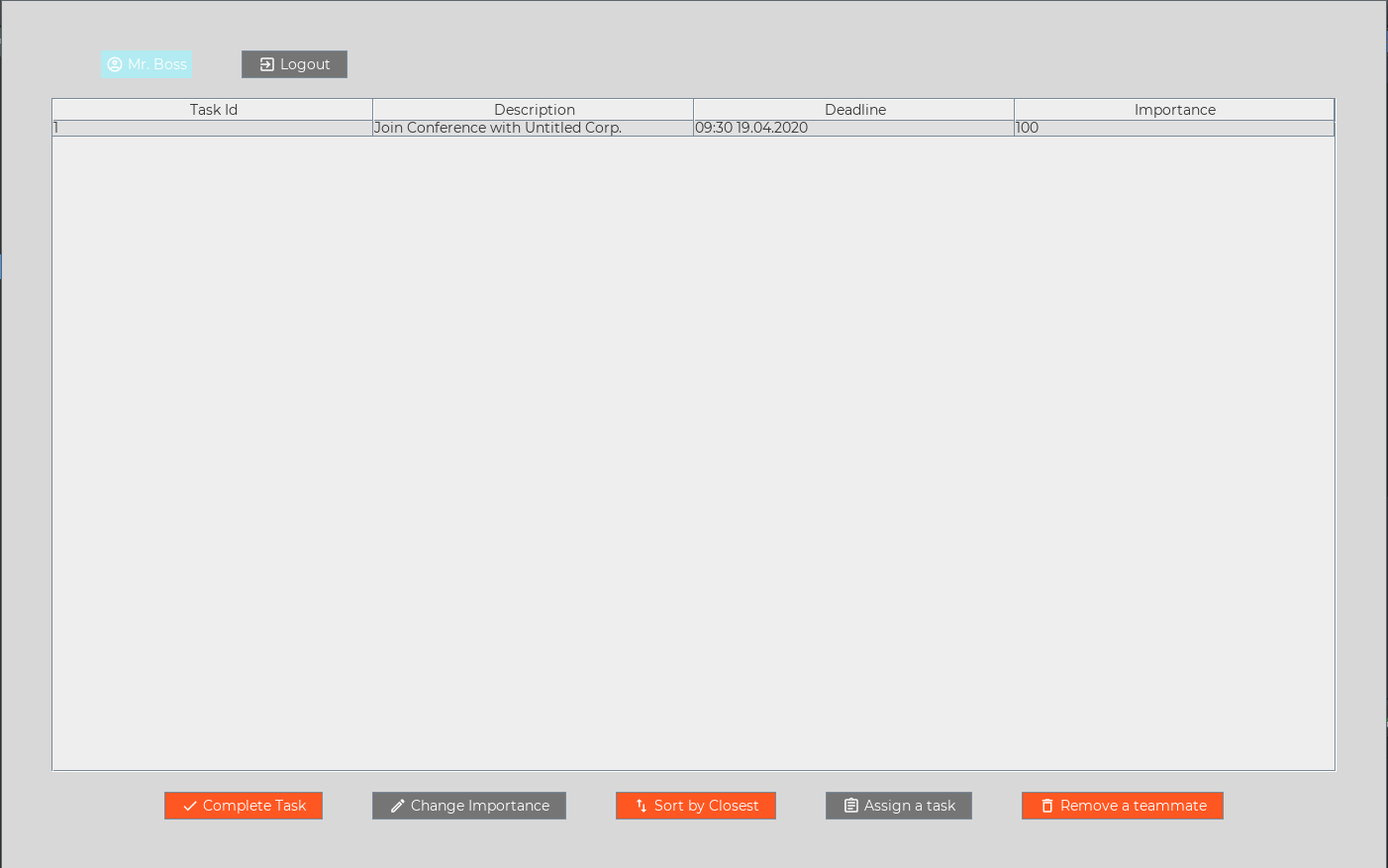


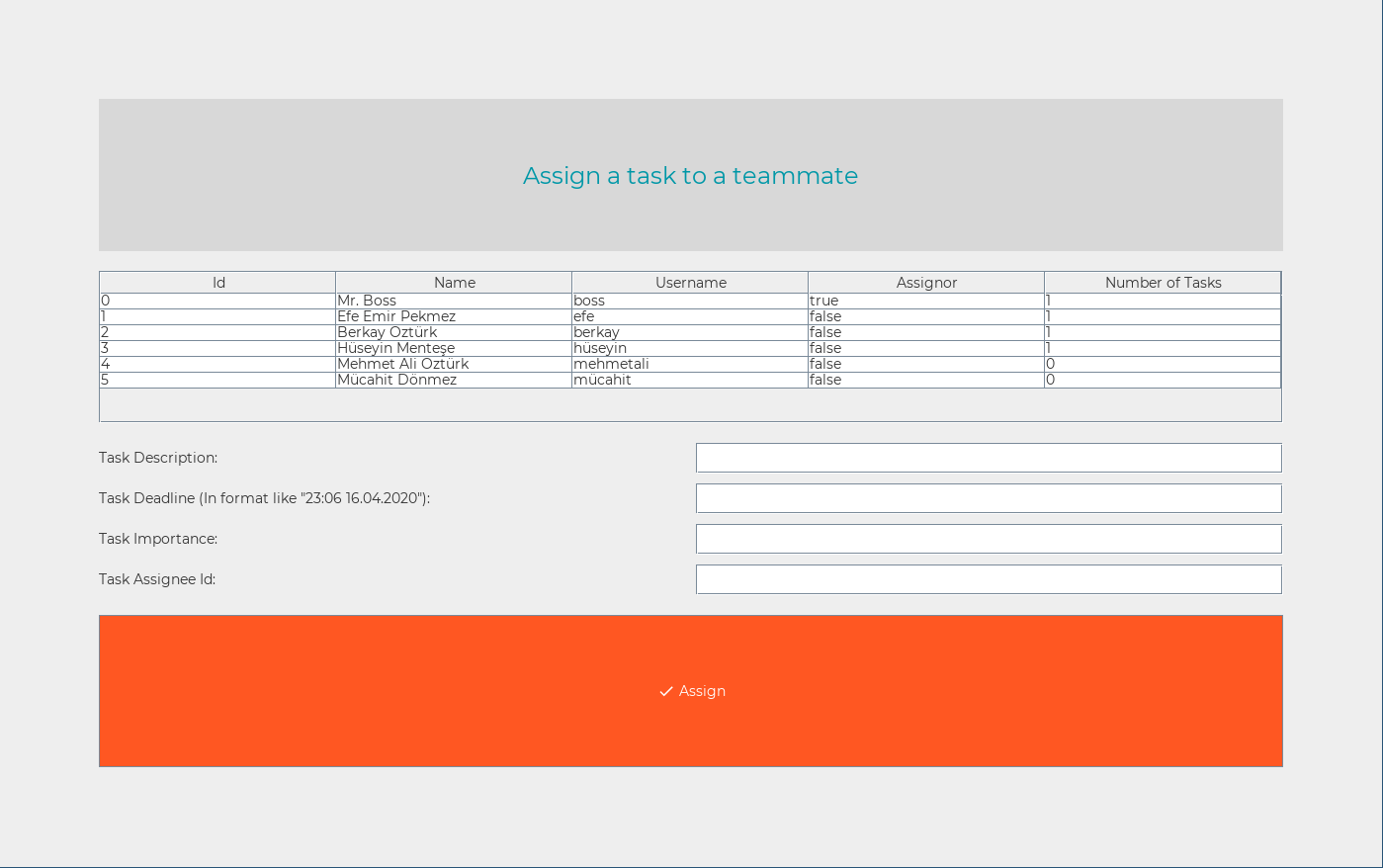
**Output Screens (GUI):**



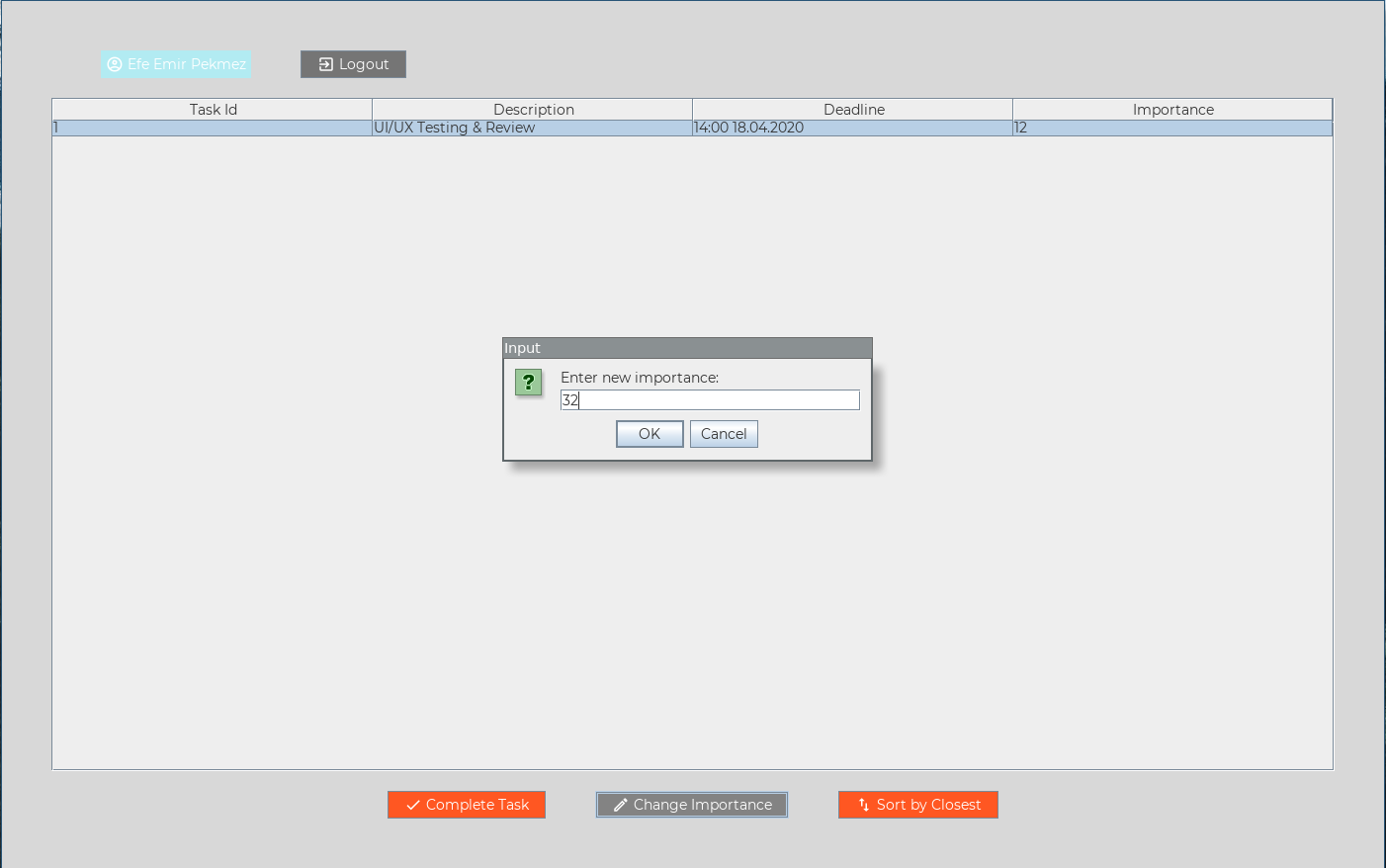
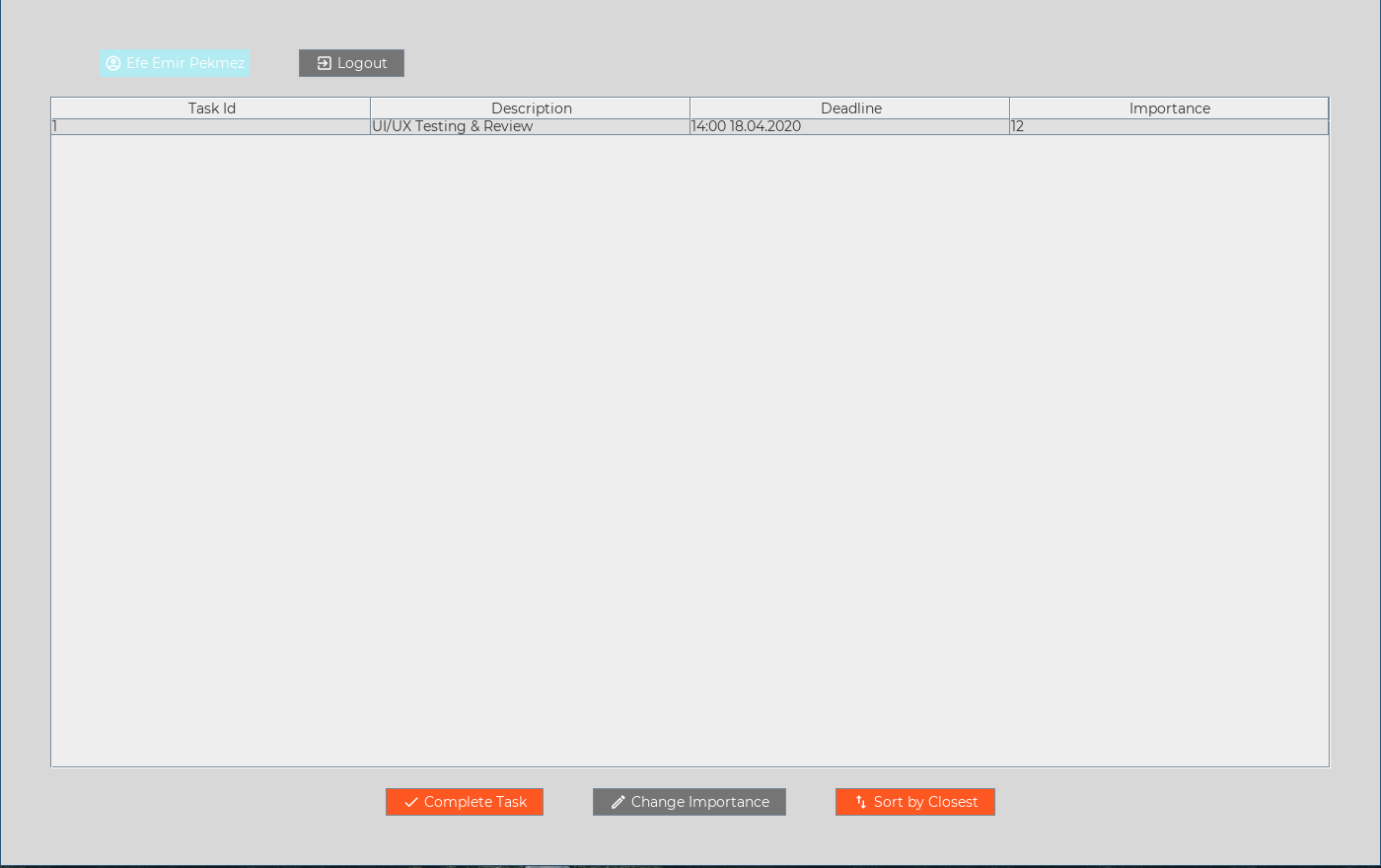




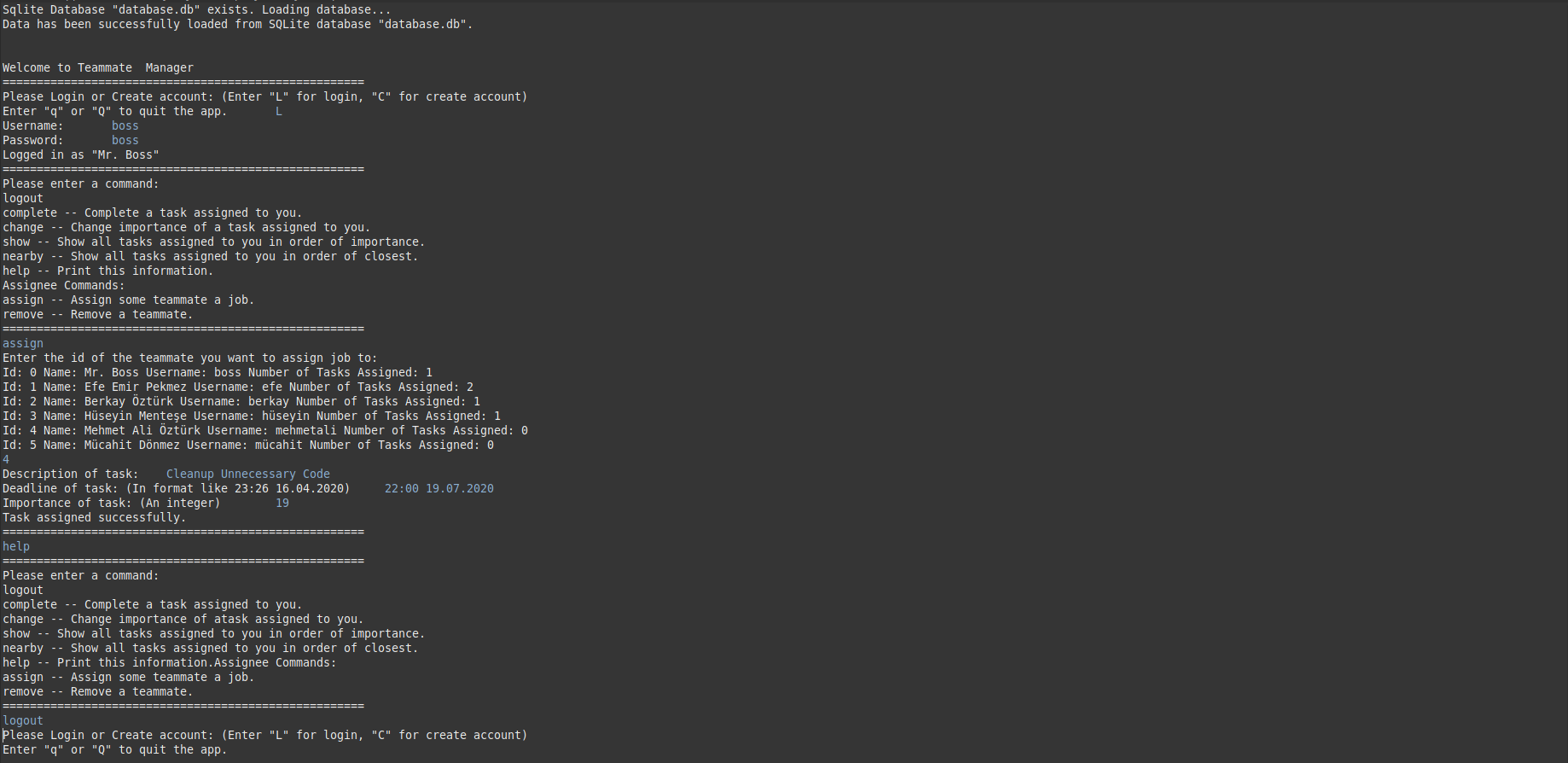


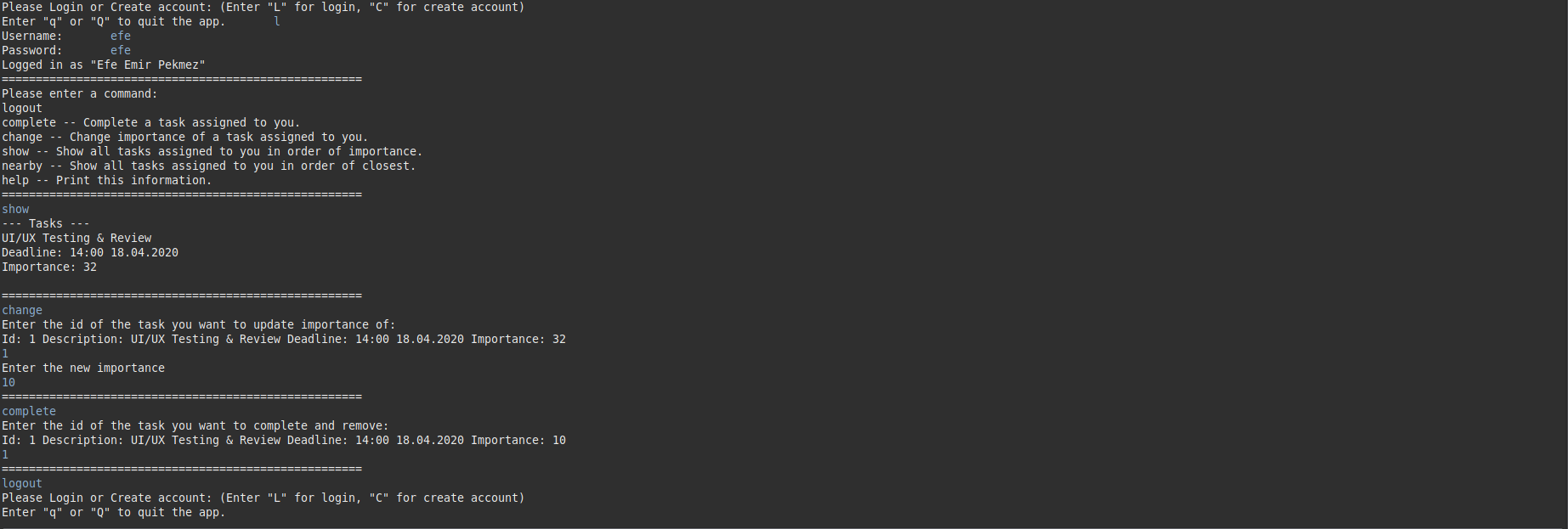






**Output Screens (Console):**







****

**Codes:**

**Note:** Download dependency: sqlite-jdbc library (Can be found from: <https://bitbucket.org/xerial/sqlite-jdbc/downloads/>)

**Task.java:**

package app;

import java.time.LocalDateTime;

import java.time.format.DateTimeFormatter;

import java.time.temporal.ChronoUnit;

import java.util.Comparator;

public class Task {

private int id;

private String description;

private String deadline;

private short importance;

private int assigneeId; // Who this task is assigned to.

Task(int id, String description, String deadline, short importance, int assigneeId) {

this.id = id;

this.description = description;

this.deadline = deadline;

this.importance = importance;

this.assigneeId = assigneeId;

}

int getId() {

return id;

}

void setId(int id) {

this.id = id;

}

String getDescription() {

return description;

}

void setDescription(String description) {

this.description = description;

}

String getDeadline() {

return deadline;

}

void setDeadline(String deadline) {

this.deadline = deadline;

}

short getImportance() {

return importance;

}

void setImportance(short importance) {

this.importance = importance;

}

int getAssigneeId() {

return assigneeId;

}

void setAssigneeId(int assigneeId) {

this.assigneeId = assigneeId;

}

@Override

public int hashCode() {

final int prime = 31;

int result = 1;

result = prime \* result + assigneeId;

result = prime \* result + ((deadline == null) ? 0 : deadline.hashCode());

result = prime \* result + ((description == null) ? 0 : description.hashCode());

result = prime \* result + id;

return result;

}

@Override

public boolean equals(Object obj) {

if (this == obj)

return true;

if (obj == null)

return false;

if (getClass() != obj.getClass())

return false;

Task other = (Task) obj;

if (assigneeId != other.assigneeId)

return false;

if (deadline == null) {

if (other.deadline != null)

return false;

} else if (!deadline.equals(other.deadline))

return false;

if (description == null) {

if (other.description != null)

return false;

} else if (!description.equals(other.description))

return false;

if (id != other.id)

return false;

return true;

}

// Compares 2 tasks; greater importance value will be before other.

// Later will be used to sort by Comparators.

public static Comparator<Task> ImportanceComparator = new Comparator<Task>() {

@Override

public int compare(Task t1, Task t2) {

return (int) (t2.importance - t1.importance);

}

};

// Compares 2 tasks; the one more closer to today will be before other.

// Later will be used to sort by Comparators.

public static Comparator<Task> DeadlineComparator = new Comparator<Task>() {

@Override

public int compare(Task t1, Task t2) {

// Parse LocalDateTime.

// Get time remains to deadlines.

// Minutes will be the order of significance.

DateTimeFormatter dtf = DateTimeFormatter.ofPattern("HH:mm dd.MM.yyyy");

LocalDateTime time1 = LocalDateTime.parse(t1.deadline, dtf);

LocalDateTime time2 = LocalDateTime.parse(t2.deadline, dtf);

return (int) (time2.until(time1, ChronoUnit.MINUTES));

}

};

}

**TaskException.java:**

package app;

public class TaskException extends Exception {

private static final long serialVersionUID = 4439619855336702008L;

TaskException(String message) {

super(message);

}

}

**Teammate.java:**

package app;

import java.util.ArrayList;

public class Teammate {

private int id;

private String name;

private String username;

private String password;

private boolean isAssignor;

private ArrayList<Task> tasks; // Array of tasks assigned to this intance Teammate.

private int taskNumber;

Teammate(int id, String name, String username, String password) {

this.id = id;

this.name = name;

this.username = username;

this.password = password;

// Initialize assigned tasks to 0.

tasks = new ArrayList<Task>();

taskNumber = 0;

}

int getId() {

return id;

}

void setId(int id) {

this.id = id;

}

String getName() {

return name;

}

void setName(String s) {

name = s;

}

String getUsername() {

return username;

}

void setUsername(String username) {

this.username = username;

}

String getPassword() {

return password;

}

void setPassword(String password) {

this.password = password;

}

boolean isAssignor() {

return isAssignor;

}

void setAssignor(boolean isAssignor) {

this.isAssignor = isAssignor;

}

ArrayList<Task> getTasks() {

return tasks;

}

// Assign a task to this instance.

void addTask(Task task) throws TaskException {

if(tasks.contains(task)) {

// Do not add same task to task array.

// Throws custom error.

throw new TaskException("Task already exists.");

} else {

// Otherwise add it.

tasks.add(task);

taskNumber++;

}

}

// Remove a task from this instance.

void removeTask(Task task) throws TaskException {

if(tasks.contains(task)) {

// Task has to be in array to be removed.

tasks.remove(task);

taskNumber--;

} else {

// Otherwise throw custom error.

throw new TaskException("Task is not on the list.");

}

}

int getTaskNumber() {

return taskNumber;

}

void setTaskNumber(int taskNumber) {

this.taskNumber = taskNumber;

}

}

**IManager.java:**

package app;

import java.util.ArrayList;

public interface IManager {

// Will be used to instantiate Teammate array to maximum size in child classes.

int maxTeammates = 100;

// Adds a teammate to system that is being managed.

void addTeammate(Teammate t);

// Removes a teammate from system that is being managed.

void removeTeammate(Teammate t);

// Assigns a task to a teammate in system that is being managed.

void assignTask(Task t);

// Removes a task from a teammate in system that is being managed.

void removeTask(Task t);

// Returns array of tasks of teammate "t".

ArrayList<Task> getTasks(Teammate t);

}

**ClientManager.java:**

package app;

import java.util.ArrayList;

public class ClientManager implements IManager {

// To give incremental id to every teammate we have to store the last id.

protected int lastTeammateId;

// Store all teammates available.

private Teammate[] teammates;

ClientManager(Teammate[] teammates, int lastIndex) {

if(teammates == null) {

// If teammate is empty then just instantiate array and set first element to null.

this.teammates = new Teammate[maxTeammates];

this.teammates[0] = null;

} else {

// Otherwise assign passed in array.

this.teammates = teammates;

}

this.lastTeammateId = lastIndex;

}

@Override

public void addTeammate(Teammate t) {

// Add new teammate then increment last teammate id.

teammates[lastTeammateId++] = t;

}

@Override

public void removeTeammate(Teammate t) {

// Remove the teammate passed in from array.

if(lastTeammateId == t.getId()) {

// If the lastTeammate is removing then just set the last element null and decrement last id.

teammates[--lastTeammateId] = null;

} else {

// Otherwise set the teammate with same id to null.

int last = maxTeammates;

for(int i = 0;i < maxTeammates;i++) {

if(teammates[i].getId() == t.getId()) {

last = i;

break;

}

}

// Then shift all teammates to fill emptied element.

for(int i=last;i < maxTeammates;i++) {

if(teammates[i] == null) {

break;

}

else {

teammates[i] = teammates[i+1];

}

}

}

}

@Override

public void assignTask(Task t) {

// Assign a task to corresponding teammate.

int id = t.getAssigneeId();

for(int i = 0;i < maxTeammates; i++) {

// Avoid empty elements.

if(teammates[i] != null) {

if(id == teammates[i].getId()) {

try {

teammates[i].addTask(t);

} catch (TaskException e) {

e.printStackTrace();

}

break;

}

}

}

}

@Override

public void removeTask(Task t) {

// Remove task from corresponding teammate.

int id = t.getAssigneeId();

for(int i = 0;i < maxTeammates; i++) {

// Avoid empty elements.

if(teammates[i] != null) {

if(id == teammates[i].getId()) {

try {

teammates[i].removeTask(t);

} catch (TaskException e) {

e.printStackTrace();

}

break;

}

}

}

}

@Override

public ArrayList<Task> getTasks(Teammate t) {

// Get tasks of teammate.

return t.getTasks();

}

public boolean usernameExists(String username) {

// Search for username. If found return true. Otherwise false.

for(int i = 0;i < maxTeammates;i++) {

if(teammates[i] != null) {

if(teammates[i].getUsername().equals(username)) {

return true;

}

}

}

return false;

}

public Teammate getTeammate(String username) {

// Get the teammate instance with same username from array.

for(int i = 0;i < maxTeammates; i++) {

if(teammates[i] != null) {

if(teammates[i].getUsername().equals(username)) {

return teammates[i];

}

}

}

return null;

}

public Teammate getTeammate(int id) {

// Find the teammate with same id.

for(int i = 0;i < maxTeammates; i++) {

if(teammates[i] != null) {

if(teammates[i].getId() == id) {

return teammates[i];

}

}

}

return null;

}

public boolean containsTeammate(Teammate t) {

// Check if the teammate with same id exists.

for(int i = 0;i < maxTeammates; i++) {

if(teammates[i] != null) {

if(teammates[i].getId() == t.getId()) {

return true;

}

}

}

return false;

}

public int getLastTaskId(Teammate t) {

// Get the last task id of spesific teammate.

int max = 0;

for(Task task: t.getTasks()) {

if(task.getId() > max) {

max = task.getId();

}

}

return max;

}

public Teammate[] getTeammates() {

// Get the array of teammates.

return teammates;

}

}

**DatabaseManager.java:**

package app;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.sql.Statement;

import java.util.ArrayList;

public class DatabaseManager implements IManager {

// This will hold the max id of teammates.

protected int lastUserId;

private String databaseName;

// Global connection to sqlite database.

private Connection connection;

DatabaseManager(String databaseName) {

// Set the name of database, set the connection to null.

connection = null;

this.databaseName = databaseName;

// Assume there is no teammate.

lastUserId = 0;

}

protected void createDatabase() {

// Initialize tables in sqlite.

try {

connection = DriverManager.getConnection("jdbc:sqlite:" + databaseName);

Statement statement = connection.createStatement();

// Table that will hold the users.

statement.execute("CREATE TABLE IF NOT EXISTS users(\n"

+ " id integer PRIMARY KEY,\n"

+ " name text NOT NULL,\n"

+ " username text NOT NULL,\n"

+ " password text NOT NULL,\n"

+ " isAssignor integer DEFAULT 0,\n"

+ " taskNumber integer DEFAULT 0\n"

+ ");");

// Table that will hold the tasks.

statement.execute("CREATE TABLE IF NOT EXISTS tasks(\n"

+ " id integer PRIMARY KEY,\n"

+ " description text NOT NULL,\n"

+ " deadline text,\n"

+ " importance integer DEFAULT 0,\n"

+ " assignee integer NOT NULL\n"

+ ");");

} catch (SQLException e) {

e.printStackTrace();

} finally {

try {

if(connection != null) {

connection.close();

}

} catch (SQLException e) {

e.printStackTrace();

}

}

}

protected Teammate[] loadDatabase() {

// Load database to "users" array that will be returned after.

Teammate[] users = new Teammate[maxTeammates];

int i = 0;

try {

// Create a connection to the database.

connection = DriverManager.getConnection("jdbc:sqlite:" + databaseName);

Statement statement = connection.createStatement();

ResultSet usersTable = statement.executeQuery("SELECT \* FROM users");

ResultSet tasks;

int id;

String name;

String username;

String password;

boolean isAssignor;

int taskNumber;

// First get all users from table.

while(usersTable.next()) {

users[i] = null;

id = usersTable.getInt("id");

name = usersTable.getString("name");

username = usersTable.getString("username");

password = usersTable.getString("password");

isAssignor = (usersTable.getInt("isAssignor") == 1) ? true : false;

taskNumber = usersTable.getInt("taskNumber");

Teammate t = new Teammate(id, name, username, password);

t.setAssignor(isAssignor);

t.setTaskNumber(taskNumber);

if(id >= lastUserId) {

lastUserId = id+1;

}

users[i++] = t;

}

PreparedStatement preparedStatement = connection.prepareStatement("SELECT \* FROM tasks WHERE assignee = ?");

i--;

// Then get tasks assigned to them from tasks table in database.

for(;i>=0;i--) {

id = users[i].getId();

preparedStatement.setInt(1, id);

tasks = preparedStatement.executeQuery();

while(tasks.next()) {

try{

users[id].addTask((new Task(tasks.getInt("id"),

tasks.getString("description"),

tasks.getString("deadline"),

(short) tasks.getInt("importance"),

tasks.getInt("assignee")

)));

} catch(TaskException e) {

e.printStackTrace();

}

}

}

System.out.println("Data has been successfully loaded from SQLite database \""+ databaseName +"\".");

} catch (SQLException e) {

e.printStackTrace();

} finally {

try {

if(connection != null) {

connection.close();

}

} catch (SQLException e) {

e.printStackTrace();

}

}

// Return null if the first element of loaded array is null.

return (users[0] != null) ? users:null;

}

protected void saveDatabase(Teammate[] users) {

// Save everything in passed array to database.

String[] sql = {"REPLACE INTO users VALUES(?,?,?,?,?,?)",

"REPLACE INTO tasks VALUES(?,?,?,?,?)"};

try {

connection = DriverManager.getConnection("jdbc:sqlite:" + databaseName);

PreparedStatement preparedStatement = connection.prepareStatement(sql[0]);

// First just update users table.

for(int i=0;i < maxTeammates;i++) {

if(users[i] == null) {

continue;

}

preparedStatement.setInt(1, users[i].getId());

preparedStatement.setString(2, users[i].getName());

preparedStatement.setString(3, users[i].getUsername());

preparedStatement.setString(4, users[i].getPassword());

preparedStatement.setInt(5, users[i].isAssignor() ? 1:0);

preparedStatement.setInt(6, users[i].getTaskNumber());

preparedStatement.executeUpdate();

}

preparedStatement = connection.prepareStatement(sql[1]);

// Then update tasks table.

for(int i=0;i < maxTeammates;i++) {

if(users[i] == null) {

continue;

}

for(Task task:users[i].getTasks()) {

preparedStatement.setInt(1, task.getId());

preparedStatement.setString(2, task.getDescription());

preparedStatement.setString(3, task.getDeadline());

preparedStatement.setInt(4, task.getImportance());

preparedStatement.setInt(5, task.getAssigneeId());

}

}

} catch (SQLException e) {

e.printStackTrace();

} finally {

try {

if(connection != null) {

connection.close();

}

} catch (SQLException e) {

e.printStackTrace();

}

}

}

@Override

public void addTeammate(Teammate t) {

// Add just one teammate to database.

String[] sql = {"REPLACE INTO users VALUES(?,?,?,?,?,?)",

"REPLACE INTO tasks VALUES(?,?,?,?,?)"};

try {

connection = DriverManager.getConnection("jdbc:sqlite:" + databaseName);

PreparedStatement preparedStatement = connection.prepareStatement(sql[0]);

preparedStatement.setInt(1, t.getId());

preparedStatement.setString(2, t.getName());

preparedStatement.setString(3, t.getUsername());

preparedStatement.setString(4, t.getPassword());

preparedStatement.setInt(5, t.isAssignor() ? 1:0);

preparedStatement.setInt(6, t.getTaskNumber());

preparedStatement.executeUpdate();

// Add assigned tasks to database too.

for(Task task:t.getTasks()) {

preparedStatement = connection.prepareStatement(sql[1]);

preparedStatement.setInt(1, task.getId());

preparedStatement.setString(2, task.getDescription());

preparedStatement.setString(3, task.getDeadline());

preparedStatement.setInt(4, task.getImportance());

preparedStatement.setInt(5, task.getAssigneeId());

preparedStatement.executeUpdate();

}

} catch (SQLException e) {

e.printStackTrace();

} finally {

try {

if(connection != null) {

connection.close();

}

} catch (SQLException e) {

e.printStackTrace();

}

}

}

@Override

public void removeTeammate(Teammate t) {

String[] sql = {"DELETE FROM users WHERE id = ?",

"DELETE FROM tasks WHERE assignee = ?"

};

try {

// Remove a teammate from database.

connection = DriverManager.getConnection("jdbc:sqlite:" + databaseName);

PreparedStatement preparedStatement = connection.prepareStatement(sql[0]);

// Set the corresponding parameter.

preparedStatement.setInt(1, t.getId());

// Execute the delete statement.

preparedStatement.executeUpdate();

// Remove corresponding tasks too.

preparedStatement = connection.prepareStatement(sql[1]);

// Set the corresponding parameter.

preparedStatement.setInt(1, t.getId());

// Execute the delete statement.

preparedStatement.executeUpdate();

} catch (SQLException e) {

e.printStackTrace();

} finally {

try {

if(connection != null) {

connection.close();

}

} catch (SQLException e) {

e.printStackTrace();

}

}

}

@Override

public void assignTask(Task t) {

// Add a task to table.

String sql = "REPLACE INTO tasks VALUES(?,?,?,?,?)";

try {

connection = DriverManager.getConnection("jdbc:sqlite:" + databaseName);

PreparedStatement preparedStatement;

preparedStatement = connection.prepareStatement(sql);

preparedStatement.setInt(1, t.getId());

preparedStatement.setString(2, t.getDescription());

preparedStatement.setString(3, t.getDeadline());

preparedStatement.setInt(4, t.getImportance());

preparedStatement.setInt(5, t.getAssigneeId());

preparedStatement.executeUpdate();

} catch (SQLException e) {

e.printStackTrace();

} finally {

try {

if(connection != null) {

connection.close();

}

} catch (SQLException e) {

e.printStackTrace();

}

}

}

@Override

public void removeTask(Task t) {

// Remove one task from table.

String sql = "DELETE FROM tasks WHERE id = ?";

try {

// Create a connection to the database.

connection = DriverManager.getConnection("jdbc:sqlite:" + databaseName);

PreparedStatement preparedStatement = connection.prepareStatement(sql);

preparedStatement.setInt(1, t.getId());

// Execute the delete statement.

preparedStatement.executeUpdate();

} catch (SQLException e) {

e.printStackTrace();

} finally {

try {

if(connection != null) {

connection.close();

}

} catch (SQLException e) {

e.printStackTrace();

}

}

}

@Override

public ArrayList<Task> getTasks(Teammate t) {

// Get tasks assigned to spesific teammate from database.

ArrayList<Task> result = new ArrayList<Task>();

String sql = "SELECT FROM tasks WHERE assignee = ?";

try {

// Create a connection to the database

connection = DriverManager.getConnection("jdbc:sqlite:" + databaseName);

PreparedStatement preparedStatement = connection.prepareStatement(sql);

preparedStatement.setInt(1, t.getId());

ResultSet tasks = preparedStatement.executeQuery();

while(tasks.next()) {

result.add(new Task(tasks.getInt("id"),

tasks.getString("description"),

tasks.getString("deadline"),

(short) tasks.getInt("importance"),

tasks.getInt("assignee")

));

}

} catch(SQLException e) {

e.printStackTrace();

} finally {

try {

if(connection != null) {

connection.close();

}

} catch (SQLException e) {

e.printStackTrace();

}

}

if(result.isEmpty()) {

return null;

} else {

return result;

}

}

}

**App.java:**

package app;

import java.io.File;

public abstract class App extends DatabaseManager {

// Will hold ClientManager instance.

protected IManager client;

// This will hold currently logged in account.

protected Teammate loggedAccount;

App(String databasePath) {

// Set up database manager.

super(databasePath);

// This will hold the loaded teammates from database.

Teammate[] teammates = null;

// Check if the specified database exists or not.

File f = new File(databasePath);

if (f.exists()) {

// Load teammates from database if available.

System.out.println("Sqlite Database \"" + databasePath + "\" exists. Loading database..." );

teammates = loadDatabase();

}

else {

// Create database otherwise.

System.out.println("Sqlite Database \"" + databasePath + "\" does not exist. Creating database...");

createDatabase();

}

// Set up ClientManager instance.

client = new ClientManager(teammates, lastUserId);

}

// This will be defined in ConsoleApp and GuiApp.

abstract void run();

// Save database before application closes.

void close() {

// Get the available teammates from client and save to sqlite database.

saveDatabase(((ClientManager) client).getTeammates());

}

}

**ConsoleApp.java:**

package app;

import java.time.LocalDateTime;

import java.time.format.DateTimeFormatter;

import java.util.ArrayList;

import java.util.Scanner;

public class ConsoleApp extends App {

private enum state{

Menu,

Create,

Login,

User,

Assignor,

Close

};

// State which app is in.

private state stateOfApp;

// Global console input.

private Scanner input;

ConsoleApp(String databasePath) {

// Set up as a application first.

super(databasePath);

// Initialize attributes.

stateOfApp = state.Menu;

input = new Scanner(System.in);

}

private void menu() {

// Menu Screen.

while(stateOfApp == state.Menu) {

System.out.print("Please Login or Create account: (Enter \"L\" for login, \"C\" for create account)\n"

+ "Enter \"q\" or \"Q\" to quit the app.\t"

);

char answer;

answer = input.nextLine().charAt(0);

if(answer == 'C' || answer == 'c') {

stateOfApp = state.Create;

} else if(answer == 'L' || answer == 'l') {

stateOfApp = state.Login;

} else if(answer == 'Q' || answer == 'q') {

stateOfApp = state.Close;

} else {

System.out.println("Option \"" + answer + "\" does not exist.");

}

}

}

private void create() {

// Create Account Screen.

String username, password, name;

boolean assignor;

do {

System.out.print("Username:\t");

username = input.nextLine();

} while(((ClientManager) client).usernameExists(username));

System.out.print("Password:\t");

password = input.nextLine();

do {

System.out.print("Password Again:\t");

} while(!input.nextLine().equals(password));

System.out.print("Name:\t");

name = input.nextLine();

System.out.print("Are you a assignor: (Enter \"Y\" for yes \"N\" for no)\t");

char answer = input.nextLine().charAt(0);

assignor = (answer == 'Y' || answer == 'y') ? true: false;

Teammate t = new Teammate(lastUserId, name, username, password);

t.setAssignor(assignor);

client.addTeammate(t);

addTeammate(t);

System.out.println("Account Created now you can login:");

stateOfApp = state.Login;

}

private void login() {

// Login Account Screen.

String username, password;

System.out.print("Username:\t");

username = input.nextLine();

Teammate t = ((ClientManager) client).getTeammate(username);

if(t != null) {

System.out.print("Password:\t");

password = input.nextLine();

if(password.equals(t.getPassword())) {

System.out.println("Logged in as \"" + t.getName() + "\"");

loggedAccount = t;

stateOfApp = t.isAssignor() ? state.Assignor: state.User;

} else {

System.out.println("Password incorrect. Try again later.");

}

} else {

System.out.println("User \"" + username + "\" does not exist.");

stateOfApp = state.Menu;

}

}

private void user() {

// When a normal user is logged in.

System.out.println("=====================================================\n"

+ "Please enter a command:\n"

+ "logout\n"

+ "complete -- Complete a task assigned to you.\n"

+ "change -- Change importance of a task assigned to you.\n"

+ "show -- Show all tasks assigned to you in order of importance.\n"

+ "nearby -- Show all tasks assigned to you in order of closest.\n"

+ "help -- Print this information."

);

// User Screen should always show while user is logged in.

while(stateOfApp == state.User) {

System.out.println("=====================================================");

ArrayList<Task> tasks = loggedAccount.getTasks();

Task selected;

String answer;

int id;

answer = input.nextLine();

switch(answer) {

case "logout":

// Logging out.

loggedAccount = null;

stateOfApp = state.Menu;

break;

case "complete":

// Completing a task.

System.out.println("Enter the id of the task you want to complete and remove:");

for(Task t: tasks) {

System.out.println("Id: "+ t.getId() + " Description: " + t.getDescription() + " Deadline: " + t.getDeadline()

+ " Importance: " + t.getImportance());

}

id = input.nextInt();

input.nextLine();

selected = null;

for(Task t: tasks) {

if(t.getId() == id) {

selected = t;

}

}

if(selected != null) {

client.removeTask(selected);

removeTask(selected);

} else {

System.out.println("Task with id:" + id + " does not exist.");

}

break;

case "change":

// Changing a task importance.

System.out.println("Enter the id of the task you want to update importance of:");

for(Task t: tasks) {

System.out.println("Id: "+ t.getId() + " Description: " + t.getDescription() + " Deadline: " + t.getDeadline()

+ " Importance: " + t.getImportance());

}

id = input.nextInt();

input.nextLine();

selected = null;

for(Task t: tasks) {

if(t.getId() == id) {

selected = t;

}

}

if(selected != null) {

short importance;

System.out.println("Enter the new importance");

importance = input.nextShort();

input.nextLine();

client.removeTask(selected);

selected.setImportance(importance);

client.assignTask(selected);

assignTask(selected);

} else {

System.out.println("Task with id:" + id + " does not exist.");

}

break;

case "show":

// Show tasks in importance order.

tasks.sort(Task.ImportanceComparator);

System.out.println("--- Tasks ---");

for(Task t: tasks) {

System.out.println(t.getDescription() + "\nDeadline: " + t.getDeadline()

+ "\nImportance: " + t.getImportance() + "\n");

}

break;

case "nearby":

// Show tasks in deadline time closest order.

tasks.sort(Task.DeadlineComparator);

System.out.println("--- Tasks ---");

for(Task t: tasks) {

System.out.println(t.getDescription() + "\nDeadline: " + t.getDeadline()

+ "\nImportance: " + t.getImportance() + "\n");

}

break;

case "help":

// Show help if needed.

System.out.println("=====================================================\n"

+ "Please enter a command:\n"

+ "logout\n"

+ "complete -- Complete a task assigned to you.\n"

+ "change -- Change importance of atask assigned to you.\n"

+ "show -- Show all tasks assigned to you in order of importance.\n"

+ "nearby -- Show all tasks assigned to you in order of closest.\n"

+ "help -- Print this information."

);

break;

default:

// Show help if unknown command entered.

System.out.println("Command \"" + answer + "\" does not exist. Try this ones:\n"

+ "=====================================================\n"

+ "Please enter a command:\n"

+ "logout\n"

+ "complete -- Complete a task assigned to you.\n"

+ "change -- Change importance of atask assigned to you.\n"

+ "show -- Show all tasks assigned to you in order of importance.\n"

+ "nearby -- Show all tasks assigned to you in order of closest.\n"

+ "help -- Print this information."

);

}

}

}

void assignor() {

// When a assignor user is logged in.

System.out.println("=====================================================\n"

+ "Please enter a command:\n"

+ "logout\n"

+ "complete -- Complete a task assigned to you.\n"

+ "change -- Change importance of a task assigned to you.\n"

+ "show -- Show all tasks assigned to you in order of importance.\n"

+ "nearby -- Show all tasks assigned to you in order of closest.\n"

+ "help -- Print this information.\n"

+ "Assignee Commands:\n"

+ "assign -- Assign some teammate a job.\n"

+ "remove -- Remove a teammate."

);

// Assignor Screen should always show while assignor is logged in.

while(stateOfApp == state.Assignor) {

System.out.println("=====================================================");

ArrayList<Task> tasks = loggedAccount.getTasks();

Teammate[] teammates = ((ClientManager) client).getTeammates();

Teammate selectedTeammate;

Task selected;

String answer, description, deadline;

int id, taskId;

short importance;

answer = input.nextLine();

switch(answer) {

case "logout":

// Logging out.

loggedAccount = null;

stateOfApp = state.Menu;

break;

case "complete":

// Completing a task.

System.out.println("Enter the id of the task you want to complete and remove:");

for(Task t: tasks) {

System.out.println("Id: "+ t.getId() + " Description: " + t.getDescription() + " Deadline: " + t.getDeadline()

+ " Importance: " + t.getImportance());

}

id = input.nextInt();

input.nextLine();

selected = null;

for(Task t: tasks) {

if(t.getId() == id) {

selected = t;

}

}

if(selected != null) {

client.removeTask(selected);

removeTask(selected);

} else {

System.out.println("Task with id:" + id + " does not exist.");

}

break;

case "change":

// Changing a task importance.

System.out.println("Enter the id of the task you want to update importance of:");

for(Task t: tasks) {

System.out.println("Id: "+ t.getId() + " Description: " + t.getDescription() + " Deadline: " + t.getDeadline()

+ " Importance: " + t.getImportance());

}

id = input.nextInt();

input.nextLine();

selected = null;

for(Task t: tasks) {

if(t.getId() == id) {

selected = t;

}

}

if(selected != null) {

System.out.println("Enter the new importance");

importance = input.nextShort();

input.nextLine();

client.removeTask(selected);

selected.setImportance(importance);

client.assignTask(selected);

assignTask(selected);

} else {

System.out.println("Task with id:" + id + " does not exist.");

}

break;

case "show":

// Show tasks in importance order.

tasks.sort(Task.ImportanceComparator);

for(Task t: tasks) {

System.out.println(t.getDescription() + "\nDeadline: " + t.getDeadline()

+ "\nImportance: " + t.getImportance());

}

break;

case "nearby":

// Show tasks in deadline time closest order.

tasks.sort(Task.DeadlineComparator);

for(Task t: tasks) {

System.out.println(t.getDescription() + "\nDeadline: " + t.getDeadline()

+ "\nImportance: " + t.getImportance());

}

break;

case "help":

// Show help if needed.

System.out.println("=====================================================\n"

+ "Please enter a command:\n"

+ "logout\n"

+ "complete -- Complete a task assigned to you.\n"

+ "change -- Change importance of atask assigned to you.\n"

+ "show -- Show all tasks assigned to you in order of importance.\n"

+ "nearby -- Show all tasks assigned to you in order of closest.\n"

+ "help -- Print this information."

+ "Assignee Commands:\n"

+ "assign -- Assign some teammate a job.\n"

+ "remove -- Remove a teammate."

);

break;

case "assign":

// Assign a task to a teammate.

System.out.println("Enter the id of the teammate you want to assign job to:");

for(int i=0;i < maxTeammates;i++) {

if(teammates[i] == null) {

continue;

}

System.out.println("Id: " + teammates[i].getId() + " Name: " + teammates[i].getName()

+ " Username: " + teammates[i].getUsername()

+ " Number of Tasks Assigned: " + teammates[i].getTaskNumber());

}

id = input.nextInt();

input.nextLine();

selectedTeammate = ((ClientManager) client).getTeammate(id);

if(selectedTeammate != null) {

taskId = ((ClientManager) client).getLastTaskId(((ClientManager) client).getTeammate(id)) + 1;

System.out.print("Description of task:\t");

description = input.nextLine();

DateTimeFormatter dtf = DateTimeFormatter.ofPattern("HH:mm dd.MM.yyyy");

System.out.print("Deadline of task: (In format like " + dtf.format(LocalDateTime.now()) + ")\t");

deadline = input.nextLine();

System.out.print("Importance of task: (An integer)\t");

importance = input.nextShort();

input.nextLine();

client.assignTask(new Task(taskId, description, deadline, importance, id));

assignTask(new Task(taskId, description, deadline, importance, id));

System.out.println("Task assigned successfully.");

} else {

System.out.println("Teammate with id:" + id + " does not exist.");

}

break;

case "remove":

// Remove a teammate.

System.out.println("Enter the id of the teammate you want to remove:");

for(int i=0;i < maxTeammates;i++) {

if(teammates[i] == null) {

continue;

}

System.out.println("Id: " + teammates[i].getId() + "\tName: " + teammates[i].getName()

+ "\tUsername: " + teammates[i].getUsername()

+ "\tNumber of Tasks Assigned: " + teammates[i].getTaskNumber());

}

id = input.nextInt();

input.nextLine();

selectedTeammate = new Teammate(id, "", "", "");

client.removeTeammate(selectedTeammate);

removeTeammate(selectedTeammate);

break;

default:

// Show help if unknown command entered.

System.out.println("Command \"" + answer + "\" does not exist. Try this ones:\n"

+ "=====================================================\n"

+ "Please enter a command:\n"

+ "logout\n"

+ "complete -- Complete a task assigned to you.\n"

+ "change -- Change importance of atask assigned to you.\n"

+ "show -- Show all tasks assigned to you in order of importance.\n"

+ "nearby -- Show all tasks assigned to you in order of closest.\n"

+ "help -- Print this information."

+ "Assignee Commands:\n"

+ "assign -- Assign some teammate a job.\n"

+ "remove -- Remove a teammate."

);

}

}

}

void run() {

// The main run loop.

System.out.println("\n\nWelcome to Teammate Manager\n"

+ "====================================================="

);

// As long as app state is not close loop should run. Otherwise "close()" function handles exiting app.

while(true) {

switch(stateOfApp) {

case Menu:

menu();

break;

case Create:

create();

break;

case Login:

login();

break;

case User:

user();

break;

case Assignor:

assignor();

break;

case Close:

close();

break;

}

}

}

void close() {

// Close console input.

input.close();

// Save database.

super.close();

// Exit the process.

System.exit(0);

}

}

**GuiApp.java:**

package app;

import java.awt.BorderLayout;

import java.awt.CardLayout;

import java.awt.Color;

import java.awt.Component;

import java.awt.Dimension;

import java.awt.FlowLayout;

import java.awt.Font;

import java.awt.FontFormatException;

import java.awt.GridLayout;

import java.awt.event.MouseAdapter;

import java.awt.event.MouseEvent;

import java.awt.event.WindowAdapter;

import java.awt.event.WindowEvent;

import java.io.File;

import java.io.IOException;

import java.time.LocalDateTime;

import java.time.format.DateTimeFormatter;

import java.util.ArrayList;

import java.util.Enumeration;

import javax.swing.BorderFactory;

import javax.swing.Box;

import javax.swing.BoxLayout;

import javax.swing.ImageIcon;

import javax.swing.JButton;

import javax.swing.JCheckBox;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JOptionPane;

import javax.swing.JPanel;

import javax.swing.JPasswordField;

import javax.swing.JScrollPane;

import javax.swing.JTable;

import javax.swing.JTextField;

import javax.swing.SwingUtilities;

import javax.swing.UIManager;

import javax.swing.plaf.FontUIResource;

import javax.swing.table.DefaultTableModel;

public class GuiApp extends App {

// Global color palette.

private final int darkBlueColor = 0x0097a7;

private final int blueColor = 0x00bcd4;

private final int lightBlueColor = 0xb2ebf2;

private final int whiteColor = 0xffffff;

private final int dirtyWhiteColor = 0xf6f6f6;

private final int backgroundColor = 0xd8d8d8;

private final int orangeColor = 0xff5722;

private final int lightOrangeColor = 0xff6332;

private final int greyColor = 0x757575;

private final int lightGreyColor = 0x838383;

private final int redColor = 0xdc1a3c;

private final int fumeWhite = 0xe0e0e0;

// Get the default system font. Will be used as global font for every GUI element.

private Font font = new JLabel().getFont();

// Main window.

private JFrame frame = new JFrame();

// CardLayout panels.

private JPanel views = new JPanel();

private JPanel menuView = new JPanel();

private JPanel createView = new JPanel();

private JPanel loginView = new JPanel();

// Setting a buttons UI look, colors.

private void setBtnUI(JButton btn, int backgroundColor, int foregroundColor, int fontSize, String iconPath, int hoverColor) {

// Set colors.

btn.setBackground(new Color(backgroundColor));

btn.setForeground(new Color(foregroundColor));

// Set font size by deriving it.

btn.setFont(btn.getFont().deriveFont(fontSize));

// Set a icon for button.

btn.setIcon(new ImageIcon(iconPath));

// Hovering effect.

btn.addMouseListener(new MouseAdapter() {

@Override

public void mouseEntered(MouseEvent me) {

btn.setBackground(new Color(hoverColor));

}

@Override

public void mouseExited(MouseEvent me) {

btn.setBackground(new Color(backgroundColor));

}

});

}

// A nested class for the layout when a normal teammate logins.

private class UserView extends JPanel {

private static final long serialVersionUID = -5907052262175253840L;

// List of tasks assigned to logged user.

protected ArrayList<Task> tasks;

// Will be shown on top left corner of app. Name of logged teammate.

protected JLabel nameLabel = new JLabel();

// Action buttons related to logged user.

protected JButton logoutBtn = new JButton("Logout");

// Remove selected task from table.

protected JButton completeBtn = new JButton("Complete Task");

// Change importance of selected task from table.

protected JButton changeBtn = new JButton("Change Importance");

// Sort tasks on table by either importance or closest (deadline).

protected JButton sortBtn = new JButton("Sort by Closest");

// Will hold action buttons on a horizontal layout.

protected JPanel editButtons = new JPanel();

// Used for JTable. To make every cell non-editable, the method "isCellEditable" is overridden.

protected DefaultTableModel model = new DefaultTableModel(new String[] {

"Task Id", "Description", "Deadline", "Importance"

}, 0) {

private static final long serialVersionUID = -8290888012737743114L;

@Override

public boolean isCellEditable(int row, int column) {

return false;

}

};

// Create the JTable for tasks.

protected JTable tasksTable = new JTable(model);

UserView() {

nameLabel.setIcon(new ImageIcon("icons/outline\_account\_circle\_white.png"));

nameLabel.setOpaque(true);

nameLabel.setBackground(new Color(lightBlueColor));

nameLabel.setForeground(new Color(whiteColor));

nameLabel.setBorder(BorderFactory.createEmptyBorder(5, 5, 5, 5));

setBtnUI(logoutBtn, greyColor, whiteColor, 12, "icons/outline\_exit\_to\_app\_white.png", lightGreyColor);

setBtnUI(completeBtn, orangeColor, whiteColor, 12, "icons/outline\_done\_white.png", lightOrangeColor);

setBtnUI(changeBtn, greyColor, whiteColor, 12, "icons/outline\_edit\_white.png", lightGreyColor);

setBtnUI(sortBtn, orangeColor, whiteColor, 12, "icons/outline\_import\_export\_white.png", lightOrangeColor);

logoutBtn.addActionListener(e -> {

cards.show(views, "menu");

loggedAccount = null;

});

tasksTable.setBackground(new Color(fumeWhite));

setBackground(new Color(backgroundColor));

completeBtn.addActionListener(e -> {

int row = tasksTable.getSelectedRow();

if(row != -1) {

int id = (int) model.getValueAt(row, 0);

ArrayList<Task> tasks = loggedAccount.getTasks();

Task selected = null;

for(Task t: tasks) {

if(t.getId() == id) {

selected = t;

}

}

if(selected != null) {

model.removeRow(row);

client.removeTask(selected);

removeTask(selected);

}

}

});

changeBtn.addActionListener(e -> {

int row = tasksTable.getSelectedRow();

if(row != -1) {

int id = (int) model.getValueAt(row, 0);

ArrayList<Task> tasks = loggedAccount.getTasks();

Task selected = null;

for(Task t: tasks) {

if(t.getId() == id) {

selected = t;

}

}

if(selected != null) {

String answer = JOptionPane.showInputDialog(frame,"Enter new importance:");

if(answer != "") {

short importance = Short.parseShort(answer);

model.setValueAt(importance, row, 3);

client.removeTask(selected);

selected.setImportance(importance);

client.assignTask(selected);

assignTask(selected);

}

}

}

});

sortBtn.addActionListener(e -> {

String s = sortBtn.getText();

if(s.equals("Sort by Closest")) {

tasks.sort(Task.DeadlineComparator);

sortBtn.setText("Sort by Importance");

}

if(s.equals("Sort by Importance")) {

tasks.sort(Task.ImportanceComparator);

sortBtn.setText("Sort by Closest");

}

model.setRowCount(0);

for(Task t: tasks) {

model.addRow(new Object[] {

t.getId(), t.getDescription(), t.getDeadline(), t.getImportance()

});

}

});

JPanel userCard = new JPanel();

userCard.add(nameLabel);

userCard.add(logoutBtn);

userCard.setLayout(new FlowLayout(FlowLayout.LEFT, 50, 0));

userCard.setBackground(new Color(backgroundColor));

editButtons.add(completeBtn);

editButtons.add(changeBtn);

editButtons.add(sortBtn);

editButtons.setLayout(new FlowLayout(FlowLayout.CENTER, 50, 0));

editButtons.setBackground(new Color(backgroundColor));

setLayout(new BorderLayout(20, 20));

add(userCard, BorderLayout.NORTH);

add(new JScrollPane(tasksTable), BorderLayout.CENTER);

add(editButtons, BorderLayout.SOUTH);

}

// Will be used to refresh model whenever needed. (When new user logins, model changes.)

public void render() {

nameLabel.setText(loggedAccount.getName());

tasks = ((ClientManager) client).getTasks(loggedAccount);

tasks.sort(Task.ImportanceComparator);

model.setRowCount(0); // Empty the table.

// Build the table from tasks assigned to logged account.

for(Task t: tasks) {

model.addRow(new Object[] {

t.getId(), t.getDescription(), t.getDeadline(), t.getImportance()

});

}

}

}

// A nested class for the layout when a assignor teammate logins.

// All the normal user functionality except there is actions such as:

// - Assigning a teammate a task.

// - Removing a teammate.

// This is why it extends UserView.

private class AssignorView extends UserView {

private static final long serialVersionUID = 5419985131438157861L;

// Holds all of the teammates available on the application.

private Teammate[] teammates;

// This buttons are for the mentioned added functionality.

// They will be added to inherited button group "editButtons".

private JButton assignBtn = new JButton("Assign a task");

private JButton removeBtn = new JButton("Remove a teammate");

// Both assigning a teammate a task and removing a teammate should be done on a separate window than main window.

private JFrame assignFrame = new JFrame();

private JFrame removeFrame = new JFrame();

// Table model for teammates that will be show in frames above.

// Same overriden for making cells non-editable.

private DefaultTableModel teammateModel = new DefaultTableModel(new String[] {

"Id", "Name", "Username", "Assignor", "Number of Tasks"

}, 0) {

private static final long serialVersionUID = 4060675525315431400L;

@Override

public boolean isCellEditable(int row, int column) {

return false;

}

};

// Separate tables for showing available teammates from model.

private JTable assignTable = new JTable(teammateModel);

private JTable removeTable = new JTable(teammateModel);

AssignorView() {

// Initialize same GUI elements from UserView.

super();

// Prepare Assign Frame

JLabel assignHeader = new JLabel("Assign a task to a teammate");

// Set the header label UI look.

assignHeader.setOpaque(true);

assignHeader.setForeground(new Color(darkBlueColor));

assignHeader.setBackground(new Color(backgroundColor));

assignHeader.setHorizontalAlignment(JLabel.CENTER);

assignHeader.setFont(assignHeader.getFont().deriveFont(24f));

assignHeader.setAlignmentX(Component.CENTER\_ALIGNMENT);

// Set the button UI's.

setBtnUI(assignBtn, greyColor, whiteColor, 12, "icons/outline\_assignment\_white.png", lightGreyColor);

setBtnUI(removeBtn, orangeColor, whiteColor, 12, "icons/outline\_delete\_white.png", lightOrangeColor);

// Create a simple form for new task to be assigned.

JPanel form = new JPanel();

JLabel taskDescriptionLabel = new JLabel("Task Description:");

JTextField taskDescriptionField = new JTextField();

DateTimeFormatter dtf = DateTimeFormatter.ofPattern("HH:mm dd.MM.yyyy");

JLabel taskDeadlineLabel = new JLabel("Task Deadline (In format like \"" + dtf.format(LocalDateTime.now()) + "\"):");

JTextField taskDeadlineField = new JTextField();

JLabel taskImportanceLabel = new JLabel("Task Importance:");

JTextField taskImportanceField = new JTextField();

JLabel taskAssigneeIdLabel = new JLabel("Task Assignee Id:");

JTextField taskAssigneeIdField = new JTextField();

form.add(taskDescriptionLabel);

form.add(taskDescriptionField);

form.add(taskDeadlineLabel);

form.add(taskDeadlineField);

form.add(taskImportanceLabel);

form.add(taskImportanceField);

form.add(taskAssigneeIdLabel);

form.add(taskAssigneeIdField);

form.setLayout(new GridLayout(4, 2, 10, 10));

JButton assignConfirmBtn = new JButton("Assign");

setBtnUI(assignConfirmBtn, orangeColor, whiteColor, 12, "icons/outline\_done\_white.png", lightOrangeColor);

assignConfirmBtn.addActionListener(e -> {

String answer = taskAssigneeIdField.getText();

if(answer != "") {

int id = Integer.parseInt(answer);

Teammate selectedTeammate = ((ClientManager) client).getTeammate(id);

if(selectedTeammate != null) {

int taskId = ((ClientManager) client).getLastTaskId(((ClientManager) client).getTeammate(id)) + 1;

String description = taskDescriptionField.getText();

String deadline = taskDeadlineField.getText();

short importance = Short.parseShort(taskImportanceField.getText());

render();

client.assignTask(new Task(taskId, description, deadline, importance, id));

assignTask(new Task(taskId, description, deadline, importance, id));

}

taskDescriptionField.setText("");

taskDeadlineField.setText("");

taskImportanceField.setText("");

taskAssigneeIdField.setText("");

render();

}

});

// Create a new panel so we can give margins for assign window.

JPanel assignPanel = new JPanel();

assignPanel.setLayout(new GridLayout(4, 1, 0, 20));

assignPanel.setBorder(BorderFactory.createEmptyBorder(100, 100, 100, 100));

assignPanel.add(assignHeader);

assignPanel.add(new JScrollPane(assignTable));

assignPanel.add(form);

assignPanel.add(assignConfirmBtn);

assignFrame.add(assignPanel);

assignFrame.setMinimumSize(new Dimension(1400, 900));

assignFrame.setTitle("Team Manager - Assign Task");

assignFrame.setIconImage(new ImageIcon("icons/outline\_people\_white.png").getImage());

// Additional functionality buttons for pop-up frames.

assignBtn.addActionListener(e -> {

assignFrame.setVisible(true);

});

removeBtn.addActionListener(e -> {

removeFrame.setVisible(true);

});

assignFrame.addWindowListener(new WindowAdapter() {

public void windowClosing(WindowEvent e) {

// Clear every input when window closes.

assignFrame.dispose();

taskDescriptionField.setText("");

taskDeadlineField.setText("");

taskImportanceField.setText("");

taskAssigneeIdField.setText("");

}

});

// Prepare Remove Frame

JLabel removeHeader = new JLabel("Assign a task to a teammate");

JButton removeConfirmBtn = new JButton("Remove selected teammate");

removeHeader.setOpaque(true);

removeHeader.setForeground(new Color(darkBlueColor));

removeHeader.setBackground(new Color(backgroundColor));

removeHeader.setHorizontalAlignment(JLabel.CENTER);

removeHeader.setFont(assignHeader.getFont().deriveFont(24f));

removeHeader.setAlignmentX(Component.CENTER\_ALIGNMENT);

BorderLayout layout = (BorderLayout)getLayout();

remove(layout.getLayoutComponent(BorderLayout.SOUTH));

add(editButtons, BorderLayout.SOUTH);

setBtnUI(removeConfirmBtn, orangeColor, whiteColor, 12, "icons/outline\_done\_white.png", lightOrangeColor);

removeConfirmBtn.addActionListener(e -> {

// Get the selected teammate and remove it from model, ClientManager and sqlite database.

int row = removeTable.getSelectedRow();

if(row != -1) {

int id = (int) teammateModel.getValueAt(row, 0);

Teammate selectedTeammate = new Teammate(id, "", "", "");

teammateModel.removeRow(row);

client.removeTeammate(selectedTeammate);

removeTeammate(selectedTeammate);

// If account deleted is currently logged account show menu in main window.

if(id == loggedAccount.getId()) {

cards.show(views, "menu");

}

}

});

// Create a panel so we can add border to remove window.

JPanel removePanel = new JPanel();

removePanel.setLayout(new BoxLayout(removePanel, BoxLayout.Y\_AXIS));

removePanel.setBorder(BorderFactory.createEmptyBorder(0, 0, 10, 0));

removePanel.add(removeHeader);

removePanel.add(new JScrollPane(removeTable));

removePanel.add(removeConfirmBtn);

removeFrame.add(removePanel);

removeFrame.setMinimumSize(new Dimension(1400, 900));

removeFrame.setTitle("Team Manager - Remove Teammate");

removeFrame.setIconImage(new ImageIcon("icons/outline\_people\_white.png").getImage());

// Place added functionality buttons of assignor account to the "editButton" button group inherited.

editButtons.add(assignBtn);

editButtons.add(removeBtn);

}

// Will be used to refresh model whenever needed. (When new user logins, model changes.)

public void render() {

// First of all call the method in super class to load necessary updates.

super.render();

teammates = ((ClientManager) client).getTeammates();

teammateModel.setRowCount(0); // Empty the table.

// Build model from all the teammates in system.

for(int i = 0; i < maxTeammates; i++) {

if(teammates[i] == null) {

continue;

}

teammateModel.addRow(new Object[] {

teammates[i].getId(), teammates[i].getName(), teammates[i].getUsername(), teammates[i].isAssignor(), teammates[i].getTaskNumber()

});

}

}

}

// Will hold instances to classes above that we extended from JPanel.

private JPanel userView;

private JPanel assignorView;

// Global CardLayout of main window.

private CardLayout cards = new CardLayout();

GuiApp(String databasePath) {

// Super class constructor call loads database as intended.

super(databasePath);

// Try to load custom font from file.

try {

File f = new File("fonts/Montserrat-Regular.ttf");

font = Font.createFont(Font.TRUETYPE\_FONT, f);

} catch(IOException e) {

System.out.println("Couldn't load custom fonts.");

e.printStackTrace();

} catch(FontFormatException e) {

System.out.println("Custom font format did not match.");

e.printStackTrace();

}

// Change default font.

Enumeration<Object> keys = UIManager.getDefaults().keys();

FontUIResource fontResource = new FontUIResource(font.deriveFont(14f));

// Change fonts of all Text elements in Swing.

while (keys.hasMoreElements()) {

Object key = keys.nextElement();

Object value = UIManager.get (key);

if (value instanceof javax.swing.plaf.FontUIResource)

UIManager.put(key, fontResource);

}

// Instantiate JPanel subclasses.

userView = new UserView();

assignorView = new AssignorView();

}

// Set up menu view. (Login or Create Account)

void setMenuView() {

JLabel header = new JLabel("Welcome to Team Manager");

JButton loginBtn = new JButton("Login");

JButton createBtn = new JButton("Create an Account");

header.setOpaque(true);

header.setForeground(new Color(whiteColor));

header.setBackground(new Color(darkBlueColor));

header.setHorizontalAlignment(JLabel.CENTER);

header.setFont(header.getFont().deriveFont(24f));

header.setAlignmentX(Component.CENTER\_ALIGNMENT);

setBtnUI(loginBtn, orangeColor, whiteColor, 14, "icons/outline\_exit\_to\_app\_white.png", lightOrangeColor);

setBtnUI(createBtn, greyColor, whiteColor, 14, "icons/outline\_person\_add\_white.png", lightGreyColor);

// Change cards when a option is choosen.

loginBtn.addActionListener(e -> cards.show(views, "login"));

createBtn.addActionListener(e -> cards.show(views, "create"));

menuView.add(header);

// Create a button group for both login and create buttons.

// Used to set a border around buttons.

JPanel buttons = new JPanel();

buttons.add(loginBtn);

buttons.add(createBtn);

buttons.setLayout(new GridLayout(2, 1, 0, 20));

buttons.setBorder(BorderFactory.createEmptyBorder(60, 400, 60, 400));

buttons.setBackground(new Color(backgroundColor));

menuView.add(buttons);

menuView.setLayout(new GridLayout(2, 1, 100, 100));

menuView.setBackground(new Color(backgroundColor));

}

// Set up new account creating view.

void setCreateView() {

JLabel header = new JLabel("Create an Account");

// Label to show errors.

JLabel status = new JLabel();

// Necessary form fields.

JLabel nameLabel = new JLabel("Name:");

JTextField nameField = new JTextField();

JLabel usernameLabel = new JLabel("Username:");

JTextField usernameField = new JTextField();

JLabel passwordLabel = new JLabel("Password:");

JPasswordField passwordField = new JPasswordField();

JLabel passwordAgainLabel = new JLabel("Password Again:");

JPasswordField passwordAgainField = new JPasswordField();

JLabel assignorLabel = new JLabel("Are you an assignor?");

JCheckBox assignorCheck = new JCheckBox();

JButton createBtn = new JButton("Create");

JButton cancelBtn = new JButton("Cancel");

header.setOpaque(true);

header.setForeground(new Color(darkBlueColor));

header.setBackground(new Color(backgroundColor));

header.setHorizontalAlignment(JLabel.CENTER);

header.setFont(header.getFont().deriveFont(24f));

header.setAlignmentX(Component.CENTER\_ALIGNMENT);

status.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));

status.setOpaque(true);

status.setVisible(false);

status.setBackground(new Color(redColor));

status.setForeground(new Color(whiteColor));

status.setFont(status.getFont().deriveFont(16f));

nameField.setBackground(new Color(dirtyWhiteColor));

nameField.setBorder(BorderFactory.createLineBorder(new Color(blueColor)));

nameField.setToolTipText("Enter your name...");

usernameField.setBackground(new Color(dirtyWhiteColor));

usernameField.setBorder(BorderFactory.createLineBorder(new Color(blueColor)));

usernameField.setToolTipText("Enter your username...");

passwordField.setBackground(new Color(dirtyWhiteColor));

passwordField.setBorder(BorderFactory.createLineBorder(new Color(blueColor)));

passwordField.setToolTipText("Enter your password...");

passwordAgainField.setBackground(new Color(dirtyWhiteColor));

passwordAgainField.setBorder(BorderFactory.createLineBorder(new Color(blueColor)));

passwordAgainField.setToolTipText("Enter your password again...");

assignorCheck.setBackground(new Color(backgroundColor));

setBtnUI(createBtn, orangeColor, whiteColor, 14, "icons/outline\_person\_add\_white.png", lightOrangeColor);

createBtn.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));

setBtnUI(cancelBtn, greyColor, whiteColor, 14, "icons/outline\_cancel\_white.png", lightGreyColor);

cancelBtn.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));

createBtn.addActionListener(e -> {

String username, password, name;

boolean assignor;

// Form validation on new account.

username = usernameField.getText();

if(!((ClientManager) client).usernameExists(username)) {

// If username is not taken continue.

password = new String(passwordField.getPassword());

if((new String(passwordAgainField.getPassword())).equals(password)) {

// If passwords are same then create new account.

name = nameField.getText();

assignor = assignorCheck.isSelected();

Teammate t = new Teammate(((ClientManager) client).lastTeammateId, name, username, password);

t.setAssignor(assignor);

client.addTeammate(t);

addTeammate(t);

// Clear fields if validated and account added.

nameField.setText("");

usernameField.setText("");

passwordField.setText("");

passwordAgainField.setText("");

assignorCheck.setSelected(false);

status.setText("");

status.setVisible(false);

// Change view to main menu.

cards.show(views, "menu");

} else {

// Otherwise password check fails.

status.setText("Passwords do not match!");

status.setVisible(true);

}

} else {

status.setText("Username already taken!");

status.setVisible(true);

}

});

cancelBtn.addActionListener(e -> {

// Cancels creating account and changes view to main menu.

cards.show(views, "menu");

// Clear fields.

nameField.setText("");

usernameField.setText("");

passwordField.setText("");

passwordAgainField.setText("");

assignorCheck.setSelected(false);

status.setText("");

status.setVisible(false);

});

// Group all fields in a form.

JPanel form = new JPanel();

form.add(nameLabel);

form.add(nameField);

form.add(usernameLabel);

form.add(usernameField);

form.add(passwordLabel);

form.add(passwordField);

form.add(passwordAgainLabel);

form.add(passwordAgainField);

form.add(assignorLabel);

form.add(assignorCheck);

form.add(createBtn);

form.add(cancelBtn);

form.setLayout(new GridLayout(8, 2, 20, 20));

form.setBackground(new Color(backgroundColor));

createView.add(header);

createView.add(Box.createRigidArea(new Dimension(0, 10)));

createView.add(status);

createView.add(Box.createRigidArea(new Dimension(0, 10)));

createView.add(form);

createView.setLayout(new BoxLayout(createView, BoxLayout.Y\_AXIS));

createView.setBorder(BorderFactory.createEmptyBorder(150, 200, 200, 200));

createView.setBackground(new Color(backgroundColor));

}

// Set up login view.

void setLoginView() {

JLabel header = new JLabel("Login");

JLabel status = new JLabel();

// Necessary form fields.

JLabel usernameLabel = new JLabel("Username:");

JTextField usernameField = new JTextField();

JLabel passwordLabel = new JLabel("Password:");

JPasswordField passwordField = new JPasswordField();

JButton loginBtn = new JButton("Login");

JButton cancelBtn = new JButton("Cancel");

header.setOpaque(true);

header.setForeground(new Color(darkBlueColor));

header.setBackground(new Color(backgroundColor));

header.setHorizontalAlignment(JLabel.CENTER);

header.setFont(header.getFont().deriveFont(24f));

header.setAlignmentX(Component.CENTER\_ALIGNMENT);

status.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));

status.setOpaque(true);

status.setVisible(false);

status.setBackground(new Color(redColor));

status.setForeground(new Color(whiteColor));

usernameField.setBackground(new Color(dirtyWhiteColor));

usernameField.setBorder(BorderFactory.createLineBorder(new Color(blueColor)));

usernameField.setToolTipText("Enter your username...");

passwordField.setBackground(new Color(dirtyWhiteColor));

passwordField.setBorder(BorderFactory.createLineBorder(new Color(blueColor)));

passwordField.setToolTipText("Enter your password...");

setBtnUI(loginBtn, orangeColor, whiteColor, 14, "icons/outline\_person\_add\_white.png", lightOrangeColor);

setBtnUI(cancelBtn, greyColor, whiteColor, 14, "icons/outline\_cancel\_white.png", lightGreyColor);

loginBtn.addActionListener(e -> {

// Form validation

String username = usernameField.getText();

Teammate t = ((ClientManager) client).getTeammate(username);

if(t != null) {

// If teammate is in ClientManager then account exists.

String password = new String(passwordField.getPassword());

if(password.equals(t.getPassword())) {

// Entered password matches password in database.

loggedAccount = t;

// Clear fields.

usernameField.setText("");

passwordField.setText("");

status.setText("");

status.setSize(0, 0);

status.setVisible(false);

// Select which type of user it is to show next view and then render it.

// Necessary data is filled before card changes.

if(t.isAssignor()) {

((AssignorView) assignorView).render();

cards.show(views, "assignor");

} else {

((UserView) userView).render();

cards.show(views, "user");

}

} else {

// Password in client database and entered does not match.

status.setText("Password incorrect!");

status.setVisible(true);

}

} else {

// Account doesn't exist.

status.setText("User doesn't exist!");

status.setVisible(true);

}

});

cancelBtn.addActionListener(e -> {

// Back to main menu.

cards.show(views, "menu");

// Clear fields.

usernameField.setText("");

passwordField.setText("");

status.setText("");

status.setVisible(false);

});

// Group fields in form.

JPanel form = new JPanel();

form.add(usernameLabel);

form.add(usernameField);

form.add(passwordLabel);

form.add(passwordField);

form.add(loginBtn);

form.add(cancelBtn);

form.setLayout(new GridLayout(3, 2, 20, 50));

form.setBackground(new Color(backgroundColor));

loginView.add(header);

loginView.add(Box.createRigidArea(new Dimension(0, 10)));

loginView.add(status);

loginView.add(Box.createRigidArea(new Dimension(0, 10)));

loginView.add(form);

loginView.setLayout(new BoxLayout(loginView, BoxLayout.Y\_AXIS));

loginView.setBorder(BorderFactory.createEmptyBorder(200, 200, 200, 200));

loginView.setBackground(new Color(backgroundColor));

}

@Override

void run() {

// Don't let items disappear on smaller sizes.

frame.setMinimumSize(new Dimension(1400, 900));

// Title and Title Icon.

frame.setTitle("Team Manager");

frame.setIconImage(new ImageIcon("icons/outline\_people\_white.png").getImage());

// Set the card layout and main panel.

views.setLayout(cards);

views.setBorder(BorderFactory.createEmptyBorder(50, 50, 50, 50));

views.setBackground(new Color(backgroundColor));

// Add each one of the cards to panel.

views.add(menuView, "menu");

views.add(createView, "create");

views.add(loginView, "login");

views.add(userView, "user");

views.add(assignorView, "assignor");

// Start application by showing menu.

cards.show(views, "menu");

// Prepare GUI of each one of the cards.

setMenuView();

setCreateView();

setLoginView();

// Set up main frame.

frame.add(views);

frame.addWindowListener(new WindowAdapter() {

public void windowClosing(WindowEvent e) {

// Save database then end the process.

frame.dispose();

close();

System.exit(0);

}

});

// Show main window.

frame.setVisible(true);

}

}

**Main.java:**

package app;

public class Main {

static App app;

static String databasePath = "database.db"; // Sqlite database name/path.

static boolean isGUI = true; // Whether the application will start in GUI mode or Console mode.

public static void main(String[] args) {

// Parse arguments passed in.

for(String s:args) {

// If there is a "c" flag then start in Console mode.

if(s.equals("-c"))

isGUI = false;

}

// Choose which object to instantiate.

if(isGUI) {

app = new GuiApp(databasePath);

}

else {

app = new ConsoleApp(databasePath);

}

// Run the application.

app.run();

}

}