STU FIIT Object-Oriented Programming (OOP)

Program a dokumentácia

„DIA“

Made by: Meredov Nazar

AIS ID: 122092

Criteria and methods used:

Use of inheritance:

1) The dependence of the Users and its types (User,Admin)

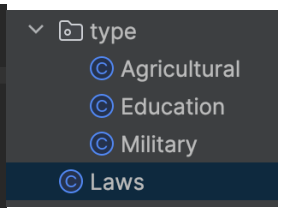
```java
public class Users {  2 usages  2 inheritors  Nazar Faustynn *
    private String login;  3 usages
    private String password;  4 usages

    private String name;  3 usages
    private String surname;  3 usages
    private int age;  3 usages


    public Users(String login,String password,int age,String name,String surname){  2 usages  Nazar Faustynn
        this.login = login;
        this.password = password;
        this.age = age;
        this.name = name;
        this.surname = surname;
    }

    public Users() {  2 usages  Nazar Faustynn
    }
```

```java
public class User extends Users implements UniversalAccount, Serializable {  3 usages  Nazar Faustynn
    private int id;  2 usages

    public User(String login, String password, String name, String surname,int age,int id) {  1 usage  Nazar Faustynn
        super(login, password, age, name, surname);
        this.id=id;
    }
    public User() {  no usages  Nazar Faustynn
    }
```

```java
public class Admin extends Users implements UniversalAccount, Serializable {  3 usages  Nazar Faustynn
    private int score;  2 usages
    private int money;  1 usage

    public Admin(String login, String password, String name, String surname,int age,int score,int money) {  1 usage  Nazar Faustynn
        super(login, password, age, name, surname);
        this.money = money;
        this.score = score;
    }
    public Admin() {  no usages  Nazar Faustynn
    }
```

2) The dependence of the Law and its types. (Military,Education,Agriculture similary)

```java
public class Military extends Laws {  2 usages  Nazar Faustynn
    private static final String type = "M";  no usages
    private final String result;  2 usages
    private final int yes;  2 usages
    private final int no;  2 usages
    public Military(String name, int votesNeeded,String type,String result,int yesCount, int noCount) {  3 usages  Nazar Faustynn
        super(name, votesNeeded);
        this.result=result;
        this.yes=yesCount;
        this.no=noCount;
    }
```




```java
public class Laws {  16 usages  3 inheritors  Nazar Faustynn
    private String name;  2 usages
    private int votesNeeded;  2 usages
    private int votesFor;  3 usages
    private int votesAgainst;  3 usages
    private int yes;  no usages
    private int no;  no usages

    public Laws(String name, int votesNeeded) {  5 usages  Nazar Faustynn
        this.name = name;
        this.votesNeeded = votesNeeded;
        this.votesFor = 0;
        this.votesAgainst = 0;
    }

    public Laws() {  2 usages  Nazar Faustynn

    }
```

Use of Polymorphism:

1) Calling the load() method for the FXMLLoader object in the changeUserRole method .The load() call loads the contents of the FXML file into the root variable. The polymorphism here is that the load() method can load different FXML files depending on what resources have been set for fxmlLoader.

```
this.stage = primaryStage;
FXMLLoader loader = new FXMLLoader(getClass().getResource( name: "/org/example/demo_oop_projectjavafx/FXML/main_page.fxml"));
Parent root = loader.load();
```

2) () -> new User(...) is a lambda expression that implements the UserSupplier functional interface demonstrating the use of polymorphism via a lambda expression.

```
@FXML  new *
protected void RegisterButtonOnAction(ActionEvent event) throws IOException {
    if (isInputValid()) {
        AccountRegistration registration = new AccountRegistration();
        registration.registerAccount(() -> new User(loginArea.getText(), passArea.getText(), nameArea.getText(), Integer.parseInt(oldArea.getText())), ifAdmin.isSelected());
    } else {
        showErrorAlert("Please fill in all fields.");
    }
}
```

```
@FunctionalInterface  1 usage  new *
private interface UserSupplier {
    User get();  1 usage  new *
}
```

The use of agragation:

The MultithreadingClass class contains a controller field that references an object of type UserVoteController.

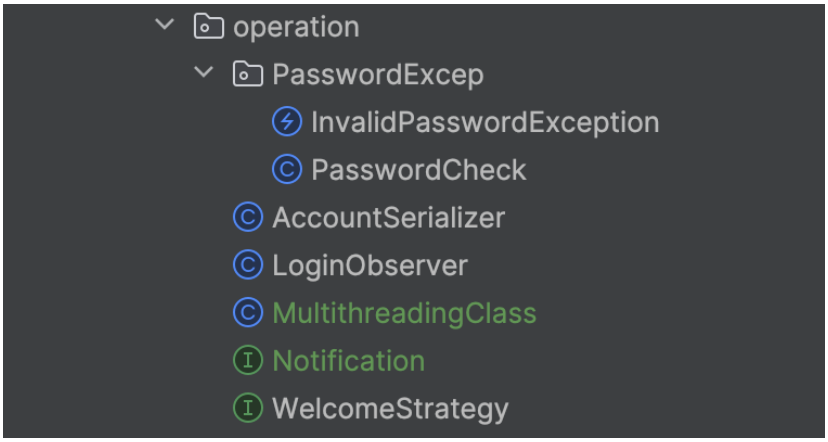
This relation shows that MultithreadingClass aggregates UserVoteController, i.e. it contains a reference to it.

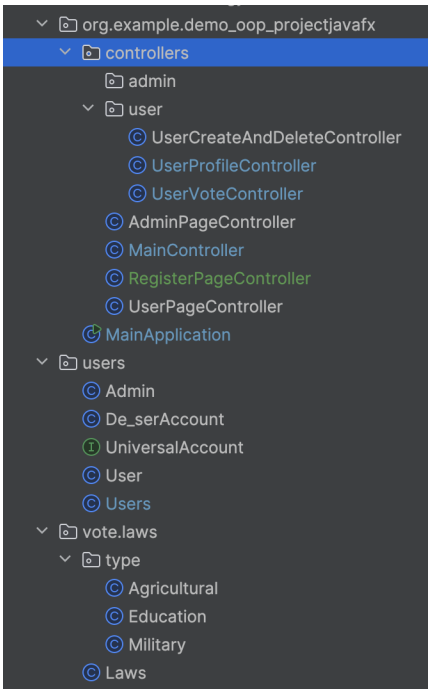
Aggregation here implies that MultithreadingClass uses UserVoteController, but they are not tightly coupled by a lifecycle.

```
public class MultithreadingClass implements Runnable {  3 usages  new *
    private int countYes;  5 usages
    private int countNo;  5 usages
    private Random random = new Random();  2 usages
    private Object lock = new Object();  4 usages
    private UserVoteController controller;  3 usages
```

Separation of application logic from the user interface & Code organised into packages:

All code is divided into a logical part which is stored in the operation package and a custom part in demo_oop_projectjavafx together with controllers.





The use of design patterns:

1) Obbserver:

```
public UniversalAccount observeLogin(String username, String password) throws InvalidPasswordException {  1 usage  Nazar Faustynn
    return findAccount(username, password);
}
```

```
private UniversalAccount findAccount(String username, String password) throws InvalidPasswordException {
    String[] fileList = {USER_LIST_FILE, ADMIN_LIST_FILE};

    for (String file : fileList) {
        try (BufferedReader br = new BufferedReader(new FileReader(file))) {
            String line;
            while ((line = br.readLine()) != null) {
                String[] parts = line.split( regex: ";");
                if (parts.length >= 2 && parts[0].equals(username) && parts[1].equals(password)) {
                    if (!password.matches( regex: "\\d+")) {
                        throw new InvalidPasswordException("Password must contain only digits.");
                    }
                    String[] userInfo = parts[2].split( regex: " ");
                    int old = Integer.parseInt(parts[3]);
                    int id = Integer.parseInt(parts[4]);
                    if (file.equals(USER_LIST_FILE)) {
                        return new User(username, password, userInfo[0], userInfo[1], old, id);
                    } else {
                        int score = id;
                        int money = Integer.parseInt(parts[5]);
                        return new Admin(username, password, userInfo[0], userInfo[1], old, score, money);
                    }
                }
            }
        } catch (IOException e) {
            throw new RuntimeException("Error reading file: " + file, e);
        }
    }
    throw new InvalidPasswordException("User not found");
}
```

2) Strategy:

In the initialize() method of the controller, the setWelcomeStrategy() method is called, which sets the welcome strategy. The printHello() method is then called, which uses the current strategy to display the welcome message on the screen.

```
public interface WelcomeStrategy {  3 usages  1 implementation  Nazar Faustynn
    String getWelcomeMessage();  1 usage  1 implementation  Nazar Faustynn

    default void LoginCheck(String role) {  1 usage  Nazar Faustynn
        System.out.println(role+" was login successful!");
    }
}
```


```
public interface WelcomeStrategy {  3 usages  1 implementation  Nazar Faustynn
    String getWelcomeMessage();  1 usage  1 implementation  Nazar Faustynn

    default void LoginCheck(String role) {  1 usage  Nazar Faustynn
        System.out.println(role+" was login successful!");
    }
}
```


Treatment of emergencies through own exemptions:

```java
public class InvalidPasswordException extends Exception {  11 usages  Nazar Faustynn

    public InvalidPasswordException(String message) { super(message); }

}
```

```java
public static void checkPassword(String password) throws InvalidPasswordException {  no us
    if (password == null || password.isEmpty()) {
        throw new InvalidPasswordException("Password cant be empty!");
    }
    if (password.length() < 3) {
        throw new InvalidPasswordException("Password must have at least 3 characters!");
    }
    if (!password.matches( regex: "[0-9]+")) {
        throw new InvalidPasswordException("Password must have only numbers!");
    }
```

Providing a graphical user interface separate from the application logic:

MultithreadingClass that has an updateGUI() method that is called to update the GUI from another thread using Platform.runLater(). This method uses an instance of UserVoteController to update the GUI text fields (numberYes and numberNo) that display the number of votes for and against.

This way, your GUI is updated asynchronously from another thread, separating it from the main application logic

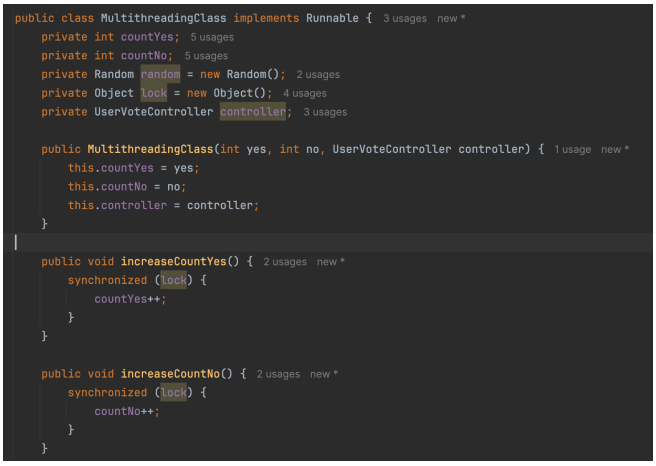
```java
private void updateGUI() {  2 usages  new *
    Platform.runLater(() -> {
        controller.numberYes.setText(Integer.toString(countYes));
        controller.numberNo.setText(Integer.toString(countNo));
    });
}
```

Explicit use of multiplicity:

The Timer class is used to schedule and execute tasks in the background. Timer creates a thread that executes a task at a certain interval.

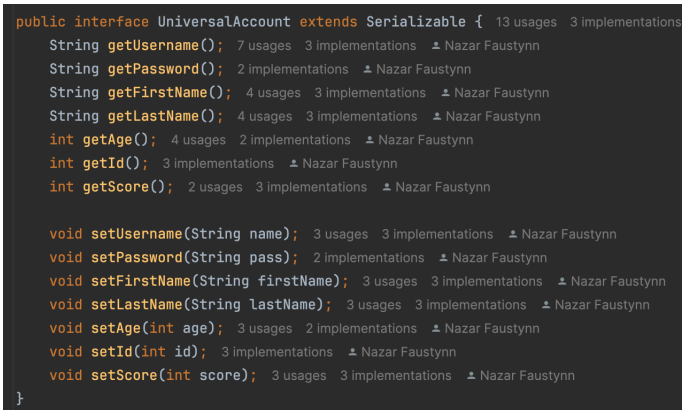
The StartVoting() method creates two timers: timerYes and timerNo, each of which starts a different task. This task randomly generates a voting result (yes or no) with a certain probability, and updates the GUI if necessary via the updateGUI() method.

The MultithreadingClass also has increaseCountYes() and increaseCountNo() methods that increment the vote counters.



The use of genericity in custom classes:

UniversalAccount interface, which defines methods for obtaining and setting various account properties. This interface allows you to create different types of accounts (for example, regular users and administrators) with a common set of features.



```java
public class User extends Users implements UniversalAccount, Serializable {  3 usages  Nazar Fa
    private int id;  2 usages

    public User(String login, String password, String name, String surname,int age,int id) {
        super(login, password, age, name, surname);
        this.id=id;
    }
```

```java
public class Admin extends Users implements UniversalAccount, Serializable {  3 usages  Nazar Faustynn
    private int score;  2 usages
    private int money;  1 usage

    public Admin(String login, String password, String name, String surname,int age,int score,int money) {  1
        super(login, password, age, name, surname);
        this.money = money;
        this.score = score;
    }
```

Explicit use of RTTI:

RTTI used to check if the loaded account is an instance of De_serAccount. If so, it casts the loadedAccount to De_serAccount and updates it; otherwise, it creates a new De_serAccount instance.

```java
public void saveNewDataProfile() throws IOException, ClassNotFoundException {
    String fullName = nameSurnameTextArea.getText();
    String[] nameParts = fullName.split( regex: "\\s+");

    De_serAccount account = null;
    if (loadedAccount instanceof De_serAccount) {
        account = (De_serAccount) loadedAccount;
    } else {
        account = new De_serAccount();
    }

    account.setUsername(loginTextArea.getText());
    account.setPassword(passwordTextArea.getText());
    account.setFirstName(nameParts[0]);
    account.setLastName(nameParts[1]);
    account.setAge(Integer.parseInt(yearTextArea.getText()));
    account.setId(Integer.parseInt(idTextArea.getText()));
    account.setScore(0);
```

Use of nested classes:

MultithreadingClass, and it uses nested anonymous classes to handle timers. These nested classes are created inside the StartVoting() method. They are instances of TimerTask used to start tasks periodically.

In code, using nested classes for timer tasks keeps the logic for starting the voting process within the MultithreadingClass class

```
private void StartVoting() {  1 usage  new *
    Timer timerYes = new Timer();
    timerYes.schedule(new TimerTask() {  new *
        @Override  new *
        public void run() {
            if (random.nextDouble() < 0.5) {
                increaseCountYes();
                updateGUI();
            }
        }
    }, delay: 0, period: 5000);

    Timer timerNo = new Timer();
    timerNo.schedule(new TimerTask() {  new *
        @Override  new *
        public void run() {
            if (random.nextDouble() < 0.25) {
                increaseCountNo();
                updateGUI();
            }
        }
    }, delay: 0, period: 3000);
}
```

Use of lambda expressions:

Lambda expression () -> {...}, which is passed to the runLater() method from the Platform class. The lambda expression here is an anonymous function with no name. It contains a block of code that is executed asynchronously in the JavaFX Application Thread, which allows GUI controls to be updated without blocking the user interface.

```
private void updateGUI() {  2 usages  new *
    Platform.runLater(() -> {
        controller.numberYes.setText(Integer.toString(countYes));
        controller.numberNo.setText(Integer.toString(countNo));
    });
}
```

```
private void StartVoting() {  1 usage  new *
    Timer timerYes = new Timer();
    timerYes.schedule(new TimerTask() {  new *
        @Override  new *
        public void run() {
            if (random.nextDouble() < 0.5) {
                increaseCountYes();
                updateGUI();
            }
        }
    }, delay: 0, period: 5000);

    Timer timerNo = new Timer();
    timerNo.schedule(new TimerTask() {  new *
        @Override  new *
        public void run() {
            if (random.nextDouble() < 0.25) {
                increaseCountNo();
                updateGUI();
            }
        }
    }, delay: 0, period: 3000);
}
```

Use of implicit method implementation in interfaces:

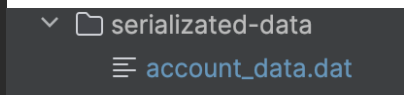
Classes implementing WelcomeStrategy can choose to override the LoginCheck() method if they want a custom implementation, but they are not required to do so. If they don't override it, they'll inherit the default implementation provided in the interface.

```
public interface WelcomeStrategy {  3 usages  1 implementation  Nazar Faustynn
    String getWelcomeMessage();  1 usage  1 implementation  Nazar Faustynn

    default void LoginCheck(String role) {  1 usage  Nazar Faustynn
        System.out.println(role+" was login successful!");
    }
}
```

Use of serialization:

```java
public class AccountSerializer {  5 usages  Nazar Faustynn
    public static void saveUniversalAccount(UniversalAccount account, String filePath) throws IOException {  2 usages
        try (ObjectOutputStream outputStream = new ObjectOutputStream(new FileOutputStream(filePath))) {
            outputStream.writeObject(account);
        }
    }

    public static UniversalAccount loadUniversalAccount(String filePath) throws IOException, ClassNotFoundException {
        try (ObjectInputStream inputStream = new ObjectInputStream(new FileInputStream(filePath))) {
            return (UniversalAccount) inputStream.readObject();
        }
    }
}
```

```java
@FXML  Nazar Faustynn
protected void handleLoginButtonAction(ActionEvent event) throws IOException {
    String username = usernameField.getText();
    String password = passwordField.getText();

    try {
        UniversalAccount universalAccount = loginObserver.observeLogin(username, password);
        De_serAccount account = new De_serAccount();
        account.setUsername(universalAccount.getUsername());
        account.setPassword(universalAccount.getPassword());
        account.setFirstName(universalAccount.getFirstName());
        account.setLastName(universalAccount.getLastName());
        account.setAge(universalAccount.getAge());
        account.setId(universalAccount.getId());
        account.setScore(universalAccount.getScore());
        try {
            AccountSerializer.saveUniversalAccount(account, ACCOUNT_FILE_PATH);
            System.out.println("Data was sucsecful serialized");
        } catch (IOException e) {
            System.err.println("Error serialize data " + e.getMessage());
        }
        if (universalAccount instanceof User) {
            mainApplication.changeUserRole( newRole: "user");
        } else if (universalAccount instanceof Admin) {
            mainApplication.changeUserRole( newRole: "admin");
        } else {
            showErrorAlert("Unknown role");
        }
    } catch (InvalidPasswordException e) {
        showErrorAlert(e.getMessage());
    }
}
```



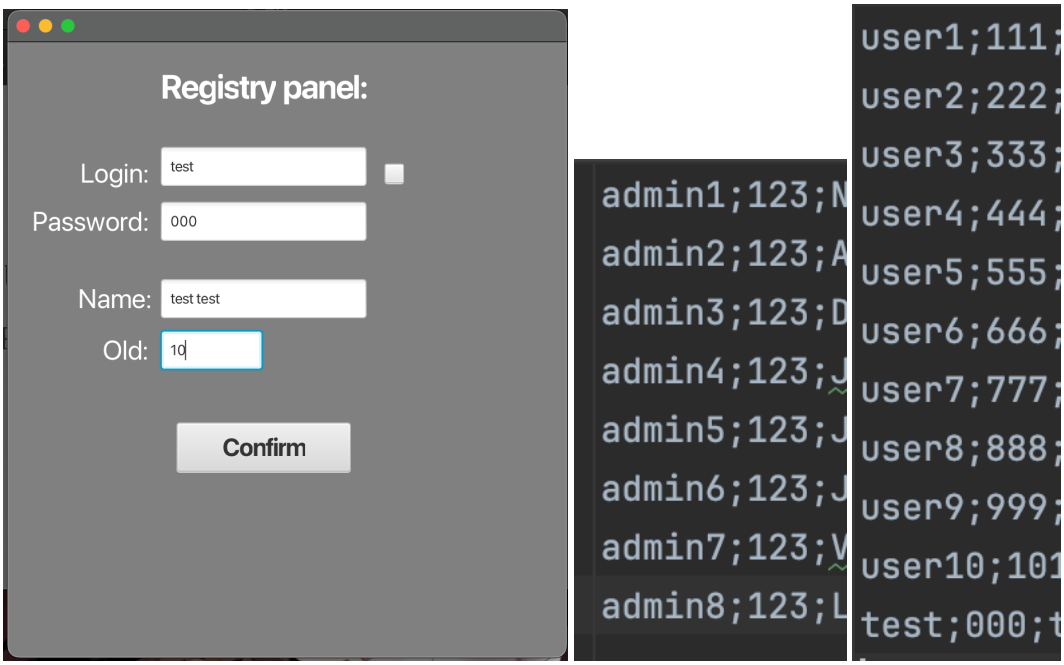
How work with program?:

1) Start Window with login function:

- You can login as User or Admin, below you can read the login and password if you want to log in to one of the accounts.

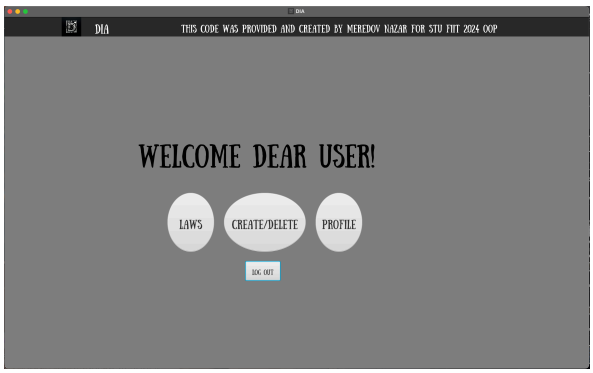
(If you enter an invalid password or username, you will not be able to log in.)

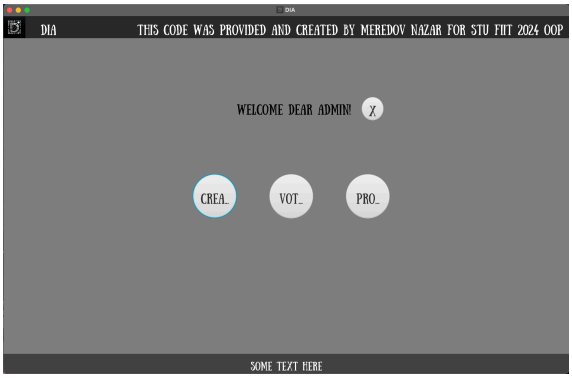
- If you want create new Account you can also do it click on REGISTER button



(Write your desired data and if you want to create an admin account click the checkbox next to the login field and then confirm your action.)

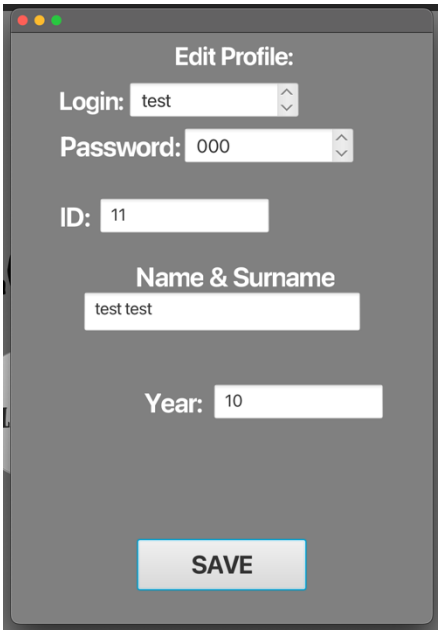
2) User & Admin Views:





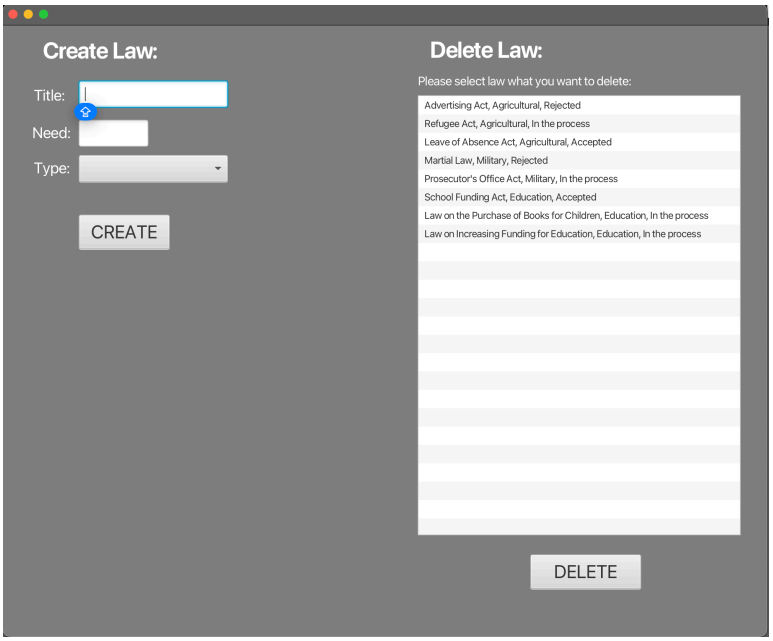
P.s Since the Idea was too big, I dont have time to fully finalize the functional for the Admin, but you can see all the requirements for the project and the essence of the voting through the User.

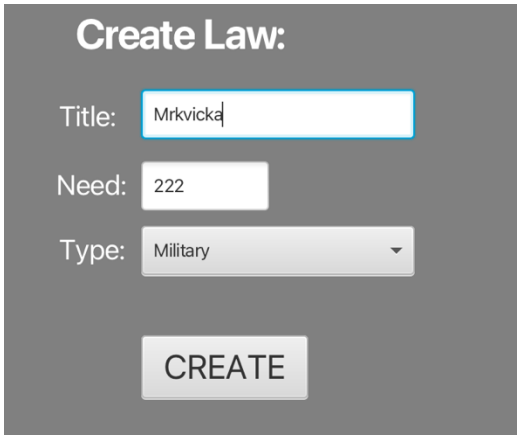
- **For edit profile information you can click button PROFILE**



Edit the information you want and then save it by clicking the SAVE button

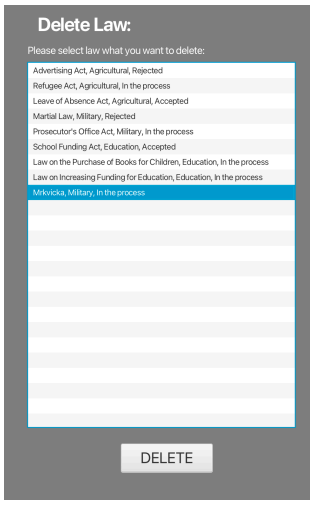
- **For create or delete laws you can click button CREATE/DELETE**



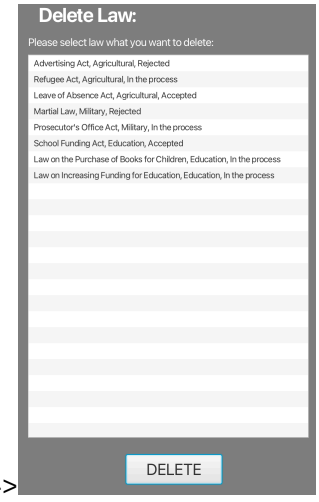


On the left, you can work on creating a new law by entering the name, the number of votes, and the type of law to be put forward. Then click on the CREATE button where the information will be saved and you can work with it.

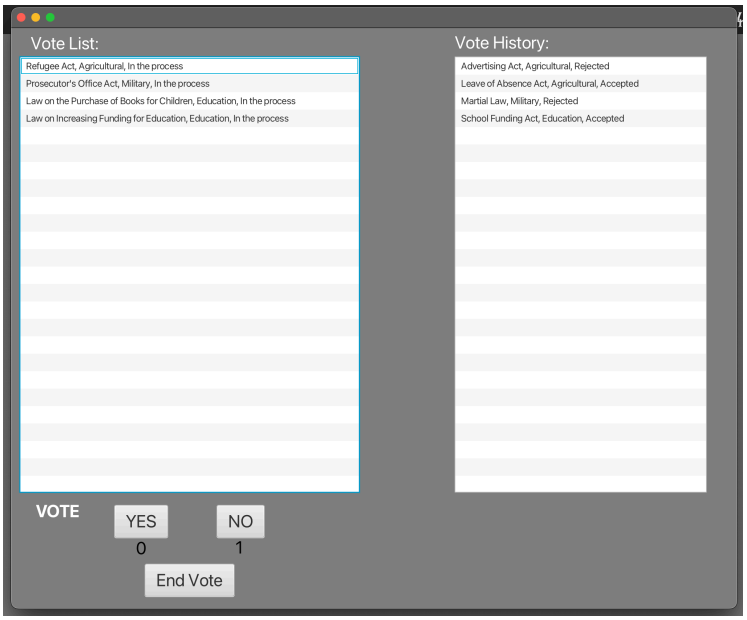
If you want to delete a law, you can select it from the list on the right and then click delete where it will be deleted from the list and from the database.



<-before & after->



- **For VOTE for laws you can click button LAWS**



- *On the right side you can see all the laws whose voting has already been completed and whether successful or not.*
- *On the left side you can select a law from the list and then vote for it yes or no, then send your results by clicking on the button, after which this vote will be closed and displayed in the history on the right. (Also votes are generated automatically from other users).*

Class Diagram: