SOCIAL HERO

SOSU

Simulation Application

FINAL EXAM

Team

HandsomeKoalas

Diogo da Costa Queijo

Nikolay Nikolov

Daniel Ponce Baez

Github

Preface

Section 1 collects all the materials regarding Introduction, background, problem statement, product vision, and strategic analysis.

Section 2 describes the Pre-Game(Sprint 0). In this sector, we explain how we manage to prepare and organise the project as a whole. We also write about our design architecture, schedule, and tools in this section.

Section 3, 4, and 5 documents Sprint 1, 2, and 3 respectively. Those specific sections follow a similar structure which is: Sprint Planning, Daily Meetings, GUI, Data Model, Implementation, Code Examples, Sprint Review, and Sprint Retrospective.

The source code is available at:
https://github.com/haker10/SOSU_HandsomeKoalas_FinalExam2ndSemCS2021B

Introduction

Practice is the best teacher. We were assigned the task by SOSU to create an application that aims to simulate a real platform that social workers use in their practice. They want to use the application as a learning tool in one of their compulsory courses called - "Fællessprog 3".

Background

SOSU Esbjerg is an educational institution that clarifies and educates to the basic level within the social and health area. Students who graduate from schools such as SOSU are more likely to become social workers.

Social workers are people who aim to improve people's lives. By helping with social and interpersonal difficulties, social workers promote human rights and well-being.

Social workers protect people in need and children from injury, they also provide support. One of the requirements of social workers is to know the so-called FS3 which we can define as a protocol through which information is collected about the patient and then this information can easily be shared with health experts and other social workers. FS3 is a mandatory subject in schools such as SOSU.

Problem Statement

Fællessprog 3 or Common Language 3 is a form of managing personal information. FS3 is a protocol that helps health professionals and social workers share information about their patients. Filling in and sharing information is done through the use of several platforms to which students do not have access.

The project aims to create an application that simulates these platforms, an application that will be used by students to play role-play situations.

Product Vision

In this assignment, we have created an application that aims to simulate the use of the applications that are used in the real practice of social workers. With this application teachers will be able to assign tasks to students during class or homework, tasks that will be performed in the application and all data about them will be stored in a database.

Our product should meet the needs of the customer through efficiency and model through CSS interface.

Requirements

The program must:

- Be functional, meaning all the intended functionalities of the program are implemented.
- Separate user access for students, teachers, and administrators(school)
- Ability to create fictional citizens for simulation purposes
- Tools for enabling/training the use of FS3
- Documenting and saving data from fictional citizen
- The System must be divided so each school has its own data

Additional requirements gathered during the sprint reviews:

- Create as many Citizens from a Template as students exist
- Overview of Health Conditions and Functional Abilities
- Admin manages only their school information.

Strategic Analysis

Risk Analysis

The result from the risk analysis shows that the human factor is with low impact on the situation. The technical part is also with very low impact due to the fact we are using such an efficient tool as GitHub. Our big problem could be the miscommunication we can possibly have with the POs. However, if the time is incorrectly estimated this may affect brutally on our Project management.

RISK	RISK management
Human	
Absence	Accepting and controlling the risk: Disease, work, social obligations must be considered. In any case in which a member of the team cannot perform his tasks on schedule. You need to contact the team and make a new schedule
Technical disagreements	Accepting and controlling the risk: The chance of disagreement is very high because of the big number of ways you can solve a problem. In case of disagreement, we will present a vote to decide to resolve the issue.
Technical	
Loss of data	Accepting and controlling the risk: Git will be used for our repository and commits will be made daily.
Merge conflicts	Accepting and controlling the risk: small portion of the code will be split between branches anyhow we will have a GitHub Master who will be the only one who is authorized to merge.
Product Owner	
Miscommunication	Accepting and controlling the risk: Miscommunication is a high-risk factor in our project due the short meetings we have assigned. Taking notes which, we will verify with the Product Owner is a good solution.
Development	
Poor estimates	Avoid the risk: Estimating the time needed is a hard task. Breaking user stories on smallest possible parts is a good practice in case of more accurate time estimating.

Figure1:Risk Analysis

Stakeholder Analysis

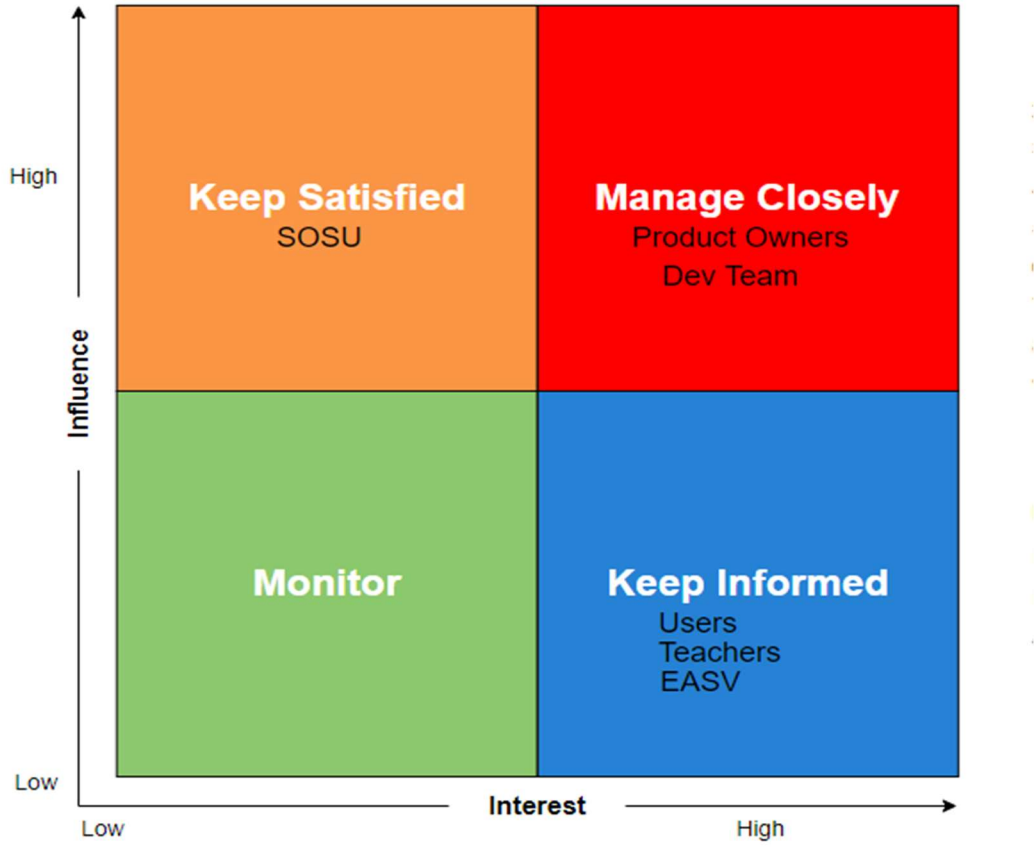


Figure 2: Stakeholder Analysis

Stakeholders:

- SOSU
- Product Owners
- The Developer Team(us)
- Users (students/teachers from SOSU)
- Teachers
- EASV

SOSU has a high influence on the project because they've provided 3 of their teachers to be representatives and owners of the product. For this reason, they are the stakeholders we want to satisfy. Although they have low interest in the project because they are likely to never use the final product.

The owners of the product as well as the developers have a strong influence and a strong interest in the project to be successful. On the one hand, there are POs (teachers from SOSU) for whom it is important that the application is exactly according to their requirements because they want to implement the application in the educational process of their students. On the other hand, the developers (we) have the strongest interest in the project being successful because it would be feedback to us that our study is going according to plan and so success is important for our evaluation.

EASV as well as teachers have a strong interest in the details of the project, so they are in a sector with a strong interest (keep informed), but they still have very little influence on how the project will go and the final result. The Users of the application have a strong interest, but they also have no influence on the project due to the fact that they will see it only when it is completed.

Pre-Game (Sprint 0)

The pre-game stage consisted of planning and scheduling the project. We started as we got introduced to the project and with the team, we defined the architecture, and we made a plan to follow using tools such as Scrum and GitHub. Our team contract was included in the planning and the roles were distributed among the participants.

Project Organisation

The HandsomeKoalas consists of a three-member team from CS2021B class - Diogo, Daniel, and Nikolay. We organised our daily work following Scrum. We met daily in Discord for a brief overview asking 3 main questions.

What did we do?

What are we going to do?

Are there any difficulties we need to discuss?

We kept most of the meetings short and productive, except for the last meeting of each sprint, where we fixed bugs and combined our code.

Roles

Scrum Master: Niko was assigned to be in charge of planning and keeping everyone on track with the project process. As a Scrum Master, he had to keep an eye on every one of us and make sure we are following the Scrum framework.

GitHub Master: Niko was responsible to clear the code and keep the repository up to date, also he had to deal with the merge conflicts.

Report editor Master: Daniel was responsible for the general design of the report. He had to keep track of editing and decorating the report to the final.

Development Team: Basically, all of us were working on the coding. Diogo with his previous programming skills got the role of a Senior engineer for this project. However, the ideas that came were voted according to the laws of democracy.

Product Owners: Social- og Sundhedsskolen Esbjerg (SOSU)

Tools

GitHub: For the version control and code storage we use GitHub. We had the main branch where we were keeping our final code (code which was 100% complete). There was also a different branch for each one of us in the team. In there, everyone could work on his specific tasks without disturbing anyone else's code.

Scrumwise: We use Scrumwise for sprint planning, user stories, and tasks.

Discord: Discord was our main tool used for our meetings. We used discord to contact the team.

Team Collaboration Policies

One of the first steps our group has taken is for us all to reach a consensus on work ethic and to create our group contract. The contract will be uploaded in the appendices.

Definition of Done

We got this project into a serious concern. So we had to change our definition of Done. Done in our group involved in this when you want to check something as completely Done you should go through several steps.

Task or even many Tasks can be assigned to 1 person in the group. They are always coming with the big responsibility of completing each step before you go to the next one. When a task is created out of a user story, we also define a set of requirements for testing the implemented functionality.

Our Scrumwise boards are displayed 4 different columns:

- To do
- Doing
- To test
- Done

Overall Project schedule

Given the size of the project and the time of 5 weeks we have, as well as the lack of information from the product owners, organizing the project was a challenge. We used the Work Breakdown Structure (WBS) shown in fig.1 to break down our work into micro parts. Scrumwise was used to record the user stories during the sprints, and we created a text file(Bugs.txt) which can be found together with the source code in GitHub and also in fig.2 in which we recorded all detected bugs so that we could fix them later.

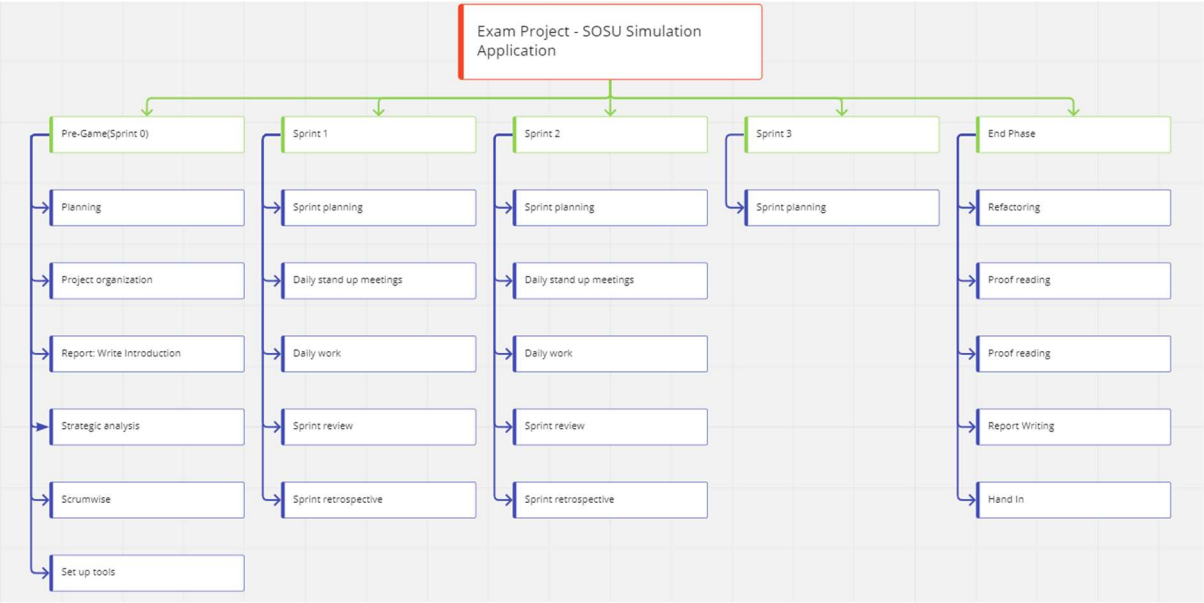


Figure 3: Work Breakdown Structure(WBS)

1. PreGame: Preparatory work, setting up Github, strategic analysis etc.
2. Sprint 1
3. Sprint 2
4. Sprint 3 Fixing bugs, small requirements finished, etc.
5. End phase: Report writing, clean up, refactoring, etc.

```
//every user should have a label with their school
//don't show ids on table views (also, delete all table columns with "school")
//admin manages admin view --> school choice box only shows schools which are not assigned to any admin and have a label with school name
//fix all filters on table views
//add all JFrames
//add name on citizen template and citizen on db
//when creating or copying citizen template, have textField to receive name
//make tab pane for citizen view
//resize windows

//clear fields after creating users - diogo
//datepicker enabled on NOT RELEVANT. Make it Disable - diogo
//check delete citizenTemplate - diogo - if deleted citizenTemplate, it deletes the ID,
and all citizens created from this template need that ID (fixed by using on delete set null on sql)
//when refresh table view, also refresh combo box (schools) - diogo
//make action event for filters - diogo
//Fix To have only 1 username ; if existing give Exception; 1/5 (schools) - diogo
//when closing citizen or citizenTemplate edit pages, go back to previous view - niko
//make citizenTemplate names unique

fix exceptions - diogo
```

Figure 4: Bugs.txt

Initial product backlog

Figure 5: Initial Backlog

Our Initial backlog contains all mandatory user stories defined by PO's given requirements.

Architecture

For this project, we used three-tier architecture implementing the MVC, BLL and DAL. We conclude that this is the best choice as this is the architecture of the material this year and it fits perfectly with the application we build. With the three-tier architecture we can easily separate data access operations, business logic, and the GUI layer.

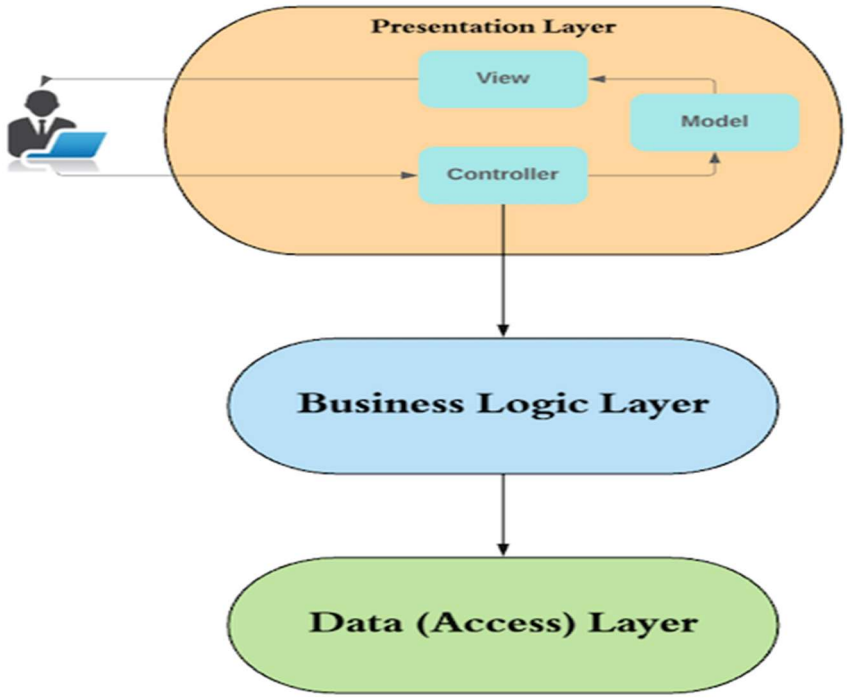


Figure 6: Three-tier architecture model

Prototype

Prototypes were a key part of the project. The process by which they were created allowed our group to successfully understand the assigned task and meet all the requirements of the PO. All prototypes were made at a very basic level of drawing (paint).

The prototypes will be discussed with the figures below:

Login screen

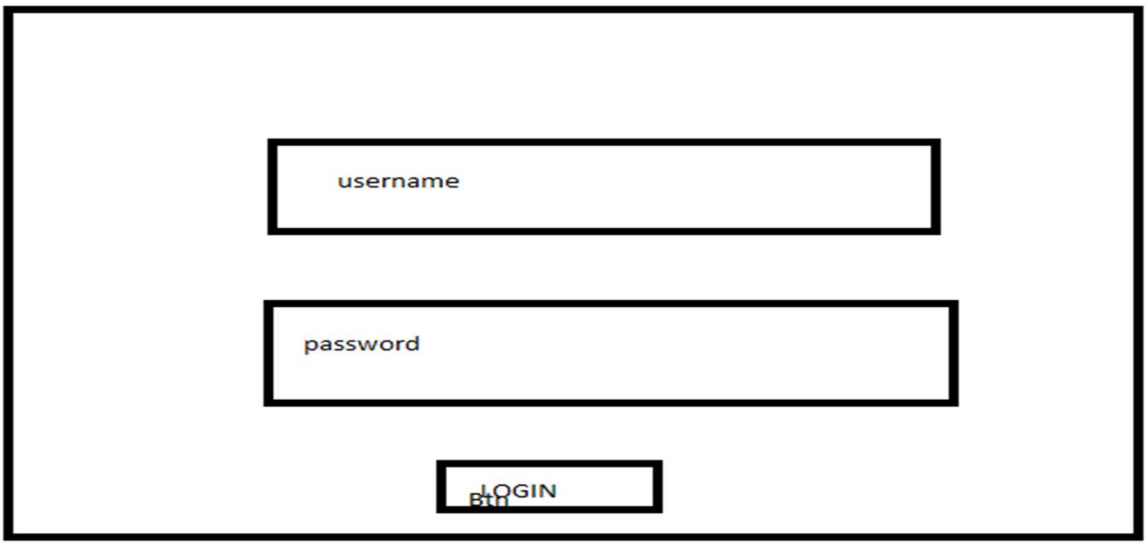


Figure 7: The login screen where all users log in

The first scene of our application is the login which is exactly the same for all the users. The idea is to log in with a username and password and as you click the login button it will redirect you to a new scene.

Admin screen

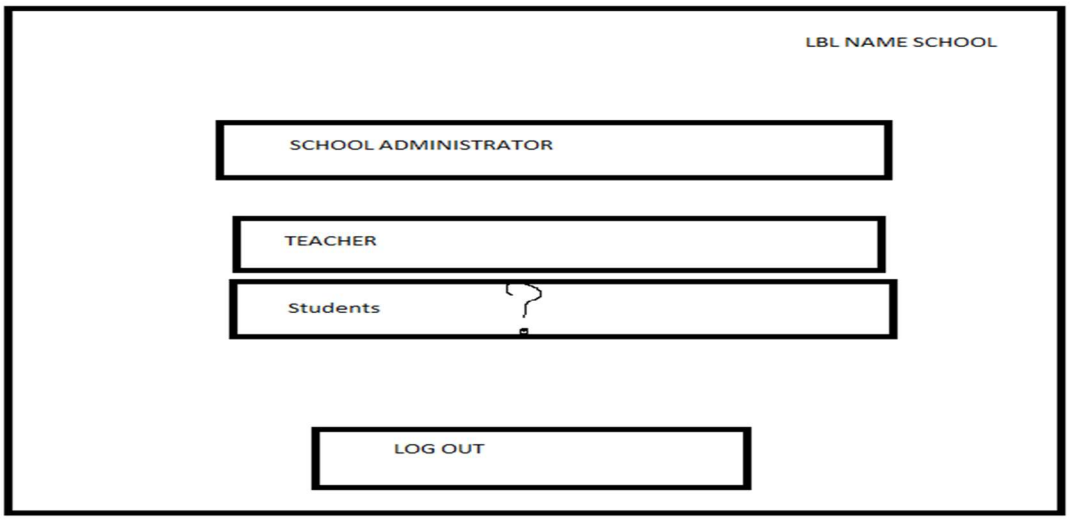


Figure 8: Administrator screen

This is the initial idea for the Administrator scene.

Teacher screen

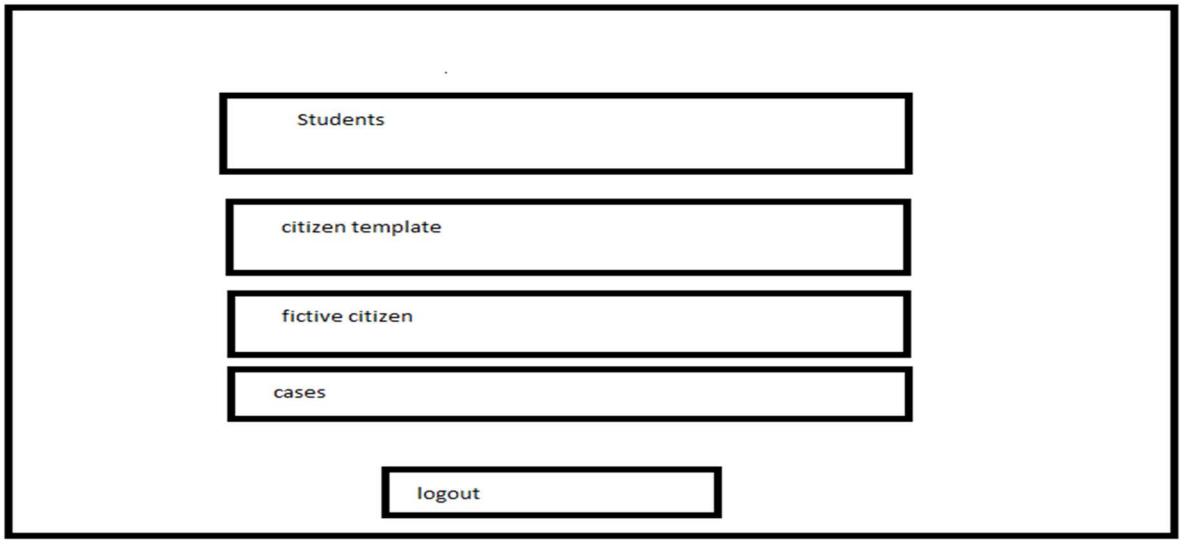


Figure 9: Teacher screen

This is the initial idea for the Teacher scene.

AdminManagesTeacher/School/Admin/Student screen

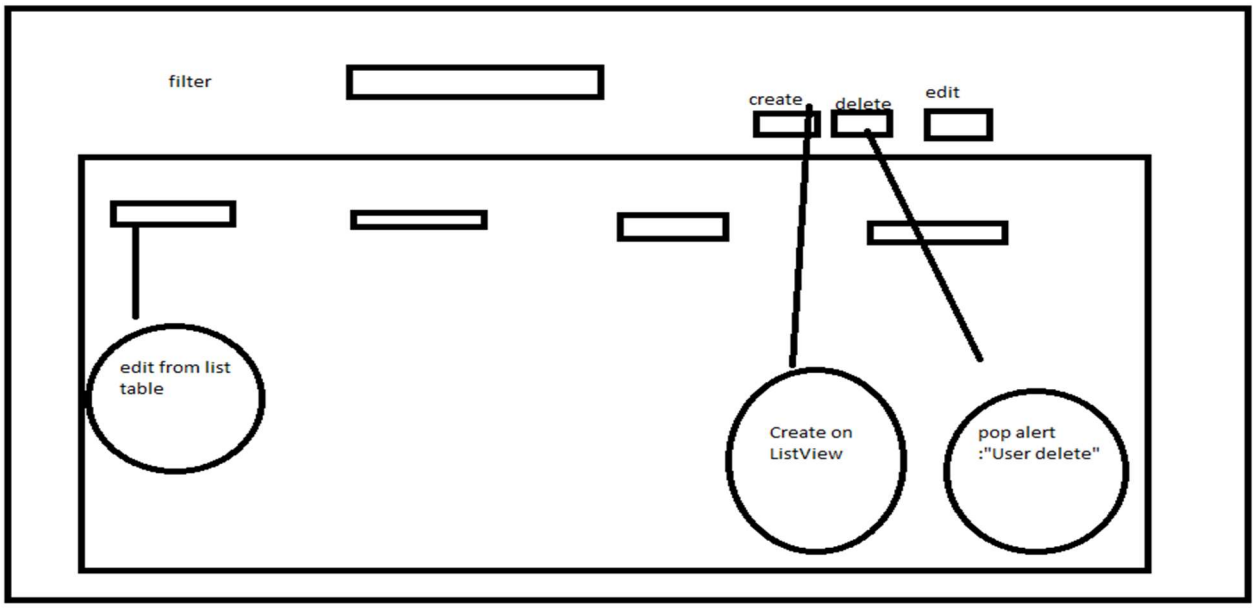


Figure 10: AdminManages screen

This prototype shows 4 different scenes which are Identical. The prototype represents a TableView/ListView that will display (for example) all teachers. You will be able to edit them by double-clicking on a specific teacher or create/delete them using the buttons above. There is also a filter that you will be able to search by keyword.

Sprint 1

Sprint planning

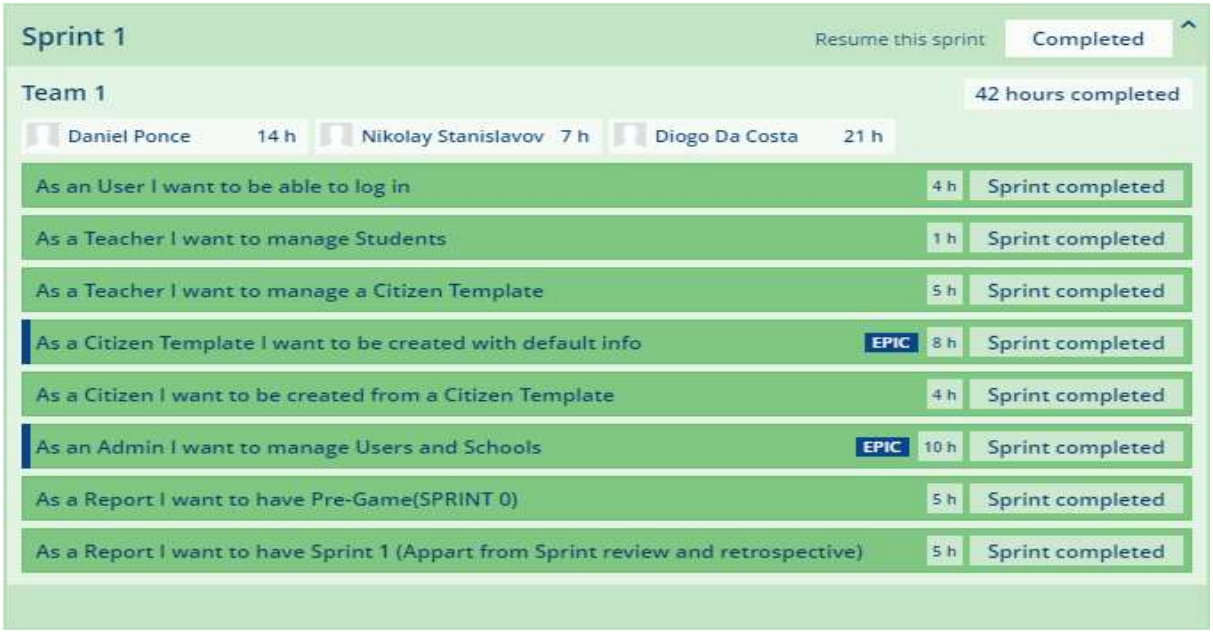
The first days were difficult for multiple reasons. We didn't have all the information we needed and the one we had was not clear enough and led to misunderstandings. Anyway, we had to get the project started so we made some assumptions with the given information and we began to plan and organise how we were going to approach this project.

Prior to starting the sprint, we had a meeting, where user stories were defined, and estimated. We then prioritized the stories, so that the least important stories would be given less attention, in case our estimates were wrong, and we would not be able to complete all the stories in sprint 1. Finally, we broke down the stories into individual tasks.

We were required to work 2 hours minimum per person, per day on the exam project, and therefore we would have a maximum of 48 total work hours available for the first sprint.

When we were done estimating all the user stories for the exam project, we concluded that it could be possible for us to complete the main structure of the project in the first sprint, if we chose to focus on implementation, and keep documentation to a minimum. The goal of Sprint 1 was to have created a base the team could build on in future sprints. Our backlog for Sprint 1 consisted of all the basic structure backlog. We decided to have a scrum master for the whole project so that the person chosen could manage and help all the others, either with tasks or time management.

Our first Sprint backlog looked like the next screenshot:



Daily meetings

On this Sprint, the meetings took more time than estimated due to the missing information and because we had to change the design and functionality of the program multiple times. These meetings consisted in discussing the new approach and what we had to do to make it possible.

GUI

For this project, we tried to make all the GUI understandable and easy to use since the users may not have knowledge about IT and programming.

We tried to fill all the text fields with the information that should be filled in them as well, using buttons and choice boxes to be user-friendly.

We also decided to put two buttons in all the views, them being a close button and a go back button, to be easy to go through the whole program.

We also added small pop-up windows for the user to understand that the operation that they just made was done successfully.

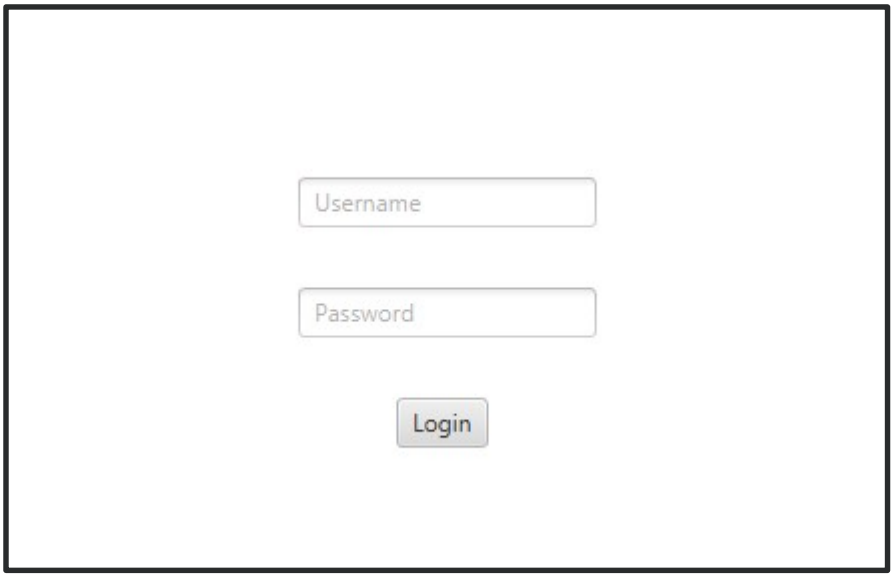


Figure 11: Login View

First of all, we made the login view (image above), which consisted of two text fields (one for username and the other for password respectively) and a button that would pop up the next view depending on which kind of user is logging in. Those views are:

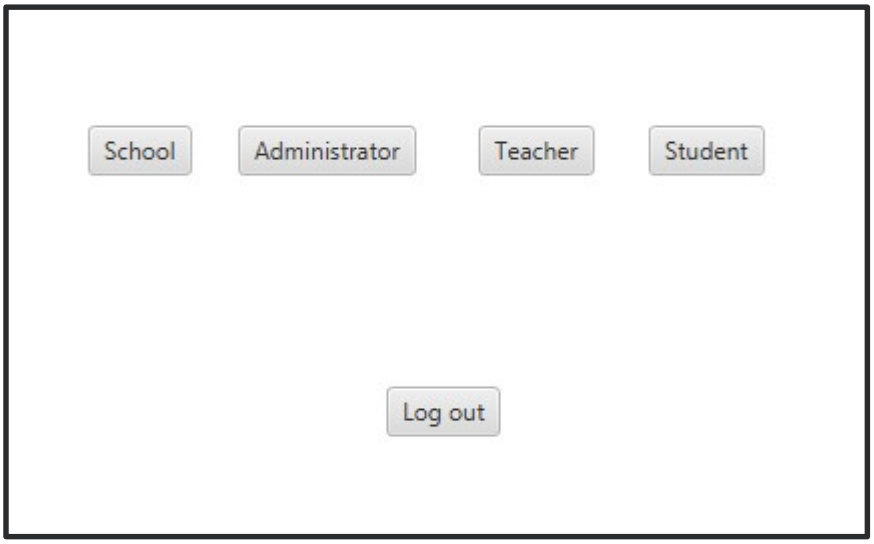


Figure 12: Admin View

- Administrator view (image above): this view has five buttons:
    - School button: pops up a view with a list of all schools, a text field and a button to create a new one.

- Administrator button: pops up a view with a list of all administrators, a choice box with all the schools without an administrator, three text fields (name, username and password) and two buttons (create and delete).
- Teacher button: pops up a view with a list of all teachers, three text fields (name, username and password) and two buttons (create and delete).
- Student button: pops up a view with a list of all students, three text fields (name, username and password) and two buttons (create and delete).
- Logout button: logs out and goes back to the login view.

Figure 13: AdminManagesStudent/Teacher/School/Administrator

On Button Clicked from the Admin Manager, it will pop a new view looking like on Figure 13.

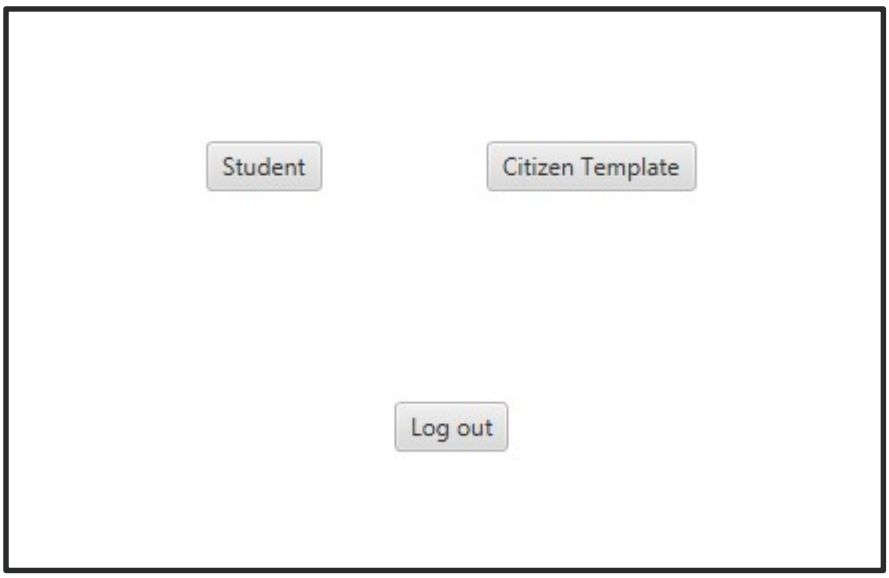


Figure 14: Teacher View

- Teacher view (image above): this view has three buttons:
    - Student button: does the same thing as the student button in the administrator view.
    - Citizen template button: pops up a view with a list of all the citizen templates, a text field for the name of a new citizen template and six buttons:
        - Create button: creates a new citizen template with the name in the text field and pops up a view with three choice box for the health condition (category choicebox, that shows all categories; subcategory choicebox, that shows all subcategories of the chosen category; and relevance choice box that shows "not relevant", "relevant" and "very relevant"), a save button for the health conditions (depending on if it is "not relevant", it will save it; if it is "relevant" it will pop up a view with a text field to fill with a professional note and a save button; and if it is "very relevant" it will pop up a view with the same professional note text field, two more text fields -current assessment and observation note respectively-, a choice box with the expected level, a date picker and a save button), six choice box for the functional abilities (category and subcategory -which do the same as the health condition category and subcategory choice boxes-, present level -shows numbers from 0 to 4 which are a reference to the levels in the image-, expected level -does the same as the previous one-, performance -shows "self performed", "self performed partly", "does not perform itself" and "not relevant"- and meaning of performance -shows "does not experience limitations" and

"experience limitations"), three text fields (proffesional note, the ciziten's wishes and goals and observation note), a date picker, a save button for the functional abilities, an image explaining the levels of the functional abilites, eleven text fields for general information (mastery, motivation, resources, roller, habits, education and jobs, life story, network, health information, assistive devices and the interior of the home), a button for every general information text field that pops up a little text with an explanation about it and a save button for the general information.

- Delete button: deletes the selected citizen template.
- Edit button: pops up the same view as the create button for the selected citizen template.
- Copy button: creates a copy of the selected citizen template.
- Create citizen button: creates a citizen from the selected citizen template for each student from their school.
- Logout button: logs out and goes back to the login view.

CSS

We will leave the CSS for the next sprint because we don't have enough time due to all the information problems so all the views are quite basic.

Data model

In our first sprint we had the following tables on our database:

- Citizen - holds the citizen ID, citizen template ID, which is a foreign key from CitizenTemplate table, and citizen name.
- CitizenTemplate - holds the citizen template ID, school ID, which is a foreign key from the School table, and citizen template name.
- FunctionalAbilitiesCitizen - holds the functional abilities citizen ID, category name, subcategory name, score, expected score, professional note, performance, limitation, goals note, observation note, date and citizen ID, which is a foreign key from Citizen table.

- FunctionalAbilitiesCitizenTemplate - holds the functional abilities citizen template ID, category name, subcategory name, score, expected score, professional note, performance, limitation, goals note, observation note, date and citizen template ID, which is a foreign key from CitizenTemplate table.
- GeneralInformationCitizen - holds the general information citizen ID, name, explanation, editable (which is the information that the user writes in the text field) and citizen ID, which is a foreign key from the Citizen table.
- GeneralInformationCitizenTemplate -holds the general information citizen template ID, name, explanation, editable (which is the information that the user writes in the text field) and citizen template ID, which is a foreign key from CitizenTemplate table.
- HealthConditionCitizen - holds the health conditions citizen ID, category, subcategory, colour (which is the relevance), professional note, assessment note, expected level, observable note, date and citizen ID, which is a foreign key from Citizen table.
- HealthConditionCitizenTemplate - holds the health conditions citizen template ID, category, subcategory, colour (which is the relevance), professional note, assessment note, expected level, observable note, date and citizen template ID, which is a foreign key from CitizenTemplate table.
- School - holds school ID and school name.
- SchoolClass - holds school class, school ID, which is a foreign key from school table, and user ID, which is a foreign key from UserData table.
- UserData - holds the user ID, type of user, which is a foreign key from UserType table; school, which is a foreign key from School table, name, username and password.
- UserType - holds the user type ID and type of user.

Implementation

As for the implementation we decided to start the first user story and do it from beginning to end and then move on to the others, using the same method for all of them.

The user stories that we had in this first sprint were:

- As an User I want to be able to log in:

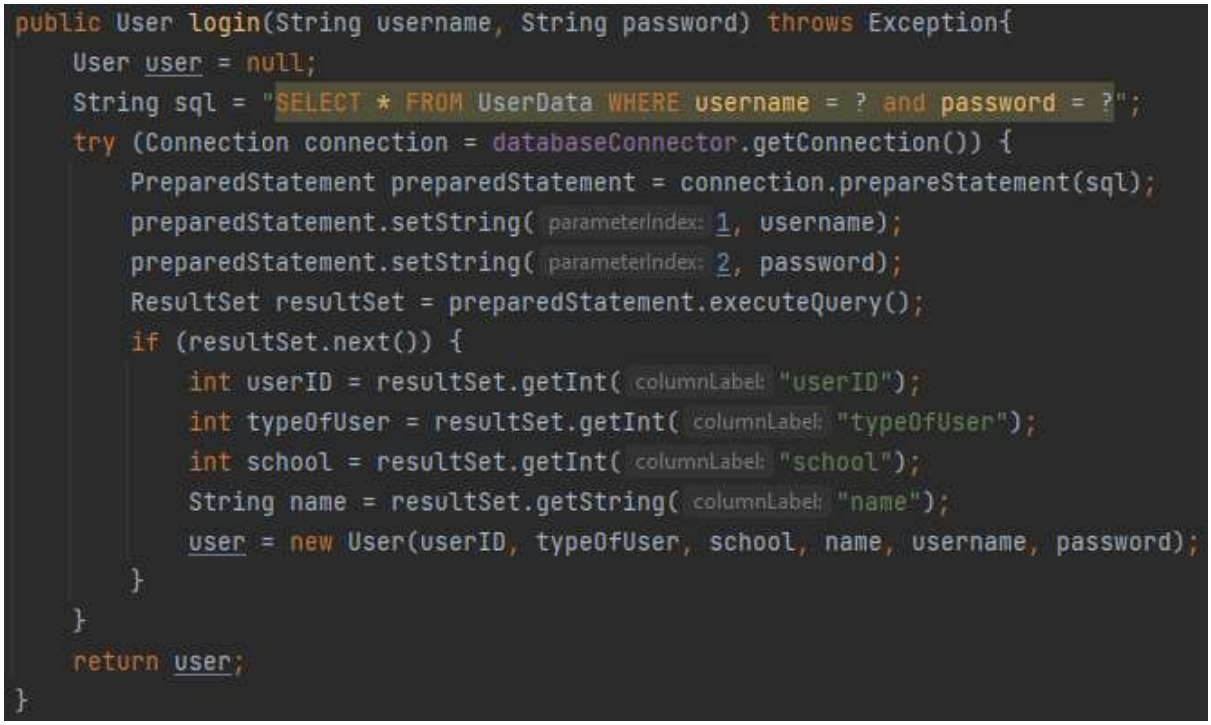
  Because it was the first implementation, we had to create the MVC layer structure as well as the databases for the users, the schools and the types of user because the user has foreign keys for school and type of user which has an influence on the view that is going to be shown. So in the controller it takes the username and the password, it sends it to the database and it receives which type of user is logging in and it pops up the view that belongs to that type of user.

```
public void login(ActionEvent event) {
    String username = usernameTxt.getText();
    String password = passwordTxt.getText();
    Stage currentStage = (Stage) logInBtn.getScene().getWindow();
    currentStage.close();
    try {
        User result = userModel.login(username, password);
        if (result == null) {
            Alert alert = new Alert(Alert.AlertType.NONE);
            alert.setTitle("Error");
            alert.setHeaderText("LOGIN FAILED !!");
            alert.getDialogPane().getStylesheets().add(getClass().getResource( name: "/gui/view/css/myDialogs.css").toExternalForm());
            alert.getDialogPane().getStyleClass().add("myDialog");
            ButtonType okButton = new ButtonType( text: "OK");
            alert.getButtonTypes().setAll(okButton);
            alert.showAndWait();
            Parent root = FXMLLoader.load(getClass().getResource( name: "/gui/view/loginView.fxml"));
            Stage stage = new Stage();
            Scene scene = new Scene(root);
            stage.initStyle(StageStyle.TRANSPARENT);
            stage.setScene(scene);
            stage.show();
            scene.setFill(Color.TRANSPARENT);
        }
```

```
else {
    if (result.getTypeOfUser() == 1) {
        schoolId1 = result.getSchool();
        try{
            Parent root = FXMLLoader.load(getClass().getResource( name: "/gui/view/adminView.fxml"));
            Stage stage = new Stage();
            Scene scene = new Scene(root);
            stage.initStyle(StageStyle.TRANSPARENT);
            stage.setScene(scene);
            stage.setUserData(schoolId1);
            stage.show();
            scene.setFill(Color.TRANSPARENT);
        }catch (Exception e){
            PopUp.showError(e.getMessage());
        }
    }
    else if (result.getTypeOfUser() == 2) {
        schoolId1 = result.getSchool();
        try{
            Parent root = FXMLLoader.load(getClass().getResource( name: "/gui/view/teacherView.fxml"));
            Stage stage = new Stage();
            Scene scene = new Scene(root);
            stage.initStyle(StageStyle.TRANSPARENT);
            stage.setScene(scene);
            stage.setUserData(schoolId1);
            stage.show();
            scene.setFill(Color.TRANSPARENT);
        }catch (Exception e){
            PopUp.showError(e.getMessage());
        }
    }
```

```
            else if (result.getTypeOfUser() == 3) {
                schoolId1 = result.getSchool();
                studentId1 = result.getUserId();
                String info = schoolId1 + "," + studentId1;
                try{
                    Parent root = FXMLLoader.load(getClass().getResource( name: "/gui/view/studentView.fxml"));
                    Stage stage = new Stage();
                    Scene scene = new Scene(root);
                    stage.initStyle(StageStyle.TRANSPARENT);
                    stage.setScene(scene);
                    stage.setUserData(info);
                    stage.show();
                    scene.setFill(Color.TRANSPARENT);
                }catch (Exception e){
                    PopUp.showError(e.getMessage());
                }
            }
        }
    } catch (Exception e) {
        PopUp.showError(e.getMessage());
    }
}
```

In these three pictures above, it sends the information the user has filled in the text fields to the database and checks what we receive in order to pop up the correct view. If the information is wrong, it will pop up an advice warning the user about it. As you can see too, we are using the stage.setUserData() as we said before to send the user information to the next view.



```
public User login(String username, String password) throws Exception{
    User user = null;
    String sql = "SELECT * FROM UserData WHERE username = ? and password = ?";
    try (Connection connection = databaseConnector.getConnection()) {
        PreparedStatement preparedStatement = connection.prepareStatement(sql);
        preparedStatement.setString( parameterIndex: 1, username);
        preparedStatement.setString( parameterIndex: 2, password);
        ResultSet resultSet = preparedStatement.executeQuery();
        if (resultSet.next()) {
            int userID = resultSet.getInt( columnLabel: "userID");
            int typeOfUser = resultSet.getInt( columnLabel: "typeOfUser");
            int school = resultSet.getInt( columnLabel: "school");
            String name = resultSet.getString( columnLabel: "name");
            user = new User(userID, typeOfUser, school, name, username, password);
        }
    }
    return user;
}
```

In this image we are sending and receiving the information to the database

- As an Admin I want to manage Users and Schools:

  Here we decided to create a different view for each type of user and for schools. So in each controller we had to send all the information that the administrator fulfilled to the database and, depending on which button they clicked, it creates, edits or deletes the user or school in the database.

```java
@Override
public void initialize(URL location, ResourceBundle resources) {
    adminTableView.setEditable(true);
    updateAdminTableView();
    try {
        schoolChoiceBox.setItems(userModel.getAllSchoolsNotAssigned());
    } catch (Exception e) {
        PopUp.showError(e.getMessage());
    }
    editAdminFromTableView();
    try {
        filterAdminTableView();
    } catch (Exception e) {
        PopUp.showError(e.getMessage());
    }
}

public void updateSchoolComboBox() throws Exception {
    schoolChoiceBox.setItems(userModel.getAllSchoolsNotAssigned());
}

public void updateAdminTableView() {
    nameColumn.setCellValueFactory(new PropertyValueFactory<>("name"));
    usernameColumn.setCellValueFactory(new PropertyValueFactory<>("username"));
    passwordColumn.setCellValueFactory(new PropertyValueFactory<>("password"));
    try {
        adminTableView.setItems(userModel.getAllAdmins());
        filterAdminTableView();
    } catch (Exception e) {
        PopUp.showError(e.getMessage());
    }
}
```

This is an example about how we initialise the adminManagesAdmins view which is pretty much the same as the other three so we will only put this one. Here we are setting all the tables and the choice box with the information that we receive from the database.

```
public void createAdmin(ActionEvent actionEvent) throws Exception {
    int count = 0;

    if(usernameTxt.getText().isEmpty() || nameTxt.getText().isEmpty() || passwordTxt.getText().isEmpty() || schoolChoiceBox.getValue() == null){
        Alert alert = new Alert(Alert.AlertType.NONE);
        alert.setTitle("Error");
        alert.setHeaderText("Please fill in all the fields");
        alert.getDialogPane().getStylesheets().add(getClass().getResource( name: "/gui/view/css/myDialogs.css").toExternalForm());
        alert.getDialogPane().getStyleClass().add("myDialog");
        ButtonType okButton = new ButtonType( text: "OK");
        alert.getButtonTypes().setAll(okButton);
        alert.showAndWait();
        filterAdminTableView();
    }
    else {
        try {
            count = 0;
            List<User> allUsers = userModel.getAllUsernames();
            for (int i=0; i< allUsers.size();i++) {
                if (allUsers.get(i).getUsername().equals(usernameTxt.getText())) {
                    Alert alert = new Alert(Alert.AlertType.NONE);
                    alert.setTitle("Error");
                    alert.setHeaderText("Username already exists, please choose a new one");
                    alert.getDialogPane().getStylesheets().add(getClass().getResource( name: "/gui/view/css/myDialogs.css").toExternalForm());
                    alert.getDialogPane().getStyleClass().add("myDialog");
                    ButtonType okButton = new ButtonType( text: "OK");
                    alert.getButtonTypes().setAll(okButton);
                    alert.showAndWait();
                    filterAdminTableView();
                    count++;
                }
            }
```

```
            if(count == 0) {
                int schoolId = schoolChoiceBox.getValue().getSchoolID();
                userModel.createAdmin(schoolId, nameTxt.getText(), usernameTxt.getText(), passwordTxt.getText());
                Alert alert = new Alert(Alert.AlertType.NONE);
                alert.setTitle("Process confirmation");
                alert.setHeaderText("Admin created");
                alert.getDialogPane().getStylesheets().add(getClass().getResource( name: "/gui/view/css/myDialogs.css").toExternalForm());
                alert.getDialogPane().getStyleClass().add("myDialog");
                ButtonType okButton = new ButtonType( text: "OK");
                alert.getButtonTypes().setAll(okButton);
                alert.showAndWait();
                nameTxt.clear();
                usernameTxt.clear();
                passwordTxt.clear();
                schoolChoiceBox.setValue(null);
                schoolChoiceBox.setItems(userModel.getAllSchoolsNotAssigned());
                updateAdminTableView();
                filterAdminTableView();
            }
        } catch (Exception e) {
            PopUp.showError(e.getMessage());
        }
    }
}
```

In these two pictures we create admins. First of all it checks if all the text fields and the choice box are fulfilled and then, if they are not, it will pop up an alert warning to do it. If they are fulfilled, it will create a new user sending all the information to the database.

```java
public void onClickDelete(ActionEvent actionEvent) {
    try {
        if (adminTableView.getSelectionModel().getSelectedItem() == null){
            Alert alert = new Alert(Alert.AlertType.NONE);
            alert.setTitle("Error");
            alert.setHeaderText("Choose an Admin! Please try again");
            alert.getDialogPane().getStylesheets().add(getClass().getResource( name: "/gui/view/css/myDialogs.css").toExternalForm());
            alert.getDialogPane().getStyleClass().add("myDialog");
            ButtonType okButton = new ButtonType( text: "OK");
            alert.getButtonTypes().setAll(okButton);
            alert.showAndWait();
            filterAdminTableView();
        }
        else {
            userModel.deleteUser(adminTableView.getSelectionModel().getSelectedItem().getUserId());
            Alert alert = new Alert(Alert.AlertType.NONE);
            alert.setTitle("Process confirmation");
            alert.setHeaderText("Admin deleted");
            alert.getDialogPane().getStylesheets().add(getClass().getResource( name: "/gui/view/css/myDialogs.css").toExternalForm());
            alert.getDialogPane().getStyleClass().add("myDialog");
            ButtonType okButton = new ButtonType( text: "OK");
            alert.getButtonTypes().setAll(okButton);
            alert.showAndWait();
            updateAdminTableView();
            filterAdminTableView();
        }
    } catch (Exception e){
        PopUp.showError(e.getMessage());
    }
}
```

Here we are deleting an administrator. It checks if the user has clicked on an admin in the table and if they don't, it will pop up an advice warning them about it. If they do, it will send the ID of the selected administrator to the database and it will delete it.

- As a Teacher I want to manage Students:

For this one teachers had to be able to create, delete and edit students (users whose type of user is "student") from their own school so in the controller of the view that we explained in the GUI part, we created methods that receive the necessary information to do it and send it to the database through the model, the manager and the DAO.

```
public void filterStudentTableView() throws Exception {

    ObservableList<User> userList = userModel.getAllStudents(schoolId1);
    FilteredList<User> filteredData = null;
    try {
        filteredData = new FilteredList<>(userList, b -> true);
    } catch (Exception e) {
        PopUp.showError(e.getMessage());
    }

    FilteredList<User> finalFilteredData = filteredData;
    filterTxt.textProperty().addListener((observable, oldValue, newValue) -> {
        finalFilteredData.setPredicate(user -> {


            if (newValue == null || newValue.isEmpty()) {
                return true;
            }
            String lowerCaseFilter = newValue.toLowerCase();

            if (user.getName().toLowerCase().contains(lowerCaseFilter) || user.getUsername().toLowerCase().contains(lowerCaseFilter) ||
                    user.getPassword().toLowerCase().contains(lowerCaseFilter))
                return true;
            else
                return false;
        });
    });


    SortedList<User> sortedData = new SortedList<>(filteredData);

    sortedData.comparatorProperty().bind(studentTableView.comparatorProperty());

    studentTableView.setItems(sortedData);
}
```

This is the filter we use for the student table view.

```java
public void editStudentFromTableView(){
    nameColumn.setCellFactory(TextFieldTableCell.forTableColumn());
    nameColumn.setOnEditCommit(
            new EventHandler<TableColumn.CellEditEvent<User, String>>() {
                @Override
                public void handle(TableColumn.CellEditEvent<User, String> t) {
                    t.getTableView().getItems().get(
                            t.getTablePosition().getRow()).setName(t.getNewValue());
                    TablePosition pos = studentTableView.getSelectionModel().getSelectedCells().get(0);
                    int row = pos.getRow();
                    int userId = studentTableView.getSelectionModel().getSelectedItem().getUserId();
                    String password = passwordColumn.getCellObservableValue(((studentTableView.getItems().get(row)))).getValue();
                    String name = nameColumn.getCellObservableValue(((studentTableView.getItems().get(row)))).getValue();
                    String username = usernameColumn.getCellObservableValue(((studentTableView.getItems().get(row)))).getValue();
                    try {
                        userModel.editUser(userId, name, username, password);
                        filterStudentTableView();
                    } catch (Exception e) {
                        PopUp.showError(e.getMessage());
                    }

                }
            }
    );
```

```java
    passwordColumn.setCellFactory(TextFieldTableCell.forTableColumn());
    passwordColumn.setOnEditCommit(
            new EventHandler<TableColumn.CellEditEvent<User, String>>() {
                @Override
                public void handle(TableColumn.CellEditEvent<User, String> t) {
                    t.getTableView().getItems().get(
                            t.getTablePosition().getRow()).setPassword(t.getNewValue());
                    TablePosition pos = studentTableView.getSelectionModel().getSelectedCells().get(0);
                    int row = pos.getRow();
                    int userId = studentTableView.getSelectionModel().getSelectedItem().getUserId();
                    String password = passwordColumn.getCellObservableValue(((studentTableView.getItems().get(row)))).getValue();
                    String name = nameColumn.getCellObservableValue(((studentTableView.getItems().get(row)))).getValue();
                    String username = usernameColumn.getCellObservableValue(((studentTableView.getItems().get(row)))).getValue();
                    try {
                        userModel.editUser(userId, name, username, password);
                        filterStudentTableView();
                    } catch (Exception e) {
                        PopUp.showError(e.getMessage());
                    }

                }
            }
    );
}
```

```
usernameColumn.setCellFactory(TextFieldTableCell.forTableColumn());
usernameColumn.setOnEditCommit(
        new EventHandler<TableColumn.CellEditEvent<User, String>>() {
            @Override
            public void handle(TableColumn.CellEditEvent<User, String> t) {
                t.getTableView().getItems().get(
                        t.getTablePosition().getRow()).setUsername(t.getNewValue());
                TablePosition pos = studentTableView.getSelectionModel().getSelectedCells().get(0);
                int row = pos.getRow();
                int userId = studentTableView.getSelectionModel().getSelectedItem().getUserId();
                String password = passwordColumn.getCellObservableValue(((studentTableView.getItems().get(row)))).getValue();
                String name = nameColumn.getCellObservableValue(((studentTableView.getItems().get(row)))).getValue();
                String username = usernameColumn.getCellObservableValue(((studentTableView.getItems().get(row)))).getValue();
                try {
                    userModel.editUser(userId, name, username, password);
                    filterStudentTableView();
                } catch (Exception e) {
                    PopUp.showError(e.getMessage());
                }

            }
        }
);
```

In these three pictures we are editing the student information directly from the table. It works by clicking on the user that you want to edit, then clicking the cell that you want to edit, then you would be able to modify it and, whenever you press enter, it will send the information in that cell to the database and it will update it.

- As a Teacher I want to manage a Citizen Template:

  We had to create the Citizen Template Database first and do pretty much the same we did for the teacher managing students but with the information that the Citizen Template needs. Additionally, we created a method that would create a copy of the selected Citizen Template in the database.

```
public void onClickEdit(ActionEvent actionEvent) {
    if(citizenTemplateTV.getSelectionModel().getSelectedItem() == null){
        Alert alert = new Alert(Alert.AlertType.NONE);
        alert.setTitle("Error");
        alert.setHeaderText("Please choose a CitizenTemplate");
        alert.getDialogPane().getStylesheets().add(getClass().getResource( name: "/gui/view/css/myDialogs.css").toExternalForm());
        alert.getDialogPane().getStyleClass().add("myDialog");
        ButtonType okButton = new ButtonType( text: "OK");
        alert.getButtonTypes().setAll(okButton);
        alert.showAndWait();
    }
    else {
        int citizenTemplateId = citizenTemplateTV.getSelectionModel().getSelectedItem().getCitizenTemplateId();
        try {
            Parent root = FXMLLoader.load(getClass().getResource( name: "/gui/view/teacherEditCitizenTemplateView.fxml"));
            Stage stage = new Stage();
            Scene scene = new Scene(root);
            stage.initStyle(StageStyle.TRANSPARENT);
            stage.setScene(scene);
            stage.setUserData(citizenTemplateId);
            stage.show();
            scene.setFill(Color.TRANSPARENT);
        } catch (Exception e) {
            PopUp.showError(e.getMessage());
        }
    }
}
```

Here, if the user has clicked on a citizen template before clicking the edit button, it will pop up a new view with all the citizen template information about health condition, functional abilities and general information of the citizen template that we are sending in the stage.setUserData().

```
public void onClickCopy(ActionEvent actionEvent) throws Exception {
    if(citizenTemplateTV.getSelectionModel().getSelectedItem() == null){
        Alert alert = new Alert(Alert.AlertType.NONE);
        alert.setTitle("Error");
        alert.setHeaderText("Please choose a CitizenTemplate");
        alert.getDialogPane().getStylesheets().add(getClass().getResource( name: "/gui/view/css/myDialogs.css").toExternalForm());
        alert.getDialogPane().getStyleClass().add("myDialog");
        ButtonType okButton = new ButtonType( text: "OK");
        alert.getButtonTypes().setAll(okButton);
        alert.showAndWait();
    }
    else {
        Stage currentStage = (Stage) createBtn.getScene().getWindow();
        schoolId1 = (Integer) currentStage.getUserData();
        int citizenTemplateId = citizenTemplateTV.getSelectionModel().getSelectedItem().getCitizenTemplateId();
        String citizenTemplateName = citizenTemplateTV.getSelectionModel().getSelectedItem().getCitizenTemplateName() + "Copy";
        citizenTemplateModel.copyCitizenTemplate(citizenTemplateId, schoolId1, citizenTemplateName);
        Alert alert = new Alert(Alert.AlertType.NONE);
        alert.setTitle("Process confirmation");
        alert.setHeaderText("Citizen Template COPIED !!");
        alert.getDialogPane().getStylesheets().add(getClass().getResource( name: "/gui/view/css/myDialogs.css").toExternalForm());
        alert.getDialogPane().getStyleClass().add("myDialog");
        ButtonType okButton = new ButtonType( text: "OK");
        alert.getButtonTypes().setAll(okButton);
        alert.showAndWait();
        updateCitizenTemplateTV();
    }
}
```

Here, if the user has clicked on a citizen template before clicking the copy button, it will send all the information about that selected citizen template to the database and it will create a new citizen template with the same information.

- As a Citizen Template I want to be created with default info:

  We had to create databases for health conditions, functional abilities and general information for Citizen Templates. After that, we created the controller of the view that we explain in the GUI part and apart from the necessary methods to send the information to the database, we had to create methods to fulfil the choice boxes, being one of them a method for the category choice box so that every time the teacher selects a category from it, the subcategory choice box would

```
if (count == 0) {
    citizenTemplate = citizenTemplateModel.createCitizenTemplate(schoolId1, citizenTemplateNameTxt.getText());
    Alert alert = new Alert(Alert.AlertType.NONE);
    alert.setTitle("Process confirmation");
    alert.setHeaderText("Citizen Template created");
    alert.getDialogPane().getStylesheets().add(getClass().getResource( name: "/gui/view/css/myDialogs.css").toExternalForm());
    alert.getDialogPane().getStyleClass().add("myDialog");
    ButtonType okButton = new ButtonType( text: "OK");
    alert.getButtonTypes().setAll(okButton);
    alert.showAndWait();
    updateCitizenTemplateTV();
    int citizenTemplateID = citizenTemplate.getCitizenTemplateId();
    try{
        Parent root = FXMLLoader.load(getClass().getResource( name: "/gui/view/teacherEditCitizenTemplateView.fxml"));
        Stage stage = new Stage();
        Scene scene = new Scene(root);
        stage.initStyle(StageStyle.TRANSPARENT);
        stage.setScene(scene);
        stage.setUserData(citizenTemplateID);
        stage.show();
        scene.setFill(Color.TRANSPARENT);
    }catch (Exception e){
        PopUp.showError(e.getMessage());
    }
}
```

  be fulfilled with all the subcategories that this category has.

As we can see on the image above, the citizen Template is created and then the edit Citizen Template view is called, because on this view if the fields are empty and we put some values in them, then it will be created in the database, so it works as a edit view and also a create view, as it can be seen on the image below:

```
if(citizenTemplateModel.checkFunctionalAbilitiesTemplateId(selectedCategory, selectedSubCategory, citizenTemplateID) == false){
    citizenTemplateModel.createFunctionalAbilitiesCitizenTemplate(selectedCategory, selectedSubCategory, selectedPresentLevel, selectedExpectedLevel, profes
}
else{
    citizenTemplateModel.updateFunctionalAbilitiesCitizenTemplate(selectedCategory, selectedSubCategory, selectedPresentLevel, selectedExpectedLevel, profes
}
```

- As a Citizen I want to be created from a Citizen Template:

  Here we had to send all the information about a Citizen Template (including the information in the health conditions database, the functional abilities database and the general information database) and create a new Citizen for each student in the teacher's school with all this information.

```
public void createCitizenFromTemplate(int citizenTemplateId, String citizenName, int schoolId) throws DatabaseException {
    try {
        List<User> allStudents = getAllStudents(schoolId);
        for (User student : allStudents) {
            Citizen citizen = citizenDAO.createCitizenFromTemplate(citizenTemplateId, citizenName);
            int citizenId = citizen.getCitizenId();
            healthConditionsCitizenDAO.copyHealthConditionsCitizen(citizenId, citizenTemplateId);
            functionalAbilitiesCitizenDAO.copyFunctionalAbilitiesCitizen(citizenId, citizenTemplateId);
            generalInformationCitizenDAO.copyGeneralInformationCitizen(citizenId, citizenTemplateId);
            assignCitizenToStudent(student.getUserId(), citizenId);
        }
    } catch (Exception exception) {
        throw new DatabaseException("Couldn't create Citizen from Citizen Template! Check database connection!", exception);
    }
}
```

As we can see on the image above, when a citizen is created from a citizen Template, the three categories (health conditions, functional abilities and general info) are copied into the citizen tables respectively. Also shown below how this copy is made:

```
public void copyHealthConditionsCitizen(int citizenId, int citizenTemplateId) throws Exception{
    String sql = "SELECT * FROM HealthConditionsCitizenTemplate WHERE citizenTemplateID = ?";
    try(Connection connection = databaseConnector.getConnection()){
        PreparedStatement preparedStatement = connection.prepareStatement(sql);
        preparedStatement.setInt( parameterIndex: 1, citizenTemplateId);
        ResultSet resultSet = preparedStatement.executeQuery();
        while(resultSet.next()){
            String category = resultSet.getString( columnLabel: "healthConditionsCitizenTemplateCategory");
            String subCategory = resultSet.getString( columnLabel: "healthConditionsCitizenTemplateSubCategory");
            String relevance = resultSet.getString( columnLabel: "healthConditionsCitizenTemplateColor");
            String proffNote = resultSet.getString( columnLabel: "healthConditionsCitizenTemplateProfessionalNote");
            String assessmentNote = resultSet.getString( columnLabel: "healthConditionsCitizenTemplateAssessmentNote");
            String expectedLevel = resultSet.getString( columnLabel: "healthConditionsCitizenTemplateExpectedLevel");
            String observationNote = resultSet.getString( columnLabel: "healthConditionsCitizenTemplateObservableNote");
            LocalDate date = resultSet.getDate( columnLabel: "healthConditionsCitizenTemplateDate").toLocalDate();
            createHealthConditionsCitizen(category, subCategory, relevance, proffNote, assessmentNote, expectedLevel, observationNote, date, citizenId);
        }
    }
}
```

So, once we need to copy, we just go to the citizenTemplate database from which this citizen was created, and we copy all the information into the citizen table. And this is done for all three categories.

Code examples

One of the coding that we used a lot in order to send information from one fxml to another was the stage.setUserData() method to send it and then the currentStage.getUserData() method to receive it. We also had to use the Platform.runLater in order to receive the information when we needed it. We tried to do it without the Platform.runLater but every time we tried it, the values were null. So we just googled if there was any way to do it and this is one of the three that we found. We tried all three and, how it was the only one that worked, we used it in the whole project.

```
stage.setUserData(sendInformation);
```

Here we are sending the information as we said above.

```
Platform.runLater(() -> {
    Stage currentStage = (Stage) categoryLbl.getScene().getWindow();
    String informationReceived = (String) currentStage.getUserData();
```

And here we receive the information and we store it in a String.

For creating a citizen template, the teacher has to set the health conditions, general information and functional abilities.

Regarding health conditions, if the teacher selects “relevant” or “very relevant” in the relevance about the category, it will pop up new views for each, but if they choose “not relevant” it will send it directly to the database.

```java
public void OnClickSaveRelevance(ActionEvent actionEvent) {

    Stage currentStage = (Stage) fACategoryComboBox.getScene().getWindow();
    int citizenTemplateId = (Integer) currentStage.getUserData();

    JFrame frame = new JFrame();

    if(hCCategoryComboBox.getValue() == null || hCSubCategoryComboBox.getValue() == null || relevanceComboBox.getValue() == null) {
        JOptionPane.showMessageDialog(frame, message: "Please fulfill all fields");
    }
    else {
        String selectedCategory = hCCategoryComboBox.getSelectionModel().getSelectedItem();
        String selectedSubCategory = hCSubCategoryComboBox.getSelectionModel().getSelectedItem();
        String selectedRelevance = relevanceComboBox.getSelectionModel().getSelectedItem();
        String sendInformation = selectedCategory + "," + selectedSubCategory + "," + selectedRelevance + "," + citizenTemplateId;
        if (selectedRelevance.equals("Not relevant")) {
            String professionalNote = "";
            String assessmentNote = "";
            String expectedLevel = "";
            String observableNote = "";
            LocalDate date = LocalDate.now();
            citizenTemplateModel.createHealthConditionsCitizenTemplate(selectedCategory, selectedSubCategory
                    , selectedRelevance, professionalNote, assessmentNote, expectedLevel, observableNote, date, citizenTemplateId);
            hCCategoryComboBox.getSelectionModel().clearSelection();
            hCSubCategoryComboBox.getSelectionModel().clearSelection();
            relevanceComboBox.getSelectionModel().clearSelection();
            JOptionPane.showMessageDialog(frame, message: "Saved");
        }
```

Here, if the teacher clicks the button, it will check if all the comboBox are fulfilled. If they are not, it will pop up a message warning them to do it. If they are, it will check what is inside the comboBoxes and then, if it is "not relevant" (code above), it sends empty Strings regarding the notes.

```
else if (selectedRelevance.equals("Relevant")) {
    try{
        hCCategoryComboBox.getSelectionModel().clearSelection();
        hCSubCategoryComboBox.getSelectionModel().clearSelection();
        relevanceComboBox.getSelectionModel().clearSelection();
        Parent root = FXMLLoader.load(getClass().getResource( name: "/gui/view/relevantView.fxml"));
        Stage stage = new Stage();
        Scene scene = new Scene(root);
        //stage.initStyle(StageStyle.TRANSPARENT);
        stage.setScene(scene);
        stage.setUserData(sendInformation);
        stage.show();
        //scene.setFill(Color.TRANSPARENT);
    }catch (Exception e){
        e.printStackTrace();
    }
}
```

Here, if the choice is "relevant", it will pop up a new view where the teacher will have to fill a textField with more information.

```
else if (selectedRelevance.equals("Very relevant")) {
    try{
        hCCategoryComboBox.getSelectionModel().clearSelection();
        hCSubCategoryComboBox.getSelectionModel().clearSelection();
        relevanceComboBox.getSelectionModel().clearSelection();
        Parent root = FXMLLoader.load(getClass().getResource( name: "/gui/view/veryRelevantView.fxml"));
        Stage stage = new Stage();
        Scene scene = new Scene(root);
        //stage.initStyle(StageStyle.TRANSPARENT);
        stage.setScene(scene);
        stage.setUserData(sendInformation);
        stage.show();
        //scene.setFill(Color.TRANSPARENT);
    }catch (Exception e){
        e.printStackTrace();
    }
}
```

And here, if the choice is "Very relevant", it will pop a new view as well, but they have to fulfil more fields of information in it.

```java
public void createHealthConditionsCitizenTemplate(String selectedCategory, String selectedSubCategory, String selectedRelevance, String professionalNote,
                                                   String assessmentNote, String expectedLevel, String observableNote, LocalDate date, int citizenTemplateId) {
    manager.createHealthConditionsCitizenTemplate(selectedCategory, selectedSubCategory, selectedRelevance, professionalNote, assessmentNote, expectedLevel,
            observableNote, date, citizenTemplateId);
}
```

```java
public void createHealthConditionsCitizenTemplate(String selectedCategory, String selectedSubCategory, String selectedRelevance, String professionalNote,
                                                   String assessmentNote, String expectedLevel, String observableNote, LocalDate date, int citizenTemplateId) {
    healthConditionsCitizenTemplateDAO.createHealthConditionsCitizenTemplate(selectedCategory, selectedSubCategory, selectedRelevance, professionalNote,
            assessmentNote, expectedLevel, observableNote, date, citizenTemplateId);
}
```

```java
public void createHealthConditionsCitizenTemplate(String selectedCategory, String selectedSubCategory, String selectedRelevance, String professionalNote,
                                                   String assessmentNote, String expectedLevel, String observableNote, LocalDate date, int citizenTemplateId) {
    String sql = "INSERT INTO HealthConditionsCitizenTemplate(healthConditionsCitizenTemplateCategory, healthConditionsCitizenTemplateSubCategory, healthConditionsCitizenTemplateColor," +
            " healthConditionsCitizenTemplateProfessionalNote, healthConditionsCitizenTemplateAssessmentNote, healthConditionsCitizenTemplateExpectedLevel," +
            " healthConditionsCitizenTemplateObservableNote, healthConditionsCitizenTemplateDate, citizenTemplateID) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)";
    try(Connection connection = databaseConnector.getConnection()){
        PreparedStatement preparedStatement = connection.prepareStatement(sql);
        preparedStatement.setString( parameterIndex: 1, selectedCategory);
        preparedStatement.setString( parameterIndex: 2, selectedSubCategory);
        preparedStatement.setString( parameterIndex: 3, selectedRelevance);
        preparedStatement.setString( parameterIndex: 4, professionalNote);
        preparedStatement.setString( parameterIndex: 5, assessmentNote);
        preparedStatement.setString( parameterIndex: 6, expectedLevel);
        preparedStatement.setString( parameterIndex: 7, observableNote);
        preparedStatement.setObject( parameterIndex: 8, date);
        preparedStatement.setInt( parameterIndex: 9, citizenTemplateId);
        preparedStatement.executeUpdate();
    } catch (Exception e){
        e.printStackTrace();
    }
}
```

Here is how we send the information to the database through the model, manager and DAO.

```java
try {
    schoolChoiceBox.setItems(userModel.getAllSchoolsNotAssigned());
} catch (Exception e) {
    PopUp.showError(e.getMessage());
}
```

This is the code in the controller to set the items on the choicebox with the schools that are not assigned to any administrator.

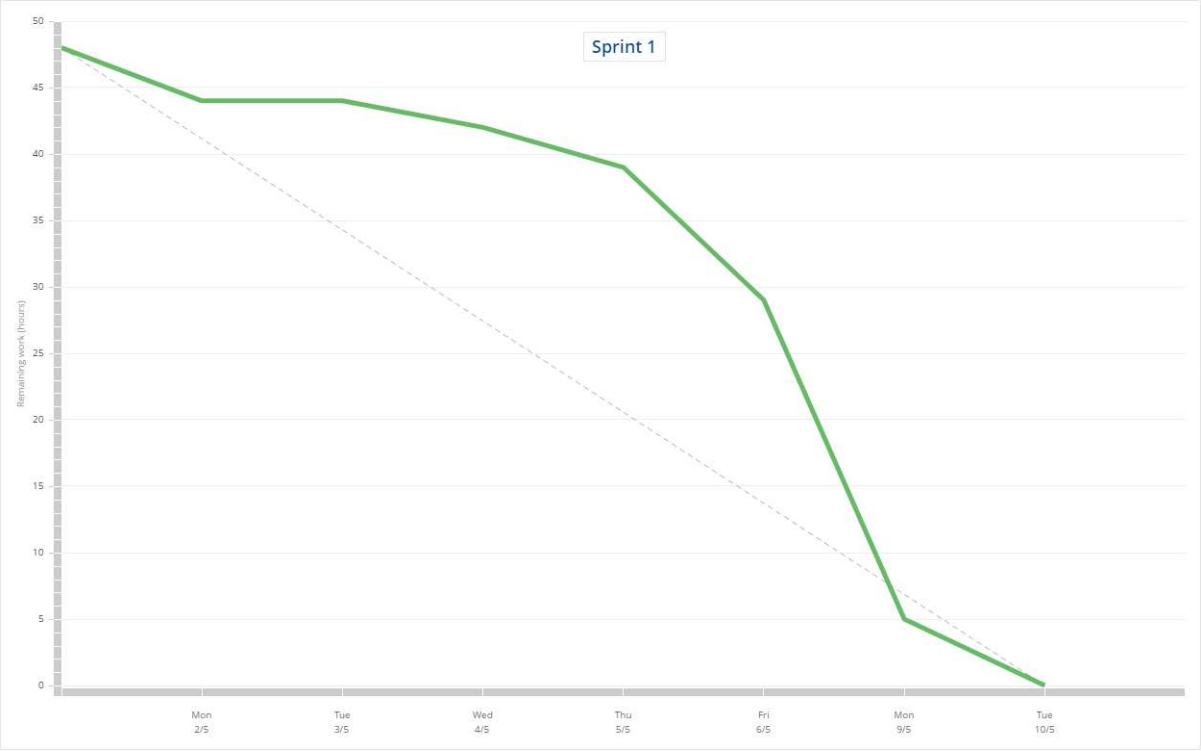
```java
public List<School> getAllSchoolsNotAssigned() throws Exception{
    List<School> allSchoolsNotAssigned = new ArrayList();
    String sql = "SELECT * FROM School WHERE schoolID NOT IN (SELECT school FROM UserData WHERE typeOfUser = ?)";
    try(Connection connection = databaseConnector.getConnection()){
        PreparedStatement preparedStatement = connection.prepareStatement(sql);
        preparedStatement.setInt( parameterIndex: 1,  x: 1);
        ResultSet resultSet = preparedStatement.executeQuery();
        while(resultSet.next()){
            int id = resultSet.getInt( columnLabel: "schoolID");
            String school = resultSet.getString( columnLabel: "school");
            School newSchool = new School(id, school);
            allSchoolsNotAssigned.add(newSchool);
        }
    }
    return allSchoolsNotAssigned;
}
```

Then, when it gets to the database we SELECT the SchoolID from the table School where this SchoolID it's not in any of the users created (meaning table UserData) and where the type of user is an administrator. In the end we just send back all the "schools that are not assigned".

Sprint Review

In the first sprint review meeting the project owners told us that our project was almost done. From their point of view, we only had left the student view and a few requests: having a list of all the health conditions or functional abilities on the citizen and citizen template view, whenever we create a citizen, create one copy for each student in the school and when an admin manages students or teachers, they will only be able to do it for their own school.

Sprint Retrospective



In this sprint, things didn't go as planned. We had to start from scratch a few times, so we had a lot of pressure on ourselves because we thought we were quite behind in progress and we tried to do at least what we planned in the backlog. We decided to leave all the CSS and design for the second sprint in order to complete all our tasks.

As you can see in the image above, this sprint burndown doesn't look nice due to all those issues mentioned earlier but we managed to complete everything in time. We worked really hard during these days trying to understand what we were requested to do and how to implement it. After the meeting, as our project was in a better state than expected we realised that everything proved to work and our efforts were rewarded.

What went well?

As mentioned before, we worked really hard and, as seen in the graph, once we completed our final prototype first task in just a few days we were able to progress and even complete everything related with the code.

What can we improve?

Mostly assigning the right amount of time for each user story, as some stories proved to take different times than expected. Not understanding spoken and written Danish was an issue when gathering information from the project owners, but that doesn't have a short term solution.

Sprint 2

Sprint planning

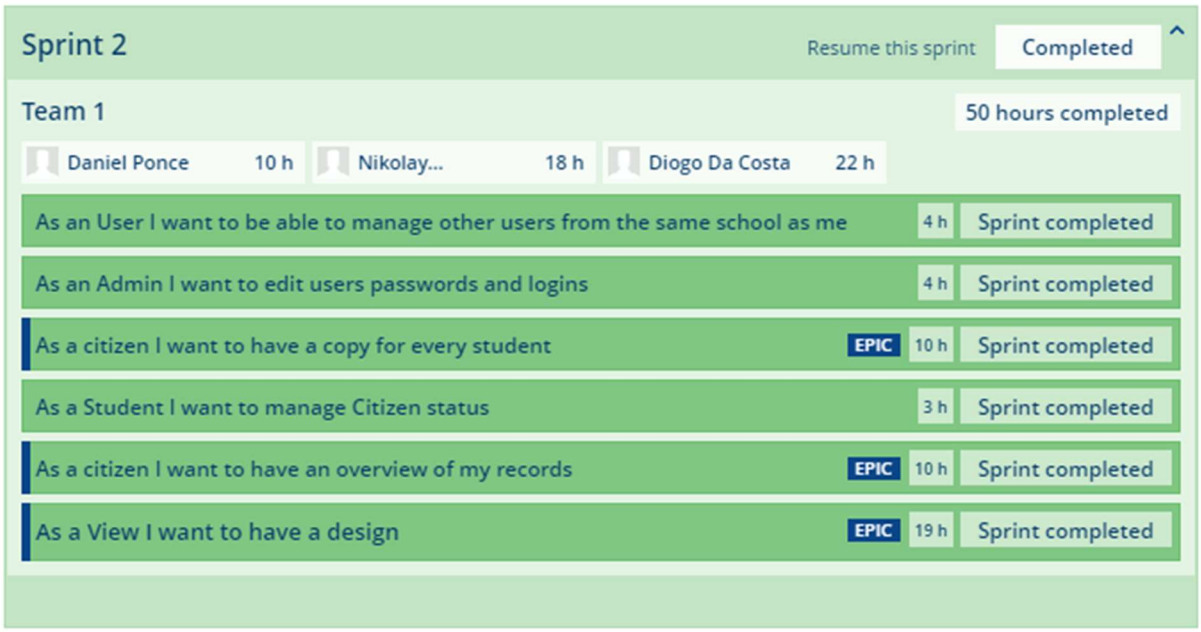
On the first sprint review meeting we had done most of our features, there was only one major feature remaining, but apparently, the information was not transmitted the best way in the beginning of the project and the feature missing was actually already implemented on the project during the first sprint by us.

So, for this sprint we had to make one last view (fxml) and we were pretty much done with the implementation part for the whole project. So we met and discussed that this could be an opportunity to make the project a bit "fancy" and as we were still missing to do all the css, we decided to go with tabs for the whole project instead of popping up new views. This would be something that we never did before and we decided it would be fun to mess around with new stuff and also learn it.

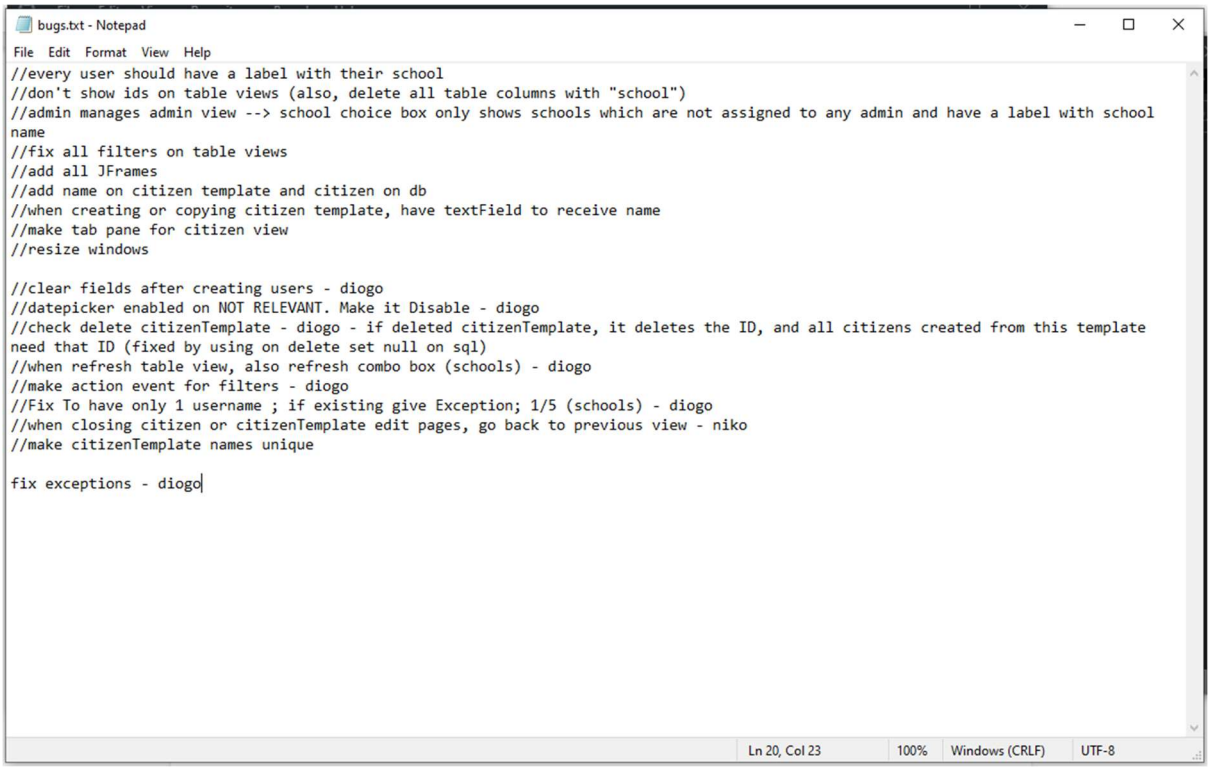
There was also some implementation done on sprint 1 that we were not sure how the project owner would like us to handle, so we also had to make it their way on sprint 2, and so we added some tasks on our scrumwise regarding these improvements.

Apart from all this there were also some bugs or implementations on the project that had to be corrected, so we decided to create a text file in our project root folder with all the bugs that needed to be handled.

So, our backlog for sprint 2 looked like the screenshot below:



And also our "bugs" file looked like the screenshot below:



Daily meetings

For the second sprint instead of creating branches again, as we had less work to do, so to say, we decided to meet everyday and work together in the main branch, so we would understand better what everyone is doing and be more prepared for the exam for when questions for things that we haven't done or participated might occur.

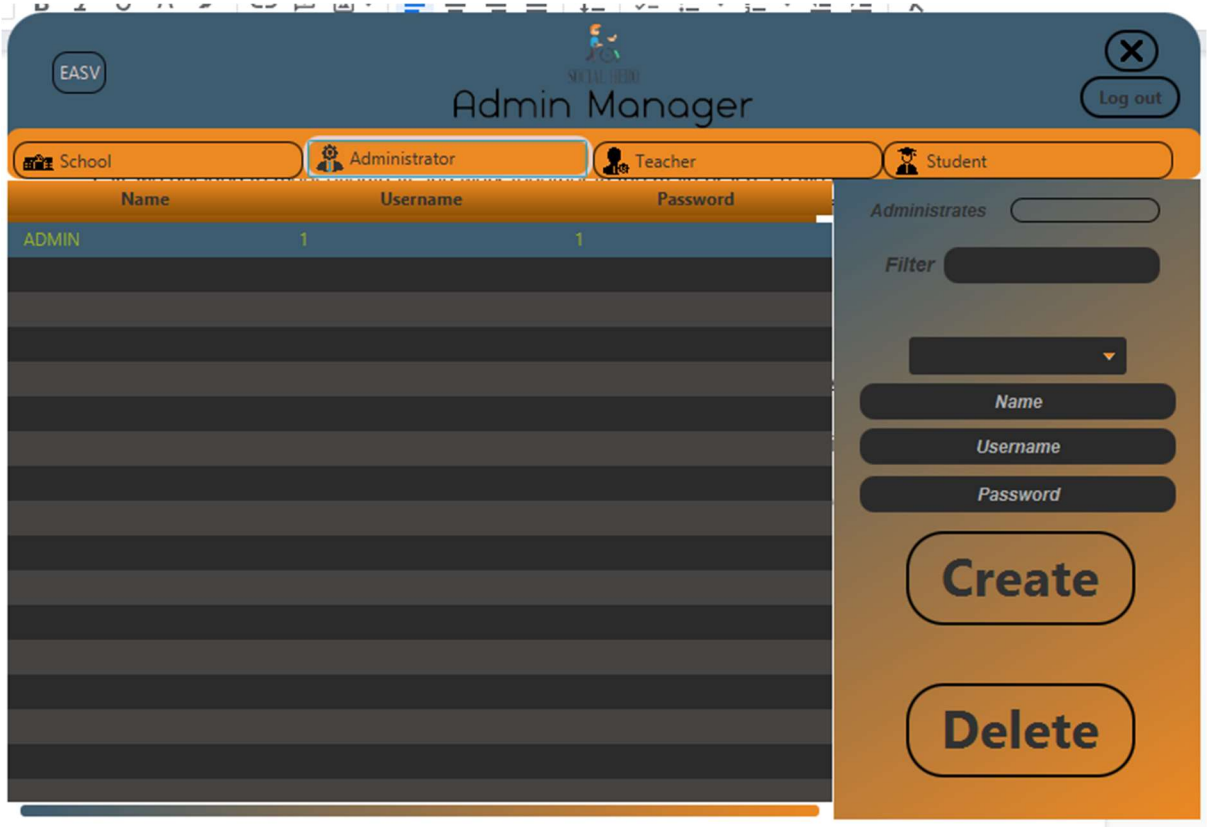
GUI

So, in this sprint we can say that the GUI changed a bit.

First of all we added the Student view and also the Student edits Citizen view. After this we decided to use tabs instead of popping new windows (views), so instead of calling them in the controllers, we made it so that once admin or teacher view initialise, we also initialise all the windows (and controllers) in all the tabs through the fxml files.

We also made all the CSS files that we needed and added them in a folder called css inside the view folder of the GUI.

So with the CSS and the views having tabs from now on, our views would look like the one below:



Data model

As for the data model, we only had to create a new table called Student gets Citizen, that would hold the student ID and the citizen ID, and in this way we could know all the citizens that a specific student has, and bring that citizen up to edit once the student chooses it from the table view that has all his citizens.

We also had to change the Citizen table for the foreign key CitizenTemplateID to be able to be null. With this we could guarantee that if a Citizen Template gets deleted, the Citizens created from that Template would still exist for the Students that had it. So, once it gets deleted we just change whatever value holds the CitizenTemplateID in the Citizen table to null. We used the clause "ON DELETE SET NULL" for that foreign key.

Another change we did in the data model was to set the Citizen Template Name to "UNIQUE", so in this way if a Teacher tries to create a Citizen Template with an existing name, the program will throw an SQL Exception saying that the information entered does not fulfil the needs of the data type in the CitizenTemplate table.

Implementation

When it comes to managing users, citizenTemplates or just Citizens we decided to have table views with the objects that each user can manage.

The first implementation we had on the second sprint was:

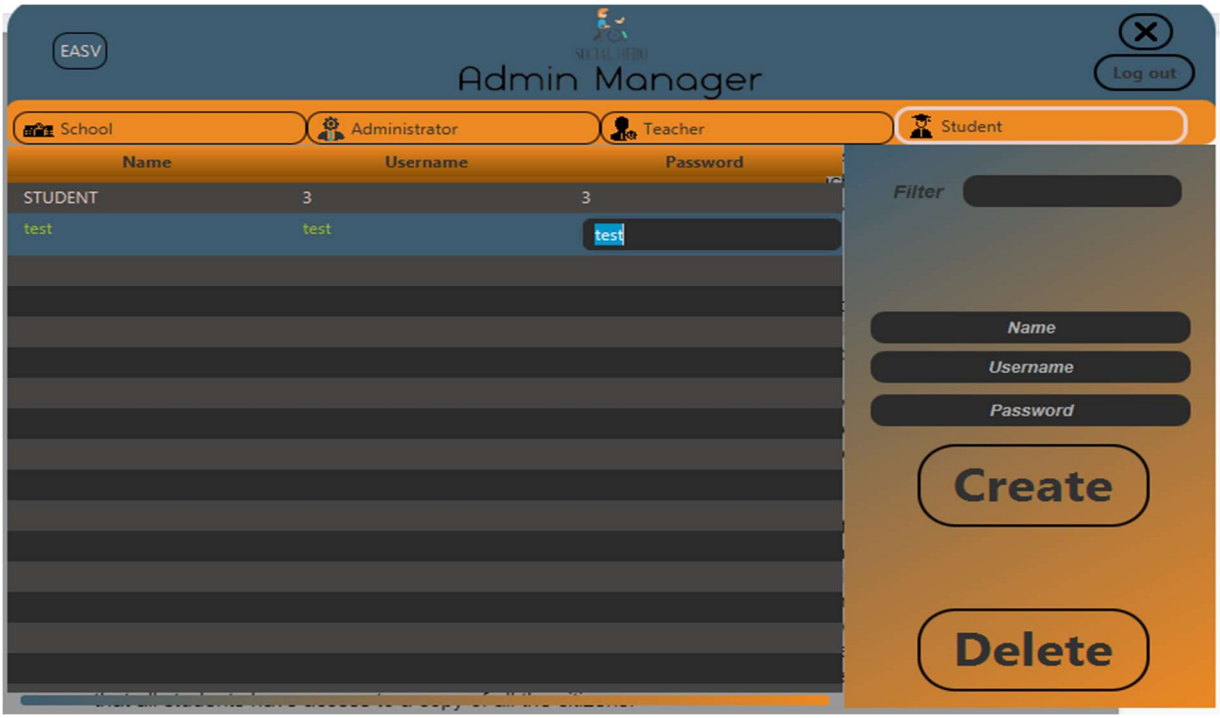
- As an User I want to be able to manage other users that only belong to the same school as I do:

This implementation would include, admins managing teachers and students, and also teachers managing students. So we made a method to send the admin or teacher school to the database and bring to the table views the objects that belong to the same school as the user.

- As an admin I want to edit users passwords and logins

We decided that only the admins would be able to edit this kind of information, so we only made it possible on the admin controller. We made it so that the table view could be edited directly in it, without popping up new windows for editing, we thought this would be a really good implementation to have. So, when we pop all the information on the table views from the databases, we can actually double click on any cell and edit the field. We will then read the cell that was edited and send the new information to the database where we UPDATE the table and refresh the table view instantly to show the new information already edited.

- As a citizen I want to have a copy for every student



This was an implementation that we discussed on the first sprint review, as the project owner wanted to have a fix amount of logins for students and then each student could have a copy of the citizen, but we gave the option to have as many logins as desired and so each student could have his citizen and also privacy and accuracy, because no other student would login in his account and change data without the other one noticing. So when a new student is created in a school and that school already has CitizenTemplates created, the program will create citizens from all the templates and assign them to the new student, so that all students have access to a copy of all the citizens.

And also, when a new citizen is created from a template, we will get a list of all students from that school and create copies from that citizen for all the students and obviously add that information to the Student gets Citizen database.

Results Messages

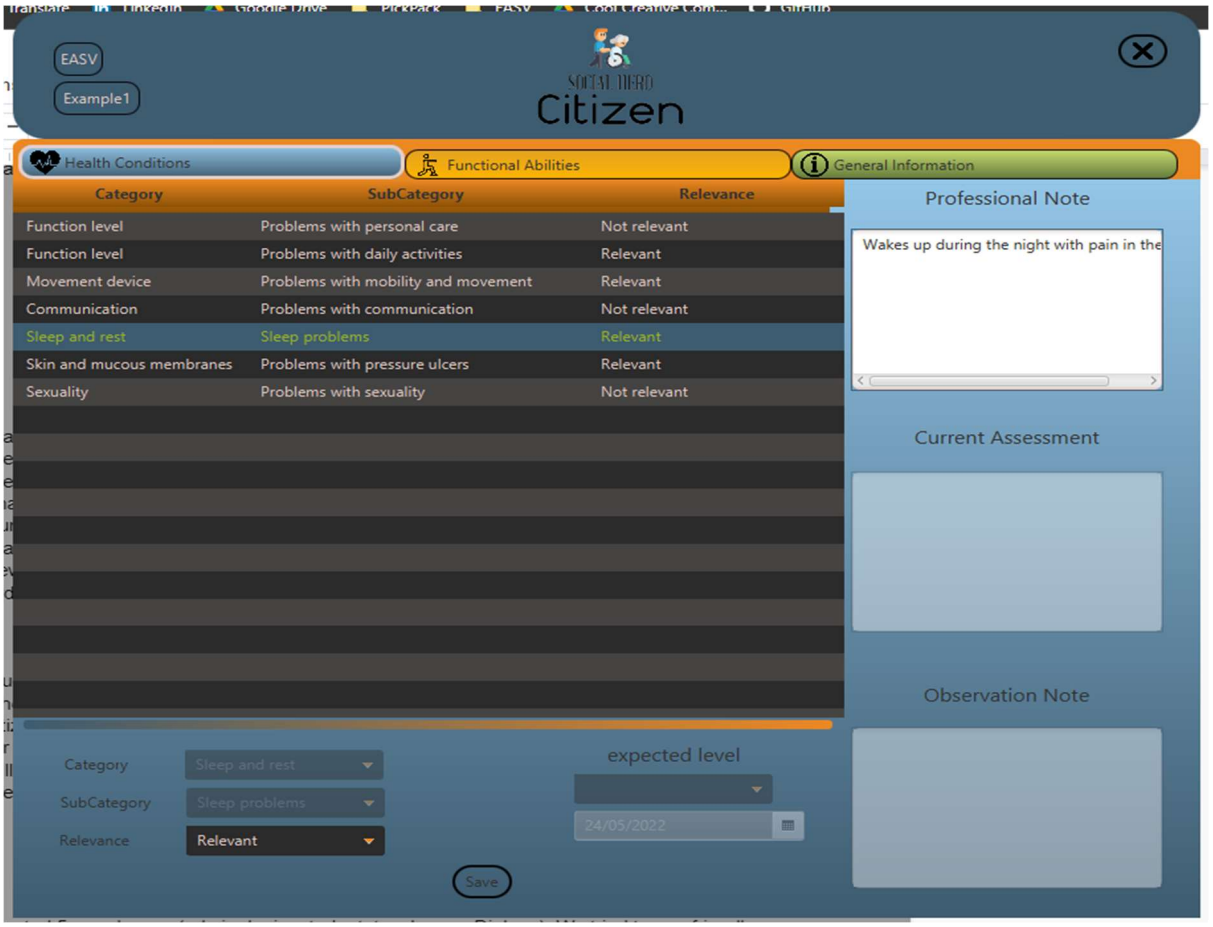
	citizenID	citizenTemplateID	citizenName
1	206	150	Example1
2	207	150	Example1
3	208	150	Example1

- As a student I want to manage Citizen status

For this one, we have a table view on the student view with all the citizens he can manage that were pre created by the teacher and once the student double clicks in one of them, the edit Citizen view pops up. This new window basically functions the same way as the teacher edit Citizen Template, but instead of changing data for the CitizenTemplate, it changes the data for the citizen chosen. On the 3 different tabs (Health Conditions, Functional Abilities and General Info) we bring all the data that is already in the database at that moment, and then the student can choose between editing that data or just creating new. Once he does this we send the info to the database and we check if it already exists in order to create or update the table.

- As a citizen I want to have an overview of my records

This is also something that the project owner asked to be implemented in order for the student to have some kind overview of the data that has already been created, either from when the citizen was created from the template, or a creation that he did before on this citizen. And so we created a table view holding these values, and this also made it possible for the student to just double click on any information inside this table view, and the values will automatically be added to the set fields assigned to them so that the student can edit them quicker.



- As a View I want to have a design

Last but not least, we had to implement the whole CSS throughout the whole project. We created 5 css classes (admin, login, student, teacher, myDialogs). We tried to use friendly colours and also we tried to keep a pattern of the same colours throughout the whole project. We also tried to put small images or icons to be more user-friendly. The myDialogs css class was made only for the pop-up windows, either error windows or process confirmation windows so that the user knows exactly what is happening and if the process he just made worked or not.

Code examples

When editing the Health Conditions tab on the CitizenTemplate or Citizen there are some fields that could only be edited depending on the option chosen on a combo box, so we had to clear and disable some fields or the other way around in order to make this work. The code is shown below:

```
if (relevanceComboBox.getSelectionModel().getSelectedItem().equals("Not relevant") && relevanceComboBox.getSelectionModel().getSelectedItem().equals(healthConditions.get(0))) {
    hCprofessionalNoteTxt.setDisable(true);
    currentAssessmentTxt.setDisable(true);
    hCObservationNoteTxt.setDisable(true);
    hCExpectedLevelComboBox.setDisable(true);
    hCdatePicker.setDisable(true);
    hCdatePicker.setValue(LocalDate.now());
} else if (relevanceComboBox.getSelectionModel().getSelectedItem().equals("Not relevant") && relevanceComboBox.getSelectionModel().getSelectedItem() != healthConditions.get(0)) {
    hCprofessionalNoteTxt.setDisable(true);
    hCprofessionalNoteTxt.setText("");
    currentAssessmentTxt.setDisable(true);
    currentAssessmentTxt.setText("");
    hCdatePicker.setDisable(true);
    hCdatePicker.setValue(LocalDate.now());
    hCObservationNoteTxt.setDisable(true);
    hCObservationNoteTxt.setText("");
    hCExpectedLevelComboBox.getSelectionModel().clearSelection();
    hCExpectedLevelComboBox.setDisable(true);
}
```

For this first one, we're doing the statements for when the combo box gets "Not relevant" as a selected item. When "Not relevant" is selected no fields should be able to be edited.

```
if ((relevanceComboBox.getSelectionModel().getSelectedItem().equals("Relevant") && relevanceComboBox.getSelectionModel().getSelectedItem().equals(healthConditions.get(0))) ||
        (relevanceComboBox.getSelectionModel().getSelectedItem().equals("Relevant") && healthConditions.get(0).equals("Not relevant") && relevanceComboBox.getSelectionModel().getSelectedItem() != healthConditions.get(0))) {
    hCprofessionalNoteTxt.setDisable(false);
    currentAssessmentTxt.setDisable(true);
    hCObservationNoteTxt.setDisable(true);
    hCExpectedLevelComboBox.setItems(expectedLevelList);
    hCExpectedLevelComboBox.setDisable(true);
    hCdatePicker.setDisable(true);
} else if (relevanceComboBox.getSelectionModel().getSelectedItem().equals("Relevant") && relevanceComboBox.getSelectionModel().getSelectedItem() != healthConditions.get(0) && healthConditions.get(0).equals("Very relevant")) {
    currentAssessmentTxt.setDisable(true);
    currentAssessmentTxt.setText("");
    hCdatePicker.setValue(LocalDate.now());
    hCObservationNoteTxt.setDisable(true);
    hCObservationNoteTxt.setText("");
    hCExpectedLevelComboBox.setItems(expectedLevelList);
    hCExpectedLevelComboBox.setDisable(true);
    hCdatePicker.setDisable(true);
}
```

For the second one, we're doing the statements for when the combo box gets "Relevant" as a selected item. When "Relevant" is selected, only the "professional note" text field should be editable.

```
if (relevanceComboBox.getSelectionModel().getSelectedItem().equals("Very relevant")) {
    hCprofessionalNoteTxt.setDisable(false);
    currentAssessmentTxt.setDisable(false);
    hCObservationNoteTxt.setDisable(false);
    hCExpectedLevelComboBox.setItems(expectedLevelList);
    hCExpectedLevelComboBox.setDisable(false);
}
```

For the third one, we're doing the statements for when the combo box gets "Very relevant" as a selected item. When "Very relevant" is selected, all the fields should be editable.

Another problem that we had was that, once we initialise the admin controller, we would also initialise all the controllers for the tabs at the same time. On the admin manages admin view/controller, we have a combo box to choose the school for which the admin we are creating is going to administrate, and we made this combo box display only schools that don't have an administrator yet, so that each school will have one administrator only. But we also have a tab to create schools, and as we initialise all the tabs at the same time, if the first thing we would do would be to create a new school, and right after this we call the admin manages admin class and tried to update that combo box with the new school, it would gives us an error saying that the combo box was null because we were setting the values on that combo box on a running later platform. So we had to somehow find the specific controller that the view was using at that specific time and call the method on that controller instead of the default controller name.

```
@FXML
AdminManagesAdminsController adminController;
```

So, we tried to create a controller object of the admin manages admin controller, which is the controller that has the combo box with the schools but at first we only called it admin because that was the fx:id on that tab, and it was giving us the same error. So, we did some research and we learned that we had to put the word Controller in front of the fx:id name when giving the name to the variable, because that is how Java runs on the backend, we can say.

```
public void adminTabSelected(Event event) {
    adminController.updateSchoolComboBox();
}
```

So, after that we only had to create a method that has an Event for when the tab is pressed, and we call the controller being used at the moment and the method to update the schools combo box.

Below is also the way we decided to check for existing usernames on the databases, in order to not make duplicates:

```
try {
    count = 0;
    List<User> allUsers = userModel.getAllUsernames();
    for (int i=0; i< allUsers.size();i++) {
        if (allUsers.get(i).getUsername().equals(usernameTxt.getText())) {
            //JOptionPane.showMessageDialog(jFrame, "Username already exists, please choose a new one");
            Alert alert = new Alert(Alert.AlertType.NONE);
            alert.setTitle("Error");
            alert.setHeaderText("Username already exists, please choose a new one");
            alert.getDialogPane().getStylesheets().add(getClass().getResource( name: "/gui/view/css/myDialogs.css").toExternalForm());
            alert.getDialogPane().getStyleClass().add("myDialog");
            ButtonType okButton = new ButtonType( text: "OK");
            alert.getButtonTypes().setAll(okButton);
            alert.showAndWait();
            filterStudentTableView();
            count++;
        }
    }
    if (count == 0) {
        userModel.createStudent(schoolId1, nameTxt.getText(), usernameTxt.getText(), passwordTxt.getText());
```

So we start by creating a counter, and then we go to the database and get a list of all the usernames. We then make a for loop to go through all the usernames and compare them to the new username inserted by the user, and if there is any username that equals that one, an error pop-up window will appear and the counter will add 1 to his value. In the end we check, if that counter is still 0 then we are ready to create the user. And we do the same thing for all users trying to be created and for the schools as well.

Below is the code example for when a Student is created:

```java
public User createStudent(int schoolId, String name, String username, String password) {
    List<CitizenTemplate> allTemplates = citizenTemplateDAO.getALlCitizenTemplates(schoolId);
    User student = null;
    if(allTemplates.size() == 0) {
        student = userDataDAO.createStudent(schoolId, name, username, password);
    }
    else {
        student = userDataDAO.createStudent(schoolId, name, username, password);
        for (CitizenTemplate citizenTemplate : allTemplates) {
            Citizen citizen = citizenDAO.createCitizenFromTemplate(citizenTemplate.getCitizenTemplateId(), citizenTemplate.getCitizenTemplateName());
            int citizenId = citizen.getCitizenId();
            healthConditionsCitizenDAO.copyHealthConditionsCitizen(citizenId, citizenTemplate.getCitizenTemplateId());
            functionalAbilitiesCitizenDAO.copyFunctionalAbilitiesCitizen(citizenId, citizenTemplate.getCitizenTemplateId());
            generalInformationCitizenDAO.copyGeneralInformationCitizen(citizenId, citizenTemplate.getCitizenTemplateId());
            assignCitizenToStudent(student.getUserId(), citizenId);
        }
    }
    return student;
    // go to all citizen template from this school, create citizen copy, add citizen to student gets citizen
}
```

First of all we get a list of all the CitizenTemplates that exist in the same school as the student belongs or will belong. Then if there are no CitizenTemplates on that school, we just create the student with the information received from the user. On the other hand, if there are CitizenTemplates created on that school we will first of all create the student and then we will make a for loop for the CitizenTemplates that exist on that school and create citizens from those templates (including all the information that they hold regarding health conditions, functional abilities and general info) and in the end assign those citizens to the student created. So we make sure that every new student has access to all the citizens created at his school.

Sprint Review

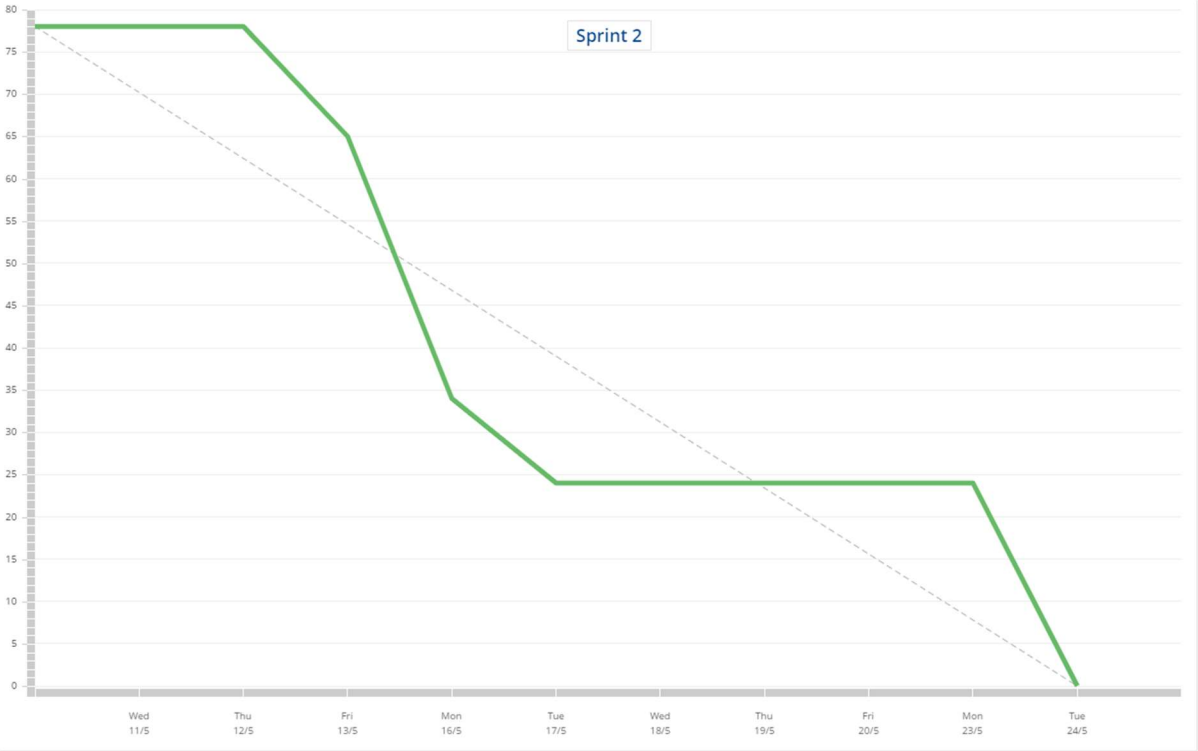
We believe this Sprint was really good, we managed to work together almost everyday and we were happy about the implementations and how in general the project was going.

Our scrumwise didn't look as good as it should because we finished the tasks for the user stories but we had a lot of bugs in our project, so we decided to create a file with all the bugs to start fixing them, and as they don't show up in our scrumwise, it looks like we did nothing for some days, which is not true.

Anyway, we think the project owners quite liked our project and asked us to make some minor changes regarding the looks (CSS) and 1 or 2 minor implementations.

At the end of the meeting we also had 2 comments from our teachers which we also took in consideration for the rest of the project.

Sprint Retrospective



As mentioned before, it looked like we did nothing for some days but as also explained before, that wasn't the case. We thought about adding a new feature along the way but we had to get some insights from the product owner first, and second, we didn't really have the time because a lot of bugs were coming up as we were in the last part of the implementation of the project.

What went well?

We believe the product owner really liked the project, and also they really liked the way we presented it. They also really liked the idea of having the overview of the records in the way that we did it where you can click on them and edit from there on. We would say that the colours chosen and the images/icons made some good impact for them.

We believe we used our time for this Sprint well and maybe we could have done a bit more but we are still happy with the results.

What can we improve?

There were some minor changes that the product owners wanted us to change in order to be able for the user to understand better where is at and some more information regarding which citizen/Template the user is currently working on. Furthermore, we can improve on how we read the information provided to be able to fulfil the product owner needs and maybe try to work better on the branches because it gives that use of more time per person and independence on the implementation. Also, we should have been more careful with our scrumwise regarding the days where we worked on the project but didn't really add any tasks to our scrum board. And, we think another good thing to improve would be to pay more attention to information received by the teacher or fellow students during the project time, where there could be some updates or essential information to conclude the project.

Sprint 3

We received some requests from the product owners regarding the project but our group had already decided that we would use our last week to focus more on the report, regardless we took the first day to fix the things that they have asked us to changed, being them:

- Use different colours for tabs and respective backgrounds
- Have a label on the CitizenTemplate and on the Citizen views with the name of the respective
- Highlight the tab that is currently open
- On the General Info we had buttons that hold the information on what should be on the text fields, and it was asked to make it show up when hover above it instead of clicking on the buttons
- Not to use Java Swing but only Javafx (asked by our teacher)

So we managed to make all these changes together on the first day and work on the report for the rest of the week.

Here we can see the alert windows (javafx) we used instead of the java swing (JFrames) we used to have and the style we use for these windows.

```
Alert alert = new Alert(Alert.AlertType.NONE);
alert.setTitle("Error");
alert.setHeaderText("FIELD IS EMPTY !! PLEASE TRY AGAIN!!");
alert.getDialogPane().getStylesheets().add(getClass().getResource( name: "/gui/view/css/myDialogs.css").toExternalForm());
alert.getDialogPane().getStyleClass().add("myDialog");
ButtonType okButton = new ButtonType( text: "OK");
alert.getButtonTypes().setAll(okButton);
alert.showAndWait();
```

```
.myDialog{
    -fx-background-color: linear-gradient(to right bottom, #3E5C70, #ED8923);
}
.myDialog > *.button-bar > *.container{
    -fx-background-color: linear-gradient(to right bottom, #3E5C70, #ED8923);
}
/*.myDialog > *.label.content{
    -fx-font-size: 14px;
    -fx-font-weight: bold;
}*/
.myDialog:header *.header-panel{
    -fx-background-color: linear-gradient(to right bottom, #3E5C70, #ED8923);

}
.myDialog:header *.header-panel *.label{
    -fx-font-size: 18px;
    -fx-font-style: italic;
    -fx-background-color: linear-gradient(to right bottom, #3E5C70, #ED8923);
    -fx-text-fill: white;
    -fx-text-alignment: center;
}
```

We tried to use the same colours as we did for the rest of the project.

Also, to highlight the tab opened, we did the following style class:

```
.tab-pane .tab:selected{
    -fx-border-color: #E8CFC9;
    -fx-border-width: 3;
    -fx-border-radius: 10;
}
```

Conclusion

In our understanding we believe that we could deliver exactly what was asked of us, which is a software, program, platform that can be used as a student in order to practice the FS3 protocol while in class or at home.

We had a lot of struggles in the beginning because we couldn't understand exactly what was being asked from us and the language was also a barrier between us and the product owner.

We ended up changing or starting over the project 2/3 times in the beginning.

We tried to use as much as we could from the sketches that were given to us by the product owner and it ended up helping us doing it and delivering exactly what was asked from us.

It was also hard for us to give estimated times for certain tasks and also to divide tasks between the members of the group.

We used branches on the first sprint and worked all together in the main branch on the second sprint in order to see what approach would work better for us, but we think that we still don't know. Both have pros and cons.

Even though we didn't have to go to school anymore, all the members of the group have jobs so sometimes it was hard to manage the time.

We always tried to listen to each other, listen to constructive criticism and take outside tips and things to look out for. Also, always take in consideration the product owner requests and the teachers' criticisms.

As this was an Exam and not just an assignment, it was hard to trust others with tasks, but in the end we could work it out and we also must say that we learned a lot with each other, everybody has their creative mind and it was something that we took advantage everytime that we could. This project took a lot of communication between the members, and by the end of it we were well integrated as a team and everybody made an effort to complete and improve the project.

We also learned new stuff with the platforms that we used for the project, them being scrum or java and sql.

We must say that we are really happy with the outcome of the whole project.

References

www.Google.com

www.StackOverFlow.com

Appendices

Appendix A: Team Contract

Team Contract

Team HandsomeKoalas

Diogo Costa
Nikolay Nikolov
Daniel Baez

This document is a collection of all agreed points about the work ethic and collaboration between us. This document will be our constitution during the final exam. All members of the team have read in advance what is written in this document and agree with it.

1. Each member is expected to work 42 hours or more per week.
2. Each member must show on time for the daily meeting in discord.
3. As a team we split the tasks between us and each one of us is responsible for finishing their tasks before the deadline
4. Each member is responsible for push/pull from GitHub before each interference in the code.
5. Each member is responsible for contributing to a friendly, productive work environment in the group
6. In case of arguments between 2 co-workers from the group the 3rd collaborator should take the role of judge and give the final shot.

Creation of Database :

```
USE [HandsomeKoalas_ExamProject]

CREATE TABLE UserType
(
		userTypeID INT IDENTITY(1,1) NOT NULL,
		typeOfUser NVARCHAR(50) NOT NULL,

		PRIMARY KEY (userTypeID)
)

CREATE TABLE School
(
		schoolID INT IDENTITY(1,1) NOT NULL,
		school NVARCHAR(100) NOT NULL,

		PRIMARY KEY (schoolID)
)

CREATE TABLE UserData
(
		userID  INT IDENTITY(1,1) NOT NULL,
		typeOfUser	INT NOT NULL,
		school  INT NOT NULL,
		name    NVARCHAR(100) NOT NULL,
		username	NVARCHAR(50) NOT NULL,
		password	NVARCHAR(50) NOT NULL,

		PRIMARY KEY (userID),
		FOREIGN KEY (typeOfUser) REFERENCES UserType(userTypeID) ON DELETE CASCADE,
		FOREIGN KEY (school) REFERENCES School(schoolID) ON DELETE CASCADE
)

CREATE TABLE CitizenTemplate
(
		citizenTemplateID INT IDENTITY(1,1) NOT NULL,
		school INT NOT NULL,
		citizenTemplateName  NVARCHAR(100) NOT NULL UNIQUE,

		PRIMARY KEY (citizenTemplateID),
		FOREIGN KEY (school) REFERENCES School(schoolID) ON DELETE CASCADE,
)



CREATE TABLE Citizen
(
		citizenID INT IDENTITY(1,1) NOT NULL,
		citizenTemplateID INT,
		citizenName NVARCHAR(100) NOT NULL,

		PRIMARY KEY (citizenID),
		FOREIGN KEY (citizenTemplateID) REFERENCES CitizenTemplate(citizenTemplateID) ON DELETE SET NULL,
)
```

```
CREATE TABLE HealthConditionsCitizenTemplate
(
		healthConditionsCitizenTemplateID INT IDENTITY(1,1) NOT NULL,
		healthConditionsCitizenTemplateCategory NVARCHAR(100) NOT NULL,
		healthConditionsCitizenTemplateSubCategory NVARCHAR(100) NOT NULL,
		healthConditionsCitizenTemplateColor NVARCHAR(50) NOT NULL,
		healthConditionsCitizenTemplateProfessionalNote NVARCHAR(1000),
		healthConditionsCitizenTemplateAssessmentNote NVARCHAR(1000),
		healthConditionsCitizenTemplateExpectedLevel NVARCHAR(100),
		healthConditionsCitizenTemplateObservableNote NVARCHAR(1000),
		healthConditionsCitizenTemplateDate DATE,
		citizenTemplateID INT NOT NULL,

		PRIMARY KEY (healthConditionsCitizenTemplateID),
		FOREIGN KEY (citizenTemplateID) REFERENCES CitizenTemplate(citizenTemplateID) ON DELETE CASCADE
)


CREATE TABLE HealthConditionsCitizen
(
		healthConditionsCitizenID INT IDENTITY(1,1) NOT NULL,
		healthConditionsCitizenCategory NVARCHAR(100) NOT NULL,
		healthConditionsCitizenSubCategory NVARCHAR(100) NOT NULL,
		healthConditionsCitizenColor NVARCHAR(50) NOT NULL,
		healthConditionsCitizenProfessionalNote NVARCHAR(1000),
		healthConditionsCitizenAssessmentNote NVARCHAR(1000),
		healthConditionsCitizenExpectedLevel NVARCHAR(100),
		healthConditionsCitizenObservableNote NVARCHAR(1000),
		healthConditionsCitizenDate DATE,
		citizenID INT NOT NULL,

		PRIMARY KEY (healthConditionsCitizenID),
		FOREIGN KEY (citizenID) REFERENCES Citizen(citizenID) ON DELETE CASCADE
)


CREATE TABLE FunctionalAbilitiesCitizenTemplate
(
		functionalAbilitiesCitizenTemplateID INT IDENTITY(1,1) NOT NULL,
		functionalAbilitiesCitizenTemplateCategoryName NVARCHAR(100) NOT NULL,
		functionalAbilitiesCitizenTemplateSubCategoryName NVARCHAR(100) NOT NULL,
		functionalAbilitiesCitizenTemplateScore INT NOT NULL,
		functionalAbilitiesCitizenTemplateExpectedScore INT NOT NULL,
		functionalAbilitiesCitizenTemplateProfessionalNote NVARCHAR(1000),
		functionalAbilitiesCitizenTemplatePerformance NVARCHAR(100) NOT NULL,
		functionalAbilitiesCitizenTemplateLimitation NVARCHAR(100) NOT NULL,
		functionalAbilitiesCitizenTemplateGoalsNote NVARCHAR(1000),
		functionalAbilitiesCitizenTemplateObservationNote NVARCHAR(1000),
		functionalAbilitiesCitizenTemplateDate DATE,
		citizenTemplateID INT NOT NULL,

		PRIMARY KEY (functionalAbilitiesCitizenTemplateID),
		FOREIGN KEY (citizenTemplateID) REFERENCES CitizenTemplate(citizenTemplateID) ON DELETE CASCADE
)
```

```
CREATE TABLE FunctionalAbilitiesCitizen
(
		functionalAbilitiesCitizenID INT IDENTITY(1,1) NOT NULL,
		functionalAbilitiesCitizenCategoryName NVARCHAR(100) NOT NULL,
		functionalAbilitiesCitizenSubCategoryName NVARCHAR(100) NOT NULL,
		functionalAbilitiesCitizenScore INT NOT NULL,
		functionalAbilitiesCitizenExpectedScore INT NOT NULL,
		functionalAbilitiesCitizenProfessionalNote NVARCHAR(1000),
		functionalAbilitiesCitizenPerformance NVARCHAR(100) NOT NULL,
		functionalAbilitiesCitizenLimitation NVARCHAR(100) NOT NULL,
		functionalAbilitiesCitizenGoalsNote NVARCHAR(1000),
		functionalAbilitiesCitizenObservationNote NVARCHAR(1000),
		functionalAbilitiesCitizenTemplateDate DATE,
		citizenID INT NOT NULL,

		PRIMARY KEY (functionalAbilitiesCitizenID),
		FOREIGN KEY (citizenID) REFERENCES Citizen(citizenID) ON DELETE CASCADE
)

CREATE TABLE GeneralInformationCitizenTemplate
(
		generalInformationCitizenTemplateID INT IDENTITY(1,1) NOT NULL,
		generalInformationCitizenTemplateName NVARCHAR(100) NOT NULL,
		generalInformationCitizenTemplateExplanation NVARCHAR(1000) NOT NULL,
		generalInformationCitizenTemplateEditable NVARCHAR(1000) NOT NULL,
		citizenTemplateID INT NOT NULL,

		PRIMARY KEY (generalInformationCitizenTemplateID),
		FOREIGN KEY (citizenTemplateID) REFERENCES CitizenTemplate(citizenTemplateID) ON DELETE CASCADE
)

CREATE TABLE GeneralInformationCitizen
(
		generalInformationCitizenID INT IDENTITY(1,1) NOT NULL,
		generalInformationCitizenName NVARCHAR(100) NOT NULL,
		generalInformationCitizenExplanation NVARCHAR(1000) NOT NULL,
		generalInformationCitizenEditable NVARCHAR(1000) NOT NULL,
		citizenID INT NOT NULL,

		PRIMARY KEY (generalInformationCitizenID),
		FOREIGN KEY (citizenID) REFERENCES Citizen(citizenID) ON DELETE CASCADE
)

CREATE TABLE StudentGetsCitizen
(
		studentID INT NOT NULL,
		citizenID INT NOT NULL,

		PRIMARY KEY (studentID, citizenID),
		FOREIGN KEY (studentID) REFERENCES UserData(UserID) ON DELETE CASCADE,
		FOREIGN KEY (citizenID) REFERENCES Citizen(citizenID) ON DELETE CASCADE
)
```