

**Project report**

**Multiplayer racing game**

**Date**:

28.5.2017

**Made by:**

Alexandru Ungureanu (224469)

Dominik Lukac (239837)

Vladimir Rocin (239854)

**Supervisors:**

Ib Havn

Lars Sorensen

Richard Brooks

Table of figures

[Abstract 5](#_Toc452635970)

[1. Introduction 6](#_Toc452635971)

[2. Analysis 7](#_Toc452635972)

[2.1 Requirements 7](#_Toc452635973)

[2.2 Key features 9](#_Toc452635974)

[2.3 Constraints 10](#_Toc452635975)

[2.4 Risks 10](#_Toc452635976)

[2.5 Use case diagram 11](#_Toc452635977)

[2.6 Actor description 12](#_Toc452635978)

[2.7 Use case description 13](#_Toc452635980)

[2.8 Activity diagram 14](#_Toc452635981)

[2.9 Class diagram 15](#_Toc452635982)

[3. Design 16](#_Toc452635983)

[3.1 Design Class Diagram 16](#_Toc452635984)

[3.2 GUI design 18](#_Toc452635985)

[3.3 Sequence diagram 20](#_Toc452635987)

[4. Implementation 21](#_Toc452635988)

[4.1 Database 21](#_Toc452635989)

[4.2 Controller 26](#_Toc452635990)

[5. Testing 29](#_Toc452635991)

[6. Discussion 30](#_Toc452635992)

[6.1 AdminWindow 30](#_Toc452635993)

[6.2 RMI 30](#_Toc452635994)

[6.3 Stop/start election 30](#_Toc452635995)

[6.4 No history of votes 30](#_Toc452635996)

[7. Results 31](#_Toc452635997)

[8. Conclusion 32](#_Toc452635998)

[9. References 33](#_Toc452635999)

Abstract

*The goal of this project is to develop a real-time application based on freeRTOS which will run on AVR micro-controller. This include also design and implementation of protocol which describes communication between the AVR and PC via usb serial connection. The application has a form of the game for two players, in our case racing/avoiding obstacles.*

1. Introduction

We decided to create a multiplayer racing game. Cars must avoid obstacles and each other. If a car hits anything in front of it, it is destroyed and the other player wins. Obstacles will be spawned at one end of the display and move towards the other one to simulate progressing movement of the cars. A car would be represented by two pixels/diodes and would be allowed to move freely around, while obstacles would move towards cars and then out of screen once the cars pass them. The goal of the game is to last longer than the opponent.

The map would be either generated randomly with some restrictions so there is always a path or it would be hard coded. This depends on how quickly it is possible to work out the rest of the project.

This supposes that a suited communication protocol be agreed upon. Furthermore, the interface and flow of the application have to be derived from the above stated conditions.

1. Analysis

The analysis phase is used for deciding the game rules, constraints and features. Also overall run of the application out of the game from the users point of view.

This phase includes making:

* + The list of requirements
  + Use case diagrams
  + Use case descriptions
  + Activity diagram
  1. Requirements
     1. Functional requirements

The following list of requirements which shows everything the system will have to be capable of is based on the first meetings of the group but had also been modified through the development process as we found out, that some things can be done better or easier. The list is divided by the type of user and ordered by the importance of the requirements.

**Player 1 (on game console):**

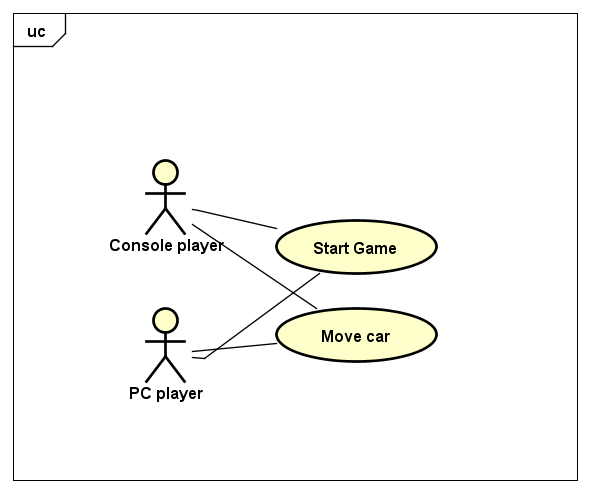
* See the movement of the car on the display
* Move with the car to sides using joystick
* At the end of the game, display who won
* Timer controlled tasks DMS or RMS

**Player 2 (on PC/Laptop):**

* Start/Load the game
* Play by using simple inputs (arrows) instead of the controller
* Can see the score, game time, progress of the game
* End/terminate the game
* At the end of the game, see who won

**Other**

* Cars can collide
* If a car makes a move towards an edge of the display it appears on the other side, but this move is possible only sideways.
* Movement into obstacle is not allowed but only counts as loss if it is frontal collision.
* Game results are displayed constantly on the PC console
  + 1. Non-funcitional requirements
* The system must be implemented with FreeRTOS
* Need to have our protocol with flow control and error detection
* Serial connection between game-console and PC
* Code should be unit tested
* We must use semaphores or mutexes
* We use only given hardware (ATmega 3247, DOT matrix, usb serial communication, joystick, R2R DA)
* PC app in C
* Use oscilloscope for computation time measurements
  1. Constraints
* Game is only for two players
* Application does not keep track of games before restart of the controller.
* No GUI
* Difficulty is fixed
  1. Use case diagram

Based on requirements the use case diagram shows all the functions that are needed. It is not too complicated since the game itself is quite simple. Also limited input options and display capabilities does not allow too much complexity.

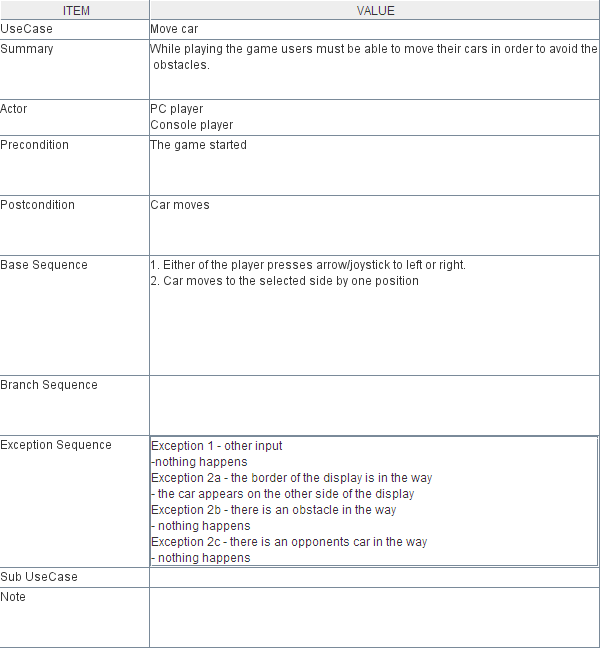
*Figure 1 – use case diagram*

* 1. Actor description

In the system there are two types of users. In the actor description they are listed and described.

* PC player: controls the car from PC keyboard but still must look at the console display to see the game.
* Console player: controls the car with joystick which is a part of the board.
  1. Use case description

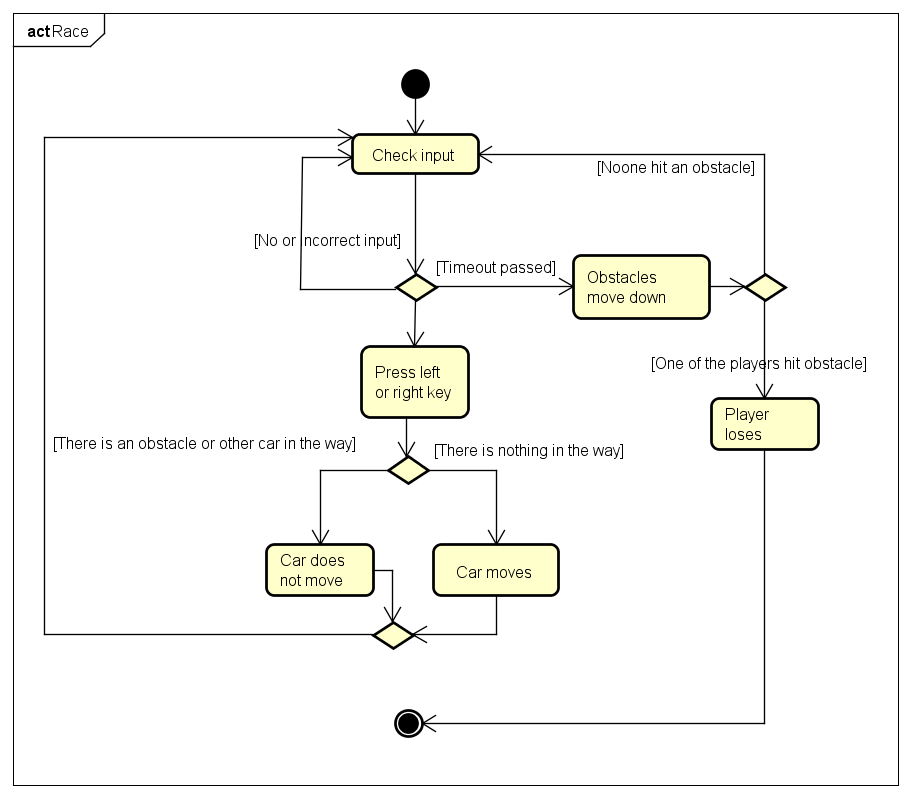
Here we show the use case description of the most essential action of the the game which is moving the car.



*Figure 2 – the use case description*

* 1. Activity diagram

The activity diagram displays how is the car movement is managed. It describes the whole game.



*Figure 3 – the activity diagram*

* 1. Class diagram

1. Design
   1. Design Class Diagram

The design phase was essential for giving the program its final and specific shape. We decided in what way we create the GUI, which tables we are going to use and what data they store.

* + 1. Model

Model package consist of key classes, which contain main entities connected with the elections and its needs. It contains **Election, Position, Candidate, *User*, Admin, Voter.**

**Election** is implemented as a singleton for the purpose of tracking if a voter can vote or if admin can add or remove candidates and positions.

***User***is an abstract class and is extended by the **Admin** and the **Voter** classes. Both these classes use username and password String and admin Boolean, which is defining the type of user. For the **Voter** there is one more important Boolean field voted which keeps track if the voter already voted or not.

**Position** class serves the purpose of keeping track of created positions so they can be selected while adding new candidates. It contains only one field which is String and carries the name of the position.

**Candidate** class contains **Position** field, a String field for candidate`s name and a counter for votes which candidate received.

* + 1. View

View contains classes which allow user to control the program and display results. Classes creates graphical user interface using JSwing. They are synchronized by the **Controller** class.

It is contained within three JFrames plus dialog messages. JFrames are **AdminWindow, VoterWindow** and **LogInWindow.**

**LogInWindow** starts with the program. Its purpose is to collect username and password inserted by the user. **Controller** then sends data to the database and receives either an admin object, voter object or nothing. Based on this acquire **Controller** closes the frame and opens a new one.

**VoterWindow** is created when the **Controller** receives a **Voter** object. It gives the user a possibility to vote for candidates at selected positions. After log out the **LogInWindow** is created again.

**AdminWindow** is created when the **Controller** receives an **Admin** object. It contains tabs that open panels for different actions like changing password, adding candidates and positions, starting election etc. After log out the **LogInWindow** is created again.

* + 1. Controller

Controller package contains **Controller** and **IController**.

**Controller** implements **IController** and gives the GUI its functionality. It communicates with both the GUI and the database through the **DBManager**. It sends all the values for filling the JLists and JComboBoxes and also values entered through the GUI into the database using the **DBManager**.

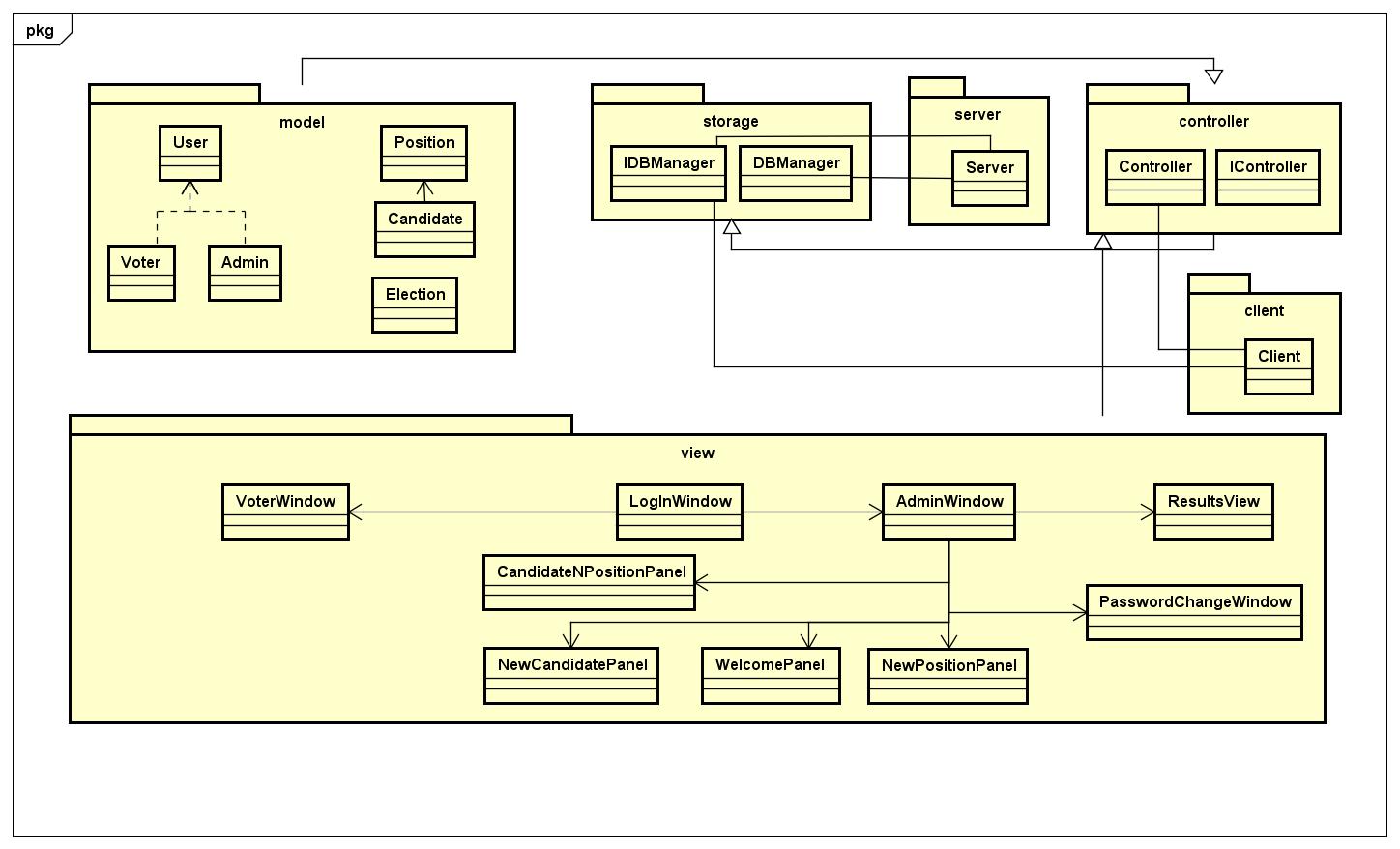
* + 1. Storage

Storage package contains **DBManager** which implements **IDBManager**. It receives requestsfrom the **Controller**, collects data from database accordingly and send them back. It also receives data from **Controller** and saves it into the database.

* + 1. Server

**Server** keeps the database running and waits for clients to connect and do what they can.

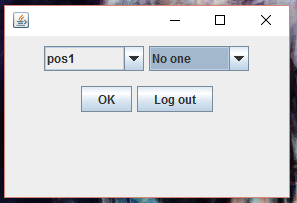
* + 1. Client

Client is a tool for communication between **Server** and the user. It uses its own **Controller** to run and the **IDBManager** to command the **Server**. It runs the GUI and do most of the functionality.

*Figure 5 – overall class diagram*

* 1. GUI design
     1. Main GUI

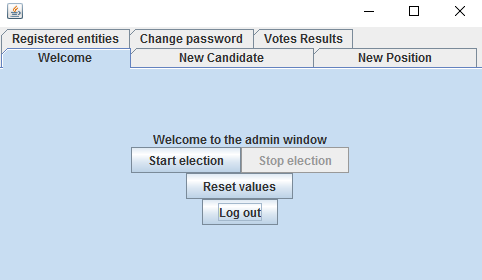
Taking into consideration the requirements of the Voting system the main screen which will be displayed for the most people will look like this:



Main GUI is divided into two parts: The middle part consisting of two buttons and the top part consisting of two dropdowns. The dropdown on the right side is changing depending in what the user chooses in the left dropdown

In the middle part the left button “OK” is the button, which calls a method vote, and that one sends the votes to the database and sets the voter status to voted. The right button logs out the voter and goes back to the login window. Further the left dropdown shows all the positions and when the user chooses first time a new position to vote he have an option to vote “No one”. If he votes somebody he cannot choose again “No one”. Furthermore the right dropdown updates every time when the voter chooses a new position to vote, unless he already chosen and voted for someone on that position, then he can only see, for whom he voted and choose someone else if he want to. After he clicks the “OK” button he cannot log in again, until the admin reset the whole election.

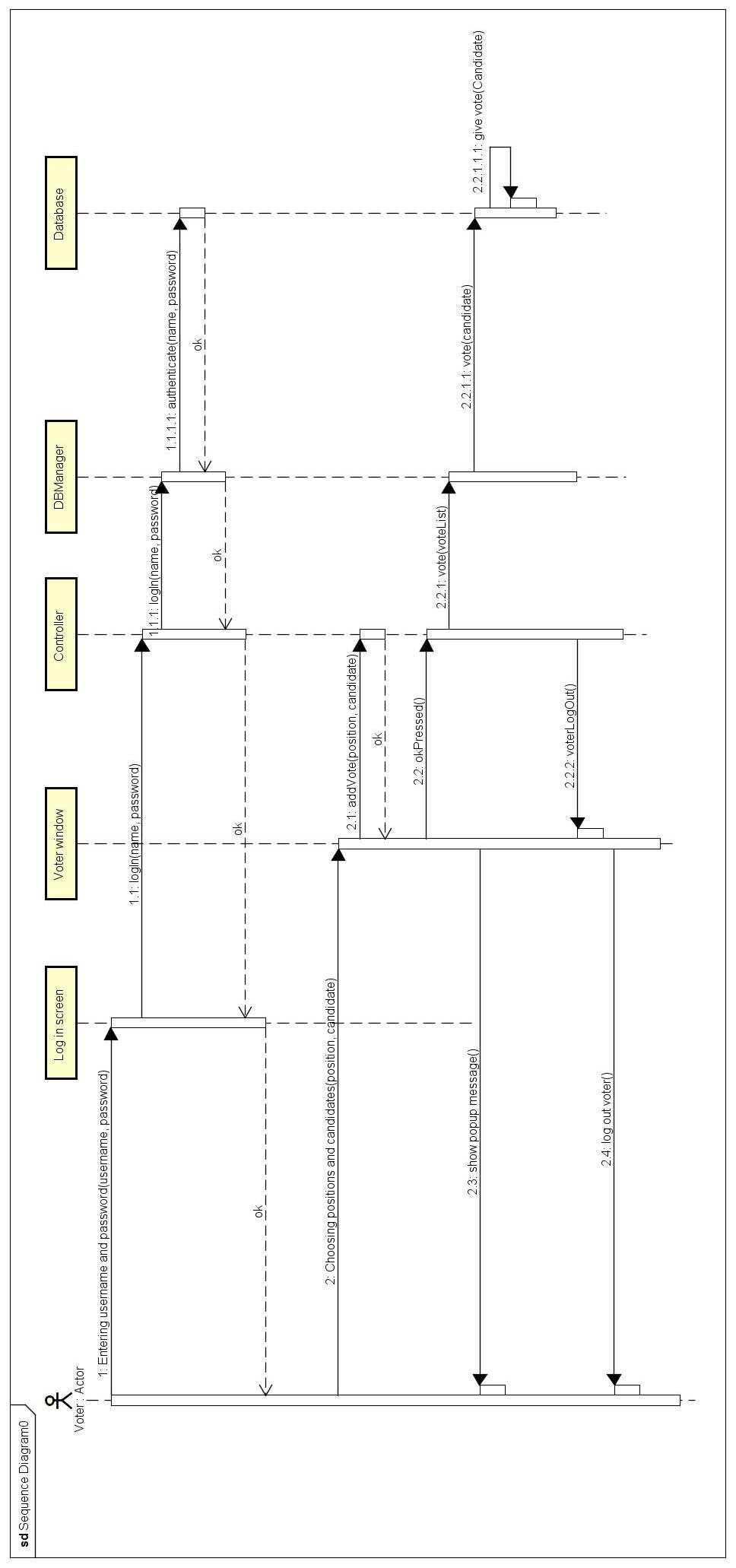
* + 1. Admin GUI



Admin GUI is divided into multiple parts. The main parts of it are the tabs, which gives us an advantage in having everything in one frame, even thou there are so many things. The main part of that admin GUI is the “welcome” tab. This tab allows to the admin to start the election, if there is any active election at the moment or stop it if there is some active election running. This tab also allows to the admin to reset the election in case it is over and he want to run a new one with new positions and candidates. This “Reset values” button is only available if there is not an active election. If the admin want to add a new candidate or a new position he just clicks a tab, whit what he want to do, and create them, or make what is necessary.

* 1. Sequence diagram

The sequence diagram shows what happens in the program step by step when a user logs in as a voter and saves a vote. The user enters their username and password, which is passed through all the way down to the database, where the input is verified. After that (if both the username and the password is correct) the voter window opens for the user, where they can choose positions, candidates and vote. The information about the user’s vote is sent to the database, and a confirmation message pops up for the user. After the user presses “ok” on the message, they are logged out automatically.



*Figure 8 – the sequence diagram*

1. Implementation

In the design phase, we developed and implemented the components and application features. While we were working on the development, we also placed a lot of emphasis to the management of our time. Which means, we tried to optimize our schedules to work as effectively as we could and also at the end to have the best quality product. We met almost every day to coordinate the development of the program and did use the “git” software for code-sharing.

* 1. Database
     1. SQL Database

For this project we have used an SQL database in order to save the data about the users and the elections. To create a connection between the database and the program itself we have used a JDBC driver and created a DBManager class, which is responsible between logical communication between the database and the controller.

The SQL code:

CREATE SCHEMA Vote;

The Users table contains information about the users using the system. A name (username), a password, whether the user is an admin or not and whether the user (voter) has already voted or not.

CREATE TABLE Users (

name VARCHAR(40),

password VARCHAR(40),

admin BOOLEAN,

voted BOOLEAN);

The Candidates table contains information about the candidates of the actual election. A name, a position (name of the position) and a vote count (number of votes the candidate has).

CREATE TABLE Candidates(

name VARCHAR(40),

position VARCHAR(40),

voteCount INT NOT NULL);

The Election table contains information about the state of the actual election, whether if it’s active or not.

CREATE TABLE Election(

active BOOLEAN);

The Positions table contains information about the positions in the election (names of the positions).

CREATE TABLE Positions(

name VARCHAR(40));

The users (admins and voters) input data into the GUI. This data is handled by the controller, which it transfers to the DBManager after certain logical functions. Then the DBManager communicates with the SQL database using up the data it has recieved.

* + 1. Database driver

The **logIn** method is one of the most important methods in the **DBManager** class, it returns a User object (Voter or Admin) or null to the Controller, which it will use to determine what action to take next.

**public** User logIn(String username, String password)

{

Boolean adminBoolean = **false**; //It will store a boolean whether the searched user in the database is an admin or not.

Boolean voted = **true**; //It will store a boolean whether the voter in the database has already voted or not.

Boolean active = **false**; //It will store a boolean whether the election is active or not.

**try**

{

connection = DriverManager.*getConnection*(

"jdbc:postgresql://localhost:5432/postgres", "postgres", "pass");

}

**catch** (SQLException e)

{

}

// Returning data from the database.

**try**

{

PreparedStatement statement = connection.prepareStatement(

"SELECT \* FROM Users WHERE name = ? AND password = ?");

statement.setString(1, username);

statement.setString(2, password);

ResultSet result = statement.executeQuery();

**while** (result.next())

{

adminBoolean = result.getBoolean("admin");

voted = result.getBoolean("voted");

}

statement = connection.prepareStatement("SELECT \* FROM election");

result = statement.executeQuery();

result.next();

active = result.getBoolean("active");

}

**catch** (SQLException e)

{

}

**finally**

{

**try**

{

connection.close();

}

**catch** (SQLException e)

{

}

}

// Creating a User object based on the state of the boolean objects.

**try**

{

**if** (adminBoolean)

{

admin = **new** Admin(username, password);

**return** admin;

}

**else** **if** (!voted && active)

{

voter = **new** Voter(username, password, voted);

**return** voter;

}

}

**catch** (Exception e)

{

}

**return** **null**;

}

The **vote** method is mainly responsible of recording the votes of the voters in the database, incrementing the vote count of a certain candidate based on the name of the candidate in the method’s argument (which is provided by the controller).

**public** **void** vote(Candidate candidate)

{

**...**

**try**

{

PreparedStatement statement = connection.prepareStatement(

"UPDATE Candidates SET voteCount = ? WHERE name = ?");

candidate.giveVote();

statement.setInt(1, candidate.getVotes());

statement.setString(2, candidate.getName());

statement.executeUpdate();

statement = connection

.prepareStatement("UPDATE Users SET voted = 'true' WHERE name = ?");

statement.setString(1, voter.getName());

statement.executeUpdate();

}

The **reset** method makes it possible to empty the Candidates and the Positions table in the database, also making sure that voter users are allowed to vote again on a new election.

**public** **void** reset()

{

...

**try**

{

PreparedStatement statement1 = connection

.prepareStatement("DELETE FROM Candidates");

PreparedStatement statement2 = connection

.prepareStatement("DELETE FROM Positions");

PreparedStatement statement3 = connection

.prepareStatement("UPDATE Users SET voted = 'false' WHERE voted = 'true'");

statement1.executeUpdate();

statement2.executeUpdate();

statement3.executeUpdate();

}

* 1. Controller

**Log in** method ask the database for the list of the users with name and password. Then it compares this list with the username and password given in the login field. After it, when it finds the user, which owns that username and password, it will open a new window depending on wat kind of user he is. If admin, then admin window, if voter then voter window.

The code is shown below.

**public** **void** logIn(String name, **char**[] passwordIn)

/\*

\* Check if the password and name are in the database by calling the database

\* opening the admin window if its a admin

\* opening the voter window if its a voter

\*/

{

**if** (!name.equals(""))

{

String password = **new** String(passwordIn);

user = **new** User(name, password);

**try**

{

**if** (dbm.logIn(name, password) **instanceof** Admin)

{

logIn.close();

adminW = **new** AdminWindow(**this**);

logIn.clearFields();

}

**else** **if** (dbm.logIn(name, password) **instanceof** Voter)

{

logIn.close();

voterW = **new** VoterWindow(**this**);

logIn.clearFields();

}

**else**

{

logIn.clearFields();

JOptionPane.*showMessageDialog*(**new** JFrame(), "Unable to log in. Check your username and password. You can only vote once.");

//The option pane shows the dialog message if something is wrong with the password or username

}

}

**catch** (Exception e)

{

}

}

}

**Vote** method is in a close relationship with another methods called **addVote** and **okPressed**. The **addVote** method is collecting the votes which have been selected by the voter in the GUI drop down. When the user clicks the ok button the **okPressed** method is being called. This method is just calling the vote method.

Vote method is shown below

**public** **void** vote()

{

**for** (**int** i = 0; i < voteList.size(); i++)

{

**try**

{

dbm.vote(voteList.get(i));

}

**catch** (Exception e)

{

}

}

voteList = **new** ArrayList<Candidate>();

}

**addVote** method is shown below

**public** **void** addVote(String name, String pos)

{

**for** (**int** i = 0; i < voteList.size(); i++)

{

**if** (voteList.get(i).getPosition().equals(pos))

voteList.remove(i);

}

**try**

{

voteList.add(dbm.getCandidate(name, pos));

}

**catch** (Exception e)

{

}

}

1. Testing

The program has been tested by the using of all the required features through the GUI with multiple clients running at one time.

|  |  |
| --- | --- |
| * Log in as an administrator with his username and password | Passed |
| * Log out | Passed |
| * Change password | Passed |
| * Create position | Passed |
| * Delete positions together with all the candidates for that position | Passed |
| * Add candidate after selecting a position | Passed |
| * Remove candidates | Passed |
| * Start or stop election | Passed |
| * Remove everything from the tables of positions and candidates while allowing all the voters in the database to vote again | Passed |
| * See the results of the election at any time | Passed |
| * See all the registered candidates and positions | Passed |
| * Log in as a voter with his username and password if he did not vote yet and if the election has been started by an admin account | Passed |
| * Select multiple positions and one candidate for every position to vote for | Passed |
| * Send his votes to the database | Passed |
| * Log out without voting | Passed |
| * Preventing voters from voting more than once | Passed |

1. Discussion

There are few design choices that should be discussed more in-deth.

* 1. AdminWindow

We decided to use JTabbedPane on the AdminWindow to avoid having to many windows opening and closing and having buttons to switch between them at the same time. We were also interested about learning how to use the JTabbedPane.

* 1. RMI

The reason for using RMI for client-server implementation of the program is that we do not have to write any protocols or anything else. We just get the remote interface to control DBManager on the server so the Controller of the Client can retrieve and save data.

* 1. Stop/start election

We decided on having possibility do stop election because it seems to be right to not be able to vote outside the election period. And also to disable administrator from inserting new candidates or positions when the election is running.

* 1. No history of votes

We decided to not keep any track or history of votes to keep any election running on our system anonymous.

1. Results

After the code was done and put together it was time to try to run it on an RMI server and RMI client on two different computers and test how it works. We created a small table of users just for testing purposes. After assuring the key features works we tested the less important ones and made sure they work as well. During the testing we realized we forgot about few features that could be useful and clever to have. So we returned to the code a few times before calling it fully functional. When we were done we decided to polish it, delete the fields we did not use in the end, unnecessary methods and code that was commented out.

The software is capable of creating an environment for running an elections. It offers an anonymous election for multiple position with multiple candidates. The database of users has to be imported through another program. It prevents voters from voting more than once and it is possible to prevent voters from voting by stopping the election through the administrator. All the values except registered voters can be deleted by resetting the system.

1. Conclusion

The project was about creating a client-server system which uses a database on the topic of our choice. We choose to implements a system for DSR elections. The system was developed according to the requirements set at the beginning of the project and the result is relatively generic election system that can be used even for different types of elections. It is even possible to run multiple clients at a time while keeping the functionality. The table of users has to be imported externally and cannot be managed through the system. The table of positions and the table of candidates can managed through the system and can be also imported.

1. References

**[1]** Gaddis, G.T. (2014). Creating a simple GUI app, Starting Out with Java. Great Britain: Pearson Education.

**[2]** Gaddis, G.T. (2014). Handling events, Starting Out with Java. Great Britain: Pearson Education.

**[3]** <https://www.youtube.com/watch?v=Ou2A-JWszVA> December 2015

**[4]** http://mathbits.com/MathBits/Java/arrays/ABCSort.htm December 2015

**[5]** <http://www.w3schools.com/xml/> December 2015

**[6]** https://docs.oracle.com/javase/tutorial/uiswing/layout/gridbag.htmlDecember 2015

**[7]** <http://stackoverflow.com/questions/5212983/jlist-add-remove-item> December 2015