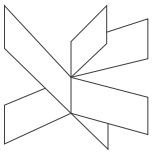


[SCRUM management system]

Students

Borislav Aleksiev 279970

Dziugas Austys 280144

Ronald Johnson 279987

Przemyslaw Regulski 280196

Supervisors

Joseph Chukwudi Okika

Jakob Knop Rasmussen

ICT ENGINEERING

THIRD SEMESTER 2019

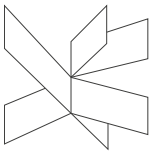
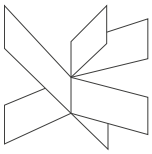


Table of content

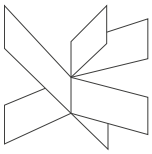


1 User stories and requirements

1.1 User stories

In this subchapter the user stories from the customer will be presented since the requirements for this report will be made based on them.

1. As an admin, I want to add employees to the system so that all the employees will have access to the system.
2. As an admin, I want to delete employees from the system so that all employees can no longer access the system.
3. As an admin, I want to edit employee's information in the system, so that all employee's data can be modified.
4. As an admin, I want to be able to create teams and assign employees and tasks to them.
5. As an admin, I want to be able to assign tasks to employees, so that employees can view their backlog.
6. As an admin, I want to be able to remove tasks from employee's backlog.
7. As an admin, I want to be able to view employee's data, so that I have access to pertinent information regarding employees.
8. As a user, I want to be able to view my backlog so that the schedule can be adhered to.
9. As a user I want to be able to view a burndown chart of the current project.
10. As a user, I want to able to denote my progress on tasks, so that the burndown chart is accurate.
11. As a user, I want to be able to check my work-related statistics, so that I can calculate my income.



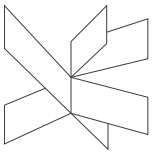
1.2 Non-functional requirements

1. The System must follow three tier architecture
2. The system must be implemented in Java and C#.
3. The usability of the system must be tested by end-users.
4. The system must store information in a database.

2 Analysis

2.1 use case diagram

Figure 1use case diagram

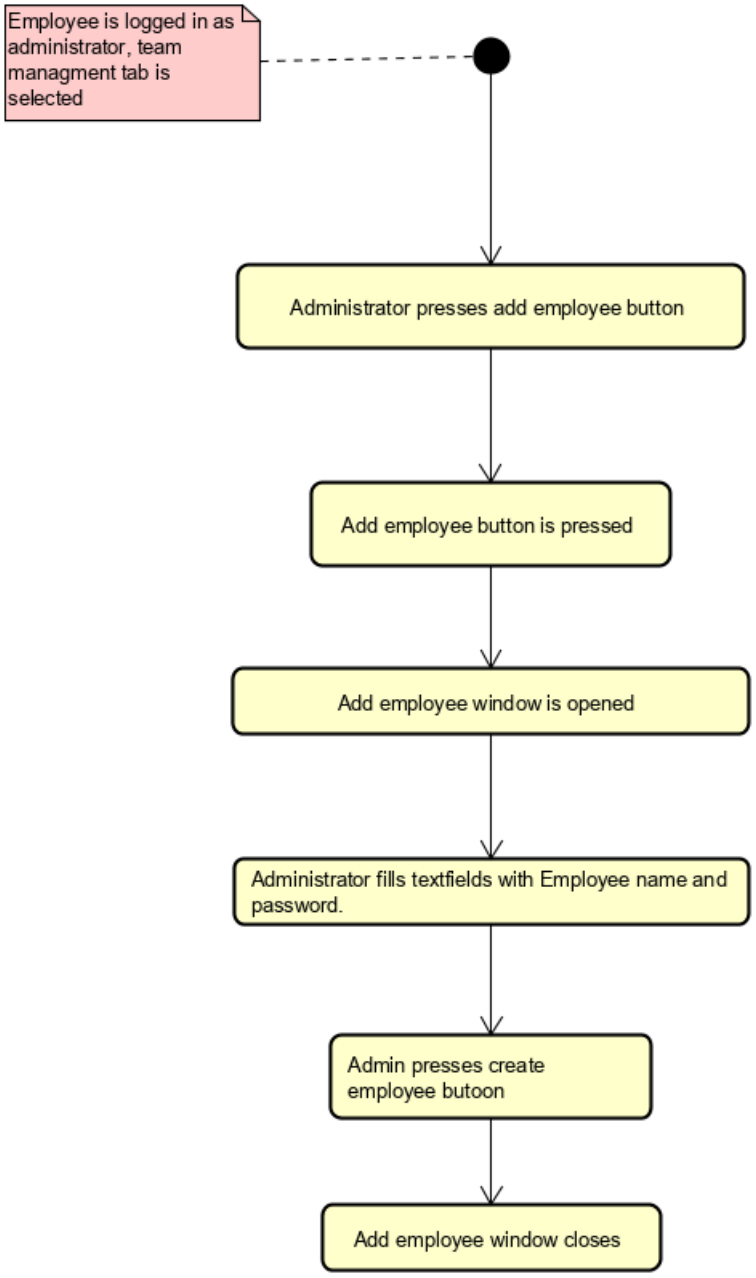


The use cases methodology has been used to classify and organize the system requirements. As it is shown in the diagram there are two different actors each has it is own set of actions that needs to be performed as a part of their daily activities.

2.2 Activity diagram

The activity diagram was created during the inception face to get a clear idea of the certain actions that need to be executed by the actor to reach his goal.

Figure 2Add Employee activity diagram



One of the Admin task is to add an Employee the activity diagram shows the steps needs to be taken to save the employee in the database.

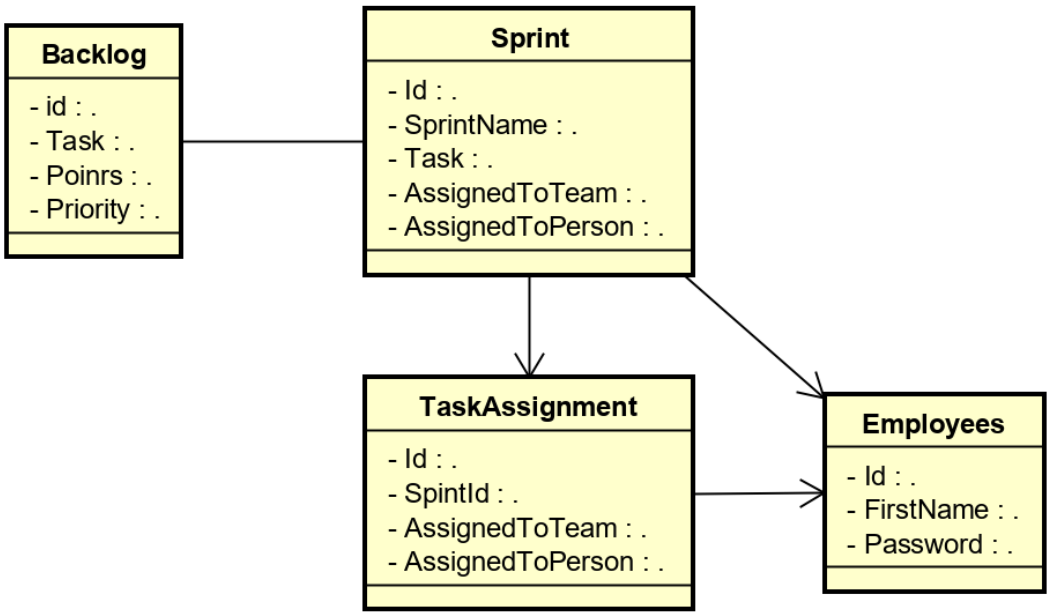
2.3 Conceptual diagram

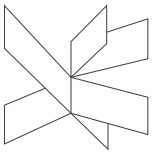
Conceptual Diagram demonstrates how all of the classes in their packages interact with each other. This is important since it provides a graphical representation on how the system will be created and how each component will interact with another to make it functional.

See apendix 1 for conceptual diagram

Figure 3conceptual diagram

2.4 Database





3 Design

3.1 Class diagram

See appendix 1 for class diagram

3.1.1 MVC Design pattern

The MVC design pattern is the most prevalent pattern throughout this project as it is how the GUI and its many windows are structured. This structure consists of the model, the view (actual interface the user sees) and the controller which dictates how the model and the view interact.

3.2 GUI

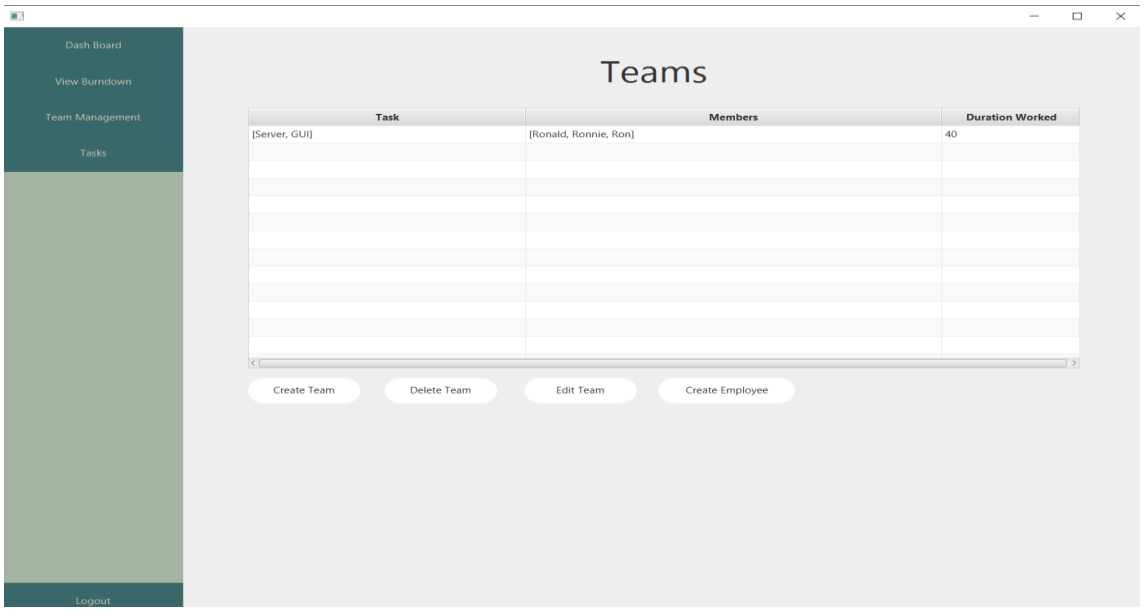
This section will show a portion of the GUI's design as it is the sole medium through which the user interacts with the software.

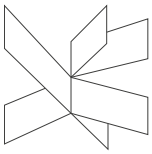
The figure below depicts the Team managment window as it is an apt example:

The FXML Code:

```
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>
<?import javafx.scene.text.*?>

<AnchorPane prefHeight="154.0" prefWidth="591.0" style="-fx-background-color: #EEEEEE;" xmlns="http://javafx.com/javafx/10.0.2-internal" xmlns:fx="htt
   <children>
      <Label layoutX="14.0" layoutY="14.0" text="Create Employee">
         <font>
            <Font size="28.0" />
         </font>
      </Label>
      <TextField fx:id="nameInput" layoutX="117.0" layoutY="63.0" prefHeight="26.0" prefWidth="206.0" style="-fx-background-radius: 30;" />
      <Button fx:id="createEmployeeButton" layoutX="488.0" layoutY="96.0" mnemonicParsing="false" onAction="#createEmployeeBtnPressed" prefHeight="42.
      <TextField fx:id="passwordInput" layoutX="117.0" layoutY="104.0" prefHeight="26.0" prefWidth="206.0" style="-fx-background-radius: 30;" />
      <Label layoutX="27.0" layoutY="65.0" text="Name">
         <font>
            <Font size="19.0" />
         </font>
      </Label>
      <Label layoutX="11.0" layoutY="105.0" text="Password">
         <font>
            <Font size="19.0" />
         </font>
      </Label>
   </children>
</AnchorPane>
```

The view the user sees

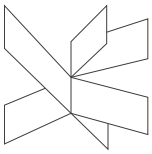




This window was chosen as it encompasses a multitude of actions that can be taken with the system and as such contains a reasonable amount of complexity.

Upon selecting an item from the list which is populated with current employees in the database, various actions can be performed on them through the use of the buttons below the table. This tab is only visible to admins, this is done during the login as two different main windows exist; one for employees and one for users.

into the actual REST server. When data is obtained on the REST server side, depending on the method called from GUI client side



4 Implementation

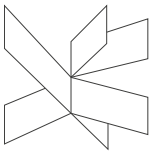
4.1 GUI Implementation

The figure below outlines one of the more interesting examples of the code with regards to the GUI as it deals with complicated tableviews and client/database requests in the controller.
The beauty of the MVC design pattern is exemplified within this example as it links the complex and fragile nature of tableViews with the dynamic nature of the database and client.

The createEmployee controller adheres to the SOLID design principle of single responsibility as its sole purpose is to be the bridge between the view and the client.

The controller's purpose is to obtain information from the fields in GUI and pass it to the Rest client that passes it further eventually getting to the database. Manage employee controller:

```
package controller.user.team;

import ...

public class createEmployeeController {
    public Client client;
    public createEmployeeView view;

    public createEmployeeController(createEmployeeView view, Client client) {
        this.view = view;
        this.client = client;
    }


    public void createEmployee() throws IOException {
        client.PostEmployee(view.getName(), view.getPassword());
        view.closeWindow();
    }
}
```

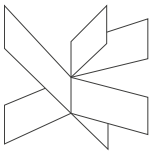


4.2 Three tier architecture implementation

4.3 Application layer

Application layer consists of two servers. First server is inside WebApp folder and is made in RESTful framework. It's responsible for passing and processing data acquired from GUI and passing it forwards into WebSocket server located inside SocketServer folder. SocketServer folder communicates with database and sends data back to RESTful server.

The GUI is calling methods inside Client class. Client class is then responsible for creating a HashMap from the arguments and sends them through REST commands



4.4 Data layer

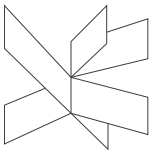
In order to receive and input information into database Socket server was created. It receives data from REST server. In order to pass the data from REST server to socket server JSON serialization and deserialization is used. JsonHandler folder inside SocketClient folder on RestServer side is responsible for serializing the data into strings. This is an example of Employee request being created:

```
1 odwołanie
public String Add(string name, string password)
{

    EmployeeEntity employee = new EmployeeEntity();
    employee.SetValues(name, password);
    JsonPackage package = new JsonPackage();
    EmployeeRequest emp = new EmployeeRequest();
    emp.Employee = employee;
    package.ForwardTo = "EmployeeController";
    package.Type = "AddRequest";
    package.Content = emp;
    String jsonpackage = JsonConvert.SerializeObject(package);

    return jsonpackage;


}
```

Socket server determines what to do with received data from ForwardTo and Type fields

5 Testing

In order to test the software two methods of testing had been used - scenario and white-box testing. White-box testing was achieved with the use of JUnit test framework. With JUnit testing most of the methods inside Client class. Special user was created

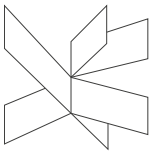


inside the database in order to allow database related methods to run the tests through it.

In addition to JUnit tests scenario testing was also used, where GUI part was used to test various user activity.

6 Conclusion

The project “SCRUM management system” was not completed as it’s missing most of its core features. The system performs a number of basic operations but due to lacks of understanding of Sockets and REST finishing the project would take it past the deadline. At the end of the time the group managed to understand all necessary components and completed the architecture with successful connection back and forth but it was to late to finish all the features.
In conclusion, we did our best and we tried as much as it is possible to deliver a good product that can be possible for future development.



7 References

Ken Schwaber, Jeff Sutherland, 2011; The Scrum Guide, [Last accessed 10/04/2018] via link: https://studienet.via.dk/Class/IT-SWE1XS18/Session%20Material/Scrum_Guide.pdf

Project report, 2017 (Appendix 3) VIA Engineering Guidelines [Last accessed 10/04/2018] via link: https://studienet.via.dk/projects/Engineering__project_methodology/General/Guidelines/2017%20Project%20Report%20(Appendix%203)%20%20VIA%20Engineering%20Guidelines.pdf Project description, 2017 (Appendix 1) VIA Engineering Guidelines [Last accessed 27/02/2018] via link: https://studienet.via.dk/projects/Engineering__project_methodology/General/Guidelines/2017%20Project%20Description%20(Appendix%201)%20%20VIA%20Engineering%20Guidelines.pdf

Process report,2017 (Appendix 2) VIA Engineering Guidelines, [Last accessed 05/04/2018] via link: https://studienet.via.dk/projects/Engineering__project_methodology/General/Guidelines/2017%20Process%20Report%20(Appendix%202)%20%20VIA%20Engineering%20Guidelines.pdf

Testing
https://www.vogella.com/tutorials/JUnit/article.html

Appendices

Appendix 1 class diagram

Appendix 2 entity relationship diagram

Appendix 3 Project description