Computer Science (AP) at Business Academy Southwest

Attendance Automation 2

Compulsory Assignment 2

April 17, 2020

Radoslav Backovsky
Louise Lauenborg
Anne Luong
Nadezhda Antoanova Miteva
Martin Emil Rune Wøbbe

Preface

The main objective of any computer science student is to get as much practical knowledge as possible. Being able to have practical knowledge by developing a program is as important as theoretical knowledge. Through the development of the project we have gotten great experience in various strategies that can be applied in future endeavors.

We are pleased to present this project and thankful for the opportunity to gain this experience.

Table of Contents

3 Introduction

3.1 Background

In education there are many management tools for students, teachers and school employees. The main goal for these tools is to make things done in a simple and effective way. Teachers should have a good overview of student's attendance in order to evaluate their work. In everyday life people have different problems they must solve. Some problems will cause that student is not able to be part of educational events. Attendance Automation System is a management tool for educational institutions to track attendance of students and help the most absent individuals if needed.

3.2 Problem statement

EASV student registration must be recorded every day for every course subject. The process should be convenient, time efficient and non-disruptive.

Currently, the students mark their own attendance with the inbuilt solution on Moodle. This solution is decent but has room for improvement. Students find it troublesome and time-consuming as it is hidden behind many clicks. Attendance registration is also done for the entire day and not for each course subject. Hence, leading to an incorrect representation of a student's actual attendance.

In this project, the goal is to develop a working program which solves the problems described. The program will also use a decentralized system and strive to improve the shortcomings of the currently used solution. The focus is ease of use and more features. During the process, SCRUM should be used for the project management and GitHub for version control. Common software design patterns, a layered architecture and tests should be incorporated in the software.

3.3 Product vision

The vision for the Product

Attendance Automation is a decentralized management system that provides easy and convenient attendance registration and supervision. The product stakeholders of this product are primarily institutions like schools and academies. It's made for users as the teachers and students of EASV, who need to track and register attendance in any given course. Unlike the current inbuilt solution in Moodle, Attendance Automation is simpler to use with more features. A teacher and a student have a better overview of course attendance and registration of students as a student registering is easily accessed.

Value proposition

EASV needs a student attendance registration system. Attendance Automation is a far more efficient way to register attendance together with getting an overall overview. This will benefit teachers and students in more access to more attendance information and a safer way to register attendance without cheating.

Epic stories

After Student login – Today Page (Figure 1):

- As a student you should be able to register your attendance for a specific lesson.
- As a student you should be able to see the name and time of the subject of the current day.
- As a student you should be able to see the current date.
- As a student you should be able to see your name.

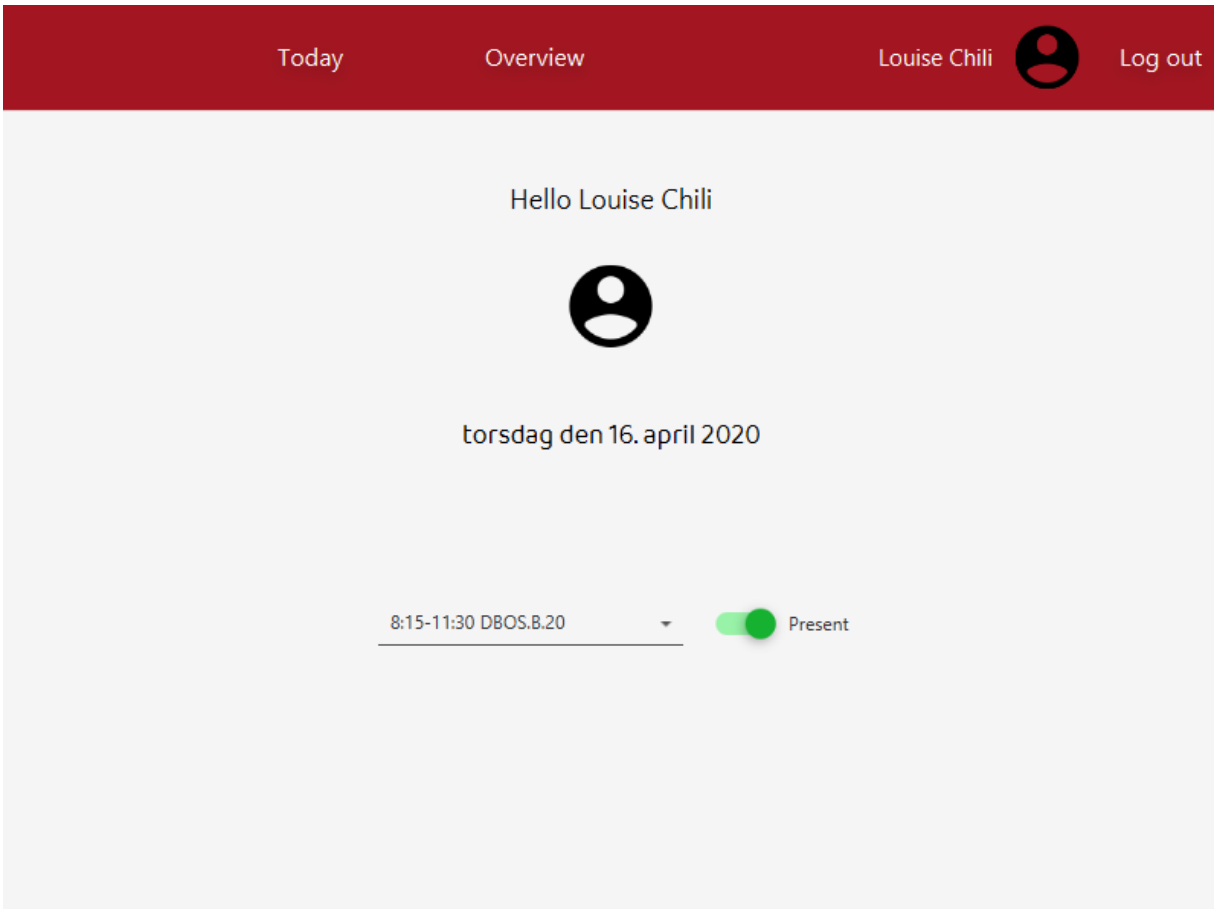


Figure 1: Today page for the student.

After Teacher login – Overview page (Figure 2):

- As a teacher you should be able to see summarized attendance for a specific course, where most absent students are at the top.
- As a teacher you should be able to see summarized attendance on each student of a given course.
- As a teacher, I should be able to see the total number of students present in the class in order to limit cheating.
- As a teacher, I should be able to select the course to be shown on the Dashboard.

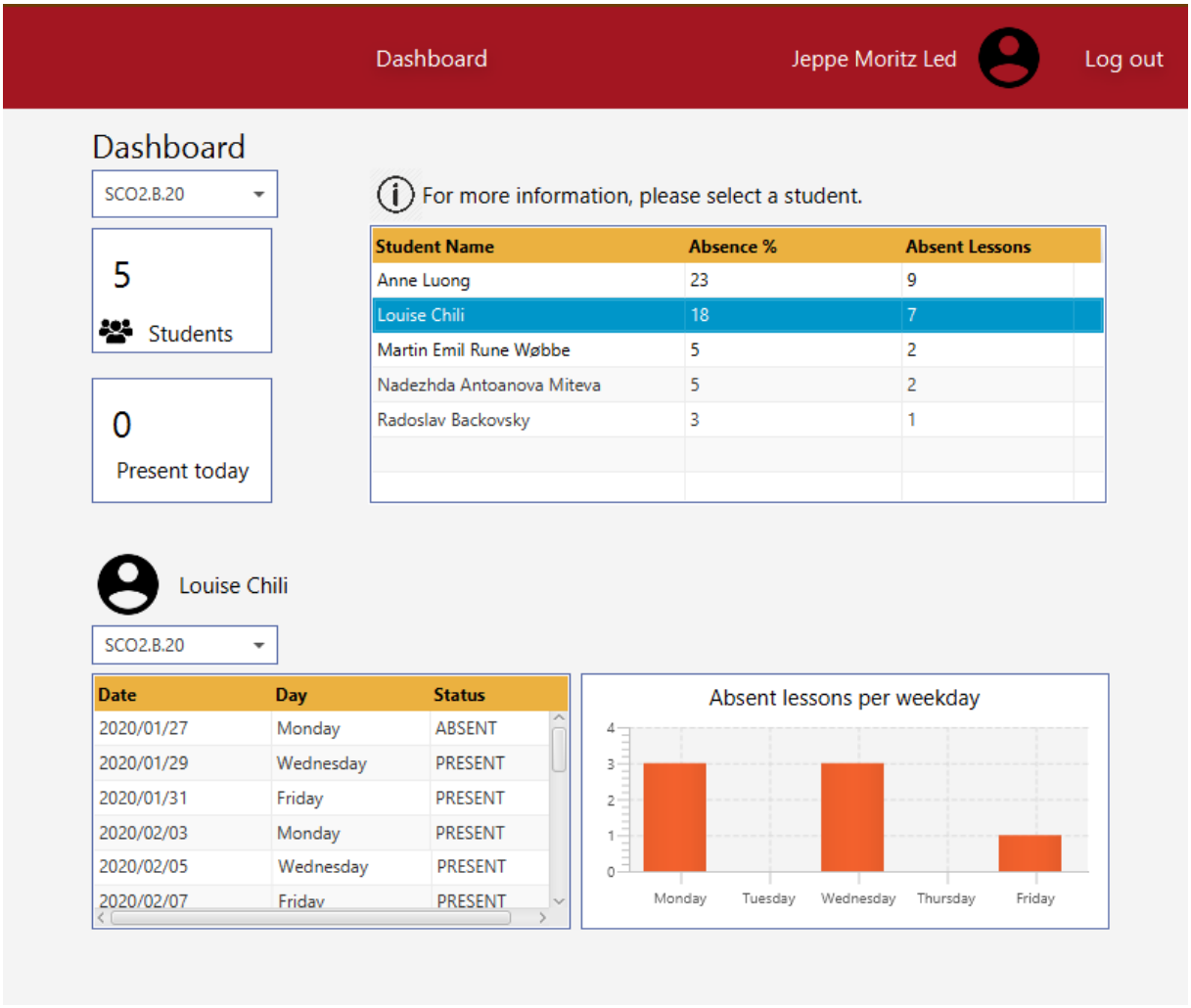


Figure 2: Dashboard page for the teacher.

Business objectives

Attendance Automation will provide the targeted customer, the company, with a beneficial way to register and overview attendance of the students attending the institution. It will be easier as a teacher to follow a student's absence behavior therefore it will be possible to help a given student succeed the study before a student is so far behind with studies that they give up or fail and aren't graduating. This will also benefit the teachers and students' options to closely follow their attendance with a better overview of different courses.

3.4 Strategic analysis

Stakeholders

Company buying software, academy or other institutions and users.

Figure 3: Power-interest grid [1].

Stakeholder analysis

High power, high interest:

The most important, highly prioritized stakeholder is the company buying this product for a bigger concern like an academy. They'd like to collaborate and keep an eye on the progress of the project.

They want to be fully engaged. The team must have close communication with the product owner.

High power, low interest:

The actual academy or educational institution which awaits this registration system. They don't need or want lots of details, but they should have influence over the project and need to be kept satisfied without too much communication. These are motivated by the opportunity of making more graduates and in making the registration and overview less time-consuming to save money.

Low power, high interest:

Lower prioritized is the users. As teachers and students can have high interest as users of the product, but they have very little power to change anything. These don't need to be heard but can come with valuable inspiration along the way. They are motivated by the idea of easy access for this specific information.

Low power, low interest:

It seems there is no low power, low interest in this power-interest grid for this product.

4 Pre-Game (Sprint 0)

4.1 Project organization

SCRUM will be used as the management framework. The roles in the team are shown in Table 1.

Role	Name
Product Owner	Martin Emil Rune Wøbbe
Scrum Master	Anne Luong
Scrum Team (Development Team)	Radoslav Backovsky Louise Lauenborg Anne Luong Nadezhda Antoanova Miteva Martin Emil Rune Wøbbe

Table 1: Overview of the SCRUM roles in the team.

All SCRUM principles and practices will be followed except some of the Product Owner's and Scrum Master's responsibilities which will be shared by the Scrum Team. As the entire team is inexperienced at SCRUM, it is necessary for everyone to participate in creating the Product Backlog (epic stories and user stories) and selecting Product Backlog items for the Sprint Backlog. The tasks for each user story will also be created together followed by planning poker for time estimation. There is no task delegation. The team members will choose tasks according to their own wishes. Each member should update the Task board daily regarding time spent and assigning new tasks. The Task board should constantly be modified according to the state of the project. The Task board will be checked every Daily Scrum meeting together with the Burndown chart. Hence, the only responsibilities of the SCRUM master will be to facilitate the Daily Scrum and ensure SCRUM practices are followed.

Various software will be used during the project (see Table 2) to support collaboration. The main communication channel will be Discord, while meetings will be held on Zoom and TeamViewer. To facilitate SCRUM practices, the team will use Scrumwise. The software will be developed in NetBeans and the database will be created and managed with Microsoft SQL Server Management Studio. Git, the version control system, will be used to track changes of the code and aid collaboration.

Name	Type
Zoom	Video communications
TeamViewer	Video communications
Discord	Text, image, video and audio communications
NetBeans	Integrated development environment (IDE)
Git	Version control
GitHub (GitHub Desktop)	Git client
GitKraken	Git client
Microsoft SQL Server Management Studio	Database administrator tool
Scrumwise	SCRUM project management tool
Google Docs	Web-based word processing
Balsamiq Wireframes	Wireframing
Lucidchart	Diagramming and visualization
Visual Paradigm	UML diagramming
Draw.io (Diagrams.net)	Diagramming

Table 2: Software used in the project.

Finally, in order to ensure a smooth collaboration, the team has made a working agreement (Appendix Z).

4.2 Overall project schedule

Table 3 shows the overall schedule for the whole project.

Week	Date	Time	Activity
11	March 9, 2020	10:45 – 12:30	Kick-off meeting: 1) Project presentation 2) How to write a report
11	March 9, 2020 – March 13, 2020		Pre-Game (Sprint 0) planning Sprint 1 planning
12	March 13, 2020 – March 19, 2020		Sprint 1
12	March 20, 2020	10:00 – 10:15	Sprint 1 Review
12	March 22, 2020		Sprint 1 Retrospective
12 – 13	March 22, 2020 – March 23, 2020		Sprint 2 planning
13 – 14	March 24, 2020 – April 02, 2020		Sprint 2
14	April 03, 2020	10:00 – 10:15	Sprint 2 Review
14	April 03, 2020		Sprint 2 Retrospective
14	April 03, 2020		Sprint 3 planning
15	April 06, 2020 – April 08, 2020		Sprint 3
16	April 17, 2020	14:00	Hand-in the report

Table 3: The overall project schedule.

4.3 Initial product backlog

The initial Product Backlog, as shown in Figure 4, 5 and 6, has been created by the team. Epic stories and user stories are created for each requirement in the assignment.

Please note that Figure 4, 5, and 6 are screenshots taken from the end of Sprint 1. Except the progress, there is no difference from the initial Product Backlog.

Figure 4: Scrumwise Product Backlog first part.

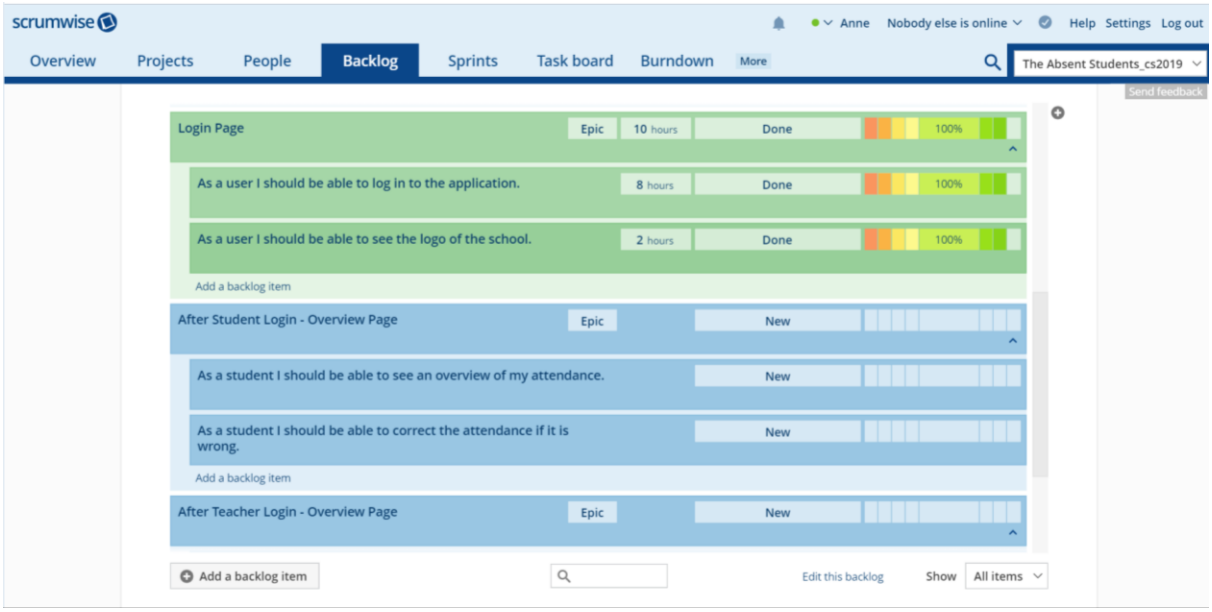


Figure 5: Scrumwise Product Backlog second part.

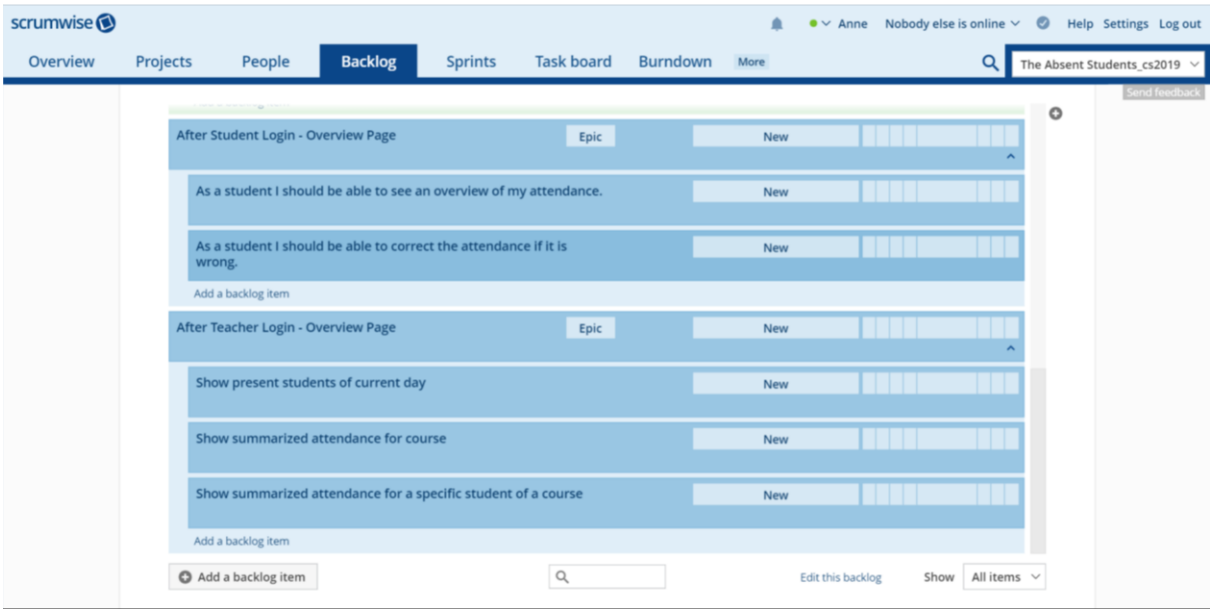


Figure 6: Scrumwise Product Backlog third part.

4.4 Architecture

The team has decided to implement 3-layered architecture with GUI, BLL and DAL layers. Every layer has access to applications business entities (BE). The main benefit from implementing this architecture is decoupling the layers which will ensure independent development of each layer.

In GUI layer the team decided to implement MVC architectural pattern (Figure 7) which is handling communication and separating responsibilities between Models, Views and Controllers. Visuals are created with help of SceneBuilder and functionality is handled with JavaFX.

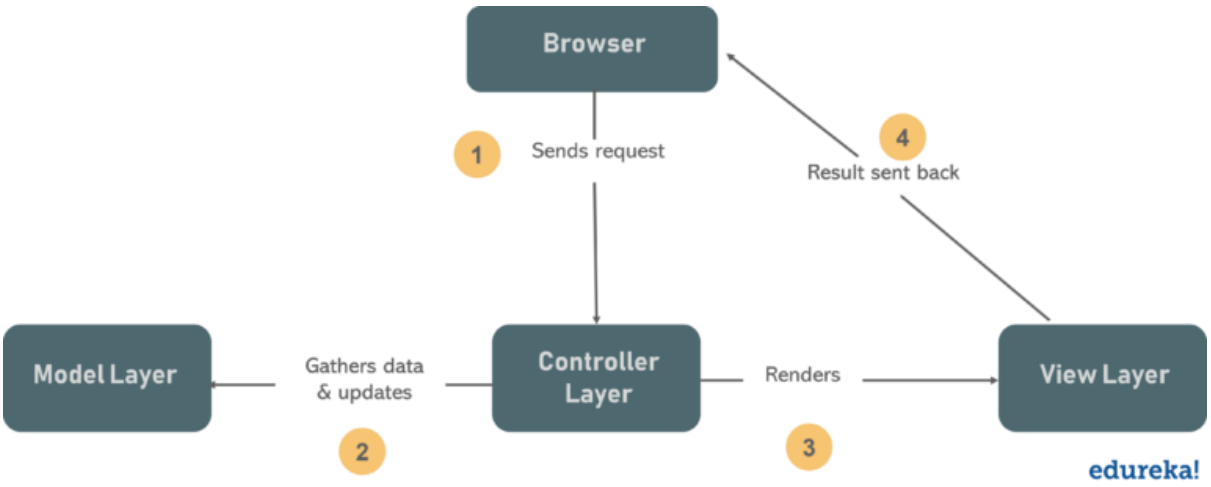


Figure 7: MVC architectural pattern's showcase of communication and responsibilities. [2]

MVC pattern could be improved by creating a facade for Models, because many controllers are using many models. That means controllers have many dependencies. Later the facade could handle what functionality of what model should be passed to the controller.

In the initial architecture the team planned to implement Facade pattern for BLL and DAL layers and Singleton design patterns for UserModel.

4.5 Preliminary usability test

The first prototype presented for usability tests can be seen in Appendix 1.

The first test group liked the user experience but questioned the usage of the column chart in the monthly overview. Displaying all days of the month in the column chart was overwhelming and not very useful. The group also wanted to get a confirmation message after attendance registration.

The second test group noted that one doughnut chart and one column chart for each overview was excessive. The charts were not intuitive and hard to understand. The group also wanted the data represented in a table instead of graphically with charts.

The usability tests showed that the user experience could be improved. Specifically, the overview was not as simple and intuitive as hoped and the addition of a confirmation message. In order to solve these issues, the user interface was changed to use a dashboard design pattern [3]. The decision was made after looking at different design patterns on ui-patterns.com.

The dashboard design pattern was found to be a better solution as it provides a high-level, visual overview of data for the user while simplifying the user experience (compared to the previous prototype) [3]. The previous prototype could not simultaneously provide daily, weekly and monthly overview. Each type of overview was accessed by the user's choice. After careful consideration, it was decided that giving the user quick access to daily, weekly and monthly overview at once glance would be a better approach. The new prototype with dashboard design pattern was directly made in Balsamiq.

5 Sprint 1

5.1 Sprint planning

The user stories in the Product Backlog were ranked and the stories which contained the core functionalities were picked to be a part of the Sprint Backlog. The allotted Sprint time was 50 hours but was mistakenly not filled out. Only 26 hours of work was a part of the Sprint at the beginning.

The Sprint Backlog can be seen in Figure 11. By mistake, no screenshots were taken at the beginning of the Sprint, so Figure 11 is the Sprint Backlog at the end of Sprint 1. The shown ranking of the user stories does not reflect the initial priority. During the Sprint, the priority was changed. The initial priority is unknown as the team did not document it anywhere and cannot remember.

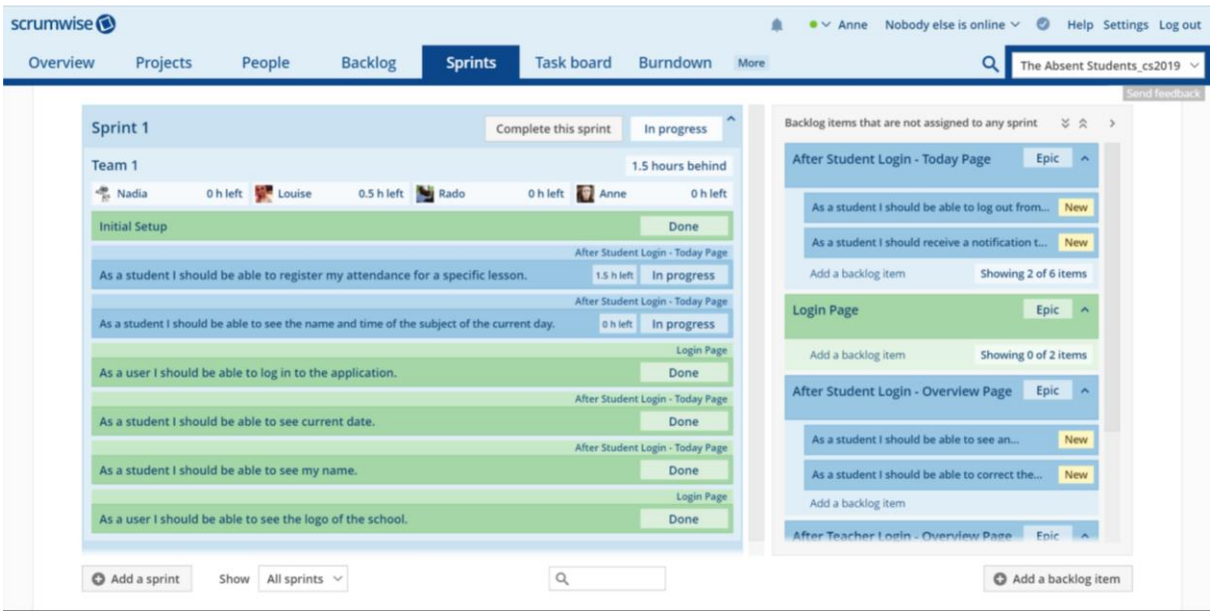


Figure 11: Scrumwise Sprint 1.

5.2 GUI (including UI-design patterns)

In the Attendance Automation System, the team has used several UI design patterns in order to make the user experience much more pleasant and less time consuming for the user.

One of the design patterns the team used was the *Input Prompt* design pattern in the login page (Figure 12). This design pattern makes it easier for the user to understand what type of data are required and where to put them. This design method also helps the program's simplicity design, by putting a prompt text instead of an extra label. [4]

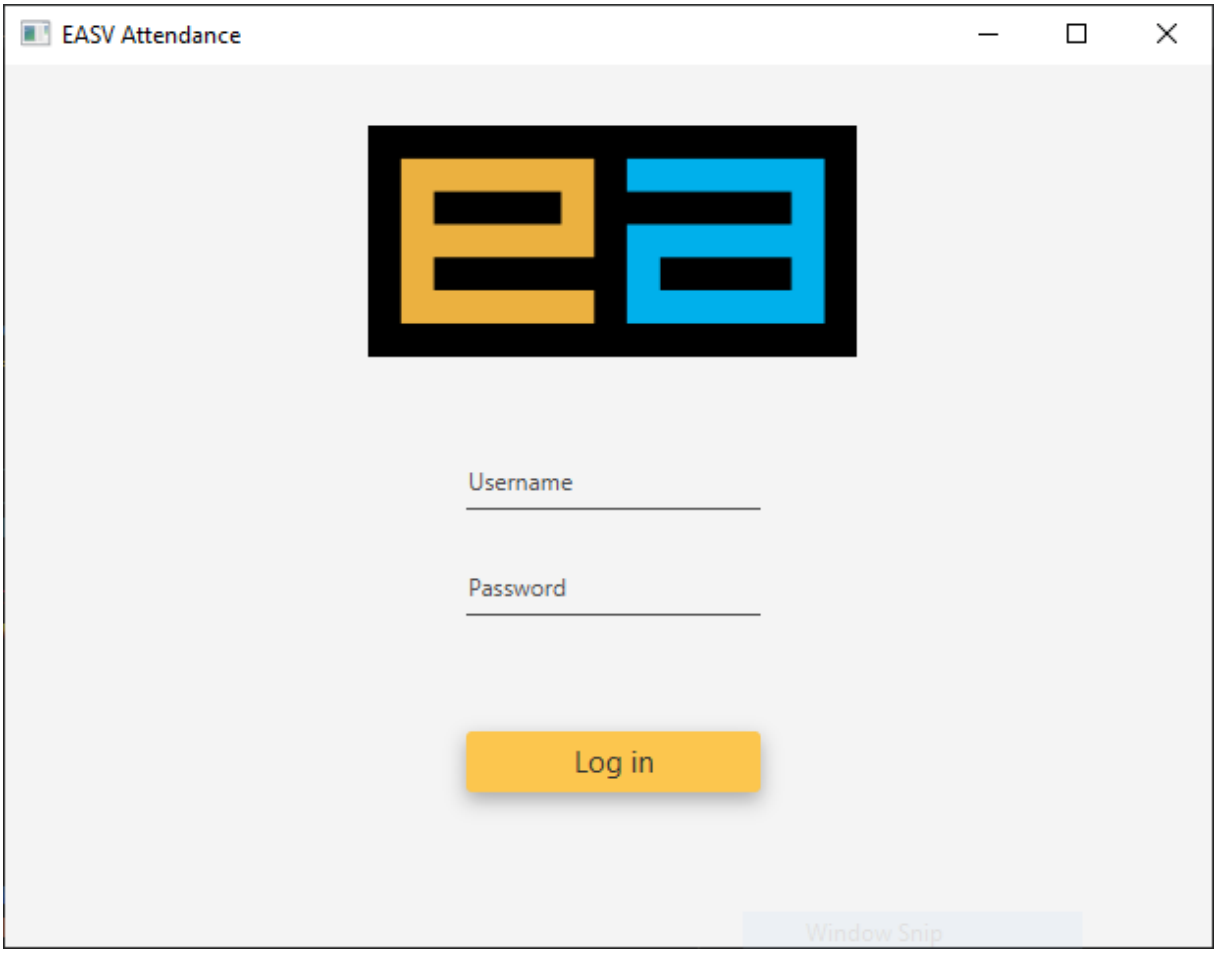


Figure 12: Input Prompt design pattern: Login page.

The second design pattern the team has used was the *Keyboard Shortcut*. The team expected the user to use a keyboard shortcut more often than clicking the button for logging in since it is widely used for convenience. [5]

Another design pattern the team used was the *Module Tabs*. Because some of the content needed to be separated into sections which has a lot of benefits one of which is that the program won't need to open a new scene each time thus making it more pleasant and less time consuming for the user as well as the functionality of the program. [6]

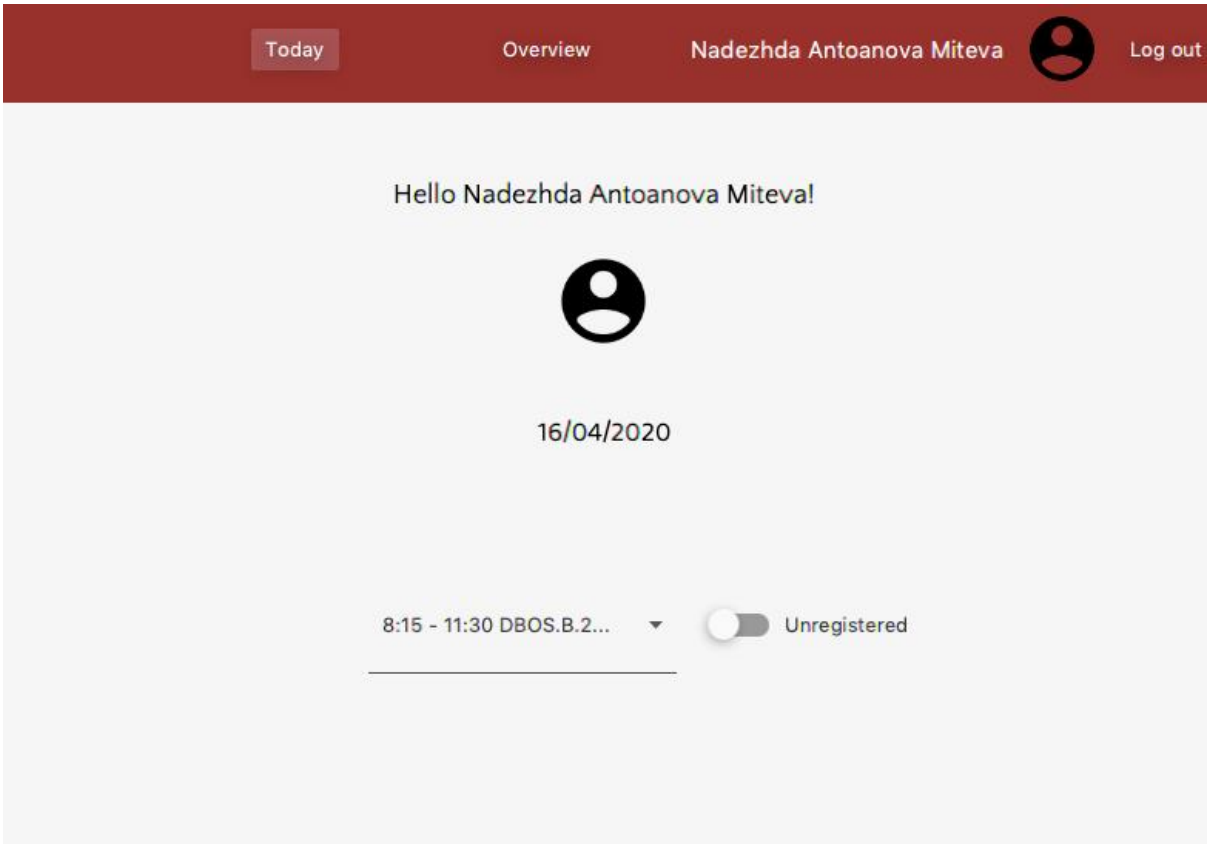


Figure 13: Module Tabs design pattern: Student Today page .

5.3 Data model

For this project the team created a database using Windows SQL Management Studio. In this database there is six relations. All the relations contain a primary key id, which is used throughout. Normalization is utilized by having several smaller, relational tables, reducing redundancy and dependency of the data. These are the relations:

- User
- UserType
- Course
- UserCourse
- CourseCalender
- AttendanceRecord

User and UserType:

User holds the primary information regarding any user in the program. It holds an id of the user, a username, a password, a name (full name of the user) and a userTypeId. The userTypeId is a char which can be used to determine what type (Teacher or Student) the user is. The name of the UserTypes are stored in the UserType relation and can easily be found through the relation between the tables using the char id.

Course, UserCourse, CourseCalender and AttendanceRecord:

Course is in itself a simple relation in the database, only consisting of a course id and a name (e.g. SCO2, SDE2 etc.). The use of the Course relation comes in its relations with UserCourse and CourseCalender. The UserCourse relation uses a userId and a courseId to store information about which users, be it teacher or student, are connected to certain courses.

The CourseCalender relation is used to store which lessons of a certain Course are on which days and at which time. Here both the startTime and endTime attributes are of the smalldatetime datatype, storing both a date and a time.

AttendanceRecord:

The AttendanceRecord is used to store what the users submitted status are for each lesson in the CourseCalender, this being either Absent or Present. This is stored as a nchar, with the time of which the status was recorded (datetime datatype) and stores it with the userId of the user in question and the courseCalenderId from the specific lesson.

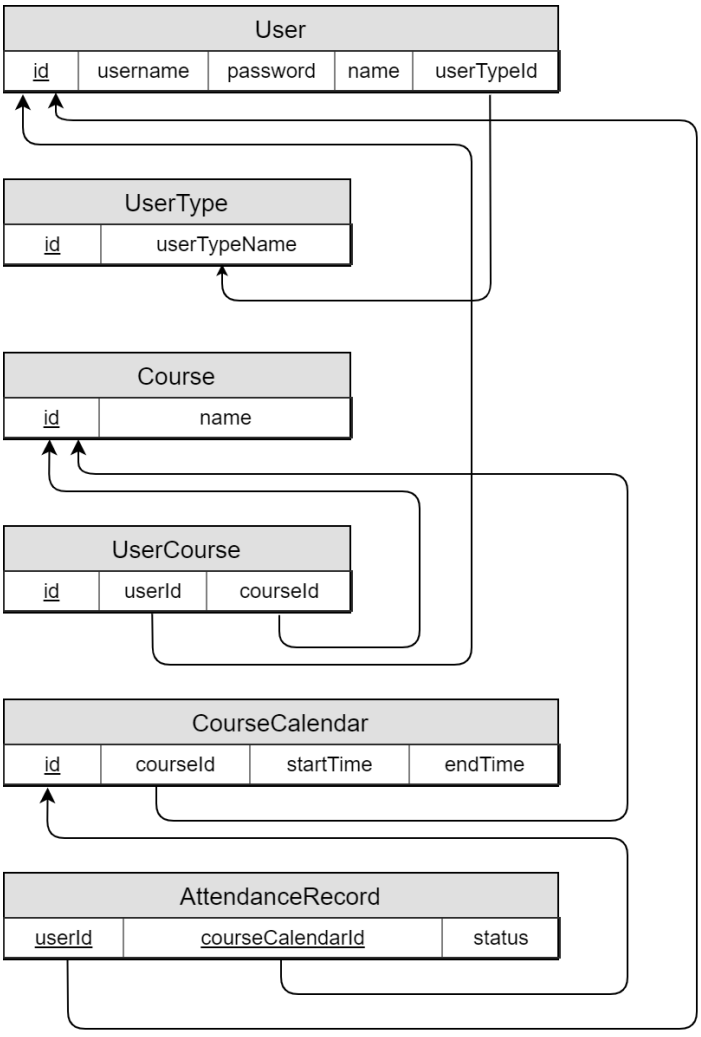


Figure 14: Relational database schema.

5.4 Implementation

As a starting point the team created a project repository on GitHub. Important thing was to update the .gitignore file. This update prevented pushing irrelevant changes in properties files while working on the project. After the creation of the project, the team started with implementation of 3-layer architecture and MVC, creating packages, interfaces and classes. After creating main packages, the team started with implementation of Facade pattern in BLL and DAL layers and Singleton pattern for UserModel.

5.4.1 Code examples

User is set after login. If the logged in user is a student, a student client will launch. If the user is a teacher, then the teacher's client will launch after selecting the initial course.

```
107     private void authentication() throws Exception {
108         try {
109             this.user = userModel.login(txtUsername.getText(), pwPassword.getText());
110         } catch (ModelException ex) {
111             showAlert(ex);
112         }
113         if (user != null) {
114             if (user.getType() == User.UserType.TEACHER) {
115                 showRoot(SUBJECT_CHOOSER);
116                 closeLogin();
117             } else if (user.getType() == User.UserType.STUDENT) {
118                 showRoot(ROOT_STUDENT);
119                 closeLogin();
120             }
121         }
122     }
```

Figure 15: Authentication of user.

This method could be optimized after implementing a facade pattern for models in future.

```
71      public void showModule(String MODULE) {
72          try {
73              FXMLLoader fxmlLoader = new FXMLLoader(getClass().getResource(MODULE));
74              Parent moduleRoot = fxmlLoader.load();
75
76              if (MODULE.equals(TODAY_MODULE)) {
77                  todayController = fxmlLoader.getController();
78                  todayController.setUser(user);
79                  todayController.injectModel(lessonModel);
80                  todayController.initializeTodayModule();
81              }
82              if (MODULE.equals(OVERVIEW_MODULE)) {
83                  StudentOverviewController controller = fxmlLoader.getController();
84                  controller.setUser(user);
85                  controller.injectModels(courseModel, recordModel);
86                  controller.initializeOverviewModule();
87              }
88              borderPane.setCenter(moduleRoot);
89          } catch (IOException ex) {
90              Logger.getLogger(RootStudentController.class.getName()).log(Level.SEVERE, null, ex);
91          }
92      }
```

Figure 16: Showing module after clicking on button in Student's client.

5.4.2 Design Patterns/principles

A *Façade Pattern* was implemented by the Team in the program in both the Business Logic Layer and the Data Access Layer. This was firstly to decouple the layers, ensuring that instead of calling methods directly from the DAO classes, we would use the Façade and Interface classes made, also giving the opportunity to make changes to the Class Methods without impacting the Client Call. This also allowed to make more complex method calls from different DAO classes from only a single Façade Class.

Singleton design pattern was implemented for UserModel because the application needed the same instance of UserModel for every controller. Since the model has global access it is easy for the controller to get the instance every time it is needed. This way the current logged in user could be set so displaying of the user's name in different parts was possible.

5.5 Sprint Review

The Sprint Review for Sprint 1 occurred on March 20, 2020. In the beginning, the team demonstrated the done user stories to the stakeholders (teachers), who were pleased with the outcome in terms of functionality and visual design. As seen in Figure 17 and 18, only 5 user stories out of 7 user stories were done. The Sprint Goal was not met because of the challenging circumstances following the current novel coronavirus (2019-nCoV) outbreak and inexperienced planning, so the stakeholders were understanding of the current situation.

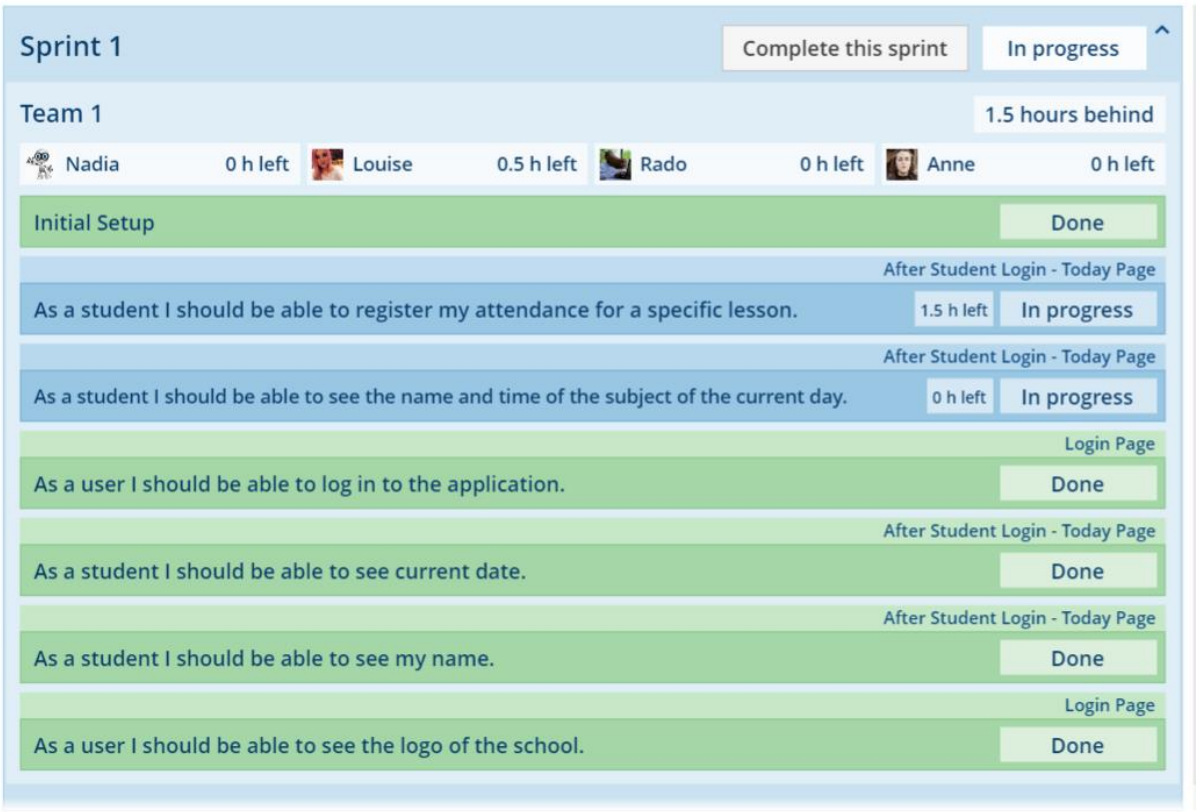


Figure 17: Scrumwise Sprint Backlog of Sprint 1.

Figure 18. Scrumwise user stories completed in Sprint 1.

In the second part of the meeting, the stakeholders discussed the SCRUM practices of the Sprint with the team. In general, the feedback was positive with praise of the Scrumwise usage. The epic stories were correct, and the user stories were well-formulated because of the format “As a [role] I should be able to…”. Inserting the time used for each task was also well-received. The Burndown chart showed that the group started to fall behind on work from March 17, 2020. The Scrum Master, Anne, was sick and could not contribute any work that day. As mentioned before, many user stories were not completed within the Sprint due to unskilled planning. The planning was based on wrong time estimates, so many tasks were harder and took longer than anticipated. The stakeholders also noticed that the Sprint hours were not fully assigned after looking at the Burndown chart in Figure 19 (26 out of 50 hours were assigned) and advised to fully allocate the next Sprint with tasks.

All the incomplete user stories were brought to the next Sprint.

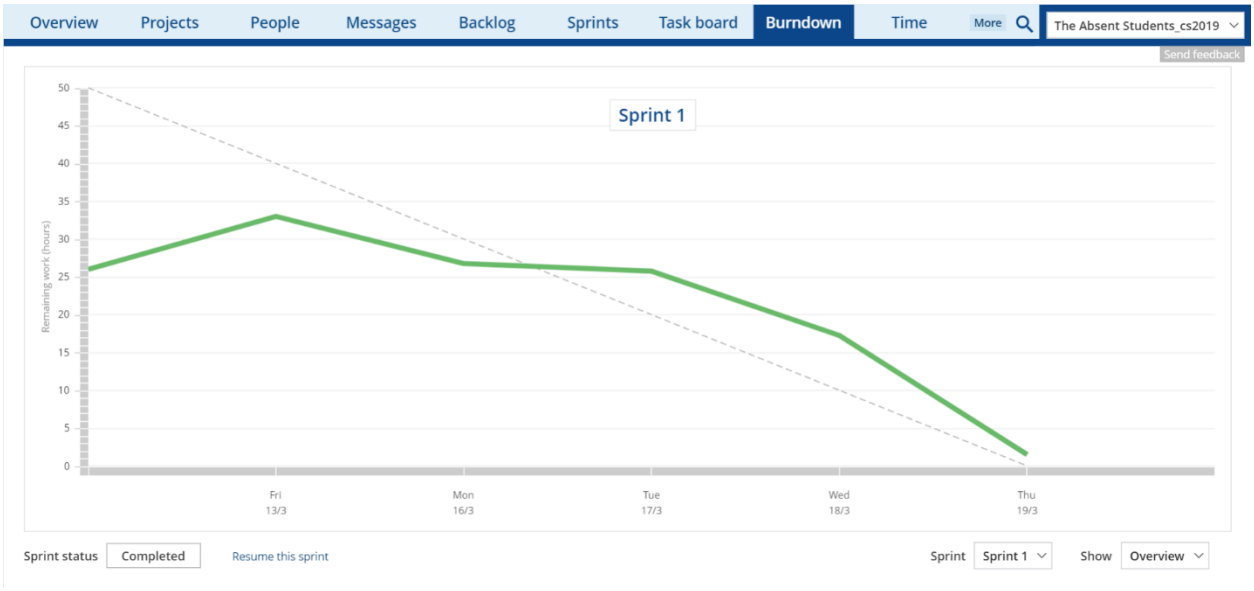


Figure 19: Scrumwise Burndown for Sprint 1.

5.6 Sprint Retrospective

For the Sprint Retrospective, the team used a template called "The 4 L's" by RealtimeBoard [7]. Each member of the team wrote down what they liked/learned/lacked/longed for in the Sprint and the team shared their reflections afterwards. Table 4 shows a summary of each category.

Liked	- SCRUM was a good planning tool. It helped coordination and division of tasks. The Task board gave a helpful overview. - Daily Scrum meetings were short and provided daily updates. - Team atmosphere was good, and everyone was understanding of each other.
Learned	- SCRUM practices in action. - The importance of planning and foresight. Many tasks were dependent on another task making the order of task completion crucial. Some tasks were also missing on the Task board. - Time estimation should be less optimistic. Many tasks required more time than anticipated and were also harder than expected. The estimated time should factor in this uncertainty. - The importance of asking for help. Several members realized their own shortcomings and lack of knowledge and skills but did not reach out for help.
Lacked	- Punctuality. Several members were not on time for the scheduled meetings and did not inform the team. - Communication. - Knowledge. All team members could feel their own shortcomings. - Foresight. Many tasks were more comprehensive than anticipated. - Inclination to ask for help and advice. Several members wished they had asked for help and advice earlier and not struggle so long by themselves.
Longed for	- Shorter Daily Scrum. In the beginning the meetings were too long as the team talked about their progress in too much detail. - A more detailed and well-planned Task board. The description was lacking and unclear for some team members. Some team members would like smaller tasks. - Coordination regarding task delegation. The team members picked their own task, which in retrospect was an unwise decision.

Table 4: Sprint Retrospective The 4 L's.

It was agreed that the team would continue doing the "liked" and "learned" observations. In order to improve and eliminate "lacked" and "longed for" findings, the team agreed on the following changes:

- Daily Scrum (maximum 15 minutes) at 7:00 PM as it would suit several team members better. Before it was in the afternoon after class.
- During the Daily Scrum, each team member should only answer the three Daily Scrum questions.
- If needed, a detailed meeting can be requested afterwards with the involved parties. All team members should be available for an hour after Daily Scrum in case they are needed.
- Always stay in contact and inform the team in advance if not attending scheduled meetings.
- Voice your honest opinion about any issues. Don't let the problems grow.
- Spend more time on thorough planning for the next Sprint.

6 Sprint 2

6.1 Sprint planning

The Sprint Backlog of Sprint 2 can be seen in Table 5. No screenshots were taken at the beginning of the Sprint, so there is no Scrumwise documentation. The assigned hours for the Sprint was 125 hours and the estimated hours for all the remaining user stories was 124 hours. The team expected to complete the Product Backlog in this Sprint.

User stories
As a student, I should be able to log out from the application.
As a student, I should be able to see the name and time of the subject of the current day.
As a student, I should be able to register my attendance for a specific lesson.
As a teacher, I should be able to select course after login.
As a teacher, I should be able to see summarized attendance for a specific course, where most absent students are at the top.
As a teacher, I should be able to select the course to be shown on the Dashboard.
As a student, I should be able to see an overview of my attendance.
As a teacher, I should be able to see the total number of students present in the class in order to limit cheating.
As a teacher, I should be able to see summarized attendance on each student of a given course.
As a student, I should be able to correct the attendance if it is wrong.
As a teacher, I should be able to correct the attendance for a student who requests his/her attendance to be corrected.
As a student, I should receive a notification to register my attendance.

Table 5: Sprint Backlog of Sprint 2.

6.2 GUI (including UI-design patterns)

For the Teacher user the team has decided to implement a *Dashboard design pattern*, more specifically an operational Dashboards pattern. The Dashboards main function is to display lots of data and nearly real time data as the data is refreshed and interactive making it easy and efficient for the user to have an overview of the data. [3]

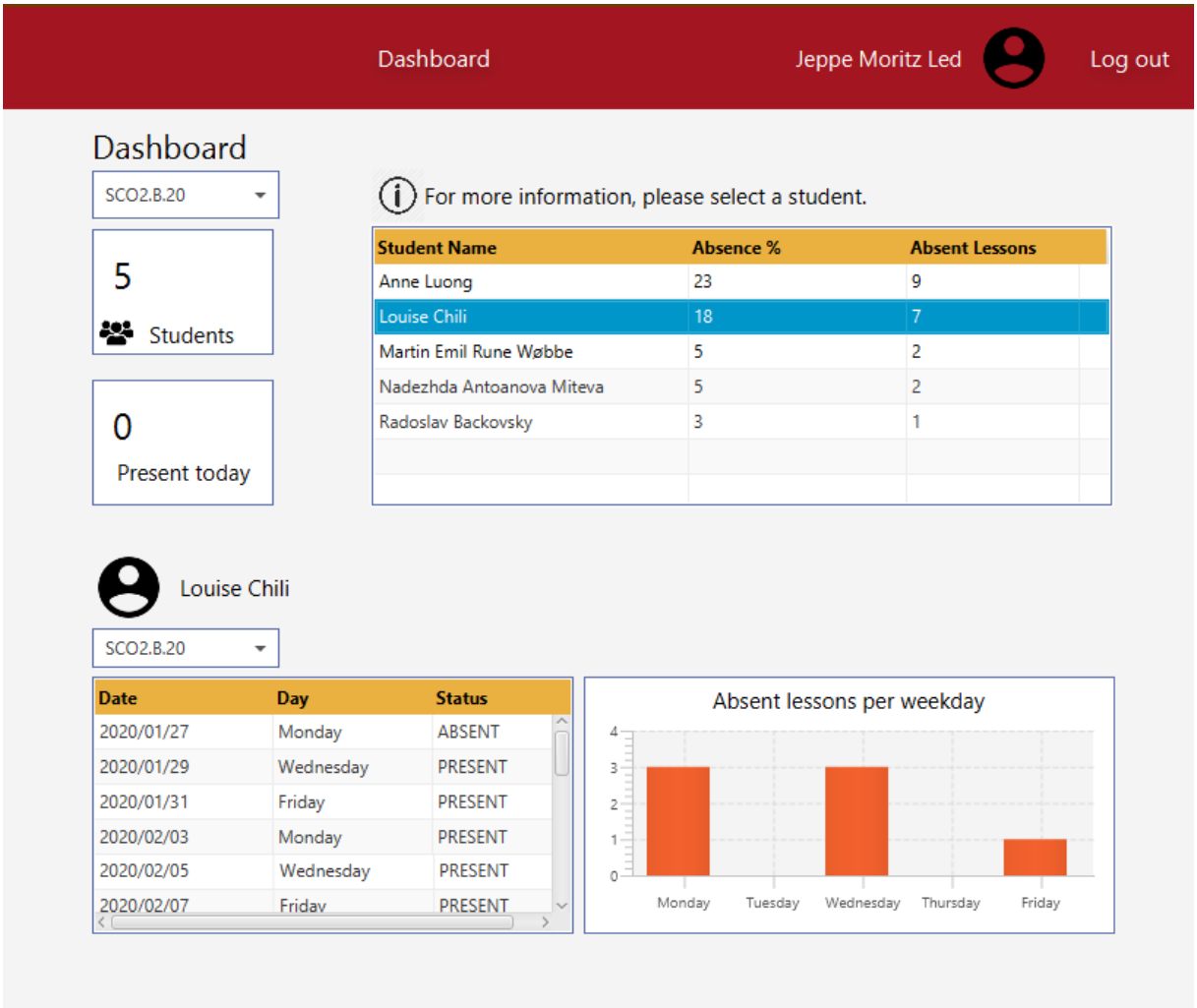


Figure 20. Dashboard design pattern: Teacher Dashboard.

6.3 Data model

A timeRecorded attribute with the datetime domain was added to the AttendanceRecord relation as it was needed for an Observer Pattern implementation. The new entity relation diagram is shown in Figure 21.

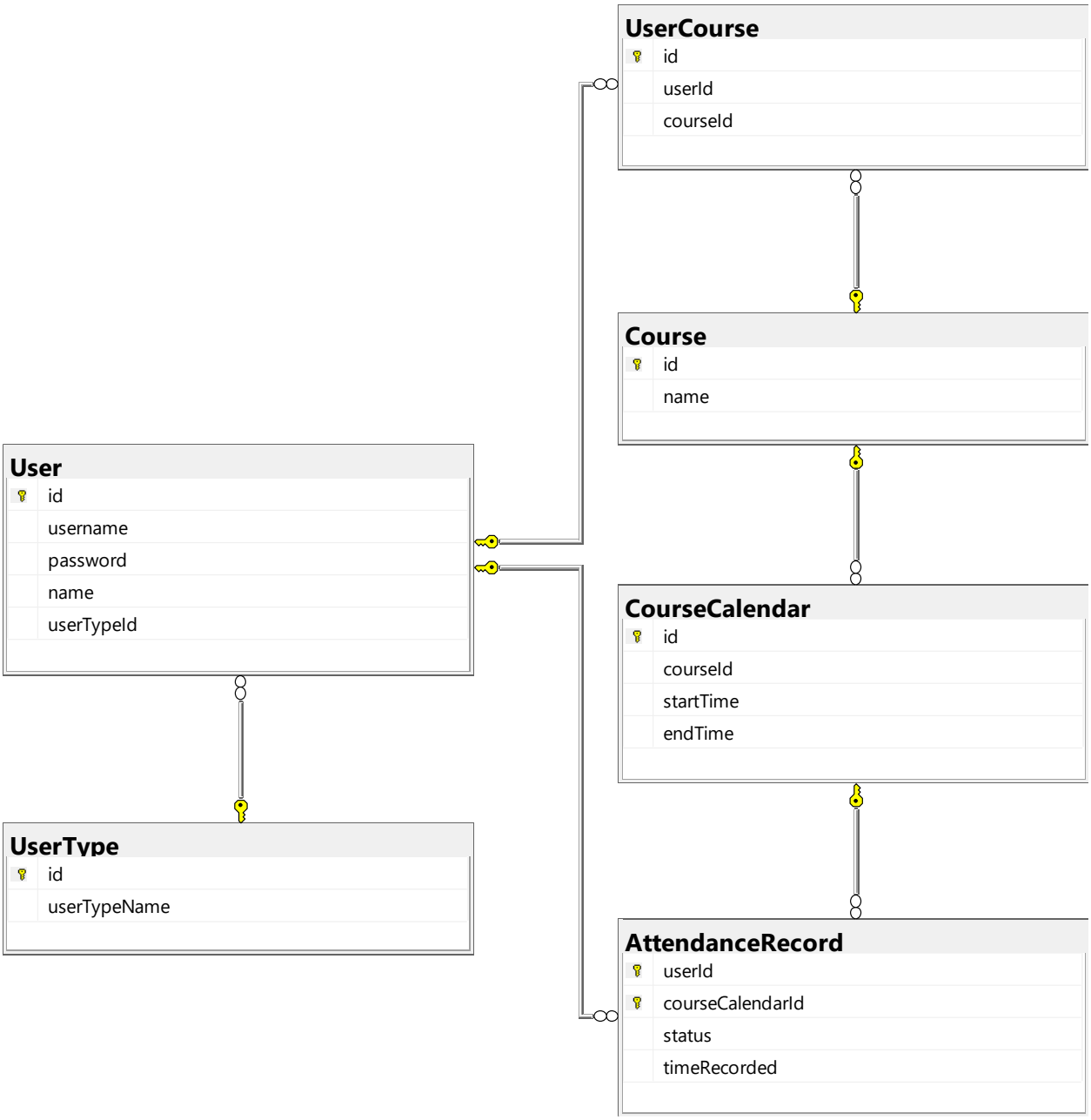


Figure 21: Entity relation diagram.

6.4 Implementation

The Team decided on adding a new business entity in the second Sprint. This Lesson business entity would store the information we needed for specific lessons, compared to the Course which contains information on the whole Course. This information would be in the form of the Instance Variables: id (int), courseName(String), startTime(LocalDateTime) and endTime(LocalDateTime).

This Entity would allow us to have the Student mark attendance for specific lessons for specific days, using the startTime and endTime variables to show only the Lessons of the current date. These Lesson's

attendance would then be stored in the Database. The attendance of these lessons is also shown in the Teacher Dashboard, getting each Lesson entity of all enrolled students, which can also be narrowed down to each specific Students Lesson attendance.

6.4.1 Code examples

A teacher can see summarized attendance for a specific course (with most absent students at the top) in a TableView in the TeacherDashboard. In order display the data, it was necessary to write methods to get the number of absent lessons and calculate the absence percentage for each student of the course.

The method to get the number of absent lessons for each student was written in the AttendanceRecordDAO (Figure 22). The method uses the SQL COUNT() function, which returns the number of rows satisfying the specified conditions.

The absence percentage needed the total number of conducted lessons for the calculation, so another method in Figure 23 was written. Finally, the absence percentage for each student was calculated using a method in the AbsencePercentageCalculator utility class (Figure 24).

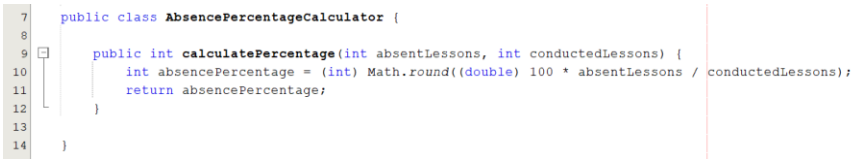
```
127     public List<Student> getNumberOfAbsentLessons(Course course) {
128         List<Student> students = new ArrayList<>();
129 
130         String sql = "SELECT T1.id, T1.name, COUNT(T2.status) AS absentLessons "
131                 + "FROM "
132                 + "     (SELECT U.id, U.name "
133                 + "     FROM [User] AS U "
134                 + "     JOIN UserCourse AS UC ON U.id = UC.userId "
135                 + "     WHERE UC.courseId = ? AND U.userTypeId = 'S') "
136                 + "     AS T1 "
137                 + "LEFT JOIN "
138                 + "     (SELECT AR.userId, AR.status "
139                 + "     FROM AttendanceRecord AS AR "
140                 + "     JOIN CourseCalendar AS CC ON AR.courseCalendarId = CC.id "
141                 + "     WHERE CC.courseId = ? "
142                 + "     AND AR.status = 'Absent') "
143                 + "     AS T2 "
144                 + "ON T1.id = T2.userId "
145                 + "GROUP BY T1.id, T1.name "
146                 + "ORDER BY absentLessons DESC";
147 
148         try (Connection con = connection.getConnection()) {
149             PreparedStatement pstmt = con.prepareStatement(sql);
150             pstmt.setInt(1, course.getId());
151             pstmt.setInt(2, course.getId());
152             ResultSet rs = pstmt.executeQuery();
153             while (rs.next()) {
154                 int id = rs.getInt("id");
155                 String name = rs.getString("name");
156                 int absentCount = rs.getInt("absentLessons");
157                 students.add(new Student(id, name, absentCount));
158             }
159             return students;
160         } catch (SQLServerException ex) {
161             Logger.getLogger(AttendanceRecordDAO.class.getName()).log(Level.SEVERE, null, ex);
162         } catch (SQLException ex) {
163             Logger.getLogger(AttendanceRecordDAO.class.getName()).log(Level.SEVERE, null, ex);
164         }
165         return null;
    }
```

Figure 22: Method from the AttendanceDAO class.

```
public int getNumberOfConductedLessons(Course course, LocalDateTime current) {
    String sql = "SELECT COUNT(id) FROM CourseCalendar WHERE courseId = ? AND endTime <= ?";

    try (Connection con = connection.getConnection()) {
        PreparedStatement pstmt = con.prepareStatement(sql);
        pstmt.setInt(1, course.getId());
        pstmt.setTimestamp(2, Timestamp.valueOf(current));
        ResultSet rs = pstmt.executeQuery();
        if (rs.next()) {
            int count = rs.getInt(1);
            return count;
        }
    } catch (SQLServerException ex) {
        Logger.getLogger(LessonDAO.class.getName()).log(Level.SEVERE, null, ex);
    } catch (SQLException ex) {
        Logger.getLogger(LessonDAO.class.getName()).log(Level.SEVERE, null, ex);
    }
    return 0;
}
```

Figure 23: Method from the LessonDAO class.



```
public class AbsencePercentageCalculator {

    public int calculatePercentage(int absentLessons, int conductedLessons) {
        int absencePercentage = (int) Math.round((double) 100 * absentLessons / conductedLessons);
        return absencePercentage;
    }

}
```

Figure 24: Method from the AbsencePercentageCalculator class.

6.4.2 Design Patterns/principles

In the Dashboard for the Teacher, all the components will change depending on the registration of a student. When there is a new attendance record, the components would need to be updated. In order to keep the UI responsive to new attendance records, it was deemed necessary to implement the Observer Pattern. The Observer Pattern keeps all dependents notified of updates, so updates can happen automatically in the UI. The implementation of the pattern was unfortunately not done by the end of the Sprint.

6.5 Sprint Review

The Sprint Review for Sprint 2 occurred on April 03, 2020. The meeting started with a demonstration of the five done user stories to the stakeholders (teachers). The completed user stories can be seen in Figure 25. The stakeholders were satisfied with the features and thought the UI was visually pleasing.

The Sprint Goal was not met as 7 out of 12 user stories were not done. The Scrum Team had failed to make realistic time estimates again in this Sprint. Several group members were struggling with their tasks due to lack of skills and their own personal struggles. This was reflected in the Burndown chart (Figure 26), as the remaining work hours decrease at a slower pace than planned. The team was advised to estimate more safely in the future.

The Scrum Team's inexperience was apparent again in this Sprint. During the Sprint, missing tasks had to be added. This is reflected on the Burndown chart where the line increases at Wed 25/3 and Tue 31/3. The team was recommended to add tasks to account for uncertainties (e.g. task for research).

The Scrum Team expressed their wish to complete the incomplete user stories in a Sprint 3.

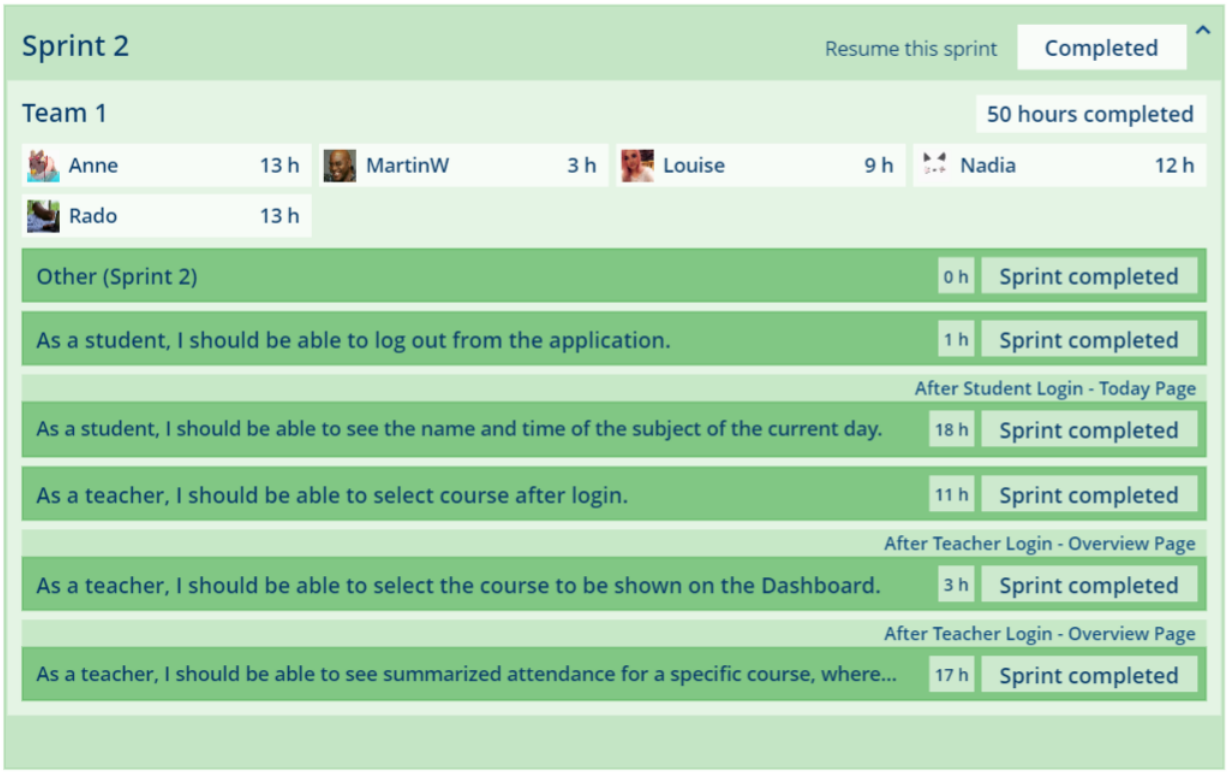


Figure 25: Scrumwise user stories completed in Sprint 2.

User stories
As a student, I should be able to register my attendance for a specific lesson.
As a student, I should be able to see an overview of my attendance.
As a teacher, I should be able to see the total number of students present in the class in order to limit cheating.
As a teacher, I should be able to see summarized attendance on each student of a given course.
As a student, I should be able to correct the attendance if it is wrong.
As a teacher, I should be able to correct the attendance for a student who requests his/her attendance to be corrected.
As a student, I should receive a notification to register my attendance.

Table 6: Incomplete user stories in Sprint 2.

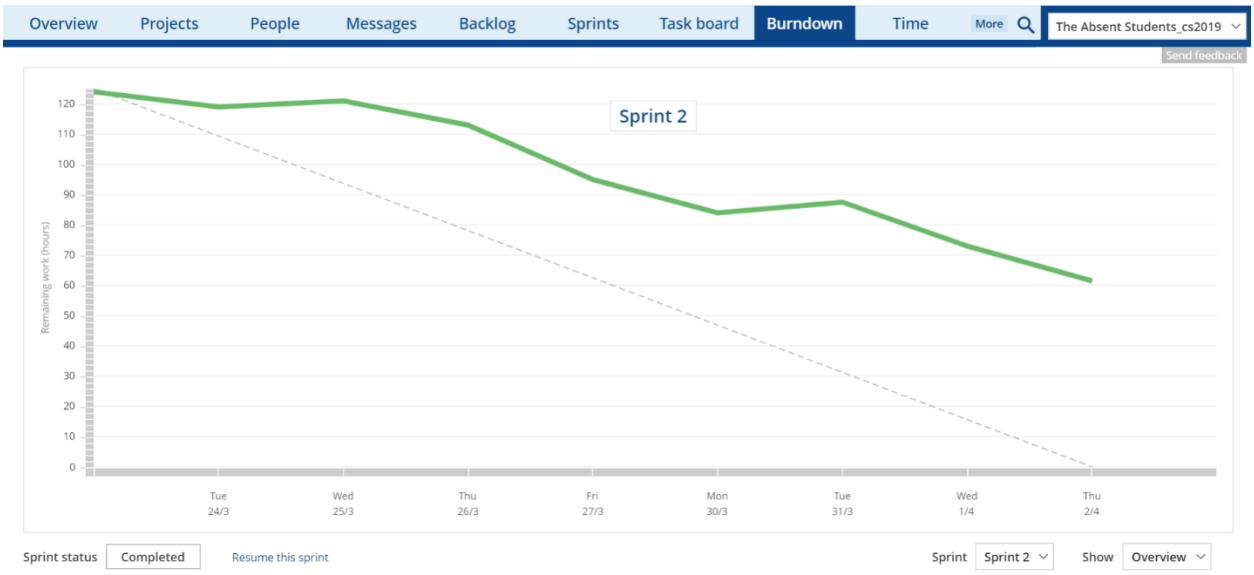


Figure 26: Scrumwise Burndown chart for Sprint 2.

6.6 Sprint Retrospective

The team used a template called "The 4 L's" by RealtimeBoard [7] for the Sprint Retrospective again for this Sprint. Table 7 shows a summary of each category.

Liked	- The improved management and coordination halfway through the Sprint. During the second week, the previously shared SCRUM responsibilities were entrusted to the Scrum Master. The Scrum Master was solely responsible for creating tasks, task delegation, overseeing the Task board and coordinating the work. - How the tasks were smaller and had better descriptions. - The improved communication and punctuality. - The support and understanding. - The close teamwork. - The shorter Daily Scrum. - The accomplishment from completing more tasks. - To ask for help and not struggle needlessly.
Learned	- The importance of a real Scrum Master with coordinating responsibilities. - To plan thoroughly and the advantages of a good plan. - The approach should be more learning focused than meeting the requirements. - Insight of own skills. - To ask for help earlier. - Time estimation is still too ambitious and not skeptical enough.
Lacked and Longed for	- Time. - Personal time management. - Knowledge. All team members could feel their own shortcomings. - Foresight. Many tasks were more comprehensive than anticipated. - Even better communication. - A coordinating role. During the first half of the Sprint, there was no coordinator to oversee the order of carrying out tasks. This was problematic for dependent tasks. - A contract/agreement, which everyone follows.

Table 7: Sprint Retrospective The 4 L's.

The Scrum Team agreed to continue the good findings and try to eliminate the bad in the next Sprint.

7 Sprint 3

7.1 Sprint planning

The priority of the user stories in the Product Backlog was reevaluated before the selection of the items for the Sprint Backlog. The changed priority can be seen in Table 8. As the assigned hours for the Sprint was 76 hours and the estimated time for the remaining user stories exceeded this number, the four highest prioritized user stories from the Product Backlog were selected for the Sprint Backlog. The Sprint Backlog of Sprint 3 can be seen in Table Y. The estimated time of the four selected user stories were 76 hours, so the Sprint was fully assigned.

User stories
As a student, I should be able to register my attendance for a specific lesson.
As a teacher, I should be able to see the total number of students present in the class in order to limit cheating.
As a teacher, I should be able to see summarized attendance on each student of a given course.
As a student, I should be able to see an overview of my attendance.
As a student, I should be able to correct the attendance if it is wrong.
As a teacher, I should be able to correct the attendance for a student who requests his/her attendance to be corrected.
As a student, I should receive a notification to register my attendance.

Table 8: Product Backlog before Sprint 3.

User stories
As a student, I should be able to register my attendance for a specific lesson.
As a teacher, I should be able to see the total number of students present in the class in order to limit cheating.
As a teacher, I should be able to see summarized attendance on each student of a given course.
As a student, I should be able to see an overview of my attendance.

Table 9: Sprint Backlog of Sprint 3.

7.2 GUI (including UI-design patterns)

No changes.

7.3 Data model

No changes.

7.4 Implementation

7.4.1 Code examples

```
83      private void setupCheckerThread() {
84          executor = Executors.newSingleThreadScheduledExecutor();
85          executor.scheduleAtFixedRate(() -> {
86              Platform.runLater(() -> {
87                  absenceGuard();
88                  refreshCombobox();
89                  tbStatusSet();
90              });
91          }, 1, 3, TimeUnit.SECONDS);
92      }
```

Figure 27: Thread for inserting absence.

After the lesson is finished and the student did not register his/her attendance, the lesson status will change from unregistered to absent. Toggle button will change the state according to the selected lesson in ComboBox. This whole checking is handled from the student's perspective only. That means if the student will not log in to the application for a few days, lessons during student's inactivity will not be set to absent and they will stay as unregistered. This could be repaired by moving the thread and deploying application on the server side. This way the application can handle the checking, then student is not required to be logged in.

7.4.2 Design Patterns/principles

Observer Pattern

The implementation of this pattern was continued in this Sprint. Two concrete observable classes implementing the same interface and one observer interface were created. All the items (two labels, two tables and a bar chart) in the TeacherDashboard are dependent on the attendance registration of the students and can change any time, so they all need to subscribe to an observable in order to be notified of any updates. The concrete observables periodically check if there are any updates by comparing the time of the last received update with time of the last attendance record entry. If an attendance record has a time later than the last received update, the observable will get all data and notify the observer to update.

The implementation in the project, successfully made it possible to automatically receive updates. However, the implementation had issues with threads and the subscription mechanism of the *Observer Pattern*. When changing the course selection in the ComboBoxes, a new observer would be created and attached to the observable. A new thread to check the database for updates would also be created each time instead of reusing an existing thread. Essentially, the threads kept increasing so it was necessary to stop the unused threads. Ideally, one thread should be created, and observers should be attached and detached. However, in the current implementation, the observer list is not used properly. Observers are never removed, and the list keeps growing.

Factory Method Pattern

During the last sprint the team decided to implement Simple factory for model creation and Factory method for Facades in BLL and DAL alongside with Dependency injection. Factories are Singleton classes so the client has the ability to get the same instance of the factory every time the client will ask for the product. Creation of objects should not be handled by client, because of high coupling. This is the reason for using the factory patterns and then injecting the dependency to the client. The patterns were implemented late in development due to lack of knowledge in Sprint 1 and Sprint 2.

The team decided to create interfaces for Models instead of keeping the Models as Singleton classes. This way the *Dependency Inversion Principle* is followed. Client (Controller) depends on expected behavior (what) rather than on specific implementation (how) of Model class [8]. The implementation can be switched without changing the code of the client.

7.4.3 Unit test

The team decided to test the two utility classes from BLL: AbsenceCounter.java and AbsencePercentageCalculator.java. For these tests the team had to create mock data.

AbsenceCounterTest

In this case a list of lessons was created inside the tested method. The test accepts the list and returns the number of absent lessons.

AbsencePercentageCalculator

In this case 2 numbers are mocked. AbsentLessons and conductedLessons. After math operation the result is returned. In this case the team created 3 test methods for the class.

If the tests are written well, development in the long run is more flexible and the bugs are introduced sooner. Truth be told the team is aware of the fact that the testing in this project was underestimated and should get more attention. The reason why the team is testing util classes and not some more complex classes is also lack of knowledge how to mock other dependencies of complex classes.

7.5 Sprint Review

No Sprint Review was conducted for Sprint 3.

It should be noted that the Sprint Goal was met this Sprint as seen in Figure 28. The Scrum Team also made safer time estimates for the tasks, which is reflected in the Burndown chart (Figure 29).

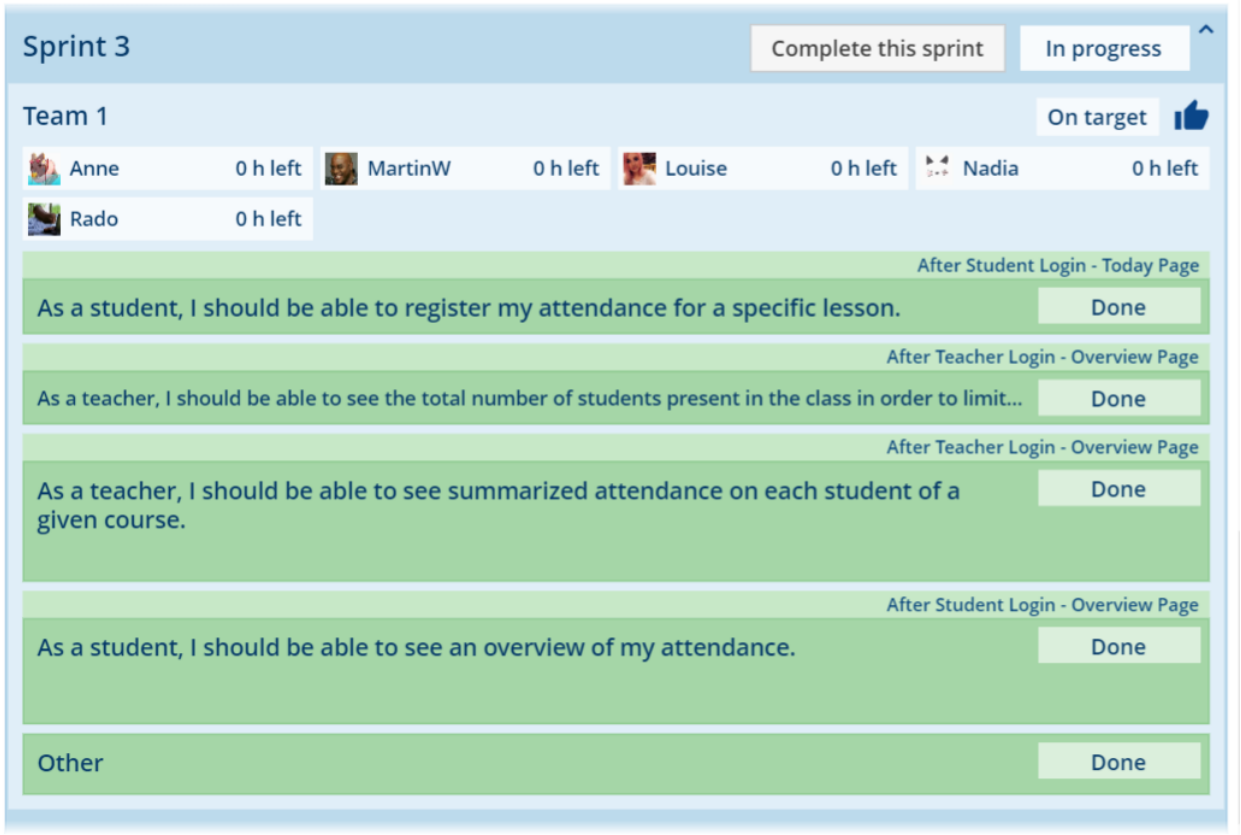


Figure 28: Scrumwise user stories completed in Sprint 3.

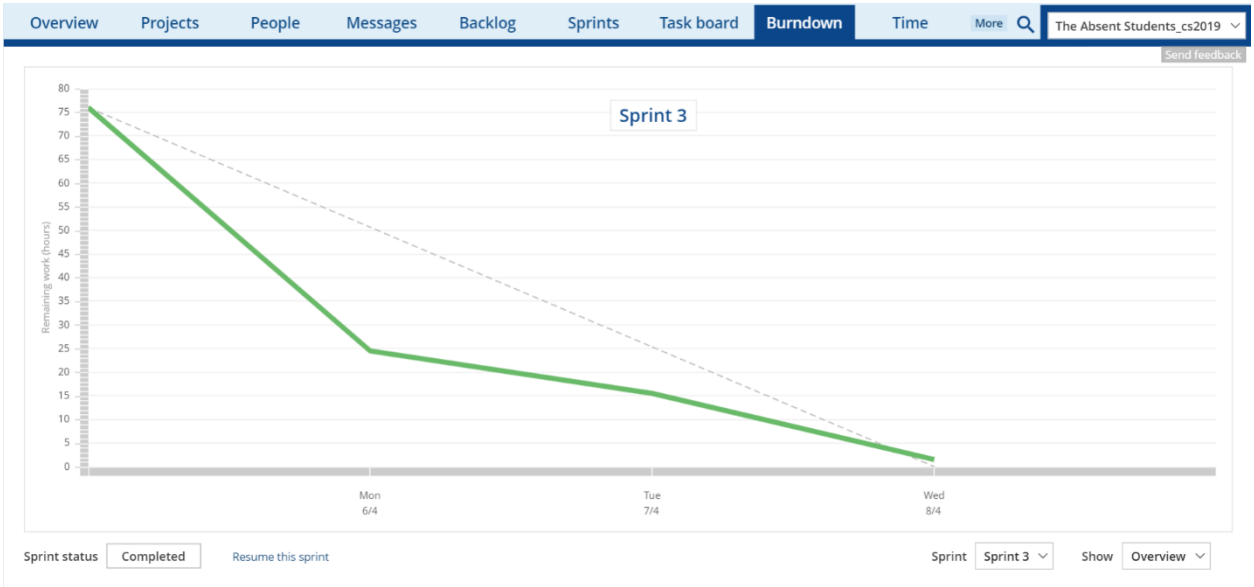


Figure 29: Scrumwise Burndown chart for Sprint 3.

7.6 Sprint Retrospective

No Sprint Retrospective was conducted for Sprint 3.

8 Conclusion

Attendance Automation System was a great challenge and rich learning experience. The team delivered a functional desktop application that serves its purpose.

The team fulfilled successfully these requirements:

1. Student should be able to register attendance for a specific course.
2. Student should be able to see his/her attendance overview and percentage of absence.
3. Teacher should be able to see overview of student's attendance and most absent student at the top.
4. Teacher should see total absence of selected student by weekday.

The team decided the requirements about correcting attendance had low priority and they were not fulfilled due to lack of time.

Every team member is aware of their shortcomings and what can be improved.

Teamwork

The team was facing management problems due to lack of leadership in the early stage. The team is aware that this can be improved by being honest to other members and speaking up if there is uncertainty. Overall the team was a supportive and hardworking unit.

Planning

The planning was difficult due to lack of knowledge of design patterns/principles and other coding concepts. For a good plan there has to be a skillset of fundamentals. The team has interest in developing scalable and cohesive solutions. That means the focus on design patterns and principles can't be underestimated in the future.

Code

Team members tend to specialize in different fields. The team is aware that sharing knowledge is important and it is worth giving it more attention.

Development of the Attendance Automation System was valuable experience. Academy provided support in different ways such as tutoring and resources which helped the team during development significantly. The team is looking forward to further development of the Exam Project.

9 References

[1] "Stakeholder Analysis," *Mind Tools*, n.d. [Online Image]. Available: https://www.mindtools.com/pages/article/newPPM_07.htm. [Accessed: Apr 15, 2020].

[2] A. Choudary, "How to Implement MVC Architecture in Java?," *Edureka*, Nov. 28, 2019. [Online Image]. Available: https://www.edureka.co/blog/mvc-architecture-in-java/. [Accessed: Apr 14, 2020].

[3] "Dashboard," *UI-Patterns.com*, n.d. [Online]. Available: https://ui-patterns.com/patterns/dashboard. [Accessed: Apr 15, 2020].

[4] "Input Prompt," *UI-Patterns.com*, n.d. [Online]. Available: https://ui-patterns.com/patterns/InputPrompt. [Accessed: Apr 15, 2020].

[5] "Keyboard Shortcuts," *UI-Patterns.com*, n.d. [Online]. Available: https://ui-patterns.com/patterns/keyboard-shortcuts. [Accessed: Apr 15, 2020].

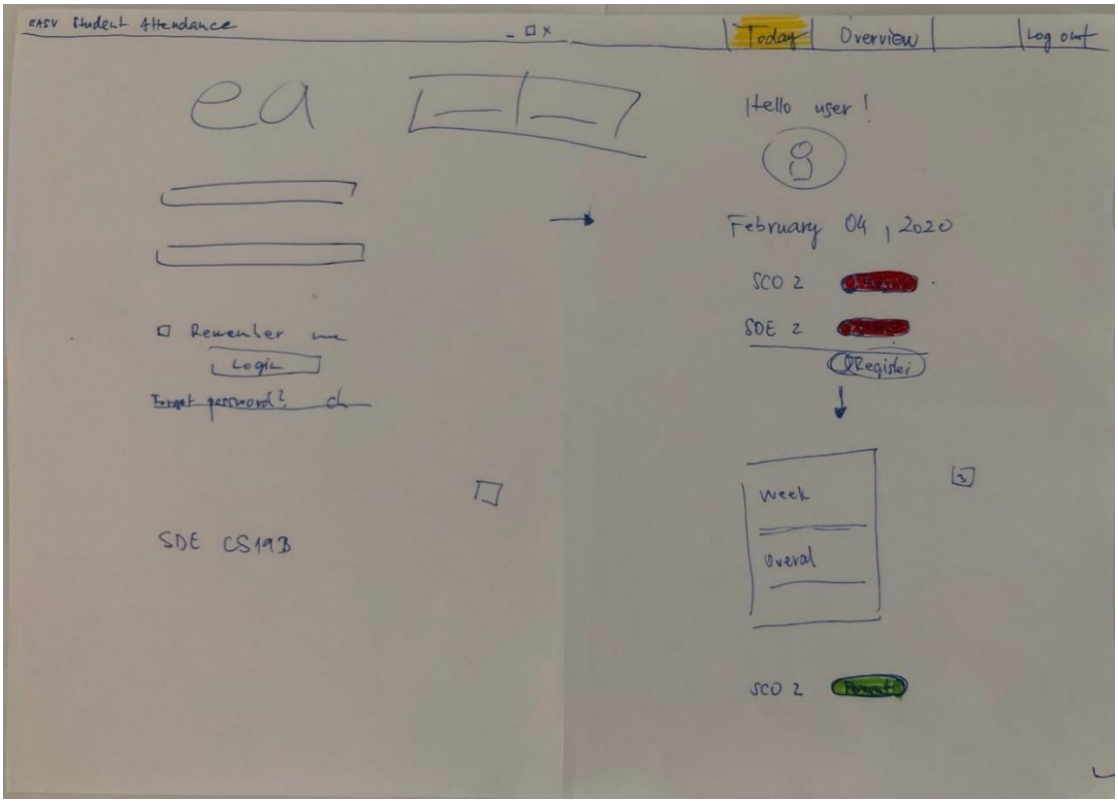
[6] " Module Tabs," *UI-Patterns.com*, n.d. [Online]. Available: https://ui-patterns.com/patterns/ModuleTabs. [Accessed: Apr 15, 2020].

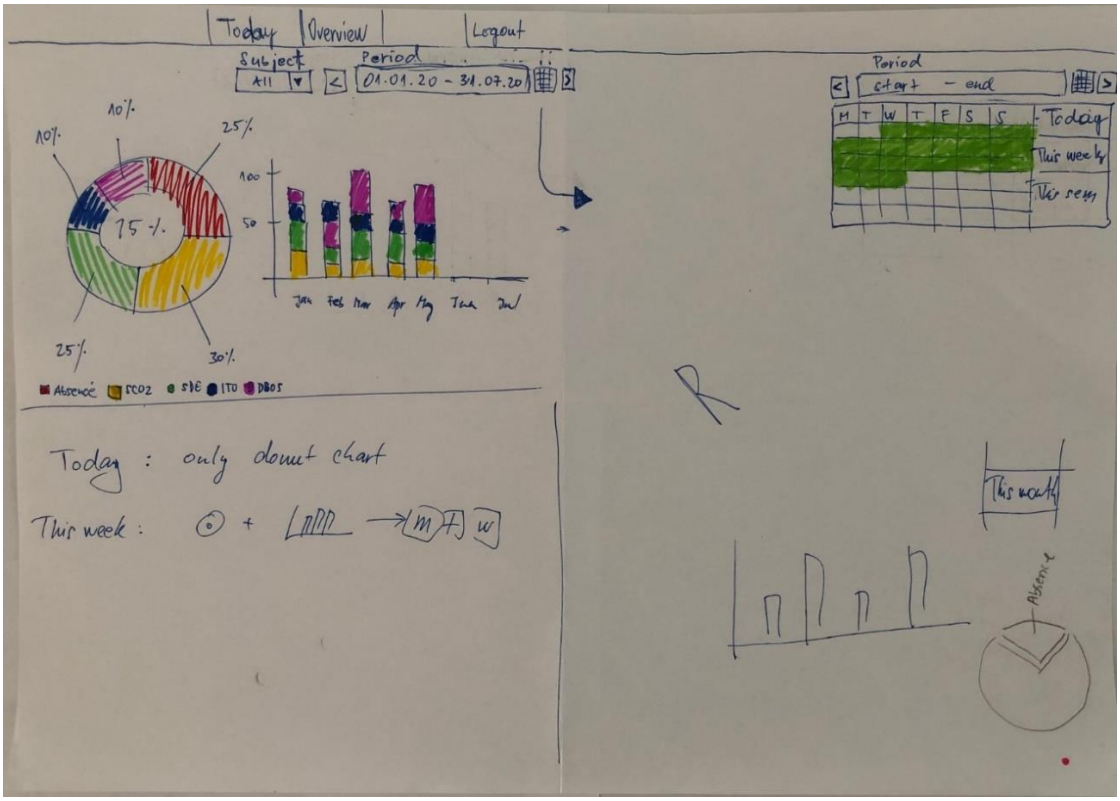
[7] RealtimeBoard, "5 fun sprint retrospective ideas with templates," *Atlassian*, Dec. 20, 2017. [Online]. Available: https://www.atlassian.com/blog/jira-software/5-fun-sprint-retrospective-ideas-templates. [Accessed: Mar 22, 2020].

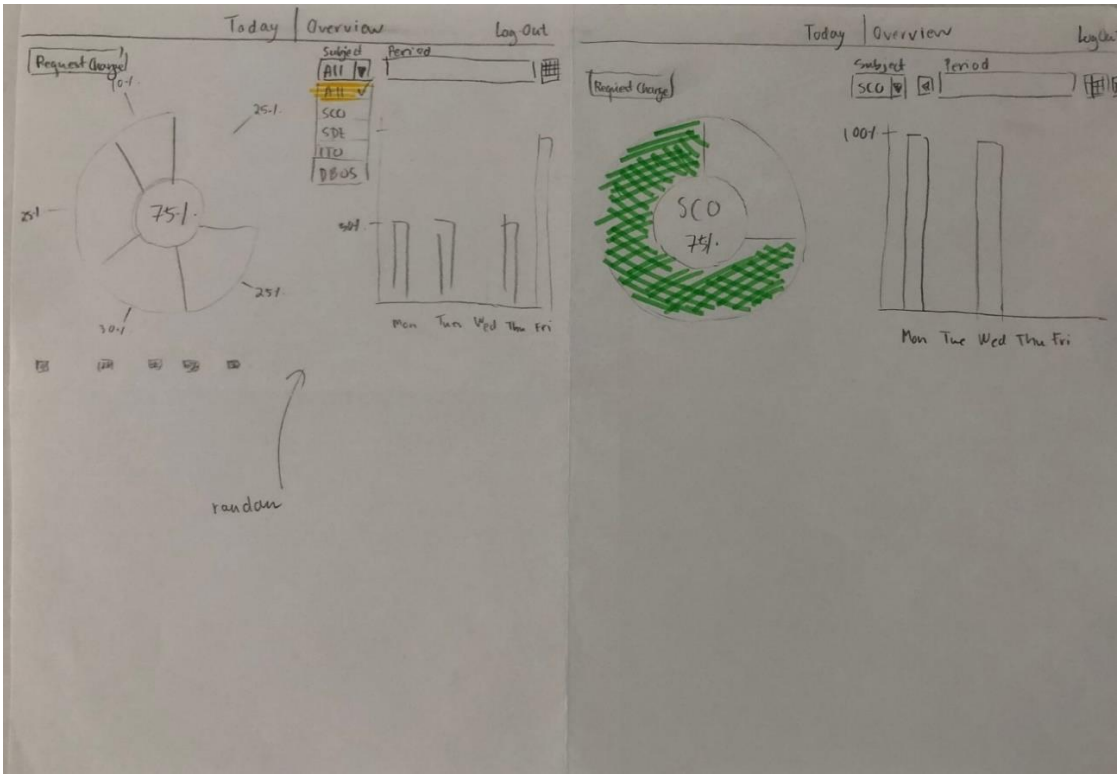
[8] J. Zhang, “Dependency Inversion Principle,” *YouTube*, Dec. 14, 2018. [Video File]. Available: https://www.youtube.com/watch?v=2d8qZbyZXc8. [Accessed: Apr. 15, 2020].

10 Appendices

10.1 Appendix 1






Today
Overview
Log Out
Request Change
10.1.
25.1.
75.1.
25.1.
30.1.
Subject
All
SCO
SDE
ITD
DBOS
Period
Mon Tue Wed Thu Fri
random
Today
Overview
Request Change
SCO
75%
Subject
SCO
Period
100%
Mon Tue Wed Thu Fri

10.2 Appendix 2