AP in Computer Science at Business Academy Southwest

bright time

First Year Examination

June 02, 2020

Radoslav Backovsky

Anne Luong

Preface

This report documents a project assignment completed for the First Year Examination in the Computer Science study at the Business Academy SouthWest (EASV).

The project took place from April 20, 2020 until June 02, 2020, where a time management system was developed for *Grumsen Development ApS*.

We were able to complete the project with the guidance and support from the lecturers of the course and would like to thank:
Stig Salskov Iversen
Søren Spangsberg Jørgensen
Jeppe Moritz Led
Peter Gaarsmand Stegger Nielsen
Trine Graungaard H. Thomsen

We also extend our gratitude to the tutors of the course:
Grzegorz Charyszczak
Máté Kiss
Nedas Šurkus

Finally, we would like to thank our family and friends for all the help and support.

Radoslav Backovsky and Anne Luong,
AP in Computer Science
Business Academy SouthWest (EASV)

Table of Contents

1. Introduction

1.1 Background

Grumsen Development ApS is a small and agile software company. The company provides digital solutions to a variety of customers from different fields. The contract terms vary for each project and oftentimes the projects run simultaneously. The owner, *Jakob Grumsen*, has been looking for a tool to monitor the time spent on each client, project and task, but the current solutions on the market does not satisfy his needs. He wants a customized solution developed for his company and expects the following:

Features for all users

- Task creation.
- Task logging using a timer with a start/pause button.
- Access to personal tasks only.
- Tabular and graphical overviews of the time spent on tasks with time frame configurations.
- Personal login.

Features for administrators:

- Client and project creation with hourly rate specification.
- User creation, update and deletion.
- Access to all tasks.

Other specifications:

- Windows or Mac based desktop client.
- MSSQL database.
- Event log in the database to ensure traceability of actions taken and errors.

1.2 Problem statement

Grumsen Development ApS does not have a suitable tool for tracking the time spent on clients, projects and tasks, but the company needs the information to monitor project expenses, employees' working hours (to determine salary) and invoicing clients. The purpose of this project is to develop a customized time management system for *Grumsen Development ApS*.

1.3 Product vision

The vision for the product (elevator pitch)

bright time is a time management system which was developed with *Grumsen Development ApS* in mind. It is a simple application which provides a convenient way to monitor and analyze time and cost for projects and tasks. No more despair, time tracking made bright!

Product stakeholders (target group)

Grumsen Development ApS is the customer. All the employees will be the users.

Value proposition

Grumsen Development ApS needs a time management system to track the time spent on each customer/project. The system should be able to log the exact time spent on each task of a project and provide a detailed overview for each project regarding the time spent on tasks and the resulting costs. The existing tools (e.g. *Clockify*) are not able to satisfy all the customer's specific needs.

An overall feature list of the product (main epic stories)

1. Log-in page
2. Time logging page for all tasks.
3. Dashboard for overview of projects.
4. Admin page.

The business goals

This product is a tool for time tracking and automatic cost calculation. It is beneficial for the company as it keeps track of project expenses, employees' working hours (which can be used to determine salary) and assists when invoicing clients.

1.4 Strategic analysis

Stakeholder analysis

A stakeholder analysis has been performed in order to provide an overview of the stakeholders in the project. Figure 1 shows the stakeholder map resulting from the analysis.

Figure 1: Stakeholder map.

Primary stakeholders:

Jakob Grumsen, Owner of *Grumsen Development ApS* (Client)

He has high power and high interest.

- The Development Team must satisfy the needs of the customer and manage contact closely. This will be done through the Product Owner and the Sprint Reviews.
- He will be an end user of the product.

Jeppe Led Moritz (Product Owner)

- He has high power and high interest.
- He has a good understanding of the project.
- He can give the Development Team support and guidance.

Development Team

- *Anne Luong* (Scrum Master, Developer)
  - She has medium power and medium interest.
  - She is the head of the Development Team.
  - She strives to satisfy the customer's requests.
- *Radoslav Backovsky* (Developer)
  - He has medium power and medium interest.
  - He strives to satisfy the customer's requests.

Secondary stakeholders:

Teachers at *EASV*

- They have high interest and low power.
- They can provide support and advice during the project.

Employees of *Grumsen Development ApS*

- They have high interest and low power.
- They will be the end users of the product.

Other student groups

- They have low interest and low power.
- They are working on the same project and competing for the customer.

Risk analysis

A risk analysis has been performed in order to identify and manage the potential risks. The analysis is shown in the table below (Table 1):

Risk	Risk Management
Human	
Illness	The risk of illness is medium considering the current novel coronavirus (2019-nCoV) outbreak and the social distancing guidelines imposed by the government.
Injury	The Development Team is not practicing any dangerous sports or activities. The team is home most of the time during the project development.
Team conflict	The Development Team has previous experience with teamwork and has no conflicts because of clear and respectful communication.
Technical	
Damage of computer	Technology can be unreliable, but the Development Team cannot mitigate this risk except using the electronic devices with care.
Loss of project data	The Development Team is using Git version control for the software and OneDrive file hosting service for backup of any documentation and the report. The team is confident that the risk of losing data is low.
Reputational	
Loss of customer	The Development Team views the risk of losing the customer at the end as high considering the high competition at the academy. The risk of losing the client during development is medium as the relationship between the customer and the academy is good.
Project	
Underestimation of time for tasks	From previous projects, the Development Team is aware of the high risk of tasks being underestimated in terms of time. The team will not underestimate the tasks and strive for thorough planning and good prioritization.
Code quality	The Development Team is composed of two students, so the quality is at high risk. As the team is still learning, there is not much the team can do other than being diligent and quality check throughout the process.
Government	
Novel coronavirus (2019-nCoV) outbreak	The possible impact of the current outbreak on the project is low. The academy has assured the project will progress smoothly despite all interaction being online.

Table 1: Risk analysis. The risk level is identified as low risk (yellow), medium risk (orange) and high risk (red).

2 Pre-Game (Sprint 0)

2.1 Project organization

Scrum will be used as the management framework throughout the project. The distribution of roles is shown in Table 2.

Role	Name
Product Owner	Jeppe Moritz Led
Scrum Master	Anne Luong
Development Team	Radoslav Backovsky Anne Luong

Table 2: Overview of the roles in the Scrum Team.

The Development Team strives to follow the Scrum principles and practices, but due to the project's nature as a school project an exception to the Product Backlog must be made. According to the Scrum Guide, the Product Owner is responsible for the Product Backlog [1]. However, in this project, the Development Team will share this responsibility. The Scrum Master will create the epic stories and user stories for the Product Backlog and the outcome will be discussed with the Development Team. Otherwise, all other practices will follow the Scrum Guide as best as possible.

Here are the most prominent Scrum practices and events:

1) The Scrum Master will be responsible for selecting Product Backlog items for the Sprint Backlog and creating the tasks for each user story.
2) The Development Team will estimate the duration of each task using planning poker.
3) The tasks will be delegated and coordinated by the Scrum Master.
4) The Task board shall constantly be modified according to the state of the project. The Development Team will update the Task board daily to reflect the actual hours spent on each task.
5) The Scrum Master will facilitate the Daily Scrum. During the Daily Scrum, the Development Team will answer the three questions (refer to 10.1 Appendix 1). Afterwards, the Task board and Burndown chart will be checked to ensure a smooth progress of the project.
6) The Scrum Team and the key stakeholders will attend a Sprint Review to evaluate the Increment and discuss the Product Backlog for future Sprints.
7) The Development Team will end a Sprint in a Sprint Retrospective to reflect on the process, outcome and the team itself.

Various software will be used during the project (see Table 3).

Name	Type
FaceTime	Video and audio communications
iMessage	Instant messaging
Discord	Text, image, video and audio communications
TeamViewer	Video communications
Zoom	Video 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
Balsamiq Wireframes	Wireframing
Lucidchart	Diagramming and visualization
Visual Paradigm	UML diagramming
Draw.io (Diagrams.net)	Diagramming
Microsoft Word	Word processing
Microsoft OneDrive	File hosting and synchronization service

Table 3: Software used in the project.

Communication

The communication channels will be *iMessage*, *FaceTime audio* and *Discord. Discord* will also be used as a space to post updates, questions and documentation for the project. In the wake of the novel coronavirus (2019-nCoV) outbreak, all meetings will be held on *TeamViewer* or *Zoom*, which both have screen sharing.

Project management

The Development Team will use *Scrumwise* as a tool to perform the Scrum practices pertaining the Product Backlog and Sprint planning/management (Sprint Backlog, Task board and Burndown chart). The team has also decided to make a daily log to document the progress of the project.

Planning

All prototypes (wireframe mockups) will be created with *Balsamiq Wireframes*. The diagramming tools *Lucidchart*, *Visual Paradigm* and *Draw.io (Diagrams.net)* will be used to create diagrams e.g. UML and ERD.

Software development

The software will be developed in *NetBeans* and persistent data will be stored on a database 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. The Git clients, *GitHub (GitHub Desktop)* and *GitKraken*, will be used in order to use *Git* efficiently.

Report writing

Microsoft Word will be used for all word processing and document creation. Through *OneDrive* and *SharePoint*, the documents will be collaborated.

Project material

All material related to the project will be stored in a shared *OneDrive* folder for easy access and backup.

Finally, the team has made a working agreement to ensure a smooth and efficient collaboration (10.2 Appendix 2).

2.2 Overall project schedule

Table 4 shows the overall schedule for the whole project.

Week	Date	Time	Activity
	April 20, 2020	9:00	Kick-off meeting: 1) Short intro to the exam project (Jeppe) 2) Problem presentation by the company owner (Jakob Grumsen) 3) Questions (Jeppe, Jakob Grumsen and students) 4) Deeper description of the exam project requirements and tips and tricks (Jeppe)
	April 20, 2020 – April 24, 2020		Pre-Game (Sprint 0) planning
	April 24, 2020 – April 25, 2020		Sprint 1 planning
	April 25, 2020 – May 06, 2020		Sprint 1
	May 07, 2020	12:40 – 13:00	Sprint 1 Review (20 min)
	May 07, 2020		Sprint 1 Retrospective
	May 07, 2020 – May 08, 2020		Sprint 2 planning
	May 08, 2020 – May 17, 2020		Sprint 2
	May 18, 2020	10:20 – 10:40	Sprint 2 Review (20 min)
	May 18, 2020		Sprint 2 Retrospective
	May 18, 2020 – May 19, 2020		Sprint 3 planning
	May 19, 2020 – May XX, 2020		Sprint 3
			Sprint 3 Retrospective
	June 02, 2020	14:00	Hand-in of the report and program on WiseFlow
	June 03, 2020	9:00	Product of the Year award: A representative from the company will choose the best product and winners will get a prize.
	June 16, 2020		Examination

Table 4: The overall project schedule.

2.3 Initial Product Backlog

Table 5 shows the initial Product Backlog with the four epic stories identified in *1.3 Product vision*. The epic stories are not prioritized in the table.

Each epic story contains user stories, which are written from the minimum requirements stipulated in the official document on *Moodle*. Additionally, *Jakob Grumsen* from *Grumsen Development* made specific and elaborate requests at the Kick-off meeting. These requests have been taken in consideration during the preliminary prototype of the Time Tracking Page and user stories has been made for each feature.

The Development Team faces limitations due to the team size, so it is important to clearly identify the minimum requirements. This distinction is shown using black and gray font in Table 5.

10.3 Appendix 3 shows some documentation from *Scrumwise* of the Product Backlog. The Product Backlog was changed frequently during Pre-Game as the prototype was changed.

Login Page
As a user, I should be able to log in to the application using a personal login.
Time Tracking Page
As a user, I should be able to create tasks in an existing project.
As a user, I should be able to view a task, which I have created.
As a user, I should be able to log the precise amount of time spent using a simple start/pause on a specific task.
As a user, I should be able to set a task as billable.
As a user, I should be able to edit the name, client, project and logged time of a specific task.
As a user, I should be able to manually log the time used on a specific task.
As a user, I should be able to delete a specific task.
As a user, I should be able to select a date (in a date picker) and view all my tasks for the selected day.
As a user, I should be able to easily select and view tasks of another day in the same week as the currently selected date.
As a user, I should be able to see the total time for all tasks in a week.
Overview Page (Dashboard)
As a user, I should be able to see the time I spent on projects in a tabular overview.
As an admin, I should be able to view data for all users.
As a user, I should be able to configure the tabular overview. (Configurations 1 week, 1 month or even specify a period of days.)
As a user, I should be able to see the time I spent on projects in a graphical overview.
As a user, I should be able to configure the graphical overview. (Configurations 1 week, 1 month or even specify a period of days.)
Admin Page
As an admin, I should be able to create clients.
As an admin, I should be able to create projects.
As an admin, I should be able to set an hourly price for a project, which is only visible for me.
As an admin, I should be able to assign projects and tasks to users.
As an admin, I should be able to create users.
As an admin, I should be able to update users.
As an admin, I should be able to delete users.
Other (user stories not belonging to an epic story)
Initial Setup
As a user, I should be able to navigate the different pages.
As a user, I should be able to log out from the application.

Table 5: The initial Product Backlog. User stories in black font are minimum requirements, while the user stories resulting from the initial prototype are in gray font. The epic stories and user stories are not prioritized.

2.4 Architecture

3-layered architecture will be implemented because it promotes high cohesion and low coupling. The benefit of decoupling the layers will ensure independent development of the layers. Every layer has access to the business entities of the application.

To further the decoupling, the **Façade pattern** will be implemented for each layer. Each façade will encapsulate and hide the complexity of the subsystem within a single interface object. From previous

projects, the Development Team has only used the pattern for BLL and DAL. However, in this project the team also wants to implement the pattern in GUI due to high coupling between the models and controllers.

The widely used **MVC pattern** (refer to Figure 2) will also be implemented in GUI to attain higher cohesion and lower coupling.

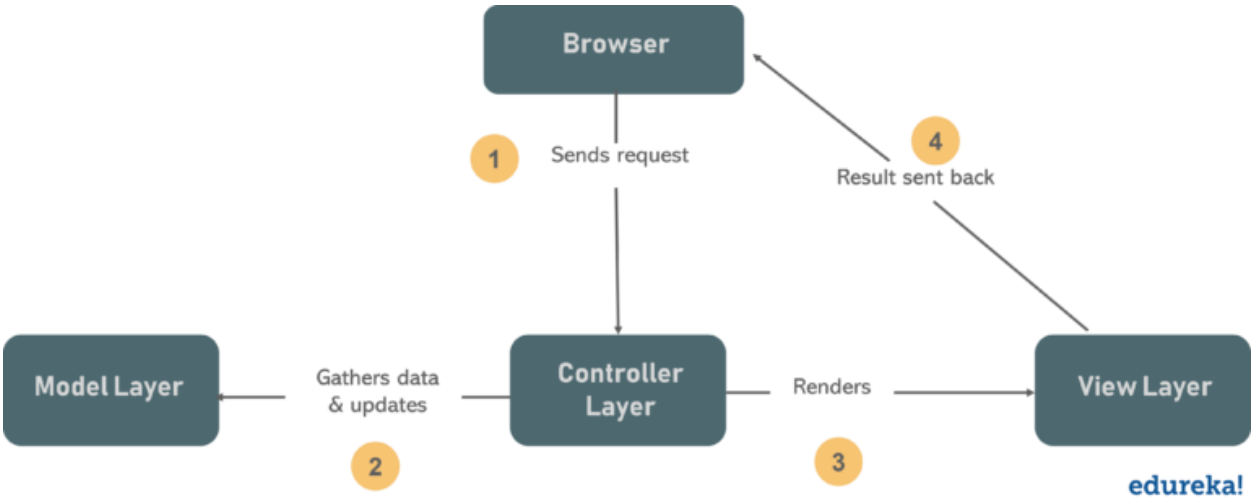


Figure 2: The communication and responsibilities between MVC components. [2]

2.5 Preliminary usability test

The core functionality of the application is tracking time of tasks, so the upcoming Sprint will focus on implementing task creation, displaying the created tasks and logging time for the tasks.

Prior to the Sprint 1 planning, the Development Team has deemed it necessary to perform market research on existing solutions, develop a prototype and perform preliminary usability tests.

Figure 3 shows the developed prototype. The high-fidelity prototype was created in *Balsamiq Wireframes* and made as interactive as possible to get better test results. The prototype used for the usability tests as well as the test results can be found in 10.4 Appendix 4.

Figure 4 and 5 show the prototype changes according to the participants feedback. The prototype would be further developed and finalized during the upcoming Sprint.

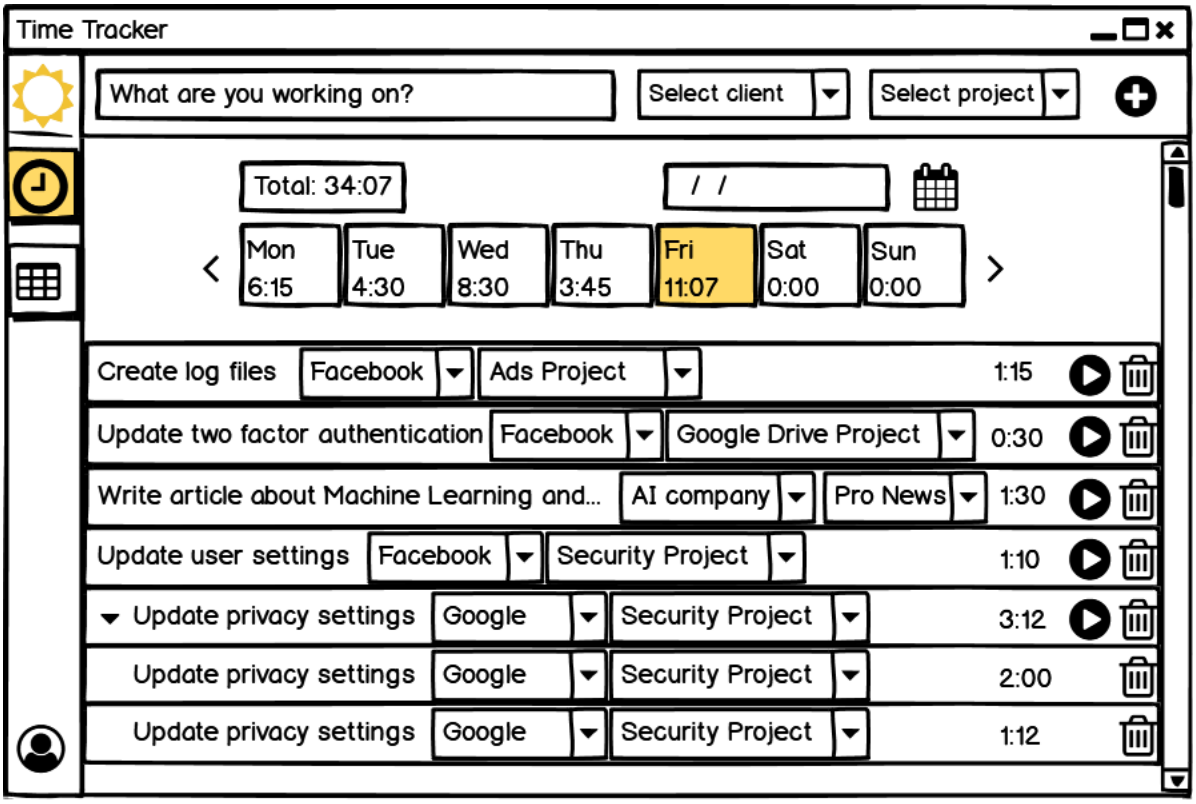


Figure 3: The preliminary prototype for the Time Tracking Page.

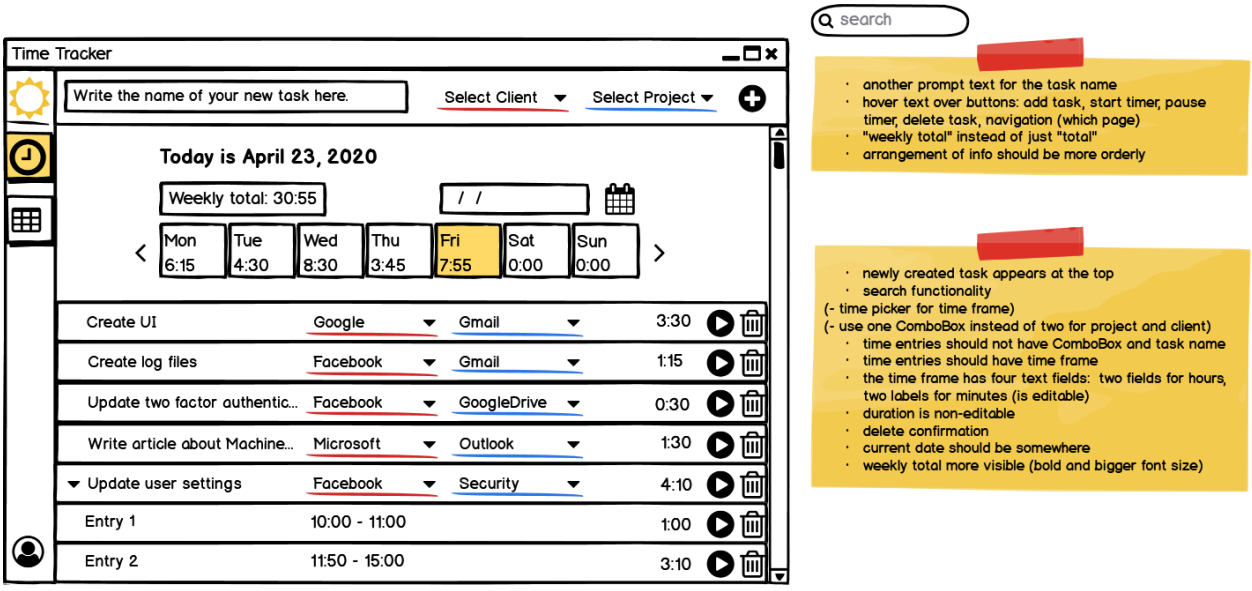


Figure 4: Usability test results of the participant Máté Kiss.

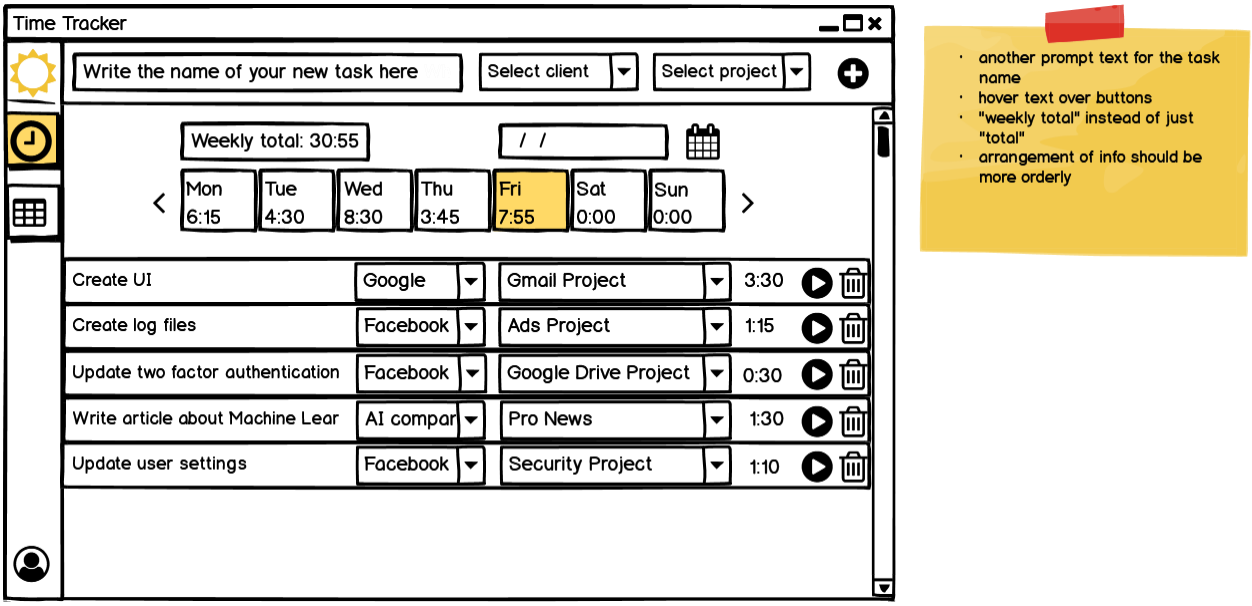


Figure 5: Usability test results of the participant Tu Nga Quan.

3 Sprint 1

3.1 Sprint planning

The planning involved the following phases:

Creation of the Sprint Goal

The Development Team identified the core functionality of the application to be time tracking of tasks during the Pre-Game. The Sprint Goal was defined as “Get the Time Tracking Page ready for release.”

Prioritization of the user stories on the initial Product Backlog

The Scrum Master prepared and proposed a prioritized list of the backlog items based on the Sprint Goal. The Development Team discussed and reviewed the proposed Sprint Goal and items. It was apparent that it would be impossible to complete all user stories in Sprint 1, so the priority was crucial and carefully considered. A preliminary prioritization was agreed upon.

Decomposition of the user stories into tasks

The Scrum Master created meaningful tasks for the user stories, which were likely to be a part of the Sprint.

Time estimation of the tasks

The Development Team started by deciding the work hours of each day of the Sprint. It was agreed that each team member would work 8 hours a day and the weekends would be designated rest days to prevent burnout. The first day of the Sprint happened to be a Saturday, so an exception was made as the Development Team wanted to do the initial set up before Monday.

The tasks in each user story was estimated. One user story was estimated at a time according to the order on the list. When there was enough work planned for the Sprint, the Development Team stopped estimating tasks.

Finalizing the Sprint Backlog

The resulting Sprint Backlog is shown in Figure 6 below.

At the end of the time estimation meeting, the Development Team reviewed the items which were not a part of the Sprint Backlog. The Scrum Master suggested to prepare prototypes for these items in order to get feedback at the Sprint Review. The Development Team aspired to do this if time permits.

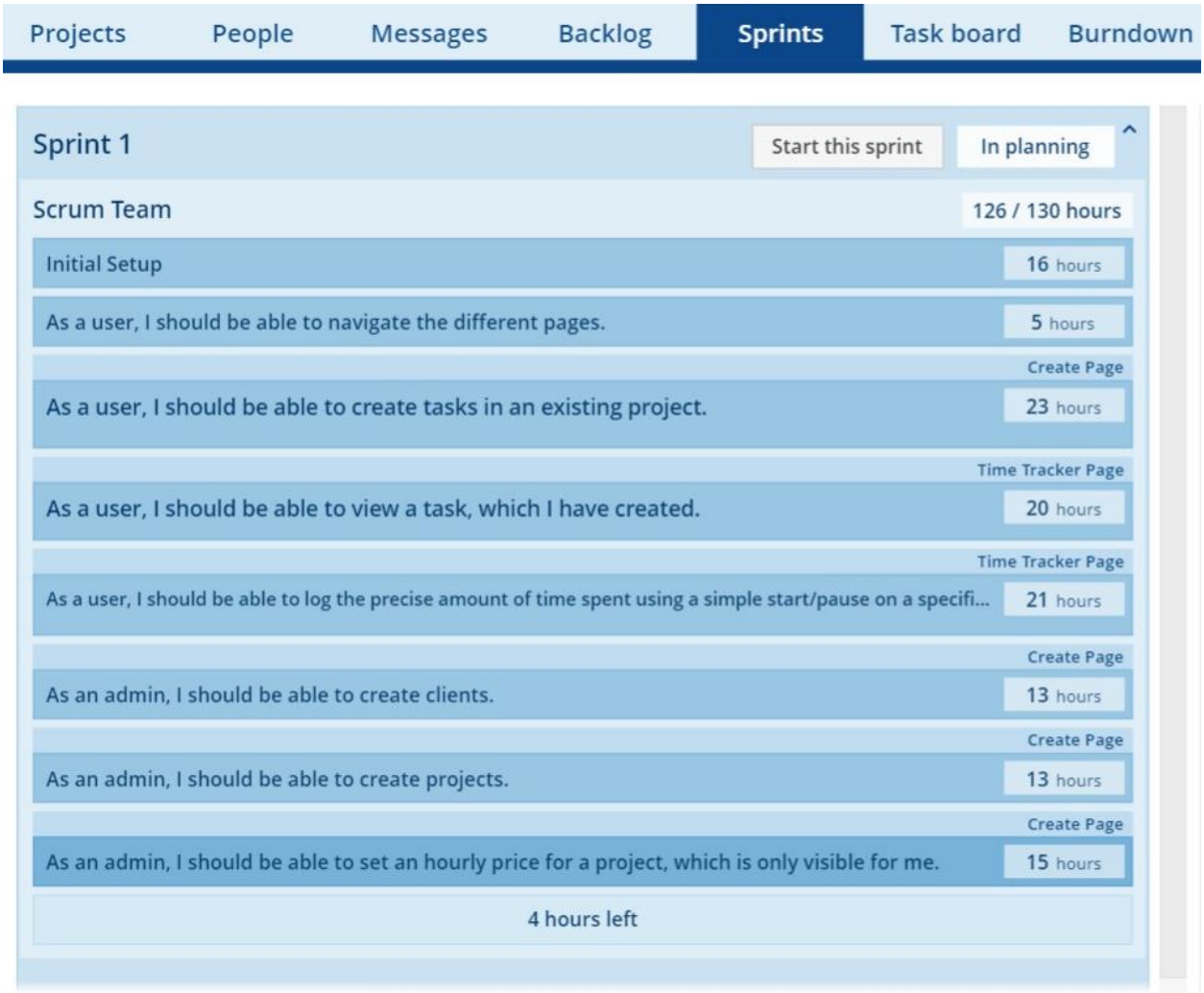


Figure 6: Sprint 1 Backlog before starting the Sprint.

3.2 GUI (including UI-design patterns)

The Development Team started with implementing the task view. Working with mock data helped with independent development of GUI and DAL. In Figure 7, the team was able to view tasks in a desired relationship. Task entries are underneath the Task as children.

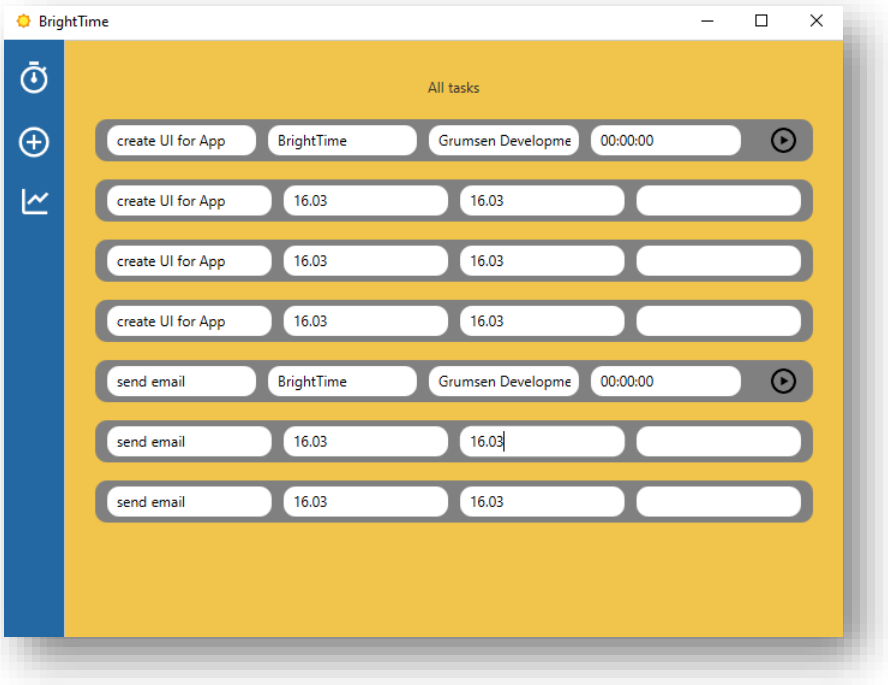


Figure 7: Initial design of the Time Tracker Page.

At the end of the Sprint 1, the team managed to set up the expanding functionality of the task revealing its children. Additional functionality of task creation and visual design has been completed as shown in Figure 8.

Figure 8: Time Tracker Page at the end of Sprint 1.

A vertical bar (sidebar) was used as the navigation choice. The vertical bar was chosen over the horizontal bar as it is more compact and uses less space. The bar was placed on the left as a study has shown that vertical bars on the left perform better, because most users are left to right readers. [3]

Drop-down menu navigation was used in combination with the vertical bar to keep the bar uncluttered, but still provide short cuts to the administrative features. [3]

In order to enhance the user experience and usability, the *Input Prompt* pattern and *Input Feedback* pattern were implemented. The *Input Prompt* pattern was used various places such as TextFields and ComboBoxes. Tooltips and alerts are examples of *Input Feedback* pattern implementations. [4,5]

Finally, it was highly prioritized that all the Scenes were responsive.

3.3 Data model

The system uses *Microsoft SQL Server Management Studio*, which is a RDBMS (relational database management system). RDBMS is a management system based on the relational database model [6].

In order to create the database for the system, ERDs (Entity Relationship Diagram) were made. An ERD can have three levels of abstraction as shown in Table 6 [7].

ERD features	Conceptual model	Logical model	Physical model
Entity (Name)	Yes	Yes	Yes
Relationship	Yes	Yes	Yes
Columns		Yes	Yes
Column's Types		Optional	Yes
Primary Key			Yes
Foreign Key			Yes

Table 6: Types of ERD. [7]

Initially, the conceptual model in Figure 9 was created to get an overview of the entities and the relationship between them. The identified entities included *Task*, *Project*, *Client* and *User* and the relationship cardinality and ordinality between the entities were expressed in crow ‘s foot notation [8].

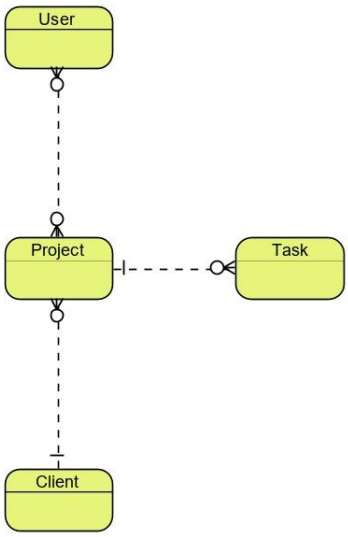


Figure 9: Conceptual ERD.

As more details were added to the conceptual model, the model developed into a logical model and in the end a physical model. Figure 13 shows the database design at the end of the Sprint.

In this Sprint, the database contained the relations *Client*, *Project*, *Task* and *TaskEntry*. At this point, the user stories did not involve users, so the *User* relation was not created yet. There is a one-to-many relationship between the relation pairs *Client* and *Project*, *Project* and *Task*, and *Task* and *TaskEntry*. In this type of cardinality, one instance of the first entity can be related to many instances of the second (any number from zero to infinity), while one instance of the second entity only is related to exactly one instance in the first.

In order to reduce data redundancy and improve data integrity, database normalization (3NF) is used to structure all entities [7]. Each relation has an auto-incremented id as a primary key and the relation between entities are referenced through foreign keys.

The *Client* and *Project* relation had no drastic changes after creation over the Sprint except adding the attribute *hourlyRate* to the *Project* relation to represent the hourly rate of a project. However, the *Task* relation underwent several changes. The reasons for the changes are explained in the timeline below:

April 25, 2020

The relations *Client*, *Project* and *Task* were created according to the design in Figure 10. In the preliminary prototype, one task would take place on one day only.

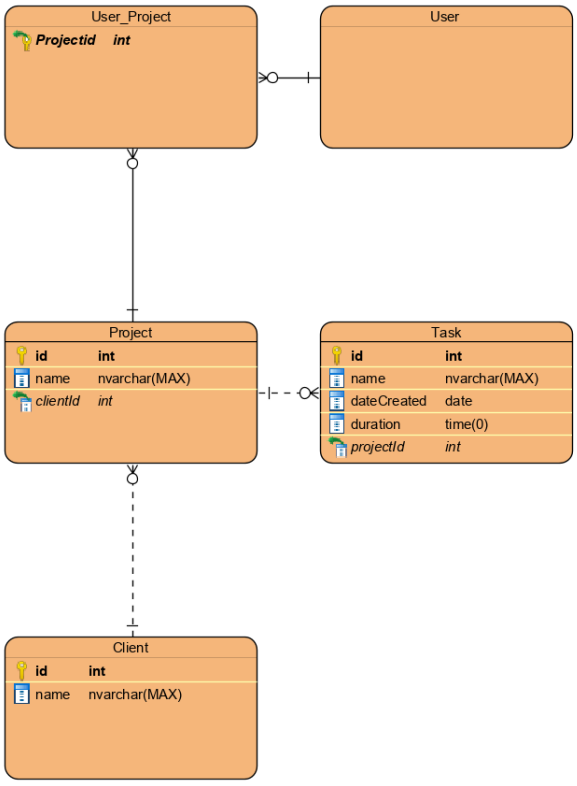


Figure 10: Physical ERD from April 25, 2020. *Client*, *Project* and *Task* relations were created in the database.

April 27, 2020

The domain of the *dateCreated* attribute was changed from date to datetime2(0) in order to display the tasks in the order of creation. The tasks could also be ordered by id to achieve this effect but adding the time component to the attribute was a more logical and simple approach.

datetime2 was chosen over datetime because it has user specified precision and a smaller storage size (6 bytes instead of 8 bytes. [9,10].

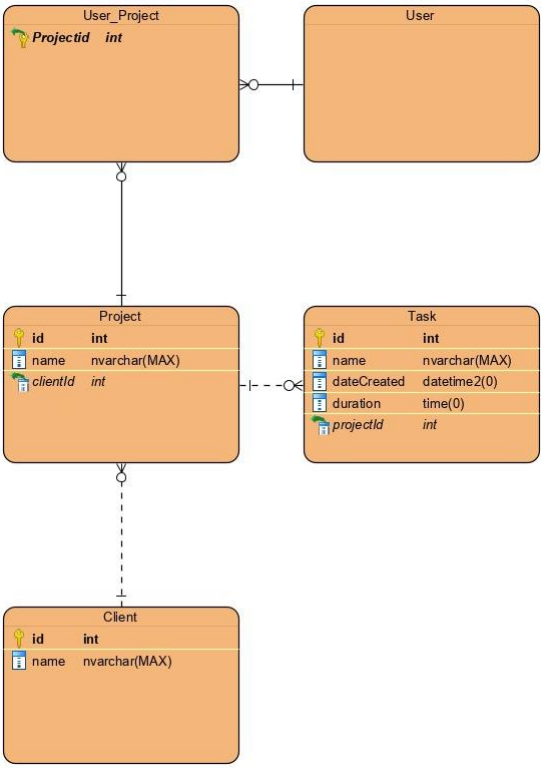


Figure 11: Physical ERD from April 27, 2020. The domain of the attribute *dateCreated* in the *Task* relation was changed to datetime2(0).

April 28, 2020

A new relation *TaskEntry* was added, since the prototype had changed. Instead of one task taking place on one day only, a task could span several days. The attribute *duration* (time(0)) would not be suitable for this setup as the time domain is limited to the range 00:00:00.0000000 through 23:59:59.9999999 [11].

The *name* attribute was renamed to *description* as it was found more suitable and explanatory.

The *dateCreated* attribute was changed to *lastUpdate*. The attribute was used to find out if a task had any updates within 30 days as any tasks logged within this time frame should be displayed on the UI. At this point, getting tasks from the database to the UI consisted of two DAO methods.

1) Get the tasks which were updated within the last 30 days.
2) Get the entries for the tasks.

In hindsight, it was not necessary to make two methods. An approach with only one DAO method is more efficient, but the SQL statement would be more complicated, and the developer decided to use the simple approach due to time constraints. One DAO method with a more complicated SQL statement was eventually implemented at a later point.

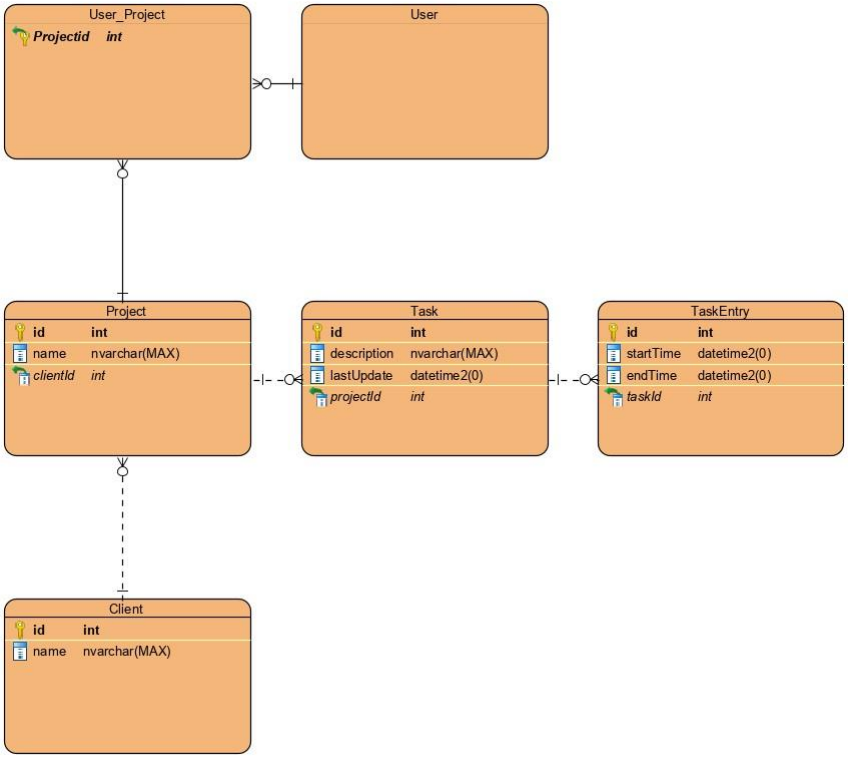


Figure 12: Physical ERD from April 28, 2020. The *TaskEntry* relation was created in the database and the *Task* relation was changed.

May 04, 2020

The *lastUpdated* attribute was renamed to *modifiedDate* as the name was deemed more suitable.

A *createdDate* attribute was added. The scenario of a task without any entries was not considered before. As the Development Team discovered this issue, a DAO method to find empty tasks (tasks without entries) was created. The simplest way was to add the *createdDate* attribute and if the *createdDate* and *modifiedDate* were the same, the task had no entries. This was not a good approach as one more DAO method resulted in one more database call. The methods for getting tasks for the Time Tracker was eventually refactored in Sprint 2, so all data is obtained through one database call.

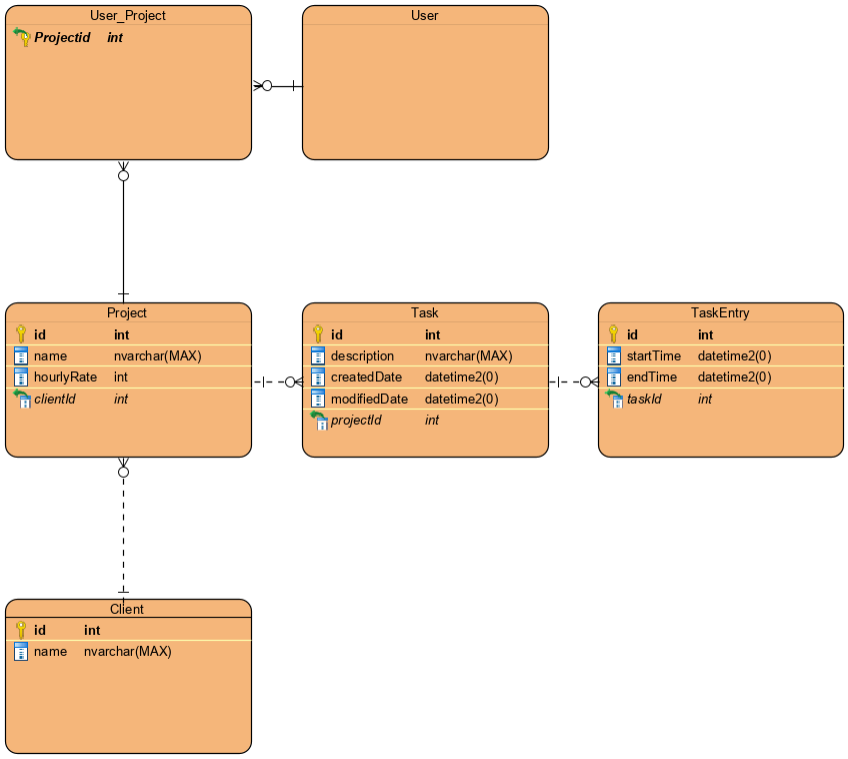


Figure 13: Physical ERD from May 04, 2020, which shows the state of the database at the end of Sprint 1. Changes were made to the *Task* relation attributes.

3.4 Implementation

As a starting point, the Development Team did the initial setup before working on any user stories. This included the initial architecture with packages and classes.

Sprint 1 was about creating, viewing and playing a task. The team had problems with deciding how to implement viewing tasks. The first idea was implementing tree table view so the **task** could be expanded, and **task entries** could be seen. After discussion the team decided to drop tree table view and try to display **tasks** and **task entries** as custom FXML files. Displaying **tasks** this way was more interesting and challenging and the team had more freedom of item's visual style and addition of different components inside the items. Task would hold its entries for the current day and reflect its data. Figure 14, Figure 15 **Task Model** and **Task Entry Model** are responsible for providing data needed for UI.

The team wanted to display time frame of entries and total duration on task item. The team decided to create additional properties inside the **Task Model** and **Task Entry Model** and bind them with properties of textfields in controllers.(Figure 17) Specific **Task Model** filters data for specific **task** in UI on specific day with specific number of **Task entries** because **Task BE** holds list of all entries from different days. The team decided to filter the list after expanding the **task** in UI with Java streams and sorting with help of Comparator interface. Figure 16

The team had issues with refreshing UI after creating new task and pausing the task (Task entry creation). The team was calling the database and initializing whole UI again. The reason could be the database wanted to fetch new data that were in the middle of being inserting.

3.4.1 Code examples

The team decided to use a HashMap for storing the data, because tasks had to be split between days. The Map holds all data about the tasks and entries. Map contains values of Task lists set on date key because the team had to know when the date label must be created for separation. Reference of specific task can appear in different lists on different keys.

```
85      private void initTasks() {
86          try {
87              mainModel.loadTasks();
88              vBoxMain.getChildren().clear();
89              Map<LocalDate, List<Task>> taskList = mainModel.getTasks();
90              Map<LocalDate, List<Task>> orderedMap = new TreeMap<>(Collections.reverseOrder());
91              orderedMap.putAll(taskList);
92              for (Map.Entry<LocalDate, List<Task>> entry : orderedMap.entrySet()) {
93                  LocalDate dateKey = entry.getKey();
94                  List<Task> taskListValue = entry.getValue();
95                  if (!dateKey.equals(date)) {
96                      String formatted = dateKey.format(DateTimeFormatter.ofLocalizedDate(FormatStyle.LONG));
97                      Label label = new Label(formatted);
98                      label.getStyleClass().add("labelMenuItem");
99                      vBoxMain.getChildren().add(label);
100                     label.translateXProperty().set(25);
101                     date = dateKey;
102                 }
103                 for (Task task : taskListValue) {
104                     addTaskItem(task);
105                 }
106             }
107         } catch (ModelException ex) {
108             alertManager.showAlert("Could not get the tasks.", "An error occured: " + ex.getMessage());
```

Figure 14: Loading and initializing the task view.

```
113      private void addTaskItem(Task task) {
114          try {
115              FXMLLoader fxmlLoader = new FXMLLoader(getClass().getResource(TASK_ITEM_FXML));
116              Parent root = fxmlLoader.load();
117              ITaskModel taskModel = ModelCreator.getInstance().createTaskModel();
118              taskModel.setTask(task);
119              taskModel.setDate(date);
120              TaskItemController controller = fxmlLoader.getController();
121              controller.injectTimeTrackerController(this);
122              controller.injectModel(taskModel);
123              vBoxMain.getChildren().add(root);
124          } catch (IOException ex) {
125              Logger.getLogger(TimeTrackerController.class.getName()).log(Level.SEVERE, null, ex);
126          }
127      }
```

Figure 15: Creating custom Task Item with its own custom TaskModel

```
103      @Override
104      public List getDayEntryList() {
105          List<TaskEntry> dayEntries = task.getTaskEntryList().stream().filter(allEntries
106                  -> allEntries.getStartTime().toLocalDate().equals(date)).collect(Collectors.toList());
107          dayEntries.sort(Comparator.comparing(o -> o.getStartTime()));
108          return dayEntries;
109      }
```

Figure 16: Filtering and sorting task entry list inside the task.

```
125      public void setTaskDetails(Task task) {
126
127          DateTimeFormatter dtf = DateTimeFormatter.ofPattern(DATE_TIME_FORMAT);
128          textFieldTaskDesc.textProperty().bind(Bindings.createStringBinding(()
129                  -> task.getDescription(), task.descriptionProperty()));
130          textFieldClient.textProperty().bind(Bindings.createStringBinding(()
131                  -> task.getProject().getClient().getName(), task.getProject().getClient().nameProperty()));
132          textFieldProject.textProperty().bind(Bindings.createStringBinding(()
133                  -> task.getProject().getName(), task.getProject().nameProperty()));
134          textFieldStartTime.textProperty().bind(Bindings.createStringBinding(()
135                  -> dtf.format(taskModel.getStartTime()), taskModel.startTimeProperty()));
136          textFieldEndTime.textProperty().bind(Bindings.createStringBinding(()
137                  -> dtf.format(taskModel.getEndTime()), taskModel.startTimeProperty()));
138          textFieldDuration.textProperty().bind(Bindings.createStringBinding(()
139                  -> taskModel.secToFormat(taskModel.calculateTaskDuration(taskModel.getDayEntryList()).toSeconds()),
140                  taskModel.stringDurationProperty()));
141      }
```

Figure 17: Initial bindings of task item to properties of TaskConcrete1 BE and properties of Task Model

```
115     private void addTask() {
116         btnAdd.setOnAction((event) -> {
117             if (!txtDescription.getText().trim().isEmpty() && !cboProject.getSelectionModel().isEmpty()) {
118                 try {
119                     Task task = new Task(txtDescription.getText().trim(), cboProject.getSelectionModel().getSelectedItem());
120                     mainModel.addTask(task);
121                     timeTrackerContr.initializeView();
122                     System.out.println("action event is working!");
123                 } catch (ModelException ex) {
124                     alertManager.showAlert("Could not create the task.", "An error occured: " + ex.getMessage());
125                 }
126             } else if (txtDescription.getText().trim().isEmpty()) {
127                 alertManager.showAlert("No task description was entered.", "Please enter a description of the new task.");
128             } else if (cboClient.getSelectionModel().isEmpty()) {
129                 alertManager.showAlert("No client is selected.", "Please select a client.");
130             } else {
131                 alertManager.showAlert("No project is selected.", "Please select a project.");
132             }
133         });
134     }
135 
136 }
```

Figure 18: Method which creates an empty task for logging time.

3.4.2 Design Patterns/Principles

The **Façade pattern** was implemented to decouple the layers and hide the implementation of the subsystem.

The **Singleton pattern** was implemented in the ModelCreator as it must hold state.

The **Factory design pattern** was implemented for model creation. The ModelCreator is responsible for dynamic creation of models when they are needed. The ModelCreator holds dependency which will be injected to models **(dependency injection)**. Because of the ModelCreator, the application uses a single instance of the BllFacade which is passed to the models. The Façade is the only gateway between GUI and BLL in the application, so no unnecessary facades are created.

Dependency injection was also implemented in the controllers. All controllers are dependent on the models.

The **Model Façade** was dropped because every TaskItemController must call methods on its own TaskModel and close contact is necessary. The Development Team is considering bringing back the ModelFaçade during refactoring to lower coupling between controllers and models.

3.5 Sprint Review

The Sprint Review occurred on May 07, 2020. Due to the current novel coronavirus (2019-nCoV) outbreak, the meeting was held on Zoom with the Scrum Team and key stakeholders in attendance. The name and roles of the attendees are shown in Table 7.

Scrum Team	**Name**
Product Owner	Jeppe Moritz Led
Scrum Master	Anne Luong
Development Team	Radoslav Backovsky Anne Luong
Stakeholders	**Name**
Customer	Jakob Grumsen
Teacher	Stig Salskov Iversen

Table 7: Attendees at the Sprint 1 Review.

The Scrum Master opened the meeting by introducing the Development Team and the prepared agenda.

1. Demonstration of the features from the Increment.
2. Discussion of the Product Backlog.
3. Feedback of prototypes (if time permits).

As opposed to the procedure described in the *Scrum Guide* [1, p. 13], *Radoslav Backovsky* from the Development Team explained the status of the user stories in the Sprint Backlog during the demonstration instead of the Product Owner. The Development Team also answered questions from *Jakob Grumsen* about the features built in the Increment.

Figure 19 shows the Sprint Backlog and the status of the user stories at the completion of the Sprint. All the user stories were done except the user story “As a user, I should be able to view all my tasks.”. It was discovered a few hours before the meeting that there was a bug, where the refresh was not working properly. When a task was created, it would not always appear. Except this bug, the Increment went well. The Development Team received praise for the implemented features.

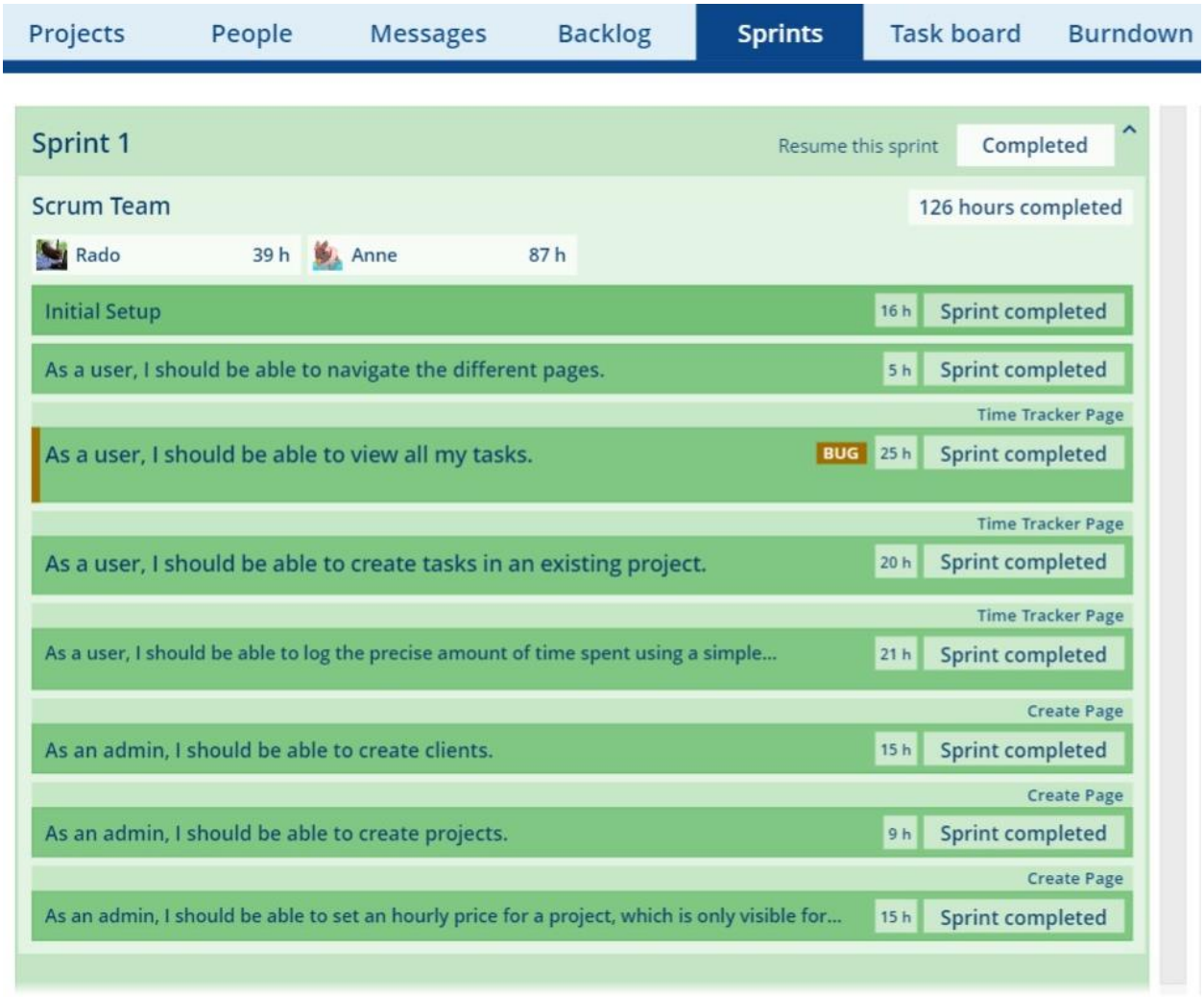


Figure 19: The Sprint Backlog at the completion of Sprint 1.

Next on the agenda was the discussion about the Product Backlog. The Development Team needed feedback on the current Product Backlog to plan the next Sprint. The proposed Product Backlog was well-received regarding user stories and prioritization. It was agreed that the number one priority would be to fix the bug in the next Sprint. *Jakob Grumsen* also emphasized that all tasks should be billable by default during task creation.

Lastly, the Development Team asked *Jakob Grumsen* to give feedback on a prototype for the feature regarding tabular overview. It was immensely helpful as the Development Team's prototype was different from *Jakob Grumsen*'s wishes. A rough prototype was created in *Balsamiq Wireframes* (see Figure 20).

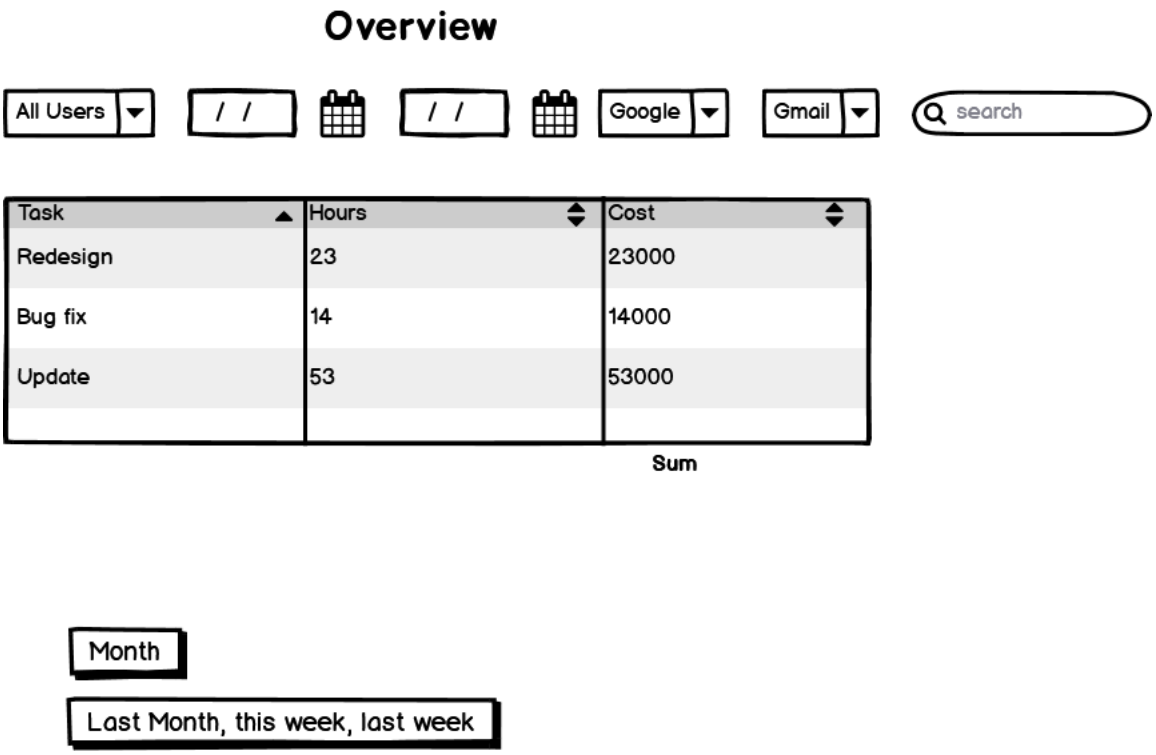


Figure 20: The *Balsamiq Wireframes* prototype for the Overview.

There was not enough time to look at *Scrumwise* together, but *Stig Salskov Iversen* had a look. He praised the Development Team's use of the tool and the Burndown chart (Figure 21) looked great.

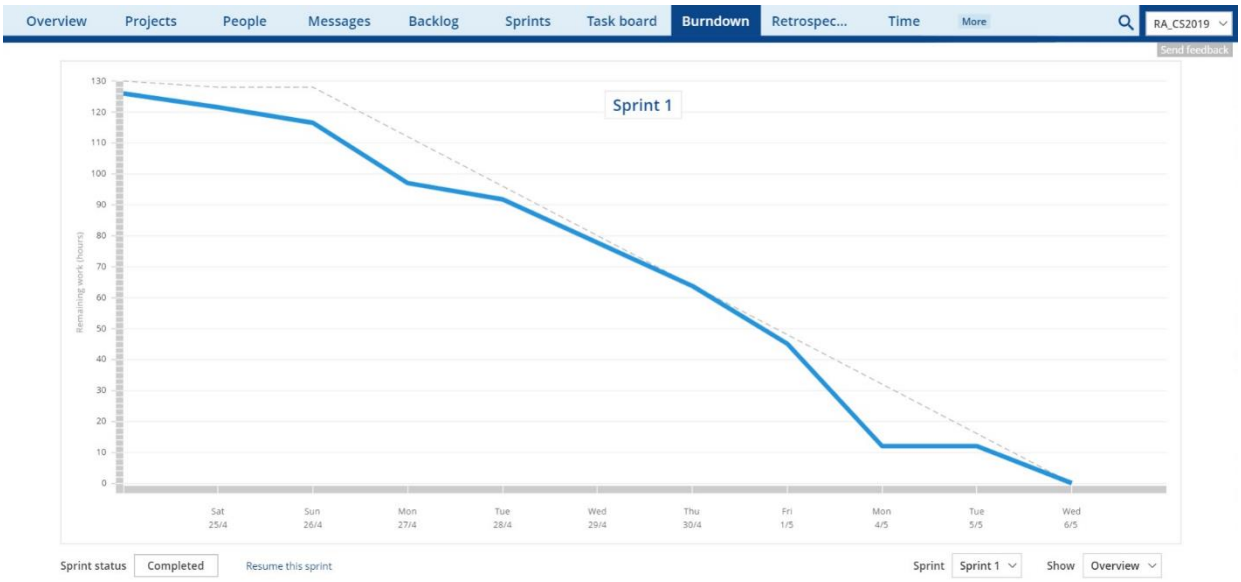


Figure 21: Sprint 1 Burndown chart after the Sprint.

3.6 Sprint Retrospective

The Development Team had a Sprint Retrospective on May 07, 2020 to reflect on the Sprint. The Retrospective was held on *Zoom* and *Scrumwise* was used to facilitate the event. Each member of the Development Team got time to reflect on the three areas:

1) What went well?
2) What went wrong?
3) What should we do?

When each person was done reflecting, cards were made to share the reflections (see Figure 22). It was agreed that the Development Team would continue to do the observations which went well and try to improve on and eliminate the bad.

The following changes were made immediately:

- Daily Scrum was moved from 9:00 AM to 10:00 AM as it would suit one of the team members better.
- The Development Team wrapped up the Sprint by organizing documentation and writing notes for the report.

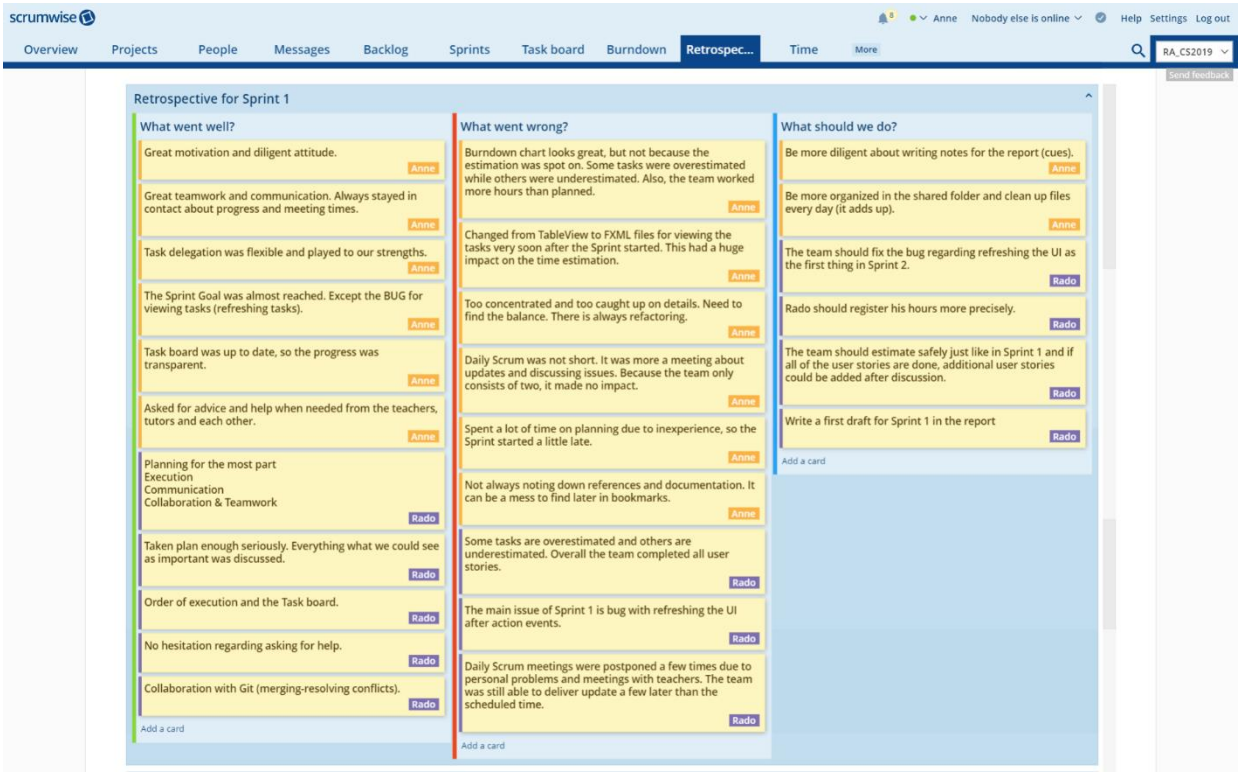


Figure 22: Sprint 1 Retrospective.

4 Sprint 2

4.1 Sprint planning

The planning involved the following phases:

Creation of the Sprint Goal

The prioritization of the remaining Backlog items was already decided at the Sprint 1 Review. By looking at the highest ranked user stories, the Sprint Goal was defined as “Fix the refresh bug of the Time Tracking Page and get the Overview Page ready for release.”

Decomposition of the user stories into tasks

The Scrum Master created tasks for the user stories, which would partake in the upcoming Sprint.

Time estimation of the tasks and finalizing the Sprint Backlog

The Development Team agreed to continue working 8 hours a day with one rest day during the weekend. The Sprint had seven full-working days and one half-working day (half of the day was spent on Sprint planning), so the available hours were 120 hours.

The tasks in each user story were estimated until there was enough work for the Sprint.

The resulting Sprint Backlog is shown in Figure 23.

Not all user stories were assigned to this Sprint, so the Development Team were expecting one more Sprint.

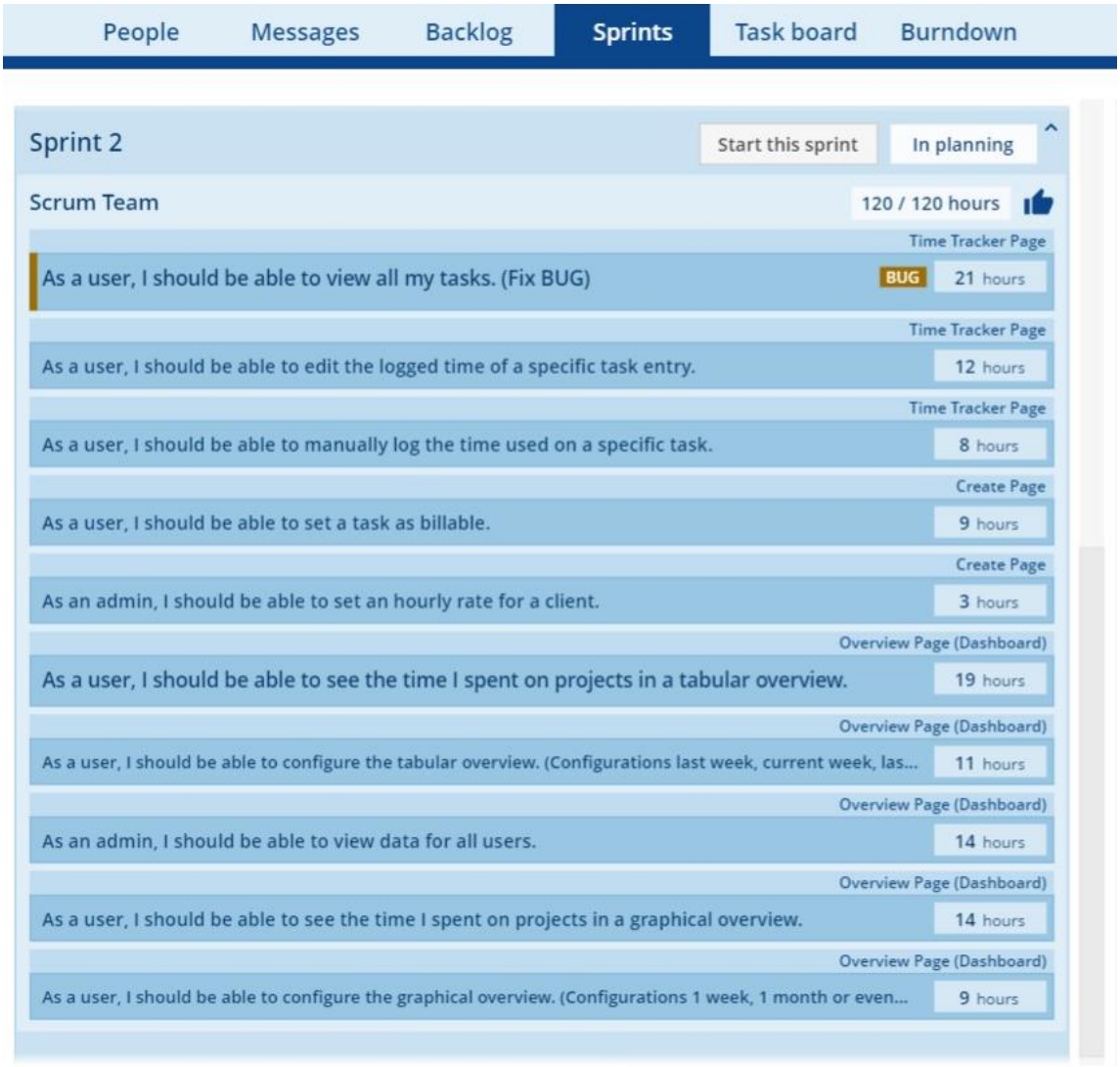


Figure 23: Sprint 2 Backlog before starting the Sprint.

4.2 GUI (including UI-design patterns)

A *Filter Bar* design pattern was used for the filters on the Overview Page. The advantages of this pattern are easy access during user scrolling and the compact size compared to a left filter panel. [12]

Right click?

Figure 24: UI design at the end of Sprint 2.

4.3 Data model

The changes to the data model are explained in the timeline below:

May 10, 2020

It was necessary to distinguish the billability of each individual tasks, so the attribute *billability* was added to the *Task* relation. The attribute is a foreign key attribute which can only contain values matching the values from the newly introduced *BillabilityLookup* relation. The *BillabilityLookup* relation is a workaround as *Microsoft SQL Server Management Studio* does not have the enum domain.

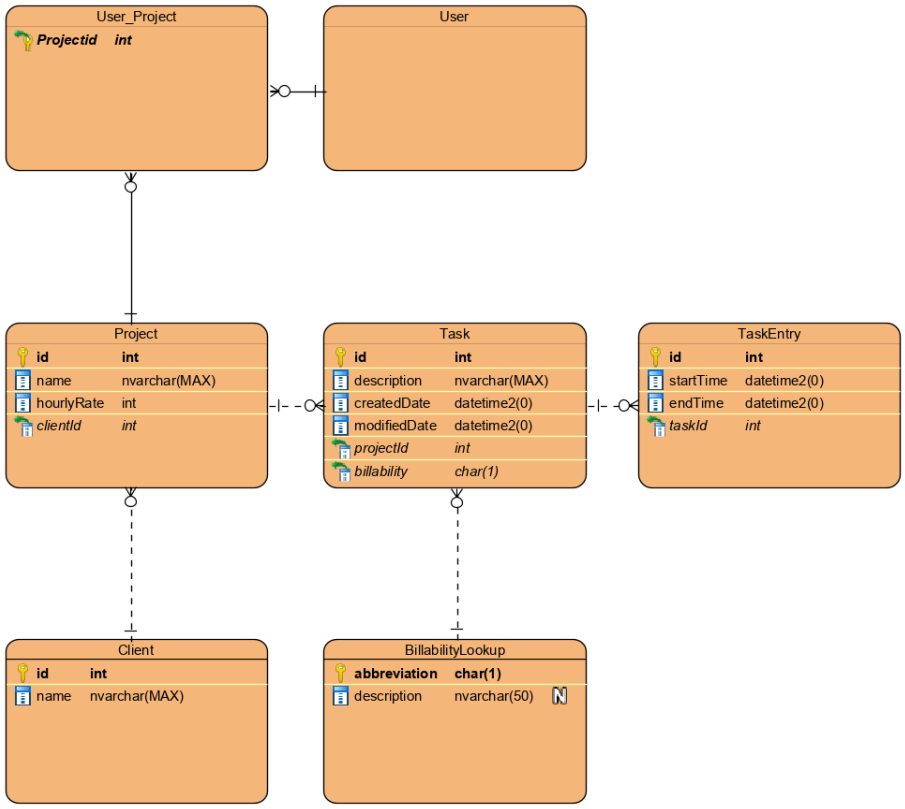


Figure 25: Physical ERD from May 10, 2020. The attribute *billability* was added to the *Task* relation and the *BillabilityLookup* relation was created.

May 13, 2020

Users were introduced in the system and the information related to users had to be stored in the database, so the *User* relation was created. A user can be either a standard user or an administrator. The type of user is indicated in the *userTypeId* attribute, which is a foreign key attribute related to the primary key in the *UserType* relation. The lookup table serves as an enumeration and enables extension for other types of users (e.g. moderator).

The *userId* attribute was added to the *Task* relation, so users would only be shown their own tasks in the Time Tracker Page.

The *hourlyRate* attribute was added to the *Client* relation to store a rate for the client.

Figure 26: Physical ERD from May 13, 2020. *User* and *UserType* relations were created in the database. A *userId* attribute was added to the *Task* relation and a *hourlyRate* attribute was added to the *Client* relation.

May 14, 2020

The JOIN table *User_Project* relation was no longer in consideration as *Jakob Grumsen* no longer wanted to assign projects to users. The relation was never created in the database, but it was added to the ERD for overview purposes.

The primary key attribute *id* for the *UserType* relation was changed from char(1) to int. The change was made as int would be more flexible for extension. Before the value STANDARD and ADMINISTRATOR would be “S” and “A” respectively. If another type of user which had the either “S” or “A” as the first letter, it would cause confusion.

Figure 27: Physical ERD from May 14, 2020. The JOIN table *User_Project* relation was no longer needed as the functionality was dropped. The domain of the *id* attribute of the *UserType* relation was changed to int.

4.4 Implementation

Sprint 2 main features were about logging the time manually (Figure 28) on a task and editing time of specific task entry inside a task (Figure 29 and 30). The team decided to try implementing time picker component from JFoenix library. The attempt was successful but details about the Task were not updated because of lack of knowledge about observer pattern,

The team was struggling with refreshing UI in the first sprint and had to bring the bug to the second sprint. UI was loading too slow and sometimes new created task would not appear after refresh. The team failed with caching individual nodes of tasks and reflecting updates of task entries on task item. The team did not continue to fix the bugs because of time constrains and priorities.

4.4.1 Code examples

```
193      private void addTask() {
194          btnAdd.setOnAction((event) -> {
195              if (validateInput()) {
196                  if (!manualMode) {
197                      try {
198                          TaskConcrete1 newTask = makeTask();
199                          mainModel.addTask(newTask);
200                          timeTrackerContr.initializeView();
201                      } catch (ModelException ex) {
202                          alertManager.showAlert("Could not create the task.", "An error occured: " + ex.getMessage());
203                      }
204                  } else {
205                      try {
206                          TaskConcrete1 newTask = makeTask();
207                          TaskConcrete1 newTaskWithEntry = makeTaskEntry(newTask);
208                          mainModel.addTask(newTaskWithEntry);
209                          timeTrackerContr.initializeView();
210                      } catch (ModelException ex) {
211                          alertManager.showAlert("Could not create the task.", "An error occured: " + ex.getMessage());
212                      }
213                  }
214              }
215          });
216      }
```

Figure 28: Creation of task expanded with manual logging in Controller.

```
84      @FXML
85      private void handleEditStartTime(Event event) throws ModelException {
86          TaskEntry taskEntry = taskEntryModel.getTaskEntry();
87          LocalTime updatedStartTime = timePickerStartTime.getValue();
88          LocalDate entryDate = taskEntry.getStartTime().toLocalDate();
89          if (updatedStartTime.isBefore(taskEntry.getEndTime().toLocalTime())) {
90              try {
91                  taskEntry.setStartTime(LocalDateTime.of(entryDate, updatedStartTime));
92                  taskEntryModel.updateTaskEntryStartTime(taskEntry);
93
94              } catch (ModelException ex) {
95                  alertManager.showAlert("An error occured", "Check your internet connection." + ex.getMessage());
96              }
97
98          } else {
99              Platform.runLater(() -> {
100                 alertManager.showAlert("Invalid input", "Start time has to be before end time");
101                 timePickerStartTime.show();
102
103             });
104         }
105
106     }
```

Figure 29: Editing start time of task entry.

```
139     @Override
140     public void setupStartTimeListener() {
141         startTime.addListener((ObservableValue<? extends LocalTime> observable, LocalTime oldValue, LocalTime newValue) -> {
142             if (newValue.isBefore(taskEntry.getEndTime().toLocalTime())) {
143                 LocalDateTime startTimeLDT = LocalDateTime.of(getDate(), newValue);
144                 taskEntry.setStartTime(startTimeLDT);
145                 stringDuration.set(secToFormat(calculateDuration(taskEntry).toSeconds()));
146             }
147         });
148     }
```

Figure 30: Setting up listener for start time property inside the model that is bound to time picker inside the controller.

4.4.2 Design Patterns/Principles

The **built-in observer pattern** was implemented as a result of using properties and observable lists with combinations of bindings and/or listeners. The usage of these throughout the application help with updating UI when changes occur.

4.5 Sprint Review

The Sprint Review was held on May 18, 2020 on *Zoom* with the Scrum Team and key stakeholders in attendance. The attendees are shown in Table 8.

Scrum Team	**Name**
Product Owner	Jeppe Moritz Led
Scrum Master	Anne Luong
Development Team	Radoslav Backovsky Anne Luong
Stakeholders	**Name**
Customer	Jakob Grumsen
Teacher	Stig Salskov Iversen

Table 8: Attendees at the Sprint 2 Review.

The agenda for the meeting was:

1. Demonstration of the features from the Increment.
2. Discussion of the Product Backlog.

The Development Team started the meeting by presenting the features of the Increment. *Jakob Grumsen* was generally pleased, but requested some improvements and changes for the upcoming Sprint:

- The bugs should be fixed before the release.
- The time notation should be 24-hour notation.
- The date pickers are hard to see, so a suitable color should be used instead.
- The active filter buttons would benefit from adding an “x” to indicate the filter removal.
- There should be a feature where the user can see the information of a task column when hovering on it. The information includes the task description, project and hours.
- The standard user should not be able to see the cost.
- The administrator should be able to see an overview of all clients.
- The administrator should be able to see an overview of all projects.

The Product Backlog prepared by the Development Team was shown and the other attendees agreed with the remaining items. Additional items representing the requests by *Jakob Grumsen* would be added as well.

Jeppe Moritz Led wrapped up the meeting by giving general feedback for this project. He praised the Development Team for the software presented in terms of visual design and functionalities. He was especially impressed due to the team consisting of two people. The Development Team's preparation for the Review meetings (agenda) and attentiveness to *Jakob Grumsen*'s requests were also appreciated. Following up on the instructions from the last Review was well-received as it gives the customer a sense of respect and aids collaboration.

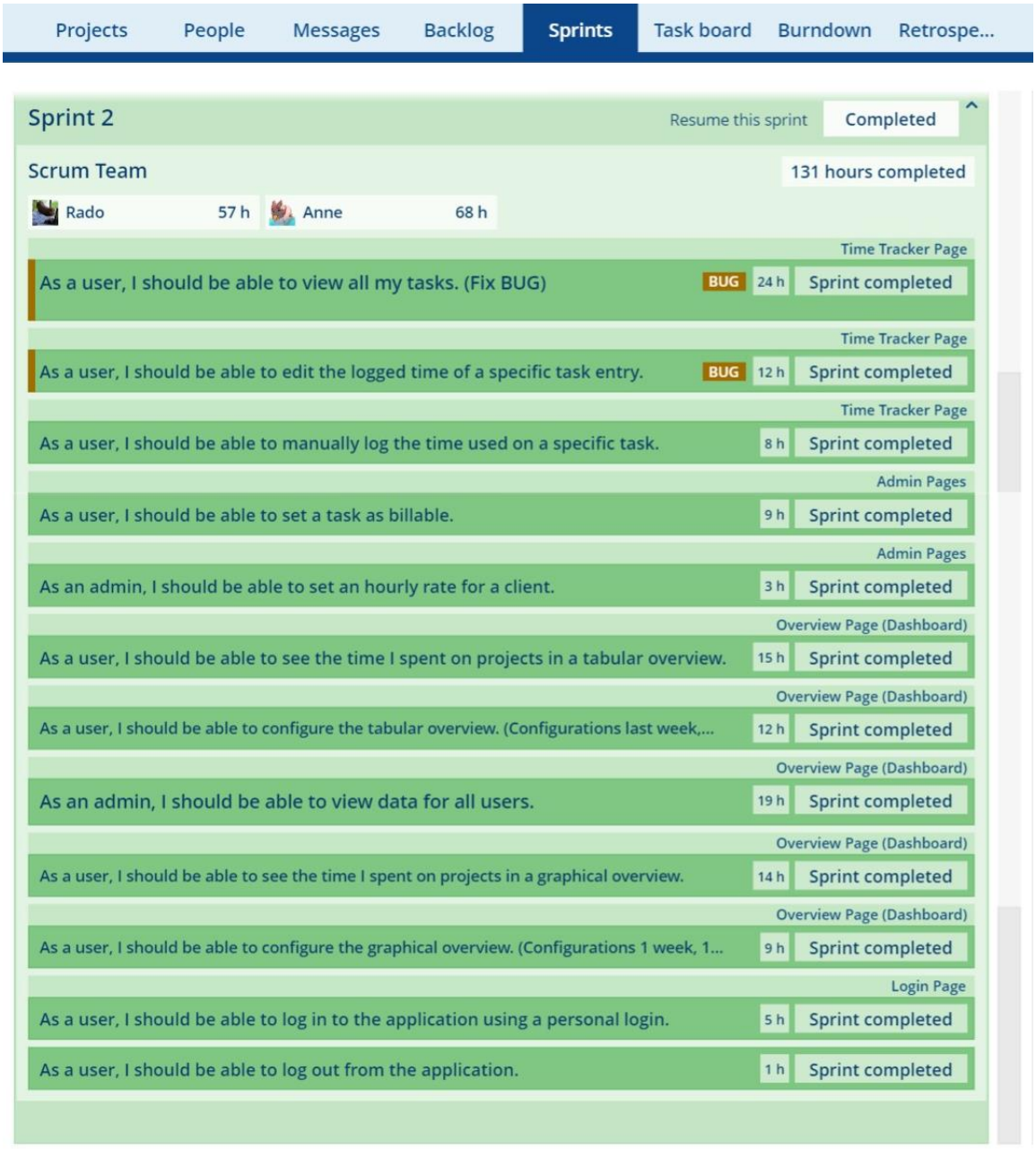


Figure 31: The Sprint Backlog at the completion of Sprint 2.

4.6 Sprint Retrospective

The Development Team held a Sprint Retrospective on May 18, 2020 to reflect on the Sprint. *Scrumwise* was used to facilitate the event again. Each member of the Development Team reflected individually and shared these reflections afterwards. The reflections can be seen in Figure 32.

Generally, the Development Team performed well despite a decrease in motivation mid-Sprint. The team was aware of the time limitations due to the team size and worked more than planned. The two user stories “As a user, I should be able to log in to the application using a personal login.” and ”As a user, I should be able to log out from the application.” were completed despite not being a part of the initial Sprint Backlog. The initial Sprint Backlog appears in Figure 23, while the final Sprint Backlog can be seen in Figure 31.

The observations and reflections from this Sprint were brought to the next Sprint.

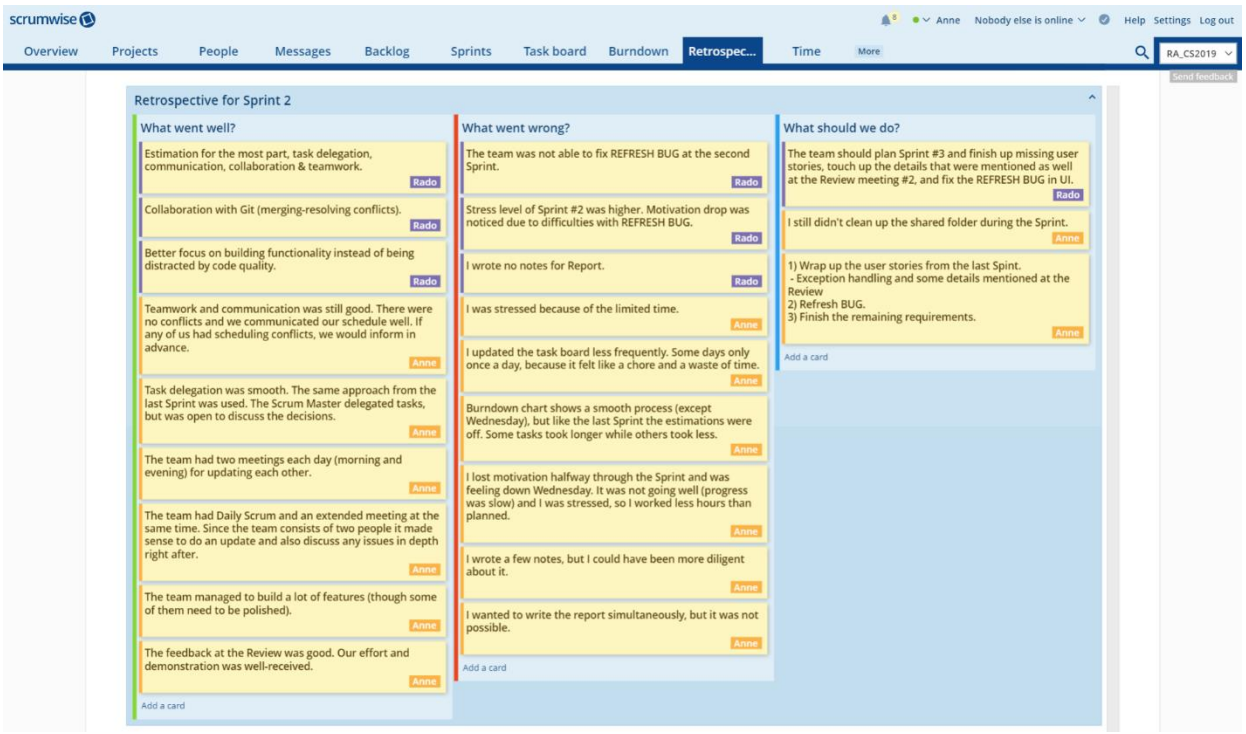


Figure 32: Sprint 2 Retrospective.

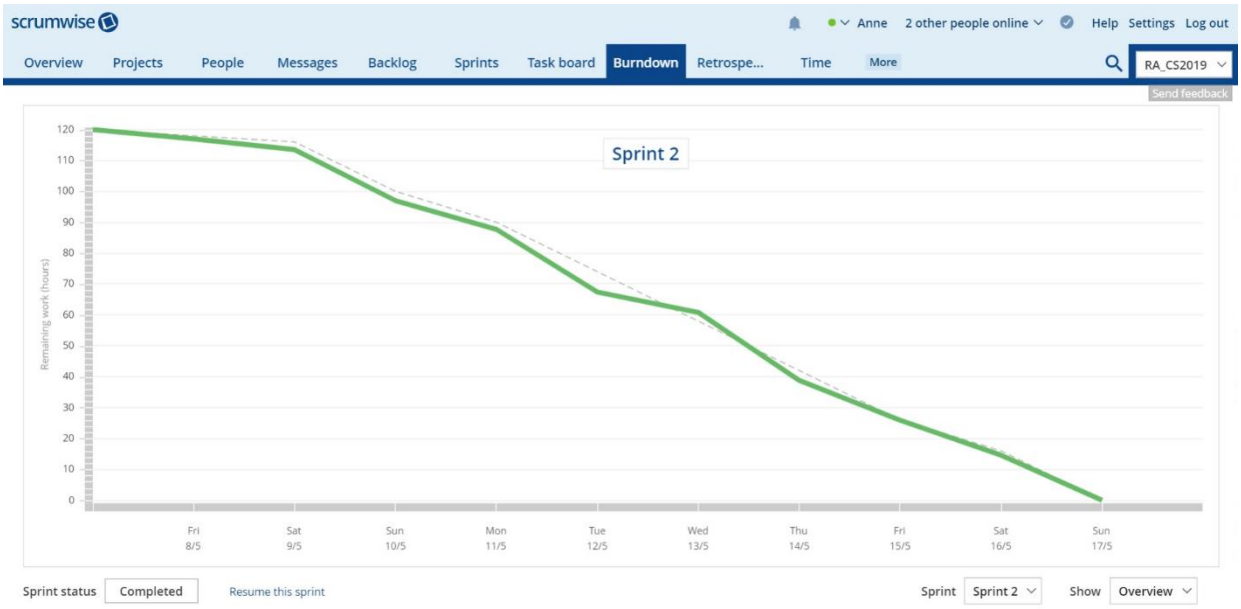


Figure 33: Sprint 2 Burndown chart after the Sprint.

5 Sprint 3

5.1 Sprint planning

The planning involved the following phases:

Creation of the Sprint Goal

The Sprint Goal was defined as “Finish up all the remaining features an requested changes, so the application is ready for release.”

Decomposition of the user stories into tasks

The Scrum Master created tasks for the remaining user stories.

Time estimation of the tasks and finalizing the Sprint Backlog

The Development Team decided to allot 60 hours of work for the final Sprint. The Sprint would take place from May 26, 2020 to May 29, 2020. The required number of works hours each day was decided to be more flexible as the report would be written simultaneously. The team was also prepared to extend the Sprint as some tasks might be challenging and other unaccounted for tasks would appear.

The tasks in each user story were estimated and the resulting Sprint Backlog is shown in Figure 34 below.

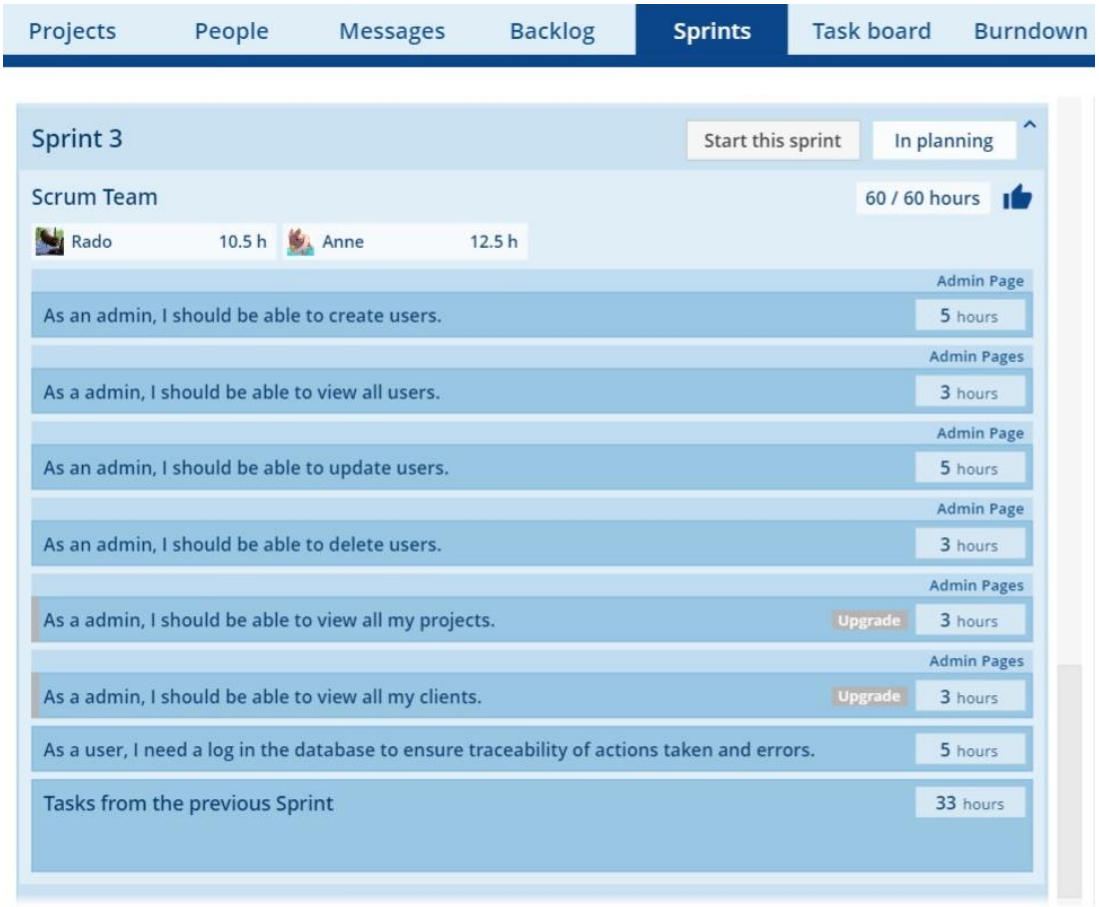


Figure 34: Sprint 3 Backlog before starting the Sprint.

5.2 GUI (including UI-design patterns)

The functionality of showing information of a task column in the bar chart when hovering over it was implemented using tooltips. The column changes color to indicate it is currently hovered on.

5.3 Data model

The changes to the data model are explained in the timeline below:

May 19, 2020

A new functionality to track actions and errors in an event log had to be implemented, so the *EventLog* relation was created in the database.

Figure 35: Physical ERD from May 19, 2020. *EventLog* relation created in the database.

May 24, 2020

A new functionality to delete users had to be implemented. Only users without tasks will be truly deleted from the database. Users with tasks will not be deleted, but their status will be changed from "ACTIVE" to "INACTIVE". The status of a user is recorded in the *statusId* attribute of the *User* relation, which is a foreign key. The foreign key is related to the primary key of the *StatusLookup* relation.

Figure 36: Physical ERD from May 24, 2020. Added a *statusId* attribute to *User* relation and created a *StatusLookup* relation in the database.

May 26, 2020

The *username* attribute of the *EventLog* relation was changed to *ipAddress*. This decision was made due to the following reasons:

1) If the username had to be stored, the logging would have to take place in the controller or a parameter containing the username would be necessary. If the event is logged in the controller, there could be a discrepancy between UI and the database. If the username was provided as a parameter, it would introduce overhead which is not worth it.
2) Recording the IP address is adequate for the purpose of the event log. In the case of errors, it is only necessary to know which computer the error happened on. There is no danger of hacking, so the username does not need to be recorded.

Figure 37: Physical ERD from May 26, 2020. Changed the *username* attribute to *ipAddress* in the *EventLog* relation.

May 28, 2020

A UNIQUE constraint was placed on the *username* attribute of the *User* relation (see Figure 38). This will ensure that all usernames are unique. The SQL exception with the code 2627 will occur if an administrator tries to make a user with an existing username (see Figure 39).

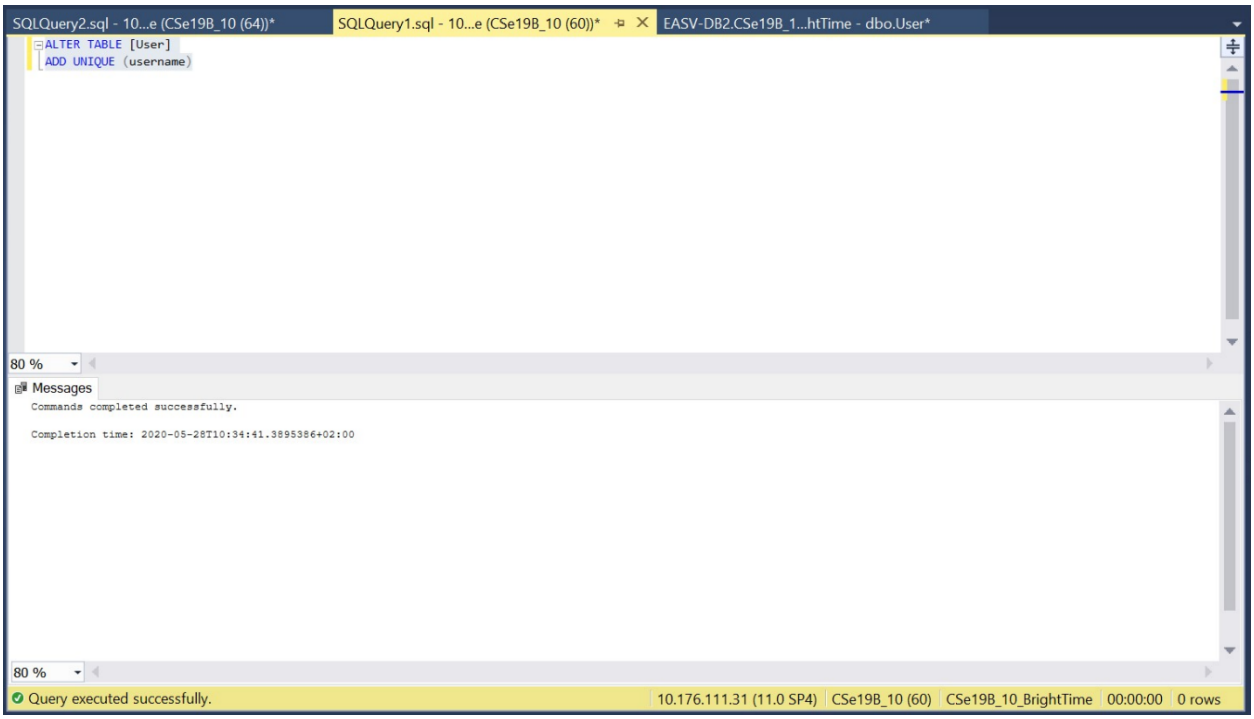


Figure 38: SQL statement to place UNIQUE constraint on the *username* attribute.

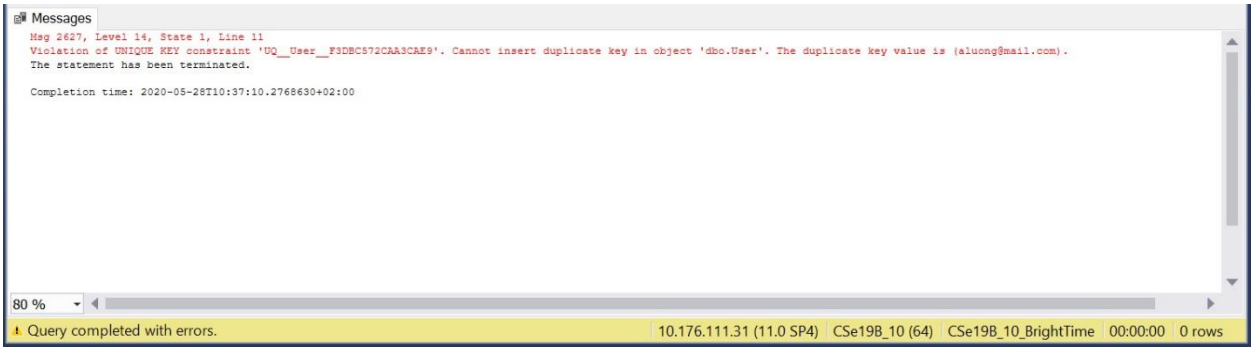


Figure 39: Error message for violation of the UNIQUE constraint.

June 01, 2020

A view was created to provide an extra layer of abstraction and hiding the complex SQL statement shown in Figure 40.

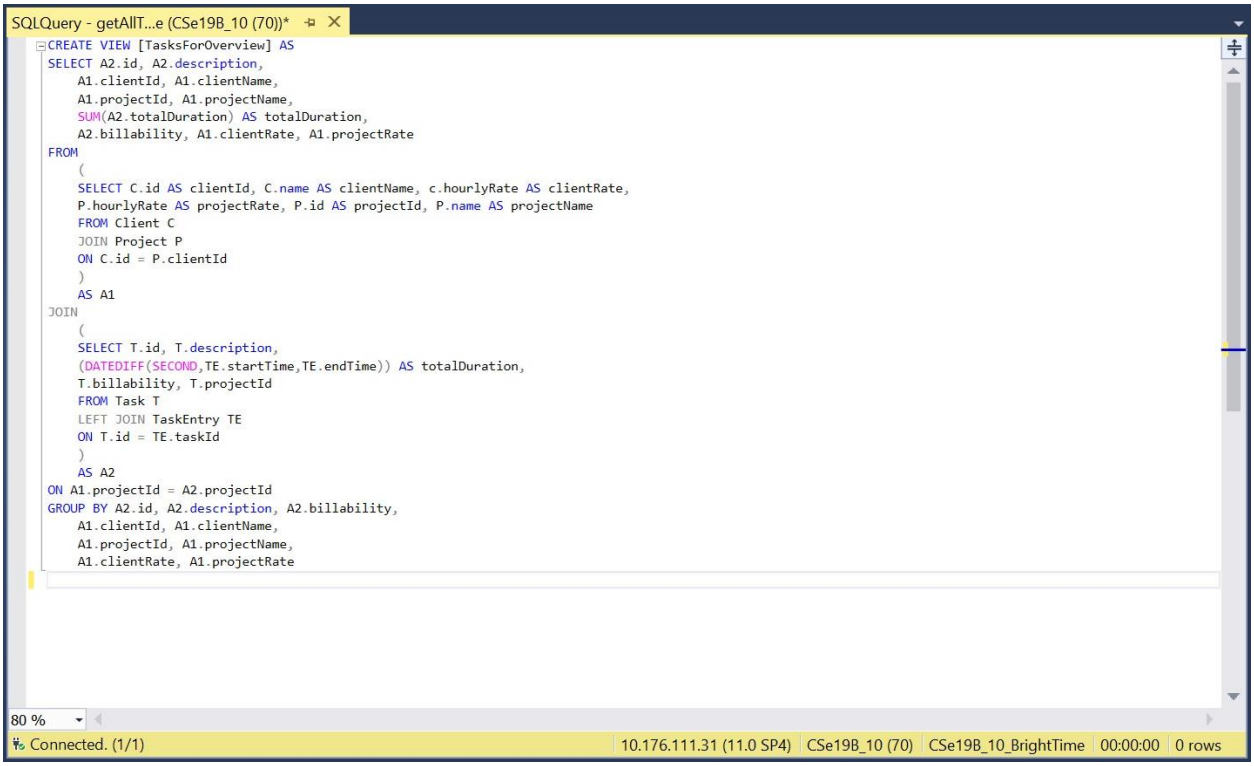


Figure 40: Creation of a view which is used in the getAllTasks() method in the TaskDAO.

5.4 Implementation

After the Sprint 2 review meeting the team received tutoring from Nedas Šurkus and managed to establish local caching of **Tasks** and **Tasks entries** (Figure 41). Listener attached to the task map that was responsible for initializing the tasks was crashing the program after re-entering the time tracker scene (Figure 42). The very next day the team received support from project's product owner Jeppe Moritz Led. Initialing tasks from local data after the change occurred on task map was fixed (Figure 43).

The problem of slow loading tasks on UI was still alive. The team was thinking that database is slow, but it was all wrong. The real reason is loading custom FXML files is slow. The team received a suggestion from the product owner that creating view in code would solve this problem but there was no time left for recreating the view. Instead the team came up with a simple solution of loading less FXML files based on filter that is predefined for the current week. Another possible improvement contains "Load more" button that will load another chunk of information and implementation of proper Observer design pattern that will fetch latest updates for specific node.

During Sprint 3 the team fixed issues of task item details being outdated after task entry updates. Task Model was refactored to set the day entry list instead of filtering it with getter. Day entry list is observable list with listener attached. Observable list is returning array list of predefined properties of task entry so the list can send notification to the listener not only when the list changes but also when the content of list changes. After the listener will update task details.

In the Sprint3 the Model Façade was not brought back.

Possible solution: Model Facade would have to be responsible for making sure Task Item Controller calls method on its own Task Model. After creating Task Model inside the Time Tracker Controller, the team would save Task Model as value on Task Item Controller key in hash map. The hash map would be placed in Model Manager so every method can pick the correct Task Model (every method called on Model Manager has also in parameter current controller so the key can be known). This is an idea that is not tested at all so it can be wrong.

The reason why the team could not eliminate coupling in GUI was basically time constrains. The team had to implement features and for design fixing there was no time left.

The team was also not able to fix the bug about updating client. When a user wants to update a client name or hourly rate, the old value is not set if the update went wrong.

5.4.1 Code examples

```
104     @Override
105     public void addTask(TaskConcrete1 task) throws ModelException {
106         try {
107             TaskConcrete1 freshTask = bllManager.createTask(task);
108             List<TaskEntry> entryList = new ArrayList();
109             freshTask.setTaskEntryList(entryList);
110             List<TaskConcrete1> taskList = taskMap.get(freshTask.getCreationTime().toLocalDate());
111             if (taskList == null) {
112                 taskList = new ArrayList<>();
113                 taskList.add(freshTask);
114                 taskMap.put(freshTask.getCreationTime().toLocalDate(), taskList);
115             } else {
116                 taskList.add(0, freshTask);
117                 taskMap.remove(freshTask.getCreationTime().toLocalDate());
118                 taskMap.put(freshTask.getCreationTime().toLocalDate(), taskList);
119             }
120         } catch (BllException ex) {
121             throw new ModelException(ex.getMessage());
122         }
123     }
```

Figure 41: Creating task in database and locally in Main Model.

```
100      private void setUpTaskMapListener() {
101          taskMapListener = (MapChangeListener.Change<? extends LocalDate, ? extends List<TaskConcrete1>> change) -> {
102              initTasks();
103              System.out.println("init tasks");
104          };
105          mainModel.addTaskMapListener(taskMapListener);
106
107      }
```

Figure 42: Setting up listener.

Command putAll on observable hash map is a loop of put commands. That means listener attached to the observable hash map is initializing tasks after each put command. That's why the team is forced to apply a workaround of temporally removing listener from map when the map is getting new data from database. This must be done because listener is set up when the user enters time tracker scene and is still alive when the user exits the scene so the "Singleton" for listener inside the Main Model is necessary (Figure 44).

```
132      @Override
133      public void loadTasks(User user) throws ModelException {
134          try {
135              Map<LocalDate, List<TaskConcrete1>> allTasks = bllManager.getAllTasksWithEntries(user);
136              //  temp removal
137              taskMap.removeListener(taskMapListener);
138
139              taskMap.clear();
140              taskMap.putAll(allTasks);
141
142              taskMap.addListener(taskMapListener);
143
144          } catch (BllException ex) {
145              throw new ModelException(ex.getMessage());
146          }
147      }
```

Figure 43: Loading tasks with task entries from database into HashMap data structure.

```
199      @Override
200      public void addTaskMapListener(MapChangeListener<LocalDate, List<TaskConcrete1>> taskMapListener) {
201          if (this.taskMapListener != null) {
202              taskMap.removeListener(this.taskMapListener);
203          }
204          this.taskMapListener = taskMapListener;
205          taskMap.addListener(taskMapListener);
206      }
207  }
```

Figure 44: Making sure the only one listener is attached to the task map.

5.5 Sprint Retrospective

The Development Team held a Sprint Retrospective on June 01, 2020 to reflect on the Sprint. The final Retrospective was a casual conversation instead of using the *Scrumwise* feature. The team agreed that the last Sprint was very rough as the report was written simultaneously.

The Sprint was extended several times, because the team discovered new things to correct while testing the application during report writing. Figure 45 shows the Sprint Backlog at the end of the Sprint and Figure 46 shows the Burndown chart. Both show that more work was added. It was very stressful, but not much could be done as the team did not want to hand in faulty software.

Unfortunately, the team was not able to finish all the remaining features and details.

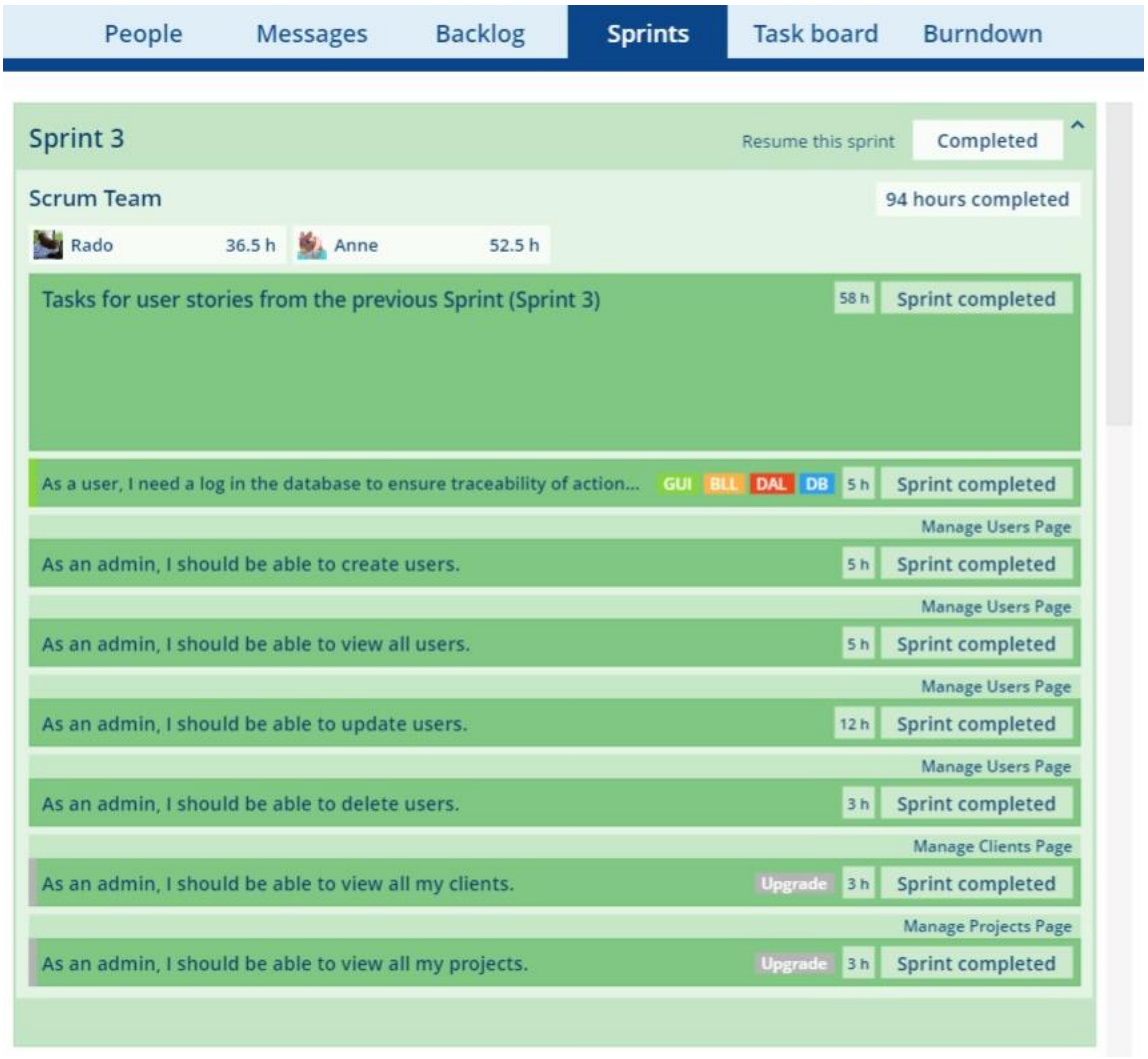


Figure 45: Sprint 3 Backlog after the Sprint.

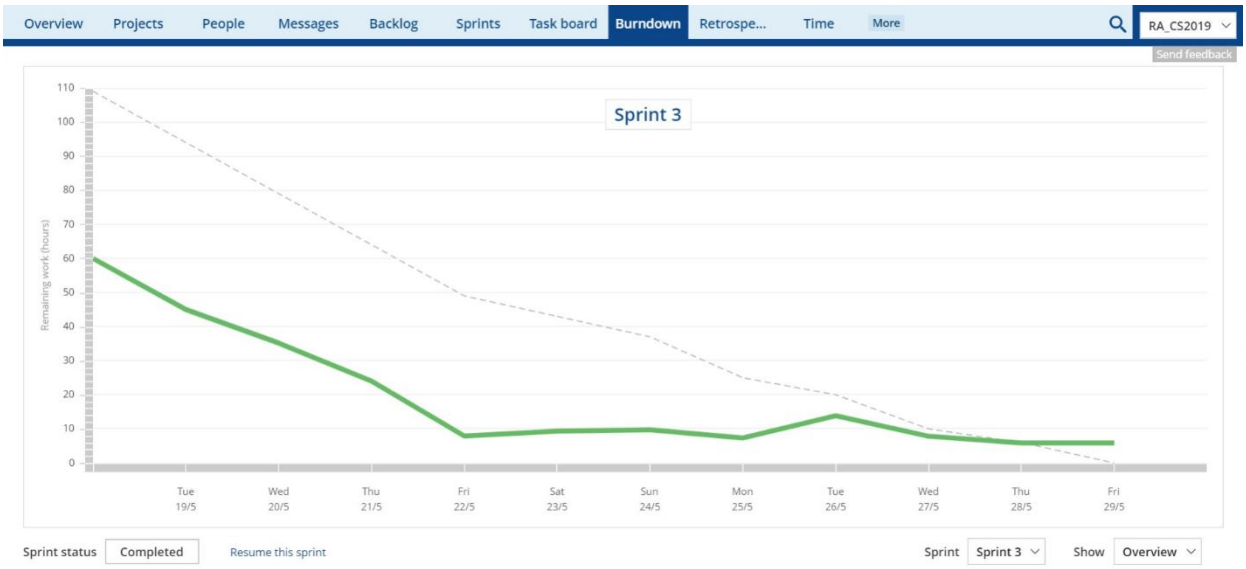


Figure 46: Sprint 3 Burndown chart after the Sprint.

6 Unit test

The TaskDurationCalculator class was tested as it is an essential class for the UI. The method calculateTaskDuration() is called when a change occurs inside the List<TaskEntry> instance variable of a Task. The task duration reflects the sum of the task entry durations.

```
44      public void testCalculateTaskDuration() {
45          List<TaskEntry> entryList = new ArrayList<>();
46          User user = new User(1, "John", "Colins", "j@gmail.com", User.UserType.STANDARD);
47          Client client = new Client(1, "Nike", 1000);
48          Project project = new Project(1, "Nike Air Max 7", client, 0);
49          TaskConcrete1 task = new TaskConcrete1(14, "update material", project, TaskBase.Billability.BILLABLE, entryList, LocalDateTime.now(), user);
50          entryList.add(new TaskEntry(task, LocalDateTime.parse("2020-05-05T09:00:00"), LocalDateTime.parse("2020-05-05T09:30:00")));
51          entryList.add(new TaskEntry(task, LocalDateTime.parse("2020-05-05T15:00:00"), LocalDateTime.parse("2020-05-05T16:00:00")));
52          entryList.add(new TaskEntry(task, LocalDateTime.parse("2020-05-05T15:00:00"), LocalDateTime.parse("2020-05-05T16:00:00")));
53          entryList.add(new TaskEntry(task, LocalDateTime.parse("2020-05-05T18:00:00"), LocalDateTime.parse("2020-05-05T20:00:00")));
54          TaskDurationCalculator instance = new TaskDurationCalculator();
55          Duration expResult = Duration.ofSeconds(16200);
56          Duration result = instance.calculateTaskDuration(entryList);
57          assertEquals(expResult, result);
58      }
59
60      @Test
61      public void testCalculateTaskDuration2() {
62          List<TaskEntry> entryList = new ArrayList<>();
63          User user = new User(2, "Mike", "Wazowski", "m@gmail.com", User.UserType.STANDARD);
64          Client client = new Client(1, "Nike", 1000);
65          Project project = new Project(1, "Nike Air Max 7", client, 0);
66          TaskConcrete1 task = new TaskConcrete1(15, "update color", project, TaskBase.Billability.BILLABLE, entryList, LocalDateTime.now(), user);
67          entryList.add(new TaskEntry(task, LocalDateTime.parse("2020-05-05T09:00:00"), LocalDateTime.parse("2020-05-05T11:30:00")));
68          entryList.add(new TaskEntry(task, LocalDateTime.parse("2020-05-05T15:00:00"), LocalDateTime.parse("2020-05-05T15:30:00")));
69          entryList.add(new TaskEntry(task, LocalDateTime.parse("2020-05-05T15:00:00"), LocalDateTime.parse("2020-05-05T16:00:00")));
70          entryList.add(new TaskEntry(task, LocalDateTime.parse("2020-05-05T18:00:00"), LocalDateTime.parse("2020-05-05T22:00:00")));
71          TaskDurationCalculator instance = new TaskDurationCalculator();
72          Duration expResult = Duration.ofSeconds(28800);
73          Duration result = instance.calculateTaskDuration(entryList);
74          assertEquals(expResult, result);
75      }
76
```

```
Test Results
BrightTime
Tests passed: 100.00 %
Both tests passed. (0.368 s)
brighttime.bll.util.TaskDurationCalculatorTest passed
testCalculateTaskDuration passed (0.1 s)
testCalculateTaskDuration2 passed (0.002 s)
calculateTaskDuration
calculateTaskDuration
```

Figure 47: Unit test of the TaskDurationCalculator class.

7 Multi-threading and Concurrency

The Development Team was able to improve the performance of task creation with the support of the Product Owner. The first iteration concluded Thread that was saving task into database, but the team had problems with catching the exceptions. The Thread is not able to return anything that's why the team must use the Future interface. As the last thing was the replacement of the Thread to ExecutorServervice that submits the Callable task which returns the result as a Future. This way the team can get result from the Future (bright time task), catch exceptions and throw new Model exception. This approach is not used in other parts because of time constrains.

```
129     @Override
130     public void addTask(TaskConcrete1 task) throws ModelException {
131         ExecutorService executorService = Executors.newSingleThreadExecutor();
132         Future<TaskConcrete1> future = executorService.submit(() -> bllManager.createTask(task));
133         executorService.shutdown();
134         try {
135             addLocally(future.get());
136         } catch (InterruptedException ex) {
137             throw new ModelException(ex.getMessage());
138         } catch (ExecutionException ex) {
139             throw new ModelException(ex.getMessage());
140         }
141     }
```

Figure 48: Implementation of Executor Service and Future.

8 Conclusion

The bright time project was a great challenge for the Development Team and the team learned a lot throughout the process. All the minimum requirements were accomplished. The team worked hard thanks to great collaboration and motivation. The guidance and support received from lecturers and tutors of the academy had a huge impact on the result.

9 References

[1] K. Schwaber and J. Sutherland, "Scrum Guide," *ScrumGuides.org*, Nov, 2017. [Online]. Available: https://www.scrumguides.org/scrum-guide.html. [Accessed: May 28, 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 31, 2020].

[3] "Guide to Website Navigation Design Patterns," *WebFX*, n.d. [Online]. Available: https://www.webfx.com/blog/web-design/navigation-design-patterns/. [Accessed: May 01, 2020].

[4] "Input Prompt," *UI-Patterns.com*, n.d. [Online]. Available: https://ui-patterns.com/patterns/InputPrompt. [Accessed: Apr 21, 2020].

[5] "Input Feedback," *UI-Patterns.com*, n.d. [Online]. Available: https://ui-patterns.com/patterns/InputFeedback. [Accessed: Apr 21, 2020].

[6] "What a Relational Database Is," *Oracle*, n.d. [Online]. Available: https://www.oracle.com/database/what-is-a-relational-database/. [Accessed: Apr 24, 2020].

[7] "What is Entity Relationship Diagram (ERD)?," *Visual Paradigm*, n.d. [Online]. Available: https://www.visual-paradigm.com/guide/data-modeling/what-is-entity-relationship-diagram/#erd-data-models-physical. [Accessed: Apr 24, 2020].

[8] "Entity-Relationship Diagram Symbols and Notation," *Lucidchart*, n.d. [Online]. Available: https://www.lucidchart.com/pages/ER-diagram-symbols-and-meaning#section_2. [Accessed: Apr 24, 2020].

[9] Ian, "datetime vs datetime2 in SQL Server: What's the Difference?," *Database.Guide*, Jul. 29, 2019. [Online]. Available: https://database.guide/datetime-vs-datetime2-in-sql-server-whats-the-difference/. [Accessed: Apr 27, 2020].

[10] "datetime2 (Transact-SQL)," *Microsoft*, Jul. 23, 2017. [Online]. Available: https://docs.microsoft.com/en-us/sql/t-sql/data-types/datetime2-transact-sql?view=sql-server-2017. [Accessed: Apr 27, 2020].

[11] "time (Transact-SQL)," *Microsoft*, Jun. 07, 2017. [Online]. Available: https://docs.microsoft.com/en-us/sql/t-sql/data-types/time-transact-sql?view=sql-server-ver15. [Accessed: Apr 28, 2020].

[12] "Why filter bars are better than left-hand filters," *UXM*, Jun. 29, 2015. [Online]. Available: http://www.uxforthemasses.com/filter-bars/. [Accessed: May 15, 2020].

10 Appendices

10.1 Appendix 1: Daily Scrum Three Questions

1. What did I do yesterday?
2. What will I do today?
3. What issues do I have?

10.2 Appendix 2: Team collaboration policies

Development Team Working Agreement

1. Be honest, responsible and respectful.
2. Don't be afraid to voice your opinion and ask for help.
3. Working schedule
   - The minimum working hours is 42 hours a week.
   - One day of the week is a non-working day.
4. Communicate progress and schedule
   - Attend the Daily Scrum at 9:00.
   - Attend the standup meeting about the report.
   - Update the working hours on the Scrumwise Task board daily.
   - Update the report as you work on your tasks.
   - Inform the team if you cannot attend a meeting at least 5 min before the scheduled time.
   - Use phone (iMessage/FaceTime) and Discord for daily communication.
5. Use the checklist below every day!
6. Participate in knowledge sharing session after sprint retrospective.

Daily Checklist

<u>General</u>

❖ Did you do any research?
   - Keep track of your references (links for webpages and online media) and the date accessed. Write clearly where each reference is used.

❖ Are you not attending Daily Scrum?
   - Inform the team at least 5 minutes before the scheduled time.
   - Inform the team of your status by posting the three questions on Discord.

❖ Are you done for today?
   - Record your working hours for all tasks on Scrumwise.
   - Write all the project related tasks you performed today in the OneDrive Word document for the Daily Log.

<u>Report</u>

❖ Is there something you think should be included in the report?
   - Write cues in the OneDrive Word document for the report.
   - Upload documentation on Discord and the OneDrive folder.

<u>Code</u>

❖ Are you starting on a new task/functionality?
   - Create and work in one GitHub branch.

❖ Are you ready to share your code?

- Always fill out the “Summary” and “Description” with clear information about the task. State any BUGs clearly. Once a task is completed and tested, merge the branch with the master branch. Test again and then push.

❖ Are you getting conflicts during a merge?

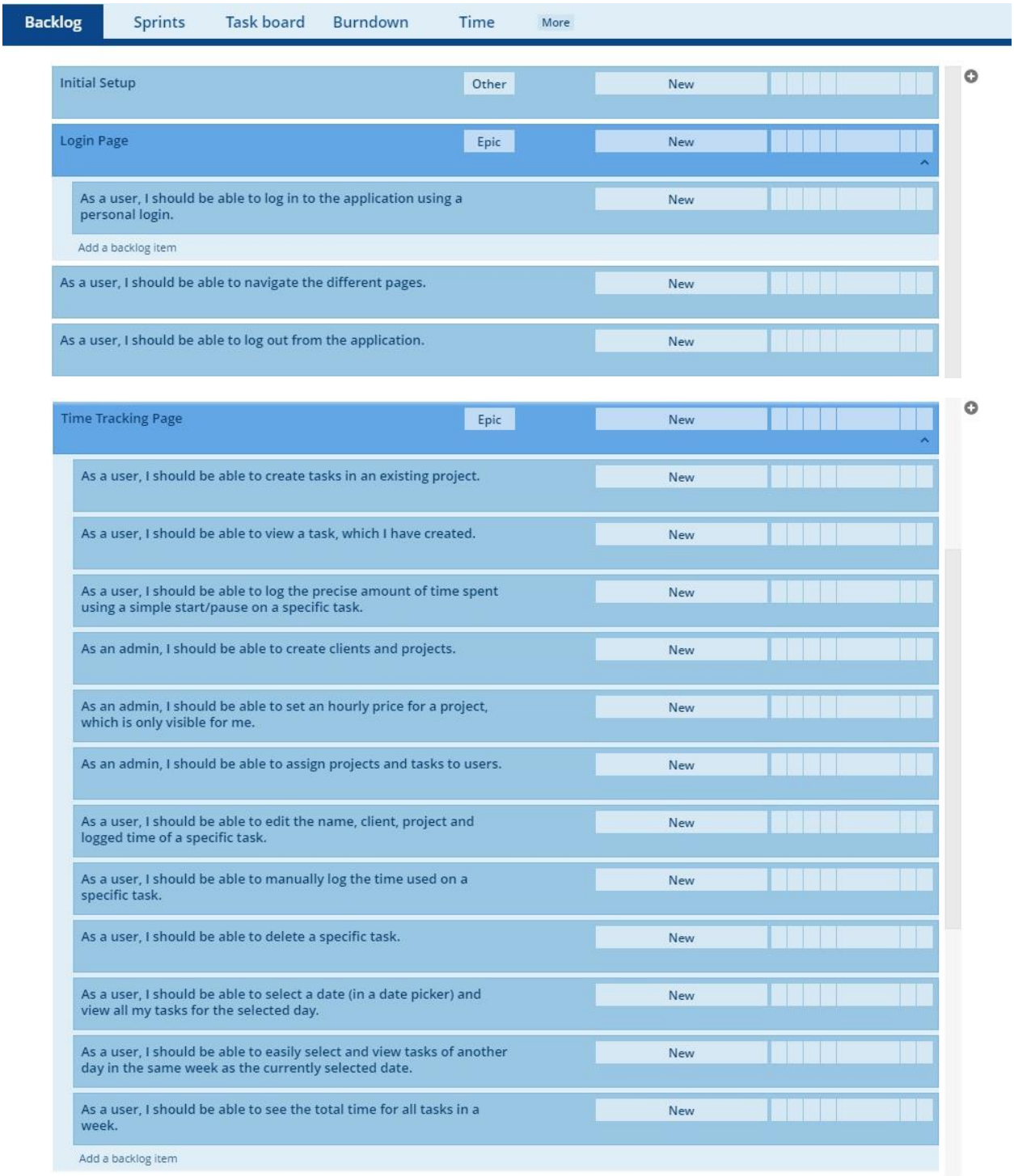
- Contact a group member and solve the conflict together. Don't edit another group member’s code. Things can go terribly wrong.

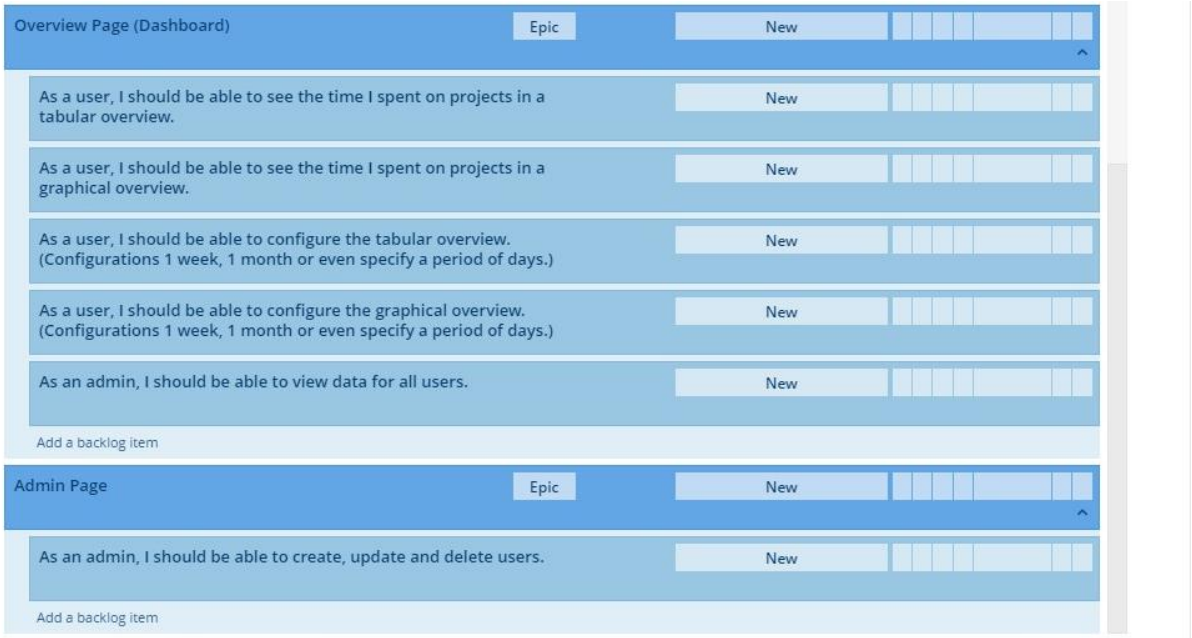
Definition of Done

1. The code has been tested functionally.
2. There are no bugs.
3. All exceptions are handled.
4. The code has been reviewed and tested by reviewer.
5. The code has been committed to the master branch.

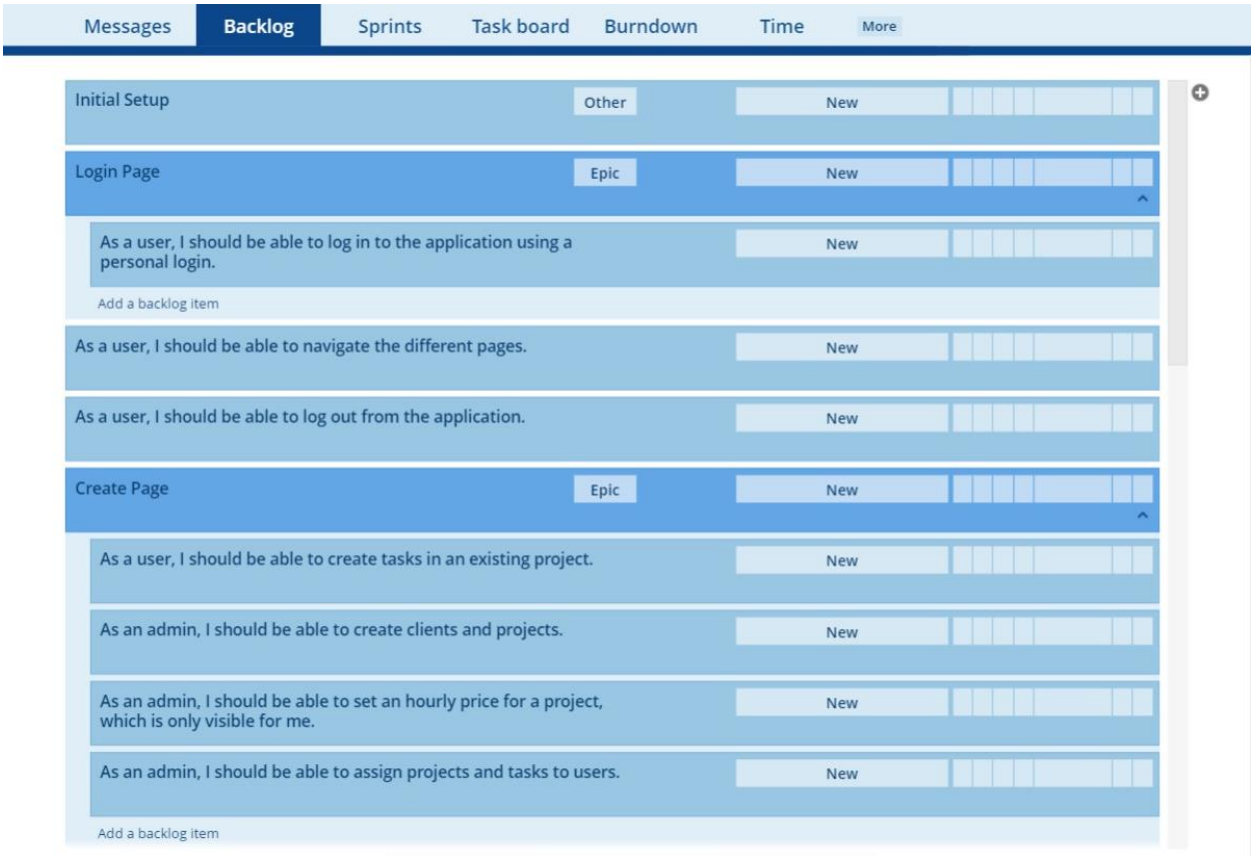
10.3 Appendix 3: Pre-Game Product Backlogs

The Product Backlog from April 21, 2020



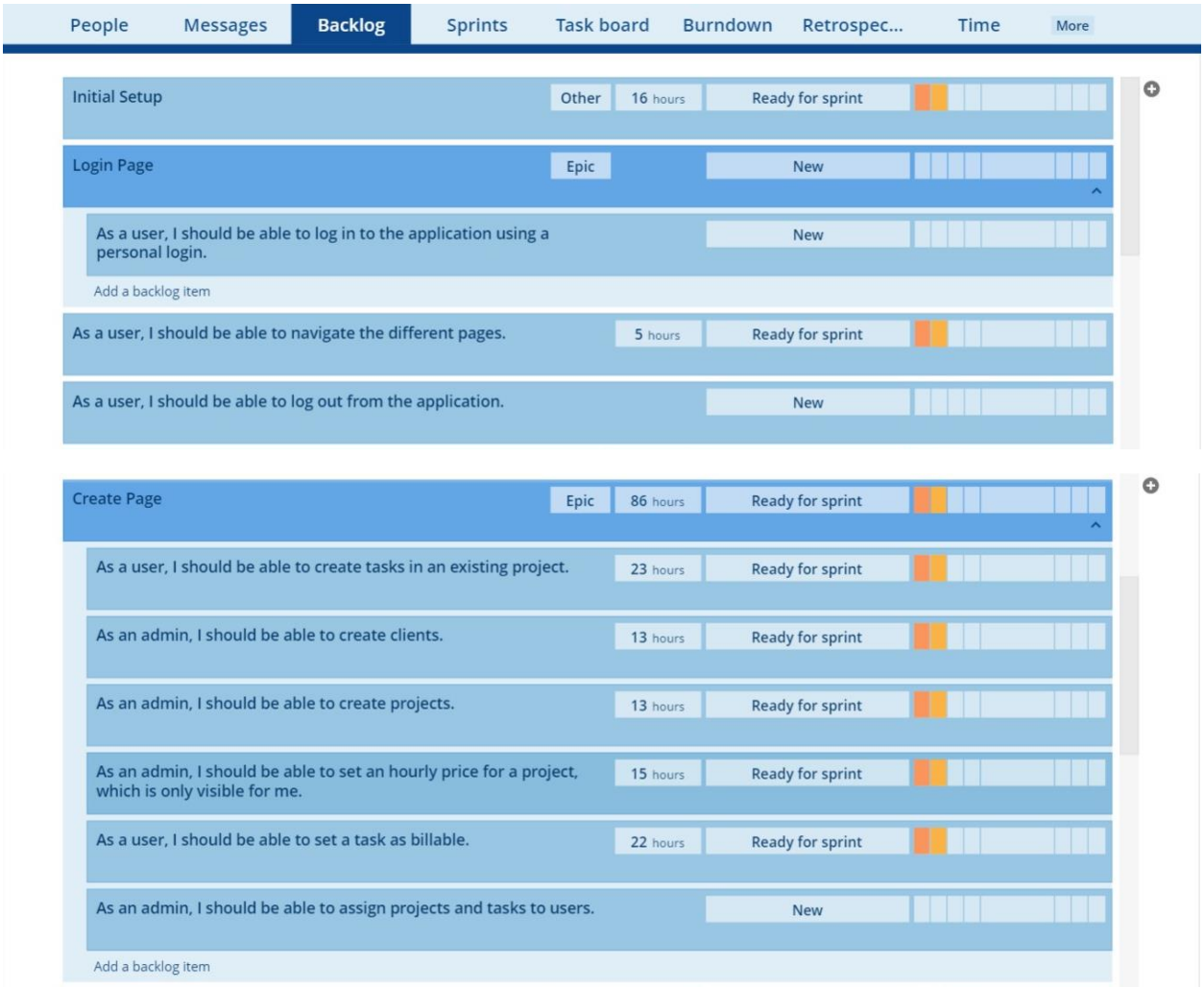


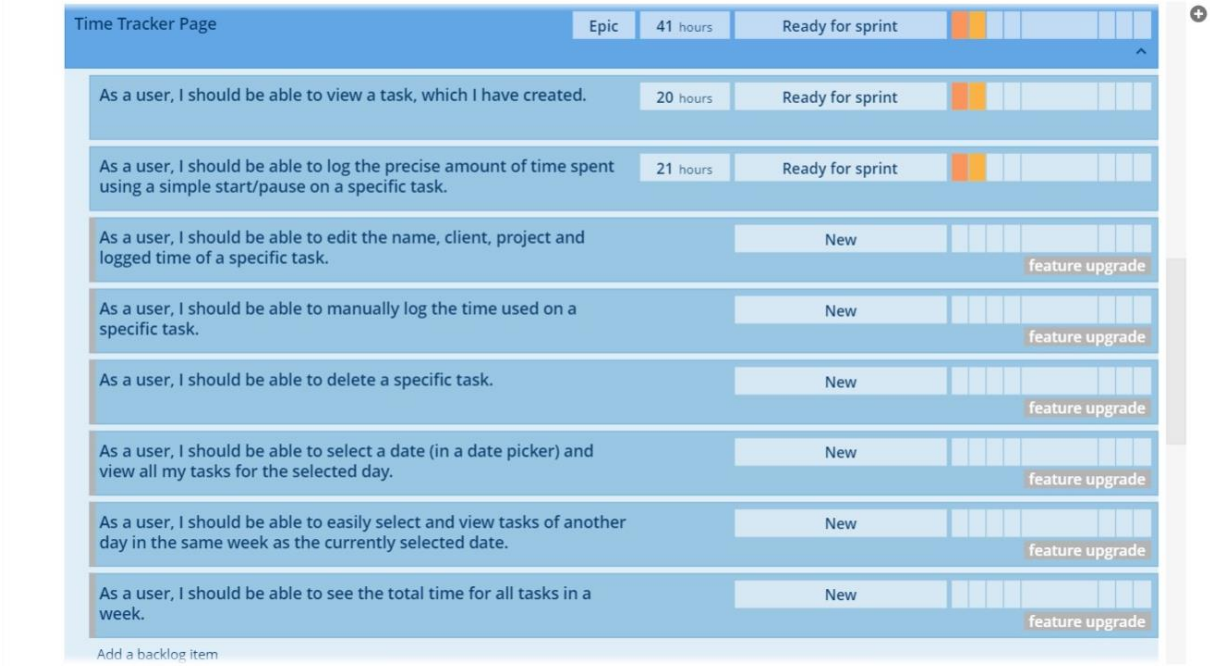
The Product Backlog from April 24, 2020

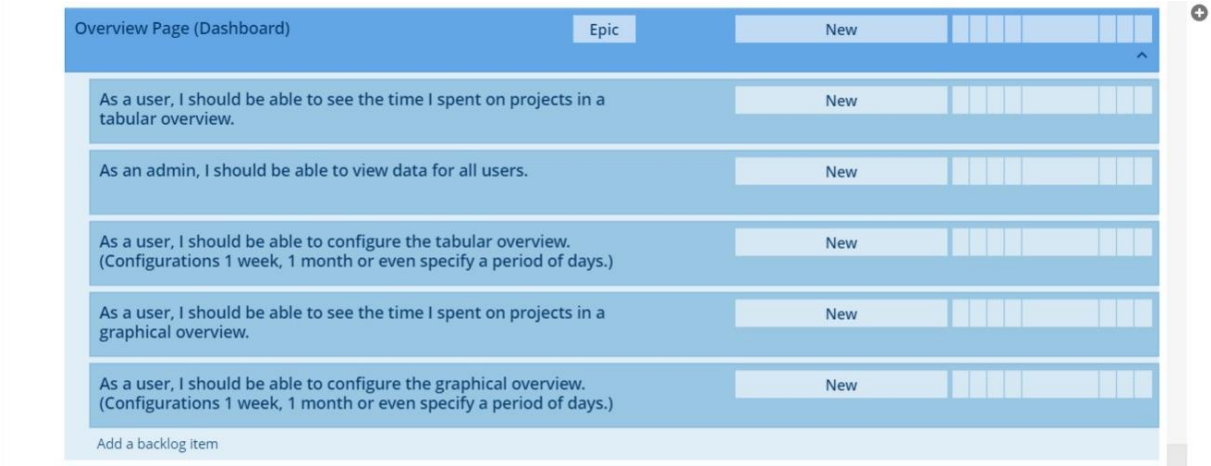


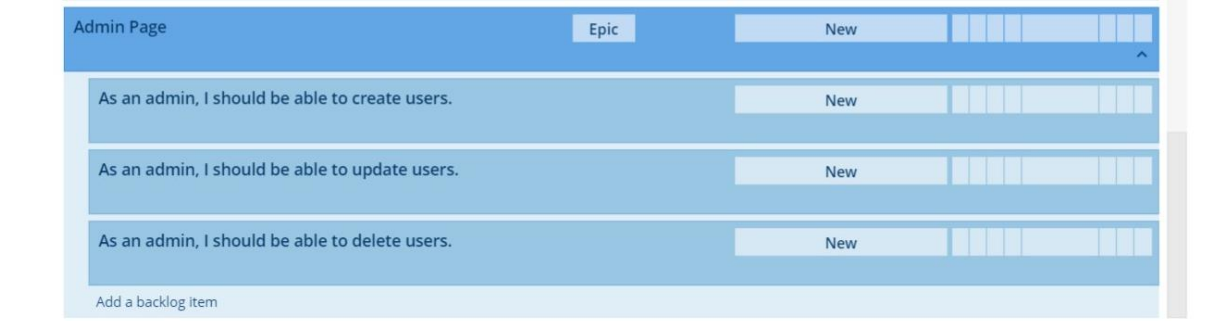

Time Tracking Page
Epic
New
As a user, I should be able to view a task, which I have created.
New
As a user, I should be able to log the precise amount of time spent using a simple start/pause on a specific task.
New
As a user, I should be able to edit the name, client, project and logged time of a specific task.
New
As a user, I should be able to manually log the time used on a specific task.
New
As a user, I should be able to delete a specific task.
New
As a user, I should be able to select a date (in a date picker) and view all my tasks for the selected day.
New
As a user, I should be able to easily select and view tasks of another day in the same week as the currently selected date.
New
As a user, I should be able to see the total time for all tasks in a week.
New
Add a backlog item
Overview Page (Dashboard)
Epic
New
As a user, I should be able to see the time I spent on projects in a tabular overview.
New
As a user, I should be able to see the time I spent on projects in a graphical overview.
New
As a user, I should be able to configure the tabular overview. (Configurations 1 week, 1 month or even specify a period of days.)
New
As a user, I should be able to configure the graphical overview. (Configurations 1 week, 1 month or even specify a period of days.)
New
As an admin, I should be able to view data for all users.
New
Add a backlog item
Admin Page
Epic
New
As an admin, I should be able to create, update and delete users.
New
Add a backlog item

The Product Backlog from April 25, 2020




Time Tracker Page
Epic
41 hours
Ready for sprint
As a user, I should be able to view a task, which I have created.
20 hours
Ready for sprint
As a user, I should be able to log the precise amount of time spent using a simple start/pause on a specific task.
21 hours
Ready for sprint
As a user, I should be able to edit the name, client, project and logged time of a specific task.
New
feature upgrade
As a user, I should be able to manually log the time used on a specific task.
New
feature upgrade
As a user, I should be able to delete a specific task.
New
feature upgrade
As a user, I should be able to select a date (in a date picker) and view all my tasks for the selected day.
New
feature upgrade
As a user, I should be able to easily select and view tasks of another day in the same week as the currently selected date.
New
feature upgrade
As a user, I should be able to see the total time for all tasks in a week.
New
feature upgrade
Add a backlog item


Overview Page (Dashboard)
Epic
New
As a user, I should be able to see the time I spent on projects in a tabular overview.
New
As an admin, I should be able to view data for all users.
New
As a user, I should be able to configure the tabular overview. (Configurations 1 week, 1 month or even specify a period of days.)
New
As a user, I should be able to see the time I spent on projects in a graphical overview.
New
As a user, I should be able to configure the graphical overview. (Configurations 1 week, 1 month or even specify a period of days.)
New
Add a backlog item

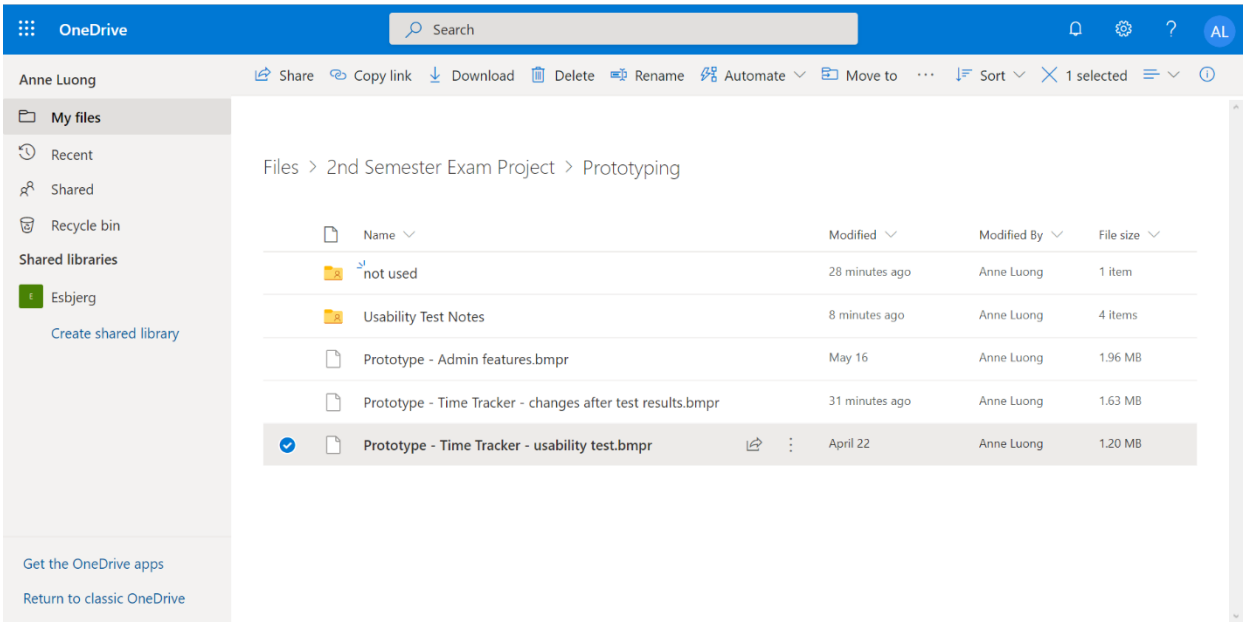

Admin Page
Epic
New
As an admin, I should be able to create users.
New
As an admin, I should be able to update users.
New
As an admin, I should be able to delete users.
New
Add a backlog item

10.4 Appendix 4: Prototypes

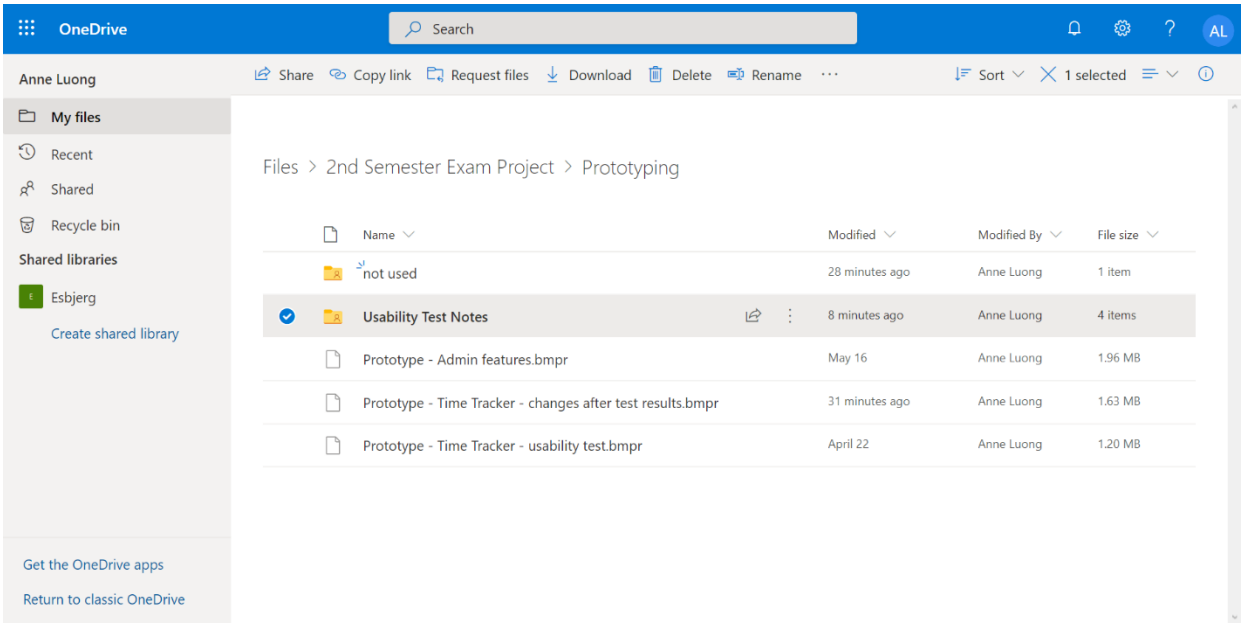
Prototypes were made throughout the project to visualize designs and perform tests. The following folder contains all the prototypes:

https://erhvervsakademisydvest-my.sharepoint.com/:f:/g/personal/anne2p31_easv365_dk/EpWRUA_VAqVKsZoP3WBQNa0B94L2JuYKkbMIRXCa08aJLA?e=Q8EEoq

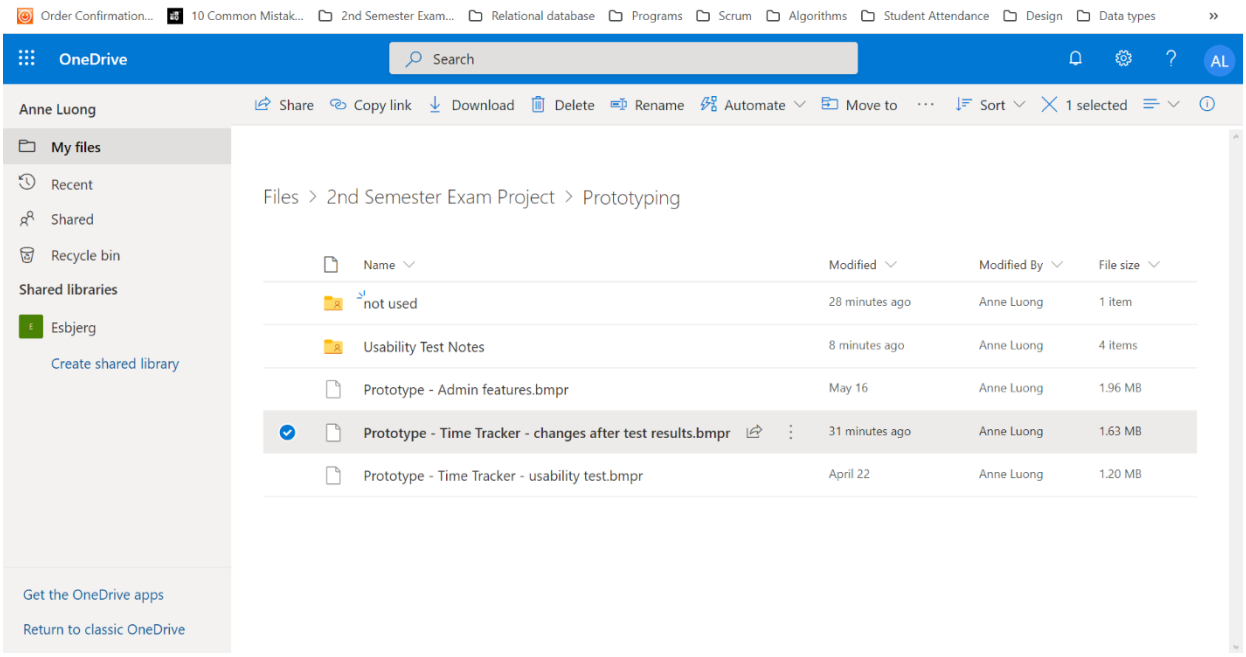
The selected file in the screenshot below is the Time Tracker prototype used for the preliminary usability tests:



The selected folder in the screenshot below contains the test results from the preliminary usability tests:



The selected file in the screenshot below contains the Time Tracker prototype with changes made from the test results as well as later prototypes.



The selected file in the screenshot below contains the prototypes for administrative features. No usability tests were performed, but the Development Team asked the teachers, friends and family for opinions and advice.