AARHUS UNIVERSITET

Preliminary Project for Bachelor Project

Spring 2020

Project title:	ORBIT Project 2: Backend Platform and Management Interface for OrbitBox		
Project number	*Project provider*	*Counselor*	*Assistant Counselor*
2020E43	ORBIT Lab / ASE	Brian Vestergaard Danielsen	Henrik Bitsch Kirk

Participant Nr.	*Student Nr.*	*Name*	*Signature*
Participant 1	201510924	Andreas Blaabjerg	
Participant 2	201611828	Christopher Bernold	

Introduction

With thorough thought and much consideration, it was firmly decided that working with new web technologies and doing so affiliated with a company, or organization would yield the greatest outcome, for a more solid future bachelor project. After researching the available options at hand, we received an opportunity to work with the Aarhus School of Engineering sub-branch ORBIT-Lab. ORBIT-Lab is an entrepreneurial tech hub that offers learning experiences and workspaces, with many experimental facilities. Each semester ORBIT Lab offers a variety of projects that can be conducted by students, as their bachelor projects. After browsing the project catalogue, we were intrigued by the ORBIT BOX platform and after a few meetings it was decided to affiliate with ORBIT-Lab, in creating a sub platform, for managing and sharing documents and other tools between teachers from Danish Gymnasiums all around. With this project, the intention is to create a type of system similar to the famous "code repository sharing" website, GitHub[1].

The cause of this project surfacing, is due to the fact that danish teachers currently do not have a platform for sharing Lab exercise documents with other teachers at other gymnasiums. Many of them have facebook groups, which can be quite counter intuitive. The system will consist of the following aspects, with the opportunity to evolve and adapt during the course of the project:

1) An integrated login system, such as Uni-login or similar.
2) Storage of documents, small files or ORBIT-Box exercise templates.
3) Ability to view publicly stored files and download these files.
4) Option to export files to chosen platforms.
5) Ability to open ORBIT-box templates directly from the management platform.

This document will explain technology considerations and other important aspects which will occur in the system.

Project Description from the Project-Catalogue

ORBIT Project 2: Backend Platform and Management Interface for OrbitBox

NB: Project can be done in English or Danish

Proposal version: 0.6

Abstract:

Technology has become such an integrated part of our daily lives that we do not think about it any longer—we just use it. The last decade has been defined by handheld devices, wearables, data collection and artificial intelligence. ORBIT Lab has been working on a concept for high schools where teachers can design assignments, present them to students and get live feedback while the students are working with them. NB: See ORBIT Project E2020-1 proposal for more details on ORBIT Box.

It needs to facilitate as many different workflows as possible: Real-time workflows where teachers can monitor progress while the students are working with assignments. Each institution handles student administration and teaching differently, so for this platform to become a success, it should be able to integrate with existing systems at the institutions, as well as providing the necessary infrastructure to institutions that wish to use the system, but do not have any integration requirements. The platform should with minimal configuration be deployable in a cloud context or locally on hardware provided by the institutions.

Challenges

- Build an extendable platform with well-defined and tested APIs
- Data compartmentalization—how do we ensure privacy, confidentiality and collaboration?
- Identify, design and implement features that can be configured and deployed in isolation.

Partners:

ORBIT Lab, Center for Computational Thinking and Design at AU

Scoping

In the following section, a rough estimate for the final products Functional- and Non-functional, user related, requirements will be described. To specify these requirements, the User Stories technique, FURPS+ will be employed. Further to prioritize these requirements in an importance based manner the MosCoW prioritization is employed. The aim of using these methods, is to better enable pinpointing the focus of the project, and scope the projects size and contents adequately.

FURPS+

FURPS+ is used to delimit the functionalities from the non-functional requirements and actively categorizes these, respectively, in Functionality for the functional requirements and Usability, Reliability, Performance and Supportability, in the Non-functional requirements. Further additions can be things such as Design Constraints, Implementation requirements, Interface requirements and Physical requirements.

MOSCOW

MoSCoW, otherwise known as MoSCoW prioritization or MoSCoW analysis. This element will help keep an easy and clear prioritizing/scoping of the specified requirements.
The four subdivisions are the following:

MoSCoW = **M**ust, **S**hould, **C**ould, **W**on't

The MoSCoW section is solely based on the user stories that are already defined for the project. These are subject to change during the project, as a proper MoSCoW is performed so a rotation occurs in some elements categorized subdivision. Eg. a could requirement turns to a should based on rescoping because a project is running on track and there's time and budget for extra implementation. Or a could becoming a won't because the outcome of implementing it turns out through analysis to be insignificant. This also goes for elements that can end up being deemed marginal changes, not implementing better performance/usage.

Requirements

User Stories

With the use of User Stories the project will be enabled to describe certain features in a simple manner, using a more relatable perspective. User Stories are often told in a specific way that tells the story from a user that wants to reach a certain property.
A story template follows this simple structure:

> As a <type of user>, I can/want <some goal> so that <some reason>. [2]

User Stories is a great tool for an addition to the different non-functional, user related, requirements in a project. User Stories therefore adds knowledge of a wanted feature, but without requirements as to how that feature is implemented.

EPIC 1 SAML BASED UNI-LOGIN & SECURITY

User Story #1 - Login

As a teacher I want to be able to login to the system using Uni-login, so it's easier to login.

Acceptance Criteria: Teachers can login with unilogin.

User Story #1 - Logout

As a teacher I want to be able to logout of my account when I am finished with my session.

Acceptance Criteria: Teachers can logout of the session.

EPIC 2 DATABASE CONFIGURATION

User Story #1 - Database modelling in server and ERD layout

As a teacher I want to be able to have my data saved securely, so I and only I can access it at a later time.

Acceptance Criteria: User data is stored in a cloud based database.

User Story #2 - Login persistence

As a teacher I want my login info to exist after leaving the webpage, so login will be easier when visiting the webpage next.

Acceptance Criteria: Any data can be fetched from a unilogin database at any given time.

User Story #2 - Account changes persistence

As a teacher I want changes to my account to be saved so I don't have to redo my changes every time I sign in.

Acceptance Criteria: Changes to profile is persisted to the profile database.

EPIC 3 GRAPHICAL USER INTERFACE

User Story #1 - Home Page

As a teacher I want to view a home page when first entering the webpage, so it's easier to have an overview of the webpage.

Acceptance Criteria: User is prompted with a home page when first entering the webpage.

User Story #2 - User Navigation

As a teacher I want to have an understandable way of navigating around the webpage, so things are easier to find on the webpage.

Acceptance Criteria: UI is easily navigable

User Story #3 - User Profile, Create

As a teacher, I want to be able to create my own userprofile, so people know it's me.

Acceptance Criteria: User can successfully create a new user.

User Story #4 - User Profile, View/Edit

As a teacher, I want to View/Edit my user profile settings through a profile management page, so my profile contains updated information.

Acceptance Criteria: User can view and manage profile settings on a profile management page.

User Story #5 - User Profile, Delete

As a teacher, I want to delete my userprofile, because I don't use the webpage anymore.

Acceptance Criteria: User can delete profile, on a profile management page.

User Story #6 - Upload content

As a teacher, I want to be able to upload my work to the platform, for others to view.

Acceptance Criteria: User can successfully upload a lesson's instructions on a course subject, for other users to view.

User Story #7 - View/edit own published content

As a teacher, I want to be able to view/edit my own work on the platform, so I can use it for teaching reasons.

Acceptance Criteria: When a user can successfully view another teachers', lesson's instructions on a course subject.

User Story #8 - View other users' content

As a teacher, I want to be able to view other teachers' work on the platform, so I can use it for inspirational reasons.

Acceptance Criteria: When a user can successfully view another teachers', lesson's instructions on a course subject.

User Story #9 - Fetch/Pull content

As a teacher, I want to be able to download another teacher's work from the platform, so it can be saved locally, for inspirational reasons, or to edit and use in my own class.

Acceptance Criteria: When a user can successfully download another teacher's lesson's instructions on a course subject.

User Story #10 - Searching for content

As a teacher, I want to be able to search for a specific subject, limiting the viewed material by selected search criteria, so it is both easier and faster finding what I'm looking for.

Acceptance Criteria: When a user can successfully find another teacher's lesson's instructions on a course subject, through a search function.

User Story #11 - Availability

As a teacher, I want to be able to access the webpage at any given time.

Acceptance Criteria: When the webpage's uptime is stable and running 24 hours a day 7 days a week.

User Story #12 - Rating

As a teacher, I want to be able to rate content based on a five star system.

Acceptance Criteria: Each content item contains a rating from one-five stars, where teachers can rate the content.

User Story #13 - Comment content

As a teacher, I want to be able to give comments to content.

Acceptance Criteria: Users can comment content which can be viewed by all users.

EPIC 4 SERVER OPERATIONS

User Story #1 - Uploading the right Content

As a teacher I want to be able to upload content, so others can draw inspiration from my work, or use/edit it themselves in their own classes.

Acceptance Criteria: Users can upload content of specified formats. It is also important to note here that users cannot upload any specific content and will be restricted to specific formats such as PNG/JPEG files, PDF, ORBIT-Box formats and so on.

User Story #2 - Exporting Uploaded Content

As a teacher I would like to be able export content that has been uploaded to the webpage, so the files can be stored locally.

Acceptance Criteria: Users can export uploaded content to formats like PDF.

User Story #3 - Export to ORBIT-Box

As a teacher I would like the options to export directly to ORBIT-Box.
Acceptance Criteria: Users can click an export to ORBIT-Box button where they will be routed to the ORBIT-Box opening the exported content.

MoSCoW

Must

- The system **must** support using UniLogin as an authentication option.
- The system **must** be able to store user information securely.
- The system **must** contain a Home page.
- The system **must** include an easy navigable interface
- The system **must** be able to recognize a returning user that's already logged in.
- Using the system, a user **must** be able to create a profile.
- Using the system, a user **must** be able to View/Edit their own profile information.
- Using the system, a user **must** be able to delete their own profile
- Using the system, a user **must** be able to upload content.
- Using the system, a user **must** be able to view content they themself published.
- Using the system, a user **must** be able to edit content they themself published.
- Using the system, a user **must** be able to view content from other users.
- Using the system, a user **must** be able to download other users' teachers', lesson's instructions.
- Using the system, a user **must** be able to export the lesson's instructions.

Should

- The system **should** support the most common variety of file formats.
- The system **should** support commenting and rating content
- The system **should** have good Performance (See FURPS+)
- The system **should** have high Reliability (See FURPS+)

Could

- The system **could** support fetching other teachers' lesson's instructions in various formats.
- Using the system, a user **could** be able to export the lesson's instructions to the ORBIT-Box.

Won't

- The system **won't** support all types of file formats.
- The system **won't** support all file sizes.

FURPS+

Functionality

- Users' data has to be stored securely.
- The system has to be able to store users' uploads.
- The system will contain a search system, enabling its users to search for specific content uploaded by other teachers.
- The system is going to support registration with authentication.
- Users will be able to use UniLogin as a login option.

Usability

- The system's User Interface is expected to be intuitive.
- The system will be containing a simple and elegant design cohering to ORBIT-labs colors to some extent.

Reliability

- The system is expected to be available 7 days a week, 24 hours a day.

Performance

- The system needs to be fast and should score high on Google PageSpeed [3]. Meaning that the webpage has to score around 90 or higher. [4]
- The system scaling of files will be according to their size, when uploading content smaller files will be faster than larger files.
- The system will also scale its exporting performance, according to the size of the file.

Supportability

- The system will be expected to work properly on Google Chrome version 83 and up and Firefox version 77 and up.
- The system will support English use of language.
- The system will be guaranteed to support uploading content with items up to 250mb.
- The system might include support for alternative login options.

Implementation(+)

- The systems client will be coded in React (or angular)
- The systems server will be coded in Node.js and Express.

Technology Analysis

This section reflects decisions and recommendations on what will be used in the bachelor project based on the idea of a teachers Github which allows sharing experiment documents/layouts and other items.

1. General Stack

1.1. Server-side technologies

1.1.1. Node.js paired with Express web application framework to serve as primary server-side technology along with all logic and routing. Node.js and Express comes with endless middleware options and npm packages which eases the burden for the developer and allows us to focus on creating the important aspects of the bachelor project.

1.1.2. Database Considerations

1.1.2.1. CONSIDERATION 1: MongoDB to serve as database

1.1.2.1.1. Reason: Very good with scaling and high performance, this is also generally our favorite database.

1.1.2.2. CONSIDERATION 2: Firebase to serve as both a realtime database and authentication method.

1.1.2.2.1. Reason: Some prior knowledge from Smartphone applications course, easy to implement different authentication methods.

1.1.2.2.1.1. Negatives: Extremely tough to implement realtime database with node.js

1.1.2. UniLogin implementation Considerations

1.1.2.1. CONSIDERATION 1: Using Uni-login as IdP [5]

1.1.2.1.1. Reason: All teachers primarily use UniLogin at each of their respective institutions. Similar setup as miniOrange [6] as IdP using Passport. [7]

1.2. Client-side technologies

1.2.1. CONSIDERATION 1: Angular

1.2.1.1. Paired with Angular Material UI

1.2.1.1.1. Reason: Little knowledge but a good framework in terms of performance. Most Orbit lab technologies use angular, which could make this the most optimal choice.

1.2.2. CONSIDERATION 2: React.js

1.2.2.1. Paired with Material-UI: React framework

1.2.2.1.1. Reason: Little knowledge but exciting library with great performance.

1.2.3. CONSIDERATION 3: React (TypeScript)

1.2.3.1. Paired with Material-UI: React framework

1.2.3.1.1. Reason: Prior knowledge.

1.2.3.1.1.1. Negatives: Very hard to implement authentication + small user base.

2. Testing and Code quality assurance

2.1. Continuous Integration

2.1.1. CONSIDERATION 1: TeamCity [8]

2.1.1.1. Reason: TeamCity is a top choice candidate when it comes to CI as it has a very simple configuration and one of us has much experience with the tool from the mandatory internship. This can be coupled with the use of Git. [9]

2.1.2. CONSIDERATION 2: Jenkins

2.1.2.1. Reason: Prior use in I4SWT, also very modifiable.

2.2. Unit Testing frameworks

2.2.1. Jest

2.2.1.1. Reason: Works well with both node.js and all considered client-side choices. Not many considerations were made here as this

framework is known to be the best choice when unit testing node based projects.

2.3. Code quality assurance tools

2.3.1. SonarQube and SonarLint.

2.3.1.1. Reason: EXTREMELY useful tool, catches both bugs which are not found by the compiler and detects code smells. Using this tool will both decrease risks which can occur in the development phase and lift the quality of our code greatly. SonarQube is a very expensive tool, which means that many enterprise companies that use this tool take full advantage of it. For students and open source projects the tool is free of charge and integrates well with many different version-control systems.

3. Version-Control

3.1. CONSIDERATION 1: Github

3.1.1. Reason: Prior Experience from school and spare time projects, Github also contains a planning board to track development progress and create new task and log system bugs. (Free)

3.2. CONSIDERATION 2: Bitbucket

3.2.1. Reason: One of us used this tool in our internship, and found out that it pairs very well with Jira and TeamCity (*Somewhat free*)

Tools

Overleaf

The Project report itself will be written using the type-setting platform Overleaf, formerly known as LaTeX. Overleaf will contribute to the consistency of the project as a hole. This will provide better tools for handling referencing and quotes.

Software testing

During our studies so far, the amount of software testing we have experienced has been very limited. Throughout the process of this project, it will be a key concern, testing the capabilities of the system. For this objective, it is crucial to research more on how different testing methods apply for the relevant software. Many different types of software testing exist where many of the most commonly practised are relevant as a hole for this project. For an example we've chosen to pair Node.js with an express web application framework (See technology analysis p. 1.1.1). Node.js is known for having lightning fast performance, however it is still possible to have designed a system wrongfully allowing for worse performance than what is the goal for this project [10]. Different types of software testing that are deemed relevant for this project, will be researched and taken in use. [11]

Time Planning

Based on some prior experience with excel and time planning, we have carefully mapped out a burndown chart which after completion of the project will contain, an estimate of when we plan to finish our bachelor project (blue), along with an unrealistic approach in the case that we had no courses on the side (grey) and how the actual project went (orange).

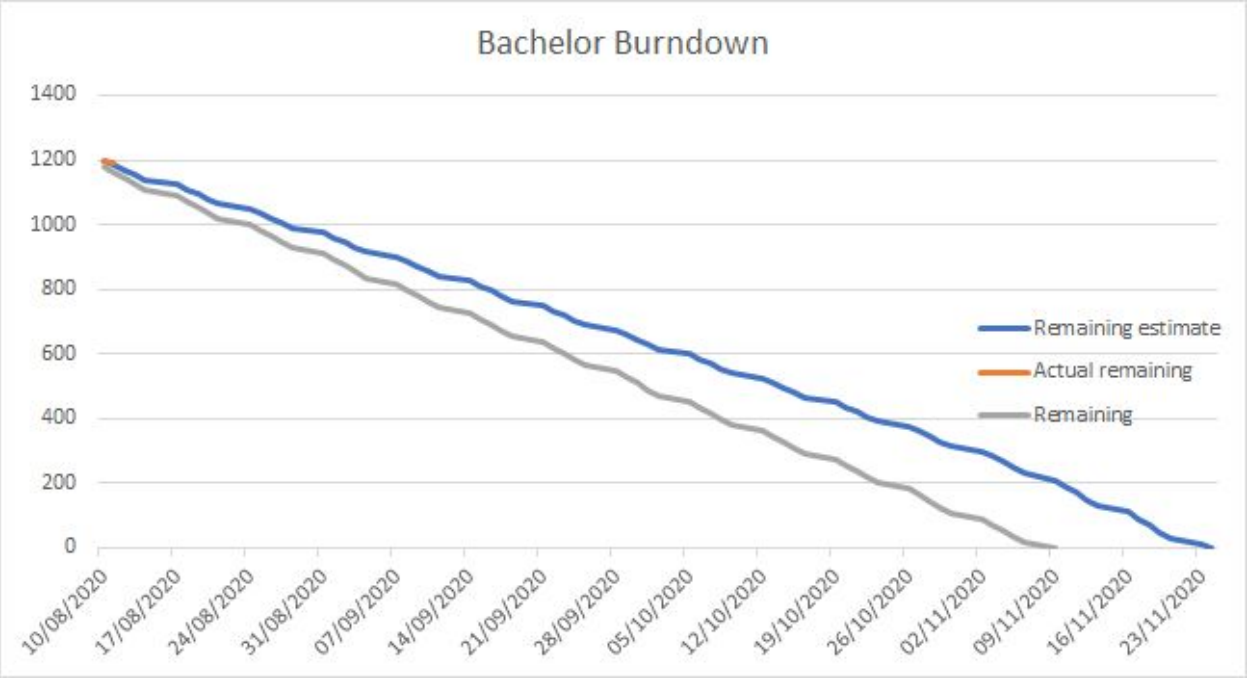


Figure 01: Burndown Chart for the expected project duration

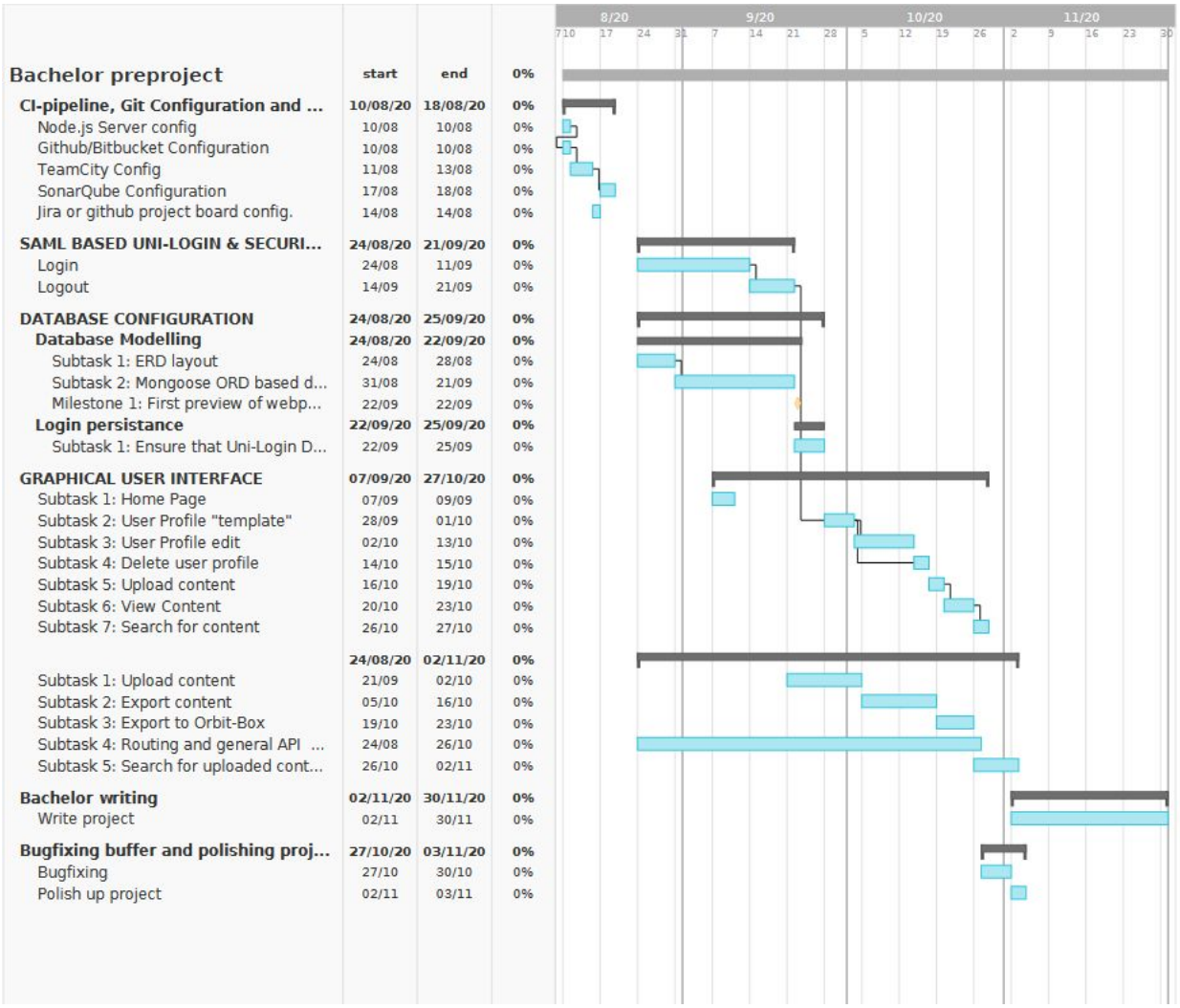
Here there are a few prerequisites in place. Before August 10th, we have created a 45 day "*pre-sprint*" where logistics of the project such as creating a pipeline between Bitbucket/Github and TeamCity, setting up Jira or a similar planning tool to view the status of pull requests, tasks, bugs and so on.

Besides the following, a simple skeleton for the project will likely be created based on the application generator tool: express-generator which creates a quick and simple web application skeleton, similar to when an ASP.NET project is created.

This decision is purely based on our desire to get started with the project so we have more time to implement functionality when we kick off the project on the 10th of August.

Gantt chart

A Gantt chart is a diagram that illustrates, using bars, a schedule for a project. It is widely used despite being invented back in the 1900. The diagram illustrates the project tasks on the vertical axis and the time spent on each phase in the project on the horizontal axis. Using this type of diagram, the plan for executing this project is much easier visualized. It also goes to show that there is a plan from day one. This cancels out the waiting time of starting the project itself with planning how to initialize it for the first days/week.



Bachelor preproject	start	end	0%
CI-pipeline, Git Configuration and ...	10/08/20	18/08/20	0%
Node.js Server config	10/08	10/08	0%
Github/Bitbucket Configuration	10/08	10/08	0%
TeamCity Config	11/08	13/08	0%
SonarQube Configuration	17/08	18/08	0%
Jira or github project board config.	14/08	14/08	0%
SAML BASED UNI-LOGIN & SECURI...	24/08/20	21/09/20	0%
Login	24/08	11/09	0%
Logout	14/09	21/09	0%
DATABASE CONFIGURATION	24/08/20	25/09/20	0%
Database Modelling	24/08/20	22/09/20	0%
Subtask 1: ERD layout	24/08	28/08	0%
Subtask 2: Mongoose ORD based d...	31/08	21/09	0%
Milestone 1: First preview of webp...	22/09	22/09	0%
Login persistance	22/09/20	25/09/20	0%
Subtask 1: Ensure that Uni-Login D...	22/09	25/09	0%
GRAPHICAL USER INTERFACE	07/09/20	27/10/20	0%
Subtask 1: Home Page	07/09	09/09	0%
Subtask 2: User Profile "template"	28/09	01/10	0%
Subtask 3: User Profile edit	02/10	13/10	0%
Subtask 4: Delete user profile	14/10	15/10	0%
Subtask 5: Upload content	16/10	19/10	0%
Subtask 6: View Content	20/10	23/10	0%
Subtask 7: Search for content	26/10	27/10	0%
	24/08/20	02/11/20	0%
Subtask 1: Upload content	21/09	02/10	0%
Subtask 2: Export content	05/10	16/10	0%
Subtask 3: Export to Orbit-Box	19/10	23/10	0%
Subtask 4: Routing and general API ...	24/08	26/10	0%
Subtask 5: Search for uploaded cont...	26/10	02/11	0%
Bachelor writing	02/11/20	30/11/20	0%
Write project	02/11	30/11	0%
Bugfixing buffer and polishing proj...	27/10/20	03/11/20	0%
Bugfixing	27/10	30/10	0%
Polish up project	02/11	03/11	0%

Figure 02: Gantt Chart for the expected project duration

Literature search

Bibliography

General references

https://github.com/ [1]

https://www.mountaingoatsoftware.com/agile/user-stories [2]

Pagespeed value definition

https://developers.google.com/speed/pagespeed/insights/?hl=da [3]

https://developers.google.com/speed/docs/insights/v5/about?utm_campaign=PSI&utm_medium=incoming-link&utm_source=PSI&hl=da-DK [4]

Articles on how to implement SAML authentication in Node.js

https://viden.stil.dk/display/OFFSKOLELOGIN/Unilogin+IdP [5]

https://www.miniorange.com/step-by-step-guide-to-setup-sso-into-nodejs-app [6]

http://www.passportjs.org/packages/passport-saml/ [7]

TeamCity articles on setup and Git integration

https://www.jetbrains.com/help/teamcity/installing-and-configuring-the-teamcity-server.html# [8]

https://medium.com/@amaya30/integrating-teamcity-and-github-with-ssh-is-actually-as-easy-as-you-would-think-it-is-3429e01112ea [9]

Performance testing and tuning

https://stackify.com/node-js-performance-tuning/ [10]

https://stackify.com/ultimate-guide-performance-testing-and-software-testing/ [11]