

Bachelor's project

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical
Engineering
Department of ICT and Natural Sciences

Aron Mar Nicholasson
Liban Bashir Nor
Bendik Nogva Uglem

AutoPacker

Automated software packaging and deployment
solution

Bachelor's project in Bachelor i ingeniørfag - Data
Supervisor: Girts Strazdins

May 2020



NTNU

Kunnskap for en bedre verden

Aron Mar Nicholasson
Liban Bashir Nor
Bendik Nogva Uglem

AutoPacker

Automated software packaging and deployment
solution

Bachelor's project in Bachelor i ingeniørfag - Data
Supervisor: Girts Strazdins
May 2020

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of ICT and Natural Sciences



Norwegian University of
Science and Technology

Obligatorisk egenerklæring/gruppeerklæring

Den enkelte student er selv ansvarlig for å sette seg inn i hva som er lovlige hjelpemidler, retningslinjer for bruk av disse og regler om kildebruk. Erklæringen skal bevisstgjøre studentene på deres ansvar og hvilke konsekvenser fusk kan medføre. Manglende erklæring fritar ikke studentene fra sitt ansvar.

<i>Du/dere fyller ut erklæringen ved å klikke i ruta til høyre for den enkelte del 1-6:</i>	
1.	Jeg/vi erklærer herved at min/vår besvarelse er mitt/vårt eget arbeid, og at jeg/vi ikke har brukt andre kilder eller har mottatt annen hjelp enn det som er nevnt i besvarelsen. <input checked="" type="checkbox"/>
2.	Jeg/vi erklærer videre at denne besvarelsen: <ul style="list-style-type: none">• ikke har vært brukt til annen eksamen ved annen avdeling/universitet/høgskole innenlands eller utenlands.• ikke refererer til andres arbeid uten at det er oppgitt.• ikke refererer til eget tidligere arbeid uten at det er oppgitt.• har alle referansene oppgitt i litteraturlisten.• ikke er en kopi, duplikat eller avskrift av andres arbeid eller besvarelse. <input checked="" type="checkbox"/>
3.	Jeg/vi er kjent med at brudd på ovennevnte er å <u>betrakte som fusk</u> og kan medføre annullering av eksamen og utestengelse fra universiteter og høgskoler i Norge, jf. Universitets- og høgskoleloven §§4-7 og 4-8 og Forskrift om eksamen §§14 og 15 . <input checked="" type="checkbox"/>
4.	Jeg/vi er kjent med at alle innleverte oppgaver kan bli plagiakkontrollert i <u>Ephorus</u> , se Retningslinjer for elektronisk innlevering og publisering av studiepoenggivende studentoppgaver <input checked="" type="checkbox"/>
5.	Jeg/vi er kjent med at høgskolen vil behandle alle saker hvor det <u>forligger</u> mistanke om fusk etter høgskolens studieforskrift §31 <input checked="" type="checkbox"/>
6.	Jeg/vi har satt oss inn i regler og retningslinjer i bruk av <u>kilder og referanser på biblioteket sine nettsider</u> <input checked="" type="checkbox"/>

Preface

Before you lies the thesis "AutoPacker - Automated software packaging and deployment solution.", which contains the results of our research and development. The thesis has been written by three computer engineering students at NTNU Ålesund to fulfil the requirements for the bachelor degree.

AutoPacker is a self-defined project where the concept has been worked on since the second year with the help of Girts Strazdins. It started with a desire to create an application to automatically build independent applications using virtualization technology, to be an idea for a platform that builds and deploys software to a centralized hub which gets used for testing, server deployment and storage.

We hope you find the thesis interesting.

Aron Mar Nicholasson

Liban Bashir Nor

Bendik Nogva Uglem

Ålesund, May 20, 2020

Acknowledgement

We want to thank Girts Strazdins for his help in establishing the idea for the project, as well as being an outstanding supervisor and acting project owner.

We also want to give a special thanks to Avento and Kenneth Gjersdal for their cooperation and mentorship.

Abstract

When a teacher has to upload a project to a server for a specific lecture, it might involve tedious and repetitive configurations and even software updates due to changes and updates in software that does not comply with the program the teacher wants to host. New students may also find it hard to start working with server configuration, and more experienced students may find simple configuration tedious in length.

Current solutions that solve these problems are platform or provider dependent and very complicated to use by providing an ocean of possibilities and tools for the user. This can be especially hard for new learners that do not have time or money for these solutions.

We want AutoPacker to be a simple, but productive and transparent platform that is cloud service and hosting independent and offers a way to manage projects, servers, deployment and storage, and being a platform for people to share projects and ideas.

AutoPacker is built using microservice architecture and consists of one web application and 4 APIs. AutoPacker can deploy to any server, which means students and teachers can use servers provided by the school. AutoPacker supports simple uploads like Java (both project and Jar), Spring-Boot, static sites, Angular and React projects. It also has a modular configuration builder that makes it easy to implement further support later on. In AutoPacker, universities can create organizations that can store lecture projects, bachelor projects and other types of projects.

Contents

Declaration	i
Preface	ii
Acknowledgement	iii
Abstract	iv
Acronyms	2
1 Introductions	10
1.1 Background	10
1.2 Problem Formulation	10
1.3 Scope	11
1.4 Objectives	11
1.5 Structure of the Report	12
2 Theoretical basis	13
2.1 Agile Development	13
2.1.1 Scrum	13
2.2 Security	13
2.2.1 Why encrypt?	13
2.2.2 Why hash?	14
2.2.3 Our needs	14
2.2.4 Types of encryption and hashing	15
2.2.5 JWT	16
2.2.6 The chosen cryptography mechanisms	16
2.2.7 Possible exploits	16

Cross-Site Script	16
SQL Injection	17
Zip Slip Vulnerability	17
2.3 Privacy	18
2.3.1 General Data Protection Regulation (GDPR)	18
2.3.2 Right To Privacy	18
2.4 Communcation and RESTful	19
2.4.1 What is RESTful	19
2.4.2 Why RESTful over conventional method	20
2.4.3 Usage of the HTTP methods	21
2.5 Docker Concepts	22
2.5.1 Containers	22
2.5.2 Docker Images	23
2.5.3 Docker Volume and Bind Mount	24
Volume	24
Bind Mount	24
2.5.4 Docker Registry	25
2.5.5 Docker Network	25
2.5.6 Swarm	26
Service	27
Secret	29
Stacks	29
2.6 Spring Boot Patterns & Principles	29
2.6.1 Beans	29
2.6.2 Inversion of Control (IoC)	29
2.6.3 IoC Container	30
2.7 Server Administration	30
2.7.1 Continuous Integration (CI)	30
2.7.2 Continuous Deployment (CD)	30
2.7.3 Blue Green Deployment	30

2.8 Database Concepts	31
2.8.1 Relational Databases	31
2.8.2 Entities and domains	31
2.8.3 Repositories	31
3 Method	32
3.1 Project Organization	32
3.1.1 Scrum	32
3.2 Work Organization	32
3.3 Security Flaw	34
3.4 Programming languages	34
3.4.1 Java	34
3.4.2 JavaScript	34
3.5 Command languages	35
3.5.1 Bash	35
3.6 Package Manager	35
3.6.1 Yarn	35
3.7 Frameworks	36
3.7.1 Spring Boot	36
3.7.2 React	36
3.7.3 React Hooks	36
3.7.4 Redux	37
3.7.5 React-router	37
3.7.6 Ant Design	38
3.8 Data	39
3.8.1 JSON	39
3.8.2 YAML	39
3.8.3 MySQL	39
3.8.4 MongoDB	40
3.9 Project Management	40

3.9.1 Management Tools	40
Confluence	40
Jira	40
Teamwork	40
Draw.io	40
Visual Paradigm Online	41
3.9.2 Development Tools	41
Docker	41
GitLab	43
Postman	43
Wireshark	43
IntelliJ	43
Visual Code	43
Git	44
Database Management Tools	44
SSH Client	44
3.10 Existing Solutions Comparisons	44
3.10.1 Azure Comparison	44
3.10.2 Heroku	45
3.10.3 GidPod	45
3.11 Documentation	46
4 Result	47
4.1 Architecture	48
4.1.1 Deployment Diagram	48
4.1.2 Use Case Diagram	49
4.2 Database overview	50
4.2.1 Authentication Server Database	50
4.2.2 Server Manager Database	51
4.2.3 General Database	52

4.2.4	File Delivery Database	53
4.3	Technology Stack survey	53
4.3.1	Go-to programming-/scripting language for different projects	54
4.3.2	Frontend	55
4.3.3	Backend	56
4.3.4	Database	56
4.3.5	Server	57
4.3.6	Extra thoughts/comment	57
4.4	Gitlab CI/CD	58
4.5	Backend Services	61
4.5.1	File Delivery API	61
Authorization	61
File Explorer	61
Docker-compose template builder	63
General builder functionality	64
Parameters	65
Placeholders and their replacement	65
4.5.2	Validation	66
4.5.3	MongoDB	66
4.5.4	Server Manager	67
Server Initialization Script	67
Connecting and transmitting data	67
Project Deployment	68
4.5.5	Authentication Server	69
JWT vs OAuth2	69
Registration	69
Authentication	70
Authorization	70
4.6	General API	71
4.7	Web Application	71

4.7.1	Routing	71
4.7.2	Application State Management	73
4.7.3	React Hooks	74
4.7.4	Custom Alert	76
4.7.5	Search Logic	77
4.7.6	Graphical Interface	78
4.7.6.1	Homepage	79
4.7.6.2	Registration Success	80
4.7.6.3	Projects	82
4.7.6.4	New Project	83
4.7.6.5	Project Overview	85
4.7.6.6	Module Selection	86
4.7.6.7	Single-Module	87
4.7.6.7.1	Setup	87
4.7.6.7.2	Upload	88
4.7.6.7.3	Building	88
4.7.6.7.4	Complete	89
4.7.6.8	Multi-Module	89
4.7.6.8.1	Summary	90
4.7.6.8.2	Type	91
4.7.6.8.3	Setup (database)	92
4.7.6.8.4	File Upload (database)	92
4.7.6.8.5	Summary (with modules)	93
4.7.6.9	Own Setup	94
4.7.6.10	Project Overview (populated)	95
4.7.6.11	Project Settings	97
4.7.6.12	Servers	98
4.7.6.13	Server Creation	99
4.7.6.14	Server Overview	100
4.7.6.15	Organizations	102

4.7.6.16 Organization Projects	103
4.7.6.17 Organization Project Requests	104
4.7.6.17.1 Decline Request	105
4.7.6.17.2 Edit Request	105
4.7.6.17.3 Accept Request	106
4.7.6.18 Organization Members	106
4.7.6.19 Organization Applicants	107
4.7.6.20 Organization - Submit Project	108
4.7.6.21 Organization Submissions	109
4.7.6.22 Search Result	110
4.7.6.23 User Profile	112
4.7.6.24 User Project	112
4.7.6.25 Organization Profile	113
4.7.6.26 Organization Membership	113
4.7.6.27 User Settings	114
4.8 Docker architecture	115
4.8.1 Docker compose (staging)	115
4.8.2 Docker Swarm (production)	117
4.9 Testing	118
4.9.1 Usability Testing	118
4.9.2 Mentor and Supervisor test	118
4.9.3 Test Results	118
5 Discussion	120
5.1 Results VS Expectation	120
5.2 Project Organization	121
5.3 RESTful Services	121
5.3.1 File Delivery API	121
5.4 Security	122
5.5 Web Application	122

CONTENTS	1
5.5.1 Design & Interface	123
5.5.2 Code Quality & Structure	123
5.6 Production	124
5.7 Limitations	124
5.8 Future Work	124
6 Conclusions	126
Bibliography	128
Appendices	134
A Preliminary Report	134
B API Specifications	154
C Gantt diagram	163
D Web Application Source Code	164
E File Delivery API Source Code	164
F General API Source Code	164
G Server Manager Source Code	164
H Authentication Server Source Code	164
I Dynamic use of docker compose	164
J Wireframes	168
K Jira Sprint Reports	172
L Retrospective Meeting Notes	186
M Meeting Notes	191
N Weekly Logs	198
O Note from tester	216

Terminology

Docker Docker is a tool for managing containers (explained in detail in section [2.5](#) and [3.9.2](#)).

Docker Image A docker image is an executable package of different software, library, tools and settings.

Docker Container A docker container is a process instanced from a docker image.

Front-end The part of a computer system or application that the user interacts directly with.

Back-end The part of a computer system or application that is not directly available to the user.

Sprint A sprint is a time-box iteration of a continuous cycle of development.

Agile A working method focused on working in iterations.

Scrum is an agile process framework where a team plans, adapts, changes, develops and deploys in iterations.

Cipher Is a general term for algorithms in cryptography.

Salt Random value added into a hashing function to secure the hash.

Crack Hacking with malicious intent.

User story Is an informal, natural description of a feature in a system described from the eyes of an end-user.

Unix An operating system. Most popular variants are MacOS and Linux.

Wizard Is a setup assistance interface to help the user achieve his goal.

Abbreviations

API Application Programming Interface

ACK acknowledge message

REST REpresentational State Transfer

URL Uniform Resource Locator

URI Uniform Resource Identifier

SSH Secure Shell

CI Continuous Integration

CD Continuous Deployment

JWT JSON Web Token

UI User Interface

UML Unified Modeling Language

HTML Hyper Text Markup Language

SQL Structured Query Language

CSS Cascading Style Sheet

JSON JavaScript Object Notation

HTTP HyperText Transfer Protocol

HTTPS HyperText Transfer Protocol Secure

SSL Secure Sockets Layer

IDE Integrated Development Environment

JPA Java Persistence API

JDBC Java Database Connectivity

VM Virtual Machine

GDPR General Data Protection Regulation

IoC Inversion of Control

DI Dependency Injection

YAML YAML Ain't Markup Language

GUI Graphical User Interface

XML Extensible Markup Language

JDK Java Development Kit

DTR Docker Trusted Registry

TCP Transmission Control Protocol

SSL Secure Sockets Layer

TLS Transport Layer Security

RSA Rivest–Shamir–Adleman

DES Data Encryption Standard

3DES Triple DES

TDEA Triple Data Encryption Algorithm

IaaS Infrastructure as a Service

PaaS Platform as a Service

SaaS Software as a Service

List of Figures

2.1	Figure showing an example of ubuntu:15.04 image layers[19].	23
2.2	Figure showing how docker containers can persist and manage generated data[27].	24
2.3	Figure showing a general docker network architecture[42].	25
2.4	Figure showing a docker swarm overlay network architecture[42].	26
2.5	Figure showing a docker swarm with nodes[18].	27
2.6	Figure showing a service object[25].	28
2.7	Figure showing the process of assigning a service to a manager and tasks being given to worker nodes[25].	28
3.1	Figure showing the pre planned thought of our infrastructure.	33
3.2	Figure showing our pre planned work on what modules we needed for our bachelor project.	33
3.3	Figure showing swot analysis on ant design and material UI.	38
3.4	Figure showing the difference of virtualization between Docker and virtual machines[10].	42
4.1	Deployment diagram of the system of micro-services.	48
4.2	A use case describing what interactions different roles can expect.	49
4.3	Figure showing the authentication server database.	50

4.4 Figure showing the server manager database.	51
4.5 Figure showing the general service database.	52
4.6 Figure showing the file delivery service database.	53
4.7 Figure showing the most used programming and scripting languages.	54
4.8 Figure showing the most popular tools and frameworks for developing front-end applications.	55
4.9 Figure showing the most popular tools and frameworks for developing back-end services.	56
4.10 Figure showing the most popular databases.	56
4.11 Figure showing the most popular servers.	57
4.12 A sequence diagram of creating a module after a project has been added. . .	62
4.13 Folder structure of the API, where the focus is on the separation of users, projects and modules.	63
4.14 The base docker-compose template where only variables on the highest scope is defined.	63
4.15 Docker-compose template of the Java 8 service.	64
4.16 Main template builder code snippet from BuilderService class.	65
4.17 Code snippet showing an example of using JSch.	67
4.18 Sequence diagram for deploying a project on to a server.	68
4.19 Sequence diagram for registration.	69
4.20 Sequence diagram for authentication.	70
4.21 Figure showing the custom route for a profile dashboard component.	72
4.22 Figure showing the module upload part of the application with component hierarchy and switching.	73
4.23 Figure showing application state management with regular React Redux. .	75
4.24 Figure showing application state management with hooks.	75

4.25 Figure showing a use example for the alert component.	76
4.26 Figure showing the implementation of the debounce hook.	77
4.27 Figure showing an example using the debounce hook.	77
4.28 Figure showing the wireframe for project overview.	78
4.29 Figure showing our temporary homepage.	79
4.30 Figure showing the registration form.	80
4.31 Figure showing the profile panel after the user is successfully registered. . .	80
4.32 Figure showing the verification email.	81
4.33 Figure showing profile dashboard when authenticated and verified.	82
4.34 Figure showing page for creating the projects.	83
4.35 Figure showing the profile panel after project creation.	84
4.36 Figure showing a project overview page.	85
4.37 Figure showing the module type selection page.	86
4.38 Figure showing the setup part of a single-module upload.	87
4.39 Figure showing the upload part of a single-module upload.	88
4.40 Figure showing the complete part of a single-module upload.	89
4.41 Figure showing the summary part of a multi-module upload.	90
4.42 Figure showing the type selection part of a multi-module upload.	91
4.43 Figure showing the setup step for a database module in multi-module upload. .	92
4.44 Figure showing the summary part with modules data in multi-module up- loading.	93
4.45 Figure showing the own setup option.	94
4.46 Figure showing a project overview with added module.	95
4.47 Figure showing a modal containing module details.	96
4.48 Figure showing a modal for deleting a module.	96

4.49 Figure showing the project settings page.	97
4.50 Figure showing all the users' servers.	98
4.51 Figure showing the form for adding a server.	99
4.52 Figure showing an overview of a server without projects assigned to it.	100
4.53 Figure showing the modal for adding projects.	101
4.54 Figure showing an overview of a server with projects assigned to it.	101
4.55 Figure showing a list of organizations the user is affiliated with.	102
4.56 Figure showing a list of projects affiliated with the organization.	103
4.57 Figure showing the dropdown menu from hovering over username.	104
4.58 Figure showing a list of project requests made to the organization.	104
4.59 Figure showing the modal for declining a request.	105
4.60 Figure showing the modal for editing a request.	105
4.61 Figure showing the modal for accepting a request.	106
4.62 Figure showing a list of all the members of an organization.	106
4.63 Figure showing a list of all the applicants for the organization.	107
4.64 Figure showing the page used to submit a project request.	108
4.65 Figure showing a list of submissions made by the user.	109
4.66 Figure showing the modal for editing an existing submission.	110
4.67 Figure showing the search results page.	111
4.68 Figure showing the search results containing users.	111
4.69 Figure showing the search results containing organizations.	112
4.70 Figure showing a public user profile.	112
4.71 Figure showing a public project overview for a user project.	112
4.72 Figure showing a public organization profile.	113
4.73 Figure showing the form to submit a membership application.	114

4.74 Figure showing the user settings page.	114
4.75 Figure showing our staging environment using docker.	115
4.76 Figure showing a concept for a production environment using docker. . . .	117
1 Figure showing the wireframe for the homepage.	168
2 Figure showing the wireframe for the dashboard panel containing user projects.	169
3 Figure showing the wireframe for creating a new project.	169
4 Figure showing the wireframe for an empty project.	170
5 Figure showing the wireframe for a project containing modules.	170
6 Figure showing the wireframe for creating a new module for a project. . . .	171
7 Figure showing the wireframe for listing all projects in an organization. . . .	171

Chapter 1

Introduction

1.1 Background

The reason why we chose this custom task is based on our own experience and that we see a benefit to this solution that can help students in particular, but also individuals who are developing as a hobby. We want to create a solution that automates the process by going from source code to a platform-independent solution that can run both locally and on a remote server that is automatically configured and runs the application in a virtualized environment. In this way, it can be used for testing, sharing and operation of older and newer projects.

1.2 Problem Formulation

When one is trying to develop, test and run a software application on different systems, it takes much time to configure the application environment so it can run without problems (as different operating systems require different configuration). Usually one performs these steps several times in which it becomes tedious.

This project solution will help automate this task and also make the application platform-independent by using virtualization tools and automatic server and local environment configuration.

1.3 Scope

The scope of this bachelor project is to be able to create a platform that can perform simple automatic server configuration, software building/bundling and project deployment. The platform architecture will be modular and scalable using microservice architecture.

The platform can provide a foundation for future tool support by having an easy to extend and generic file builder for uploading modules to projects.

The project is well maintained and documented so that others will be able to contribute in the future.

The platform can provide a platform for users to create, share and deploy projects as well as explore other users projects, join organizations and were both students and lecturers can store submissions, lectures and bachelor projects.

1.4 Objectives

The objectives for this bachelor thesis are:

- To research different ways to implement automatic environment and application setup.
- To create a standalone system that can automatically configure a virtualized environment for a remote server or local computer.
- To achieve good test result.
- To make a solution that can be used for software based bachelor projects.
- To make a solution that can be used by NTNU for bachelor hosting, distribution and operations.
- To create an independent platform that can be used by all individuals.
- To create a solution that can be used my multiple organizations (preferably schools and universities).

1.5 Structure of the Report

The rest of the report is structured as follows.

Chapter 2 - Theoretical basis: Chapter two gives an introduction to the theoretical background for the concepts and tools needed to prepare the reader for the following chapters.

Chapter 3 - Materials & Methods: Contains a description of the methodology and materials that were considered throughout the project.

Chapter 4 - Result: Contains a description of the finished product. What results we achieved and what decisions were made.

Chapter 5 - Discussion: Contains a reflection on the work that has been done, choices that have been made and what could have been done differently.

Chapter 6 - Conclusions: This chapter presents the overall conclusion.

Chapter 2

Theoretical basis

2.1 Agile Development

Agile development is a work process that focuses on delivery in iterations, rather than delivering it all at once. With agile development, the goal is to create a minimum viable product first, then keep building upon that product through sprints. The team keeps a backlog of what features is needed in order to create the minimum viable product. In each sprint, the team chooses issues from the backlog and add it to their sprint. This process is repeated typically biweekly until the product is finished[57].

2.1.1 Scrum

Scrum is a variant of Agile development but with a greater focus on team communications and meetings. At the beginning of every day, the team has daily stand-up meetings where they inform about what they have done, what were difficult, and what to work with next [57].

2.2 Security

2.2.1 Why encrypt?

In a perfect world without people with malicious intent, there would be no need for encryption. Human readable data gets sent between two parties, and there would be no need to worry. In

the real world, however, the case is different. The data could contain private information or other information that the sender does not want others to know. It can be anything from once real name to once credit card information. Information like this is almost always sent when performing online purchases. This makes it extremely important that the information is transmitted securely to the receiver. Encryption is a security layer that prevents the data from being compromised between point A and B.

2.2.2 Why hash?

Hashing is a one-way function that changes the original value to a value that is not understood by anyone who reads it. In the case of a security breach, this technique prevents the intruder from getting the raw information quickly. Common ways to find the original value is using dictionary attacks, where one hashes the words in a dictionary, or brute force, where one hashes any combination of symbols. The hashes generated from these attacks are then compared with the target hash to find the original value. Therefore it is essential not to use hashing as the primary security and use it with a combination of other security mechanisms. Hashing delays the end part of an attack, so one has time to react and fix the issues, and alternatively, disclose it to the affected parties.

2.2.3 Our needs

In our systems, we are going to transmit a lot of data between our users, frontend and backend. All data is going to be sent using the protocols TCP and HTTP. HTTP itself is an unsafe protocol because all the packets sent through the protocol is visible for whoever got the packets.

Since we will store the user password in a database, we do not want this sensitive information in plain text in case someone manages to hack into the system. The hashing function does not need to be quick, but it does require good strength. Any hashing function used for passwords with a commonly used bit length of 256 is proper. It should also add salt when hashing the password.

2.2.4 Types of encryption and hashing

- **RSA** - An asymmetrical encryption method that uses public/private key pairs to encrypt and decrypt messages. RSA's security is based on factorization of very large numbers with large prime numbers, making it difficult to compute[40][55].
- **DES** - A symmetrical encryption introduced in year 1974[14]. The bit size of the cipher is 64 and 8 of them is used for parity checks and the remaining 56 bits are the key size. Because of its bit size, this is not a secure algorithm. DES encryption can be cracked in less than a day.
- **Double DES and TDEA** - These encryptions are an improvement of DES. Since the biggest flaw of its predecessor was its cipher block length, double DES and TDEA encryption increases it. The difference between double DES and TDEA is that double DES uses the DES encryption twice and TDEA (or 3DES) uses it 3 times, giving the final key length of 112 and 168 bits. According to National Institute of Standards and Technology, double DES and TDEA using two keys has been deprecated, where the ladder being disallowed from 2018[9].
- **bcrypt** - Bcrypt is a hashing function that uses Blowfish, an older method, as base. The appeal of bcrypt is that it has built-in salt that prevents it from dictionary attacks. The hashing function has an iteration variable which defines how many times the hashing will be ran. This makes cracking the hash slower and makes bcrypt remain secure for years[1].
- **SHA-2** - This hashing function is considered a secure function today. It was designed by the National Security Agency as an improvement of SHA-1. The difference between SHA-2 and SHA-1, its predecessor, is not just an increase in bit length, but also the fundamentals of the hashing function. SHA-2 refers to a set of hashing that uses the same mechanism and has different bit lengths. They can also be named SHA-256, SHA-512 and ect., where the numeric part of the name is the size. Since it is quick to generate the hashes, it can also be cracked faster. This makes it a disadvantage when using SHA to encrypt passwords[4].

2.2.5 JWT

JWT is an access token used for authorization and is commonly used in APIs. In addition to authorization, it can also transport additional data between client and server. One can sign tokens to validate the integrity of the token. The token is signed using a secret key and can either utilize symmetric or asymmetric cryptography methods.

2.2.6 The chosen cryptography mechanisms

The traffic between the client and server will use SSL/TLS with RSA for the handshake. During the handshake, both parties agree on a session key for the symmetrical encryption and they both store this key. The reason symmetrical encryption is used over the public cryptography RSA after the handshake is because it is faster.

We will secure our JWT tokens with HMAC512. HMAC512 is a combination of two mechanisms, SHA-512 and HMAC. Essentially, HMAC is a signature that is used with hashes to verify that the data inside has not been modified. This cryptography is the most commonly used to generate JWT tokens[41]. One flaw of the cryptography method is that it uses a symmetrical key. Anyone who has that key, e.g. cracks it from a token, can generate valid tokens with a modified token payload. The risk of impersonation is high at this point. To reduce this risk, it is crucial to use strong keys and optionally regenerate new keys often.

Passwords that are stored in our system will be hashed using bcrypt. It is slow, meaning it is more resistant to brute-force attacks. This is the preferred hashing function for password because of the security, especially more preferred than SHA and similar functions.

2.2.7 Possible exploits

Cross-Site Script

Cross-site scripting (XSS) happens when a hacker manages to inject a script into a web application[7]. The script is only executed when someone visits the site on their machines. User actions, data and input can be captured by the hacker, and the hacker can abuse it. To guard against such attacks, one can sanitize the user input or encode it.

SQL Injection

SQL Injection, or SQLi for short, is a vulnerability where one can manipulate SQL queries in a way it is not intended[5]. A hacker supplies a SQL query instead of a proper value through the application to the SQL server. The query that the developers wrote is closed using SQL syntax, and the malicious query is followed by it. Basically, the hacker is granted full access to the database and can retrieve sensitive information, change or delete data from the database. Many modern solutions have implemented a countermeasure against this called prepared statements. It is also possible to sanitize the values.

Zip Slip Vulnerability

Zip Slip is a type of path traversal attack[8]. This attack occurs when the target machine unpacks a zip, and a file during that process escapes the originally intended file system location. This file may override system files or configuration. It can be executed either by remote command or by the system itself as a part of its regular system routine. Zip slip attack works by adding a reference to a parent directory multiple in the file name. Regular files, for example named 'run.sh', can be named '../.../run.sh' to start the exploitation. This is a risk when the system tries to treat files when a module is uploaded with a zip. To prevent this kind of attacks, file names must be sanitized.

2.3 Privacy

2.3.1 General Data Protection Regulation (GDPR)

The General Data Protection Regulation (GDPR) is a regulation in EU law made in 2016 and enforced in Norway in 2018. The regulation surrounds the protection of natural persons concerning the processing of personal data and on the free movement of such data [33].

Due to the exponential digitalization and rapid development of technology, the world needed a regulation that protects users privacy.

"The principles of data protection should apply to any information concerning an identified or identifiable natural person." - REGULATION (EU) 2016/679 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL

The regulation enforces businesses/processors to take the right decisions regarding data protection. As mentioned in Article 28 g): "*at the choice of the controller; deletes or returns all the personal data to the controller at the end of the provision of services relating to processing*". This means a user/consumer can ask to get all information saved about him or delete all that data as long as the union or member state does not require storage of the personal data [33].

In our application, the only mandatory personal information we need is the email address. This is used to verify the users' identity.

2.3.2 Right To Privacy

The right to privacy is about the right to private personal life and the right to decide on once own personal data[15]. We find this very important, and we want the user to be able to know at all times what data we have stored, and also be able to delete the data if needed. We will not push this application to production unless we have these options ready for the user.

2.4 Communcation and RESTful

2.4.1 What is RESTful

REST is a design pattern that web services follow to communicate with each other[3]. A RESTful API or service refers to services that use the REST conventions. In order to achieve a RESTful state, these guidelines must be met:

- **Client-Server Model** - The app using the RESTful API does not need to worry about how the data is stored, what technology it is using or how it is implemented. The only requirement is that the HTTP response schema format corresponds[11].
- **Stateless** - The API does not save nor require any previous calls to interpret the calls. The client must provide the minimum necessary data when performing calls. Any session data is only kept on the client-side.
- **Caching** - Caching reduces the load on the server by saving the data static. The API then does not need to reproduce the data which is going to use more hardware resources.
- **Uniform interface** - Uniform interface is a design that constrains the API, so each operation is understandable and consistent[44]. There are four principles for a uniform interface:
 1. Each resource has it's own URI
 2. Actions on Resources Through Representations
 3. Messages are self explanatory
 4. *HATEOAS*, or Hypermedia as the Engine of Application State
- **Layered system** - A layered system is a system where the services deployed on different servers have different purposes. One server can store data, and the other is the REST API.

2.4.2 Why RESTful over conventional method

When using REST, one has separate back-end and front-end. This leads to lower coupling and higher cohesion. Front-end can focus on how to parse the data and display them while the back-end focuses on creating/updating/deleting the data and structuring the response. It creates abstraction where the front-end does not need to know how the data is processed. The abstraction, combined with the REST standards makes the API usable with services with modern languages and frameworks.

A resource is an object that has data in it. An example is an employee that has, along with other data, a name, phone number, age and address. Each resource has its own URI. Let us take employees as an example. If one wants to perform actions, one can do GET `api.example.com/employees`. Usually, each resource has its own controller. Any operations done on the resource is handled by the corresponding controller. Again, making it low coupling and high cohesion.

The URLs are verbose. It makes it easier to understand what the endpoint is for. Objects can have objects inside them. An example of that is an employee that has a development project ongoing. Both the employee and the projects are objects. Example:

`api.example.com/employees/2/projects/1/` tells that the action will be performed on the second employee's first development project. The action depends on what HTTP method one uses.

All the separation, abstraction, cohesion and coupling makes the development process on the API simple and faster. This is very effective when the API needs to be upscaled with new features.

2.4.3 Usage of the HTTP methods

REST APIs follows CRUD, a set of functions to store data with persistence[6]. CRUD stands for create, remove, update and delete. Here are the HTTP methods implementation of CRUD standard:

1. **GET** - GET is the most commonly used HTTP methods on APIs. The method retrieves data from a specific resource[53]. I.e, if you wish to get all employees, you can send a GET request to the URI `.../employees`. Narrowing down the result is possible, either by using a filter algorithm (`.../employees?age=22`) to get data from multiple employees or using the employees' numerical identifier (`.../employees/3`) from one employee.
2. **POST** - To submit new data to a resource, you use a POST request. Most of the data is submitted in the request body. The file format of the body is JSON, XML or form data. JSON is the most common. The data is sent using POST directly on a resource (`.../employees`). POST can also be used to update existing data.
3. **DELETE** - The DELETE method removes one or multiple data records from a resource.
4. **PUT** - PUT is often used to update an existing data record. I.e. one employee has moved and therefore needs to update his address in the API's database. POST and PUT are similar, where the only distinction is that PUT is idempotent.
5. **Other** - We have other methods, but they are not used frequently. These are PATCH, HEAD, OPTION.

2.5 Docker Concepts

Virtualization has become a must know field for many information technology fields and is used daily in areas like server consolidation, information security and cloud computing. This is largely due to an increase in hardware performance, and the goal to fully maximize the potential of the hardware[37].

Virtualization has paved the path for several technologies. A relatively new technology that utilizes virtualization to manage and maintain software applications is containerization tools like Docker (More on Docker as a tool in section [3.9.2](#)).

2.5.1 Containers

Containerization has become a major trend in software development and operations. It involves encapsulating or packaging up a software application and all its tools, libraries, configuration files and dependencies to run uniformly and consistently on any infrastructure[32].

This eliminates OS and machine dependent errors and bugs as the application is abstracted away from the host operating system, and therefore becomes portable and platform independent.

2.5.2 Docker Images

A Docker image is a read-only template with instructions for creating a docker container. These instructions are executed in order when initializing a container from the image. One can also have images based on other images to get customized behaviour[22].

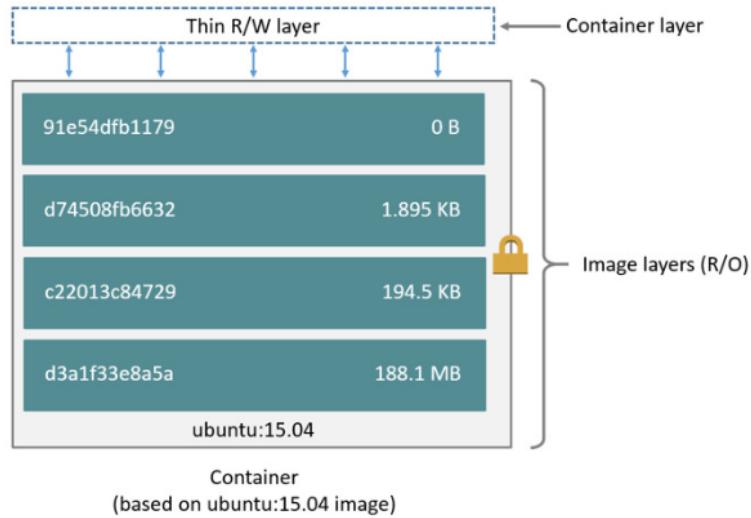


Figure 2.1: Figure showing an example of ubuntu:15.04 image layers[19].

The image is built up from a series of layers, as shown in figure 3.13. Each layer is only a set of differences from the layer before it. As git repositories have version control on software projects, layers have control of the different versions of the image.

When instantiating a new container, a new layer will be added on the top of the stack of layers. This layer contains all the changes made to the running container, such as new changes, modifications and deletions[19].

2.5.3 Docker Volume and Bind Mount

Docker provides two different mechanisms for persisting data generated and used by Docker containers.[\[49\]](#) This is useful when one has changes one wants to keep even when removing the container.

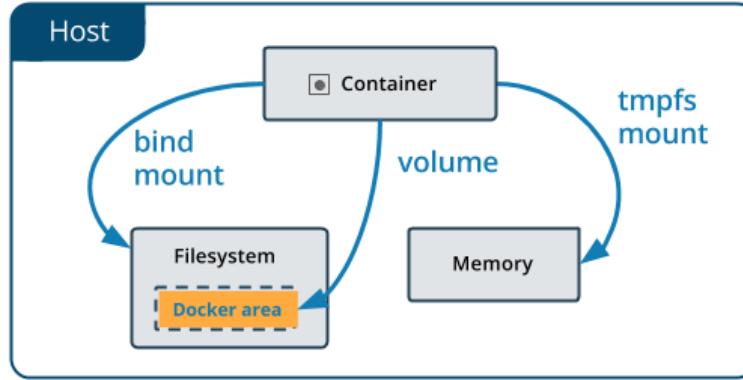


Figure 2.2: Figure showing how docker containers can persist and manage generated data[\[27\]](#).

Volume

Volumes are the preferred mechanism for persisting data. This is the best option when one is trying to persist in complex data like databases containing tables and rows. As shown in figure 2.2, these volumes exist outside the container in the host filesystem within a Docker area, which makes it easy to transfer the volume to another host if needed.

The volume is not dependent on a container. So if we were to delete the container that used this volume or even created the volume, the volume would still exist in the file system. One can attach another container to it if needed[\[27\]](#).

Bind Mount

A bind mount usually gets used to mount a file or directory on the host machine into a container. As one can see in figure 2.2 a bind mount happens on the file system outside of the docker area. It is therefore dependent on the host machine's filesystem having a specific directory structure available.

An example use of a bind mount is when one has a website one wishes to host. Then one can mount the website directory into an Nginx container. When a change occurs on the directory or file on the host machine, the website will automatically get updated inside the container[17].

2.5.4 Docker Registry

A Docker registry is a stateless, highly scalable server-side application that stores and lets one distribute Docker images. Docker has its official registry called docker hub, but one can create own docker registries, local and on cloud solutions, but then one should use the Docker Trusted Registry (DTR) due to security[23].

A docker registry works similarly to a git repository. It contains all the images (and their layers) that have been pushed to the registry. By comparing it to a git repository, the image itself is the source code while the layers are the commits. For every change pushed to the image a new layer is added on top of the stack. These images can then be pulled and used by others.

2.5.5 Docker Network

Docker networks is a powerful tool within docker that gets used to connect containers and services, or even connect them to non-Docker workloads. Docker provides several network drivers by default[20].

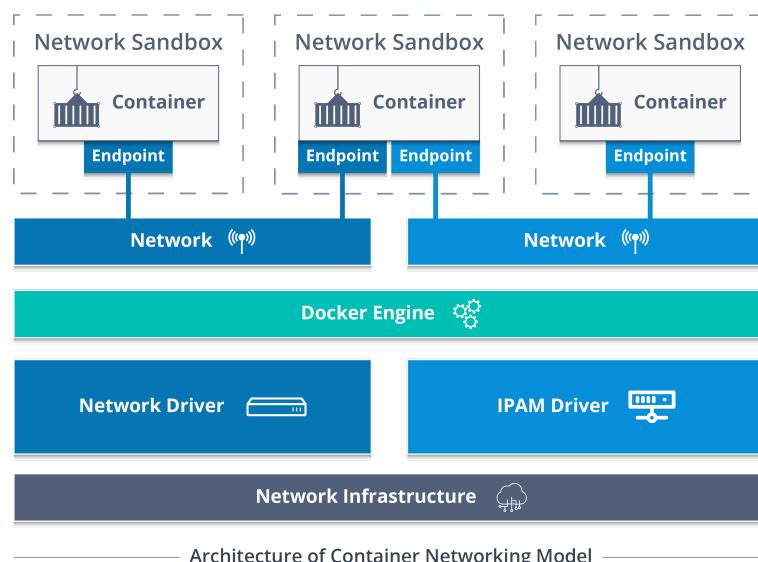


Figure 2.3: Figure showing a general docker network architecture[42].

The basic idea is that we use networks to connect containers or separate containers that do not have anything to do with each other. In figure 2.3, we can see a general network architecture.

We have two networks within the docker engine and three containers, two of them in separate networks and one connected to both networks. The container connected to both networks can communicate with the two other containers, but the containers only connected to one network can not talk to the container in the other network[20]. An example of this is when one wants to separate a web application, API and database. Then one can set the web application container in an own network, API in both networks, and database in the other network. This way, the API can connect to both containers, but the web application has no way of connecting to the database container.

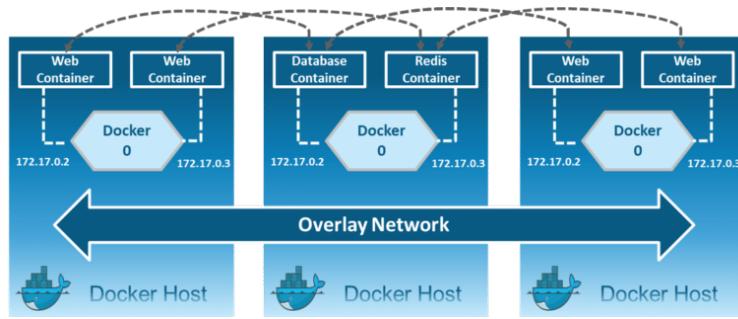


Figure 2.4: Figure showing a docker swarm overlay network architecture[42].

In docker swarm, we have a network driver called "overlay". This driver lets multiple docker hosts connect to the same network to enable communication between containers on different nodes[20].

Docker transparently handles routing of each packet to and from the correct docker daemon host and the correct destination container. By default, all the traffic between nodes connected to an overlay network gets encrypted using the AES algorithm. The different manager nodes in the swarm rotate the key used to encrypt the data every 12 hours[21].

2.5.6 Swarm

Docker swarm is a feature introduced in Docker engine 1.12. This is a feature that lets one create a cluster of one or more servers/virtual machines running the Docker daemon in swarm mode. There are two types of nodes. As shown in figure 2.5, we have Manager nodes and Worker nodes.

In a swarm, we also use the overlay network driver to make the different containers talk to each other despite being on different host machines, as shown in figure 2.15[18].

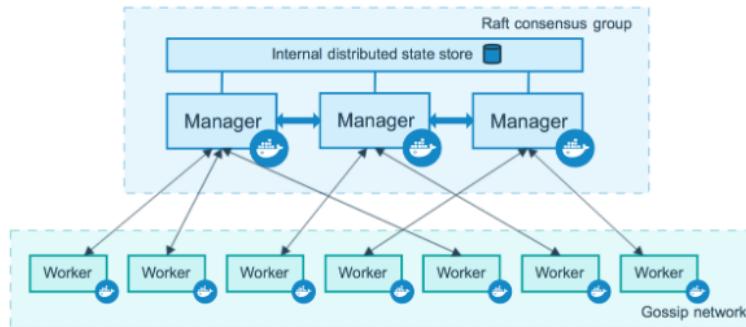


Figure 2.5: Figure showing a docker swarm with nodes[18].

Managers use a [Raft](#) implementation to maintain the state of the entire swarm and all the services (explained in section [2.5.6](#)) running in the swarm. In a test environment, one usually has one manager to distribute services and tasks to worker nodes, while in production, one has multiple manager nodes for fault-tolerance purposes[18].

Worker nodes sole purpose is to execute containers. One usually has multiple worker nodes containing replicas for the different containers making up the application to achieve a form of redundancy. This also makes it a lot easier to have a green-blue deployment.

Service

We can think of a docker service to represent a specification to acquire a desired state. In figure [2.6](#), we see that we have a service specification that instructs the creation of 3 Nginx replicas. When we create a service object, we specify what port the containers defined in the tasks should run on. We can also specify which overlay network the service should get assigned[25].

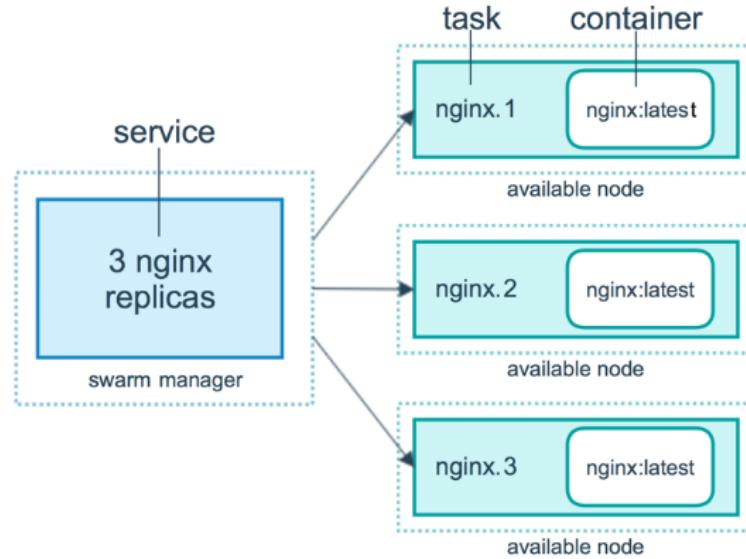


Figure 2.6: Figure showing a service object[25].

The service is then given to a manager node which will create tasks from the service object, and then instruct a worker to run that task, as shown in figure 2.7.

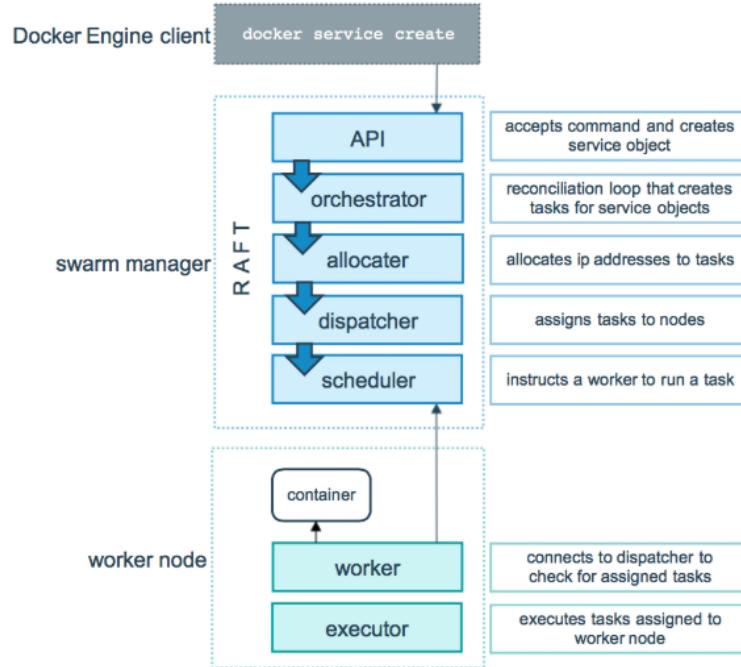


Figure 2.7: Figure showing the process of assigning a service to a manager and tasks being given to worker nodes[25].

Secret

Secrets are blobs of data which is used to store private information like a password, SSH private key, certificates or other data that should not be transmitted or stored unencrypted. Docker sends the secret to a swarm manager over a mutual TLS connection. The secret then gets stored in the Raft log, which is encrypted and replicated across the other managers[[24](#)].

Stacks

Stacks are the swarm alternative to docker-compose. By running "docker stack deploy" on any docker-compose file of version 3.0 or above one will create the necessary services and tasks specified in the docker-compose file. Even though stack gets used on a compose file does not mean one can use docker-compose and stack deploy interchangeably as they interpret the file differently, and docker-compose does not work with docker swarm. One usually test the configuration locally with "docker-compose up" first, if that works one can run "docker stack deploy" on a swarm manager node to orchestrate the services. Docker will automatically ignore the fields that are specific for docker-compose[[26](#)].

2.6 Spring Boot Patterns & Principles

2.6.1 Beans

A bean is an object containing metadata managed by the spring boot IoC container. This metadata contains the instructions on how to create the bean, its scope and lifecycle, the references to other beans (also called dependencies)[[30](#)].

2.6.2 Inversion of Control (IoC)

Inversion of Control, also known as dependency injection is a principle in software engineering that inverts the flow of control as compared to traditional control flow. It is a process where the objects define their dependencies through arguments set on the object instance after it is constructed or returned from a factory method. The IoC container then injects those dependencies when it creates the bean[[28](#)].

2.6.3 IoC Container

The Spring IoC container is the interface which is responsible for instantiating, configuring and assembling the beans. The container gets these instructions from the beans metadata[29].

2.7 Server Administration

2.7.1 Continuous Integration (CI)

Continuous integration is a practice for automating the integration of code changes in a software project. CI removes unnecessary time used on integration[52].

2.7.2 Continuous Deployment (CD)

Continuous deployment is a process for deploying changes that use automated testing to validate the changes made to the codebase, and deploys the solution to usually a staging environment first and then manually deployed to the production environment from which the users can see and interact with the changes. By having automated testing, rolling deployments, and proper monitoring, the maintainers will have a more stable and faster workflow as well as the user will have a lot less downtime. By containerizing the project, we also make sure that the application will work on any environment it gets deployed to[48].

2.7.3 Blue Green Deployment

Blue-green deployment is a model for releasing updates with zero downtime. This is done by redirecting the traffic from the old version of the service one wants to update to a replica that contains the newer version one wants to use. In the end, one transfers all the traffic to the new version and removes the old version. Docker swarm has a "build-in" feature that makes this type of deployment model easy to execute[38].

2.8 Database Concepts

2.8.1 Relational Databases

Relational databases are based on a relational model, which is an intuitive way of representing data in tables. The rows represent the actual data stored in the table, while the columns represent the attributes for the given entity. Each row has an attribute called a primary key which is a unique value used to identify the row/entry[46].

2.8.2 Entities and domains

In our spring boot services, we have two ways of identifying "objects" in our application. These are entities and domains. An entity represents an object which is stored in the database. So the entity is really the table, while entity objects are the rows within that table. The second term we use is domains. These are regular Java Classes that we use in the API for object oriented programming.

2.8.3 Repositories

In spring boot, we use repositories as an interface to communicate with the databases. These works differently depending on what type of interface one is using. In most of our APIs, we used JPA with JpaRepository, which automatically generated queries from methods containing a specific syntax within a class defined as a repository with @Repository.

Chapter 3

Materials and methods

3.1 Project Organization

3.1.1 Scrum

Our group agreed to use scrum as the working process due to the focus of meetings and team communication. This meant that everyone involved could be kept up to date, and it gives the flexibility in our project that we desired. We can adjust the workload that best suits us, compared to the waterfall method where its more difficult to move back to change something.

3.2 Work Organization

Before we started implementing the services, we planned how our infrastructure should look. So we created a simple diagram for visualizing how the different parts work together and the system as a whole. This diagram gets shown in figure [3.1](#).

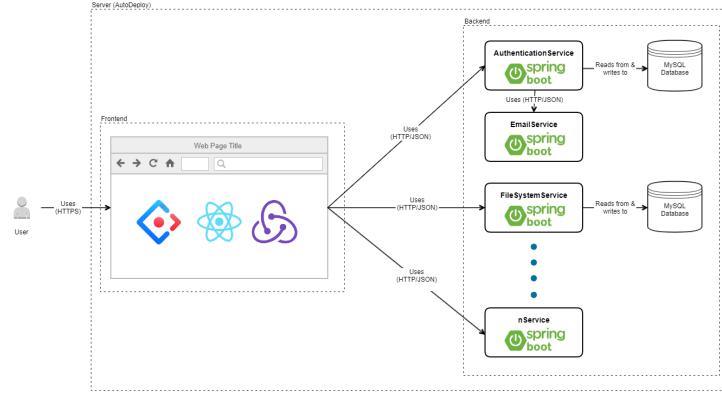


Figure 3.1: Figure showing the pre planned thought of our infrastructure.

From this, we built a module diagram that displayed all the parts we had planned for the bachelor. It included the services we needed to create as well as other parts like the different type of configuration files we needed and the administration part, as shown in figure 3.2. We also used this diagram to separate the workload. Each member got assigned specific modules to focus on, and then we assigned some modules to multiple team members.

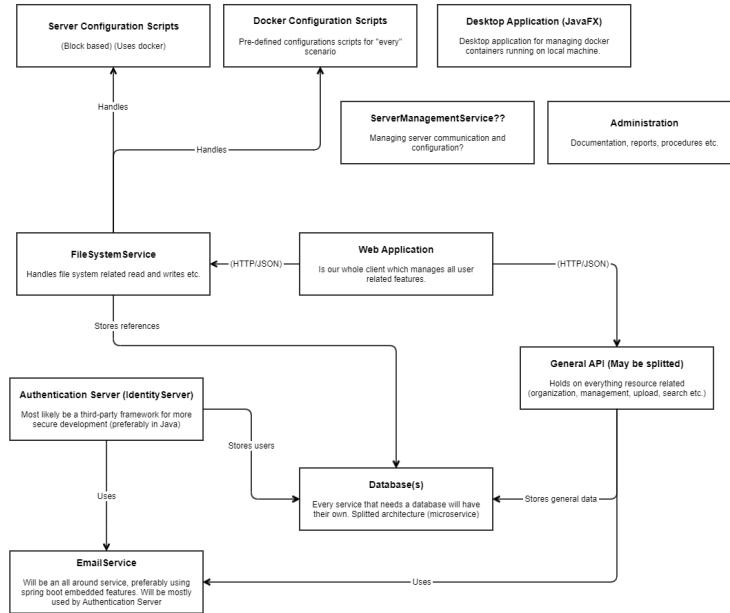


Figure 3.2: Figure showing our pre planned work on what modules we needed for our bachelor project.

3.3 Security Flaw

Because our users can upload any code, we have introduced a significant security flaw in our system. Since everyone can use our application and upload whatever they want, people can upload malicious software. We are aware of this and have some ideas on how to prevent it. The first idea is only to support open source projects so that others can see the code before downloading the code. The second option is to have moderators in organizations that have access to the code that is being uploaded to the organization and can then verify if the code is malicious or not. Another option is to have a closed platform (not what we want), that only people with an invitation key can use. This option might be a solution if this platform were to be organization dependent.

3.4 Programming languages

3.4.1 Java

We decided to use Java since we are all familiar with it and it is the standard programming language used at NTNU Ålesund. The main reason why we chose it, is because other students might want to continue working on this project as their bachelor project, or that other students and NTNU employees might want to develop further and maintain the software. Therefore Java is the obvious choice for us as it is easier for others to take over or collaborate on the project when we are finished. In figure 4.7, one can see a diagram showing the most used programming language which ensures us that the choice of Java was correct.

3.4.2 JavaScript

JavaScript has quickly become the most popular and used programming (scripting) language for web development because of its simpleness and the development of the most popular web development framework of today: React, Vue and Angular.

3.5 Command languages

3.5.1 Bash

Bash is a unix shell and command language, it look similar to a text editor, but it allows the user to perform actions from the users input. Bash can also read and execute files, these are called shell scripts, which makes it easier for developers to perform predefined actions on behalf of the user. We have to use Bash in order to execute Docker commands. [[https://en.wikipedia.org/wiki/Bash_\(Unix_shell\)](https://en.wikipedia.org/wiki/Bash_(Unix_shell))]

3.6 Package Manager

3.6.1 Yarn

Yarn is one of the most used package managers for JavaScript alongside npm. We chose to use Yarn over npm due to its speed. Yarn caches every package it downloads, so it never needs to download it again. It also parallelizes operations to maximize resource utilization, so install times are faster than using npm[13].

3.7 Frameworks

3.7.1 Spring Boot

Spring boot is an open-source framework and also the most popular framework for Java. Spring boot makes it easy to create Spring-based applications that can "just run". The main reason why we choose Spring as our framework is that we have previous experience with the framework. It is also the most popular framework for Java other than Java EE. Spring-boot uses an IoC Container for managing beans lifecycle and has a term called automagic which is used for all the automatic configuration that happens under the hood.

3.7.2 React

React is a open source web framework maintained by Facebook and a community of developers. It's one of the three most used frameworks creating single-page applications (React, Angular and Vue). React is only concerned with rendering data to the DOM, and so creating React applications usually requires the use of additional libraries. The most famous are named below[60]. We choose to use React due to its huge community and the endless amount of resources online. We also have some familiarity with the framework which makes it easier to start with then picking something that no one is familiar with.

3.7.3 React Hooks

React hooks got introduced in React 16.8, and as explained on the official react documentation: "Hooks provide a more direct API to the React concepts you already know: props, state, context, refs, and lifecycle"[34].

React hooks has quickly become the most used API for managing the concepts mentioned above and is a valuable asset. React hooks can only get used with functional components and not the usual class components. By no means does it replace class components, but for many, they are much more comfortable to use.

From experience, we know that react state can be challenging to grasp for newcomers. Especially if one has to pass state to child components or access the react lifecycle. So by using React hooks, we can remove a complex layer of management, and develop faster than before. We will use React hooks both for state management, accessing lifecycle, redux, routing and custom logic.

3.7.4 Redux

Redux is one of the must-haves when creating a application with React. Redux is an open-source JavaScript library for managing application state. When using redux with react you usually use react redux, which is the official react binding for redux. It lets your react components read data from a redux store, and dispatch actions to the store to update data[[51](#)].

3.7.5 React-router

React router is also one of the must-haves when creating a react based application. As react renders data to the DOM and redux takes care of application state management. React router takes care of the navigational parts of the application by providing a collection of navigational components.

3.7.6 Ant Design

Ant Design is one of the most used component libraries for React. With its growing community and interest, it's become the second most popular component library right after Material UI. This UI library (created by the Chinese conglomerate Alibaba) includes a vast collection of modern easy-to-use react components, which is one of the main reasons we chose it.

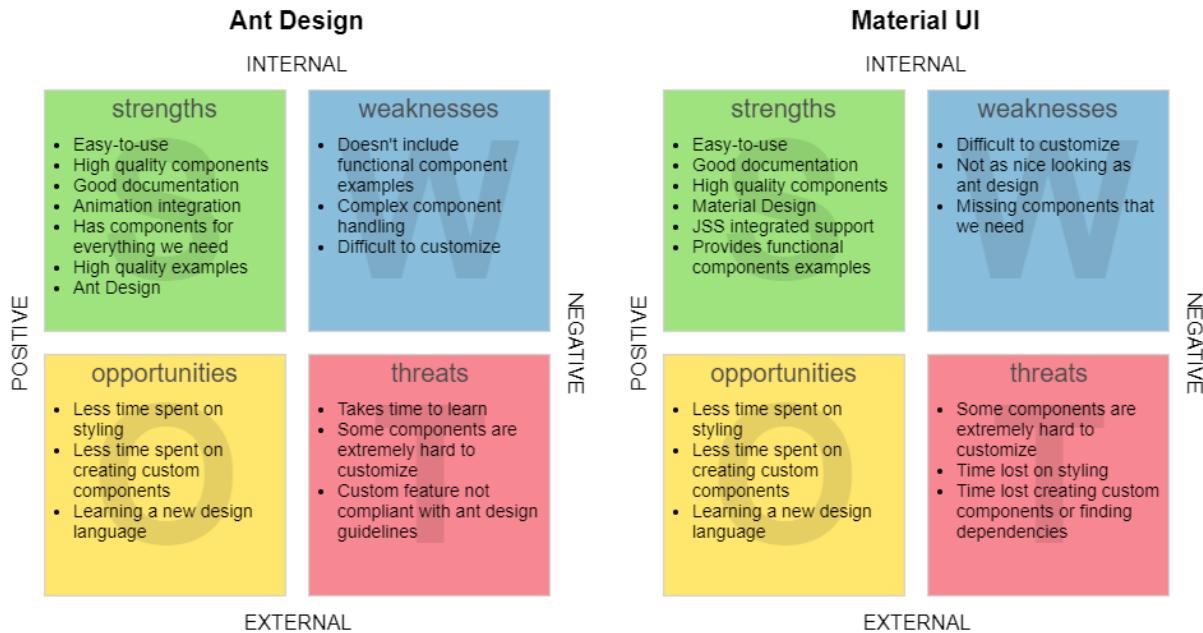


Figure 3.3: Figure showing swot analysis on ant design and material UI.

In figure 3.3, we have done a SWOT analysis for both Ant Design and Material UI, and then we compared them to find which did the best. The SWOT analysis is taken basis from own experience with Material UI and research for Ant Design. Both of them have quite similar strengths. They are easy-to-use; they deliver high-quality components and have excellent documentation. The main advantage of Ant Design is that they deliver several more components than Material UI, which was quite essential for us due to the fact we have limited time to implement an extensive application. We wanted to spend as little time possible on custom styling and finding dependencies for other components. There were multiple threats we had to consider. No one had any experience with Ant Design. The components they provide can be challenging to customize, and some might be immutable. We could also have a problem with implementing a component that does not meet the Ant Design guidelines.

3.8 Data

3.8.1 JSON

JSON is used to store data and uses markup structure. What makes JSON one of the most popular data storage language is because it is easy to use and is human readable and has a logical structure[58]. This is a format we have worked with a lot.

The markup uses key and value pairs to store data, where each key has none or many values. These values can be numbers, strings, boolean, objects, null and arrays. There are some limitations few of them being that one cannot store functions and dates[12]. There are workarounds for both, e.g., one can store dates as a string or number (as JSON do not support date data types) and create an interpreter on your application.

3.8.2 YAML

Similar to JSON, YAML is human readable. However, it is designed to be more understandable than JSON and XML[43]. YAML is mostly used as a configuration file and we will use it for Docker. It is simple to deserialize and process data to any programming language since it uses a key/value structure. Like UNIX configuration files, YAML also has one key/value pair each line, but has a possibility to extend to new lines using carrot symbol ">". One beneficial feature is that one can explicitly define what data type a value has inside YAML. The markup language the same data types as JSON with the addition of time and dates[56].

3.8.3 MySQL

MySQL is a database system by Oracle to store data. It is the most popular open-source SQL database system[47]. Each set or collection of data is stored inside tables where each data entry follows the standard structure of the table. MySQL uses SQL as language and is reliable, quick and flexible[54]. All members are most familiar with this database.

3.8.4 MongoDB

Unlike MySQL, MongoDB is a NoSQL database. It means that it does not adhere to the traditional relational database principles and does not operate with the query language found in SQL databases. MongoDB appeals to applications that require flexibility and storage of huge sets of data. Furthermore, the data is stored as JSON/BSON and uses key-value mapping to identify each record, which is called documents in MongoDB[2].

3.9 Project Management

3.9.1 Management Tools

Confluence

We use Confluence to document the work process and manage documents. Retrospective notes, meeting notes, requirements specifications, diagrams, wireframes, conventions and other types of documents.

Jira

To keep track of the work process and issue tracking, we used Jira. This tool is closely related to Confluence and makes it easier to work in sprints as well as having an apparent backlog of work that needs to be done.

Teamwork

We used [Teamwork](#) to create and manage our Gantt diagram, as it provided a powerful easy-to-use Gantt diagram design tool with collaboration support and task management system.

Draw.io

[Draw.io](#) is a free online diagram tool for creating all sorts of diagrams.

Visual Paradigm Online

[Visual Paradigm Online](#) is an online subscription-based tool that provides a large number of components, examples and templates for creating different types of diagrams.

3.9.2 Development Tools

Docker

Docker is a tool designed to create, deploy, share and run applications by using containers. Containers are processes instantiated by images (a package containing everything needed to run the application. libraries, tools, configuration settings and software) that runs in a virtualized environment configured by the Docker engine[59].

In a way, Docker is similar to virtual machines. However, instead of creating a whole virtual operating system, Docker uses the same kernel that the host machine is running and will only install the necessary libraries, tools and software needed for the application to run. Doing this, Docker abstracts the application's needs for software, tools and libraries installed on the host machine and only needs the ones specified in the container. This way a Docker container can run on any OS that has Docker support which makes testing, sharing and deployment much easier as a Docker container is consistent over every platform (as long as it has the Linux kernel. So for windows and macOS, Docker runs in a Linux VM)[45].

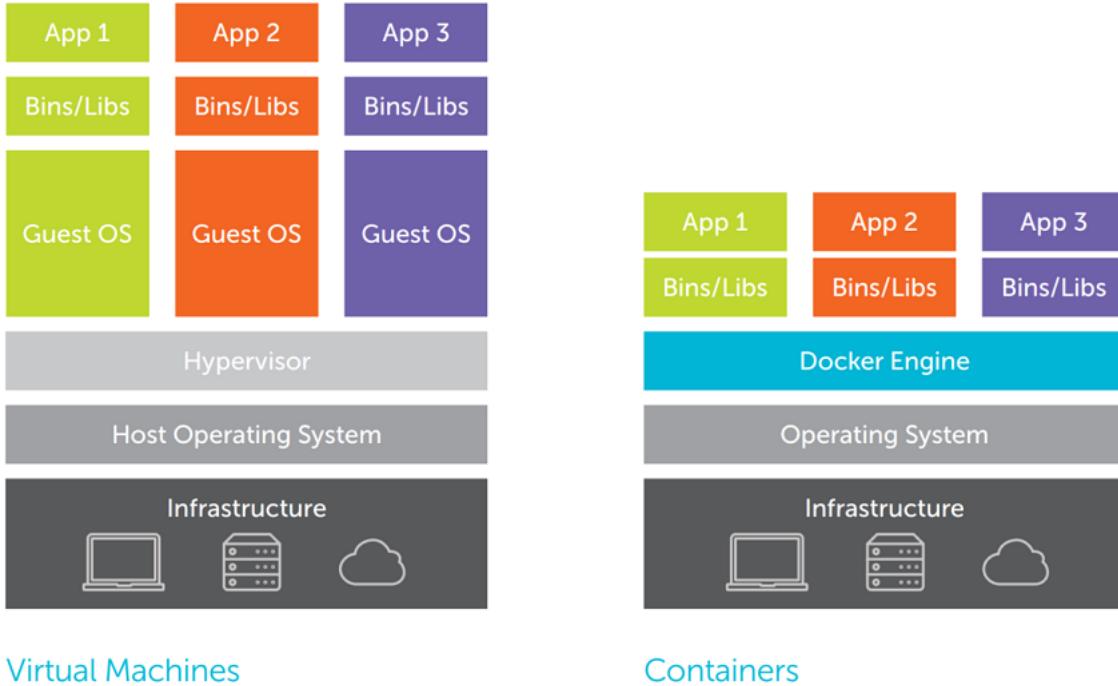


Figure 3.4: Figure showing the difference of virtualization between Docker and virtual machines[10].

The most popular alternatives to Docker are Vagrant and Virtual Box. Virtual Box is used to create virtual machines, and as described in the figure above, a Docker container is much more lightweight and portable as it relies only on a Linux kernel. Vagrant is used to create and configure lightweight, reproducible and portable development environments. In some way, they try and solve the same issue as Docker by removing the application's needs for the host machine installed software and libraries. The main difference is that Vagrant provisions any machine (typically a virtual machine) with the tools needed, and Docker containerizes applications[31].

In our case, we could have used both Docker and Vagrant for local development for those running Windows 10 Home. However, for Windows 10 Pro Docker provides an easy-to-use installer that sets up the environment and makes Docker easy to access. So all in all, for local development there are alternatives, but Docker has some solutions. Furthermore, for our Linux server, we can install Docker and run our application as containers without a problem.

GitLab

GitLab is one of the most popular web-based DevOps lifecycle tools. GitLab provides a Git-repository manager, issue-tracking system, kanban-boards, auto-deployment and a comprehensive CI/CD pipeline system. GitLab offers a ton of tools included, so one does not have to use multiple third-party tools to manage their application lifecycle. GitLab also offers free private organization creation which is the main reason why we went with GitLab versus GitHub. The second reason is that we are familiar with the GitLab pipeline tool from the course ID304112 System Administration[35].

Postman

We used Postman to test and develop our RESTful APIs. Postman delivers a sophisticated collaboration platform with many features which made development and testing of our APIs a lot easier and quicker.

Wireshark

Wireshark is a widely known and used network protocol analyzer, which we used to see what was happening on our network while using the application. Wireshark was used to see if correct messages got sent and received as well as to debug and fix errors[61].

IntelliJ

IntelliJ IDEA is a industry leading IDE (mostly for java) for software development developed by JetBrains.

Visual Code

Visual Studio Code is a lightweight, customizable code editor developed by Microsoft. We choose to use VS Code for developing the React web application due to its many extensions, customizability, efficiency and easy to use interface.

Git

Git is a distributed system for versioning source code. We choose to use Git as all the bachelor members have experience with this tool. We also used GitKraken, Sourcetree and Git CLI for interacting with the Git system. Every bachelor member was free to choose their favourite tool for versioning the bachelor project.

Database Management Tools

- **DBeaver** - [DBeaver](#) is a database administration tool to create, manage and modify existing databases and connects. Can also be used to generate ER-diagrams.
- **PhpMyAdmin** - [PhpMyAdmin](#) is a database administration tool for MySql and MariaDB.

SSH Client

We used the native ssh command in the terminal to connect to remote servers. This is the simplest and easiest way to connect to both our own servers and the servers used to test our application.

3.10 Existing Solutions Comparisons

3.10.1 Azure Comparison

Microsoft Azure delivers a massive library of tools to manage IaaS, PaaS and SaaS. While AutoPacker's primary function is to help build and configure projects as well as assisting with server and environment configuration and deployment, Azure is used for production-grade and production-ready applications. AutoPacker is by no means a platform for hosting production-grade software solutions. While Azure delivers their own servers for the user to use, AutoPacker gives the user the freedom to choose whatever he wants to use.

Azure also has a service called "App Service" that lets you build, deploy and scale web apps on a fully managed platform which is a bit same as AutoPacker and uses containers as well to run the code. As mentioned above, the difference is that AutoPacker is a platform-independent tool

with universities and students in mind that provides a transparent and helpful way to build, deploy and manage projects and servers.

3.10.2 Heroku

Then we have services like Heroku, which is a PaaS that lets users build, run and operate applications entirely in the cloud. One of Heroku's slogan is "Focus on your apps. invest in apps, not ops." Heroku will handle the operation side of deployment, while you, the user only needs to worry about developing your application[39]. Heroku delivers their servers with, of course, a payment option depending on size, power and other specifications. AutoPacker as said earlier, does not care about who provides the server, its capacity and other technical details. It is a tool for the user to reach the means to an end.

3.10.3 GidPod

Then we have GitPod, which is a SaaS online code editor that launches ready-to-code dev environments for GitHub and GitLab projects[36]. It lets one write, test and run code straight in the browser with automatic update and deployment to a hosted (testing) page using SSL for secure communication. GitPods editor is based on the online IDE Theia made by Eclipse. While GitPod is an excellent tool for development, making changes to a project quickly. AutoPacker is more of a platform to store and manage projects, as well as helping with building/packaging of projects and configuration of servers and environments.

3.11 Documentation

Everything that is documentation related will be stored in our Confluence area, which is neatly sorted into folder structure that makes it easy to navigate for information.

Supervisor / mentor meeting notes	For each meeting with the supervisor and/or mentor, a meeting record must be written that summarizes the objectives of the meeting, what was reviewed and what was concluded.
Log report	We can retrieve Jira's log report after each sprint documenting what we have done, how long it took and who did what.
Weekly report	Each week, a weekly report is written that summarizes what has been done so that both team members, supervisors and mentors are updated before meetings.
Routines	In cases where it is necessary to have a routine, it must be documented jointly so that everyone does the same without any inconsistencies.
Decisions	For each important decision that contributes to the modification of the task, that decision must be documented with: what is the reason, how do we solve it and possible conclusion.
Requirements specification	The requirements specification should define what user functions and general requirements the system should have. We use "user stories" to define user functions
UML Documents	If there is a need for UML charts and wireframes this should also be stored and described in our Confluence area.
Retrospective meeting notes	At the end of each sprint, the team will have a retrospective meeting where we go through the sprint internally. What has been done, what can be changed, how was the work process, what should we do next.
Milestone report	Every time we reach a milestone, we have to write a separate report that summarizes the functionality that is available and some kind of "release" summary

Chapter 4

Result

4.1 Architecture

4.1.1 Deployment Diagram

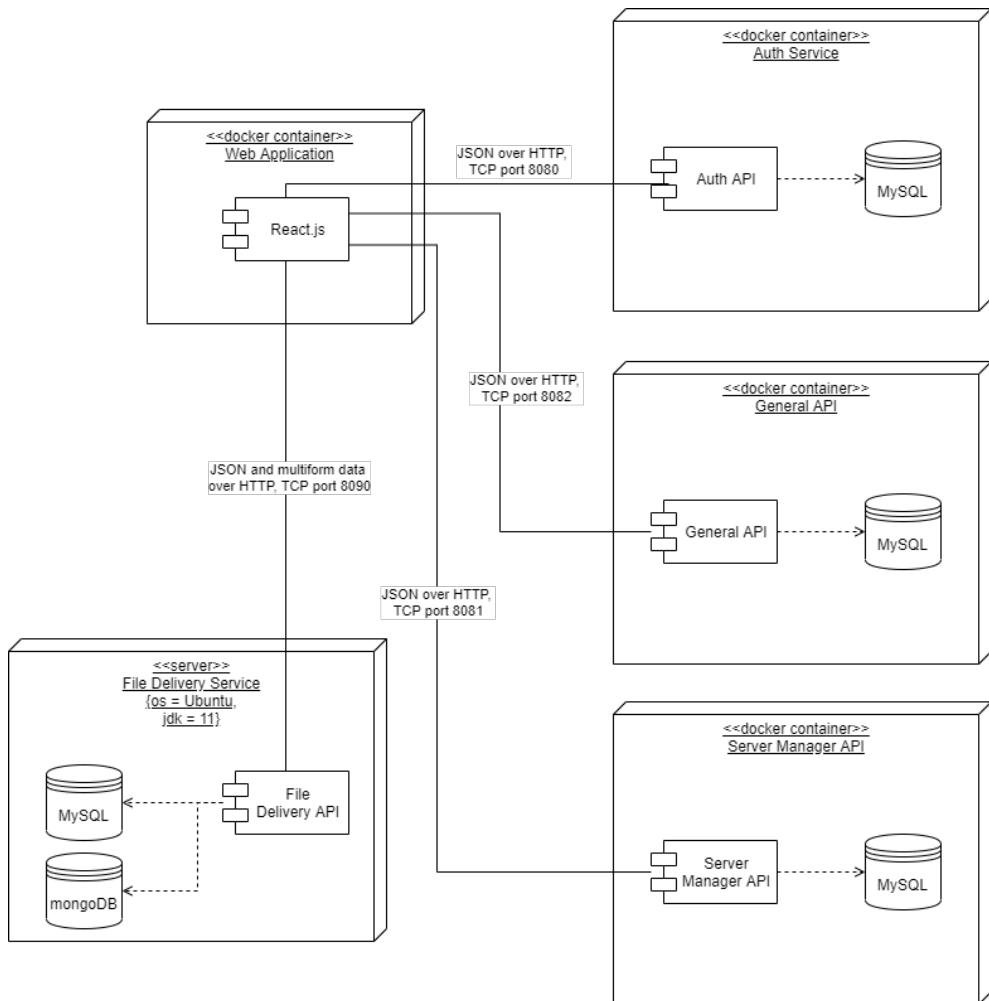


Figure 4.1: Deployment diagram of the system of micro-services.

The whole system got split up in 5 services, one web application and the rest are APIs. The APIs only communicates with the web app and never with each other. Most of the services are in docker containers and can run on any platform with docker installed. The exception is the File Delivery API, which needs to run outside docker on a Ubuntu environment with Java 11 installed.

4.1.2 Use Case Diagram

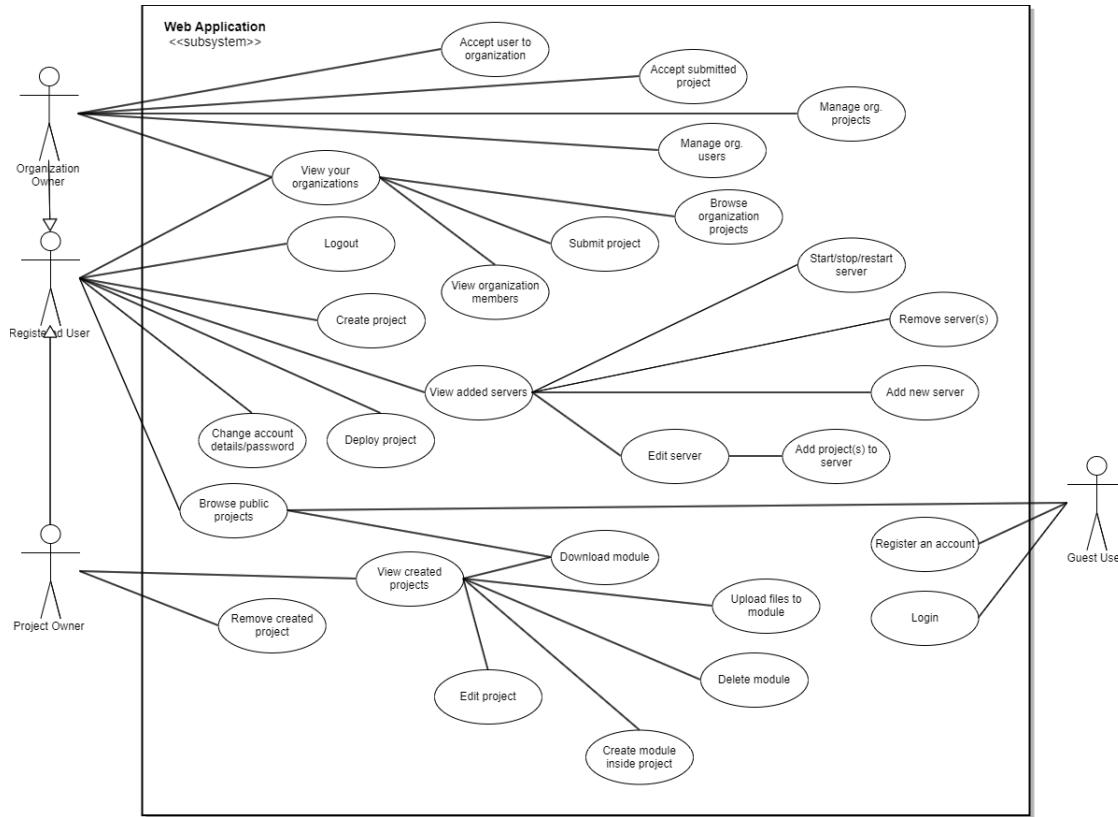


Figure 4.2: A use case describing what interactions different roles can expect.

Our system consists of different roles. When you first enter the website, you become a guest and have the possibility to create an account, login or browse public projects. Registered user is the default role and here you can create your own projects, apply to organization and manage your servers. You become a project owner once you create a project and when you create an organization, you become a organization owner.

4.2 Database overview

Since we are using a microservice architecture, every service has its own database. We make this work by using the users' username as a unique identifier as it is needed in most services as well. In the subsections, we will go through our databases, how they look, how they work and reflect on what could have been better.

4.2.1 Authentication Server Database

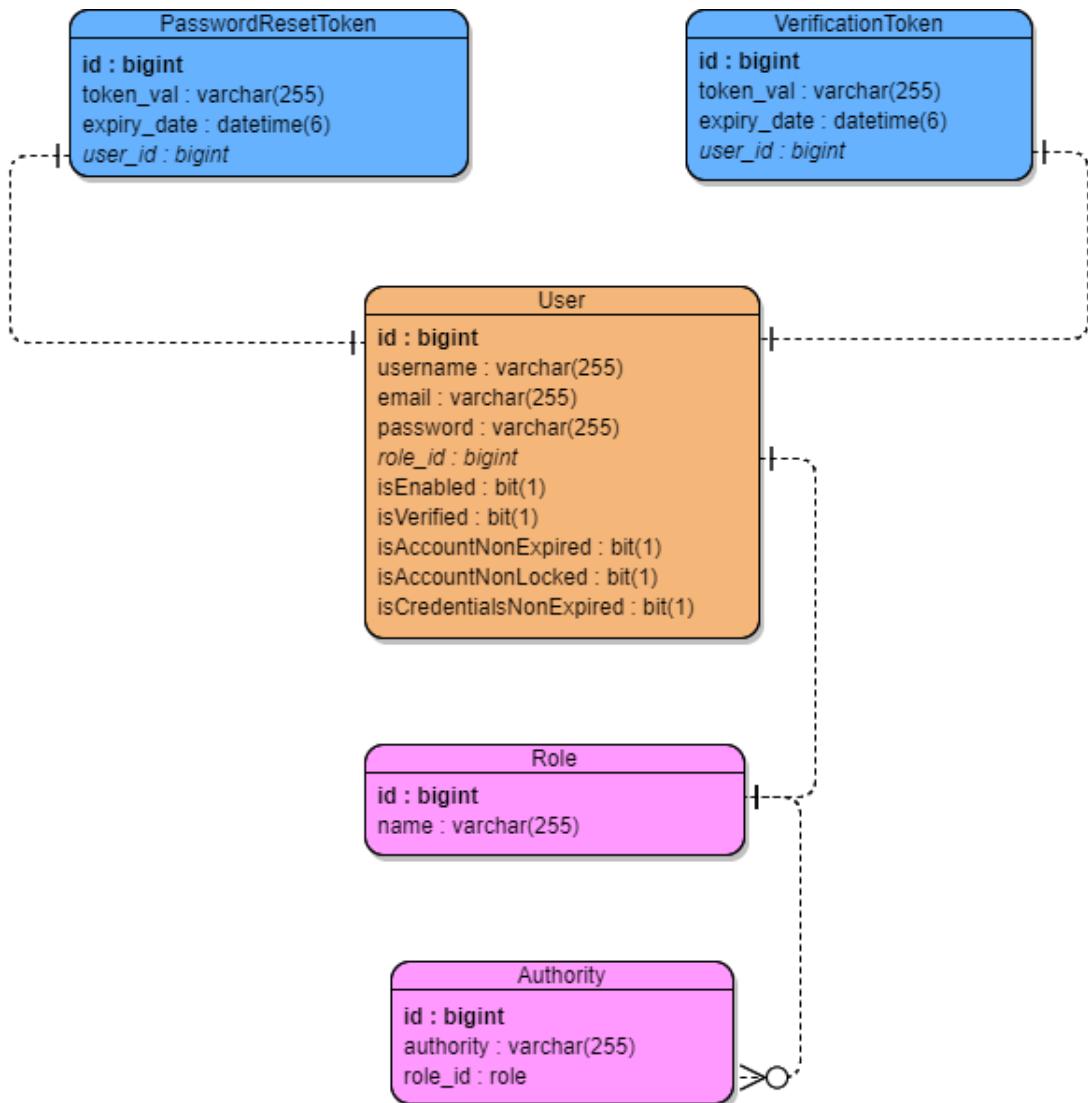


Figure 4.3: Figure showing the authentication server database.

The authentication server has a database with five tables. The token tables have a one-to-one relationship as the user can only have one valid token of each type. At the time of writing, the user can only have one role. This role has one or many authorities that defines what the user has access to do. A future feature would be to implement a following/follower type system, where users can follow one another. This can be implemented by extending this database.

4.2.2 Server Manager Database

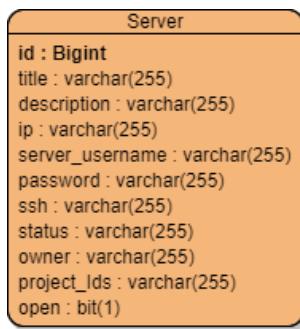


Figure 4.4: Figure showing the server manager database.

The only data the server manager stores are the information regarding servers, and the references to the projects that the server can deploy that are located in the file delivery database. Therefore the API only has one table, `server`, that holds all the information needed.

4.2.3 General Database

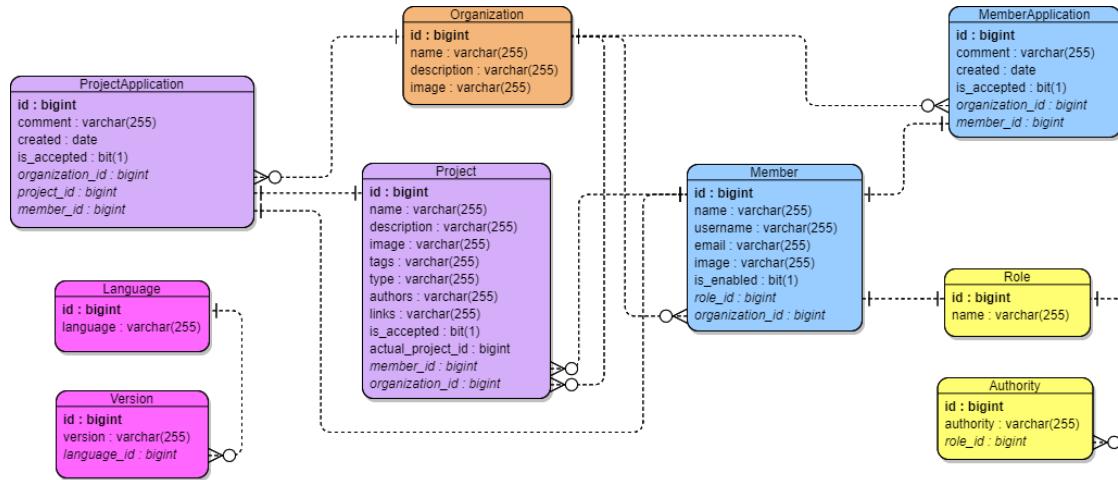


Figure 4.5: Figure showing the general service database.

The General APIs database holds the data for both the organization related information, but also the language and version support. The language and version data are easily managed by having a one-to-many relation so that a language can have many versions. The organization part, however, is a little bit more complicated. At first look, it might seem like much redundancy as both the authentication server holds information about the user, his role and authorities.

Furthermore, File Delivery API holds information regarding projects. The reason for this is that the organization system is more of a concept, and the organization might want to store more information about the user and the projects that our system does. The organization might also have their own specific roles and authorities, and therefore these tables exist here as well. An organization can as well have membership applications and project requests made from users that want to be or are members of the organization.

4.2.4 File Delivery Database

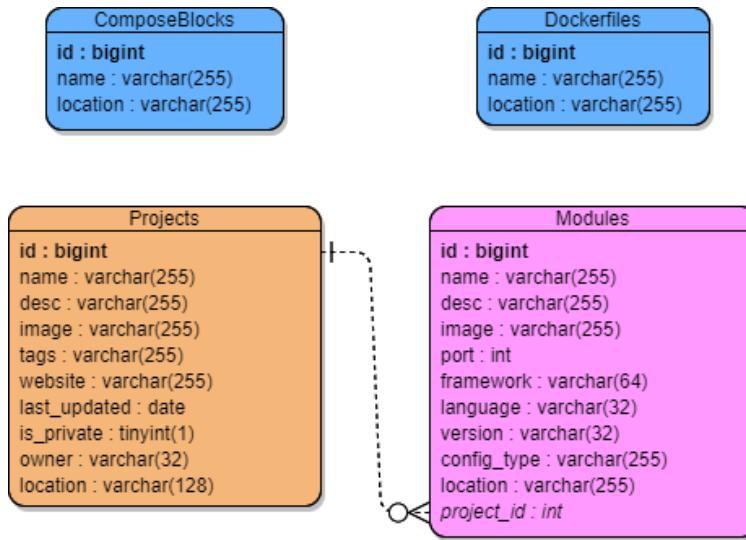


Figure 4.6: Figure showing the file delivery service database.

The file delivery APIs database holds the information for projects and their modules, but also the configuration files (docker-compose blocks and dockerfiles) needed to build and run uploaded projects. A project can have one or more modules, while the modules can have only one project (a module is unique to a project). The dockerfiles and compose-blocks tables hold the references to the actual files that live in the file system of our server. It is important to note that those tables are only there to index these files, so the API does not need to look through all the files on the hard disk to find it.

4.3 Technology Stack survey

We wanted to get an overview of what programming-/scripting languages students, hobby coders and professionals use for daily projects, student projects and hobby projects. To achieve this, we decided to create a technology survey with google forms to establish an overview. We sent the survey to 1st, second and 3rd-year computer engineering students at NTNU Ålesund as well as students at other universities and people in the industry. In total we got 33 responses where 16 (49%) of them were computer engineering students, 2 (6%) were other types of students, and 15 (45%) of them were either hobby programmers or employees in the industry. The reason

we wanted a wide range of different people, their profession and interest, is because we want the target audience for our application to be mainly students and hobby programmers, but also people in the industry that needs a fast deployment and testing solution. By including people from the industry, we also get data on what programming languages get used and what is popular, which might be interesting for the students to learn. So by gaining this knowledge, we can narrow down the initial support in our application to meet the specification we get from this survey. In the next sections, we will go through the results for each question.

4.3.1 Go-to programming-/scripting language for different projects

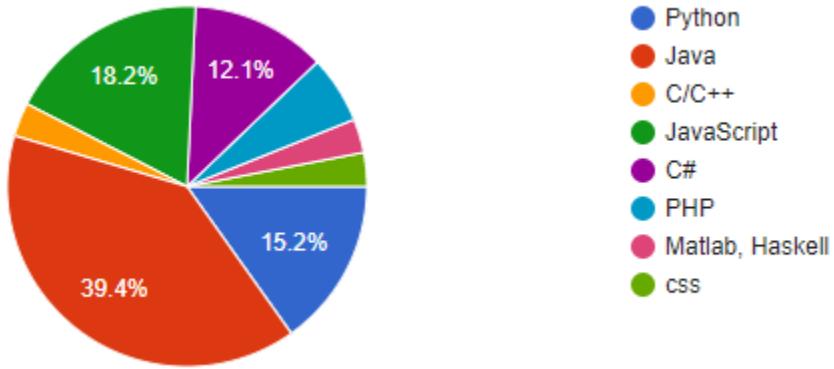


Figure 4.7: Figure showing the most used programming and scripting languages.

These results were expected as the most used programming language in courses, is Java. The top languages after Java are JavaScript, Python and C#. So when we were working on implementing upload and deployment support for different tools and languages, this is the order we used to prioritize.

4.3.2 Frontend

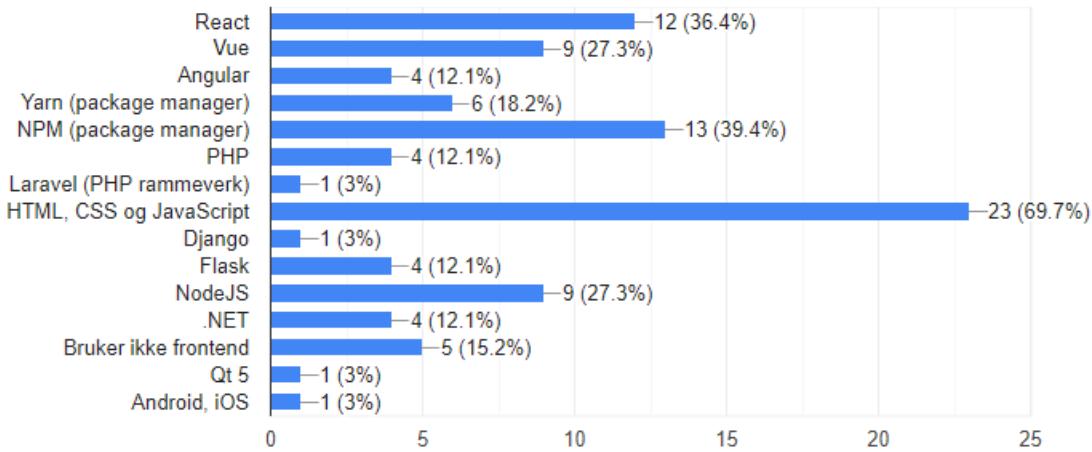


Figure 4.8: Figure showing the most popular tools and frameworks for developing frontend applications.

This question was a multiple choice question with the option of specifying other languages and tools as well. In this diagram we can see that of all the 33 respondents we had, 23 of them use regular HTML, CSS and JavaScript for developing frontend applications, which is not a shocker since these get used in conjunction with most frameworks out there. Of the significant frameworks, ReactJS is the most popular choice. For package managers, NPM is most widely used. In section 3.6, we explain the difference between Yarn and NPM and the reason why we choose Yarn over NPM.

4.3.3 Backend

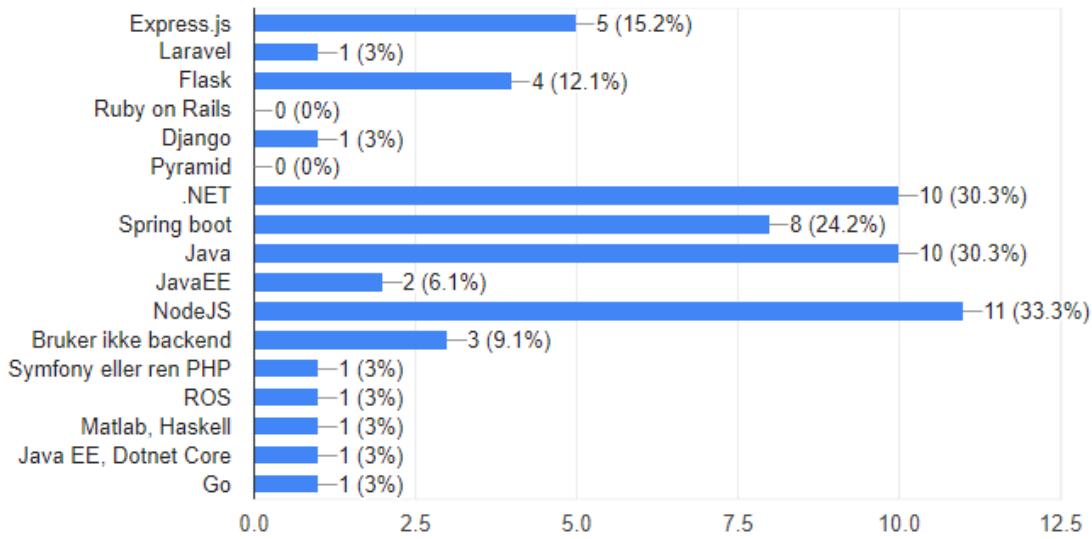


Figure 4.9: Figure showing the most popular tools and frameworks for developing backend services.

This question was a multiple choice question with the same option as before to specify other languages and tools. The most popular choice here is NodeJS, followed by .NET and Java, and then Spring boot. Due to the fact we are using Spring Boot, we focused on support for that and Java first, then NodeJS followed by .NET.

4.3.4 Database

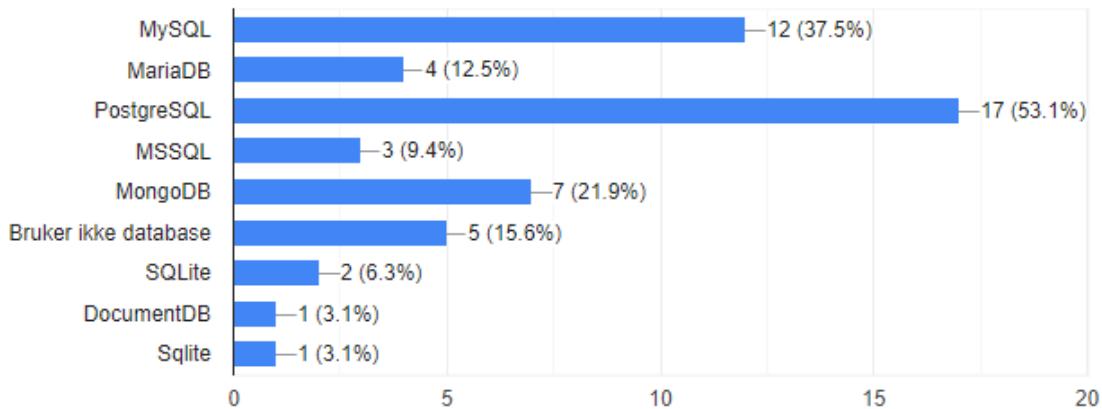


Figure 4.10: Figure showing the most popular databases.

This question was also a multiple choice question with the option to specify other database types. In this diagram, we can see that PostgreSQL is the most popular, followed by MySQL, MongoDB, and then MariaDB (skipping the "does not use database" responses). Since we are using MySQL, we choose to create support for that first, then PostgreSQL followed by MongoDB.

4.3.5 Server

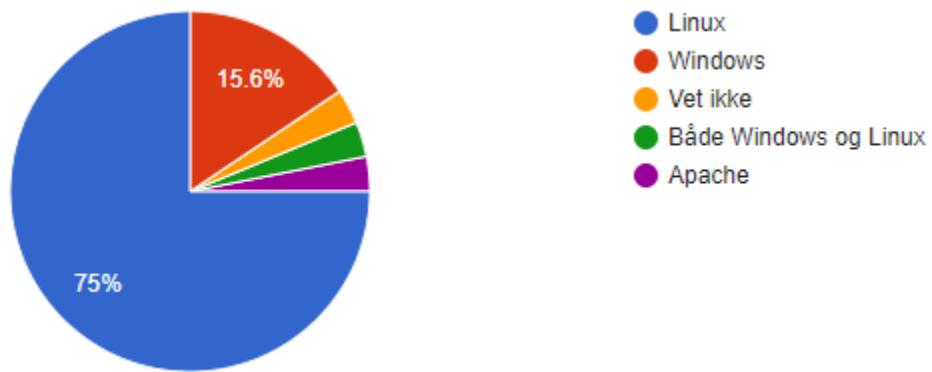


Figure 4.11: Figure showing the most popular servers.

This question is a single choice question and might have been unclear. The question was what server they use and if there are any specific versions which one. One answered Apache and another one "do not know" so we are ignoring those choices. So then we get that 83% uses Linux and 17% uses Windows. (excluding "do not know", "Apache" and "both Windows and Linux").

4.3.6 Extra thoughts/comment

Our last question was if they have any extra thoughts/comments regarding the application. Here we got four responses, and these have gotten taken into consideration.

4.4 Gitlab CI/CD

We used GitLab's' continuous integration (CI) & Continuous Delivery (CD) pipelines to build, test and deploy our code on given conditions. We choose to use Gitlabs' CI/CD pipelines as we have some experience with it. There is a ton of other alternatives here, but since our bachelor project was more of a concept we were developing, Gitlabs' solution was more than enough.

All of the services used the GitLab pipeline to continuously build, test and deploy when commits where made, and changes were merged. As we did not need any advanced integrations and customized handling for the different services, most of our pipelines were the same for all our services. Below is a listing showing an example of a gitlab-ci.yml file configuration.

```
1 default:
2   image: docker
3
4 services:
5   - docker:dind
6
7 stages:
8   - package
9   - build
10  - test
11  - deploy
12
13 variables:
14   IMAGE_TAG: autopacker/service:latest
15
16 package:
17   image: maven:3.6.3-jdk-11
18   stage: package
19   script:
20     - mvn package -DskipTests
21   artifacts:
22     paths:
```

```
23      - target/*.jar
24
25 build:
26   image: docker
27   stage: build
28   script:
29     - cp target/*.jar .
30     - docker build -t $IMAGE_TAG .
31     - docker push $IMAGE_TAG
32   before_script:
33     - echo "$DOCKER_TOKEN" | docker login --username $DOCKER_USERNAME --
34       password-stdin
35   when: on_success
36
37 testing:
38   image: tmaier/docker-compose
39   stage: test
40   script:
41     - docker-compose -f docker-compose.yml -f docker-compose.test.yml up -
42       d
43     - docker-compose down
44   when: on_success
45
46 deploy_staging:
47   stage: deploy
48   environment:
49     name: stage
50   script:
51     - ssh {USER}@{IP_ADDRESS} 'cd /home/dev/service/
52       && docker-compose down
53       && docker-compose pull
54       && docker-compose -f docker-compose.yml -f docker-compose.stage.yml
55       up -d'
56   when: manual
57   only:
58     - develop
```

```

56      - master
57 before_script:
58     # install ssh-agent
59     - 'which ssh-agent || (apk add --update --no-cache openssh-client)'
60     # run ssh-agent
61     - 'eval $(ssh-agent -s)'
62     # add ssh key stored in SSH_PRIVATE_KEY variable to the agent store
63     - echo "$SSH_PRIVATE_KEY" > /tmp/gitlab_ci_ssh
64     - chmod 600 /tmp/gitlab_ci_ssh
65     - ssh-add /tmp/gitlab_ci_ssh
66     # disable host key checking (NOTE: makes you susceptible to man-in-the
67     # middle attacks)
68     # WARNING: use only in docker container, if you use it with shell you
69     # will overwrite your user's ssh config
70     - mkdir -p ~/.ssh
71     - echo -e "Host *\n\tStrictHostKeyChecking no\n\n" > ~/.ssh/config

```

Listing 4.1: Example of a gitlap-ci file

In the top part of the listing (line: 7), we can see the four different stages in our pipeline.

- **package** - In this stage, we run mvn package to build and package the application into a jar file. This jar file is then set as an artefact with a path so it can be used in other stages.
- **build** - In this stage, we use the jar file created from the previous stage with the Dockerfile for that particular service (more on dockerfiles in the next section) to build a docker image and push the image to docker hub. The before_script: segment is run before the script and authenticates us to docker hub with a user we have created for this purpose, and its login token.
- **test** - This stage is currently unfinished. The reason with this stage is to run tests we have created and to see if the containers are successfully created and can communicate with each other.
- **deploy** - This is a manual step only available for commits pushed to either develop or master branch. In this stage, we deploy our update to the staging server (environment: stage).

We use an ssh-agent with a given ssh private key to connect to our staging server and pull the latest docker images from our docker hub organization and replace the containers that were running for that service with updated containers running the latest version.

In the first pipeline we created we used a Gitlab runner. This is an excellent solution if one runs one pipeline on one server. When one uses Gitlab runner, one needs to install some software on the platform the runner is going to run on. In our case, it was the testing server. Then one can create an own user for the GitLab runner. In the gitlab-ci file, one specifies what it should do when it is executed. So if we were to push an update to our application, the Gitlab pipeline would execute, and the GitLab runner would automatically run and execute its tasks on the server it has been assigned. In our case, as we have multiple pipelines and will have multiple servers to deploy to, this was not the best way. So instead we choose to run everything we could in the pipeline and use ssh-agent to connect to the servers with ssh private key and then run the scripts defined in the gitlab-ci.yml configuration file.

4.5 Backend Services

4.5.1 File Delivery API

Authorization

Roles and permissions were not implemented. Instead, the project owner is the only one who has rights to do changes in a project and all its contents. If the project is public, everyone can view them, if not, no one else can view them except the project creator.

File Explorer

We realised that adding a file explorer feature may be too much work and we had work that had much higher priority. At the start, we did implement a “workspace”, i.e. dedicate a folder for each user containing their projects and modules. These folders are mapped directly to the URL. That made it easy to separate projects from each other and made it easier to add future imple-

mentations.

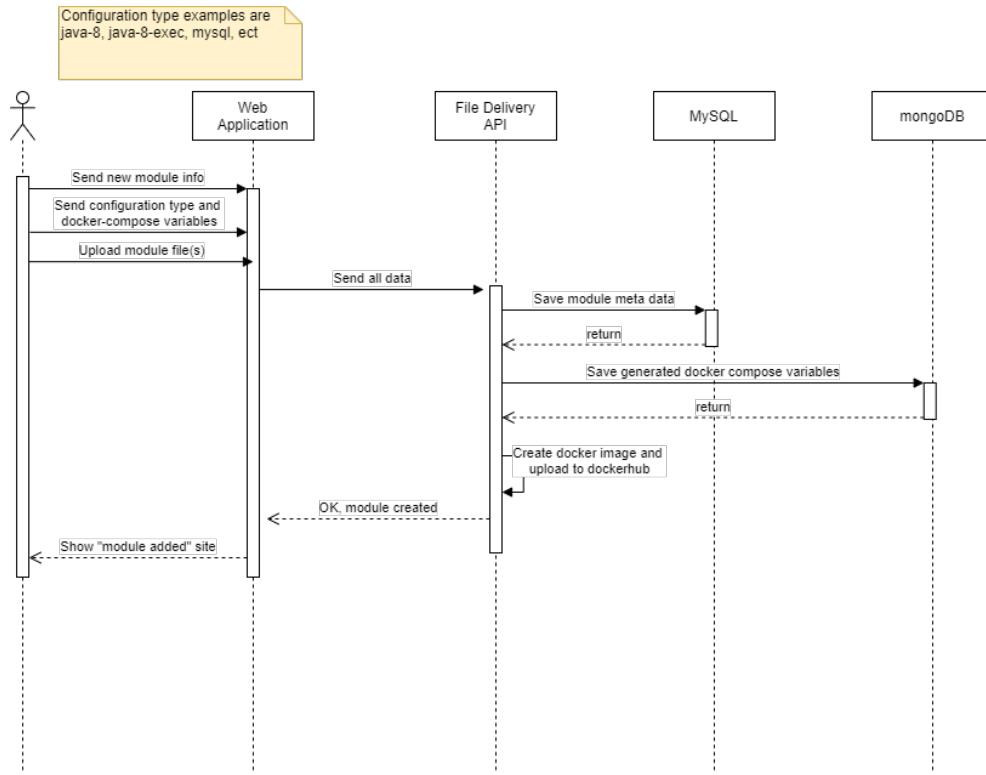


Figure 4.12: A sequence diagram of creating a module after a project has been added.

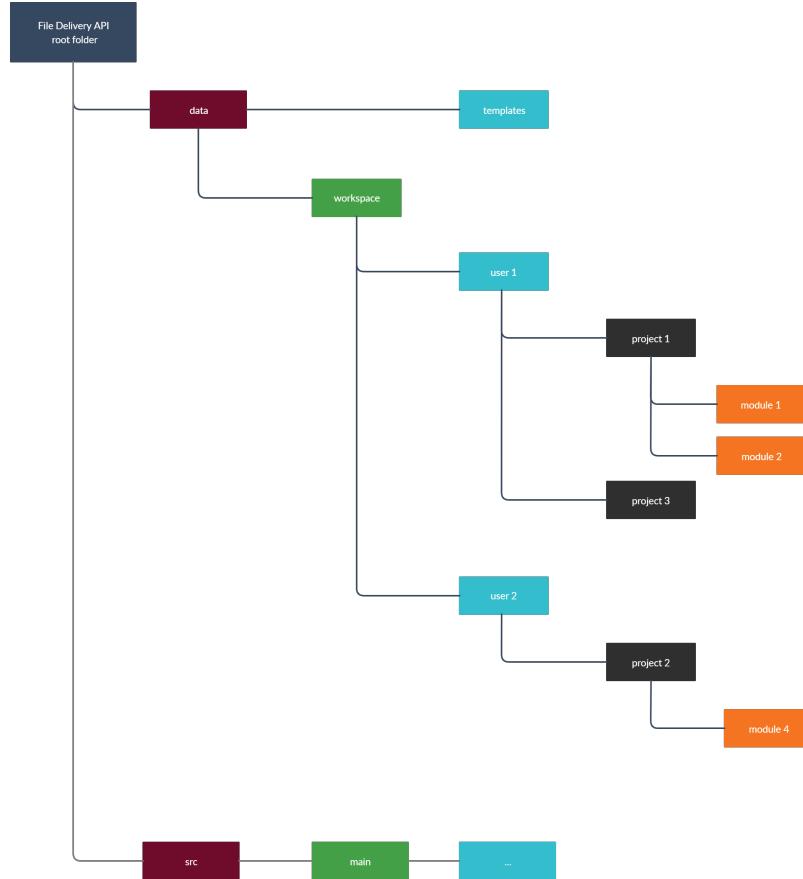


Figure 4.13: Folder structure of the API, where the focus is on the separation of users, projects and modules.

Docker-compose template builder

The builder uses two types of file for docker-compose to create a docker-compose file for a project:

1. The main docker-compose file. This defines the first scopes variables, ex. version: 3.3.0 and services:

```

1  version: "3"
2
3  services:
4    ${SERVICES}
  
```

Figure 4.14: The base docker-compose template where only variables on the highest scope is defined.

2. The server docker-compose block. This is the template for each service, and the set of variables depends on what service is being dealt with. For example, a java service has different variables than a MySQL database service.

```

1      # Module java-8
2      ${CONFIG_TYPE}:
3          container_name: ${CONTAINER_NAME}
4          image: ${IMAGE_NAME}
5          ports:
6              - ${PORT}:${PORT}
7

```

Figure 4.15: Docker-compose template of the Java 8 service.

Before we explain the main function for this job, we must know how the building works.

General builder functionality

The general gist of the docker-compose compilation is that each module in a project has a configuration type, i.e. Java, MySQL, React set when the module is uploaded. The configuration type is linked to the corresponding docker-compose template block with their unassigned variables. The uploader of the module has already defined these variables and is stored. When someone requests the docker-compose.yml for a project, using the function, merge the compose template block with their variables. Lastly, the template block with the variables are put under the service section of the main docker-compose file.

Here we have a code example of the main function of layering the docker-compose files.

```

92     /**
93      * Takes in the template content as string and replaces the placeholder with the values from the JSON object.
94      * NOTE: The placeholder keys and the json object key must be the same for the placeholder you want to switch out.
95      *
96      * @param templateContent template content
97      * @param json json object with all the values
98      * @param validate if true, this switch casts an exception when all placeholders has not been replaced
99      * @return a string with the template content replaced with the values from json
100     * @throws JSONException if json data couldn't be processed
101    */
102   public String buildFromTemplate(String templateContent, JSONObject json, boolean validate) throws JSONException {
103     // Format: ${VARIABLE} and its case-insensitive
104     Pattern pattern = Pattern.compile("\\$\\\\{(?i)([A-z0-9]*)\\\\}");
105     Matcher m = pattern.matcher(templateContent);
106     StringBuilder result = new StringBuilder();
107
108     while (m.find()) {
109       String key = m.group(1);
110       String value = null;
111
112       if (json.has(key.toLowerCase())) {
113         value = json.getString(key.toLowerCase());
114       } else if (json.has(key.toUpperCase())) {
115         value = json.getString(key.toUpperCase());
116       }
117
118       if (value != null) {
119         m.appendReplacement(result, Matcher.quoteReplacement(value));
120       }
121     }
122
123     m.appendTail(result);
124
125     // Check if placeholders weren't replaced and throw exception if so
126     if (result.toString().contains("${") && validate) {
127       throw new JSONException("All placeholder values has not been assigned");
128     }
129
130     return result.toString();
131   }
132 }
```

Figure 4.16: Main template builder code snippet from BuilderService class.

Parameters

The function takes in the compose template block as a string, usually read from a file. Variables for the template are sent as a map, where the key of the map is the placeholder value of the template. The last boolean parameter is used to check if all the variables inside the template have been assigned.

Placeholders and their replacement

In the beginning, we create a [regular expression](#) to find and target the placeholder values that will be replaced. For this part, the symbols of the placeholders are ignored. For each of the

placeholder, it finds the key that has the same name as the placeholder and replaces it with the value from the map. The function ignores if the variable map keys are capitalized or not.

4.5.2 Validation

Before the built template is sent, depending on the validate boolean, it checks if there are remaining placeholders that aren't replaced. If there are placeholders that still remains, it throws an exception. The reason behind this check is that it is common for the system to leave few placeholders, especially when the system tries only to compile all the compose block templates, and not put in all variables yet.

4.5.3 MongoDB

In addition to a MySQL database, we have MongoDB. The purpose of MongoDB is to store the variables that will be combined with the docker-compose templates. It is combined to a full docker-compose.yml when a client wants to deploy the project. Examples of variables are in [4.15](#).

Each compose blocks has its own configuration type. The configuration type is linked to a set of variables. The variables may be similar to other variables inside other configuration type and is not a guarantee. For example, a spring-boot configuration may require a port and a reference to an SSL certificate, and a MySQL configuration may require port and database credentials. It is possible to store the variables inside a MySQL table where each configuration type has its table with variables. However, one must declare the table structure before one can insert the variable values. In MongoDB, one only need to add the document in the collection (where a document is the same as a MySQL row and collection as a MySQL table,) and the column structure of the collection will automatically be made. If documents with new columns are added, the collection will also automatically expand. This made it easier to work with MongoDB compared to MySQL since much work is automated by default.

For now, the variable values for a template is set by whoever uploaded the module. It makes the implementation of the MongoDB less intuitive. The original intent was that every user who

wanted to deploy an existing module/project could define the values of the variables. Although we did not make it like that, the current implementation is set up so that the feature can be extended that way.

4.5.4 Server Manager

Server Manager is the service that handles all server related features. From adding servers, installing essentials and adding/deploying projects to a server.

Server Initialization Script

For a server to be able to run user projects, they need to have both docker and docker-compose installed. For this, we created a preparation script for Ubuntu servers that installs the essential tools as well as enabling all incoming and outgoing connections. We did this to make sure that all ports and transmissions were able to open. As a future feature, we would like to implement a management tool for managing the server ports and other server configuration from this service.

Connecting and transmitting data

For connecting to remote servers and exchanging data between the connections, we used a library called [JSch](#) (Java Secure Channel). Using this library, we could transfer files and execute commands on a remote server through an SSH connection.

```
channel.setCommand(
    "sudo docker-compose -v; touch docker-compose." + username + "_" + project
    + ".yml; curl -X GET " + dockerComposePath + " >> docker-compose."
    + username + "_" + project + ".yml; sudo docker-compose -f docker-compose."
    + username + "_" + project + ".yml up -d");
runCommand(channel, server);
```

Figure 4.17: Code snippet showing an example of using JSch.

Project Deployment

In figure 4.18, we can see a sequence diagram showing the sequence in which the actions get executed when deploying a project on to a server. After the user has selected what project to deploy the server manager will connect the target server which will then request the docker-compose.yml file for building and starting the project from the File Delivery API. Then the server manager will execute a series of commands that will, in the end, run the project on that target server using the fetched docker-compose.yml file.

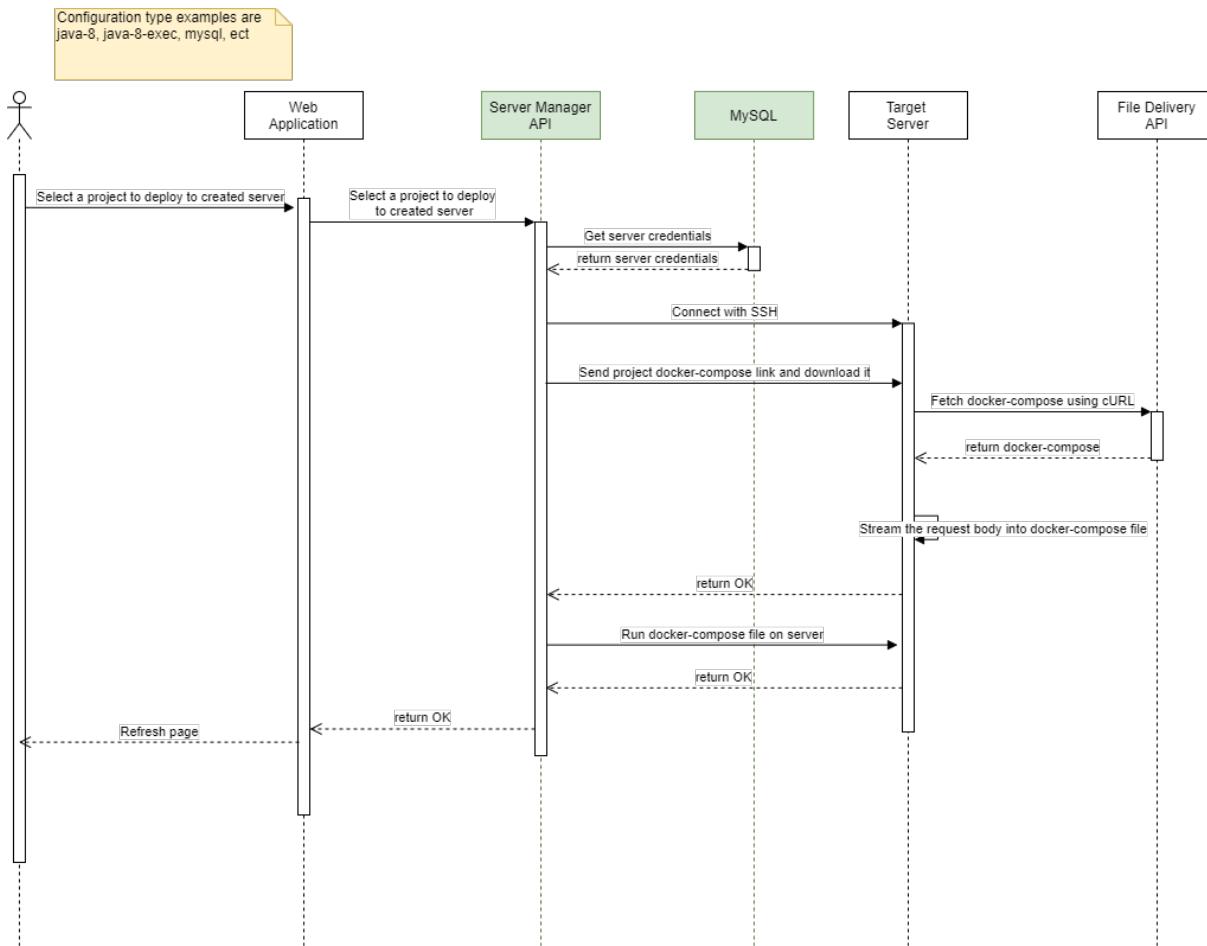


Figure 4.18: Sequence diagram for deploying a project on to a server.

4.5.5 Authentication Server

Authentication Server is responsible for handling all user-specific features and authentication. It takes care of registration, authentication, password changes and resets as well as JWT token management.

JWT vs OAuth2

We started out developing a working authentication service using JWT but had planned to change and use OAuth2 with JWT. By using the OAuth2 protocol, we could have integrated Feide with our service. OAuth2 is something we have planned on implementing if we are to continue development on the platform.

Registration

In figure 4.19, we see the sequence of actions when registering a new user. When the user has submitted the registration form, the data gets sent to the Authentication Server where it gets validated. If the form is valid, the service will persist the user to the database and issue a JWT, which gets returned to the user.

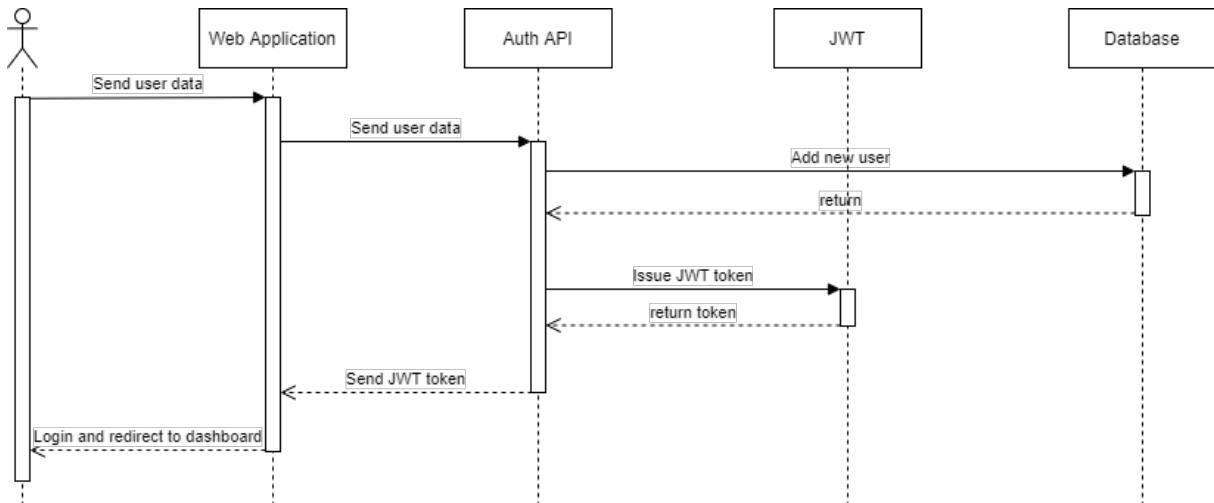


Figure 4.19: Sequence diagram for registration.

Authentication

In figure 4.20, we see the sequence of actions when the user is authenticating himself. The credentials get sent to the Authentication Server, which will check if they return a user. If the credentials are valid, the service will issue a JWT, which gets returned to the user. If the credentials are invalid, the service will respond with HTTP status 401 Unauthorized.

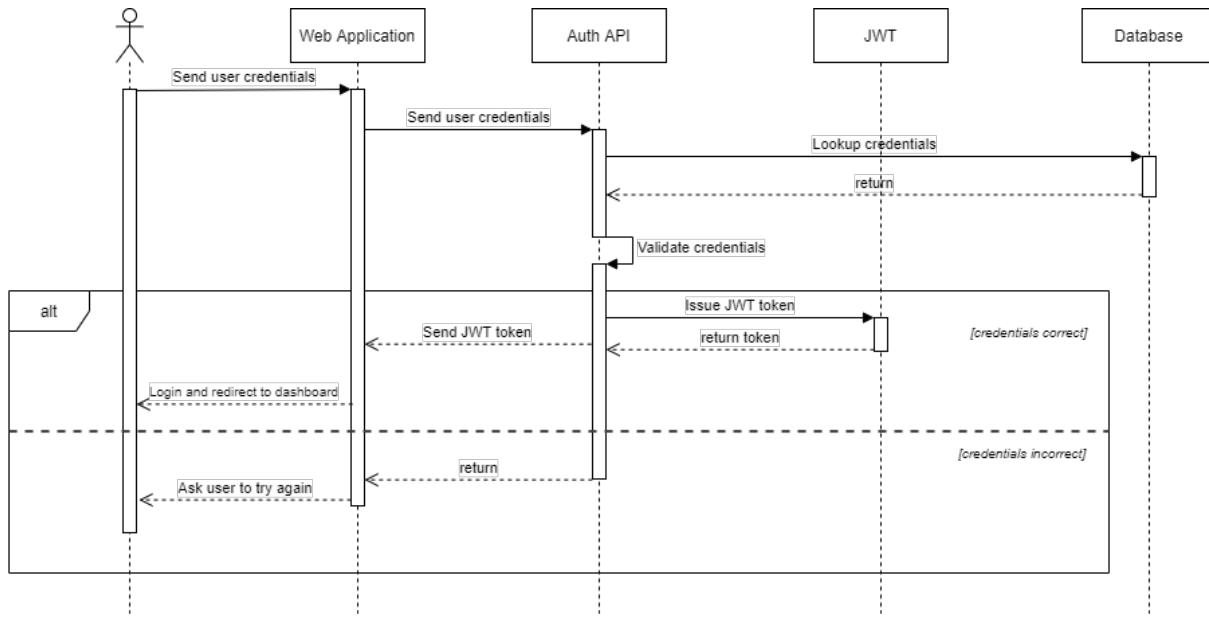


Figure 4.20: Sequence diagram for authentication.

Authorization

There are parts of the other microservices that has restrictions. We do not want everyone to access every possible actions. There are multiple levels of security and these are: authentication, user affiliation rights and user roles.

To perform any modification on a project or a module, it's required that the user is authenticated and be the one who created it. Other people can use it granted the project is set to public. To access those project, you do not need to be logged in. Users with the role "Admin" can modify any project despite not having any development ties to the project or module. All of the endpoints for deployment of project to a server can only serve users that has been authenticated. Users can create, manage and delete servers they have provided.

4.6 General API

General API started as a general-purpose API, where we implemented features that were supposed to be general. It first handled all the supported language and versions related features, but now holds all the logic for organization related functionalities.

4.7 Web Application

Our web application is a standalone React application that communicates with our RESTful services through HTTP(S). We use React Router for routing, React Redux for managing application state and Ant Design as our component library so we can easily access high quality modern react components to use in our application.

We ended up with an extensive modern web application that both looks good, meets our minimum product requirements and is built to be scalable and contains some logic that is ready to use if we were to extend the system with the features.

We have a test-version up and running at <http://stage.autopacker.no>. Do not use personal password as the site is currently only running HTTP.

4.7.1 Routing

We use [React Router](#) to manage the routing in our application. The routes are defined and structured in the App.js file, which is the entry point for the application. The current management of routes is messy and not that efficient. A better way would be to group and nest routes that have something in common. This way, we can create a more dynamic, efficient and more readable structure. We could also then define different "404 Page Not Found" warnings or redirects for the different scenarios.

Another great feature which we had planned to implement, but did not have the time to, is lazy loading. Our application is built up of many different pages and routes, and some of them might never be used in one user session. Therefore it would be logical to implement lazy loading on almost all of our pages to increase efficiency and remove unnecessary redundancy.

This is easily done by using [Suspense](#), but as stated we did not prioritize as it is a "nice to have" sort of feature.

We also created two custom routes for our application. In our application, we have a lot of similar pages which contains the same logic as the navigation bar and side panels. By creating custom routes, we can pre-specify these components to render for specific routes and remove redundancy. We could also add custom logic which is executed when the route is mounted as seen in figure 4.21.

```
const ProfileDashboardRoute = ({ component: Component, isAuthenticated, ...rest }) => {
  return (
    <Route
      {...rest}
      render={(props) =>
        isAuthenticated === true ? (
          <ProfileDashboardLayout>
            <Component {...props} />
          </ProfileDashboardLayout>
        ) : (
          <Logout />
        )
      }
    />
  );
};
```

Figure 4.21: Figure showing the custom route for a profile dashboard component.

Here the route will act as a regular route retrieve values used to set properties, but it will also check if the user is authenticated (This only checks if the token !== null). If the token does not exist the route will render the `<Logout />` component which will execute the logout logic and redirect the user to the homepage. If the token does exist the route will render the `<ProfileDashboardLayout />` with its children, which is the layout showing the navigation bar and side panel shown in figure 4.33.

4.7.2 Application State Management

We have three different main ways of managing state in our web application. The most used is, of course, the component that the state regards will hold the state itself, and in some cases might pass the state update method as properties to a child component.

The second most used is the use of Redux for application state management. By using Redux, we can store the state we want to be accessed globally in a Redux store which is manipulated by dispatching actions. Example of fetching a value from Redux state as well as dispatching an action is shown in figure 4.24.

In some cases, we have a parent component (more like a container) that holds n number of components that alternates between being used and not. An example of this is when a user is uploading one or more modules, as shown in figure 4.38. In this case, we might save the majority of states in the parent component and pass them as properties to be used and updated by the child components. Some of the child components may have their own state as well, which we might want to store as a JSON object in parent state, unavailable to the rest when the component gets swapped out.

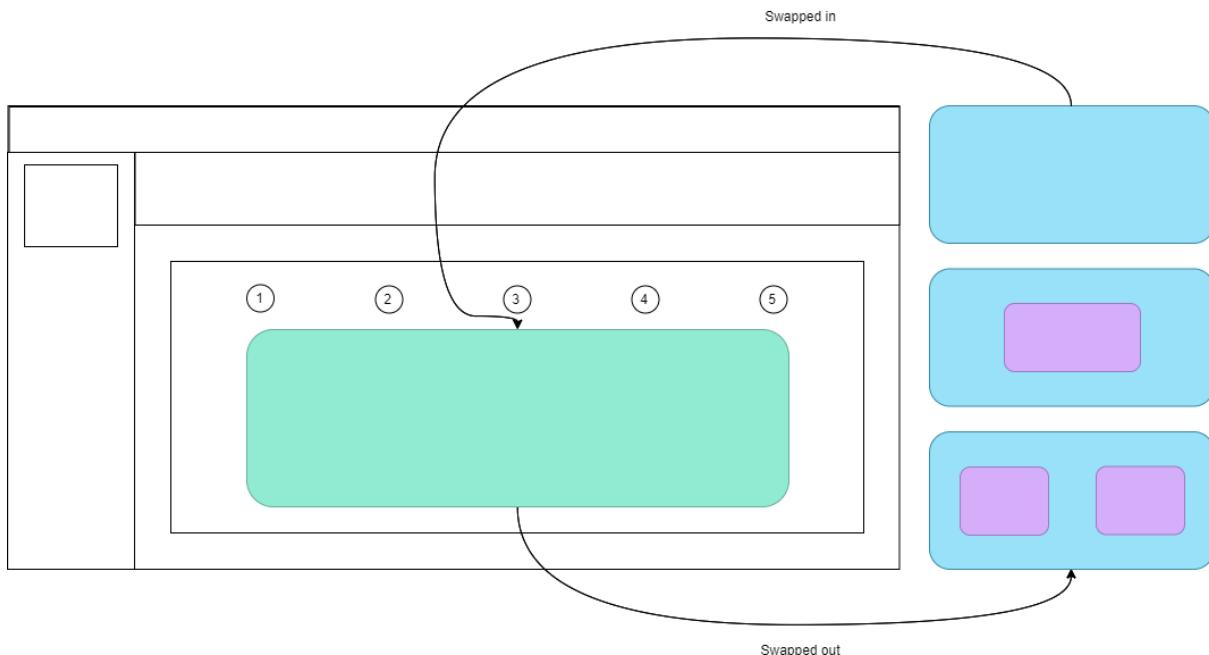


Figure 4.22: Figure showing the module upload part of the application with component hierarchy and switching.

Figure 4.22 visually shows this. The white website is the parent container which holds the parent state as well as some of the "children's" states. The green area is the currently selected and visible component that is rendered, while the blue areas are components managed by the parent, but not in use. In some cases, the child of a parent might have a child component of its own. This is shown by the purple areas within the blue areas. In some cases, even this child of a child component might want to update a state that the parent holds. For this, we have the blue parent hold the state which is needed by the white area parent, and the purple child can update the state of the blue parent, and when the blue parent is switched out of the green area, the blue parent will then store the state needed in a JSON object state in the white parent.

4.7.3 React Hooks

We mentioned React hooks a bit in section 3.7.3. We did have prior knowledge with the use of React hooks for state management and accessing React lifecycle. Using hooks, we were able to develop components a lot quicker than usual. We could have a more complex hierarchy of parent and children components passing state and update methods as props, without any complex logic. We were also able to use hooks with React Redux, which made fetching application state and dispatching actions to update state a lot easier[16]. The examples shown in the figures below are code written only to display the difference between using hooks or not with React Redux.

```

import React from "react";
import { connect } from "react-redux";
import { doSomething } from "../some/actions/folder/file";

function Component({ isAuthenticated, doSomething }) {
    const handleSomething = () => {
        if (isAuthenticated) {
            doSomething();
        }
    };
    ...
}

// Mapping state to properties to use within the component
const mapStateToProps = (state) => {
    return {
        isAuthenticated: state.auth.token !== null,
    };
};

// Mapping dispatch actions to props to use within the component
const mapDispatchToProps = (dispatch) => {
    return {
        onAuth: () => dispatch(doSomething()),
    };
};

export default connect(mapStateToProps, mapDispatchToProps)(Component);

```

Figure 4.23: Figure showing application state management with regular React Redux.

In figure 4.23, we can see an implementation of retrieving a value from the Redux store and dispatching actions using the old way.

```

import React from "react";
import { useSelector, useDispatch } from "react-redux";
import { doSomething } from "../some/actions/folder/file";

function Component() {

    // Check if the token in store !== null
    const isAuthenticated = useSelector((state) => state.auth.token !== null);

    // Get dispatch reference from the redux store
    const dispatch = useDispatch();

    const handleSomething = () => {
        if (isAuthenticated) {
            dispatch(doSomething());
        }
    };
    ...

}

export default Component;

```

Figure 4.24: Figure showing application state management with hooks.

In figure 4.24, we can see the code is shorter and easier to understand. This shows an implementation for retrieving a value from the Redux store and dispatching actions using hooks.

4.7.4 Custom Alert

In AutoPacker, the user has multiple choices and options available. For the user to understand when an action has fired, something is loading or finished executing; we need some form of feedback. To achieve this, we decided to use Ant Designs alert component globally in a fixed position on the website with global application state management (more in detail in section 4.13). This way, any point in the application can dispatch an action which will then create an alert and display it at the fixed location. This alert can take in different types of properties: error, warning, success, different text, closable or not.

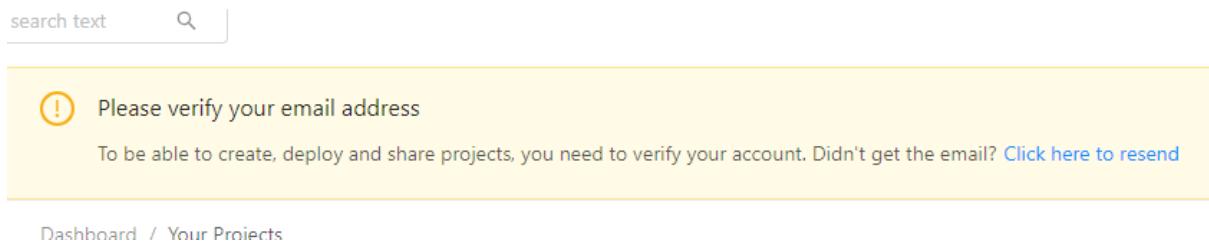


Figure 4.25: Figure showing a use example for the alert component.

Figure 4.25 shows an example use of the alert, which displays a verification notification to the user. An extension to this custom component is the possibility to add a timer to it as well. For now, the alert is closable or not. By adding a timer property later, we can automatically close the alert after the set time. This way, the user does not have to close all the alerts manually. We did choose to prioritize functionality over feedback, so for the time being, some of the alert messages may contain vague replies. This is something that should get fixed before deploying the project unto a production environment

4.7.5 Search Logic

In our application, we want the user to be able to search for whatever is available in the application: projects, users, organizations, available emails and username. Therefore, we heavily rely on an effective way of performing search requests, especially when we have multiple users searching simultaneously.

For this, we were heavily inspired by how GitHub implemented the email and username typing validation delay when registering. Then we found an article that both implemented a custom hook to achieve this as well as an example to use it[50].

```
import { useState, useEffect } from "react";

export default function useDebounce(value, delay) {
  const [debouncedValue, setDebouncedValue] = useState(value);
  useEffect(() => {
    const handler = setTimeout(() => {
      setDebouncedValue(value);
    }, delay);
    return () => {
      clearTimeout(handler);
    };
  }, [value, delay]);
  return debouncedValue;
}
```

Figure 4.26: Figure showing the implementation of the debounce hook.

So what this does is that it takes in two arguments, value and delay. Whenever the value or delay gets modified, we return the value after the timeout set by the delay has expired.

```
const [search, setSearch] = useState("");
const debouncedSearchTerm = useDebounce(search, 500);

useEffect(() => {
  if (debouncedSearchTerm) {
    console.log("this code is fired 500ms after search has been modified");
  } else {
    console.log("This code is fired on component mount");
  }
}, [debouncedSearchTerm]);
```

Figure 4.27: Figure showing an example using the debounce hook.

So in figure 4.27, we pass in the search value to the hook. Whenever the user types, the value gets changed, and the hook gets fired. The hook then returns the value when the timeout expires and the logic inside the if statement gets executed.

4.7.6 Graphical Interface

In this subsection, we will go through our web applications graphical interface, what functionality is available, what functionality might be available in the future and our different thoughts on the solution. Before we started creating the web application, we created some wireframes to get an idea of how the layout and UI should be. After creating the wireframes and implementing some pages, we stopped creating wireframes as we could see how the pages should look like in advance to be consistent with the existing ones. In figure 4.28, we can see the wireframe for the project overview, which resulted in the UI shown in figure 4.46. The rest of the wireframes are available in appendix J.

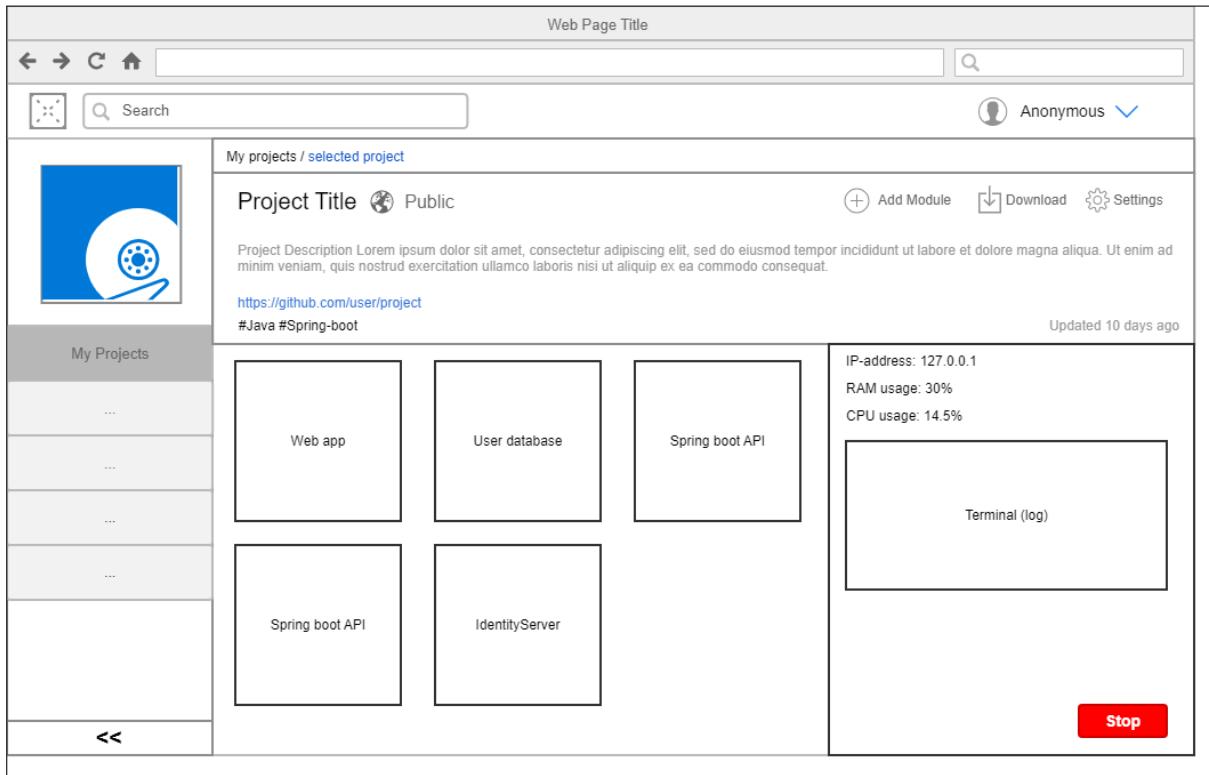


Figure 4.28: Figure showing the wireframe for project overview.

4.7.6.1 Homepage

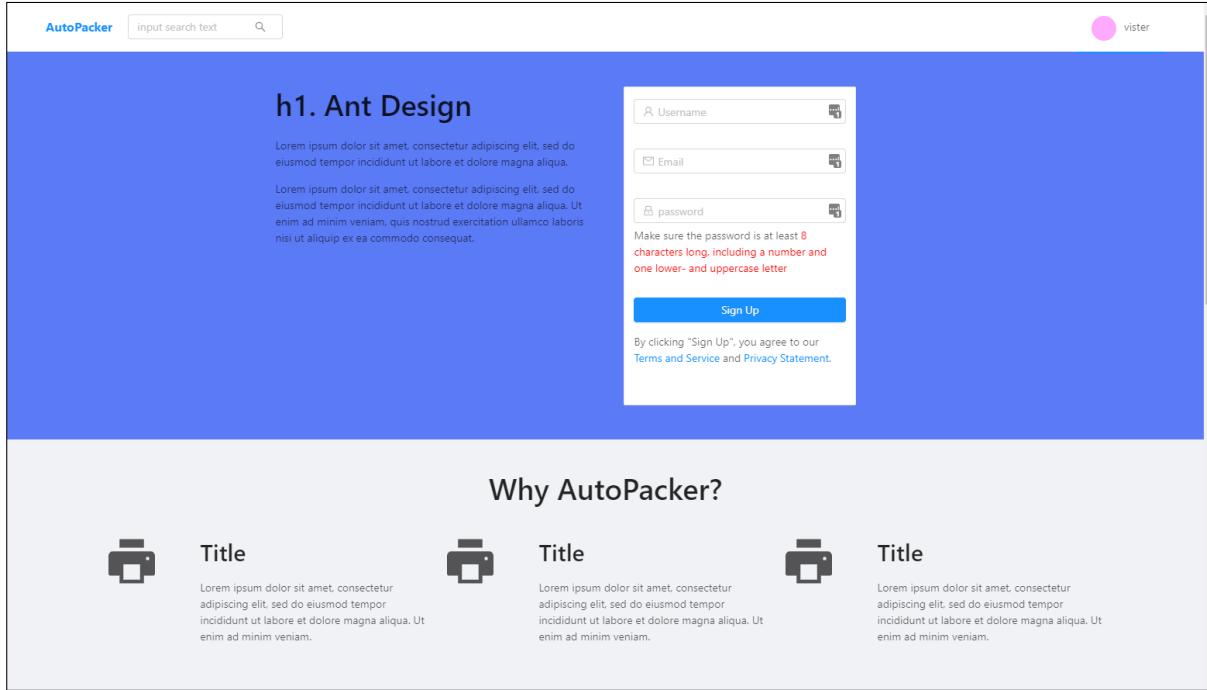


Figure 4.29: Figure showing our temporary homepage.

The first thing the user will meet when entering our website is our homepage. We did not put that much effort into this part of the application as its purpose is only to give an introduction. The most important feature on this page is the registration form.

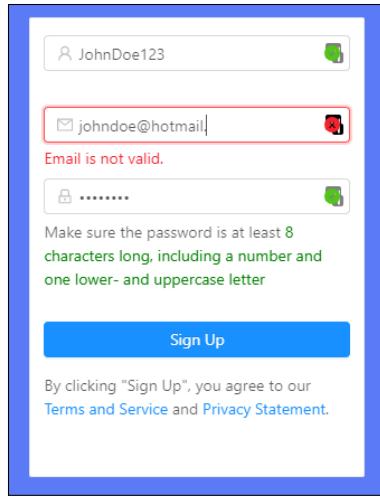
A screenshot of a registration form. At the top, there is a field with a user icon and the placeholder "JohnDoe123". Below it is an email input field containing "johndoe@hotmail.com", which has a red border and a small error icon. An error message "Email is not valid." is displayed below the field. Next is a password input field with a lock icon and the placeholder "*****". A validation message below it states: "Make sure the password is at least 8 characters long, including a number and one lower- and uppercase letter". At the bottom of the form is a blue "Sign Up" button. Below the button, a note says: "By clicking "Sign Up", you agree to our [Terms and Service](#) and [Privacy Statement](#)".

Figure 4.30: Figure showing the registration form.

In this form, the username and email are validated and checked for availability using the search logic explained in section 4.7.5. The user also has to meet a minimum password strength required to be able to register and use our service.

4.7.6.2 Registration Success

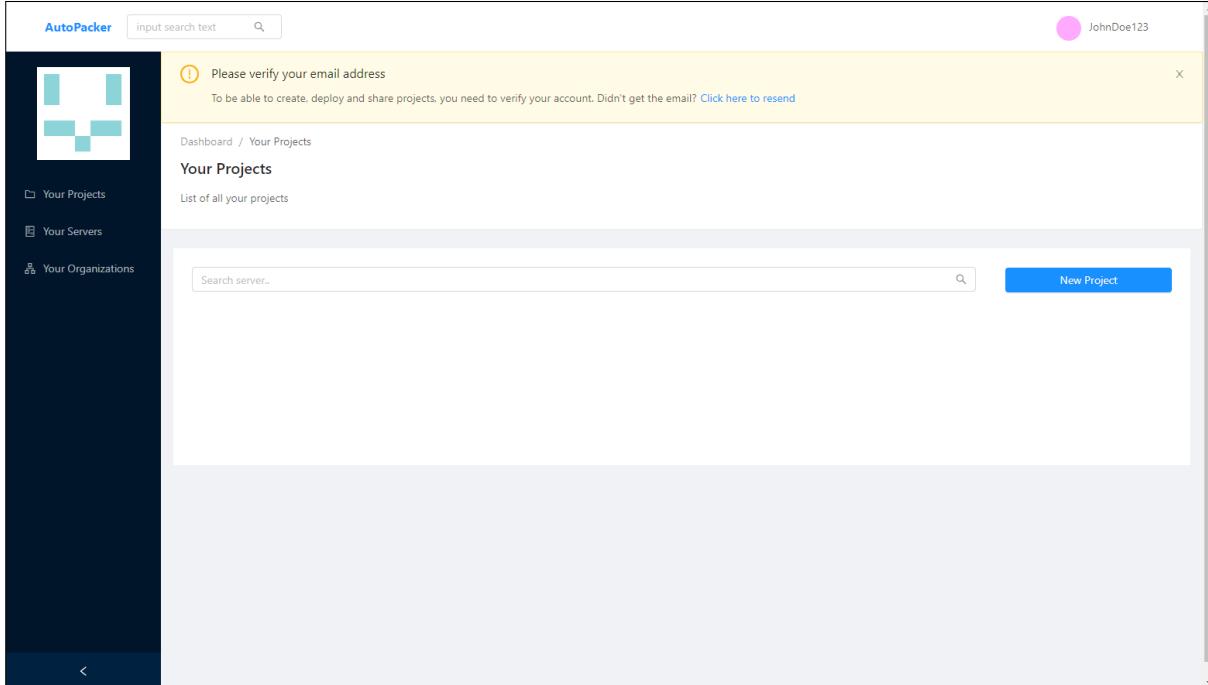


Figure 4.31: Figure showing the profile panel after the user is successfully registered.

If the registration is successful, then the user will be redirected to the page displayed in figure 4.31. The user will also get a notification to check his/her email and verify the account. The user must verify the account before being able to use any of the features the application provides. Without verifying the account, the only difference between a guest and the user would be that the unverified user can preview the profile dashboard environment.

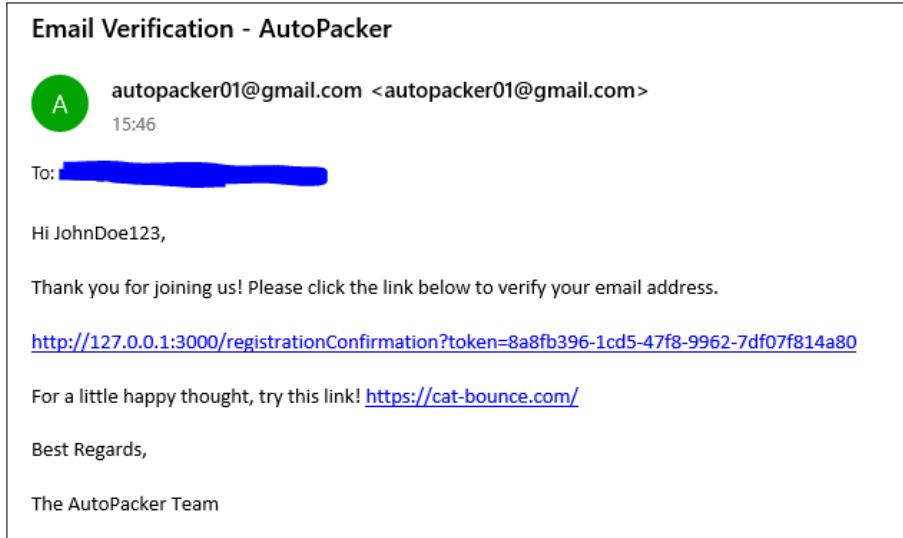


Figure 4.32: Figure showing the verification email.

4.7.6.3 Projects

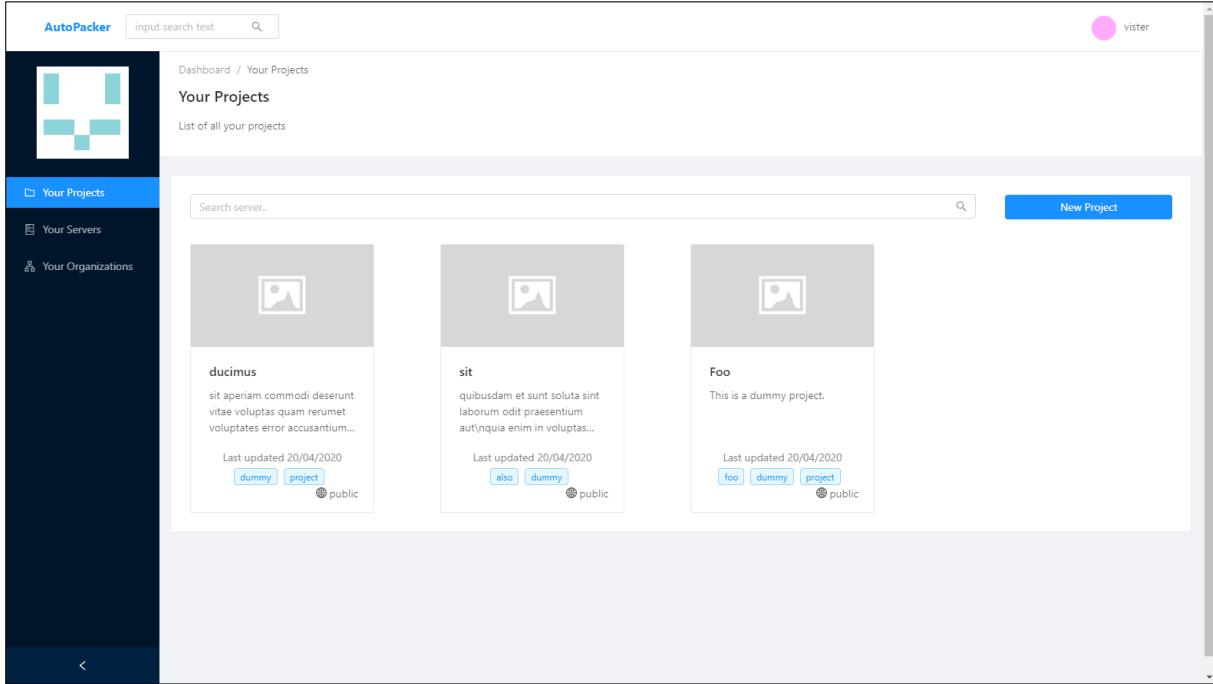


Figure 4.33: Figure showing profile dashboard when authenticated and verified.

In figure 4.33, we have our current "dashboard" which the user is presented with when authenticated. In this example, we have three projects that have been created by the user. The user can use the search input field to find specific projects. From this page, the user can create new projects.

4.7.6.4 New Project

The screenshot shows the 'Create New Project' page of the AutoPacker application. The left sidebar has a dark theme with icons for 'Your Projects' (selected), 'Your Servers', and 'Your Organizations'. The main area has a light background. At the top, there's a search bar with placeholder 'input search text' and a magnifying glass icon. A user profile icon for 'vister' is in the top right. The breadcrumb navigation shows 'Dashboard / Your Projects / Create new project'. The title 'Create New Project' is centered above a form. The form instructions say 'Fill in the form below to create a new project'. It contains fields for 'Project Name' (set to 'Dummy Project'), 'Project Description' (placeholder 'project description'), 'Website' (empty), 'Tags' (empty), and a 'Private project' toggle (set to 'No'). Below the form are three buttons: 'create X', 'a X', and 'project X'. A large blue 'Create Project' button is at the bottom.

Figure 4.34: Figure showing page for creating the projects.

Here the user can input details about the project he wants to create. A project must have a name and can have a description, website (for example GitHub repo or the server hosting it) and tags. For now, we currently only support the creation of public projects since we are using the docker hub registry for storing docker images. If the project gets created successfully, the user gets redirected to the page listing all the users' projects with an alert message displaying a success message (more on the alert in section 4.7.4). If the user is unable to create the project, he will get an error message.

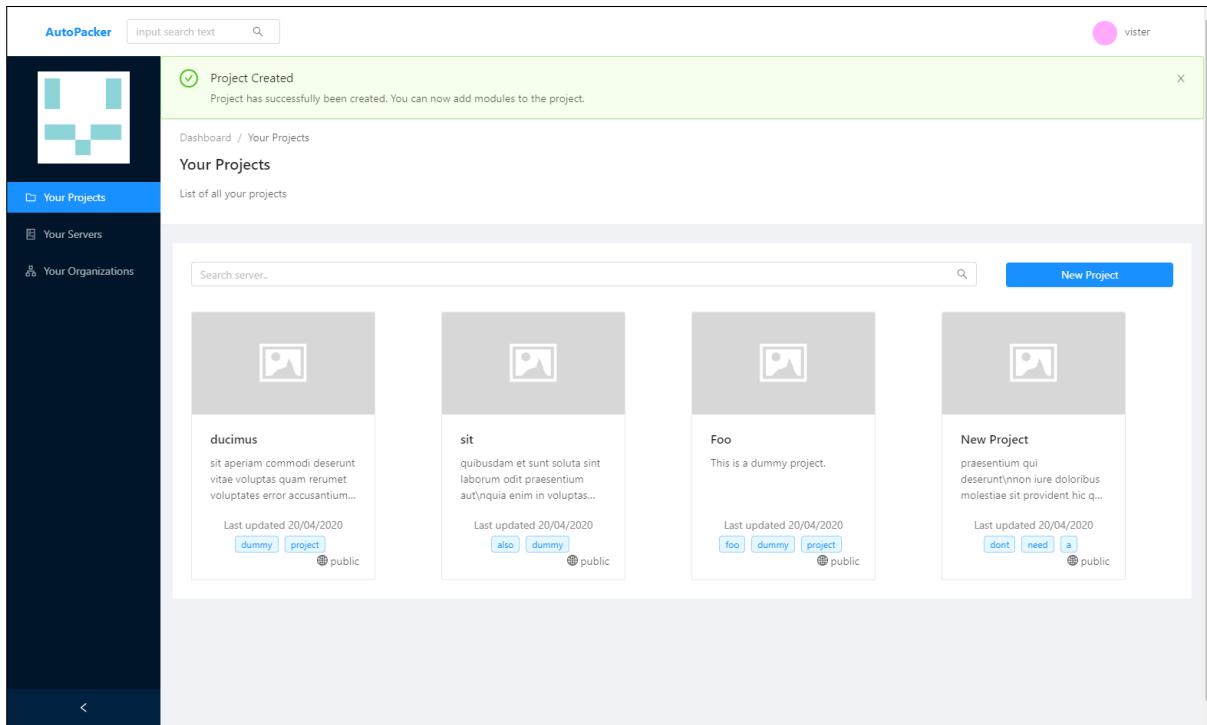


Figure 4.35: Figure showing the profile panel after project creation.

4.7.6.5 Project Overview

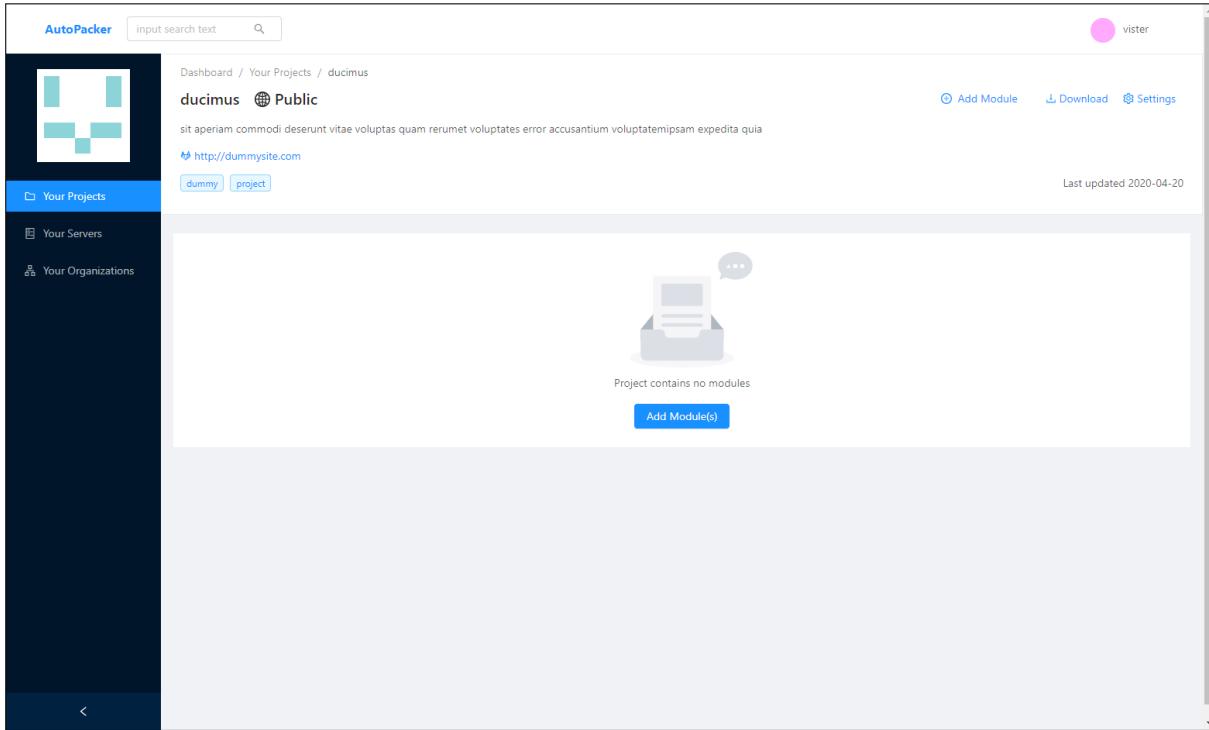


Figure 4.36: Figure showing a project overview page.

Figure 4.36 shows the project overview displayed when a user clicks on one of his projects. In this page, the user can view more information, manage his project settings and add modules to the project to be deployed to a server which we will see later.

4.7.6.6 Module Selection

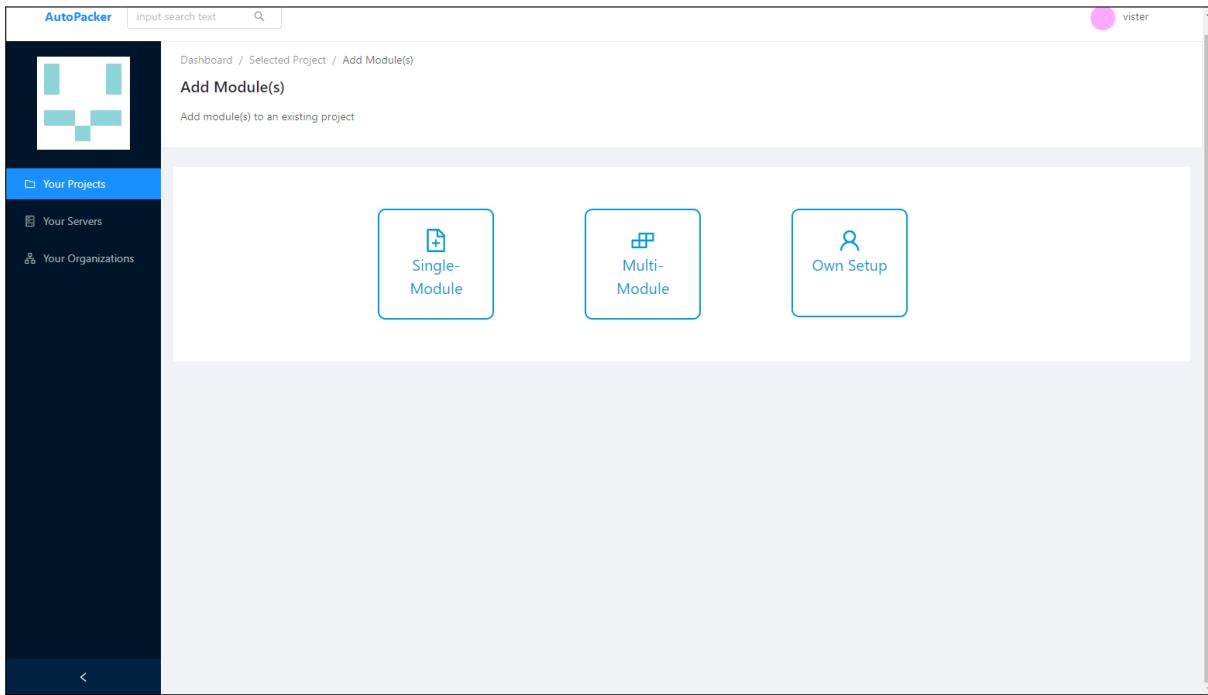


Figure 4.37: Figure showing the module type selection page.

If the user clicks the "Add Module(s)" button displayed in figure 4.36, he will get three options. These different options have different "wizards" to help the user create a single module, multiple modules or even upload a pre-configured setup script. At the moment of writing, we have currently only implemented support for single module upload and only the visual parts for the "multi-module" and "own setup" (as we will see in the sections below). An alternative is to use the single-module upload option multiple times to have multiple modules in the project. This alternative works for application and service-specific programs, but not for databases. We will go through the step by step process for uploading a module in "single-module" first, followed by "multi-module" and lastly "own-setup".

4.7.6.7 Single-Module

4.7.6.7.1 Setup

The screenshot shows the AutoPacker web interface. In the top left, there's a logo consisting of four teal squares arranged in a 2x2 grid. To its right is a search bar with placeholder text 'input search text' and a magnifying glass icon. On the far right, there's a user profile icon with the name 'vister'.

The main title 'Dashboard / Selected Project / Add Module(s)' is at the top. Below it, the sub-section title 'Add Module(s)' is displayed, followed by the instruction 'Add module(s) to an existing project'. A vertical sidebar on the left contains three items: 'Your Projects' (selected), 'Your Servers', and 'Your Organizations'.

The main content area is titled 'Setup' (step 1 of 4). It includes a progress bar with four steps: 'Setup' (highlighted), 'Upload', 'Building', and 'Complete'. The 'General Information' section contains fields for 'Module name:' (set to 'module') and 'Module Description:' (set to 'module description'). The 'Technology' section contains a 'Framework?' field with two radio button options: 'Yes' (unchecked) and 'no' (checked). The 'Specification' section contains a 'Language:' dropdown set to 'Java' and a 'Version:' dropdown set to '8'. Finally, there's a 'Port:' field containing '1300'. At the bottom is a blue 'Forward >' button.

Figure 4.38: Figure showing the setup part of a single-module upload.

In the figure above we can see the setup part for a single-module upload. This part gets used to define the needed information to know what tools and versions are necessary so our services can create a functioning build for that module. Currently, we only have support for simple uploads that only need language, version, and the port.

4.7.6.7.2 Upload

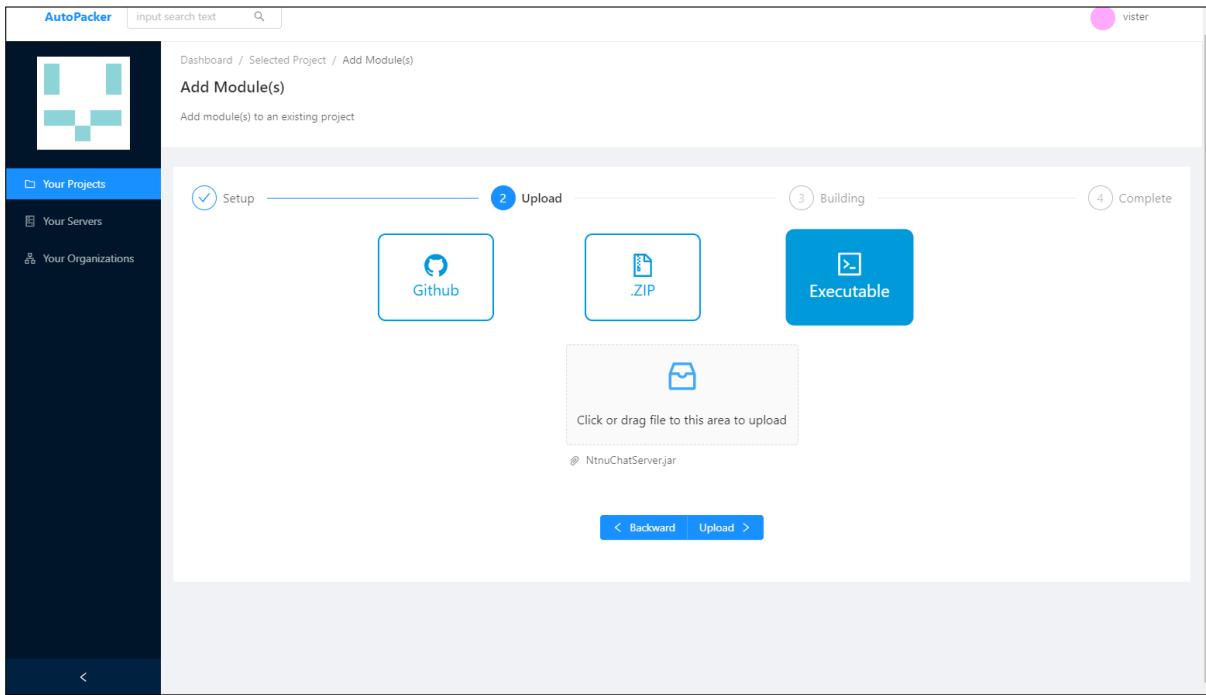


Figure 4.39: Figure showing the upload part of a single-module upload.

In the figure above we can see the upload part for a single-module upload. In this step, the idea was that the user could choose how to upload the module. We currently only have support to upload an executable and .zip, so GitHub does not work at the moment. Here the user has to upload an executable or a zip to be able to proceed.

4.7.6.7.3 Building

When the user clicks the upload button, he will go to the building step. The only thing showing in this step is a loading screen. Here the data (both specification and program/code) is sent to the file delivery API for handling. When this process finishes the user will receive either *HTTP 200 OK* or *HTTP 400 BAD REQUEST*. This status code gets sent to the next step, which is "complete".

4.7.6.7.4 Complete

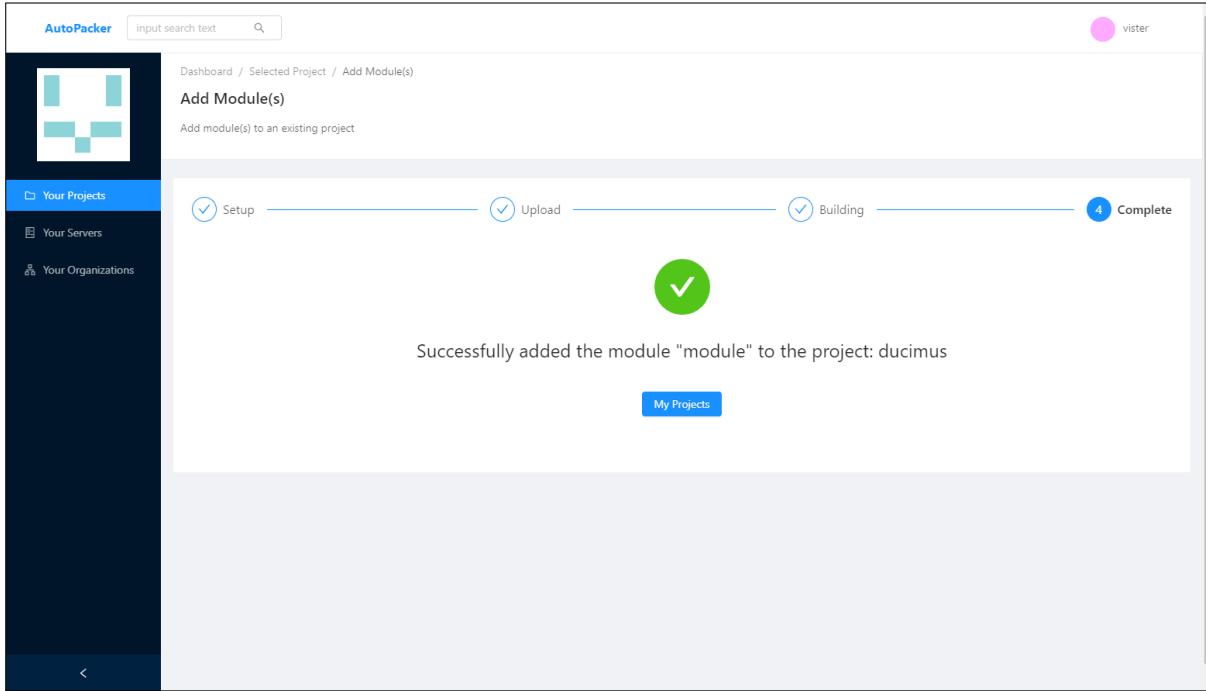


Figure 4.40: Figure showing the complete part of a single-module upload.

This step will extract the status code sent in the previous step and will render some information accordingly. In this example, we see that the module upload was a success and therefore, will get a success message.

4.7.6.8 Multi-Module

Multi-module uploading is currently only in the concept stage. We started out thinking that this feature would be the only possible way of having a multi-module project up and running on a server. Instead, we created such a scalable way of adding modules in the backend services that one can also add multiple modules using the single-module upload multiple times. So the meaning of this feature has now become to make it faster to implement multiple modules, as well as more clear what a person is adding by having the summary section. We might merge the single- and multi-module upload wizards into one after the deadline.

4.7.6.8.1 Summary

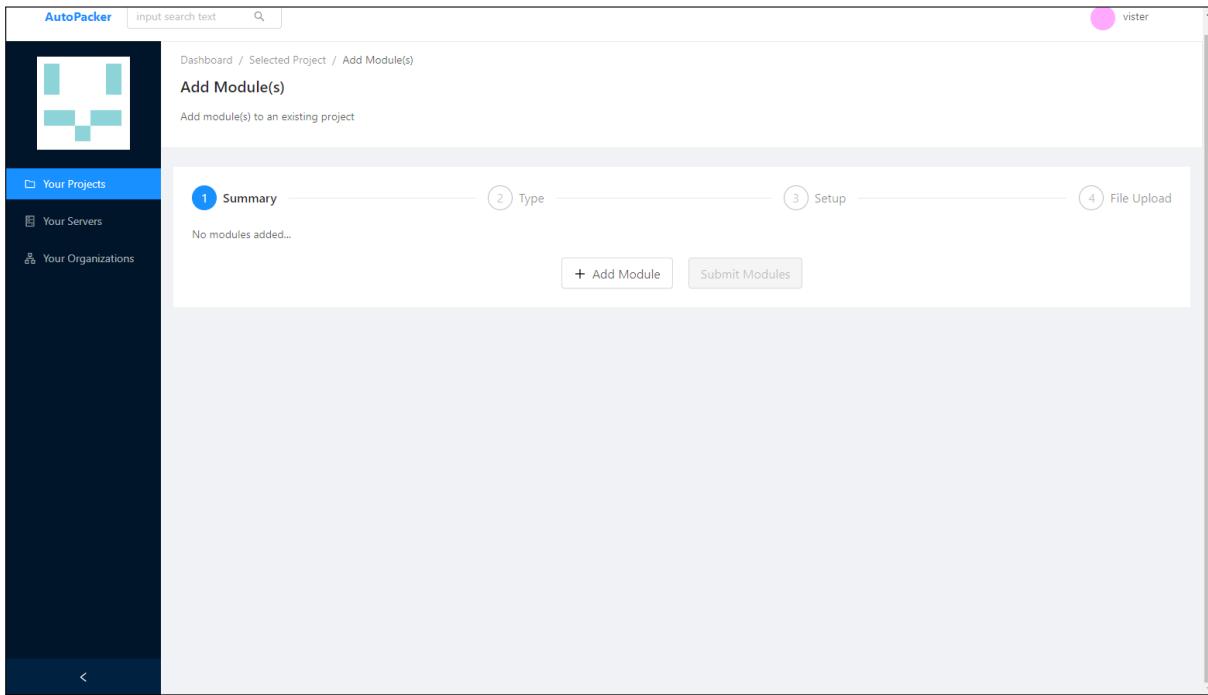


Figure 4.41: Figure showing the summary part of a multi-module upload.

Figure 4.41 shows the first page the user will see when trying to perform a multi-module upload. This step is there to give the user a summary of what he is uploading. This is useful when the user is uploading multiple modules and might be concerned with what data, port and settings have gotten set.

4.7.6.8.2 Type

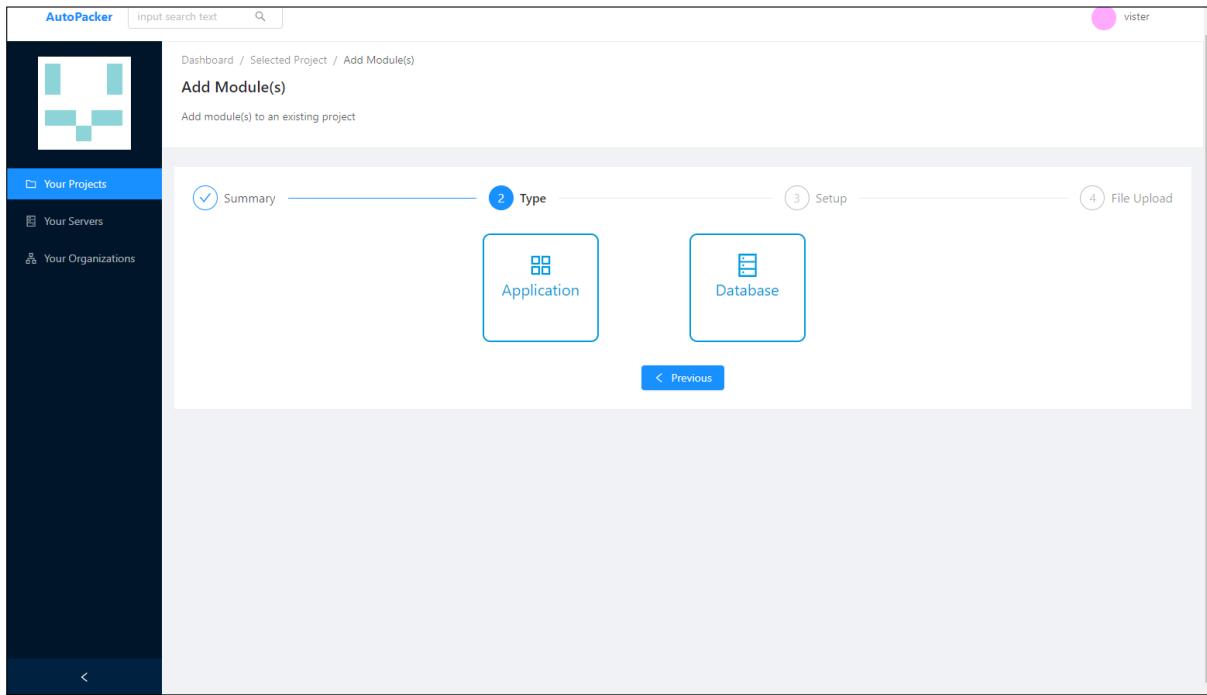


Figure 4.42: Figure showing the type selection part of a multi-module upload.

After clicking "Add Module" in the previous step, the user can now choose what type of module he wants to upload. The application module will show the same steps that the single-module upload does in figure 4.38 and figure 4.39. The database selection currently only has the interface developed and not backend support (yet).

4.7.6.8.3 Setup (database)

The screenshot shows the AutoPacker application interface. On the left, there's a sidebar with 'Your Projects' (selected), 'Your Servers', and 'Your Organizations'. The main area has a header 'Dashboard / Selected Project / Add Module(s)'. Below the header, it says 'Add module(s) to an existing project'. A progress bar at the top indicates four steps: 'Summary' (done), 'Type' (done), 'Setup' (active), and 'File Upload' (not yet started). The 'Setup' tab contains several sections: 'General Information' with 'Module name: MyDatabase' and 'Module Description: database description, if needed'; 'Technology' with 'Database: MySQL' and 'Version: 8'; and 'Specification' with 'Port: 3306'. At the bottom of the setup form are 'Backward' and 'Forward' buttons.

Figure 4.43: Figure showing the setup step for a database module in multi-module upload.

In figure 4.43 we can see the setup form for a database application. It looks quite similar to the setup form shown in figure 4.38. We have some common parts and some unique parts, and by using react components, we can split things up in parts and reuse parts as much as needed. This way, we have the upper parts containing the name and description as a general part which gets always included. So the unique part here is the database name, username and password. The user also needs to specify the database, version and port to run on.

4.7.6.8.4 File Upload (database)

The file upload part for a database module is also the same as the one shown in figure 4.39. The only difference is that the user is not required to upload a file. In some cases, one might have a service which generates the tables and data needed on runtime. Then one does not need a .sql

file to initialize it. So in the file upload step for a database module, there will be a help text for the user explaining this in detail.

4.7.6.8.5 Summary (with modules)

The screenshot shows the AutoPacker interface with the following details:

- Left Sidebar:** Contains 'Your Projects' (selected), 'Your Servers', and 'Your Organizations'.
- Header:** Shows 'AutoPacker' logo, search bar, and user 'vister'.
- Current Path:** Dashboard / Selected Project / Add Module(s)
- Section:** Add Module(s) - Add module(s) to an existing project
- Step Progress:** 1 Summary (highlighted), 2 Type, 3 Setup, 4 File Upload
- Module 1 (MyDatabase):**
 - Module Name: MyDatabase
 - DB Name: db
 - Port: 3306
 - Module Description: database description, if needed
 - DB Username: root
 - Type: database
 - Database & version: MySQL 8
- Module 2 (TestModule):**
 - Module Name: TestModule
 - Programming language & version: Java 8
 - Module Description: module desc
 - Port: 3000
 - Type: application
 - File: NtnuChatServer.jar
- Buttons:** + Add Module, Submit Modules

Figure 4.44: Figure showing the summary part with modules data in multi-module uploading.

When the user finishes specifying the necessary details for a module, he will return to the summary step at the start, which will show a list of all added modules. In figure 4.44, the user has added two modules, one database and one application. So the idea here is that when the user has added two or more modules, the "submit modules" button will become enabled and the user can upload the modules and add them to the project.

4.7.6.9 Own Setup

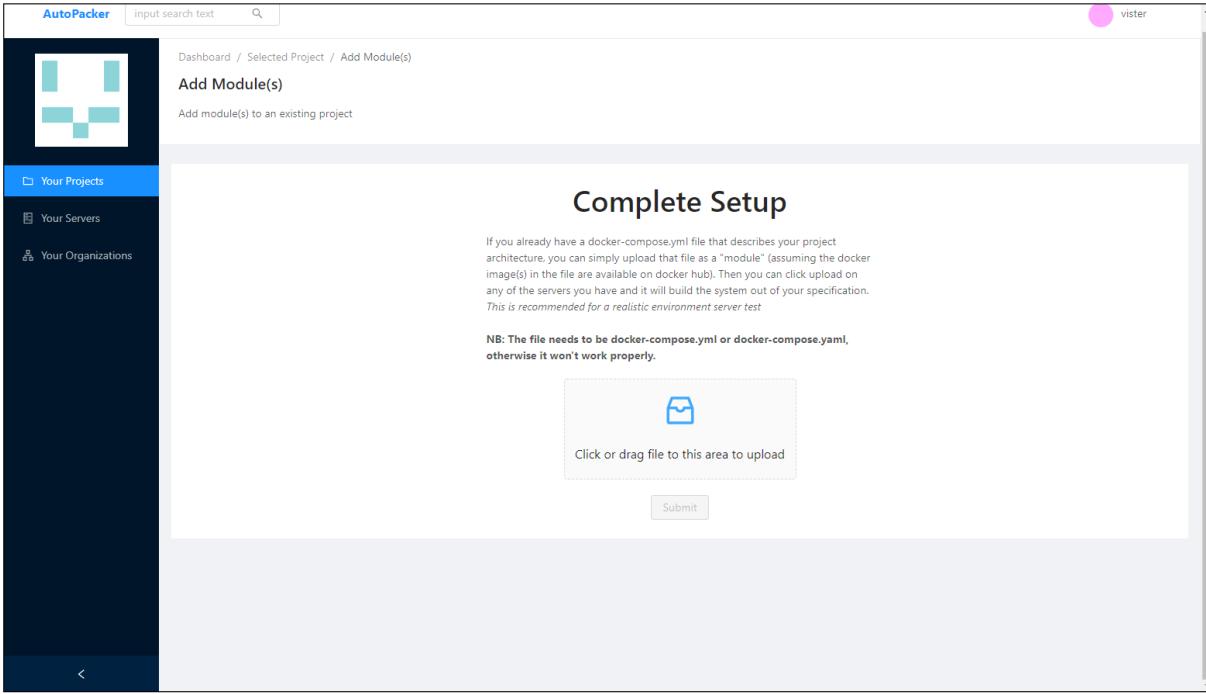


Figure 4.45: Figure showing the own setup option.

The third and last module upload option is the "own setup" option shown in figure 4.45. In some cases, the user might have a full docker-compose setup already configured and only want to deploy a project to a server quickly. If so, then this option is the best. It can also be a better option if the configuration is quite complex, and we do not offer support for that complexity in the setup wizards.

4.7.6.10 Project Overview (populated)

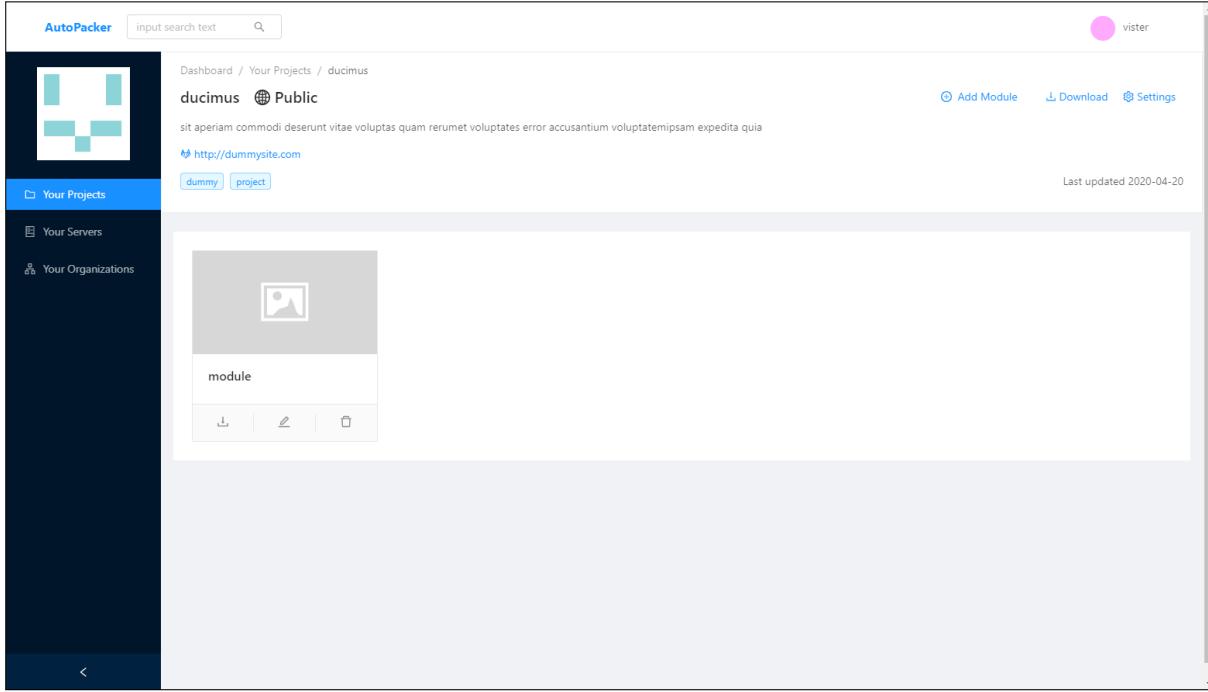


Figure 4.46: Figure showing a project overview with added module.

In figure 4.46, we can see that the module we entered in section 4.7.6.7 has been added as a module to the project. There are two specific actions we can perform on that module from this page. We can select it to get more details about that specific module (figure 4.47), or click the trash icon to delete the module from the project (figure 4.48).

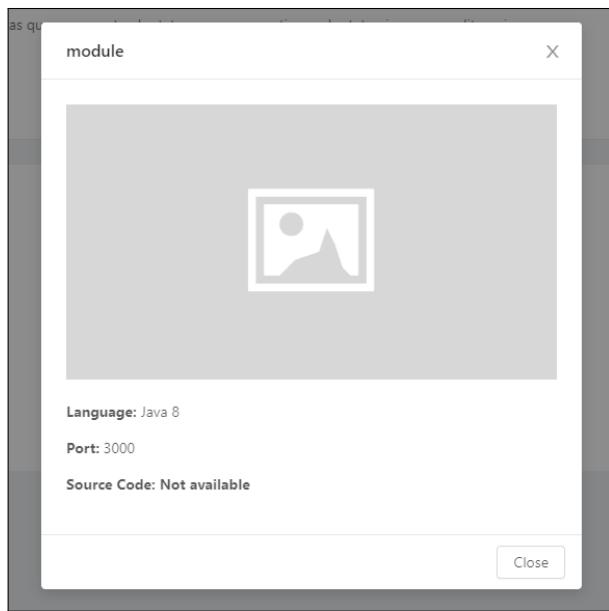


Figure 4.47: Figure showing a modal containing module details.

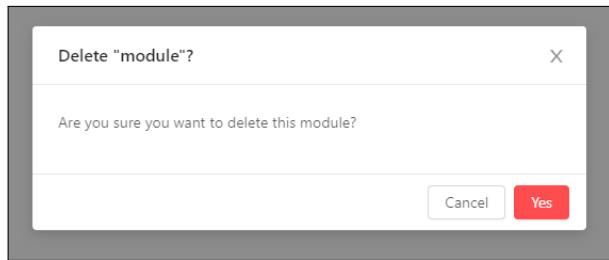


Figure 4.48: Figure showing a modal for deleting a module.

4.7.6.11 Project Settings

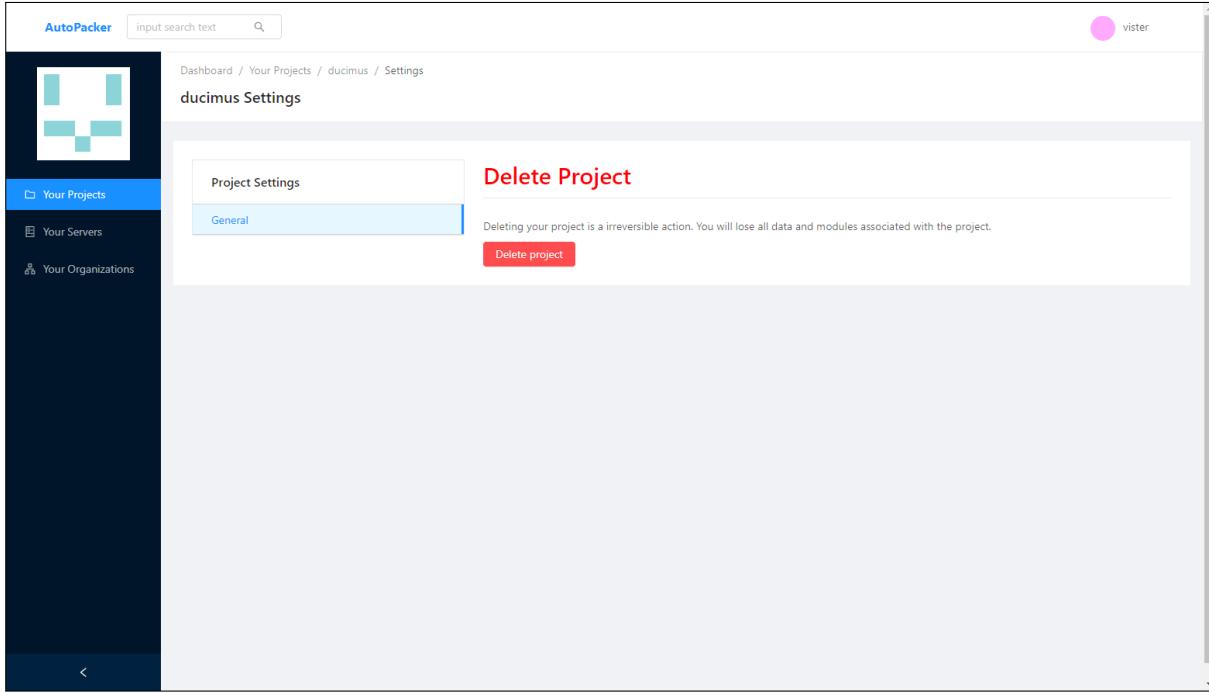
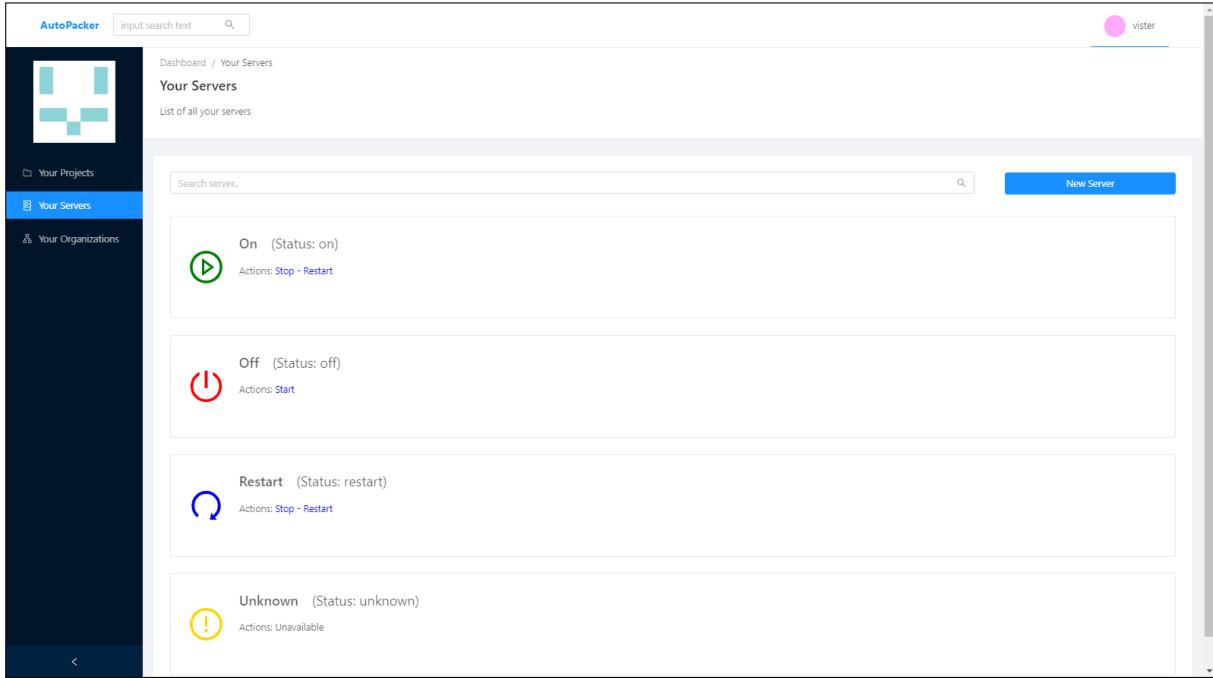


Figure 4.49: Figure showing the project settings page.

If the user clicks the settings option in figure 4.36 he will see the settings page shown in figure 4.49. The only option available is to delete the project. If the user clicks on the "delete project" button a modal to delete the project will come up. The user then has to write "delete" in the input field to approve the deletion.

4.7.6.12 Servers

A screenshot of the AutoPacker application interface. The top navigation bar includes 'AutoPacker', a search bar, and a user profile icon labeled 'vister'. The left sidebar has three items: 'Your Projects' (disabled), 'Your Servers' (selected and highlighted in blue), and 'Your Organizations'. The main content area is titled 'Your Servers' with the subtitle 'List of all your servers'. It features a search bar at the top right and a 'New Server' button. Below, there are four server entries: 1) 'On (Status: on)' with a green play icon, showing actions 'Stop - Restart'. 2) 'Off (Status: off)' with a red power-off icon, showing action 'Start'. 3) 'Restart (Status: restart)' with a blue circular arrow icon, showing actions 'Stop - Restart'. 4) 'Unknown (Status: unknown)' with a yellow exclamation mark icon, showing action 'Unavailable'.

Status	Action	Icon	Status Description	Action Description
On	Stop - Restart	Green play icon	(Status: on)	
Off	Start	Red power-off icon	(Status: off)	
Restart	Stop - Restart	Blue circular arrow icon	(Status: restart)	
Unknown	Unavailable	Yellow exclamation mark icon	(Status: unknown)	

Figure 4.50: Figure showing all the users' servers.

In our application, the user can have several servers added. These servers get used for deploying the users' projects on to them. Currently, our application can add servers, execute an initialization/preparation script on the server on demand, and deploy any projects that our application supports. In figure 4.50 we have different icons, colours and actions for the different states a server can have. This is just a concept for some functionality we can add later. For now, every server created will have a stock icon, and no actions are available for managing server state.

4.7.6.13 Server Creation

The screenshot shows the AutoPacker web application interface. On the left, there is a sidebar with three items: 'Your Projects' (with a gear icon), 'Your Servers' (with a server icon, which is highlighted in blue), and 'Your Organizations' (with a user group icon). At the top, there is a search bar labeled 'input search text' with a magnifying glass icon. In the top right corner, there is a user profile icon with the name 'vister'. The main content area is titled 'Add Server' and contains the following fields:

- * Server Title: Dummy Server
- Server Description: servers can have a description as well
- * IP Address: 255.255.255.255
- * Username: dummy
- * Password:
- SSH?: Yes no

At the bottom of the form is a blue 'Add Server' button.

Figure 4.51: Figure showing the form for adding a server.

Like creating a project, a user can easily add a server. In figure 4.51, we can see the form for adding a server. The username and password that the user enters are needed for our application to connect to the server. When the user has finished entering all the details and clicks "add server", the user will get a success alert displaying a success if the creation succeeds. If not, the user will get an error message (similar to the one displayed in figure 4.35).

4.7.6.14 Server Overview

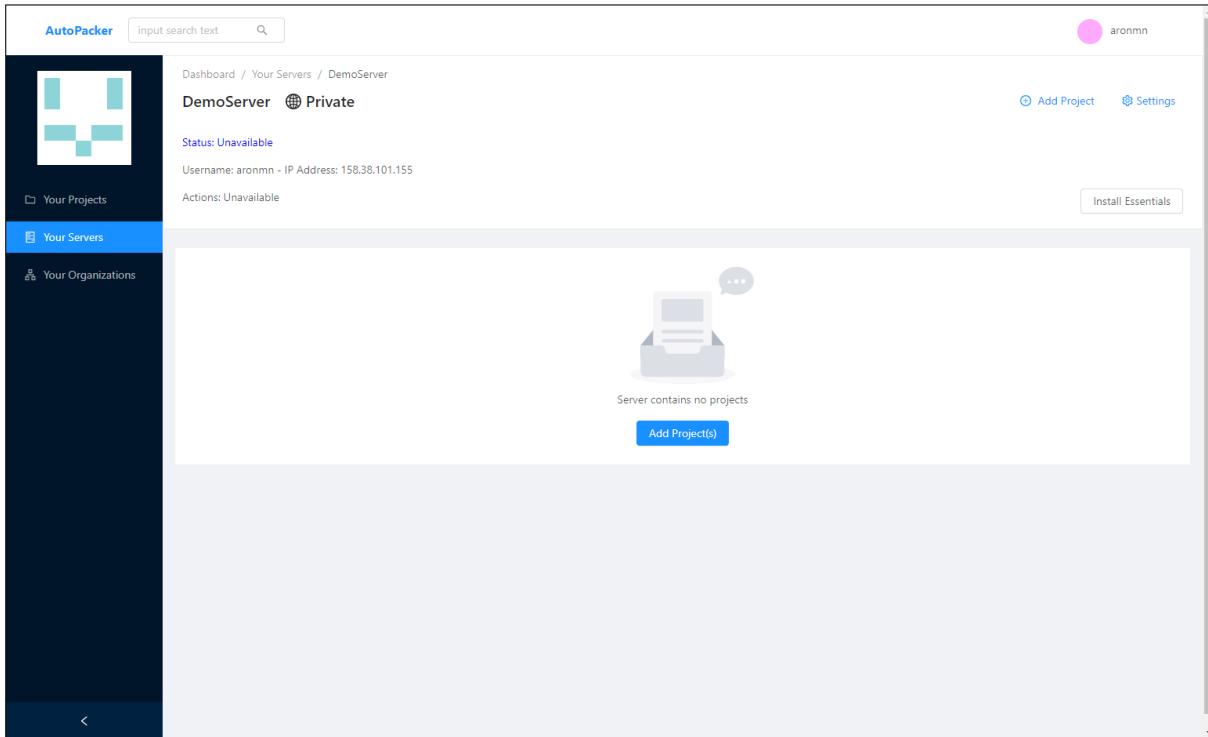


Figure 4.52: Figure showing an overview of a server without projects assigned to it.

From here, the user can get more details about the server they have. Furthermore, it is from here they can assign which projects they want to deploy to that server. By clicking the "add project" button, a modal will open displaying all the projects the user owns (figure 4.53), and from here, the user can select the projects he wants to assign to the server. By clicking the "install essentials" button the application will perform a request to the server manager API which will try and connect to the server and execute an installation script (more details on the script in section 4.5.4).

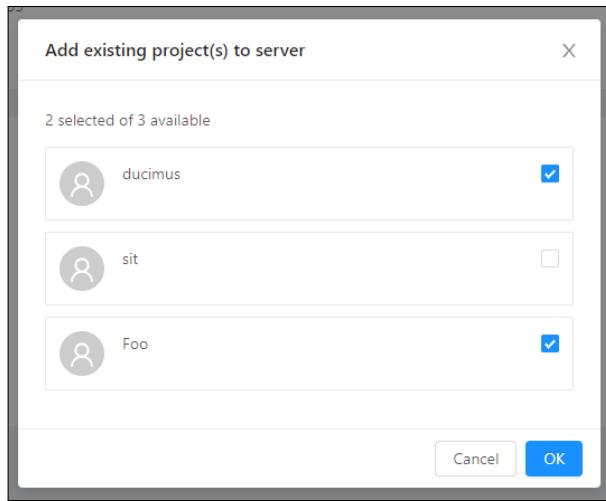


Figure 4.53: Figure showing the modal for adding projects.

When the user has successfully added projects to the server, the server overview will be populated by those projects, as shown in figure 4.54.

The screenshot shows the AutoPacker interface. On the left is a sidebar with navigation links: 'Your Projects', 'Your Servers' (which is currently selected), and 'Your Organizations'. The main content area displays the 'DummyServer' overview. It includes the server's name ('DummyServer'), its status ('Private'), a status message ('This is just a dummy server used to display the server overview page.'), and its details ('Status: Unavailable', 'Username: dummy - IP Address: 255.255.255.255'). Below this, two project cards are listed:

- ducimus**: 'sit aperiam commodi deserunt...'. Edit | Delete | ...
- Foo**: 'This is a dummy project.'. Edit | Delete | ...

Figure 4.54: Figure showing an overview of a server with projects assigned to it.

One can delete a server by clicking the settings button in the top right corner, going to the general option and hitting the delete button. This will open a delete modal (same as for project deletion in figure 4.48), and when the user enters delete and clicks "yes" the server will be

deleted. If the user has deployed a project unto a server, this program will still run even if the server gets "deleted" from the application. If the user wants to delete a specific project from the server, he can click the trashcan icon on the project followed by "yes". This will remove the project and delete the docker-compose file that was created on the server. For the time being, we have no way to stop or remove the running docker container on the server after deployment, even if the user removes the project from the server.

4.7.6.15 Organizations

For the sections regarding organization-specific features, everything is viewed from an organization admin perspective. This will make every organization option visible in the sidebar.

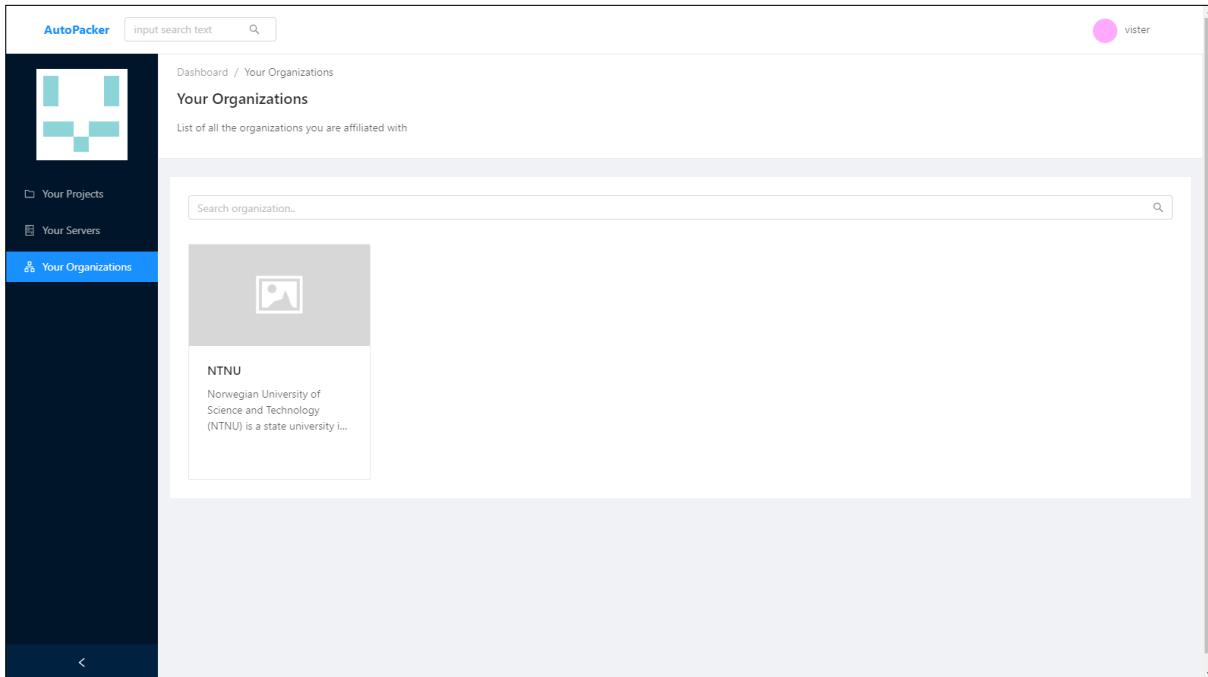


Figure 4.55: Figure showing a list of organizations the user is affiliated with.

In our application, we also have a subsystem that allows users to be part of organizations. For the time being, there is no available option to create an organization other than that we (the developers) create it. This is because the organization part is more of a prototype that we can further extend. The things we have implemented though does work as intended. For the time being, we have implemented an organization that can be used by NTNU to hold students, bachelor project references and software used in lectures.

4.7.6.16 Organization Projects

The screenshot shows the 'AutoPacker' application interface. At the top left is the 'AutoPacker' logo. To its right is a search bar with the placeholder 'input search text' and a magnifying glass icon. In the top right corner, there is a pink circular profile picture with the word 'vister' next to it. The main header reads 'Dashboard / Organization Projects' and 'NTNU Projects'. Below the header, a sub-header states: 'Norwegian University of Science and Technology (NTNU) is a state university in Norway, and from 2016 the country's largest. It has headquarters in Trondheim and campuses in Gjøvik and Ålesund.' A search bar labeled 'Search server...' with a magnifying glass icon is positioned above a list of projects. The first project in the list is titled 'Test Project' with a small image placeholder. Below the title, a note says 'This is just for displaying data from db'. Underneath the note, it specifies 'Type: Bachelor'. At the bottom of the project card are two blue rectangular buttons labeled 'test' and 'project'. On the left side of the screen, there is a dark sidebar with the 'NTNU' logo at the top. Below the logo, the sidebar is divided into sections: 'Projects' (with 'Overview' selected), 'Project Requests', 'Members' (with 'Overview' selected), 'Applicants', and 'Personal'. Each section has a small icon and a dropdown arrow indicating more options.

Figure 4.56: Figure showing a list of projects affiliated with the organization.

The first thing the user will see when accessing the organization from the users' profile panel is the organization panel. This panel has another sidebar as well as organization-specific options as seen in figure 4.56. For the time being, we do not have a "dashboard feed" so the natural "dashboard" at the moment is a list of all the projects associated with the organization. To go back to the profile dashboard instead of viewing the organization related features the user can hover over the username in the top right corner of the screen and click on one of the options shown in figure 4.57.

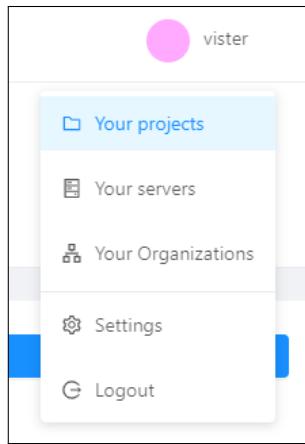


Figure 4.57: Figure showing the dropdown menu from hovering over username.

4.7.6.17 Organization Project Requests

A screenshot of a web application interface. On the left is a dark sidebar with the "AutoPacker" logo at the top, followed by a search bar with placeholder text "input search text" and a magnifying glass icon. Below the search bar are several menu items: "Dashboard / Organization Project Requests", "NTNU Project Requests", "List of all the project requests that wants to be affiliated with the organization". The main content area shows a list of project requests. The first request is titled "Arro1990 - My Project" and has a submission date of "2020-04-20". It includes details such as "Submitted By: Arro1990", "Project Name: My Project", "Type: Personal", "Tags: My project", "Description: Just a simple test project", "Authors: Bethany", "Links: http://localhost, http://myproject.no", and "Comment: Bachelor project". Below this are buttons for "Decline Request", "Edit", "Accept Request", and "Accept & Verify". The second request listed is "Tionve - Test2 Project" with a submission date of "2020-04-20". The sidebar also contains sections for "Projects" (Overview, Project Requests selected), "Members" (Overview, Applicants), and "Personal".

Figure 4.58: Figure showing a list of project requests made to the organization.

To be able to have a project displayed as one of the organizations' projects, one has to submit a project request (more on this in section 4.7.6.20). In figure 4.58 we can see a list of project requests. This page is only available to organization admins. From here, the admin has three choices.

4.7.6.17.1 Decline Request

If the admin were to click the "decline request" button, the module in figure 4.59 would show up. Here the admin can add a comment which will be sent with the email informing the requester that the organization has declined the request.

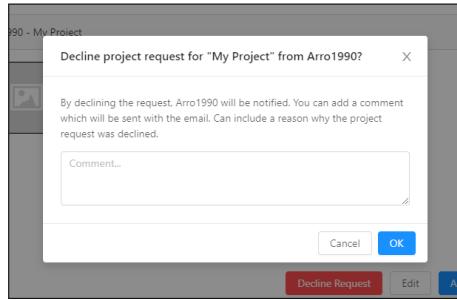


Figure 4.59: Figure showing the modal for declining a request.

4.7.6.17.2 Edit Request

If the admin wishes to edit the request, he can click the "edit" button which will open up the module in figure 4.60. The only thing the admin can edit at the moment is the tags (add and remove). If the admin were to alter the request, he should notify the user in the comment field when accepting the request.

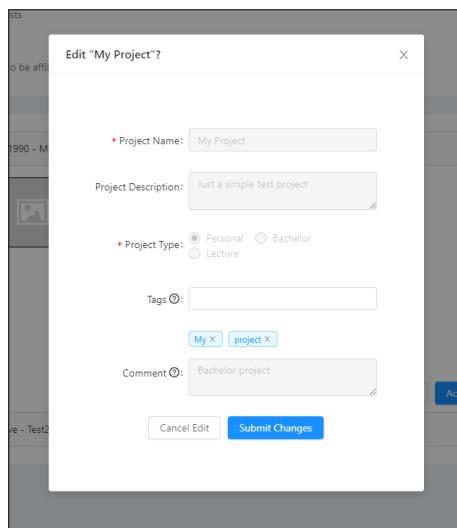


Figure 4.60: Figure showing the modal for editing a request.

4.7.6.17.3 Accept Request

If the admin is ready to accept the request, he can click the "accept request" button and the modal shown in figure 4.61 would show up. Here he can also add a comment which will be sent with the email to inform the user that the organization has accepted the project request.

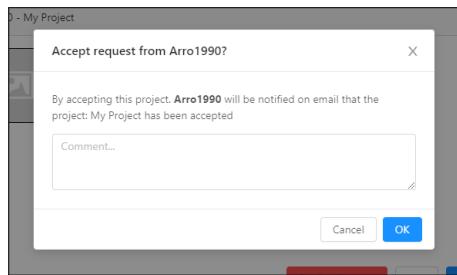


Figure 4.61: Figure showing the modal for accepting a request.

4.7.6.18 Organization Members

The screenshot shows the AutoPacker application interface. The left sidebar has a dark theme with white text and icons. It includes sections for "Projects", "Members" (which is currently selected and highlighted in blue), "Applicants", and "Personal". The main content area has a light gray background. At the top, there's a search bar with placeholder text "input search text" and a magnifying glass icon. Below the search bar, the page title is "Dashboard / Organization Members" and the sub-section title is "NTNU Members". A descriptive text states "Table containing all the members affiliated with the organization in some way." To the right of this text is a user profile icon with the name "aronmn". The main content is a table with two columns: "Username" and "Role". The table lists the following data:

Username	Role
vister	ADMIN
Arro1990	ADMIN
Tionve	MAINTAINER
Boodsom	EMPLOYEE
Jone1994	EMPLOYEE
Funt1959	STUDENT
Sithered	STUDENT
aronmn	ADMIN
libanbn	ADMIN

At the bottom right of the table area, there are navigation buttons for pagination: '<', '1' (highlighted in blue), and '>'.

Figure 4.62: Figure showing a list of all the members of an organization.

As a member of an organization, one can see all the other members and their role, as shown in figure 4.62. This list can be sorted alphabetically (ascending and descending) by username and role.

4.7.6.19 Organization Applicants

The screenshot shows the AutoPacker application interface. On the left, there is a dark sidebar with the NTNU logo at the top. Below the logo, the sidebar has several sections: 'Projects' (with 'Overview' and 'Project Requests'), 'Members' (with 'Overview' and 'Applicants' which is highlighted in blue), and 'Personal'. At the bottom of the sidebar are navigation arrows. The main content area is titled 'Dashboard / Organization Applicants' and 'NTNU Applicants'. It displays a list of applicants with their names, roles, and a comment. The first applicant listed is 'Hatiou1983' with a role of 'STUDENT' and a comment 'Please accept me :D'. The second applicant listed is 'chu3llzahkai'. At the bottom right of the list, there are 'Decline Request' and 'Accept Request' buttons. The top right corner of the main window shows a user profile icon and the word 'vister'.

Figure 4.63: Figure showing a list of all the applicants for the organization.

To become a member of an organization, one has to submit a membership application (how you do so is described in section 4.7.6.25). In figure 4.63, we can see a list of membership applications (this page is only accessible for organization administrators. Here the administrator can either accept or decline the applicant (the modals' that will come up when clicking decline or accept are similar to the ones in figure 4.59 and 4.61). In both cases, the applicant will be informed by an email.

4.7.6.20 Organization - Submit Project

The screenshot shows the 'Submit Project' page on the AutoPacker platform. The left sidebar has a dark theme with categories: Projects (Overview, Project Requests), Members (Overview, Applicants), Personal (Submit Project, Submissions). The 'Submit Project' item is highlighted. The main content area is titled 'Submit Project' and describes it as a 'Form to submit a project to the organization'. It contains the following fields:

- * Project Name:
- Project Description:
- * Project Type: Bachelor Personal Lecture
- Tags: example project
- Author(s):
- Link(s):
- * Project:
- Comment:

A blue 'Submit Project' button is at the bottom right.

Figure 4.64: Figure showing the page used to submit a project request.

As mentioned in section 4.7.6.17, the user has to submit a project request to have his project displayed on the organization project hub. The page containing the form to submit a project is displayed in figure 4.64.

4.7.6.21 Organization Submissions

The screenshot shows a web application interface for 'AutoPacker'. The left sidebar has a dark blue background with white text and icons. It includes links for 'Projects', 'Members', 'Personal', 'Submit Project', and 'Submissions', with 'Submissions' being the active tab. At the top, there is a header with the NTNU logo, a search bar, and a user profile icon labeled 'vister'. A green notification bar at the top indicates a successful update: 'Project request updated' with the message 'You have successfully updated the project information. You will receive a notification on email when the request has been handled.' Below the notification, the main content area shows the title 'Your Project Submissions' and a subtitle 'List of all the project submissions you have with this organization'. A specific submission is highlighted with a modal window titled 'vister - TestSubmission' dated '2020-04-20'. The modal displays 'Project Application Details' with fields: 'Submitted By: vister', 'Project Name: TestSubmission', 'Type: Bachelor', 'Tags: test submit', 'Description: creating a test submission', 'Authors: me', 'Links: http://testlink.no', and 'Comment: test comment'. At the bottom of the modal are two buttons: 'Delete Request' (red) and 'Edit'.

Figure 4.65: Figure showing a list of submissions made by the user.

After the user has made one or more submissions, they will appear when clicking the submissions option in the sidebar. Doing this will render a list containing all of the users' submission, as shown in figure 4.65. From here the user can either delete a submission or edit an existing submission. The delete button will open a delete verification modal where the user has to accept to delete the project. If the user were to click the Edit button the modal shown in figure 4.66 would show up. Here the user can edit everything except what project he wants to submit. Because if the user wanted to change the project as well, he can delete this submission and create a new one.

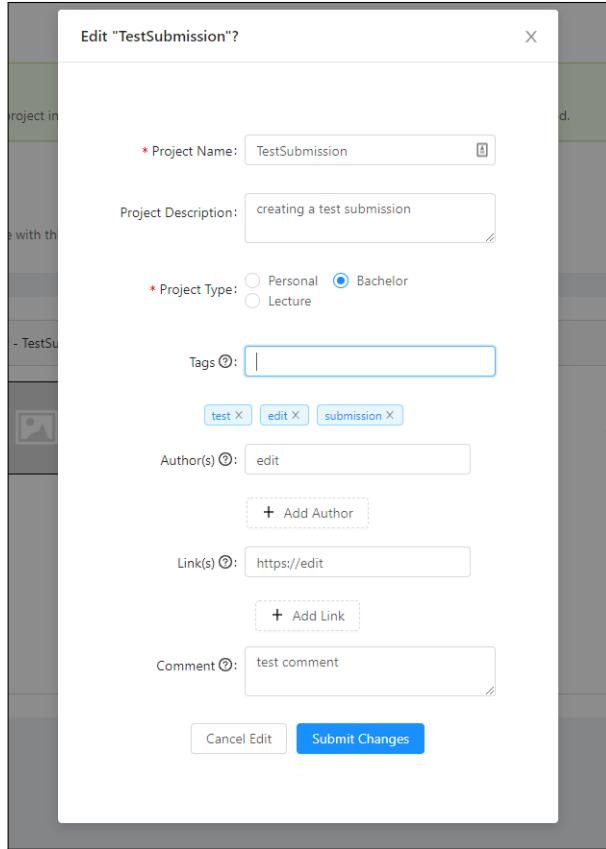


Figure 4.66: Figure showing the modal for editing an existing submission.

4.7.6.22 Search Result

In addition to having all these personal related features, we have also implemented the possibility to see other users' profiles, project as well as finding public organizations. To do this, one has to search for something in the input field in the top navigation bar. The search field can be left empty to search for everything, or one can search for something containing search input.

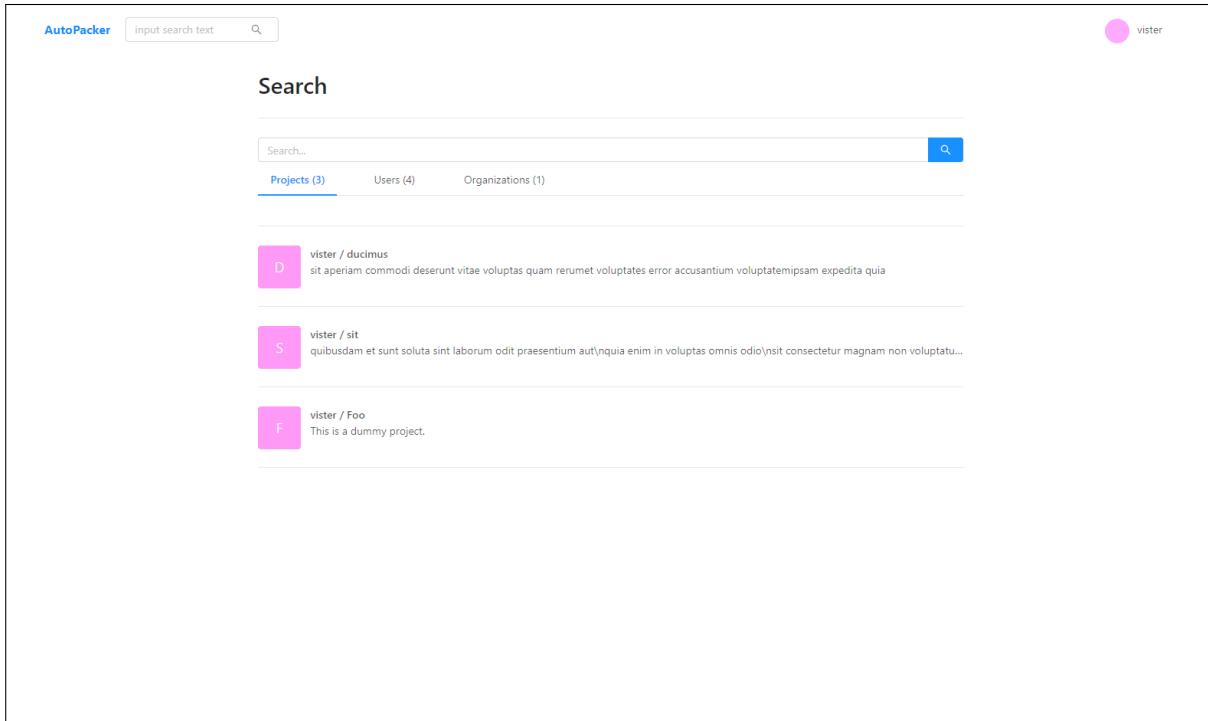


Figure 4.67: Figure showing the search results page.

In figure 4.67, one can see the results we get when entering an empty search. This will list all the available public projects, users and organizations. By changing the selected tab, one can see the different results.

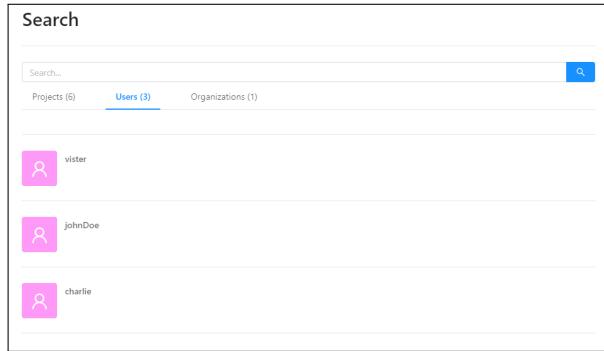


Figure 4.68: Figure showing the search results containing users.

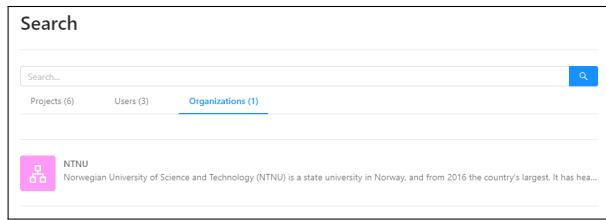


Figure 4.69: Figure showing the search results containing organizations.

4.7.6.23 User Profile

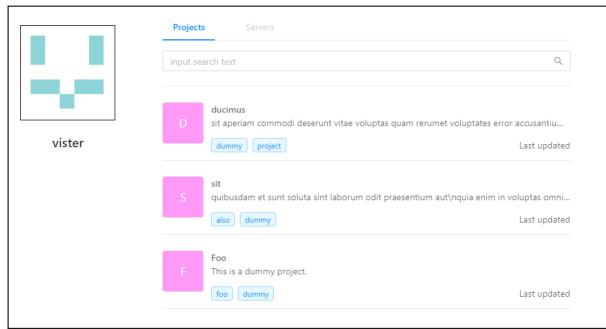


Figure 4.70: Figure showing a public user profile.

If the user were to click on one of the users shown in figure 4.68, one would see a profile page that looks like the one in figure 4.70. We do not store any information relevant to the public in our application. However, a possible future feature would be to allow the user to add a bio, add their real name if they want, personal website, email and address. These implementations would be all up to the user to use.

4.7.6.24 User Project



Figure 4.71: Figure showing a public project overview for a user project.

If the user clicks on one of the projects on a user profile, one would see a project overview like the one in figure 4.71. The project overview includes some public information about the project as well as the modules.

4.7.6.25 Organization Profile

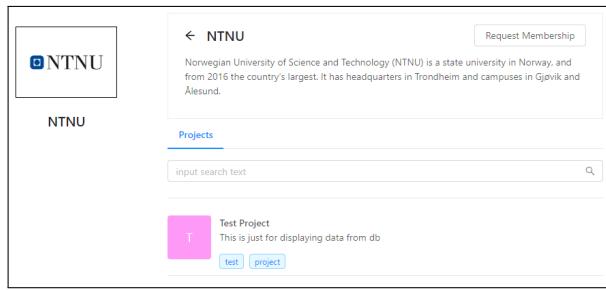


Figure 4.72: Figure showing a public organization profile.

Public organizations also have a profile page which can be accessed by clicking on an organization shown in figure 4.69.

4.7.6.26 Organization Membership

If the user is authenticated, the "request membership" button on the organization profile will be visible. By clicking this button, one will be presented with the page shown in figure 4.73. The username is automatically set to be the same as the one the user is authenticated with. When one presses the submit button, the application will send the data to the organization for handling. When the application has been processed, the user will get notified on the results in the form of an email.

The screenshot shows a membership application form titled "NTNU Member Application". The form is part of the "NTNU" website. It includes fields for "Username" (with placeholder "vister"), "Real name", "Email", "Role" (a dropdown menu with placeholder "Please select a role"), and "Comment". A "Submit Form" button is at the bottom.

Figure 4.73: Figure showing the form to submit a membership application.

4.7.6.27 User Settings

The user also has access to some user settings, as shown in figure 4.74. Currently, the only setting available is the option to change password. The user settings are accessible by hovering over the username in the top right corner, as shown in figure 4.57, and clicking the settings option.

The screenshot shows the "User Settings" page for "AutoPacker". On the left, there's a sidebar with icons for "Your Projects", "Your Servers", and "Your Organizations". The main area has a navigation bar with "Personal settings", "Profile", "Account", and "Password" (which is highlighted). To the right, there's a "Change password" form with fields for "Old password", "New password", and "Confirm password". Below these fields are validation messages: "Make sure the password is at least 8 characters long, including a number and one lower- and uppercase letter" and "Make sure the two passwords matches". At the bottom of the form are "Update password" and "I forgot my password" buttons. In the top right corner, there's a user profile icon with the name "vister".

Figure 4.74: Figure showing the user settings page.

4.8 Docker architecture

Through the whole working process, we have used Docker in our development environment. By utilizing Docker, it is much easier to maintain the tools needed for development. It is easier to work with, more comfortable to change/update and because we use it in our development, staging and production environment. It is much easier to set things up and make it work on all our platforms (personal computer(s) and servers).

4.8.1 Docker compose (staging)

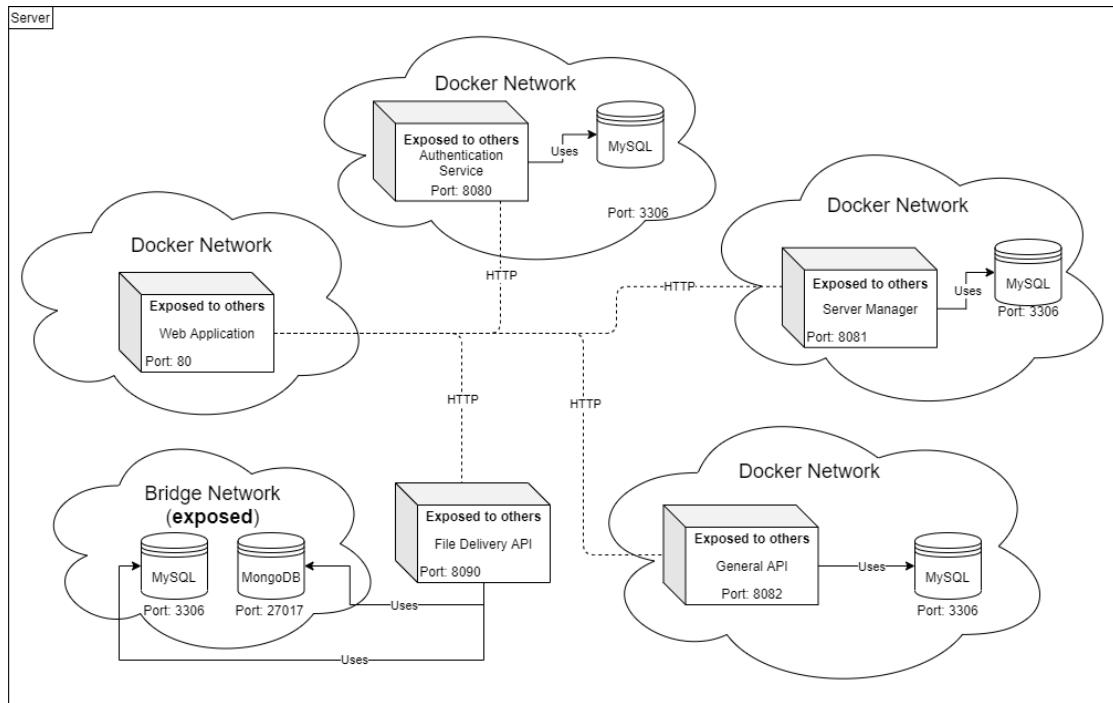


Figure 4.75: Figure showing our staging environment using docker.

The figure above shows our staging setup using docker-compose. For each of our services, we have multiple docker-compose configurations that build our local development environment, testing environment and staging environment.

As every service has its own docker-compose files, they will all be separated into their own docker networks with their own internals. So when one runs "docker-compose up" on one of the docker-compose configurations for one of the services, the docker engine will create a des-

ignated docker network for that service and create the containers needed and attach them to that network.

So in the case of the Authentication Server, there are two containers within the same network. Only the Authentication Service container itself can be accessed outside the network, while the MySQL container gets only exposed within that network. So all the containers that are marked with "exposed to others" or "exposed" are available to the host machine, which also makes them available at host ports.

This is not good for the file delivery API databases that are connected to the bridge network and accessible for the host. The reason they are is that the file delivery API has to run on the host and not inside a docker container due to an issue we had when trying to connect to docker hub that is inside a docker container within a docker container.

The nice thing with having these separated networks for the other services however, is that the inside of the network can be equal to another, but they will not affect one another. For example, we have a MySQL container running on port 3306 in almost all of the networks, but because they are only exposed to the network and not outside, we can do this.

4.8.2 Docker Swarm (production)

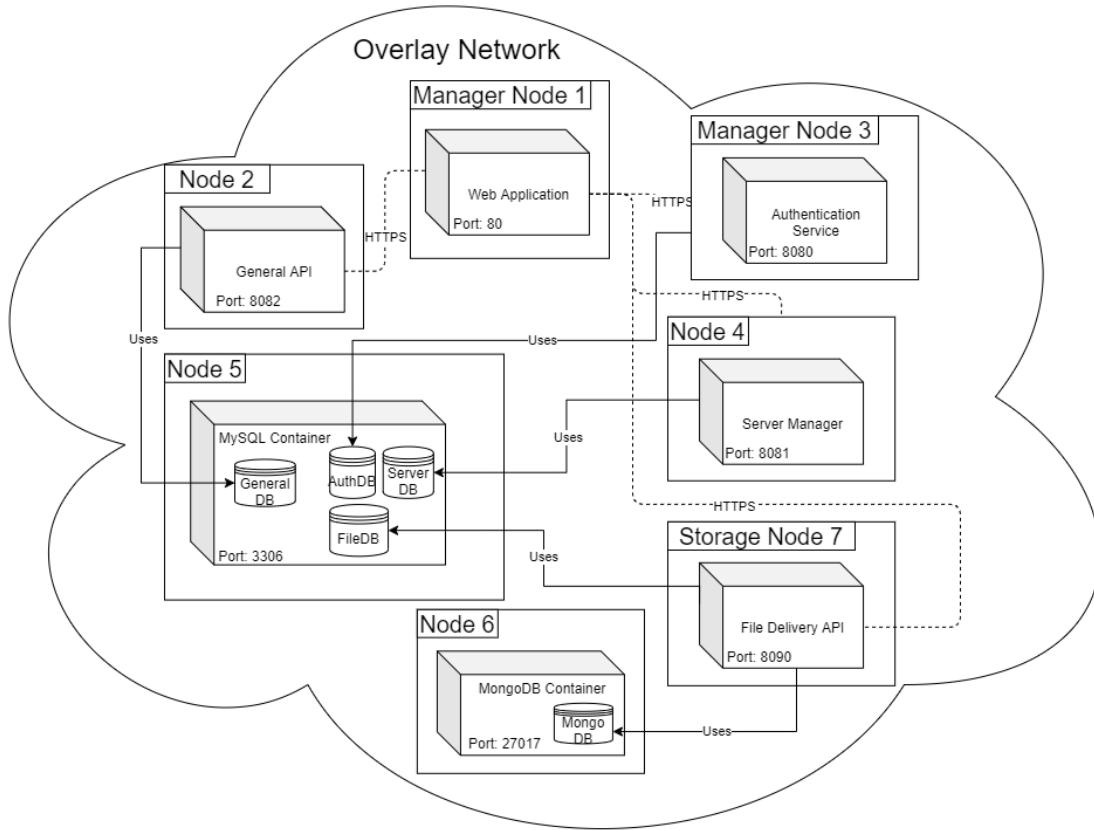


Figure 4.76: Figure showing a concept for a production environment using docker.

The figure shows a concept for a production environment where we use docker swarm to manage our services. This scenario could be something we would pursue in the early stages of production. In this example, we have a server/node for every service (demonstration purposes) that are attached to a shared overlay network that lets the services talk to each other as if they were on the same server. If the system were to scale to many users and more activity, we would have to increase the number of replicas for each service and maybe add more nodes. This for load balancing, redundancy in case of failure (both server and service failure) and blue-green deployment.

For something early production-ready (as it is just a concept), this setup would be good enough with fewer nodes containing all the services. Due to our services having different needs, we can also have different nodes with different specifications. In figure 4.76, we can see that there is a storage node for the file delivery API. This node could have had a lot more storage,

while the others have less, but more CPU or memory, for example.

4.9 Testing

4.9.1 Usability Testing

On May 15th and 16th, we performed usability tests on three participants. One second-year student, one third-year student and a graduate from 2019. These participants had no prior knowledge of the application other than an introduction to the purpose of the application. The participants got told to have a project they wanted to test ready. If they had no projects to use, we provided them with one or more, but to achieve the most realistic results we wanted the participants to use projects we have not tested. Each participant also got an AutoDeploy server if they were unable to provide a server themselves.

The test got performed using Discord with screen sharing and voice chat. The participant performing the test got told to think out loud when navigating through the application. The purpose of this was to gain insight into the applications user-interface friendliness, and user thought process. If the participant came to a dead-end, the observer would then give hints to the user so he could proceed with the test.

4.9.2 Mentor and Supervisor test

We also ran a test with our supervisor Girts Strazdins on May 14th. Girts have been updated quite frequently with the changes done to the application, and therefore the test with him was more to receive critic and feedback seen from a neutral professor/user.

4.9.3 Test Results

Summarizing our test sessions, we found out that the AutoPacker platform might take some time to figure out. It is a platform that takes much navigation to achieve what the user wants to achieve. There should be a more red thread between what the user has done and what to do

next. Some ideas given were to create a "tutorial", or add more tooltips, hints and feedback on actions.

Some actions should be disabled/enabled depending on a given condition, like initializing server only has to be done once, but the button can still be clicked. The usability test also uncovered bugs we were not aware of that were quite critical.

All in all, we got a lot of feedback on visual changes that would make the platform more accessible to understand, easier to navigate between and easier to use. We also got great feedback on how the concept is a great idea, and that they might even use the system for personal projects due to its simpleness of deployment if the platform supported more features, more configuration and became more secure. The project had a lot of potential. Given below is a quote from a note that one of our testers wrote:

"For min egen del så er dette noe jeg kan bruke på mine freelance prosjekt, men for at det skal gå så må der legges til mer options og sikkerheten må bli mye bedre. Foreløpig passer nettsiden best til studierelaterte prosjekt, ikke kommersielle prosjekt."

The full note is available in appendix [O](#).

Chapter 5

Discussion

5.1 Results VS Expectation

If we look at what we have accomplished in the short amount of time we had, we must say we are happy with the results. There are some parts of the system that is either missing or not finished, but on the other side, we were able to implement functionality that we originally had planned for as a future extension. For example, we wanted to implement solutions for checking out a GitHub repo, a following/follower system, forking functionality and supporting multiple languages, frameworks and databases. Due to the lack of time, we were unfortunately not able to implement this. On the other side, we have a concept for organization management and a flexible module builder which can be used to extend language and framework support further later on.

Our system architecture is also entirely modular as we are using micro-services, which makes it easy to implement additional features, split up the complexity and separate workload more efficiently.

We only support public projects as we are using docker hub as registry for storing our images. In the future we would setup our own server with our own registry for storing all images. This way we can make projects private.

We prioritized functionality over security due to the fact that this bachelor project is more of a concept/prototype developed to achieve a specific goal. If we had set the goal to make a

production ready secure application we would have had to let go of a lot of the functionality we currently have due to time restrictions.

5.2 Project Organization

Initially, we had a proper plan for how we should work. What days we were working together, what time we were meeting up and when to have meetings. We could have been better at having stand-up meetings, but other than that we were early to plan our roadmap.

In March, corona started to affect the regular school days, and on March 12th all students were told to leave campus, and the school has been closed since. Luckily as our project is software-based, we were not profoundly affected. The biggest issue we faced with the corona shutdown was that we had to find an everyday work schedule. We started having virtual chat sessions at specific times where we were all working and being available to the rest when we were working on more essential parts, and merging things. All members were able to meet on all meetings with both supervisor and mentor.

5.3 RESTful Services

5.3.1 File Delivery API

It was not easy to develop this API because many techniques had to be applied in order for the service to work. Some of the new implementations make other parts of the system somewhat non-functional. We did, however, manage to fix them without it taking a significant toll on our time. The good thing about this is that most of the problems came because of something we did not know and gave us the opportunity to learn more, which is essential.

Much time went into creating the processing of the uploaded module. There were many scenarios to account for. The docker-compose builder works with simple docker services but requires a change when extending docker services by, for example adding volumes, networks, etc. However, this implementation serves us well for now.

One major issue was trying to dockerize this API. The File Delivery API itself needs to run docker commands in order to dockerize the modules. This is a problem because one is not supposed to run docker commands inside a docker container since it is inconvenient and recommended not to do so. Since having the API package the modules into images was deemed more important than dockerizing the actual API, we decided not to dockerize the API after all.

There is a lot of I/O operations happening on the API. It is vital that race conditions are prevented. Having a folder for each user for their projects and modules greatly reduces this risk. Most likely, a user will not be able to upload two or more modules at the same time. If that happens, both requests must have the same project and module name for it to be an issue. In case that is not enough, we have implemented a simple flag semaphore with the MySQL database, checking first if a module has been added with the same name, project and user. It is very little probable that a race condition occurs with this implementation.

If we were going to improve this API, we would start with supporting generating more complex docker-compose files. Then the pool of projects that can be made with the API dramatically increases. After finishing that, it would be wise to go through the API and strengthen the stability of the module uploading, and if possible, dockerize the API too.

5.4 Security

We should ask the user for permission when connecting to the users' servers and doing things. If we were to lose control of our system, or if somebody were to get control of our system, they could access all of our users' servers and use them for malicious intent. Therefore we should have consent from the user and show them a notice on what will happen if they use the server deployment functions on the website.

5.5 Web Application

As mentioned in section 5.1 we did overall end up with a solution we are quite happy with, even though some parts are missing. In the beginning, we did not think we would be able to

create such a polished and nice looking web interface. The reason we were able to develop so many high-quality web components and interface pieces is due to the scalability, reusability and simplicity of React and the high-quality react component library, Ant Design.

5.5.1 Design & Interface

We agreed on the start not to spend too much time on visuals and design, which is the main reason why we choose to use a component library like Ant Design. This way, we spent less time dealing with styling and more time on implementing features.

Even though we had already agreed to have the layout we have implemented early on, we were heavily inspired by the Ant Design Pro panel layout. Most of the parts in our interface is built using existing or customized Ant Design components. Most of the components do have more than enough properties to alter the behaviour, but many did not have the features we wanted. In this case, much time went into finding a hacky way of combining ant design components with custom components and logic which was quite hard in some cases, as Ant Design components are hard coded and can be challenging to alter. In these scenarios, we might have been better off designing the components from scratch or using a CSS framework like [Tailwind CSS](#) or [Bootstrap](#).

A poor design choice is the use of constant inline JSX styling to style components or override existing styles. The main idea was to use this to prototype designs but quickly turned in to becoming the primary way to style components. In future modifications and maintenance, we would like to use [Styled Components](#) to style our components.

5.5.2 Code Quality & Structure

In some places, the code can be quite messy as we did try and implement as many features possible in the time we had. Unfortunately, this affected the quality and structure. In most places, the code is well written, but the structure could have been better. Some files contain code that does not make sense to have there, and some files contain old logic for dealing with a problem. The overall file system structure could also need reformatting before making the project open

source to make it easier to navigate through the application. After the web application passed 100 files and 60 folders, things started to become challenging to find and navigate through.

5.6 Production

There are several steps and demands that need to be met before even considering deploying the application to a real production environment. First of all, we need to configure HTTPS between all services and the web application. We also need a better way of handling secrets and keys vital to different services. We need restrictions for who can upload what, and a type of project validation so that a person does not upload malware that other than downloads.

5.7 Limitations

When a user wants to upload a module, the user is not presented with many options regarding supported frameworks, script and programming languages. There may be a case where the user wants to upload a connected multi-service project, for example, a frontend, backend and some form of a database. If the technology of the frontend is unsupported, the project will not work as intended, and in the worst scenario, the project is not working at all. We have collected survey data on the most popular languages among our participants. The data showed us what languages we should prioritize implementing support for, reducing the issue stated previously.

Deploying these projects to a server will, for the most part, work. Our preparation script for the server is primitive and could cause problems if certain conditions are not met. The target server must use a 64-bit Ubuntu distribution and have a package named [ufw](#) installed managing the firewall. If any of the port is occupied, the deployment could fail, and no error message will show up.

5.8 Future Work

Supporting too few languages could be problematic if more people use the solution. However, this will not become a significant task as the system is developed such that adding new config-

urations is quick and straightforward.

Extending the system with simple configurations will not be an issue, but having a more complex configuration may break the system if implemented at the current state. Because of the continuous surfacing of unexpected bugs, much time was lost. This led us to implement the fixes as simple as possible because it was faster. This left some part of our system inefficient, and the stability reduced. In the future, increasing stability is crucial, so maintaining the system gets more manageable, and the users stumble upon fewer problems.

Since we keep all the files in the File Delivery API, half the work was done to implement the file exploration. We did have an overview on how to add the exploration. Implementation of this feature was already in progress. Later into development, the realization came that it would take time we did not have. The implementation under development on the back-end was scraped, but it is still possible to extend it again in the future.

A helpful feature would be to create a dynamic Dockerfile builder similar to the compose template builder. It can be used to create a more flexible and complex build of a module upload. Users will have more control over the build by being able to change more of the configuration. This results in a much more reliable upload that is tailored to the environment requirements of the users.

Putting the underlying parts of the system away, a significant future improvement is updating the visuals of the frontend application. Our user testing has given us pointers on how that can be done. Implementing UX/UI concepts besides the test results lower the learning curve and gives the user a more pleasant experience using AutoPacker.

Chapter 6

Conclusions

The purpose of this project was to develop a solution that would make it easier to build, distribute and deploy different types of projects using today's virtualization technology.

We did meet our minimum product requirement. The application supports simple server provisioning, project creation, uploading of a JAR executable and ZIP containing either a Java, Spring-boot, React, Angular or a straightforward HTML, CSS and JavaScript project. A server can hold and deploy multiple projects. We also have a subsystem concept for organizations, which the user can join, submit projects and if the user is an admin, he can manage organizational data. We were able to build a foundation for our docker image/compose builder as well as a working wizard for configuring modules and so much more.

This solution is primarily targeted towards a lectures where the students can with little to no hassle run a project distributed by the lecturer. Time used to setup the project is reduced, dedicating more time to the actual lecture. The solution can also be used outside lectures where students can deliver their assignments and allow the lecturer to deploy and grade the assignment quicker. Other developers can upload their projects if they want to easier build and distribute them.

There are not many configurations that's available to the users and could be a problem when a user uploads multiple modules where half of the technologies are supported. We have taken into account the survey to find out the most used technologies and implemented those

configurations. It is not possible to collaborate with people on projects, and only the project owner can manage it. If a user wants to deploy projects to a server, it needs to be clean, i.e. not have docker installed.

Workload compared to the time available and the occurrence of unexpected bugs has led us to implement and fix bugs fast. Some corners had to be cut in order to complete tasks. As a result of that, some code is inefficient. Improving stability is the next step of AutoPacker alongside adding more configurations for the users. For the sake of the user, applying more UX/UI concepts to reduce the learning curve. We are happy with the results, even though we were not able to finish all the functionality we had hoped to implement.

Bibliography

- [1] URL <https://en.wikipedia.org/wiki/Bcrypt>.
- [2] What is mongodb? introduction, architecture, features & example. URL <https://www.guru99.com/what-is-mongodb.html>.
- [3] What is rest. URL <https://restfulapi.net/>.
- [4] Sha-2. URL <https://en.wikipedia.org/wiki/SHA-2>.
- [5] Sql injection. URL <https://portswigger.net/web-security/sql-injection>.
- [6] Create, read, update and delete. URL https://en.wikipedia.org/wiki/Create,_read,_update_and_delete.
- [7] Cross-site scripting. URL <https://portswigger.net/web-security/cross-site-scripting>.
- [8] Zip slip vulnerability. URL <https://snyk.io/research/zip-slip-vulnerability>.
- [9] Elaine Barker and Allen Roginsky. Transitioning the use of cryptographic algorithms and key lengths. Technical report, National Institute of Standards and Technology, 2018.
- [10] Igor Beliaiev. Containers vs. virtual machines, 2017-06-27. URL <https://www.softserveinc.com/en-us/blog/security-containers-vs-virtual-machines>.
- [11] Bivás Biswas. How not to blow your rest interview. URL <https://medium.com/future-vision/the-principles-of-rest-6b00deac91b3>.

- [12] SQLizer Blog. Json store data, 2017.05.04. URL <https://blog.sqlizer.io/posts/json-store-data/>.
- [13] Jamie Kyle Christoph Nakazawa, Sebastian McKenzie. Yarn: A new package manager for javascript, 2016.10.11.
- [14] Don Coppersmith. The data encryption standard (des) and its strength against attacks. *IBM journal of research and development*, 38(3):243–250, 1994.
- [15] Datatilsynet. What is privacy?, 2019-07-17. URL <https://www.datatilsynet.no/rettigheter-og-plikter/hva-er-personvern/>.
- [16] React Redux Official Docs. Hooks, 2020-02-03. URL <https://react-redux.js.org/api/hooks>.
- [17] Docker Official Documentation. Use bind mounts, n.d. URL <https://docs.docker.com/storage/bind-mounts/>.
- [18] Docker Official Documentation. How nodes work, n.d. URL <https://docs.docker.com/engine/swarm/how-swarm-mode-works/nodes/>.
- [19] Docker Official Documentation. About storage drivers, n.d. URL <https://docs.docker.com/storage/storagedriver/>.
- [20] Docker Official Documentation. Networking overview, n.d. URL <https://docs.docker.com/network/>.
- [21] Docker Official Documentation. Use overlay networks, n.d. URL <https://docs.docker.com/network/overlay/>.
- [22] Docker Official Documentation. Docker overview, n.d. URL <https://docs.docker.com/get-started/overview/>.
- [23] Docker Official Documentation. Docker registry, n.d. URL <https://docs.docker.com/registry/>.

- [24] Docker Official Documentation. Manage sensitive data with docker secrets, n.d. URL <https://docs.docker.com/engine/swarm/secrets/>.
- [25] Docker Official Documentation. How services work, n.d. URL <https://docs.docker.com/engine/swarm/how-swarm-mode-works/services/>.
- [26] Docker Official Documentation. Deploy a stack to a swarm, n.d. URL <https://docs.docker.com/engine/swarm/stack-deploy/>.
- [27] Docker Official Documentation. Use volumes, n.d. URL <https://docs.docker.com/storage/volumes/>.
- [28] Spring Boot Documentation. Introduction to the spring ioc container and beans, n.d. URL <https://docs.spring.io/spring/docs/current/spring-framework-reference/core.html#beans>.
- [29] Spring Boot Documentation. Container overview, n.d. URL <https://docs.spring.io/spring/docs/current/spring-framework-reference/core.html#beans>.
- [30] Spring Boot Documentation. Bean overview, n.d. URL <https://docs.spring.io/spring/docs/current/spring-framework-reference/core.html#beans-definition>.
- [31] Adam DuVander. Docker vs vagrant: What you need to know, 2016-04-12. URL <https://www.ctl.io/developers/blog/post/docker-vs-vagrant>.
- [32] IBM Cloud Education. Containerization, 2019-05-15. URL <https://www.ibm.com/cloud/learn/containerization>.
- [33] EUR-Lex. General data protection regulation, 2016-05-04. URL <https://eur-lex.europa.eu/eli/reg/2016/679/oj>.
- [34] Facebook. Introducing hooks, n.d. URL <https://reactjs.org/docs/hooks-intro.html>.
- [35] GitLab. Devops tools landscape, n.d. URL <https://about.gitlab.com/devops-tools/>.

- [36] GitPod. Homepage, n.d. URL <https://www.gitpod.io/>.
- [37] Charles Graziano. A performance analysis of xen and kvm hypervisors for hostig the xen worlds project, 2013-01-29. URL <https://lib.dr.iastate.edu/cgi/viewcontent.cgi?referer=https://en.wikipedia.org/&httpsredir=1&article=3243&context=etd>.
- [38] Red Hat. What is blue green deployment?, n.d. URL <https://www.redhat.com/en/topics/devops/what-is-blue-green-deployment>.
- [39] Heroku. Homepage, n.d. URL <https://www.heroku.com/>.
- [40] David Ireland. Rsa algorithm. *DI Management*, 2011.
- [41] Michael Jones, Brain Campbell, and Chuck Mortimore. Json web token (jwt) profile for oauth 2.0 client authentication and authorization grants. *May-2015.[Online]. Available: https://tools.ietf.org/html/rfc7523*, 2015.
- [42] Sahiti Kappagantula. Docker networking – explore how containers communicate with each other, 2019-11-27. URL <https://www.edureka.co/blog/docker-networking/>.
- [43] Xah Lee. What is yaml, 2013-06-04. URL http://xahlee.info/comp/what_is_YAML.html.
- [44] Guy Levin. Restful api basic guidelines. URL <https://blog.restcase.com/restful-api-basic-guidelines/>.
- [45] opensource.com. What is docker?, n.d. URL <https://opensource.com/resources/what-docker>.
- [46] Oracle. What a relational database is, n.d. URL <https://www.oracle.com/database/what-is-a-relational-database/>.
- [47] Oracle. What is mysql, n.d. URL <https://dev.mysql.com/doc/refman/8.0/en/what-is-mysql.html>.

- [48] Sten Pittet. What is continuous deployment?, n.d. URL <https://www.atlassian.com/continuous-delivery/continuous-deployment>.
- [49] Rajesh Radhakrishnan. Docker architecture and components, 2018-08-04. URL <https://vmarena.com/docker-architecture-and-components/>.
- [50] Gabe Ragland. Debouncing with react hooks, 2019-12-19. URL https://dev.to/gabe_ragland/debouncing-with-react-hooks-jci.
- [51] Redux. Usage with react, n.d. URL <https://redux.js.org/basics/usage-with-react>.
- [52] Max Rehkopf. What is continuous integration, n.d. URL <https://www.atlassian.com/continuous-delivery/continuous-integration>.
- [53] Cody Reichert. 7 http methods every web developer should know and how to test them. URL <https://assertible.com/blog/7-http-methods-every-web-developer-should-know-and-how-to-test-them>.
- [54] SiteGround. Mysql tutorial, n.d. URL <https://www.siteground.com/tutorials/php-mysql/mysql/>.
- [55] Belinda Smith. Prime numbers keep your encrypted messages safe. URL <https://www.abc.net.au/news/science/2018-01-20/how-prime-numbers-rsa-encryption-works/9338876>.
- [56] Symfony. Yaml format, n.d. URL https://symfony.com/doc/current/components/yaml/yaml_format.html.
- [57] Visual-Paradigm. What is agile development, n.d. URL <https://www.visual-paradigm.com/scrum/what-is-agile-software-development/>.
- [58] w3schools. What is json, n.d. URL https://www.w3schools.com/whatis/whatis_json.asp.
- [59] Wikipedia. Docker (software), 2020-04-23. URL [https://en.wikipedia.org/wiki/Docker_\(software\)](https://en.wikipedia.org/wiki/Docker_(software)).

- [60] Wikipedia. React (web framework), 2020-05-11. URL [https://en.wikipedia.org/wiki/React_\(web_framework\)](https://en.wikipedia.org/wiki/React_(web_framework)).
- [61] Wireshark. About wireshark, n.d. URL <https://www.wireshark.org/>.

Appendices

A Preliminary Report

FORPROSJEKT - RAPPORT

FOR BACHELOROPPGAVE



Kunnskap for en bedre verden

TITTEL:

AutoPacker

KANDIDATENE:

Aron Mar Nicholasson, Liban B. Nor & Bendik Uglem Nogva

DATO:	EMNEKODE:	EMNE:	DOKUMENT TILGANG:
08.01.2020	IE303612	Bacheloroppgave	- Åpen
STUDIUM:		ANT SIDER/VEDLEGG:	BIBL. NR:
INGENIØRFAG - DATA		19/2	- Ikke i bruk -

OPPDAGSGIVER(E)/VEILEDER(E):

Avento (mentor), Girts Strazzins (hovedveileder), Mikael Tollesen (biveileder)

OPPGAVE/SAMMENDRAG:

Oppgaven er egendefinert og går ut på å gjøre det lettere å publisere, administrere og distribuere et programvareprosjekt. Ved hjelp av en web applikasjon kan man lage offentlige eller private "prosjekt". Her kan man laste opp enten kildekode fra lokal PC eller URL for GitHub og GitLab. Når dette er valgt kan man generere et kjørbart "docker" image og/eller en docker compose fil. Disse kan lastes ned og kjøres lokalt eller ved å deploye til en bruker valgt server og kjøres der. Denne tjenesten skal også kunne holde på organisasjoner som igjen holder på flere slike "prosjekt". På denne måten kan NTNU ha en organisasjon som bevarer på alle programvare relaterte bachelorene gjort.

Denne oppgaven er en eksamensbesvarelse utført av studenter ved NTNU i Ålesund.

Postadresse
Høgskolen i Ålesund
N-6025 Ålesund
Norway

Besøksadresse
Larsgårdsvegen 2
Internett
www.hials.no

Telefon
70 16 12 00
Epostadresse
postmottak@hials.no

Telefax
70 16 13 00

Bankkonto
7694 05 00636
Foretaksregisteret
NO 971 572 140

INNHOLD

INNHOLD	2
1 INNLEDNING	3
2 TERMINOLOGI.....	4
2.1 BEGREPER	4
2.2 FORKORTELSER	4
3 AVTALER	5
3.1 ARBEIDSSTED	5
3.2 GRUPPENORMER – SAMARBEIDSREGLER – HOLDNINGER	5
4 PROSJEKTBESKRIVELSE	6
4.1 PROBLEMSTILLING – MÅLSETTING	6
4.2 KRAV TIL LØSNING ELLER PROSJEKTRESULTAT – SPESIFIKASJON.....	7
Funksjonelle krav	7
Ikke-funksjonelle krav	11
4.3 PLANLAGT FRAMGANGSMÅTE(R) FOR UTVIKLINGSARBEIDET – METODE(R).....	12
4.4 INFORMASJONSINNSAMLING – UTFØRT OG PLANLAGT.....	13
4.5 VURDERING – ANALYSE AV RISIKO	14
4.6 FRAMDRIFTSPLAN – STYRING AV PROSJEKTET	15
4.6.1 <i>Hovedplan</i>	15
4.6.2 <i>Styringshjelpe midler</i>	15
4.6.3 <i>Utviklingshjelpe midler</i>	15
4.6.4 <i>Intern kontroll – evaluering</i>	16
4.7 BESLUTNINGER – BESLUTNINGSPROSSESS	16
5 DOKUMENTASJON	17
5.1 RAPPORTER OG TEKNISKE DOKUMENTER.....	17
6 PLANLAGTE MØTER OG RAPPORTER.....	18
6.1 MØTER.....	18
6.1.1 <i>Prosjektmøter</i>	18
7 PLANLAGT AVVIKSBEHANDLING	18
8 UTSTYRSBEHOV/FORUTSETNINGER FOR GJENNOMFØRING	19
VEDLEGG.....	19

1 INNLEDNING

Bakgrunnen for at vi valgte denne egendefinerte oppgaven er på grunnlag av egen erfaring og at vi ser at dette er en løsning som kan hjelpe spesielt studenter, men også enkeltpersoner som har utvikling som en hobby. Vi ønsker å lage en løsning som automatiserer prosessen ved å gå fra kildekode til en plattform uavhengig løsning som kan kjøres både lokalt på egen maskin og på en spesifisert ekstern server som er automatisk konfigurert og kjører applikasjonen i et virtualisert miljø. På denne måten kan det brukes for testing, utprøving og drift av tidligere og nyere prosjekter.

Oppdragsgiver – Oppgaven er egendefinert, men vi har valgt å involvere Avento som mentor grunnet deres kompetanse innenfor den teknologien og plattformen vi skal ta i bruk.

2 TERMINOLOGI

2.1 Begreper

Docker	Docker er et verktøy for å lettare lage, distribuere og kjøre applikasjoner ved hjelp av docker containers.
Docker Image	Et docker image er en kjørbar pakke av forskjellig software, bibliotek, verktøy og innstillinger.
Docker Container	En docker container er en prosess instansiert fra et docker image.
Frontend	Den delen av et datasystem eller applikasjon som brukeren samhandler direkte med.
Backend	Den delen av et datasystem eller applikasjon som ikke er direkte tilgjengelig av brukeren.
Sprint	En sprint er en tidsbokset iterasjon av en kontinuerlig utviklingssyklus ¹
Agile/smidig	En arbeidsmetode med fokus på iterasjoner av arbeidet.
Scrum	En smidig arbeidsmetode hvor man i syklus planlegger, tilpasser, endrer og utvikler i iterasjoner.
Chiffer	Fellesbetegnelse på krypterings algoritmer
User stories	User stories er beskrivelse av features.

2.2 Forkortelser

API	Application Programming Interface
SSH	Secure Shell
CI	Continuous Integration
CD	Continuous Deployment
JWT	JSON Web Token
PUKA	Plattform Uavhengig Kjørbar Applikasjon

¹ <https://yodiz.com/help/what-is-sprint/>

3 AVTALER

3.1 *Arbeidssted*

Vi kommer hovedsakelig til å oppholde oss på lab rommet L167 siden det rommet er reservert for studenter som skriver bachelor oppgave. Vi har også mulighet for å jobbe på Avento sine lokaler ved behov. Denne muligheten kommer vi i hovedsak til å utnytte i sammenheng med retrospektivt møte etter møte med mentor.

3.2 *Gruppenormer – samarbeidsregler – holdninger*

Det viktigste for å oppnå en god arbeidsprosess er kommunikasjon. Uten kommunikasjon er det lite fremgang og mye som kan slå feil. I hovedsak kommer vi til å jobbe sammen på skolen i hverdager. Dette for å engasjere diskusjon, kompetansedeling og forskjellige syn som kan bidra til mer effektive løsninger, nye ideer og bedre sosialt samvær.

Alle i gruppen skal være dedikert til oppgaven, det skal legges inn en innsats til å fullføre prosjektet. Dette kan lett gjøres ved at gruppen er streng angående tilstedeværelse, og at man skal være aktiv i prosjektarbeid og møter av alle slag.

Vi har ingen form for prosjektledelse struktur og dermed ingen hierarki på ansvarsfordelinger. Dermed vil fordeling av oppgaver og hvem som har ansvar for hva diskuteres frem til og delegeres i interne gruppemøte.

Dersom det skulle oppstå konflikter i gruppen skal det tas opp i et internt møte. Dersom det ikke blir løst internt vil vi ta kontakt med veileder for retningslinjer og eventuelt innkalling til møte.

4 PROSJEKTBESKRIVELSE

4.1 Problemstilling - målsetting

Problemstilling:

Hvordan gjøre det lettere for studenter/enkeltindivid å gjøre kildekode om til en plattform uavhengig kjørbar applikasjon, og som gjør distribusjon, testing og drift av denne applikasjonen lettere.

I vårt prosjekt har vi satt følgende mål:

<i>Effektmål</i>	Løsningen skal eliminere mest mulig steg fra kildekode lastet opp, til en plattform uavhengig kjørbar applikasjon (PUKA), der man ikke trenger stor teknisk kunnskap.
<i>Resultatmål</i>	Utvikle en selvstendig sikker applikasjon som spesielt studenter som skriver bachelor kan ta i bruk, og som har støtte for de mest vanlige kodeprosjektene innen midten av mai.
<i>Prosessmål</i>	Gruppen skal følge den framgangsmåten beskrevet, helst med så få komplikasjoner som mulig. Og dersom det skulle oppstå noe, klare å komme seg inn på riktig bane kjappest mulig.

4.2 Krav til løsning eller prosjektresultat – spesifikasjon

Funksjonelle krav

#	Tittel	Bruker funksjonalitet	Prioritet	Notater
1	Besøke nettområde	Som en bruker vil jeg se informasjon/hjemmestedet når jeg først besøker nettområdet som "gjest".	KRITISK	
2	Konsoll side	Som en bruker vil jeg ha tilgang til en konsoll side, hvor jeg kan søke, se og laste opp/ned forskjellige prosjekter	KRITISK	Profil "dashboard" siden ser litt ut som et admin dashboard.
3	Opprette prosjekt	Som en bruker vil jeg kunne opprette en hoved mappe som inneholder alle modulene prosjektet er bygd opp av	KRITISK	Bruker kan gjennom profil "dashboard" velge "nytt prosjekt" som han/hun kan opprette. I prosjektet kan man ha en eller flere tjenester.
4	Modul Implementasjon	Som bruker vil jeg kunne tilknytte moduler til et eksisterende prosjekt.	KRITISK	Disse modulene kan for eksempel være: nettapplikasjon, API, mobilapp, database osv.
5	Valg av avhengigheter / moduler	Som en bruker vil jeg finne og velge de nødvendige avhengigheter prosjektet behøver for å kunne kjøre	KRITISK	Brukeren kan (med hjelp av installeringsveiviser for å finne og velge avhengighetene som trengs for å bygge prosjektet. Dette kan også

				gjøres ved at brukeren legger ved en "docker compose file"
6	Slette prosjekt	Som en bruker kan jeg slette prosjekter jeg har opprettet	KRITISK	Se at brukeren har rettighetene til å slette prosjekt, Hvis det er et delt prosjekt og brukeren har rettighet til å slette prosjekt som er opprettet av andre.
7	Søke etter prosjekt	Som en bruker vil jeg kunne søke etter eksisterende prosjekter med gitt navn og tags	KRITISK	Brukeren burde kunne finne prosjekt ved å søke etter dem, enten etter navn eller tag
8	Nedlastning	Som bruker vil jeg kunne laste ned prosjekter fra nettapplikasjonen	KRITISK	
9	Autentisering	Som en bruker vil jeg få tilgang til applikasjonen etter autentisering	KRITISK	Lages med Spring Boot (Spring Security)
10	Registrering	Som en bruker kan jeg registrere meg selv slik at jeg kan bruke tjenesten	KRITISK	Brukeren burde kunne registrere seg med minst mulig informasjon (email, brukernavn og passord)
11	Utlogging	Som en bruker kan jeg logge ut av applikasjonen	KRITISK	
12	Email Verifisering	Som en bruker kan jeg verifisere brukerkontoen min via en email sendt til meg	VIKTIG	Implementerer en EmailService
13	Endre passord	Som en bruker vil jeg, kunne endre mitt passord	VIKTIG	Må legges til autentiseringsserver tjenesten

14	Glemt passord	Som en bruker vil jeg kunne tilbakestille passordet mitt.	<u>VIKTIG</u>	Må legges til autentiseringsserver tjenesten
15	NTNU (Feide) integrasjon	Som en bruker vil jeg kunne verifisere brukerkontoen min ved å logge inn via Feide	<u>VIKTIG</u>	Vi skal se nærmere på hvordan vi kan utføre Feide integrasjonen.
16	Brukerinnstilling	Som en bruker kan jeg endre brukerinnstillinger	<u>TRIVIELL</u>	
17	Tilknytninger	Som en bruker kan jeg tilknytte meg til en organisasjon	<u>TRIVIELL</u>	Eksempel på dette kan være NTNU, UIO osv.
18	Sikkerhet	Som en bruker vil jeg vite at det jeg laster ned er trygt	<u>KRITISK</u>	Dette kan utføres ved at enkelte brukere "signerer" prosjektene, eller ved hjelp av crawlers for å unngå skadelig programvarer lastes ned.
19	Predefinert konfigurasjon	Som en bruker vil jeg kunne definere konfigurasjonen når jeg laster opp prosjekt.	<u>KRITISK</u>	Hvis brukeren allerede har en konfigurasjon som trengs, trenger vi ikke å generere den. En brukerdefinert kan være mer spesifik og innstilt på prosjektet sitt
20	Kjøre lokalt	Som en bruker vil jeg kunne kjøre prosjekt lokalt på maskinen min	<u>VIKTIG</u>	Vi behøver en desktop brukergrensesnitt for å håndtere docker containers. I første omgang kan brukeren bare bruke installert docker og bruke de

				samme scripts som eksterne "modulen" av prosjektet.
21	Server konfigurasjon	Som en bruker vil jeg kunne beskrive nok server informasjon slik at applikasjonen kan konfigurere serveren for meg	KRITISK	Brukeren oppgir SSH, brukernavn, passord slik at vi kan tilkoble en server via SSH. Deretter kan vi bruke verktøy slik som Ansible til å konfigurere serveren.
22	Kjøre eksternt	Som bruker vil jeg kunne kjøre et prosjekt på en gitt server (se story ovenfor)	KRITISK	Dette er på en måte det samme som ovenfor. Når serveren er konfigurert, skal prosjektet deretter kjøres
23	Offentlige prosjekt	Som en bruker vil jeg kunne opprette prosjekt som er tilgjengelig for alle	KRITISK	For å dele prosjekt med andre brukere, så må prosjektet være åpen kildekode.
24	Private prosjekt	Som en bruker vil jeg kunne opprette prosjekt som er kun tilgjengelig for meg	VIKTIG	

Ikke-funksjonelle krav

#	Tittel	Forklaring
1	Lett forståelig brukergrensesnit	<ul style="list-style-type: none">• Lett å forstå, brukeren trenger ingen kjennskap til applikasjon for å kunne bruke den
2	Responsivt brukergrensesnitt	<ul style="list-style-type: none">• Brukgrensesnittet responderer på inputs, vindu størrelse og enhet.• Brukeren får indikator når applikasjon holder på å kjøre. (ventetid)
3	Kompatibel med ulike nettlesere.	<ul style="list-style-type: none">• Nettapplikasjonen kan kjøre på alle typer nettlesere
4	Sikkerhet	<ul style="list-style-type: none">• Forsikre brukeren at passord og brukernavn er sendt og lagret sikkert/kryptert<ul style="list-style-type: none">◦ Alt av kommunikasjon vil bruke HTTPS kommunikasjon◦ Alt av passord vil bli lagt til SALT når det lagres i database.◦ Implementer chiffers (SHA256)
5	Kvalitetssikring	<ul style="list-style-type: none">• Kildekode skal være lett å forstå• For å sikre at systemet fungerer som den er beskrevet, benytter vi oss av flere teknikker for testing<ul style="list-style-type: none">◦ Black box◦ White box◦ Acceptance testing

4.3 Planlagt framgangsmåte(r) for utviklingsarbeidet – metode(r)

Gruppen har blitt enig om å bruke scrum som arbeidsprosess "rammeverk" for *smidig* utvikling i dette bachelorprosjektet. Vi har planlagt å ha sprints som spenner seg over to uker. Siden vi har møte med veileder/mentor hver annen uke så passer det bra med to ukers sprints. Vi kommer også til å kjøre et internt standup møte helst før hver lab økt. Her kommer vi litt til å se an behovet. Dersom alle er oppdatert på arbeidet til hverandre er det ikke nødvendig å ha det hver gang.

Hovedgrunnen til at vi valgte å bruke scrum som arbeidsprosess "rammeverk" er på grunn av at den gir oss mye fleksibilitet i prosjektet vårt. Vi kan sette opp arbeidsfremgangen vår slik at den passer oss, veileder og vår mentor. Og dersom det skjer en endring kan vi kjapt tilpasse prosjektet.

Det som kan være den største svakheten med å bruke denne arbeidsprosessen er at siden prosjektet er egendefinert, så kan det være vanskelig å se for seg hvordan det faktisk ferdige produktet skal se ut. Dette kan føre til en dårligere langtidsplan som kan føre til at vi ender opp med å bruke mye tid på å tilpasse prosjektet senere. Et annet problem kan være å fordele arbeidsoppgaver. Noen implementasjoner er avhengig av at andre implementasjoner har tatt sted og noen ikke. Det kan dermed være vanskelig å delegere arbeidsoppgaver mellom oss som er "uavhengig" eller som vi vet vi rekker å fullføre slik at en annen kan ta i bruk løsningen.

Å bruke scrum krever innsats fra alle involverte, og det krever god planlegging. For å dermed unngå å havne i fellene over så er det kjempe viktig at vi har en god taktikk for hvordan vi skal arbeide og en god plan som vi klarer å holde oss til. Vi tenker at ukentlige møter, retrospektivt møte, og scrum møter er nok til at vi klarer å holde oss på stø kurs.

4.4 **Informasjonsinnsamling – utført og planlagt**

Det finnes ingen løsning som gjør nøyaktig det vi har planlagt, men det finnes mange tjenester som forenkler prosessen av å administrere docker containers. Her har du for eksempel Docker desktop og portainer.io. Så har man også løsninger som gjør om en kommando linje om til docker-compose yml som for eksempel Composerize.

Vi kommer til å bruke flere tjenester for å sikre oss informasjon som er nødvendig for at vi finner løsninger og forbedringer til vårt prosjekt. Blant disse er:

- [StackExchange](#) (I hovedsak [StackOverflow](#)) - StackExchange er nok en av de største Q&A samfunnene tilgjengelig og er en “go-to” for programvare utviklere. StackOverflow har en gigantisk database med spørsmål og svar, og i de tilfellene hvor man ikke finner noe svar til sitt problem kan man stille spørsmål og få veldig gode svar
- [Docker documentation](#) - Dokumentasjonen viser hvordan man skal benytte seg av Docker.
- [Spring Boot Reference](#) – Dokumentasjon som viser hva løsninger Spring Boot har å tilby, samt hvordan vi kan oppnå ønskede resultater
- [Spring Security Reference](#) – Informasjon om hvordan man skal implementere sikkerhet i våre Spring Boot applikasjoner.
- [NTNU Open](#) - NTNU Open er en oppbevaringsbase som holder på masteroppgaver, bacheloroppgaver, forsknings artikler, tidsskrifter og mye annet vi kan lese og ta inspirasjon fra til vår bachelor skriving.
- [Ei Engineering Village](#) - Engineering Village er en plattform som inneholder journaler, tekniske rapporter, vitenskapelige artikler og dokument og er en ressurs som kan brukes for kvalitetssikre kilder.
- [ScienceDirect](#) - ScienceDirect er også en plattform som inneholder journaler, vitenskapelige artikler og elektroniske bøker. Alt som kan finnes her er autentisk og dermed trenger man ikke å bruke masse tid på å verifisere kildene.
- [Baeldung](#) - baeldung.com er en go-to nettside for java utviklere som ønsker seg å lære mer om java økosystemet, spring microservices, spring security og mye mer i form av guides og courses.
- [React Docs](#) - Dokumentasjon over hva react har å tilby og hvordan man kan implementere de inn i prosjekt (med eksempler).
- [Redux Docs](#) (with React) – Dokumentasjon som viser hvordan man kan bruke redux med React.

- [Docker Mastery \(Udemy course\)](#) - Docker Mastery er et ~20t kurs som er tilgjengelig på Udemy, som er den mest kjente plattformen for å finne betalte kurs av instruktører (gjerne sertifiserte).
- [Google Scholar](#) – Google Scholar er en søkemotor som er ment for å finne akademisk litteratur.

I noen av de overnevnte kildene er det viktig at man som enkeltperson kvalitet sikrer det innholdet man finner og eventuelt velger å ta i bruk. Man må finne om dette er noe man kan implementere, ta utgangspunkt i eller forbedre slik at det passer inn i prosjektet. Man må også tenke over hvilken virkning det har, om det påfører risiko eller det har konsekvenser.

4.5 *Vurdering – analyse av risiko*

Vi har ikke god nok teknisk forståelse på hvordan alle elementer av systemet skal fungere sammen. Der vi har estimert en viss tid til en funksjonalitet, er det mulig at mer tid må brukes for implementasjon. Da må vi enten velge å bruke mer tid eller å forkaste funksjonen og jobbe med annet som er viktigere.

Systemet består av mange elementer og det kan være for mye arbeid som må gjøres innen tidsfristen. Mye tid må dedikeres til prosjektet dersom vi skal oppnå alle målene.

Backend funksjonalitet er prioritert høyere enn frontend og har vi færre arbeidsoppgaver. Frontend er helt avhengig av servere og er ubruklig uten.

Risikomatrisen er vedlagt som vedlegg 1.

4.6 **Framdriftsplan – styring av prosjektet**

4.6.1 Hovedplan

Vi bruker scrum som smidig arbeidsmetodikk prosessrammeverk som går ut på å jobbe i iterasjoner hvor vi hele tiden vil tilpasse oss. Dermed er det litt vanskelig å spå lengre enn noen uker/måned i forkant. Vi kommer til å ha kontinuerlig planlegging og tilpasning av prosjektet. Som det er sagt så har vi laget et gantt diagram som viser et utgangspunkt for hvordan vi kan jobbe med de forskjellige delene av applikasjonen gjennom dette semesteret. En PDF-utskrift ligger vedlagt.

4.6.2 Styringshjelpe midler

Confluence	Vi bruker Confluence for å dokumentere arbeidsprosessen og holde styring av dokumenter/referat/retrospektiv, møtenotater, kravspesifikasjoner og andre typer dokumenter.
Jira	For å ha oversikt over arbeidsprosessen og issue-tracking kommer vi til å bruke Jira. Dette verktøyet er tett knyttet til Confluence og gjør det lettere å jobbe i sprints samt ha en oversiktlig backlog med arbeid som må gjøres.

4.6.3 Utviklingshjelpe midler

GitLab	GitLab er en web basert plattform som forsørger hosting av programvare med Git til versjon styring.
Postman	Postman er en samarbeids plattform for testing og utvikling av API. Her kan vi dele collections av rest kall som blir brukt med tjenestene våre.
IntelliJ	For å utvikle og kjøre Java applikasjoner og programmer så bruker vi IntelliJ.
Visual Code	Er en code editor optimalisert for web og cloud applikasjoner
GitKraken / Sourcetree / Git CLI	Vi kommer til å bruke Git klienter som GitKraken, Sourcetree og Git CLI for å versjonsstyre prosjektene våre.

4.6.4 Intern kontroll – evaluering

Oppfølging av framdrift vil bli gjort annenhver uke med veileder og etter behov med mentor (Må avtales). Her vil vi gå gjennom hva som har blitt gjort, hva som er blitt endret og om det har oppstått noen problemer. Vi vil også holde et internt retrospektivt møte etter hver endt sprint.

Det som kjennetegner at et mål er blitt nådd er når vi har gjort alle oppgavene relatert til implementasjonen, at både veileder og mentor er enig med implementasjonen og at den går igjennom både automatiske tester og bruker tester uten problem.

4.7 Beslutninger – beslutningsprosess

Vi har som mål å fullføre hovedsynet for prosjektet allerede nå i forprosjekts arbeidet slik at vi forhåpentligvis klarer å unngå store endringer i oppgaven senere. Endring av oppgavetekst og applikasjon tar lang tid og vi prøver dermed å ta alle slike avgjørelser nå i starten.

Dersom det oppstår en situasjon hvor vi må ta en viktig beslutning så kommer vi til å samle hele gruppen, samt veileder og mentor (dersom det er nødvendig). Så diskuterer vi frem til en felles beslutning og deretter dokumenterer den endringen i Confluence (hvor vi holder på alt av dokumentasjon).

5 DOKUMENTASJON

5.1 Rapporter og tekniske dokumenter

Alt som er dokumentasjons relatert, vil bli lagret i vårt Confluence område, som er ryddig sortert i mappestruktur som gjør det lett å navigere etter informasjon.

Veileder/mentor møtereferat	For hvert møte med veileder og/eller mentor skal det skrives et møtereferat som oppsummerer målene med møtet, hva som ble gjennomgått og hva som ble konkludert.
Loggrapport	Vi kan hente ut loggrapport fra Jira etter hver sprint som dokumenterer hva vi har gjort, hvor lang tid det tok og hvem som har gjort hva.
Ukentlig rapport	Hver uke skal det skrives en ukes rapport som oppsummerer hva som er blitt gjort slik at både gruppemedlemmer, veiledere og mentor er oppdatert.
Rutiner	I de tilfellene det er nødvendig å ha rutine så skal det dokumenteres i felleskap slik at alle gjør det samme uten uoverensstemmelser.
Beslutninger	For hver viktig beslutning som er med på å modifisere oppgaven skal den beslutningen dokumenteres med: hva er grunnen, hvordan løser vi det og eventuelt konklusjon
Kravspesifikasjon	Kravspesifikasjonen skal definere hvilke brukerfunksjoner og generelle krav systemet skal ha. Vi bruker "user stories" til å definere brukerfunksjoner
UML dokument	Dersom det er behov for UML diagrammer og wireframes skal dette også lagres og beskrives i vårt Confluence område.
Retrospektivt møtereferat	I slutten av hver sprint skal det holdet et retrospektivt møte hvor vi internt går gjennom sprinten som var, hva som er gjort, hva kan endres, hvordan var arbeidsprosessen, hva skal vi gjøre videre og i slutten skrive et møtereferat.
Milepæls rapport	Hver gang vi oppnår en milepål skal vi skrive en egen rapport som ramser opp den funksjonaliteten som er tilgjengelig og en slags "release" oppsummering

6 PLANLAGTE MØTER OG RAPPORTER

6.1 *Møter*

6.1.1 Prosjektmøter

Vi har planlagt faste møter annenhver uke med veiledere og et møte i måneden med Avento (dersom de har mulighet) hvor vi vil gå gjennom sprinten som var (hva som er gjort, endret), og hva vi skal gjøre videre.

Kort fortalt er hensikten å gi alle inkludert en oversikt over tilstanden til prosjektet. Hva som er blitt gjort, hva som er endret, hva kunne gjøres bedre, hvordan vi skal gå videre.

7 PLANLAGT AVVIKSBEHANDLING

Ut ifra størrelsen/risikograden for avviket har vi flere valg å ta:

- Dersom en kommer over avviket og har mulighet til å fikse det uten å konsultere de andre kan man gjøre en oppdatering som fikser avviket.
- Dersom avviket er stort nok til at vi må gjøre endringer i selve oppgavebeskrivelsen, og utviklingsprosessen må vi først identifisere hva eller hvor avviket ligger og hva det påvirker. Så må vi konsultere hverandre å finne den beste løsningen. Dersom vi klarer å komme frem til en løsning vi er sikre på vi kan implementere vil vi gjøre de nødvendige endringene for å få det fikset. Etter dette vil vi føre inn denne endringen inn i en ukentlig rapport som oppdaterer både mentor og veileder på det som skjedde. Dersom vi føler at vi ikke klarer å finne en god løsning på problemet vil vi ta kontakt med veileder og/eller mentor og prøve å komme frem til en løsning sammen. Når vi kommer frem til en løsning vil vi dokumentere det på samme vis som nevnt over.

Ansvaret vil bli delt ut ifra hvilke tiltak som blir tatt. Dersom en utvikler velger å fikse problemet, kjøre en oppdatering og jobbe videre, så vil han ta ansvar for avviket. Dersom problemet kommer frem i gruppen og gruppen jobber sammen for å finne en løsning, så vil gruppen i sin helhet ta ansvaret.

8 UTSTYRSBEHOV/FORUTSETNINGER FOR GJENNOMFØRING

Vi kommer til å bruke AutoDeploy for å administrere test servere i starten, men så gå over til noe litt mer reliable med backup, disaster recovery og redundancy senere. Her kan vi for eksempel bruke Azure.

VEDLEGG

Vedlegg 1 PDF-utskrift av Gantt diagrammet

Vedlegg 2 Excel fil som inneholder risikoanalyse

B API Specifications

The API specifications given below are of PDF format and only shows the overview of our API documentation. The actual interactable Open API specification with swagger for each service can be found at these addresses:

Authentication Server: <http://stage.autopacker.no:9000>

Server Manager: <http://stage.autopacker.no:9001>

General API: <http://stage.autopacker.no:9002>

File Delivery API: <http://stage.autopacker.no:9003>



Authentication Server

1.0.0-oas3 OAS3

Spring Boot API that delivers authentication server functionality to the users of AutoPacker.

Servers

`http://dev.libane.tk:8080``Authorize`

members

 Operations available to all our members. `POST /auth/login` Authenticate an existing user `GET /auth/resendVerificationToken` Resend a verification token`GET /auth/registrationConfirmation` Verifies the user if token is valid`GET /auth/resetPasswordRequest` Performs a request to reset a users' password`POST /auth/resetPasswordChange` Reset a users' password`POST /auth/changePassword` Change an authenticated users' password

guests

 Operations available to all our members, but also our guests. `POST /auth/register` Register a new user`GET /auth/usernameAvailability/{username}` Verifies if the username is available, used in registration`GET /auth/emailAvailability/{email}` Verifies if the email is available, used in registration

GET /auth/users

GET /auth/search Returns a list of all the users using AutoPacker that contains the search string. Used when searching for users.

Schemas



User



Role



Authority



Token





File ▾ Edit ▾ Insert ▾ Generate Server ▾ Generate Client ▾

Server Manager 1.0.0 OAS3

Spring Boot API that delivers server management functionality to the users of AutoPacker.

Servers

<http://dev.libane.tk:8081>

Authorize

members

Operations available to all our members. ▼

GET /server/deployProject/{username}/{serverId}/{ projectName}

POST /server/new-server Add a new server associated with the user making the request

GET /server/init/{serverId} Runs a initialization script on the given server

DELETE /server/delete/{serverId} Deletes the given server if the user authorized owns the server

GET /server/server-overview/{serverId}

GET /server

GET /server/{search}

POST /server/add-project

POST /server/remove-project

Schemas

Server



General API 1.0.0 OAS3

Spring Boot API that delivers some general functionality to the users of AutoPacker.

Servers

`http://dev.libane.tk:8082`

Authorize

admins Operations available only to admins.

POST `/languages/new` Add a new supported language to the application

organization admins Operations available only to organization admins

POST `/organization/acceptMemberRequest` Accepts a given applicant to become a member of the organization

POST `/organization/declineMemberRequest` Declines a membership application to an organization

POST `/organization/acceptProjectRequest` Accepts a project request and adds that project to the organization

POST `/organization/declineProjectRequest` Declines a project request for an organization

GET `/organization/{organization}/member-applications` Returns all the member applications made to the organization

GET `/organization/{organization}/project-applications` Returns all the project requests made to the organization

GET `/organization/{organization}/delete-project-applications/{projectId}` Deletes a specified project request

organization members Operations available to all our organization admins, but also our members.

POST `/organization/requestMembership` Request a membership with an organization

POST `/organization/submitProject` Submit a project to a given organization

POST `/organization/updateProjectSubmission` Updates an existing project submission

GET `/organization/{organization}/members` Returns a list of all the members and their roles in an organization

GET `/organization/{username}/isMember` Returns all the organizations the user is part of

GET `/organization/{username}/isMember/search` Returns all the organizations the user is part of

GET `/organization/{organization}/projects` Returns all the projects of the organization

GET `/organization/{organization}/projects/search` Returns all the projects that are affiliated with the organization that match the search criteria

GET `/organization/{organization}/project-applications/{username}` Returns all the project requests that a user has made

members

Operations available to all of the members in our application, part of an organization or not.

GET `/languages/all`

guests

Operations available to all our members, but also our guests.

GET `/organization` Returns a list of all the public organizations

GET `/organization/{organization}` Returns information about an organization

Schemas

Member

MemberApplication

Organization



Project



ProjectApplication



Role



Authority



Language



Version



[/swagger.json](#)[Explore](#)

Project & Module

1.0.0-stage-oas3 OAS3

[/swagger.json](#)

This is a collection of endpoints for managing projects and modules.

Servers

<http://158.38.101.87:8090>[Authorize](#)

default



POST /projects Add Empty Project



GET /projects/all Get All Projects



GET /projects/{username} Get All Projects From User



GET /projects/{username}/{project} Get Single Project



DELETE /projects/{username}/{project} Delete Project



DELETE /projects/{username}/{project}/{module} Delete Module



POST /projects/{username}/{project}/{module}/add Add Module To Project



GET /projects/{username}/{project}/docker-compose.yml Get docker-compose for project



Schemas



Project

Module

config-params

docker-compose

Project create format

body

INVALID {…}

C Gantt diagram

D Web Application Source Code**E File Delivery API Source Code****F General API Source Code****G Server Manager Source Code****H Authentication Server Source Code****I Dynamic use of docker compose**

In this document, we describe the general use on how to create a dynamic docker-compose.yml file with a preset template and dynamic values. This is the general idea we built upon when creating our docker-compose builder in our backend services.

Dynamic use of docker-compose blocks

In this file, we explain the concept of how to run a common jar file as a standalone docker container. To achieve this we need this structure:

User-project/

```
.env | "Contains the values specifying the module specification"  
docker-compose.block.yml "The docker-compose block that has been picked out by an algorithm"  
Dockerfile "The dockerfile used to build a docker image out of the program the user want to upload"  
.jar | "The program in case the image hasn't been built by docker yet"
```

Now that we have the structure of the folder in which we're going to work in we can go through step by step how to make this .jar program run. (**example explains scenario of using .jar**)

Step 1: Build the image

When the user have uploaded a application as a single module we first need to find out (from the users criteria) a dockerfile that matches his/hers criteria. In our example the uploaded application is a .jar file using java-8, so our dockerfile that we hopefully picked out would look like this:

Dockerfile

```
FROM openjdk:8  
ARG JAR_FILE=.*.jar  
COPY ${JAR_FILE} app.jar  
ENTRYPOINT [ "java", "-jar", "/app.jar" ]
```

So our application/docker process builder (which builds the image automatically) then builds the docker image: (if of course we have the file structure above)

```
→ Girts Jar docker image build -t autopacker/project-name .  
Sending build context to Docker daemon 14.85kB  
Step 1/4 : FROM openjdk:8  
--> cdf26cc71b50  
Step 2/4 : ARG JARFILE=.*.jar  
--> Using cache  
--> 948001627947  
Step 3/4 : COPY ${JARFILE} app.jar  
--> Using cache  
--> b8dad95e9174  
Step 4/4 : ENTRYPOINT [ "java", "-jar", "/app.jar" ]  
--> Using cache  
--> a0cf7dea10d9  
Successfully built a0cf7dea10d9  
Successfully tagged autopacker/project-name:latest
```

Step 2: Finding the correct docker-compose block

When image building is completed we need to find out what type of docker-compose block we need for our project. This search and find process must be handled by a type of searching algorithm. In this case we don't need a advanced docker-compose setup. We only need a docker image and what port to use for port-forwarding. So in our scenario a simple docker-compose block is enough:

docker-compose.simple.yml

```
# docker-compose block for a simple setup that only needs an image and port
version: "3"

services:
  client_app:
    container_name: ${CLIENT_APP}
    image: autopacker/${CLIENT_APP}
    ports:
      - ${PORT}:${PORT}
```

Step 3: Using the correct values with the docker-compose block

As you can see in the code block our docker-compose block needs variables to function. That's where the .env file comes in to place. The .env file can store n number(s) of environment variables that can be "echoed" into the file and used in the docker-compose blocks. So for our scenario our .env file would have looked like this:

.env

```
CLIENT_APP=project_name
PORT=3000
```

(When we get to the point we want to run this compose file it will automatically use the .env file so we don't have to specify it.)

Step 4: Validate the configuration

Now we want to validate that our docker-compose block is correct. We can achieve this by running:

validate

```
docker-compose -f docker-compose.simple.yml config
```

So in our case we would have gotten this result:

```
→ compose-blocks git:(feature/add-compose-block-for-java) ✘ docker-compose -f docker-compose.java8.yml config
services:
  jar_app_8:
    container_name: test
    image: autopacker/test
    ports:
      - 3000:3000/tcp
version: '3.0'
```

You'll get an error if something goes wrong.

Step 5: Run the application

When everything above is correct we can finally execute the given application with:

```
→ compose-blocks git:(feature/add-compose-block-for-java) ✘ docker-compose -f docker-compose.simple.yml up
Creating network "compose-blocks_default" with the default driver
Creating test ... done
Attaching to test
test      | Chat server started on port 1300
```

The End.

J Wireframes



Figure 1: Figure showing the wireframe for the homepage.

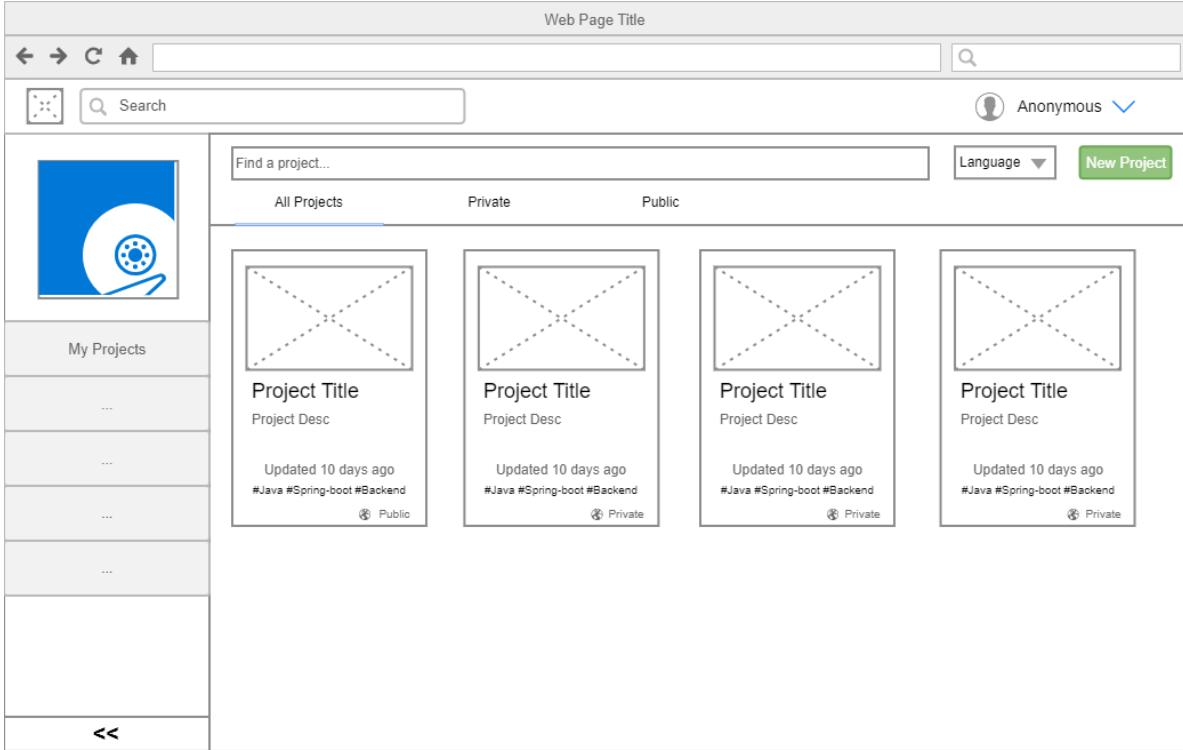


Figure 2: Figure showing the wireframe for the dashboard panel containing user projects.

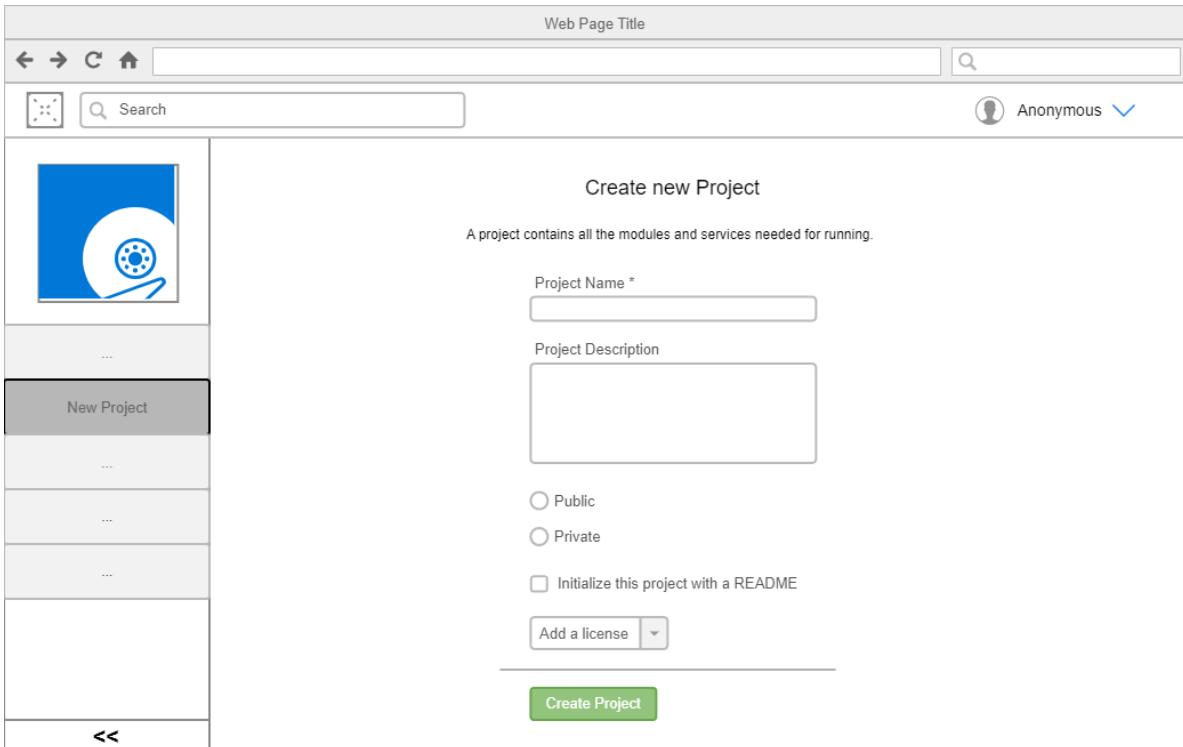


Figure 3: Figure showing the wireframe for creating a new project.

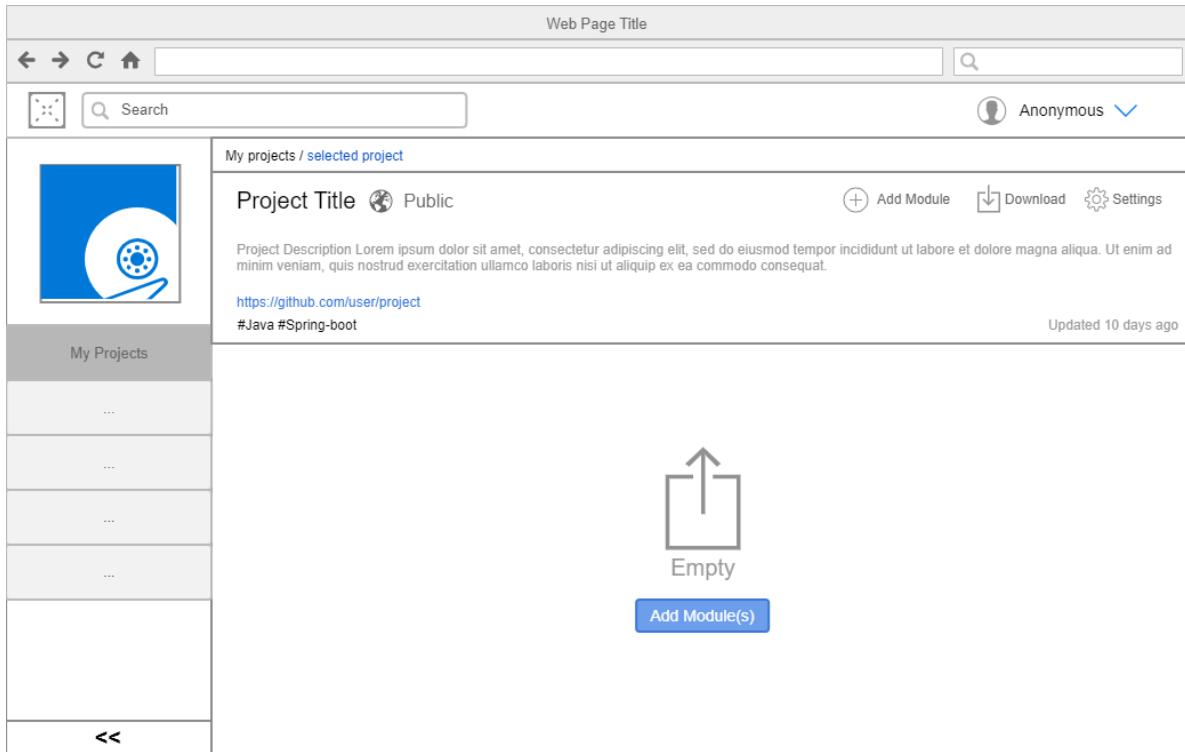


Figure 4: Figure showing the wireframe for an empty project.

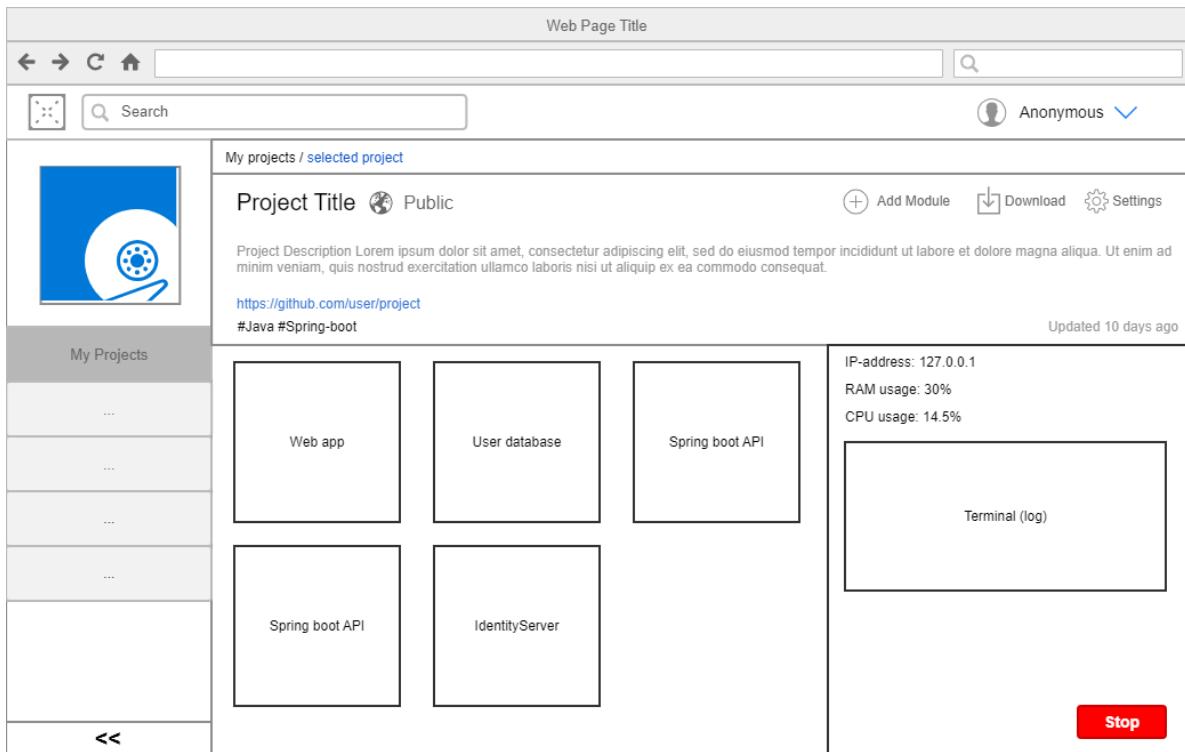


Figure 5: Figure showing the wireframe for a project containing modules.

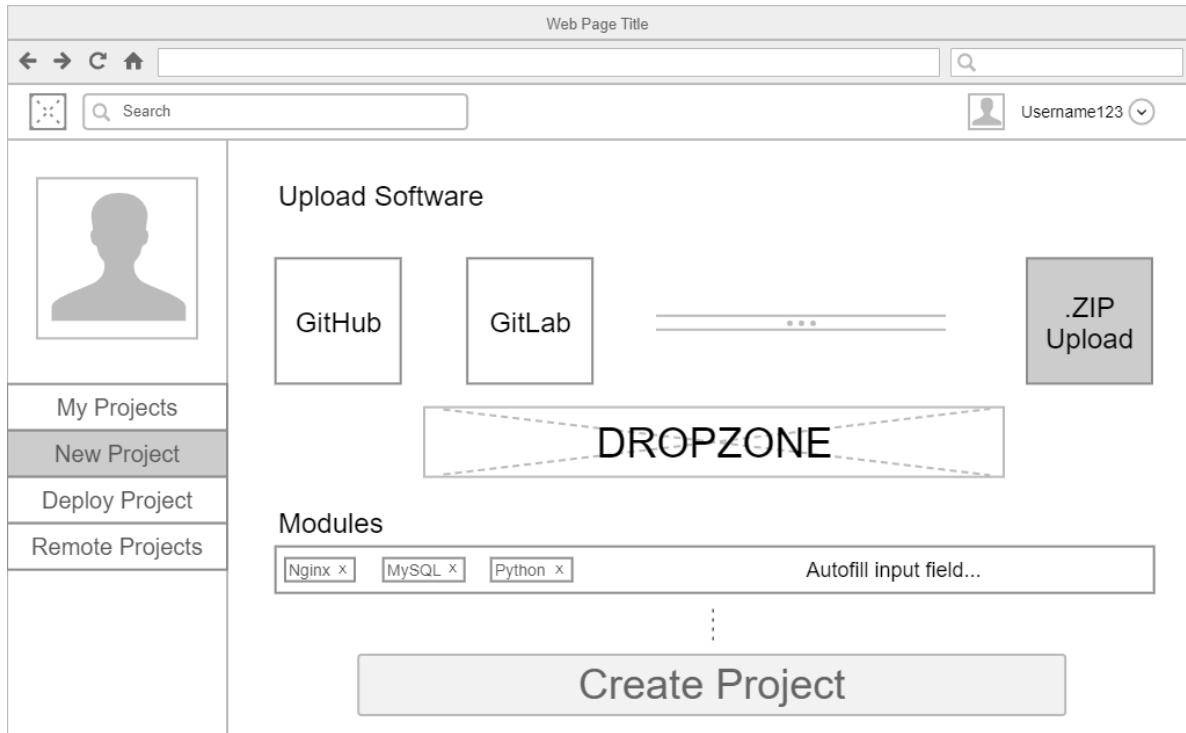


Figure 6: Figure showing the wireframe for creating a new module for a project.

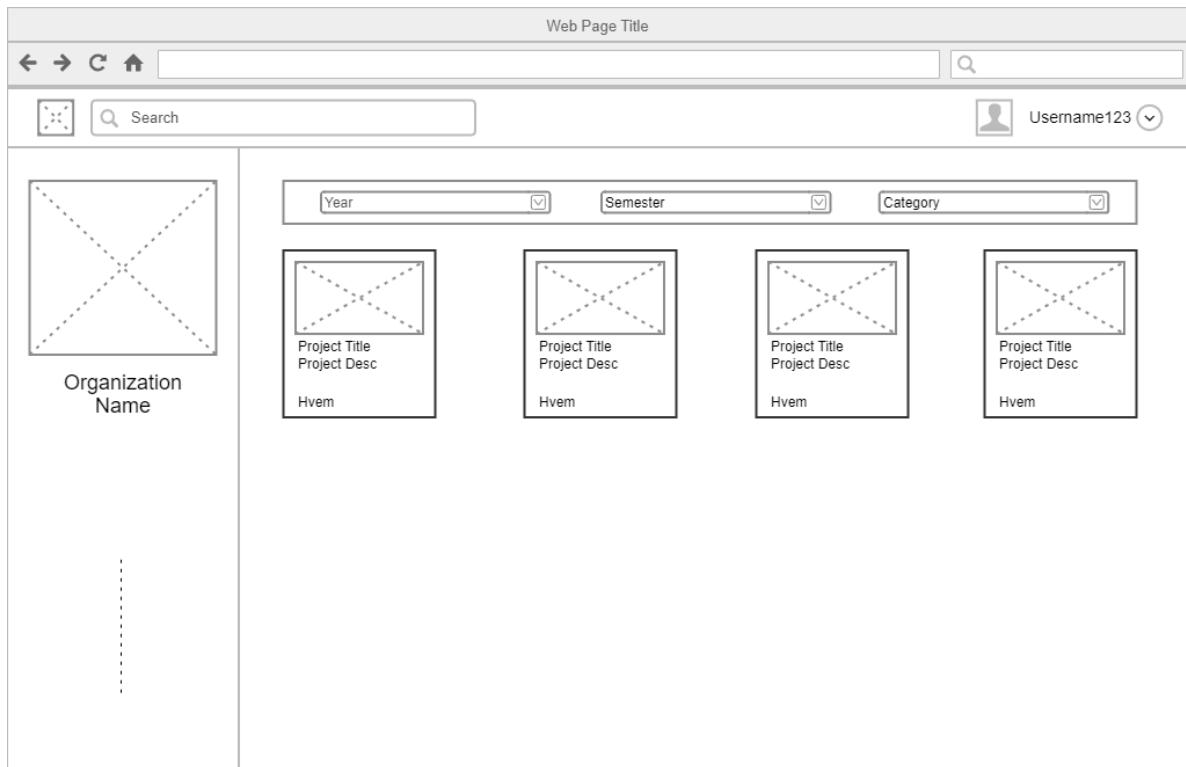


Figure 7: Figure showing the wireframe for listing all projects in an organization.

K Jira Sprint Reports

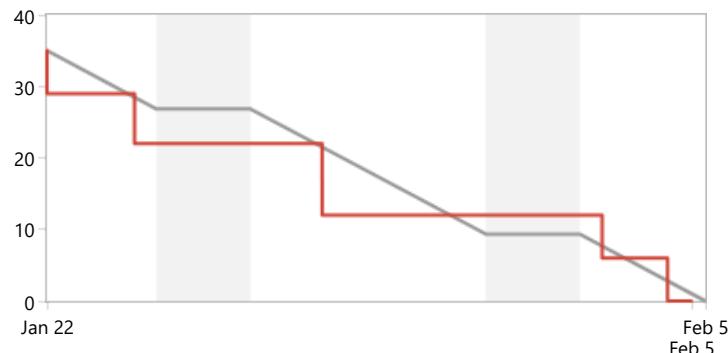
Main

Sprint Report [Switch report ▾](#)

Main Sprint 1

Closed Sprint, started by Aron Nicholasson, ended by Aron Nicholasson 22/Jan/20 4:02 PM - 05/Feb/20 9:33 AM

[View linked pages](#)



Status Report

Completed Issues

[View in Issue navigator](#)

Key	Summary	Issue Type	Priority	Status	Story Points (35)
AUT-12	Finish pre-project report	<input checked="" type="checkbox"/> Task	Critical	DONE	-
AUT-13	Finish Gantt diagram (version 1)	<input checked="" type="checkbox"/> Task	Critical	DONE	-
AUT-16	As a developer I want to authenticate myself so I can start implementing security in other parts of the system	Story	Critical	DONE	7
AUT-17	Create contribution guide	<input checked="" type="checkbox"/> Task	Major	DONE	-
AUT-18	Update use-case diagram to meet new minimum-value product specifications	<input checked="" type="checkbox"/> Task	Critical	DONE	-
AUT-20	Create weekly report for week 4	<input checked="" type="checkbox"/> Task	Critical	DONE	-
AUT-21	Create weekly report for week 5	<input checked="" type="checkbox"/> Task	Critical	DONE	-
AUT-22	Create developer guidelines for formatting and codestyle	<input checked="" type="checkbox"/> Task	Critical	DONE	-
AUT-24	As a user I want to be able to contact support or technical when i have questions about or wishes for the application	Story	Major	DONE	6
AUT-26	As a developer, I want users to be able upload any files so it can be downloaded in the future	Story	Critical	DONE	6
AUT-28	As a developer, I want to be able to download files so it can be used later to download docker/script files	Story	Critical	DONE	5
AUT-29	As a developer, I want users to have a workspace so they can store all their projects and modules	Story	Critical	DONE	5
AUT-30	As a developer, I want to view the file meta data of each	Story	Critical	DONE	3

user workspace

AUT-32	As a developer, I want to delete any files on the server so it's no longer available	 Story	 Critical	DONE	3
AUT-35	Create presentation for IF300114 Ingeniørfaglig systemteknikk og systemutvikling	 Task	 Major	DONE	-

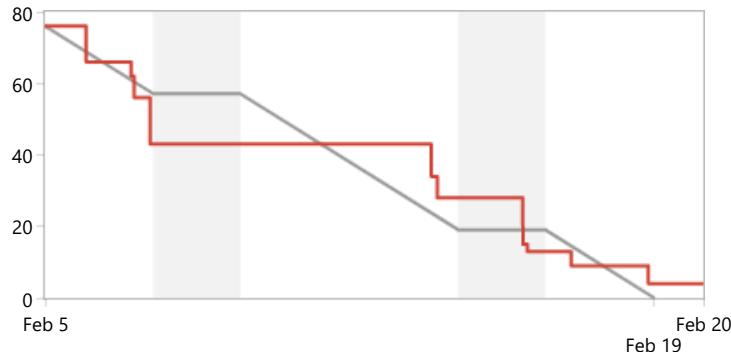
Main

Sprint Report [Switch report ▾](#)

Main Sprint 2

Closed Sprint, started by Aron Nicholasson, ended by Aron Nicholasson 05/Feb/20 12:17 PM - 20/Feb/20 3:11 PM

[View linked pages](#)



Status Report

Completed Issues

[View in Issue navigator](#)

Key	Summary	Issue Type	Priority	Status	Story Points (72)
AUT-14	As a developer I want to store file system references in a database because it makes it easier to find files	Story	Critical	DONE	5
AUT-23	As a guest I want to be welcomed in a homepage so I can read information about the service	Story	Major	DONE	8
AUT-34	As a developer I want a resource template with authorization which i can use to create new resource services	Story	Major	DONE	5
AUT-37	As a user, I don't want others to be able to modify/delete my projects I've made because I've made it	Story	Critical	DONE	4
AUT-38	As a user, I want to decide if I want to make my project(s) private so only I have private access to it	Story	Critical	DONE	3
AUT-39	As a user, I can create modules in my project(s) so I can extend my system	Story	Critical	DONE	5
AUT-40	As a user, I can prove that I am the legitimate owner by clicking on a verification link	Story	Minor	DONE	5
AUT-41	As a user, I want to receive a new verification link when the other has expired	Story	Minor	DONE	4
AUT-42	As a user I want to be able to reset my password when forgotten	Story	Major	DONE	6
AUT-43	As a user I want to be able to change my password when needed	Story	Minor	DONE	4

AUT-46	As a developer, I want a weekly report for week 6 that specifies what have been done	 Story	 Major	DONE	1
AUT-47	As a developer, I want a weekly report for week 7 that specifies what have been done	 Story	 Major	DONE	1
AUT-48	As a user, I want to be able to specify server credentials for the API to connect to	 Story	 Major	DONE	5
AUT-49	As a user, I want to see a profile dashboard when I have authenticated myself	 Story	 Major	DONE	6
AUT-50	As a guest, I want to be able to register an account so I can use more of the system	 Story	 Critical	DONE	5
AUT-53	As a user, I can authorize myself in the service	 Story	 Major	DONE	5

Issues Not Completed

[View in Issue navigator](#)

Key	Summary	Issue Type	Priority	Status	Story Points (4)
AUT-52	As a student, I want to sketch out a material and methods section (list of used technologies/theoretical concepts)	 Story	 Critical		4

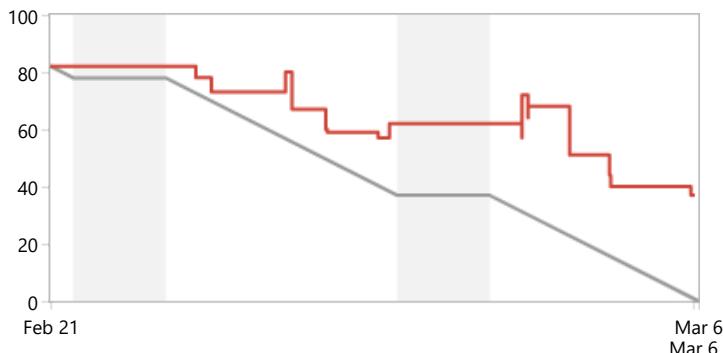
Main

Sprint Report [Switch report ▾](#)

Main Sprint 3

Closed Sprint, started by Aron Nicholasson, ended by Aron Nicholasson 21/Feb/20 12:07 PM - 06/Mar/20 10:12 AM

[View linked pages](#)



Status Report

* Issue added to sprint after start time

Completed Issues

[View in Issue navigator](#)

Key	Summary	Issue Type	Priority	Status	Story Points (76)
AUT-15	As a user I can view my projects, so I know which projects I have	Story	● Blocker	DONE	6
AUT-52	As a student, I want to sketch out a material and methods section (list of used technologies/theoretical concepts)	Story	● Critical	DONE	4
AUT-55	As a developer, I want the server manager to execute configuration scripts on given server	Story	● Critical	DONE	5
AUT-56	As a developer, I want wireframes for the profile-dashboard	Story	● Major	DONE	7
AUT-58	As a user, I want to be able to create an empty "project" that contains meta-data	Story	● Blocker	DONE	4
AUT-59	As a developer, I want a weekly report for week 8 that specifies what have been done	Story	● Major	DONE	1
AUT-64	As a user, I want to be able to create and upload a simple java project	Story	● Critical	DONE	3
AUT-68	As a user, I want to be able to specify which java version the project uses	Story	● Critical	DONE	2
AUT-73	As a developer, I want the API to be managed by a DevOps pipeline	Story	● Minor	DONE	4
AUT-75	As a user, I want to be able to upload "modules" to a	Story	● Critical	DONE	9

specified project

AUT-78 *	As a user, I want to get an overview of the project i select	Story	Blocker	DONE	7
AUT-79 *	Cant forward properties to a functional component which is a child of a child	Bug	Blocker	DONE	-
AUT-80 *	As a developer, I want the pipeline to be more robust, effective and less resource consuming	Story	Major	DONE	5
AUT-81 *	As a user, I want to see a list of all my managed servers	Story	Major	DONE	4
AUT-82 *	As a user, I want to see server specific information when i click on it	Story	Major	DONE	4
AUT-85 *	As a developer, I want to store information about user defined servers	Story	Major	DONE	7
AUT-86 *	As a user, I want to be able to add a server to my "account"	Story	Major	DONE	4
AUT-87 *	Change port from 8080 to 8081	Bug	Major	DONE	-

Issues Not Completed

[View in Issue navigator](#)

Key	Summary	Issue Type	Priority	Status	Story Points (37)
AUT-54	As a developer, I can provision server with the necessary tools and configurations	Story	Critical	BACKLOG	4
AUT-57	As a user, I want to be presented with a modern informative homepage that is userfriendly	Story	Major	BACKLOG	9
AUT-60	As a developer, I want a weekly report for week 9 that specifies what have been done	Story	Major		1
AUT-61	As a student, I want to sketch out chapter 4: results in the bachelor thesis	Story	Major	BACKLOG	4
AUT-65	As a developer, I want a docker-compose template for a project with one api and one database	Story	Major	BACKLOG	3
AUT-66	As a developer, I want a docker-compose template for database	Story	Major	BACKLOG	4
AUT-69	As a user, I want to be able to specify which mysql version to use	Story	Major	BACKLOG	2
AUT-70	As a user, I want to be able to specify which postgresql version to use	Story	Major	BACKLOG	2
AUT-71	As a developer, I want to be able to store which docker images(s) and versions the application supports	Story	Critical		4
AUT-72	As a user, I want to be able to see what tools and versions that are available	Story	Critical		4

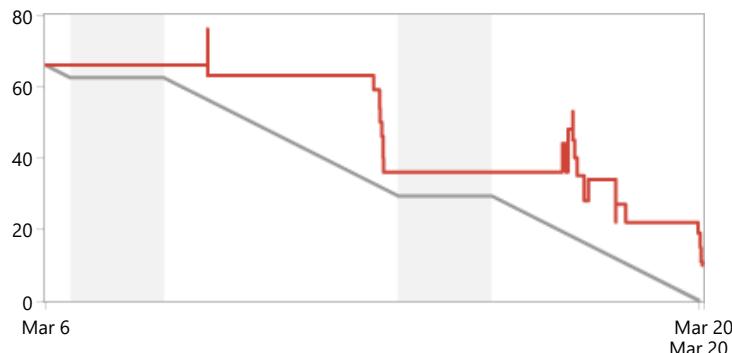
Main

Sprint Report [Switch report ▾](#)

Main Sprint 4

Closed Sprint, started by Aron Nicholasson, ended by Aron Nicholasson 06/Mar/20 10:52 AM - 20/Mar/20 12:40 PM

[View linked pages](#)



Status Report

* Issue added to sprint after start time

Completed Issues

[View in Issue navigator](#)

Key	Summary	Issue Type	Priority	Status	Story Points (98)
AUT-60	As a developer, I want a weekly report for week 9 that specifies what have been done	Story	Major	DONE	1
AUT-61	As a student, I want to sketch out chapter 4: results in the bachelor thesis	Story	Major	DONE	4
AUT-71	As a developer, I want to be able to store which docker images(s) and versions the application supports	Story	Critical	DONE	4
AUT-72	As a user, I want to be able to see what tools and versions that are available	Story	Critical	DONE	4
AUT-83	As a user, I want to be able to add project to server	Story	Major	DONE	4
AUT-84	As a user, I want a deploy option so I can deploy a project assigned to a server	Story	Major	DONE	6
AUT-89	As a user, I want feedback that describes if server is successfully created and configured or not	Story	Major	DONE	3
AUT-90	As a user, I want to add a module to an existing project	Story	Major	DONE	5
AUT-91	As a user, I want to be able to create a new project	Story	Major	DONE	4
AUT-92	As a developer, I want a weekly report for week 10 that specifies what have been done	Story	Major	DONE	1
AUT-93	As a developer, I want a weekly report for week 11 that specifies what have been done	Story	Major	DONE	1

AUT-94	Create an API documentation and add JavaDoc to each class and methods needed	 Story	 Major	DONE	4
AUT-96	Add documentation to each component and methods needed	 Story	 Major	DONE	6
AUT-97	Create an API documentation and add JavaDoc to each class and methods needed	 Story	 Major	DONE	5
AUT-98	Create an API documentation and add JavaDoc to each class and methods needed	 Story	 Major	DONE	4
AUT-100 *	As a developer, I want the API to receive more information about projects and modules	 Story	 Major	DONE	6
AUT-102 *	As a developer, I want to store dockerfile location reference in database for quick finding	 Story	 Major	DONE	4
AUT-103 *	As a developer, I want to store docker-compose blocks location reference in database for quick finding	 Story	 Major	DONE	4
AUT-104 *	As a developer, I want a isVerified field in user so i can see if user is verified or not	 Story	 Major	DONE	5
AUT-105 *	As a user, I want to be able to see system before verifying myself	 Story	 Major	DONE	7
AUT-106 *	As a developer, I want the database to automatically be imported when testing locally and with Docker	 Story	 Major	DONE	5
AUT-107 *	As a user, I want feedback on input when signing up	 Story	 Major	DONE	6
AUT-108 *	As a developer, I want the system to be managed by a gitlab-ci pipeline	 Story	 Major	DONE	5
AUT-109 *	The API tries to run select queries (from DBInit) before the database has been created in the main class	 Bug	 Major	DONE	-

Issues Not Completed

View in Issue navigator

Key	Summary	Issue Type	Priority	Status	Story Points (10)
AUT-88	As a user, I want a newly "connected" server to be provisioned on creation	 Story	 Blocker	IN PROGRESS	6
AUT-95	Create an API documentation	 Story	 Major	IN PROGRESS	4
AUT-101 *	Invalid Cookie	 Bug	 Trivial	IN PROGRESS	-

Issues Removed From Sprint

View in Issue navigator

Key	Summary	Issue Type	Priority	Status	Story Points (4)
AUT-99 *	As a developer, I want the File Delivery API to be run on Windows and UNIX systems without needing system specific implementations	 Story	 Major	DONE	4

Main

Sprint Report [Switch report ▾](#)

Main Sprint 5

Closed Sprint, started by Aron Nicholasson, ended by Aron Nicholasson 20/Mar/20 1:56 PM - 03/Apr/20 12:02 PM

[View linked pages](#)



Status Report

* Issue added to sprint after start time

Completed Issues

[View in Issue navigator](#)

Key	Summary	Issue Type	Priority	Status	Story Points (154 → 165)
AUT-19 *	As a user I can view settings section, so I can know which settings are active for my profile	Story	Critical	DONE	- → 7
AUT-54	As a developer, I can provision server with the necessary tools and configurations	Story	Critical	DONE	8
AUT-88	As a user, I want a newly "connected" server to be provisioned on creation	Story	Blocker	DONE	6
AUT-110	As an admin, I want to be able to upload templates for dockerfile and docker-compose	Story	Blocker	DONE	5
AUT-111	As a developer, I want to be able to view another users' profile	Story	Major	DONE	8
AUT-112	As a developer, I want an endpoint that can generate a simple docker-compose file with my variables	Story	Blocker	DONE	7
AUT-116	As a user, I want generated docker-compose files to be stored for later downloads	Story	Major	DONE	4
AUT-117	As a user, I can search for public projects	Story	Major	DONE	4
AUT-118	Logout functionality not working	Bug	Blocker	DONE	-
AUT-119	As a user, I can logout of the application	Story	Blocker	DONE	5
AUT-120	As a developer, I want user project module to be built into a docker image using a dockerfile	Story	Blocker	DONE	4

and upload it to docker hub

AUT-121	As a user, I can delete one of my owned projects	 Story	 Major	DONE	5
AUT-127	As a user, I can view another users' projects on his/hers profile	 Story	 Major	DONE	6
AUT-128	As a user, I can filter search result by users	 Story	 Major	DONE	6
AUT-129	As a user, I can filter search results by project name	 Story	 Major	DONE	6
AUT-130	As a user, I can filter search results by pagination	 Story	 Major	DONE	7 → 5
AUT-132 *	As a developer, I want to upload more than 1 file as a module	 Story	 Major	DONE	- → 6
AUT-133 *	As a developer, I want split up the controller class for project and module into two	 Story	 Minor	DONE	2
AUT-134 *	As a user, I want to view an organization page	 Story	 Major	DONE	6
AUT-135 *	As a user and admin, I want to be able to see all members in an organization	 Story	 Major	DONE	4
AUT-136 *	As a user, I want to see all the projects that are associated with an organization	 Story	 Major	DONE	4
AUT-137 *	As a user, I want to make a request of becoming a member of a given organization	 Story	 Major	DONE	6
AUT-138 *	As a organization member, I want to be able to submit a project to the organization	 Story	 Major	DONE	6
AUT-139 *	As a organization member, I want to be able to see all of my submissions	 Story	 Major	DONE	5
AUT-140 *	As a organization admin, I want to be able to see all applicants	 Story	 Major	DONE	6
AUT-141 *	As a organization admin, I want to be able to see all project submissions	 Story	 Major	DONE	9
AUT-142 *	As a developer, I want the API to support functionality for organizations	 Story	 Major	DONE	10
AUT-143 *	As a developer, I want the API to contain a organization database	 Story	 Major	DONE	7
AUT-144 *	As a developer, I want the application to handle global organizational state	 Story	 Major	DONE	8

Issues Not Completed

View in Issue navigator

Key	Summary	Issue Type	Priority	Status	Story Points (47 → 51)
AUT-57	As a user, I want to be presented with a modern informative homepage that is userfriendly	 Story	 Major	BACKLOG	10

AUT-95	Create an API documentation	 Story	 Major	IN PROGRESS	4 → 8
AUT-101	Invalid Cookie	 Bug	 Trivial	IN PROGRESS	-
AUT-113	As a developer, I want docker-compose variables from users to be saved in mongoDB	 Story	 Critical	BACKLOG	7
AUT-131	As a developer, I want the server manager to connect to remote server, fetch a docker-compose file and run docker-compose up	 Story	 Blocker	IN PROGRESS	8
AUT-145 *	Create a Confluence document describing how upload, building and download of projects happens	 Story	 Critical	BACKLOG	6
AUT-146 *	As a developer, I want to store references of dockerfile and docker-compose in database	 Story	 Major	BACKLOG	4
AUT-147 *	As a developer, I want to use autopacker.no email for DockerHub on all APIs	 Story	 Major	BACKLOG	8

Main

Sprint Report [Switch report ▾](#)

Main Sprint 6

Closed Sprint, started by Bendik Nogva, ended by Aron Nicholasson 03/Apr/20 1:24 PM - 17/Apr/20 4:41 PM

[View linked pages](#)



Status Report

* Issue added to sprint after start time

Completed Issues

[View in Issue navigator](#)

Key	Summary	Issue Type	Priority	Status	Story Points (103)
AUT-65	As a developer, I want a docker-compose template for a project with one api and one database	Story	Major	DONE	5
AUT-66	As a developer, I want a docker-compose template for database	Story	Major	DONE	4
AUT-69	As a user, I want to be able to specify which mysql version to use	Story	Major	DONE	2
AUT-70	As a user, I want to be able to specify which postgresql version to use	Story	Major	DONE	2
AUT-95	Create an API documentation	Story	Major	DONE	8
AUT-113	As a developer, I want docker-compose variables from users to be saved in mongoDB	Story	Critical	DONE	7
AUT-114	As a user, I can view a project settings section, so I can modify my project	Story	Major	DONE	5
AUT-122	As a user, I can delete one of my servers	Story	Major	DONE	5
AUT-131	As a developer, I want the server manager to connect to remote server, fetch a docker-compose file and run docker-compose up	Story	Blocker	DONE	8
AUT-146	As a developer, I want to store references of dockerfile and docker-compose in database	Story	Major	DONE	4
AUT-147	As a developer, I want to use autopacker.no email for	Story	Major	DONE	8

DockerHub on all APIs

AUT-148	As a user, I want to be able to specify database related information when uploading a database module	 Story	 Major	DONE	8
AUT-149	As a user, I want the option to use a multi module upload	 Story	 Major	DONE	10
AUT-150	As a user, I want to be able to upload a finished configured system	 Story	 Major	DONE	7
AUT-151	Issue when uploading modules to Windows type file-systems (No write permissions)	 Bug	 Blocker	DONE	-
AUT-152	As a user, I want to see the side menu changes when I click on menu options	 Story	 Major	DONE	6
AUT-153	As a developer, I want the system to be on the azure cloud instead of autopacker (more reliable)	 Story	 Major	DONE	8
AUT-154	As a developer, I want a google style configuration XML file for intelliJ	 Story	 Major	DONE	6
AUT-155 *	Authorization happens unintentionally on all endpoints despite some endpoints not requiring it at all	 Bug	 Critical	DONE	-

Issues Not Completed

[View in Issue navigator](#)

Key	Summary	Issue Type	Priority	Status	Story Points (6)
AUT-145	Create a Confluence document describing how upload, building and download of projects happens	 Story	 Critical	IN PROGRESS	6

L Retrospective Meeting Notes

Sprint 3

Dato	06.mar.2020
Deltakere	Aron Nicholasson Bendik Nogva Liban Nor

Tilbakeblikk

Hva fikk vi bra til?

- Mye produksjon av koding
- God kommunikasjon
- Produsert mye GUI
- Mye gjort klart for integrasjon
- Automatisert flere repositories (CD)
- Lært mye docker relatert

Hva burde vi ha gjort bedre?

- Planlagt sprinten bedre
- Sett for oss hvordan det faktisk ville fungere å laste opp en .jar til å ha den kjørende som en docker container på en valgt server
- Ha et mer teknisk grunnlag før vi starter på sprinten
- Dokumentert bedre (bachelor og in-code document)

Sprint 4

Dato	20.mar.2020
Deltakere	Liban Nor Aron Nicholasson Bendik Nogva

Tilbakeblikk

Hva fikk vi til bra?

- Vi fikk gjort unna mye selv om vi hadde eksamen i denne sprinten
- Har komt godt igang med koding og begynt på bachelor.
- Flink med manuelle tester
- Godt fungerende CI/CD system
- Gode commit navn
- God dokumentasjon

Hva burde vi ha gjort bedre?

- Kommunikasjon (burde legge faste tidspunkt)
- Brukt mer tid på å se hva som må til for å oppnå et minimum-value product

Retrospective - Sprint 5

Dato	03.apr.2020
Deltakere	Aron Nicholasson Liban Nor Bendik Nogva

Retrospectives

What did we do well?

- Produced a lot of new functionality
- Squashed many existing bugs
- Improved existing functionality
- Good documentation
- We worked well due to the fact that we are in the middle of a pandemic (Corona pandemic)
- Established great goals for the future of the project

What could have been done better?

- Estimated user points better
- Tried and keep the planned course better (We were supposed to complete upload of modules, but this hasn't been done, but we added a lot of other functionalities instead)
- Remember to have stand-up meetings?

Retrospective - Sprint 6

Dato	17.apr.2020
Deltakere	Aron Nicholasson Liban Nor Bendik Nogva

Tilbakeblikk

Hva fikk vi bra til?

- Vi har så og si klart å lage vårt minimum-value-product.
- Vi har fått til mye på kort tid.
- Vi har klart å kommunisert og fikset det som er nødvendig å fikse.
- Alle har vært flink å møte opp i møter og felles samtaler (gjelder egentlig alle sprinter).
- Vi fikk integrert hele systemet sammen, testet den og funnet ut hva som må fikses.
- Vi var gode til å definere hvor mange story points vi kom til å gjøre ferdig i sprinten.
- Vi har fikset mye og lagt til mye på kort tid.
- Overraskende stabil applikasjon. Det meste fungerer uten å kræsje.

Hva burde vi ha gjort bedre?

- Vi burde ha testet systemet mer før vi hadde møte med produkteier (16.04.2020).
- Hatt flere daily stand-up meetings gjennom internett (minst 1m avstand).
- Burde begynt å fokusert mer på selve bachelor rapporten.
- Er dårlig med automatiske tester (tar mye tid å gjennomføre manuelle tester).
- Det ble litt lite stand-up meetings.

M Meeting Notes

2020-01-10 Første møte

Dato

10.jan.2020 kl 08:30 - 09:30

Deltakere

- [Aron Nicholasson](#)
- [Liban Nor](#)
- [Bendik Nogva](#)
- [Girts Strazdins \(Veileder\)](#)

Mål

- Se på den egendefinerte oppgaven og se på forbedringer, punkter for å realisere og generelt gå gjennom bachelor relaterte punkter

Notater

Beskrive den painfulle versjonen for at en bruker skal kunne gjøre det som vi har tenkt (idag må man)

- Brukeren må selv skrive "docker run" i terminalen plassering av hvor docker image ligger

Fancy pancy versjon:

- Bruker laster ned en slags desktop client som gjør det mulig å starte, stoppe, installere, lettere og mer oversiktlig (JAR)

Punkter tatt opp på møte

- Ha en sentral server der alle docker containers kjøres. Dette kan være vanskeligere å gjøre pga. mindre kontroll og mye konfigurasjon som må til. (Alternativ (port forwarding))
- For hver instanse av et prosjekt en bruker ønsker å kjøre så vil det opprettes en AutoDeploy server ut ifra en template som kjører prosjektet brukeren ønsker. (ressurskrevende alternativ)
- Beste alternativet er at brukeren laster ned og kjører prosjekt lokalt på sin egen maskin. Dette kan gjøres på flere måter, men trenger god dokumentasjon uansett
- Kan bruke docker compose til å beskrive og konfigurere systemet. Når brukeren skal laste opp et program kan han enten inkludere en docker compose fil eller så kan han gå igjennom en wizard som hjelper han å sette opp en docker compose fil
- Ingen tilpasset søkefilter, men kan filtreres etter år og tags
- Notere problemer som brukere kan ha når brukeren konfigurerer/deployer applikasjonen sin
- Skrive en minimum value product beskrivelse sett fra brukerens perspektiv (Viktig)⚠
- Skrive en egen kravspesifikasjon for hver fase med user stories (Viktig å sende til Mentor og veiledere) (viktig å inkludere prioritet)⚠
- Alt unntatt forprosjektrapport skal være på engelsk
- Skisser primitiv UML diagrammer, eks: use case, deployment
- Siden vi kjører microservice architecture har vi en database per tjeneste som trenger
- EmailService: bruke dependency? eller ikke.
- Kjører alle test servere via autodeploy, product servere kjører vi kanskje gjennom en mer reliable tjeneste som for eksempel Azure
- Det er lettere å kjøre docker containers på en server vi konfigurerer og administrerer for brukeren da dette gir kjappere resultat og mindre anstrengelse fra brukeren sin side.
- Dersom vi har en microservice tjeneste som direkte må jobbe med docker kan den kjøres som en docker container
- En løsning er også å lage et java program (JAR) som kjører på brukerens lokale maskin og handler (og installerer) docker relaterte stuff som fjerner brukerens behov for å styre med det.
- Girts nevnte at webappen kan ha inkludert wizard slik at bruker definerer hvilke dependancies, ports, mapper.
- Bachelor programvarer blir lagret sentralt, selve oppgavene får tags, slik det blir enklere å filtrere og finne bacheloroppgaver
- Vi lagrer alle prosjekt som brukeren laster opp (tilgjengelig for alle)
- Hvem er kunden? -vi ta utgangspunkt at vi er kundene.
- Beskrive problem, hva vi skal automatisere.
- Sluttprodukt: det viktigste er at appen fungerer, ikke fokusere på fancy versjon.
- Trenger bruker for opplastninger, private/public.

Sprints:

- centralisert som produkt,
- iterasjon 0, hvor vanskelig er det å kjøre et produkt,
- hver sprint gir mer funksjonalitet
- første fase, droppe innlogging,
- vi har for mange features, men etter hver sprint skal vi implementere nye,

2020-01-16 Første møte med Avento

Date

16.jan.2020 kl: 08.30

Attendees

- [Aron Nicholasson](#)
- [Liban Nor](#)
- [Bendik Nogva](#)
- [Girts Strazdins](#)

Goals

- Introduksjon til selve bachelor oppgaven (hva, hvem, hvorfor)
- Oppsummering fra møte med Girts (10/01)
- Feedback på tankegang
- Er oppgaven realistisk? Er det et bruksområde her?
 - Eventuelle endringer som gjør det mer realistisk at dette er en god oppgave, at det finnes et bruksområde
- Bli enig om et minimum-value product
- Bli enig om et mer fullstendig produkt (syn)

Notater

(Skal endres)

- Se på identity server istedenfor egen auth server pga sikkerhetshull
- Bruk ikke så mye tid på det som ikke er nødvendig å ha med i produktet. Trengs det email service på starten?
- Man kan ha et lokalt register på systemet og gjøre et lookup mot skolens database for dobbeltsjekking av student
- Kan ha egen konfigurasjons-format for å definere hva slags elementer som skal være med i et prosjekt (yaml eller json)
- En mulig løsning er å bruke BitBucket Validator, og scripts for bygging og deploy av program.
- Predefinerte konfigurasjoner
- Vår applikasjon vil fungere slik som en slags appstore/windows store.
- Vi burde fokusere på å lage et produkt som er tilrettelagt for NTNU,
- Et pluss om hele systemet fungerer seamless med NTNU, ved bruk av FEIDE
- Dersom minimum kravet er å kunne laste opp/ned docker, kan man lage en desktop app som kontrollerer prosjektene
- Om man kan laste ned docker og skriptet, kan man selv laste ned begge og kjøre dem på egen server dersom man allerede kan om servere
- Mulig å kontakte de som laget autodeploy og be de lage en predefinert ubuntu image med det vi trenger. For å unngå brukeren logger inn med passord/brukernavn.
- Vi kan dele opp systemet i forskjellige moduler (f.eks. ha en modul for å laste opp til AutoDeploy) slik at neste bachelor folk kan videreutvikle den
- Transparency på prosjekt for trust (hvordan kan du stole på studenter?)
- Brukere kan spesifisere sine server detaljer slik at vår applikasjon kan konfigurer for dem
- Vi må tenke på sikkerhet av det som blir laster opp til webapp. og nedlastet iom. programvarer kan være malicious, derfor behøver vi en måte å verifisere at programvaren er OK.

2020-02-20 "Oppsummering" fra møte med mentor og girts (delt møte)

Dato

20.feb.2020 KI 14.00

Deltakere

- Aron Nicholasson
- Liban Nor
- Bendik Nogva
- Girts Strazdins

Mål

Tenker å gå gjennom de undernevnte punktene:

- Hva som er blitt gjort
 - Autentiserings server
 - Filsystem service
 - Server Manager
- Hva kan gjøres annerledes
 - Cookie service for håndtering av token (backend)
- Hvilke valg som er tatt
 - Spring boot for autentisering
- Teknologi brukt (videre valg)
 - Docker (docker-compose)
 - Spring-boot
 - Dropper Ansible, går for Java
- Hva vi tenker å fokusere på fremover
 - Web applikasjon
 - Integrasjon mellom webapp og APIer
 - Server konfigurasjon
 - Docker compose "blocks" builder
- Beskrive vårt konsept av docker-compose "module builder" (Forvente at prosjekt består av flere moduler)
- Input for UI design av systemet

Diskusjon

1. Kan ha åpen Authorization token uten problem, så lenge alt går gjennom HTTPS og ikke har XSS exploits
2. Kan starte med å ha offentlige images på Docker istedenfor privat på starten (lettare og mindre tidsbruk)
3. Saniter alle verdier til backend, aldri stol på informasjon fra frontend
4. KISS - Chris Brown

2020-02-05 3. møte med Girts

Dato

05.feb.2020 kl. 09.00 - 10.30

Deltakere

- [Aron Nicholasson](#)
- [Bendik Nogva](#)
- [Liban Nor](#)
- [Girts Strazdins](#)

Oppsummering

- Analyse delen av rapporten (hvorfor vi valgte den og den teknologien)
- Prøve å stripe bort alle dem fancy non-prioritized features og få noe som funker
- Story points på alt unntatt bugs (bugs har 0 poeng pga sprint rapport "hopp")
- Legge inn liste av den analyse delen under material og metode
- Se på word template av bachelor for å ta utgangspunkt i rekkefølge og dokumentstruktur

2020-03-19 BB Colab møte med veileder og mentor

Dato

19.mar.2020

Deltakere

- [Aron Nicholasson](#)
- [Liban Nor](#)
- [Bendik Nogva](#)
- [Girts Strazdins](#)

Mål for møte med begge parter

- Ta opp det som er blitt gjort fra forrige møte (Se på ukentlig rapport)
- Ta opp beslutningane og valgene som vi har tatt (MongoDB, mikroservice arkitektur, verifikasjons funksjonalitet osv.)
- Ta imot feedback

Mål for internt møte med veileder

- Gjennomgå situasjonen vi er ovenfor nå (starten av korona pandemien)
- Gå gjennom om han eventuelt har tanker om det målet vi satt men ikke rakk (gått to sprinter, og var forventet 1)
- Legge en plan for korsn me ska jobbe som gruppe (og veileder) fremmover
- Planlegge neste sprint (Blir vell å jobbe med det som me jobber med no)

Diskusjon

2020-04-03 BB Colab meeting with supervisor and mentor

Date

03.apr.2020

Participants

- [Aron Nicholasson](#)
- [Bendik Nogva](#)
- [Liban Nor](#)
- [Girts Strazdins](#)

Goals

- Establish a common project overview with supervisor and mentor for what has been done, what is missing and what goals/wishes we have for the project
- Gain feedback for the progress that has been done

Discussion

- Talked about how we can perform a type of "logout" functionality and JWT handling methods in general
- Talked about how the application has developed, what is missing, what is good. Does it give a good user experience
- We went through a possibility of using cross platform "path" handling for file delivery API

N Weekly Logs

Uke 2

Uke	2
Document Status	COMPLETED

- Vi har hatt vårt første møte med veileder (møtefererat her: [2020-01-10 Første møte](#))
- Vi har blitt enige om hvilke teknologier vi skal bruke for å utvikle applikasjonen vår
- Vi har blitt enige om hvilke verktøy vi skal bruke for å dokumentere arbeidsprosessen våres
- Vi har satt opp discord server (kommunikasjon), confluence space (dokumentering) og Jira (issue-tracking og loggføring). Vi har også laget GitLab gruppe og lagt til alle involverte i prosjektet.
- Vi har begynt såvidt på forprosjektrapporten, men begynner å få en anelse over hva som må gjøres
- Vi har også (sammens med veileder) fått oversikt hvordan vi kan dele prosjektet opp i tre faser. Fra et minstekrav til et "fancy" produkt
- Vi har også planlagt vårt første møte med mentor (Avento)

Uke 3

Uke	3
Document Status	COMPLETED

- Vi har i hovedsak jobbed med forprosjektsrapport og diverse dokumentasjon.
- Vi har laget ferdig et use-case diagram for et fase 1 minimum value prout.
- Vi Har gjennomført vårt første møte med Avento, vår mentor (Møtereferat her: [2020-01-16 Første møte med Avento](#))
- Vi har gjennom møtet med Avento endret vårt syn for et minimum value product og har dermed gått vekk ifra 3 fase planleggingen vår til en mer to delt fase.
- Vi har blitt enig om hvordan vi kan modulere systemet og hvem som kan (initially) ha ansvar for hvilken del. Slik at vi kan jobbe mer uavhengig når man ikke har mulighet for å møte opp på skolen.
- Vi har laget et Postman team slik at gruppen kan gjøre samme REST API tester

Uke 4

Uke	4
Document Status	COMPLETE

- Vi har hatt vårt andre møte med Girts Strazdins. Her gikk vi gjennom det som ble tatt opp på møte med Avento og hvordan vi burde legge opp arkitektur, bruke docker og autentiserings server.
- Vi har også startet vår første sprint
- Vi har laget v.1.0 Gantt diagram som viser hva vi skal jobbe med når i dette semesteret
- Vi har laget ferdig en simpel in-memory autentiserings tjeneste som kan lett tas i bruk til utvikling og testing
- Vi har begynt på selve hjemmesiden til applikasjonen
- Vi har begynt å laget et basic brukergrensesnitt til desktop applikasjonen
- Vi har implementert opp- og nedlasting av filer på filsystem APIen (FS API)
- Vi har gjort slik at alle brukere som laster opp filer til FS API får en egen mappe der alt de laster opp ligger

Uke 5

Uke	5
Document Status	COMPLETE

- Vi har oppdatert use-case diagrammet for å møte den nye minimum-product requirement
- Vi har migrert autentiserings serveren til å bruke h2 database og fikset registrerings funksjonalitet
- Vi har så og si gjort de fleste tasks i sprinten
- Vi har så og si fullført fil system service
- Vi har lagt til en simpel database til filsystem APIen og implementert den
- Vi har laget ferdig powerpoint til fremføringen som var på onsdag (29.01)
- Vi har fullført forprosjektsrapporten
- Vi har skrevet ferdig coding convention
- Vi har skrevet ferdig contribution guide
- Vi har funnet ut at noe har skjedd med confluence som har gjort slik at vi har mistet noen av dokumentene våres og må mest sannsynlig skrive det på nytt

Uke 6

Uke	6
Document Status	COMPLETE

- Har fikset utsending av registrerings verifisering token etter at registrering er fullført, denne token på trykkes på for å verifisere sin oppretta konto
- Har fikset utsending av passord reset token dersom noen har glemt passordet sitt.
- Har også fikset reset passord funksjonalitet etter man har trykt reset token
- Har bygget en basic pipeline for authenticationserver inntil videre
 - (OBS! Alt nevnt ovenfor er bare API relatert, har ikke blitt lagt til i frontend enda)
- Har laget en simpel innloggings side i web applikasjonen
- Har laget et registrerings form på hjemmesiden i "banneren"
- Filesystem API kan laste opp moduler
- Filesystem databasen inneholder generell informasjon om prosjekt

Uke 7

Uke	7
Document Status	COMPLETE

- Har implementert autentisering i web applikasjonen ved hjelp av redux for state management
- Har fikset CORS konfigurasjon for autentiserings APIen slik at den kan akkesseres og har exposet Authorization headeren inntil videre slik at frontend kan plukke ut token og legge den i en cookie (Her skal det nok endres etterhvert)
- Har oppdatert gitlab-ci til å være mer generisk og kan brukes til å lett akkessere prod server når den tiden kommer.
- Har laget en docker-compose fil for å boote opp autentiseringsserver med database
- Har lastet opp autentiserings server til staging server og fikset CI/CD
- Brukere kan sette sine prosjekter til offentlig eller privat
- Har laget en autoriserings mal for Spring Boot applikasjoner
- Har lagt til auth malen i filesystem APIen og implementert autorisering på fleste endpoints
- Har lagt til administrator rettigheter på alle sikre endpoints på filesystem APIen

Uke 8

Uke	8
Document Status	COMPLETE

- Tried to change the gitlab-ci file to use the dev user instead of root, but autodeploys' template have a "bug" or something that restricts ssh authentication just for the root user without password. And since this is just a test server, we decided to keep it at root user.
- Dockerize the filesystem API with docker compose
- Almost finish materials and methods section on Bachelor document
- Create a Linux bash script that configures remote servers be able to run projects with docker
- Har laget en profile-dashboard custom route og ryddet litt i web applikasjonen
- Har lagt til passord endring "on-demand" som krever at bruker er autentisert
- Implementert metoder på desktop applikasjonen for å overføre og kjøre scripts på en remote maskin.

Uke 9

Uke	9
Document Status	COMPLETED

- Finished sketching out the material and methods section on the bachelor thesis
- Server manager can now execute configuration scripts on a given server
- A user can now view a project overview
- A user can now get a list of all the projects he/she has
- We have finished creating wireframes for the profile dashboard
- A user can specify language and version when uploading a module
- Created a more robust and resource friendly automated pipeline for authentication server
- Finished setting up "add module" implementation so its ready for integration
- Added application profiles to authentication server
- Fix issue on File Delivery API where you couldn't have universal file structure across operating systems
- Add endpoint on File Delivery API where admins can delete and view all projects

Uke 10

Uke	10
Document Status	COMPLETE

- A user can see a list of all of her/his servers
- A user can view a server overview when clicking on one of his/hers managed servers
- A user can create an empty "project" that contains project meta-data
- A user can add a server to his/hers account
- A user can upload modules to a specific project (Only web-client has been implemented)
- We have created a API for handling server related information and tasks
- We have created a gitlab-ci pipeline for ServerManager API
- We have created Dockerfiles for Java8 and Java11
- We have implemented roles and authorities in the authentication server
- We have fixed some database related issues and added new tables (roles and authority) as well as initial script in authentication server
- Added custom axiosRequest that sends with credentials on every request
- Implemented environment variables in web application
- Added dockerfile and other necessary configuration for dockerizing the web application
- Added listing of all projects and servers as well as searching functionality in the web application
- Added gitlab-ci automatic pipeline to the web application
- Added search project and get user project support in file delivery api
- Change how File Delivery API generates a path for modules
- Add basic swagger documentation on File Delivery API

Uke 11

Uke	11
Document Status	COMPLETE

- Implemented a solution for adding projects to a specified server in the web application
- Added the possibility to create a new project
- Added Documentation to web application
- Added JavaDoc and Open API 3 documentation to Authentication server
- Added JavaDoc and Open API 3 documentation to Server Manager
- Added JavaDoc and Open API 3 documentation to File Delivery API
- Updated Server Manager to accept a string containing project Ids to add to the server. With this we can define n numbers of projects to add to the specified server
- Added "show all projects" in File Delivery API
- Added "delete all projects" in File Delivery API
- Updated entities in File Delivery API to meet the new requirements
- We can now [use only 1 config for all systems instead of different dirs for different OSs in the File Delivery API](#)

Uke 12

Uke	12
Document Status	COMPLETE

- All projects are packaged into an image and uploaded to DockerHub
- Users can upload a dockerfile template
- Users can upload a docker-compose template
- Users can generate a docker-compose file from a template
- Generated docker-compose files can be stored and downloaded later
- Added logout option in the web application
- Added unverified support and alert reminder for verification
- Added validation and feedback on register form
- Finished project creation
- Added verification page
- Finished authenticated verification checking
- Fixed resending of verification mail, and added visual feedback regarding verification
- Added page for displaying search results (projects & users)
- Added backend support for storing and retrieving supported languages and versions
- Added gitlab-ci pipeline for General API
- Added OpenAPI specification
- Added isVerified field to the user in auth service. (This way a user can be authenticated while not verified, but can't use any features until user is verified)
- Added necessary support for verification related tasks in authentication server
- Added logic to search for existing users in authentication server
- Fixed security configuration to accept member, and admin, and not unverified users actions in server-manager
- When a new server is created, the server runs a preparation script for docker on it

Uke 13

Uke	13
Document Status	COMPLETE

- Splitted up the single controller in file system API into two for accessibility
- Added more documentation on file system API
- Implemented uploading of directories or multiple files as module instead of one standalone file
- Add a few docker-compose and dockerfiles as template for builder
- Added page for displaying public profiles
- Added project overview when looking at public projects
- Added pagination to search results to navigate between results
- Added a account settings page and implemented functionality for changing user password.
- Added dynamic navigation bar, so it changes from when user is authenticated or not
- Added organization panel for interacting with the organization.
- Added project settings menu. Here the user can delete their project
- Added option to submit a project submission to an organization
- Added public organization profile page
- Updated gitlab-ci pipeline for Server Manager
- Updated password change logic in authentication server
- Did some small chores in authentication server code
- Added organizational database structure and relationship in general api service and their corresponding endpoints

Uke 14

Uke	14
Document Status	COMPLETE

- Added Authorization filter to General API for authorizing organizational requests
- Added backend logic to handle member and project requests as well as giving an organization member the option to view, edit and delete their own requests
- Added upload of multiple files to module
- Implemented logic to upload, build and fetch option for docker-compose files.
- Created the first docker-compose blocks and dockerfile templates to use for building applications
- Added database for storing dockerfile and docker-compose blocks references.
- Did some other chore type of work on the File Delivery API
- Added dynamic fetching of organization data in web application
- Created a global navigation bar that is used on all interfaces.
- Added logic for managing own project requests, as well as editing them, deleting and creating more.
- Added interface for managing application submissions and project requests
- Added Redux State management for handling sidebar selection state.

Uke 15

Uke	15
Document Status	COMPLETE

Web Application

- Added option for an organization admin to edit some parts of a project submission
- Added concept page for uploading a finished docker-compose.yml file for full system uploads
- Added utility functions for fetching active browser windows width and height
- Added concept interface and implementation for handling database module uploads
- Added concept multi module upload support for "application" modules
- Performed a massive cleanup of redundant logic, redundant imports and unnecessary files, functions and others.
- Fixed server and project overview search interfaces in web application

File Delivery API

- Added testing templates for docker-compose and dockerfile build
- Fixed dockerhub repo connection issues and add use template database reference in DockerService
- Fixed missing files after uploading files without directories as module
- Added endpoints for searching and fetching public user projects
- Added MongoDB to store configuration and build module information on module upload
- Fixed internal error when processing empty cookies

Server Manager

- Updated docker-compose files for building project

General API

- Added option to comment on project request handling for admins
- Added endpoint for searching all public organizations

Uke 16

Uke	16
Document Status	COMPLETE

Web Application

- Updated how single module upload works and changed the predefined axios request to send Header with token instead of Cookie
- Fixed some visual anomalies
- Added option to delete modules and deploy projects
- Added server deletion option
- Fixed an issue with logout

File Delivery API

- Fix an issue where the API does not properly check when deleting projects/modules
- Updated Authorization Filter
- Replaced AuthenticationUser class with UsernamePasswordAuthenticationToken
- Removed mongodb autoconfiguration to prevent startup exception
- Implemented restriction for project names so that they can only contain alphanumeric values and dashes.
- Fixed compose-block building for config param values that are not capitalized
- Implemented restriction for module names so that they can only contain alphanumeric values and dashes.
- Implemented logic for deleting module mongodb data for every module inside a project when the project is deleted.

Server Manager

- Updated Authorization filter
- Updated the linux server preparation script
- Implemented logic for uploading a project to a server
- Changed how docker containers are created on the server

Authentication Server

- Updated authorization filter

General API

- Updated Authorization filter
- Replaced AuthenticationUser class with UsernamePasswordAuthenticationToken

Uke 17

Uke	17
Document Status	COMPLETE

Web Application

- Performed another massive cleanup, fixed bugs and made the user experience better

File Delivery API

- Updated the docker-compose builder to have a unique container name for each compose-block

Server Manager

- Added option for removing projects from a server, delete a deployed docker-compose.yml file and option to deploy multiple projects to same server

Authentication Server

- Performed some bug fixes and cleanup

General API

- Performed some bug fixes and cleanup

Uke N (Siste logg og inneholder resterende frem til 10.05)

Uke	N
Document Status	COMPLETE

Web Application

- Added option to upload a zip
- Edited the gitlab-ci pipeline
- Added more language and version options
- Added button to initialize the selected server with the installation script

File Delivery API

- Fixed validation not working when building templates with validation
- Added dockerfiles and corresponding compose-blocks for react, angular, static sites, spring boot and java projects.
- Added logic for unzipping a zip when uploaded as a module
- Replaced IOUtils copy method when uploading module
- Fixed template unique entry check always failing
- Did a cleanup of the service as a whole

Server Manager

- Added own endpoint for initializing the server. So now the user has to explicitly tell the service to initialize the server instead of initializing on server creation

O Note from tester

"AutoPackager er en fin løsning på hvordan man kan simplifisere deployment og pakking av prosjekter til server. Førsteinntrykket mitt når jeg kom inn på nettsiden var at den såg veldig fin og ryddig ut. Etter jeg opprettet konto ble jeg tatt videre til mine prosjekter. Jeg hadde litt problemer med det å finne frem til ting, men syntes det gikk raskt å lære seg hvor du fant de ulike funksjonene. Der var en del bugs som gjorde at noen ting ble litt merkelig, men antar at dette blir fikset etterhvert. For min egen del så er dette noe jeg kan bruke på mine freelance prosjekt, men for at det skal gå så må der legges til mer options og sikkerheten må bli mye bedre. Foreløpig passer nettsiden best til studierelaterte prosjekt, ikke kommersielle prosjekt."

