TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies
Computer Science

Oskar Pihlak 185147IAIB

Programming assignment management registry Aurora

Bachelor's Thesis

Supervisor: Ago Luberg
PhD

Tallinn 2021

TALLINNA TEHNIKAÜLIKOOL

Infotehnoloogia teaduskond

Tarkvarateaduse instituut

Oskar Pihlak 185147IAIB

PROGRAMMEERIMISÜLESANNETE HALDAMISE REGISTER AURORA

Bakalaureusetöö

Supervisor: Ago Luberg

PhD

Tallinn 2021

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature, and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author:	Oskar Pihlak	....
		(signature)
Date:	25.05.2021	

Annotatsioon

PROGRAMMEERIMISÜLESANNETE HALDAMISE REGISTER AURORA

Mitmed kursused Tallinna Tehnikaülikoolis kasutavad õppeaine läbiviimiseks lisanduvalt loengutele programmeerimisülesandeid. Antud lähenemist on praktiseeritud üle pikema aja ning ülesannete arv on kasvanud peaaegu tuhandetesse. Õpetajatele on esitatud väljakutse määrata kursusele parim ülesanne, kuna iga kursus on erinev ja neil puudub ülevaade juba olemasolevatest ülesannetest.

Lõputööks on ülesannete haldamise register TalTechi töötajatele, mis võimaldab töötajatel lihtsalt valida ülesanne kindla teema ja raskustasemega. Parima tulemuse saavutamiseks analüüsitakse olemasolevaid lahendusi ning teadusartikleid. Pärast esialgset analüüsi ja nõuete kogumist alustatakse arendust toetudes *Agile* ja *DevOps* kultuuridele, ning *Continous Delivery* ja Ekstreemse programmeerimise tavadele. Töö eesmärk on lihtsustada õpetajate ja abiõpetajate tööd.

Rakenduse veebipõhine kasutajaliides on saadaval õpetajatele ja abiõpetajatele leheküljel `cs.ttu.ee/services/aurora/front/`.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 34 leheküljel, 10 peatükki, 8 joonist, 1 tabelit.

Abstract

Multiple courses in TalTech teach their subjects by giving students programming assignments. Throughout semesters the number of assignments the course teaching staff has to choose from - grows. After a while, the management of sources can become hard to manage. It is a challenge for teachers to determine the best assignment for the current course since every course is different.

In this thesis, an assignment management registry is developed for TalTech staff dealing with assignments in courses. Existing research and solutions are analyzed. After the initial analysis and requirements gathering, the progressing development is conducted with the Agile and DevOps cultures, taking adaptation from the Continuous Delivery and Extreme programming practices. The goal of the work is to simplify operational activities for teachers and teaching assistants working on assignments.

The application's client interface is available for teachers and teaching assistants on `cs.ttu.ee/services/aurora/front/`.

This thesis is written in English and is 34 pages long, including 10 chapters, 8 figures, and 1 table.

List of abbreviations and terms

API	Application Programming Interface
BEM	Block Element Modifier
CD	Continuous Delivery
CI	Continuous Integration
CLI	Command Line Interface
CRUD	The four basic functions of persistent storage - create, read, update, delete
CSS	Cascading Style Sheets
DAO	Data Access Object
DevOps	A set of practices that combines software development (Dev) and IT operations (Ops)
DTO	Data Transfer Object
DSC	Desired State Config
GMT	Greenwich Mean Time
HTML	HyperText Markup Language
HTTP	HyperText Transport Protocol
IDE	Integrated Development Environment
JDBC	Java Database Connectivity.
JPA	Java Persistence API
JSON	JavaScript Object Notation
MVC	Model View Controller
NFR	Non-functional requirement
NPM	Node Package Manager
ORM	Object Relational Mapper
OSI	Open Systems Interconnection
PostgreSQL	Relational database system.
PWA	Progressive Web App
REST	Representational State Transfer in an architectural methodology, which is a guideline.
SASS	Syntactically Awesome Style Sheets
SCSS	Sassy CSS
SIL	Service Interface Layer
SQL	Structured Query Language

TCP	Transmission Control Protocol
UI	User Interface
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UX	User Experience
SSH	Secure Shell Protocol
VM	Virtual Machine
YAML	(A recursive acronym for "YAML Ain’t Markup Language") is a human-readable data-serialization language for presenting data.

Table of Contents

List of Figures

List of Tables

1. Introduction

Students in TalTech IT degree programs solve lots of programming assignments during semesters. Current methods of managing created assignments between Git, Moodle, Codera, and other TalTech toolings are time-consuming. Manual management without a complete overview of all assignment sources and content makes it possible to accidentally introduce duplicates into the system, additionally to the intentional duplicates introduced due to current management methods.

For results validation, a weekly feedback loop is established with teachers and teaching assistants who will be using the software alongside development. The expected outputs are the following:

1. Time reduction in assignment management overhead;
2. Achieving a better overview of assignments;
3. Improving the jobs of teachers and teaching assistants who come in contact with assignment creation or management.

The registry will enable TalTech to integrate Git and other possible storing mediums for programming challenges and materials. Jetbrains[1] IDEs (Integrated Development Environment) were used in the registry development.

This paper consists of ten chapters. The introduction composes the first. The second and third chapters cover planning, introducing used methodologies, and the analysis. The fourth, fifth, sixth, and seventh chapters are the core of the paper. The fourth goes over the constructed system's infrastructure side. The fifth introduces the architecture of the software system. The sixth gives an overview of the back-end. The seventh chapter gives an overview of the front-end. An introduction to the most prominent features is given in the eighth chapter. The ninth and tenth chapters look at the results, summarise the paper, and propose further development plans.

[1]`https://www.jetbrains.com/`

1.1 Domain overview

An assignment is a piece of work given to someone as a part of their studies or job [1]. In Aurora, it is defined as a programming challenge with a description and tests. It is stored in any TalTech Git repository where challenges are eligible to be chosen for students to solve. Assigning programming assignments to students is practiced across TalTech IT courses. The following are examples of a few:

1. ITI0202 - Computer programming;
2. ITI0002 - Refresher Course in Programming;
3. ITI0204 - Algorithms and Data Structures;
4. ITI0211 - Logical Programming;
5. ITI0102 - Introduction to Programming.

A course can accumulate many Git repositories, and the assignment management can become tedious. The following are examples of courses whose assignment management extends across multiple repositories:

- ITI0202 - Computer programming;
    1. iti0202-2021,
    2. iti0202-2020,
    3. iti0202-2019,
- ITI0102 - Introduction to Programming;
    1. iti0102-2021,
    2. iti0102-2020,
    3. iti0102-2019.

1.2 Task setting

TalTech teachers create repositories to store assignments for every course. Throughout many classes, the amount of repositories created eventually decreases assignment maintainability, and achieving a complete overview is challenging. Searching for the most up-to-date assignment is time-consuming and troublesome, especially if the correct repository is unknown.

1.3 Goal setting

The author's primary goal is to simplify the work of teachers and teaching assistants who maintain, create or test assignments. This work aims to construct TalTech's assignment management registry named Aurora. The name is an inspiration from Roman mythology. The registry will unify assignment management into a single tool. The goal is divided into the following subtasks:

1. Build and maintain a production environment;
2. Develop the Assignments API (Application Programming Interface);
3. Develop the UI (User Interface);
4. Document the work done;
5. Establishing registry-specific software development practices and processes;
6. Conducting business analysis with supervisor feedback;
7. Creating an evolutionary system design.

1.4 The scope of the thesis

The thesis covers the maintenance, development, analysis, and system design of the application. Existing programming assignment management algorithms in TalTech's Computer Science department are also analyzed. An initial set of requirements are established upon the analysis. The thesis scope includes but is not limited to the following:

1. Analyzing similar applications;
2. Requirement gathering;
3. Application environment setup and maintenance;
4. Development and architecture of the application;
5. Project management.

1.5 Existing work and research

Before starting the registry development, some related applications were analyzed to understand what has worked for different business use cases. Many relevant applications are developed as online judges, focusing on programming interview passings. Regardless of the slight difference in domain focus, these applications have relevant functionality for the assignment registry.

1.5.1 Hackerrank

Hackerrank[2] was founded in 2009. The technology company focuses on competitive programming assignments for developers and businesses [2]. The product holds unnecessary complexity regarding the thesis need.
This software is used in the course Advanced Programming (ITI0214) for some assignment solving. The following are the most relevant pros and cons:

- + Possibility for partnerships
- + Many programming assignments
- + SaaS service does not require self-hosting and maintenance.
- - No possible integrations with Moodle
- - No possible integrations with Gitlab or Github
- - Not possible to develop custom functionality according to needs

1.5.2 Codera

Potentially a similar system to the current thesis development is Codera[3]. It is a TalTech programming assignments application developed as a Bachelor's thesis[4] by Kirill Denisov for holding public programming assignments. Codera manages tasks but is created with the end goal of publicly sharing programming assignments [3]. The current Bachelor's thesis goal is to be more of a tool for teachers and teaching assistants. Since both applications are meant for different business needs, writing an extension for Codera in the scope of this thesis is not reasonable and will introduce unnecessary coupling. The following are the most relevant pros and cons:

- + Possible storage for assignments
- - Difficult to dynamically add assignments
- - Difficult to search assignments
- - Different end goal

[2]`https://www.hackerrank.com/`
[3]`https://codera.cs.ttu.ee/`
[4]`https://digikogu.taltech.ee/et/Item/49b19a90-a516-434b-80d4-addaf5178bb6`

1.5.3 UVa Online Judge

The automated online judge UVa[5] Hosted by the University of Valladolid [4]. This software is used in the course Advanced Programming (ITI0214) for some assignment solving.

+ Many programming challenges (4000+)
+ User registration for everyone
+ Possibility to test assignments
- No direct integration with the search of TalTech's Git repositories

1.5.4 LeetCode

LeetCode[6] was founded in 2015 [5]. It is one of the most well-known online judge platforms.

+ Many programming challenges (1000+)
+ Many programming languages (15+)
+ User registration for everyone
+ Possibility to test assignments
- No direct integration with the search of TalTech's Git repositories

1.5.5 Exercism.io

Exercism[7] is a coding challenge website created by Katrina Owen in 2013 as a tool to be used by her programming students [6]. The website is being evolved in 2021, but the current analysis is the following:

+ Many programming challenges
+ Many programming languages
- Difficult to search and solve assignments
- No direct integration with the search of TalTech's Git repositories

[5]https://onlinejudge.org/
[6]https://leetcode.com/
[7]https://exercism.io/

1.6 Projects

The thesis content is held on TalTech GitLab[8] servers. This Git provider was chosen because the registry is developed as their native tool, and it is seamless to iterate processes in the client environments. Doing so prevents conflicts that may arise with any environment migrations to TalTech platforms in the future. The thesis has been developed for around 600 hours between three projects.

The Aurora Operations[9] project holds the Docker Swarm containers, documentation, code, system cleanup jobs, and assets for keeping the registry working correctly. For example, the Aurora-specific proxy is deployed here. This project is in semi-active management, meaning tasks are created, but sprints are not made since the amount of work is not consistent.

The Aurora Web UI[10] project (122 issues, 252,3 hours, 8870 lines of code) is the browser-based front-end web interface, exposing the registry to users. It communicates with the registry APIs, currently only with the Aurora Assignments API. This project is in active development. Sprints are organized weekly, a backlog of issues exists.

The Aurora Assignments API[11] project (147 issues, 232,7 hours, 7208 lines of code) is a back-end service, which provides data for the Aurora Web UI and any other UI type. It communicates with persistence solutions and holds complex business logic. This project is in active development. Sprints are organized weekly, a backlog of issues exists.

[8]`https://gitlab.cs.ttu.ee/`
[9]`https://gitlab.cs.ttu.ee/aurora/aurora-devops`
[10]`https://gitlab.cs.ttu.ee/aurora/aurora-frontend`
[11]`https://gitlab.cs.ttu.ee/aurora/aurora-backend`

2. Methodologies

Methodologies are systems of practices, techniques, procedures, and rules used by those who work in a discipline [7]. The project development has adapted properties from the Agile and DevOps cultures.

2.1 Values

Practices describe how things are done, and values add the reasoning. The registry development is following these values:

- Minimalism - Easy to understand, deal with, and use;
- Feedback - Weekly cycles, CI (Continuous Integration)/CD (Continuous Delivery), Incremental design, small builds, ruthless prioritization;
- Transparency - The client and the development team know what is being worked on;
- Sustainability - Iterative effort, development at a maintainable velocity.

Minimalism proved challenging to maintain with the registry development velocity. Moreover, later required multiple phases of refactoring, which dragged down new feature development speed. This shows that technical debt is initially good in the lifecycle of a feature, but after features are established, the debt should be removed to maintain speed.

The feedback cycles were beneficial beyond expectations. Thanks to them, many initially developed ideas were improved, re-prioritized, and re-tailored to more accurately match user needs. Feedback cycles were established with the experienced teaching assistants Enrico Vompa, Timo Loomets and the supervisor Ago Luberg.

Transparency was easy to keep with the GitLab project group management tooling. Since the client and relevant personnel have access to the registry project group, they can see the tasks in development. A weekly overview was also given to the client and supervisor Ago Luberg.

Sustainability was the starting value. The project idea was already established in May 2020. Starting from the end of 2020, the projects have been in constant development.

2.2 Project management

Choosing a management approach depends on project characteristics and the type of people in the team. The thesis is developed in a dynamic environment. Feature requirements and priorities could have changed weekly, depending on feedback. As a result, the Agile set of principles were adapted [8].

2.2.1 Development iterations

Up until the start of 2021, the development was done more in a kanban way. The first sprints in the API and UI projects were accumulating all the work done in 2020. After the first sprints, the ongoing sprints were made in weeks or two weeks. Sprints are focused, sustainable, planned, and measured. Upon completing a sprint, a release version tag was added to the final commit, and a release was made. Every sprint moved the version by 0.1. Since the API is complex, in some weeks, only that was developed. This results in version differences between the UI and API projects. The sprints made are listed in Table 1

Table 1. Sprints.

V.	Front-end	Back-end
0.1	Initial release (34 issues, 118h)	Initial release (16 issues, 89h 20m)
0.2	Assignments display (4 issues, 3h 5m)	Assignments API (13 issues, 44h 19m)
0.3	Repositories & Assignments (10 issues, 15h 10m)	Database & Assignments API (5 issues, 27h)
0.4	Tags & repositories (13 issues, 26h 13m)	Assignments API & Git improvements (6 issues, 13h)
0.5	Repository & Assignment view (22 issues, 15h 35m)	Assignments API & Further Git improvements (3 issues, 4h)
0.6	Handle no API communication (16 issues, 34h 47m)	Gitlab API, assignments & repositories (6 issues, 18h)
0.7	Automatic tagging & similar assignments (10 issues, 23h 35m)	Assignments, repositories & maintenance (7 issues, 14h 55m)

Continues...

Table 1 – *Continues...*

V.	Front-end	Back-end
0.8	-	Tags & repositories (9 issues, 28h 24m)
0.9	-	Assignment files & performance (26 issues, 45h 35m)
1.0	-	Tags, Tests, Repositories, Assignments, Filesystem (20 issues, 47h 5m)
1.1	-	Similar files, Dynamic tagging (16 issues, 32h)

2.2.2 Prioritization

Prioritization is the most significant and critical portion of requirements analysis and project planning due to restrictions on development time and resources. Techniques can be categorized into three scales called nominal, ordinal, and ratio [9]. A variety of approaches can be found in appendix 7.

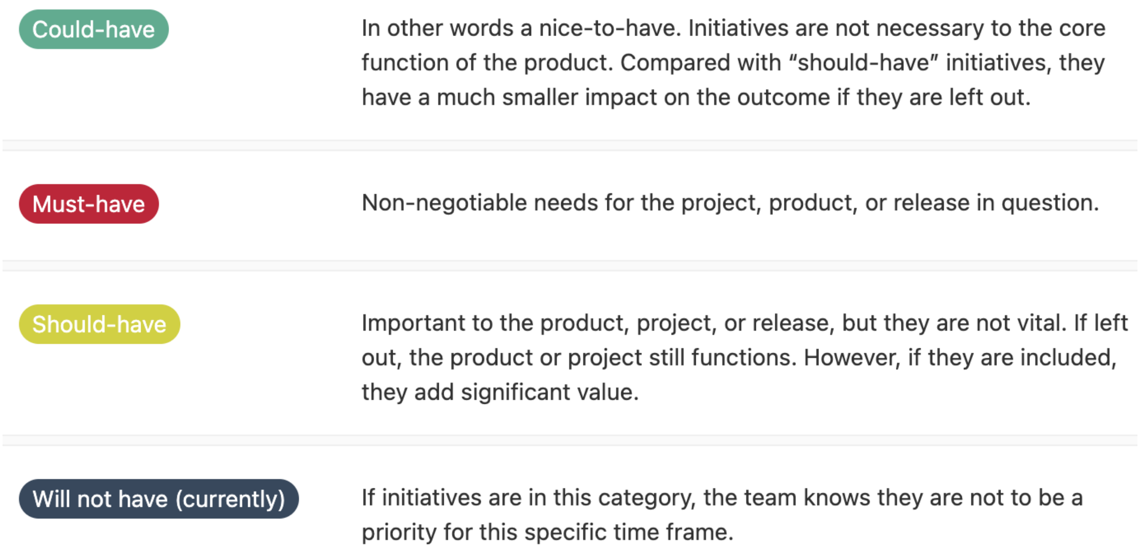
This project uses the MoSCoW[1] prioritization technique on development tasks. It is a collaboration-based nominal scale technique and is one of the easiest methods, originating from the dynamic software development method [9]. The MoSCoW categories are implemented as GitLab tags seen in Figure 1. A category is assigned to a task based on the implementation importance. Requirements in the same group represent similar priorities.

The Kano model[2] is used for a more customer-centric view. Requirements can be divided into the three following categories:

- Baseline Expectations - Must be present for the product to be successful (e.g., assignment search);
- Linear Satisfiers - The better they are – the more customer satisfaction they bring, the worse they are, the more dissatisfaction they bring (e.g., assignment search speed);
- Delighters - Details that will typically not be missed if they are absent but will impress the client (e.g., displaying search speed and loading spinners);

Combining the simplicity of MoSCoW with Kano's customer-centric view results in a

[1]`https://www.productplan.com/glossary/moscow-prioritization/`
[2]`https://www.career.pm/briefings/kano-model`



Could-have	In other words a nice-to-have. Initiatives are not necessary to the core function of the product. Compared with “should-have” initiatives, they have a much smaller impact on the outcome if they are left out.
Must-have	Non-negotiable needs for the project, product, or release in question.
Should-have	Important to the product, project, or release, but they are not vital. If left out, the product or project still functions. However, if they are included, they add significant value.
Will not have (currently)	If initiatives are in this category, the team knows they are not to be a priority for this specific time frame.

Figure 1. MoSCoW labels in the GitLab project.

robust lexicon to explore the mix of requirements that need to be delivered [10]. This proved beneficial when prior sprint planning was conducted with the supervisor. The MoSCoW tags were added to GitLab tasks, but Kano tags were kept on a feature-basis on notes instead of task-basis in GitLab since Kano is more general and MoSCoW works with individual task tagging better.

2.3 Continuous Integration and Continuous Delivery

This section will introduce techniques for keeping development speed and software release roll-outs stable and agile.

2.3.1 Continuous delivery and deployment

Delivery is the ability to deploy on-demand. Deployment is the instant production deployment of a change upon a passed pipeline. CD practices lower deployment risks enable real progress and regular user feedback [11].

All the deployable projects are developed using trunk-based development and started out using Continuous Deployment, meaning a production deployment on every commit. This was convenient but restricted committing flexibility. After projects increased in complexity enough, they were switched to Continuous Delivery, starting with the API, then the UI project. CD allowed pushing commits up to the remote more often without the fear of affecting the production environment. A pipeline failure is handled by rolling forward. Doing so leaves behind a trail of the broken version for potential auditing. There are various deployment strategies that were considered: for example, Dark launching[3], GitHub, Rolling Update[4], Blue/Green deployment/Phoenix deployment[5], Canary deployment[6]. The projects Swarm services will use the rolling updates deployment method since Docker Swarm has a built-in mechanism for this approach[7] [11].

2.3.2 Continuous integration

CI (Continuous Integration) goal is to get fast, automated feedback on the correctness of the application every time there is a code change. CD cannot be done without good CI practices in place.
Feature toggles is a tool for aiding trunk-based development and an easy way to test business features. Each toggle's technical debt and life span are different [12], [13]. The application used a release toggle with the file authoring feature implementation. The application projects will use CI in the provided VM (Virtual Machine) with trunk-based development.

[3]https://martinfowler.com/bliki/DarkLaunching.html
[4]https://link.springer.com/chapter/10.1007/978-3-540-30225-4_13
[5]https://martinfowler.com/bliki/BlueGreenDeployment.html
[6]https://martinfowler.com/bliki/CanaryRelease.html
[7]https://docs.docker.com/engine/swarm/swarm-tutorial/rolling-update/

3. Analysis

This section will cover the domain background check, functional requirements, non-functional requirements, alternative requirement classifications, possible requirement gathering methodologies, and the ones used.

3.1 Requirements

Requirements are usable representations of needs, which focus on understanding what kind of value could be delivered if they are fulfilled [14]. Initial requirements are gathered and then iterated with the supervisor. After an iteration, they are classified into functional and non-functional requirements and persisted on notes.

3.1.1 Requirements classification

Requirements will be differentiated between behavior (e.g., assignment search response time) and representation (e.g., programming languages). Representational NFRs (Non-functional Requirements) are how systems are syntactically or technically represented, described, structured, implemented, or executed [15]. Behavioral NFRs are additionally classified according to the system view (interface, architecture, or state) and the expressive behavior theory, which are the following [16], [17], [18]:

- Syntactic - structure on which behavior can be described;
- Logical - set of interaction patterns;
- Probabilistic - probabilities for a set of interaction patterns;
- Timed - set of interaction patterns with relation to time.

User requirements are covered in appendix 12, and system requirements are distributed according to the ISO 9126–1[1] in the following way:

- Functionality
  - Interface
    1. Syntactic - A list of assignments will be displayed when navigating to the assignments page;
    2. Logical - Assignments will be returned according to the filtering criteria that the user provides;
    3. Logical - If the assignment file's content matches the search criteria, a snippet of the file will be displayed;
    4. Syntactic - It has to be possible to search for assignment repositories.
- Usability
  - Representational
    1. The application can be comfortably used from a device with a screen size of 1200px x 700px;
    2. The application displays notifications about successful and failed requests to the user;
- Reliability
  - Interface
    1. Logical - Application maintainers will be notified of system failures.
  - Architecture
    1. Probabilistic - The system availability shall be 90%;
    2. Logical - Application containers shall be restarted upon failure when feasible.
  - State
    1. Timed - The system should recover from an incident within two hours.
- Efficiency
  - Interface
    1. Timed - The application shall respond to any TCP (Transmission Control Protocol) request within 10s.

    1. Logical - Real-time communication that takes longer than 10s to complete should use an alternate communication form (e.g., WebSockets).

[1]`https://www.iso.org/standard/22749.html`

- Maintainability
  - Representational
    1. The expected lifespan of the software is at least two years;
    2. The back-end of the application shall be written in Java 11 or higher, and Spring-boot;
    3. The front end of the application shall be written in Angular 11 or higher;
    4. The source code for the applications will be stored in the client's Git environment.
- Portability
  - Interface
    1. Timed - The administrator should be able to migrate the application manually between different VM's at most in 2 hours.
  - Representational
    1. Applications shall be placed inside Docker containers;
    2. A VM shall have at least a single container of each application container;
    3. The application shall run on a Linux VM.

4. IT Operations

The application is deployed to a TalTech server VM and is reachable on the base Aurora URL[1] (Uniform Resource Locator). It supports reading from multiple SSH (Secure Shell Protocol) key files for receiving credentials, which are needed to access TalTech GitLab repositories. Multiple SSH key support was implemented because many sprints went into debugging Git credentials. This also simplifies local development's key management if prior keys exist in the local machine. Furthermore, distributing repository access between different keys in production increases security. Although it is possible, a single SSH key is used in production. The software is planned to operate within TalTech, so no remote URI (Uniform Resource Identifier) whitelisting was done.

4.1 Virtualization

Virtualization provides a platform to run different services of operating systems. It facilitates building multiple virtual machines on a single primary physical machine, either in the form of hypervisors or containers [19].

The supervisor provided a single TalTech server VM, and the registry author accessed it through SSH. The time zone was set to GMT+3 (Greenwich Mean Time) to match Tallinn, SSH keys for the registry were inserted, Docker tooling was installed, and GitLab runners were set up for CI/CD. The application's production environment, pipelines, and monitoring all operate within a single Ubuntu 18.04[2] VM instance with 10 GB of storage and 4 GB of RAM (Random Access Memory). Containers running on the VM instance can be viewed in appendix 6.

4.2 Containerization

Containerization technology combines the application and related dependencies into a deployable package. Such a platform is, for example, Docker[3] [19]. It enables application separation from the infrastructure, providing the ability to run applications in a loosely isolated environment [20].

[1] `https://cs.ttu.ee/services/aurora`
[2] `https://releases.ubuntu.com/18.04/`
[3] `https://www.docker.com/`

The deployed application runs in a single node cluster Docker Swarm[4]. An alternative would have been to use Kubernetes[5], but Swarm matches the thesis requirements closest due to the development scope and no detailed configuration needs. In the future, it is possible to split the cluster between multiple nodes. A single node is kept since the VM has enough capability to serve the users, and a multi-node setup will unnecessarily increase complexity. The Swarm topology is shown in Figure 2. The Swarm cluster is project agnostic, meaning containers and networks can be deployed from various projects. The cluster-wide overlay network is called aurora-network. The back- and front-end bridge networks are called back-end, and front-end respectively. To prevent the VM from being overloaded, some of the Swarm container's resource consumption is limited. At one point, the registry proxy failed to serve the API due to resource restrictions, but this was fixed by increasing the maximum possible amount of resources to consume. The Docker container description is written in YAML(YAML Ain't Markup Language), and the operations project file can be seen in appendix 4.

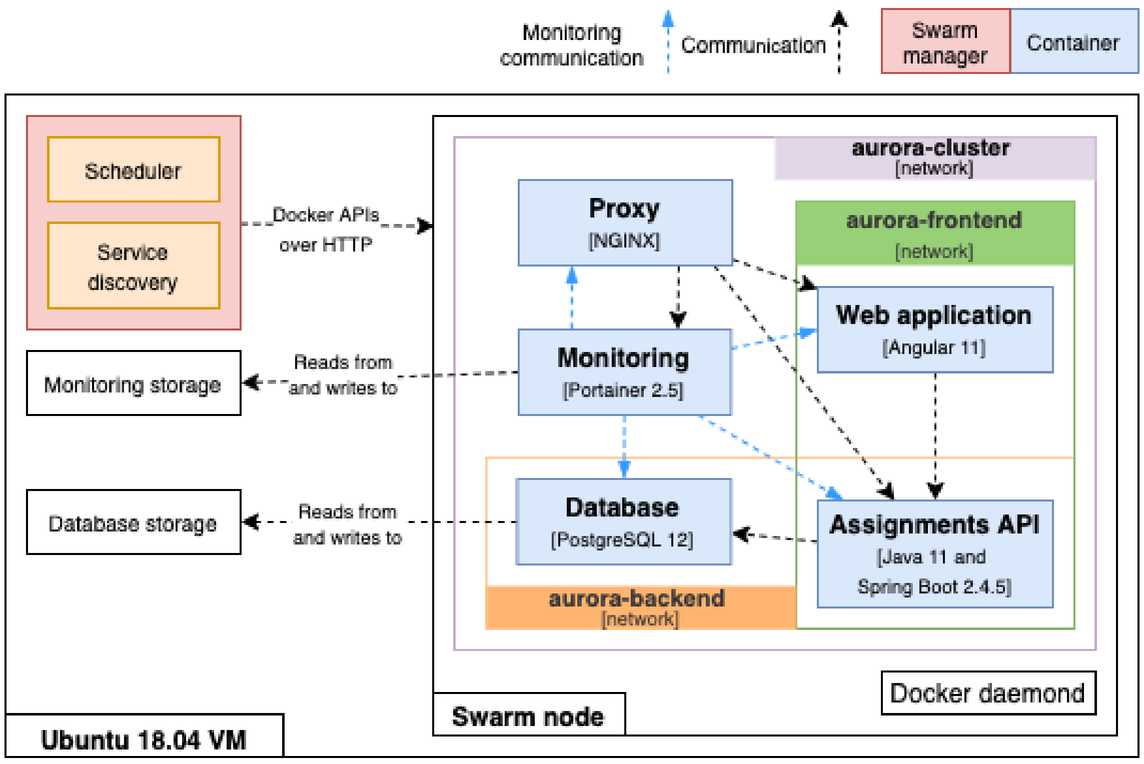


Figure 2. Docker Swarm.

[4]https://docs.docker.com/engine/swarm/
[5]https://kubernetes.io/

4.3 Proxies

All the proxies relevant to the registry are HTTP (Layer 7 of the OSI networking model). The application reverse proxy was chosen from vendors who could support distributed systems. Initially, a Cloud-Native networking stack called Traefik[6] was chosen. However, the configuration and maintenance were too general purpose and complex for the project. Instead, it was decided to match the host's reverse proxy with NGINX[7]. It is used as a proxy for load balancing, back-end routing, caching, and serving web content.

The application is reached through two NGINX reverse proxies, the TalTech, and the application proxy. The TalTech proxy detects for the Aurora base URL[8]. After receiving a request, it is directed through an internal network to the application's Swarm proxy container, which listens to port 8080. The base URL suffix /front displays the web application UI, /api exposes the APIs; currently, only the Assignments API and /portainer reveals the Docker Swarm monitoring interface.

4.4 Pipelines

Every commit is considered a release candidate. A commit or a group of commits trigger a pipeline run on the UI, API, and Operations projects. CD has the purpose of killing bad release candidates [12]. After a pipeline passes on any project, there should be confidence for deployment. If any of the application project pipeline jobs fail before deployment, the production environment remains running. A failing deployment signals operational problems (e.g., VM's memory is full). The application and its architectures increase in complexity throughout development (e.g., separating layers into different projects), and the pipelines become various but simpler or as complex.

The pipelines center around dealing with DockerHub Docker containers. All projects except Aurora Resources have a pipeline. The main application pipelines are in Aurora Web UI and Aurora Assignments API. The project Aurora Operations focuses on operational aspects. After the operations project pipelines reach DockerHub's anonymous user pull rate limit[9] (100 pulls/6h), all Docker networking operations on every project are conducted from an authenticated user account, which rate limit is higher (200 pulls/6h). A fallback image pulling policy "if-not-present" is added into GitLab runners[10]. The pipelines and the production environment run on the same VM. Since there have been various production

[6] `https://traefik.io/`
[7] `https://www.nginx.com/`
[8] `https://cs.ttu.ee/services/aurora`
[9] `https://www.docker.com/increase-rate-limit`
[10] `https://docs.gitlab.com/runner/executors/docker.html`

outage incidents due to the VM memory getting close to full, the pipelines are designed to take up minimal storage space. The pipeline's storage consumption optimization is to conduct storage and Docker cache checking and clean up before and after every pipeline. As of the time of writing, no other pipelines have caused the VM to fail.

In Figure 3 the Aurora Operationspassed pipeline is presented. The given pipeline updates the monitoring and proxy setups when needed.

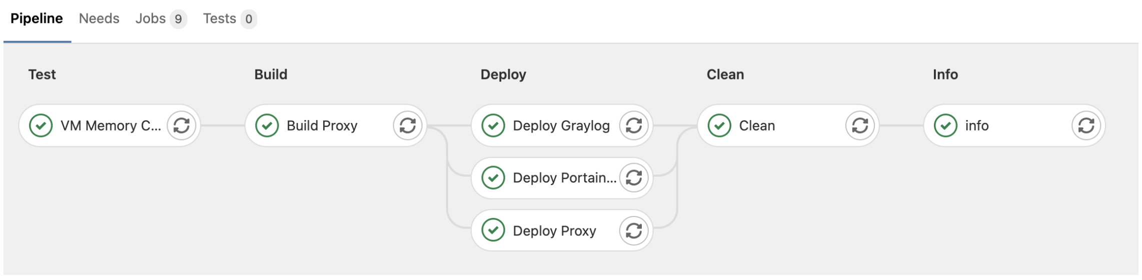


Figure 3. Aurora Operations pipeline.

In Figure 4 the Aurora Web UIpassed pipeline is presented. The service update job updates the Swarm NGINX container. The pipeline compiles the code into a production state and deploys the image to Docker Hub[11]. The deployment pulls the new image into the production environment and runs it. There is a minor downtime ≈ 1 min between deployments.

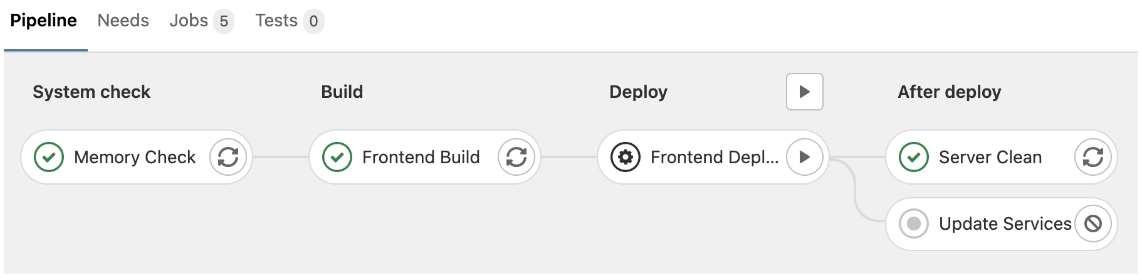


Figure 4. Aurora Web UI pipeline.

In Figure 5 the Aurora Assignments APIpassed pipeline is presented, which begins and ends with a memory cleanup. This is the most complex pipeline of the application. The testing phase goes over the VM's memory state and the API unit tests. The build phase creates a Docker image and dry-runs the database schema changes. The deployment is triggered manually and consists of the database and API deployments, which are done separately. The order of the pipeline jobs matters since the following jobs may depend on previous ones, displayed in appendix 11. After a database schema update failure, the

[11]`https://hub.docker.com/`

transaction will be rolled back. The new API version could break if it were deployed before the database with data model changes, and the database update failed. There is a minor downtime $\approx$ 1 min between deployments.

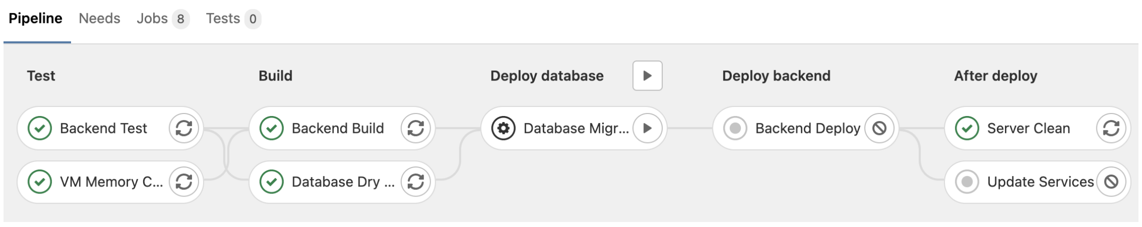


Figure 5. Aurora Assignments API pipeline.

5. Architecture

It is a shared understanding of the system's most essential and hard-to-change aspects. Every project has to decide what is architectural for their case. The types can be categorized as technical, data, security, and domain. Some architectural patterns are monolithic (e.g., layered), service-based (e.g., microservices), and distributed (e.g., event-driven) [21].

5.1 Architectural overview

The application uses a monolithic approach in the thesis scope. The system containers can be found in Figure 6. This is done since it is easier to fix modeling decisions and layer bordering within the same application and extract when clear domain borders have been established. Distributing a system introduces lots of complexity which is not required in the current scope. The application components are organized and assembled into a top-down four-layered architectural pattern. The pattern organizes itself into layers that are specific to significant system responsibilities. Layers are independently isolated portions of the system that behave like external entities with the rest of the system. They are named presentation, business, services, and data/infrastructure.

Interactions between other system components are done via well-defined interfaces, e.g., REST (Representational State Transfer). A layer exposes its functionalities through an abstract API. An example of an n-layered pattern is the OSI (Open Systems Interconnection) model.

Developing without a traditional architecture in place could result in the "Big Ball-of-Mud" anti-pattern. The thesis application layers will be closed, meaning each layer responds only to the layer above it. The system will be designed with the evolution of the architecture kept in mind, meaning it will be possible to easily extract components from the monolith. After demand within an extractable component increases, it could be pulled into a microservice. Consistency and coherency are the critical aspects of a system design.

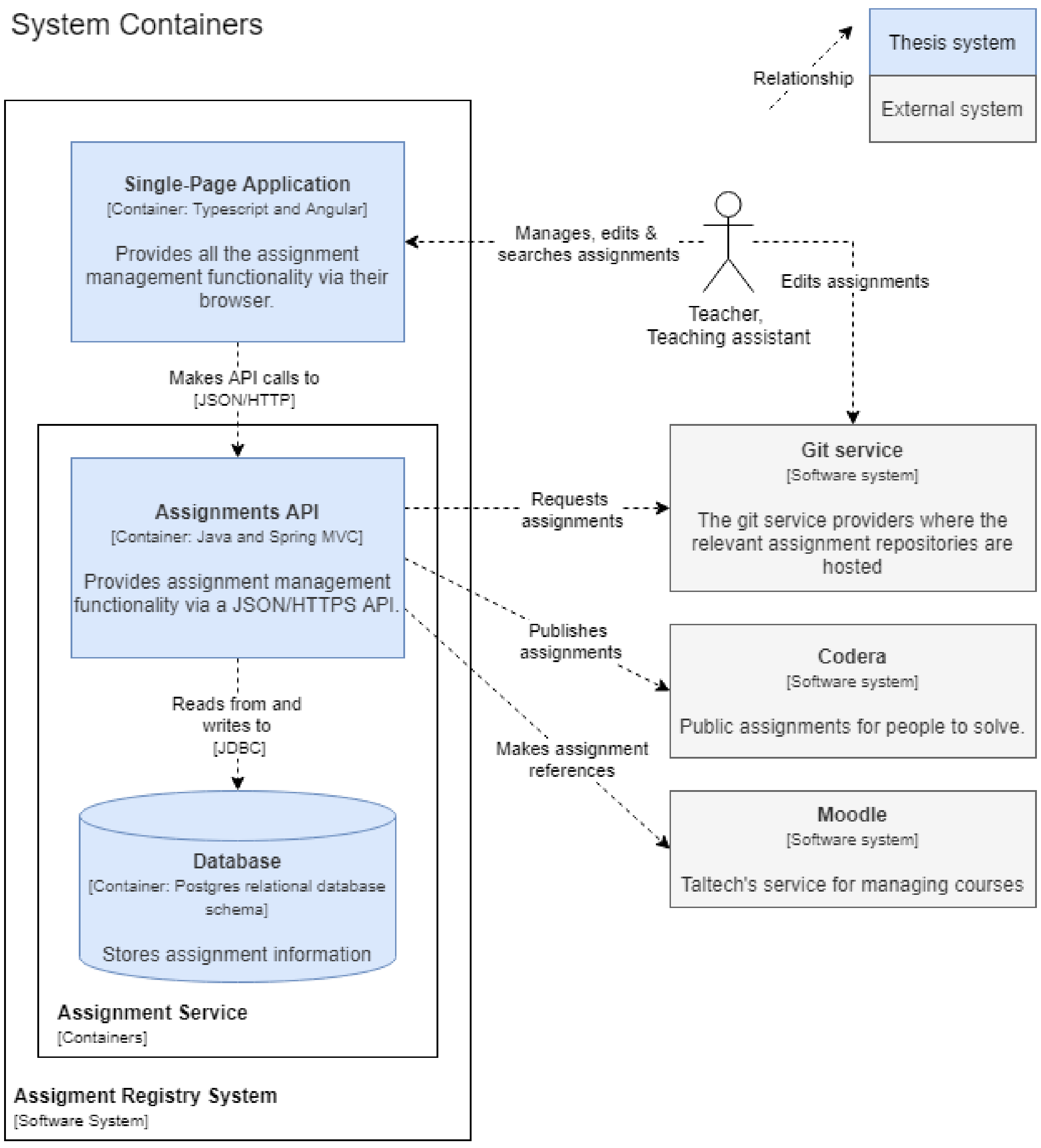

System Containers
Relationship
Thesis system
External system
Single-Page Application
[Container: Typescript and Angular]
Provides all the assignment management functionality via their browser.
Manages, edits & searches assignments
Teacher, Teaching assistant
Edits assignments
Makes API calls to [JSON/HTTP]
Assignments API
[Container: Java and Spring MVC]
Provides assignment management functionality via a JSON/HTTPS API.
Requests assignments
Git service
[Software system]
The git service providers where the relevant assignment repositories are hosted
Publishes assignments
Codera
[Software system]
Public assignments for people to solve.
Makes assignment references
Moodle
[Software system]
Taltech's service for managing courses
Reads from and writes to [JDBC]
Database
[Container: Postgres relational database schema]
Stores assignment information
Assignment Service
[Containers]
Assigment Registry System
[Software System]

Figure 6. Architectural system containers.

6. Back-end

The back-end composes a single JDK 11 based API with a dedicated relational database. It is named the Aurora Assignments API, and handles the assignments part of the software.

6.1 API

The namings have to be clear and consistent throughout the system, indicating intent, action, and the service provided. The correct design of methods, classes, systems, APIs, services, and microservices is connected [22].

The Assignments API[1] is written in Java 11 using Spring Boot since the author has the most experience with them. Java is a strongly and statically typed, general-purpose, concurrent, class-based, object-oriented language related to C and C++ and intended to be a production, not a research language [23]. Spring Boot simplifies Spring development by handling explicit boilerplate configuration [24]. The build tool used with Spring Boot is Gradle since it can have faster build times compared to other tooling [25].

The web-API is implemented relying on the OpenAPI[2] specification and is designed with some parts course-grained to prevent network congestion (e.g., repository update requests), where the API consumers have the requirement to call multiple endpoints for a task. Simple API calls are fine-grained (e.g., assignment tagging) [22]. The Assignments API web controllers have 38 endpoints distributed between 10 controllers. The most extensive controllers are the assignments controller (11 endpoints), and the git repository controller (12 endpoints), and the tag controller (6 endpoints).

The back-end primary modules are organized around the following operational layers: persistence (communicates with PostgreSQL, holds DAOs (Data Access Object), entities, and configurations), domain (holds DTOs, and enums), services (holds business logic between services and interactors), utils (globally accessible tooling by functionality), web-api (the interface between the API and services). The Liquibase module is separated from the others, since it is not part of the API, but evolves alongside it.

[1]`https://cs.ttu.ee/services/aurora/api/swagger-ui/`
[2]`https://swagger.io/specification/`

The developed system's business logic modules act as a black box, exposed to external clients by a public Domain Facade pattern[3]. The built system is agnostic of the context in which it runs, meaning any SIL[4] (Service Interface Layer) could be implemented. During the period of the thesis development, a Web-API SIL[5] is implemented.

6.1.1 Testing

Software testing is the process of verifying computer code functions to match the intended design. Software should be predictable and consistent, presenting no surprises to clients [26].

The application is tested manually and with module (unit) testing. Manual testing is done in both local and production environments. An excellent manual integration test was with the GitLab API to press a button on the repository page, which refreshed all the registry repositories. For example, unit tests cover the cloned repositories file traversal logic, which classifies assignments. During the tests, a mock filesystem is created. Tests have helped discover the differences between file path conventions in windows and Linux-based systems. This saved much manual debugging time. The file system traversal functionality was written prior to the tests, but debugging was done with tests. Prior to every release, the software was manually tested.

6.2 Persistence

The Aurora Assignments API uses a SQL (Structured Query Language)-based open-source relational database system called PostgreSQL[6] as the persistence technology. Persisted data objects are defined with classes called entities shown in appendix 2. The transition from class objects to database rows is done with an ORM (Object Relational Mapper) called Hibernate[7], which implements JPA (Java Persistence API).

The API uses Git as a secondary persistence medium and downstream. Maintaining assignment data integrity is difficult due to every part of a file within a Git repository having the possibility of being changed. For example, file and repository renaming, path changes, and content updates on the registry update would create a new assignment reference. The solution is to traverse the file system and map through the file path. The

[3]https://www.geeksforgeeks.org/facade-method-python-design-patterns/

[4]https://www.researchgate.net/publication/267690101_Implementation_of_MDA_Method_into_SOA_Environment_for_Enterprise_Integration

[5]https://www.youtube.com/watch?v=3JAwNtNthbU

[6]https://www.postgresql.org/

[7]https://hibernate.org/

deletion of files is rarely done, and validations occur prior to that. A repository change event that causes a duplicate assignment can be mapped back to the original assignment. The mapping feature is yet to be implemented in Aurora.

6.2.1 Schema

The Assignments API database is named aurora_db_s1, and the content is inside the public schema, which has 16 entity tables, seen in Figure 7. Triggers were created to check for duplicate inserts on the similar_repository and tagging_job tables. The database tables are named using a singular form to convey the power scale relations between tables clearly. Understanding power scales clearer makes table joining easier to conduct. Primary keys are prefixed with pk, foreign keys - fk, unique constraints - uk, alternative keys - ak, domain values - d, and views - v. Two views were created, the assignments view seen in appendix 5, and the repository view.

The API began with testing JPA and Spring generated schema approaches. In the beginning the JPA approach was chosen, but as complexity grew, JPA was replaced with an SQL scripting-based evolutionary tool. For evolving the relational database schema with SQL, the considered options were Liquibase[8] and Flyway[9]. Liquibase was favored since it offers more in the free tier, and the author has more work experience with the tool.

6.2.2 Continuous Database Integration and Delivery

Data incidents are more complex to recover than code incidents. To prevent data losses, machines should do the database evolution and deployment instead of humans. There are two methodologies of doing this: Migrations and DSC (Desired State Config) [27].

The Assignments API database is evolved with Liquibase migration scripts. This achieves manual control over the created scripts since in DSC, migration scripts are generated automatically [27]. The Liquibase tooling is used locally for development, migrations are triggered by a custom made Gradle command dbUpdate. Production schema evolution is conducted by the API CD pipeline, which spins up a Docker container and runs the update command against the production Postgres Docker instance. Liquibase migration scripts are stored and written in the respective API project since the API and the data model often evolve together. An example of a migration script can be seen in appendix 3.

[8]`https://www.liquibase.org/`

[9]`https://flywaydb.org/`

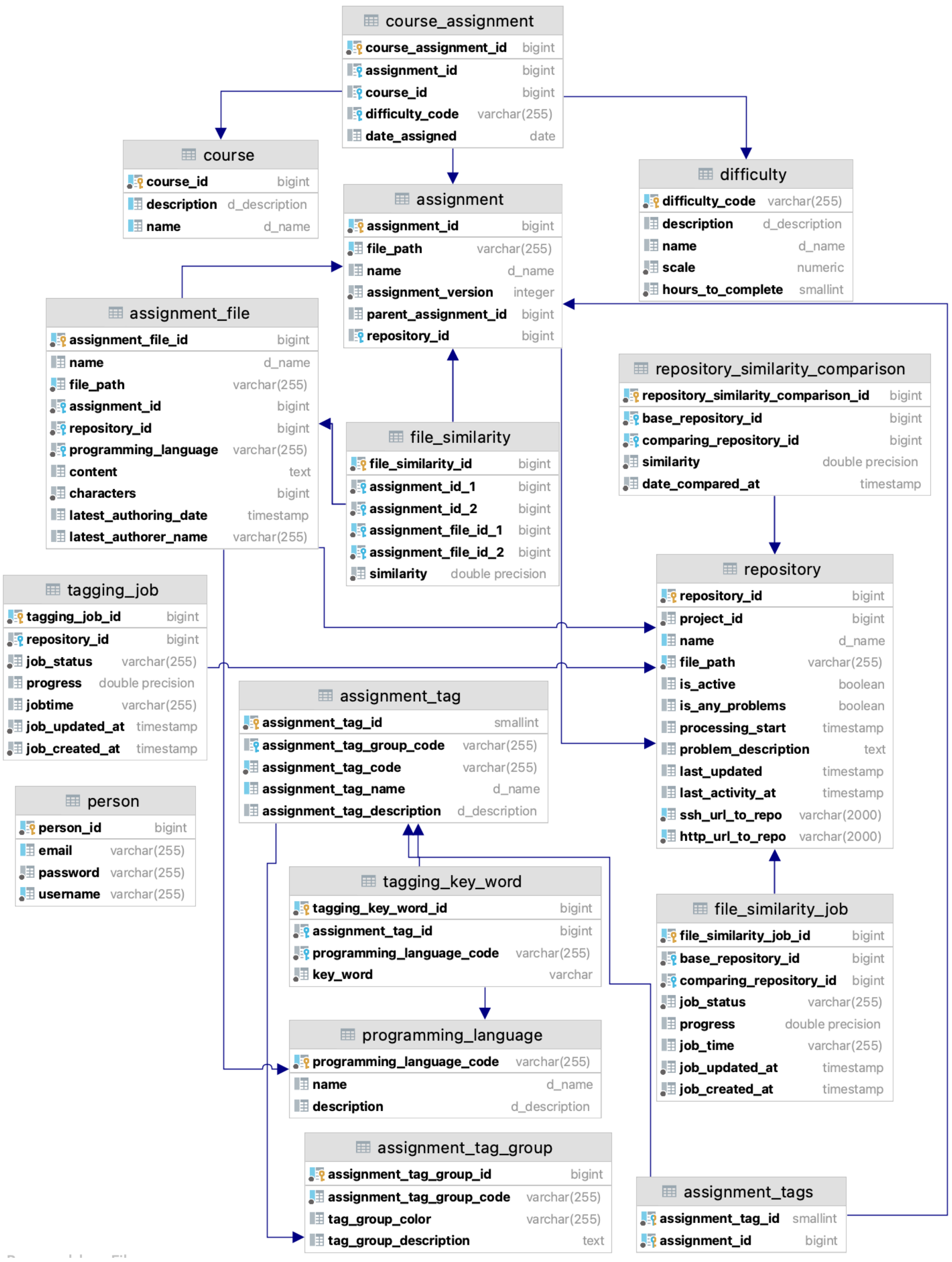

course_assignment
course_assignment_id bigint
assignment_id bigint
course_id bigint
difficulty_code varchar(255)
date_assigned date
course
course_id bigint
description d_description
name d_name
difficulty
difficulty_code varchar(255)
description d_description
name d_name
scale numeric
hours_to_complete smallint
assignment
assignment_id bigint
file_path varchar(255)
name d_name
assignment_version integer
parent_assignment_id bigint
repository_id bigint
assignment_file
assignment_file_id bigint
name d_name
file_path varchar(255)
assignment_id bigint
repository_id bigint
programming_language varchar(255)
content text
characters bigint
latest_authoring_date timestamp
latest_authorer_name varchar(255)
repository_similarity_comparison
repository_similarity_comparison_id bigint
base_repository_id bigint
comparing_repository_id bigint
similarity double precision
date_compared_at timestamp
file_similarity
file_similarity_id bigint
assignment_id_1 bigint
assignment_id_2 bigint
assignment_file_id_1 bigint
assignment_file_id_2 bigint
similarity double precision
tagging_job
tagging_job_id bigint
repository_id bigint
job_status varchar(255)
progress double precision
jobtime varchar(255)
job_updated_at timestamp
job_created_at timestamp
repository
repository_id bigint
project_id bigint
name d_name
file_path varchar(255)
is_active boolean
is_any_problems boolean
processing_start timestamp
problem_description text
last_updated timestamp
last_activity_at timestamp
ssh_url_to_repo varchar(2000)
http_url_to_repo varchar(2000)
assignment_tag
assignment_tag_id smallint
assignment_tag_group_code varchar(255)
assignment_tag_code varchar(255)
assignment_tag_name d_name
assignment_tag_description d_description
person
person_id bigint
email varchar(255)
password varchar(255)
username varchar(255)
tagging_key_word
tagging_key_word_id bigint
assignment_tag_id bigint
programming_language_code varchar(255)
key_word varchar
file_similarity_job
file_similarity_job_id bigint
base_repository_id bigint
comparing_repository_id bigint
job_status varchar(255)
progress double precision
job_time varchar(255)
job_updated_at timestamp
job_created_at timestamp
programming_language
programming_language_code varchar(255)
name d_name
description d_description
assignment_tag_group
assignment_tag_group_id bigint
assignment_tag_group_code varchar(255)
tag_group_color varchar(255)
tag_group_description text
assignment_tags
assignment_tag_id smallint
assignment_id bigint

Figure 7. Database entity diagram.

7. Front-end

It is the architecture's presentation layer, which exposes a web UI to users. Compared to a CLI (Command Line Interface), it is more complex to build but enables a better UX (User Experience) while interacting with the functionality provided by the application's back-end.

7.1 Web user interface

The web UI is rendered on the client-side to reduce server load, keep the web page more dynamic, and decrease the client and server coupling. The web page content is rendered through JavaScript, HTML (HyperText Markup Language), and CSS (Cascading Style Sheets). The development uses TypeScript, SCSS (Sassy CSS), and HTML.

For enabling PWA (Progressive Web App) features and improving performance, a service worker is implemented. It is JavaScript running separately from the main browser thread, intercepting network requests. It caches or retrieves resources from the cache, enables the application to control network requests, cache those requests to improve performance and provide offline access to cached content [28].

7.2 Styling

The CSS development uses the SASS (Syntactically Awesome Style Sheets)[1] preprocessor, with SCSS syntax and adaptations from the BEM (Block Element Modifier) naming convention for writing custom styles. It is a styling language for reducing repetition and maintainability challenges of traditional CSS, allowing to scale styles faster and more efficiently to write reusable styles from scratch. SASS uses variables, nesting, mixins, and functions to write reusable styles [29].

At the beginning of the UI development, a comprehensive SCSS architecture seen in appendix 13 was designed to be agnostic to the used development framework and external styling libraries. The aim was to write all application styles without third-party tooling, using only SASS. After a sprint, it was realized that doing so would leave little time for

[1] `https://sass-lang.com/`

developing higher priority features. This resulted in the adaptation of Bootstrap 5[2], and Angular Material component library[3]. The custom styles have higher precedence over third-party libraries, and the component styles have the highest priority.

7.3 Angular

Angular is used to write the SPA (Single Page Application). It was chosen since the author has prior experience using the platform. Angular[4] is a development platform built on TypeScript, which is a superset of JavaScript. It enables the separation of concerns with an MVC (Model View Controller) architecture. As a platform, Angular includes a component-based framework, a well-integrated collection of libraries, and a suite of developer tools [30]. In an Angular production build, TypeScript, SCSS, and HTML will be built into HTML, JavaScript, and CSS.

The NgRx[5] framework is used for managing the global application state. Initially all API requests were routed through state management, but due to management overhead this was discontinued. In future developments, it is planned to keep user preferences in the global state.

The front-end project was the first where active development began. Since the API was not ready from the start, the beginning requests were mocked with `json-server`, Node.js, and `faker`, running on port 3000. There are three environment configurations: development, production, and mock, which configure the used API paths, versioning, logging, and allowed authorization methods.

The application `app` directory is structured into `core`, `dto`, `features`, and `shared` modules. The `core` spans over the entire application and holds global enums, constants, factories, alerting, authorization, language selection, and WebSocket services. Multilingual support is built-in, but only English is supported. `dto` content matches the APIs DTO objects. `features` hold the business components, and each sub-folder here is a separate page, which has its own routing module. An example of assignments routing module is given in Figure 8.

[2]`https://getbootstrap.com/docs/5.0/getting-started/introduction/`
[3]`https://material.angular.io/`
[4]`https://angular.io/`
[5]`https://ngrx.io/`

```
{
    path: 'assignments',
    canActivate: [AuthGuard],
    loadChildren: () => {
        // @ts-ignore
        return import('./features/assignments/assignments.module')
        .then(m => m.AssignmentsModule);
    },
},
```

Figure 8. Assignments module routing in the base rooting module

7.4 Node.js

Node.js[6] is an asynchronous event-driven JavaScript runtime, and it is designed for building scalable network applications. HTTP (HyperText Transport Protocol) is a first-class citizen in Node.js, designed with streaming and low latency in mind, which makes it well suited for the foundation of a web library or framework [31].

NPM[7] (Node Package Manager) is the default package manager for Node.js, which enables sharing packaged modules of code. The NPM registry is a public collection of packages of open-source code for Node.js [32].

Node.js and NPM are used for the web UI project development. They do not run production, but are used in building the production package.

[6]https://nodejs.org/en/
[7]https://www.npmjs.com/

8. Core Features

These are the most significant features implemented in the registry in terms of time, complexity, and importance. The application has three primary pages: assignments search, repository management, and tag management.

Currently in progress are the following: dashboard, administration, automatic tagging, and settings. The dashboard, settings, and administration are waiting for the user system implementation. Browsing between the pages is close to instant. Many requests are not made before, for example opening specific tabs. This saves resources and keeps browsing as seamless as possible.

8.1 Repository Management

This is one of the core administrative features operating with Git. The UI is seen in appendix 9. Aurora sees and uses the available repositories at the moment only through the GitLab API and via a Personal Access Token provided to the Aurora GitLab user `aurora-user` account. When `aurora-user` is added into a repository with cloning/pulling permissions, it becomes visible for the registry, and in the production environment, the repository becomes available for cloning into the registry's file system. The API enables to push changes up to repositories, but this functionality is not yet implemented by the API-consuming UI. This feature will be fully-added after the user permission system is implemented so that git logs show the logged-in user as the file changer. Only administrators will be allowed to change files under the `aurora-user`.

There is an issue with repository persistence in production, which is related to entity recursion. This is why repository update requests are rate limited to 1 per 30 minutes. The recursion causes the API to crash, but Swarm redeploys an instance, occurring in downtime of around 3 minutes. This will be fixed outside of the thesis scope since the problem discovery has already taken over 10h.

Git systems offer file authoring (blame) annotations. The Assignments API uses the GitLab APIs to extract additional data for assignments. The GitLab API handling blame requests only returns a single blame response per request, meaning that it takes around 5 seconds per file to request and persist the record. The blame annotation persistence accumulative time is linear depending on the number of files. Initially, blame was added upon a repository update. Currently, it is updated manually through an API call, and in the following sprints, it will be made a cron job.

8.2 Assignment Search

The search UI has gone through 2 remodels thanks to user feedback. The current visual is seen in appendix 8. Assignment searching is the core functionality that provides the most significant business value. Many features are built alongside to help with searching. The query response is paginated in the Assignments API to 10, 25, 50, and 100 results, defaulting to 10. The initial load without query criterion search ignores file content search for performance reasons.

After a more detailed search-criterion is given, and the database has returned a query result, the API goes into a multithreaded mode and splits the queried results pairing between the available threads in the pool, which conduct file snippet extraction, analysis, and response mapping. Since the biggest file in the database has 4 217 441 characters, the analyzable files are kept a standard deviation over the average character size in the system, resulting in the limit of 40 000 characters per file, excluding Text, CSV, and JSON files. Information about file sizes is stored in the database alongside the files.

In the search result, the assignment file snippets matching the search criteria are displayed, with three lines before and after the found result. Found snippets count is displayed in the expanded assignments navigation header and the root node in parentheses. In the case of similar files, also a tag is displayed in the assignment root, with the similar assignments count shown in the expanded header. In the similar file listing, the similarity percentage is displayed with the same file path format as in the root assignment files.

Initially, the assignment search content was a single table query executed via a JPA repository. As features were added, a view was created, which replaced the table query. As additional features were added, the search query performance started to suffer. Initial page loading took close to 10s, following queries took 5s, which is below the requirement, but realistically very slow. At this point, a custom SQL query service was implemented and integrated with the view. It created the most optimal query depending on the given criterion from the request. The assignment search base view can be viewed in appendix 5.

8.3 Assignment Classification

The default assignments classification is one of the functionalities that is covered in unit tests since file traversal logic may differ depending on the host system file path structuring methods. By default, Aurora searches in the cloned repositories file tree leaf folders, which have at least a single file inside them. The traversal excludes the Git and Python system files. A set of keywords written in the traversal service act as the parent for the following assignment files. Such keywords are, for example, KT, EX, PR. The keywords are searched for in the file path with Regular expression.

To make the assignment classification through the filesystem more customizable, it is possible to add metadata files into Git repositories named `aurora.json` which the registry will take into account while traversing the repository file tree. This is primarily added to enable overriding of the system default assignment classification algorithm. Currently, the metadata is not persisted and only taken into account if the type of KT is described. The metadata will be used as a second persistence solution for tags, descriptions, and custom properties in upcoming sprints.

8.4 Assignment Similarity

Since administrative staff often copy assignments from existing repositories, there are duplicates in the system.

Aurora implements a similarity scanning feature to cross-compare all the assignment file contents between repositories, comparing two repositories files at a time. The comparison uses Levenshtein distance computation and persists file relations that are over 0.55 in similarity. Comparable file contents are viewed as two strings, a and b, and the Levenshtein distance[1] between them is given by $lev_{a,b}(|a|, |b|)$. Since the comparison can take an extended amount of time (10 000 comparisons/1h), it is run agnostic to the UI.

This is achieved through implementing a queue in the form of a database table. There is a separate table for storing the queued jobs and the comparison results. The trouble with this is determining when a comparison job has encountered errors or is stuck. The running comparison job is updated over 3 minutes. If the job has not been completed and 10 minutes have passed from the latest update, it is considered no longer running. During development, around 250000 file pairs have been compared in 18h 12m.

[1]https://www.cuelogic.com/blog/the-levenshtein-algorithm

8.5 Tagging

The tagging feature has gone through three remodeling iterations. The logic in the UI is complex and has been completely remade twice. In addition to tags being a search query input, they also enable assignment tag editing. It started out as a single assignment edit, then to speed up tagging - support for multiple assignment edits was added. Tags provide quick context about the type of assignments.

Tags have a management page seen in appendix 10 where users can create or edit tag groups with custom colorings by providing a correct color hex value.

A feature in the development roadmap is to implement automatic tagging on assignments based on custom user-defined input. The user-given configuration will be persisted in a keyword table in Postgres. The API side is complete, but the UI is yet to be created.

9. Results analysis

By May 2021 the registry has been used for a few months. During development, the production environment managed 8 GitLab repositories, with 523 classified assignments and 1335 files (125 Restructured Text, 673 Python, 206 Java, 255 Markdown, 49 Text, 6 JSON, 18 CSV, 2 not classified). 250000 similar files were compared, and 2000 tags were added.

The initial assignment search takes a bit longer than the average load speed (3-6 seconds), if no prior searches have been made. This Exceeds the required request-response ceiling time of 10s. This is because the database connection pool needs to provide a connection which takes more time. Following requests were within the 10s response bounds, with up to 100 assignments per query.

Since the application runs in Docker Swarm, migration durations between VMs are kept under the 2h requirement.

10. Summary

The goal of the work was to create an assignment registry for teachers and teaching assistants for managing assignments. For the registry development an analysis was made, methodologies and disciplines were adopted, requirements were continuously gathered and modified, and the application was iteratively developed with the client.

The created registry is a client-server application. The API is written in Java 11 with the Spring Boot framework. The Postgres database system and Git repositories were used as persistence mediums. The web UI is written in Angular 11. The application is running inside a single node cluster, Docker Swarm. Thanks to the adopted DevOps culture, changes were easily deployed through the GitLab CI/CD pipelines. The registry core features are the following:

1. Repository management;
2. Assignment search;
3. Assignment classification;
4. Assignment similarity;
5. Tagging.

The web UI[1], Assignments API[2], and monitoring[3] are accessible from the TalTech page[4].

The application is designed to be extensible, and an overview of the approaches, technologies, and experiences were given in the thesis. The following are additional objectives to reach that were left out of the thesis scope:

1. Programming language detection from the assignment file content;
2. User Authentication and authorization with uniid;
3. Administration tooling;
4. Possibility to test assignments with the Testing service Arete;
5. Increased registry proxy resilience to upstream connection losses.

[1]`cs.ttu.ee/services/aurora/front/`
[2]`cs.ttu.ee/services/aurora/api/`
[3]`cs.ttu.ee/services/aurora/portainer/`
[4]`cs.ttu.ee`

Bibliography

[1] Cambridge University Press. *Cambridge Dictionary*. Apr. 7, 2021. URL: https://dictionary.cambridge.org/dictionary/english/.

[2] HackerRank. *About Us*. May 23, 2021. URL: https://www.hackerrank.com/about-us/.

[3] Kirill Denisov. *Development of a Code Writing Practice Platform*. 2019.

[4] Aaron Bloomfield and Borja Sotomayor. “A Programming Contest Strategy Guide”. In: *SIGCSE* (2016).

[5] Leetcode. *Overview*. Apr. 20, 2021. URL: https://www.linkedin.com/company/leet-code/about/.

[6] Exercism. *About Exercism*. Apr. 24, 2021. URL: https://exercism.io/about.

[7] Axelsoft. *Agile Project Management: Best Practices and Methodologies*. Apr. 16, 2021. URL: https://www.altexsoft.com/whitepapers/agile-project-management-best-practices-and-methodologies/.

[8] *Manifesto for Agile Software Development*. Apr. 15, 2021. URL: https://agilemanifesto.org/.

[9] Amjad Hudaib et al. “Requirements Prioritization Techniques Comparison”. In: *Modern Applied Science* 12 (Jan. 2018). DOI: 10.5539/mas.v12n2p62.

[10] Hot PMO. *MoSCoW or Kano Models – how do you prioritize?* Apr. 17, 2021. URL: https://www.hotpmo.com/management-models/moscow-kano-prioritize/.

[11] Martin Fowler. *ContinuousDelivery*. May 30, 2021. URL: https://martinfowler.com/bliki/ContinuousDelivery.html.

[12] Ken Mugrage. *It’s Not Continuous Delivery If You Can’t Deploy Right Now*. Mar. 23, 2021. URL: https://www.youtube.com/watch?v=po712VIZZ7M.

[13] Pete Hodgson. *Feature Toggles (aka Feature Flags)*. Apr. 20, 2021. URL: https://martinfowler.com/articles/feature-toggles.html.

[14] International Institute of Business Analysis. *BABOK v3: A GUIDE TO THE BUSINESS ANALYSIS*. 2015.

[15] Jonas Eckhardt, Andreas Vogelsang, and Daniel Méndez Fernández. “Are "Non-Functional" Requirements Really Non-Functional? An Investigation of Non-Functional Requirements in Practice”. In: *Proceedings of the 38th International Conference on Software Engineering*. ICSE ’16. Austin, Texas: Association for Computing Machinery, 2016, pp. 832–842. ISBN: 9781450339001. DOI: `10.1145/2884781.2884788`. URL: `https://doi.org/10.1145/2884781.2884788`.

[16] M. Broy. “Rethinking Nonfunctional Software Requirements”. In: *Computer* 48.5 (2015), pp. 96–99. DOI: `10.1109/MC.2015.139`.

[17] J. Eckhardt, D. Mendez Fernandez, and A. Vogelsang. “How to Specify Non-Functional Requirements to Support Seamless Modeling? A Study Design and Preliminary Results”. In: *2015 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. 2015, pp. 1–4. DOI: `10.1109/ESEM.2015.7321200`.

[18] S. Robertson J. Robertson. *Volere Requirements Specification Template*. 2016.

[19] Amit M Potdar et al. “Performance Evaluation of Docker Container and Virtual Machine”. In: *Procedia Computer Science* 171 (2020). Third International Conference on Computing and Network Communications (CoCoNet’19), pp. 1419–1428. ISSN: 1877-0509. DOI: `https://doi.org/10.1016/j.procs.2020.04.152`. URL: `https://www.sciencedirect.com/science/article/pii/S1877050920311315`.

[20] Docker. *Docker overview*. Feb. 25, 2021. URL: `https://docs.docker.com/get-started/overview/`.

[21] Frank Buschmann, Kevlin Henney, and Douglas C. Schmidt. *Pattern-Oriented Software Architecture, Volume 5: On Patterns and Pattern Languages*. Chichester, UK: Wiley, 2007. ISBN: 978-0-471-48648-0. URL: `https://www.safaribooksonline.com/library/view/pattern-oriented-software/9780471486480/`.

[22] Matthias Biehl. *RESTful API Design*. May 22, 2021.

[23] James Gosling et al. *The Java Language Specification, Java SE 16 Edition*. Feb. 12, 2021.

[24] Craig Walls and Andrew Glover. *Spring Boot in action*. Manning Publications Co., 2016.

[25] Gradle. *Gradle vs Maven: Performance Comparison*. May 6, 2021. URL: `https://gradle.org/gradle-vs-maven-performance/`.

[26] Glenford J. Myers, Corey Sandler, and Tom Badgett. *The Art of Software Testing*. 3rd. Wiley Publishing, 2011. ISBN: 1118031962.

[27] Jimmy Bogard NDC Conferences. *Continuous Integration and Delivery for Databases*. Feb. 25, 2021. URL: https://www.youtube.com/watch?v=HdXDSjWe2Q8.

[28] Goolge. *Introduction to Service Worker*. Apr. 27, 2021. URL: https://developers.google.com/web/ilt/pwa/introduction-to-service-worker.

[29] Jane O'Donnell. "SASS (Syntactically Awesome Style Sheets)". In: *J. Comput. Sci. Coll.* 34.4 (Apr. 2019), pp. 101–102. ISSN: 1937-4771.

[30] Angular. *What is Angular?* Apr. 27, 2021. URL: https://angular.io/guide/what-is-angular.

[31] Nodejs. *About Node.js*. Apr. 27, 2021. URL: https://nodejs.org/en/about/.

[32] Npmjs. *About npm*. Apr. 27, 2021. URL: https://www.npmjs.com/about.

[33] https://www.career.pm/briefings/product-prioritization-techniques. *20 Product Prioritization Techniques - A Map and Guided Tour*. Mar. 25, 2021.

[34] GitLab. *Directed Acyclic Graph*. May 3, 2021. URL: https://docs.gitlab.com/13.11/ee/ci/directed_acyclic_graph/.

[35] XP123. *INVEST in Good Stories, and SMART Tasks*. May 5, 2021. URL: https://xp123.com/articles/invest-in-good-stories-and-smart-tasks/.

Appendix 1 - Non-exclusive license for reproduction and publication of a graduation thesis[5]

[5]The non-exclusive license is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive license, the non-exclusive license shall not be valid for the period.

Appendix 2 - Spring entity

```
package ee.taltech.aurora.persistence.entity.assignment;

import io.swagger.annotations.ApiModel;
import lombok.*;
import org.hibernate.annotations.DynamicUpdate;

import javax.persistence.*;
import java.io.Serializable;

@ApiModel(value = "Assignment",
          description = "The assignments that we get")
@Builder
@Data
@Entity
@NoArgsConstructor
@AllArgsConstructor
@DynamicUpdate
@Table(name = "assignment",
   uniqueConstraints=@UniqueConstraint(columnNames = {"file_path"}))
@SequenceGenerator(name = "assignment_assignment_id_seq", sequenceName
   = "assignment_assignment_id_seq", allocationSize = 1)
public class Assignment implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(generator="assignment_assignment_id_seq",
       strategy=GenerationType.SEQUENCE)
    @Column(name = "assignment_id", unique = true)
    private Long assignmentId;

    @Column(name = "name")
    private String name;

    @Column(name = "repository_id")
    private Long repositoryId;

    @Column(name = "file_path", unique = true, nullable = false)
    private String filePath;

    @Column(name = "assignment_version", nullable = false)
    private Integer assignmentVersion;

    @Column(name = "parent_assignment_id")
    private Long parentAssignmentId;
}
```

Appendix 3 - Liquibase migration script

```sql
--liquibase formatted sql
--changeset oskar:create-table__tagging_job context:prod
--runOnChange:false
--logicalFilePath:db/07-changes/changes/2021/2021_05_14/
--  03-ddl-create-table__tagging_key_word.sql

CREATE TABLE IF NOT EXISTS tagging_key_word
(
    tagging_key_word_id BIGSERIAL NOT NULL,
    assignment_tag_id BIGINT NOT NULL
        CONSTRAINT fk_tagging_key_word_assignment_tag_assignment_tag_id
            REFERENCES assignment_tag(assignment_tag_id)
            ON UPDATE CASCADE ON DELETE SET NULL,
    programming_language_code VARCHAR(255) NOT NULL
        CONSTRAINT fk_tagging_key_word_prog_lang_prog_lang_code
            REFERENCES programming_language(programming_language_code)
            ON UPDATE CASCADE ON DELETE SET NULL,
    key_word VARCHAR(255) NOT NULL
);

CREATE UNIQUE index tagging_key_word_tagging_key_word_id_uindex
    ON tagging_key_word (tagging_key_word_id);

ALTER TABLE tagging_key_word
    ADD CONSTRAINT pk_tagging_key_word
        PRIMARY KEY (tagging_key_word_id);
```

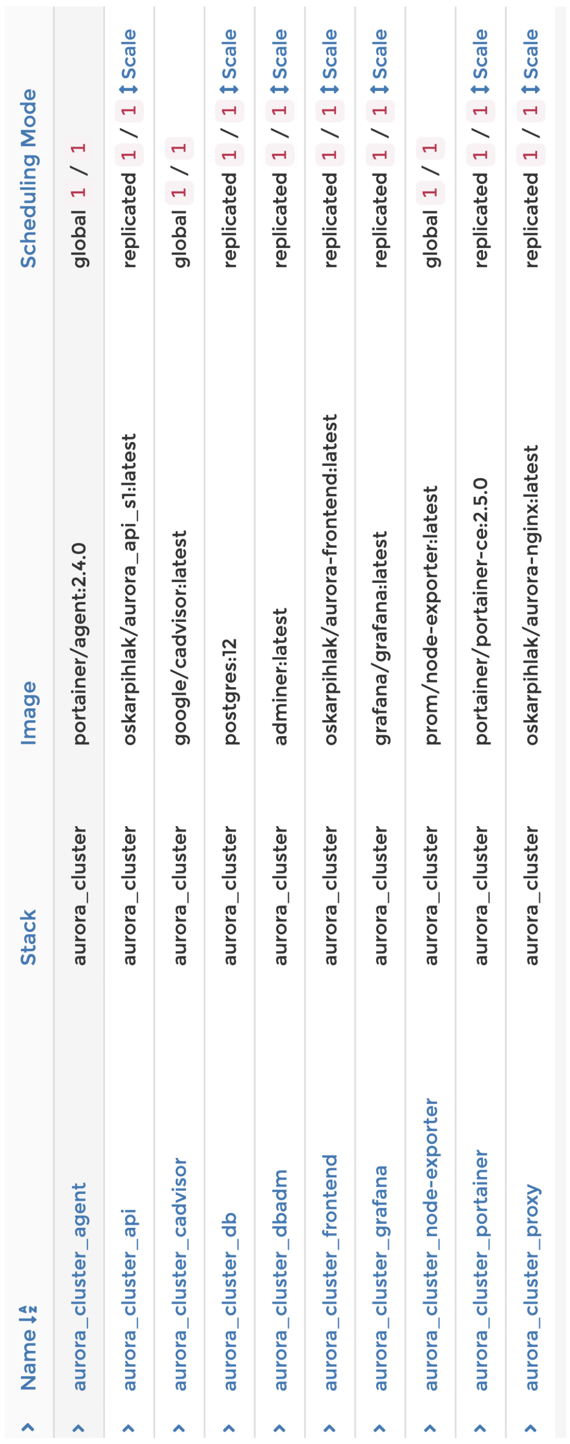
Appendix 4 - Operations project Docker stack file

```yaml
# docker stack deploy -c stack-proxy.yaml aurora_cluster
# docker stack rm traefik
# docker stack deploy -c stack-proxy.yaml aurora_cluster & docker stac

version: "3.6" # important, so that we can name our network

services:
  proxy:
    image: oskarpihlak/aurora-nginx:latest
    networks:
      - aurora-network
    ports:
      - "8080:8080"
    deploy:
      mode: replicated
      replicas: 1
      # service resource management
      resources:
        # Hard limit - Docker does not allow to allocate more
        limits:
          cpus: '0.5'
          memory: 1024M
        # Soft limit - Docker makes best effort to return to it
        reservations:
          cpus: '0.5'
          memory: 512M
      # service restart policy
      restart_policy:
        condition: on-failure
        delay: 5s
        max_attempts: 3
        window: 120s
      # service update configuration
      update_config:
        parallelism: 1
        delay: 10s
        failure_action: continue
        monitor: 60s
        max_failure_ratio: 0.3

networks:
  aurora-network:
      name: aurora-network
      driver: overlay
      attachable: true
```

Appendix 5 - Assignments SQL view

```sql
SELECT * FROM ( SELECT a.assignment_id AS assignment_id, a.file_path,
(SELECT COALESCE(t.tags, '[]'::jsonb)
FROM (SELECT (JSONB_AGG(DISTINCT JSONB_BUILD_OBJECT(
        'assignmentTagId', at.assignment_tag_id,
        'assignmentTagGroupCode', at.assignment_tag_group_code,
        'assignmentTagCode', at.assignment_tag_code,
        'assignmentTagName', at.assignment_tag_name,
        'assignmentTagDescription', at.assignment_tag_description,
        'tagGroupColor', atg.tag_group_color
    ))) AS tags
        FROM assignment_tags ats
LEFT JOIN assignment_tag at ON at.assignment_tag_id =
   ats.assignment_tag_id
LEFT JOIN assignment_tag_group atg ON at.assignment_tag_group_code =
   atg.assignment_tag_group_code
        WHERE ats.assignment_id = a.assignment_id) t) AS tags,
REPLACE(rl.http_url_to_repo, '.git', '/') || '-/tree/master'
|| REPLACE(REPLACE(a.file_path, './repositories/', ''),
        rl.file_path, '') AS repository_https_reference,
REPLACE(a.file_path, '/', ' / ') AS name,
(SELECT COALESCE(t.p_lang, '[]'::jsonb)
FROM (SELECT (JSONB_AGG(JSONB_BUILD_OBJECT('programmingLanguage',
   af2.programming_language, 'characters', af2.characters) ORDER BY
   af2.characters DESC)) AS p_lang
        FROM (SELECT af1.programming_language, SUM(af1.characters) AS
            characters FROM assignment_file af1 WHERE
            af1.assignment_id = a.assignment_id GROUP BY
            af1.programming_language) af2) t) AS programming_languages,
(SELECT COALESCE(t.author, '{}'::jsonb)
FROM (SELECT (JSONB_BUILD_OBJECT(
        'authoredDate', af.latest_authoring_date,
        'authorName', af.latest_authorer_name
)) AS author
    FROM assignment_file af WHERE af.assignment_id = a.assignment_id
    GROUP BY af.latest_authoring_date, af.latest_authorer_name
    ORDER BY af.latest_authoring_date DESC LIMIT 1) t) AS authoring,
(SELECT ARRAY_AGG(DISTINCT CASE WHEN (a.assignment_id = fs
.assignment_id_1) THEN fs.assignment_id_2 ELSE fs.assignment_id_1 END)
FROM assignment_file af
INNER JOIN file_similarity fs ON af.assignment_file_id =
   fs.assignment_file_id_1 OR af.assignment_file_id =
   fs.assignment_file_id_2
WHERE af.assignment_id = a.assignment_id) AS similar_assignment_ids
FROM assignment a
INNER JOIN repository rl ON rl.repository_id = a.repository_id
GROUP BY a.assignment_id, rl.http_url_to_repo, rl.file_path
) t3 ORDER BY t3.authoring ->> 'authoredDate' DESC NULLS LAST;
```

Appendix 6 - Ubuntu node with containers



Name	Stack	Image	Scheduling Mode
aurora_cluster_agent	aurora_cluster	portainer/agent:2.4.0	global 1 / 1
aurora_cluster_api	aurora_cluster	oskarpihlak/aurora_api_s1:latest	replicated 1 / 1 Scale
aurora_cluster_cadvisor	aurora_cluster	google/cadvisor:latest	global 1 / 1
aurora_cluster_db	aurora_cluster	postgres:12	replicated 1 / 1 Scale
aurora_cluster_dbadm	aurora_cluster	adminer:latest	replicated 1 / 1 Scale
aurora_cluster_frontend	aurora_cluster	oskarpihlak/aurora-frontend:latest	replicated 1 / 1 Scale
aurora_cluster_grafana	aurora_cluster	grafana/grafana:latest	replicated 1 / 1 Scale
aurora_cluster_node-exporter	aurora_cluster	prom/node-exporter:latest	global 1 / 1
aurora_cluster_portainer	aurora_cluster	portainer/portainer-ce:2.5.0	replicated 1 / 1 Scale
aurora_cluster_proxy	aurora_cluster	oskarpihlak/aurora-nginx:latest	replicated 1 / 1 Scale

Appendix 7 - The periodic table of Prioritization Techniques

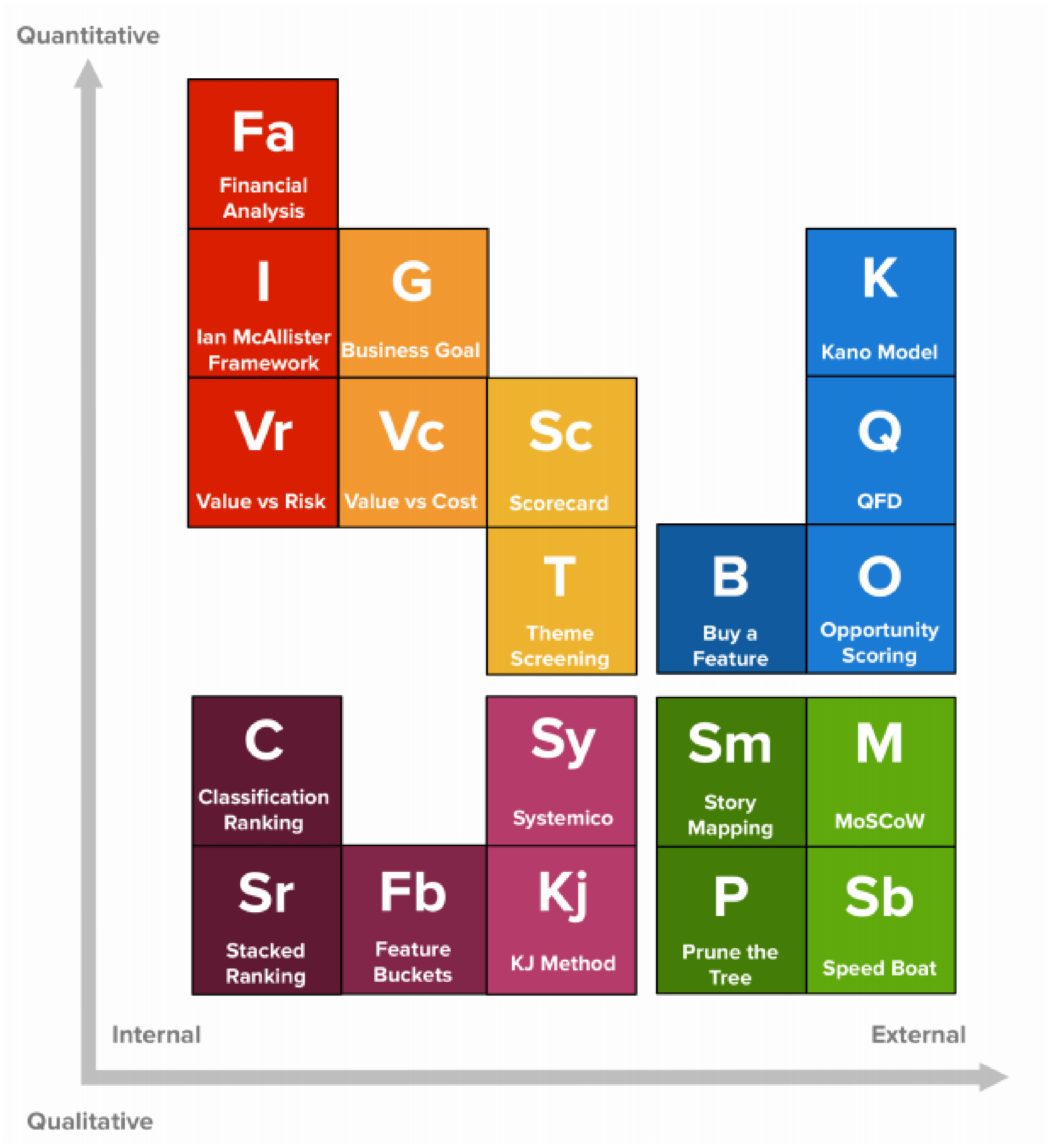
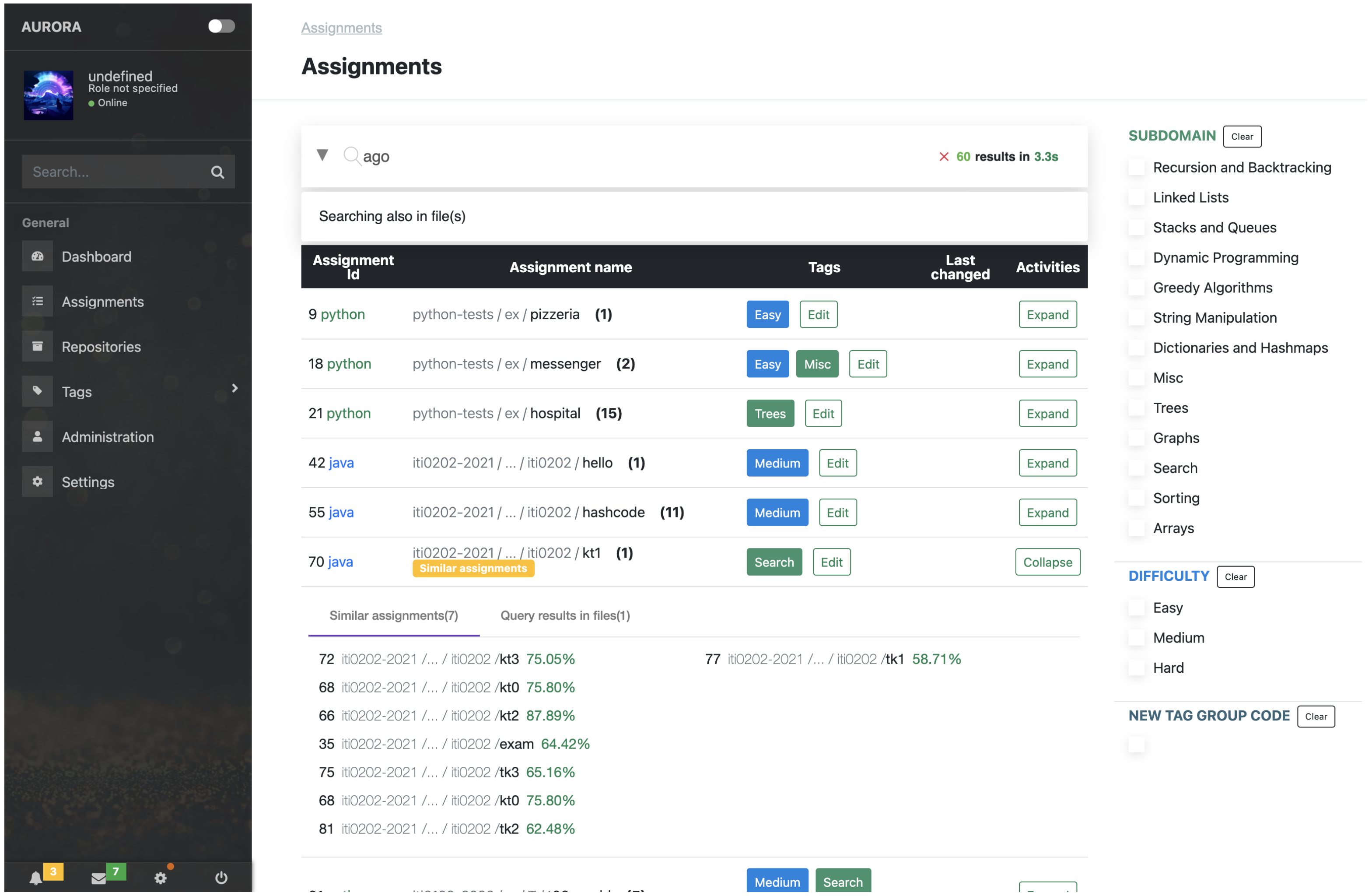


Figure 7-1 Prioritizations table [33].

Appendix 8 - Assignment search page



Assignments

ago

60 results in 3.3s

Searching also in file(s)

Assignment Id	Assignment name	Tags	Last changed	Activities
9 python	python-tests / ex / pizzeria (1)	Easy Edit		Expand
18 python	python-tests / ex / messenger (2)	Easy Misc Edit		Expand
21 python	python-tests / ex / hospital (15)	Trees Edit		Expand
42 java	iti0202-2021 / ... / iti0202 / hello (1)	Medium Edit		Expand
55 java	iti0202-2021 / ... / iti0202 / hashcode (11)	Medium Edit		Expand
70 java	iti0202-2021 / ... / iti0202 / kt1 (1) Similar assignments	Search Edit		Collapse

Similar assignments(7) Query results in files(1)

72 iti0202-2021 /... / iti0202 /kt3 75.05%
68 iti0202-2021 /... / iti0202 /kt0 75.80%
66 iti0202-2021 /... / iti0202 /kt2 87.89%
35 iti0202-2021 /... / iti0202 /exam 64.42%
75 iti0202-2021 /... / iti0202 /tk3 65.16%
68 iti0202-2021 /... / iti0202 /kt0 75.80%
81 iti0202-2021 /... / iti0202 /tk2 62.48%
77 iti0202-2021 /... / iti0202 /tk1 58.71%

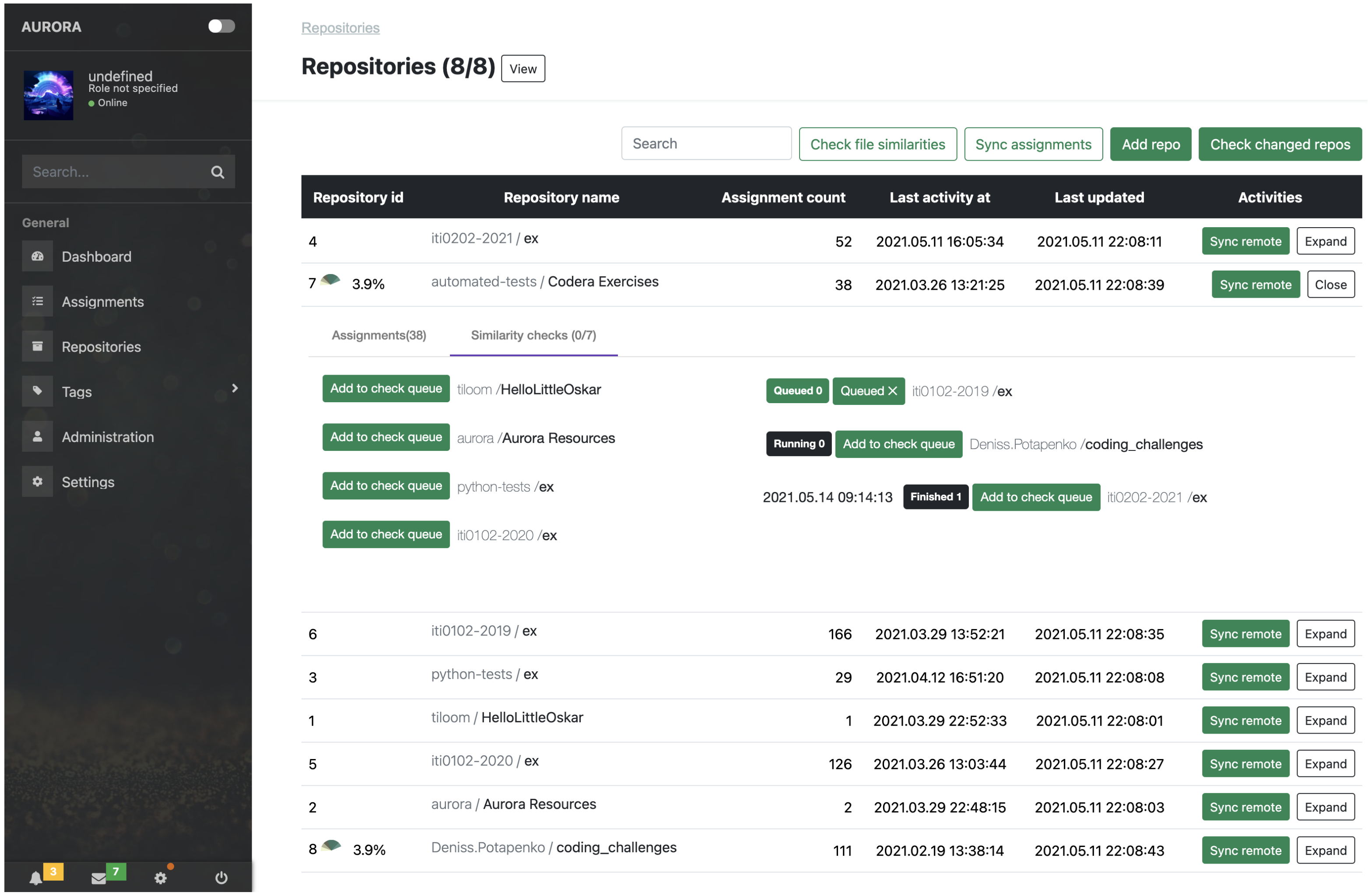
Medium Search

SUBDOMAIN Clear
- Recursion and Backtracking
- Linked Lists
- Stacks and Queues
- Dynamic Programming
- Greedy Algorithms
- String Manipulation
- Dictionaries and Hashmaps
- Misc
- Trees
- Graphs
- Search
- Sorting
- Arrays

DIFFICULTY Clear
- Easy
- Medium
- Hard

NEW TAG GROUP CODE Clear

Appendix 9 - Repository page



Repositories (8/8)

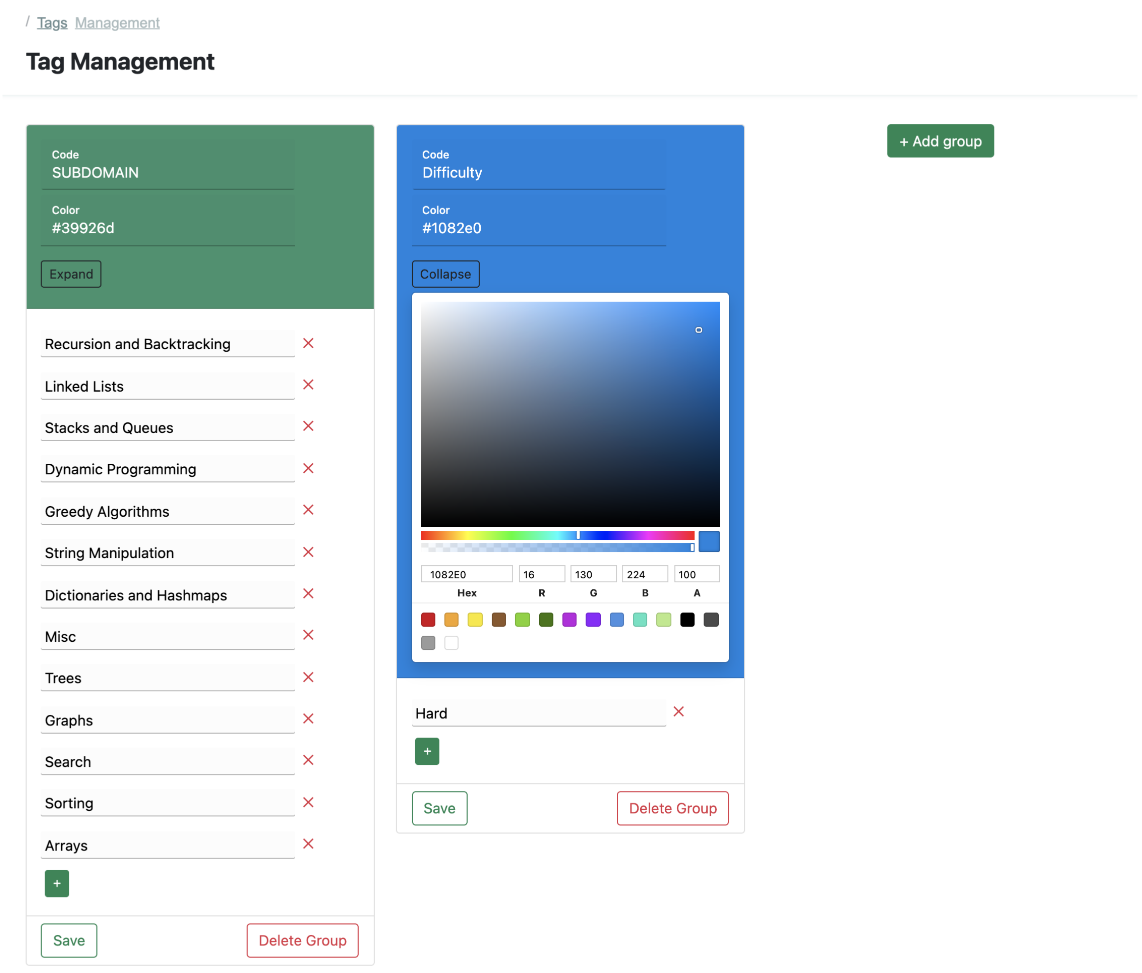
Repository id	Repository name	Assignment count	Last activity at	Last updated	Activities
4	iti0202-2021 / ex	52	2021.05.11 16:05:34	2021.05.11 22:08:11	Sync remote Expand
7 3.9%	automated-tests / Codera Exercises	38	2021.03.26 13:21:25	2021.05.11 22:08:39	Sync remote Close

Assignments(38) Similarity checks (0/7)

Add to check queue tiloom /HelloLittleOskar	Queued 0 Queued ✕ iti0102-2019 /ex
Add to check queue aurora /Aurora Resources	Running 0 Add to check queue Deniss.Potapenko /coding_challenges
Add to check queue python-tests /ex	2021.05.14 09:14:13 Finished 1 Add to check queue iti0202-2021 /ex
Add to check queue iti0102-2020 /ex	

Repository id	Repository name	Assignment count	Last activity at	Last updated	Activities
6	iti0102-2019 / ex	166	2021.03.29 13:52:21	2021.05.11 22:08:35	Sync remote Expand
3	python-tests / ex	29	2021.04.12 16:51:20	2021.05.11 22:08:08	Sync remote Expand
1	tiloom / HelloLittleOskar	1	2021.03.29 22:52:33	2021.05.11 22:08:01	Sync remote Expand
5	iti0102-2020 / ex	126	2021.03.26 13:03:44	2021.05.11 22:08:27	Sync remote Expand
2	aurora / Aurora Resources	2	2021.03.29 22:48:15	2021.05.11 22:08:03	Sync remote Expand
8 3.9%	Deniss.Potapenko / coding_challenges	111	2021.02.19 13:38:14	2021.05.11 22:08:43	Sync remote Expand

Appendix 10 - Tag management page



/ Tags Management

Tag Management

Code
SUBDOMAIN

Color
#39926d

Expand

Recursion and Backtracking ×
Linked Lists ×
Stacks and Queues ×
Dynamic Programming ×
Greedy Algorithms ×
String Manipulation ×
Dictionaries and Hashmaps ×
Misc ×
Trees ×
Graphs ×
Search ×
Sorting ×
Arrays ×

+

Save Delete Group

Code
Difficulty

Color
#1082e0

Collapse

Hex	R	G	B	A
1082E0	16	130	224	100

Hard ×

+

Save Delete Group

+ Add group

Appendix 11 - Assignments API pipeline needs

The needs visualization shown in Figure 11-1 makes it easier to visualize the relationships between dependent jobs in a DAG (Directed Acyclic Graph) [34].

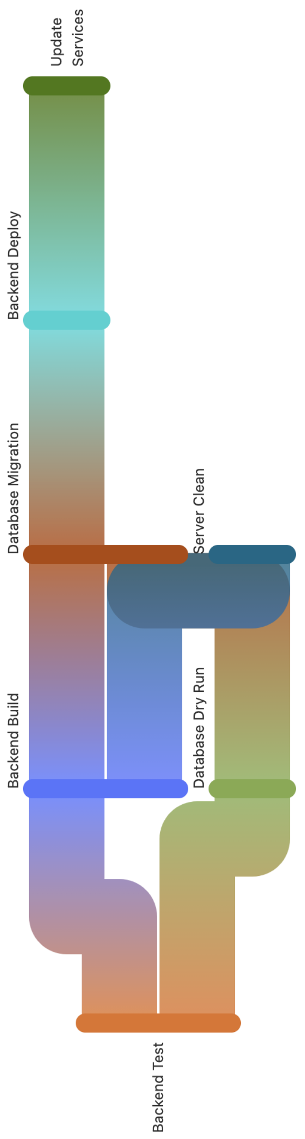


Figure 11-1 Assignments API pipeline needs

Appendix 12 - User requirements

User requirements are analyzed on an epic basis. Table 12-1 covers the epic sizes, and Table 12-2 lists the epics.

Table 12-1 Epic sizes.

Epic size label	**Weeks**	**Numeric identifier**
XS	0–1	0,5
S	1–2	1,0
M	2–4	2,0
L	4–8	5,0
XL	8–16	8,0

Table 12-2 List of epics.

Epic name	**Business value (1-10)**	**Kano category**	**Risk/Value**	**Epic sizing (Table 12-1)**
Assignment search	10	Delighted	Low risk / high value	L
Assignment tagging	3	Satisfied	Low risk / medium value	M
Repository overview	7	Satisfied	Low risk / high value	M
Assignment similarity automatic assessment	6	Satisfied	Low risk / high value	M
Assignment grouping	5	Satisfied	Low risk / high value	M
Assignment editing	6	Neutral	Medium risk / high value	L

Continues...

Table 12-2 – *Continues...*

Epic name	**Business value (1-10)**	**Kano category**	**Risk/Value**	**Epic sizing (Table 12-1)**
Assignment automatic tagging	6	Neutral	Medium risk / high value	L

The stories in Table 12-3 are analyzed with the INVEST methodology. The letters in the acronym stand for the following: I (Independent), N (Negotiable), V (Valuable), E (Estimable), S(Small), T (Testable) [35].

Table 12-3 User stories.

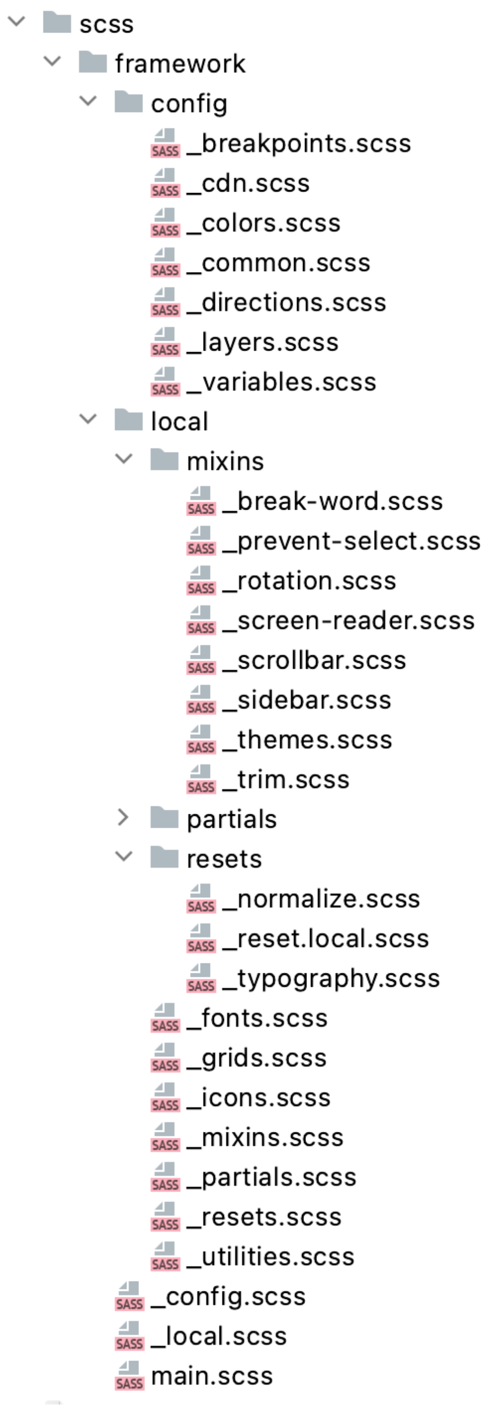
Nr.	**User story**	**I**	**N**	**V**	**E**	**S**	**T**
1	I as a user, I want to search for assignments by name so that I can find them	true	true	true	true	true	true
2	I as a user, I want to search for assignments by file contents so that I can get a better overview of the assignments input	true	true	true	true	true	true
3	I as a user, I want to see the file snippets where the search query was found, to get more information	false	true	true	true	true	true
4	I as a user, I want to add tags to assignments so that I can have a better understand of the contents	true	true	true	true	true	true
5	I as a user, I want to search for assignments via tags, so that I can get wished results quicker	false	true	true	true	true	true
6	I as a user, I want to see the repositories listed in the system so that I can get an overview of the assignments in the system	true	true	true	true	true	true

A user story is considered done when all acceptance criteria (e.g., Table 12-4) are completed, the client has accepted the stories, and the feature is in production.

Table 12-4 Acceptance criteria.

User story nr.	**Acceptance criteria**
1	User sees searched assignments
	The searched result of the assignment name is highlighted
2	User sees searched assignments
	The number of substrings found in files is shown on assignment as a number
3	User expands the file snippet viewing section and can see the code snippets
	A green dotted line separates the snippets if there are many in a single file
4	User can add tags to an assignment or multiple assignments
	Added tags are persisted
5	User can search via tags
	Active tags are highlighted
6	Repositories are displayed for the user when navigating to the repositories page
7	Assignments are automatically tagged & tags are persisted and displayed

Appendix 13 - SCSS file structure



```
scss
  framework
    config
      _breakpoints.scss
      _cdn.scss
      _colors.scss
      _common.scss
      _directions.scss
      _layers.scss
      _variables.scss
    local
      mixins
        _break-word.scss
        _prevent-select.scss
        _rotation.scss
        _screen-reader.scss
        _scrollbar.scss
        _sidebar.scss
        _themes.scss
        _trim.scss
      partials
      resets
        _normalize.scss
        _reset.local.scss
        _typography.scss
      _fonts.scss
      _grids.scss
      _icons.scss
      _mixins.scss
      _partials.scss
      _resets.scss
      _utilities.scss
    _config.scss
    _local.scss
    main.scss
```