MentorHub

High Level Design Document

Berktuğ Kaan Özkan

Version v1.0

August 17, 2021

Table of Contents

List of Figures

1 Introduction

1.1 Purpose

This software design document describes the architecture, implemented features and system design of MentorHub application. *MentorHub is a platform application which enables it's users to find or to become mentors about their expertise.*

1.2 Scope

This document outlines the high level functional design of MentorHub application. It highlights/refers the high level flows/use cases in design of components, along with the rationale for the same.
It serves as an input to the low level design documents that would further elaborate on the application design.

1.3 Reference Material

1. OBSS Company 2021 Java Summer Internship Directive (July 2021 v1.0)

1.4 Definitions and Acronyms

Term	Definition
LDAP	Lightweight Directory Access Protocol
SMTP	Simple Mail Transfer Protocol
API	Application Programming Interface
JVM	Java Virtual Machine
Docker	Docker is a set of platform as a service products that use OS-level virtualization to deliver software in packages called containers.

2 System Overview

2.1 Product Perspective

In a broader sense, users in the system will be able to provide mentoring support to other users about their expertise.

Experts, mentoring on the subject of expertise will provide mentoring support to the person concerned, based on the mentoring request received after the application is approved.

When the mentoring process begins, the mentor and the user (mentee) will come together, either online or offline, to determine the phases of the relevant process together with their due dates and enter them into the system. After entering the system, they will start the process and will be able to manage the phase changes over the system. In a mentoring process, only one phase can be active at any time t.

The mentor or mentee will finish the relevant phase through a button in the system when each phase is finished. After finishing the phase a review button will be activated. After clicking this button, a display containing a text field where users can write their comment and a field where users can give a score between 1 and 5 will open on the screen. After the phase is over, the person who has not closed the phase will be able to read the evaluation made by the person who closed it and make an evaluation comment himself/herself and give a score.

A user can become a mentor or a mentee at the same time.

The system will be notify the mentor and the mentee of a phase, 1 hour before the end of the active phase.

2.2 Tools Used

The detailed list will be presented in the next sections. In general, the following tools have used:

- Maven based Java Spring Boot Framework for the main entry point.
- Thymeleaf, HTML, CSS and JavaScript used for the end-user interface design. For additional support following libraries used,
  - JQuery Library for enhancing interactivity.
  - Bootstrap 5 Library for the industrial standards in visuals.
- Actual persistent data is stored in NoSQL database, MongoDB.
- Search engine selected as Elasticsearch due to advanced search capabilities (i.e. full–text search).
- The connection between MongoDB and Elasticsearch is established with Monstache.
- In order to quick and stable distribution of the software Docker containers used.

2.3 Assumptions

This project is designed with an educational mindset. Therefore, the resulting program should not be used in real applications, as is. Main concern is the lack of security, because the feature implementation was the first goal. Also in order to run application with email support an SMTP server needs to be configured.

3 System Architecture

3.1 User Requirements

3.1.1 Functional Requirements

1. **Users and Roles**

   There will be two type of user profiles: Admins and Users.

   There is no administrative interface to manage user profiles. User definitions will be obtained from Google Authentication and LDAP server.

2. **Subject Definitions**

   Admins will be able to add new subjects, edit and remove existing subjects from the system.

3. **Mentorship Requests**

   Users can apply to become mentor by selecting the subjects which expertise in and writing about themselves.

4. **Accepting and Declining the Mentorship Requests**

   The application requests will be presented to Admins after they logged in. They can either accept or decline waiting requests made by Users.

5. **Mentor Searching**

   Users can search among accepted mentors in real time by entering a search text or by selecting relevant subjects from a filter. Mentorship creation will be done by selecting a mentor from this page.

   One mentee can only work with one mentor under the same major subject. However, one mentor can only work with 2 mentees at the same time.

6. **Phase Planning**

   After mentorship creation, users will create/edit/remove phases of that mentoring process by specifying a name, an end date and end time.

   This process can be called as long as the mentoring process hasn't begun.

7. **Displaying Mentorship Details**

   Each user will see their mentoring processes after selecting a mentor. They can start the process from here and see the comments/ratings about a specific phase.

8. **Phase Reviews**

   After each phase completion, both mentor and mentee will write a short comment and give a rating between 1 to 5 stars about finished phase.

9. **Email Reminder**

   If there is less than one hour to end of an active phase, an email will be sent to both mentor and mentee.

   Mailhog container used for testing email reminder feature.

3.1.2 Non–Functional Requirements

1. **Authentication and Authorization**

   Users can securely log into the system and access appropriate information based on their role and account. Authentication of users should be done in following ways:

   - Login by LDAP
   - Login by Google Authentication

2. **Security**

   In order to access a page/resource users must be logged in. Logged in users can only access and view/change their own data, they cannot view/change others.

3. **Performance**

   Application should be fast enough to handle at least 1K users and 10K of mentorship process. It should be stable while performing.

4. **Accessibility**

   Application should be designed with flexible, user–friendly UI/UX and it should be responsive (i.e. work on mobile phones and tablet).

5. **User interface**

   The user interface is a very simple plain layout with little to no graphics. It will display information very clearly for the user and will primarily output information to the user through HTML pages formed with Thymeleaf framework. Administrative screens are use mainly for input through text fields in HTML pages. Screen shots have been provided to demonstrate the user and administrative interface, see section 3.1.3.

6. **Error Handling**

   Should errors be encountered, an explanation will be displayed as to what went wrong. An error will be defined as anything that falls outside the normal and intended usage.

7. **Maintainability**

   Very little maintenance should be required for this setup. An initial configuration will be the only system required interaction after system is put together. For example, defining user roles.

8. **Portability**

   This system should have the ability that, once it is together, the entire system should be able to be moved to any location, easily. Since the system deployed as Docker containers it is very easy to do so.

3.1.3 Process Flow

According to user requirements as defined in section 3.1, application workflow and interfaces can be seen from the below figure 1.

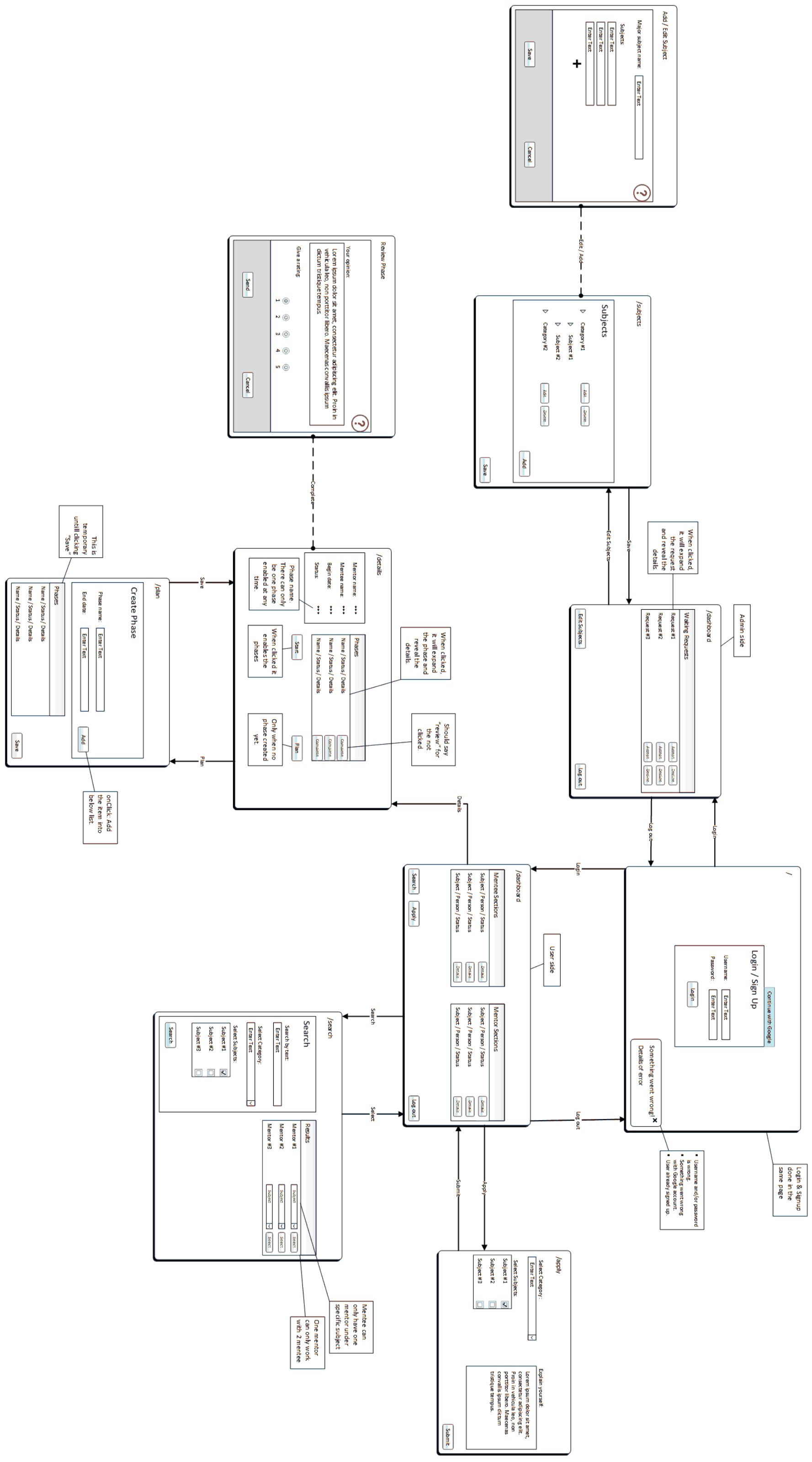

Figure 1: Process flow diagram

3.2 Data Requirements

3.2.1 Data Model

The data model designed as shown in the below figure 2.

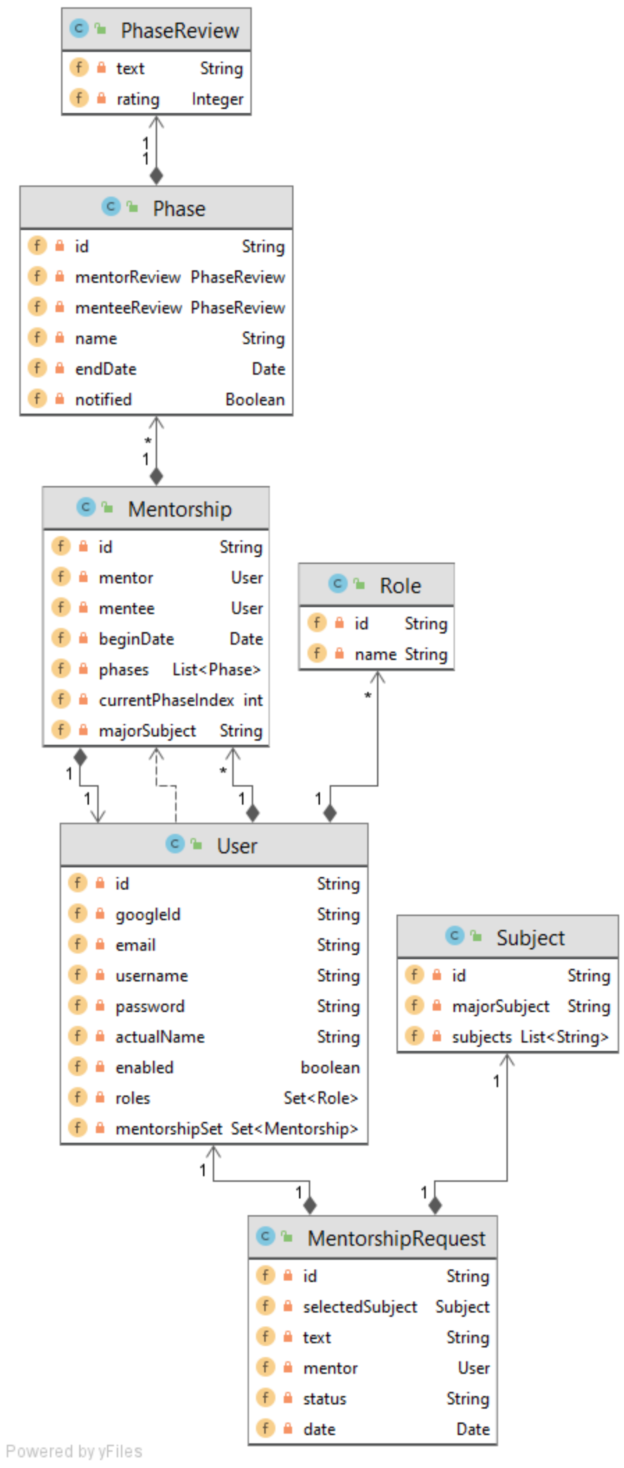

Figure 2: Entity relationships diagram

In database, only the classes with `id` field which is auto–generated by the database, have their own collections/tables. Instances of these classes stored as is, in database.

However, if a field references an entity which is already in the database; for example, `mentor` field for `Mentorship` entity (references to the `User` entity), is stored as reference (`DBRef` for MongoDB) inside the database. Otherwise they are stored as embedded documents, such as `PhaseReview` objects inside the `Phase` entities.

In practice the framework will solve this abstraction but know that they are stored differently in database.

3.2.2 Data Access Mechanism

Data access done by API calls. The API endpoints are as follows:

Endpoint	Method	Description
`/api/subjects/`	`PUT`	Creates a new subject or update the existing subject.
`/api/subjects/{idd}`	`DELETE`	Remove existing subject.
`/api/search/method/text?searchTxt`	`GET`	Retrieve mentors by full–text search.
`/api/search/method/filter?majorSubjectName&subjectList`	`GET`	Retrieve mentors by filtering by subjects.
`/api/requests/`	`PUT`	Create mentorship request.
`/api/requests/{id}/{answer}`	`POST`	Accept or decline mentorship requests.
`/api/mentorships/`	`PUT`	Create mentoring process.
`/api/mentorships/{id}/phases/`	`PUT`	Creates a new phase or update the existing phase for mentoring process with `id`.
`/api/mentorships/id/nextPhase`	`POST`	Start the next phase in a mentoring process.
`/api/mentorships/{mentorshipId}/phases/{phaseId}`	`DELETE`	Remove the phase from mentoring process.
`/api/mentorships/{mentorshipId}/phases/{phaseId}/reviews/`	`PUT`	Create a review for phase.

Only data changes handled by the endpoints. Retrieval of data such as subjects, user and mentorship details, reviews or phases etc. remained inside the application context. They are not explicitly shared, they are rendered on the page. This is mainly because to restrict unwanted access to API endpoints.

3.2.3 Search Implementation

As previously explained, actual data stored inside MongoDB. In order to abstract search layer, Monstache container used. Basically, the purpose of this container is to synchronize the mentorship request part of the data with Elasticsearch database. So that, the search can be done with Elasticsearch.

In implementation of the feature, application only needs to know the address of the Elasticsearch instance. Searching is handled by a dependency of Spring framework. Importing thing is that the result of the search query is the ids of the relative mentorship requests, converted into `SearchHitResponse` class. Then, it needs to be fetched from the database in order to get additional information. Since the `id` field is indexed in MongoDB it does not burden the system.

The separation of database and search engine is due to in future patches, which may result in a database migration, search feature of the design will not be affected (i.e. only the "connector" configuration will be changed).

3.3 Deployment

Project can be deployed as Docker image or standalone service. In Docker image all necessary services and configurations stacked together. Of course, according to necessity each container can be separated. Such Docker deployment diagram can be seen from the below figure 3.

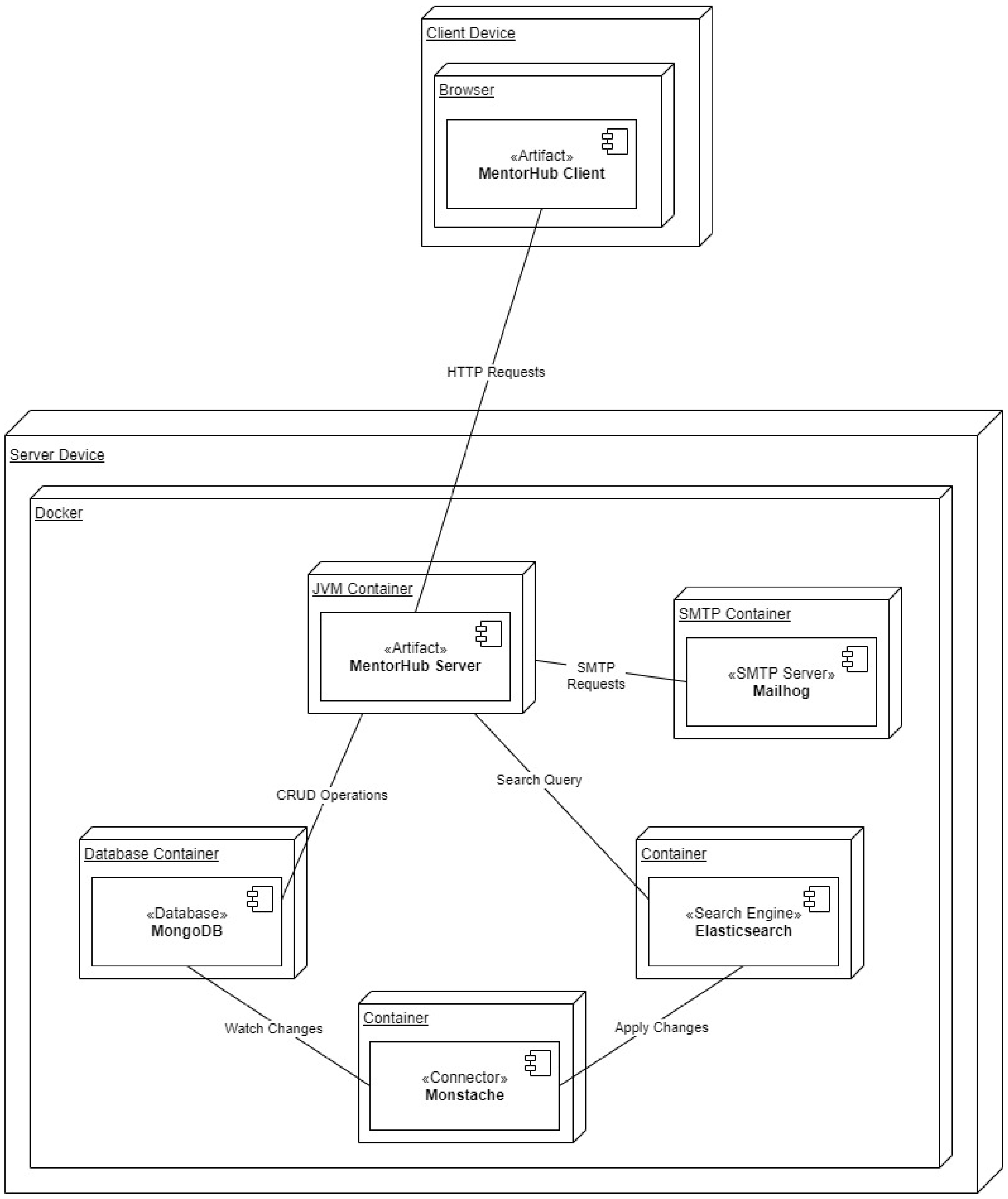

Figure 3: Physical deployment diagram

In development, the program has two main branches: `master` and `docker-release`. `master` branch mainly used for separated development environment for easier debugging and the changes made here (the patches) are applied to `docker-release` branch.

So, in order to release product select `docker-release` branch, compile a JAR file, copy it to `<project_dir>/docker/mentorhub-app/app.jar` and run `docker-compose up` command.