CS 319 - Object Oriented Software Engineering

Instructor: Eray Tüzün, TA: Muhammad Umair Ahmed & Elgun Jabrayilzade

BilHealth

Final Report

Implementation

Mehmet Alper Çetin
21902324

Vedat Eren Arıcan
22002643

Uygar Onat Erol
21901908

Recep Uysal
21803637

Efe Erkan
21902248

May 10, 2022

Contents

1 Introduction

Our project is a health center management software system, built in the form of a web application. The main goal is to provide online, remote attention to patients to increase productivity for all parties involved. The primary method of interaction is through *case*s, which contain all relevant information for a given medical situation. Patients use the system to open cases and request appointments, and the health center staff acts on these requests to provide medical services. These interactions also function as a variety of medical records for the health center.

1.1 Current Implementation State

- The calculator page was not implemented. It would consist of simple BMI and such calculations.
- The report generation feature was not fully developed and does not exist in the UI.
- Email notifications were not implemented. A rudimentary notification system was though.
- Some rough edges exist in many parts of the back-end and front-end implementation. Polish would definitely be required before actual production use.
- Some of the access security is lacking granularity to be per-user.
- Some of the UI has not been hooked up to the implementation on the back-end.
- The UI should be using more human-readable representations in some places where user IDs are currently used. This may require certain input fields to support searching through users in an auto-complete fashion.

2 Lessons Learned

First, we learned the difficulties of finding a team. Finding teammates with appropriate qualities was a hard job, although it was made easier by Peerpanda. We do believe we now have more experience in finding requirements for a given software system. We see that going out there and dealing with real-life problems was different from given academic tasks. We believe the hardship of getting requirements were communication problems, such as the barrier between engineers and current users. Not always do users help and not always do engineers understand their help.

In terms of code, we learned how difficult it might be to enter a totally different style of coding. Most of us had not used .NET and C# up until now which made it harder to get started. Some of us learned how to approach these new fields and learned not to underestimate learning new fields and allocating enough time. Design patterns were another important topic of consideration, and we observed the added difficulty of integrating design patterns, along with their benefits.

We also gained a little experience about how a group project plays out. Communicating with people has its difficulties and a decentralized group can be a problem when deciding

and acting. Since all of us had similar backgrounds it might sometimes be hard to convince a teammate when we think what we are doing is right.

Lastly given a small amount of time we find out that not everything works out as we planned it be. We had to make compromises as a result.

In terms of the engineering that went into the implementation, there have been numerous refactors. Some of these had to do with integrating design patterns. Another problem was the existing implementation becoming rigid and hard to extend over time, which ended up requiring us to create more classes. In other words, we observed the trade-offs between concise and rigid code against complex and lengthened code.

3 Build & Execution Instructions

There are two build flavors in which the application can run: development and production. The production build is more stable, but also requires more initial effort to run. Both builds are highly recommended to be run through **Docker**.

The project runs best on a Linux-based environment, despite using Docker. Through WSL and its integration with Docker, Windows is also able to run the project reasonably well. Note that the project source must be located within the WSL filesystem. In some instances, Docker causes WSL to consume unnecessarily high amounts of RAM. This can be solved through the WSL configuration as seen here.

For both of the build flavors, the prerequisites are that:

- you obtain the project source through tools such as *git clone*.
- you have installed Docker and docker-compose.

Then, begin by changing your working directory to the root of the project directory.

For the most up to date version of these instructions, see the README on the project's repository.

3.1 Development Build

1. Build the Docker image:

```
docker-compose -f docker-compose.devel.yml build
```

2. Any time after the first image build, run the application via:

```
docker-compose -f docker-compose.devel.yml up
```

Once the containers are fully up, the project should be visible at `https://localhost:7257/`. This development image mounts the project folder into the container, allowing changes to the code to be seen in real-time.

Note that the HTTPS support is through self-signed certificates, and your web browser may warn you about it. Ideally, your browser should give you the option to add an exception for the certificate. This is completely safe to do.

3.2 Production Build

The production build process is slightly more involved, at least on the first run. You will need to have additionally installed .NET Core SDK v6.0.x and EF Core tools.

Then;

```
# Build image
docker-compose -f docker-compose.prod.yml build

# Perform database migration (do this only on first run or new migration)
## Bring up only the database container
docker-compose -d -f docker-compose.prod.yml up dbpostgres
## Generate idempotent SQL script and copy into container
dotnet ef migrations --idempotent -o migrate.sql
docker cp migrate.sql bilhealth_db_postgres_1:/migrate.sql
## Enter the database environment and execute migration
docker exec -it bilhealth_dbpostgres_1 bash
cd / && psql -U postgres -d bilhealthprod -f migrate.sql && exit
## Bring down database
docker-compose -f docker-compose.prod.yml down

# Bring up all containers
docker-compose -d -f docker-compose.prod.yml up
```

Once the containers are fully up, the project should be visible at `http://localhost:5000/`. Note that unlike the development image, this one may need to be rebuilt upon every change to the project. The database migration process needs to be done only if the database is not up to date with the latest migration.

There is currently no support for HTTPS in the production build. If desired, a reverse proxy (such as Nginx) may be used to support HTTPS.

4 User's Guide

The overall user interface design of the system is actually parallel between the several actors. This means that the shown components have slight variations depending on the logged in user type. The overall page and URL remain mostly the same. There are a few pages that are specific to users of elevated privilege, such as the staff panel.

4.1 Login Page

This is the **only** page accessible by non-authenticated users that fall outside our actor type range. These guests can view the announcements made on the system. The only other functionality is to log in.

4.2 Profile Page

This page is basically a record of the user and their medical interactions on the system. The test results are accessed from this page. The vaccination history and physical measurements of the patient are found here. Some typical settings such as changing passwords are presented as well.

The user's profile can be modified on this page, but some of the sensitive properties are limited to modification by a privileged user.

The user can grant temporary profile access to other users on their profile. This is one of the forms of access control on the system. The staff can manually blacklist the user on this page.

4.3 Notifications Page

This page lists all of the notifications the user has received as a result of interactions. The user can mark notifications as *read*.

4.4 Announcements Page

User actors other than the patients can add, modify, and remove announcements on this page.

4.5 Cases Page

The case list on this page displays only the cases to which the user has access permissions. For example, a doctor will only see cases to which they have been assigned. Some overall details about a case are presented in the form of cards.

4.6 Staff Panel Page

This page is only visible to the staff and admin users. Several cards exist for distinct functionalities. New users of all actor types are registered on this page. The user list is how the staff members access patient profiles on the system. It is searchable by username, full name, and user ID.

Lastly, there is an audit trail card, which is one of our access control strategies. This is a common feature in health record systems around the globe. It displays the logs of whoever accessed a patient's profile.

5 Work Allocation

- **Mehmet Alper Çetin:** Interview with the Bilkent University health center. Use Case, State and Activity Diagrams in Requirements Report. Lessons Learned in Final Report. Evaluation of Sibling Group's Design Report. Presentation Outline. Requirements Engineering.

- **Vedat Eren Arıcan:** Interview with the Bilkent University health center. Implementation of the backend and frontend code bases. DevOps tasks such as integrating Docker and a GitHub CI workflow. Writing textual report content through LaTeX. Access matrix and deployment diagram.

- **Uygar Onat Erol:** Sequence Diagrams in Requirement Reports. Boundary Conditions and User Interface Layer Class Diagram in Design Report. Preparation of presentation slides.

- **Recep Uysal:** State and Activity Diagrams in Requirement Reports. Design Goals and Object Design Trade-offs in Design Report.

- **Efe Erkan:** Use Case Diagram in Requirements Report. Evaluation of Sibling Group's Requirements Report. Implementation of the backend code bases. Subsystem Decomposition and Class Diagrams in Design Report.