Quality Assurance Document

Nikola (nk465)

Introduction

This document provides an overview of the quality assurance practices implemented during the development of our medical appointment scheduling system. The practices aimed to ensure that our project structure is logical and understandable, all the java files look alike, and the code is easy to read, functions properly and is maintainable.

Code Reviews

I, Nikola, conducted code reviews throughout the development process. I conducted a general code review twice per sprint to ensure that the code adhered to best practices and was consistent throughout the project. IntelliJ’s code review functionality was used to track any missed issues and to fix them. It caught many typos, inefficient if-else statements, redundant imports, etc. Examples of what I did in a code review conducted during the project include:

* Going over the database schema design to ensure normalization.
* Review of the different GUIs to enforce consistency in the font, font-size, spacing between the elements, background, contrast color, and window size.

My feedback from the code reviews was incorporated into the development process by communicating it in our group chat to discuss it and then making changes based on the suggestions

Refactoring

Refactoring is an essential process in software development that involves improving the internal structure of the code without changing its external behavior. Such techniques were used to improve the quality and readability of our codebase. Throughout the project, I identified areas where refactoring was necessary and performed the did the following:

* Renaming variables to be more descriptive. This was important for code readability and helped to prevent errors caused by confusion over variable names. For example, I changed ‘u’ to ‘user’ and ‘p’ to ‘password’.
* Renaming method names so that they are self-explanatory. This was strongly enforced on public methods, to eliminate the need to jump back and forth between files to find out what a particular function does. For instance, I renamed the method ‘init’ to ‘initializeTable’.
* Extracting duplicate code into functions and classes. I identified repeated areas of code and extracted them into functions and classes to eliminate redundancy and improve maintainability. For example, I refactored the method that sets the look and feel of a GUI, which shortened the code by almost 300 lines (this was the 2nd commit on the 30th of March, I’ve given a detailed description there).
* Breaking down complex functions. I identified lengthy and complex methods and separated them into multiple smaller functions to make the code easier to understand and modify. For instance, I divided the main method in RegistrationCheck into the test, firstnameCheck, surnameCheck, ageCheck, phoneNumberCheck, doctorCheck, and detailsCheck boolean methods.

Project Structure Maintenance

We used a package-oriented structure to make the project more logical. The packages are a combination of logically connected files with a similar purpose. Each of them contains a FileFormat.txt that provides a general template that all the files in the package should follow. The following packages were used:

* Databases package – This package contains the files that initialize the DB and its tables. There is a file for each table, such as ‘BookingsDB’ for the Bookings table and ‘DoctorsDB’ for the Doctors table. The tables should be accessed only through the methods declared in these files.
* Checks package – This package contains all the files that perform specific checks on the information the user provided. These checks include general data format checks and logical checks. The data format checks watch for data inconsistencies such as if a DB table requires an integer value for some column, but the user provided a string. The logical checks watch if the user tries to book an appointment out of working hours or with a non-existing doctor.
* Functionality package – This package links the GUIs, the Checks, and the Databases. The Functionality files receive information from a GUI, send it to a Check, and if it returns true, then they send it to a Database and update the GUI. There is a Functionality file for every GUI that has buttons.
* GUIs package – This package contains the different GUIs for our interface.
* Session package – This package contains important information, that all other files often refer to. It has two files – Info.java and General.java. The former contains important information regarding the current session, such as the current user’s id and his doctor’s name. The latter one has general information and methods that other files often use.
* Tests package – It contains all the unit tests for our project which help us make sure we don’t break some old functionality by accident, when committing new code.

Using this package-oriented structure, we kept our codebase organized and modular. Each package has a clear purpose, and files within each package follow a consistent naming convention and structure. This made it easier for developers to locate specific files and understand how they fit into the overall project.

Issue Tracing

Whenever an issue was found, a team member noted it in our group chat on WhatsApp, describing the problem encountered and providing relevant details such as the affected component, steps to reproduce, and expected behavior. The issue was then assigned to a team member responsible for fixing it. We also used the issue tracker to track the progress of bug fixes and updates to ensure that all issues were resolved before the project’s final submission.

Conclusion

By following these quality assurance practices, we were able to ensure that our project code was of high quality, easy to read and maintain, and was structured logically. We were also able to track and resolve any issues and bugs encountered during the project’s development, which allowed us to submit a final product that met the project’s requirements.