Development Guide for The Smart Medical Management System.

By Circle Lab

Contents

[Planning 3](#_Toc38300193)

[Analysis 3](#_Toc38300194)

[Design 3](#_Toc38300195)

[Implementation 3](#_Toc38300196)

[Testing & Integration 3](#_Toc38300197)

[Maintenance 3](#_Toc38300198)

# Planning

We used the Belbin self-perception test to identify our strengths, weaknesses, and behaviours. The test consists of questions which have several answers each with a score waiting. These answers are tallied to give a score which links to role types based on which type of answer was given the most often.

We used CmapTools to generate a concept map which is a graphical tool for organising and representing knowledge. We used this to show how the smart medical management system is linked together, from stakeholders to body parts to appointments. The concept map can be found in the cmap folder.

We developed a pert chart to plan and layout an approximation of how long each step and stage of this project should take individually and cumulatively with the rest of the stages. Each stage displays the earliest start time, earliest finish time, latest start time, latest finish time, duration, and slack time. The pert chart can be found in the critical path analysis folder.

# Analysis

For the analysis stage, we had a one to one meeting with the user to establish what they wanted from the application. From this interaction, we created a problem statement. An example problem statement might be, “Doctors can book appointments”.

Using the meeting with the client and the problem statements noted from them we created a use case diagram which is a simple view of how the user interacts with the application and how the application deals with the actions of the user by passing information between the upper and lower levels of the application. The use case texts and use case diagram can be found in the use case folder.

Using the information from the problem statements we could speculate about possible classes the application is going to require and the relationships between these classes. For example, a person may have a prescription which has a drug, which shows 3 possible classes, person, prescription, and drug. The domain model can be found in the domain model folder.

# Design

We created a robustness diagram for several stakeholders and use cases. The purpose of the robustness diagram is to show how the layers of the application are linked together. It also shows which class is called when certain methods are called. The robustness diagrams can be found in the robustness folder.

We created a sequence diagram for several stakeholders and use cases. Sequence diagrams are used to see information being passed through the layers of the application. The sequence is a more in-depth view of the use case diagrams created in the earlier analysis stage.

We created a class diagram which is used to get a complete overview of the classes of the application and how the classes are linked together. For each class, the diagram lists the methods and attributes the class has along with the required input of each method and the output of each method.

# Implementation

We developed the database for the application using MariaDB version 10.4 to meet the system requirements laid out in the analysis stage. It was developed using the MySQL workbench software to create the code and entity-relationship diagram. This can be found in the entity-relationship diagram folder and is titled ERD.png the workbench model can be found in the same folder titled Smart Medical Management System Model. There are also SQL scripts in the SQL folder to test the database and set up the base database from scratch.

The user interface was implemented using PHP version 7, HTML5, and CSS3. All work has been completed in the IDE PHP storm and has been completed in line with the UX designs created in the design phase. These designs can be found in the UX folder and then in the subfolder UX screenshots.

The main application has been developed using PHP version 7 and XAMPP version 3.2.4 there are 15 DTO’s (Data Transfer Object) classes and 15 DAO’s (Data Access Object) classes. Along with one service façade class that ties everything together. All software has been developed using Git and GitHub and has been coded to international standards.

# Testing & Integration

During the development of this application, we have run several tests to make sure that it will function correctly when out in the real world. All the 15 DAO’s and DTO’s have been unit tested along with the service façade. All unit tests can be found in the application folder and then in the unit tests folder.

The database was tested as well using the Test DB SQL script in the entity-relationship diagram folder and the subfolder SQL scripts. It was also manually tested by two engineers to make sure that everything was functioning correctly according to the dummy data files that can be found in the Entity-Relationship Diagram folder and the subfolder Database Data.

We have also written a test plan to help test the final integration of the system modules this can be found in the test plan folder titled test plan. This test plan was then run by another engineer who had the following results from the test plan that can be found in the same test plan folder titled test plan results.

# Maintenance

In the final stage of development, we wrote a user guide for helping new users understand how to use the system. This can be found in the user guide folder and is titled user guide. We have also written this development guide to help onboard new people and allow them to understand the development process. Or to allow someone who was not part of the original team to take over maintenance of this application.