**Solo Covid Project**

**Project Report**

**Created by Dominic Smith**

**Index**

|  |  |
| --- | --- |
| **Topic** | **Page** |
| Introduction   * Purpose * Scope | 2 |
| Phase Plan | 3 |
| Product Perspective | 3 |
| Problem Statement | 3 |
| Problem Analysis | 4 |
| Project Limitations | 5 |
| List of Requirements | 5 |
| Feasibility Study | 6 |
| Risk Analysis | 7 |
| SWOT Analysis | 7 |
| Stakeholder Analysis | 8 |
| Use Case Diagram | 9 |
| Use Case Description | 10-13 |
| Domain Model | 14 |
| ER Diagram | 14 |
| Class Diagram | 15 |
| Database Design & Normalisation | 15-16 |
| Extra Specifications | 17 |
| FURPS | 17 |
| UML | 17-18 |
| Activity Diagram | 18 |
| System Sequence Diagram | 18 |
| Sequence Diagram | 18 |
| State Diagram | 18 |
| Package Diagram | 18 |
| GRASP | 19 |
| Quality Concepts | 19 |
| Software Testing Techniques | 20 |
| Construction | 20 |
| Conclusion & Reflection | 20-21 |
| References | 21 |

**Introduction**

**Purpose**

The purpose of this project and this report document is to begin to outline the requirements for a booking service that would allow users to book tests for coronavirus (covid-19) and to build and design a web-based system that facilitates the outlined requirements in a reliable manner. This system will be designed and produced for use by the Danish Health Authority (Sundhedsstyelsen) and will aim to maintain user friendliness, usability and will focus on an intuitive User Interface (UI).

**Scope**

This report will focus on the test booking website and the considerations that are made in the production cycle. This system will be designed with the previously mentioned requirements in mind.

**Phase Plan**

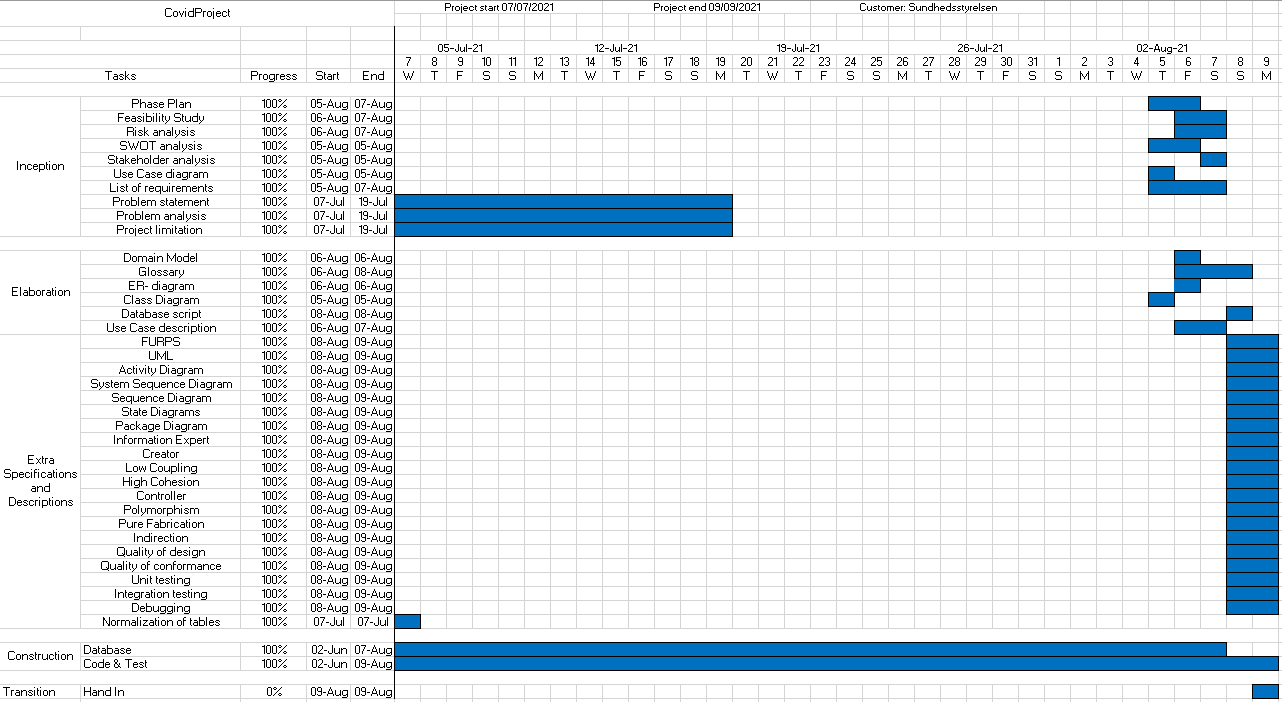
****

Figure 1: Phase plan

**Product perspective**

Upon completion and approval, the Covid-19 test center website will be made available to users online to alleviate the load and compliment the current system of test-booking available to the general population. Alongside this user functionality, the system will also allow users to change and cancel their appointments. The system also allows administrators to add and remove users, alter their details and view a full list of appointments and users.

**Problem Statement:**

Covid-19 has devastated the lives and livelihoods of every person on the planet since it became a worldwide pandemic in March 2020. Over the course of the past year and two months, there has not been a single person alive that hasn’t felt the devastating effects of this virus, whether that be due to closure of their leisure activities, their place of work, or their favourite store or the loss of a loved one from the virus’ devastating symptoms and complications. Even when the virus’ infectivity is receding, the effects still make themselves known, in the form of regular required testing that is often a prerequisite to taking part in everyday activities such as going to the gym or clothes shopping.

|  |  |
| --- | --- |
| The problem of | There isn’t a reliable and quick way to book a covid-19 test. |
| affects | Every person who needs to possess a negative test for their job or leisure activities |
| the impact of which is | Long lines and wait times at quick-test centers around the city and overloaded testing centers |
| a successful solution would | Reduce the waiting times, both online and in person for testing for the virus. |

**Problem Analysis:**

As the virus has dominated every aspect of most people’s lives for the past 18 months, it’s almost impossible to look anywhere without some new news story, new research, new strain or new breakthrough being front and center. Largely, this has bolstered the ability to research the problem and understand its effects and identify some key areas where the current system that is in place could be supported or improved.

Every government on every continent has some form of public information system to inform their citizens of the current state of the virus. In Denmark, it is sst.dk, where you can find a plethora of facts and figures about the novel coronavirus, whether that be the current spread rate, travel restrictions, the vaccine schedule or the reopening plan. These governmental information centers are also supported by similar, more global statistics from other nation’s governments as well as supranational organisations like the World Health Organization (WHO).

Research into the problem was largely based on our own experiences with the current situation and the testing process. As both a user and designer of a test booking system, I have a perspective that may offer some useful insights into what users want, need and can handle. The biggest issue I found when using the currently provided services were long waiting times, unnecessary complexity and lack of flexibility. What if I only had 5 minutes free to book a test but the queue was 25 minutes long? Why does the booking system require so many details when the information is already stored elsewhere? What if I want to change my booked test/vaccine appointment, or cancel it?

All of these questions were considerations when designing the booking system and vitally important targets to hit when building this system.

**Project Limitations:**

When embarking on a project, it’s sometimes difficult to foresee all of the limitations and potential problems that could occur over the course of the project development. Personal issues are rarely insurmountable, but some things are simply unavoidable. Some of the limitations faced in this project were:

Scheduling issues – With the summer months approaching and all kinds of leisure activities opening again following the Covid-19 Lockdown that started in December 2020, it was inevitable that scheduling conflicts would arise between when work was planned to be done and other responsibilities like a job. To combat this, the previously built system was used to free up a large amount of time with alterations and improvements being made up to and after the deadline.

Group of one – Combined with the previous limitation, a major factor in the speed at which this project was completed is the lack of other group members. It was decided that it would be completed on an individual basis and as a result there was nobody that could pick up the slack or take up the work when I was unavailable. There was no real plan to remedy this, and instead work was completed as and when it could be.

Lack of prior knowledge – One of the key limitations of this project was the introduction of an entirely new software platform in the form of Spring and Spring Boot, and more recently Spring security. With little to no prior knowledge of Spring or the Spring Boot framework, and a short amount of available time to acquire the necessary knowledge for the project, the majority of the hours spent on this development were spent trying to gain an understanding of Spring and its components.

**Requirements List**

Here is a list of the requirements for this project. This list may not be exhaustive, and may be expanded in the future;

Computer

Active Internet connection

GitHub

MySQL workbench

Java Development Kit

Java-based IDE, IntelliJ preferred

Google Docs/Drive

**Feasibility Study**

At its heart, a feasibility study aims to determine 3 things;

1. Does the customer actually want or need what they’re asking for?
2. Should we actually make the product or service?
3. Can we make the product or service they’re asking for?

The following few paragraphs are an attempt to answer these questions;

1. Does the customer want/need this?

The countless figures being released by governments worldwide is overwhelmingly supportive of the idea of regular testing of citizens, young and old, to best facilitate the prevention of the spread of coronavirus (covid-19). The need for testing then is clear, but what about whether they want the system? Testing people en-masse on a free-for-all & walk-in basis can be hectic, stressful and time-consuming, potentially leading to longer waits at the testing centers and longer processing times for results. Therefore to help with some semblance of order and organisation, a test booking system would be highly advantageous.

2. Should we make this?

As mentioned in the previous paragraph, the need for testing members of the population from all demographics is imperative to combatting the pandemic of Covid-19. With Sundhedsstyrelsen wanting to improve the efficiency and organisation of their testing facilities it would be highly advantageous to have a booking system.

An online web-based booking system for tests would not only improve the statistics for Sundhedsstyrelsen, but also aid members of the public with quicker test administration, allowing them to plan their days more effectively and cutting the time spent queueing drastically. In short: yes, we should make this.

3. Can we make this?

The tools to make this product, namely; Java, JDK and Spring Boot, are all available for use, and although the technology of Spring Boot is new and unfamiliar, with some research this project is more than possible.

So… is it feasible?

With the previous paragraphs in mind, it has been determined that this project is very feasible.

**Risk Analysis**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Risk Factor** | **Probability** | **Impact** | **Severity** | **Mitigation** |
| **1** | Team member falls ill | 1 | 2 | Low | Due to being fully vaccinated against covid-19, this is highly unlikely. If illness does occur, working from home is possible. |
| **2** | Internet unavailable | 2 | 3 | Medium | To mitigate this risk, the developer has multiple locations where they can complete their work without hindrance |
| **3** | Change in government stance/requirements | 2 | 2 | Medium | To mitigate this risk, the system being built will be flexible, allowing the changing of features and services fairly quickly. |
| **4** | Data loss | 1 | 3 | Medium | To mitigate the risk of losing data in the database, the database is saved on a hosting service as well as maintaining a local copy to minimise the risk of complete data loss |

Figure 2: Risk Analysis

**SWOT Analysis**

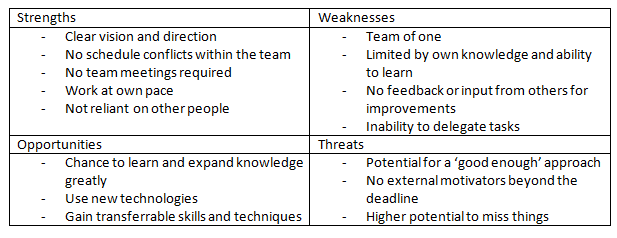
****

Figure 3: SWOT Analysis

**Stakeholder Analysis**

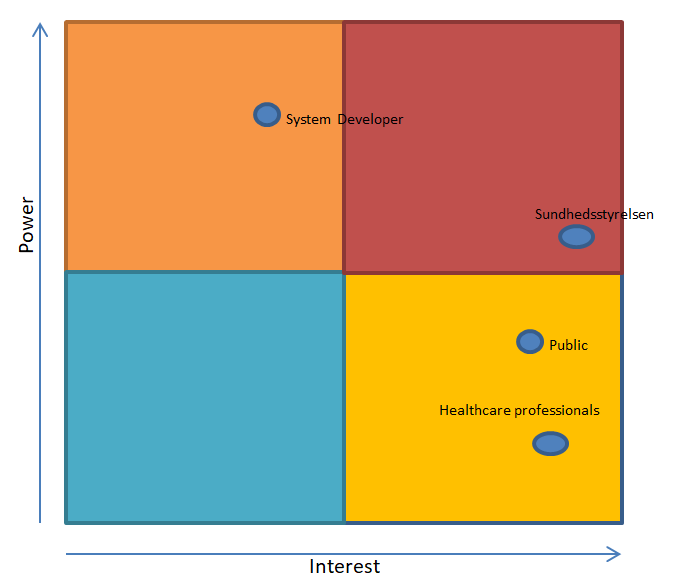
****

Figure 4: Stakeholder Analysis

**Use Case Diagram**

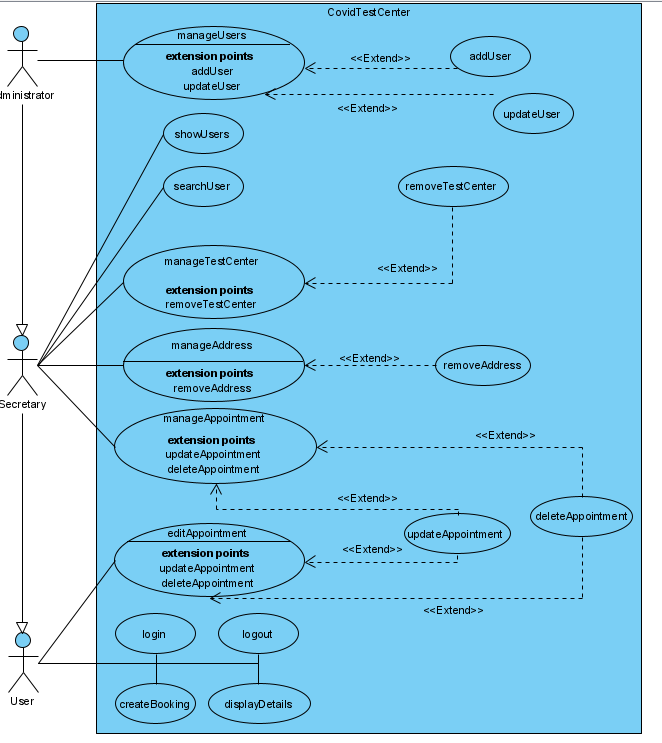


Figure 5: Use Case Diagram

**Use Case Descriptions**

**Add User**

**Actor(s)**

Administrator

**Stakeholders**

User

Secretary

Administrator

Sundhedsstyelsen

**Pre-Conditions**

1. Administrator is logged in

2. Administrator has been authenticated successfully

3. User details to be added are present

**Post-Conditions**

1. User details added to database

2. Database updated successfully

**Main Success Scenario**

1. Administrator selects the option to add user

2. System redirects to the 'add new user' page

3. 'add new user' page requests new user details

4. Administrator inputs new user details

5. Administrator presses the submit button

6. System accepts new user details

7. Database is updated with new user details

8. Administrator is redirected to the index page

9. Administrator exits.

**Alternative Scenarios**

7a. Database connection is disrupted

7b. New User details are not updated in the database

7c. Error page is displayed

7d. Administrator starts the process again.

**Remove Test Center**

**Actor(s)**

Administrator/Secretary

**Stakeholders**

User

Secretary/Administrator (Authenticated User(AU))

Sundhedsstyelsen

**Pre-Conditions**

1. AU is logged in

2. AU has been successfully authenticated

**Post-Conditions**

1. Test Center is removed from the database

2. System is updated to reflect database removal

**Main Success Scenario**

1. AU selects 'management' option

2. AU is presented with a selection of entity's to manage

3. AU selects 'Manage Test Center'

4. AU locates the Test Center they wish to remove

5. AU selects the 'remove' option

6. System sends request to database to remove the test center

7. Database removes the test center

8. AU is redirected to the index page

9. AU exits.

**Alternative Scenarios**

7a. Database connection is disrupted

7b. Test Center is not removed from database

7c. System displays error page

7d. AU begins the process again.

**Update Appointment**

**Actor(s)**

Administrator/Secretary

**Stakeholders**

User

Secretary/Administrator (Authenticated User(AU))

Sundhedsstyelsen

**Pre-Conditions**

1. AU is logged in

2. AU has been successfully authenticated

**Post-Conditions**

1. User appointment is successfully updated

2. User is able to see updated appointment information

**Main Success Scenario**

1. AU selects 'management' option

2. AU is presented with a selection of entity's to manage

3. AU selects 'Manage Appointments'

4. AU searches for the appointment they wish to update

5. AU selects the appointment to be updated once located.

6. AU inputs the new details for the appointment

7. AU selects update

8. System sends the new appointment information to the database

9. Database removes old appointment data

10. Database adds new appointment data

11. AU is taken back to the appointments page

12. AU exits.

**Alternative Scenarios**

10a. New appointment information is in incorrect format

10b. Database is unable to input the new appointment information

10c. Database does not remove old appointment data

10d. AU is informed of the error

10e. AU is prompted to input the new appointment details again

**Domain Model**

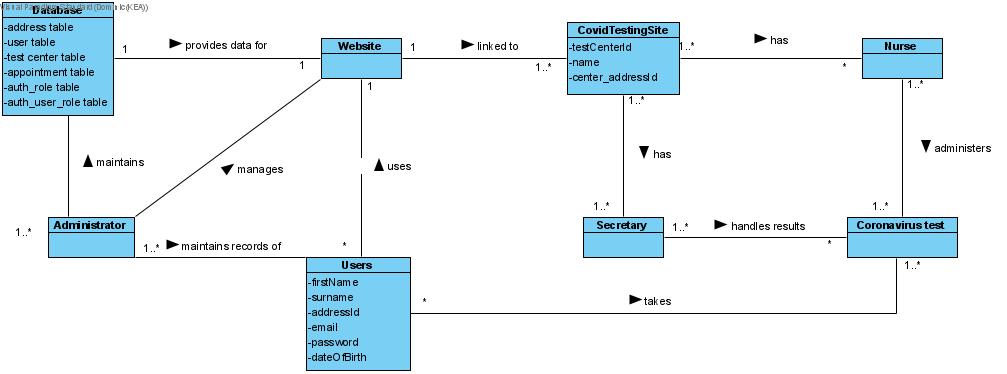
****

Figure 6: Domain Model

**Entity-Relationship Diagram**

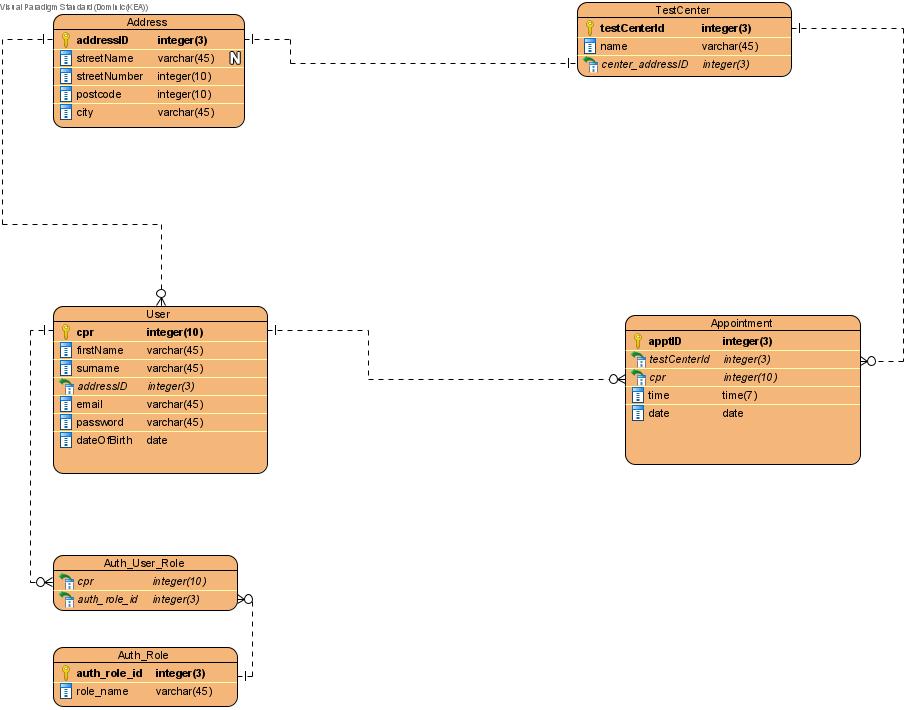
****

Figure 7: Entity-Relationship Diagram

**Class Diagram**

Due to the large size of the class diagram, it makes for impractical viewing within the confines of this document. Instead, the Class Diagram can be found as both a jpeg or in its original form (Requires Visual Paradigm 16.1 to open) at the following link: <https://github.com/Sintry1/CovidProject/blob/master/UML/Class%20Diagram.jpg> (image), <https://github.com/Sintry1/CovidProject/blob/master/UML/Visual%20Paradigm%20files/CovidProject.vpp> (original form).

**Database design and normalisation**

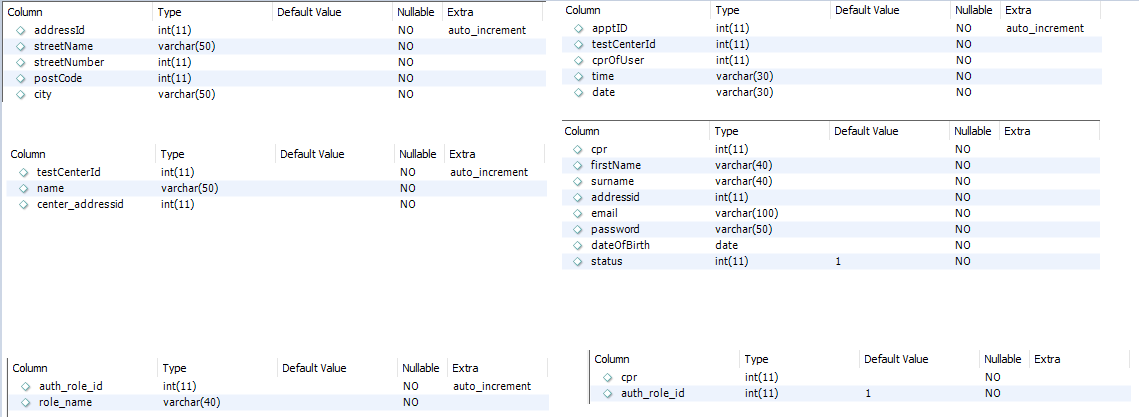


Figure 8: Database Layout

No matter what industry, country or time zone we find ourselves in, in today's databases are a huge part of the everyday life of most people. Your bank account is part of a database of bank accounts, if you have stocks and shares, they’re all part of another database, the purchases you make at the store are stored in their database and the list goes on.

It therefore stands to reason that databases need to be well designed and laid out in order for the information contained within to not become a mess of incomprehensible numbers and letters.

“A well designed database is easy to understand and query, while a poorly designed database is difficult to work with” (Murach, 2019, p.306).

To construct a database, Murach (2019, p.307) suggests that it should be modeled as close to a real world system as it is possible to get. To do this, he suggests 6 steps for designing data structures within the database. Below, I will list these steps and provide examples from within this project where possible:

1. Identify the data elements.

In our database, these data elements would be users, test centers, appointments and addresses.

1. Subdivide each element into its smallest useful components

Taking the user element as an example, they are subdivided into multiple items; cpr, firstName, surname and so on.

1. Identify the tables and assign columns

The tables in our system are the names of our data elements, while the columns within these tables are the smallest useful component.

1. Identify the primary and foreign keys

A primary key is used in a database to identify a single data element within a table in the user table, this would be the first column, cpr, as this is an identification number unique to each individual person in the system, whereas in our address table, the primary key is the addressId column, which is auto-incremented every time a new address is added.

The primary keys in one table are usually used as foreign keys in another. For example, each test center has a testCenterId and an addressId to represent the address associated with the center. The addressId is the foreign key,

1. Review whether the data structure is normalised

Normalisation is the act of separating data in a database into related tables and reducing data redundancy. In a normalised data structure, each table should contain information about only one entity, although there can be multiple entries in each table. This also means that each piece of information is stored in only one place (Murach, 2019, p.319).

1. Identify the indexes

An index in a database, like an index in a book, helps to locate information more quickly. When using MySQL, as in this project, indexes are automatically created for keys, both primary and foreign. Indexes are designed to improve performance and speed up searches.

**Extra Specifications**

**FURPS**

FURPS is an acronym that represents 5 components of software quality attributes that are used in the business and software development world to describe functional and non-functional requirements of software systems. The 5 components of FURPS are; Functionality, Usability, Reliability, Performance and Supportability.

According to Larman (2004, p.42), the 5 components of FURPS relate can be broken down as follows;

Functionality - Functionality relates to the features and capabilities of a system, along with its security features.

Usability - The Usability aspect of FURPS is tied to the UX, or User Experience, of the product or system. This includes any sort of help/guidance available to assist with the use of the system, along with documentation on the system.

Reliability - The Reliability of a system is, rather predictably, how reliable the system is. A system with poor reliability may have a high occurrence of errors and faults, experience long periods of downtime or experience unrecoverable data loss.

Performance - The Performance of the system is how well the system can carry out the tasks for which it is designed. For example, is the data that is stored done so accurately? Does the system use a lot of resources when in operation? It can also relate to the speed at which these are taking place. If a system takes 1 minute to log in, it’s unlikely that the system is well optimised and performance is likely to be poor in the whole system.

Serviceability - The S in FURPS relates to serviceability, which can be described as “how well can this system be maintained, adapted or changed to meet the needs of new customers, or the emerging requirements of current customers.

Overall, the components of FURPS are a good way to ensure that requirements for a system are being met, and while not exhaustive, they provide an excellent starting point for defining a systems’ requirements.

**UML**

Unified Modelling Language, or UML for short is a graphical modelling language used to illustrate a system in a graphical way. UML is extremely useful in all aspects of system design and development as it is a language agnostic way of representing a system, that is, it is not only representative of one particular programming language, such as Java or Python, instead it can be used regardless of programming language.

Some examples of UML appear earlier in this report, namely; Class Diagram, Use Case Diagram, Entity Relationship Diagram and Domain Model. Further examples are the Activity Diagram, System Sequence Diagram, Sequence Diagram, State Diagram and Package Diagram, which are described below.

**Activity Diagram**

An Activity Diagram is a UML Diagram that shows the flow of activities in a system. It is considered a behaviour diagram, because it shows what must happen in the system. According to the author of Applying UML & Patterns, Craig Larman (2004, p.607) an activity diagram is useful for visualising business workflows and processes and use cases.

**System Sequence Diagram**

A System Sequence Diagram is used to visualise a use-case and the steps that take place during that specific use case. System Sequence Diagrams can be thought of as a visual representation of a use-case description.

**Sequence Diagram**

Sequence Diagrams can often be mixed up with System Sequence Diagrams and while they bear some similarities, a Sequence Diagram is slightly more granular, and visualises a single method in a system and also focuses on the messages being sent back and forth from the system.

**State Diagram**

A state diagram is another behavioural diagram in UML and is used to illustrate the states, state transitions and actions of a system. State Diagrams are useful for systems with objects that go through multiple states in their life cycle. An example of one such object, is a bankAccount that is inactive until some form of open() method is invoked, at which point the state will change to active (Chonoles and Schardt, 2003, p. 262). An example of an object with such a state in the covid testing system would be a testCenter. Once created, the testCenter becomes active (a hidden attribute within the object) and eventually, once the pandemic passes, or testing requirements are lessened, the need for the testing centre will disappear, and thus it will be discontinued, or made inactive.

**Package Diagram**

In UML, a package diagram is a diagram used to represent a system in a slightly less granular fashion than a class diagram. A package diagram may show the classes within a system on a non-granular basis, i.e. it will usually not show methods and attributes, however its primary purpose is to give a real overview of the structure of a program without the minutiae of a class diagram.

**GRASP**

GRASP is an acronym that stands for General Responsibility Assignment Software Patterns. GRASP are patterns that help with how we build and design systems and the objects that make them up. Larman (2004, p.216) describes GRASP as a “learning aid to help understand essential object design and apply design reasoning in a methodical, rational and explainable way.

GRASP can be broken down into 9 principles; Information Expert, Creator, High Cohesion, Low Coupling, Controller, Indirection, Polymorphism, Pure Fabrication and Law of Demeter. Some of these principles are defined below;

Controller - The controller pattern is the class that controls the flow of data within the system. This class is responsible for ferrying the inputs of users to the system and carrying the system response back to the user, ready for the view to display to them in an understandable way.

High Cohesion - “*In terms of object design, cohesion (or more specifically, functional cohesion) is a measure of how strongly related and focused the responsibilities of an element are”* (Larman, 2004, p.232). A good general rule of thumb for assigning responsibilities is that elements should be tasked with related responsibilities, while also being assigned a low amount of work. An object that follows these rules would be defined as possessing high cohesion.

Low coupling - Coupling, as the name may suggest, refers to how closely a class or entity depends on any number of other classes or entities. According to Larman (2004, p.230), low coupling is a principle that should always be kept in mind when making design decisions. Placing too much responsibility on one class, or making multiple entities reliant on one class, while unproblematic in smaller systems, can quickly spiral out of control and become unwieldy in larger, more complex systems.

**Quality Concepts**

In the world of software, Quality Concepts refer to how ‘fit-for-purpose’ the software is. This can be further broken down to separate components of quality of design, and quality of conformance, both of which are closely linked.

Quality of design refers primarily to how well the software is designed and built. In a well designed software system, GRASP patterns are much more likely to have been followed/implemented, and the system is likely to have been designed with the GRAS patterns in mind. A poor quality of design system will not conform closely with these principles, and is likely to be designed in a somewhat opposing manner to these principles.

Quality of conformance refers to how well the system conforms to its intended purpose. This aspect of quality is tied quite closely to quality of design. If a system is well designed, it is likely that it will also conform quite well to the needs of the customer. Quality of conformance is more subjective than quality of design, and ultimately the quality of conformance metric is best judged by the customer. That being said, part of a developers’ job is helping the customer to understand what they need before the building process begins.

**Software Testing**

Software testing is a process by which parts, or all of a software system are tested in order to reduce bugs, improve usability, increase performance and reduce development costs later down the line by reducing or eliminating the need to re-develop parts of the system that are buggy or non-functional.

There are a few different types of software testing, the smallest of which is unit testing. A unit is the smallest testable part of a software system, more than likely a method. Unit testing aims to check that each unit performs their task as expected. In other words - checking that methods do what they were intended to do.

Usability testing is another facet of software testing that is important to perform. Usability testing is often performed by a small subset of users that conform to the demographics of the target audience and the testing is intended to highlight how user-friendly the system is.

Acceptance testing is one more aspect of software testing that is conducted to verify whether the whole system is working as intended. Acceptance testing will usually be the last phase of a testing cycle and is the phase at which the customer is able to have input and determine whether they accept the application/system, or whether they are not happy and want more work to be done.

**Construction**

The construction of the system was done incrementally, and with the assistance of evolving UML diagrams, such as the Class Diagram, while also keeping in mind the overall goals and design of the system that were set out in the Use-Case Diagram. GitHub was used as a repository for the code and the UML documents in order to keep track of the updates and changes. On top of this, to help with development, commenting of the code was frequent and intended to help the developer pick up where they left off.

An example of this would be the comments that are included at the top of some classes and above some methods to help illustrate and describe what the classes and methods do.

**Conclusion & Reflection**

In conclusion to this report, I believe a fit-for-purpose web based application was created for the purpose of booking covid tests, thereby fulfilling the project brief and satisfying the requirements set out.

On a personal note, as somewhat of a reflection on my own performance during this project, the development of this system has highlighted some major flaws in both my character and my ability when it comes to programming and the road towards being able to call myself a developer. As with any project, there were areas where familiarity and comfort were present, in the case of this project, these would be the SQL elements and the basics of the Java language.

Major flaws however reared their head when it came time to program in Spring. A lack of knowledge of Spring meant that I had to learn a lot regarding the language in a very short amount of time, combined with learning Spring Boot, and being relatively unfamiliar with HTML and CSS, the task was a momentous one.

As a result of the above mentioned lack of knowledge, a major character flaw appeared, in that I allowed this momentous task to overwhelm me, and instead of throwing myself into it to learn as much as I could, I instead perpetuated the problem by not starting.

In terms of the system itself, there are a few improvements I believe could be made, and will be worked on, even after hand-in is complete. For example, it should be possible to add testCenters, appointments and addresses within the system itself, under the management tab. This is currently possible to do for the user entity, but not for the other three. This is something I would add, and something I will be working on after this project is handed in.

Another improvement I plan to make is the ability to preview addresses when creating a user. As it stands right now, when the administrator creates a user (which is also currently the only way to add a user - more on that below) and is inputting their address, the administrator has to know the addressId beforehand, in order to assign the correct address to the User. An improvement I would, and plan to make, is that I would allow the administrator to select a dropdown menu which displayed all of the current addresses in the database. If possible, I would also like to add a functionality to add an address while creating a user, but I believe this is beyond the scope of my current capability.

One final improvement I believe could be made, is that when a new user attempts to log into the system, they would instead be directed to a new user page, where they can create a new user account for themselves, instead of relying on the administrator to create their user account for them. This would help to emulate the real-world, as Sundhedsstyrelsen already has information on most individuals, and you merely need to confirm your details in order to access their systems. In this case it would allow a user to emulate the ‘confirmation’ of their details, when in actuality they were creating a user account and logging in immediately after.

Overall, I believe the objectives of this project have been satisfied, and though the goal was achieved, there are some glaring issues that need to be fixed for the next project, which will likely be more challenging, but nonetheless more rewarding once the issues present during this project are rectified.

**References**

Chonoles, Michael & Schardt, James. 2003. *UML2 for Dummies*. Wiley Publishing Inc.

Larman, Craig. 2004. *Applying UML & Patterns*. *An introduction or object-oriented analysis and design and the unified process*. 2nd ed.

Murach, Joel. 2019. *Murach’s MySQL*. 3rd ed. Mike Murach & Associates Inc.