High Level Design Document

Online Doctor Visit Appointment (Web Application)

Shubham Gantayat, Suraj Kumar

Version 1.0

14/08/2021

Version Control

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Description | Responsible Party | Date |
| 1.0 | Initial version | Shubham Gantayat,  Suraj Kumar | 14-08-2021 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# Abstract

In these recent times, COVID-19 has affected the general population very seriously. People need medical care now more than ever. But to get an appointment they must go a hospital and get in a long queue to get a slot. And sometimes till you reach the counter the slots are all taken. Then you must repeat the same procedure at different hospital till you get an appointment. This application is made to help you with these problems so that you can check an empty slot for any doctor you want and book a slot. No waiting in long queues to reach the counter. Now you can stay home and stay safe and go only when there’s an appointment scheduled.

# Introduction

## Why this is a High-Level Design Document?

The purpose of this High-Level Design (HLD) Document is to add the necessary detail to the current project description to represent a suitable model for coding. This document is also intended to help detect contradictions prior to coding andcan be used as a reference manualfor how the modules interact at a high level.

* + 1. Present all the design aspects and define them in detail
    2. Describe the user interface being implemented
    3. Describe the hardware and software interfaces
    4. Describe the performance requirements
    5. Include design features and the architecture of the project
    6. List and describe the non-functional attributes like
       - Security
       - Reliability
       - Maintainability
       - Portability
       - Reusability
       - Application compatibility
       - Resource utilization
       - Serviceability

## Scope

The HLD documentation presents the structure of the system, such as the database architecture, application architecture (layers), application flow (Navigation), and technology architecture. The HLD uses non-technical to mildly-technical terms which should be understandable to the administrators of the system.

## Definition

|  |  |
| --- | --- |
| Term | Description |
| ODVA | Online Doctor Visit Appointment Application |
| Database | Collection of all information monitored by this application |
| AWS | Amazon Web Services |
| IDE | Integrated Development Environment |

# General Description

## Product perspective

The ODVA is a web application designed to help patients book appointments with doctors of their choice.

## Problem Statement

Create a web application to facilitate the functionality for the patient to book an appointment with a doctor online.

Approach: Implement the below feature in your application.

1. Doctor Registration portal. (Accept detail of doctors even their specialty, experience)

2. Allow patients to book an appointment with the doctor based on their needs.

3. Implement payment for each functionality.

4. Allow patient to Cancel appointment before 3 days.

5. Allow the patient to book an appointment within 15 days.

6. Every doctor can only accept 30 appointments.

7. If any patient tried and due to limitations of appointment they were not able to book an appointment suppose somehow if the booked appointment has been cancelled then all patient whosoever tried should receive notification regarding availability of appointment so that they can book again.

8. Allow the doctor to cancel all or reschedule appointment of a patient if somethings come up and notify same to all patient.

## Proposed Solution

The solution proposed here is an ODVA web application that allows patients to effortlessly book an appointment with their doctor within 15 days. After a slot is confirmed by a patient he/she is redirected to the payment portal where we the patient can pay using a variety of methods given, i.e., UPI, credit cards, debit cards, QR scanner, etc. All the payments are then transferred to the doctors account except a small amount of Rs.50 which we charge as a convenience fee. A patient in case of emergency can cancel his/her appointment before 3 days to appointment. In this case the patient will get a partial refund, i.e., the amount he paid minus our convenience fee. Doctor can also take a leave between for a specified period during which all patients can either be rescheduled or cancelled as per the choice of doctor. If rescheduled all the patients will be moved to the nearest empty slot and be notified via email and on the notification bar on patient homepage. For some reason if a rescheduling of all patients is not possible due to unavailability of slots the appointments are cancelled. The patients whose appointments are cancelled would get a full refund including the convenience fee.

Doctors on signup provides a proof of his/her medical license as an attachment. Unless the proof is verified the doctor is not allowed to take appointments. The doctor also must add his account details so that transfers can take place. Doctors must set their working time along with how much time they want to give per patient. A break time is also added which is optional. A doctor can only take up to 30 appointments a day.

Admin can activate and deactivate doctors accounts in case of a dispute or unfair use of our application.

## Further Improvements

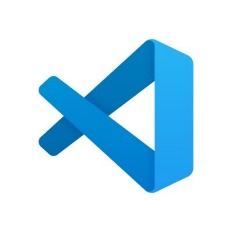
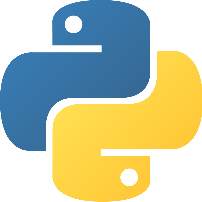
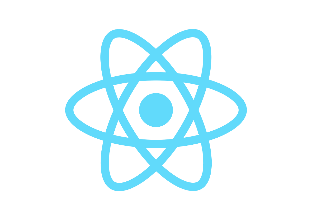
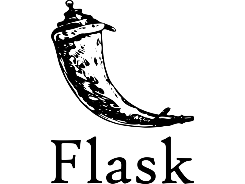
Filter options can be added to the pages so searching of appointments for a date can be easy and effortless. Also, NLP can be added to make search more effective, e.g., if a patient search for heart doctor he/she should get cardiologists as a result. We can also use machine learning to suggest doctors to patients based on their medical history.

## Technical Requirements

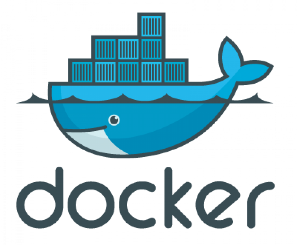
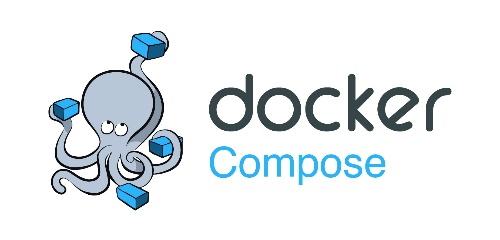
* + 1. Proper computing power to process requests and fetch data from server.

## Tools Used

* PyCharm and VS Code as IDE.
* Python Flask as backend.
* React as frontend.
* Apache Cassandra to retrieve, insert, create and update databases.
* Razorpay as payment portal.
* GitHub is used as version control system.
* AWS is used for deployment.
* NGINX is used for connecting flask to react using reverse web proxy.
* Docker used to containerize our application.
* Docker-compose is used to link the docker containers.



## Constraints

The ODVA web application must be user friendly, as automated as possible and users should not be required to know any of the workings.

# Design Details

## Process Flow

We have used a react based web application as frontend and flask as backend along with Cassandra as database. For more information on flow diagrams visit our software architecture document –

<https://github.com/shubhamgantayat/MedHub/blob/main/Documents/Software%20Architecture%20Document.docx>

## Event Handling

1. The System identifies at what step logging required

2. The System can log each system flow.

3. We have used database logging in Cassandra.

4. System does not hang even after using so many loggings. Logging is mandatory just because we can easily debug issues.

## 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.

# Performance

## Reusability

The code written, and the components used should have the ability to be reused with no problems

## Application Compatibility

The different components for this project will be using Python as an interface between them. Each component will have its own task to perform, and it is the job of the Python to ensure proper transfer of information.

## Resource Utilization

When any task is performed, it will likely use all the processing power available until that function is finished.

## Deployment

