Link to the Web App : https://aarogya-hetu.ml

DESIGN REPORT

Aarogya Hetu: Mercari's Large Scale System Design Hackathon

Team 10

Introduction

India is a large country with many diverse people and diverse needs. The health needs of such a country need a highly scalabale system that can track the health records as well as cater to the needs of the doctors and the multi-chain hospitals. We present a highly scalable and innovative solution, *Aarogya Hetu*, which can scale as per the needs of the hospitals, and provide the patients with some highly useful features like booking appointments, generating prescriptions, automation of slots, etc.

Aarogya is a Sanskrit word which means "overall well-being". *Hetu* is also a Sanskrit word which means "for". The name of our application signifies that we have built this application for helping with the overall well-being of its users.

With this main goal in mind, we can derive the requirements from our app. We essentially wanted the following features from our app:

- Users should be able to request appointments with doctors or labs, and the appointments should be automatically scheduled based on the schedule of the doctor or lab.
- Users should be able to view their own health records, and doctors should be able to view the records of those patients who have an appointment with them.
- Doctors and labs should be able to upload test results, prescriptions and other health records for patients.
- Users should be able to see their medical bills at one single location, and hospitals should be able to see the automatically generated bills for any patient.

Besides these main goals, we also needed to have management capabilities for handling hospitals, labs, doctors etc.

Our application Aarogya Hetu achieves all the required goals, with a highly scalable architecture. In this report, we shall explain the details of our design and implementation.

We can effectively divide our report into three elementary units:

- **Entities**: This section covers the basic entities in our application and their uses.

- **Microservices**: This section covers the fundamentals of our design architecture, and the different microservices we have.
- **Containerization**: This section throws some light on our containerization method.
- **Continuous Deployment**: This section will describe how we use GitHub Actions for continuous deployment.

Design Architecture & Microservices

We focus on a microservice based architecture in order to ensure the scalability of our solution. To get started with the problem, we first identified the different entities which are present in a Hospital Management System. To the best of our efforts, we managed to come up with the following four entities :

- Patients
- Doctors
- Hospitals
- Labs

The next order of business was to determine the different functions of each of the entities. We explored the different possible functions of each of the entities, and came up with the following functions :

Patients: The patient should have an option to register their NHIDs (when onboarding for the first time), so as to ensure the addition of new patients. Some other information like Patient Name, Phone, Email and Aadhar Number are also required at this step. Once this step is done, the patient is assigned an NHID. Some of the other functions of the Patient includes viewing historical reports and booking appointments from doctors/labs

Doctors: The doctors can have the option to delete an appointment, as well as block their own calendars (so our algorithm doesn't schedule an appointment in that slot). We provide the patients with a feature to choose their own time slots(if free) for a particular doctor, of a custom duration. Security is of utmost importance in our application, so the doctors have access to the past medical records of only those patients who have booked an appointment under them. The doctors also have the option to refer the patient to another doctor. The doctor has the abilities to generate prescription, as well as request lab tests for a particular patient.

Labs: The labs serve as separate entities in their structural form, but are considered to be affiliated to the hospitals. The labs have the methods to upload the results of a patient, and to schedule lab appointments of a patient.

Hospitals: The Hospitals can have the option to view the patient bills, and collect them. The hospitals can manage the doctors, that is, add them, or remove them. Hospitals can also manage the beds and other inventory items, thus allowing them to take note of their current occupancy.

Microservices

On the basis of the above discussion, we concluded to have the following microservices in our Application:

- NHID
- Filter Systems
- Appointments
- Bills
- Calendar
- Main Server

A comprehensive detailed analysis of each of the microservices, along with the endpoints in those microservices can be found in the section below:

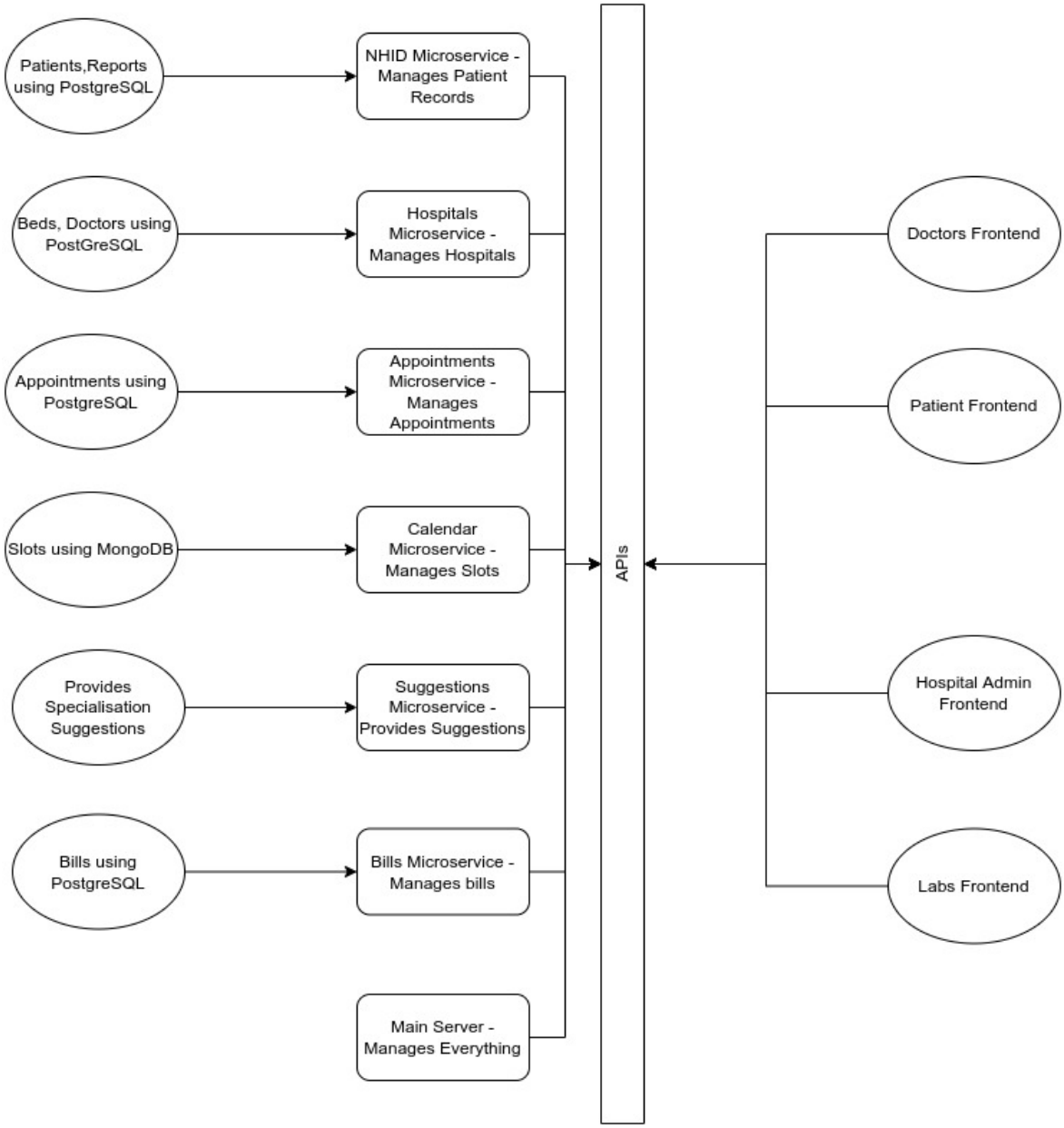
NHID

The NHID microservice deals with the management of the patient records and the general information of the patients. The microservice uses MongoDB as a database, owing to the nature of queires which the microservice was going to receive. The database for this microservice has 2 tables, namely PatientInfo and Reports. This needs to be a microservice in its own right because this data may be stored centrally by a government organisation, thereby separating its concerns from the other features of our app.

Specialization Filter Suggestions

The patient, in a general case, books an OPD appointment, if he doesn't know the nature of the disease he is suffering from. This leads to some redundancy in the cycle, as

Fig. 1. System Architecture



the OPD doctor then finally directs the patient to the specialist of the disease. We eliminate this redundancy through our specialization suggestion microservice. This microservice has a predict disease endpoint, which takes in the symptoms input from the user, and predicts the department of the specialist which the patient should consult to.

Since this is just an implementation of an ML based model, it doesn't have any update/insert queries. It, however, needs to handle a large number of requests as fast as possible. So we don't need a large storage for this feature, but faster processing. Hence, this is a microservice in its own right.

Appointments

The Appointments microservice deals with the process of booking and modifying appointments for the doctors and the labs. The appointment microservice interacts with the calendar microservice in order to provide a full in-sync scenario to the patients as well as to the doctors. When a patient tries to book an appointment with a doctor, he/she provides a start time and duration of the consultation. This

request is then routed to the appointments microservice, which further routes it to the calendar microservice, and checks if the slot is available or not. If yes, the patient then books an appointment, and the corresponding entries are made to the calendar database as well as to the appointment database. The microservice uses PostgreSQL as the database.

Bills

The Bills Microservice contains the methods to get the bills for a particular patient, as well as to add additional charges for a patient. The microservice uses PostgreSQL as the database, owing to the highly dynamic nature of the database.

Calendar

The Calendar Microservice contains the methods to check and fill slots for the doctors as well as labs. This Microservice uses MongoDB as the database, owing to the relational structure of the data. This database has a separate document(MongoDB) for each doctor, as well as contains methods to add a new doctor database, in order to make the system scalable. Currently, this is our own custom built microservice. But it is possible to implement the same using Google calendar, or other calendar apps. Such modifications would be major changes for the user, but it would not affect how other microservices interact with this microservice.

Frontend Server

The Frontend server is for deploying the compiled frontend code for the web app.

Some Additional Services

File Server: This server is used to upload and manage the files onto the server. The server contains 2 endpoints. The first endpoint is used for uploading the file onto the server, and it returns a file ID. Once a file has been uploaded, this fileID is then stored in the NHID database of the patient. The second endpoint is used to retrieve the file, by appending the fileID to the URL We used an open source tool `skx/sos` for this. THis needs to be a microservice on its own, because instead of this, we may use AWS S3, or Azure Storage or some other storage service, and our app would not be affected by the change.

Authentication We perform Authentication using the JSON Web Token. This is because there is no centralized database across the microservices. So JWT becomes the authorization method of choice to enable different microservices to verify a user's identity. When the patient/doctor/hospital first logs into the server, an authentication check is performed using the login endpoint of the server. If the authentication is successful, the server provides with a cookie, which is then used subsequently in combination with a public key(obtained from the server) to perform authentication.

Continuous Deployment

We are using GitHub Actions for Continuous Deployment. Since we are using AWS EC2 for running the microservices, we needed to be able to automatically publish the code from the GitHub repository to the EC2 VM, and then compile and deploy it on EC2 itself. For this purpose, we made use of two open source GitHub Actions:

- `easingthemes/ssh-deploy` : This is used to copy the code from the GitHub repository to the EC2 machine through rsync.
- `appleboy/ssh-action` : This is used to execute the build scripts on EC2 once the code has been synced.

Each one of our microservice repositories has a GitHub Actions Workflow defined, which is executed whenever there is a new commit on the main branch. This ensures that we are always running the latest tested code, and our features are shipped to the users as soon as possible.

Conclusion

Aarogya Hetu is a fully functional and scalable system for helping the healthcare industry of India. It would be really beneficial for all of us if something like this is actually built and used in India.