

---

# Design and Development of SMS based Health Services for Urban Poor and Chat Service Integration

---

## **M.Tech Project Report**

*Submitted in partial fulfillment of requirements for the degree of*

## **Master of Technology**

by

**Chinmay Prakash Kulkarni**

**(20305R004)**

*under the guidance of* **Prof. Kameswari Chebrolu**



Department of Computer Science and Engineering  
**INDIAN INSTITUTE OF TECHNOLOGY, BOMBAY**

## **ABSTRACT**

We live in the Information Age, where the exchange of information plays a crucial role in the development of a nation. While smartphones and smart devices play a significant role in this knowledge exchange, in developing nations such as India, access to smartphones is restricted chiefly to only a segment of the population within urban areas. On the other hand, mobile phones with SMS facilities are more widely available. While they can help with information exchange in a variety of scenarios, health care is one very important area.

Many NGOs work for this specific purpose. SNEHA foundation is one such NGO and works prominently in Mumbai region in 7 corporations. In this project, we design and build a health check-up appointment booking system based on the SMS service. Our solution offers a simple way for field agents to book appointments for users with health issues, by sending an SMS to a Virtual Mobile Number. We design the system such that it also simplifies the interaction between the members of the SNEHA foundation working in different roles. As part of this work, we built an android application for the SNEHA officers to help them keep track of tasks. Additionally, we have designed a dashboard that allows SNEHA admins to monitor the entire workflow. Along with this, we have integrated a chat application for the SNEHA admins and the clients to have seamless communication.

# Acknowledgments

I would like to thank Prof. Kameswari Chebrolu for providing me the opportunity to be a part of this huge and exciting project and also for her continuous guidance which helped me a lot in working on this project. The frequent meetings with her throughout the semester were really fruitful. Also, I would like to express my gratitude to the SNEHA members for their coordination and support.

# Contents

<b>List of Figures</b>	<b>vi</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Problem Statement . . . . .	2
1.3 Key Challenges . . . . .	3
1.4 Outline . . . . .	4
<b>2 Previous Work</b>	<b>5</b>
2.1 SNEHA foundation . . . . .	5
2.2 Roles . . . . .	6
2.3 Hierarchy of Roles . . . . .	7
2.4 Working of SNEHA . . . . .	7
2.5 Drawbacks . . . . .	8
<b>3 Methodology</b>	<b>10</b>
3.1 Scope of this project . . . . .	10
3.2 Proposed Solution . . . . .	10
3.3 Architecture . . . . .	11

3.3.1	System diagram . . . . .	11
3.4	Workflow . . . . .	13
3.4.1	Data Preparation . . . . .	14
3.4.2	Appointment Booking . . . . .	14
3.4.3	Post Appointment Booking . . . . .	16
3.4.4	Task creation and Chat service . . . . .	18
<b>4</b>	<b>Implementation</b>	<b>25</b>
4.1	Background Technologies . . . . .	25
4.1.1	SMS third-party service . . . . .	25
4.1.2	Django . . . . .	26
4.1.3	Django Channels . . . . .	26
4.1.4	Svelte . . . . .	28
4.1.5	SvelteKit . . . . .	29
4.1.6	Websockets . . . . .	29
4.2	Modules . . . . .	30
4.2.1	Centralized server . . . . .	30
4.2.2	Android . . . . .	41
4.2.3	Chat Service . . . . .	43
<b>5</b>	<b>Future Work</b>	<b>47</b>



# List of Figures

2.1	SNEHA hierarchy . . . . .	7
2.2	Referral slip . . . . .	8
3.1	Architecture . . . . .	12
3.2	SMS format sent by Agent . . . . .	15
3.3	SMS format received by Patient . . . . .	16
3.4	Appointment booking process . . . . .	16
3.5	WhatsApp Reply . . . . .	18
3.6	Form or Chat . . . . .	19
3.7	Fill Form . . . . .	19
3.8	Sign in page . . . . .	20
3.9	Task list . . . . .	21
3.10	Chat Window . . . . .	22
3.11	Message Input . . . . .	22
3.12	Accepted Task List . . . . .	23
3.13	Accepted Task . . . . .	23
3.14	Chat . . . . .	24

4.1	Dashboard Schema . . . . .	33
4.2	Auth Schema . . . . .	33
4.3	MSGs Schema . . . . .	34
4.4	Main page template. . . . .	34
4.5	Input facility page. . . . .	35
4.6	View all facilities. In each row there is an edit button using which one can edit the facility details . . . . .	35
4.7	Input case type . . . . .	35
4.8	View all case types. Every row has an edit button which can be used to edit the details. . . . .	36
4.9	Input SNEHA master. . . . .	36
4.10	View all SNEHA masters. Every row has an edit button which can be used to edit the details. . . . .	37
4.11	Input Agents . . . . .	37
4.12	View all Agents. Every row has an edit button which can be used to edit the details. . . . .	38
4.13	Input slots . . . . .	38
4.14	View all slots. Every row has an edit button which can be used to edit the details. . . . .	39
4.15	View all valid tasks . . . . .	39



4.16 View all flagged tasks. These are the tasks that are marked invalid by the system . . . . .	40
4.17 Dashboard: Here the user can see all the categories of the tasks. . . . .	41
4.18 Sidebar: Menu with options to explore. . . . .	41
4.19 Accepted tasks: Here the user can see the list of accepted tasks. . . . .	42
4.20 Accepted: Here the user can see the details of an accepted task after clicking on it. . . . .	42
4.21 Completed tasks: Here the user can see the list of completed tasks. . . . .	43
4.22 Completed task details: Here the user can see the details of a completed task after he clicks on it. . . . .	43

# Chapter 1

## Introduction

India has progressed a lot in use of technology in recent years. For example, the increase in digital transactions via means like UPI, the usage of DigiLocker systems and so on. India has seen a sharp increase in smartphone users and as per a report by Deloitte[1], the number of smartphone users in India is predicted to reach 1 billion by 2026. Despite this, a substantial segment of India is still not benefited by it. While the number of smartphone users has increased, the percentage of people with access to smartphones is not satisfactory, especially since many people living in rural areas or the suburbs have very limited or no access to smartphones. Even in the cases where it is available, due to a lack of education and awareness about it, people fail to use them efficiently.

### 1.1 Motivation

- **Lack of Awareness/Education:** The growth of the country can only happen when people have access to information related to health, education, agriculture, etc. A large segment of people in India is not exposed to this information because of a lack of awareness or lack of education.
- **Helping NGOs:** NGOs play a crucial role in providing education and medical counseling in suburban and rural areas. There are volunteers in these NGOs who visit different places to spread awareness among people. But visiting places and spreading

awareness is hardly scalable.

- **Need for Digitization:** Also, after the COVID-19 pandemic hit the world, many organizations realized the importance to go online and digitize themselves. May it be government institutions, educational institutes, or NGOs. They faced various difficulties in communicating and meeting with people.
- **Significance of Healthcare:** Healthcare is a basic need for every citizen, be it a person from the urban or rural class. It is not only to be made accessible, but also needs to be affordable. Technology is one of the best ways to make this happen.
- **Easy communication:** Helpers and clients should be able to communicate easily without using any third-party software. A chat application can suffice the need.

This suggests the need for a scalable, easy-to-use solution that can provide the NGOs the necessary support to make healthcare accessible via technology.

## 1.2 Problem Statement

This project focuses on the design and development of health care services with reliance over the widely available technology (SMS) that makes it easily accessible to the urban poor. The vision has always been to make the system cost efficient and easily scalable. As part of this work, the following modules were to be developed

- A health-care appointment booking system, that maps patient cases to the earliest available time-slots in healthcare centers when ASHA agents book appointments.
- An android application through which SNEHA officers can access the data that belongs to their corporation and keep track of their tasks.

- A web-based dashboard for SNEHA admins to monitor all the tasks.
- A form through which clients can create a task.
- A chat service that can help ease communication between client and helper.

## 1.3 Key Challenges

To design such a system that is cost-efficient and is also easy to use gives rise to certain challenges.

- Choosing the medium: When it comes to some digital medium, the first thought that anyone would get is, a mobile phone. And it is right. Mobile will surely do. But even on a mobile phone, we have options like SmartPhone or feature phones. Even though no of smartphone users is increasing drastically, currently there are many people who don't have access to any sort of smartphone and hence we have to rely on feature phones to serve their needs.
- Now the options that we have are IVRs(Interactive Voice Response) and SMS. IVR and SMS can both serve our purpose. But in this particular case, SMS is found to be more convenient to the ASHA worker/Helper as it is asynchronous.
- For the helpers, they can have an android application or a website that can help them monitor the work.

## 1.4 Outline

The report is organized as follows. Section 2 contains the previous work, that explains how the process was earlier, its architecture and working and drawbacks. Section 3 contains the current work, explaining the architecture and working details. Section 4 explains the background technologies that have been used and other implementation details. Section 5 provides future work of the project

# Chapter 2

## Previous Work

In this chapter we describe the SNEHA foundation's working and architecture that existed to achieve their goals, prior to using our developed system

### 2.1 SNEHA foundation

SNEHA believes that for the development of a community, it is important to have health equity[2]. And, this can be done by strengthening the capacity of the community to access health services while working with public health systems for providing quality services. There are also other ways like imparting necessary and related knowledge to them and spreading awareness which can certainly help in changing attitudes and behavior. SNEHA is working for bridging the gap between health services and the needy, and is not just confined to the spaces of fighting an illness but is also about raising general awareness and health in public. SNEHA works for different people and serves different purpose as following:

- Pregnancy:
  - Enhancing public system response to maternal and newborn healthcare
  - Improving coordination across public health facilities for high risk pregnancies.
- Birth

- Encouraging institutional deliveries through the public health system
- Ensuring full immunisation in children and exclusive breastfeeding
- Childhood
  - Identifying, preventing and treating malnutrition
  - Capacity building to deliver better public services for child health and nutrition
- Adolescence
  - Educating adolescents on nutrition, reproductive health and gender equity and improving emotional resilience
  - Creating a cadre of youth to become community change agents
- Adulthood
  - Empowering communities and partnering with public systems to respond to gender-based violence
  - Providing mental health and legal services to women facing violence, and improved knowledge and access to planned parenthood

## 2.2 Roles

There are various different roles for SNEHA members with every role having its own responsibility. The roles and responsibilities are as explained below:

- Project director (PD)
- Associate project director (APD)

- Program coordinator (PC)
- Program Officer (PO)
- Community Organizer (CO)

## 2.3 Hierarchy of Roles

The hierarchy of the roles is as shown in the diagram.

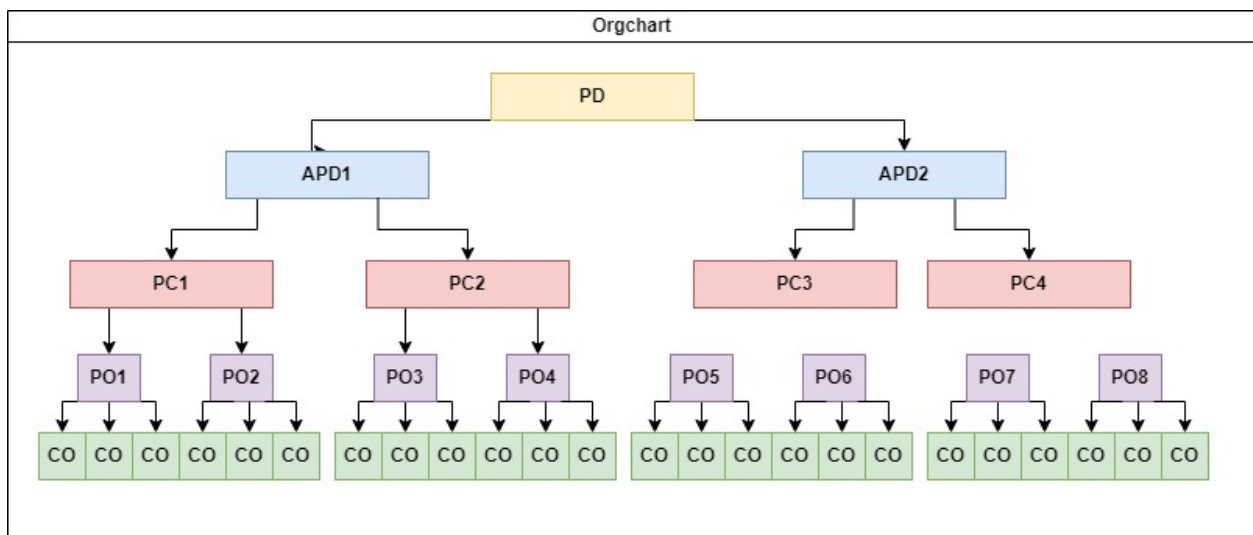


Figure 2.1: SNEHA hierarchy

## 2.4 Working of SNEHA

The workflow of SNEHA can be described in following steps:

- The community officers go to different places and try to identify different cases in the community.



- Then, these community officers identify the case types of these people and then give them a referral slip.


 <b>सोसायटी फॉर न्यूट्रिशन एज्युकेशन अँड हेल्थ अक्शन (स्नेहा)</b>	
सेवा सुविधा का नाम	तारीख :-
बस्ती का नाम	
महिला का नाम	
किस विधे भेजा है ?	<input type="checkbox"/> ऐन.सी. रजिस्ट्रेशन <input type="checkbox"/> ऐन.सी. चेक अप <input type="checkbox"/> पी.एन.सी फॉलो <input type="checkbox"/> टिकाकरण
	<input type="checkbox"/> परिवार नियोजन <input type="checkbox"/> आय. क. ऐ. मोलिया <input type="checkbox"/> अन्य
सी.ओ. का नाम	सही

Figure 2.2: Referral slip

- Now, the patient is supposed to take this referral slip to the hospital recommended by the community officer.
- For this, the community officer should also know the available dates of appointments for the given case type in all the health posts.
- The process ends when the patient visits the health post.
- The other SNEHA members are responsible for monitoring the entire process.

## 2.5 Drawbacks

Following were the drawbacks of the older system:

- Every SNEHA community officer is supposed to visit different places and identify different cases. Travelling to different places and carrying out this process is cumbersome, and is also inefficient as one community officer might not reach every needy person.
- Tracking patients is very difficult. After booking an appointment for a patient, knowing whether the patient visited the hospital and communicating it with the higher authorities is very difficult.
- Maintaining a record of all the cases handled, all the patients visited, etc, is very cumbersome.
- In case if there is a change in the appointment schedule, it becomes very difficult to communicate it to all COs as they are the ones who schedule the appointments with beneficiaries.

# Chapter 3

## Methodology

### 3.1 Scope of this project

As studied in [2](#), the previous model had its drawbacks and to enable digitization, a few changes to the previous workflow are required. Also, for the digitization, we need to design the architecture and the database.

### 3.2 Proposed Solution

To fulfill all the requirements, we have categorized the user base in 4 categories. And handled all the three of them differently. The categories of different bodies involved in SNEHA are:

- SNEHA officers(COs POs)
- SNEHA admin
- Agents(ASHA workers)
- Patients/Clients

We created 3 platforms for these 4 different roles. These three major parts are:

- SMS: This is designed for the Agent and patient side

- Dashboard: This part is designed for the SNEHA admins for monitoring and also for the clients to chat.
- Android app: This part is designed for SNEHA officers

## **3.3 Architecture**

The architecture includes two parts, the system architecture and the database design. In the system architecture we'll see all the different components and how these components interact to make the app work. While in the database design we'll slightly dive into the details of implementation to see how the database schema for these components are made to achieve the desired output.

### **3.3.1 System diagram**

The system consists of various components as listed below.

- Patient
- Agent
- SNEHA Admin
- SNEHA Officer
- SMS service
- Server

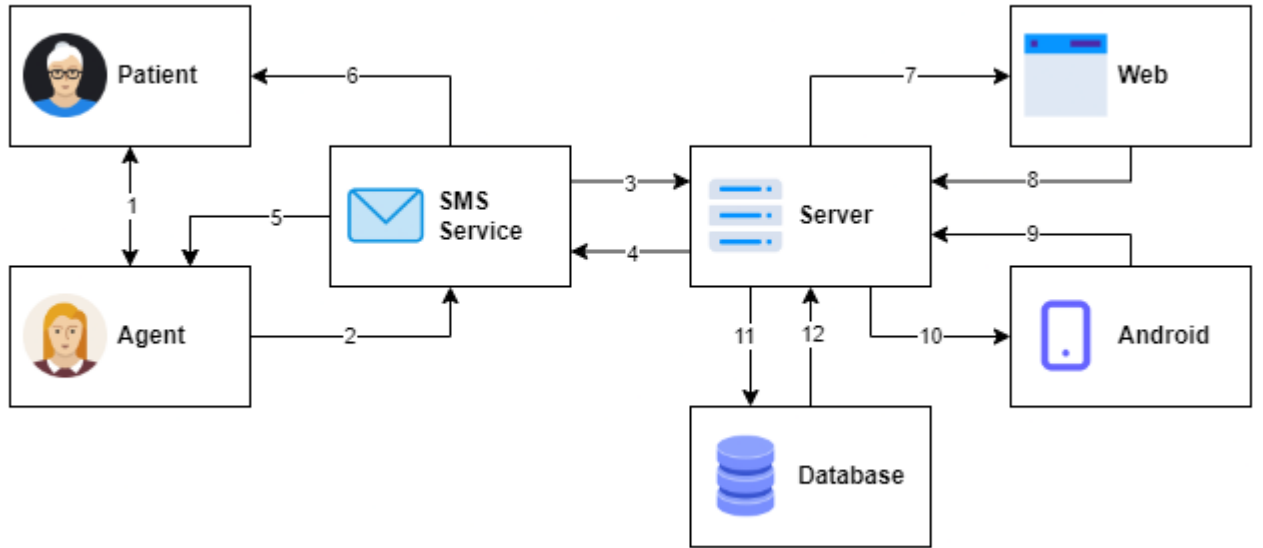


Figure 3.1: Architecture

## Patient

The patient is identified by an agent and is registered in the system by the agent. For this the agent sends an SMS to the VMN that belongs to the SMS service provider. Now this service provider makes a GET request to our server and we get the details of the SMS. The server parses the SMS and performs a tasks accordingly.

## Agent

Agents are preregistered in the system. Every agent has an auto incremental agent id using which one agent can be identified uniquely. Agent is responsible for adding the patient in the system. An agent should know the entire process, he should at least be aware of all the case types, their identification and the short codes designed for each of them.

## **SNEHA Admin**

SNEHA admins are involved in the database creation and in monitoring the working in all the corporations. They use the website and have access to all the database.

## **SNEHA Officer**

SNEHA officers monitor the corporations that are assigned to them. In case of any unhandled cases, they play a major role. Like in case if an agent sends an SMS with some error in it, or in wrong format, then an officer can see this on the dashboard. He can see the details of the agent and the patient. He can contact the agent to resolve the issue.

## **SMS Service**

This is the SMS service which sends the SMS data sent by agent to server and vice-versa. For an SMS to be sent via this service, one has to register the template of the SMS with them. Then the SMS is sent only if it follows the template else discarded.

## **Server**

Server is the main component throughout, which stitches the entire system together and also manages the database and the chat service.

## **3.4 Workflow**

There are three modules in the platform. SMS part, Dashboard, and android application. We'll see how these all work together to make the application work.

### 3.4.1 Data Preparation

For the system to work it needs data. The data is input by SNEHA admins or officers mostly. The data can be added in the following way.

- **Corporations:** Firstly, we add corporations to the database. This includes the details of the corporation like name, address, and Area.
- **Facility:** After adding the corporation, we need to add the facilities that belong to those corporations. In facilities also add the details like address, name, etc. This address is sent to the patient when an appointment is scheduled for the facility.
- **Agents:** Then we have to add agents, one of the most important parts of our system. Every agent has details as shown in figure [4.1](#). Every agent belongs to one or more facilities.
- **Casetype:** This is the case and the case type/code that identifies the case uniquely. Which can be DS for Danger sign, IM for Immunization, etc.
- **SNEHA Master:** There is also an option to add other SNEHA master users to distribute the data addition task. These users are officers that can monitor the process via a dashboard or Android app.

### 3.4.2 Appointment Booking

- **SNEHA officers:** They have the responsibility to add available slots for different HPs. For adding a slot, the required details are the health facility, case type, time, and date.

- **Agent:** Agents are responsible for initiating the process of appointment booking. These agents first identify needy people from society. These people identify the case of the patient.
  - The next step is to get their details like phone number, Name, Case Type, and health post.
  - Now the Agent has to send a message to our NGO number in a given format.
  - The format is:

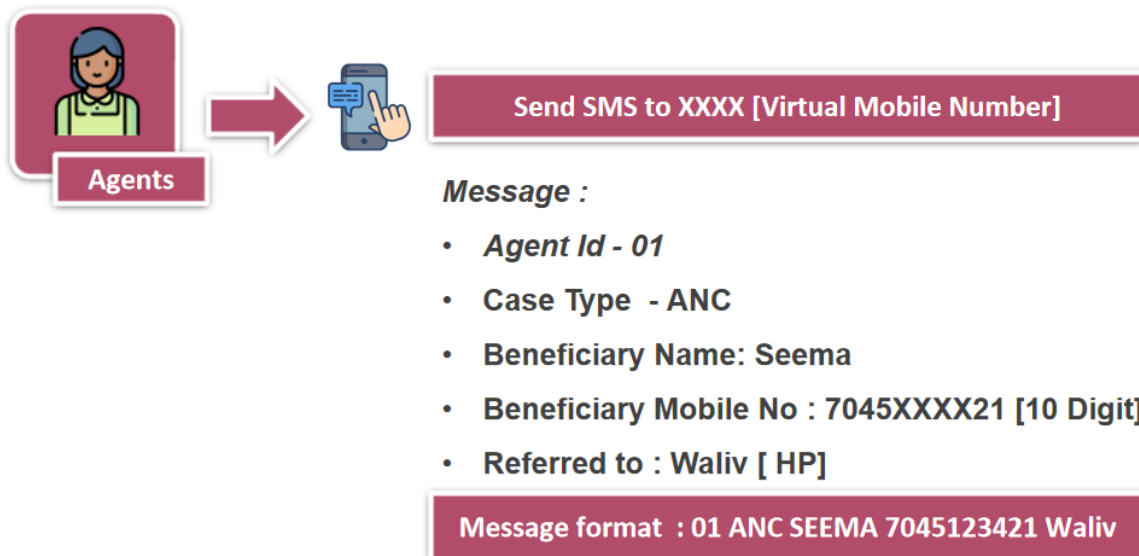


Figure 3.2: SMS format sent by Agent

- **Patient:** Once the SMS is sent to the number, the server checks if there is any slot available for the requested date, facility, and case type. If there is any slot available, the server sends an SMS to the patient. The SMS has the details of the appointment like the date, time, and address of the health facility.



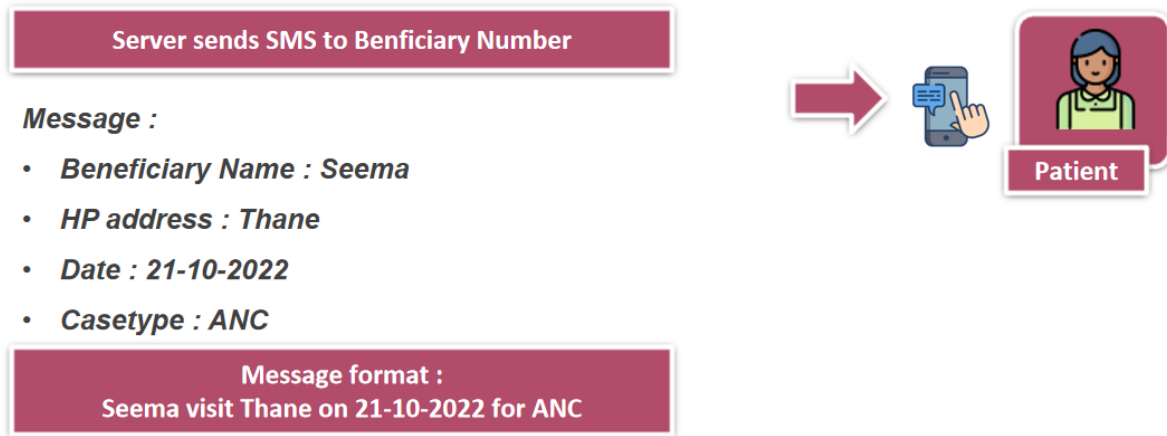


Figure 3.3: SMS format received by Patient

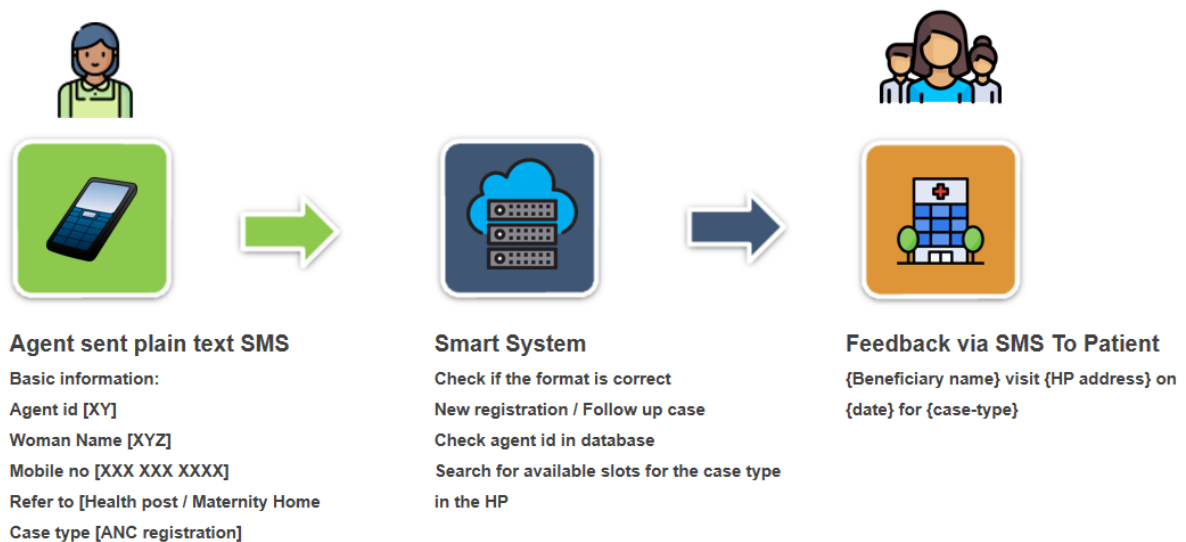


Figure 3.4: Appointment booking process

### 3.4.3 Post Appointment Booking

- **Tracking**

- After an appointment is booked, the important part is tracking the patients. For this, every sneha officer can see all the created tasks, also all flagged tasks.

Here, in all tasks, all the correct tasks are shown and in flagged tasks, the ones which have some issues are shown. The sneha officers can contact the agents who created those tasks and get the issues resolved.

- The android app can also be used for monitoring. In the android app, one can see all the tasks belonging to the corporations that are monitored by the sneha master.

- **Follow-up**

- At the end of every day, a cronjob runs and checks all the available slots of the day.
- For every slot on that day, a message is sent to the agent who created the task, asking an update of the task status.
- The agent is supposed to reply to the SMS in a given format.
- The reply SMS is of the following format:

**< -1 status-code phone name tracking-id >**

- \* -1 signifies that the SMS is sent for follow-up and not first-time booking. If the agent code is -1 the server considers this as a follow-up task.
- \* Where the status code can take values SC, RS. SC stands for success and RS stands for reschedule. If the status code is SC the task is marked as completed. Else if the code is RS, the appointment is rescheduled.
- \* A tracking id is also sent along with the message from the server. This tracking id is used to identify the reply to the SMS.

### 3.4.4 Task creation and Chat service

- Our organization has a dedicated WhatsApp business account that is set up and registered.
- When a client wants to communicate with us, they send a message to our WhatsApp business account. This can be done through the WhatsApp application on their phone or any other device that supports WhatsApp.
- In response to the client's message, our system generates a reply. This reply contains a URL (a clickable web link) and a pair of credentials (login details) that are specific to the client.
- The client receives the reply, which includes the URL and credentials. They can access this reply through the WhatsApp application on their device.

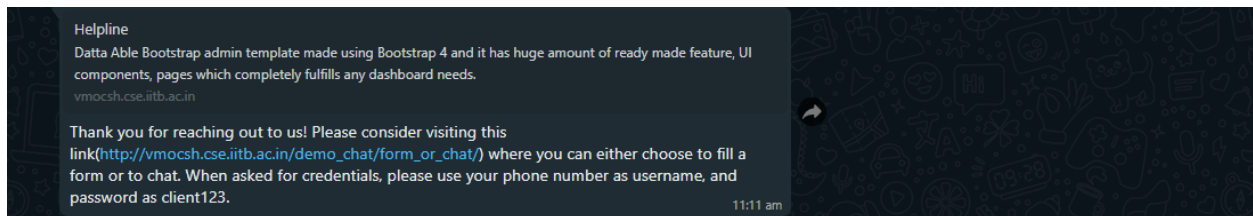


Figure 3.5: WhatsApp Reply

- To continue the conversation, the client clicks on the URL provided in the reply. This URL directs them to a website that is designed specifically for this chat feature.
- Upon arriving at the website, the client is presented with two options, Fill out a form or Chat.

1. Filling form:

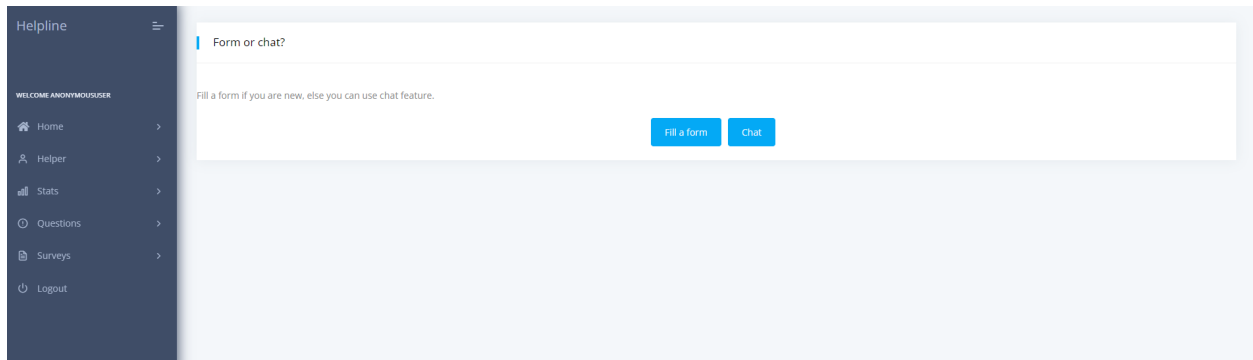


Figure 3.6: Form or Chat

- (a) The client is prompted to enter information such as their phone number, name, and select a category that best represents the nature of their request or issue.
- (b) Once the client fills out the form and submits it, the system processes the information provided.

Figure 3.7: Fill Form

- (c) Based on the submitted details, a corresponding task is created within the system.
- (d) This task serves as a record of the client's request or issue, containing the relevant information they provided in the form.
- (e) The task can then be assigned to an available helper who is qualified and

capable of addressing the client's specific needs.

- (f) The assigned helper can access the task details, review the client's information, and begin working on resolving the request or providing assistance.

## 2. Chat:

- (a) After the client opts to use the chat feature, they are prompted to fill in the credentials provided to them in the WhatsApp reply.
- (b) The client enters the credentials, which may include a username and password, to log into the chat system.

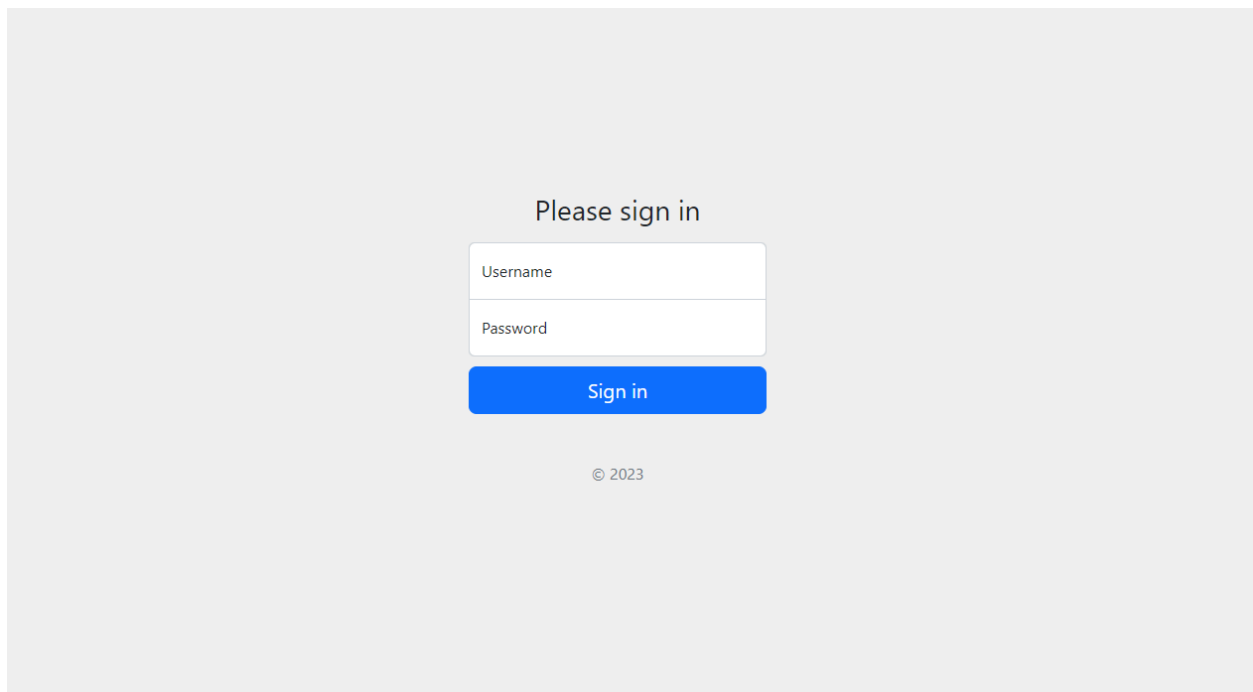
The image shows a sign-in page with a light gray background. At the top center, the text "Please sign in" is displayed. Below this, there is a white rectangular form with two input fields: "Username" and "Password". Below the form is a blue button with the text "Sign in". At the bottom center, there is a small copyright notice "© 2023".

Figure 3.8: Sign in page

- (c) Upon successful login, the client is presented with their personal dashboard or task management interface.
- (d) Within the dashboard, the client can view the tasks that have been created by them.

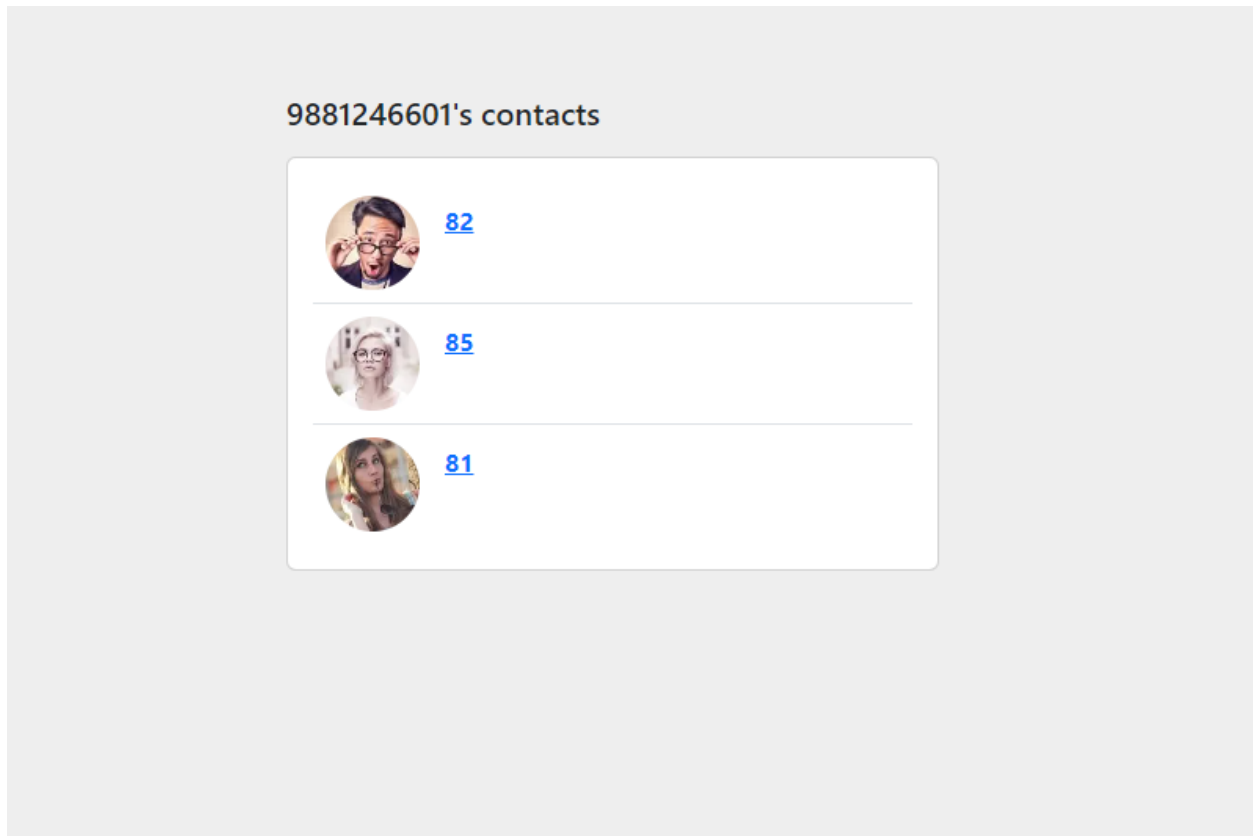


Figure 3.9: Task list

- (e) The client can select a specific task from the list, and upon clicking on it, a chat window is opened.
- (f) The chat window displays the entire history of the conversation that has taken place for that particular task.
- (g) In the chat window, the client can send and receive text messages, allowing for real-time communication with the assigned helper.
- (h) Additionally, the client can exchange file messages, such as documents, images, or any other relevant files, within the chat window.
- (i) The chat interface provides a user-friendly environment where the client can interact with the assigned helper, ask questions, provide updates, or receive

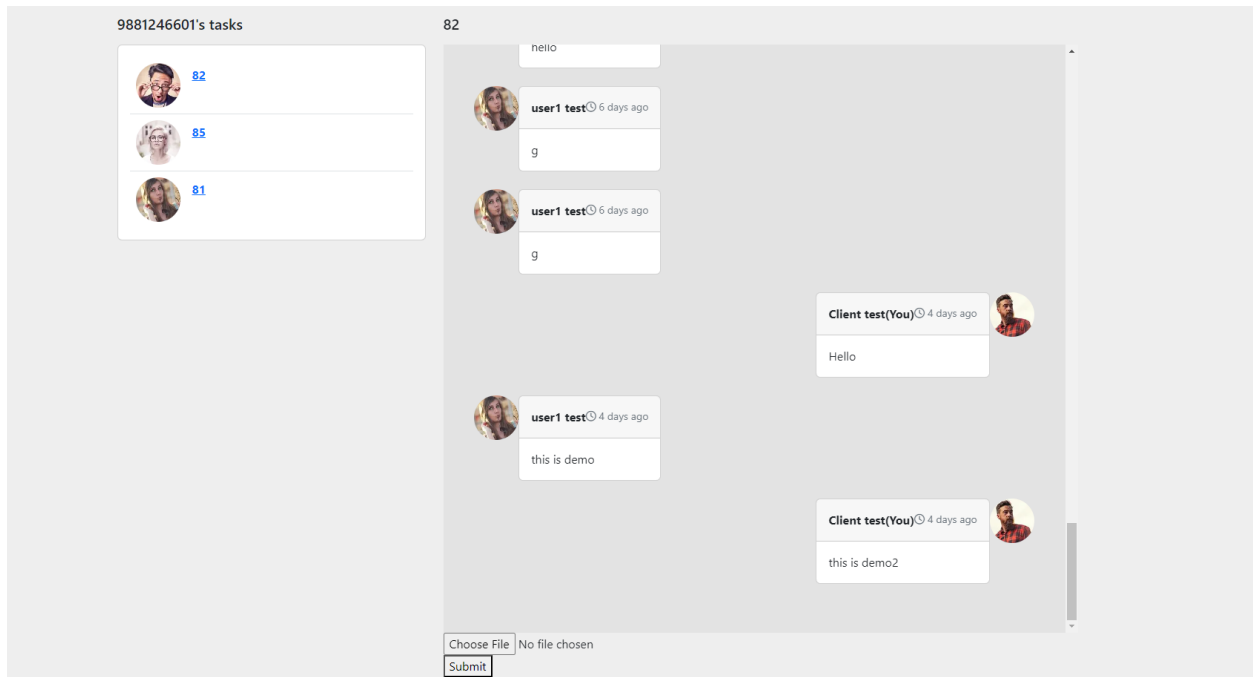


Figure 3.10: Chat Window

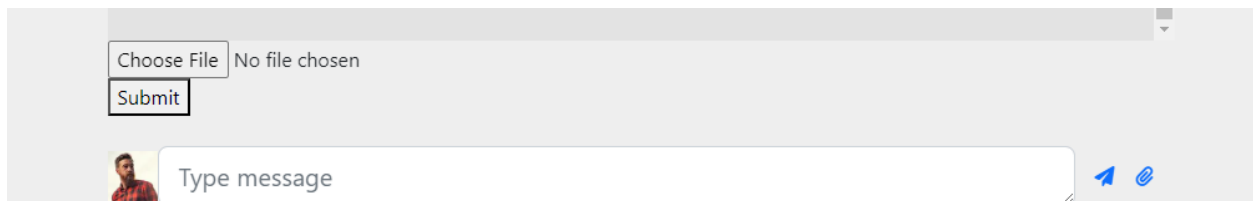


Figure 3.11: Message Input

guidance and support.

- (j) The chat feature ensures that all communication related to a specific task is organized and easily accessible within the chat window, allowing the client to refer back to previous conversations if needed.
- The helpers can also chat to these clients through the Android app. The clients can use the same app to chat with the helpers. The accepted tasks have all the tasks listed.
- There is a chat option on the accepted task page for each task.

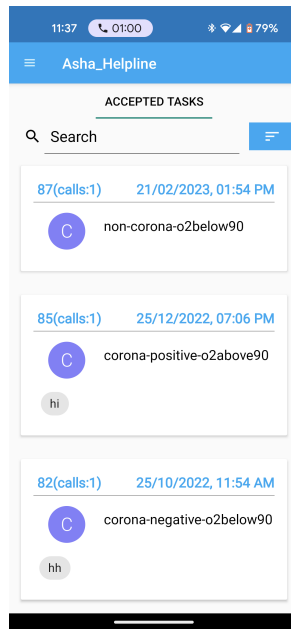


Figure 3.12: Accepted Task List

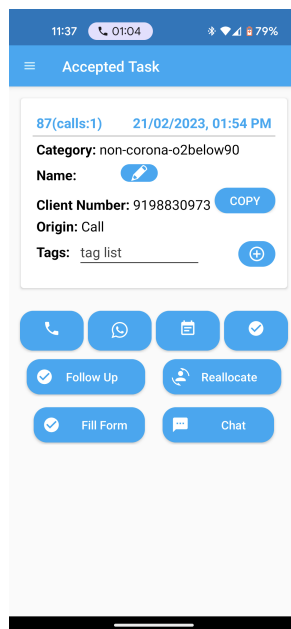


Figure 3.13: Accepted Task

- Once helper clicks on this chat button, new page is rendered along with the chat history specific to the task.



- Also the text input box is there to send messages.

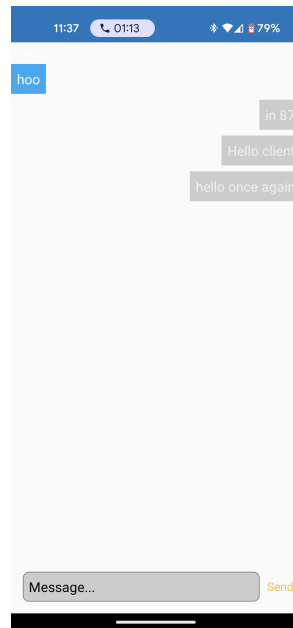


Figure 3.14: Chat

# Chapter 4

## Implementation

This section speaks about the implementation of the centralized server and the android app. This also includes the dashboard end. We have used the SMS service and integrated it with the task management system that is developed in Django.

### 4.1 Background Technologies

In this part, some of the tools and technologies that are used, are explained.

#### 4.1.1 SMS third-party service

These kind of services help us get the details of an SMS sent to a number that is connected to our server. The number has to be pre-registered in the SMS service provider, and it acts as a Virtual Mobile Number(VMN). There are two types of SMS, inbound and outbound SMS. For outbound SMS we have to register a template. So, whenever an outbound SMS request is sent to any number from VMN, the provider first checks if the formats matches with the registered template, if it does, then only SMS is sent else it is rejected.

### 4.1.2 Django

Django is a web development framework built in Python[3]. In Django, the famous Model-View-Controller (MVC) paradigm has been implemented as Model-Template-View (MTV) paradigm. Django enables us to create complex database driven websites by providing a rich set of libraries which helps in rapid development of these websites. Django is highly scalable and provides lots of in-built security mechanisms such as validation of users, avoiding Cross Site Request Forgery(CSRF) etc.

- Models in Django are python classes that help us manage database-related information without actually writing any SQL query.
- Templates are the HTML pages that are rendered based on the URL accessed. Django has a in-built templating system that provides a rich set of template tags that can be used directly with HTML.
- Views takes in an HTML Request and generates an HTML Response. This response can be a simple rendering of a static HTML template to complex database querying and rendering.

Django also provides a library Django-Rest framework which can be used to create Restful APIs. Data in the form of JSON Request can be sent as query to these APIs and a JSON Response containing the result of the query can be obtained

### 4.1.3 Django Channels

Django Channels is a powerful extension for the Django web framework that allows the handling of real-time web applications. While Django is traditionally designed for request-

response cycle-based applications, Django Channels enables the development of applications with persistent connections and asynchronous behavior.

With Django Channels, developers can build applications that require real-time updates, such as chat applications, notifications systems, collaborative editing tools, and more. It introduces the concept of "channels" that represent individual connections and facilitate communication between clients and the server.

One of the key features of Django Channels is its support for WebSockets, a protocol that enables bidirectional communication between the client and the server. WebSockets allow for real-time data transfer and can significantly enhance the user experience by enabling instant updates without the need for constant page refreshing.

Django Channels integrates seamlessly with other Django components, such as models, views, and authentication, making it easy to incorporate real-time functionality into existing Django projects. It also provides a straightforward API for handling WebSockets, allowing developers to define consumers to process incoming messages and send responses to clients.

Under the hood, Django Channels leverages asynchronous frameworks like `asyncio` and provides a scalable architecture for handling large numbers of concurrent connections efficiently. It supports various backends for channel layer communication, including in-memory, database, and message brokers like Redis or RabbitMQ[4].

By utilizing Django Channels, developers can extend the capabilities of their Django applications and create interactive, real-time experiences for their users. Whether it's building a chat room, live notifications, or any other application requiring instant updates, Django Channels provides the necessary tools and infrastructure to handle the complexities of real-time web development within the Django ecosystem.

#### 4.1.4 Svelte

Svelte is a modern JavaScript framework for building user interfaces (UIs) for web applications. It takes a unique approach compared to traditional frameworks like React or Angular. Rather than running in the browser, Svelte compiles components during build time into highly efficient JavaScript code.

One of the standout features of Svelte is its reactive programming model. It allows developers to write expressive and reactive code that automatically updates the UI whenever the underlying data changes. This reactive nature eliminates the need for complex state management libraries or explicit data bindings, resulting in simpler and more concise code.

Svelte also prioritizes performance and bundle size. By compiling components into optimized JavaScript code, Svelte generates smaller bundle sizes, leading to faster loading times and improved performance for the end-users. This lightweight approach makes it an excellent choice for creating performant applications, particularly in situations where bandwidth or device capabilities are limited[5].

The framework's simplicity and ease of use are also worth mentioning. Svelte's syntax is straightforward and intuitive, making it accessible for both beginners and experienced developers. Its component-based architecture allows for easy reuse and composition, promoting code reusability and maintainability.

Furthermore, Svelte has a growing ecosystem of third-party libraries and tools, enabling developers to extend its functionality and integrate with existing ecosystems seamlessly.

### 4.1.5 SvelteKit

SvelteKit is an extension of the Svelte JavaScript framework specifically designed for building complete web applications. It offers features like file-based routing, server-side rendering (SSR), and client-side rendering (CSR). SvelteKit simplifies application development by providing a smooth transition from Svelte and a file-based routing system. It allows developers to choose between SSR and CSR based on their needs. With SvelteKit, developers can create high-performing web applications with efficient code generation, improved SEO, and a rich plugin ecosystem for additional functionality[6].

### 4.1.6 Websockets

WebSockets is a communication protocol that enables real-time, bidirectional communication between a web browser and a server. Unlike traditional HTTP requests that follow a request-response model, WebSockets establish a persistent connection that allows for continuous data exchange between the client and the server.

One of the key advantages of WebSockets is its ability to facilitate instant and efficient communication. Once a WebSocket connection is established, data can be transmitted in both directions without the need for constant HTTP requests. This enables real-time updates, interactive messaging, and collaborative features in web applications.

WebSockets use a single TCP connection, reducing the overhead associated with establishing multiple connections for each interaction. This results in lower latency and improved efficiency compared to traditional HTTP-based approaches.

WebSockets support a wide range of applications, including chat systems, real-time notifications, multiplayer games, financial trading platforms, collaborative document editing,

and more. By enabling seamless, low-latency communication, WebSockets greatly enhance the user experience and enable more interactive and dynamic web applications.

In terms of implementation, WebSockets are supported by most modern web browsers and can be utilized with various programming languages and frameworks on the server side. The WebSocket API provides methods for opening, closing, sending, and receiving data over the WebSocket connection, making it easy for developers to incorporate real-time functionality into their applications.

However, it's important to note that WebSockets require server-side support as well. Servers need to implement the WebSocket protocol and handle the incoming WebSocket connections, along with managing the data exchange and broadcasting updates to connected clients.

## **4.2 Modules**

In this section we'll see how different modules are implemented in Django. The working and APIs are explained.

### **4.2.1 Centralized server**

The server is implemented using the Django Web Framework and for database, SQLite is used. In Django, modular components called Web Apps can be built. The server has the following components.

- Authentication Module
- MSGS module

- Dashboard module
- Helper Module

### **Authentication module**

This module is responsible for providing APIs related to authentication of credentials of those who access the app. Using this API, we create session corresponding to a user after verification of the credentials and after logging out, the session is deleted.

### **MSGs module**

This module is responsible for all SMS related tasks. Receiving SMS on server, parsing it to extract meaning, creating a task corresponding to the received SMS. If the received SMS is in correct format and for the given case, slot in an HP is available, a task is created, else, the task is marked as a flagged task. That means, manual intervention is required for managing these kind of tasks.

This is also used to send SMS. We just do a GET request to a url provided by the service provider and, then, the service provider checks if it is in the correct format. If the format is correct, SMS is sent to the target customer else it is dropped.

### **Dashboard module**

This module has most of the important APIs for accessing relevant data. This has the APIs for adding, editing or viewing Corporation, Health facilities, SNEHA master, Agent, Case Type, and Doctor slots. This API has appropriate validators which helps to get only those inputs that are in correct format.



This also has the follow-up task scheduler. We have used django-crontab for serving the purpose. At the end of every day, it checks all the tasks of the day and sends an SMS to all the agents who created those tasks. This SMS is for a follow-up of the appointment. The agent again replies to this and the decision, whether to mark the task as completed or to reschedule the appointment, depends on the reply SMS.

While designing an API for adding available doctor slots to database, one requirement was to add recurring slots. Which means an officer can add available slot that will repeat weekly or monthly. This could have been done, by creating that many individual and independent slots for each week. But in case if all these slots needs to be cancelled or deleted later, then the challenge of identifying which all slots were created together, arises. To tackle this, we designed a parent doctor slot. This parent object stores date and time of the slot, the frequency of repetition (weekly, monthly or never), and a few other details. Now, for every doctor slot addition, we first create a parent doctor slot and then, for every parent slot we create an entry in doctor slots with this parent slot as a foreign key. Every doctor slot has only its own date and time.

## **Helper Module**

This module manages the access to the android app. This can also be merged with the dashboard module with the SNEHA master table. But for convenience of changing and accessing the app, it is kept separate. It is responsible for creating different credentials for android app.

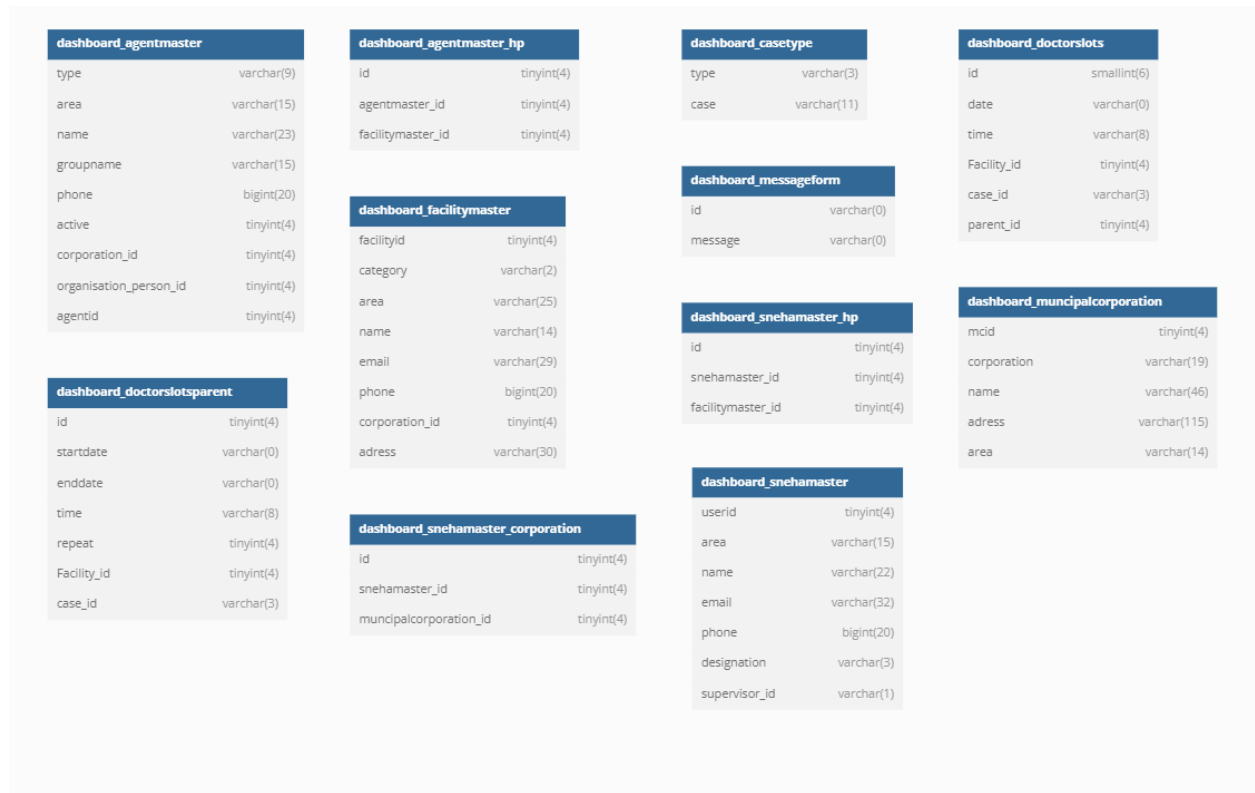


Figure 4.1: Dashboard Schema



Figure 4.2: Auth Schema

msgs_flagtask		msgs_ivr_message		msgs_task	
id	tinyint(4)	id	varchar(0)	id	tinyint(4)
agentid	tinyint(4)	cid	varchar(0)	case	varchar(13)
message	varchar(40)	message	varchar(0)	status	tinyint(4)
problem	varchar(33)			agentid	tinyint(4)
created	varchar(10)			name	varchar(16)
				phone	bigint(20)
				hospital	varchar(14)
				created	varchar(10)
				modified	varchar(10)
				slot_id	smallint(6)

Figure 4.3: MSGs Schema

Helpline		Facilities			
WELCOME WWH		Name	Category	Corporation	Area
Home		2	Greater Mumbai	Municipal Corporation of Greater Mumbai (MCGM)	6RPR+68H, Yogi Nagar, Borivali West, Mumbai 40
Corporation		3	Thane	Thane Municipal Corporation (TMC)	5XX8+2V9, New Administrative Building, Chandan Wadi, Pachpakhadi, Mahapalika I
Facility		4	Ulhasnagar	Ulhasnagar Municipal Corporation (UMC)	Chopra Rd, Vitthalwadi, Ulhasnagar, Maharashtra 4
SNEHA Master		5	Kalyan-Dombivali	Kalyan Dombivali Municipal Corporation (KDMC)	64RH+7FH, Bhoiwada, Kalyan, Maharashtra 421:
Agent		6	Bhiwandi- Nizampur	Bhiwandi Nizampur Municipal Corporation (BNMC)	73X7+HPM, Juna ST Road, Old SP Stand, New Building, opp. Rajiv Gandhi Flyov
Case type		7	Mira-Bhayandar	Mira Bhayandar Municipal Corporation (MBMC)	8R2X+P4R, Indira Gandhi Bhawan Chhatrapati Shivaji Maharaj Marg, Bhayandar Wes
Hospital slots					
Flagged Tasks					
All Tasks					
Clients					
Logout					

Figure 4.4: Main page template.

## Input Facility

Name	Category
<input type="text"/>	<input type="text" value="-----"/>
Email	Corporation
<input type="text"/>	<input type="text" value="-----"/>
Area	Adress
<input type="text"/>	<input type="text"/>
Phone	
<input type="text"/>	

Add Facility

Figure 4.5: Input facility page.

Facilities							
Name	Category	Corpotation	Area	Address	Email	Phone	
Waliv UPHC	HP	Vasai Virar Municipal Corporation (VVMC)	Waliv Nagri Arogya Kendra	Waliv Nagri Arogya Kendra VVMC	uhpwaliv@gmail.com	7045260121	Edit
Wagle HP	HP	Thane Municipal Corporation (TMC)	Near TMC Bus Depo	Wagle estate, Thane West	nadeem.shalkh@snehamumbai.org	7045260121	Edit
SNEHA Facility	HP	Municipal Corporation of Greater Mumbai (MCGM)	Santacruz	BMC Colony, Santacruz West	nadeem.shalkh@snehamumbai.org	7045260121	Edit
Dummy Faci	HP	Thane Municipal Corporation (TMC)	test facility area	test address	dummy@mail.com	9876598763	Edit

Figure 4.6: View all facilities. In each row there is an edit button using which one can edit the facility details

## Input Case type

Type	Case
<input type="text"/>	<input type="text"/>

Add Case type

Figure 4.7: Input case type

Case Type

Case	Type	
ANC Case	ANC	Edit
Danger Sign	DS	Edit
test-case	TS	Edit

Figure 4.8: View all case types. Every row has an edit button which can be used to edit the details.

Input Sneha

Name

Email

Area

Hp

☐ Waliv UPHC
 ☐ Wagle HP
 ☐ SNEHA Facility
 ☐ Dummy Faci

Supervisor

Designation

Phone

Corporation

☐ Municipal Corporation of Greater Mumbai (MCGM)
 ☐ Thane Municipal Corporation (TMC)
 ☐ Ulhasnagar Municipal Corporation (UMC)
 ☐ Kalyan Dombivali Municipal Corporation (KDMC)
 ☐ Bhiwandi Nizampur Municipal Corporation (BNMC)
 ☐ Mira Bhayandar Municipal Corporation (MBMC)
 ☐ Vasai Virar Municipal Corporation (VVMC)

Add Sneha

Figure 4.9: Input SNEHA master.

SNEHAs						
Sneha ID	Name	Corporation	Email	Phone	HP	Area
1	test Sneha master Name	Municipal Corporation of Greater Mumbai (MCGM) Thane Municipal Corporation (TMC)	test_sneha_master@gmail.com	9876598764	Waliv UPHC Wagle HP SNEHA Facility	test area
2	Nadeem Shaikh	Municipal Corporation of Greater Mumbai (MCGM) Thane Municipal Corporation (TMC) Ulhasnagar Municipal Corporation (UMC) Kalyan Dombivli Municipal Corporation (KDMC) Bhiwandi Nizampur Municipal Corporation (BNMC) Mira Bhayandar Municipal Corporation (MBMC) Vasai Virar Municipal Corporation (VVMC)	nadeem.shaikh@snehamumbai.org	7045260121	Waliv UPHC Wagle HP	Central
3	Sarita Patel	Municipal Corporation of Greater Mumbai (MCGM) Thane Municipal Corporation (TMC) Ulhasnagar Municipal Corporation (UMC) Kalyan Dombivli Municipal Corporation (KDMC) Bhiwandi Nizampur Municipal Corporation (BNMC) Mira Bhayandar Municipal Corporation (MBMC) Vasai Virar Municipal Corporation (VVMC)	sarita@snehamumbai.org	9930456313	Waliv UPHC Wagle HP	Thane Office
4	Kanchan Jagdhane	Municipal Corporation of Greater Mumbai (MCGM) Thane Municipal Corporation (TMC) Ulhasnagar Municipal Corporation (UMC) Kalyan Dombivli Municipal Corporation (KDMC) Bhiwandi Nizampur Municipal Corporation (BNMC) Mira Bhayandar Municipal Corporation (MBMC) Vasai Virar Municipal Corporation (VVMC)	kanchan.jagdhane@snehamumbai.org	9930658028	Waliv UPHC Wagle HP	Thane Office

Figure 4.10: View all SNEHA masters. Every row has an edit button which can be used to edit the details.

## Input Agent

Corporation

-----

Type

-----

Organisation person

-----

Phone

Name

Groupname

Area

Hp

☐ Waliv UPHC  
☐ Wagle HP  
☐ SNEHA Facility  
☐ Dummy Faci

Add Agent

Figure 4.11: Input Agents

Agents							
Agent ID	Name	corporation	Type	Area	HP	Status	
2	test Name	Municipal Corporation of Greater Mumbai (MCGM)	ASHA	Near thane	Wagle HP	Deactivate	Edit
3	test Name2	Thane Municipal Corporation (TMC)	ASHA	Near thane	Wagle HP	Deactivate	Edit
4	test Name3	Thane Municipal Corporation (TMC)	ASHA	Near thane	Waliv UPHC	Deactivate	Edit
5	Nadeem	Vasai Virar Municipal Corporation (VVMC)	ASHA	Thane Office	SNEHA Facility	Deactivate	Edit

Figure 4.12: View all Agents. Every row has an edit button which can be used to edit the details.

## Input New slot

Case	Facility
<input type="text" value="-----"/>	<input type="text" value="-----"/>
Startdate	Enddate
<input type="text" value="mm/dd/yyyy"/>	<input type="text" value="mm/dd/yyyy"/>
Time	Repeat
<input type="text" value="--:-- --"/>	<input type="text" value="NEVER"/>
<input type="button" value="Add New slot"/>	

Figure 4.13: Input slots







All slots					
Facility	Case	Date	Time		
Waliv UPHC	ANC Case	Oct. 18, 2022	9 a.m.	797	
Wagle HP	ANC Case	Oct. 18, 2022	1 p.m.	829	
Waliv UPHC	ANC Case	Oct. 25, 2022	9 a.m.	798	
Wagle HP	ANC Case	Oct. 25, 2022	1 p.m.	830	
Waliv UPHC	ANC Case	Nov. 1, 2022	9 a.m.	799	
Wagle HP	ANC Case	Nov. 1, 2022	1 p.m.	831	

Figure 4.14: View all slots. Every row has an edit button which can be used to edit the details.







All cases					
Agent ID 	Case 	Beneficiary Name 	Beneficiary no 	Hospital 	Created 
1	test case	Chinmay	9881246601	test Name	Aug. 9, 2022, 3:55 p.m.
1	Danger Sign	Chinmay	9881246601	test Name	Aug. 9, 2022, 4:02 p.m.
2	Danger Sign	Chinmay	9881246601	Wagle HP	Aug. 19, 2022, 1:28 a.m.
3	Danger Sign	Chinmay	9881246601	Wagle HP	Aug. 19, 2022, 1:36 a.m.
3	Danger Sign	Chinmay	9881246601	Wagle HP	Aug. 19, 2022, 1:39 a.m.
3	test-case	Chinmay	9881246601	Wagle HP	Aug. 19, 2022, 1:40 a.m.
4	Danger Sign	Chinmay	9881246601	Waliv UPHC	Aug. 19, 2022, 2:16 a.m.

Figure 4.15: View all valid tasks



Unhandled cases			
Agent ID	Message	Problem	Recieved on
1	1 TC Chinmay 9881246601	Invalid case	Aug. 9, 2022, 3:52 p.m.
1	1 TC Chinmay 9881246601	Slots for test case not found	Aug. 9, 2022, 3:53 p.m.
1	1 TC Chinmay 9881246601	Invalid case	Aug. 9, 2022, 3:56 p.m.
1	1 AN Chinmay 9881246601	Invalid case	Aug. 9, 2022, 3:59 p.m.
1	1 DS Chinmay 9881246601	Slots for Danger Sign not found	Aug. 9, 2022, 4:01 p.m.
2	2 AN Chinmay 9881246601	Invalid case	Aug. 19, 2022, 1:24 a.m.

Figure 4.16: View all flagged tasks. These are the tasks that are marked invalid by the system

## 4.2.2 Android

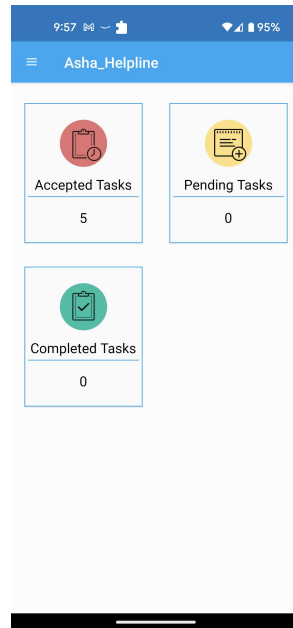


Figure 4.17: Dashboard: Here the user can see all the categories of the tasks.

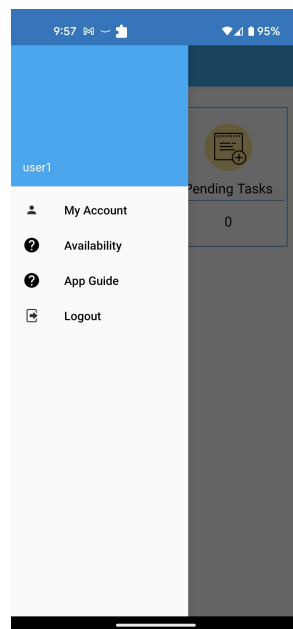


Figure 4.18: Sidebar: Menu with options to explore.

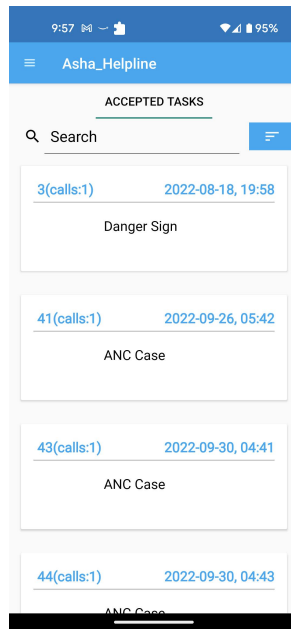


Figure 4.19: Accepted tasks: Here the user can see the list of accepted tasks.

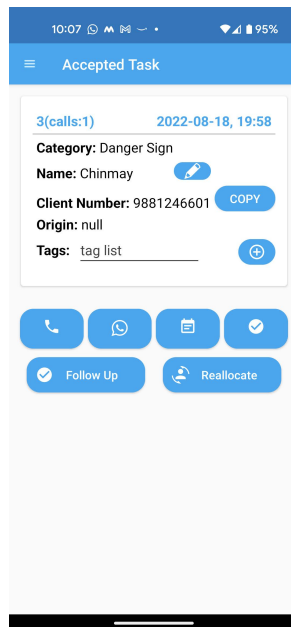


Figure 4.20: Accepted: Here the user can see the details of an accepted task after clicking on it.

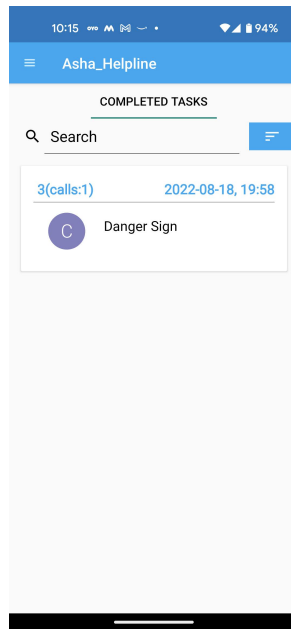


Figure 4.21: Completed tasks: Here the user can see the list of completed tasks.

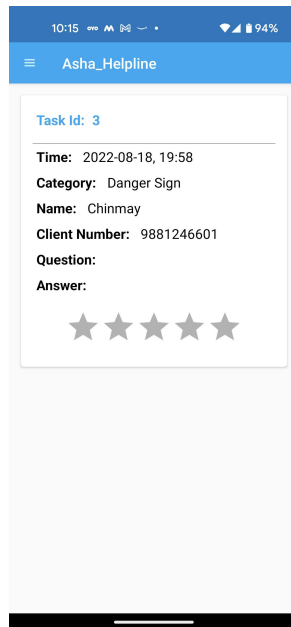


Figure 4.22: Completed task details: Here the user can see the details of a completed task after he clicks on it.

### 4.2.3 Chat Service

The chat feature plays a crucial role in facilitating communication between helpers and clients within the system. It provides a means for direct and real-time interaction, allowing

clients to seek assistance, ask questions, and receive guidance from helpers.

By integrating a chat service into the system, clients can easily initiate conversations with helpers, discussing their needs, concerns, or any issues they may be facing. This direct line of communication enables a personalized and interactive support experience.

For clients, the chat feature offers convenience and accessibility. They can access the chat service from within the system, eliminating the need to switch to external communication platforms or methods. This centralized communication channel streamlines the support process and ensures that all relevant information is captured within the system itself.

Helpers benefit from the chat feature as well. They can provide timely responses, address client queries, offer guidance, and share relevant resources or information. The real-time nature of the chat service allows helpers to provide immediate assistance, ensuring a smooth and efficient support experience for clients.

Moreover, the chat feature enables the exchange of multimedia content, such as screenshots, images, or files, which can be valuable in troubleshooting or providing visual aids during the support process. Helpers can share resources, links, or step-by-step instructions directly through the chat, enhancing the clarity and effectiveness of the assistance provided.

## **Implementation details of Chat service**

We have used WebSocket protocol instead of http protocol. The reason being Websocket is a full duplex and persistent protocol while HTTP is not.

- Website
  - Frontend: In the frontend, we use SvelteKit as our primary tool for implementing the chat functionality.

1. When a user chooses to engage in a chat, they are directed to a login page where they enter their credentials. Once the backend verifies these credentials, a GET request is sent to retrieve the list of tasks associated with the provided phone number.
  2. Upon selecting a specific task, two actions take place. Firstly, a GET request is sent to the backend to fetch the messages related to that task. Secondly, a WebSocket connection request is made to establish a real-time communication channel with the backend. The user interface adapts to the type of message being displayed, which can be one of four types: sent text message, received text message, sent file, or received file.
  3. Once the WebSocket connection is successfully established, the user can effortlessly send and receive messages through this connection, enabling seamless communication in real-time.
- Backend: For this, we have used Django for the backend. And Django Channels framework for the backend required for WebSockets. Below are the listed modules that have been designed.
1. ChatBackend: This contains all the Django-related settings. ASGI( Asynchronous Server Gateway Interface) configuration is specified here.
  2. ChatAPI: All the API calls are defined in this module. APIs like fetching all tasks, task details, messages, etc. This is mainly for handling all the HTTP requests.
  3. ChatWebsockets: This module has all the WebSocket protocol-related code. This has several parts like the routing of websockets. Routing is written in a file routing.py. It is similar to urls.py in standard Django but is for WebSocket requests. The file named consumers.py plays a crucial role in

managing WebSocket connections. It is responsible for managing the incoming requests. It creates consumers or peers and propagates messages to other peers connected to the WebSocket. A few important functions that the file has are, connect, disconnect, reject, receive, and send. Connect, Disconnect, and Reject are used during connection establishment, while Receive and Send are used for transferring messages.

4. Static: This has all the CSS and JS files.
5. Templates: All the HTML files are stored in this.

- Android:

- On the Android side Java is used.
- The helper can see all the tasks on his screen. On clicking one of the tasks he can see the option to chat.
- After clicking on chat, a GET request is made to the server with the task id as the parameter.
- The server replies with the task chat history.
- The task chat history is then rendered in the UI. Similar to the Web, here also we have 4 types of designs for 4 types of different messages.
- There is a textbox where the user can add the message that is to be sent. Once send button is clicked, the WebSocket connection process is also initiated.
- Once the WebSocket is connected, the user can send and receive messages and files in real-time.

# Chapter 5

## Future Work

- **Monthly report:** Adding a feature where every agent will get a monthly report of all the tasks. Also all the SNEHA members will get a report of all the tasks that got completed or not with the task details.
- **Hindi support:** Since many users are not used to English, Hindi or Devanagiri is required for SMS. But for this to function, first the template has to be registered with the SMS service provider. Once done, this can benefit many such patients who otherwise won't understand English text.
- **Field Trials:** The project is in process for field trials. SNEHA foundation is already testing it in Mumbai in seven corporations. The demos and training sessions have already been conducted by us with them.
- **Read Receipts:** Read receipts can be added to the sent and received messages in the chat application.
- **Integration in different apps:** The chat service can be integrated in any other application with minimal configuration changes.



# Bibliography

- [1] “<https://economictimes.indiatimes.com/industry/cons-products/electronics/india-to-have-1-billion-smartphone-users-by-2026-deloitte/articleshow/89750324.cms>,”
- [2] “<https://snehamumbai.org/>,”
- [3] “<https://www.djangoproject.com/>,”
- [4] “<https://channels.readthedocs.io/en/stable/>,”
- [5] “<https://svelte.dev/>,”
- [6] “<https://kit.svelte.dev/docs/introduction>,”