iJRASET
International Journal For Research in Applied Science and Engineering Technology



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 12 **Issue:** IV **Month of publication:** April 2024

DOI: https://doi.org/10.22214/ijraset.2024.60110

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 12 Issue IV Apr 2024- Available at www.ijraset.com

A Fullstack Web Application to Provide On-demand Household Services

Sammed C Jain[1], Shreyas N Athreya[2], Vishak Koundinya N[3], Yashwanth S Gowda[4]
Department of Information Science & Engineering, The National Institute of Engineering, Mysore, Karnataka, India

Abstract: ***A Fullstack Web Application to Provide On-demand Household Services is a paper focused on designing and creating a full stack web application. The application aims to help users find nearby household service providers, such as electricians, plumbers, gardeners, carpenters etc. Users can easily browse the available services, book the appointments, track the location of service professionals, and share feedback about completed work. For service professionals, the application offers tools to manage appointments and track earnings. The application incorporates advanced technologies like Blockchain for secure storage of customer ratings, reviews, and work history generated by service professionals. Additionally, Machine Learning is utilized to develop chatbots for real-time user assistance. Few technological domains which are covered in this application are Full Stack development, Blockchain technology and Machine Learning. In summary, the goal of this application is to improve the skilled workforce connection system, making it efficient, secure, and engaging for all users.***
Keywords: Household services, Fullstack web application, Real time location tracking, Service bookings, Chatbot.

I. INTRODUCTION

This paper explores the creation of a user-friendly full stack web application aimed at connecting customers with household service providers seamlessly. By utilizing modern technologies such as Full Stack development, Blockchain, and Machine Learning, the platform aims to redefine how household services are accessed and managed. One key focus of this project is to simplify service selection for customers. Unlike traditional methods requiring extensive research and phone calls, our online system provides an easy-to-use interface for browsing and booking services in a specific area, saving time and effort and ultimately reducing costs. Manual appointment scheduling and management processes often involve time-consuming tasks such as phone calls, paperwork, and coordination efforts. The online system simplifies and automates these processes, allowing service professionals to efficiently manage their appointments and maximize their earning potential. This automation also reduces administrative overhead, offering a cost-effective solution for workers. Furthermore, the digital platform opens up new job opportunities for service providers. By reaching a wider online customer base without expensive advertising, it offers a cost-effective way for workers to find employment. Moreover, the integration of Machine Learning-powered chatbots enhances user support without the need for extensive human resources. This cost-effective solution improves user experience and boosts operational efficiency.

In summary, this research paper highlights the potential of a web application to revolutionize the household service industry by making it more accessible, efficient, and cost-effective for both customers and service providers.

II. OBJECTIVES

1) To develop a web application that connects skilled working professionals with customers, offering a straightforward registration process, location based service browsing and one click appointment booking.
2) To create real time location tracking and navigation system that enables customers and service professionals to locate each other and access maps and directions for service appointments.
3) Implement blockchain technology to securely store work records or completed booking details, reviews, ratings and other sensitive information, enhancing trust and transparency of the platform.
4) Utilize machine learning technology to develop intelligent chatbots equipped with natural language processing to handle customer complaints effectively, while also providing relevant information and support on other inquiries.

III.PROPOSED SYSTEM

In the proposed system, there are three key actors: customers who are looking to book a service, service professionals who are seeking appointments from customers, and admins who control the system. The functionalities provided to each of these three actors are discussed in this section of proposed system.

©IJRASET: All Rights are Reserved

A. Customers

In the proposed system customers can login with their credentials and can browse through the available services in their locality. They can book the service appointment after selecting their desired category of service. They can also track the worker's real time location on maps after booking an appointment. They can view their booking history and are also allowed to leave feedback for the completed work. Ratings and reviews given by customers are securely stored in the blockchain, making the reviews tamper-proof. Customers can also interact with the chatbot to raise the complaints related to the active service bookings and can also get other relevant information.

B. Service Professionals

Service professionals or workers can activate or deactivate their profile based on their availability and working time. After receiving the appointment, they can view the location of the customer and can decide whether to accept the service or not. Work records or completed booking data will be securely stored in the blockchain, making it tamper proof. Workers can also view their work history along with the ratings and reviews received from their customers. Workers can also track the earnings made from the completed work.

C. Admins

Admins in the system have the authority to register new workers. They can also view the feedback and complaints given by the customers and can remove the workers from the system if necessary.

Overall, the proposed system empowers customers with efficient service selection, seamless appointment booking, location tracking, genuine feedbacks and interactive assistance. Service professionals benefit from efficient appointment handling, customer location visibility, and expense tracking. This comprehensive system aims to enhance the overall experience for customers and facilitate service professionals in efficiently discovering and managing new service appointments.

IV. LITERATURE SURVEY

A. A Web Application Based Administration Panel For Handyman Services

This research paper is published by G. Baskaran, K. Saundariya, D. Prabakaran and R.Senthilkumaran in the 2022 IEEE Delhi Section Conference (DELCON), NewDelhi, India, 2022. The paper introduces a web application that enables users to book handyman services for various household tasks. The web application uses MongoDB, Express, Node.js, React, Redux, Axios, JWT, and Node mailer as the main technologies. The paper also describes an administration panel that allows the admin to manage the information and functionality of the web application. The paper explores the features and advantages of the web application and administration panel. These include a user-friendly interface, location-based searching and booking, pre-defined service costs, verification of service providers, a notification and feedback system, a chatbot for user queries, tracking and cancellation of service requests, and secure authentication and authorization. Additionally, the admin benefits from data visualization and analysis tools. The paper concludes that the proposed web application can provide a reliable admin panel that can manage the content and functionality of the website, as well as improve the performance and user interaction of the web page.

B. An Android Application for Home Services

This research paper is published by Hegde Sharaj Bhaskar Shyamala, Krishnamoorthy Rao, Padmanabha Bhandarkar, Prateek Prakash Vetekar, Geetha Laxmi in International Journal of Engineering Research and Technology (IJERT). The paper discusses how on-demand apps have disrupted many traditional industries and created a demand for home services apps that connect customers to service professionals for various household chores and errands. The paper also states the objectives and scope of the project, which is to develop an Android application for home services that covers all the cities in India and offers a contract-based model for flats and apartments. The paper proposes a system that consists of two actors: a customer and a service provider. The paper describes the functionalities and features of the system, such as registration, login, service selection, payment, rating, review, return policy, and location tracking. The paper also explains the roles and responsibilities of the administrator, who has the access and authority to manage the website and the database. It presents the design and development of the system, using various diagrams and tools like architectural diagram, the use case diagram, the data flow diagram, sequence diagram, and the control flow diagram to illustrate the structure, behavior, and interaction of the system. The paper also mentions the tools and technologies used for the implementation, such as Android Studio IDE, Java, XML, Firebase Realtime Database, Firebase Authentication, and the Google Maps API.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

It concludes that the developed system is a feasible and efficient solution for the home services market, as it provides a platform for customers to find and hire reliable and skilled service providers for various tasks. The paper also suggests some future enhancements, such as adding more functionalities, integrating with other payment gateways, and providing chat and call options.

C. Household Veritas - A platform that provides household services

This research paper is published by Apeksha Adekar, Aakash Dalvi, Pratik Gharat, Pushti Ratanghayra in International Journal of Engineering Research and Technology (IJERT). The paper introduces the need and motivation for developing an Android app for household services, such as plumbing, electrical, electronic, mechanical, pest control, home paint, and machine repairing. The paper states that the app aims to provide convenience and ease to customers and service providers, especially in urban areas where finding reliable and efficient service providers can be a challenge. The paper describes the steps involved in creating the Android app, such as understanding the requirements, conducting research, designing the solution, building the platform, testing and validating, deploying, and evaluating and improving. The paper also explains the tools and technologies used in the app development process, such as Android Studio, Java. The paper presents the architectural design, modular design, use cases diagram, and sequence diagram of the app, which illustrate the structure, components, interactions, and order of operations of the system. It also discusses the implementation details of the app, by including the source code for designing the database, the server, and the user interface. The paper also mentions the challenges and limitations faced during the app development, such as compatibility issues, performance issues, and security issues. The paper concludes that the Android app for household services has the potential to significantly impact both customers and service providers, by providing a user-friendly, secure, and efficient platform for accessing and delivering household services.

V. SYSTEM REQUIREMENTS

The software requirements for the proposed system encompass a comprehensive stack of technologies and libraries to facilitate efficient development and robust functionality. At the backend, the runtime environment is based on Node.js, coupled with the Express.js framework for seamless server-side operations. Authorization is ensured through the integration of JWT (JSON Web Tokens) for secure access control.

On the frontend, React.js serves as the primary framework, augmented by libraries such as react-chatbot-kit for interactive chatbot features, react-chartjs-2 for dynamic data visualization, and Leaflet for mapping functionalities. Navigation and directions are facilitated by the OSRM (Open Source Routing Machine) library, ensuring accurate route planning. Additionally, Bootstrap and CSS are utilized for frontend designs, alongside other styling libraries to refine the visual presentation and enhance user interface consistency.

In the context of blockchain integration, the system employs Motoko as the programming language for ICP blockchain development, with the DFINITY Canister SDK facilitating seamless interaction with the blockchain network. Furthermore, machine learning capabilities are enabled through a library called react-chatbot-kit. For sending the response in natural language, few other ML tools and frameworks are used. This comprehensive tech stack, supplemented by essential libraries and packages, ensures the development of a scalable, feature-rich system with enhanced functionality and user experience.

For efficient ML development, a quad-core processor or higher is recommended, alongside at least 8 GB of RAM to handle large datasets. A dedicated GPU with a minimum of 4 GB of VRAM accelerates ML computations, particularly for larger data sets. These specifications ensure faster model training and inference, crucial for delivering intelligent functionalities within the proposed system.

VI. SYSTEM DESIGN

A. Use Case Diagram

The use case diagram shows the interactions between the three key actors: customer, service professionals, and an admin for an online on-demand household services system. The customer can browse services, book appointments, track services, give ratings and feedback, and interact with the chatbot. The service professional can confirm appointments, view customer's location, view feedback, and request the admin to register. The admin can register the service professional, remove the service professional, view feedback, and view system statistics.

The include relationship indicates that a use case is mandatory. For example, the "Book appointments" use case includes the "Confirm appointment" use case. This means that once the customer books an appointment, the service professional must confirm or reject the appointment. The extend relationship indicates that a use case is optional. Viewing the ratings and feedback, viewing the customer’s location remains optional for service professionals.

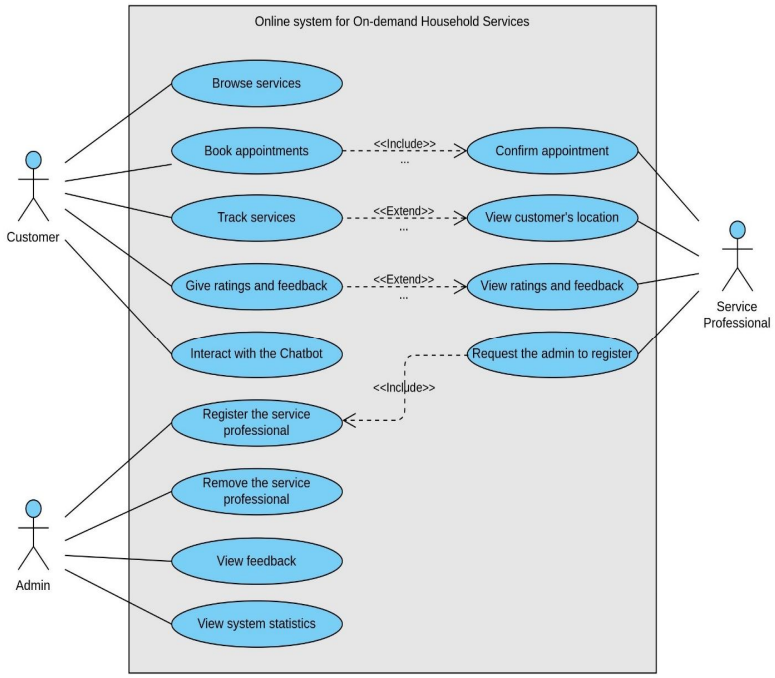


Fig. 1 Use Case Diagram

B. Designing the register logic

The Customer or Worker or Admin would fill out the username, password, and confirm password fields. On the client side, the password and confirm password fields are checked to see if they match; if not, an alert message will be displayed. The username and desired password will be submitted to the server side through a form post request. Passwords are hashed using the Bcrypt package and stored in the database. A JSON Web Token (JWT) is generated using the username and a secret key, and an expiry time is set for the token. The predefined redirect URL and the generated token are passed to the client side. The token is stored in the local storage on the client side, and the user is redirected to the main page.

C. Designing the login logic

Similar to the registration logic, here the customer fills out the username and password fields. These credentials are submitted to the server side through a post request from the client side. The password entered by the user is hashed, and the credentials are checked against the database of existing users. If there is no match, an alert message is sent to the client side; otherwise, if there is a match, a JWT token is generated using the credentials, and a desired expiry time is set for the token. The Redirect URL and the token are passed to the client side. The JWT is stored in the local storage of the client for further authentication purposes, and the user is redirected to the main page.

D. Designing the POST request routes

The user triggers a post request by either hitting a submit button or filling out a form. The JWT token is retrieved from the local storage and is passed to the server side as an authentication header of the request along with the payload data. The server verifies the token, and if the token is not valid, an alert message is displayed. Otherwise, the required operation is performed on the data, and the response is sent back to the client side.

E. Designing the GET request routes

When the user types the URL or is redirected to a page from another page, a GET request is made to the server to retrieve the page to be displayed. Similar to the post request, a JWT is retrieved from the local storage and passed to the server side as an authentication header of the request. The server checks if the route requested by the user is secure; if it is not secured, the token is not verified. Otherwise, if the route is secured, the token is verified. The response generated from the server is then passed to the client side, and the desired website is displayed to the user.

F. Designing the Blockchain Logic

The reviews and ratings provided by the customer, along with work records, are stored in the ICP blockchain to ensure its integrity. A transaction is created using the data intended for blockchain storage and the canister ID, which is assigned during blockchain deployment. The transaction is signed with the user's private key and sent to the server side. The server verifies the canister ID, and if the verification fails, an alert message is displayed, and the transaction is discarded. If the ID is successfully verified, the function within the canister is called to store the data. A new block, containing the updated canister state, is added to the local ICP blockchain, and confirmation is sent to the client side.

G. Designing the Chatbot Logic

The user interacts with the chatbot interface to file a complaint or to get any relevant information. For the frontend representation of the chatbot an open source package called react-chatbot-kit is used which helps in parsing the user's messages and in calling the specific functions whenever required. The request or message sent by the user is parsed by the message parser component of the react-chatbot-kit and identifies which exact function has to be called. The chatbot has access to the data present in the database and makes requests to the database if required. The response from the chatbot server is then passed to the Natural Language Processing software, which converts the response into readable natural language, displayed as the reply from the chatbot.

VII. DATABASE DESIGN

A. Customer Model

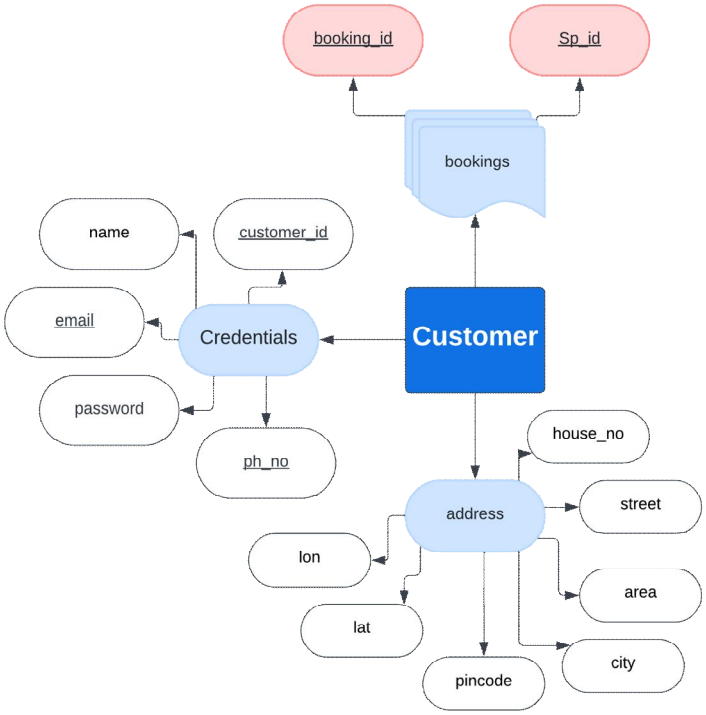


Fig.2 Customer's data base model

In the Customer model, credentials of the customers are stored inside a JSON nested object called "Credentials," which includes the customer's Name, Email, Phone number, Password, and Customer ID. These fields are predominantly used in the authentication routes like login and register routes. Among these fields, email, phone number, and customer ID are primary keys. The complete address of the customer is stored in a separate JSON nested object, which includes the house number, street, area, city, pincode, and the location's latitude and longitude data. These fields help in setting the customer's location and address. Booking data is stored in an array document known as bookings. Only booking_id and Sp_id {service professional's ID} are stored inside each element of this array. This is predominantly used to handle active bookings and in viewing the booking history. Additionally, both booking_id and Sp_id are identified as foreign keys, referencing the Service Professional model.

B. Service Professional Model

Credentials of the service professional are stored similarly to the previously explained Customer model. This includes basic details of the professional such as Name, Email, Phone number, Password, and ID. Among these fields, the service professional's ID (or just _id), email, and phone number are defined as primary keys. All booking data, including active and inactive services, is stored inside the service_history array document. Each entry of this array document includes several individual fields like active, accepted, rejected, cancelled by customer, date, allotted time, finished time, ratings, review, amount, OTP, customer ID, and booking ID. Among these individual fields, booking ID is defined as the primary key, and customer ID is defined as a foreign key which references the Customer's model. A few nested fields present inside this array are completed field which include user's and SP's boolean data, and complaints field which include all the complaint details, and customer_location field which includes the latitude and longitude data of the customer's location. The service history array plays an important role in accessing the booking history and also in managing the active services for each service professional or customer.

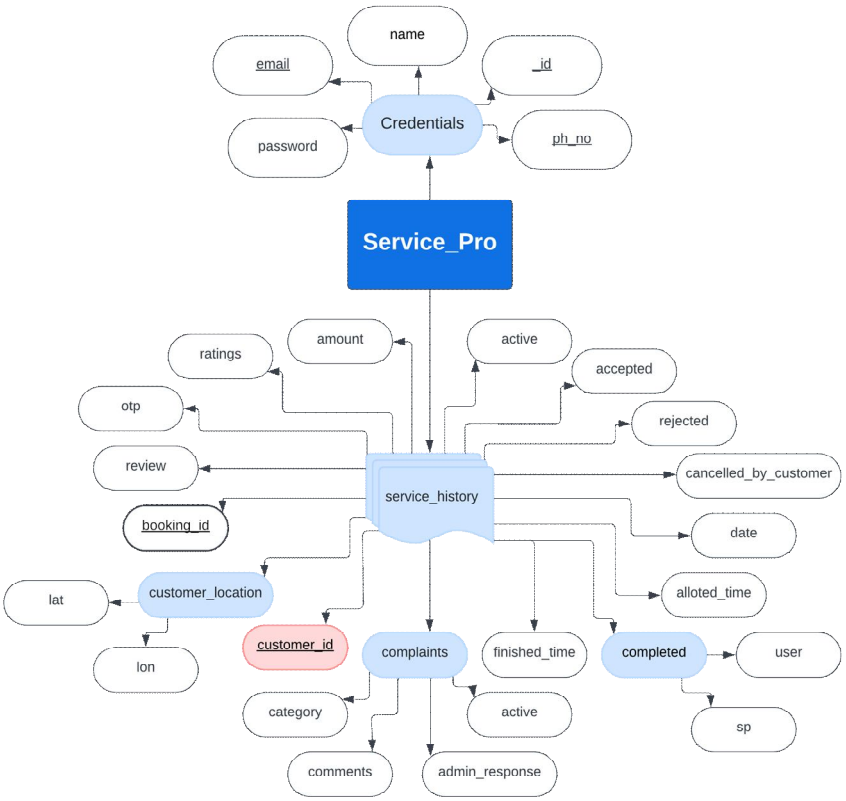


Fig.3 Service professional's data base model

C. Admin's Model

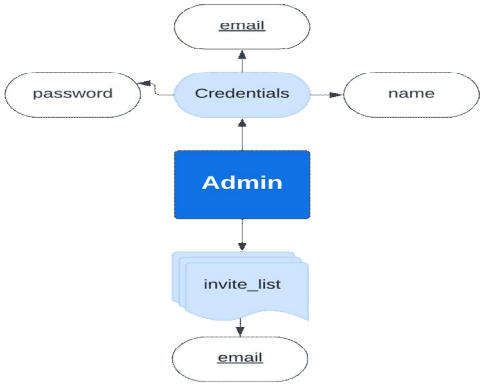


Fig.4 Admin's data base model

Admin's database model is the simplest of all. Admin's details are present within a nested document named "credentials," akin to the other two models. It includes the admin's Name, Email, and Password, among which email serves as the primary key. Additionally, the Admin's model comprises an array called "invite_list," which stores the email of invitees, each of which should be unique within the entire array.

VIII. IMPLEMENTATION AND RESULTS

A. Authentication Module

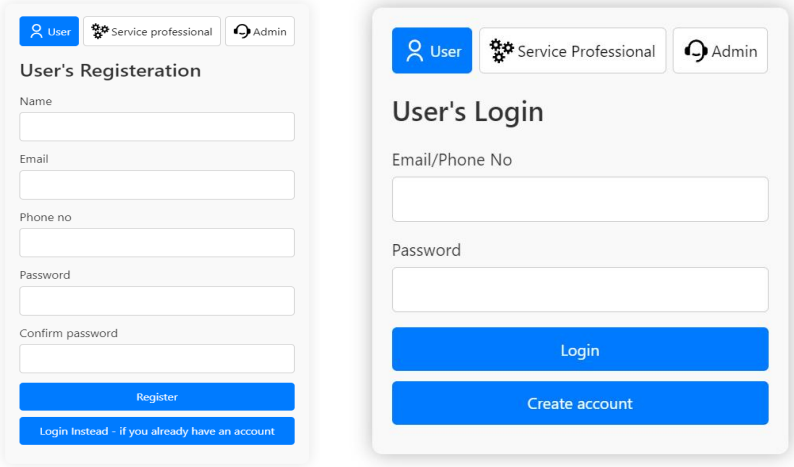


Fig.5 Registration and Login forms

To ensure secure access to the platform, an authentication system has been implemented. Users, whether they are customers, service professionals, or admins, are required to authenticate themselves before accessing their respective dashboards. Users can either log in with their existing credentials or register with a new account. During the login process, users are required to provide their email or phone number along with their password. For registration, additional information such as name, both email and phone number, and the desired password is required for customers and administrators. Service professionals are additionally prompted to specify their default working location, service category, and the city where they primarily serve. When the user hits the login or register button, a function gets triggered in the React component, and all the above-mentioned credentials are sent as payload to the backend via an Axios post request. In the Node.js {backend} post request code, if the user is registering, then his entered password is hashed and stored in the database. Otherwise, if he is logging in, the entered password is hashed and checked with the existing entry in the database. If a match is found, then a JWT token {bearer token} is generated, which contains the user's email, role, issued time, and expiry time. This token will be sent as a response to the frontend. It will be stored in the browser's local storage for further authentication.

B. Customer's Module

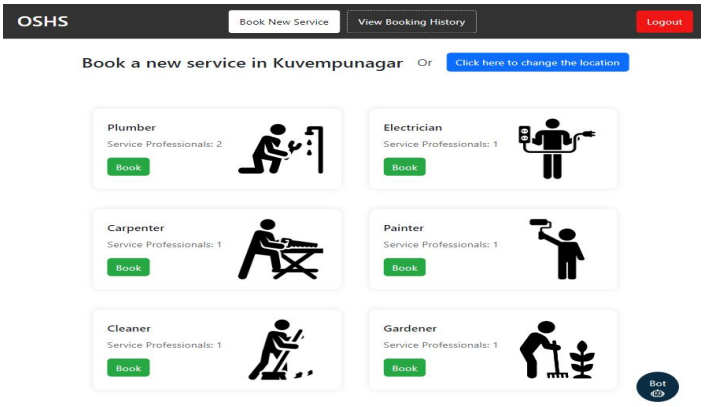


Fig.6 Customer's Dashboard

1) *Setting the Customer' Location*: Customers can access their dashboard page after logging in. In their dashboard, first, they have to set their location. To set the location, the browser's geolocation API is used to display the map of the entire area where the customer is located. Once the customer points the pin to the exact location, the latitude and longitude of the marker are retrieved and stored. After pointing out the exact location, the customer needs to enter their detailed address like house number, street, area, pin code, etc. After hitting the submit button, the address data is sent to the backend via an axios POST request and gets stored in the database.
2) *Browsing the available Services*: After setting the location, customers can browse all the services available in their city along with the number of service professionals who are available to be booked under every service category. This information is fetched from the backend by making an Axios GET request with the city name as a query parameter and the customer's JWT token as the authorization header. In the backend GET request route, the city name is fetched from the passed query parameter, and a MongoDB find query is run to get the available services in the particular city which will be sent as the response to the frontend.
3) *Booking a Service*: Customers can book their desired service by hitting the book button under any one of the displayed services. A confirmation modal is displayed to confirm the booking from the customer's side. Again, an axios POST request is made to the backend containing the service category, city, latitude and longitude of the customer's location, and time as payload and the customer's JWT token as the authorization token. The backend route parses all the sent data and matches the customer with the service professional who is nearer to the customer's location. The Haversine formula is used to calculate the shortest distance between the customer's location and the service professional's default working location. Once the match is made, the booking details are generated along with the booking ID. The booking ID and the matched service professional's ID are stored in the customer's DB document, other booking details along with the customer's ID are stored inside the service history array of the Service professional's DB document. Customers can only book the service if at least one service professional is active in any of the categories. The algorithm also maintains a 30-minute gap between the allotted services for a particular service professional. If a service is allotted to a service professional, they can only receive their next booking after 30 minutes have passed, or if the previous service is rejected by either the customer or the service professional. This cool time of 30 mins is defined so that the service professional gets time to fulfill the accepted service, and also to avoid overwhelming of bookings for a single service professional.
4) *Viewing the Booking History*

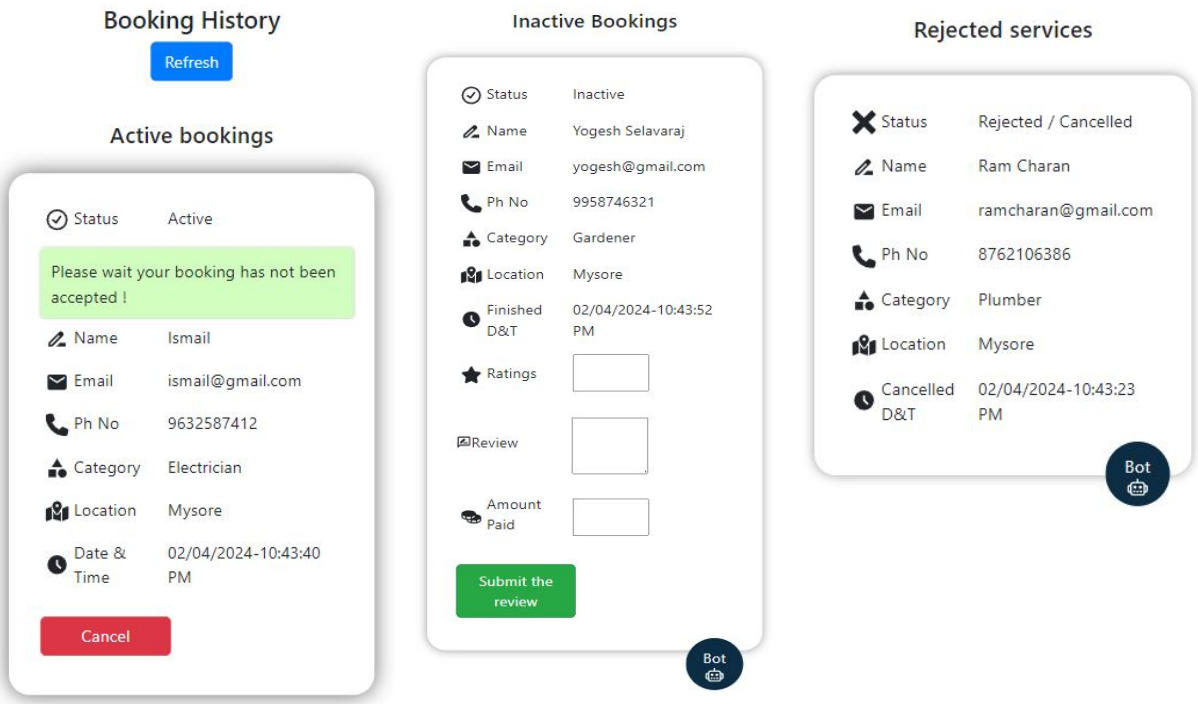


Fig.7 Customer's booking history section

Customers can view all their bookings including active, inactive, and rejected or canceled services in the booking history section. When the customer visits this section, a GET request is made to the backend to fetch all the past and active bookings made by the customer. All the booking history is stored inside the service history array of each service professional, and the service professional's ID along with the booking ID is stored in the bookings array of the customer's DB document. An aggregate query is used to obtain the final results by matching these IDs stored in the customer's DB document with those of the service professional's document. The obtained booking history is further categorized into active, inactive, and rejected services with active services appearing at the top and rejected services at the bottom.

5) Managing the Active Services

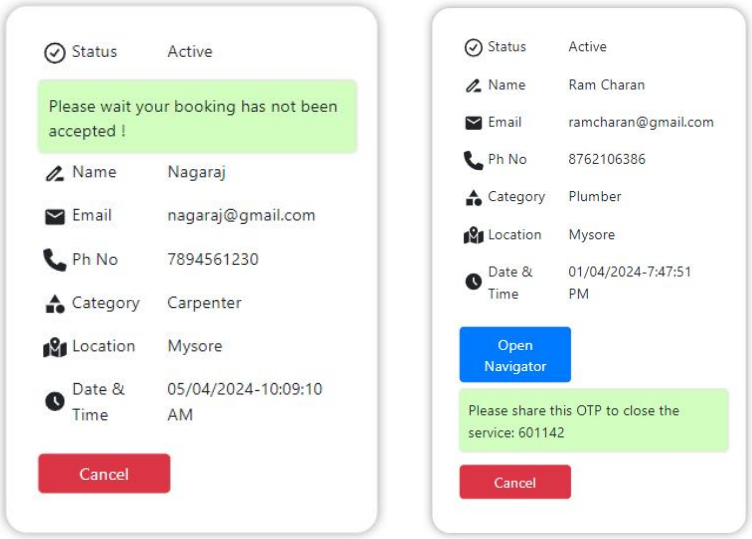


Fig.8 Active services section for customers

The booking history section also lets the customer manage their active bookings. All the relevant details of active services like allotted service professional's name, phone number, email, allotted date and time etc, are displayed in the active services section. Customers have to wait initially until their request is accepted by the service professional. Once the request is accepted, customers will be able to track the service professional's real-time location. Customers are also allowed to cancel the active service anytime they want. Also, when the service is completed, a unique 6-digit OTP is generated and displayed to the customer, which has to be shared with the service professional to mark the service as completed. All these requests are handled by the update service route in the backend, which will fetch the booking ID and service professional's email to find the exact booking, and other variables like completed and cancel to update the services.

6) Rating the Service



Fig.9 Rating screen

Once the service is closed, the service details will appear in the inactive section of the booking history page. Customers can fill the ratings, review, and the amount paid to the service professional. This will also be handled by the update service route in the backend, which will fetch the booking ID and service professional's email to find the bookings, and other data like rating, review, and amount to update the booking entries.

C. Service Professional's Module

1) Activating or Deactivating the Profile

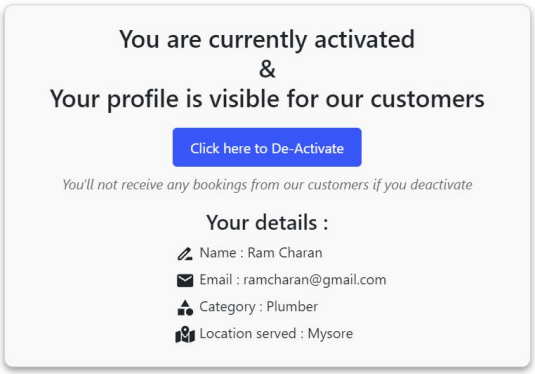


Fig.10 Service professional's dashboard

Service professionals can access their dashboard after logging in with their credentials. Initially, after creating the account, their profile has to be approved by the admin. Once their profile is approved, they can switch between an active or inactive state by hitting the activate/deactivate button. If their profile is deactivated, they will not be able to receive any bookings from customers as their profile will not be visible to customers while booking. If they wish to receive bookings from customers, then their profile should be activated. This can be achieved with a simple POST request which takes the active variable as payload along with the service professional's JWT token as the authorization token. This token is decoded in the backend, and the service professional's email is fetched from the token, which will be used to search for the exact document in the database. The available status is updated in the service professional's document based on the active variable passed.

2) Displaying the Statistics



Fig.11 Statistics screen

Statistics related to the booking data are displayed using the react-chartjs-2 library. A pie chart is used to represent the amount earned by the service professional in every month of the current year, while a line chart is used to represent the number of allotted services, completed services, and rejected services in every month of the current year. If the data is unavailable, a message is displayed indicating that the data is insufficient to depict the chart. The booking data is fetched from the database through a POST request. The obtained data is filtered according to the needs of the graph and then passed to the graph component for rendering the chart.

3) *Viewing the Booking History:* In the bookings section, service professionals will be able to view all their past and active bookings allotted to them. This is achieved with a simple GET request with JWT token as the header. The service professional's email is decoded from the token, and all the DB entries present in his service history array are sent as a response to the frontend. Active services are displayed at the top, inactive and rejected services are displayed after the active section. Service professionals can also view the ratings and reviews given by the customers regarding the service in the inactive services section.
4) *Managing the Active Services:* Service professionals can view all the allotted services in the active services section. Customer's details with location is displayed when they receive the service booking. Service professionals can either accept or reject the bookings allotted to them. If they reject the booking, the booking will be marked as rejected and displayed in the rejected section. After accepting the booking, they can view the customer's location along with the directions to reach the location. After completing the service, they can hit the "mark service as completed" button, after which a 6-digit OTP is displayed to the customer. The service professional has to ask for the OTP from his customer and enter the same in the form. After verifying the OTP, the service will be marked as closed. All these service handling functions are carried out in the update service route in the backend. The required entries are updated in the service professional's DB document based on the variables passed.

D. Admin's Module

1) Inviting new Admins

Admins can access their dashboard after logging in with their credentials. In their dashboard, in the "invite admins" section, admins will be able to add emails of other admins who are willing to register to the system. Other admins can register only if their email is present in the invite list; otherwise, they will not be able to register as admins. When the very first admin is registering in the system, then his/her email need not be present in any of the invite lists as there would be no admins to add the email to the invite list. This process of adding emails to the invite list is achieved with a simple POST request having the invitee's email as payload and the admin's JWT token as the authentication header. In the backend route, the invitee's email will be added to the invite list of the respective admin.

2) Approve Service Professionals

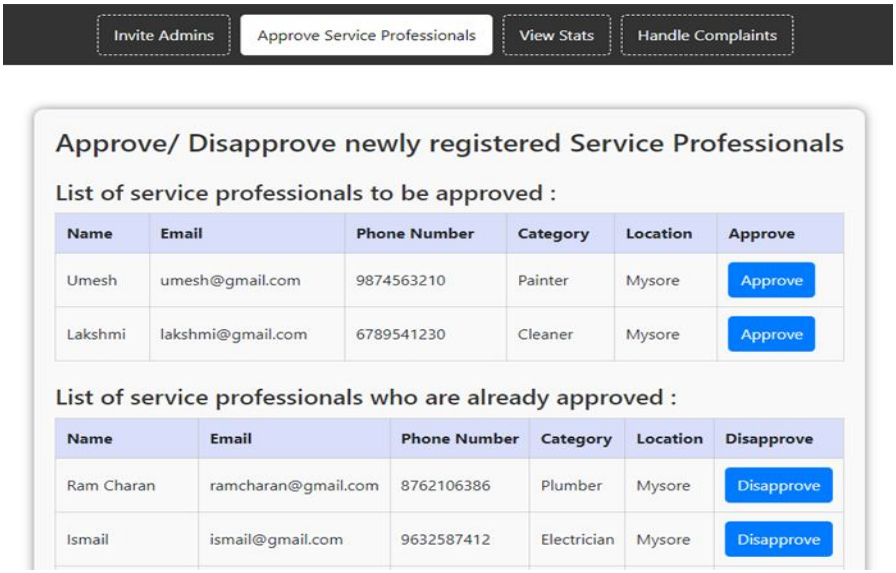


Fig.12 Admin's approving section

In this system, admins will have the power to activate or deactivate service professionals whenever required. Also, newly registered service professionals have to be approved by admins in order to receive bookings from customers. Service professionals who have to be approved are displayed in one table, and those who are already approved are displayed in another table. Additionally, other details of the service professionals like their name, email, phone number, etc, are displayed to the admin. Once the admin clicks on the activate or deactivate button, the required function is performed, and the respective service professional's profile will be marked as activated in the DB.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

3) *Handle Complaints:* Complaints submitted by the customers in the chatbot are displayed to the admin in the complaints section. Details of the customer like name, phone number, email, and complaint details are displayed in a table. Admins can add their answer to the complaint raised by the customer. Once the admin submits his reply, it can be viewed by the customer in the "show complaint status" section of the chatbot. This is handled again by a simple POST request which takes the reply given by the admin, service professional's email, and booking ID as payload and adds the same to the complaint details of the respective booking.

E. Real Time Location Tracking Module

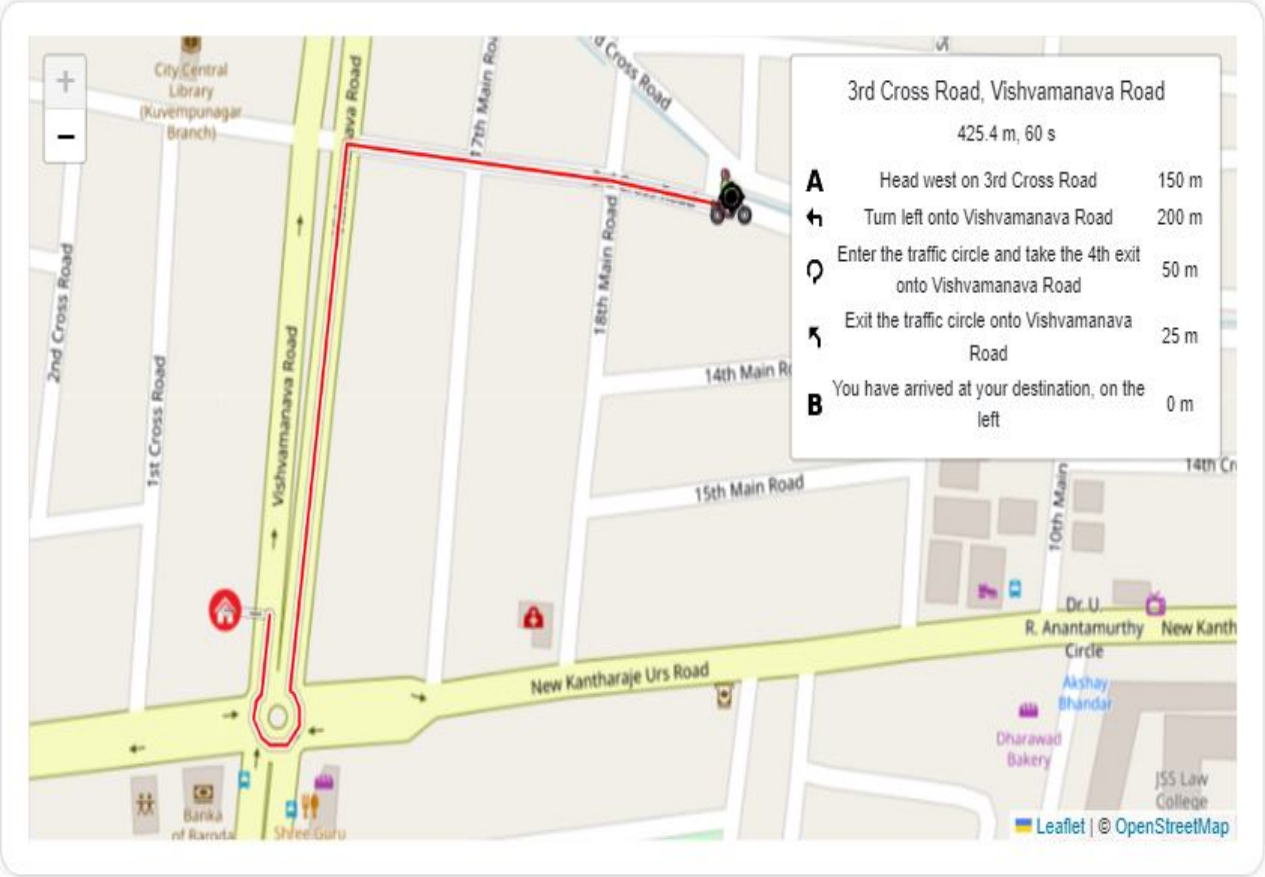


Fig.13 Location tracking screen

As mentioned in the previous section, when the service is accepted by the service professional, customers will be able to view the real-time location of the service professional, and the service professional will be able to view the customer's location along with the directions to reach the location. An open-source JavaScript map library called Leaflet is used for maps. Another open-source C++ library called OSRM (Open Source Routing Machine) is used to calculate the shortest path and display the directions to reach the destination. Initially, when the service professional hits the "Open Navigator" button, the customer's location is fetched from the database and marked as the destination on the map. Using OSRM, directions to reach the customer's location are also displayed. Also, when the service professional enters the location tracking frontend route, a socket connection is initiated, and a socket room is created with the service professional's email as the room name. The service professional's coordinates (latitude and longitude) are fetched continuously every 15 seconds using the browser's geolocation API, and these coordinates are broadcasted to the connected clients in the same room. Once the service professional's coordinates are updated, the updated location of the service professional along with his past coordinates are represented using marker icons on the map. When the user opens the navigator, he joins the socket room with the service professional's email and starts listening to the location updates which are sent every 15 seconds in that room. Once the service professional's coordinates are broadcasted to the connected room, it is read by the customer's end as well, which will again be updated on the map.

F. Chatbot Module

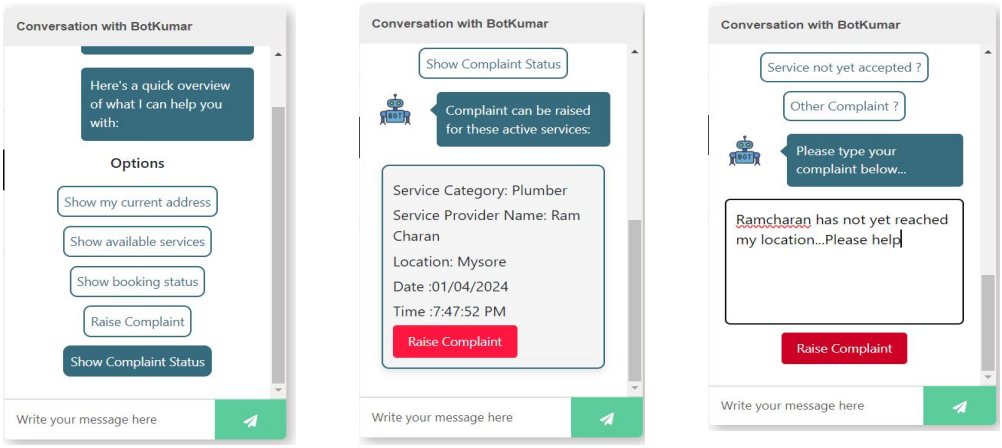


Fig.14 Chatbot interface

An open-source npm package called react-chatbot-kit is used to develop the chatbot. Customers will be able to interact with the chatbot through the chatbot interface on the dashboard page. Through this chatbot, customers can raise complaints for active services, view complaint status, view the available services in their location, view their booking status, and perform other functions. The customer's message will be parsed by the message parser react component in the frontend. The message parser identifies the exact option selected by the customer and triggers the required function present in the Action Provider component of the chatbot. A list of functions is present inside the action provider component which is associated with options chosen by the customer and also with the widget components where the response to each message can be handled efficiently in a separate react component.

1) *Raising a Complaint:* Customers will open the chatbot and select the "raise complaint" option in the displayed menu. Once the customer selects the option, the function present in the action provider component gets triggered, and its associated widget is called. Inside the widget, using React's Context API, the required data is passed from the customer's component to the widget component. The list of active services upon which complaints can be raised is displayed inside the chatbot. The customer is asked to select the complaint category and to write detailed complaint comments. Once the customer hits the submit complaint button, a POST request is made to store the complaint details in the DB and is sent to the admin to provide a resolution to the complaint.
2) *Showing the Complaint Status:* When the customer selects the "complaint status" option, the associated widget is called to display all the raised complaints. Once the customer selects the particular complaint, the response given by the admin for that raised complaint is displayed back to the customer in the chatbot; otherwise, a message saying that the admin has not yet responded is displayed. Again, the details of the complaint status and the reply given by the admin are fetched from the DB through an axios GET request. If the customer views the response given by the admin, then the complaint is marked as closed.
3) *Other Chatbot Responses:* As mentioned above, the chatbot is also equipped to provide other relevant information like available services in the customer's location, booking status regarding the number of active services and it's acceptance details, customer's currently selected address, etc. The data which is already present in the customer's react component is sent to the chatbot's component through the context api. Also the complaint details is forwarded to the widget providing the response through context api. The data sent is imported using UseData and useStateContext hook inside chatbot component, and all the responses to the customer's chatbot messages are given by accessing this data obtained from context api, along with the data obtained by accessing the database through post requests.

G. Blockchain Module

Blockchain development is done inside the Dfinity Canister SDK, which combines the Motoko backend module and React frontend module. All the ICP blockchain functions (smart contracts) and the stable variables that store the data are defined inside the Motoko file. The blockchain functions are called separately whenever required inside the React frontend module. Initially, before running the React module, the ICP blockchain has to be deployed by issuing the dfx start and dfx deploy commands. The dfx start command initiates a local instance of the ICP replica. This local testnet can be used to deploy and test canisters on the local machine. The dfx deploy command is used to deploy the smart contracts defined inside the Motoko file.

1) *Adding Completed Booking Details to the Blockchain:* In the Service Professional's booking history page, when the correct OTP is submitted, all the details of the respective booking are added to the local blockchain. The blockchain update call is defined inside the submitOTP function, which aggregates all the required data to be added to the blockchain and makes an update call when the data is ready. ICP's update calls have high resource consumption and cost cycles; hence, it takes more time to store the data in the ICP blockchain canister. Once the data is stored in the blockchain, a response is sent confirming that the data has been successfully added to the blockchain.
2) *Adding Review Details to the Blockchain*: In the User's booking history page, when the user hits the submit review button, the review details are added to the blockchain and also to the DB. The blockchain update call is defined inside the SubmitReview function, which aggregates the review details to be added to the blockchain and makes the update call when the user hits the submit review button. Again, it takes more than 3 seconds to make the update calls to the blockchain. Once the review details are added to the blockchain, a response is sent confirming that the review has been successfully added to the blockchain.
3) *Displaying Blockchain Data in the Admin's Dashboard*: The admin will be able to view all the completed bookings in the stats section of the dashboard page. A table containing all the details of the booking, including booking ID, customer's details, service professional's details, and review given by the customer, is displayed. All this data is fetched from the blockchain through a query call. Whenever admins enter the stats section, the ICP query call is made, fetching all the required data stored in the blockchain. Users cannot change the existing blockchain data through query calls, hence this type of function call has low resource consumption, low cost, and fast response time.

IX. SYSTEM TESTING

A. *Error Handling Testing*

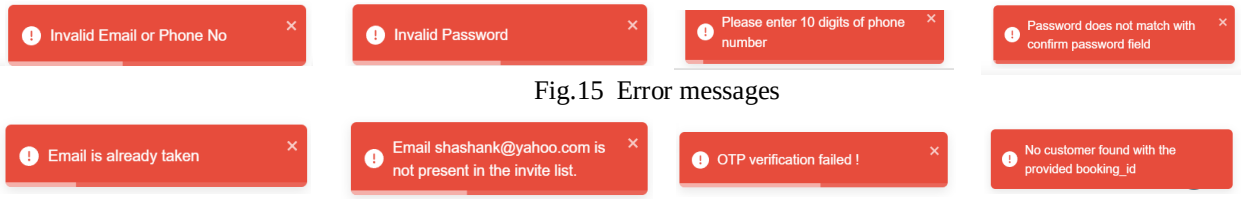


Fig.15 Error messages

Errors occurring due to invalid inputs, invalid authentication token, and other exceptions are properly handled in all backend routes. If an error occurs in a backend route or regarding user inputs, the same error is displayed using the react-toastify npm package.

On the login page, if the user enters an invalid email or phone number, an error message is displayed. If the phone number is not 10 digits long, the "Invalid phone number" error is triggered in the frontend logic itself. Similarly, if the user enters an invalid password for the correct email, an error message indicating that the password is invalid is displayed. If the user leaves any input field blank, a small popup indicates that the field is required.

On the register page, the password and confirm password fields are checked, and if they do not match, an alert message is displayed. If the user attempts to register with an email that is already registered, an alert message is displayed indicating that the email already exists.

In the admin's login, if a random user whose email is not in the invite list attempts to register, an error message is displayed indicating that the email is not present in the invite list.

In the service professional's booking section, when entering the OTP, if an incorrect OTP is entered, an alert message is displayed indicating that the OTP is incorrect.

While adding the user's reviews to the blockchain, if the booking details are not present in the blockchain, an error message is displayed indicating that the booking details are not present in the blockchain.

In the server routes, if any unexpected exception occurs or if there is any server error, all such errors are caught in the catch block of the code, and the error details are logged to the console.

B. *Regression Testing*

To implement blockchain functionality, several React components were modified and additional functions were added to existing components. Many of the ICP functions were added to the service professional's bookings component, customer's bookings component and admin’s component to include update and query calls. These code modifications did not introduce any unintended side effects or break existing functionality. All previously functioning features will continue to work after these modifications.

©IJRASET: All Rights are Reserved

C. Integration Testing

Throughout the system, various interactions occur between components such as Node.js, React, MongoDB, and the ICP blockchain canister. Extensive manual testing has been conducted to verify these interactions between components. Axios POST and GET requests made from React to Node.js were logged in the Node.js terminal, and MongoDB query results processed in Node.js were also logged, ensuring smooth integration between Node.js, React, and MongoDB. The blockchain data added from React was manually verified using the Candid UI interface provided by the Dfinity Canister SDK. All tests conducted confirmed that all components worked together as expected. Any errors related to integrations between components were resolved through this integration testing process.

D. API Testing

The Postman tool is used to test all the responses obtained from the backend routes for all possible inputs. Initially, after the development of backend routes, every route is tested for all possible inputs. If there is any error in the response or if the response does not match the expected response, then such cases are handled properly in the route code. Before integrating React with Node.js, all the API routes have been thoroughly tested, and all possible errors have been fixed.

X. FUTURE ENHANCEMENTS

A. Integrating the system with WhatsApp or Telegram API

Currently, as this is a full-stack web application, users will have to access the system by interacting with the website. However, it will become easier for users to access the system if the backend part of this system is integrated with widely used mobile applications like WhatsApp or Telegram.

B. Developing a system to send acknowledgments /confirmations through SMS and emails

Currently, all the confirmations and acknowledgments are displayed on the website's dashboard itself. For more convenience, acknowledgments/confirmations like booking details, OTPs, bills, etc., can be sent through SMS or emails to the users.

C. Implementing messaging feature

In the existing system, it is not possible for customers to chat with the allotted service professional. For more convenience, a messaging feature can be developed which allows customers to chat with the service pro and prevents the customer from calling.

D. Developing an Android application

An Android application can be developed for this system so that everything can be accessed by users conveniently at their fingertips. Also, in the existing system, location tracking can only be done if the service pro actively keeps the website open. However, with the Android app, the location details can be updated in the background as well, enhancing the location tracking system and the user's convenience in accessing the system.

XI. CONCLUSION

In conclusion, our Full Stack Web Application for On-Demand Household Service promises a user-friendly, efficient and secure experience for all. Customers will effortlessly discover, book, and track the services, access transparent ratings and reviews, and enjoy a secure and seamless experience. Service professionals will efficiently manage appointments and track earnings. Admins play a crucial role in ensuring the reliability of the system by overseeing worker registrations , managing user complaints and maintaining the integrity of the platform. Additionally, the implementation of blockchain technology will provide tamper-proof storage for customer reviews and service professional’s work records. Furthermore, the integration of a Machine Learning Chatbot will enhance user interactions, providing timely and relevant information to the customers.

All the proposed designs have been successfully implemented and thoroughly tested. A live working website has been developed and deployed, incorporating all the proposed features. The website is designed with attractive CSS, enhancing the overall user interface. Additionally, this web app is made responsive so that it can be accessed even on mobile phones. Blockchain implementation has been carried out locally as described, and a chatbot has also been developed as designed. In conclusion, our Full Stack Web Application for On-Demand Household Services is poised to transform the skilled workforce connection system, making it efficient, secure, and engaging for all users.

©IJRASET: All Rights are Reserved

XII.ACKNOWLEDGMENT

First and foremost, we ought to pay due regard to our renowned institution, The National Institute of Engineering, Mysuru, which provided us a platform and an opportunity to present the paper.

We are extremely thankful to Dr. Rohini Nagapadma, Principal of NIE, Mysuru, for providing us the academic ambiance and everlasting motivation to carry out this work and shape our careers.

We express our sincere gratitude to Dr. Girish, Head of the Department of Information Science & Engineering at NIE, Mysuru, for his stimulating guidance, continuous encouragement, and motivation throughout the course of the present work.

We extend our gratitude to our Guide Dr.Shashank Dhananjaya and Co-Guide Mr.Rajesh N, Dept. of Information Science and Engineering, NIE, Mysuru for their support and guidance over the entire course of work.

We take this opportunity to thank all our friends and classmates who always stood by us in difficult situations, helped us with some technical aspects, and last but not least, we wish to express our inspiration and sense of gratitude to our parents who were a constant source of encouragement and stood by us as pillars of strength for completing this work and course successfully.

REFERENCES

[1] K. Saundariya, M. Abirami, K. R. Senthil, D. Prabakaran, B. Srimathi and G. Nagarajan, "Webapp Service for Booking Handyman Using Mongodb, Express JS, React JS, Node JS," 2021 3rd International Conference on Signal Processing and Communication (ICPSC), Coimbatore, India, 2021, pp. 180-183.

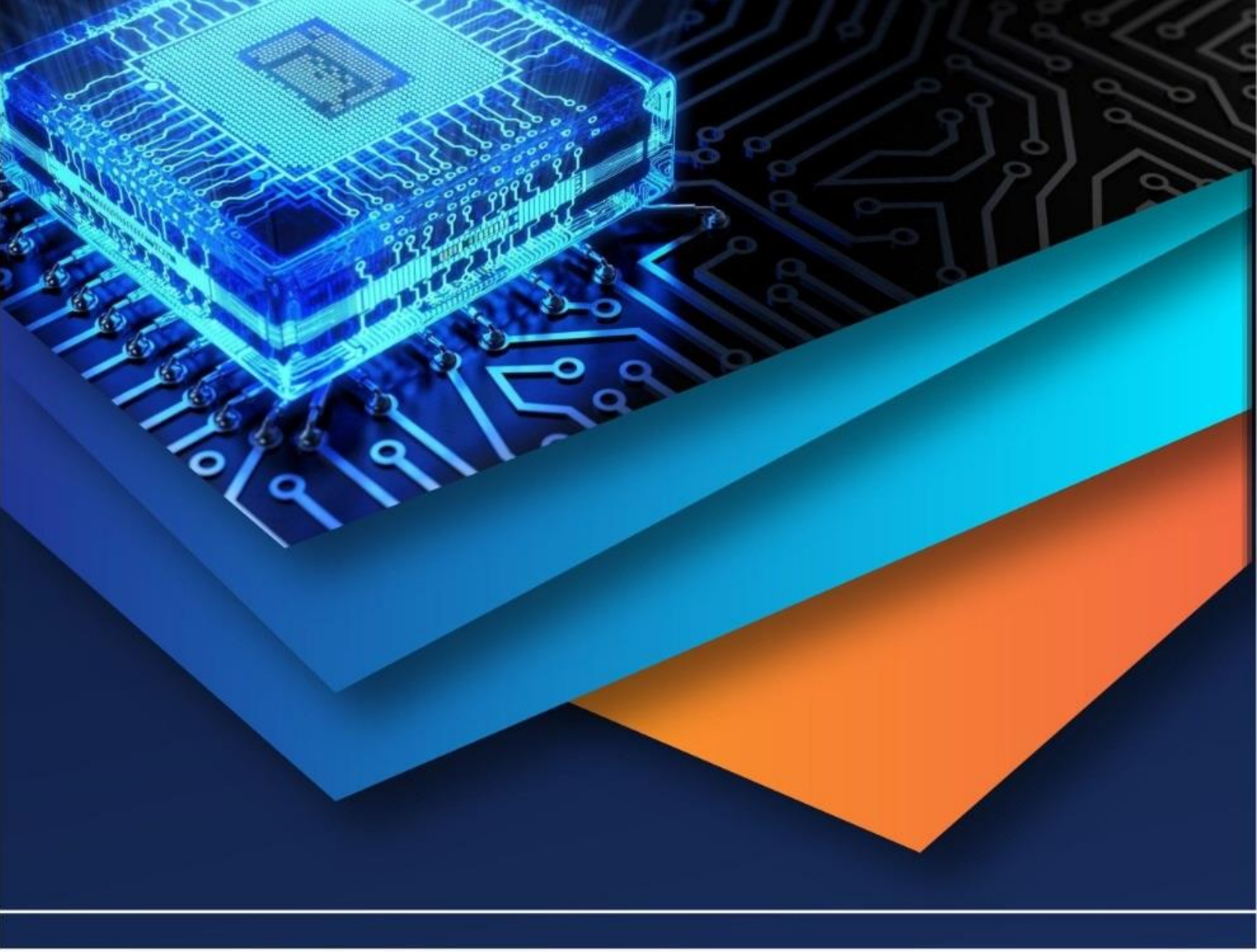
[2] K. Aravindhan, K. Periyakaruppan, T. S. Anusa, S. Kousika and A. L. Priya, "Web Application Based On Demand Home Service System," 2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS), Coimbatore, India, 2020, pp. 1458-1462.

[3] G. Baskaran, K. Saundariya, D. Prabakaran and R. Senthilkumaran, "A Web Application Based Administration Panel For Handyman Services," 2022 IEEE Delhi Section Conference (DELCON), New Delhi, India, 2022, pp. 1-5.

[4] Hegde Sharaj Bhaskar Shyamala, Krishnamoorthy Rao, Padmanabha Bhandarkar, Prateek Prakash Vetekar, Geetha Laxmi, “An Android Application for Home Services,” May 2020, International Journal of Engineering Research and Technology (IJERT).

[5] Apeksha Adekar, Aakash Dalvi, Pratik Gharat, Pushti Ratanghayra, “Household Veritas - A platform that provides household services,” Mar 2023, International Journal of Engineering Research and Technology (IJERT).

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)