CureSphere

Andreas Loutzidis
Computer Science
Columbia University
New York, USA
al4419@columbia.edu

Shwetha Subramanian
Computer Science
Columbia University
New York, USA
ss6618@columbia.edu

Weiyao Li
Statistics
Columbia University
New York, USA
wl2872@columbia.edu

Lynn Kim
Computer Science
Columbia University
New York, USA
ck3127@columbia.edu

***Abstract*—The complexity of the healthcare system often results in patients struggling to navigate through it and manage their healthcare needs effectively. This paper presents a serverless, full-stack, cloud-based web application implemented in Amazon Web Services that aims to simplify the process of finding and receiving healthcare services for patients. The platform not only helps patients search for doctors and book appointments but also assists them throughout their treatment process. The application incorporates a unique medicine price comparison functionality, addressing the issue of cost. The paper discusses the motivation, literature review, target audience, success criteria, data sources, APIs, and architecture of the application, with a focus on its implementation using AWS services such as Lambda, DynamoDB, and OpenSearch.**

***Index Terms*—serverless architecture, Amazon Web Services, healthcare platform, patient management, doctor search, appointment booking, treatment assistance, medicine price comparison, Lambda, DynamoDB, OpenSearch, cloud-based application, full-stack, user-friendly**

I. Introduction

Healthcare systems worldwide have become increasingly complex, often making it difficult for patients to navigate through the intricate processes of finding the right doctors, scheduling appointments, receiving treatment, and managing their healthcare needs. To address these challenges, we have developed a user-friendly, full-stack, cloud-based web application on a serverless architecture implemented in Amazon Web Services (AWS). Our platform allows patients to search for doctors, book appointments, receive treatment, and manage their healthcare needs, all in one place.

The unique value proposition of our application lies in providing patients with a comprehensive solution that enables them to complete their entire treatment cycle through the platform. Unlike other healthcare-related apps, our platform not only helps patients find and book appointments with doctors but also assists them throughout their treatment process, allowing doctors to provide short comments on medication and other treatments. The platform also includes a medicine price comparison functionality, which aims to address the issue of cost and help patients save money on their healthcare expenses.

In this paper, we provide an overview of the motivation, literature review, target audience, success criteria, data sources, APIs, and architecture of the application. We will also discuss the implementation of the platform using AWS services such as Lambda, DynamoDB, and OpenSearch, providing insights into the development process and the overall effectiveness of the platform in meeting the needs of the target audience. By streamlining the healthcare process and providing a more convenient and efficient way for patients to manage their healthcare needs, our platform aims to significantly improve the healthcare experience for patients.

II. Architecture

The architecture of this project is centered around a serverless, full-stack, cloud-based web application designed to simplify and enhance the healthcare experience for patients. Utilizing a modular and coherent code structure, the architecture is organized into key components, including Lambda functions for core API functionalities and a frontend folder for user interface implementation. By leveraging AWS services such as Lambda, DynamoDB, and OpenSearch, the platform achieves scalability, efficiency, and cost-effectiveness. This well-organized architecture ensures that the project remains maintainable and responsive to user feedback, ultimately contributing to a more streamlined and satisfying patient experience within the healthcare domain.

III. Key Design Details

A. doctor submit : POST

When a user(doctor) clicks on an appointment from the current appointment lists, it redirects to a different page where he can submit a medicine/feedback form. After entering medicine and feedback and clicking on a submit button, it is submitted. Then 'doctor input' lambda function is triggered, with feedback, medicine, and appointment id in the header. With the appointment id, we query the results in the open search, and the data is updated with feedback and medicine. Lastly, the time slot gets freed as the appointment has ended.

B. get user information : GET

When the user portal page loads, based on the user role, getPatient or getDoctor lambda function is triggered. After parsing each data section, full user data is displayed in a table.

C. get current/past appointments : GET

With email(user id), user role(patient or doctor), appointment type(current or past) passed, we query all of the appointment data in open search with the email. After checking if the email matches, if feedback or medicine is an empty string, the data is appended to current appointments. If feedback or medicine is not an empty string, then it is appended to the past appointments. The result is sorted in time order and one of the result is returned and displayed in the portal based on the appointment type.

D. book appointment : POST

It facilitates appointment booking by querying relevant information from a DynamoDB table, sending confirmation emails to the patient and doctor using Amazon SES, storing appointment details in an Elasticsearch index, and updating the doctor's availability in DynamoDB by marking the booked time slot as ”taken”. The API leverages AWS services like Lambda, DynamoDB, Elasticsearch, and SES to provide these functionalities.

E. medicine comparison : GET

It queries a DynamoDB table named ”medicine data” to find the lowest three prices for a given medicine name. The API takes in the medicine name from an HTTP GET request and calls the get medicine comparison function to query DynamoDB for the lowest three products with the same medicine name. It then sorts the results by price and returns the lowest three prices along with the seller name and zip code as a JSON response. The API also includes custom JSON encoding to handle decimal types. If the medicine name is not found in the database, it returns an error response with a message. If no medicine name is provided in the request, it returns an error response indicating that no latest medicine name was found for the given appointment ID. The API uses AWS services such as Lambda and DynamoDB to provide these functionalities.

F. upload to DB : POST

It inserts data from a JSON file named ”sampleMedicine-Data.json” into a DynamoDB table named ”medicinedata”. The API uses the insert data function to perform the insertion, which takes a list of data and inserts it into the specified table. The API uses the boto3 library to interact with DynamoDB and json library to read the data from the JSON file. If no database connection is provided, the API uses the default boto3 resource to connect to DynamoDB. The function overwrites any items with the same index as provided in the JSON data. The API doesn't return any response to the caller, but logs the DynamoDB response to CloudWatch. The API uses AWS services such as Lambda and DynamoDB to provide these functionalities.

G. call lex : POST

It serves as an interface between the frontend and AWS Lex, a conversational AI service. Upon receiving a text message from the frontend, the Lambda handler function processes the user's input, sending it to the Lex bot for further interpretation. The Lex bot, identified by its specific 'bot Id' and 'bot AliasId', processes the text in English and returns a response. The Lambda function then formats the response, including any messages from Lex, with the necessary headers and structure. Finally, it sends the formatted response back to the frontend, allowing seamless interaction between the user and the conversational AI service.

H. lex get recommended doctors validation : GET

It is designed to handle user requests to recommend doctors based on their symptoms and location. The code initializes a logger, configures a DynamoDB client, and defines helper functions for validating user input, querying the database, and calculating the distance between zip codes. The main handler, lambda handler, routes incoming requests based on their intent. The primary intent, Get Recommended Doctors, processes user input to obtain the patient's zip code and symptoms, validates the zip code, and queries the database to find relevant doctors. It retrieves specialty information based on the symptoms, and finds top doctors within the patient's vicinity who specialize in the identified specialties. Finally, the function returns a list of recommended doctors as a response.

I. get doctor availability : GET

It retrieves the availability of a doctor for a specific date. It initializes a DynamoDB client, extracts the doctorId from the incoming event's headers and the date from the queryStringParameters. The function then queries the doctors table for the specified doctor and extracts their availability windows. It converts the input date into a day of the week, and checks if the doctor is available on that day. If the doctor is available, it filters the available slots and returns them in the response. If the doctor is not available, it returns a message stating that booking is unavailable. The response includes appropriate headers for CORS configuration, allowing cross-origin requests.

IV. CODE STRUCTURE

The code is organized into a coherent and modular structure. It comprises a lambdas folder (API), a frontend folder, .gitignore, project architecture documentation, README.md, and configuration YAML. The structure and functionality of each component are described below:

- **Lambda folder:** This folder contains 16 Lambda functions (APIs) that are designed to carry out the core functionalities of the application. These essential functions encompass:
  - Book Appointment
  - Call Lex
  - Create Doctor
  - Create Patient
  - Doctor Input
  - Get Appointment Link
  - Get Doctor Availability
  - Get Feedback
  - Get Medicine Comparison
  - Get Patient
  - Get Doctor
  - Get Recommended Doctors Validation from Lex
  - Lex Validation Fulfillment

  - Login Handler
  - Put Symptom to Specialty
  - Upload Medicine to DB
- **Frontend folder**: Contains the user interface implementation for the system, including 11 HTML screens, the generated API Gateway SDK, and corresponding JavaScript and CSS files. The screens include:
  - Book Appointment
  - Chat
  - Display Doctor
  - Doctor's Past Appointments
  - Doctor's Portal
  - Doctor's Submission
  - Index
  - Login
  - Medicine Comparison
  - Patient Past Appointments
  - Patient Portal
- **.gitignore**: Specifies the files and directories that should be ignored by Git version control.
- **Project architecture**: Describes the overall architecture of the system.
- **README.md**: Provides an overview of the project, instructions for use, and other important information.
- **Configuration.yaml**: Contains configuration settings for the project.

V. Results

The project's success is evaluated based on the target audience's ability to complete the following tasks:

- Users must register by providing a valid email address, creating a unique username, and establishing a secure password. This process ensures the protection of user data and the maintenance of confidentiality.
- Once registered, users can securely log in to their accounts using the credentials they previously set up. This authentication process safeguards against unauthorized access.
- After successful login, user-provided data is stored in the patient record associated with their account, enabling a comprehensive and easily accessible health history for both patients and healthcare providers.
- Users can schedule a consultation with a recommended doctor based on their needs by selecting from the available time slots. This streamlined process facilitates convenient appointment booking and enhances the overall user experience.
- Upon scheduling an appointment, users will receive email notifications containing pertinent details, ensuring they stay informed and well-prepared for their upcoming consultations.
- The system offers users the option to create calendar invites and reminders for scheduled appointments, thereby promoting punctuality and minimizing the risk of missed consultations.
- During the allocated time slot, users can interact with their healthcare providers through chat or video calls, enabling a personalized and engaging consultation experience, regardless of their physical location.
- After the consultation, any prescribed medications or treatments will be stored in the patient's record, granting easy access for future reference and promoting better adherence to treatment plans.
- Following the treatment session, healthcare providers have the ability to leave feedback and recommend medications or treatments tailored to the patient's specific needs. This fosters a comprehensive understanding of the patient's condition and enables more effective treatment plans.
- Patients can access the doctor's feedback through their account, ensuring they remain informed about their health status and can take the necessary steps towards improvement.
- To assist patients in obtaining the recommended medications, the system offers a medicine comparison functionality. By utilizing this feature, patients can receive information on the three lowest-priced options for the prescribed medication available in their zip code area.
- This comparison tool not only considers the matching medicine name but also takes into account the proximity of the pharmacies or sellers, ultimately providing patients with convenient and cost-effective solutions for their medication needs.

VI. Conclusion

In this paper, we have presented a serverless, full-stack, cloud-based web application implemented in Amazon Web Services that simplifies the process of finding and receiving healthcare services for patients. The platform provides a comprehensive solution, allowing patients to search for doctors, book appointments, receive treatment, and manage their healthcare needs, all in one place. Furthermore, the application includes a unique medicine price comparison functionality to address the issue of cost and help patients save on their healthcare expenses.

We have discussed the application's architecture, which leverages AWS services such as Lambda, DynamoDB, and OpenSearch to create a scalable and efficient platform. In the code structure and results section, we demonstrated the implementation details of the platform, showcasing how different AWS services are combined to provide the desired functionalities. Additionally, we have provided a video recording uploaded on YouTube to exhibit the platform in action, highlighting its user-friendliness and effectiveness.

In conclusion, our web application aims to provide patients with a more positive and satisfactory healthcare experience by streamlining the process and making it more efficient. The serverless architecture and the use of AWS services enable the platform to be scalable, maintainable, and cost-effective. By continuously refining the platform based on user feedback and incorporating new features, we hope to further improve the healthcare experience for patients, ultimately contributing to better overall patient outcomes and satisfaction.