

Cloud-based Patient Engagement Platform

*A Practice School Report submitted to
Manipal Academy of Higher Education
in partial fulfilment of the requirement for the award of the degree of*

BACHELOR OF TECHNOLOGY

in

Computer Science & Engineering

Submitted by

Abhijeet Sahdev

170905316

Under the guidance of

Mr. Parth Srivastav
Chief Technical Officer
MyHealthToday LLC

&

Dr. Geetha M
Professor



MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

July 2021



MANIPAL INSTITUTE OF TECHNOLOGY

MANIPAL

(A constituent unit of MAHE, Manipal)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Manipal

31st June 2021

CERTIFICATE

This is to certify that the project titled **MyHealthJournal** is a record of the bonafide work done by Abhijeet Sahdev (*Reg. No. 170905316*) submitted in partial fulfilment of the requirements for the award of the Degree of Bachelor of Technology (B.Tech.) in **COMPUTER SCIENCE & ENGINEERING** of Manipal Institute of Technology, Manipal, Karnataka, (A Constituent Institute of Manipal Academy of Higher Education), during the academic year 2021.

Parth Srivastav

Chief Technology Officer

MyHealthToday LLC

Prof. Dr. Ashalatha Nayak

HOD, CSE Dept.

M.I.T, MANIPAL

Internship Offer Letter



TO WHOM IT MAY CONCERN

This is to confirm that ABHIJEET SAHDEV will be working at myHealthToday LLC as an Software Developer Intern from December 14, 2020 to June 4, 2021.

During this Internship, he will work on the implementation of backend services, mobile application and Web frontend application for MyHealthToday.

A handwritten signature in brown ink, appearing to read "P. Srivastav".

Parth Srivastav

Chief Technology Officer
MyHealthToday LLC.

Project Completion Letter



June 30th, 2021

To whom it may concern

This is to certify that Mr. **Abhijeet Sahdev** has interned with MyHealthToday LLC on the project MyHealthJournal from December 2020 to June 2021. Mr Abhijeet Sahdev has completed the project satisfactorily during his time at MyHealthToday

For MyHealthToday LLC

A handwritten signature in dark ink, appearing to read "P. Srivastav".

Parth Srivastav
Chief Technology Officer
MyHealthToday LLC

ACKNOWLEDGMENTS

The completion of this project would not be possible without Dr D Srikanth Rao, Director and Dr Ashalatha Nayak, HOD, Department of Computer Science and Engineering, who helped me in securing this internship. I also convey my sincere gratitude to Dr Geeta M, Professor, for her guidance and support towards completing several reports and documents that had to be submitted. Lastly, I would like to thank Mr Parth Srivastav, CTO, MyHealthToday LLC, for his mentorship throughout this internship.

ABSTRACT

A recent study shows greater patient engagement leads to improved health outcomes. Thus, empowering senior patients to self-report health information can be an effective measure to combat chronic conditions. Thus, the objective of this project was to develop a chronic disease-specific symptom tracking service that enables patients to use Amazon Alexa to voice report their disease-related symptoms or simply log their symptoms on their phones/websites. Then, the collected information will be analyzed based on predefined rules and send an actionable report to the physician for patient follow up.

As a part of this project, the existing DynamoDB Database design was modified, and we built new REST APIs to store data securely and ensure fast response to the front end. For scalability, serverless architectures and design patterns are used. These challenges were dealt with using Agile and Test-Driven Development paradigms.

These new APIs and tweaks to the database designs helped us reduce the creation time of new resources via the frontend and reduced the response time of existing APIs.

By the end of this internship, we successfully managed to deploy the mobile application on Google Play Store in the US, and it's under review in the Apple Store. The website is now responsive across different sizes and now has two primary users: providers and patients.

The tech stack of this project comprises NodeJS, DynamoDB, React Native, React.js, and AWS.

Table of Contents		
		Page No
Acknowledgement		i
Abstract		ii
List Of Tables		iv
List Of Figures		v
Chapter 1	INTRODUCTION	1
1.1	Introduction	1
1.2	General Discussion and Present-Day Scenario	1
1.3	Motivation	1
1.4	Objective	2
1.5	Project Work Schedule	2
Chapter 2	BACKGROUND THEORY	
2.1	Introduction to the project title	3
2.2	Background Theory	3
2.2.1	React	3
2.2.2	React Native	4
2.2.3	AWS Lambda	4
2.2.4	AWS Amplify	5
2.2.5	AWS Dynamo DB	5
2.2.6	AWS Cognito	5
2.2.7	Alexa Skills	6
2.2.8	CI/CD on AWS	6
2.3	Conclusions	6
Chapter 3	METHODOLOGY	
3.1	Introduction	8
3.2	Methodology	8
3.2.1	Agile Software Development	8
3.2.1.1	Trello	9
3.2.2	Service Layer Architecture	9

	3.2.3	Components	9
	3.2.3.1	AlertComp	10
	3.2.3.2	SymptomCard	10
	3.2.3.3	Feedbacks	11
	3.2.4	Designing, Building and Deploying APIs	11
	3.2.4.1	Feedbacks	11
	3.2.4.2	Symptom Creation	12
	3.2.4.3	Storing Review History	14
	3.3	Conclusions	14
Chapter 4	RESULT ANALYSIS		
	4.1	Introduction	15
	4.2	Result Analysis	15
	4.2.1	Reports	15
	4.2.2	Symptoms	16
	4.3	Conclusions	17
Chapter 5	CONCLUSION AND FUTURE SCOPE		
	5.1	Summary	18
	5.2	Conclusions	18
	5.3	Future Scope of Work	18
REFERENCES			19
ANNEXURES (OPTIONAL)			21
PLAGIARISM REPORT			30
PROJECT DETAILS			32

LIST OF TABLES

Table No	Table Title	Page No
1	Feedbacks Database Design	11
2	Review History Database Design	14
3	Report Creation Time over five months	15
4	Symptom Creation Time over five months	16

LIST OF FIGURES

Figure No	Figure Title	Page No
1.	How Lambda Works	4
2.	AWS CodePipeline	6

CHAPTER 1

INTRODUCTION

1.1 Introduction

This chapter explains the need for the project. It highlights the problems faced today in the healthcare sector and how this project aims to resolve them. We shall also discuss the main objectives and how we spent the last six months achieving them.

1.2 General Discussion and Present-Day Scenario

Approximately **85%** of seniors have at least one chronic condition (CC), and **60%** have at least two CCs [1]. According to Premiers, about 30% of emergency department visits among patients with common chronic conditions are potentially unnecessary, leading to **\$8.3 billion** in additional costs for the healthcare industry [2]. It's expensive. Moreover, COVID-19 has negatively affected the patient-physician relationship in obvious ways, like limitations to in-person visits and physician burnout. Given these circumstances, there's a need to adapt to telemedicine.

1.3 Motivation

By focusing on the pre-clinic phase and between visits, this service empowers patients to record their health concerns through a mobile app, voice assistant and securely transform the data into reports/notes that doctors can utilize in their designated electronic health records (EHR).

Thus, empowering senior patients to self-report health information can be an effective measure to combat chronic conditions.

1.4 Objective

The main objective of this project was to develop a chronic disease-specific symptom tracking service that enables patients to use Amazon Alexa to voice report their disease-

related symptoms or simply log their symptoms on their phones/websites. Then, the collected information will be analyzed based on predefined rules and send an actionable report to the physician for patient follow up.

1.5 Project Work Schedule

The following timeline provides a brief overview of the project schedule -

➤ *January 2021*

- Learn prerequisites (React Native).
- Mobile application development

➤ *February 2021*

- Mobile application development.
- Learnt React.

➤ *March 2021*

- Web application development.

➤ *April 2021*

- Submission of report & evaluation (4 Months project)
- Learn prerequisites (AWS course on Udemy)
- Backend Development

➤ *May 2021*

- Backend and Web Development.

➤ *June 2021*

- Alexa and Chrome Voice Interface Development

CHAPTER 2

BACKGROUND THEORY

2.1 Introduction to the Project Title

As mentioned earlier, the main objective is to build a symptom tracking service. To track something, it must be created and stored. Stored records help us analyze a particular condition or how a symptom has unfolded over the past X months or years, what changes has the patient observed over the last few weeks, etc. These records can help a doctor diagnose a patient more efficiently since a patient would take their time to log in the details of a symptom as and when they occur. They no longer need to remember these details for the next appointment.

Moreover, frequent visits to their doctors can now drop once or twice in a few months since the doctors can now access these records. All records are stored and processed in a remote data centre. In our case, this remote data centre is owned and managed by Amazon – AWS. Now, let's discuss the tech stack used in this project in brief.

2.2 Background Theory

2.2.1 React

A top-of-the-line JavaScript library for creating user interfaces. Some of its features are as follows: -

- Declarative: React allows creating interactive UIs a breeze. Create simple views for each state of our app and React will update and make the appropriate components as data changes. Declarative views improve the predictability and debuggability of our code.
- Component-based: Compose encapsulated modules that maintain their state to create complex user interfaces. We can quickly transfer rich data through our app and hold the state out of the DOM since component logic is written in JavaScript rather than templates.
- Learn Once, write anywhere: Compose encapsulated modules that maintain their state to create complex user interfaces. We can quickly transfer rich data through

your app and hold the state out of the DOM since component logic is written in JavaScript rather than templates.[3]

2.2.2 *React Native*

React Native blends the best features of native development with React. We can use it in our current Android and iOS projects right now, or we can start from scratch and create a brand-new app. Our app can use the same native platform APIs as other apps since React primitives render to native platform UI. We can create platform-specific versions of modules to code from one codebase across several platforms. React Native allows a single team to manage two networks while sharing a similar technology—React. It allows us to create genuinely native apps without compromising the user experience. It includes a core set of platform-independent native components, such as View, Text, and Image, that map to the platform's native UI building blocks. React components use React's declarative UI paradigm and JavaScript to wrap existing native code and communicate with native APIs. This opens native app development to entirely new developers' teams and allows current native teams to work even more quickly.[4]

2.2.3 *AWS Lambda*

It is a serverless compute service that allows us to run code without providing or managing servers, writing workload-aware cluster scaling logic, keeping event integrations up to date, or managing runtimes. We can run code for practically any application or backend service with Lambda, and we don't have to worry about administration. Simply upload our code as a ZIP file or container image. Lambda will automatically and precisely allocate compute execution power and run our code in response to the incoming request or event, regardless of traffic volume. We can make your code activate automatically from 140 AWS services, or you can call it directly from any site or mobile app. [5]

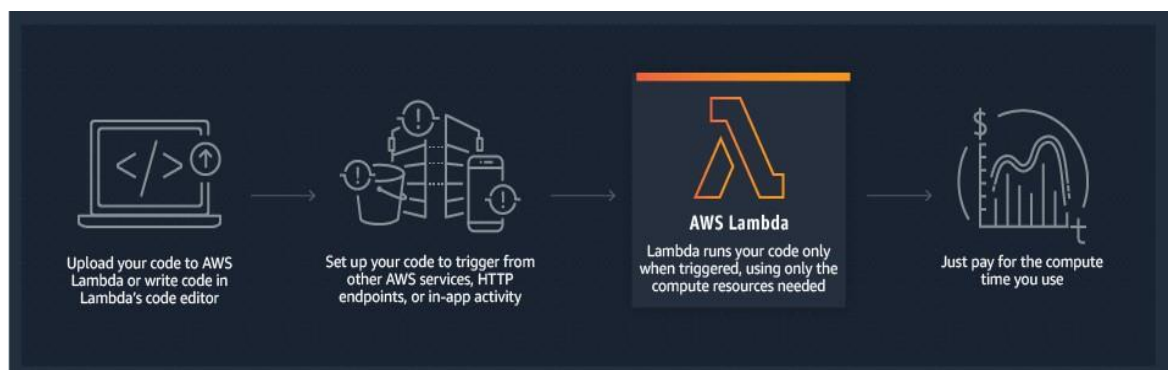


Fig 1: How Lambda Work

2.2.4 *AWS Amplify*

It is a collection of tools and resources that can be used together or separately to assist frontend web and mobile developers in developing scalable full-stack applications using AWS. We can configure app backends and bind your app in minutes with Amplify, deploy static web apps in a few clicks with Amplify, and control app content outside the AWS console with Amplify. Amplify works with various web frameworks and mobile platforms, including JavaScript, React, Angular, Vue, Next.js, Android, iOS, React Native, Ionic, and Flutter. With AWS Amplify, you will get to market quicker. Under the amplify umbrella, we make use of DynamoDb, Aws Cognito and other services.[6]

2.2.5 *AWS Dynamo DB*

It's a key-value and document database with performance in the single-digit milliseconds at any scale. It's a fully managed, multi-region, multi-active, persistent database for internet-scale applications with built-in security, backups and restores, and in-memory caching. DynamoDB can handle more than 10 trillion requests per day and 20 million requests per second at its peak.[7]

2.2.6 *AWS Cognito*

Amazon Cognito allows us to add user sign-up quickly and easily, sign-in, and access management to your online and mobile apps. Amazon Cognito provides sign-in with social identity providers such as Apple, Facebook, Google, and Amazon and grows to millions of users. It provides access control solutions for AWS resources. We can build roles and assign users to them so that your app can only access the resources that have been granted to each user. Alternatively, we can utilize identity provider attributes in AWS Identity and Access Management permission policies to restrict resource access to users who fulfil specific attribute criteria. [8]

2.2.7 *Alexa Skills*

They're like applications that give us a new way to share our content and services with Alexa. Customers may use their voices to do things like check the news, listen to music, play a game, and more using skills (In our case, an example could be to create a new

symptom). Skills may be published in the Alexa Skills Store by businesses and individuals to reach and delight consumers across hundreds of millions of Alexa devices. [9]

2.2.8 CI/CD on AWS

New code is submitted on one end, tested via several phases (source, build, try, staging, and production), and then published as production-ready code. Fig 2 shows how CI/CD (continuous integration/ continuous delivery) can be pictured like a pipeline. [10]

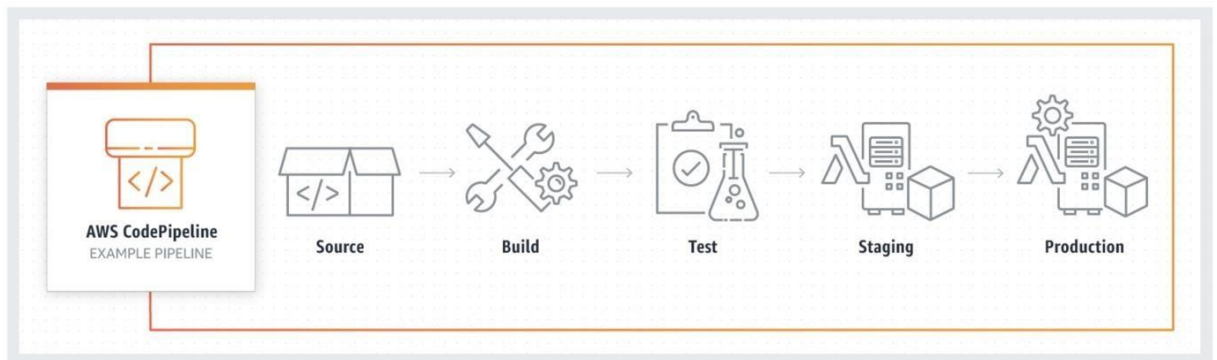


Fig 2: AWS CodePipeline

2.3 Conclusion

To sum it up, the frontend for the web was built using React. React Native was used to building the IOS and Android Apps. Across both platforms, we made use of several JavaScript packages and created scalable components to meet our design specs and ensured that the UI was responsive. Some of these components were made to clean up the code. This helps reduce the time required to make changes in existing features in the future, especially those reused across different sections.

User authentication was carried out using AWS Amplify. It employs AWS Cognito as its authentication provider. Their data is stored in a dynamo dB table, called dev_patient (for dev/testing) and prod_patient (for production), with the primary key being userId and recordTime as the sort key. Other indexes were introduced to view the table in a particular way and fetch desirable data. To access data in this table, APIs were designed and built using AWS SAM CLI that provides lambda-like execution. After locally testing them, they were deployed using CI/CD on AWS.

Alexa skill was developed separately on Alexa developers' consoles. It has not been integrated yet.

CHAPTER 3

METHODOLOGY

2.1 Introduction

In this chapter, we shall discuss section 2.3 in detail. Also, we followed agile Software Development during these six months with tasks or 'tickets' assigned on Trello and meetings were held via Zoom to build apps that adhere to the Service Layer Architecture. Both concepts shall be discussed, along with some features and components built to serve them.

2.2 Methodology

2.2.1 Agile Software Development

It involves self-organizing and cross-functional teams working together with their customers/end-users to identify requirements and create solutions (s). It promotes agile responses to change and adaptive preparation, evolutionary growth, early implementation, and continuous improvement. Let's have a look at some of the principles:

- Early and consistent delivery of valuable applications ensures customer loyalty.
- Even late in production, it's okay to accept changing requirements.
- Deliver working apps regularly (weeks rather than months)
- Working software is the most critical indicator of success.
- Sustainable development is characterized by the ability to sustain a steady rate of change.
- Consistent emphasis on technological performance and aesthetics
- Simplicity, or the art of minimizing the amount of work that isn't finished, is critical.
- Self-organizing teams provide the best architectures, specifications, and designs.
- The team considers how to become more successful regularly and adjusts as required.

To meet these principles, we had a standup call daily, sometimes on weekends too, at a time of our convenience. Here, we discuss what we had done on that day, followed by a code review and then I was assigned tasks for the following day. A part of my routine was to accommodate the suggestions that come up during the code review and then deliver on the assigned task.

3.2.1.1 Trello

We used it to manage our tasks. Trello uses a 'ticketing' concept to assign tasks to a specific member within a Trello board. This ticket consists of a short description of the job to be done, followed by a few images as hints and then tag that member to the ticket. Trello then sends an email to notify the member about their task.

2.2.2 *Service Layer Architecture*

A way to critique applications is the architecture of their codebase. There must be a clear distinction between the service layer, business logic, and finally, where we render components. It enhances code readability and helps in debugging.

In the service layer, we interact with our backend via APIs. The web application already had a network service layer, so my job required me to create one for mobile. The service layer is subdivided into Patients, Symptoms and Reports; basically, different endpoints have a separate division for themselves. Here, we make the right calls to interact with our backend. There is a separate file that holds all the common constants that are required by these three files.

With GET requests in the business logic layer, we manipulate data from the API and then send it to the UI layer to render it. With POST/PUT/PATCH/DELETE, we manipulate the information acquired from a user into the desired format that adheres to our database design and invokes the desired API.

To support this layered architecture, we created components or modules. In the next section, we shall dive into some of these components.

2.2.3 *Components*

They are functional units that perform a specific-repeatable task. They must be scalable. Scalable components are those that can accommodate changes in the future without breaking the existing code. Moreover, the parameters of a component must be intuitive and easy to grasp. Here are some examples.

2.2.3.1 *AlertComp*

As the name suggests, this component displays Alert Dialogs to a user on the mobile app. Alert Dialogs can be considered a way of conveying messages from the server (error or success). It has the following parameters:

- i. Title: This is the title of the alert dialogue and conveys the sole purpose of displaying an alert.
- ii. Message: This part describes the issue that triggered the alert or a task that a user must perform.
- iii. Type: This parameter classifies alerts into mainly four classes in our app – success, error, warning, verify.
- iv. isActionable: A boolean that indicates whether alert dialog must act.
- v. Action: It is the name of a task that is displayed.
- vi. Callback: This is the function that is triggered when an action is selected on an alert.
- vii. onDismiss: Function performed upon dismissing an alert.

2.2.3.2 *SymptomCard*

This component displays a symptom card. In general, it shows details such as name, severity, symptom location (image) and onset date. Symptoms are displayed all over our app and can be categorized as follows: review, incomplete, non-chief and general. They take their definitions from their names themselves. The following showSymptomImage is a boolean to display symptom location or not, which requires a gender parameter to show a female or male body.

- i. Item: holds the symptom object.
- ii. isReview: a boolean to categorize a symptom as review; defaults to false.
- iii. showReviewOptions: a function passed from a parent class that adds a review symptom to a reviewProcessList that sets toggleReview to true for this item, indicating that a user can now see various options to review this symptom
- iv. showSymptomImage: a boolean to display symptom location or not, and this requires a gender parameter to portray a female or male body

2.2.3.3 Feedbacks

This component was built on the website to take feedbacks from providers. Feedbacks are associated with reports, patients and can be general feedback too. Moreover, it can be scaled to take feedbacks from patients, which we will discuss in the next section. It is a button that, when tapped, displays a dialog containing a set of questions that seek a provider's opinion on the overall experience of using the software. Coming to its parameters:

- i. Style: This aligns with the feedback button.
- ii. AssociatedWith: This is an object that contains two keys, entity and uniqueIdentifier. The entity can be reports or patients, and uniqueIdentifier holds the recordTime for report or userId, respectively.

2.2.4 Designing, Building and Deploying APIs

In this section, we shall discuss building new features in an existing application from scratch. The Feedbacks component discussed in the previous section will serve as the primary example. The other two examples, Symptom Creation and Storing Review History indicate the convenience of Dynamo Db.

2.2.4.1 Feedbacks

Table 1: Feedbacks Database Design. 'scenario' isn't a parameter.

userID	recordTime	associatedWith	effectiveness	postVisitDocumentation	scenario
123	PROVIDERFEEDBACK#TIME1	REPORT#time	5	5	feedback for one report
1234	PROVIDERFEEDBACK#TIME2	PATIENT#userId	5	5	feedback for one patient

The first step should be modelling the table (Table 1). Modelling gives us a clear idea about the parameters that will be needed. As mentioned earlier, the primary key for our table is userID and the sort key is record time. #TIME differentiates one feedback from the other by the same provider. It is the time when a user invoked the API associated with this feature. We discussed other parameters in the previous section. It was also mentioned earlier how we could scale this same design to patients. For patients, the

recordTime will simply begin with 'PATIENTFEEDBACK'. Using constant is a straightforward and convenient way of uniquely identifying records in a Dynamo Db.

After this, endpoints should be designed. The way employees previously set up APIs, /providers/feedbacks seemed to be the aptest endpoint. Now, let's have a look at the entire API with various methods and request-response bodies.

Endpoint: /provider/feedbacks

Method: POST

Body Params:

	Attribute:	Format
{		
	associatedWith:	REPORT#TIMESTAMP,
	effectiveness:	Numeric [0,5],
	postVisitDocumentation:	Numeric [0,5]
}		

Response Body

→ Success:

```
{
  statusCode: 201,
  message: "Thank you for your valuable feedback."
}
```

→ Error: This is returned when the attributes do not adhere to their formats.

```
{
  statusCode: 400,
  message: "Sorry, your feedback was not submitted. Please try
again later."
}
```

Method: GET

Response Body:

→ Success:

```
{
  statusCode: 200,
  data: List of feedback objects associated with this provider,
  message: "Get feedback."
}
```

→ Error:

```
{
  statusCode: 400,
  message: "Failed to load feedback from this provider."
}
```

2.2.4.2 Symptom Creation

Before we discuss the implementation details, let's discuss the symptom creation flow. To create a new symptom, a user must answer a set of 12 questions about the symptom. Now, answering twelve questions in one sitting can be tedious. So, we've provided users with an option to 'Save & Quit' after answering eight questions and is categorized as an incomplete symptom. Thus, a minimum of eight parameters is required to create a symptom resource in our table. `completionTime` differentiates between these two symptoms. It's 0 for complete symptoms, and the timestamp of symptom creation is stored for incomplete symptoms.

Considering the frontend, clicking two buttons, 'Save & Quit' and a button at Symptom Summary (displayed after answering all questions), should create a new resource, i.e., symptom in our table. To tackle this situation at the backend, we initially made two API calls; POST requested to create an incomplete symptom with eight parameters (answers to first 8), followed by a PATCH request to complete this symptom. In simple terms, to create a new symptom, first, an incomplete symptom was created and then updated immediately. However, this solution isn't cost-effective.

Ideally, to create one resource, we should use only one API. For a cost-effective solution, we utilized the `completionTime` attribute and passed all the parameters. In case of incomplete symptoms, we represented the unanswered questions with default values in our table.

3.2.4.3 Storing Review History

In the last step of symptom creation, users can choose to review this symptom shortly, or providers resolved them. They may check it within a day, a week, a month or a few months later. The main reason for reviews is to track how this symptom has affected this user over time. For now, we prompt users only to update their severity. Now, the objective was to reflect these reviews on our table. Each review should be associated with only one main symptom, and that should be understood when we look at the table. Table 2 depicts it. Rt1, rt2 and rt3 are timestamps in ascending order. The main symptom and latest severity store the newest review.

Table 2: Review History Database Design

userId	recordTime	severity	initialSeverity
asd	SYMPTOM#RTS	7	10
asd	SYMPTOM#RTS#REVIEW#rt1	5	
asd	SYMPTOM#RTS#REVIEW#rt2	6	
asd	SYMPTOM#RTS#REVIEW#rt3	7	

3.3 Conclusions

Once the designs were ready and the code was pushed to the master branch, AWS CodePipeline took over from here and deployed the changes to production. Thus, making AWS Codepipeline are a convenient tool for developers.

In the examples discussed in the previous section, we've paid more attention to designing our features instead of implementing them or simply writing the actual code mainly because implementation is a straightforward translation-to-code task once viable solutions are discovered. Moreover, Dynamo Db tables are easy to use because of their key-value based nature.

CHAPTER 4

RESULT ANALYSIS

4.1 Introduction

This section shall look into the different stacks of our mobile app (Symptom, Report and Profile, Setup Flow). The main focus is on time taken to finish tasks on each of these sections. I will represent the results in the form of a 'time taken vs app versions' graph.

We made monthly versions.

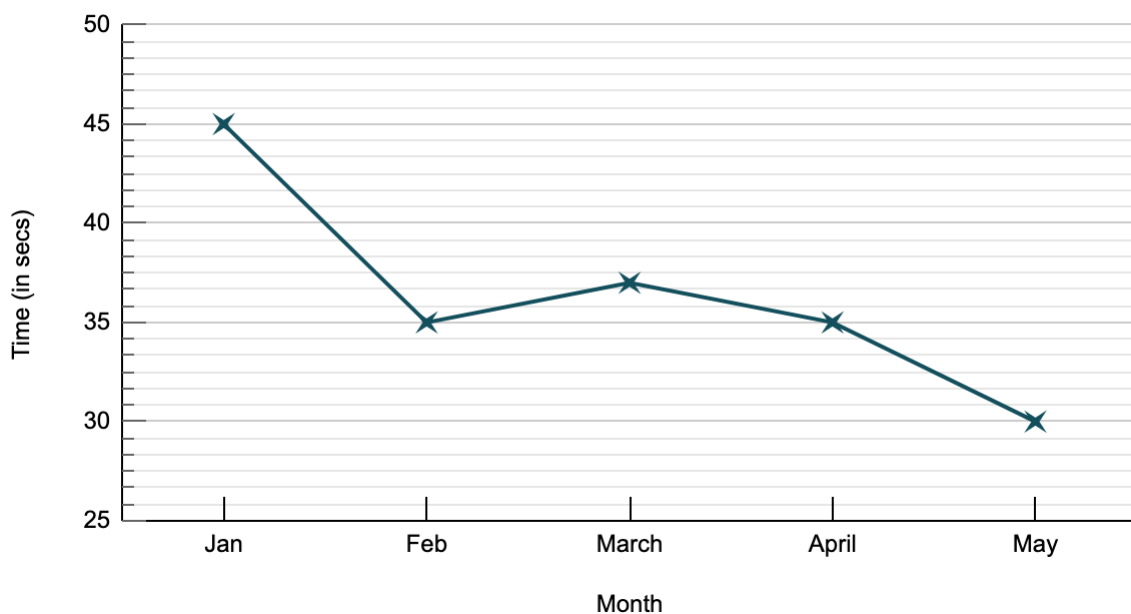
4.2 Result Analysis

4.2.1 Reports

Table 3: Report creation time over five months

Month	Time (in secs)
Jan	45
Feb	35
March	37
April	35
May	30

Report Creation Time vs App Versions



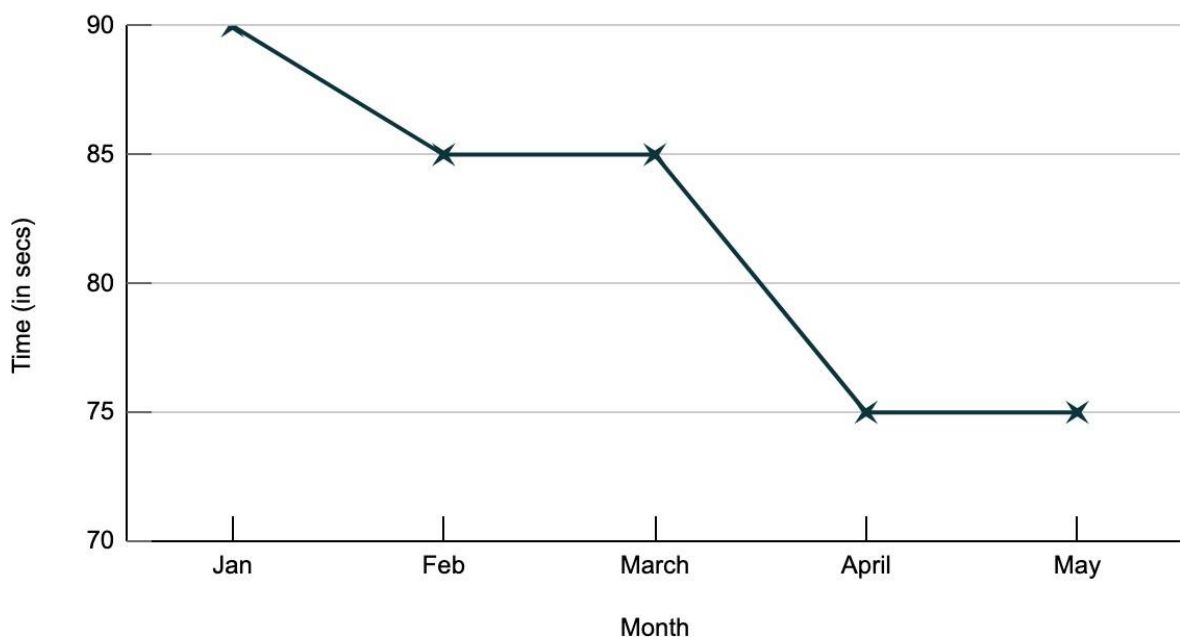
Once the first version was ready and tested, creating a report took around 45 seconds. However, over the next month, we managed to bring it down to 35 seconds. We achieved this by loading the no. of symptoms within a period; for five different periods in parallel. Then it remained the same by the end of April, but there were new functionalities added, such as storing images that represented the chief symptoms of the report. These images were stored in our S3 buckets right after a user uploaded them. Storing these images at this stage instead of doing it while storing the report itself accounted for the extra 5 seconds compared to the time in May.

4.2.2 Symptoms

Table 4: Symptom creation time over five months

Month	Time (in secs)
Jan	90
Feb	85
March	85
April	75
May	75

Symptom creation time vs App Versions



At first glance, we can observe that the time taken to create a symptom is much longer than the time taken to generate a report that consists of these symptoms. This is mainly because symptom creation is a 12-step long process, whereas it takes only three steps to create a report. As we reached April, we observed a significant drop in the symptom creation time because we dropped redundant APIs that were involved here. One of them has been discussed in Section 2.2.4.2, and the other being Load Common Symptoms API. As the name suggests, it contained all the commonly occurring symptoms in our table. Upon further examination, we realized that these common symptoms will be specific to those living in similar conditions and would not be nationwide.

4.3 Conclusions

There were no predefined time limits for the functionalities discussed in this chapter, and the goal was to ensure that they were intuitive, easy-going and non-tedious for users [11]. By the end of May, we arrived at a satisfying version of our app.

CHAPTER 5

CONCLUSION AND FUTURE SCOPE

5.1 Summary

The main objective of this project was to develop a chronic disease-specific symptom tracking service that enables patients to use Amazon Alexa to voice report their disease-related symptoms or simply log their symptoms on their phones/websites. While the Alexa skill has not been integrated with the app, other main functionalities of the app have been tested and deployed to production.

As mentioned earlier, we followed the Agile software development paradigm. Meetings were held on zoom where we would discuss new features, have code reviews and, when needed, resolve bugs.

5.2 Conclusions

We launched the app on Google Play Store in early April. However, it is still under review for the Apple Play store. The website has been launched in beta and can now be used by both users and doctors.

5.3 Future Scope of Work

Recollect section 3.2.4.3, where we had discussed the need for stored review records and how we could implement it on the backend. Currently, these different reviews are not used at all. However, we could use these values to display a 'severity vs time' graph. An API has to be designed to fetch data with recordTime that begins with SYMPTOM#Timestamp to implement this. The endpoint in this case is `base_url/symptoms/{timestamp}`. At the server, this API will first retrieve all entries in the table corresponding to that symptom. Then, we can return a list of objects in the following format:

```
[ {'severity' : value, 'date' : dateFormat}],
```

where dateFormat is a string in YY-MM-DD format, obtained from recordTime of each symptom. On the frontend (web or mobile), an icon button, preferably in the shape of a graph, can be added to the symptom card. This button can trigger a dialog or take the user to a new screen on mobile and render the desired graph.

In section 3.2.4.1, we discussed the Feedbacks feature implemented for providers on the website only in detail and mentioned that patient feedback could be represented in the table by

storing recordTime in PATIENTFEEDBACK#Timestamp format. The endpoint, in this case, is base_url/users/ feedbacks. For mobile, we can display dialogs after completing symptom creation flow and report creation flow to pursue users and gain their valuable feedback on these key components of our app. We can ask for generic feedback within the Profile stack itself.

After gaining some traction, the next step is to integrate this software with EHR. An electronic health record (EHR) is a digital version of a patient's paper chart. EHRs are real-time, patient-centred records that make information available instantly and securely to authorized users. It was built to go beyond standard clinical data collected in a provider's office and include a broader view of a patient's care. One of the most key traits of an EHR is that authenticated physicians can create and manage health information in a digital format that we can share with other providers across several health care organizations. We can utilize the redox engine to integrate this software with EHR.

REFERENCES

Reference

[11] I. Van de Poel and L. Royakkers, Ethics, technology, and engineering: An introduction. John Wiley & Sons, 2011

Web

[1] "Supporting Older Patients with Chronic Conditions", *National Institute on Aging*, 2021. [Online]. Available: [Website 1](#).

[2] "Unnecessary ED visits from chronically ill patients cost \$8.3 billion", *Modern Healthcare*, 2021. [Online]. Available: [Website 2](#).

[3] "Build Mobile & Web Apps Fast | AWS Amplify | Amazon Web Services", *Amazon Web Services, Inc.* [Online]. Available: [Website 5](#).

[4] "React Native · Learn once, write anywhere", *Reactnative.dev*. [Online]. Available: [Website 7](#).

[5] "AWS Lambda – Serverless Compute - Amazon Web Services", *Amazon Web Services, Inc.* [Online]. Available: [Website 3](#).

[6] "Build Mobile & Web Apps Fast | AWS Amplify | Amazon Web Services", *Amazon Web Services, Inc.* [Online]. Available: [Website 4](#).

[7] "Amazon DynamoDB | NoSQL Key-Value Database | Amazon Web Services", *Amazon Web Services, Inc.* [Online]. Available: [Website 6](#).

[8] "Amazon Cognito - Simple and Secure User Sign Up & Sign In | Amazon Web Services (AWS)", *Amazon Web Services, Inc.* [Online]. Available: [Website 8](#).

[9] "Create Alexa Skills Kit | Amazon Alexa Voice Development", *Amazon (Alexa)*. [Online]. Available: [Website 9](#).

[10]"CI/CD on AWS - Continuous Integration and Continuous Delivery for 5G Networks on AWS", *Docs.aws.amazon.com*. [Online]. Available: [Website 10](#).

Annexures

Code Snippets have been included for components discussed in this report.

```
const AlertComp = ({title, message, type,isActionable, callback, action, onDismiss}) => {
  var jsonPath;
  switch (type) {
    case "error":
      jsonPath = require("../assets/error.json");
      break;
    case "success":
      jsonPath = require("../assets/success-tick.json");
      break;
    case "verify":
      jsonPath = require("../assets/email-sent.json");
      break;
    case "warning":
      jsonPath = require("../assets/warning.json");
    default:
      console.log("no such lottie file exists in assets");
  }
  return (
    <Modal
      animationType="slide"
      transparent={true}
      visible={true}
      style={{ backgroundColor: "rgba(0,0,0,0.5)" }}
    >
      <View style={styles.view}>
        <View style={styles.modal}>
          <Text style={styles.modalText}> {title} </Text>
          <View
            style={{
              width: "100%",
              height: 0.5,
              backgroundColor: "gray",
              marginVertical: 15,
            }}
          > </View>
          {!type ? null : (
            <View style={{ width: "50%", height: 50 }}>
              <LottieView
                source={jsonPath}
                autoPlay
                backgroundColor="white"
              > </LottieView>
            </View>
          )}
          <Text style={styles.text}> {message} </Text>
        </View>
      </View>
    >
  )
}
```

```

{isActionable ? (
  <View style={styles.buttonRow}>
    <TouchableHighlight
      style={{ ...styles.buttons, backgroundColor: "white" }}
      onPress={() => {
        callback();
      }}
    >
      <Text style={styles.delete}> {action} </Text>
    </TouchableHighlight>
    <TouchableHighlight
      style={{ ...styles.buttons, backgroundColor: "#F4992C" }}
      onPress={() => {
        onDismiss();
      }}
    >
      <Text style={styles.ok}> Cancel </Text>
    </TouchableHighlight>
  </View>
) : (
  <TouchableHighlight
    style={{ ...styles.button, backgroundColor: "#F4992C" }}
    onPress={() => {
      onDismiss();
    }}
  >
    <Text style={styles.ok}> Okay </Text>
  </TouchableHighlight>
))
</View>
</Modal>
);
}

const styles = StyleSheet.create({
  view: {
    justifyContent: "center",
    alignItems: "center",
    alignContent: "center",
    flex: 1,
    backgroundColor: "rgba(0,0,0,0.7)",
  },
  modal: {
    width: "80%",
    margin: 10,
    backgroundColor: "white",
    borderRadius: 10,
    padding: "5%",
    alignItems: "center",
    shadowColor: "black",
    shadowOffset: {
      width: 0,
      height: 2,

```



```

    },
    shadowOpacity: 0.25,
    shadowRadius: 3.85,
    elevation: 5,
    zIndex: 5,
  },
  button: {
    backgroundColor: "#B0B0B0",
    borderRadius: 10,
    padding: 10,
    elevation: 2,
    width: "80%",
  },
  buttons: {
    backgroundColor: "#B0B0B0",
    borderRadius: 10,
    padding: 10,
    elevation: 2,
    width: "35%",
  },
  text: {
    color: "black",
    textAlign: "center",
    fontSize: 20,
    marginTop: 20,
  },
  ok: {
    color: "white",
    textAlign: "center",
    fontSize: 20,
    padding: 10,
  },
  delete: {
    color: "red",
    textAlign: "center",
    fontSize: 20,
    padding: 10,
  },
  modalText: {
    textAlign: "center",
    fontWeight: "bold",
    fontSize: 34,
    shadowColor: "black",
    shadowOffset: {
      width: 0,
      height: 2,
    },
    shadowOpacity: 0.3,
    shadowRadius: 3.8,
    elevation: 5,
  },
  buttonRow: {
    flexDirection: "row",

```

```

        justifyContent: "space-evenly",
        width: "100%",
    },
});

export default AlertComp;

```

```

const { width } = Dimensions.get("window");
const getSymptomImage = (gender, symptomLocation) => {
    let uri =
        "https://myhealthtoday-body-parts.s3.us-west-2.amazonaws.com/public/" +
        gender.toString().toLowerCase() +
        "_mob_" +
        symptomLocation.toString().toLowerCase() +
        ".png";
    return uri;
};

const onBetterPress = (item, navigator) => {
    navigator.push("Better", {
        symptomNumber: item.recordTime.split("#")[1],
        conditionType: "better",
        severity: item.severity,
        symptom: item
    });
};

const onWorsePress = (item, navigator) => {
    navigator.push("Better", {
        symptomNumber: item.recordTime.split("#")[1],
        conditionType: "worse",
        severity: item.severity,
        symptom: item
    });
};

const noChangeInReview = async (item, navigator) => {
    navigator.navigate("Review", {
        reviewProcess: true,
        severity: item.severity,
        symptomNumber: item.recordTime.split("#")[1],
        symptom: item
    });
};

const onResolvedPress = (item, navigator) => {
    navigator.push("ResolvedReview", {
        symptomNumber: item.recordTime.split("#")[1],
        symptom: item
    });
};

```

```

const SymptomCard = ({item,isReview = false, showReviewOptions, toggleReview = false, showSymtomImage = true,
navigator, gender, isIncompleteSymptom = false}) => {

return (
  <Card
    style={styles.card}
    onPress={() => {
      if(isIncompleteSymptom)
      {
        navigator.push("FeelBetter", {
          incompleteSymptomNumber: item.recordTime.split("#")[1],
          severity: item.severity,
          symptomName: item.symptomName,
          symptomLocation: item.symptomLocation,
          symptomLocationDescription: item.symptomLocationDescription,
          startDate: item.startDate,
          symptomConsistency: item.symptomConsistency,
          symptomDescription: item.symptomDescription.toString(),
          symptomMovement: item.symptomMovement,
          frequency: item.frequency,
        });
      }
      else {
        navigator.push("SymptomSummary",item);
      }
    }}
  >
    {showSymtomImage ? (
      <Card.Cover
        source={{
          uri: item.symptomLocation ? getSymptomImage(gender, item.symptomLocation) : "https://myhealthtoday-body-parts.s3.us-west-2.amazonaws.com/public/" +
            gender.toString().toLowerCase() +
            "_mob_na.png",
        }}
        style={{
          borderTopRightRadius: moderateScale(10),
          borderTopLeftRadius: moderateScale(10),
          overflow: "hidden",
        }}
      />
    ) : null}

    {toggleReview ? (
      <View>
        <View
          style={{
            flexDirection: "row",
            justifyContent: "space-between",
            flex: 1,
          }}
        >
          <Text style={([styles.cardHeading, { marginLeft: 15 }])}>

```

```

    {item.symptomName}
  </Text>
  <View
    style={{ flexDirection: "row", marginTop: verticalScale(25) }}
  >
    <Text>Severity {item.severity}</Text>
    <Rating style={{ marginHorizontal: 15 }} rating={item.severity} />
  </View>
</View>
<View>
  <Divider width="100%" style={{ marginTop: "5%" }} />
  <Text style={styles.cardActionHeader}>
    How has your symptom changed?
  </Text>
  <Card.Actions style={styles.row}>
    <Button
      style={styles.conditionButton}
      color="#505050"
      mode="outlined"
      uppercase={false}
      labelStyle={styles.conditionlabel}
      onPress={() => onBetterPress(item, navigator)}
    >
      Improved
    </Button>
    <Button
      style={styles.conditionButton}
      color="#505050"
      mode="outlined"
      uppercase={false}
      labelStyle={styles.conditionlabel}
      onPress={() => onWorsePress(item, navigator)}
    >
      Worse
    </Button>
  </Card.Actions>
  <Card.Actions style={styles.row}>
    <Button
      style={styles.conditionButton}
      color="#505050"
      mode="outlined"
      uppercase={false}
      labelStyle={styles.conditionlabel}
      onPress={() => noChangeInReview(item, navigator)}
    >
      Same
    </Button>
    <Button
      style={styles.conditionButton}
      color="#505050"
      mode="outlined"
      uppercase={false}
      labelStyle={styles.conditionlabel}

```

```

        onPress={() => onResolvedPress(item, navigator)}
      >
        Resolved
      </Button>
    </Card.Actions>
  </View>
</View>
): (
  <View>
    <View style={{ marginLeft: 15, paddingBottom: 23 }}>
      <Text style={styles.cardHeading}>{item.symptomName}</Text>
      <View style={{ flexDirection: "row" }}>
        <Text>Severity {item.severity}</Text>
        <Rating style={{ marginLeft: 10 }} rating={item.severity} />
      </View>
      <Text style={{ marginVertical: 8, fontSize: moderateScale(16) }}>
        Started on {DateHelper.formatDateShortMonth(item.startDate)}
      </Text>
    </View>
    {isReview ? (
      <View style={{ flex: 1, }}>
        <Divider width="100%" />
        <TouchableOpacity
          style={{ alignSelf: "center", paddingVertical: "6%" }}
          onPress={() => showReviewOptions()}
        >
          <Text
            style={{
              color: "#F4892C",
              alignSelf: "center",
              fontWeight: "600",
            }}
          >
            Click here to review
          </Text>
        </TouchableOpacity>
      </View>
    ) : null}
  </View>
)}
</Card>
);
}

const styles = StyleSheet.create({
  card: {
    width: width - scale(30),
    borderRadius: moderateScale(10),
    marginVertical: verticalScale(15),
    shadowColor: "#000",
    shadowOffset: {
      width: 0,
      height: 4,

```

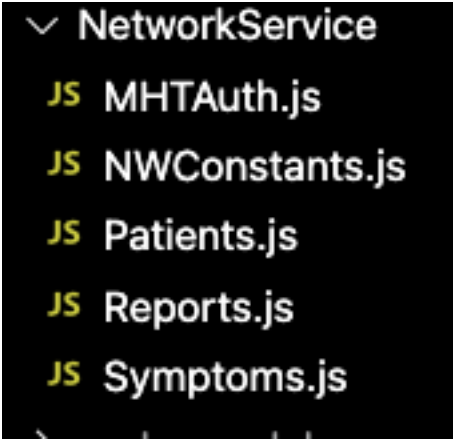
```

    },
    shadowOpacity: 0.32,
    shadowRadius: 5.46,
    elevation: 5,
  },
  cardHeading: {
    fontSize: moderateScale(20),
    fontWeight: "600",
    marginTop: verticalScale(20),
    textTransform: 'capitalize'
  },
  link: {
    color: "blue",
    marginVertical: 5,
  },
  conditionButton: {
    marginHorizontal: scale(10),
    borderRadius: 20,
    width: 130,
  },
  reviewButton: {
    //marginLeft: 10,
    paddingHorizontal: 10,
  },
  searchText: {
    color: "white",
    fontSize: 20,
    //fontFamily: "sans-serif",
    paddingHorizontal: 10,
  },
  row: {
    flexDirection: "row",
    alignItems: "center",
    justifyContent: "center",
    marginVertical: verticalScale(5),
  },
  conditionlabel: {
    fontSize: 18,
  },
  cardActionHeader: {
    fontSize: 14,
    color: "#505050",
    fontWeight: "600",
    alignSelf: "center",
    marginVertical: '5%',
  },
  text:{
    alignSelf:'center'
  },
});

export default SymptomCard

```

Network module created for mobile.



✓ NetworkService

- JS MHTAuth.js
- JS NWConstants.js
- JS Patients.js
- JS Reports.js
- JS Symptoms.js

NetworkService

Plagiarism Report

9	Student Paper	1 %
10	reactnative.dev Internet Source	1 %
11	Submitted to Griffith College Dublin Student Paper	1 %
12	Submitted to University of Bolton Student Paper	1 %
13	developer.amazon.com Internet Source	<1 %
14	www.coursehero.com Internet Source	<1 %
15	aws.amazon.com Internet Source	<1 %
16	hdl.handle.net Internet Source	<1 %
17	Submitted to Aspen University Student Paper	<1 %
18	peer.asee.org Internet Source	<1 %
19	Submitted to University of Kelaniya Student Paper	<1 %
20	Submitted to University of West London Student Paper	<1 %

21	www.bootdey.com Internet Source	<1 %
22	Submitted to Technological Institute of the Philippines Student Paper	<1 %
23	krishikosh.egranth.ac.in Internet Source	<1 %

Exclude quotes On

Exclude matches < 10 words

Exclude bibliography On

PROJECT DETAILS

<i>Student Details</i>			
Student Name	Abhijeet Sahdev		
Register Number	170905316	Section / Roll No	A / 42
Email Address	sahdevjeet@gmail.com	Phone No (M)	+91 9741323117
<i>Project Details</i>			
Project Title			
Project Duration	January 2021- June 2021	Date of reporting	14/12/2020
<i>Organization Details</i>			
Organization Name	MyHealthToday LLC		
Full postal address with pin code	1105 E Katella Ave Unit 366 Anaheim CA, USA - 92805		
Website address	https://myhealthtoday.care		
<i>External Guide Details</i>			
Name of the Guide	Parth Srivastav		
Designation	Chief Technology Officer		
Full contact address with pin code	218 Rockview, Irvine, CA, USA - 92612		
Email address	Parths1@uci.edu	Phone No (M)	+1 949 774 9777
<i>Internal Guide Details</i>			
Faculty Name	Dr Geetha M		
Full contact address with pin code	Dept of Computer Science & Engg, Manipal Institute of Technology, Manipal – 576 104 (Karnataka State), INDIA		
Email address	geetha.maiya@manipal.edu		

