CONTENT

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCES,

PILANI HYDERABAD CAMPUS

DEPARTMENT OF COMPUTER SCIENCE

PROJECT REPORT ON

Clinical Decision Support System for Diagnosis of Allergic Rhinitis

By- MEHUL JAIN

ID No: 2019A3PS1315H

Under the Guidance of

DR. JABEZ CHRISTOPHER

CSIS

Hyderabad, India

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my parents for providing me with the resources to study at such an esteemed institution and also for their constant support and motivation. I would like to thank Dr Jabez J Christopher sir and Ramisetty Kavya Ma’am for letting me work on this project and for their continuous guidance and support.

Department of Computer Science

BIRLA INSTITUTE OF TECHNOLOGY

AND SCIENCE, PILANI

CERTIFICATE

This is to certify that the project report entitled "Clinical Decision Support System for Diagnosis of Allergic Rhinitis" submitted by Mr Mehul Jain (ID No. 2019A3PS1315H) in fulfilment of the requirements of the course CS F366, Lab Oriented Project Course, embodies the work done by him under my supervision and guidance.

Dr. Jabez Christopher
Department of CSE
BITS Pilani, Hyderabad Campus

ABSTRACT

The primary objective of the project is developing an easy to use, portable and efficient platform for patients to fill up their allergens and symptoms. The first section of the report explains the problem statement and the objectives to build a successful application. The second section describes the architecture of our full stack application with more details about the backend and the frontend design, implementation and documentation for the same. The third section explains the improvements that can be made to our application in the near future to make it more efficient and feature rich.

Chapter 1

1. INTRODUCTION

1.1 Problem Statement

To design and implement a user-friendly application which can take in input from the patient and give back the results based on the input from the users which can help in the diagnostics of the same in a better way.

1.2 Objectives

• Build a Back-end Server with NodeJS to act as the main API service.

• Build a Flask Server as a Microservice to help us with the data processing of symptoms and allergens.

• Build the Front-end application in React Native to give cross platform support over Android and iOS.

• Logic implementation to get probability of multi-class classification.

Chapter 2

Design Of The Application

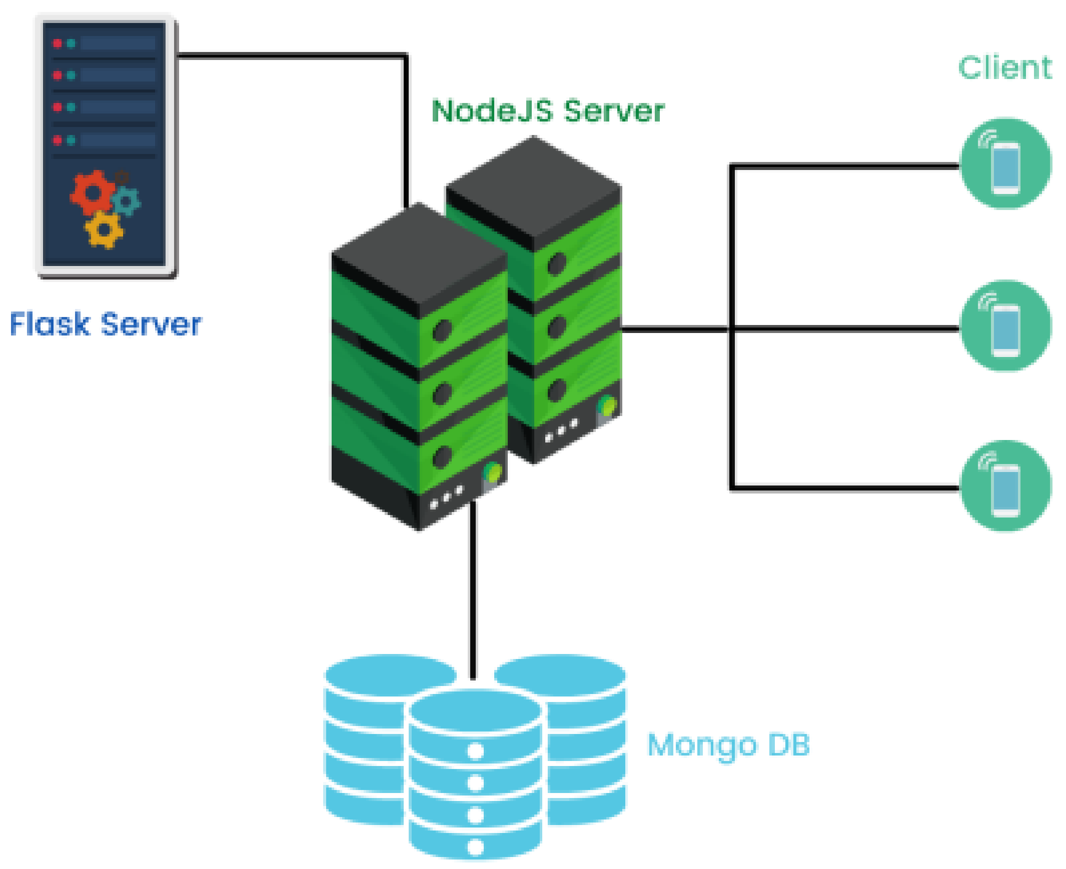
The application has been designed keeping in mind the processing requirements as well as the users preferences to deliver the best experience.

2.1 Tech Stack

The tech stack used for the application has the following components

- Front-End
  - React Native
  - Expo
- Back-End
  - NodeJS
  - Express
  - Flask
  - MongoDB

2.2 Architecture



The application architecture has 4 main components:

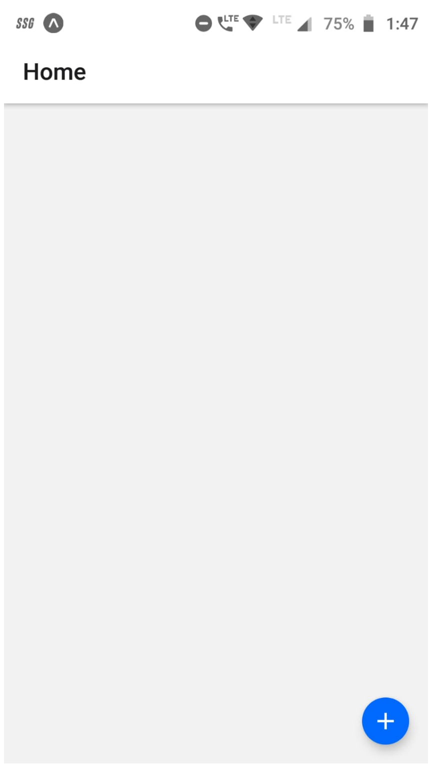
- NodeJS server - Main Application Server
- Flask Server - Data Processing Server
- MongoDB database - Database hosted on MongoDb Atlas
- React-Native Client Application

The Client application built using React-Native communicates with the API via the NodeJS server. This NodeJS server is further connected to the Python Flask server and the MongoDB database hosted on MongoDB Atlas. So, the NodejS server listens to incoming API requests from the client application and the Python Flask Server listens to incoming requests from this NodeJS server. All database queries are made through the NodeJS server. When the NodeJS server receives an API request from the client application the routing mechanism on the server decides what architecture component needs to be accessed next. If the incoming request requires database access, the server sends a query to the MongoDB database. If the request requires data processing, the NodeJS server sends a request to the Flask Server which responds with the required processed data. This response is then forwarded to the client as response to the original request made by the client.

The front-end of the application is built in React-Native using expo. Expo's built-in publish command was used to generate the final .apk files.

2.3 GUI Design

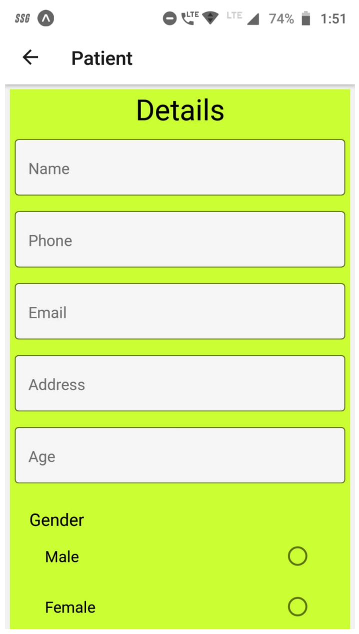
2.3.1 Home Page

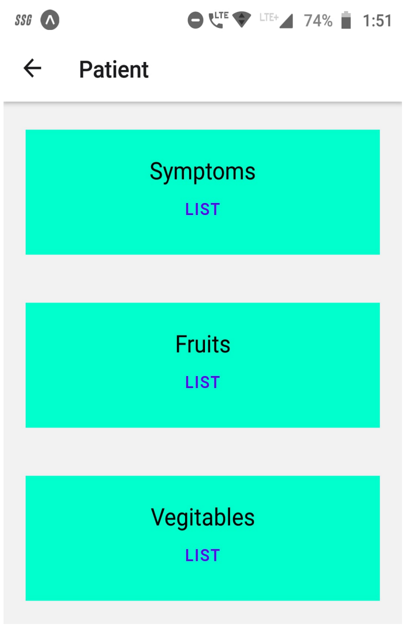


The home page is the first thing which comes when a user first installs the app. We are implementing the “Add Patient” feature here where the blue floating button with the plus sign is implemented.

2.3.2 Details Page

After the user clicks on the floating plus button, they are taken to this details page where they are given different fields to be filled which looks something like this.





The “LIST” button is used to collapse and uncollapse the list of fields to be filled in each category.

2.3.3 Filling in details of patients

1. The data type of “NAME” is String. The keyboard is a normal keyboard which is opened.

2. The data type of “PHONE” is Integer. The keyboard is a number keyboard which is opened.

3. The data type of “EMAIL” is String. The keyboard is a normal keyboard which is to be opened.

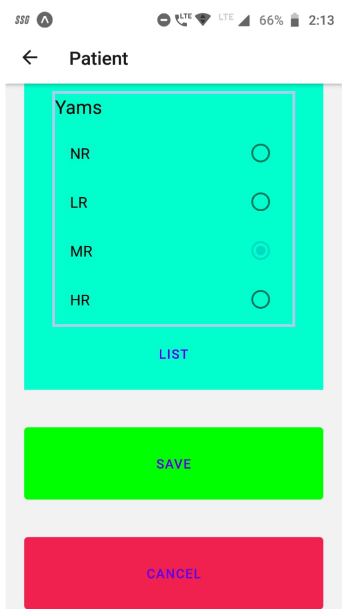
4. The data type of “ADDRESS” is String. The keyboard is a normal keyboard which is to be opened.

5. The data type of “AGE” is Integer. The keyboard is a number keyboard which is to be opened.

6. The data type of “AGE” is Integer. The keyboard is a normal keyboard which is to be opened.

7. The data type of all other fields which have an input type of radio button is boolean where False is if the radio button is not chosen and True if the radio button is chosen.

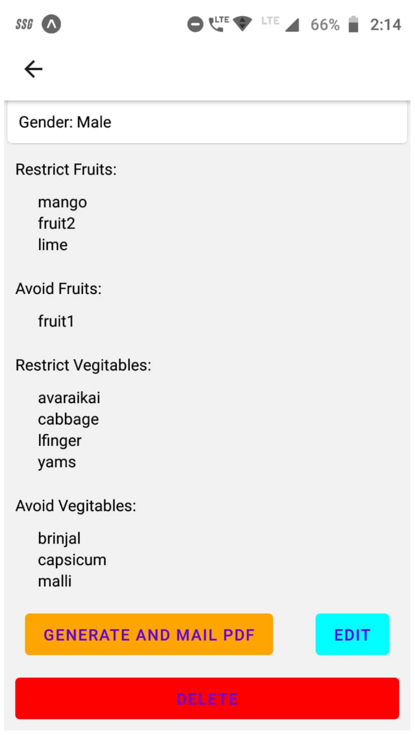
After the user fills in the information as follows, the user can save the data by clicking on the “SAVE” button.



The user can now click on the save button to add a patient.

On refreshing the screen, the user can now see the patient name which is a clickable card.

2.3.4 View Details


Gender: Male

Restrict Fruits:

mango
fruit2
lime

Avoid Fruits:

fruit1

Restrict Vegitables:

avaraikai
cabbage
lfinger
yams

Avoid Vegitables:

brinjal
capsicum
malli

On clicking the card on the home page for a particular patient, the details of the patient can be seen as above. Clicking on the email, the user can mail the patient. The user can also call the patient by clicking on the phone number section of the details page.

2.3.5 Edit Page

After the user clicks on the edit button, the user can change any values entered and update. The database at the backend is also updated. The entries previously filled by the patient are autofilled as it is retrieved and rendered from the database before the user can further update.

Chapter 3

Developer Documentation

3.1 Requirements

- NPM
- NodeJS
- Expo
- Python 3
- Flask
- React Native

The project is divided into 2 repositories,

- Backend
- Frontend

Refer to part-1 for the above.

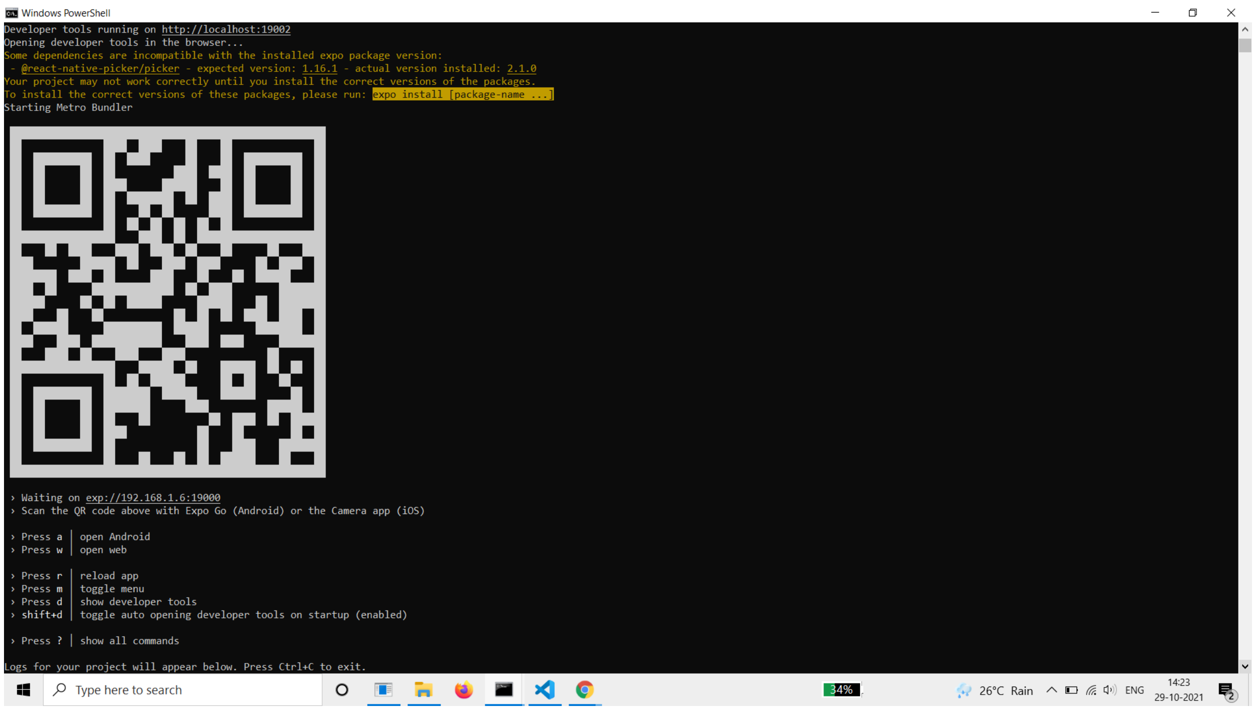
3.2 Setting up the Frontend

1. Install the dependencies using NPM by running the following command on your terminal

```
$ npm install
```

2. Start the expo server by running the following command on your terminal

   $ expo start



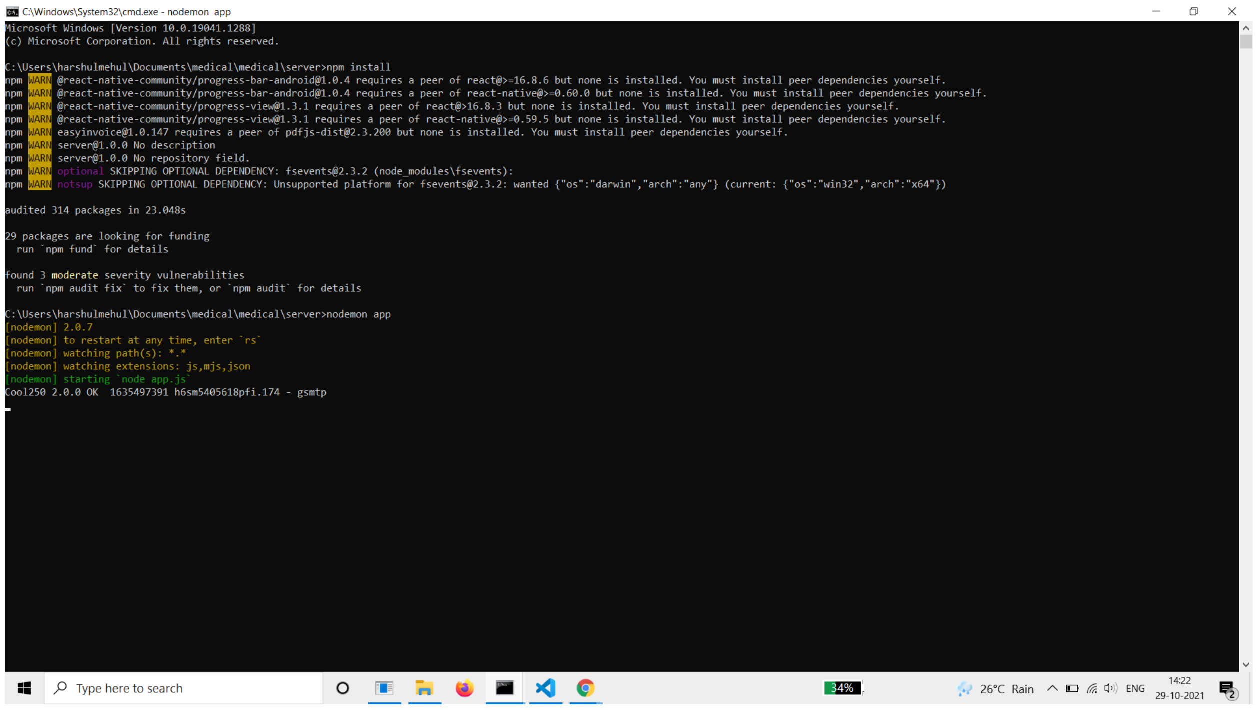
```
Developer tools running on http://localhost:19002
Opening developer tools in the browser...
Some dependencies are incompatible with the installed expo package version:
 - @react-native-picker/picker - expected version: 1.16.1 - actual version installed: 2.1.0
Your project may not work correctly until you install the correct versions of the packages.
To install the correct versions of these packages, please run: expo install [package-name ...]
Starting Metro Bundler

 › Waiting on exp://192.168.1.6:19000
 › Scan the QR code above with Expo Go (Android) or the Camera app (iOS)

 › Press a │ open Android
 › Press w │ open web

 › Press r │ reload app
 › Press m │ toggle menu
 › Press d │ show developer tools
 › shift+d │ toggle auto opening developer tools on startup (enabled)

 › Press ? │ show all commands

Logs for your project will appear below. Press Ctrl+C to exit.
```

3. Run the application on your mobile phone using the expo go app available on the play store. After downloading the application, launch the app and scan the qr code on your terminal from the app to run the front-end application on your phone. Scan the above QR code using any device having Expo-Go installed. Make sure that the system on which the React-Native and NodeJS is running and the scanning device (mostly a mobile phone scanner) are in the same internet connection (same hotspot or WiFi connection).

3.3 Setting up the Backend

1. Start the NodeJs server

   $ node server.js



```
C:\Windows\System32\cmd.exe - nodemon app
Microsoft Windows [Version 10.0.19041.1288]
(c) Microsoft Corporation. All rights reserved.

C:\Users\harshulmehul\Documents\medical\medical\server>npm install
npm WARN @react-native-community/progress-bar-android@1.0.4 requires a peer of react@>=16.8.6 but none is installed. You must install peer dependencies yourself.
npm WARN @react-native-community/progress-bar-android@1.0.4 requires a peer of react-native@>=0.60.0 but none is installed. You must install peer dependencies yourself.
npm WARN @react-native-community/progress-view@1.3.1 requires a peer of react@>16.8.3 but none is installed. You must install peer dependencies yourself.
npm WARN @react-native-community/progress-view@1.3.1 requires a peer of react-native@>=0.59.5 but none is installed. You must install peer dependencies yourself.
npm WARN easyinvoice@1.0.147 requires a peer of pdfjs-dist@2.3.200 but none is installed. You must install peer dependencies yourself.
npm WARN server@1.0.0 No description
npm WARN server@1.0.0 No repository field.
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@2.3.2 (node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@2.3.2: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})

audited 314 packages in 23.048s

29 packages are looking for funding
  run `npm fund` for details

found 3 moderate severity vulnerabilities
  run `npm audit fix` to fix them, or `npm audit` for details

C:\Users\harshulmehul\Documents\medical\medical\server>nodemon app
[nodemon] 2.0.7
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node app.js`
Cool250 2.0.0 OK  1635497391 h6sm5405618pfi.174 - gsmtp
```

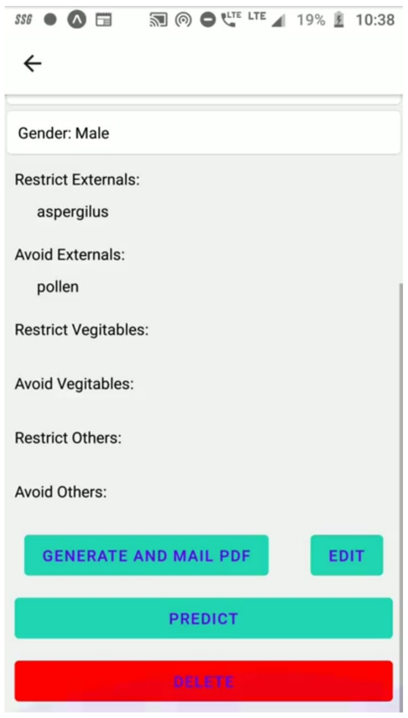
Note: Starting both the front-end and the back-end projects requires environment variables which are not pushed to the repositories for security reasons.

Chapter 4

Updates on the project and data processing

There are a few updates done on the previous version of the application.

4.1 Adding microservice



Gender: Male

Restrict Externals:
aspergilus

Avoid Externals:
pollen

Restrict Vegitables:

Avoid Vegitables:

Restrict Others:

Avoid Others:

Referring to app.py, the application has the functionality to predict the diseases and their combinatorial probabilities. Below is the output of the report generated when the “PREDICT” button is clicked.

```
"{'U': '0.8783280416722686', 'R': '0.11782557281320567', 'O': '0.003846385513901896', 'UR':
'6.196973044850788e-13', 'UO': '4.162664591312975e-15', 'RO': '2.0931213089555564e-18', 'N':
'6.551916190265631e-22'}"
```

4.1.1 The algorithm behind the multi-class classification

The BPA matrix has values for all symptoms and allergens which is used to find the combinations to get the disease diagnosed. The input, X is the user entered data and the data output is found using Dempster Shafer Theory's mass function. Bayes theorem is used as follows

$$P(X \mid Y) = \frac{P(Y|X) * P(X)}{P(Y)}$$

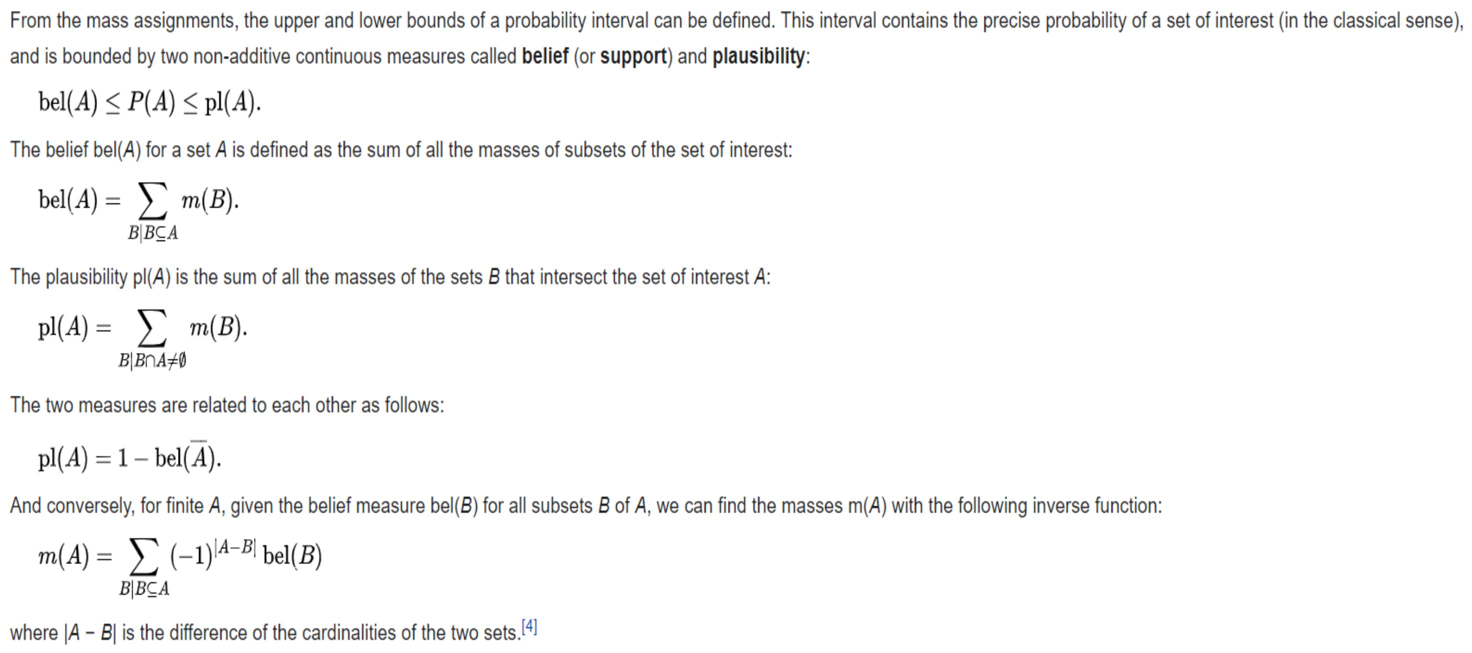
as $P(A \mid B) = \frac{P(A \cap B)}{P(B)}$ and for any independent event

$$P(A \cap B) = P(A) * P(B)$$

where P(A) is the probability of event A to occur and P(B) is the probability of

event B to occur. P(A ∩B) is the probability of both events, A and B to occur.

P(A | B) is the probability for the event A to occur given event B has already occurred. The Dempster Shafer Theory is about the following



From the mass assignments, the upper and lower bounds of a probability interval can be defined. This interval contains the precise probability of a set of interest (in the classical sense), and is bounded by two non-additive continuous measures called **belief** (or **support**) and **plausibility**:

$$\operatorname{bel}(A) \le P(A) \le \operatorname{pl}(A).$$

The belief bel(*A*) for a set *A* is defined as the sum of all the masses of subsets of the set of interest:

$$\operatorname{bel}(A) = \sum_{B \mid B \subseteq A} m(B).$$

The plausibility pl(*A*) is the sum of all the masses of the sets *B* that intersect the set of interest *A*:

$$\operatorname{pl}(A) = \sum_{B \mid B \cap A \neq \emptyset} m(B).$$

The two measures are related to each other as follows:

$$\operatorname{pl}(A) = 1 - \operatorname{bel}(\overline{A}).$$

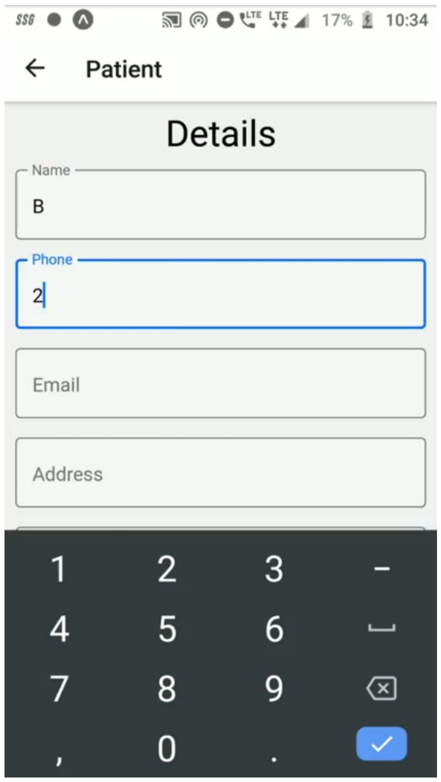
And conversely, for finite *A*, given the belief measure bel(*B*) for all subsets *B* of *A*, we can find the masses m(*A*) with the following inverse function:

$$m(A) = \sum_{B \mid B \subseteq A} (-1)^{|A-B|} \operatorname{bel}(B)$$

where |*A* − *B*| is the difference of the cardinalities of the two sets.[4]

4.2 Changing the User Interface

The changes are made to make it theme related rather than applying styles to every component of the application. Below is the theme applied to the application.



The “CLEAR CHOICE” button is also added to make changes like removing or updating choices.

4.3 Making changes in backend on MongoDB-Atlas

The application of CRUD (Create, Retrieve, Update, Delete) is used in the backend. These are reflected in the MongoDB-Atlas database or as the output.

CREATE:

The create action occurs when the user clicks on the “Plus” button on the home screen and on adding details, the patient details are added to the MongoDB database.

RETRIEVE:

The retrieve action occurs when the user clicks on the “Generate Input” button where the patient gets the PDF format of input details mailed to the respective entered email ID and the “Predict” button is when the patient gets the PDF format of the generated output from the microservice mailed to the respective email ID entered by the user.

UPDATE:

The update action occurs when the user clicks on the “Update” button when the user wants to edit the patient details by clicking on the “Edit” button.

DELETE:

On clicking the “DELETE” button the patient is removed from the database.

The complete code for the final project can be found here:

1. Project
2. Demo

The previous report can be found here:

1. Report-1

Chapter 5

Scope for improvement

There are some improvements which can be implemented in the future to make this application better and efficient. They are:

- Better data processing algorithms for live data.
- New model schema to store previous data
- Better and more attractive UI for better user experience.