

# CONTENT

## [ACKNOWLEDGEMENTS](#)

## [ABSTRACT](#)

### [Chapter 1](#)

#### [1. INTRODUCTION](#)

##### [1.1 Problem Statement](#)

##### [1.2 Objectives](#)

### [Chapter 2](#)

#### [Design Of The Application](#)

##### [2.1 Tech Stack](#)

##### [2.2 Architecture](#)

##### [2.3 GUI Design](#)

###### [2.3.1 Home Page](#)

###### [2.3.2 Details Page](#)

###### [2.3.3 Filling in details of patients](#)

###### [2.3.4 View Details](#)

###### [2.3.5 Edit Page](#)

### [Chapter 3](#)

#### [Developer Documentation](#)

##### [3.1 Requirements](#)

##### [3.2 Setting up the Frontend](#)

##### [3.3 Setting up the Backend](#)

### [Chapter 4](#)

#### [Updates on the project and data processing](#)

##### [4.1 Adding microservice](#)

###### [4.1.1 The algorithm behind the multi-class classification](#)

##### [4.2 Changing the User Interface](#)

##### [4.3 Making changes in backend on MongoDB-Atlas](#)

### [Chapter 5](#)

#### [Scope for improvement](#)



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

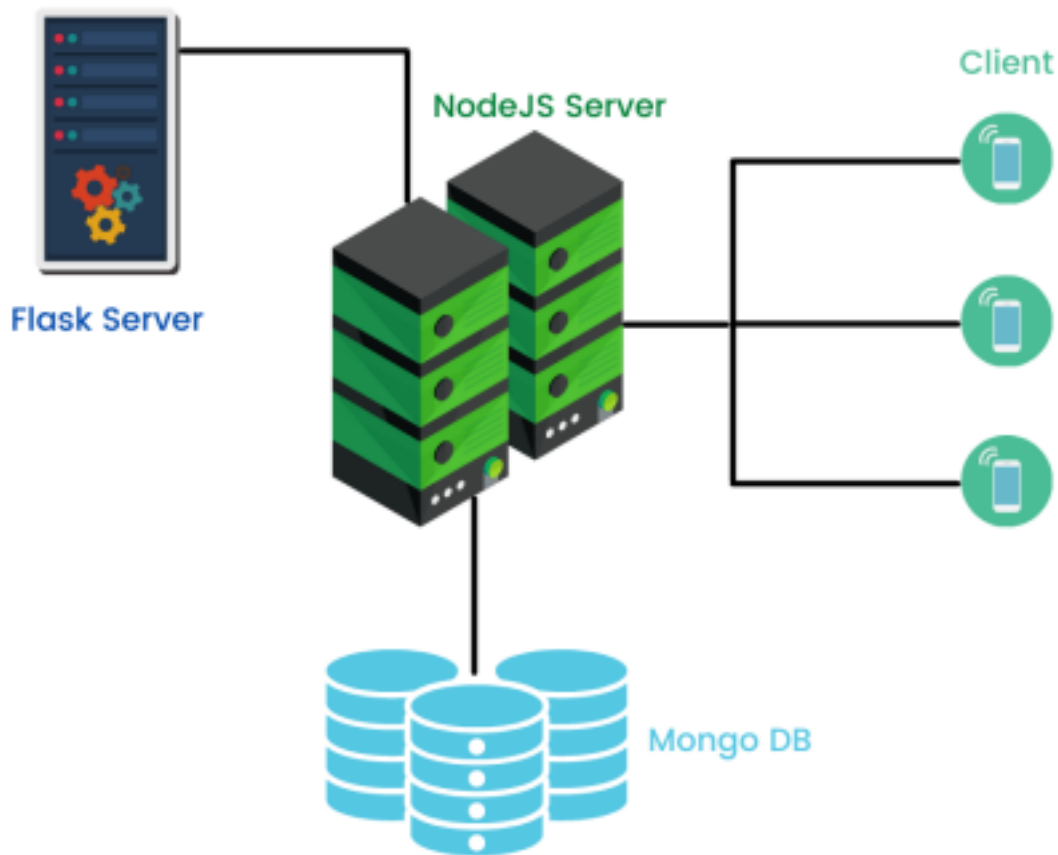
The application has been designed keeping in mind the processing requirements as well as the users preferences to deliver the best experience.

### 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

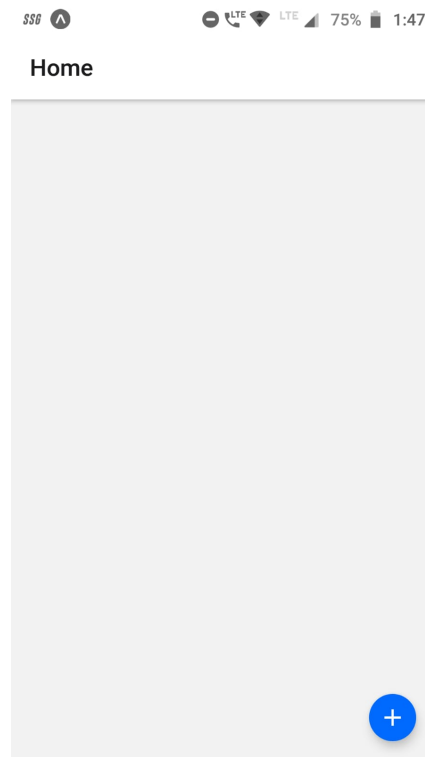


from the client application and the Python Flask Server listens to incoming requests from this NodeJS server. All database queries are made through the NodeJS server. When the NodeJS server receives an API request from the client application the routing mechanism on the server decides what architecture 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.

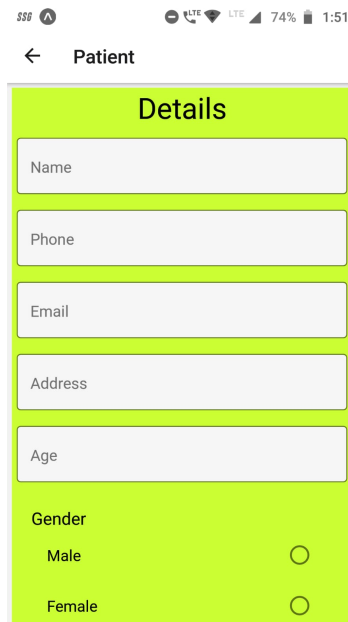
## 2.3 GUI Design

### 2.3.1 Home Page



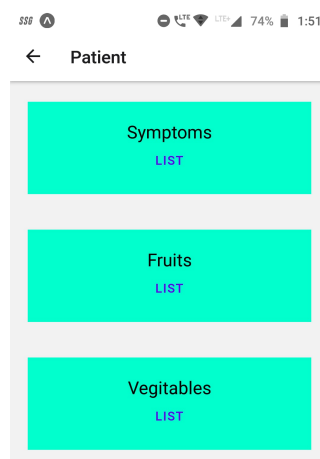
The home page is the first thing which comes when a user first installs the app. We are implementing the “Add Patient” feature here where the blue floating button with the plus sign is implemented.

### 2.3.2 Details Page



The screenshot shows a mobile application interface for a patient's details. At the top, there is a status bar with the text "SSB" and a signal strength icon. Below the status bar, there is a navigation bar with a back arrow and the text "Patient". The main content area is titled "Details" and contains several input fields for patient information: Name, Phone, Email, Address, and Age. Below these fields, there is a section for "Gender" with two radio button options: "Male" and "Female".

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 screenshot shows a mobile application interface for a patient's details. At the top, there is a status bar with the text "SSB" and a signal strength icon. Below the status bar, there is a navigation bar with a back arrow and the text "Patient". The main content area is titled "Details" and contains three sections: "Symptoms", "Fruits", and "Vegetables". Each section has a "LIST" button below it.

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

The screenshot shows a mobile application interface for entering patient details. At the top, there is a status bar with icons for signal strength, LTE, battery level (71%), and time (1:59). Below the status bar is a navigation bar with a back arrow and the title 'Patient'. The main content area is titled 'Details' and contains four input fields: 'Name', 'Phone', 'Email', and 'Address'. The 'Name' field is currently active, and a keyboard is open over it, showing a standard QWERTY layout. The keyboard has a blue checkmark button on the right side.

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

SSB LTE 71% 1:59

← Patient

### Details

Name  
A

Phone  
1

Email

Address

1	2	3	-
4	5	6	_
7	8	9	✕
,	0	.	✓

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

SSB 7 LTE 70% 2:02

← Patient

### Details

Name  
A

Phone  
1

Email  
f20191315@hyderabad.bits-pilani.ac.in

Address

> in to and

1 2 3 4 5 6 7 8 9 0

@ # \$ % & \* + ( ) /

=\< " ' : ; ! ?

ABC , 12 34 . ✓

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

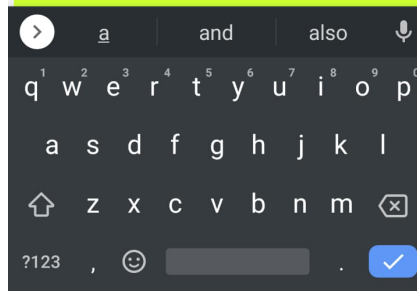
← Patient

Phone  
1

Email  
f20191315@hyderabad.bits-pilani.ac.in

Address  
a

Age



← Patient

Email  
f20191315@hyderabad.bits-pilani.ac.in

Address  
a

Age  
1

Gender

Male ☒

Female ☐

Symptoms  
LIST

← Patient

Symptoms

Running Nose

YES☒

Sneeze

YES☐

Cough

YES☒

Wheeze

YES☐

Headache

← Patient

Fruits

Banana

NR☐

LR☒

MR☐

HR☐

Mango

NR☐

LR☐

MR☒

HR☐



SSG LTE 67% 2:13

← Patient

**Vegitables**

**Avaraikai**

NR	<input type="radio"/>
LR	<input type="radio"/>
MR	<input checked="" type="radio"/>
HR	<input type="radio"/>

**Beans**

NR	<input checked="" type="radio"/>
LR	<input type="radio"/>
MR	<input type="radio"/>
HR	<input type="radio"/>

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

SSG LTE 66% 2:13

← Patient

**Yams**

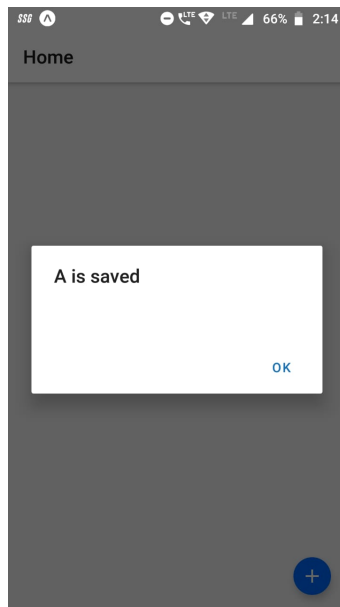
NR	<input type="radio"/>
LR	<input type="radio"/>
MR	<input checked="" type="radio"/>
HR	<input type="radio"/>

**LIST**

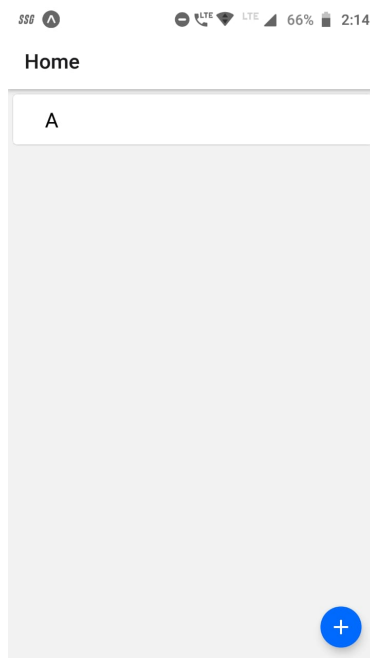
**SAVE**

**CANCEL**






The screen after clicking on the “SAVE” button is




On refreshing the screen, the user can now see the patient name.



## 2.3.4 View Details

SSO     66%  2:14



Report
Name: A
Phone: 1
Email: f20191315@hyderabad.bits-pilani.ac.in
Address: a
Age: 1
Gender: Male
Restrict Fruits: mango fruit2 lime
Avoid Fruits: fruit1
Restrict Vegetables: avaraikai cabbage

SSS 66% 2:14

←

Gender: Male

Restrict Fruits:

- mango
- fruit2
- lime

Avoid Fruits:

- fruit1

Restrict Vegetables:

- avaraikai
- cabbage
- lfinger
- yams

Avoid Vegetables:

- brinjal
- capsicum
- malli

[GENERATE AND MAIL PDF](#) [EDIT](#)

[DELETE](#)

On clicking the card on the home page for a particular patient, the details of the patient can be seen as above.

#### 2.3.5 Edit Page

After the user clicks on the edit button, the user can change any values entered and update.

← Patient

Symptoms

Running Nose

YES

Sneeze

YES

Cough

YES

Wheeze

YES

Headache

← Patient

Fruits

Banana

NR

LR

MR

HR

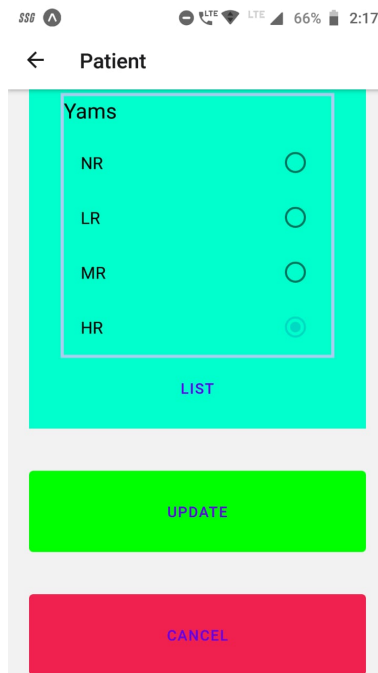
Mango

NR

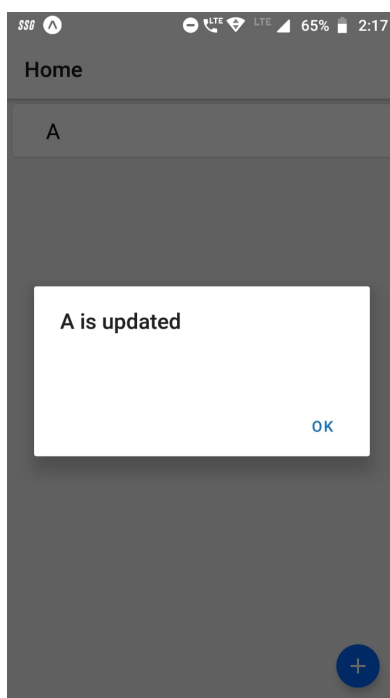
LR

MR

HR



On clicking the update button, the output is



# 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.

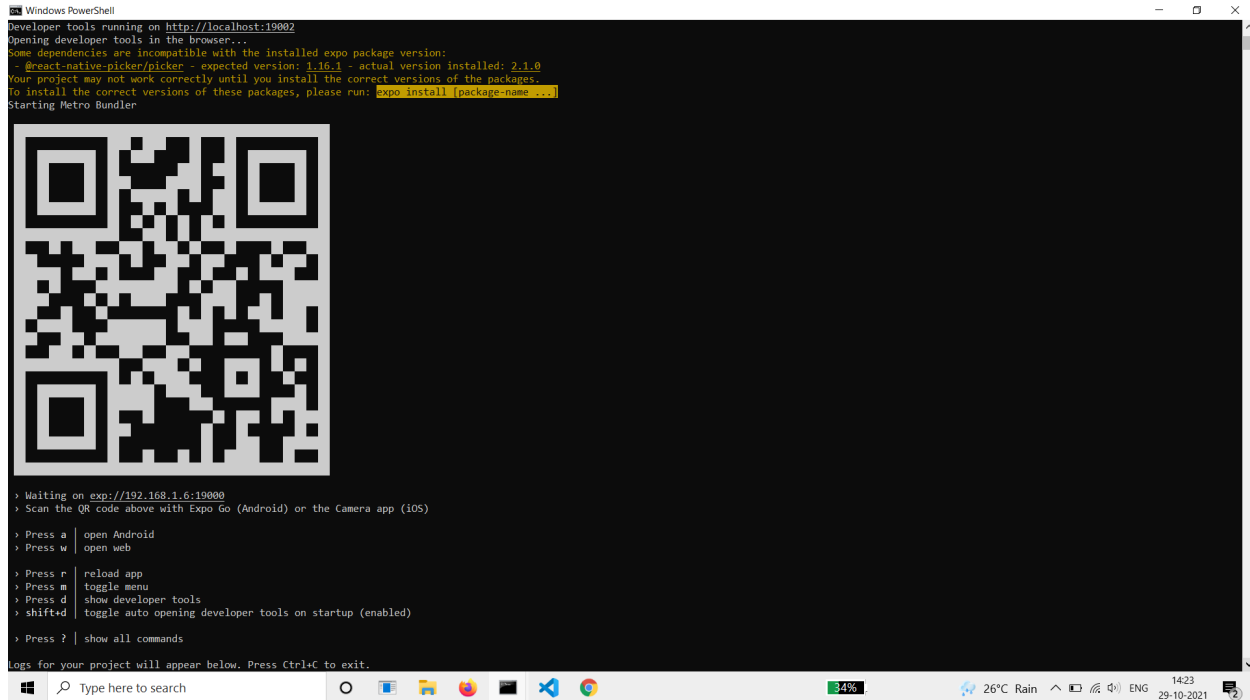
### 3.2 Setting up the Frontend

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



```
Windows PowerShell
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

[QR Code]

> 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 - nodeemon 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-native@>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 - gsmtip
```

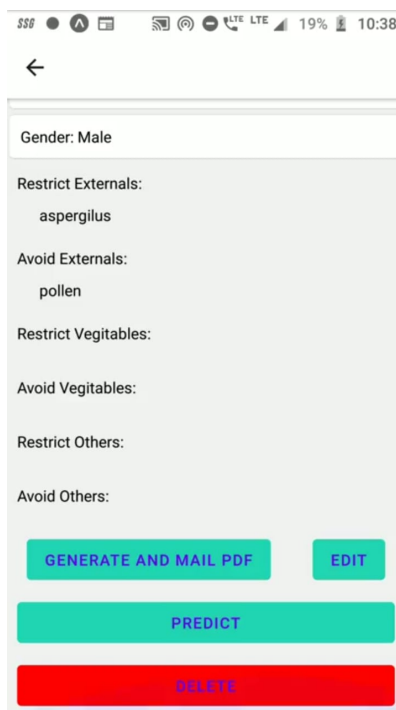
Note: Starting both the front-end and the back-end projects requires environment variables which are not pushed to the repositories for security reasons.

# 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



The screenshot shows a mobile application interface with a status bar at the top displaying 'SSB', signal icons, 'LTE', '19%', and '10:38'. Below the status bar is a back arrow icon. The main content area is a light gray box with the following text: 'Gender: Male', 'Restrict Externals: aspergilus', 'Avoid Externals: pollen', 'Restrict Vegetables:', 'Avoid Vegetables:', 'Restrict Others:', and 'Avoid Others:'. At the bottom of the gray box are two buttons: 'GENERATE AND MAIL PDF' and 'EDIT'. Below the gray box is a large teal button labeled 'PREDICT', and at the very bottom is a large red button labeled 'DELETE'.

Referring to [app.py](http://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

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

as  $P(A | B) = \frac{P(A \cap B)}{P(B)}$  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**:

$$\text{bel}(A) \leq P(A) \leq \text{pl}(A).$$

The belief  $\text{bel}(A)$  for a set A is defined as the sum of all the masses of subsets of the set of interest:

$$\text{bel}(A) = \sum_{B|B \subseteq A} m(B).$$

The plausibility  $\text{pl}(A)$  is the sum of all the masses of the sets B that intersect the set of interest A:

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

The two measures are related to each other as follows:

$$\text{pl}(A) = 1 - \text{bel}(\bar{A}).$$

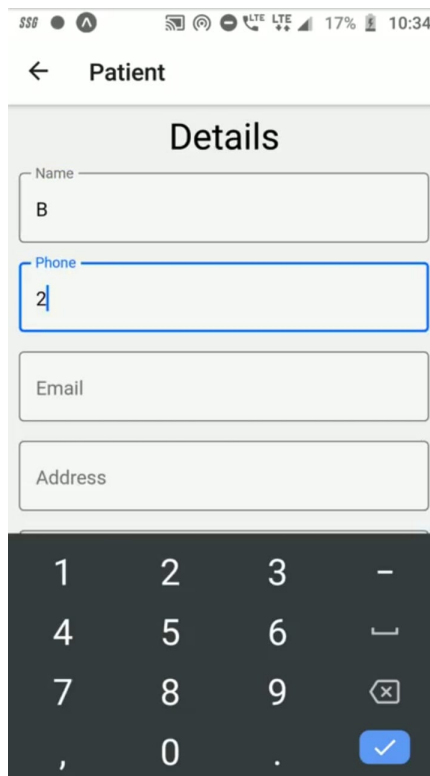
And conversely, for finite A, given the belief measure  $\text{bel}(B)$  for all subsets B of A, we can find the masses  $m(A)$  with the following inverse function:

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

where  $|A - B|$  is the difference of the cardinalities of the two sets.<sup>[4]</sup>

## 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.



The screenshot shows a mobile application interface for a 'Patient' form. At the top, there's a status bar with various icons and the time 10:34. Below it, a navigation bar shows a back arrow and the title 'Patient'. The main content area is titled 'Details' and contains four text input fields: 'Name' (containing 'B'), 'Phone' (containing '2'), 'Email', and 'Address'. A numeric keypad is overlaid at the bottom of the screen, featuring digits 1-9, 0, a decimal point, a comma, a minus sign, an equals sign, a clear button (X), and a confirmation button (checkmark).

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 database initially looks like

## Cluster0

Overview Real Time Metrics Collections Search Profiler Performance Advisor

DATABASES: 1 COLLECTIONS: 1

+ Create Database

Q NAMESPACES

▼ myFirstDatabase

| patients

### myFirstDatabase.patients

COLLECTION SIZE: 0B TOTAL DOCUMENTS: 0 INDEXES TOTAL SIZE: 12KB

Find

Indexes

Schema Anti-Patterns 0

Aggregation

Sea

FILTER { field: 'value' }

QUERY RESULTS 0

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:

```
_id: ObjectId("6177d0f86e876967dd3d2d76")
name: "B"
phone: "2"
email: "f20191315@hyderabad.bits-pilani.ac.in"
address: "b"
age: "2"
gender: "Male"
runningnose: "YES"
sneeze: ""
cough: "NO"
wheeze: ""
headache: ""
itch: ""
swell: ""
redrashes: ""
familyhistory: ""
banana: ""
mango: ""
fruit1: ""
fruit2: ""
lime: ""
avaraikai: "KR"
beans: ""
beetroot: ""
brinjal: ""
cabbage: ""
capsicum: ""
cauliflower: ""
carrot: ""
chowchow: ""
corn: ""
cucumber: ""
drumstick: ""
greens: ""
gourds: ""
kovaikai: ""
kothavarai: ""
lfinger: ""
malli: ""
mushroom: ""
nuckol: ""
onion: ""
peas: ""
potroot: ""
potato: ""
pumkin: ""
pudina: ""
radish: ""
tomato: ""
tondaikai: ""
vazpoo: ""
yams: ""
__v: 0
```

---

## RETRIEVE:

```
{
  _id: new ObjectId("61a1bd2ca8fc04fd21eefd62"),
  name: 'B',
  phone: '2',
  email: 'mehuljain.blr@gmail.com',
  address: 'b2#B',
  age: '2',
  gender: 'Male',
  housedust: 'NR',
  cottondust: 'LR',
  aspergilus: 'MR',
  pollen: 'HR',
  parthenium: 'NR',
  cockroach: "",
  catdander: "",
  dogfur: "",
  roaddust: "",
  olddust: "",
  psdust: "",
  milkp: 'NR',
  milkc: 'LR',
  curd: 'MR',
  coffee: 'HR',
  tea: 'NR',
  beef: "",
  chicken: "",
  mutton: "",
  egg: "",
  fisha: "",
  fishb: "",
  crabs: "",
  prawns: "",
  shark: "",
  avaraikai: 'NR',
  banana: "",
  beans: "",
  beetroot: "",
  brinjal: "",
  cabbage: "",
  capsicum: "",
  chillie: "",
  cauliflower: "",
  carrot: "",
  chowchow: "",
  corn: "",
  cucumber: "",
  drumstick: "",
  greens: "",
  gourds: "",
  kovaikai: "",
  kothavarai: "",
  lfinger: "",
  malli: "",
  mango: "",
  mushroom: "",
  nuckol: ""
}
```

## UPDATE:

```
_id: ObjectId("6177cebf6e876967dd3d2d6c")
name: "A"
phone: "1"
email: "f20191315@hyderabad.bits-pilani.ac.in"
address: "a"
age: "1"
gender: "Male"
runningnose: "YES"
sneeze: "NO"
cough: "NO"
wheeze: "NO"
headache: ""
itch: ""
swell: ""
redrashes: ""
familyhistory: ""
banana: "HR"
mango: "KR"
fruit1: ""
fruit2: "MR"
lime: ""
avaraikai: "NR"
beans: "NR"
beetroot: ""
brinjal: "NR"
```

```
_id: ObjectId("6177cebf6e876967dd3d2d6c")
name: "A"
phone: "1"
email: "f20191315@hyderabad.bits-pilani.ac.in"
address: "a"
age: "1"
gender: "Male"
runningnose: "YES"
sneeze: "YES"
cough: "NO"
wheeze: "NO"
headache: ""
itch: ""
swell: ""
redrashes: ""
familyhistory: "NO"
banana: "HR"
mango: "KR"
fruit1: ""
fruit2: "MR"
lime: ""
avaraikai: "NR"
beans: "NR"
beetroot: ""
brinjal: "NR"
```

## DELETE:

The screenshot shows a web application interface. At the top, there is a filter bar with a button labeled "FILTER" and a text input field containing the JSON query "{name: 'B'}". Below the filter bar, there is a section titled "QUERY RESULTS" followed by the number "0", indicating that no results were found for the given query.

On clicking the “DELETE” button



SSS ● ▲ ☑ 📶 📶 📶 LTE 19% 10:38

←

Gender: Male

Restrict Externals:  
aspergilus

Avoid Externals:  
pollen

Restrict Vegetables:

Avoid Vegetables:

Restrict Others:

Avoid Others:

GENERATE AND MAIL PDF EDIT

PREDICT

DELETE

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.