

*A project report on*

# **SMART CROP PREDICTION AND MONITORING SYSTEM**

**A CAPSTONE PROJECT REPORT**

*Submitted in partial fulfillment of the  
requirement for the award of the  
Degree of*

## **BACHELOR OF TECHNOLOGY IN COMPUTER SCIENCE ENGINEERING WITH SPECIALIZATION IN DATA ANALYTICS**

*by*

**Lakshya Kumar(17BCD7037)**



**SCHOOL OF COMPUTER SCIENCE AND ENGINEERING  
VIT-AP UNIVERSITY  
AMARAVATI- 522237**

*JUNE 2022  
A project report on*

# **SMART CROP PREDICTION AND MONITORING SYSTEM**

**A CAPSTONE PROJECT REPORT**

*Submitted in partial fulfillment of the  
requirement for the award of the  
Degree of*

## **BACHELOR OF TECHNOLOGY IN COMPUTER SCIENCE ENGINEERING WITH SPECIALIZATION IN DATA ANALYTICS**

*by*

**Lakshya Kumar(17BCD7037)**



**SCHOOL OF COMPUTER SCIENCE AND ENGINEERING  
VIT-AP UNIVERSITY  
AMARAVATI- 522237**

*JUNE 2022*



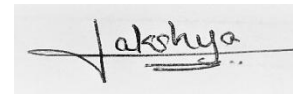
## **DECLARATION**

I here by declare that the thesis entitled “ SMART CROP PREDICTION AND MONITORING SYSTEM” submitted by me, for the award of the degree of Bachelor of Computer Science Engineering with Specialization in Data Analytics VIT is a record of bonafide work carried out by me under the supervision of Dr. Karthika Natarajan.

I further declare that the work reported in this thesis has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place: Amaravati

Date: Jun 12<sup>th</sup>, 2022

A handwritten signature in black ink, appearing to read 'Jashya', is written over a horizontal line.

Signature of the Candidate

## **CERTIFICATE**

This is to certify that the Capstone Project work titled “**SMART CROP PREDICTION AND MONITORING SYSTEM**” that is being submitted by **LAKSHYA KUMAR (17BCD7037)** is in partial fulfillment of the requirements for the award of Bachelor of Technology, is a record of bonafide work done under my guidance. The contents of this Project work, in full or in parts, have neither been taken from any other source nor have been submitted to any other Institute or University for award of any degree or diploma and the same is certified.



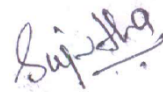
Dr. Kartika Natarajan

Guide

**The thesis is satisfactory**



**Internal Examiner**



**Internal Examiner**

**Approved by**



**PROGRAM CHAIR**

B. Tech. CSE-DA



**DEAN**

School of Computer Science and Engineering

## **ACKNOWLEDGEMENTS**

It is my pleasure to express with deep sense of gratitude to Dr. Karthika Natarajan, Assistant Professor, School of Computer Science and Engineering, VIT-AP, for her constant guidance, continual encouragement, understanding; more than all, she taught me patience in my endeavor. My association with her is not confined to academics only, but it is a great opportunity on my part of work with an intellectual and expert in the field of Data Analytics.

I would like to express my gratitude to Dr. G. Viswanathan, Sankar Viswanathan, Dr. Sekar Viswanathan, G. V. Selvam, Dr. S. V. Kota Reddy, and Dr. Sudha S V, School of computer science and engineering, for providing with an environment to work in and for his inspiration during the tenure of the course.

In jubilant mood I express ingeniously my whole-hearted thanks to Dr. Mehfooza M. Program Chair CSE DA, all teaching staff and members working as limbs of our university for their not-self-centered enthusiasm coupled with timely encouragements showered on me with zeal, which prompted the acquirement of the requisite knowledge to finalize my course study successfully. I would like to thank my parents for their support.

It is indeed a pleasure to thank my friends who persuaded and encouraged me to take up and complete this task. At last but not least, I express my gratitude and appreciation to all those who have helped me directly or indirectly toward the successful completion of this project.

Place: Amaravati

**Lakshya Kumar**

Date: June 12, 2022

**Name of the student**

## **ABSTRACT**

In this paper an efficient system is presented which can predict the ideal crop for a particular piece of land based on the input provided by the customer, for a perfect yield. The yield of a crop is based on numerous factor and a Smart Monitoring system is built in the project, which will keep track of the various factors in the field and provide real-time updates to the user through a web application. Sensors connected with an Arduino constantly monitor the nutrients of the soil and update them in a firebase real-time database. Relying on the farmer's vast experience and the input provided by the application several human errors can be minimized, and farming can happen in a systematic and efficient way.

## **TABLE OF CONTENTS**

<b>S.No.</b>	<b>Chapter</b>	<b>Title</b>	<b>Page Number</b>
<b>1.</b>		<b>Acknowledgement</b>	<b>3</b>
<b>2.</b>		<b>Abstract</b>	<b>4</b>
<b>3.</b>		<b>List of Figures and Table</b>	<b>6</b>
<b>4.</b>	<b>1</b>	<b>Introduction</b>	<b>8</b>
	<b>1.1</b>	<b>Objectives</b>	<b>9</b>
	<b>1.2</b>	<b>Background and Literature Survey</b>	<b>9</b>
		<b>Organization of the Report</b>	
	<b>1.3</b>		<b>11</b>
<b>5.</b>	<b>2</b>	<b>SMART CROP PREDICTION AND MONITORING SYSTEM</b>	<b>12</b>
	<b>2.1</b>	<b>Proposed System</b>	<b>12</b>
	<b>2.2</b>	<b>Working Methodology</b>	<b>13</b>
	<b>2.3</b>	<b>System Details</b>	<b>14</b>
	<b>2.3.1</b>	<b>Software</b>	<b>14</b>
	<b>2.3.2</b>	<b>Hardware</b>	<b>26</b>
<b>6.</b>	<b>3</b>	<b>Cost Analysis</b>	<b>29</b>
	<b>3.1</b>	<b>List of components and their cost</b>	<b>29</b>
<b>7.</b>	<b>4</b>	<b>Results and Discussion</b>	<b>30</b>
<b>8.</b>	<b>5</b>	<b>Conclusion &amp; Future Works</b>	<b>33</b>
<b>9.</b>	<b>6</b>	<b>Appendix</b>	<b>34</b>
<b>10.</b>	<b>7</b>	<b>References</b>	<b>64</b>



## List of Tables

Table No.	Title	Page No.
1.	Cost Analysis	29

## List of Figures

Figure No.	Title	Page No.
1	Share of labour force employed in agriculture in India	9
2	System Component Diagram	13
3	System Deployment Diagram	14
4	Package.json file	15
5	Folder Structure	16
6	login.ejs web-page	16
7	register.ejs web-page	17
8	settings.ejs web-page	19
9	index.ejs web-page	20
10	Firebase Add project option	25
11	Firebase console	26
12	Arduino	27
13	ESP8266 Module	27
14	NPK Sensor	28
15	pH Sensor	28
16	Open Weather API call	30
17	User input and recommended crop	30
18	Flask API call	31
19	Graphs for sensors	31
20	Firebase Database	32

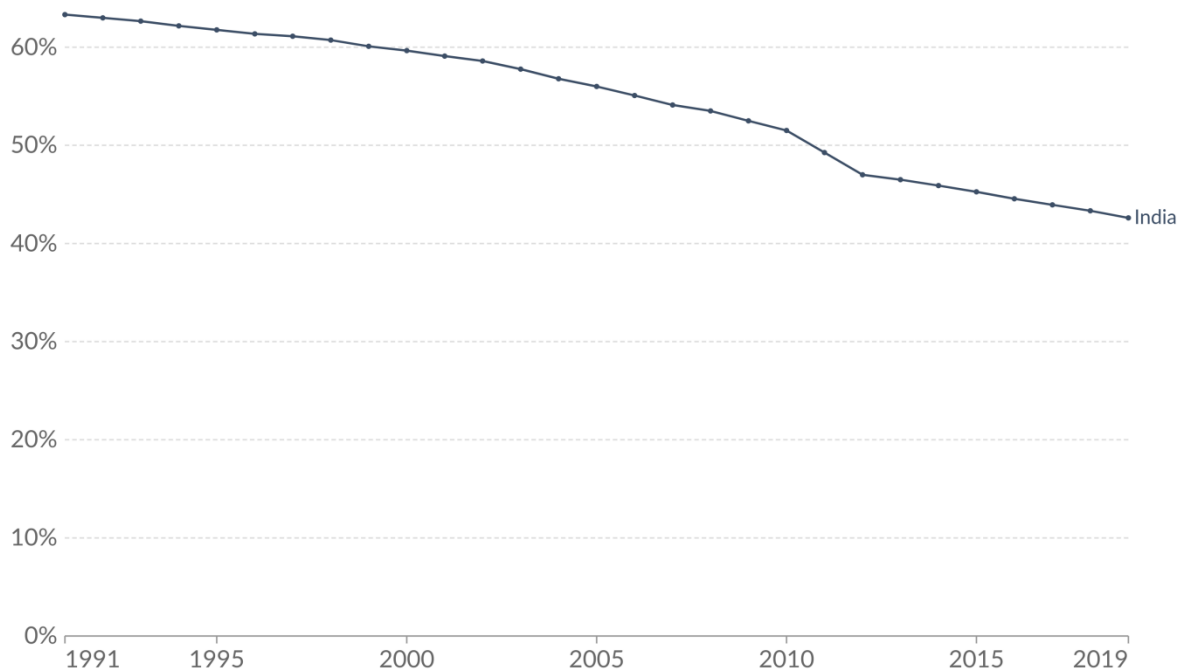
## CHAPTER 1

### INTRODUCTION

In the global age of modern medicine, humans have managed to optimize the mortality rate of the world, but with these advancements also comes the burden of global food shortages and poverty, making the agriculture industry one of the most important profession. In India the agriculture industry employs 41% of people, and amounts to 18% of the country's GDP. Still the ways of farming have remained relatively outdated. Farmers relying on their traditional experience have generally produced great yields but in many cases have destroyed huge harvests due to wrong judgement calls as the traditional method are prone to human errors.

#### Share of the labor force employed in agriculture

Share of people of working age who were engaged in any activity to produce goods or provide services for pay or profit in the agriculture sector (agriculture, hunting, forestry and fishing).



Source: Our World in Data based on International Labor Organization (via the World Bank) and historical sources  
OurWorldInData.org/employment-in-agriculture • CC BY

Fig 1: Share of labour force employed in agriculture in India

These errors can range from over watering the crops to over-estimating the nutrients of the soil, using excess pesticides or even planting the wrong crop. These errors can be easily avoided if

the farmers have exact data on their field and then utilize their judgement to adjust the various supplements they provide to their crops, which not only brings the risk of human errors down but also eliminates the extra cost they would have to endure due to excess addition of fertilizers to the soil.

## **1.1 Objectives**

The following are the objectives of this project:

- To design an efficient system which can predict the ideal crop for a given land based on the inputs provided by the user.
- Monitoring system constantly checks the state of the field after frequent intervals, and updates the user on changes in the values of various factors of their field.
- Provide accurate weather forecasts for accurate decision making.
- To provide customers with a user friendly interpretation of the through a web application.

## **1.2 Background and Literature Survey**

A similar project on the prediction of ideal crop was carried out by M.Kalimuthu, P.Vaishnavi, M.Kishore at Bannari Amman Institute of Technology, Tamil Nadu, India. This is my base paper; the title of the base paper is “Crop Prediction using Machine Learning”. It discusses the use of a Naïve Bayes to predict the ideal crop by considering the temperature, humidity, pH and rain fall. Although the approach was simplistic the accuracy of the model was calculated to be 97%.

In order to study the various factors that the yield of crops I referred the paper by Tandzi Ngoune Liliane and Mutengwa Shelton Charle, named “Factors Affecting Yield of Crops”. In the paper it was discussed that the factors affecting the crop yield can be divided into three categories which are technological, biological and environmental. the paper mentioned in detail the affects of soil fertility and its affects on the yield. Fertile soil provides the crops with micro-nutrients (Fe, B, Cl, Mn, Zn, Cu, Mo, Ni) and macro-nutrients (N, P, K, Ca, S, Mg, C, O, H) that are needed for plant growth, lack of these nutrients can lead to deficiencies in plants and excess of these nutrients can cause toxicity. The biological factors included the diseases the seeds might be carrying and the

effects of pests. Technological factors included the genetic improvements in seeds, adaptive microbiological techniques. Since the technological and biological factors are out of our control on an individual level I decided to use the soil nutrient factors in my model.

For the selection of a suitable model for prediction I referred a paper from Rakesh Kumar, M.P. Singh, Prabhat Kumar and J.P. Singh from NIT Patna, India, named “Crop Selection Method to Maximize Crop Yield Rate using Machine Learning Technique”. Here they discussed various models that can be applicable namely ANN, SVM, KNN, Decision trees, Random forest, Gradient Boosted Decision Tree and Regularized Greedy Forest. They analyzed the advantages and disadvantages of using each of the model, which aided me in deciding which model to use for my paper. According to the paper “An artificial neural network is used when number of input attributes is lesser.” And a random forest classifier is based on tree ensemble machine learning method. It generates multiple tree of randomly sub-sampled features. The output of forest is evaluated by taking average value of the prediction of individual trees. Since it is using random sub-sampled features, Random Forest can be used in high dimension input predictor”. Since random forest classifiers are fast and highly accurate and in the future scope of the project the features will be increased, I decided to use random forest classifier to predict my recommended crop for this project.

To study the workings of various sensors I referred a paper named “IMPLEMENTATION OF SOIL NUTRIENT MEASUREMENT USING RASPBERRY PI” by K.Deepika, A.Dharani, S.Diviya shree, P.Madhavan of the Muthayammal Engineering College, Tamilnadu, India. Here they discussed the workings of the NPK and pH sensors along with temperature, moisture and humidity sensors.

Also for the development of the application was a learning process. I have used Node.js (For deployment of web application), flask (for the deployment of the classifier), Firebase Real time Database (For storing the sensor values). Along with these frameworks I have used express js, passport and open weather api. The documentations for each of these libraries and APIs were referred.

### **1.3 Organization of the Report**

The remaining chapters of the project report are described as follows:

- Chapter 2 contains the proposed system, methodology, hardware and software details.
- Chapter 3 gives the cost involved in the implementation of the project.
- Chapter 4 discusses the results obtained after the project was implemented.
- Chapter 5 concludes the report.
- Chapter 6 consists of codes.
- Chapter 7 gives references.

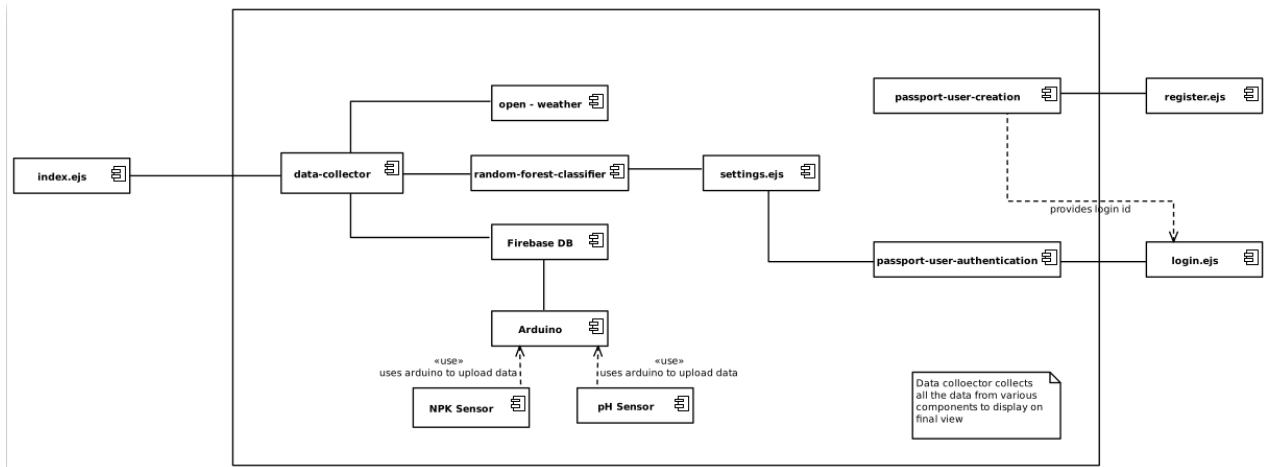
## CHAPTER 2

### SMART CROP PREDICTION AND MONITORING SYSTEM

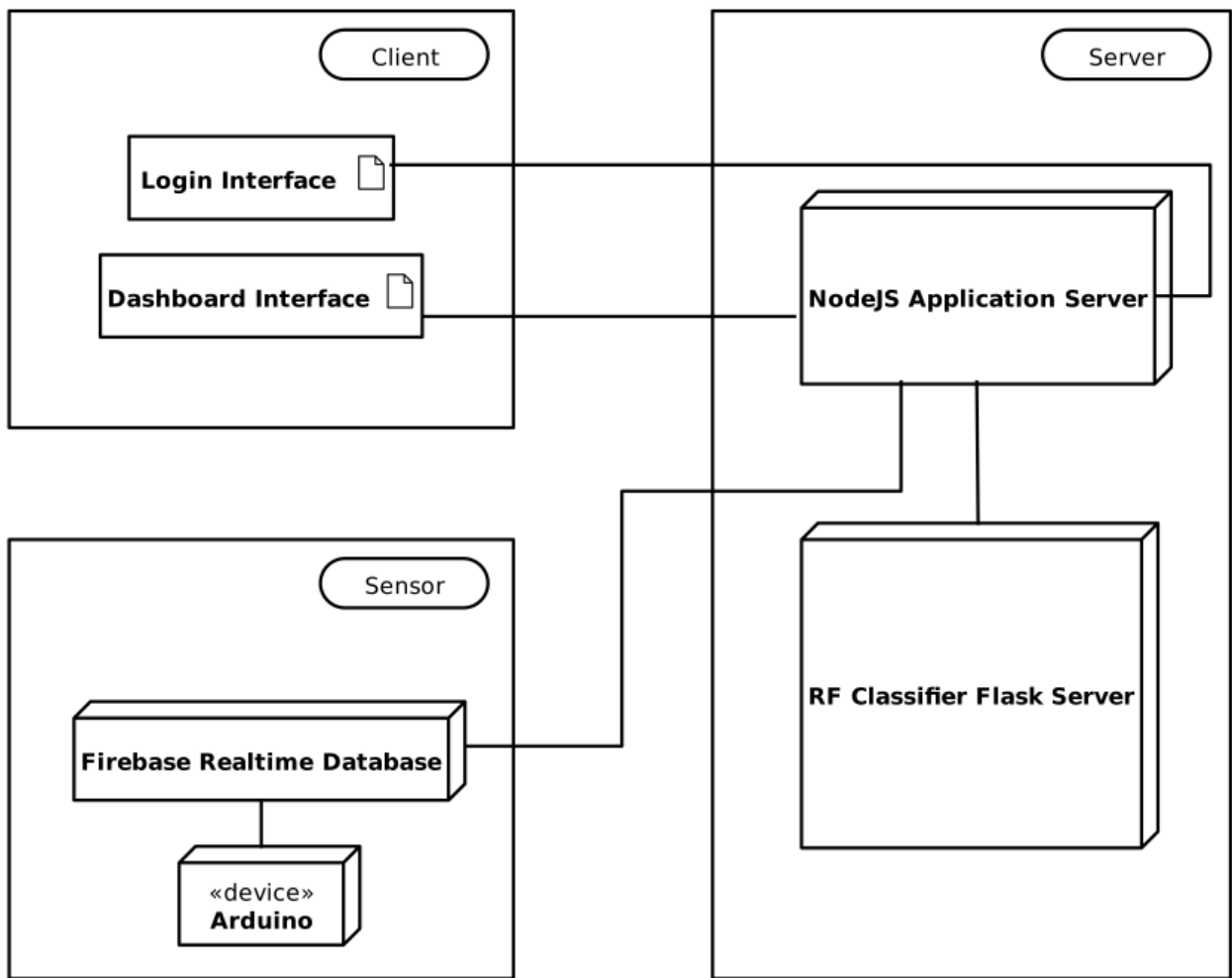
This Chapter describes the proposed system, working methodology, software and hardware details.

#### 2.1 Proposed System

The following block diagram (figure 2) shows the system architecture of this project.



**Figure 2 System Component Diagram**



**Figure 3 System Deployment Diagram**

## 2.2 Working Methodology

The system has two sections, hardware and software. Hardware consists of Arduino which is connected to NPK sensor and pH sensor and an ESP8266 module which allows the Arduino to upload data to firebase.

NPK sensor monitors the nitrogen, potassium and phosphorous levels in the soil. pH sensor monitors the pH levels of the soil. All these values are updated on firebase real-time database.

The software component includes a node js application that takes inputs from the user about the field parameters passes it on to the flask application via a HTTP request to utilize a random forest classifier to predict an ideal crop. The application uses express js, passport and

bcrypt libraries to create a secure authentication system. Firebase and charts.js are used to display the sensor data in real-time. Open-weather API is used to monitor and display the accurate weather information to the user.

A flask application that uses random forest classifier to predict the ideal crop given the input parameters.

## **2.3 System Details**

This section describes the software and hardware details of the system:

### **2.3.1 Software Details**

Node.js web application, flask application, firebase real-time database.

#### **i) Web Application**

The web application is built using the node.js framework. It is highly compatible with the firebase database.

#### **Developing Web Application**

- We start by initializing a node.js project, First we create a new directory and run the command
- - *npm init*
  -
- This command initializes the project and creates the package.json file. package.json file stores the meta data for any project. It also records the dependencies required for the project.
- 
- Second step of the project is to install the dependencies for the project. It can be done by running the command:
-



*npm i bcrypt charts dotenv ejs express express-session firebase passport  
openweather-apis request*

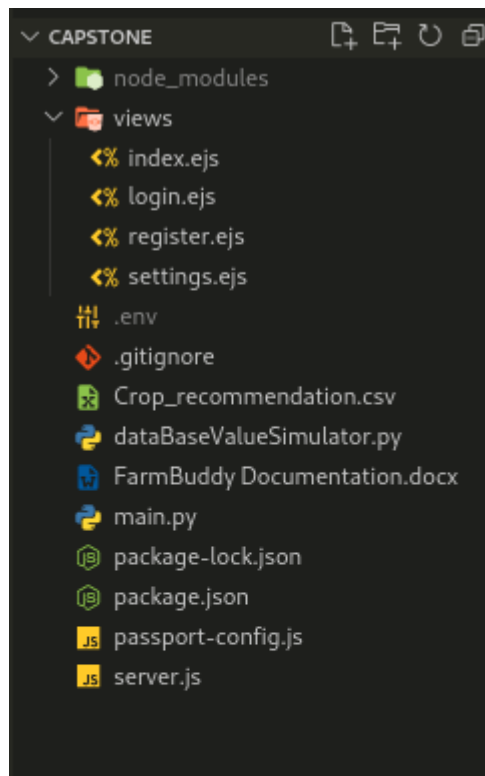
```
package.json > ...
1  {
2    "name": "capstone",
3    "version": "1.0.0",
4    "description": "",
5    "main": "server.js",
6    "scripts": {
7      "devStart": "nodemon server.js"
8    },
9    "keywords": [],
10   "author": "",
11   "license": "ISC",
12   "dependencies": {
13     "bcrypt": "^5.0.1",
14     "chart.js": "^3.7.1",
15     "convert-csv-to-json": "^1.3.3",
16     "dotenv": "^16.0.0",
17     "ejs": "^3.1.7",
18     "express": "^4.18.1",
19     "express-flash": "^0.0.2",
20     "express-oauth2-jwt-bearer": "^1.1.0",
21     "express-openid-connect": "^2.7.2",
22     "express-session": "^1.17.2",
23     "firebase": "^9.7.0",
24     "method-override": "^3.0.0",
25     "openweather-apis": "^4.4.2",
26     "passport": "^0.5.2",
27     "passport-local": "^1.0.0",
28     "plotly.js": "^2.12.0",
29     "random-forest-classifier": "^0.6.0",
30     "request": "^2.88.2"
31   }
32 }
33
```

**Figure 4 Package.json file**

the “main” key in package.json indicates the entry point for the application, in our case it is the server.js file.

For the next step we create the server.js file.

server.js file has to be created in the root of the folder because when we start a node application it looks for the file declared in main key in the source directory.



**Figure 5 Folder Structure**

Create the views folder which contains the static file (HTML, CSS and JS files) for the project. These files are used to create the client side of the application.

A .gitignore file is then created, which allows us to skip certain files from being uploaded on GitHub.

Now a file .env is created which will contain the session secret for authentication

This file should not be made public for security reasons. Hence it is added in the .gitignore file.

Now the four views that are needed for the application are declared, namely index.ejs (Application Dashboard), login.ejs (The login web-page), register.ejs (the register web-page) and the settings.ejs (User input for crop prediction).

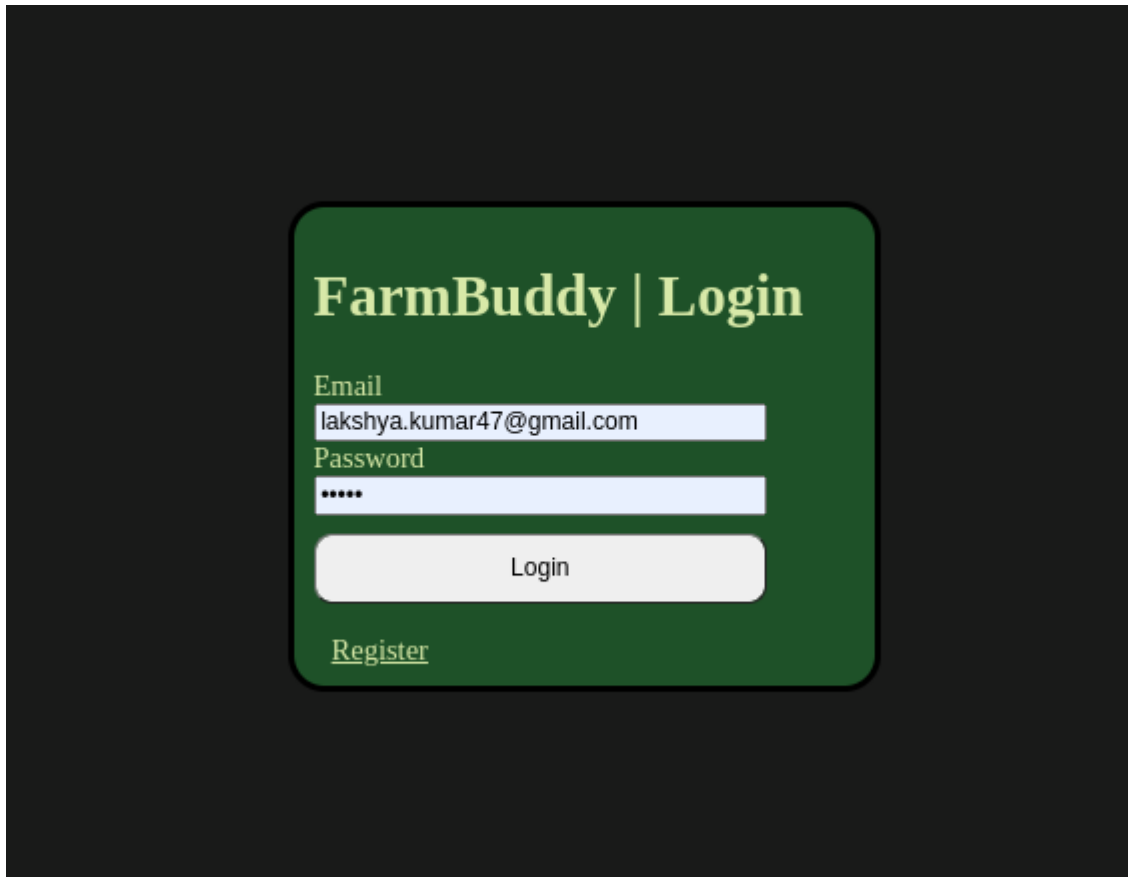
First step in the application will be to build the authentication system for users, for which we must create the login, register, settings and the index ejs files.

With this we have created all the files necessary to build our project. Now we shall begin the code of our program.

- User Authentication: We first write the code for the four web pages created.

Login.ejs will contain a form which demands the users email, password and will have two buttons one to redirect to the registration page and another to login.

In case the user is not registered the page will show an error message asking the user to register. The web-page upon completion will look like figure 6.

The image shows a login form for 'FarmBuddy'. The form is centered on a dark background. It has a title 'FarmBuddy | Login' in a light green font. Below the title, there are two input fields: 'Email' with the value 'lakshya.kumar47@gmail.com' and 'Password' with masked characters '\*\*\*\*\*'. A light green 'Login' button is positioned below the password field. At the bottom of the form, there is a light green link labeled 'Register'.

**Figure 6 login.ejs web-page**

Then we build the register.ejs web-page. register.ejs will contain a form which asks the users name, email and password. It will contain two buttons one to redirect to the login page and another to register the user. The end web-page will look like figure 7.

The image shows a web page titled "FarmBuddy | Register" with a dark green background. It features a registration form with three input fields: "Name", "Email", and "Password". The "Email" field contains the text "lakshya.kumar47@gmail.com" and the "Password" field contains six dots. Below the fields is a light gray "Register" button. At the bottom left, there is a link labeled "Login".

**FarmBuddy | Register**

Name

Email

Password

[Login](#)

**Figure 7 register.ejs web-page**

Upon logging in the user will be redirected to the settings.ejs page where they have to input the factors about their land. The page will contain a form for the respective factors and a button to predict the crop and redirect the user to the dashboard. The web-page will look like figure 8.

Home

### User Input

Please fill out your field data here

pH

N

P

K

Temperature

Rainfall

Humidity

Predict Now

**Figure 8 settings.ejs web-page**

The Dashboard can be divided into three parts, the user information, weather data and the chart representation of the sensor data. The user information part will contain a welcome message, name of the application and logout and settings button. The weather data part will contain three factors temperature, humidity and pressure. The chart representation will contain the four graphs for pH, nitrogen, potassium and phosphorous. The web-page will look like figure 9.



**Figure 9 index.ejs web-page**

After developing the web-pages we can begin writing the server side code for the project. We start by importing the necessary libraries needed for authentication and navigation between the web-pages.

```
const express = require('express')
const app = express()
const bcrypt = require('bcrypt')
const passport = require('passport')
const flash = require('express-flash')
const session = require('express-session')
const methodOverride = require('method-override')
```

Express is used for creating the application.

Bcrypt is a password hashing function.

Passport is the authentication middleware for node.js applications.

Express-flash exposes the getter and setter methods for a flash message of the form.

Express-session is used to manage the sessions in your browser.

The view engine of the application is set and ejs. Embedded JavaScript is a framework that is used in express applications so that the JavaScript code can be rendered on the client side (Used to display the weather details and name of the user). It can be done by running the following command.

```
app.set('view-engine', 'ejs')
```

We now set the functions that will be called upon to navigate through the application.

```
app.get('/', checkAuthenticated, (req, res) => {
  res.render('index.ejs', { name: req.user.name, temp: temprature, hum: humidity, press: pressure})
})
app.get('/login', checkNotAuthenticated, (req, res) => {
  res.render('login.ejs')
})
app.post('/login', checkNotAuthenticated, passport.authenticate('local', {
  successRedirect: '/settings',
  failureRedirect: '/login',
  failureFlash: true
}))
app.get('/register', checkNotAuthenticated, (req, res) => {
  res.render('register.ejs')
})

app.get('/settings', checkAuthenticated, (req, res) => {
  res.render('settings.ejs')
})
app.post('/register', checkNotAuthenticated, async (req, res) => {
  try {
    const hashedPassword = await bcrypt.hash(req.body.password, 10)
    users.push({
      id: Date.now().toString(),
      name: req.body.name,
      email: req.body.email,
      password: hashedPassword
    })
    res.redirect('/login')
  } catch {
    res.redirect('/register')
  }
})
```

```

}))
app.delete('/logout', (req, res) => {
  req.logout()
  res.redirect('/login')
})
function checkAuthenticated(req, res, next) {
  if (req.isAuthenticated()) {
    return next()
  }
  res.redirect('/login')
}

```

Open Weather API: This API is used to fetch the weather data. First we create an account on the following website: [https://home.openweathermap.org/users/sign\\_up](https://home.openweathermap.org/users/sign_up).

Upon signing up we receive an API key that will be used to fetch the data from their servers based on the services you have selected on their website.

Now we set a city and the API key that we acquired from the previous step and using request method we generate an HTTP request to fetch the data in a JSON format. We extract the temperature, humidity and pressure values and store them in global variables, which can be then rendered on the dashboard.

```

let apiKey = '5aaf45c3611c6f7c2dee540a13b1efc1';
let city = 'Delhi';
let url =
`http://api.openweathermap.org/data/2.5/weather?q=${city}&units=imperial&appid=${apiKey}`

request(url, function (err, response, body) {
  if(err){
    console.log('error:', error);
  } else {
    var data = JSON.parse(body);
    global.temprature = data.main.temp;
    global.humidity = data.main.humidity;
    global.pressure = data.main.pressure;
    console.log("open weather working");
  }
});

```



Predicting the crop: Using settings page we collect the user information through a post request and store it in a JSON variable which will be passed as a parameter in our HTTP request to the flask server which contains our random forest classifier and returns a single crop based on the parameter.

```
app.post("/result", async function(req,res) {  
  global.testData = {  
    N: Number(req.body.N),  
    P: Number(req.body.P),  
    K: Number(req.body.K),  
    temp: Number(req.body.temprature),  
    hum: Number(req.body.humidity),  
    pH: Number(req.body.pH),  
    rain: Number(req.body.rainfall)  
  };  
  
  var url = "http://127.0.0.1:5000/flask"
```

```
  request({url:url, qs:testData}, function (error, response, body) {  
    console.log('statusCode:', response && response.statusCode); // Print the response status code if a  
    response was received  
    console.log('body:', body);  
    global.recommendedCrop = body;  
    console.log(recommendedCrop);  
  });  
  res.redirect("/")  
});
```

Flask application:

The flask application is based on a random-forest classifier, we start by creating the python file that will contain the code, which is main.py in our case.

All the packages necessary are imported.

- pandas : For dataset operations
- flask : to create a flask application
- request : to receive HTTP requests
- metrics : to evaluate the accuracy of the classifier
- RandomForestClassifier and train\_test\_split from sklearn module

We follow the following steps to create the application :

1. Reading the dataset using pandas
2. Creating two lists containing the features of the classifier and target of our classifier
3. Splitting the test and training dataset in the ratio 7:3
4. Initializing the random forest classifier with n\_estimators as 20
5. Training the model on training dataset
6. Calculating the accuracy score for the model.
7. Initialize the flask application
8. Create the route /flask which was used by the node.js application to sent the request. The method type is GET.
9. Define a function index
10. Extract the arguments from the request and store them in a dictionary, since the parameters were passed as a JSON object.
11. Create a list of the feature values using the dictionary
12. Pass the list as a parameter to predict the ideal crop using the model and return the result.
13. The app runs on port 5000.

Firebase :

We start by installing the firebase library on the computer by running the command `npm i firebase`.

Then we copy the initial configuration of our firebase application from the firebase console and paste it in our application and create a firebase app using that configuration.

Since the firebase code can run on the front-end we will write the code in our index.ejs using internal js script tags.

1. We create lists for the n,p,k and pH values that will hold 20 values each.
2. Using the firebase application we call the on snapshot function which will run as soon as any new sensor value is updated on the database and fetch the latest values and append them to their respective lists.
3. Now we slice the list in a manner so that only the latest 20 values are left in the list

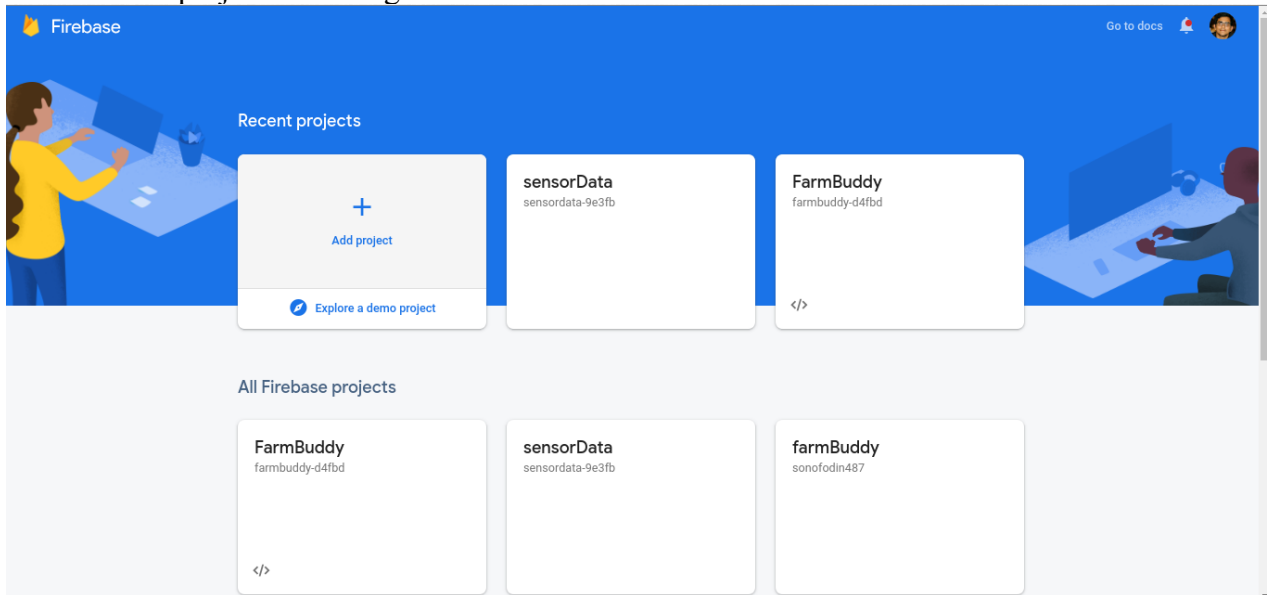
- Using the Chart function from charts.js we plot line graphs for each of the four sensors and display them on the web-page using canvas

## ii) Firebase

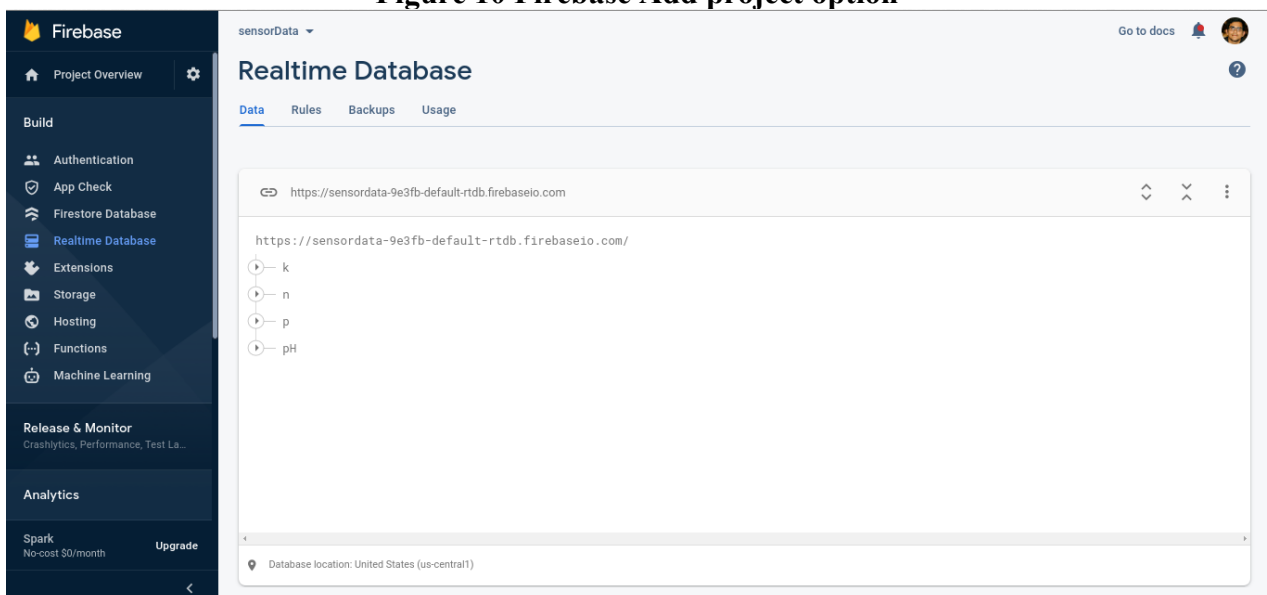
Firebase provides Back end as a Service (BaaS). In this project I have utilized the real-time database service they provide for storage and display of the data.

We start by creating a firebase account on [www.firebase.google.com](http://www.firebase.google.com)

Click on add project and navigate to the real-time database



**Figure 10 Firebase Add project option**



### Figure 11 Firebase console

As you can see the database stores the data in the form of key-value pairs. In our case the n,p,k and pH value gets appended in the corresponding keys, which can be later accessed in the web application.

#### 2.3.2 Hardware Details

As shown in Figure 1 we have various hardware components being used in this system. The details of each component is as follows

##### i) Arduino

Arduino is an open source microcontroller as shown in Figure 27 which can be easily programmed, erased and reprogrammed at any instant of time. Introduced in 2005 the Arduino platform was designed to provide an inexpensive and easy way for hobbyists, students and professionals to create devices that interact with their environment using sensors and actuators. Based on simple microcontroller boards, it is an open source computing platform that is used for constructing and programming electronic devices.



Figure 12 Arduino

The main aim of Arduino is to collect the data from various sensors i.e. NPK sensor and pH sensor, and upload them to the firebase database.

##### ii) ESP8266 Module

ESP8266 is a low cost WiFi module that has a 1mb flash allowing it to connect to a WiFi network, It uses TCP/IP protocol to communicate the WiFi signals.



**Figure 13 ESP8266 Module**

The main aim of this module is to connect the Arduino to WiFi, in order to upload sensor values to the database.

iii) NPK Sensor:

NPK sensors detect the levels of Nitrogen, Potassium and Phosphorous in the soil and supply it to the Arduino to be uploaded onto firebase



**Figure 14 NPK Sensor**

iv) pH Sensor:

pH sensor detects the pH levels of the soil. The pH Sensor will produce a voltage of approximately 1.75 volts in H7 buffer. The voltage will increase by about 0.25 volts for every pH number decrease.



## Figure 15 pH sensor

## **CHAPTER 3**

### **COST ANALYSIS**

#### **3.1 List of components and their cost**

The costs of the various components used in this project are given below in Table 3.1.

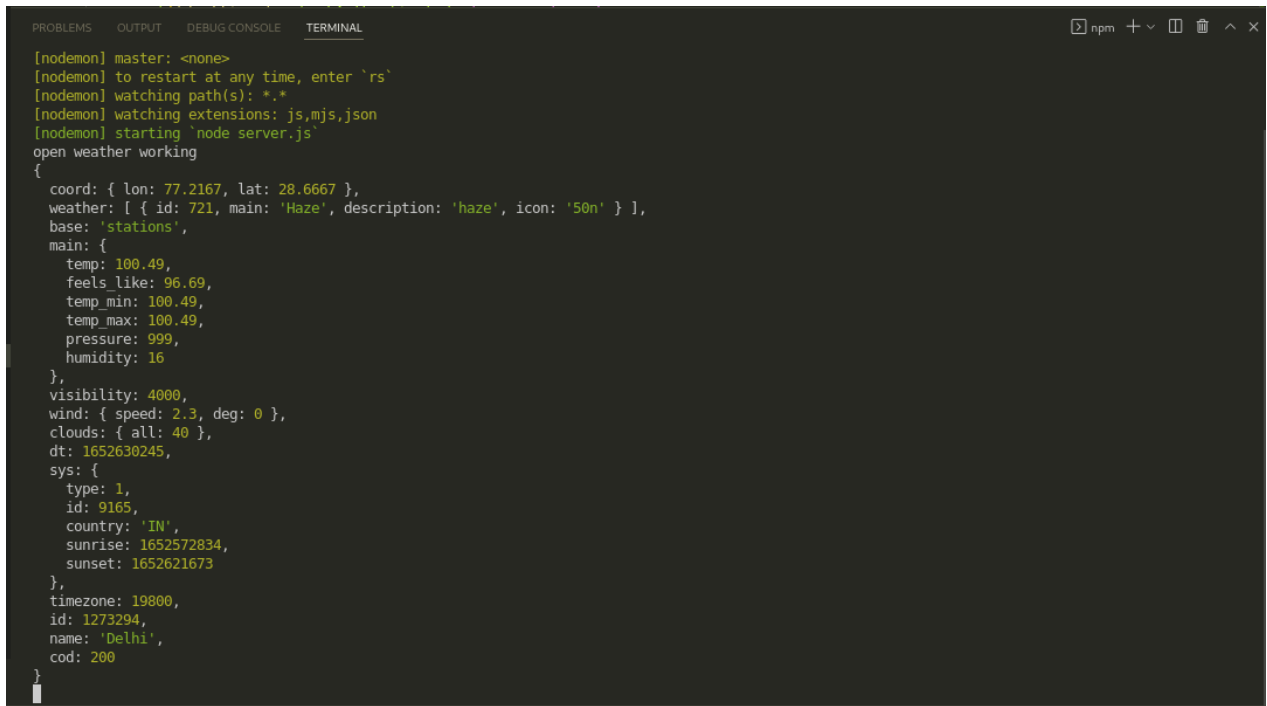
**Table 3.1 List of components and their costs**

<b>COMPONENT</b>	<b>COST</b>
Arduino (1 nos.)	₹ 500
NPK sensor	₹ 500
Bread board	₹ 100
pH sensor	₹ 1000
Jumper wires	₹ 500
ESP8266 Module	₹ 700
Miscellaneous	₹ 500
<b>TOTAL</b>	<b>₹ 3800</b>

## **CHAPTER 4**

### **RESULTS AND DISCUSSIONS**

#### **a) Weather data from open-weather API**



```
[nodemon] master: <none>
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node server.js`
open weather working
{
  coord: { lon: 77.2167, lat: 28.6667 },
  weather: [ { id: 721, main: 'Haze', description: 'haze', icon: '50n' } ],
  base: 'stations',
  main: {
    temp: 100.49,
    feels_like: 96.69,
    temp_min: 100.49,
    temp_max: 100.49,
    pressure: 999,
    humidity: 16
  },
  visibility: 4000,
  wind: { speed: 2.3, deg: 0 },
  clouds: { all: 40 },
  dt: 1652630245,
  sys: {
    type: 1,
    id: 9165,
    country: 'IN',
    sunrise: 1652572834,
    sunset: 1652621673
  },
  timezone: 19800,
  id: 1273294,
  name: 'Delhi',
  cod: 200
}
```

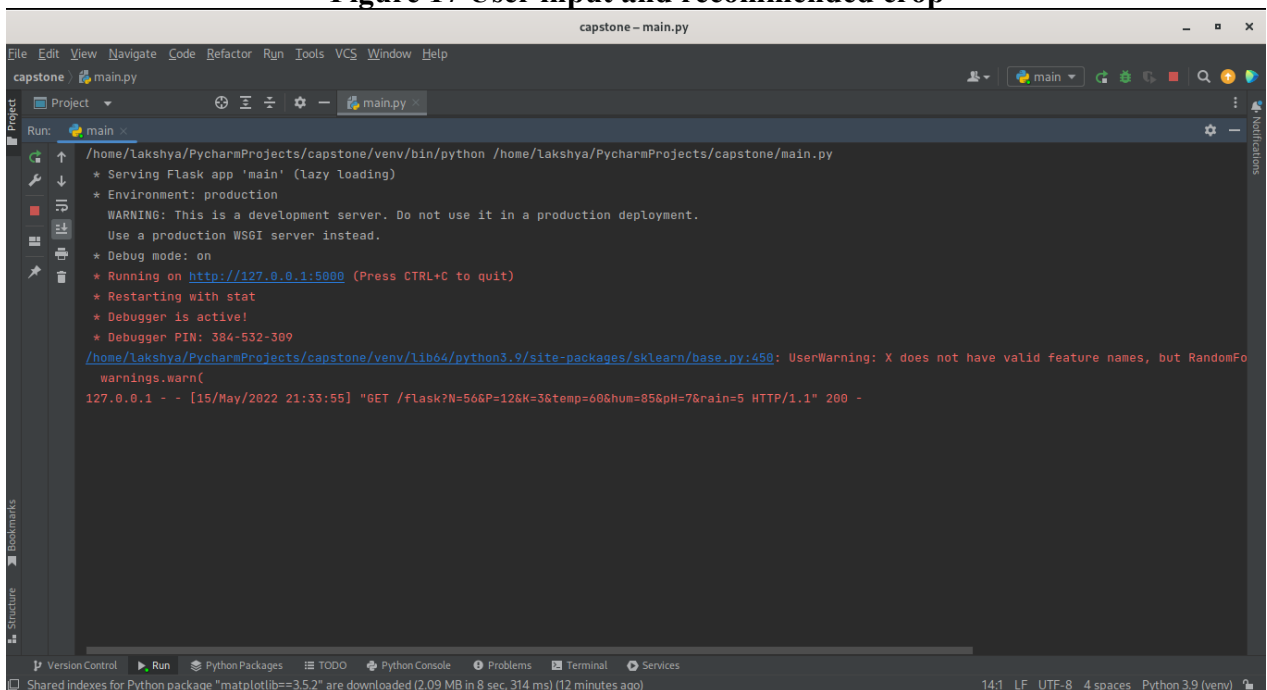
Figure 16 Open Weather API call

**b) HTTP request to flask server and corresponding recommended crop using random forest classifier**



```
{ N: 56, P: 12, K: 3, temp: 60, hum: 85, pH: 7, rain: 5 }
statusCode: 200
body: mungbean
mungbean
```

Figure 17 User input and recommended crop



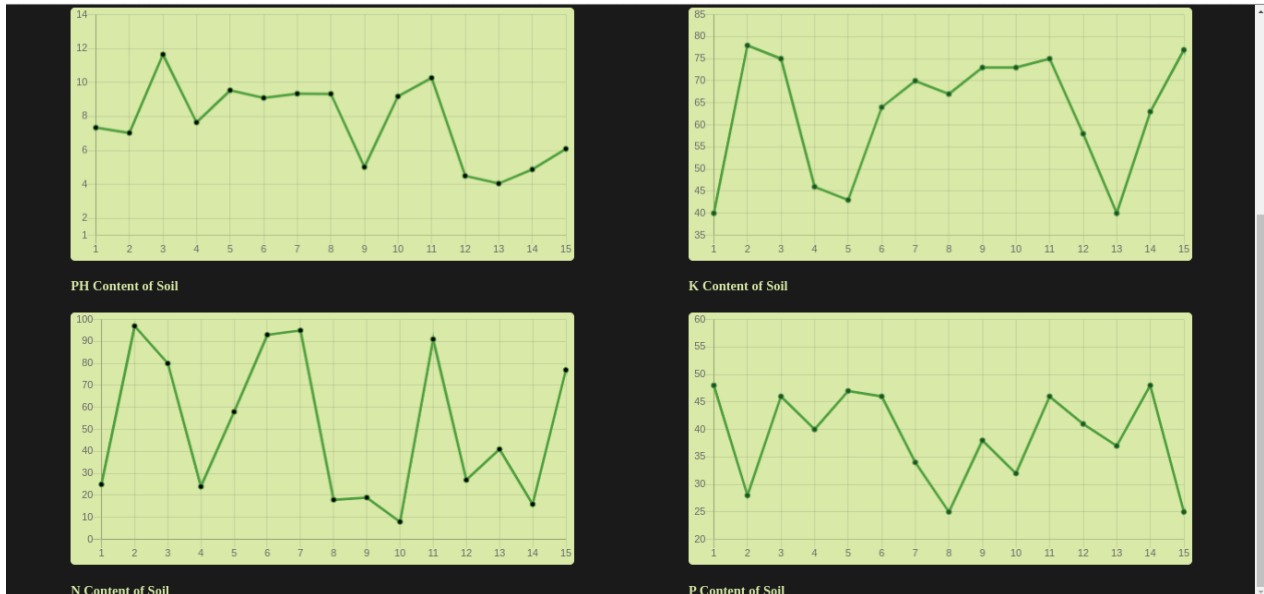
The screenshot shows the PyCharm IDE interface with the 'Run' tab active. The terminal displays the following output:

```
/home/lakshya/PycharmProjects/capstone/venv/bin/python /home/lakshya/PycharmProjects/capstone/main.py
* Serving Flask app 'main' (lazy loading)
* Environment: production
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
* Debug mode: on
* Running on http://127.0.0.1:5000 (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 384-532-309
/home/lakshya/PycharmProjects/capstone/venv/lib64/python3.9/site-packages/sklearn/base.py:450: UserWarning: X does not have valid feature names, but RandomForestClassifier has one
warnings.warn(
127.0.0.1 - - [15/May/2022 21:33:55] "GET /flask?N=56&P=12&K=3&temp=60&hum=85&pH=7&rain=5 HTTP/1.1" 200 -
```



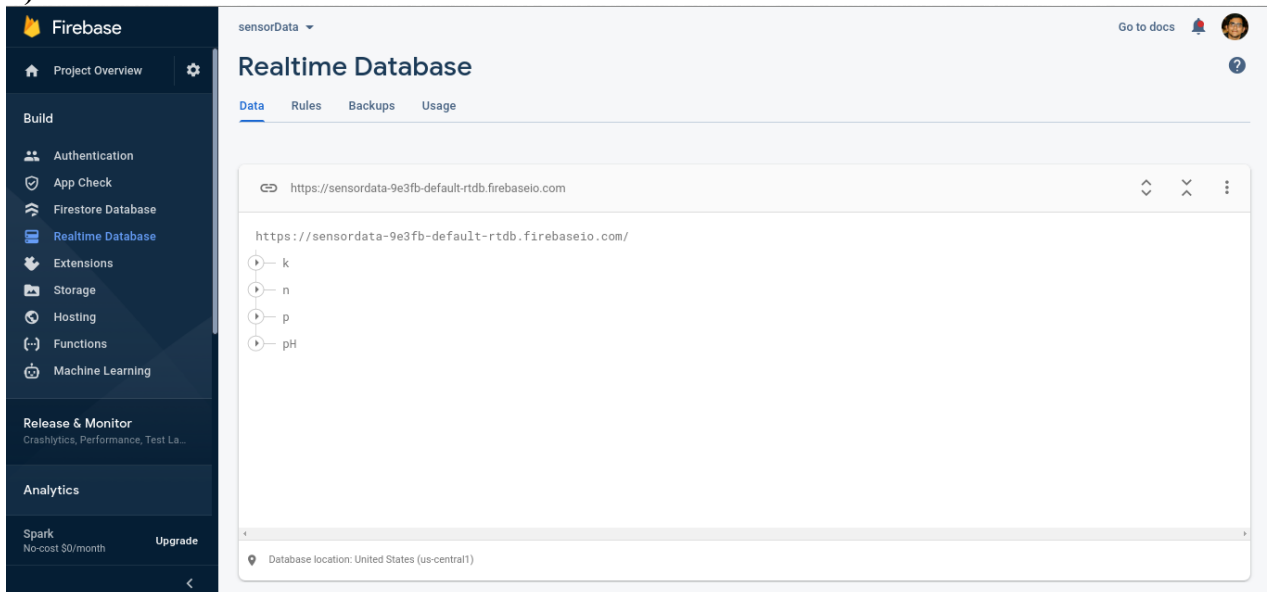
**Figure 18 Flask API call**

**c) Graphical representation of sensor values**



**Figure 19 Graphs for sensors**

**d) Firebase Real-time Database**



**Figure 20 Firebase Database**

## **CHAPTER 5**

### **CONCLUSION AND FUTURE WORK**

Agriculture being one of the biggest employment sectors in our country and one of the most important sectors in the world, is due an upgrade considering the modernization in various fields. Using this project farmers will be able to make informed decisions about their harvest. This will benefit them by cutting costs of fertilizers, maximizing the yield and minimizing the risk of failure in harvest due to excess addition of fertilizers. This project will also help the consumer of those harvests as excess of certain nutrients in the soil can cause the crop to become toxic, which can cause diseases.

In future we can work on including various other technological and biological factors by considering the genetic composition of seeds and the variety of pests local to a region to name a few. We can also consider the land size in comparison to the profits from the harvest to maximize the profit per square feet of land.

## CHAPTER 6

### APPENDIX

#### Web Application Code

##### login.ejs

```
<style>
body {
background-color: rgb(25, 26, 25);
}
.link{
padding: 10px;
}
.wrapper{
position: fixed;
align-items: center;
justify-content: center;
top: 25%;
left: 38%;
padding: 10px;
width: 300px;
height: auto;
background-color: rgb(30, 81, 40);
color: rgb(216, 233, 168);
border: solid black;
border-radius: 20px;
}
.email{
display: flex;
flex-direction: column;
justify-content: stretch;
align-items: flex-start;
}
.password{
display: flex;
flex-direction: column;
justify-content: stretch;
align-items: flex-start;
}
.btn{
margin-top: 10px;
padding: 10px;
width: 250px;
border-radius: 10px;
}
```

```

</style>
<div class="wrapper">
  <h1>FarmBuddy | Login</h1>
  <% if (messages.error) { %>
    <%= messages.error %>
  <% } %>

  <form class="form" action="/login" method="POST">
    <div class="email">
      <label for="email">Email</label>
      <input style="width: 250px;" type="email" id="email" name="email" required>
    </div class="password">
    <div class="password">
      <label for="password">Password</label>
      <input style="width: 250px;" type="password" id="password" name="password" required>
    </div>
    <button class="btn" type="submit">Login</button>
  </form>
  <a class="link" style="color: rgb(216, 233, 168);" href="/register">Register</a>
</div>

```

## register.ejs

```

<style>
  body {
    background-color: rgb(25, 26, 25);
  }
  .link {
    padding: 10px;
  }
  .wrapper {
    position: fixed;
    align-items: center;
    justify-content: center;
    top: 25%;
    left: 38%;
    padding: 10px;
    width: 320px;
  }

```

```
height: auto;
background-color: rgb(30, 81, 40);
color: rgb(216, 233, 168);
border: solid black;
border-radius: 20px;
}
.email{
  display: flex;
  flex-direction: column;
  justify-content: stretch;
  align-items: flex-start;
  width: auto;
}
.password{
  display: flex;
  flex-direction: column;
  justify-content: stretch;
  align-items: flex-start;
  width: auto;
}
.name{
  display: flex;
  flex-direction: column;
  justify-content: stretch;
  align-items: flex-start;
  width: auto;
}
.btn{
  margin-top: 10px;
  padding: 10px;
```

```

width: 250px;
border-radius: 10px;
}

</style>

<div class="wrapper">

  <h1>FarmBuddy | Register</h1>
  <form action="/register" method="POST">
    <div class="name">
      <label for="name">Name</label>
      <input style="width: 250px;" type="text" id="name" name="name" required>
    </div>
    <div class="email">
      <label for="email">Email</label>
      <input style="width: 250px;" type="email" id="email" name="email" required>
    </div>
    <div class="password">
      <label for="password">Password</label>
      <input style="width: 250px;" type="password" id="password" name="password" required>
    </div>
    <button class="btn" type="submit">Register</button>
  </form>
  <a style="color: rgb(216, 233, 168);" href="/login">Login</a>

</div>

```

```
<script src="https://cdn.firebase.com/js/client/2.3.2/firebase.js"></script>
```

```
<script
```

```
src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.9.4/Chart.js">
```

```
</script>
```

```
<script>
```

```
var pH = [], k = [], n = [], p = [];
```

```
var xValues = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15];
```

```
var firebase = new Firebase("https://sensordata-9e3fb-default-rtdb.firebaseio.com/");
```

```
firebase.on('value', function(snapshot) {
```

```
for(let i in snapshot.val().pH){
```

```
    pH.push(snapshot.val().pH[i]);
```

```
}
```

```
for(let i in snapshot.val().k){
```

```
    k.push(snapshot.val().k[i]);
```

```
}
```

```
for(let i in snapshot.val().n){
```

```
    n.push(snapshot.val().n[i]);
```

```
}
```

```
for(let i in snapshot.val().p){
```

```
    p.push(snapshot.val().p[i]);
```

```
}
```

```
pH = pH.slice(pH.length- 20, pH.length);
```

```
k = k.slice(k.length- 20, k.length);
```

```
n = n.slice(n.length- 20, n.length);
```

```
p = p.slice(p.length- 20, p.length);
```

```
new Chart("pHChart", {
```

```

type: "line",
data: {
  labels: xValues,
  datasets: [{
    fill: false,
    lineTension: 0,
    backgroundColor: "rgba(30, 81, 40)",
    borderColor: "rgb(78, 159, 61)",
    data: pH
  }]
},
options: {
  legend: {display: false},
  scales: {
    yAxes: [{ticks: {min: 1, max:14}}],
  }
}
});

new Chart("nChart", {
  type: "line",
  data: {
    labels: xValues,
    datasets: [{
      fill: false,
      lineTension: 0,
      backgroundColor: "rgba(30, 81, 40)",
      borderColor: "rgb(78, 159, 61)",
      data: n
    }]
  },

```



```

options: {
  legend: {display: false},
  scales: {
    yAxes: [{ticks: {min: 0, max:100}}],
  }
}
});

```

```

new Chart("pChart", {
  type: "line",
  data: {
    labels: xValues,
    datasets: [{
      fill: false,
      lineTension: 0,
      backgroundColor: "rgb(30, 81, 40)",
      borderColor: "rgb(78, 159, 61)",
      data: p
    }]
  },
  options: {
    legend: {display: false},
    scales: {
      yAxes: [{ticks: {min: 20, max:60}}],
    }
  }
});

```

```

new Chart("kChart", {
  type: "line",

```

```

data: {
  labels: xValues,
  datasets: [{
    fill: false,
    lineTension: 0,
    backgroundColor: "rgb(30, 81, 40)",
    borderColor: "rgb(78, 159, 61)",
    data: k
  }]
},
options: {
  legend: {display: false},
  scales: {
    yAxes: [{ticks: {min: 35, max:85}}],
  }
}
});

});

```

```
</script>
```

```
<style>
```

```

body {
  background-color: rgb(25, 26, 25);
  color: rgb(216, 233, 168);
}

.weather{

```

```

display: flex;
flex-direction: row;
padding: 10px;
justify-content: space-around;
margin: 10px;
}
.graphs{
display: flex;
flex-direction: row;
justify-content: space-around;

}
.column{
height: 175px;
width: 600px;
}

.cHeightWidth{
background-color: rgb(216, 233, 168);
border-radius: 5px;
}

.top{
display: flex;
flex-direction: row-reverse;
margin-right: 65px;
}
.temprature{
width: 350px;
padding: 20px;

```

```

border-radius: 20px;
}
.humidity{
width: 350px;
padding: 20px;
border-radius: 20px;
}
.Pressure{
width: 350px;
padding: 20px;
border-radius: 20px;
}

```

```

</style>

```

```

<div class="top">

```

```

  <form action="/logout?_method=DELETE" method="POST">

```

```

    <button style=" padding: 5px; margin: 100x; border-radius: 7px; " type="submit">Log
    Out</button>

```

```

  </form>

```

```

  <a style=" padding: 10px; margin: 100x; color: rgb(216, 233, 168); "
  href="/settings">Settings</a>

```

```

</div>

```

```

<h1 style="position: absolute; top: 1px; left: 45%; font-size: xx-large; text-decoration: underline;
padding-top: 0px;">FarmBuddy</h1>

```

```

<div style=" font-size: large; margin-left: 65px ;" class="name">

```

```

  <h2>Welcome Mr. </h2> <%= name %>

```

```

</div>

```

```

<div class="weather">

```

```

<div style="background-color: rgb(30, 81, 40)" class="temprature">
  <h4>Temprature:</h4><br>
  <%= temp %> <p> Fahrenheit</p>
</div>

<div style="background-color: rgb(30, 81, 40)" class="humidity">
  <h4>Humidity:</h4><br>
  <%= hum %> <p>Percent</p>
</div>

<div style="background-color: rgb(30, 81, 40)" class="Pressure">
  <h4>Pressure:</h4><br>
  <%= press %><p>hPa</p>
</div>
</div>

<div class="graphs">
  <div class="column">
    <canvas class="cHeightWidth" id="pHChart" style="border:1px solid"></canvas>
    <h4> PH Content of Soil</h4>
    <canvas class="cHeightWidth" id="nChart" style="border:1px solid"></canvas>
    <h4> N Content of Soil</h4>
  </div>
  <div class="column">
    <canvas class="cHeightWidth" id="kChart" style="border:1px solid"></canvas>
    <h4> K Content of Soil</h4>
    <canvas class="cHeightWidth" id="pChart" style="border:1px solid"></canvas>
    <h4> P Content of Soil</h4>
  </div>
</div>
</div>

```

```

<style>
  body{
    background-color: rgb(25, 26, 25);
  }
  .wrapper{
    position: fixed;
    align-items: center;
    justify-content: center;
    top: 25%;
    left: 38%;
    padding: 10px;
    width: 300px;
    height: auto;
    background-color: rgb(30, 81, 40);
    color: rgb(216, 233, 168);
    border: solid black;
    border-radius: 20px;
  }
  .btn{
    margin-top: 10px;
    padding: 10px;
    width: 250px;
    border-radius: 10px;
  }
</style>

<script src="/server.js"></script>

<a style="position: absolute;top: 10%; right: 10%; color:rgb(216, 233, 168) ;"
href="/">Home</a>

```

```

<form class="wrapper" action="/result" method="POST">
  <h1>User Input</h1>
  <h4>Please fill out your field data here</h4>
  <input id="pH" name="pH" type="text" placeholder="pH"> <br><br>
  <input id="N" name="N" type="text" placeholder="N"><br><br>
  <input id="P" name="P" type="text" placeholder="P"> <br><br>
  <input id="K" name="K" type="text" placeholder="K"> <br><br>
  <input id="temprature" name="temprature" type="text" placeholder="Temprature">
<br><br>
  <input id="rainfall" name="rainfall" type="text" placeholder="Rainfall"> <br><br>
  <input id="humidity" name="humidity" type="text" placeholder="Humidity"> <br><br>

  <button class="btn" id='settingsSubmitButton' type="submit">Predict Now</button>
</form>

```

### server.js

```

//----- Authentication -----

if (process.env.NODE_ENV !== 'production') {
  require('dotenv').config()
}

const express = require('express')
const app = express()
const bcrypt = require('bcrypt')
const passport = require('passport')
const flash = require('express-flash')
const session = require('express-session')
const methodOverride = require('method-override')

```

```

const initializePassport = require('./passport-config')
initializePassport(
  passport,
  email => users.find(user => user.email === email),
  id => users.find(user => user.id === id)
)

const users = []

app.set('view-engine', 'ejs')
app.use(express.urlencoded({ extended: false }))
app.use(flash())
app.use(session({
  secret: process.env.SESSION_SECRET,
  resave: false,
  saveUninitialized: false
}))
app.use(passport.initialize())
app.use(passport.session())
app.use(methodOverride('_method'))

app.get('/', checkAuthenticated, (req, res) => {
  res.render('index.ejs', { name: req.user.name, temp: temprature, hum: humidity, press:
pressure})
})

app.get('/login', checkNotAuthenticated, (req, res) => {
  res.render('login.ejs')
})

```



```

app.post('/login', checkNotAuthenticated, passport.authenticate('local', {
  successRedirect: '/settings',
  failureRedirect: '/login',
  failureFlash: true
}))

```

```

app.get('/register', checkNotAuthenticated, (req, res) => {
  res.render('register.ejs')
})

```

```

app.get('/settings', checkAuthenticated, (req, res) => {
  res.render('settings.ejs')
})

```

```

app.post('/register', checkNotAuthenticated, async (req, res) => {
  try {
    const hashedPassword = await bcrypt.hash(req.body.password, 10)
    users.push({
      id: Date.now().toString(),
      name: req.body.name,
      email: req.body.email,
      password: hashedPassword
    })
    res.redirect('/login')
  } catch {
    res.redirect('/register')
  }
})

```

```

app.delete('/logout', (req, res) => {
  req.logout()
  res.redirect('/login')
})

```

```

function checkAuthenticated(req, res, next) {
  if (req.isAuthenticated()) {
    return next()
  }

```

```

  res.redirect('/login')
}

```

```

function checkNotAuthenticated(req, res, next) {
  if (req.isAuthenticated()) {
    return res.redirect('/')
  }
  next()
}

```

```

app.listen(3000);

```

//----- Open Weather API -----

```

let request = require('request');

```

```

let apiKey = '5aaf45c3611c6f7c2dee540a13b1efc1';

```

```

let city = 'Delhi';

```

```

let url =

```

```

`http://api.openweathermap.org/data/2.5/weather?q=${city}&units=imperial&appid=${apiKey}`

```

```

request(url, function (err, response, body) {
  if(err){
    console.log('error:', error);
  } else {
    var data = JSON.parse(body);
    global.temprature = data.main.temp;
    global.humidity = data.main.humidity;
    global.pressure = data.main.pressure;
    console.log("open weather working");
  }
});

//----- Getting the user data -----
//-----Random forest classifier -----

app.post("/result", async function(req,res){
  global.testData ={
    N: Number(req.body.N),
    P: Number(req.body.P),
    K: Number(req.body.K),
    temp: Number(req.body.temprature),
    hum: Number(req.body.humidity),
    pH: Number(req.body.pH),
    rain: Number(req.body.rainfall)
  };

  var url = "http://127.0.0.1:5000/flask"

  request({url:url, qs:testData}, function (error, response, body) {

```

```

        console.log('statusCode:', response && response.statusCode); // Print the response status
        code if a response was received

        console.log('body:', body);

        global.recommendedCrop = body;

        console.log(recommendedCrop);

    });

    res.redirect("/")

});

```

//-----Connecting to Firebase -----

```

var firebase = require("firebase/app");

// TODO: Add SDKs for Firebase products that you want to use

const firebaseConfig = {
  apiKey: "AIzaSyCGkAefRUTEP7GWCb-lZoBZVbMnCbjqK00",
  authDomain: "farmbuddy-d4fbd.firebaseio.com",
  databaseURL: "https://farmbuddy-d4fbd-default-rtdb.asia-southeast1.firebaseio.com",
  projectId: "farmbuddy-d4fbd",
  storageBucket: "farmbuddy-d4fbd.appspot.com",
  messagingSenderId: "853236971949",
  appId: "1:853236971949:web:70ca38c65366a5e91f8924",
  measurementId: "G-CL2KY8F2D4"
};

const firebaseApp = firebase.initializeApp(firebaseConfig);

```

//-----

### **passport-config.js**

```
const LocalStrategy = require('passport-local').Strategy
const bcrypt = require('bcrypt')

function initialize(passport, getUserByEmail, getUserById) {
  const authenticateUser = async (email, password, done) => {
    const user = getUserByEmail(email)
    if (user == null) {
      return done(null, false, { message: 'No user with that email' })
    }

    try {
      if (await bcrypt.compare(password, user.password)) {
        return done(null, user)
      } else {
        return done(null, false, { message: 'Password incorrect' })
      }
    } catch (e) {
      return done(e)
    }
  }

  passport.use(new LocalStrategy({ usernameField: 'email' }, authenticateUser))
  passport.serializeUser((user, done) => done(null, user.id))
  passport.deserializeUser((id, done) => {
    return done(null, getUserById(id))
  })
}
```

```
module.exports = initialize
```

### **main.py**

```
"""
```

Author: Lakshya Kumar

Registration ID: 17BCD7037

Task : Flask server for predicting the ideal crop using random forest classifier

```
"""
```

```
# Importing all the necessary libraries
```

```
from __future__ import print_function
```

```
import pandas as pd
```

```
from flask import Flask
```

```
from flask import request
```

```
from sklearn import metrics
```

```
from sklearn.ensemble import RandomForestClassifier
```

```
from sklearn.model_selection import train_test_split
```

```
# Reading the dataset
```

```
crop = pd.read_csv("/home/lakshya/Documents/capstone/Crop_recommendation.csv")
```

```
crop.columns = crop.columns.str.replace(' ', '')
```

```
# Declaring the features and label
```

```
features = crop[['N', 'P', 'K', 'temperature', 'humidity', 'ph', 'rainfall']]
```

```
target = crop['label']
```

```
acc = []
```

```

model = []

# splitting the training and testing dataset

x_train, x_test, y_train, y_test = train_test_split(features, target, test_size=0.3, random_state=2)

# Declaring the random forest classifier and training it

RF = RandomForestClassifier(n_estimators=20, random_state=0)
RF.fit(x_train, y_train)

# Testing the classifier

predicted_values = RF.predict(x_test)

# calculating the accuracy of the classifier

x = metrics.accuracy_score(y_test, predicted_values)
acc.append(x)
model.append('RF')

#####
#####

# Creating the flask app

app = Flask(__name__)

```

```
# Declaring route for API call that receives a JSON object as a parameter in the form of an HTTP request
# First the argument is extracted and converted into an array
# Then it is passed through the classifier and the recommended is returned with a 200 response code
# The app runs at port 5000
```

```
@app.route('/flask', methods=['GET'])
```

```
def index():
```

```
    args = request.args
```

```
    # print(args)
```

```
    dictionary = args.to_dict()
```

```
    # print(dictionary)
```

```
    test_values = [dictionary.get('N'), dictionary.get('P'), dictionary.get('K'), dictionary.get('temp'),
                    dictionary.get('hum'), dictionary.get('pH'), dictionary.get('rain')]
```

```
    # print(test_values)
```

```
    result = RF.predict([test_values])[0]
```

```
    return result
```

```
if __name__ == "__main__":
```

```
    app.run(port=5000, debug=True)
```

### **package.json**

```
{
  "name": "capstone",
  "version": "1.0.0",
  "description": "",
  "main": "server.js",
  "scripts": {
```



```

    "devStart": "nodemon server.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "bcrypt": "^5.0.1",
    "chart.js": "^3.7.1",
    "convert-csv-to-json": "^1.3.3",
    "dotenv": "^16.0.0",
    "ejs": "^3.1.7",
    "express": "^4.18.1",
    "express-flash": "^0.0.2",
    "express-oauth2-jwt-bearer": "^1.1.0",
    "express-openid-connect": "^2.7.2",
    "express-session": "^1.17.2",
    "firebase": "^9.7.0",
    "method-override": "^3.0.0",
    "openweather-apis": "^4.4.2",
    "passport": "^0.5.2",
    "passport-local": "^1.0.0",
    "plotly.js": "^2.12.0",
    "random-forest-classifier": "^0.6.0",
    "request": "^2.88.2"
  }
}
}
.gitignore
.env
node_modules

```

**.env**

SESSION\_SECRET=secret

### **Arduino Code**

```
#include <FirebaseObject.h>
#include <FirebaseHttpClient.h>
#include <FirebaseError.h>
#include <FirebaseCloudMessaging.h>
#include <FirebaseArduino.h>
#include <Firebase.h>

#include <ESP8266WiFi.h>

#include <SoftwareSerial.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels
#define OLED_RESET -1 // Reset pin # (or -1 if sharing Arduino reset pin)
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

#define RE 8
#define DE 7

//const byte code[] = {0x01, 0x03, 0x00, 0x1e, 0x00, 0x03, 0x65, 0xCD};
const byte nitro[] = {0x01, 0x03, 0x00, 0x1e, 0x00, 0x01, 0xe4, 0x0c};
const byte phos[] = {0x01, 0x03, 0x00, 0x1f, 0x00, 0x01, 0xb5, 0xcc};
const byte pota[] = {0x01, 0x03, 0x00, 0x20, 0x00, 0x01, 0x85, 0xc0};
```

```

byte values[11];
SoftwareSerial mod(2, 3);

// Setting Values
#define FIREBASE_HOST "https://sensordata-9e3fb-default-rtdb.firebaseio.com/"
#define FIREBASE_AUTH "JYnAIyZlkKXWITAmKiaTS0FmYZoDKSzMqyw7PfpM"
#define WIFI_SSID "17BCD7037"
#define WIFI_PASSWORD "17BCD7037"

void setup() {
  Serial.begin(9600);

  // connect to wifi.
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
  Serial.print("connecting");
  while (WiFi.status() != WL_CONNECTED) {
    Serial.print(".");
    delay(500);
  }
  Serial.println();
  Serial.print("connected: ");
  Serial.println(WiFi.localIP());

  Firebase.begin(FIREBASE_HOST, FIREBASE_AUTH);

  // _____
  delay(1000);
  mod.begin(9600);
  pinMode(RE, OUTPUT);

```

```

pinMode(DE, OUTPUT);

    display.begin(SSD1306_SWITCHCAPVCC, 0x3C); //initialize with the I2C addr 0x3C
(128x64)
    delay(500);
    display.clearDisplay();
    display.setCursor(25, 15);
    display.setTextSize(1);
    display.setTextColor(WHITE);
    display.println(" NPK Sensor");
    display.setCursor(25, 35);
    display.setTextSize(1);
    display.print("Initializing");
    display.display();
    delay(3000);
}

// _____

void loop() {
    byte val1, val2, val3;
    val1 = nitrogen();
    delay(250);
    val2 = phosphorous();
    delay(250);
    val3 = potassium();
    delay(250);

```

```
Serial.print("Nitrogen: ");  
Serial.print(val1);  
Serial.println(" mg/kg");
```

```
Serial.print("Phosphorous: ");  
Serial.print(val2);  
Serial.println(" mg/kg");  
Serial.print("Potassium: ");  
Serial.print(val3);  
Serial.println(" mg/kg");  
delay(2000);
```

```
int fristCommaIndex = values.indexOf(',');  
int secondCommaIndex = values.indexOf(',', fristCommaIndex+1);  
int thirdCommaIndex = values.indexOf(',', secondCommaIndex + 1);
```

```
String n_value = values.substring(0, fristCommaIndex);  
String p_value = values.substring(fristCommaIndex+1, secondCommaIndex);  
String k_value = values.substring(secondCommaIndex+1);
```

```
//store ultrasonic sensor data as string in firebase
```

```
Firebase.setString("n",ultrasonic_value);  
delay(10);
```

```
//store IR sensor 1 data as string in firebase
```

```
Firebase.setString("p",IR_sensor1_value);  
delay(10);
```

```
//store IR sensor 2 data as string in firebase
```

```
Firebase.setString("k",IR_sensor2_value);
```

```
}
```

```
byte nitrogen() {  
    digitalWrite(DE, HIGH);  
    digitalWrite(RE, HIGH);  
    delay(10);  
    if (mod.write(nitro, sizeof(nitro)) == 8) {  
        digitalWrite(DE, LOW);  
        digitalWrite(RE, LOW);  
        for (byte i = 0; i < 7; i++) {  
            //Serial.print(mod.read(),HEX);  
            values[i] = mod.read();  
            Serial.print(values[i], HEX);  
        }  
        Serial.println();  
    }  
    return values[4];  
}
```

```
byte phosphorous() {  
    digitalWrite(DE, HIGH);  
    digitalWrite(RE, HIGH);  
    delay(10);  
    if (mod.write(phos, sizeof(phos)) == 8) {  
        digitalWrite(DE, LOW);  
        digitalWrite(RE, LOW);  
        for (byte i = 0; i < 7; i++) {  
            //Serial.print(mod.read(),HEX);  
            values[i] = mod.read();  
            Serial.print(values[i], HEX);  
        }  
    }  
    return values[4];  
}
```

```

    }
    Serial.println();
}
return values[4];
}

byte potassium() {
    digitalWrite(DE, HIGH);
    digitalWrite(RE, HIGH);
    delay(10);
    if (mod.write(pota, sizeof(pota)) == 8) {
        digitalWrite(DE, LOW);
        digitalWrite(RE, LOW);
        for (byte i = 0; i < 7; i++) {
            //Serial.print(mod.read(),HEX);
            values[i] = mod.read();
            Serial.print(values[i], HEX);
        }
        Serial.println();
    }
    return values[4];
}

```

## REFERENCES

1. M. Kalimuthu, P. Vaishnavi and M. Kishore, "Crop Prediction using Machine Learning," 2020 Third International Conference on Smart Systems and Inventive Technology (ICSSIT), 2020, pp. 926-932, doi: 10.1109/ICSSIT48917.2020.9214190.
2. A. Nigam, S. Garg, A. Agrawal and P. Agrawal, "Crop Yield Prediction Using Machine Learning Algorithms," 2019 Fifth International Conference on Image Information Processing (ICIIP), 2019, pp. 125-130, doi: 10.1109/ICIIP47207.2019.8985951.
3. R. Kumar, M. P. Singh, P. Kumar and J. P. Singh, "Crop Selection Method to maximize crop yield rate using machine learning technique," 2015 International Conference on Smart Technologies and Management for Computing, Communication, Controls, Energy and Materials (ICSTM), 2015, pp. 138-145, doi: 10.1109/ICSTM.2015.7225403.
4. Liliane, Tandzi & Mutengwa, Charles. (2020). Factors Affecting Yield of Crops. 10.5772/intechopen.90672.
5. IMPLEMENTATION OF SOIL NUTRIENT MEASUREMENT USING RASPBERRY PI K.Deepika<sup>1</sup>, A.Dharani<sup>2</sup>, S.Diviya shree<sup>3</sup>, P.Madhavan<sup>4</sup> 1, 2 & 3UG Students, Department of ECE, Muthayammal Engineering College, Tamilnadu, India 4Assistant professor, Department of ECE, Muthayammal Engineering College, Tamilnadu, India
6. <https://create.arduino.cc/projecthub/pulasthi-Narada/send-multiple-sensors-data-to-firebase-using-esp8266-f2f38b>
7. <https://how2electronics.com/measure-soil-nutrient-using-arduino-soil-npk-sensor/>
8. <https://www.kaggle.com/code/kiran004/crop-recommendation/data>
9. <https://www.passportjs.org/docs/>
10. <https://www.chartjs.org/docs/latest/>
11. <https://firebase.google.com/docs/database>
12. <https://readthedocs.org/projects/flask/>



## BIODATA



Name : Lakshya Kumar

Mobile Number : 9620792339

E-mail : lakshya.kumar@vitap.ac.in

Permanaent Address : 202 Sri Sai Nilayam, 7<sup>th</sup> cross, Chinnapanhalli Bangalore 560037