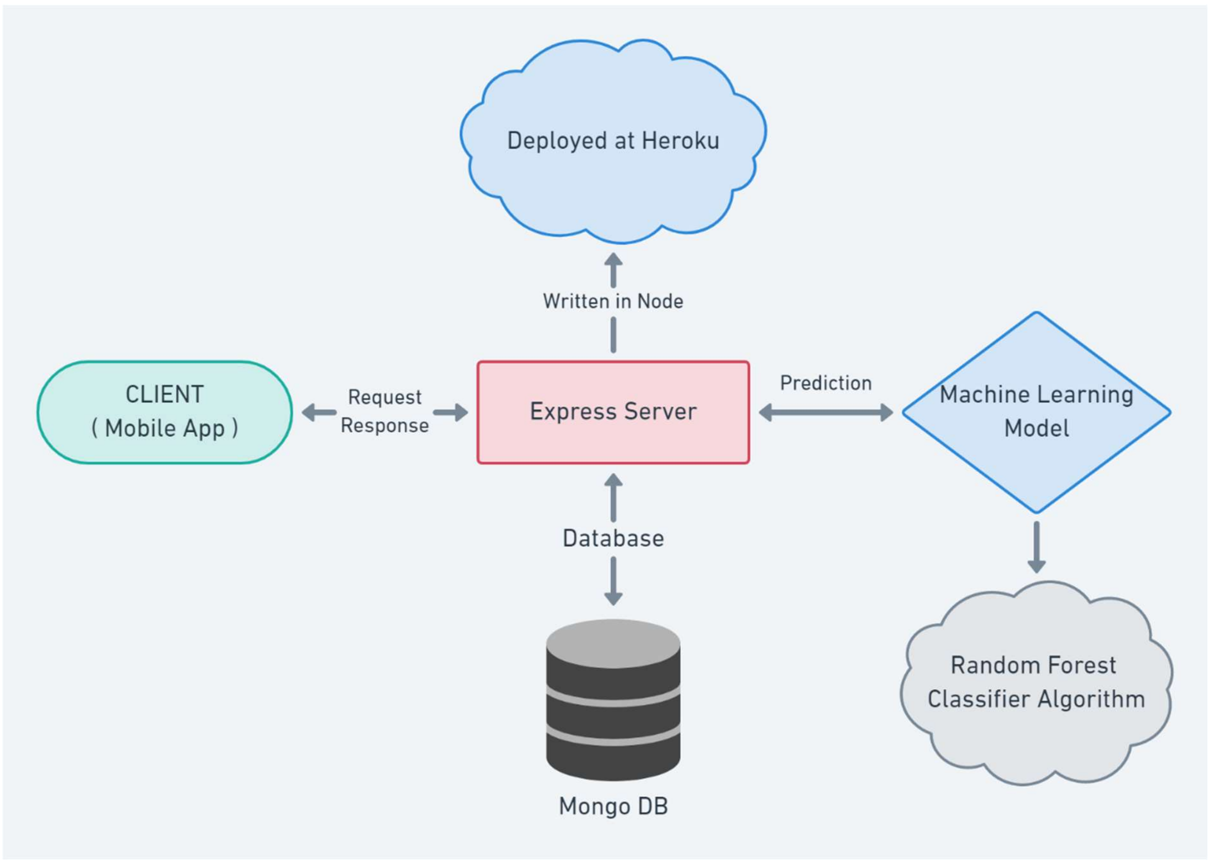
TASK 2

Ayush Srivastava

Product Design and Details

‘Grow-More’ is a complete mobile application project with a prediction Machine Learning algorithm deployed on back-end. We have tested various Machine Learning algorithms for our project and picked up the best performing one. The algorithm floats on a backend built using Express and written in Node.JS which has features like authentication and recommendation history too. Backend is deployed on Heroku. Further the front-end of the app is built using React Native. React native Framework uses React library to build native mobile apps for Android and iOS. Expo Managed Flow was used in development of mobile application.



Github Links:

App Repo: https://github.com/entrymaster/grow-more

Bckend Repo: https://github.com/entrymaster/grow-more-backend

ML- Model: https://github.com/entrymaster/predictor

Tech Stack

#) FRONT-END

IDE: VS Code

Language: JavaScript + JSX

Framework: React Native (Expo Managed Workflow)

Packages: Expo (SDK 44), React Navigation@6.x,

React-native-async-storage@1.15.14

#) BACK-END

IDE: VS Code

Language: JavaScript

Framework: Node JS

Libraries: Express.JS, Mongoose.JS, Router, MongoDB, MongoDB Atlas, Heroku

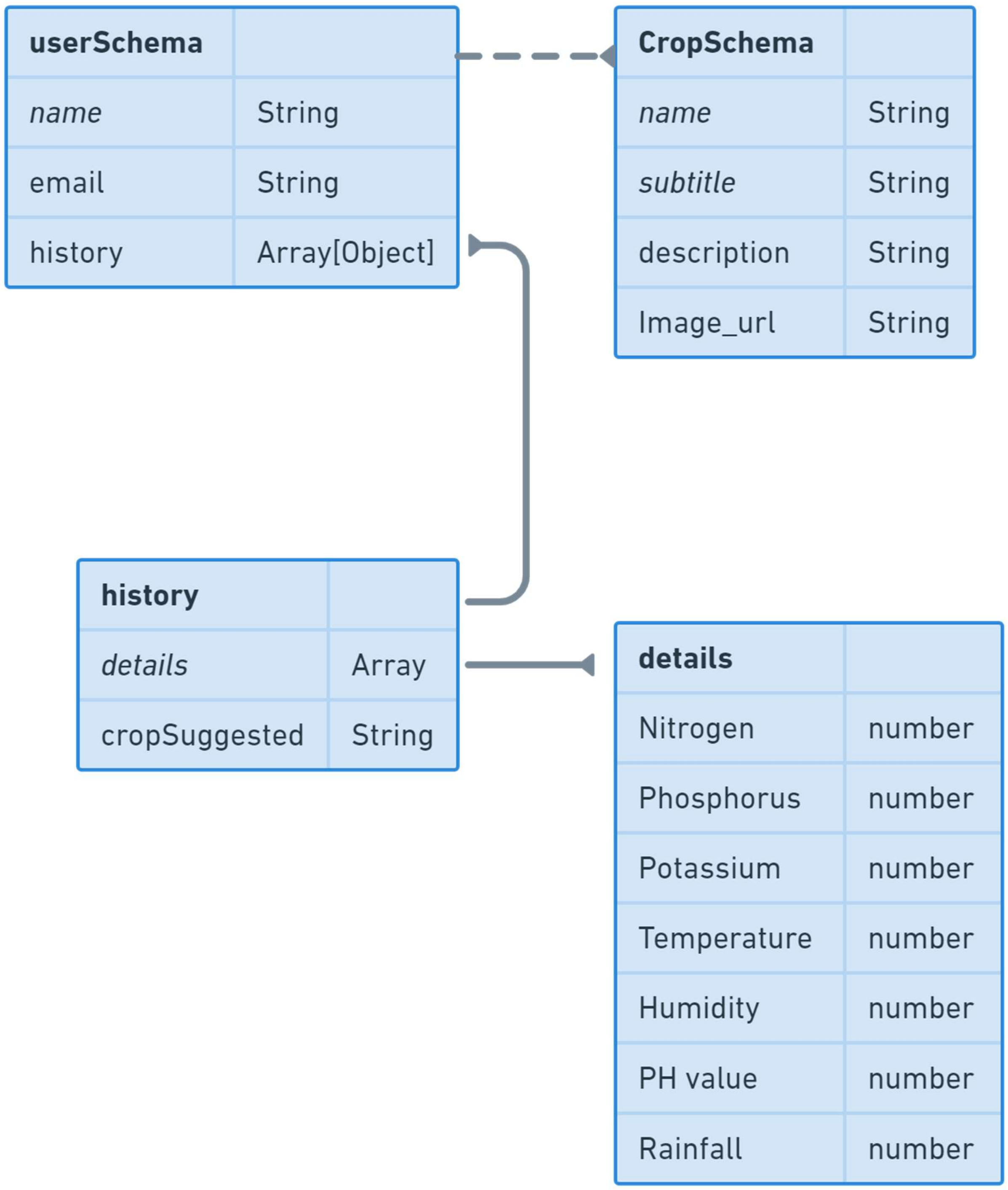
#) ML-MODEL

IDE: VS Code, Jupyter Notebook

Language: Python

Libraries: Pandas, SkLearn, Numpy

Db Schema



userSchema	
name	String
email	String
history	Array[Object]

CropSchema	
name	String
subtitle	String
description	String
Image_url	String

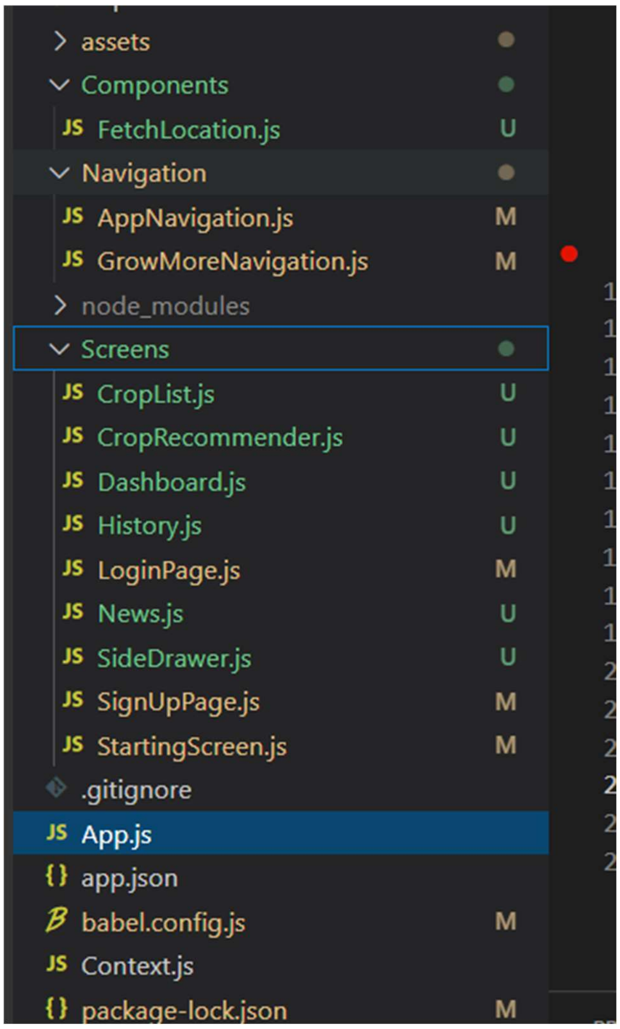
history	
details	Array
cropSuggested	String

details	
Nitrogen	number
Phosphorus	number
Potassium	number
Temperature	number
Humidity	number
PH value	number
Rainfall	number

Mobile Application

Component Structure

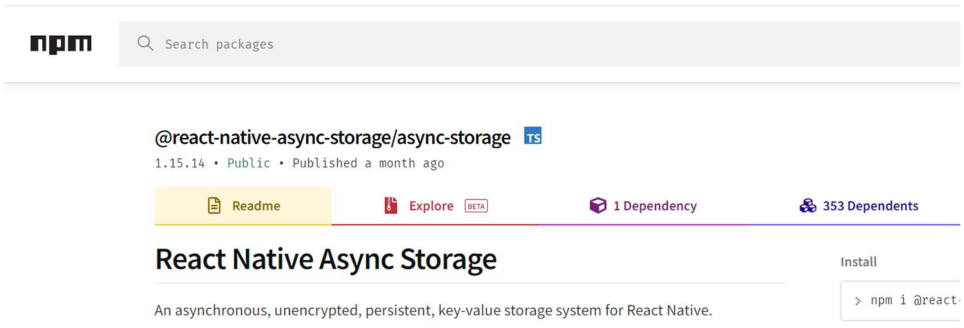
We followed a component-based structuring where every necessary component of our app is divided and kept in separate folder which can be further recalled thus enabling easy de-bugging and maintenance. Screens are built with built-in or custom components so that they can be reused. Assets are used to keep all media and JSON in one place.



```
> assets
v Components
    FetchLocation.js          U
v Navigation
    AppNavigation.js          M
    GrowMoreNavigation.js     M
> node_modules
v Screens
    CropList.js               U
    CropRecommender.js        U
    Dashboard.js              U
    History.js                U
    LoginPage.js              M
    News.js                   U
    SideDrawer.js             U
    SignUpPage.js             M
    StartingScreen.js         M
  .gitignore
  App.js
  app.json
  babel.config.js             M
  Context.js
  package-lock.json           M
```

Storage Used

React Native Async Storage is used to store user data in the app temporarily and later send it to the database server.



npm Search packages

@react-native-async-storage/async-storage TS
1.15.14 • Public • Published a month ago

Readme | Explore BETA | 1 Dependency | 353 Dependents

React Native Async Storage

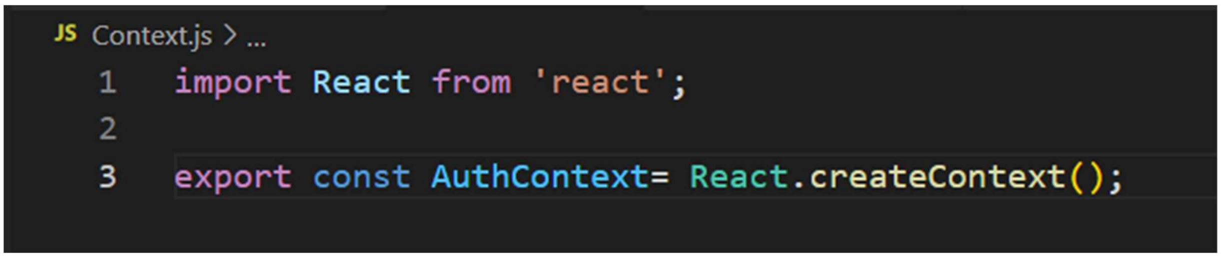
An asynchronous, unencrypted, persistent, key-value storage system for React Native.

Install

```
> npm i @react
```

React Context API

Context provides a way to pass data through the component tree without having to pass props down manually at every level. This prevents prop drilling. I have created a context separately in Context.js



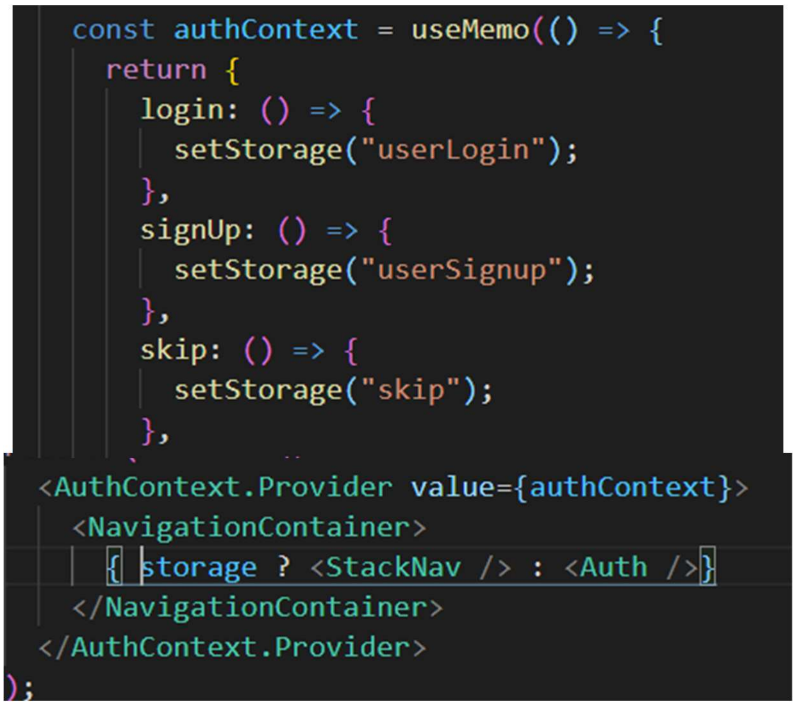
JS Context.js > ...

```
import React from 'react';

export const AuthContext= React.createContext();
```

Context.Provider

Context comes with a Provider React component allows customer components to subscribe to context changes. All components which are descendants of a Provider will re-render whenever the Provider's value prop changes.

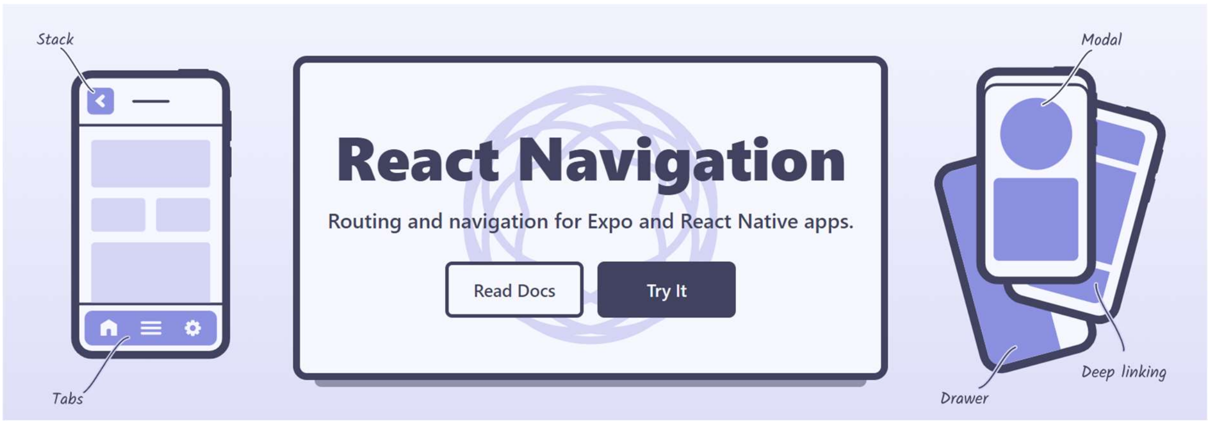


```
const authContext = useMemo(() => {
  return {
    login: () => {
      setStorage("userLogin");
    },
    signUp: () => {
      setStorage("userSignup");
    },
    skip: () => {
      setStorage("skip");
    },
```

```
<AuthContext.Provider value={authContext}>
  <NavigationContainer>
    { storage ? <StackNav /> : <Auth />}
  </NavigationContainer>
</AuthContext.Provider>
);
```

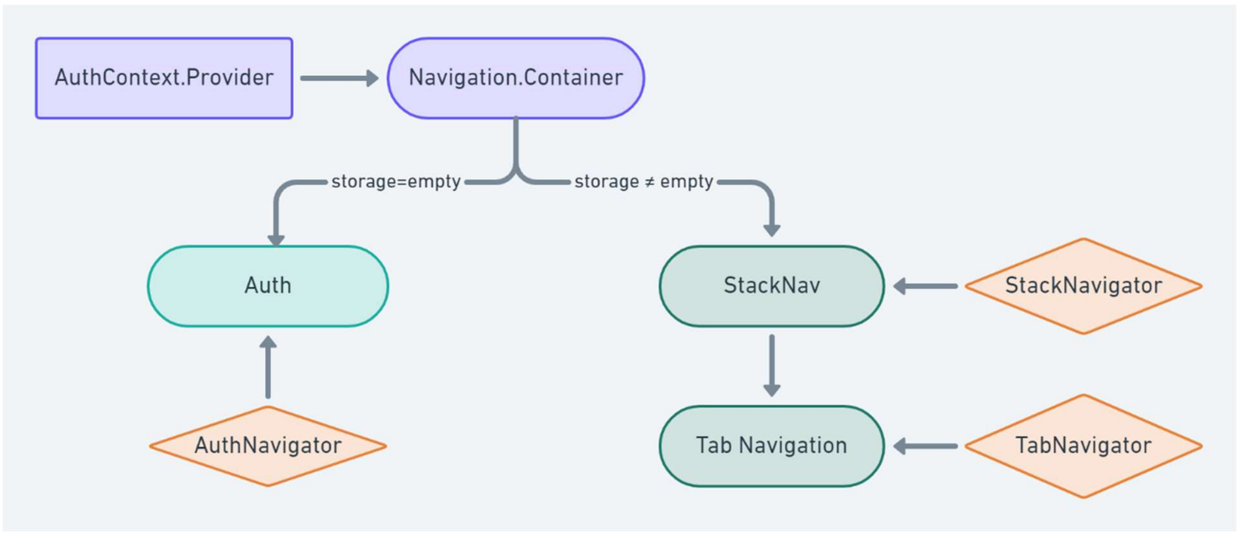
App Navigation

We used react navigation to build the navigation structure of our app.

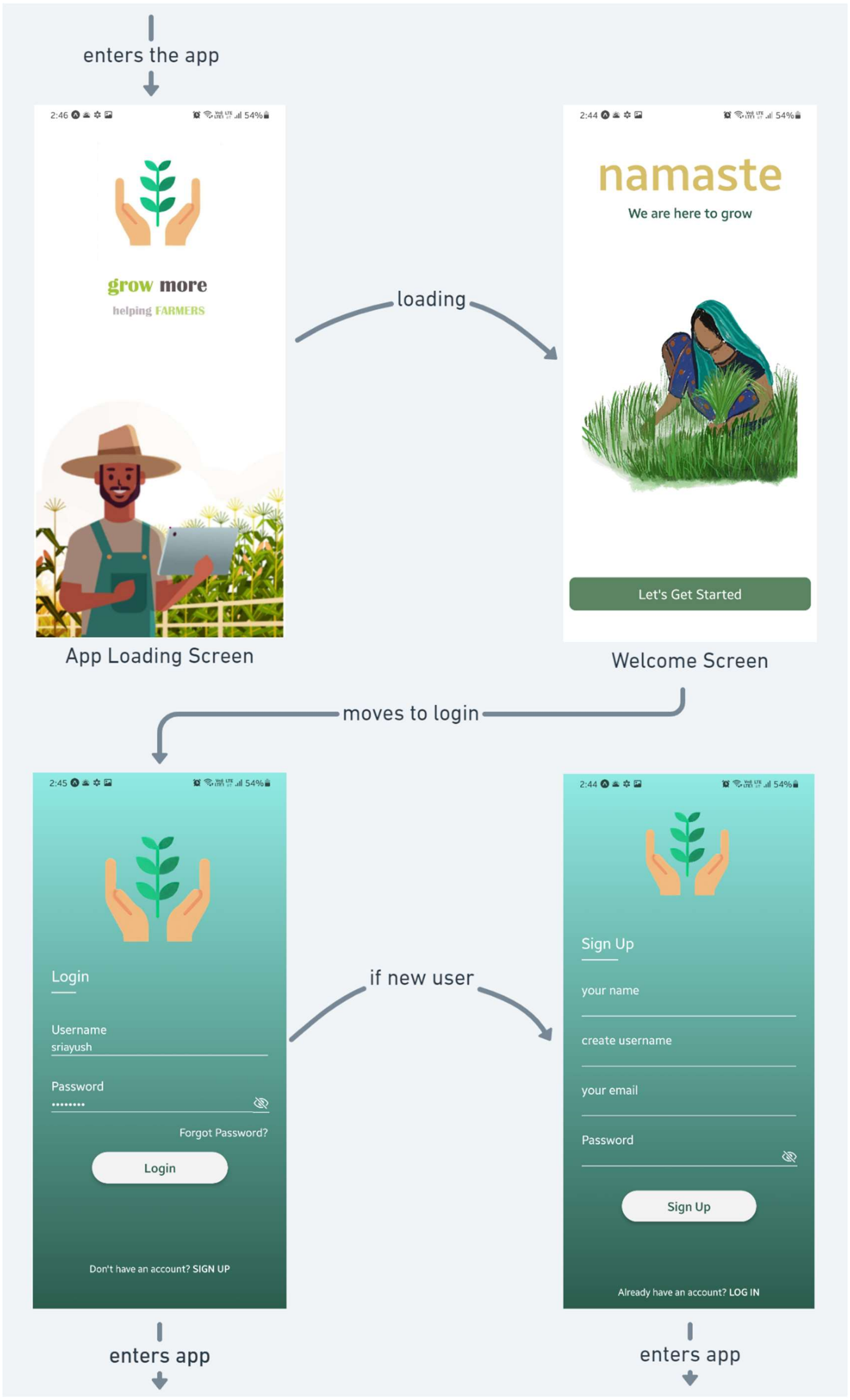


Root file (App.js) contains <AppNavigation /> Component which exports either <StackNav /> or <Auth /> based on value of storage from Context saved in async storage. StackNav and Auth are imported from GrowMoreNavigation.

Auth contains login and signup screens whereas StackNav provides access to our app.

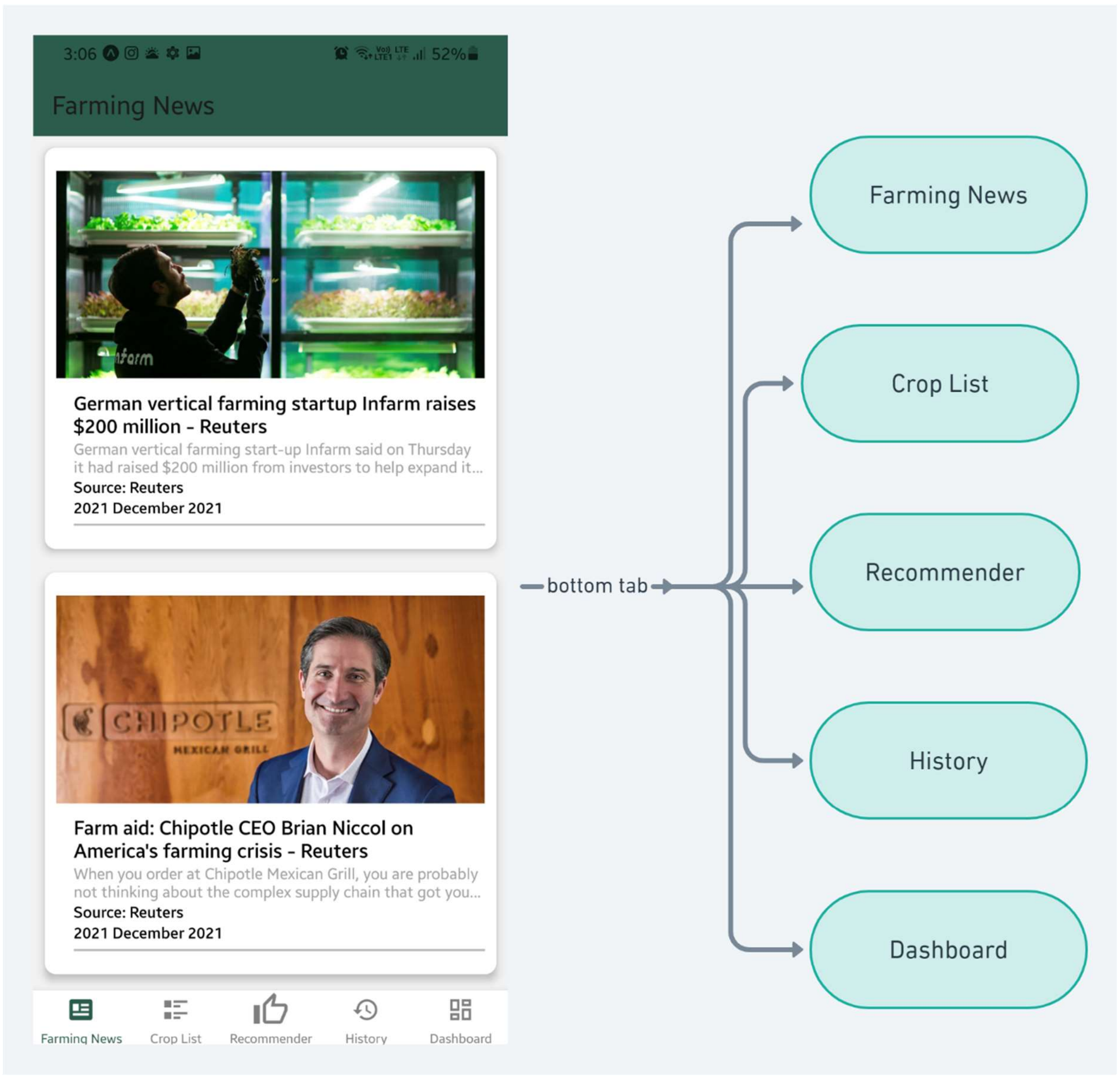


➪ Authentication Screens



Grow More Main

After auth, user is redirected to Home Page of the app which displays Farming News fetched from News.Api. User can navigate via Bottom tab to different Screens like Crop Recommender, Previous History, Crop List and Dashboard. Output of recommender system generated at server is displayed in a Overlay or Modal.



3:06 52%

Farming News

German vertical farming startup Infarm raises $200 million - Reuters
German vertical farming start-up Infarm said on Thursday it had raised $200 million from investors to help expand it...
Source: Reuters
2021 December 2021

Farm aid: Chipotle CEO Brian Niccol on America's farming crisis - Reuters
When you order at Chipotle Mexican Grill, you are probably not thinking about the complex supply chain that got you...
Source: Reuters
2021 December 2021

Deployment

Because we need a mongo database for our service to run.

We have created a database in the mongo cloud. So, instead of creating a database on our local system, we have used MongoDB atlas cloud for setting up the database.

Backend deployment to Heroku

Till now we were running backend locally. Now we need to deploy this to the backend publicly so that it can be used by a publicly deployed frontend.

We have deployed the backend to Heroku:

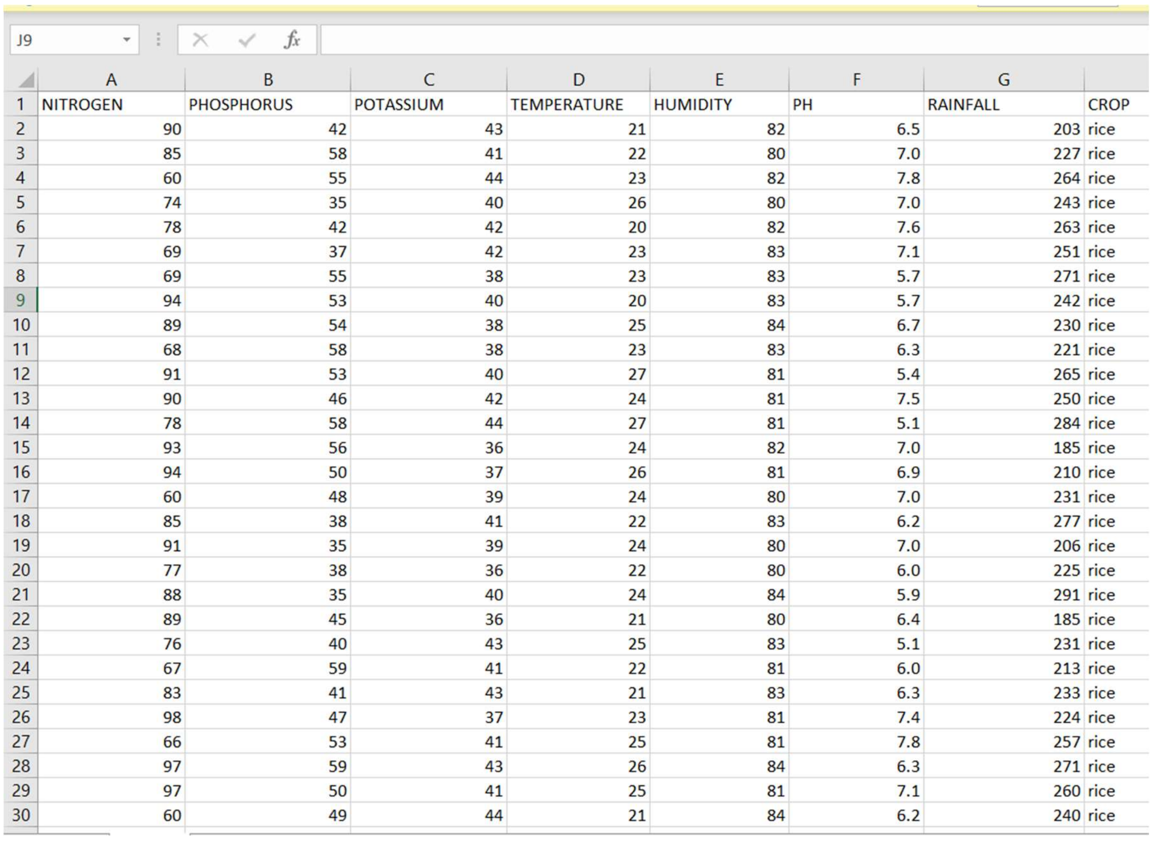
Heroku is a container-based cloud platform as a Service(PaaS).

Developers are using Heroku to deploy, manage, and scale modern apps.

Analysis: Machine Learning

#) About Data Set:

Our data set consists of 2201 number of entries each consisting of 7 number of features namely:
1)Nitrogen
2)Phosphorus
3)Potassium
4)Temperature
5)Humidity
6)PH value
7)Rainfall
These features are our independent variables.
Our target is to predict a suitable "crop" so, this acts as our target variable.
There are no empty values in our data set.
According to our data our problem is a multiclass classifier problem where new entries are to be classified across 22 crops or classes



J9

	A	B	C	D	E	F	G	
1	NITROGEN	PHOSPHORUS	POTASSIUM	TEMPERATURE	HUMIDITY	PH	RAINFALL	CROP
2	90	42	43	21	82	6.5	203	rice
3	85	58	41	22	80	7.0	227	rice
4	60	55	44	23	82	7.8	264	rice
5	74	35	40	26	80	7.0	243	rice
6	78	42	42	20	82	7.6	263	rice
7	69	37	42	23	83	7.1	251	rice
8	69	55	38	23	83	5.7	271	rice
9	94	53	40	20	83	5.7	242	rice
10	89	54	38	25	84	6.7	230	rice
11	68	58	38	23	83	6.3	221	rice
12	91	53	40	27	81	5.4	265	rice
13	90	46	42	24	81	7.5	250	rice
14	78	58	44	27	81	5.1	284	rice
15	93	56	36	24	82	7.0	185	rice
16	94	50	37	26	81	6.9	210	rice
17	60	48	39	24	80	7.0	231	rice
18	85	38	41	22	83	6.2	277	rice
19	91	35	39	24	80	7.0	206	rice
20	77	38	36	22	80	6.0	225	rice
21	88	35	40	24	84	5.9	291	rice
22	89	45	36	21	80	6.4	185	rice
23	76	40	43	25	83	5.1	231	rice
24	67	59	41	22	81	6.0	213	rice
25	83	41	43	21	83	6.3	233	rice
26	98	47	37	23	81	7.4	224	rice
27	66	53	41	25	81	7.8	257	rice
28	97	59	43	26	84	6.3	271	rice
29	97	50	41	25	81	7.1	260	rice
30	60	49	44	21	84	6.2	240	rice

Deployed model

Random forest model has been deployed since it is more robust by nature i.e. similar level of accuracy on both lesser training data set and larger training data set. This is mainly due to the inherent randomness of the model, while growing the trees. Instead of searching for the most important feature while splitting a node, it searches for the best feature among a random subset of features and this also reduces the overfitting of the decision tree.

Authentication

It is very important to secure the endpoint to fetch the user's data. If not secured anyone can read the data easily. Here in this app, we have secured our endpoint via a token authentication mechanism.
Many websites allow users to set a password during sign-up. This password is then used to authenticate the user.
In this app, the steps we have followed to authenticate the user using a password are:

- Store users' passwords which they have entered during sign-up after hashing in MongoDB.
- When a user tries to log in using the password, the hash of this password is matched with the password stored in MongoDB earlier.
- Hashing is a mechanism by which a password in plain text is converted into a jumbled-up string of a specific length.

Token authentication:
It allows access to resources via a token, instead of a password. This is like adding a layer of security over password-based authentication.
Some advantages of tokens are:

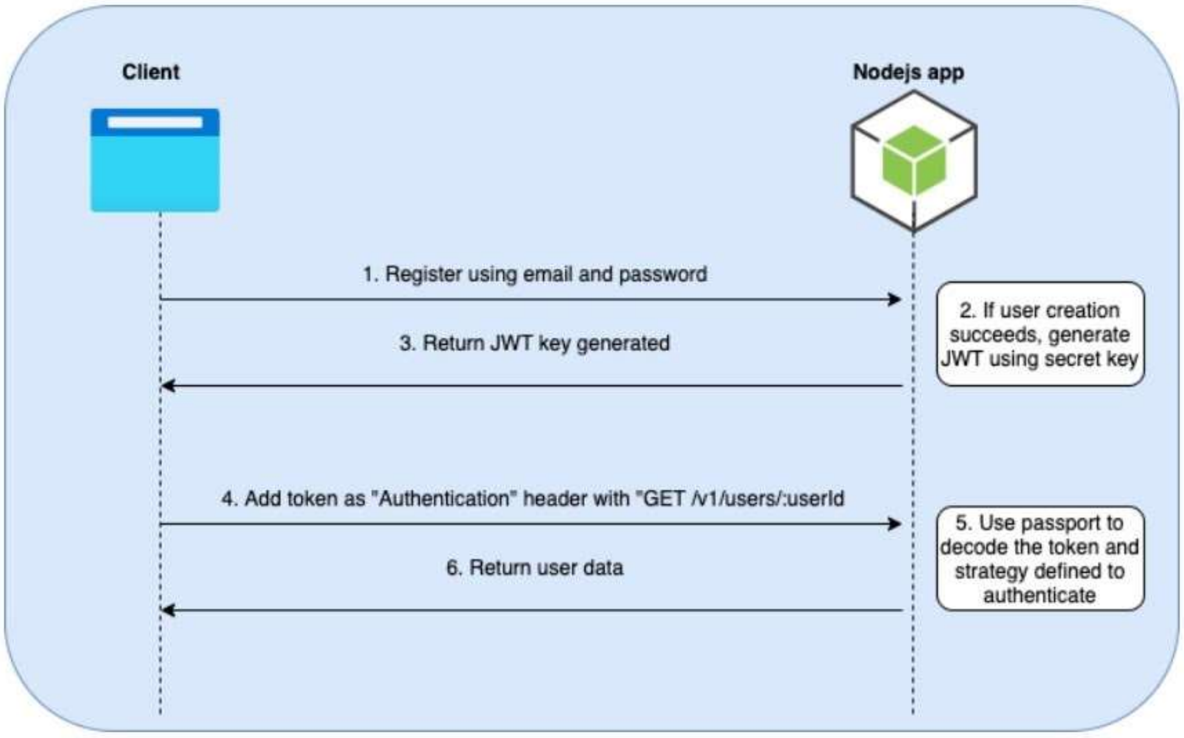
- Tokens can be assigned with an expiration date.
- Tokens can even store some data within it.

A screenshot of code for token generation is below:

On a higher level, token authentication involves:

- Users start with registering or logging in using their email and password.
- Create a token by signing some user-specific data using a secret key.
- Send token to the client and is stored at client side.
- The client sends the token with requests to any secured route.

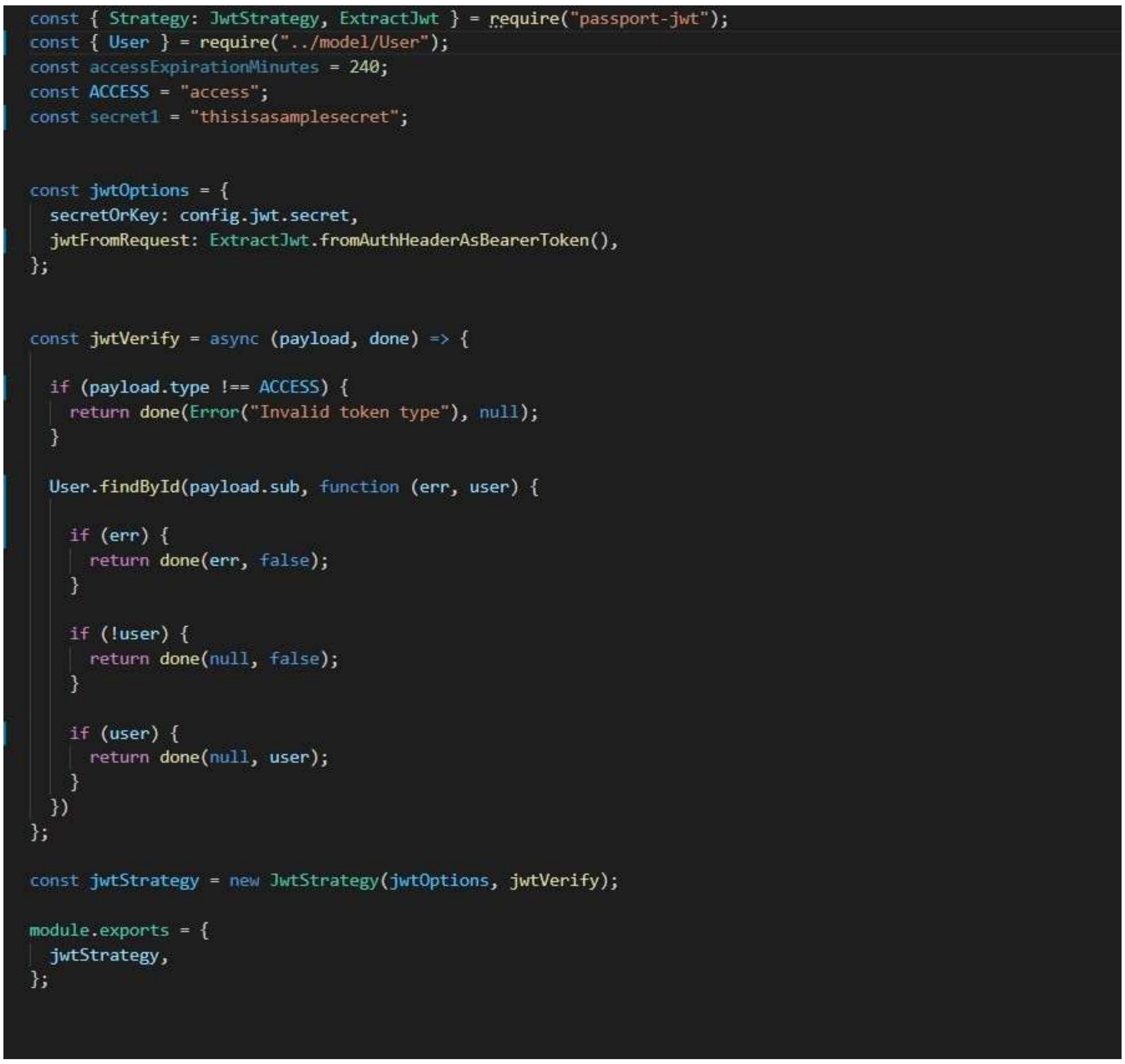
- This token is extracted from the request and decoded. Its signature is only valid if the correct secret key is available.
- If the token is valid, that particular requested action is performed and a specific response is returned.



Client
Nodejs app
1. Register using email and password
2. If user creation succeeds, generate JWT using secret key
3. Return JWT key generated
4. Add token as "Authentication" header with "GET /v1/users/:userId
5. Use passport to decode the token and strategy defined to authenticate
6. Return user data

Configuring and registering the Passport strategy

Until this point we have a mechansim to generate authentication tokens or jwt tokens, now we can utilize it to secure endpoints. Passport is a library which is used to authenticate user requests using custom strategies.
Format used to register a passport strategy with a name.
passport.use("<Name>", <strategy>);



```
const { Strategy: JwtStrategy, ExtractJwt } = require("passport-jwt");
const { User } = require("../model/User");
const accessExpirationMinutes = 240;
const ACCESS = "access";
const secret1 = "thisisasamplesecret";


const jwtOptions = {
  secretOrKey: config.jwt.secret,
  jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),
};


const jwtVerify = async (payload, done) => {

  if (payload.type !== ACCESS) {
    return done(Error("Invalid token type"), null);
  }

  User.findById(payload.sub, function (err, user) {

    if (err) {
      return done(err, false);
    }

    if (!user) {
      return done(null, false);
    }

    if (user) {
      return done(null, user);
    }
  })
};

const jwtStrategy = new JwtStrategy(jwtOptions, jwtVerify);

module.exports = {
  jwtStrategy,
};
```