

Project ON

Personal Recipe Book

BACHELOR OF TECHNOLOGY IN
COMPUTER SCIENCE AND
ENGINEERING

Name of the student: Aarushi

Registration number: 11701801

Roll number: 11

Section: km013



School of Computer Science and Engineering

Lovely Professional University Phagwara ,
Punjab (India)

Pantry Pal

It is a web application designed for users to search through a collection of recipes collected through the web, matching based on their ingredients.

Technologies used:

- Frontend: Angular
- Backend: Node/Express
- Database: MongoDB

Features

Ingredient Search

Users are able to add ingredients in their house. The results are updated every time the user adds or deletes an ingredient from their list. Recipes are sorted based on the least ingredients they are missing.

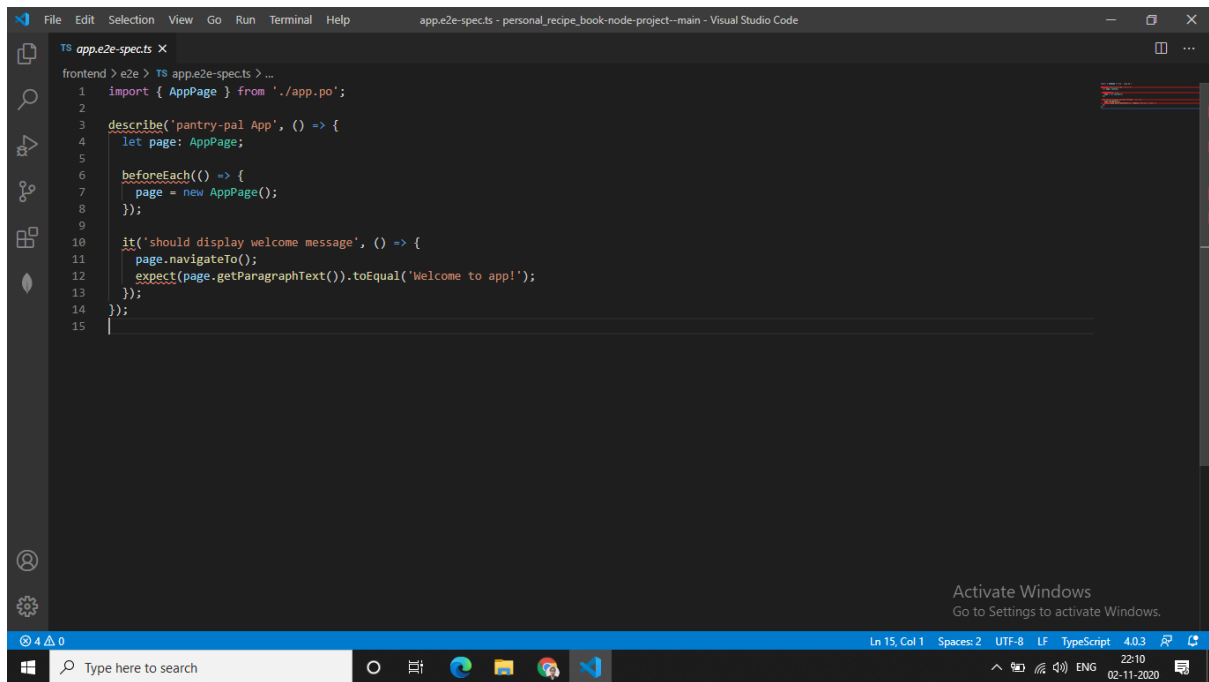
Backend API

One concern during the creation of our web application was the security of the API key. The JSON Web Token key was already stored securely in the backend as a environment variable.

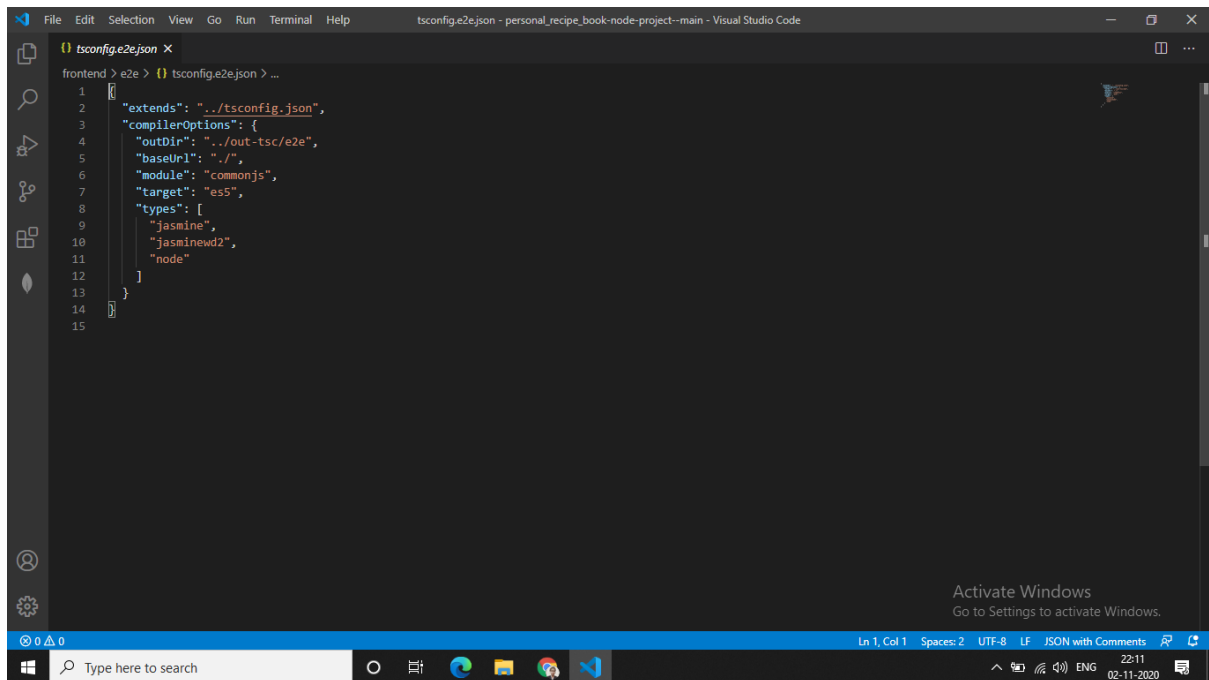
Originally, the Angular frontend handled the external API requests, but risked malicious users grabbing the API key for their own purposes. The application was then refactored to use a backend route which has access to `process.env` variables to keep the API key safe. Because GET requests do not normally carry bodies, recipe IDs and ingredients were sent through wild card params and queries.

Github Link - https://github.com/aarushi04/personal_recipe_book-node-project-

Code Snippets



```
1 import { AppPage } from './app.po';
2
3 describe('pantry-pal App', () => {
4   let page: AppPage;
5
6   beforeEach(() => {
7     page = new AppPage();
8   });
9
10  it('should display welcome message', () => {
11    page.navigateTo();
12    expect(page.getParagraphText()).toEqual('Welcome to app!');
13  });
14 });
15
```



```
1 {
2   "extends": "../tsconfig.json",
3   "compilerOptions": {
4     "outDir": "../out-tsc/e2e",
5     "baseUrl": ".",
6     "module": "commonjs",
7     "target": "es5",
8     "types": [
9       "jasmine",
10      "jasminewd2",
11      "node"
12    ]
13  }
14 }
15
```

```
1 <!doctype html>
2 <html lang="en">
3 <head>
4 <!-- Global site tag (gtag.js) - Google Analytics -->
5 <script async src="https://www.googletagmanager.com/gtag/js?id=UA-114133598-1"></script>
6 <script>
7   window.dataLayer = window.dataLayer || [];
8   function gtag(){dataLayer.push(arguments);}
9   gtag('js', new Date());
10
11   gtag('config', 'UA-114133598-1');
12 </script>
13 <script async src="https://www.googletagmanager.com/gtag/js?id=UA-114286597-1"></script>
14 <script>
15   window.dataLayer = window.dataLayer || [];
16   function gtag() { dataLayer.push(arguments); }
17   gtag('js', new Date());
18
19   gtag('config', 'UA-114286597-1');
20 </script>
21 <script async src="https://www.googletagmanager.com/gtag/js?id=UA-115560858-2"></script>
22 <script>
23   window.dataLayer = window.dataLayer || [];
24   function gtag() { dataLayer.push(arguments); }
25   gtag('js', new Date());
26
27   gtag('config', 'UA-115560858-2');
28 </script>
29
30 <meta charset="utf-8">
31 <title>Pantry Pal</title>
32 <base href="/">
33
```

```
25 vertical-align: baseline;
26 font-family: 'Roboto', sans-serif, Helvetica, Arial;
27
28 /* HTML5 display-role reset for older browsers */
29 article, aside, details, figcaption, figure,
30 footer, header, hgroup, menu, nav, section {
31   display: block;
32 }
33 body {
34   /* line-height: 1; */
35 }
36 ol, ul {
37   list-style: none;
38 }
39 blockquote, q {
40   quotes: none;
41 }
42 blockquote:before, blockquote:after,
43 q:before, q:after {
44   content: '';
45   content: none;
46 }
47 table {
48   border-collapse: collapse;
49   border-spacing: 0;
50 }
51 button:focus{
52   outline: none;
53 }
54 h1:focus{
55   outline: none;
56 }
```

This screenshot shows the Visual Studio Code editor with the file `tests.ts` open. The Explorer sidebar on the left shows the project structure, with `tests.ts` selected under the `src` directory. The main editor area displays the following TypeScript code:

```
1 // This file is required by karma.conf.js and loads recursively all the .spec and framework files
2
3 import 'zone.js/dist/zone-testing';
4 import { getTestBed } from '@angular/core/testing';
5 import {
6   BrowserDynamicTestingModule,
7   platformBrowserDynamicTesting
8 } from '@angular/platform-browser-dynamic/testing';
9
10 declare const require: any;
11
12 // First, initialize the Angular testing environment.
13 getTestBed().initTestEnvironment(
14   BrowserDynamicTestingModule,
15   platformBrowserDynamicTesting()
16 );
17 // Then we find all the tests.
18 const context = require.context('./', true, /\.spec\.ts$/);
19 // And load the modules.
20 context.keys().map(context);
21
```

The status bar at the bottom indicates the file is at Line 1, Column 1, using UTF-8 encoding and LF line endings. The system tray shows the date as 02-11-2020.

This screenshot shows the Visual Studio Code editor with the file `tsconfig.app.json` open. The Explorer sidebar on the left shows the project structure, with `tsconfig.app.json` selected. The main editor area displays the following JSON configuration:

```
1 {
2   "extends": "../tsconfig.json",
3   "compilerOptions": {
4     "outDir": "../out-tsc/app",
5     "baseUrl": "./",
6     "module": "es2015",
7     "types": []
8   },
9   "exclude": [
10    "test.ts",
11    "**/*.spec.ts"
12  ]
13 }
14
```

The status bar at the bottom indicates the file is at Line 1, Column 1, using UTF-8 encoding and LF line endings. The system tray shows the date as 02-11-2020.

```
frontend > src > app > landing page > # landing-page.component.css > landing-page
16 url(https://images.unsplash.com/photo-156376849114-976667568b02?ixlib=rb-0.3.5&ixid=eyJhcHBfawQ1OjEYMDd9&s=8348
17 background-size: cover;
18 }
19
20 .content-container {
21   width: 60%;
22   display: flex;
23   align-items: center;
24   flex-direction: column;
25 }
26
27 .title-and-body {
28   display: flex;
29   align-items: center;
30   flex-direction: column;
31 }
32
33 #welcome {
34   font-family: 'Open Sans Condensed', sans-serif;
35   font-weight: 100;
36   font-size: 15px;
37   font-weight: 100;
38   padding-bottom: 2px;
39   border-bottom: 1px solid white;
40 }
41
42 #title {
43   font-family: 'Roboto Slab', serif;
44   font-size: 100px;
45   font-weight: 700;
46   line-height: 100px;
47 }
48 }
```

```
frontend > src > app > login > # login.component.css > login-modal
8 min-width: 500px;
9 min-height: 300px;
10 z-index: 2000;
11 visibility: hidden;
12 backface-visibility: hidden;
13 transform: translateX(-50%) translateY(-50%);
14 }
15
16 .md-show {
17   visibility: visible;
18 }
19
20 .md-overlay {
21   position: fixed;
22   width: 100%;
23   height: 100%;
24   visibility: hidden;
25   top: 0;
26   left: 0;
27   z-index: 1000;
28   opacity: 0;
29   background: rgba(50,50,50,0.8);
30   transition: all 0.3s;
31 }
32
33 .md-show ~ .md-overlay {
34   opacity: 1;
35   visibility: visible;
36 }
37
38 .md-perspective,
39 .md-perspective body {
40   height: 100%;
41 }
```

```
login.component.html X
1 <div class="login-modal md-effect-1" id="modal-2">
2   <div class="md-content">
3     <button class="close-button" (click)="closeModal()">X</button>
4     <button class="demo-login" (click)="demoLogin()">Demo</button>
5
6     <p>Log In!</p>
7
8     <form class="login-form" (submit)="handleLogin()">
9       <div class="login-username-total">
10        <label id="login-username">Username:</label>
11        <input id="login-username-input" type="text" [(ngModel)]="username" name="username" />
12      </div>
13      <div class="login-password-total">
14        <label id="login-password">Password:</label>
15        <input id="login-password-input" type="password" [(ngModel)]="password" name="password" />
16      </div>
17      <input id="login-button" type="submit" value="Log In" />
18    </form>
19    <p id="login-error">{{error}}</p>
20  </div>
21</div>
22
23<div class="md-overlay" (click)="closeModal()"></div>
24
```

```
# navbar.component.css X
1 .nav {
2   display: flex;
3   justify-content: space-between;
4   height: 50px;
5   width: 100vw;
6   box-shadow: 0px 2px 25px 0px #90909075;
7   z-index: 10;
8   position: fixed;
9   top: 0;
10  background-color: #FFF2DD;
11}
12
13.nav-title {
14  font-size: 25px;
15  font-weight: 700;
16  color: #659DBD;
17  margin: auto 0px auto 25px;
18  cursor: pointer;
19}
20
21.nav-buttons {
22  margin: auto 25px auto 0px;
23}
24
25.nav-button {
26  cursor: pointer;
27  height: 25px;
28  color: #FFFFFF;
29  background-color: #659DBD;
30  border: 1px solid #FFFFFF;
31  /* box-shadow: 0px 0px 2px 2px #FFFFFF; */
32  border-radius: 10px;
33  margin-right: 5px;

```

The screenshot shows the Visual Studio Code editor with the file explorer on the left displaying the project structure. The active file is `recipe-details.component.css`. The code defines styles for a button, a recipe detail container, a sidebar, a title, and an image.

```
# recipe-details.component.css
1 p {
2   font-weight: 400;
3   font-size: 18px;
4   color: #232323;
5   margin: 15px 0px;
6 }
7
8 button {
9   cursor: pointer;
10 }
11
12 .recipe-detail {
13   margin-top: 50px;
14   display: flex;
15   overflow-y: scroll;
16   min-height: calc(100vh - 50px);
17 }
18
19 .recipe-sidebar {
20   width: 400px;
21   box-shadow: 2px 0px 10px 0px #90909075;
22   background: #FBECE125;
23 }
24
25 .recipe-title {
26   font-weight: 700;
27   font-size: 45px;
28   margin: 15px 30px;
29   color: #60708B;
30 }
31
32 .recipe-img {
33   margin: 15px 30px;
```

The screenshot shows the Visual Studio Code editor with the file explorer on the left. The active file is `recipe-details.component.html`. The code is an Angular template that renders the recipe details, including a sidebar, main content area, and additional information.

```
recipe-details.component.html
1 <div class="recipe-detail" *ngIf="recipe">
2   <div class="recipe-sidebar">
3     <h2 class="recipe-title">{{ recipe.title | uppercase }}</h2>
4     
5     <ul class="ingredients-list">
6       <li class="ingredient-item" *ngFor="let ingredient of recipe.extendedIngredients">
7         
8         <p>{{ ingredient.originalString }}</p>
9       </li>
10    </ul>
11  </div>
12  <div class="recipe-main">
13    <h3 class="directions-title">Directions</h3>
14    <ul *ngIf="recipe.instructions === null">
15      <p>Oops... Directions could not be found.</p>
16    </ul>
17    <ul *ngFor="let instructions of recipe.analyzedInstructions">
18      <li *ngFor="let steps of instructions.steps">
19        <p>{{ steps.step }}</p>
20      </li>
21    </ul>
22    <p>For more details, visit <a href="{{ recipe.sourceUrl }}" target="_blank">{{ recipe.sourceName }}</a></p>
23    <button (click)="goBack()">go back</button>
24  </div>
25  <div class="additional-info">
26    <ul class="nutrition-list">
27      <li>Nutrition Information</li>
28      <p>Vegetarian: {{ recipe.vegetarian }}</p>
29      <p>Vegan: {{ recipe.vegan }}</p>
30      <p>Gluten Free: {{ recipe.glutenFree }}</p>
31      <p>Dairy Free: {{ recipe.dairyFree }}</p>
32      <p>Healthy Choice: {{ recipe.veryHealthy }}</p>
33      <p>Weight Watcher Smart Points: {{ recipe.weightWatcherSmartPoints }}</p>
```


The screenshot shows the Visual Studio Code interface with the file explorer on the left displaying a project structure. The 'server' directory is expanded, showing 'models' and 'routes'. The 'models' directory is selected, and the 'user.js' file is open in the editor. The code defines a Mongoose schema for a user with fields for username, password, and ingredients. It includes validation rules and a unique validator for the username. The terminal at the bottom shows the command 'server > models > JS user.js > ...'.

```
server > models > JS user.js > ...
1 const mongoose = require('mongoose');
2 const uniqueValidator = require('mongoose-unique-validator');
3 const _ = require('lodash');
4 const bcrypt = require('bcryptjs');
5 const jwt = require('jsonwebtoken');
6
7 const UserSchema = new mongoose.Schema({
8   username: {
9     type: String,
10    required: [true, 'Username cannot be blank'],
11    maxlength: [12, 'Username cannot be greater than 12 characters'],
12    trim: true,
13    unique: true
14  },
15  password: {
16    type: String,
17    trim: true,
18    required: [true, 'Password cannot be blank'],
19    minlength: [6, 'Password cannot be less than 6 characters']
20  },
21  ingredients: {
22    type: Array,
23  }
24 });
25
26 UserSchema.plugin(uniqueValidator, { message: 'Username already exists' });
27
28 // Instance methods
29 UserSchema.methods.toJSON = function () {
30   const user = this;
31   const userObject = user.toObject();
32
33   return _.pick(userObject, ['id', 'username', 'ingredients']);
```

The screenshot shows the Visual Studio Code interface with the file explorer on the left displaying a project structure. The 'server' directory is expanded, showing 'models' and 'routes'. The 'routes' directory is selected, and the 'recipes.js' file is open in the editor. The code defines Express routes for finding recipes by ingredients and getting a recipe by ID. It includes axios for API calls and error handling. The terminal at the bottom shows the command 'server > routes > JS recipes.js > ...'.

```
server > routes > JS recipes.js > ...
1 const express = require('express');
2 const router = express.Router();
3 const axios = require('axios');
4 const _ = require('lodash');
5
6 const request = axios.create({
7   baseURL: 'https://spoonacular-recipe-food-nutrition-v1.p.mashape.com/recipes/',
8   headers: {
9     'Accept': 'application/json',
10    'X-Mashape-Key': process.env.SPOON || 'getyourownapikeyatmashape'
11  }
12 });
13
14 // Get recipes
15 router.get('/findByIngredients', (req, res) => {
16   request.get(`findByIngredients?fillIngredients=false&ingredients=${req.query.ingredients}&limitLicense=true&number`
17     .then(recipes => res.send(recipes.data))
18     .catch(e => res.status(400).json({
19       message: 'Request to Spoonacular failed/unauthorized'
20     }));
21 });
22
23 router.get('/:id', (req, res) => {
24   request.get(`${req.params.id}/information?includeNutrition=true`)
25     .then(recipe => res.send(recipe.data))
26     .catch(e => res.status(400).json({
27       message: 'Request to Spoonacular failed/unauthorized'
28     }));
29 });
30
31 module.exports = router;
```

The screenshot shows the Visual Studio Code editor with the file `users.js` open. The Explorer sidebar on the left shows the project structure, with `users.js` selected under the `server` directory. The main editor area displays the following JavaScript code:

```
10
11 const authenticate = passport.authenticate('jwt', { session: false });
12
13 // Signup
14 router.post('/', (req, res) => {
15   const body = _.pick(req.body, ['username', 'password']);
16   const newUser = new User(body);
17   const existingUser = User.find({ username: body.username });
18
19   newUser.save().then(() => {
20     const user = newUser.toJSON();
21     const token = newUser.generateToken();
22     res.json({
23       user,
24       token
25     });
26   }).catch((e) => {
27     res.status(400).send(e);
28   });
29 });
30
31 router.post('/login', (req, res) => {
32   const body = _.pick(req.body, ['username', 'password']);
33
34   User.findByCredentials(body.username, body.password).then(user => {
35     const token = user.generateToken();
36
37     res.json({
38       user,
39       token
40     });
41   }).catch(() => {
42     res.status(401).json({
```

The status bar at the bottom indicates the file is at line 1, column 1, with 2 spaces, in UTF-8 encoding, using the LF line ending, and is a JavaScript file. The system tray shows the date and time as 22:14 on 02-11-2020.

The screenshot shows the Visual Studio Code editor with the file `passport.js` open. The Explorer sidebar on the left shows the project structure, with `passport.js` selected under the `server` directory. The main editor area displays the following JavaScript code:

```
1 const JwtStrategy = require('passport-jwt').Strategy;
2 const ExtractJwt = require('passport-jwt').ExtractJwt;
3
4 const User = require('../models/user');
5
6 const jwtAuth = passport => {
7   let opts = {};
8   opts.jwtFromRequest = ExtractJwt.fromAuthHeaderWithScheme('jwt');
9   opts.secretOrKey = process.env.JWT_TOKEN || 'supersecretkey';
10  passport.use(new JwtStrategy(opts, (jwtPayload, done) => {
11    User.findById(jwtPayload._id).then(user => {
12      return done(null, user);
13    }).catch(() => {
14      return done(null, false);
15    });
16  }));
17 };
18
19 module.exports = jwtAuth;
```

The status bar at the bottom indicates the file is at line 1, column 1, with 2 spaces, in UTF-8 encoding, using the LF line ending, and is a JavaScript file. The system tray shows the date and time as 22:15 on 02-11-2020.

