

MARATHA VIDYA PRASARAK SAMAJ'S

KARMAVEER ADV. BABURAO GANPATRAO THAKARE
COLLEGE OF ENGINEERING,
NASHIK-422013

DEPARTMENT OF INFORMATION TECHNOLOGY
SEMESTER VI [2021-22]



WEB APPLICATION DEVELOPMENT (LP-II)
MINI PROJECT ON

Topic name: “ Nashik Food Web”

Submitted by

1. **Pranjal Anil Kunde (27)**
2. **Shravani Ashokrao Lajurkar (28)**

Table of Contents

- 1. ABSTRACT**
- 2. INTRODUCTION**
 - a. Problem definition**
 - b. Block Diagram**
- 3. SPECIFIC REQUIRMENTS**
- 4. IMPLEMNTATION**
- 5. RESULT**
- 6. DEPLOYMENT ON AWS (steps)**

Abstract

The project entitled “NASHIK FOOD WEB” has been designed to fill a specific niche in the market by providing home delivery management with the ability to offer the customers an online ordering option without having to invest large amounts of time and money in having custom website designed specifically for them. The system which is highly customizable, allows the restaurant employees to easily manage the site content, most importantly the menu, themselves through a very intuitive graphical interface. The website, which is only component seen by the restaurant customers, is then built dynamically based on the current state of the system, so any changes made are reflected in real time.

Visitors to the site, once registered, are then able to easily navigate this menu, add food items to their order, specify delivery options with only a few clicks, greatly simplifying the ordering process.

INTRODUCTION

Nashik Food Web is mainly designed primarily function for use in the food delivery industry. This system will allow the hotels and restaurants to increase online food ordering which will help the customers to select the food menu items just in few minutes. In this modern generation, Online food ordering is a mobility of food delivery or takeout from a local restaurant. As now there is rapid growth in the use of internet and the technologies associated with it, several opportunities are coming up on the web or mobile application. This is made possible through the use of electronic payment system. In addition, it can also provide efficiency for the restaurant by reducing human errors and minimizing the time.

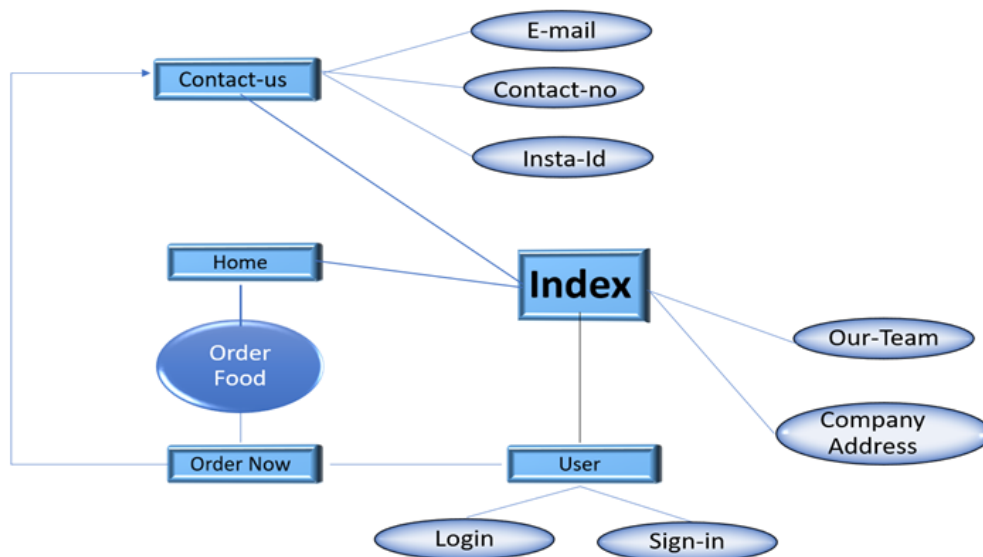
"Nashik Food Website " is considered as an increasingly used application in restaurant management. Just a few years ago, people used to spend long time waiting for a table to be ready for them at a restaurant. And after that, they would wait longer time to order their food. There was no surprise that the food took too long to be prepared. In short, the whole enjoyment of eating did eventually result in boredom and tiredness for customers. From another side, this is the restaurant management and staff who used to stand much pressure to handle a lot of customers at the same time. All in all, both customer and menu information management as well as report are the very wonderful aspects of the modern world in restaurant industry. Technically, Prototyping Model is what is applied in Online Website. Visual Studio 2013, and Microsoft Office software's are used to develop the project.

2.1 Problem definition

Problem Statement: Developing an Efficient and User-Friendly Online Food Ordering Website

Description: The objective of this project is to create an online food ordering website that addresses the challenges and shortcomings of existing platforms, providing a seamless and satisfactory experience for users. The current online food ordering landscape is marred by various issues, such as complex user interfaces, long delivery times, limited restaurant choices, and inadequate customer support.

2.2 Block Diagram



SPECIFIC REQUIRMENTS

Hardware requirements

Processor: - Any processor above i3.

Memory: - 4 GB Ram.

Hard-disk: - 512 GB.

Display: Minimum: 1280 x 1024 24-bit

Software Requirements

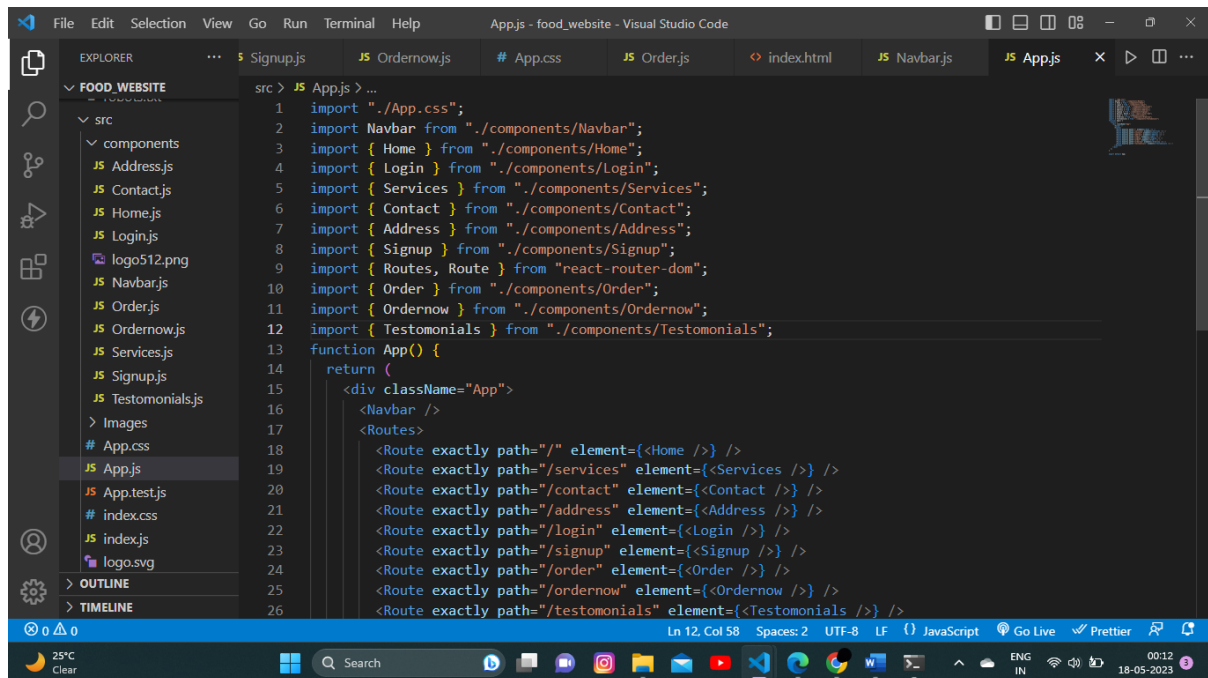
Operation System: Windows or Ubuntu

Coding Language: HTML,CSS,BOOTSTRAP,Javascript,ReactCoding

IDE: VS code

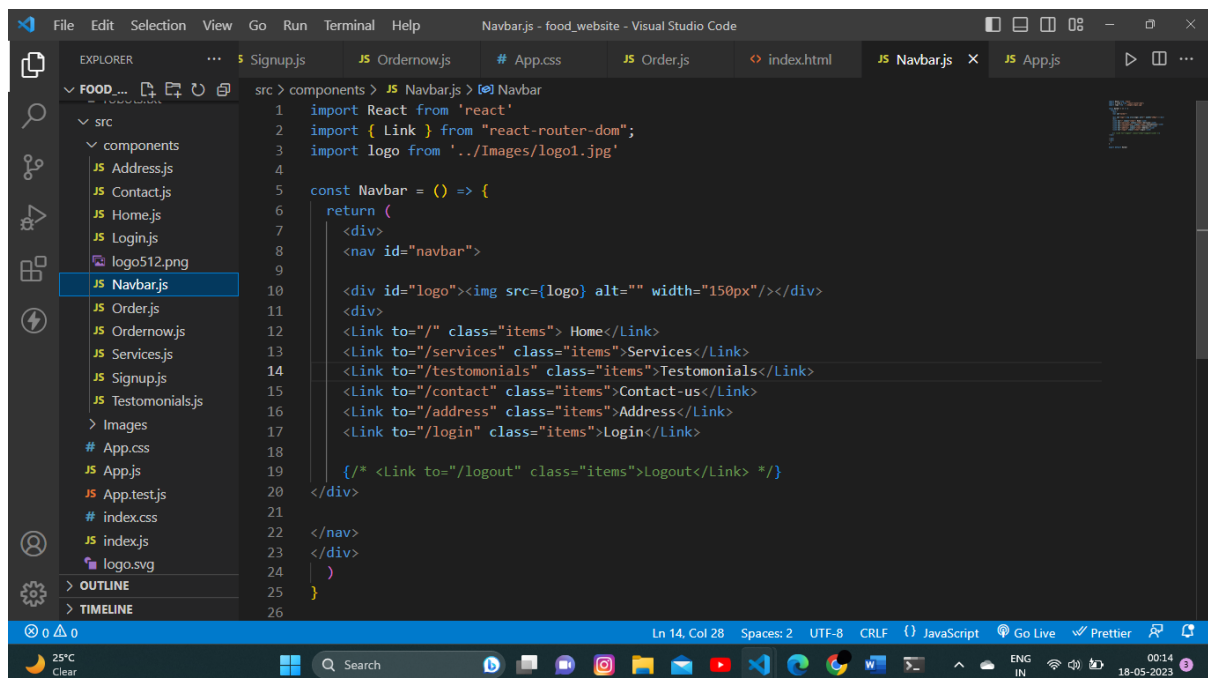
IMPLEMENTATION

Frontend:



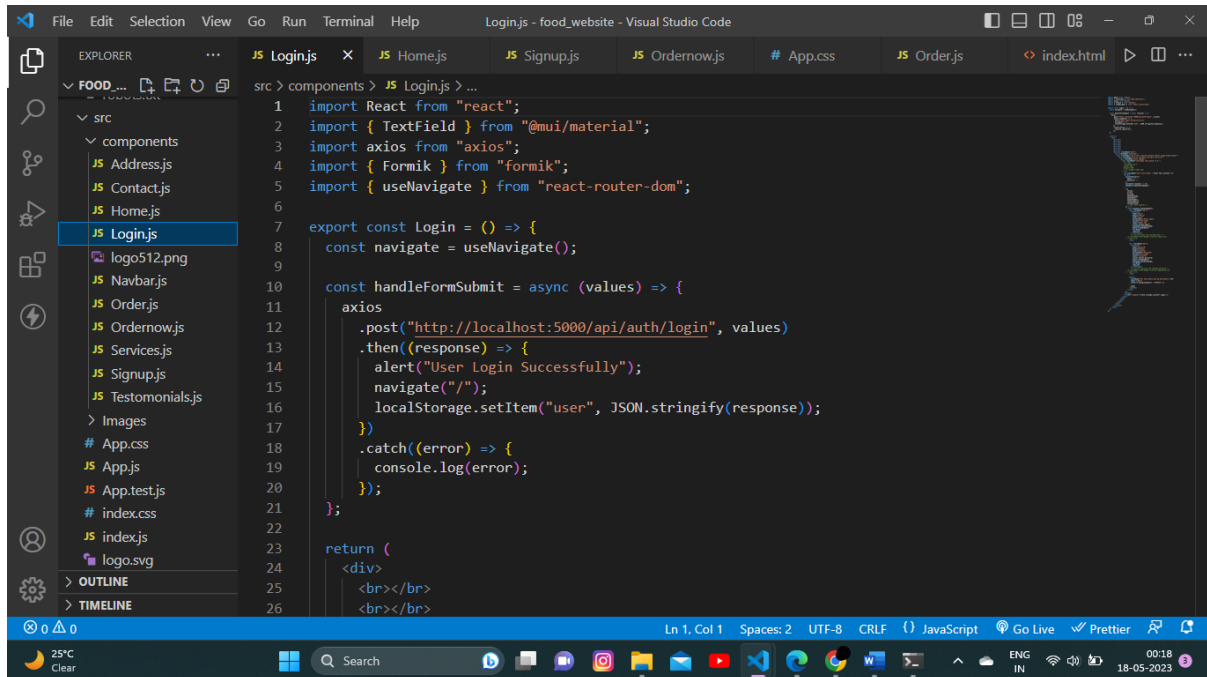
```
1 import './App.css';
2 import Navbar from './components/Navbar';
3 import { Home } from './components/Home';
4 import { Login } from './components/Login';
5 import { Services } from './components/Services';
6 import { Contact } from './components/Contact';
7 import { Address } from './components/Address';
8 import { Signup } from './components/Signup';
9 import { Routes, Route } from 'react-router-dom';
10 import { Order } from './components/Order';
11 import { Ordernow } from './components/Ordernow';
12 import { Testimonials } from './components/Testimonials';
13 function App() {
14   return (
15     <div className="App">
16       <Navbar />
17       <Routes>
18         <Route exactly path="/" element={<Home />} />
19         <Route exactly path="/services" element={<Services />} />
20         <Route exactly path="/contact" element={<Contact />} />
21         <Route exactly path="/address" element={<Address />} />
22         <Route exactly path="/login" element={<Login />} />
23         <Route exactly path="/signup" element={<Signup />} />
24         <Route exactly path="/order" element={<Order />} />
25         <Route exactly path="/ordernow" element={<Ordernow />} />
26         <Route exactly path="/testimonials" element={<Testimonials />} />
```

Fig 1:App.js



```
1 import React from 'react'
2 import { Link } from 'react-router-dom';
3 import logo from '../Images/logo1.jpg'
4
5 const Navbar = () => {
6   return (
7     <div>
8       <nav id="navbar">
9
10        <div id="logo"><img src={logo} alt="" width="150px"/></div>
11        <div>
12          <Link to="/" class="items"> Home</Link>
13          <Link to="/services" class="items">Services</Link>
14          <Link to="/testimonials" class="items">Testimonials</Link>
15          <Link to="/contact" class="items">Contact-us</Link>
16          <Link to="/address" class="items">Address</Link>
17          <Link to="/login" class="items">Login</Link>
18
19          { /* <Link to="/logout" class="items">Logout</Link> */ }
20        </div>
21      </nav>
22    </div>
23  )
24 }
25
26
```

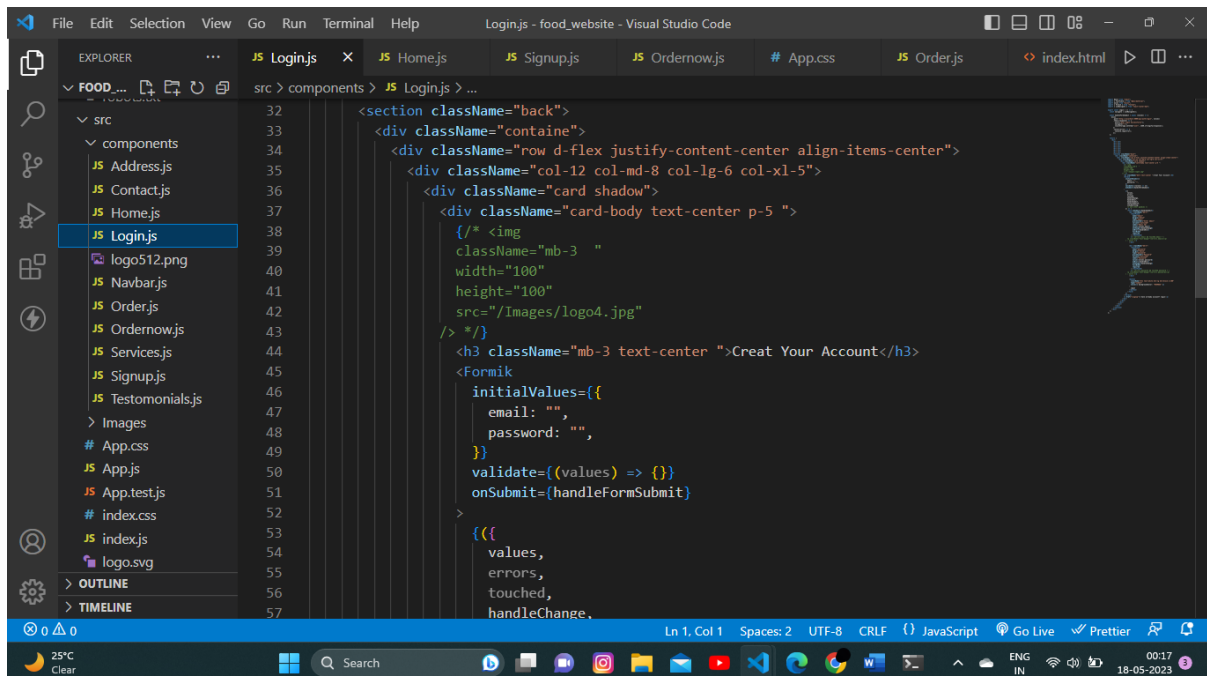
Fig2:Navbar.js



The screenshot shows the Visual Studio Code editor with the 'Login.js' file open. The file is located in the 'src > components' directory. The code defines a 'Login' function that uses 'react', '@mui/material', 'axios', 'formik', and 'react-router-dom'. It includes a 'handleFormSubmit' function that sends a POST request to 'http://localhost:5000/api/auth/login'. The status bar at the bottom indicates 'Ln 1, Col 1', 'Spaces: 2', 'UTF-8', 'CRLF', 'JavaScript', 'Go Live', and 'Prettier'.

```
1 import React from "react";
2 import { TextField } from "@mui/material";
3 import axios from "axios";
4 import { Formik } from "formik";
5 import { useNavigate } from "react-router-dom";
6
7 export const Login = () => {
8   const navigate = useNavigate();
9
10  const handleFormSubmit = async (values) => {
11    axios
12      .post("http://localhost:5000/api/auth/login", values)
13      .then((response) => {
14        alert("User Login Successfully");
15        navigate("/");
16        localStorage.setItem("user", JSON.stringify(response));
17      })
18      .catch((error) => {
19        console.log(error);
20      });
21  };
22
23  return (
24    <div>
25      <br></br>
26      <br></br>
```

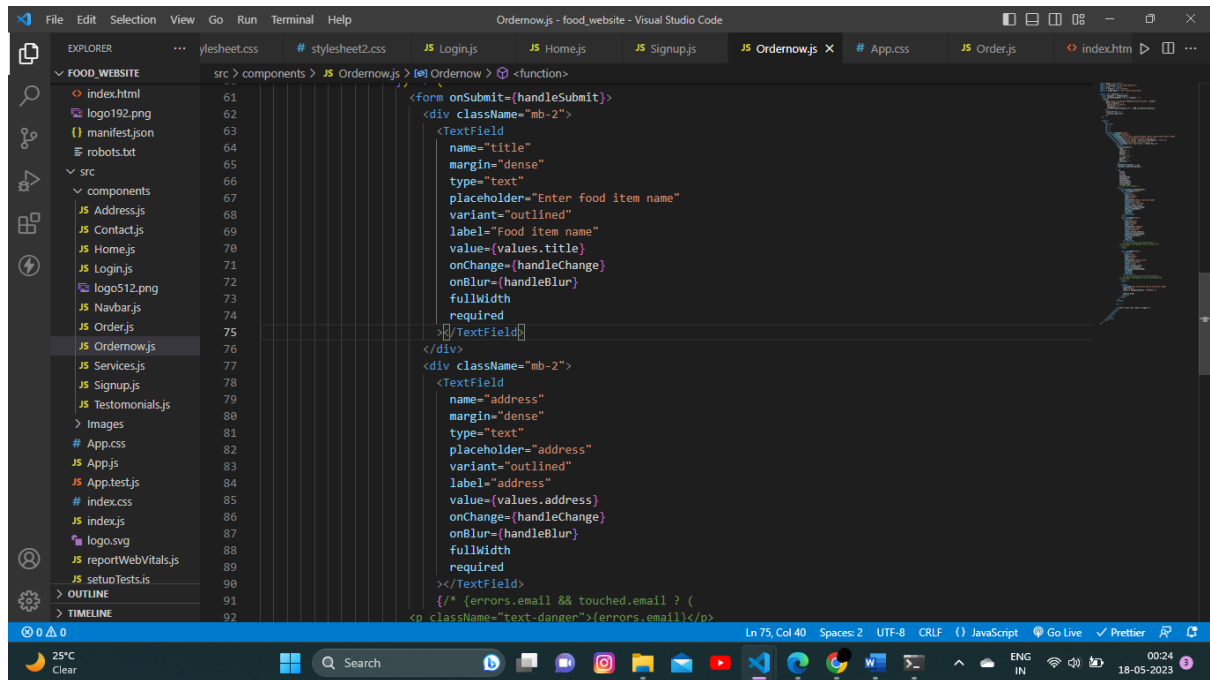
Fig 3:Login.js(a)



The screenshot shows the Visual Studio Code editor with the 'Login.js' file open, displaying the JSX part of the code. The code uses 'classnames' for styling and 'formik' for form handling. It includes a 'Create Your Account' button and a form with 'email' and 'password' fields. The status bar at the bottom indicates 'Ln 1, Col 1', 'Spaces: 2', 'UTF-8', 'CRLF', 'JavaScript', 'Go Live', and 'Prettier'.

```
32 <section className="back">
33   <div className="containe">
34     <div className="row d-flex justify-content-center align-items-center">
35       <div className="col-12 col-md-8 col-lg-6 col-xl-5">
36         <div className="card shadow">
37           <div className="card-body text-center p-5">
38             /*  */
44             <h3 className="mb-3 text-center">Creat Your Account</h3>
45             <Formik
46               initialValues={{
47                 email: "",
48                 password: "",
49               }}
50               validate={values => {}}
51               onSubmit={handleFormSubmit}
52             >
53               {({
54                 values,
55                 errors,
56                 touched,
57                 handleChange,
```

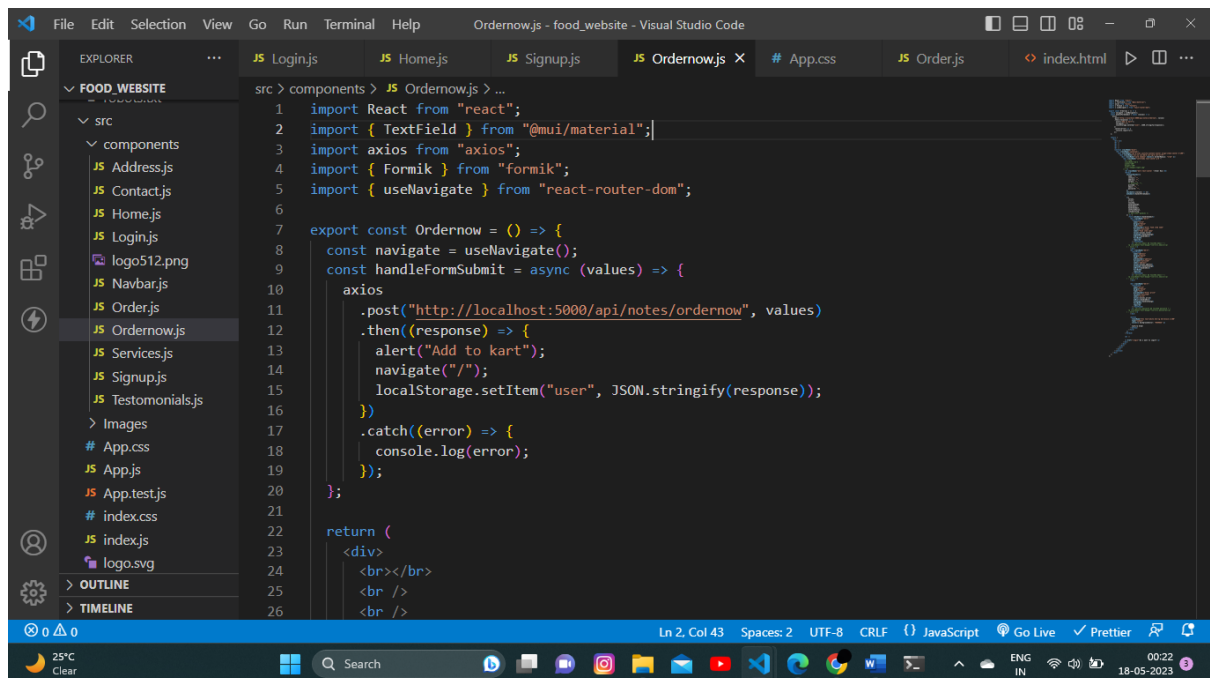
Fig 4:Login.js(b)



This screenshot shows the Visual Studio Code editor with the file `OrderNow.js` open. The Explorer sidebar on the left shows the project structure for `FOOD_WEBSITE`, including files like `index.html`, `manifest.json`, `src`, `components`, `App.css`, `App.js`, `App.test.js`, `index.css`, `index.js`, `logo.svg`, `reportWebVitals.js`, `setupTests.js`, and `Timeline`. The main editor area displays the following JSX code for the `OrderNow` component:

```
src > components > JS OrderNow.js > <function>
61 <form onSubmit={handleSubmit}>
62 <div className="mb-2">
63 <TextField
64   name="title"
65   margin="dense"
66   type="text"
67   placeholder="Enter food item name"
68   variant="outlined"
69   label="Food item name"
70   value={values.title}
71   onChange={handleChange}
72   onBlur={handleBlur}
73   fullWidth
74   required
75 >>/TextField]
76 </div>
77 <div className="mb-2">
78 <TextField
79   name="address"
80   margin="dense"
81   type="text"
82   placeholder="address"
83   variant="outlined"
84   label="address"
85   value={values.address}
86   onChange={handleChange}
87   onBlur={handleBlur}
88   fullWidth
89   required
90 >>/TextField>
91 { /* {errors.email} && touched.email ? (
92 <p className="text-danger">{errors.email}</p>
```

Fig 5:OrderNow.js(a)

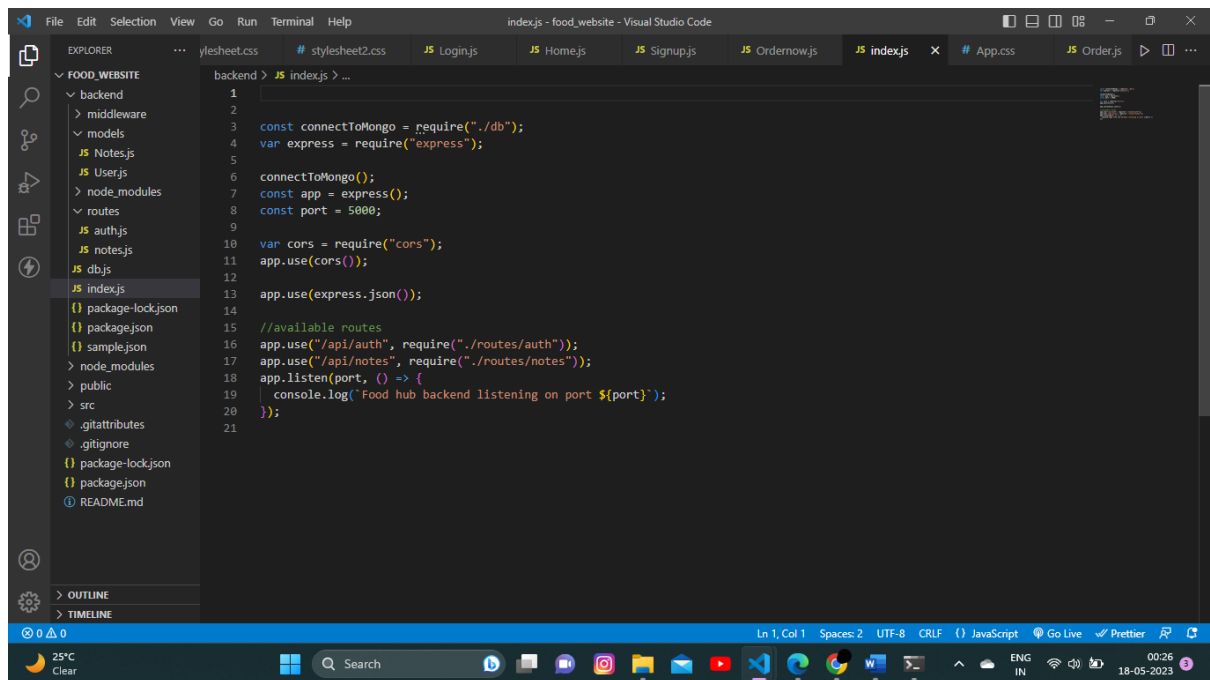


This screenshot shows the Visual Studio Code editor with the file `OrderNow.js` open. The Explorer sidebar on the left shows the project structure for `FOOD_WEBSITE`, including files like `index.html`, `manifest.json`, `src`, `components`, `App.css`, `App.js`, `App.test.js`, `index.css`, `index.js`, `logo.svg`, `reportWebVitals.js`, `setupTests.js`, and `Timeline`. The main editor area displays the following JavaScript code for the `OrderNow` component:

```
src > components > JS OrderNow.js > ...
1 import React from "react";
2 import { TextField } from "@mui/material";
3 import axios from "axios";
4 import { Formik } from "formik";
5 import { useNavigate } from "react-router-dom";
6
7 export const Ordernow = () => {
8   const navigate = useNavigate();
9   const handleFormSubmit = async (values) => {
10     axios
11       .post("http://localhost:5000/api/notes/ordernow", values)
12       .then((response) => {
13         alert("Add to kart");
14         navigate("/");
15         localStorage.setItem("user", JSON.stringify(response));
16       })
17       .catch((error) => {
18         console.log(error);
19       });
20   };
21
22   return (
23     <div>
24       <br></br>
25       <br />
26       <br />
```

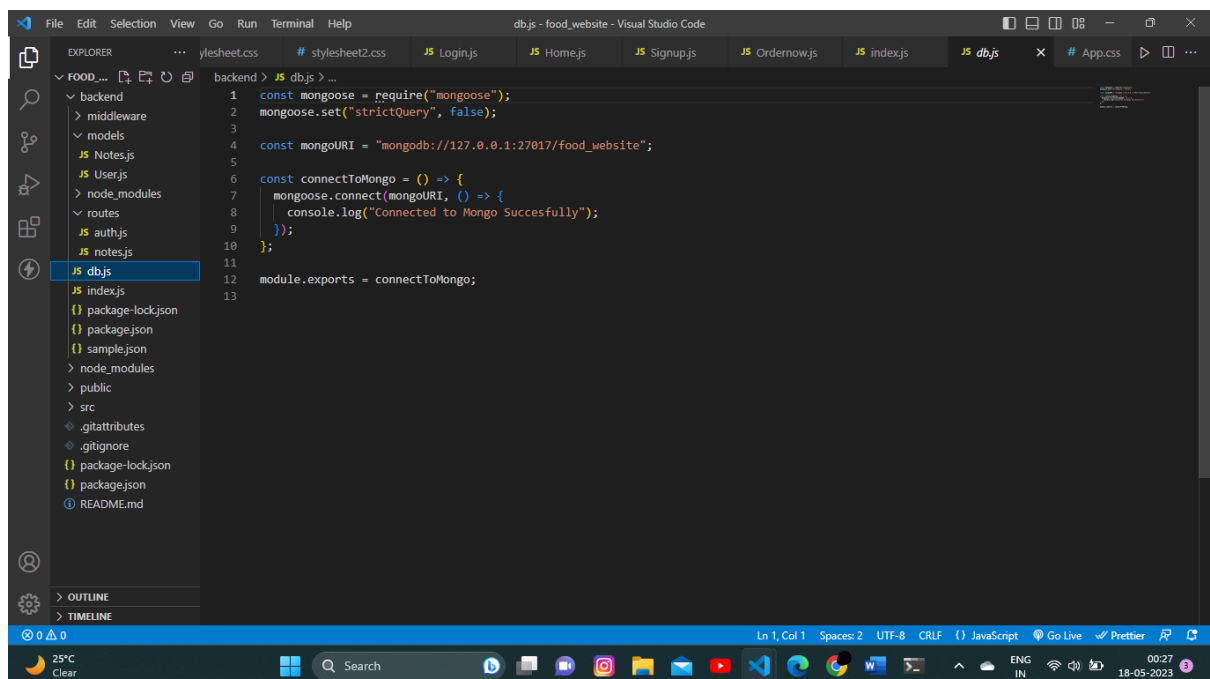
Fig 6:OrderNow.js(b)

Backend



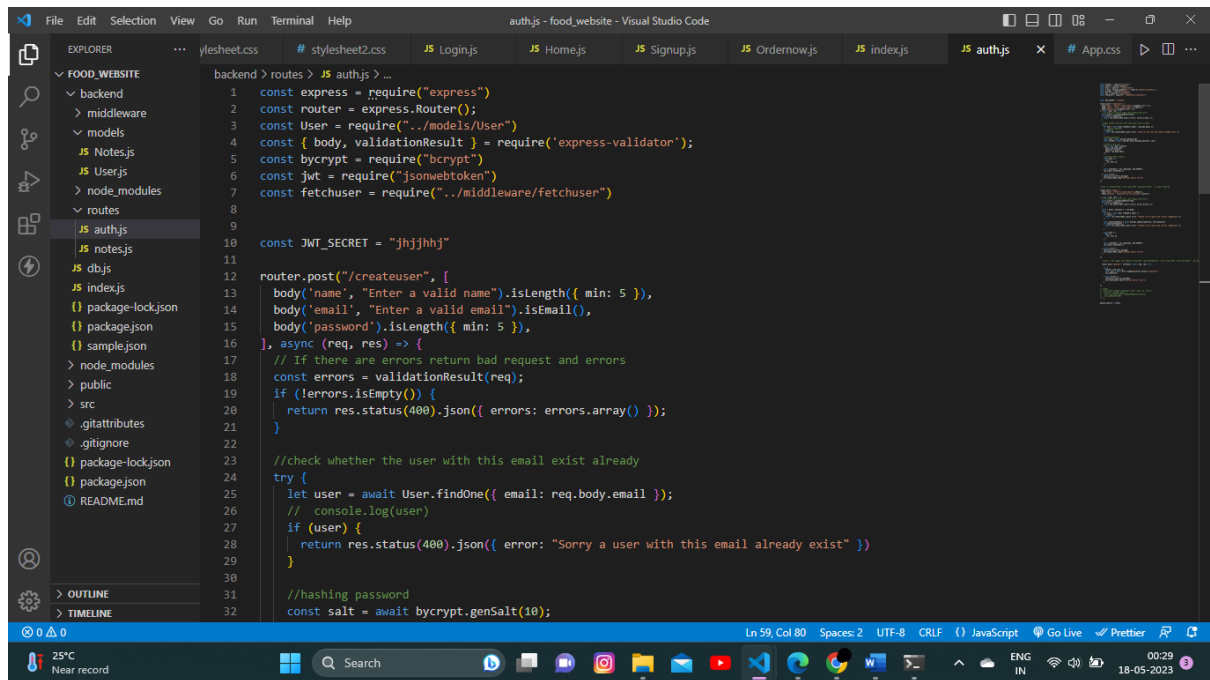
```
1
2
3 const connectToMongo = require("../db");
4 var express = require("express");
5
6 connectToMongo();
7 const app = express();
8 const port = 5000;
9
10 var cors = require("cors");
11 app.use(cors());
12
13 app.use(express.json());
14
15 //available routes
16 app.use("/api/auth", require("../routes/auth"));
17 app.use("/api/notes", require("../routes/notes"));
18 app.listen(port, () => {
19   console.log("Food hub backend listening on port ${port}");
20 });
21
```

Fig 1:Index.js



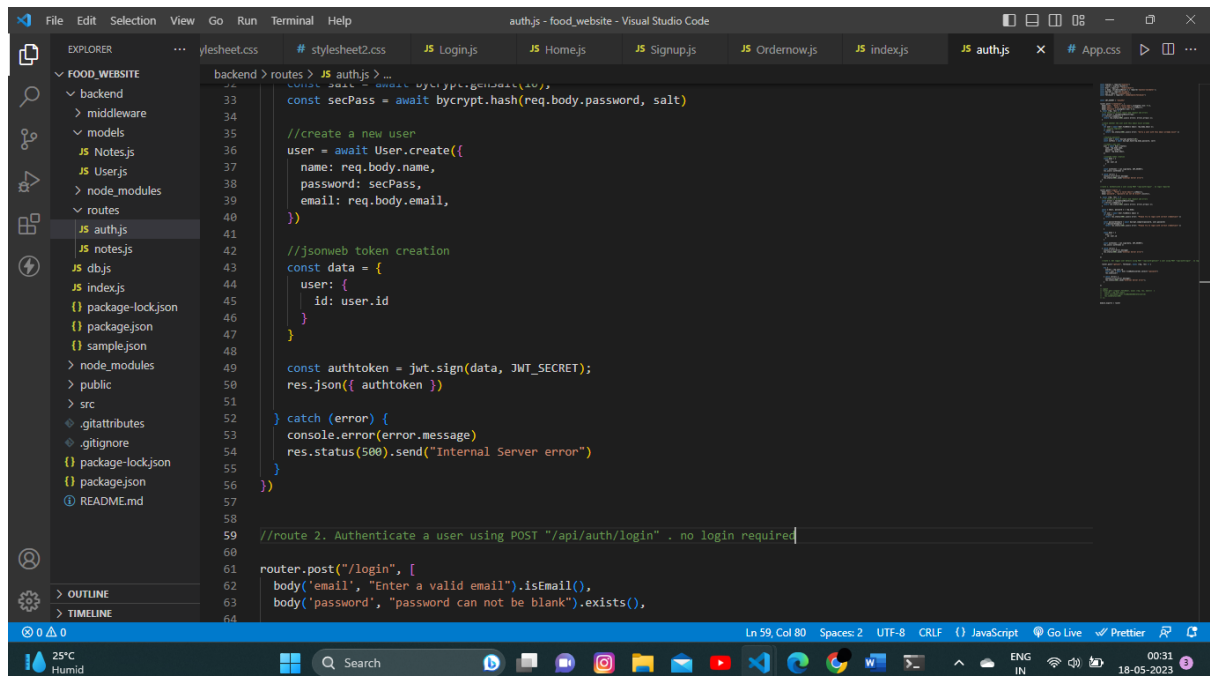
```
1 const mongoose = require("mongoose");
2 mongoose.set("strictQuery", false);
3
4 const mongoURI = "mongodb://127.0.0.1:27017/food_website";
5
6 const connectToMongo = () => {
7   mongoose.connect(mongoURI, () => {
8     console.log("Connected to Mongo Succesfully");
9   });
10 };
11
12 module.exports = connectToMongo;
13
```

Fig 2:db.js



```
1 const express = require("express")
2 const router = express.Router()
3 const User = require("../models/User")
4 const { body, validationResult } = require('express-validator');
5 const bcrypt = require("bcrypt")
6 const jwt = require("jsonwebtoken")
7 const fetchuser = require("../middleware/fetchuser")
8
9
10 const JWT_SECRET = "jhjjhhj"
11
12 router.post("/createuser", [
13   body('name', "Enter a valid name").isLength({ min: 5 }),
14   body('email', "Enter a valid email").isEmail(),
15   body('password').isLength({ min: 5 }),
16 ], async (req, res) => {
17   // If there are errors return bad request and errors
18   const errors = validationResult(req);
19   if (!errors.isEmpty()) {
20     return res.status(400).json({ errors: errors.array() });
21   }
22
23   //check whether the user with this email exist already
24   try {
25     let user = await User.findOne({ email: req.body.email });
26     // console.log(user)
27     if (user) {
28       return res.status(400).json({ error: "Sorry a user with this email already exist" });
29     }
30
31     //hashing password
32     const salt = await bcrypt.genSalt(10);
```

Fig 3:Routes/auth.js (a)



```
33   const secPass = await bcrypt.hash(req.body.password, salt)
34
35   //create a new user
36   user = await User.create({
37     name: req.body.name,
38     password: secPass,
39     email: req.body.email,
40   })
41
42   //jsonweb token creation
43   const data = {
44     user: {
45       id: user.id
46     }
47   }
48
49   const authToken = jwt.sign(data, JWT_SECRET);
50   res.json({ authToken })
51
52 } catch (error) {
53   console.error(error.message)
54   res.status(500).send("Internal Server error")
55 }
56 })
57
58 //route 2. Authenticate a user using POST "/api/auth/login" . no login required
59
60 router.post("/login", [
61   body('email', "Enter a valid email").isEmail(),
62   body('password', "password can not be blank").exists(),
63 ])
```

Fig 4:Routes/auth.js (b)

The screenshot shows the Visual Studio Code editor with the file explorer on the left displaying the project structure. The file explorer shows a folder named 'FOOD_WEBSITE' with subfolders 'backend', 'middleware', 'models', 'node_modules', 'public', 'src', and 'gitattributes'. The 'models' folder contains 'Notes.js', 'User.js', and 'db.js'. The 'routes' folder contains 'auth.js', 'notes.js', and 'db.js'. The 'notes.js' file is selected and its content is displayed in the editor. The code defines a POST route for '/ordernow' that validates the request body, creates a new order, and returns a JSON response with the order details.

```
1 const express = require("express")
2 const router = express.Router();
3 const Order = require("../models/Notes")
4 const { body, validationResult } = require('express-validator');
5 const fetchuser = require("../middleware/fetchuser")
6 const Notes = require("../models/Notes")
7
8 router.post("/ordernow", [
9   body('title', "Enter a valid title").isLength({ min: 5 }),
10  body('address', "Description must be atleast 5 character").isLength({ min: 5 }), async (req, res) => {
11
12    const errors = validationResult(req);
13    if (!errors.isEmpty()) {
14      return res.status(400).json({ errors: errors.array() });
15    }
16
17    //create a new order
18    const order = await Order.create({
19      title: req.body.title,
20      address: req.body.address,
21      price: req.body.price,
22    })
23
24    //jsonweb token creation
25    const data = {
26      order: {
27        id: order.id
28      }
29    }
30    console.log(order)
31  })
32
```

Fig 5: Routes/notes.js

The screenshot shows the Visual Studio Code editor with the file explorer on the left displaying the project structure. The file explorer shows a folder named 'FOOD_WEBSITE' with subfolders 'backend', 'middleware', 'models', 'node_modules', 'public', 'src', and 'gitattributes'. The 'models' folder contains 'Notes.js', 'User.js', and 'db.js'. The 'Notes.js' file is selected and its content is displayed in the editor. The code defines a Mongoose schema for the 'Notes' model, including fields for 'user', 'title', 'address', 'price', and 'date'. It also defines a Mongoose model for the 'Notes' collection and exports it.

```
1 const mongoose = require("mongoose")
2 const { Schema } = mongoose;
3
4 const NotesSchema = new Schema({
5   // user: {
6   //   type: mongoose.Schema.Types.ObjectId,
7   //   ref: 'user',
8   // },
9   title: {
10     type: String,
11     required: true
12   },
13   address: {
14     type: String,
15     required: true,
16   },
17   price: {
18     type: Number,
19     required: true,
20   },
21   date: {
22     type: Date,
23     required: true,
24     default: Date.now
25   }
26 })
27
28 const User = mongoose.model("notes", NotesSchema)
29 // User.createIndexes()
30 module.exports = User
31
```

Fig 6: Models/Notes.js(Schema)

RESULT

