1. Frontend (User Interface)

For the frontend, we can use React for the UI, as it provides flexibility and scalability. We will also use libraries like Axios for making API requests to the backend and React Router for handling routes.

Tech Stack for Frontend:

- **React**: For building the user interface.
- **React Router**: For managing routes in the application.
- **Axios**: For making HTTP requests to the backend.
- **CSS/SCSS**: For styling the application.

Folder Structure for Frontend:

```
pgsql
Copy
frontend/
  ├── public/
  ├── src/
  │    ├── components/
  │    │    ├── Header.js
  │    │    ├── FoodList.js
  │    │    ├── RestaurantList.js
  │    │    └── Cart.js
  │    ├── pages/
  │    │    ├── Home.js
  │    │    ├── Login.js
  │    │    └── Dashboard.js
  │    ├── App.js
  │    ├── index.js
  │    └── api.js
  └── package.json
```

Frontend Code:

`App.js` (Main Entry of Application):

```
jsx
Copy
import React from "react";
import { BrowserRouter as Router, Route, Switch } from "react-router-dom";
import Home from "./pages/Home";
import Login from "./pages/Login";
import Dashboard from "./pages/Dashboard";

function App() {
  return (
    <Router>
      <Switch>
        <Route exact path="/" component={Home} />
        <Route path="/login" component={Login} />
        <Route path="/dashboard" component={Dashboard} />
      </Switch>
    </Router>
  );
}
```

```
export default App;
```

`Home.js` (Homepage with List of Restaurants and Food Items):

```
jsx
Copy
import React, { useState, useEffect } from "react";
import axios from "axios";
import RestaurantList from "../components/RestaurantList";
import FoodList from "../components/FoodList";

const Home = () => {
  const [restaurants, setRestaurants] = useState([]);
  const [foods, setFoods] = useState([]);

  useEffect(() => {
    // Fetch restaurant and food data from API
    axios.get("/api/restaurants").then((res) => {
      setRestaurants(res.data);
    });
    axios.get("/api/foods").then((res) => {
      setFoods(res.data);
    });
  }, []);

  return (
    <div>
      <h1>Food Delivery</h1>
      <RestaurantList restaurants={restaurants} />
      <FoodList foods={foods} />
    </div>
  );
};

export default Home;
```

`RestaurantList.js` (Restaurant Component):

```
jsx
Copy
import React from "react";

const RestaurantList = ({ restaurants }) => {
  return (
    <div>
      <h2>Restaurants</h2>
      <ul>
        {restaurants.map((restaurant) => (
          <li key={restaurant.id}>{restaurant.name}</li>
        ))}
      </ul>
    </div>
  );
};

export default RestaurantList;
```

`FoodList.js` (Food Item Component):

```
jsx
Copy
import React from "react";

const FoodList = ({ foods }) => {
  return (
    <div>
      <h2>Food Items</h2>
      <ul>
        {foods.map((food) => (
          <li key={food.id}>{food.name} - ${food.price}</li>
        ))}
      </ul>
    </div>
  );
};

export default FoodList;
```

2. Backend (API & Server)

For the backend, we can use **Node.js** with **Express** to build the API and a **MongoDB** database for storing user and food data.

Tech Stack for Backend:

- **Node.js**: Runtime environment for building the server.
- **Express.js**: Web framework for handling routes.
- **MongoDB**: NoSQL database for storing user data, restaurants, and food items.
- **JWT (JSON Web Token)**: For user authentication.
- **bcryptjs**: For hashing passwords.

Folder Structure for Backend:

```
pgsql
Copy
backend/
  ├── controllers/
  │    ├── authController.js
  │    ├── foodController.js
  │    └── restaurantController.js
  ├── models/
  │    ├── User.js
  │    ├── Food.js
  │    └── Restaurant.js
  ├── routes/
  │    ├── authRoutes.js
  │    ├── foodRoutes.js
  │    └── restaurantRoutes.js
  ├── config/
  │    └── db.js
  ├── server.js
  └── package.json
```

Backend Code:

`server.js` (Main Entry of Backend Server):

```
js
Copy
const express = require("express");
const mongoose = require("mongoose");
const authRoutes = require("./routes/authRoutes");
const foodRoutes = require("./routes/foodRoutes");
const restaurantRoutes = require("./routes/restaurantRoutes");

const app = express();

// Middleware
app.use(express.json());

// Routes
app.use("/api/auth", authRoutes);
app.use("/api/foods", foodRoutes);
app.use("/api/restaurants", restaurantRoutes);

// Connect to MongoDB
mongoose
  .connect("mongodb://localhost/food-delivery", { useNewUrlParser: true,
useUnifiedTopology: true })
  .then(() => console.log("MongoDB connected"))
  .catch((err) => console.log(err));

// Server
const PORT = process.env.PORT || 5000;
app.listen(PORT, () => console.log(`Server running on port ${PORT}`));
```

`authController.js` (User Authentication):

```
js
Copy
const bcrypt = require("bcryptjs");
const jwt = require("jsonwebtoken");
const User = require("../models/User");

// Register new user
exports.registerUser = async (req, res) => {
  const { username, password } = req.body;
  try {
    const salt = await bcrypt.genSalt(10);
    const hashedPassword = await bcrypt.hash(password, salt);
    const newUser = new User({ username, password: hashedPassword });
    await newUser.save();
    res.status(201).json({ message: "User created" });
  } catch (err) {
    res.status(500).json({ message: err.message });
  }
};

// Login user
exports.loginUser = async (req, res) => {
  const { username, password } = req.body;
  try {
    const user = await User.findOne({ username });
    if (!user) return res.status(400).json({ message: "User not found" });

    const isMatch = await bcrypt.compare(password, user.password);
    if (!isMatch) return res.status(400).json({ message: "Invalid
credentials" });
```

```
    const token = jwt.sign({ id: user._id }, "secret", { expiresIn: "1h"
});
    res.json({ token });
  } catch (err) {
    res.status(500).json({ message: err.message });
  }
};
```

`foodController.js` (Handle Food Data):

```
js
Copy
const Food = require("../models/Food");

exports.getFoods = async (req, res) => {
  try {
    const foods = await Food.find();
    res.json(foods);
  } catch (err) {
    res.status(500).json({ message: err.message });
  }
};
```

`restaurantController.js` (Handle Restaurant Data):

```
js
Copy
const Restaurant = require("../models/Restaurant");

exports.getRestaurants = async (req, res) => {
  try {
    const restaurants = await Restaurant.find();
    res.json(restaurants);
  } catch (err) {
    res.status(500).json({ message: err.message });
  }
};
```

Models (MongoDB Schema):

`User.js` (User Model):

```
js
Copy
const mongoose = require("mongoose");

const userSchema = new mongoose.Schema({
  username: { type: String, required: true },
  password: { type: String, required: true },
});

module.exports = mongoose.model("User", userSchema);
```

`Food.js` (Food Item Model):

```
js
Copy
```

```
const mongoose = require("mongoose");

const foodSchema = new mongoose.Schema({
  name: { type: String, required: true },
  price: { type: Number, required: true },
  restaurantId: { type: mongoose.Schema.Types.ObjectId, ref: "Restaurant"
},
});

module.exports = mongoose.model("Food", foodSchema);
```

`Restaurant.js` (Restaurant Model):

```
js
Copy
const mongoose = require("mongoose");

const restaurantSchema = new mongoose.Schema({
  name: { type: String, required: true },
  location: { type: String },
});

module.exports = mongoose.model("Restaurant", restaurantSchema);
```

3. Setting up MongoDB:

Ensure that you have MongoDB set up locally or use a cloud-based MongoDB service like MongoDB Atlas. If you're using MongoDB locally, make sure it's running on `mongodb://localhost/food-delivery`.

4. Run the Application:

Frontend:

- Install dependencies: `npm install`
- Run the React app: `npm start`

Backend:

- Install dependencies: `npm install`
- Run the backend server: `node server.js`