Full Stack Development Assignment

Objective:
This assignment will guide students through the process of building a full-stack application. It covers essential aspects of full-stack development, including API development, user authentication (login/signup), CRUD operations, and a frontend to display data.

Technology Stack:

- **Backend:** Node.js, Express.js, MongoDB (You can use any database like PostgreSQL, MySQL as an alternative)
- **Frontend:** React (or any frontend framework like Vue.js/Angular)
- **Tools:** Postman for API testing, Git for version control

---

TODO#1: Setting Up the Backend and API Development

Step 1: Project Initialization and Setup (20 Minutes)

1. **Initialize the Project:**
    - Create a new project directory.
    - Initialize a Node.js project:

    ```
    npm init -y
    ```

2. **Install Required Packages:**

    ```
    npm install express mongoose bcrypt jsonwebtoken cors
    ```

3. **Set Up Express Server:**
    - Create a file named server.js:

    ```
    const express = require('express');
    const app = express();
    const PORT = 5000;

    app.use(express.json());
    app.listen(PORT, () => {
      console.log(`Server running on http://localhost:${PORT}`);
    });
    ```

4. **Test the Server:**
    - Run the server using:

      ```
      node server.js
      ```

    - Open http://localhost:5000 in your browser to verify the server is running.

Step 2: Database Setup (20 Minutes)

1. **Set Up MongoDB:**
    - Use MongoDB Atlas or a local MongoDB setup.
    - Install MongoDB and connect using Mongoose:

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

      mongoose.connect('mongodb://localhost:27017/ecommerce', {
        useNewUrlParser: true,
        useUnifiedTopology: true,
      })
      .then(() => console.log('Database connected'))
      .catch(err => console.error('Database connection error:', err));
      ```

2. **Define User and Product Models:**
    - Create models/User.js:

      ```
      const mongoose = require('mongoose');
      const UserSchema = new mongoose.Schema({
        username: String,
        email: String,
        password: String,
      });
      module.exports = mongoose.model('User', UserSchema);
      ```

    - Create models/Product.js:

      ```
      const mongoose = require('mongoose');
      const ProductSchema = new mongoose.Schema({

        name: String,
      ```

```
  price: Number,
  description: String,
  category: String,
});
module.exports = mongoose.model('Product', ProductSchema);
```

Step 3: User Authentication API (Login/Signup) (40 Minutes)

1. **Signup API:**
   - Create routes/auth.js and add the following code:

```
const express = require('express');
const bcrypt = require('bcrypt');
const User = require('../models/User');
const router = express.Router();

router.post('/signup', async (req, res) => {
  const { username, email, password } = req.body;
  const hashedPassword = await bcrypt.hash(password, 10);
  const newUser = new User({ username, email, password: hashedPassword
});
  await newUser.save();
  res.status(201).json({ message: 'User created successfully' });
});
module.exports = router;
```

2. **Login API:**

```
router.post('/login', async (req, res) => {
  const { email, password } = req.body;
  const user = await User.findOne({ email });
  if (!user || !(await bcrypt.compare(password, user.password))) {
    return res.status(401).json({ message: 'Invalid credentials' });
  }
  res.status(200).json({ message: 'Login successful', user });
});
```

3. **Test Using Postman:**

- Test /signup and /login endpoints using Postman.

TODO#2: CRUD API Development for Products

Step 4: CRUD APIs for Products (40 Minutes)

1. **Create Product Routes:**
   - Create routes/products.js:

```
const express = require('express');
const Product = require('../models/Product');
const router = express.Router();

// Create a new product
router.post('/', async (req, res) => {
  const newProduct = new Product(req.body);
  await newProduct.save();
  res.status(201).json(newProduct);
});

// Get all products
router.get('/', async (req, res) => {
  const products = await Product.find();
  res.json(products);
});

// Update a product
router.put('/:id', async (req, res) => {
  const updatedProduct = await Product.findByIdAndUpdate(req.params.id,
req.body, { new: true });
  res.json(updatedProduct);
});

// Delete a product
router.delete('/:id', async (req, res) => {
  await Product.findByIdAndDelete(req.params.id);
  res.json({ message: 'Product deleted' });
});
```

```
module.exports = router;
```

2. **Test All CRUD Endpoints Using Postman:**
    - POST /products
    - GET /products
    - PUT /products/:id
    - DELETE /products/:id

TODO#3: Frontend Development Using React (1 Hours)

Step 5: Frontend Setup and User Interface Development

1. **Initialize React App:**

```
npx create-react-app ecommerce-frontend
```

2. **Build a Product List Page:**
    - Create a ProductList.js component and fetch products from the API:

```
import React, { useEffect, useState } from 'react';
function ProductList() {
  const [products, setProducts] = useState([]);

  useEffect(() => {
    fetch('http://localhost:5000/products')
      .then(response => response.json())
      .then(data => setProducts(data));
  }, []);
  return (
    <div>
      <h2>Product List</h2>
      {products.map(product => (
        <div key={product._id}>
          <h3>{product.name}</h3>
          <p>{product.description}</p>
          <p>Price: ${product.price}</p>
        </div>
```

```
        ))}
      </div>
    );
}

export default ProductList;
```

TODO#4: Integration and Testing

Step 6: Integration and Deployment (40 minutes)

1. **Integrate Backend and Frontend:**
    - Update the React app to use API endpoints for login, signup, and product display.
2. **Deploy the Application:**
    - Use Heroku or Vercel for frontend deployment and MongoDB Atlas for database.
3. **Testing:**
    - Test the entire application flow, from signup to viewing the product list.

Wrap-Up and Submission:

- **Review Questions:**
    1. How does the frontend communicate with the backend?
    2. What are the benefits of using a separate API layer?
    3. Explain how state is managed in the React component.
- **Submission:**
    - Submit the GitHub repository link with both backend and frontend code.
