Assignment Submission

BSc (Hons) Computer Systems Engineering

University of Sunderland

ISMT College Butwal

Module Name: CET 139 Full stack Development

Topic Name: CSS

Name : Pratik Poudel
user id : bj03gj

Full Stack Development Code

Introduction

Full Stack Development involves building both the front-end and back-end of web applications, requiring expertise in technologies like HTML, CSS, JavaScript, serverside languages, and databases. My ePortfolio (index.html) describes a full stack workflow, referencing a Node.js/Express.js back-end with MongoDB and JWT authentication integrated with a front-end built using HTML, CSS, Bootstrap, and JavaScript. This document revisits the example from the ePortfolio, provides a complete full stack code implementation, and explains its functionality, fulfilling the request for a "fullstack code ss." As screenshots are not supported, the code is presented in text form, ready for execution or visualization in an editor. This document is designed for developers seeking practical insights into full stack development.

Original Example from ePortfolio

The ePortfolio describes a full stack application with the following flow:

Front-End: Sends HTTP requests (e.g., via Fetch API) to the back-end.

Back-End: Processes logic, queries the database, and returns data.

Database: Stores and retrieves data (e.g., user profiles)

The example mentions a login feature where a front-end form collects credentials, validated by a Node.js/Express.js back-end against a MongoDB database, using JSON Web Tokens (JWT) for authentication (Karki, 2025). The HTML snippet from index.html includes a login form:

```
<form>
  <div class="mb-3">
    <label for="username" class="form-label">Username:</label>
    <input type="text" class="form-control" id="username">
  </div>
  <div class="mb-3">
    <label for="password" class="form-label">Password:</label>
    <input type="password" class="form-control" id="password">
  </div>
  <button type="submit" class="btn btn-primary">Login</button>
</form>
```

This form, styled with Bootstrap, demonstrates the front-end component but lacks the back-end code in the original file.

Full Stack Code Implementation

Below is a complete, minimal full stack implementation based on the ePortfolio's description, using the MERN stack (MongoDB, Express.js, Node.js, and a Bootstrap front-end). This example implements a login system with JWT authentication.

Front-End (HTML with JavaScript)

File: index.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Full Stack Login</title>
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css" rel="stylesheet">
</head>
<body>
  <div class="container mt-5">
    <h2>Login</h2>
    <form id="loginForm">
      <div class="mb-3">
        <label for="username" class="form-label">Username:</label>
        <input type="text" class="form-control" id="username" required>
      </div>
      <div class="mb-3">
        <label for="password" class="form-label">Password:</label>
        <input type="password" class="form-control" id="password" required>
      </div>
      <button type="submit" class="btn btn-primary">Login</button>
    </form>
    <div id="message" class="mt-3"></div>
  </div>
  <script>
    document.getElementById('loginForm').addEventListener('submit', async (e) => {
      e.preventDefault();
      const username = document.getElementById('username').value;
      const password = document.getElementById('password').value;
      try {
        const response = await fetch('http://localhost:3000/api/login', {
          method: 'POST',
          headers: { 'Content-Type': 'application/json' },
          body: JSON.stringify({ username, password })
        });
        const data = await response.json();
        document.getElementById('message').innerText = data.message || 'Login successful!';
        if (data.token) localStorage.setItem('token', data.token);
      } catch (error) {
        document.getElementById('message').innerText = 'Error: ' + error.message;
      }
    });
  </script>
</body>
</html>
```

Back-End (Node.js/Express.js)

```
const express = require('express');
const mongoose = require('mongoose');
const jwt = require('jsonwebtoken');
const cors = require('cors');
const app = express();

app.use(cors());
app.use(express.json());

// MongoDB connection
mongoose.connect('mongodb://localhost:27017/fullstackdb', { useNewUrlParser: true, useUnifiedTopology: true });

// User schema
const userSchema = new mongoose.Schema({
  username: String,
  password: String // In production, use bcrypt for password hashing
});
const User = mongoose.model('User', userSchema);

// Login endpoint
app.post('/api/login', async (req, res) => {
  const { username, password } = req.body;
  const user = await User.findOne({ username, password });
  if (!user) {
    return res.status(401).json({ message: 'Invalid credentials' });
  }
  const token = jwt.sign({ username }, 'secret_key', { expiresIn: '1h' });
  res.json({ message: 'Login successful', token });
});

app.listen(3000, () => console.log('Server running on port 3000'));
```

Prerequisites

Install Node.js, MongoDB, and dependencies (npm install express mongoose jsonwebtoken cors).

Run MongoDB locally (mongod).

Start the server (node server.js).

Serve the HTML file using a local server (e.g., npx serve).

Explanation

Front-End: The HTML form uses Bootstrap for styling and JavaScript's Fetch API to send a POST request to the back-end. The response displays a success message or error, storing the JWT in localStorage upon successful login (Richardson and Ruby, 2007).

Back-End: The Express.js server connects to MongoDB, validates user credentials, and generates a JWT for authentication. Note: This example uses plain-text passwords for simplicity; in production, use bcrypt for hashing (OWASP, 2021).

Database: MongoDB stores user data, queried during login to verify credentials.

This implementation mirrors the ePortfolio's described flow, connecting a Bootstrap front-end to a Node.js/Express.js back-end with MongoDB.

Advantages and Challenges

This code demonstrates key full stack advantages:

> **Integration**: Seamless front-end/back-end communication via REST API (Richardson and Ruby, 2007).
>
> **Efficiency**: Bootstrap accelerates front-end development, while Express.js simplifies server logic.

Challenges include:

> **Security**: Plain-text passwords and a simple JWT secret require enhancement (e.g., bcrypt, environment variables) (OWASP, 2021).
>
> **Scalability**: The example lacks advanced features like middleware or error handling.

Advanced Practices

> **Security**: Implement password hashing and secure JWT storage (OWASP, 2021).
>
> **Performance**: Use Redis for caching and NGINX for load balancing (Bass et al., 2015).
>
> **Testing**: Add Jest for unit tests and Cypress for end-to-end testing.

Conclusion

The full stack implementation expands the ePortfolio's conceptual example into a functional login system, showcasing the integration of HTML, Bootstrap, JavaScript, Node.js, Express.js, and MongoDB. This practical demonstration highlights the power of full stack development in creating responsive, secure, and dynamic applications. Developers can extend this foundation with advanced practices to build robust solutions.

References

> Bass, L., Weber, I. and Zhu, L., 2015. *DevOps: A Software Architect's Perspective*. Boston: Addison-Wesley.
>
> Karki, A., 2025. *Aman Karki's ePortfolio*. [online] Available at: [ePortfolio source, internal reference].

OWASP, 2021. *OWASP Top Ten Web Application Security Risks*. [online] Available at: https://owasp.org/www-project-top-ten/ [Accessed 29 August 2025].

Richardson, L. and Ruby, S., 2007. *RESTful Web Services*. Sebastopol: O'Reilly Media.