

Laporan Praktikum
Mata Kuliah Pemrograman WEB



PERTEMUAN 6. TUGAS 1
PRAKTIKUM QUIZ SESSION

Dosen Pengampu:
Wildan Aprizall Arifin, S.Pd., M.Kom.

Disusun Oleh:
Nelita Maharani
2307052

PROGRAM STUDI SISTEM INFORMASI KELAUTAN
UNIVERSITAS PENDIDIKAN INDONESIA

2024

I. PENDAHULUAN

CRUD merupakan konsep fundamental dalam dunia pengembangan aplikasi yang mengacu pada empat operasi dasar dalam pengelolaan data: Create (membuat), Read (membaca), Update (memperbarui), dan Delete (menghapus). Konsep ini menjadi pondasi penting dalam hampir setiap aplikasi yang melibatkan manipulasi data, mulai dari aplikasi sederhana hingga sistem yang kompleks.

Dalam konteks pengembangan aplikasi, operasi Create memungkinkan pengguna untuk menambahkan data baru ke dalam sistem. Misalnya, ketika seseorang mendaftar di sebuah platform, data mereka "dibuat" dan disimpan dalam database. Operasi Read digunakan untuk mengambil dan menampilkan data yang telah tersimpan, seperti ketika aplikasi menampilkan daftar pengguna atau detail produk. Update memungkinkan modifikasi data yang sudah ada, contohnya ketika pengguna mengubah profil mereka atau admin memperbarui harga produk. Sementara itu, Delete digunakan untuk menghapus data yang tidak diperlukan lagi dari sistem.

Implementasi CRUD dapat dilakukan melalui berbagai cara, tergantung pada teknologi yang digunakan. Dalam konteks database SQL, operasi Create dilakukan menggunakan perintah INSERT, Read menggunakan SELECT, Update dengan UPDATE, dan Delete menggunakan DELETE. Pada pengembangan API RESTful, operasi-operasi ini umumnya dipetakan ke metode HTTP: POST untuk Create, GET untuk Read, PUT atau PATCH untuk Update, dan DELETE untuk operasi penghapusan.

Ketika mengimplementasikan CRUD, ada beberapa praktik terbaik yang perlu diperhatikan. Pertama, validasi data sangat penting untuk memastikan integritas data yang disimpan. Kedua, aspek keamanan harus diutamakan, termasuk implementasi autentikasi dan otorisasi untuk mencegah akses tidak sah ke operasi CRUD. Ketiga, memberikan umpan balik yang jelas kepada pengguna setelah setiap operasi CRUD sangat penting untuk pengalaman pengguna yang baik. Terakhir, optimasi performa, terutama untuk operasi yang melibatkan data dalam jumlah besar, tidak boleh diabaikan.

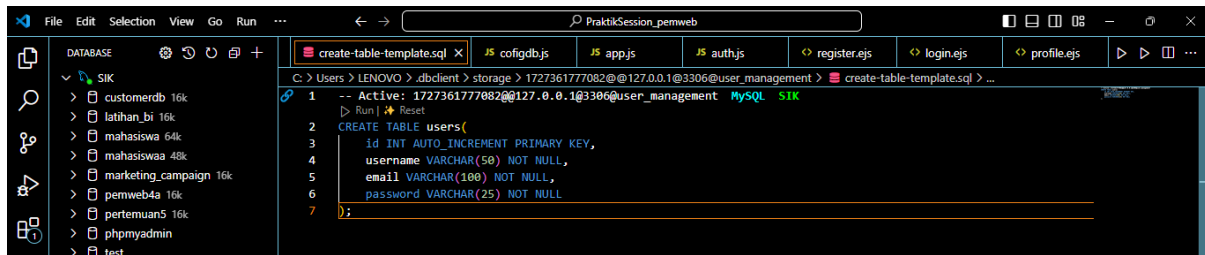
Sebagai contoh implementasi, dalam pengembangan aplikasi web menggunakan Node.js dan MySQL, operasi CRUD dapat direalisasikan menggunakan query database yang sesuai. Untuk operasi Create, aplikasi akan menggunakan query INSERT untuk menambahkan data baru. Operasi Read menggunakan SELECT untuk mengambil data yang diperlukan. Update dilakukan dengan query UPDATE untuk memodifikasi data yang sudah ada, dan Delete menggunakan query DELETE untuk menghapus data.

II. ALAT DAN BAHAN

1. Komputer/Laptop
2. Visual Studio Code
3. XAMPP
4. MySQL
5. Chrome

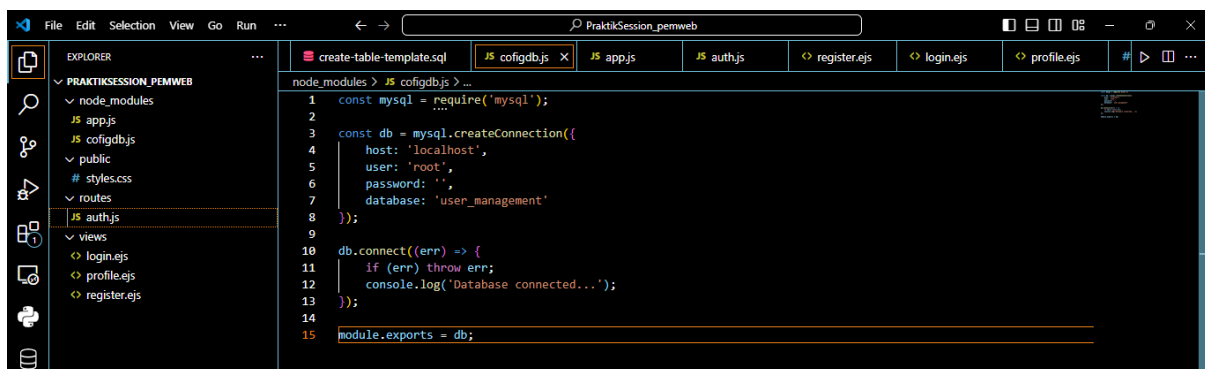
III. LANGKAH KERJA

1. Membuat database baru dan table users



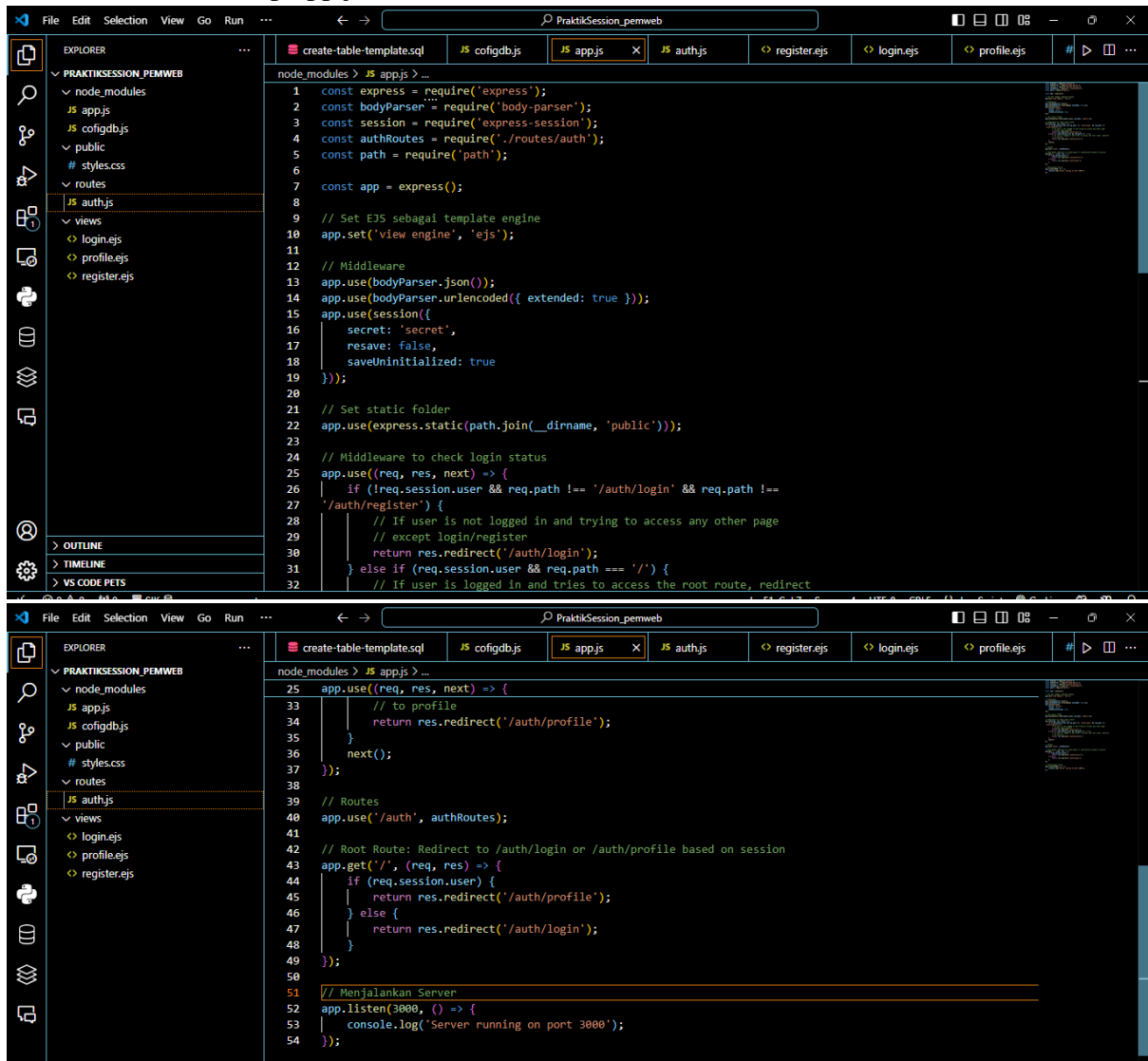
Langkah pertama membuat database bernama user_management dan tabel users di dalamnya. Tabel users ini memiliki empat kolom utama. Kolom pertama adalah id yang bertipe integer dan diatur sebagai PRIMARY KEY, di mana nilainya akan otomatis bertambah untuk setiap pengguna baru melalui fitur AUTO_INCREMENT. Kolom kedua adalah username yang bertipe VARCHAR dengan panjang maksimal 50 karakter, yang wajib diisi. Kolom ketiga adalah email, juga bertipe VARCHAR dengan panjang maksimal 100 karakter dan wajib diisi. Terakhir, terdapat kolom password yang bertipe VARCHAR dengan panjang maksimal 255 karakter, yang juga wajib diisi.

2. Membuat konfigurasi



Kode JavaScript ini menggunakan modul mysql untuk menghubungkan aplikasi Node.js dengan database MySQL. Pertama, modul mysql diimpor dengan menggunakan require('mysql'), yang memungkinkan penggunaan fungsi MySQL yang terkait. Selanjutnya, mysql.createConnection() digunakan untuk membuat koneksi ke database, dan objek konfigurasi disertakan. Objektif ini mengandung informasi penting seperti host, yang dalam kasus ini adalah "localhost", yang menunjukkan bahwa database berada di komputer lokal; user, yang diwakili oleh "root", dan password yang diatur kosong, yang digunakan dalam kasus ini; dan nama database yang akan dihubungkan, user_management. Setelah itu, db.connect() digunakan untuk menguji koneksi; jika koneksi berhasil, akan ditampilkan pesan "Database connected...". Perintah "throw err" akan menghentikan program dan menampilkan kesalahan. Terakhir, module mengeksport koneksi untuk digunakan di bagian aplikasi, gunakan exports = db.

3. Membuat setup app.js

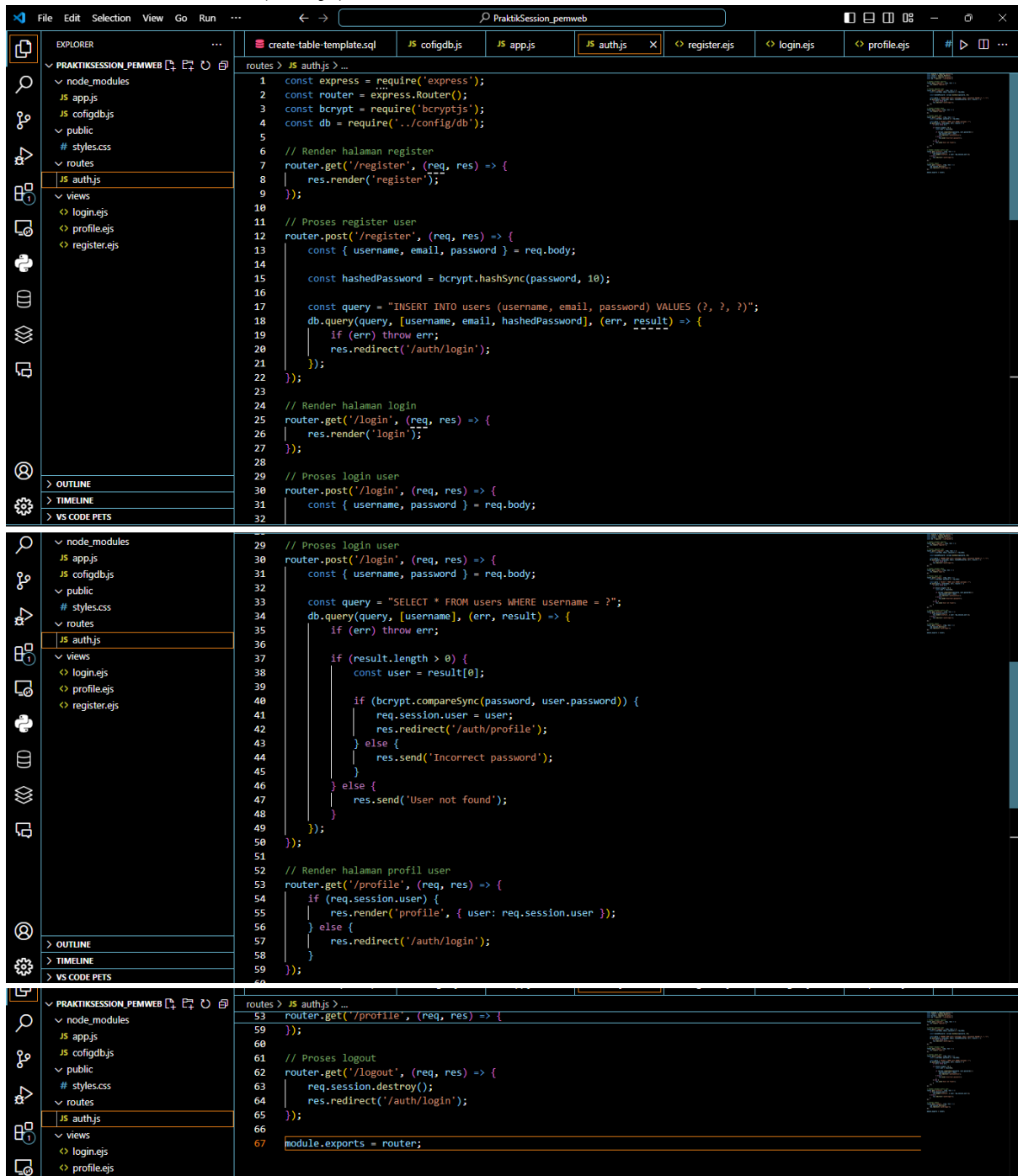


The image shows two screenshots of a VS Code editor window. The top screenshot displays the initial setup of the `app.js` file, which includes imports for `express`, `body-parser`, `express-session`, and `authRoutes`. It also shows the configuration of the EJS template engine, middleware for session management, and static file serving. The bottom screenshot shows the continuation of the `app.js` file, including route definitions for `/auth/login`, `/auth/register`, and `/auth/profile`, as well as a root route that redirects users based on their session status. The server is configured to listen on port 3000.

```
node_modules > JS app.js > ...
1  const express = require('express');
2  const bodyParser = require('body-parser');
3  const session = require('express-session');
4  const authRoutes = require('./routes/auth');
5  const path = require('path');
6
7  const app = express();
8
9  // Set EJS sebagai template engine
10 app.set('view engine', 'ejs');
11
12 // Middleware
13 app.use(bodyParser.json());
14 app.use(bodyParser.urlencoded({ extended: true }));
15 app.use(session({
16   secret: 'secret',
17   resave: false,
18   saveUninitialized: true
19 }));
20
21 // Set static folder
22 app.use(express.static(path.join(__dirname, 'public')));
23
24 // Middleware to check login status
25 app.use((req, res, next) => {
26   if (!req.session.user && req.path !== '/auth/login' && req.path !==
27     '/auth/register') {
28     // If user is not logged in and trying to access any other page
29     // except login/register
30     return res.redirect('/auth/login');
31   } else if (req.session.user && req.path === '/') {
32     // If user is logged in and tries to access the root route, redirect
33
34
35
36
37
38
39 // Routes
40 app.use('/auth', authRoutes);
41
42 // Root Route: Redirect to /auth/login or /auth/profile based on session
43 app.get('/', (req, res) => {
44   if (req.session.user) {
45     return res.redirect('/auth/profile');
46   } else {
47     return res.redirect('/auth/login');
48   }
49 });
50
51 // Menjalankan Server
52 app.listen(3000, () => {
53   console.log('Server running on port 3000');
54 });
```

Kode di atas merupakan aplikasi server sederhana yang dibuat menggunakan Node.js dengan framework Express yang menangani autentikasi pengguna. Pertama, modul-modul seperti "express", "body-parser", "express-session", dan "authRoutes" diimpor. Aplikasi ini menggunakan template engine EJS untuk tampilan dinamis. "BodyParser" digunakan untuk memproses data dari form dan JSON, dan "express-session" digunakan untuk mengatur sesi pengguna. Untuk melayani file publik seperti gambar dan CSS, direktori statis diatur dengan "express.static". Untuk mengetahui status login pengguna, middleware khusus ditambahkan. Ketika server berhasil dijalankan, pesan "Server berjalan pada port 3000" ditampilkan.

4. Membuat route (auth.js)



The image displays three sequential screenshots of a Visual Studio Code editor window, showing the development of an authentication module in a file named `auth.js`. The Explorer sidebar on the left shows the project structure, including `node_modules`, `app.js`, `configdb.js`, `public`, `routes`, `views`, `login.ejs`, `profile.ejs`, and `register.ejs`.

Top Screenshot: The `auth.js` file is open, showing the initial setup and the `register` route. The code includes imports for `express`, `bcrypt`, and `db`. It defines a `register` GET route that renders the `register` page and a POST route that handles user registration by hashing the password and inserting it into the `users` table.

```
1 const express = require('express');
2 const router = express.Router();
3 const bcrypt = require('bcryptjs');
4 const db = require('../config/db');
5
6 // Render halaman register
7 router.get('/register', (req, res) => {
8   res.render('register');
9 });
10
11 // Proses register user
12 router.post('/register', (req, res) => {
13   const { username, email, password } = req.body;
14
15   const hashedPassword = bcrypt.hashSync(password, 10);
16
17   const query = "INSERT INTO users (username, email, password) VALUES (?, ?, ?)";
18   db.query(query, [username, email, hashedPassword], (err, result) => {
19     if (err) throw err;
20     res.redirect('/auth/login');
21   });
22 });
23
24 // Render halaman login
25 router.get('/login', (req, res) => {
26   res.render('login');
27 });
28
29 // Proses login user
30 router.post('/login', (req, res) => {
31   const { username, password } = req.body;
```

Middle Screenshot: This screenshot shows the continuation of the `login` POST route. It includes a database query to find a user by username and a comparison of the provided password with the stored hashed password. If the password is correct, the user is logged in and redirected to the profile page; otherwise, an error message is sent.

```
32   const query = "SELECT * FROM users WHERE username = ?";
33   db.query(query, [username], (err, result) => {
34     if (err) throw err;
35
36     if (result.length > 0) {
37       const user = result[0];
38
39       if (bcrypt.compareSync(password, user.password)) {
40         req.session.user = user;
41         res.redirect('/auth/profile');
42       } else {
43         res.send('Incorrect password');
44       }
45     } else {
46       res.send('User not found');
47     }
48   });
49 });
50
51 // Render halaman profil user
52 router.get('/profile', (req, res) => {
53   if (req.session.user) {
54     res.render('profile', { user: req.session.user });
55   } else {
56     res.redirect('/auth/login');
57   }
58 });
59
60 module.exports = router;
```

Bottom Screenshot: This screenshot shows the `logout` route and the final export statement. The `logout` GET route destroys the session and redirects the user back to the login page.

```
61 // Proses logout
62 router.get('/logout', (req, res) => {
63   req.session.destroy();
64   res.redirect('/auth/login');
65 });
66
67 module.exports = router;
```

Modul rute autentikasi di atas menggunakan Express dan bcrypt.js untuk hashing password. Proses pendaftaran, login, profil, dan logout pengguna dilakukan melalui rute ini.


5. Register pages



```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <link rel="stylesheet" href="/styles.css">
7   <title>Register</title>
8 </head>
9 <body>
10   <div class="container">
11     <h2>Register</h2>
12     <form action="/auth/register" method="POST">
13       <label for="username">Username</label>
14       <input type="text" id="username" name="username" required>
15
16       <label for="email">Email</label>
17       <input type="email" id="email" name="email" required>
18
19       <label for="password">Password</label>
20       <input type="password" id="password" name="password" required>
21
22       <button type="submit">Register</button>
23     </form>
24     <p>Already have an account? <a href="/auth/login">Login here</a></p>
25   </div>
26 </body>
27 </html>
```

Kode tersebut merupakan halaman HTML yang akan digunakan untuk form registrasi pengguna. Halaman ini ditulis dalam bahasa Inggris dan memiliki tag HTML dasar seperti "" untuk menentukan metadata. Ini termasuk pengaturan charset UTF-8, pengaturan viewport, dan tautan ke file CSS luar seperti "/styles.css". Halaman bernama "Daftar". Sebuah div dengan class "container" ada di bagian "body", di mana form pendaftaran dapat disimpan. Dengan menggunakan metode POST, formulir ini mengirim data ke rute "/auth/register". Form memiliki input untuk username, email, dan password, yang semuanya harus diisi (ditandai dengan atribut "harus"). Data akan dikirim untuk diproses setelah pengguna mengisi formulir dan menekan tombol "Register". Jika pengguna sudah memiliki akun, ada juga link yang akan membawa mereka ke halaman login.

6. Login pages



```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <link rel="stylesheet" href="/styles.css">
7   <title>Login</title>
8 </head>
9 <body>
10   <div class="container">
11     <h2>Login</h2>
12     <form action="/auth/login" method="POST">
13       <label for="username">Username</label>
14       <input type="text" id="username" name="username" required>
15
16       <label for="password">Password</label>
17       <input type="password" id="password" name="password" required>
18
19       <button type="submit">Login</button>
20     </form>
21     <p>Don't have an account? <a href="/auth/register">Register here</a></p>
22   </div>
23 </body>
24 </html>
```

Kode tersebut merupakan halaman HTML yang akan digunakan untuk form login pengguna. Pada bagian "head", halaman dikonfigurasi dengan charset UTF-8 dan dioptimalkan untuk ponsel dengan pengaturan viewport. Halaman ini diberi nama "Login" dan terhubung ke file CSS eksternal "/styles.css" untuk menyesuaikan. Terdapat sebuah div dengan class "container" di bagian "body", yang dapat digunakan untuk menyimpan form login. Dengan menggunakan metode POST, formulir ini akan mengirimkan data ke rute "/auth/login". Form ini mengandung input untuk "username" dan "password", yang keduanya harus diisi dengan tanda "perlu". Data dikirim untuk proses autentikasi setelah pengguna

mengisi formulir dan menekan tombol "Login". Jika pengguna belum memiliki akun, di bagian bawah mereka akan menemukan tautan yang akan membawa mereka ke halaman registrasi.

7. Profile pages

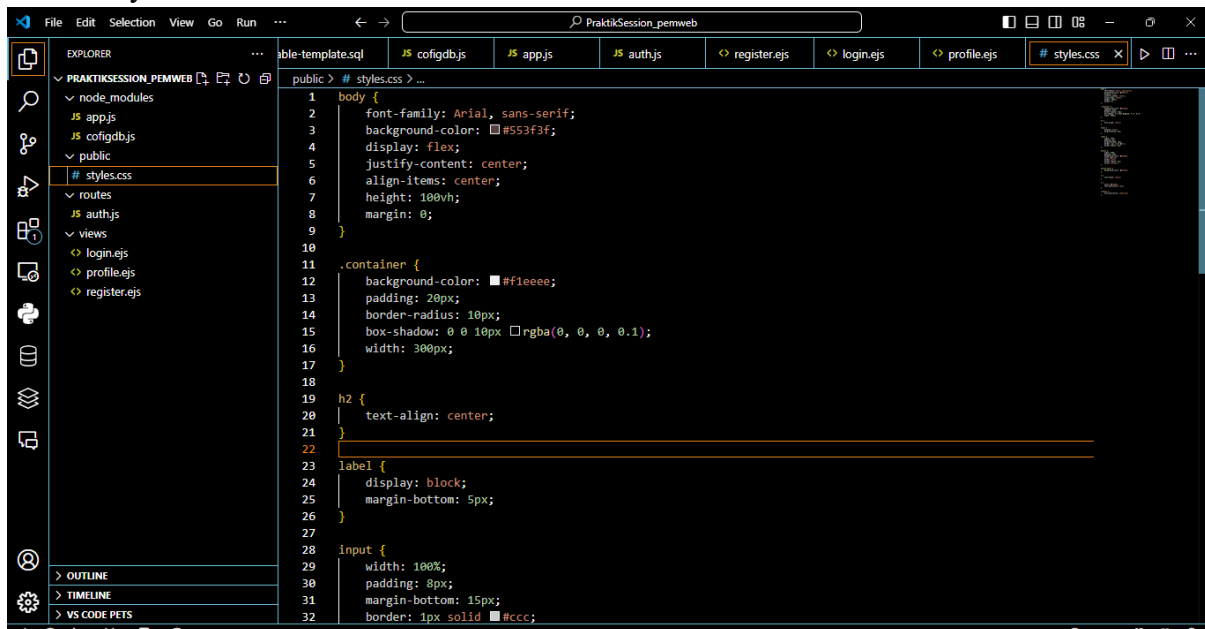


The screenshot shows the VS Code interface with the Explorer panel on the left displaying the file structure of 'PRAKTIKSESSON_PEMWEB'. The Views panel on the right shows the 'profile.ejs' file. The code in 'profile.ejs' is as follows:

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <link rel="stylesheet" href="/styles.css">
7   <title>Profile</title>
8 </head>
9 <body>
10  <div class="container">
11    <h2>Welcome, <%= user.username %></h2>
12    <p>Email: <%= user.email %></p>
13    <a href="/auth/logout">Logout</a>
14  </div>
15 </body>
16 </html>
```

Halaman profil pengguna ini dibuat menggunakan EJS (JavaScript Tertanam). Pada bagian "", halaman diatur menggunakan charset UTF-8, viewport, dan terhubung ke file CSS luar (/styles.css) untuk gaya. Halaman ini disebut "Profil". Di bagian "body", ada div dengan class "container" yang digunakan untuk menampilkan pesan selamat datang kepada pengguna yang baru saja masuk. Sintaks EJS "<%= user.username %>" digunakan untuk menampilkan nama pengguna secara dinamis, dan email pengguna ditampilkan di bawahnya dengan "<%= user.email %>". Di bagian akhir, pengguna dapat menemukan tautan ke rute "/auth/logout" untuk memulai login. Halaman ini menampilkan detail akun pengguna dan opsi untuk keluar dari sesi.

8. Styles CSS



The screenshot shows the VS Code interface with the Explorer panel on the left displaying the file structure of 'PRAKTIKSESSON_PEMWEB'. The Views panel on the right shows the 'styles.css' file. The code in 'styles.css' is as follows:

```
1 body {
2   font-family: Arial, sans-serif;
3   background-color: #553f3f;
4   display: flex;
5   justify-content: center;
6   align-items: center;
7   height: 100vh;
8   margin: 0;
9 }
10
11 .container {
12   background-color: #f1eccc;
13   padding: 20px;
14   border-radius: 10px;
15   box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
16   width: 300px;
17 }
18
19 h2 {
20   text-align: center;
21 }
22
23 label {
24   display: block;
25   margin-bottom: 5px;
26 }
27
28 input {
29   width: 100%;
30   padding: 8px;
31   margin-bottom: 15px;
32   border: 1px solid #ccc;
```




Kode di atas adalah file CSS yang digunakan untuk memberikan gaya pada halaman HTML yang berkaitan dengan profil pengguna, formulir login, atau registrasi.

IV. KESIMPULAN

Hasil dari materi praktikum ini adalah bahwa kita telah membangun dan mengelola aplikasi web sederhana menggunakan Node.js sebagai framework, Express sebagai basis data, dan bcrypt.js untuk hashing kata sandi untuk keamanan. Aplikasi ini memiliki fitur autentikasi pengguna dasar, seperti registrasi, login, logout, dan tampilan profil pengguna. Untuk menjalankannya, kami membuat jalur untuk menjalankan berbagai fungsi tersebut, menggunakan express-session untuk mengelola sesi pengguna, dan menerapkan middleware untuk memastikan hanya pengguna yang telah login yang dapat mengakses halaman tertentu. Halaman frontend dibuat menggunakan template engine EJS dan diformat dengan CSS untuk tampilan yang bersih dan responsif. Praktikum ini mengajarkan cara membuat aplikasi autentikasi yang aman dan berfungsi dengan menggabungkan berbagai komponen.

Link Github:

<https://github.com/maharaninelita/praktikSession.git>