Experiment 10: Node.js JWT Authentication

Objective

Secure your Express API using JWT tokens.

Technologies Used

- Node.js
- Express
- JSON Web Token (JWT)

Features

- User login
- Protected routes

Steps to Execute

1. Run: `npm install express jsonwebtoken`
2. Start with `node authServer.js`
3. Test login, get token, and access protected routes

Folder Contents

- `authServer.js`: Login & JWT logic

Express.js REST API with JWT Authorization

This project extends the student data REST API (from Experiment 9) by implementing JSON Web Token (JWT) based authorization. This ensures that only authenticated and authorized users can access and perform CRUD (Create, Read, Update, Delete) operations on the student data endpoints.

Features

- **User Authentication:** A dedicated `/api/login` endpoint allows users to authenticate and receive a JWT.
- **JWT Generation:** Secure JWTs are generated using the `jsonwebtoken` library, containing a user payload and an expiration time.
- **Route Protection:** Middleware is implemented to intercept requests to protected student data endpoints, verifying the presence and validity of the JWT.
- **Access Control:** Only requests with a valid JWT in the `Authorization` header (Bearer token format) are granted access to student management functionalities.
- **Password Hashing:** Uses `bcryptjs` for securely hashing dummy user passwords, demonstrating a fundamental security best practice.

- **Environment Variables:** Utilizes `dotenv` to load the JWT secret key from a `.env` file, keeping sensitive information out of the codebase.

Technologies Used

- Node.js
- Express.js (web application framework)
- `jsonwebtoken` npm package (for JWT creation and verification)
- `bcryptjs` npm package (for password hashing)
- `dotenv` npm package (for managing environment variables)
- Postman (or any API testing tool) for testing endpoints

Setup and Running

1. **Prerequisites:**
    - Node.js (LTS version recommended) and npm installed on your system.
    - Postman (or similar API testing tool) installed.
2. **Clone the repository (or create manually):**

```
git clone <repository_url>
cd express-jwt-auth # or your project directory name
```

3. **Initialize Project and Install Dependencies:** Open your terminal or command prompt, navigate to the project root directory, and run:

```
npm init -y
npm install express jsonwebtoken bcryptjs dotenv
```

4. **Create `.env` file:** In the root of your project directory, create a new file named `.env`. Add the following line to it, **replacing the placeholder with a strong, random secret key (e.g., 32+ characters)**:

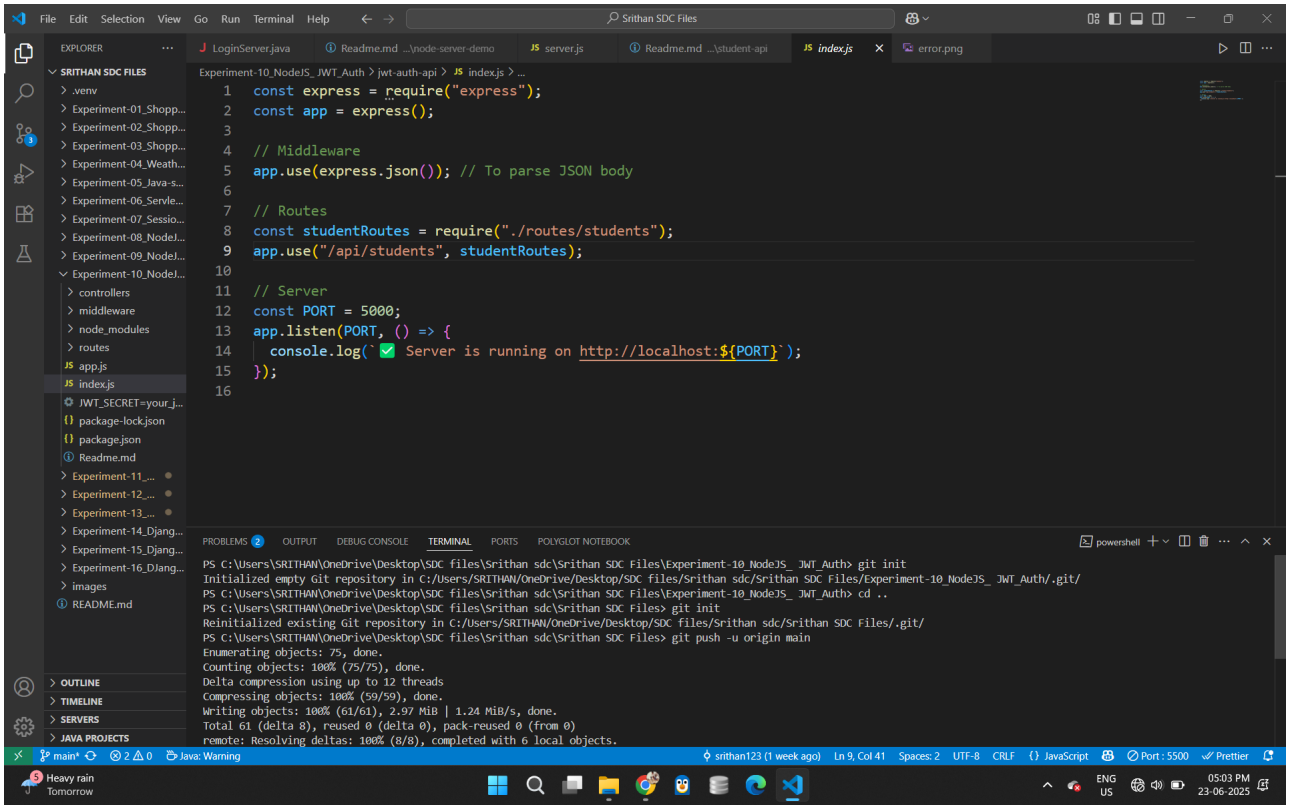
```
JWT_SECRET=YOUR_VERY_STRONG_JWT_SECRET_HERE_USE_A_RANDOM_STRING
```

(Tip: You can generate a strong secret using `openssl rand -base64 32` in a Unix-like terminal).

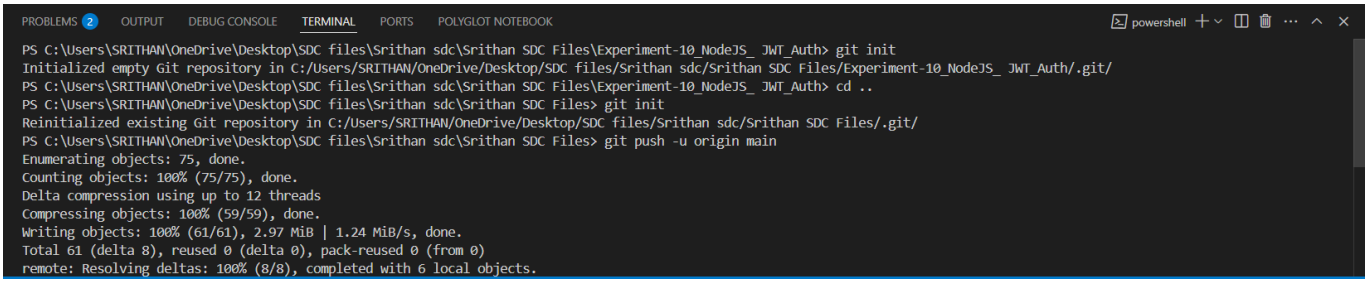
5. **Create `server.js`:** Create a file named `server.js` in the project root and paste the provided conceptual code.
6. **Start the API Server:** In your terminal, run:

```
node server.js
```

You should see messages indicating the server is running and dummy passwords are hashed.



API Endpoints and Usage (with Postman)



The API server will be running on `http://localhost:3000`.

1. User Login (Obtain JWT)

This is the first step. You need to log in to get a token to access other protected endpoints.

- **Method:** `POST`
- **URL:** `http://localhost:3000/api/login`
- **Headers:** `Content-Type: application/json`
- **Body (raw JSON):**

```
{
    "username": "testuser",
    "password": "password123"
}
```

(A second dummy user is `admin` with password `adminpass`)

- **Expected Response (200 OK):**

```
{
    "token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOjEsInVzZXJuYW1lIjoidGVzdH
VzZXIiLCJpYXQiOjE2..."
}
```

Copy the entire `token` string from the response.

2. Access Protected Student Endpoints (Using the JWT)

For all the following student CRUD operations, you **MUST** include the obtained JWT in the `Authorization` header.

- **Headers to add for all protected requests:**
  - `Authorization: Bearer <PASTE_YOUR_OBTAINED_JWT_TOKEN_HERE>`
  - `Content-Type: application/json` (for `POST` and `PUT` requests only)

Example: Get All Students (Protected)

- **Method:** `GET`
- **URL:** `http://localhost:3000/api/students`
- **Headers:**

```
Authorization: Bearer eyJhbGciOiJIUzI1Ni... (Paste your token here)
```

- **Expected Response (200 OK):** An array of student objects (if authenticated).
- **Error Responses:**
  - `401 Unauthorized`: If `Authorization` header is missing or token is malformed.
  - `403 Forbidden`: If the token is invalid (e.g., expired, tampered with).

Other Student Endpoints (POST, PUT, DELETE):

Apply the same `Authorization` header and request bodies (as detailed in Experiment 9) to the following URLs:

- `POST /api/students`
- `PUT /api/students/:id`
- `DELETE /api/students/:id`

Project Structure

. ├── server.js # Main Express application file with JWT authentication logic ├── .env # Contains the JWT_SECRET environment variable ├── package.json # Project metadata and dependencies ├── package-lock.json # Records exact dependency versions └── README.md