JWT Authentication for Secure Banking API Endpoints

1. Aim

The aim of this project is to implement a robust and stateless authentication mechanism using **JSON Web Tokens (JWT)** to secure sensitive API endpoints simulating a modern banking application. This involves generating tokens upon successful login and validating these tokens in a custom middleware to ensure only authenticated users can access financial data.

2. Objectives

Upon completion of this project, the following objectives should be met:

1. **JWT Generation:** Successfully generate a cryptographically signed JWT containing user claims (payload) upon a simulated user login.
2. **Secure Middleware Development:** Create and integrate a custom middleware capable of extracting a JWT from the Authorization header and verifying its signature and integrity.
3. **Stateless Security:** Demonstrate the principle of stateless authentication where the server relies solely on the token's signature without needing to query a session store or database on every request.
4. **Route Protection:** Apply the JWT authentication middleware selectively to banking routes (e.g., balance check) to enforce security.

3. Theory

A. JSON Web Token (JWT)

A JWT is a compact, URL-safe means of representing claims to be transferred between two parties. The claims in a JWT are encoded as a JSON object and are digitally signed using a secret (or a public/private key pair).

A JWT consists of three parts, separated by dots (.):

1. **Header:** Contains the token type (JWT) and the signing algorithm (e.g., HS256).
2. **Payload (Claims):** Contains the user data (e.g., user ID, roles) and metadata (e.g., expiration time, issuance time).
3. **Signature:** Created by taking the encoded Header, the encoded Payload, and a secret key, and signing them with the algorithm defined in the header. The signature is used by the server to verify that the token has not been tampered with.

B. JWT Authentication Flow

1. **Login:** The client sends credentials (username/password) to a public /login endpoint.
2. **Generation:** The server validates the credentials and, if successful, uses a secret key to sign and issue a JWT back to the client.
3. **API Access:** The client stores the JWT and sends it in the Authorization: Bearer <token> header for subsequent requests to protected routes.
4. **Validation (Middleware):** The middleware on the server extracts the token, uses the same secret key to verify the signature, and, if valid, grants access to the route handler.

4. Procedure

This project utilizes Node.js and the Express framework, along with the widely used jsonwebtoken library.

1. **Environment Setup & Dependencies:**
   - Initialize project: npm init -y.
   - Install necessary packages: npm install express jsonwebtoken.
2. **Secret Key Management:** Define a secure, secret key (JWT_SECRET) which must be kept confidential and stored in an environment variable (simulated in jwt_server.js).
3. **Login Endpoint:** Implement a /login route that accepts mock credentials and uses jwt.sign() to create a token.
4. **Authentication Middleware:** Implement an authenticateToken function that:
   - Checks for the Authorization header.
   - Extracts the token.
   - Uses jwt.verify() with the secret key to check validity.
   - If valid, attaches the decoded user information to the request object (req.user).
5. **Secure Routes:** Create mock banking API endpoints (e.g., /api/account/balance) and apply the authenticateToken middleware to them.
6. **Server Execution:** Run the server using: node jwt_server.js.
7. **Testing:** Test the flow: request a token from /login, then use that token to access the protected API routes.

5. Code

The complete Node.js/Express server implementation is provided in the jwt_server.js file, demonstrating the login, token generation, and token validation logic.

6. Output

A. Successful Login and Token Generation

Action	Request (Body)	Response Status	Response Body
Login	{ "username": "user1",	200 OK	{ "token": "eyJhbGciOiJIUzI1N

	"password": "password123" }		iI..." } (Example token)

B. Accessing Protected Endpoint

Action	Request (Header)	Response Status	Response Body
Authorized Access	Authorization: Bearer <Valid JWT>	200 OK	{ "account_number": "123456", "balance": 15000.75 }
Unauthorized (Missing Token)	(No Authorization header)	401 Unauthorized	Access denied. No token provided.
Unauthorized (Invalid Token)	Authorization: Bearer <Tampered JWT>	403 Forbidden	Invalid Token.

7. Learning Outcomes

1. **Cryptographic Signature & Statelessness:** Learned how JWTs enable scalable, stateless authentication by offloading session data storage to the client and ensuring integrity using a shared secret key for verification.
2. **Error Handling for Security:** Gained experience in implementing robust security checks, including parsing headers, differentiating between missing tokens (401 Unauthorized) and invalid/expired tokens (403 Forbidden), and providing appropriate feedback.
3. **Practical Library Usage:** Mastered the use of the jsonwebtoken library for the two core functions: jwt.sign() for generation and jwt.verify() for validation, a critical skill for backend development.