Note of Backend

2025年11月12日 21:39

1. Set Up Backend folder
    a. Addition tools:
        i. express mongoose dotenv jsonwebtoken bcryptjs cookie-parser cloudinary socket.io
2. Starts an HTTP server (index.js)
3. Create an **Route** for…
    a. An "address book"
    b. Map a URL path + HTTP method(GET,POST, PUT…) to a function.
4. Create a **Controller** for…
    a. The "Chef" who cooks the logic
    b. Actual code: input, talk to DB, hash pw, create JWT, set cookie, sd res
5. Setup Database
6. Create **Model**
    a. Defines "Schema"
    b. A schema/blueprint for data in the database. Ensures data consistency, enables secure auth, powers CRUD
    c. E.g. email, fullName, password, profilePic
7. Handle **Cookies** and **Tokens** function
8. Create a **Middleware**
    a. A function that runs btw req & res
    b. Read JWT cookie, parse req body, "Protected route", "AsyncHandler"(Clean error), "ErrorHandler"

Testing in backend:
Respond: res.status()
if, try, catch

Security

Hashing password:
Use bcrypt to generate a random "salt" to combine with user's password.
Then runs through bcrypt algorithm, and produces a long, unreadable hash.
Save only hashedPassword in DB.

Sign in: type password, generate Salt, hash, save into DB
Login : type password, compare password with hashed password

bcrypt.compare(): extract salt from stored hash. Rehash input with same salt. Compare 2 hashes.

Hash String: Algorithm identifier($2b$), cost factor(10), salt(base64 encoded), the actual hash
Rehash same salt: With same salt and cost of stored hashed, rehash the input password.

JWT(Json Web Token):
Core of secure authentication, a secure way to send user identity btw frontend & backend
A small, signed string that proves: "User logged in & has permission."
Can sign expire day.
User logs in:
Server checks password (bcrypt.compare) -> generate JWT(jwt.sign({id}, 'secret')) -> send JWT in HTTP-only cookie -> Frontend makes API calls w/ cookie -> Server verifies JWT (jwt.verify()) -> Allow access to protected routes

HTTP-only cookie:
Store JWT in an HTTP-only cookie.

Cookie-parser:
how Express server reads the JWT from the HTTP-only cookie that the browser automatically sends.
Cookie-parser read JWT from cookies -> verify user -> allow access

Browser Stores & sends cookies:
Server send Cookie(with token in it) -> Browser save in memory -> Browser auto-attaches cookie

API calls
Frontend asking Backend for data or action

Protected routes
API endpoints that only logged-in users can access.

Cloudinary:
A cloud platform to upload, store, transform, and deliver images & videos

expressJS = Node.js web application framework
For: build web app& APIs
Offer routing system (good for backend)
HTTP helpers (redirect, caching)
Support middleware to reapond to HTTP rendering
Error handling middleware

Mongoose: opensource NoSQL File database

Dotenv: zero-dependency module that loads environment variable from .env file into process.env

Jsonwebtoken(jwt): transfer data through-> json object encode
Encode through HMAC, RSA, ECDS algtm…
Used in users authorization.
1. User req server for jwt.
2. Req source from source server w/ that JWT
3. If JWT approved, source get

Decoded JWT include: Header, payload and verify signature

Bcryptjs: salt(random add some ramdon data into it) the user datas to avoid database leak
Dis: inreverse

cookie-parser: parse cookies from the Cookie header in incoming requests
Signed Cookies (Secure) to prevent tampering

Cloudinary: cloud-based media management platform that provides an SDK and API to easily upload, store, transform, optimize, and deliver images and videos in your Node.js applications

socket.io: real-time, bidirectional, event-based communication library for Node.js
enables live data exchange between clients and servers using WebSockets

Middleware(Node.js): add and reuse
common functionality across routes &
endpoint

Application-level Middleware: using app.use() or app.METHOD()
Cases: Logging, authentication, request parsing, and other operations that should run for every req.

Router-level Middleware: express.Router()
Cases: Grouping route-specific middleware, API versioning, and orginizing routes into logical groups
Ads: code orgiz, modular routing, apply middleware to each route gp.

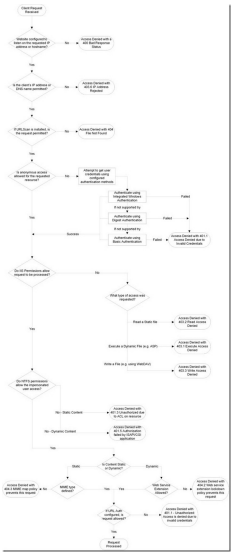
Error- handling Middleware: (err, req, res, next)
Cases: handle error that occur during req processing

Built-in Middleware:
Express.json(), express.urlencoded(), express.static(), express.Router()

Third-party Middleware:
Helmet: Secure app by setting various HTTP headers,
Morgan: HTTP request logger,
CORS: Enable CORS w/ various options
Compression: compress HTTP res
Cookie-parser: Parse Cookie header & populate req.cookies

Signed cookies: HTTP cookies that have a cryptographic signature attached to them to prevent tampering by the client (browser).

IIS 6.0 HTTP Request Processing



Request processing: lifecycle of handling a client's input.
The sequence of steps a system takes to receive, interpret, handle, and respond to an incoming request from a client.
Core concept: Web development, API design, networking, and distributed systems

Request Arrival:
Client send req to server include…
1. Method
2. URL/Endpoint
3. Headers
4. Body

Routing:
server determines which part of the application should handle the request.

Middleware:
Pre-processing steps run before the main logic

Business Logic Execution:
The core functionality runs…
1. Query a database
2. Call external APIs
3. Perform calculations
4. Generate dynamic content

Response Construction:
The server builds a response…
1. Status code (200 OK, 404 Not Found, 500 Error, etc.)
2. Headers (Content-Type: application/json, etc.)
3. Body (HTML page, JSON data, file stream, etc.)

Response Sent:
The response is transmitted back to the client over the network…

Request parsing: Early step in request processing.
process of analyzing and extracting meaningful data from an incoming client request (usually HTTP) so that the server or application can understand and act on it.