ExpressJS

Express.js is a web application framework for Node.js. It simplifies building web applications and APIs by providing a set of features and tools to handle HTTP requests, manage routing, and integrate middleware.

Setting Express.js :

- Install Node.js
- Initialize a node.js project
- Install express.js (npm i express)
- Create an Entry File (eg = app.js)
- Set Up a Basic Express Server

Basic Routing :

Syntax ; app.method(path,handler)

- App : instance of express
- Method : An HTTP method(get, post, put, delete)
- Path : Route path (url)
- Handler : function to execute when route is matched.

Parameters :

- *Route Parameters* :
  Route parameters are part of the URL itself and are used to capture values in dynamic URLs. They are defined by adding a colon : before the parameter name in the route.
  Eg ;
```
app.get('/user/:id', (req, res) => {
    const userId = req.params.id;
    res.send(`User ID is: ${userId}`); });
```
- *Query Parameters :*
  They are appended to the URL after a question mark (?) and are typically used for filtering, searching, or sorting data.

Syntax;
http://example.com/path?key1=value1&key2=value2&key3=value3

- *Body Parameters:*
  They are used to send data in the body of POST, PUT, or DELETE requests.

Router :

It acts like a "mini-application" that allows you to define routes and middleware handlers and then attach them to your main app instance.

Middlewares :

It's a function that is going to run between the time the server gets the request and the time that the sends the response out to the client. They have access to the request , response , and the next middleware function in the request-response cycle. These functions can modify the req and res objects, end the request-response cycle, or pass control to the next middleware function using next()[order of execution of middleware is same as the order in which they are defined].

Syntax ;

```
app.use((req, res, next) => {
  // Middleware logic
  next();  // Call next() to pass control to the next middleware
});
```

Handling HTTP Methods :

- GET : Retrieves data from the server.
- POST : Submits data to the server.
- PUT : Updates or replaces existing data entirely
- DELETE : Deletes data from the server

Template Engines :

It allow you to create dynamic HTML pages by inserting data from the server into your HTML files [Template engines make it easier to inject dynamic data into static HTML pages in Express.js] .

- Pug : uses indentation-based syntax instead of HTML tags to render HTML
- EJS (embedded JS) : embed JavaScript directly in your HTML using <%= %> syntax

Static Files Serving :

In Express, you can serve static files (like images, CSS, JavaScript, etc.) using the built-in express.static middleware.

Syntax : app.use(express.static("public"))

Here, its going to serve all the files from public folder

JSON Responses :

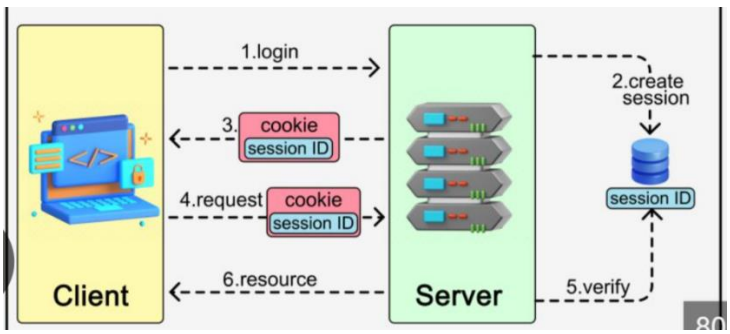
Express provides a built-in method called res.json() to send JSON-formatted data as a response.

Syntax : res.json(data)

In express, to parse JSON data, we use **express.json()** [it's a middleware]

Syntax : app.use(express.json());

Session Management and Cookies :



1. A user makes a request to the server
2. The server creates a session and generates a unique **session ID**.
3. The session ID is sent back to the client (browser) as a **cookie**
4. For subsequent requests, the client sends the session ID back to the server, allowing the server to retrieve session data for that user.

Express.session middleware is used for handling sessions.

Environmental variables and configuration :

Store sensitive information (like API keys) outside the codebase.

dotenv : A popular package for loading environment variables from a .env file

Middleware Libraries :

body parser :

Express.js comes with express.json() and express.urlencoded() built-in body parsers.

It parses incoming request bodies.

bodyParser.json() = will convert incoming JSON data to js object.

bodyParser.urlencoded() = converts url encoded data(key=value) to a js object.

morgan : Logging middleware

It logs incoming requests and responses(method, status code, response time, and more).

cors : cross origin resource sharing

When you need your Express.js server to handle requests from other domains, **CORS** must be enabled.

It helps maintain security while allowing necessary interactions between different domains. By using the CORS middleware in Express,

you can easily manage cross-origin requests and ensure that your application functions as expected.

File Uploads Handling :

One of the most commonly used middleware for handling file uploads is **Multer**.

Multer configuration =

- **Storage**: You define how files are store.
- **Destination**: The folder where uploaded files will be saved.
- **Filename**: The naming convention for the uploaded file.

Authentication and Authorization(passport.js) :

Authentication is the process of verifying a user's identity.

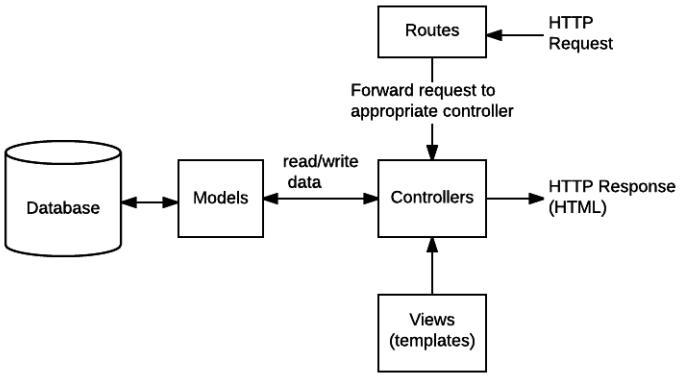
Authorization is the process of determining what an authenticated user is allowed to do

Passport.js is a popular authentication middleware for Node.js that helps you manage authentication in your Express.js applications.

Authentication and Authorization(jwt) :

JWT (JSON Web Token) in a Node.js/Express application typically involve creating a secure authentication system where users can log in, receive a token, and access protected routes based on the token.

Router Handling Architecture :



RESTful Api :

REST, which stands for Representational State Transfer, is an architectural style for designing networked applications.

RESTful APIs have a uniform interface, which simplifies communication between clients and servers. This interface typically involves the use of standard HTTP methods, resource identifiers (URLs), and representations.

In ExpressJS, you define routes to handle incoming requests for specific resources and HTTP methods. Each route specifies a callback function to process the request and send an appropriate response.

Securing Express.js Applications :

Helmet Js :

Helmet helps secure Express apps by setting various HTTP headers, making it harder for attackers to exploit vulnerabilities in the app.

Ratelimit :

express-rate-limit helps protect your app from brute-force attacks by limiting the number of requests a client can make to your server in a given timeframe.

Syntax :

```
const limiter = rateLimit({
 windowMs: 15 * 60 * 1000,
 max: 100,
message: "Too many requests from this IP, please try again after 15 minutes." });
```

Validating Input data :

- express-validator :

Testing Express Applications :

Mocha :

A feature-rich JavaScript test framework that runs on Node.js and allows you to structure your test cases in a simple way.

Chai :

An assertion library for writing clear and expressive test assertions. Can be used with different testing frameworks.

Supertest :

A library for testing HTTP servers by sending requests and asserting responses.

Jest :

Jest is a JavaScript testing framework that allows developers to write and execute tests for their code. It is often used in combination with front-end libraries like React but can also be used for back-end testing (Node.js and Express).

Characteristics of jest :

- Zero config
- Isolated
- Snapshots
- Rich Api