**Week 3**

**3.0 Overview**

* Express advance for authentication. Username, password verification.
* app.use(express.json()) 🡪 related to middlewares.
* global caches 🡪 if there’s a crash on our server.
* There are multiple NoSQL and SQL databases. MongoDB is one NoSQL database.
* MongoDB compass is a GUI that helps you to visualize your MongoDB database.
* MongoDB cluster password: melIMNJefhE7t0t0
* MongoDB Compass connection string: mongodb+srv://admin:melIMNJefhE7t0t0@cluster0.bulkj26.mongodb.net/
* Postgres NeonTech connection string: postgresql://saad.salman11123:McdX4xqmWY0U@ep-mute-heart-17643668.us-east-2.aws.neon.tech/test?sslmode=require

**3.1 Middlewares, Zod, and Global Catches**

* The use case of middleware is to do pre-checks. Pre-checks are of two types: authentication and input validation.
* There are multiple ways to send input to a route handler like through query parameters, body, headers etc.
* Cleaner way to write is do your checks and return early.
* We can have multiple callback functions in each route handler function. Each callback function also has next along with req and res. next isa function which routes the control to the next pre-check.
* You cannot respond to an HTTP request more than once.
* To get the data, you use req in the middleware function.
* res.send is used for sending text and res.json is used for sending a JSON object. Usually, we use JSON.
* Rate limiter and calculate requests are commonly used middlewares.
* Using app.use(middleware) means that every route handler after that line would have the middleware called. express.json() returns a function.
* Every middleware should have next.
* Our backend server is on the internet, and anyone can hit it. If we don’t do input validations, our server can break.
* Global catch needs to come at the end. It gets called whenever there’s an exception. They are called error-handling middleware.
* Next is also present in global catch to send the control to some other place.
* Coercion means to push someone to do something.
* Normal middleware in app.use will hypothetically take first slot among callbacks in all routes.
* You can never trust frontend checks; you should always implement them on the backend.
* If the error is in middleware1, then the control would move to the exception middleware and no more middlewares would be called.
* Tokens are usually stored in the browser (local storage).
* Server usually doesn’t check the username and password, but checks the token that is send each time.
* Authorization in headers is a token.
* Check Zod in detail: [Basic usage (zod.dev)](https://zod.dev/?id=basic-usage)
* safeParse does not generate error, but parse does.
* If middlewares is an array of middlewares, then we can write …middlewares to spread the individual middlewares and pass them to the route. 3 dots basically spread the array.
* Whenever we’re sending get requests, we use query parameters. Whenever we’re sending post request, we use body. Whenever we’re sending sensitive information like tokens, we use headers.
* Cookies is also sensitive information and is also in header.
* When you need to send data from middleware1 to middleware2, you shove it in req using req.data = 1. In middleware2, you now have access to req.data.
* This is all backend logic.
* Good practice is to have all your middlewares, then your routes, then your global catches. This is also the order in which they run.
* You have to be smart to not write logic after the respond is sent back.

**3.2 Fetch, Database, and Auth**

* Create a script tag and write whatever JS you want to write.
* See how to create a button and code.
* The default method for fetch is GET.
* Fetch is an asynchronous call.
* Console.log in an HTML file would produce output in browser’s console.
* This is all frontend.
* Rendering data means putting data on the screen 🡪 DOM.
* Eventually frontend and backend will be connected through fetch.
* Difference between encryption and hashing is that encryption is two ways. Encryption uses a key to lock the data and then the same key is used to decrypt the data. Images are encrypted, chats are encrypted. Hashes are irreversible.
* JSON Web Tokens (JWT) 🡪 works only on JSON, significantly different from hashing and encryption, the long string it gives you back is called the token, anyone can look at the string and figure out what were the original contents.
* Use jwt.io to try it on your ChatGPT’s authorization.
* JWT also use a password like encryption.
* The encoding returns a string which has 3 parts. This string is called the token. Only the website that has produced the token can verify the token using the specific password to produce the original object.
* In encryption, the string can only be decoded by the person who has the password. In JWT, the string can be decoded by anyone, but only be verified by the person who has the password. These passwords are never exposed.
* The token usually resides in browser’s local storage. Even if the browser is closed or the machine is shut down, the token remains. Whenever a request is sent, the token is sent, until and unless you log out.
* This is one way of doing authentication (local storage way). Another way is using cookies.
* The backend creates the JWT.
* Only the backend which created the JWT can verify it, because it has access to the specific password. Only the server that signed it can verify it. However, it can be decoded by anyone.
* JWT saves the query call for verification.
* Check code for example of JWT. We have used two functions: sign and verify.
* Full Stack applications usually use SQL.
* NodeJS process need to connect to MongoDB Compass to write data to database 🡪 Mongoose.
* Different JWTs would be created for different passwords. Although all of them can be decoded, only the one with the correct password can be verified.
* The text upfront in some tokens tells the type of the token.
* Why DBs are better than files:

1. You don’t have a standard way to store data.
2. Hard to distribute.
3. DBs are optimized for read and write.
4. DBs have logic for indexing to make some queries faster.

* Cookies are slightly better but are only browser based.

**3.5 DOM Introduction**

* DOM stands for Document Object Model.
* DOM is language independent.
* The main object is the document object, which in turn houses other objects which also house their own objects, and so on.
* The objective of DOM is to connect HTML with JS.
* document.querySelectorAll(‘p.intro’). Anything after . denotes class. This returns a node list.
* W3 Schools CSS Selectors.
* For live list, we use HTML collections.
* documentElement returns the root of the document. For HTML document, it is the HTML.
* Only a for loop can access the elements returned by querySelectorAll().
* Do not use document.write after your document is loaded because that would overwrite your document.
* When updating style, it is better to use .style instead of setAttribute because setAttribute would overwrite all of the styles.
* Document node, element nodes, attribute nodes, text nodes.
* Element nodes do not hold text, but rather hold text nodes.
* Bubbling is how a created method is applied. That is it is from the innermost element to the outermost element. Bubbling is how events are usually applied.
* By default, useCapture is false.