Agenda

- express Router
- Middleware
- bCrypt
- ~~JWT~~

express.Router

- express.Router() is a mini Express app, just for handling routes.
- used to create modular, mountable route handlers.
- It lets us break our app into modules so instead of defining all routes in server.js, we can separate them into different files (like users.js,products.js,etc).
- A Router instance is a complete middleware and routing system.

```
// In user.js file

const express = require("express");
const router = express.Router();
// GET /users/
router.get("/", (req, res) => {
  res.send("User List");
});

// GET /users/:id
router.get("/:id", (req, res) => {
  res.send(`User ID: ${req.params.id}`);
});
module.exports = router;
```

```
// In server.js
const express = require("express");
const app = express();
const userRouter = require("./user");

app.use("/users", userRouter); // Mount user routes at /users

app.listen(3000, () => console.log("Server running on port 3000"));
```

SUNBEAM INFOTECH

Middleware

- Middleware is a function that runs between the request and the response in a web application.
- It is commonly used in backend frameworks like Express.js to modify, process, or control HTTP requests before sending a response.
- It Access Request (req) and Response (res) Objects
- It Modify Requests/Responses → Example: Adding headers, parsing JSON

- Run Code Before Sending a Response → Example: Authentication, Logging
- Call next() to Continue Execution → Passes control to the next middleware
- It is commonly used in express for

1. Handling CORS (cors package)
2. Parsing JSON data (express.json())
3. Authentication & authorization

Password Hashing

- For storing user passwords, you should always use bcrypt (or a similar specialized hashing library like Argon2) instead of CryptoJS.
- While CryptoJS offers a wide range of cryptographic tools, it is not designed for the specific needs of password hashing and has known vulnerabilities in that application
- It is specifically designed to be slow and resilient against attacks, automatically handles salting, and has an adaptive cost factor to stay ahead of advancing hardware.
- For a secure authentication system, a specialized password-hashing library like bcrypt is the definitive best practice.

crypto-js

1. add/install the crypto-js module

```
npm install crypto-js
#OR
yarn add crypto-js
```

2. Usage

```
// import the module crypto-Js
const cryptoJs = require("crypto-js")

// use the encryption method to encrypt the password
const encrypted = cryptoJs.SHA256(password)
```

SUNBEAM INFOTECH

bcrypt

1. add/install the bcrypt module

```
npm install bcrypt
#OR
yarn add bcrypt
```

2. Usage

```
// import the module bcrypt
const bcrypt = require("bcrypt")

// In the registration or signup part of express
// use the hashing techique method to hash the password
bcrypt.hash(password, saltRounds)
    .then((hashPassword) => {
        // If hashPassword is generated then insert in the db
    }).catch((error) => {
        // If hashing is not able to generate
    })


// compare the user given password with the hash password in db
const sql = `SELECT * FROM users WHERE email = ?`
pool.query(sql, [req.body.email], (error, data) => {
if (data != '') {
    const dbuser = data[0]
    bcrypt.compare(req.body.password, dbuser.password)
    .then((result) => {
        if (result)
            res.send({ status: "success", data })
        else
            res.send({ status: "error", error: 'Invalid password' })
    })
} else
    res.send({ status: "error", error: 'Invalid email' })
})
```

SUNBEAM INFOTECH

How CryptoJS works

1. Collection of Algorithms:
    - CryptoJS offers a variety of algorithms for different use cases.
    - For hashing, it provides functions like SHA256() or MD5().
    - For encryption, it has functions like AES.encrypt() and AES.decrypt().
2. PBKDF2 Functionality:
    - While CryptoJS includes PBKDF2, a password-based key derivation function, its default settings are highly insecure for password hashing and verification.
    - To use it securely, a developer must manually generate a salt and configure a sufficiently high number of iterations.
3. General-purpose API:
    - The library's methods are not specialized for one task.
    - This means developers must have a strong understanding of cryptographic best practices to use it securely for any given purpose.

How bcrypt works

1. Salt Generation:
    - The bcrypt library automatically generates a unique, random salt for each password.

- This salt is combined with the user's password before hashing.
- This ensures that even two users with the same password will have completely different hashes stored in the database, protecting against "rainbow table" attacks.

2. Adaptive Work Factor (Cost):
    - A customizable "cost factor" or "salt rounds" determines how computationally expensive the hashing process is.
    - As computing power increases over time, the cost factor can be increased, ensuring the hashes remain secure against evolving hardware.
    - The bcrypt npm package provides simple methods to generate a hash with a work factor and compare a plain-text password against a hash.
3. Output Format:
    - The final output is a single string that combines the algorithm version, the cost factor, and the salt and hash values.
    - This simplifies storage because you only need one column in your database to store everything necessary to verify a password.

SUNBEAM INFOTECH