Tech hub – Gray Paper

Utilizing express js to render each page. Each folder structure is set individually, while the view engine renders the ejs files according to the get methods which renders each linked html page.

```
const app = express()
// convert into json format
app.use(express.json());

app.use(express.urlencoded({extended: false}));

app.set('view engine', 'ejs');
app.use(express.static("public"));
app.use(express.static("Images"));
app.use(express.static("src"));
app.use(express.static("views"));
//get each page and render
app.get("/", (req, res) => {
    res.render("index");
})

app.get("/index", (req, res) => {
    res.render("index");
})
app.get("/techspot", (req, res) => {
    res.render("techspot");
})
app.get("/forum", (req, res) => {
    res.render("forum");
})
app.get("/help", (req, res) => {
    res.render("help");
})
app.get("/login", (req, res) => {
    res.render("login");
})
```

We listen on a port, which is used to locally host the website.

```
// port
const port = 3000;
app.listen(port, ()=> {
    console.log(`Port: ${port} is connected...`);
})
```

Async/await is used in a method to obtain the username and password. The dom is used to access the html document (receives input data), which in turn accepts user input on the webpage. The async function is used; however, the await keyword is used to resolve the promise before continuing, as a check condition is used to see if the user exists. The await keyword ensures that the username and password are collected before continuing with the execution of the code. Bcrypt is then utilized to hash the accepted password before pushing the data to the database. This also uses await in order to ensure the data is encrypted by the hashing algorithm, also setting a number of salt rounds to take place to hash the password.

```
// registration
app.post("/registration", async (req, res) => {
    const data = {
        name: req.body.username,
        password: req.body.password
    }
    // check existing user
    const existingUser = await collection.findOne({name: data.name});

    if(existingUser) {
        res.send("User already exists");
    } else {
        // bcrypt
        const saltRounds = 10; // number of salt rounds for bcrypt
        const hashedPassword = await bcrypt.hash(data.password, saltRounds);
        data.password = hashedPassword; // replace hashed pw with orginal
        // add to database
        const userdata = await collection.insertMany(data);
        console.log(userdata);
    }
})
```

On the login page, async/await it used to make sure the data is collected and/or hashed before checking to see if the username and password exists or whether the password entered is correct.

```
//login
app.post("/login", async (req, res) => {
    try {
        const check = await collection.findOne({name: req.body.username});
        if(!check) {
            res.send("Username cannot be found")
        }
        // compare hashed password
        const isPasswordMatch = await bcrypt.compare(req.body.password, check.password);
        if(isPasswordMatch) {
            res.render("index");
        } else {
            req.send("Wrong password");
        }
    } catch {
        res.send("Wrong credentials")
    }
})
```

The chat bot on the FAQ page uses the dom to register user input , sending it to the openai api, and having a response generated. The generated response is returned. The code uses json format to acquire, send, and receive data.

```
// dom to handle chat messages
const chatInput = document.querySelector(".chat-input textarea");
const sendChatBtn = document.querySelector(".chat-input span");
const chatbox = document.querySelector(".chatbox");
// user message
let userMessage;
// open ai
const API_KEY = "sk-proj-X5UPLpviCijGJ9uWmTEsT3BlbkFJIWsz4vRtJtbcu39Awd4s";
// create element to pass message and class name
const createChatLi = (message, className) => {
    const chatLi = document.createElement("Li");
    chatLi.classList.add("chat", className);
    let chatContent = className === "outgoing" ? `<p>${message}</p>` : `<span class="material-sym
    chatLi.innerHTML = chatContent;
    return chatLi;
}
// api response
const generateResponse = (incomingChatLi) => {
    const API_URL = "https://api.openai.com/v1/chat/completions";
    const messageElement = incomingChatLi.querySelector("p");

    const requestOptions = {
        method: "POST",
        headers: {
            "Content-Type": "application/json",
            "Authorization": `Bearer ${API_KEY}`
        },
        body: JSON.stringify ({
            model: "gpt-3.5-turbo-0125",
            messages: [{role: "user", content: userMessage}]
        })
    }
```

A setTimeout function is used to delay the message to simulate the chat box typing/thinking a response. A listener event is used to register when the submit button is clicked or the user uses the enter key to submit

```
// message handler
const handleChat = () => {
    userMessage = chatInput.value.trim();
    if(!userMessage) return;
    // append the message to the chatbox
    chatbox.appendChild(createChatLi(userMessage, "outgoing"));
    // show chat bot is typing
    setTimeout(() => {
        const incomingChatLi = createChatLi("Tech hub bot is typing...", "incoming")
        chatbox.appendChild(incomingChatLi);
        generateResponse(incomingChatLi);
    }, 500);
}
sendChatBtn.addEventListener("click", handleChat);

chatInput.addEventListener('keypress', function(event) {
    // Check if the key pressed is Enter
    if (event.key === 'Enter') {
        // Call the handleChat function
        handleChat();
    }
});
```

A setInterval function is set to delay the transition to the next photo, creating a scroll of pictures.

```
function startAutoScroll() {
    intervalId = setInterval(function() {
        moveToNextSlide();
    }, 5500); // interval in milliseconds
}
```

Creates a module to connect to the mongo database in order to push the username and password. A schema is used to organize the username and password data in the database. That data is then pushed to a collection within the database to store and display the information enter by the user.

```
const mongoose = require("mongoose");
const connect = mongoose.connect("mongodb://localhost:37017/TechHub");

//check database connection
connect.then(() => {
    console.log("Database connected successfully")
})
.catch(() => {
    console.log("Database cannot be connected")
})

// create schema
const LoginSchema = new mongoose.Schema({
    name: {
        type: String,
        required: true
    },
    password: {
        type: String,
        required: true
    },

});
// collection
const collection = new mongoose.model("users", LoginSchema);

module.exports = collection;
```

Challenges include:

- formatting the site to have symmetric structure
- Understanding Express, its syntax, and folder structure
- Setting up database using Docker and Mongodb, pulling images
- Utilizing javascript to send data to database
- Setting up chatbot to input, send, and receive response

Solutions:

- Restructing html and using more classes to ease formatting within CSS
- Utilized several resources to understand how to implement different APIs
- Understanding nodemon, bcrypt, express, mongoose, and ejs
- Trial and error: e.g, getting the database to register user input