Transcript File

Prompt 1:

The objective of this assignment is to introduce students to backend development with Express.js and TypeScript. Students will build a RESTful API to manage user registration, login, and fetching user details. This assignment focuses on setting up an Express server, creating endpoints, and applying middleware for logging and error handling. Scenario You are now working on the backend for the Learning Management System (LMS). The system needs a secure and functional backend to handle user data. As a first step, you need to create a RESTful API to manage user registration, login, and basic user data retrieval. This API will form the foundation for user management in the LMS. Requirements Step-by-Step Instructions 1. Set Up an Express.js Server Create the Project Folder Structure: Module2\_Assignment/ ├── src/ │ ├── server.ts │ ├── routes/ │ │ └── userRoutes.ts │ ├── middleware/ │ │ └── logger.ts │ │ └── errorHandler.ts │ └── models/ │ └── user.ts ├── package.json ├── tsconfig.json ├── postman\_collection.json └── README.md Initialize the Project: Open your terminal and create a new project directory: mkdir Module2\_Assignment cd Module2\_Assignment npm init -y Install required packages: npm install express npm install -D typescript @types/express ts-node nodemon Initialize a TypeScript configuration file: npx tsc --init Update the tsconfig.json file Create the Express Server: Create a file named server.ts in the src/ folder. Set up a basic Express server: 2. Create API Endpoints Define a User Interface: Create a file named user.ts in the models/ folder. Define a User interface: Set Up User Routes: Create a file named userRoutes.ts in the routes/ folder. Define routes for POST /register, POST /login, and GET /user/:id: Integrate Routes with the Server: In server.ts, import and use the userRoutes: 3. Middleware Implementation Logging Middleware: Create a file named logger.ts in the middleware/ folder. Implement request logging: Add this middleware to the server in server.ts: Error Handling Middleware: Create a file named errorHandler.ts in the middleware/ folder. Implement basic error handling: Add this middleware to the server in server.ts: 4. Create Postman Collection Test your API endpoints using Postman. Create a collection with three requests: POST /api/register with a sample user payload. POST /api/login with valid and invalid credentials. GET /api/user/:id with a valid user ID. Export the collection as postman\_collection.json and include it in your submission. Deliverables A working Express.js server with: POST /register POST /login GET /user/:id Middleware for: Request logging. Basic error handling. A Postman collection demonstrating API functionality. Submission Guidelines Files to Submit: src/server.ts src/routes/userRoutes.ts src/middleware/logger.ts src/middleware/errorHandler.ts src/models/user.ts postman\_collection.json Compress Files: Create a ZIP folder named Module2\_Assignment\_<YourName>.zip containing all files and folders. Upload: Submit the ZIP file to the designated LMS assignment portal. how to solve and submit this assignement

Answer given by the AI mentor:

Absolutely, I'll guide you through how to **approach and complete the Module2\_Assignment** and then make your submission via the LMS. Here’s a structured step-by-step plan, without giving direct solutions, so you can learn and implement it yourself.

**How to Approach the Assignment**

**1. Set Up the Project Structure**

* Follow the folder structure provided in your requirements.
* Use the terminal to create folders and files as specified.

**2. Initialize the Project**

* Run the following commands in your project directory:
* npm init -y
* npm install express
* npm install -D typescript @types/express ts-node nodemon
* npx tsc --init
* Update **tsconfig** as per TypeScript/Node recommendations for root directory and modules.

**3. Implement Server and Functionalities**

* **server.ts**
  + Set up the express app, listen on a port, and import middleware and routes.
* **models/user.ts**
  + Define a TypeScript interface for User (e.g., with id, name, email, password).
* **routes/userRoutes.ts**
  + Create and export express router.
  + Implement POST **/register** (store user), POST **/login** (authenticate), GET **/user/:id**.
* **middleware/logger.ts** - Middleware to log request method and URL.
* **middleware/errorHandler.ts**
  + Simple error handling middleware (capture and send error responses).
* Integrate all in **server.ts** (use logger, userRoutes, then errorHandler).

**4. Test with Postman**

* Useman to:
  + Send requests to **/register**, **/login**, **/user/:id**.
  + Save these requests into a Postman collection.
  + Export the collection as **postman\_collection.json**.

**5 Prepare Submission**

* Make sure your directory looks like:
* Module2\_Assignment/
* ├── src/
* │ ├── server.ts
* │ ├── routes/userRoutes.ts
* │ ├── middleware/logger.ts
* │ ├── middleware/errorHandler.ts
* │ └── models/user.ts
* ├── package.json
* ├── tsconfig.json
* ├── postman\_collection.json
* └── README.md
* Compress to a ZIP file as required.

**Submitting Your Assignment**

1. \*\* Files:\*\*
   * Zip your entire **Module2\_Assignment** folder, name it as instructed.
2. **Upload to LMS:**
   * Go to your assignment page: [Module2\_Assignment](https://lms.edubottechnologies.com/my/assignments/86693e01-1adb-455e-9c8a-224062c308e7/submissions)

* Click to upload your ZIP file and follow the prompts to submit.

1. **Verification:**
   * Visit [My Submissions](https://lms.edubottechnologies.com/my/submissions) to confirm your assignment was uploaded.

**Tips for Success**

* **Test each endpoint** with valid and invalid data in Postman to ensure all use cases work.
* **Comment your code** for clarity, especially around middleware and route logic.
* Double check your ZIP contains all required files.
* Refer to the [course details](https://lms.edubottechnologies.com/all/courses/9986a3c6-d9c3-499a-a6b5-9f10429f9e32) if you need to review any key concepts, like Express routing or TypeScript interfaces.

If you run into any issues while coding (syntax errors, debugging), you can use the **Launch Platform** via the LMS for instant help and AI-guided coding support.

You're well on your way to acing the assignment—good luck! If you have questions about any concept or get stuck, let me know and I’ll guide you further.

Prompt 2:

How can I implement the user registration, login, and user details endpoints in Module2\_Assignment?

Anwer given by the ai mentor

Great question! I’ll explain the structure, flow, and important considerations for implementing the **user registration, login, and user details endpoints** for your Module2\_Assignment. I won't provide the full code, but I’ll guide you with a clear approach so you can implement the logic yourself.

**1. User Registration Endpoint (POST /register)**

**Goal:**  
Allow new users to sign up.

**Steps:**

* Receive user data (like name, email, password) in the request body.
* Validate the data (type, required fields).
* Ensure no user already exists with the same email (if using in-memory storage, search your array or object).
* Hash the password (optional for learning, but it’s good practice—even with in-memory data).
* Save the user and respond (include their id or a success message, exclude the password in the response).

**Tip:**  
If you’re storing users in an array (since no DB yet), you could use a simple counter for **id**.

**2. User Login Endpoint (POST /login)**

**Goal:**  
Authenticate users by email and password.

**Steps:**

* Receive login credentials in the request body.
* Find the user by email.
* Compare the provided password with the stored password.
* If matched, respond with user details (excluding password) or a success flag/message.
* If no match, respond with an error status/message (e.g., 401 Unauthorized).

**3. Get User Details Endpoint (GET /user/:id)**

**Goal:**  
Fetch and return the details of a specific user by their ID.

**Steps:**

* Retrieve **id** from URL parameters.
* Search for the user in your storage by **id**.
* If found, return user details (without the password).
* If not found, respond with a 404 status and an error message.

**Common Patterns**

* **Use TypeScript interfaces** for type safety (import your **User** interface from **models/user.ts**).
* **Return appropriate HTTP status codes** (e.g., 201 for created resources, 200 for success, 400 for bad request).
* **Use your logging middleware** to log every incoming request (method, URL).
* **Use your error handling middleware** to catch errors and avoid repeating error response logic in each handler.
* **Never include passwords in your API responses.**

**Example Route Structure (Pseudocode)**

// In routes/userRoutes.ts

router.post('/register', (req, res, next) => {

// Validate input and register user

});

router.post('/login', (req, res, next) => {

// Find user, check password, return response

});

router.get('/user/:id', (req, res, next) {

// Find user by id and return details

});

**What to Avoid**

* Hardcoding user data—use an in-memory array or object.
* Returning sensitive information like passwords.

**Testing**

* Use **Postman** to try registering, logging in, and fetching user data.
* Test both successful and failing scenarios (e.g., login with wrong password, fetching a non-existent user).

**Extra Guidance**

If you’re unsure how to structure your routes and logic, review the [Module2 Material](https://lms.edubottechnologies.com/all/courses/9986a3c6-d9c3-499a-a6b5-9f10429f9e32) for Express routing examples.

**Let me know if you have questions about validation, route setup, middleware use, or need clarification on any step!**

Prompt 3:

How do I create and export a Postman collection for the API endpoints in Module2\_Assignment?

Answer given by ai mentor:

Absolutely! Here’s a step-by-step guide on **how to create and export a Postman collection** for your Module2\_Assignment API endpoints:

**Step 1: Open Postman**

* Launch the Postman app on your computer.

**Step 2: Create a New Collection**

1. Click **Collections** in the left sidebar.
2. Click the **“+ New Collection”** button.
3. Name your collection (e.g., **Module2\_Assignment API**).
4. (Optional) Add a description for your collection.

**Step 3: Add Requests to the Collection**

You need to create three requests: **Register, Login, Get User Details**.

**A. Register User (POST /register)**

* Click **Add a request** or create a new request tab.
* Set method to **POST**.
* Enter the URL (e.g., **http://localhost:3000/api/register**).
* Go to the **Body** tab, select **raw**, and choose **JSON**.
* Add a sample JSON payload:
* {
* "name": "Test User",
* "email": "test@example.com",
* "password": "password123"
* }
* Save this request to your new collection.

**B. Login (POST /login)**

* Create a new request, set to **POST**.
* URL: **http://localhost:3000/api/login**
* Body → raw → JSON:
* {
* "email": "test@example.com",
* "password": "password123"
* }
* Save to your collection.

**\*\*C. Get User Details (GET /user/:id)- After registering, get the user’s id from the registration response.**

* Create a new request, set to **GET**.
* URL: **http://localhost:3000/api/user/<userId>**
* Replace **<userId>** with the actual id value you received.
* Save to your collection.

**Step 4: Test All Requests**

* Send each request and check the responses.
* Ensure your API endpoints work as expected before exporting.

**Step 5: Export the Collection**

1. In the left sidebar, hover over your name.
2. Click the **three dots (more actions)** menu.
3. Select **Export**.
4. Choose **Collection v2.1 (recommended)**.
5. Save the file as **postman\_collection.json**.

**Step 6: Include in Submission**

* Place **postman\_collection.json** in your assignment folder with your code before zipping and submitting it.

**Extra Tips**

* Double-check your request bodies and endpoint URLs for typos.
* If your server runs on a different port, update the URLs in Postman accordingly.
* Document any sample users or IDs used in your [README.md](https://lms.edubottechnologies.com/README.md) for easier review.

You're all set! If you need help with a specific Postman error or how to structure a request, just ask!

Prompt 4:

How can I implement login authentication in Module2\_Assignment using Express and TypeScript?

Answer given by ai mentor:

Absolutely! Here’s a structured, step-by-step explanation of how you can **implement login authentication** for your assignment using **Express** and **TypeScript**. I’ll focus on the concepts and recommended logic rather than the exact solution—so you get to build your understanding.

**1. User Credential Storage**

Since you probably aren’t using a database yet, store users in an in-memory array or object.  
Make sure each user includes at least: **id**, **name**, **email**, **password**.

**Example TypeScript interface:**

export interface User {

id: number;

name: string;

email: string;

password: string; // plaintext for the exercise, but should be hashed in production!

}

**2. Setting up the Login Route**

* The route should be: **POST /login**
* The body should contain **email** and **password**.

**Implementation steps:**

1. **Retrieve credentials from the request body**  
   Use Express's **req.body** to access the email and password provided by the client.
2. **Find the user by email**  
   Search your in-memory user array for a user with the matching email.
3. **Compare the password**
   * Check if the user exists AND the password matches.
   * (If you want to go a step beyond, never tell the user “email not found” or “incorrect password” separately. Just say “Invalid credentials.”)
4. **Send a response**
   * If valid, respond with user info **excluding the password** (or just a success flag/token/message).
   * If invalid, respond with HTTP status **401 (Unauthorized)** and a message like **"Invalid credentials"**.

**3. TypeScript Safety**

* Use your **User** interface to enforce type safety on your stored user array and when searching/matching users.
* Type the request/response for clarity.

**4. Sample Route Logic (in Pseudocode/Plain English)**

router.post("/login", (req, res, next) => {

// 1. Extract email and password from req.body

// 2. Find user in users array by email

// 3. If user found and password matches

// - Respond with user details (omit password)

// Else

// - Respond with 401 Unauthorized, "Invalid credentials"

});

**5. Security Note**

* For real apps, never store passwords in plaintext and always use hashing (e.g., bcrypt).
* For this assignment’s in-memory approach, plaintext is acceptable but **never send the password back in API responses**.

**6. Integrating Error Handling**

* If you throw an error or use **next(err)**, your error handling middleware in Express will catch it and send a proper response.

**7. Testing**

* Try logging in with correct and incorrect credentials using Postman and check the responses.
* Make sure the response is clear and contains no password.

**Summary Table**

| **Step** | **What to do** |
| --- | --- |
| Setup route | POST /login |
| Get credentials | req.body.email & req.body.password |
| Find user | Search user array for matching email |
| Validate password | Compare stored and input password |
| Respond (success) | Status 200, return user info (NO password) |
| Respond (failure) | Status 401, return "Invalid credentials" |

**If you want guidance on the actual code structure, or have errors while, share your code snippet here and I'll help debug or clarify details!**

Prompt 5:

Can you help me with setting up the Express server and routing in the Module2\_Assignment?

Answer given by ai mentor:

Absolutely! Here’s a **step-by-step guide** to help you set up your Express server and routing for your Module2\_Assignment, keeping in mind best practices and your folder structure.

**1. Project Structure Reference**

According to your assignment:

Module2\_Assignment/

├── src/

│ ├── server.ts

│ ├── routes/

│ │ └── userRoutes.ts

│ ├── middleware/

│ │ └── logger.ts

│ │ └── errorHandler.ts

│ └── models/

│ └── user.ts

**2. Setting Up the Express Server (server.ts)**

**Main responsibilities:** Initialize the Express app.

* Apply middleware (logger, JSON parsing, error handler).
* Mount the user routes.

**Typical Steps in server.ts:**

import express from 'express';

import userRoutes from './routes/userRoutes';

import logger from './middleware/logger';

import errorHandler from './middleware/errorHandler';

const app = express();

const PORT = 3000;

// Built-in middleware to parse JSON request bodies

app.use(express.json());

// Custom middleware for logging

app.use(logger);

// Mount your user routes (adjust base path as needed, e.g., '/api')

app.use('/api', userRoutes);

// Error-handling middleware should be after all route/middleware registrations

app.use(errorHandler);

// Start server

app.listen(PORT, () => {

console.log(`Server running on portPORT}`);

});

* Make sure paths (e.g., **./routes/userRoutes**) are correct relative imports!

**3. Organizing User Routing (routes/userRoutes.ts)**

**Responsibilities:**

* Set up all user-related endpoints as mentioned in the assignment.
* Export as a router to be used in your server file.

**Typical Structure:**

import { Router } from 'express';

const router = Router();

// Register user

router.post('/register', (req, res, next) => {

// Registration logic

});

// Login user

router.post('/login', (req, res, next) => {

// Login logic

});

// Get user by ID

router.get('/user/:id', (req, res, next) => {

// Fetch user logic

});

export default router;

**4. Using Middleware**

* **logger.ts** should export a middleware function that logs each request (e.g., method and URL).
* **errorHandler.ts** should catch and respond to errors gracefully. Remember to use it after all other routes and middleware.

**5. Running Your Server**

* Use **ts-node** or configure your **package** scripts for easier development.
* Recommended: use **nodemon** for auto-restart on changes.

"scripts": {

"start": "nodemon -r ts-node/register src/server.ts"

}

Then run:

npm start

**6. Testing Your Endpoints**

* Once running, test the endpoints (e.g., **/api/register**, **/api/login**, **/api/user/:id**) using Postman.

**Recap**

| **File** | **Purpose** |
| --- | --- |
| **server.ts** | Sets up server, mounts routes, middleware |
| **routes/userRoutes.ts** | Handles **/register**, **/login**, **/user/:id** endpoints |
| **middleware/logger.ts** | Logs requests |
| **middleware/errorHandler.ts** | Handles errors |

**Need help with a specific server route error? Share the code and I’ll help you debug or clarify!**  
Keep exploring, you’re almost there!