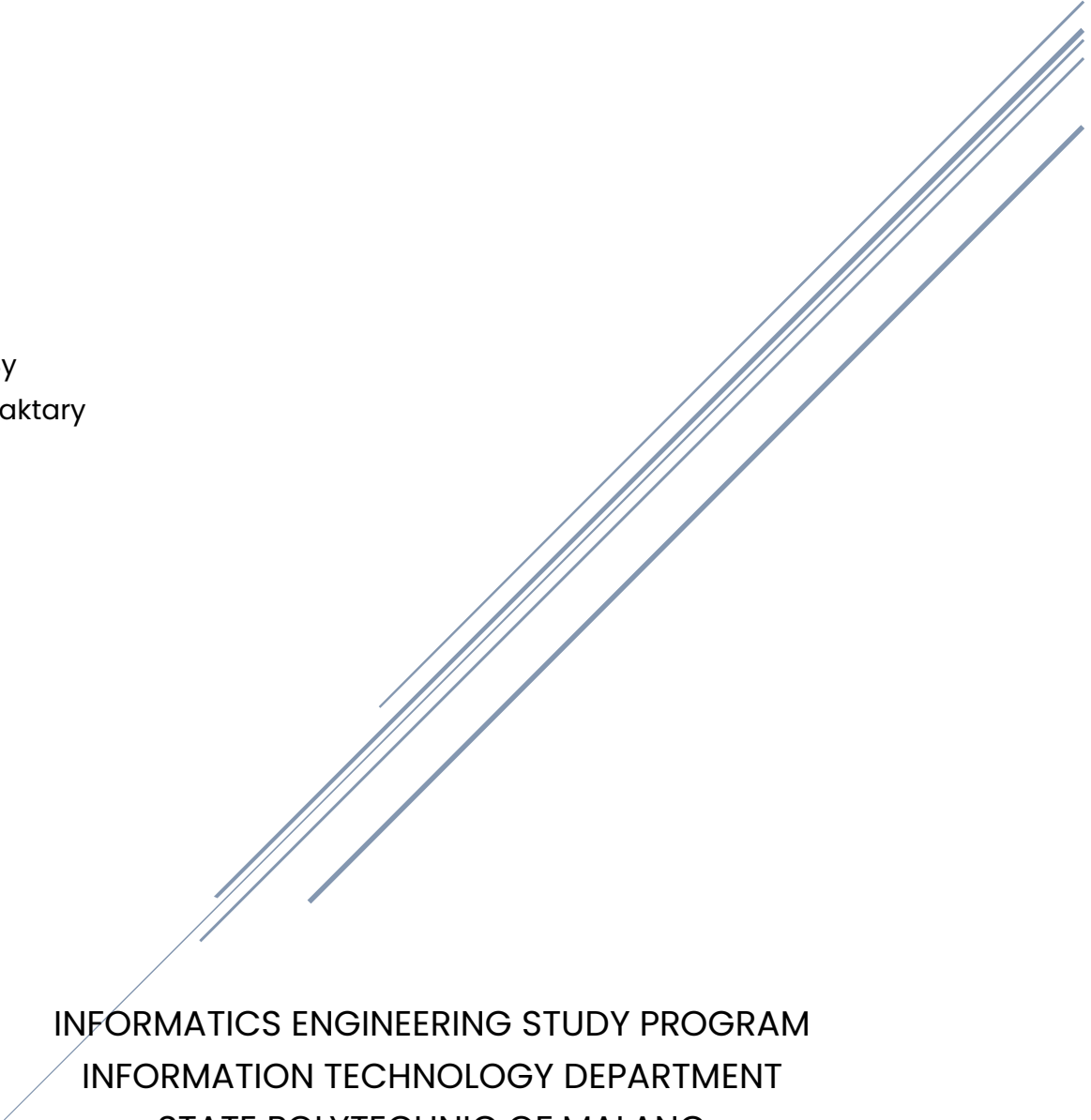


GUIDE B03

Create an Endpoint for Fetching The User Profile

Arranged By
Omar Al-Maktary



INFORMATICS ENGINEERING STUDY PROGRAM
INFORMATION TECHNOLOGY DEPARTMENT
STATE POLYTECHNIC OF MALANG

2023

Contents

Objectives.....	1
Requirements.....	1
Hardware Specifications.....	1
Minimum Requirements.....	1
Recommended Requirements	1
Software required	2
NPM Packages	2
Resource.....	2
Task Description.....	3
Start Coding.....	3
Running The API Application.....	5
Testing The API Application.....	5
Using Postman.....	5
Running The API Test File	6
Creating The Web Interface.....	7
Running and Testing The Web Interface.....	9
Results	10

Create an Endpoint for Fetching The User Profile

Objectives

1. Students can create a GET endpoint for users of the application to fetch their profile data.
2. Students can create a web page to view users' data.

Requirements

Having the correct hardware and software components is essential for ensuring the successful execution of the tasks outlined in this guide. The hardware configuration and software required for completing this guide tasks are as the following:

Hardware Specifications

The minimum hardware specifications for running a NodeJS API application on the Windows operating system and using software such as Postman and Visual Studio Code are the following:

Minimum Requirements

- Processor: Intel Core i3 or equivalent.
- RAM: 4 GB.
- Storage: 500 GB HDD with at least 20 GB of available storage.
- Graphics: Integrated graphics card.
- Connectivity: Ethernet and Wi-Fi capabilities.

Recommended Requirements

- Processor: Intel Core i5 or equivalent.
- RAM: 8 GB or more.
- Storage: 256 GB SSD with at least 20 GB of available storage.
- Graphics: Integrated graphics card.
- Connectivity: Ethernet and Wi-Fi capabilities.

Software required

It is important to have the correct software installed on your system to ensure that the application runs smoothly and meets performance expectations. The software required is as follows:

- Operating System: Windows 10 or later.
- NodeJS: Latest stable version installed.
- Visual Studio Code: Latest stable version installed.
- Postman: Latest stable version is installed.

NPM Packages

- nodemon: Automatically restarts Node application on file changes.
- cross-env: Sets environment variables in a cross-platform way.
- jest: Creates and executes tests.
- jest-expect-message: Enhances Jest assertions with custom error messages.
- jest-image-snapshot: Adds image snapshot testing to Jest.
- puppeteer: Node library to control a headless Chrome or Chromium browser.
- supertest: Makes HTTP queries to the application and checks results.
- dotenv: Simplifies management of environment variables.
- express: NodeJS framework for creating apps with routing and middleware.
- ejs: Embedded JavaScript templating.
- express-ejs-layouts: Layout support for EJS in Express.
- mongoose: MongoDB object modeling library for NodeJS.
- bcryptjs: NodeJS library for hashing passwords with the bcrypt algorithm.
- cookie-parser: Parses cookies attached to the incoming HTTP requests.
- cors: Middleware for enabling Cross-Origin Resource Sharing (CORS) in Express.
- express-unless: Defining exceptions to other middleware functions in Express.
- jsonwebtoken: Manage authentication by creating and verifying tokens.

Resource

- Documents: Guide B03
- Tests: api/testB03.test.js, web/testB03.test.js

Task Description

Students understand how to create an endpoint for fetching a user's profile data. This endpoint will return the user data by using an access token. Students should also understand how to create a web interface that shows the profile data and also buttons to act such as updating the profile data, changing the password, logging out, and deleting the user's account.

Start Coding

To create an endpoint for fetching user data, students should understand the following table which outlines the structure and goals of the endpoint. Note that there is an additional header for the authorization which is the token obtained from the login or registration process. This token has the user id encoded which can be used to fetch the user's data from the database.

GET <code>"/api/v1/profile"</code> ENDPOINT STRUCTURE			
API Endpoint Path	Request Method	Response Format	Description
<code>"/api/v1/profile"</code>	GET	JSON	Return the user's data.
Additional Headers			
Key	Value	Description	
<code>"Authorization"</code>	<code>"Bearer accessToken"</code>	AccessToken is the value of the token obtained from the login process	
Request Parameters (no parameters needed)			
Response Parameters			
Parameter	Type	Description	
<code>"user"</code>	Object	An object for the user data.	
<code>"message"</code>	String	A message indicating the status of the response.	
Success Responses			
HTTP Status Code	Response		
200	<pre>{ "user": { "name": "John Doe", "username": "johndoe", "email": "johndoe@gmail.com", "createdAt": "2023-02-20T07:32:14.786Z", "updatedAt": "2023-02-20T07:32:14.786Z", "id": "6424370fe2a9f3e77c1573ee" }, "message": "Profile Retrieved Successfully" }</pre>		

	}
Error Responses	
HTTP Status Code	Response
401	{message "Unauthorized"}
500	{ error object }

Table 1 GET "/api/v1/profile" ENDPOINT STRUCTURE

Follow the steps below to complete the code for this guide document:

1. In the "auth.service.js" file, copy and complete the following code.

```
async function getProfile(id) {
  // Find the user using the findById method
  const user = // write your code here...;
  // Return the user's data using the toJSON method
  // Also return a message that says "Profile Retrieved Successfully"
  return { // write your code here...};
}
```

Figure 1 Get Profile Data Function Code

2. Export the "getProfile" function at the end of the "auth.service.js" file.
3. In the "controllers/api/auth.controller.js" file, copy and complete the following code.

```
function getProfile(req, res, next) {
  // Get the user id from the request req.user.id
  const id = // Write your code here...;
  // Call the authServices.getProfile(id) function
  authServices
    .getProfile(id)
    .then((results) => // if the user is found, return the results object)
    .catch((err) => // if the user is not found, return the error object in the next
function);
}
```

Figure 2 Get Profile Controller Function Code

4. Export the "getProfile" function at the end of the "controllers/api/auth.controller.js" file.
5. In the "routes/api/auth.routes.js" file, import the getProfile function from the API controller.
6. Finally, Create a new GET route with the "/profile" path.

Running The API Application

For this guide and development purposes the command “npm run dev” is used to execute the command “nodemon server.js” which will run the “server.js” using the nodemon package. This package allows the server to reload if any changes occur in the code of the application.

Run the development command “npm run dev” in the terminal and notice the console message.

Testing The API Application

In this section, several tests in different ways will be explored to verify the results of the student's work on this document.

Using Postman

To test results from this guide on Postman, follow these steps:

1. In the “auth-experiment” collection, create a GET request with the name “GET /api/v1/profile”.
2. Make sure that the environment created is being used by selecting it from the top right option and then fill in the URL in the POST request as the following: “{{protocol}}{{host}}{{port}}{{version}}/profile”

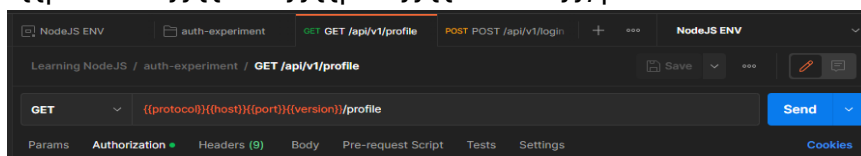


Figure 3 Postman Request Configuration Bar

3. In the Authorization tab, choose the “Bearer Token” as the type of authorization and fill the token field with “{{token}}” which should be created in the environment variables.
4. Request the login route created in the previous guide and save the token returned into the environment variable as the following:

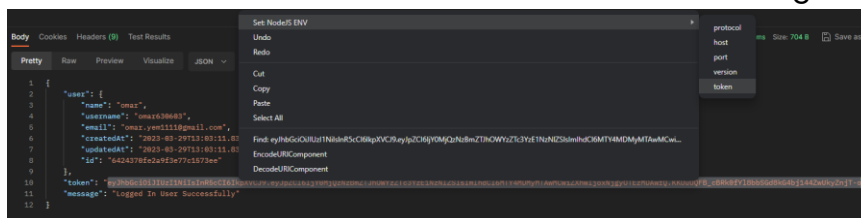


Figure 4 Postman Saving Token

5. Go back to the GET profile request and click “Send” then wait for the response. Postman should show results as the following if everything is working correctly:

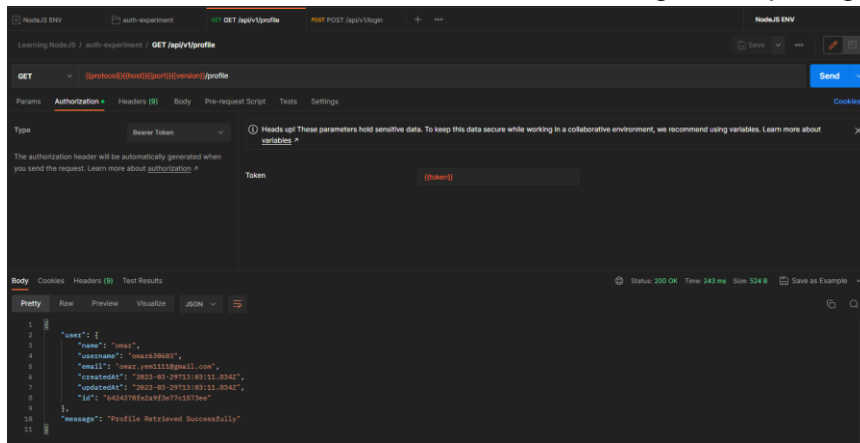


Figure 5 Postman Request Results

Running The API Test File

Note: Sometimes the test will have an error of time limit, try to re-run the test or increase the testTimeout in scripts of the “package.json” file.

Verify results by following these steps:

1. Copy the file "testB03.test.js" from the "api" folder within the "tests" folder for this material to the "tests/api" folder of your project base directory.
1. Run the fill in the VSCode integrated terminal by running this command “npm run api-testB03” and then wait for results.
2. If everything is correct and working well the results in the terminal should look as the following:

```
OMAR@LAPTOP-N1SUC5AB MINGW64 /d/Coding/Thesis/auth-experiment (main)
$ npm run api-testB03

> auth-experiment@1.0.0 api-testB03
> cross-env NODE_ENV=test jest -i tests/api/testB03.test.js --testTimeout=20000

console.log
  Database connected successfully

    at log (tests/api/testB03.test.js:31:15)

PASS tests/api/testB03.test.js
  Testing GET /api/v1/profile
    ✓ should login a user with username (149 ms)
    ✓ should return a user with the given token (86 ms)
    ✓ should return a 401 status code if the token is invalid (6 ms)

Test Suites: 1 passed, 1 total
Tests: 3 passed, 3 total
Snapshots: 0 total
Time: 3.047 s
Ran all test suites matching /tests\\api\\testB03.test.js/i.
```

Figure 6 Successful Test Results

3. If the test failed and it shows an error similar to the following figure, the error shows feedback for the cause of the error:

```
OWA@LAPTOP-N1SUCSAB MINGW64 /d/Coding/Thesis/auth-experiment (main)
$ npm run api-testB03

> auth-experiment@1.0.0 api-testB03
> cross-env NODE_ENV=test jest -i tests/api/testB03.test.js --testTimeout=20000

console.log
  Database connected successfully
    at log (tests/api/testB03.test.js:31:15)

FAIL tests/api/testB03.test.js
  Testing GET /api/v1/profile
    ✓ should login a user with username (147 ms)
    ✗ should return a user with the given token (86 ms)
    ✓ should return a 401 status code if the token is invalid (7 ms)
    • Testing GET /api/v1/profile > should return a user with the given token
      The returned user should not have a property called "password", but it does, change the response body in the function that handles the GET /api/v1/profile route
      expect(received).not.toHaveProperty(path)
      Expected path: not "password"
      Received value: "$2a$10$Vktb4P0Gf0hyaGwIacrr.NEBy3y2OE9Sc81spJN/09Ytd8YzHMF."
Test Suites: 1 failed, 1 total
Tests: 1 failed, 2 passed, 3 total
Snapshots: 0 total
Time: 2.954 s, estimated 3 s
Run all test suites matching /tests/api/testB03.test.js/i.
```

Figure 7 Failed Test Results

Try to find out why the test failed and fix it until the test result shows successful results.

Creating The Web Interface

In this section, the web interface for the profile page will be created. The same basic endpoint explained in the previous sections will be implemented in a web page that can show the user's data.

To start working on the web interface, follow these steps:

1. In the "controllers/web/auth.controller.js" file, copy and complete the following code:

```
function getProfile(req, res, next) {
  // Render the profile page
  // Title: Auth-Experiment | Profile
  // User: req.user.data
  // Message: req.body.message
  // Hint: Use the auth/profile.ejs file in views/auth
  // Write your code here
}
```

Figure 8 "controllers/web/auth.controller.js" Get Profile Function Code

2. Export the "getProfile" function at the end of the "controllers/web/auth.controller.js" file.
3. In the "routes/web/auth.routes.js" file, import the "getProfile" function from the web controller.

4. Add this new route as the following.

```
router.get("/profile", isLoggedIn, getProfile);
```

Figure 9 "routes/web/auth.routes.js" New Routes

5. In the "web/view/auth" folder create a new file named "profile.ejs".
6. In the "profile.ejs" file, copy the following code.

```
<% if (typeof message !== 'undefined') { %>
<div class="alert">
  <p class="message"><%= message %></p>
</div>
<% } %>
<div class="container">
  <h1 class="title">Account details</h1>
  <p class="description">Here you can see the details of your account</p>
</div>
<div class="container">
  <div class="card">
    <div class="card-body">
      <h5 class="card-title">Name: <%= user.name %></h5>
      <h6 class="card-subtitle">Username: <%= user.username %></h6>
      <p class="card-text">Email: <%= user.email %></p>
      <div class="action">
        <a href="/profile/update" class="btn btn-primary">Update Data</a>
        <a href="/profile/update/password" class="btn btn-primary">
          >Update Password</a>
      </div>
      <a href="/logout" class="btn btn-danger">Logout</a>
      <form action="/profile/delete" method="POST">
        <button type="submit" class="btn btn-danger">Delete</button>
      </form>
    </div>
  </div>
</div>
</div>
```

Figure 10 "profile.ejs" View Code

Note that this code has buttons for action that can be used to update the user profile, update the password, log out, and delete the user's account.

Running and Testing The Web Interface

If the application still running from the previous exercise then try to visit the following link <http://localhost:8080/>. If the application is not running in the terminal then use the command “npm run dev” to start the app.

If the user is not logged in then log in using the login page and it will automatically redirect to the profile page if the code is working correctly.

Upon visiting the URL, the web interface should look similar to the following:

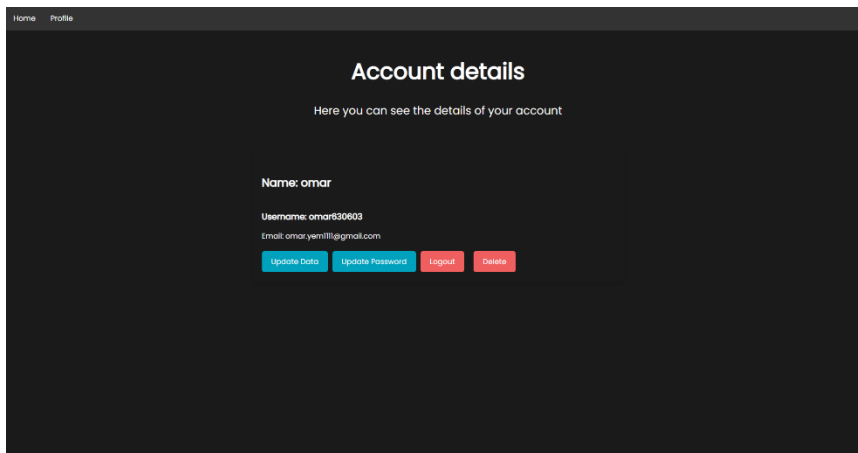


Figure 11 The Profile Page

Copy the file “testB03.test.js” and paste it to the folder “/tests/web” in your project directory. After that, run the command “npm run web-testB03” and notice the results.

If everything is correct and working well the results in the terminal should look as the following:

```
OMAR@LAPTOP-M1SUC5AB MINGW64 /d/Coding/Thesis/auth-experiment (main)
$ npm run web-testB03

> auth-experiment@1.0.0 web-testB03
> cross-env NODE_ENV=test jest -i tests/web/testB03.test.js --testTimeout=20000

PASS tests/web/testB03.test.js
  Testing the profile page
    ✓ should login a user (461 ms)
    ✓ should have the name of the user in the index page (162 ms)
    ✓ should have the name of the user in the profile page (164 ms)
    ✓ should have the right title in the profile page (169 ms)
    ✓ should have the right buttons and form in the profile page (181 ms)
  Testing the login page image snapshots
    ✓ matches the expected styling for the profile page (477 ms)

Test Suites: 1 passed, 1 total
Tests: 6 passed, 6 total
Snapshots: 1 passed, 1 total
Time: 4.865 s, estimated 9 s
Ran all test suites matching /tests\\web\\testB03.test.js/i.
```

Figure 12 Successful Web Test Results

```

QW@APTOP-N15K5SAR MINGW64 /d/Coding/Thesis/auth-experiment (main)
$ npm run web-test003

> auth-experiment@1.0.0 web-test003
> cross-env NODE_ENV=test jest -i tests/web/test003.test.js --testTimeout=20000

FAIL tests/web/test003.test.js (6.015 s)
  Testing the profile page
    ✓ should login a user (618 ms)
    ✓ should have the name of the user in the index page (181 ms)
    ✓ should have the name of the user in the profile page (171 ms)
    ✗ should have the right title in the profile page (164 ms)
    ✓ should have the right buttons and form in the profile page (190 ms)
  Testing the login page image snapshots
    ✓ matches the expected styling for the profile page (530 ms)

  ● Testing the profile page › should have the right title in the profile page

    The title "Profile" is wrong it should be "Auth-Experiment | Profile" Make sure that the function handling the GET "/profile" route is sending the right title
    expect(received).toBe(expected) // Object.is equality

    Expected: "Auth-Experiment | Profile"
    Received: "Profile"

Test Suites: 1 failed, 1 total
Tests:      1 failed, 5 passed, 6 total
Snapshots: 1 passed, 1 total
Time:       6.114 s
Run all test suites matching /tests/web/test003.test.js/i.

```

Figure 13 Failed Web Test Results

If you face a similar error try to figure out the reason for the problem until the test shows successful results.

Results

This document outlines the intended outcomes of the third meeting for the second material on the topic of web programming using NodeJS. Students be able to create an API endpoint for fetching users' data. Students should learn how to create a web interface to show the profile data.