



**INFORMATION SYSTEMS 3D
INSY7314
LAB GUIDE 2025
(First Edition: 2021)**

This manual enjoys copyright under the Berne Convention. In terms of the Copyright Act, no 98 of 1978, no part of this manual may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or any other information storage and retrieval system without permission in writing from the proprietor.



The Independent Institute of Education (Pty) Ltd is registered with the Department of Higher Education and Training as a private higher education institution under the Higher Education Act, 1997 (reg. no. 2007/HE07/002). Company registration number: 1987/004754/07.

Table of Contents

1	Introduction.....	3
2	Backend	4
2.1	<i>MongoDB</i>	4
2.2	<i>OpenSSL</i>	11
2.3	Backend API	16
2.4	<i>Integrating with MongoDB</i>	31
3	Frontend	49
3.1	<i>First React Application.</i>	49
4	Further Development	69

1 Introduction

Application Development Security (APDS) introduces the web developer to security concepts and how to incorporate them into development projects. The most efficient way to achieve this is by practical implementation, hence this guide. The concepts are generic and can be applied across any development stack; however, it is essential to understand that application security is not static and can be considered a “moving target”. New threats are constantly surfacing, and constant research is required to keep current. Most importantly, security should be built into the code from inception and not considered an afterthought.

In this document, you will be guided on how to build a basic MERN (Mongo Express React Node) application with an emphasis on security.

The guide assumes you are familiar with the following technologies.

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/A_re-introduction_to_JavaScript [Accessed 10 July 2024].
- https://developer.mozilla.org/docs/Learn/HTML/Introduction_to_HTML [Accessed 10 July 2024.]
- https://developer.mozilla.org/docs/Learn/CSS/First_steps [Accessed 10 July 2024].

Other recommended reading for the course.

- <https://developer.mozilla.org/en-US/docs/Web/Security> [Accessed 10 July 2024].
- <https://react.dev/learn> [Accessed 10 July 2024].

It is also assumed that you have the following development environment setup.

- Visual Studio Code <https://code.visualstudio.com> [Accessed 10 July 2024].
- Node <https://nodejs.org> (LTS version is fine) [Accessed 10 July 2024].

We will build the application in two parts – a backend and a frontend.

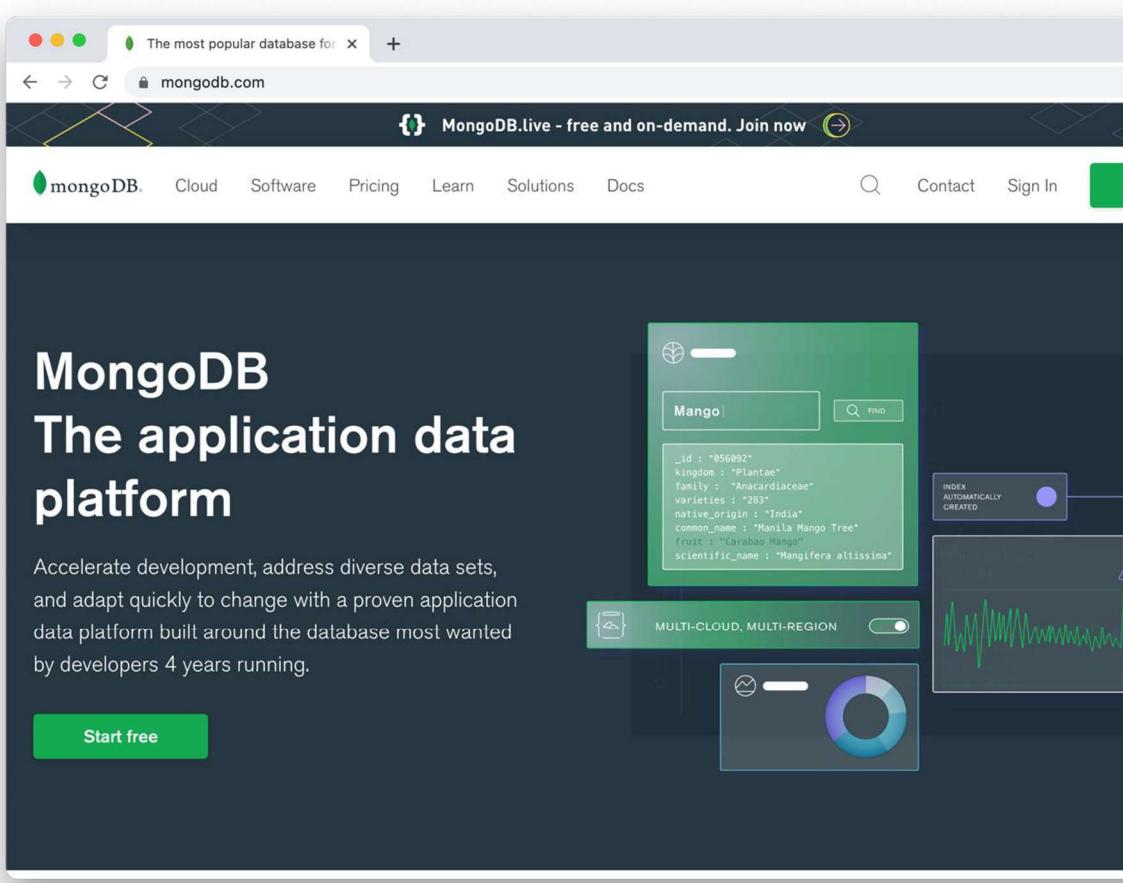
2 Backend

2.1 MongoDB

We start by setting up the cloud NOSQL database <https://www.mongodb.com> [Accessed 10 July 2024].

Remember, websites change, so the screenshots/steps may be different from these. Think in terms of outcomes.

- Click the green “Try Free” button at the top right.



- Click on “Cloud” and fill out your details on the right-hand side – use your student email address.

The screenshot shows the MongoDB Try interface. At the top, there are four main deployment options: Cloud (MongoDB as a service), On-premises (MongoDB locally), Tools (Boost productivity), and a partially visible option (N E R D). Below this, the "MongoDB Atlas" section is highlighted, featuring a brief description of its benefits and a "Features" list. To the right of the main content area, there is a sidebar with input fields for "Your Company (optional)", "How are you using MongoDB?", "Your Work Email", "First Name", and "Last Name".

Choose which type of deployment is best for you

Cloud
MongoDB as a service

On-premises
MongoDB locally

Tools
Boost productivity

MongoDB Atlas

Deploy MongoDB in the cloud in just a few clicks. Atlas is built on best-in-class automation and proven practices that help provide continuous availability, elastic scalability, fast performance, and support with regulatory compliance. It is the easiest way to try out MongoDB for free on AWS, Google Cloud, and Azure.

Features

- Fully managed database lifecycle backed by uptime SLA
- Auto-pilot capabilities such as auto-scale and auto-generated tuning guidance
- Elastic deployments with on-demand
- Integrated full-text search powered by

Your Company (optional)

How are you using MongoDB?
I'm learning MongoDB

Your Work Email
45094@iieconnect.co.za

First Name
Bruce

Last Name
Swart

Password

✓ 8 characters minimum

I agree to the [terms of service](#) and [privacy policy](#).

Get started free

Already have an account? [Sign in](#).

- Click on the “Get started free” button.

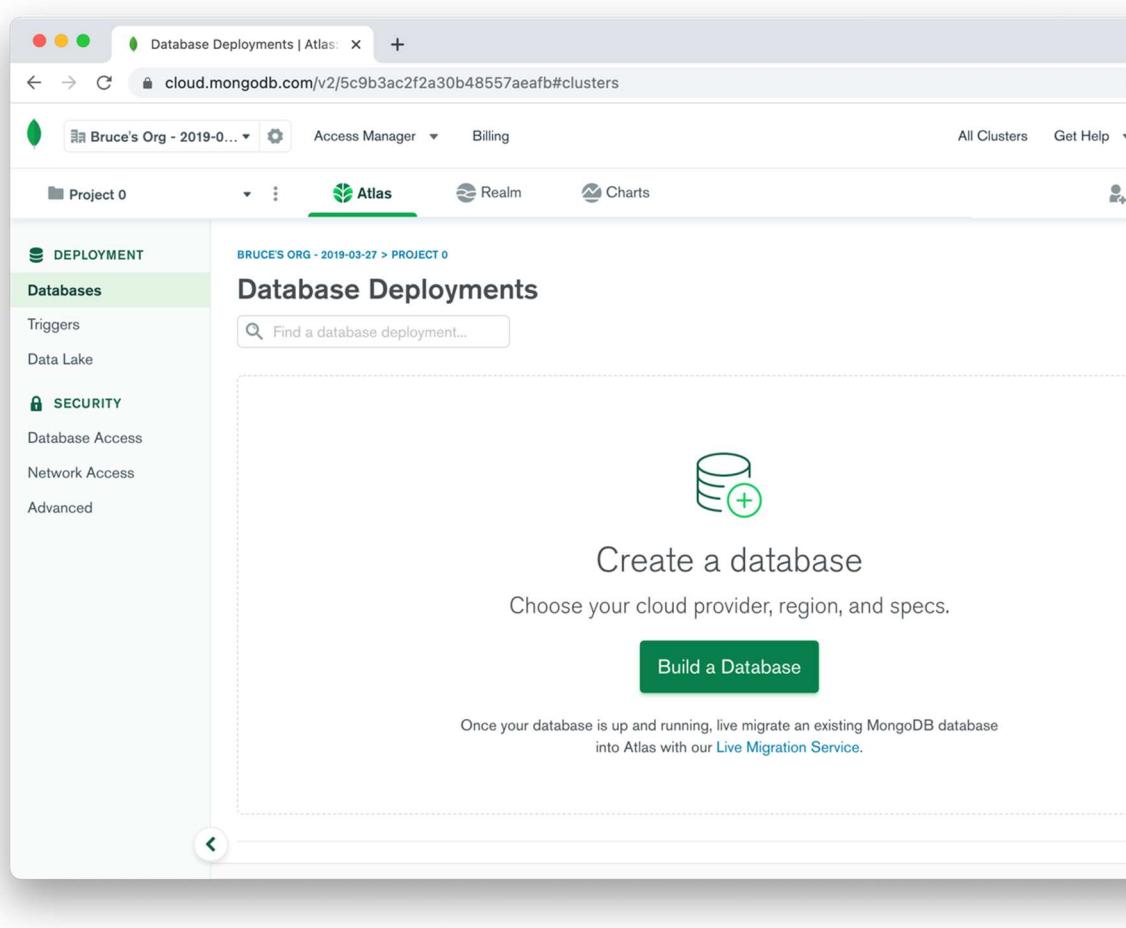
This is a detailed view of the "Get started free" form from the previous screenshot. The fields are pre-filled with sample data:

- Your Company (optional):
- How are you using MongoDB?: I'm learning MongoDB
- Your Work Email: 45094@iieconnect.co.za
- First Name: Bruce
- Last Name: Swart
- Password: *****
- ✓ 8 characters minimum
- I agree to the [terms of service](#) and [privacy policy](#).

Get started free

Already have an account? [Sign in](#).

- Click on “Build a database”.



- Select FREE/Shared and click the “Create” button.

The screenshot shows the MongoDB Atlas 'Choose a Path' interface. At the top, it says 'Choose a Path | Atlas: MongoDB' and the URL is 'cloud.mongodb.com/v2/5c9b3ac2f2a30b48557aeafb#clusters/pathSelector'. The MongoDB Atlas logo is at the top center. Below it, the title 'Deploy a cloud database' is displayed. A sub-instruction reads: 'Experience the best of MongoDB on AWS, Azure, and Google Cloud. Choose a deployment option to get started.' Three options are presented in cards:

- Serverless**: A green icon of a server. Description: 'For serverless applications that aren't critical with variable traffic. Minimal configuration required.' Benefits: Pay only for the operations you run, Resources scale seamlessly to meet your workload, Always-on security and backups. **Create** button.
- Dedicated**: A green icon of a bucket. Description: 'For production applications with sophisticated workload requirements. Advanced configuration controls.' Benefits: Network isolation and fine-grained access controls, On-demand performance advice, Multi-region and multi-cloud options available. **Create** button.
- Shared**: A green icon of a server. Description: 'For learning and exploring MongoDB in a cloud environment. Basic configuration options.' Benefits: No credit card required to start, Explore with sample datasets, Upgrade to dedicated clusters for full functionality. **Create** button.

- Click the “Create Cluster” button (all defaults are fine but make sure you are on Shared as illustrated).

The screenshot shows the MongoDB Atlas 'Create Deployment' interface for creating a new cluster. The title bar reads 'Create Deployment | Atlas: M0'. The URL in the address bar is 'cloud.mongodb.com/v2/5c9b3ac2f2a30b48557aeafb#clusters/edit?filter=starter'. The main heading is 'Create a Shared Cluster'. Below it, a welcome message says 'Welcome to MongoDB Atlas! We've recommended some of our most popular options, but feel free to customize your cluster to your needs. For more information, check our [documentation](#)'. There are three cluster type options: 'Serverless' (disabled), 'Dedicated', and 'Shared' (selected). A tooltip for 'Shared' indicates it's 'FREE'. A note below the cluster types states: 'For learning and exploring MongoDB in a sandbox environment. Basic configuration controls.' It also mentions 'No credit card required to start. Upgrade to dedicated clusters for full functionality. Explore with sample datasets. Limit of one free cluster per project.' Under 'Cloud Provider & Region', 'AWS, Ireland (eu-west-1)' is selected. The AWS logo is highlighted with a green border. Other providers like Google Cloud and Azure are also listed. A 'FREE' badge is present, along with a note: 'Free forever! Your M0 cluster is ideal for experimenting in a limited sandbox. You can upgrade to a production cluster anytime.' Buttons for 'Back' and 'Create Cluster' are at the bottom.

- Now we need to get our connection string. Click on the “Connect” button

The screenshot shows the MongoDB Atlas interface for a project named "Project 0". The left sidebar includes sections for Deployment (selected), Databases, Triggers, Data Lake, Security (Database Access, Network Access, Advanced), and Billing. The main content area is titled "Database Deployments" and displays a green banner stating "Sample dataset successfully loaded. Access it in Data Explorer by clicking the Collections button, or with the Mongo Shell.". Below this, there are four monitoring charts: R 0 (Read operations), W 0 (Write operations), Connections 0, and In 0.0 B/s (Network traffic). At the bottom, detailed cluster information is provided: VERSION 4.4.8, REGION AWS / N. Virginia (us-east-1), CLUSTER TIER M0 Sandbox (General), TYPE Replica Set - 3 nodes, BACKUPS Inactive, and LINKED REALM None Linked.

- Choose the “Connect your application” option

Connect to Cluster0

[✓ Setup connection security](#) [Choose a connection method](#) [Connect](#)

Choose a connection method [View documentation](#)

Get your pre-formatted connection string by selecting your tool below.



Connect with the MongoDB Shell

Interact with your cluster using MongoDB's interactive Javascript interface



Connect your application

Connect your application to your cluster using MongoDB's native drivers



Connect using MongoDB Compass

Explore, modify, and visualize your data with MongoDB's GUI

[Go Back](#)

[Close](#)

- Copy the connection string somewhere safe.

2 Add your connection string into your application code

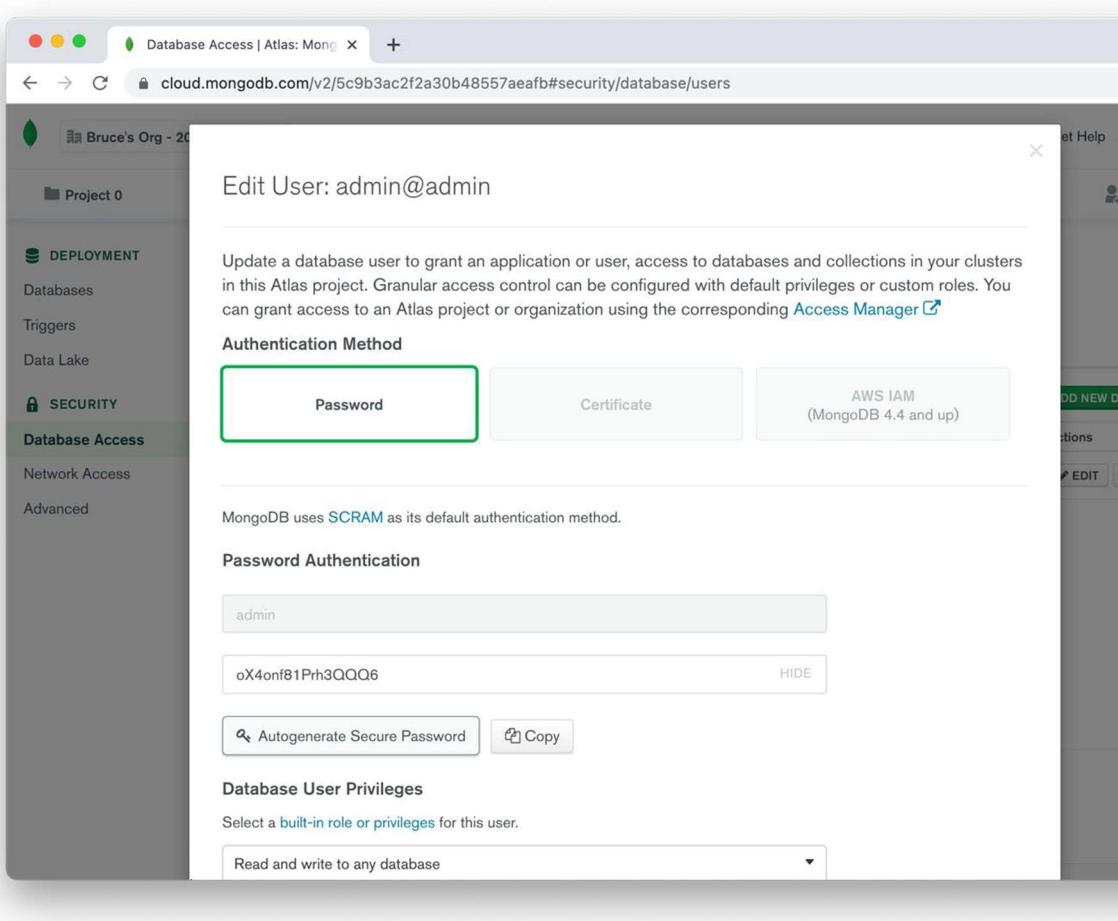
Include full driver code example

```
mongodb+srv://admin:<password>@cluster0.u597s.mongodb.net/myFirstDatabase?  
retryWrites=true&w=majority
```

Replace **<password>** with the password for the **admin** user. Replace **myFirstDatabase** with the name database that connections will use by default. Ensure any option params are **URL encoded**.

Lastly – generate a password for your database admin user.

- Using the menu on the left-hand side, choose “Database Access” under “Security” – click the “Autogenerate Secure Password” button and keep the generated password somewhere safe.
- Click on the “Update User” button at the bottom once done.



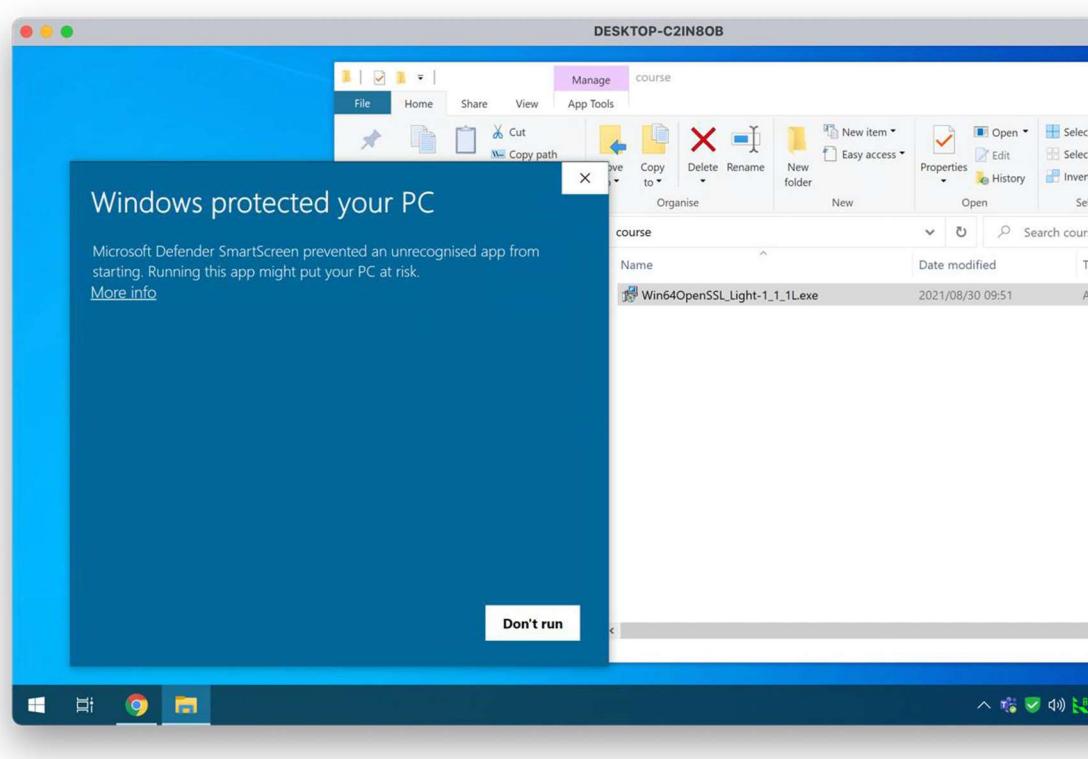
2.2 OpenSSL

Next, we generate security certificates by downloading software from <https://slproweb.com/products/Win32OpenSSL.html> [Accessed 10 July 2024].

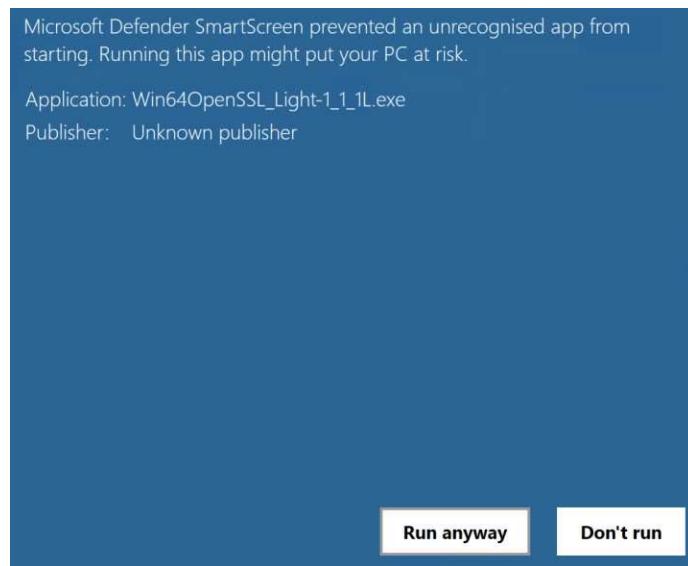
- Choose the Win64 Light version (the first one in this screenshot). Remember – screen layout and/or version number may be different.

File	Type	Description
Win64 OpenSSL v1.1.1L Light EXE MSI	3MB Installer	Installs the most commonly used essentials of Win64 OpenSSL v1.1.1L (Recommended for use OpenSSL). Only installs on 64-bit versions of Windows. Note that this is a default build of OpenSSL and is subject to local and state laws. More information can be found in the legal agreement of the installation.
Win64 OpenSSL v1.1.1L EXE MSI	63MB Installer	Installs Win64 OpenSSL v1.1.1L (Recommended for software developers by the creators of OpenSSL on 64-bit versions of Windows. Note that this is a default build of OpenSSL and is subject to local and state laws. More information can be found in the legal agreement of the installation).
Win32 OpenSSL v1.1.1L Light EXE MSI	3MB Installer	Installs the most commonly used essentials of Win32 OpenSSL v1.1.1L (Only install this if you need 32-bit OpenSSL for Windows. Note that this is a default build of OpenSSL and is subject to local and state laws. More information can be found in the legal agreement of the installation).
Win32 OpenSSL v1.1.1L EXE MSI	54MB Installer	Installs Win32 OpenSSL v1.1.1L (Only install this if you need 32-bit OpenSSL for Windows. Note that this is a default build of OpenSSL and is subject to local and state laws. More information can be found in the legal agreement of the installation).

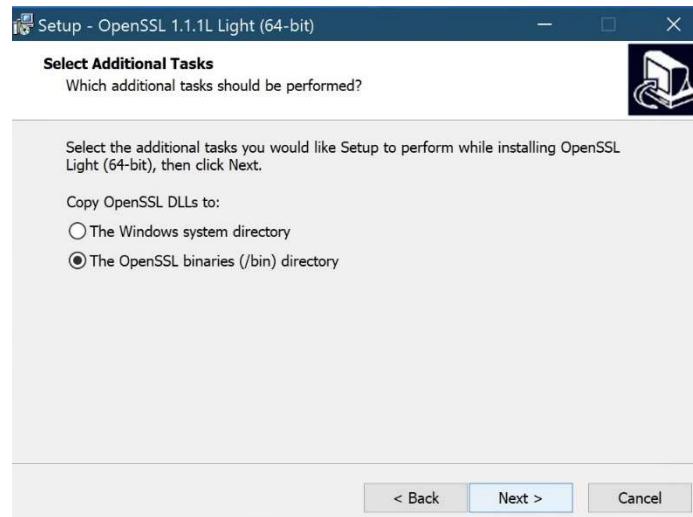
- When installing, Windows 10 or Windows 11 may give you some issues. Click on the “More info” link.



- Click the “Run anyway” button – it is safe!



- Accept most defaults when installing, except for install location – make this somewhere that you can normally access – especially on the campus PC's. Also, when asked about the DLL location choose the option as below.

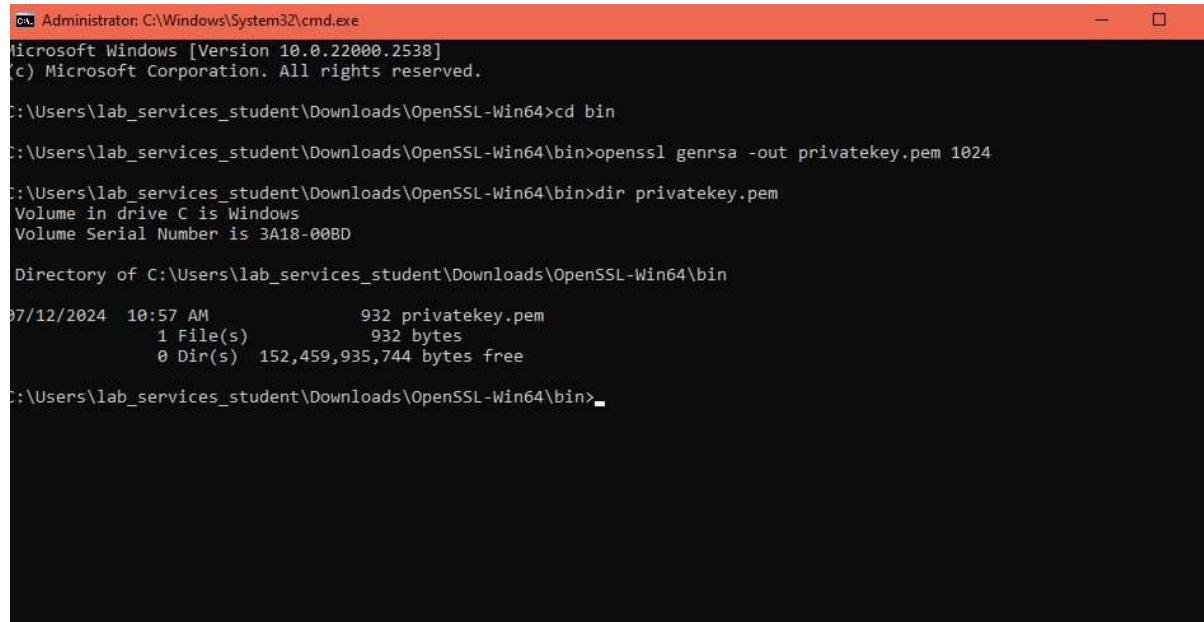


- Startup Visual Studio Code and open a terminal window. Terminal->New Terminal

This is effectively the Windows command line prompt. If this and its commands are new to you – do some research. Windows currently has two command shells: traditional command shell, and PowerShell.

- Navigate to wherever you installed OpenSSL and change to the “bin” folder as shown.

- First generate the private key
openssl genrsa -out privateKey.pem 1024



Administrator: C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.22000.2538]
(c) Microsoft Corporation. All rights reserved.
C:\Users\lab_services_student\Downloads\OpenSSL-Win64>cd bin
C:\Users\lab_services_student\Downloads\OpenSSL-Win64\bin>openssl genrsa -out privateKey.pem
C:\Users\lab_services_student\Downloads\OpenSSL-Win64\bin>dir privateKey.pem
Volume in drive C is Windows
Volume Serial Number is 3A18-00BD
Directory of C:\Users\lab_services_student\Downloads\OpenSSL-Win64\bin
07/12/2024 10:57 AM 932 privateKey.pem
1 File(s) 932 bytes
0 Dir(s) 152,459,935,744 bytes free
C:\Users\lab_services_student\Downloads\OpenSSL-Win64\bin>

- Before the next step you may have to create an “openssl.cfg” file in the “bin” folder. Do this using your favorite text editor such as Notepad (or Visual Studio Code of course!). The contents of the file come from here <https://www.openssl.org/docs/man1.1.1/man1/req.html> [Accessed 10 July 2024].

- Scroll down until you find the text as below and paste everything in the black block into your “openssl.cfg” file and save.

Sample configuration file prompting for field values:

```
[ req ]  
default_bits      = 2048  
default_keyfile   = privkey.pem  
distinguished_name = req_distinguished_name  
attributes        = req_attributes  
req_extensions    = v3_ca  
  
distrstring_type = nobmp  
  
[ req_distinguished_name ]  
countryName          = Country Name (2 letter code)  
countryName_default  = AU  
countryName_min      = 2  
countryName_max      = 2  
  
localityName         = Locality Name (eg, city)  
  
organizationalUnitName = Organizational Unit Name (eg, section)  
  
commonName           = Common Name (eg, YOUR name)  
commonName_max       = 64  
  
emailAddress         = Email Address  
emailAddress_max     = 40  
  
[ req_attributes ]  
challengePassword    = A challenge password  
challengePassword_min = 4  
challengePassword_max = 20  
  
[ v3_ca ]  
  
subjectKeyIdentifier=hash  
authorityKeyIdentifier=keyid:always,issuer:always  
basicConstraints = critical, CA:true
```

- Now you can run the next command – this creates the certificate signing request.
openssl req -new -key privatekey.pem -out certrequest.csr
-

```
Administrator: C:\Windows\System32\cmd.exe

C:\Users\lab_services_student\Downloads\OpenSSL-Win64\bin>openssl req -new -key privatekey.pem -out certrequest.csr
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:ZA
State or Province Name (full name) [Some-State]:KZN
Locality Name (eg, city) []:DBN
Organization Name (eg, company) [Internet Widgits Pty Ltd]:APDS
Organizational Unit Name (eg, section) []:STUDENT
Common Name (e.g. server FQDN or YOUR name) []:Yusuf
Email Address []:yparuk@varsitycollege.co.za

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:friedgreenmatatoes
An optional company name []

C:\Users\lab_services_student\Downloads\OpenSSL-Win64\bin>dir certrequest.csr
Volume in drive C is Windows
Volume Serial Number is 3A18-00BD

Directory of C:\Users\lab_services_student\Downloads\OpenSSL-Win64\bin

07/12/2024  11:04 AM           754 certrequest.csr
               1 File(s)      754 bytes
               0 Dir(s)  152,503,558,144 bytes free

C:\Users\lab_services_student\Downloads\OpenSSL-Win64\bin>
```

- Now the final command – signing the certificate using your private key
openssl x509 -req -in certrequest.csr -signkey privatekey.pem -out certificate.pem

```
C:\Users\lab_services_student\Downloads\OpenSSL-Win64\bin>dir *.pem
Volume in drive C is Windows
Volume Serial Number is 3A18-00BD

Directory of C:\Users\lab_services_student\Downloads\OpenSSL-Win64\bin

07/12/2024  11:06 AM           1,022 certificate.pem
07/12/2024  10:57 AM           932 privatekey.pem
               2 File(s)      1,954 bytes
               0 Dir(s)  152,501,813,248 bytes free

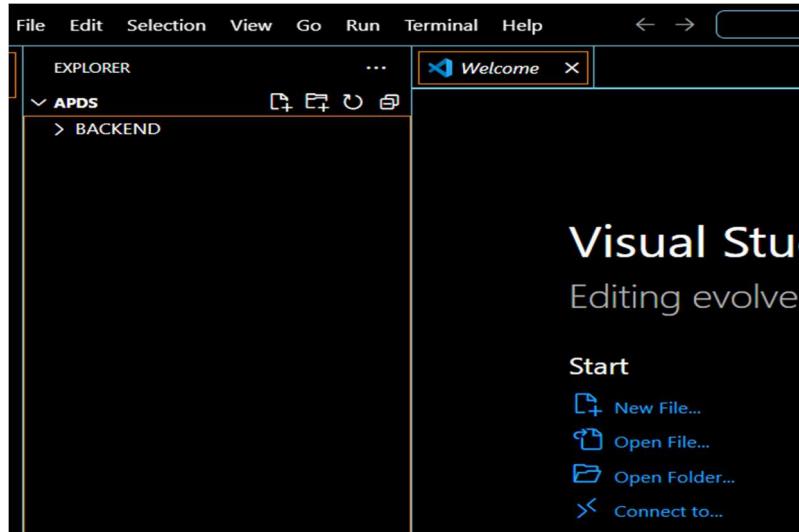
C:\Users\lab_services_student\Downloads\OpenSSL-Win64\bin>
```

- Keep these two files certificate.pem and privatekey.pem somewhere safe.

2.3 Backend API

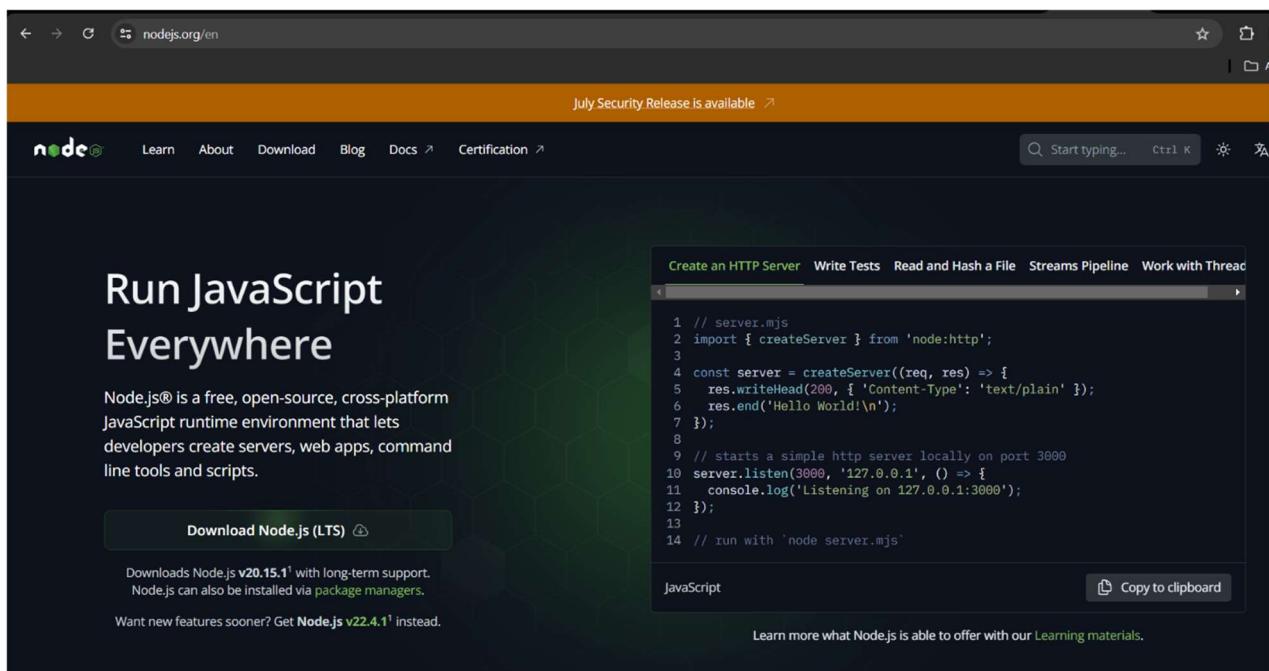
Next, we will start building out the standalone backend API that interacts with the MongoDB database in the cloud and provides data to the frontend.

- Create a folder in Visual Studio Code called “BACKEND” –where all your project files will be stored.



- -
 -
 - Install node.js which we use to write server-side applications.

Run JavaScript Everywhere - Node.js (2024)



In a new terminal window, make sure you are in the backend folder that you created earlier.

- Type this command (npm stands for node package manager <https://docs.npmjs.com>) `npm init -y`

This will create a `package.json` file. It lists all the dependent packages your projects require which are stored in a folder called `node_modules`.

The screenshot shows the Microsoft Visual Studio Code interface. On the left is the Explorer sidebar with a tree view showing a folder named 'BACKEND' containing a single file 'package.json'. The main area is the 'Welcome' screen, which includes sections for 'Start' (with options like 'New File...', 'Open File...', 'Open Folder...', 'Clone Git Repository...', and 'Connect to...') and 'Recent' (listing 'Desktop (Workspace)', 'APDS', 'APDS7312Material-master', and 'test'). Below the Welcome screen are tabs for 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL' (which is selected), and 'PORTS'. A terminal window at the bottom displays the command `npm init -y` being run in the directory `C:\Users\lab_services_student\Desktop\APDS\BACKEND`. The output shows the creation of a `package.json` file with the following content:

```
{  
  "name": "backend",  
  "version": "1.0.0",  
  "description": "",  
  "main": "index.js",  
  "scripts": {  
    "test": "echo \\\"Error: no test specified\\\" && exit 1"  
  },  
  "keywords": [],  
  "author": "",  
  "license": "ISC"  
}
```

- Add a package to your project using the “npm” install command
(i=install, D=dev only dependencies)
npm i -D nodemon

<https://www.npmjs.com/package/nodemon>

“nodemon is a tool that helps develop node.js based applications by automatically restarting the node application when file changes in the directory are detected.”

You can see below this now appears at the bottom of the package.json file along with its version.

- In addition, change the package.json “scripts” section as below. This will allow us to run our backend in either development or production environments.

```
{ } package.json > { } dependencies > nodemon
1  {
2    "name": "backend",
3    "version": "1.0.0",
4    "description": "",
5    "main": "index.js",
6    ▷ Debug
7    "scripts": {
8      "start": "node server.mjs",
9      "dev": "nodemon server.mjs"
10     },
11    "keywords": [],
12    "author": "",
13    "license": "ISC",
14    "dependencies": {
15      "nodemon": "^3.1.4"
16    }
17 }
```

- Create a new file in your backend folder “server.js” and insert a console.log command as below. Remember this command – you will use it extensively throughout this exercise to debug.

Make sure to “Save All” files from the menu. “Save” only saves the current tab you are on. This is a common issue that students encounter.

The screenshot shows the Visual Studio Code interface. On the left is the Explorer sidebar with a 'BACKEND' folder containing 'node_modules', 'package-lock.json', 'package.json', and 'server.mjs'. The 'server.mjs' file is selected. In the center is the code editor with the following content:

```
JS server.mjs
1 console.log('APDS makes me cry!')
```

Below the code editor is the Terminal panel, which displays the output of an npm install command:

```
PS C:\Users\lab_services_student\Desktop\APDS\BACKEND> npm i express
added 64 packages, and audited 65 packages in 4s
12 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities
PS C:\Users\lab_services_student\Desktop\APDS\BACKEND>
```

- Run the code as follows

```
npm run start
```

This runs “node server” as per your scripts tag in “package.json”
You can see in the terminal that it displays our log.

The screenshot shows the Visual Studio Code interface. On the left is the Explorer sidebar with a 'BACKEND' folder containing 'node_modules', 'package-lock.json', 'package.json', and 'server.mjs'. The 'server.mjs' file is selected. The main area shows the code:

```
JS server.mjs X {} package.json
JS server.mjs
1 console.log('APDS makes me cry!')
```

At the bottom, the Terminal tab is active, showing the command line output:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Node.js v21.6.1
PS C:\Users\lab_services_student\Desktop\APDS\BACKEND> npm run start
> backend@1.0.0 start
> node server.mjs
APDS makes me cry!
PS C:\Users\lab_services_student\Desktop\APDS\BACKEND>
```

Our code is not however running as a HTTP server which listens on a port and is accessed with a web browser.

- Change server.js as follows.

[MERN Stack Explained | MongoDB](#) (Accessed 12 July 2024).

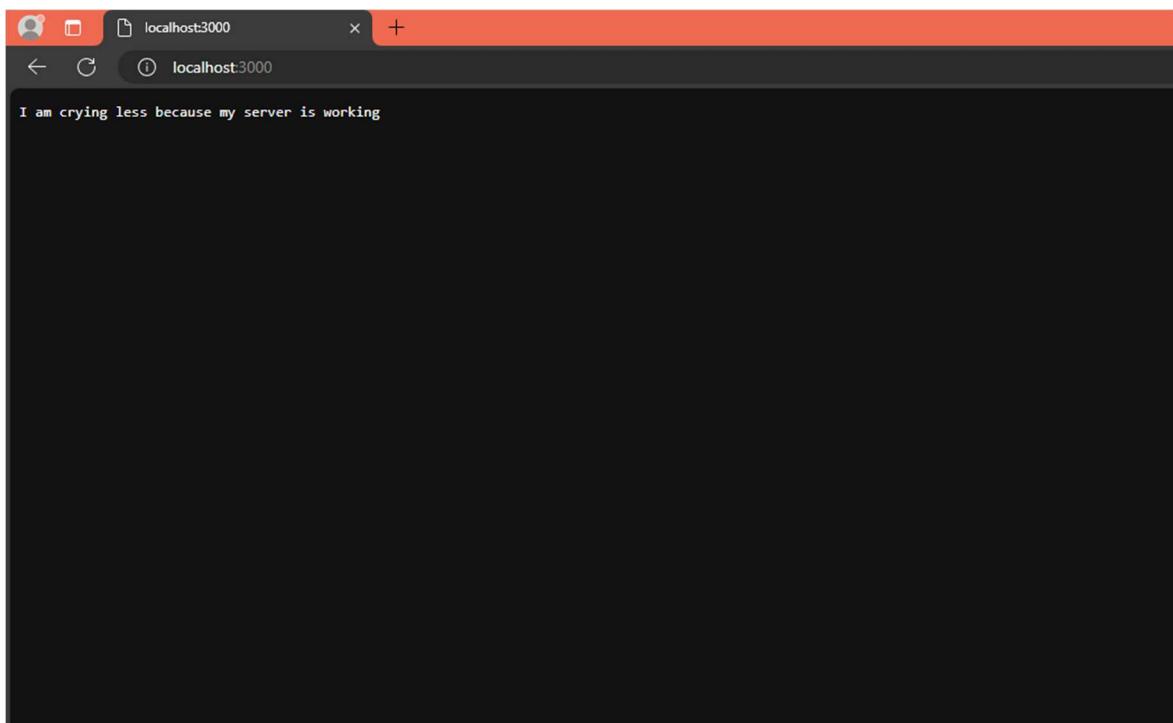
The screenshot shows the Visual Studio Code interface. On the left is the Explorer sidebar with a 'BACKEND' folder containing 'node_modules', 'package-lock.json', 'package.json', and 'server.mjs'. The 'server.mjs' file is selected and open in the main editor area. The code is as follows:

```
1 import http from "http";
2
3 //set the port
4 const PORT = 3000;
5
6 // start the server
7 const server = http.createServer((req,res)=>{
8     res.end('I am crying less because my server is working')
9 })
10
11 server.listen(PORT)
```

Below the editor is a terminal window showing the command line output:

```
PS C:\Users\lab_services_student\Desktop\APDS\BACKEND> npm run start
> backend@1.0.0 start
> node server.mjs
```

This time we must open a web browser to see the output.



From now on, instead of running “npm run start” we run (as we would in production)
npm run dev

This will restart the server every time we make a change to our code and save – thanks to
nodemon
(and the fact that we changed our scripts tag in package.json)

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

run `npm fund` for details

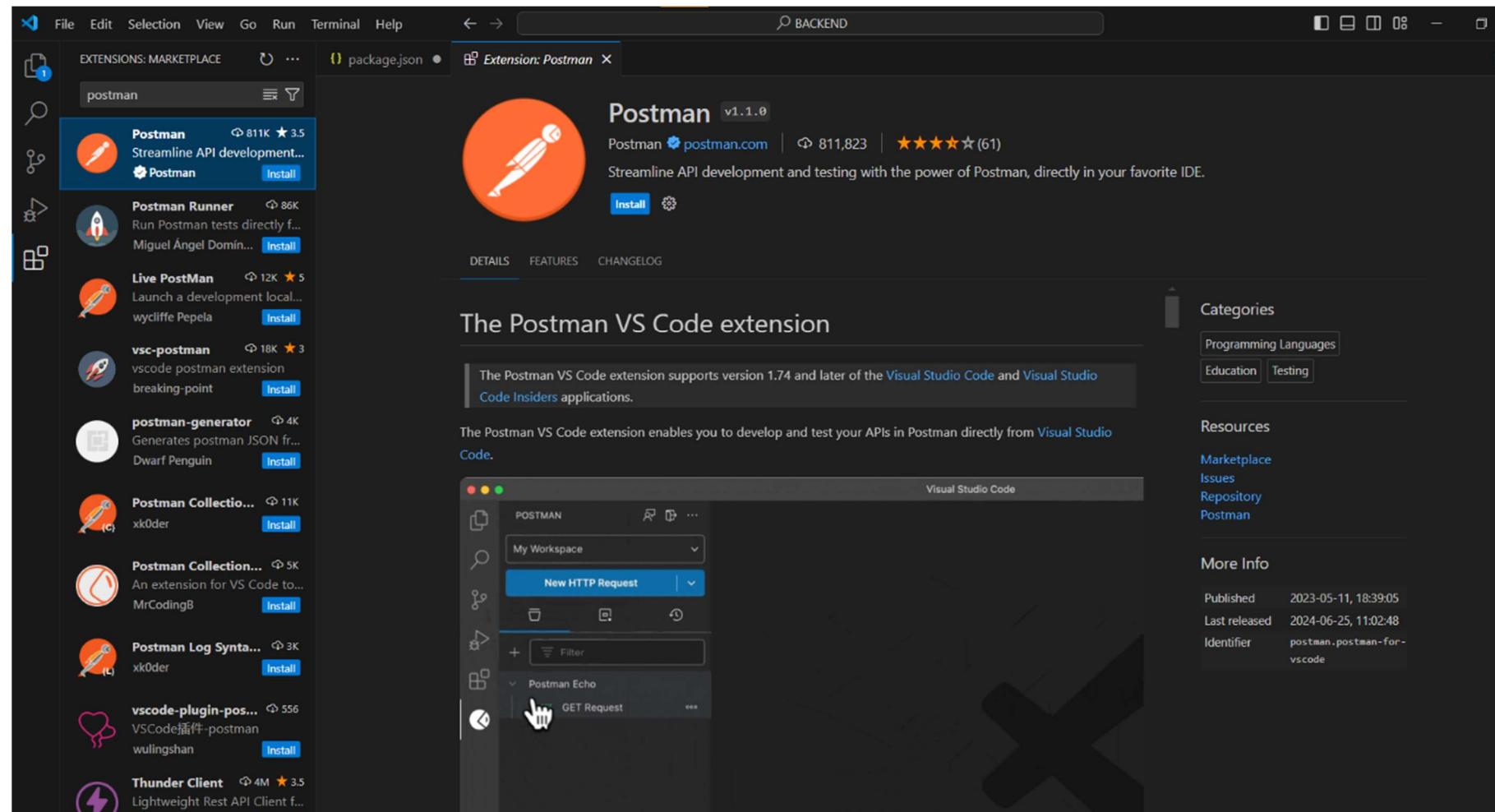
found 0 vulnerabilities
PS C:\Users\lab_services_student\Desktop\APDS\BACKEND> npm run dev

> backend@1.0.0 dev
> nodemon server.mjs

[nodemon] 3.1.4
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): ***!
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node server.mjs`
```

Before we continue, we must install a Visual Studio Code extension called “Postman”(you can also download the desktop app). This will allow us to call code in our backend API server without a frontend (or web browser)!

[Get started in Postman | Postman Learning Center](#) Accessed 10 July 2024.



Once the Postman extension is installed, we need to login, once logged in you can click new request and perform a get request to your server's port.

POSTMAN

File Edit Selection View Go Run Terminal Help

BACKEND [Administrator]

JS server.mjs http://localhost:3000

Save No Environment

New HTTP Request

GET http://localhost:3000

Send

Params Authorization Headers (7) Body Pre-request Script Tests Settings

Code Cookies

Query Params

Key	Value
Key	Value

Body Cookies Headers (4) Test Results

Status: 200 OK Time: 21 ms Size: 168 B

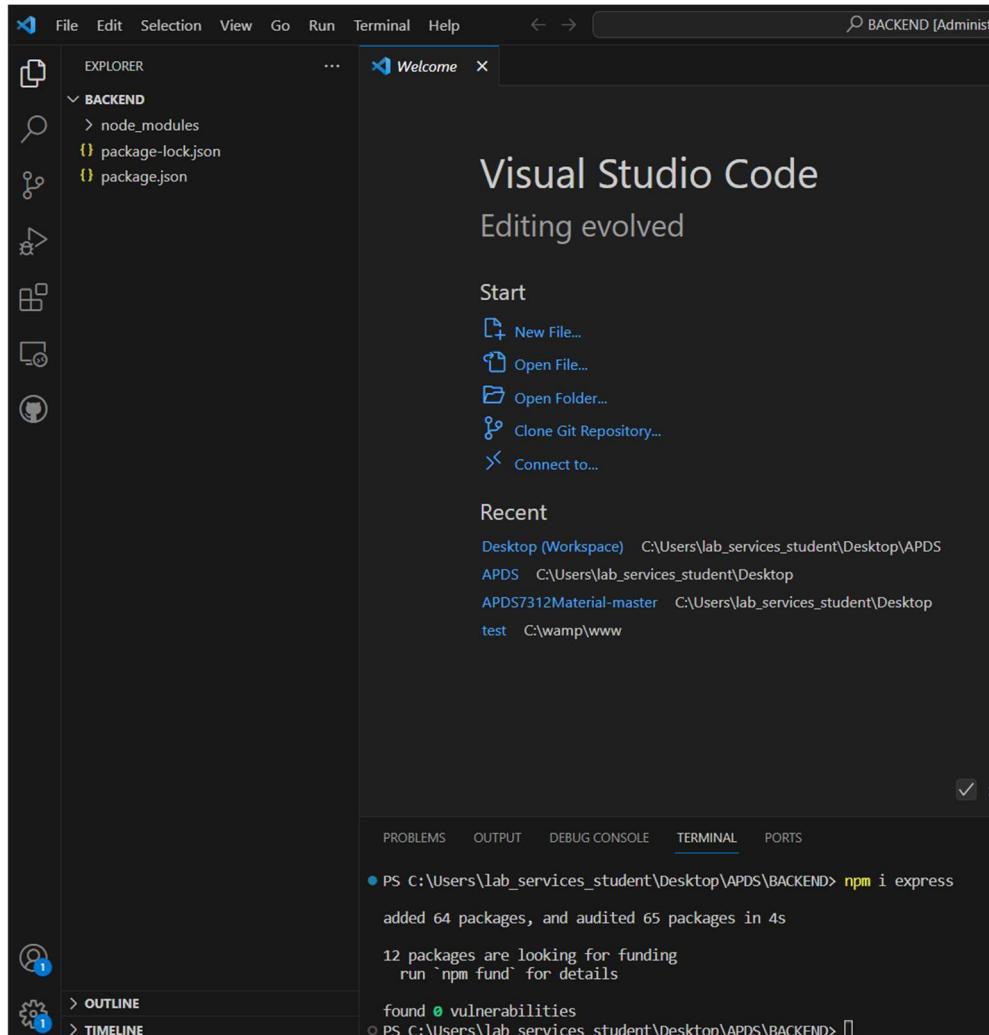
Pretty Raw Preview Text

1 I am crying less because my server is working

Create Collection

Now hand coding our “server.mjs” in this way would be very tedious. Fortunately for us “there’s a package for that” called Express (yes, the “E” in MERN, along with N for Node).

- We can install it by running the following command in the terminal
npm i express



- Change server.mjs to look as follows in the screenshot below.

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows a project structure with a folder named "BACKEND" containing "node_modules", "package-lock.json", "package.json", and a file named "server.mjs".
- Code Editor:** Displays the content of the "server.mjs" file:

```
1 import express from "express";
2 const PORT = 3000;
3 const app = express();
4
5 app.use(express.json());
6
7 app.get('/',(req, res)=>{
8 | res.send('I am finally figuring this out, no more crying')
9 })
10
11 // start the Express server
12 app.listen(PORT);
```
- Terminal:** Shows the output of running the application with nodemon:

```
> backend@1.0.0 dev
> nodemon server.mjs

[nodemon] 3.1.4
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): ***!
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node server.mjs`
[nodemon] restarting due to changes...
[nodemon] starting `node server.mjs`
```

- Send Request" to test your server in Postman again. <https://expressjs.com/en/starter/hello-world.html> [Accessed 13 July 2024]

Update server.mjs as follows to create a testing endpoint.

Both these endpoints will now work.

<http://localhost:3000/test> [Accessed 13 July 2024].

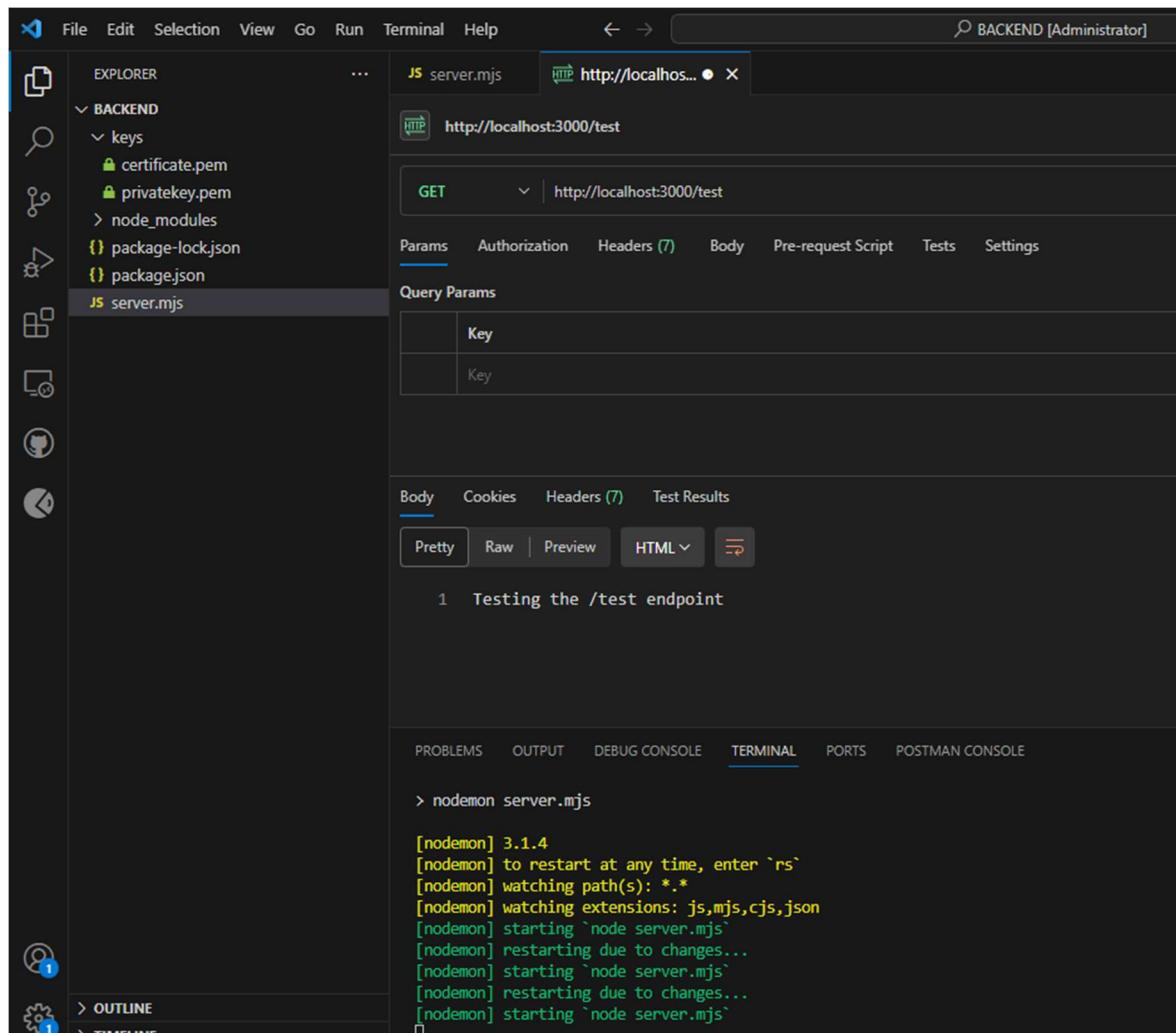
The screenshot shows the Visual Studio Code interface. The left sidebar displays a file tree with a folder named 'BACKEND' containing files like 'keys', 'certificate.pem', 'privatekey.pem', 'node_modules', 'package-lock.json', 'package.json', and 'server.mjs'. The main editor area shows the 'server.mjs' file content:

```
1 import express from "express";
2 const PORT = 3000;
3 const app = express();
4
5 app.use(express.json());
6
7 app.get('/',(req, res)=>{
8 | res.send('I am finally figuring this out, no more crying')
9 })
10
11 app.get('/test',(req, res)=>{
12 | res.send('Testing the /test endpoint')
13 })
14
15 // start the Express server
16 app.listen(PORT);
```

The bottom right panel shows the terminal output:

```
> nodemon server.mjs
[nodemon] 3.1.4
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): ***!
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node server.mjs`
[nodemon] restarting due to changes...
[nodemon] starting `node server.mjs`
[nodemon] restarting due to changes...
[nodemon] starting `node server.mjs`
```

Do not forget to test with Postman



Now let's return a more complex string like JSON and modify our endpoint

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows a project structure for "BACKEND" containing "keys" (with files "certificate.pem" and "privatekey.pem"), "node_modules", and "server.mjs".
- Terminal:** Shows the command "nodemon" starting the application.
- Code Editor:** Displays the file "server.mjs" which contains Node.js code for an Express application. The code defines two endpoints: one returning a simple message and another returning a JSON object with a list of fruits.

```
import express from "express";
const PORT = 3000;
const app = express();
const urlprefix = '/api';

app.use(express.json());

app.get(urlprefix + '/', (req, res) => {
  res.send('I am finally figuring this out, no more crying')
})

app.get(urlprefix + '/orders', (req, res) => {
  const orders = [
    {
      id: "1",
      name: "Orange"
    },
    {
      id: "2",
      name: "Apple"
    },
    {
      id: "3",
      name: "Pear"
    }
  ]
  res.json(
    {
      message: "Fruits",
      orders: orders
    }
  )
})

app.listen(PORT);
```

- Status Bar:** Shows "BACKEND [Administrator]" and the status bar with tabs: PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (which is underlined), PORTS, and POSTMAN CONSOLE.

Now Let's Test to see if our updates worked.

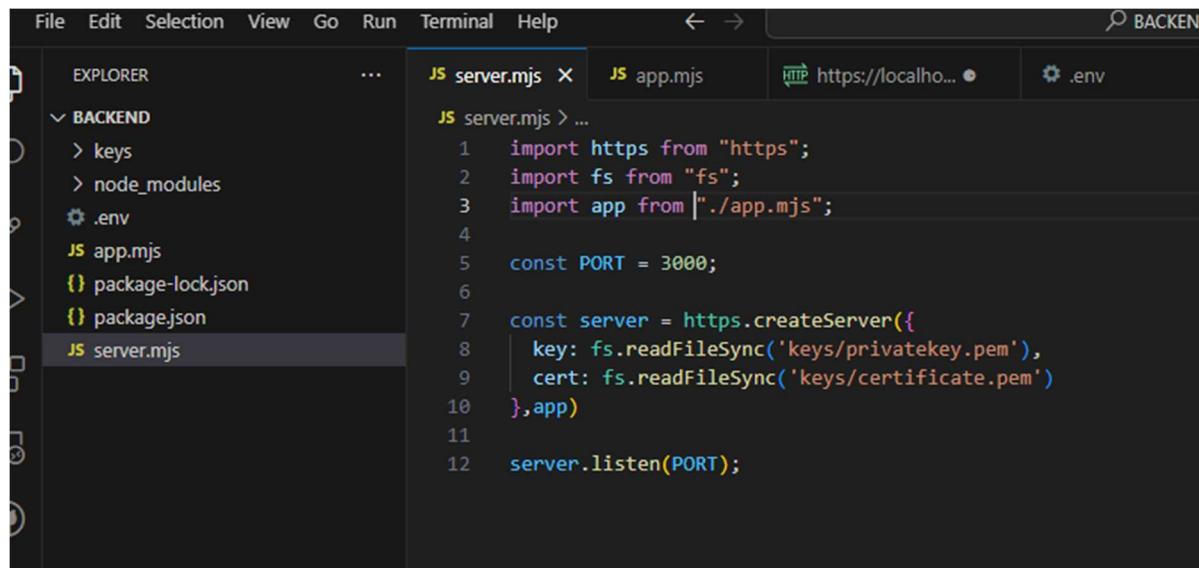
The screenshot shows a Node.js development environment with the following details:

- File Explorer:** Shows files like `certificate.pem`, `privatekey.pem`, `node_modules`, `package-lock.json`, `package.json`, and `server.mjs`.
- Terminal:** Shows the command `[nodemon] starting `node server.mjs``.
- HTTP Client:** A browser-like interface for testing API endpoints.
 - URL:** `http://localhost:3000/api/orders`
 - Method:** GET
 - Headers:** Authorization, Headers (7), Body, Pre-request Script, Tests, Settings
 - Query Params:** Key, Value
 - Body:** Pretty, Raw, Preview, JSON
 - Response:** JSON data representing an array of fruit orders:

```
1 [ { "message": "Fruits", "orders": [ 2 { "id": "1", "name": "Orange" }, 3 { "id": "2", "name": "Apple" }, 4 { "id": "3", "name": "Pear" } ] } ]
```

In order to start securing our server with SSL we need to:

- Start by creating a new folder called “keys” and copying the two .pem files you created earlier using OpenSSL into it.
- Then change your server.js as below.



```

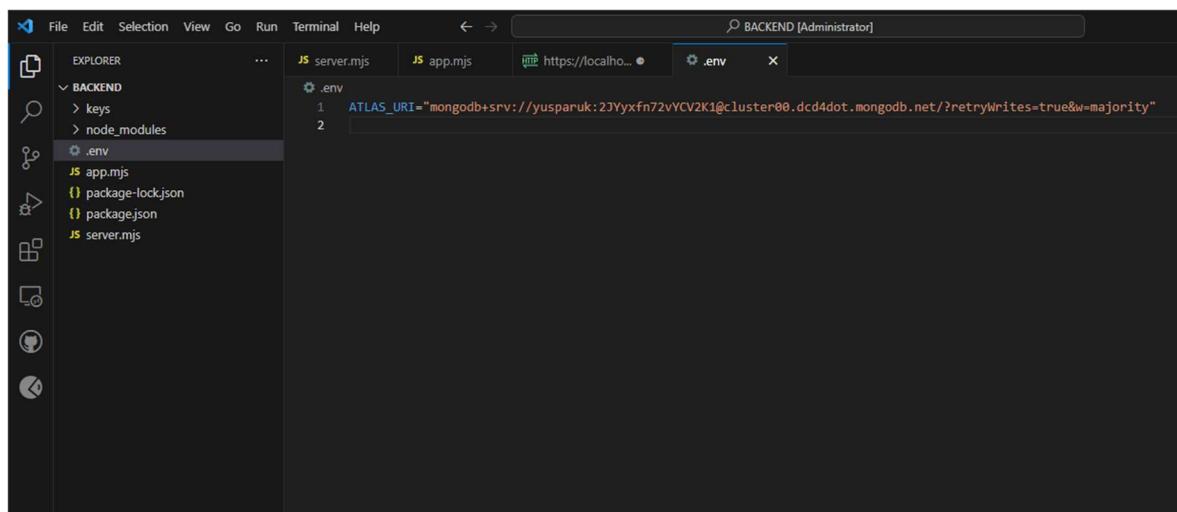
File Edit Selection View Go Run Terminal Help ← → 🔍 BACKEND
EXPLORER ...
BACKEND
  > keys
  > node_modules
  .env
  JS app.mjs
  {} package-lock.json
  {} package.json
JS server.mjs
server.mjs > ...
1 import https from "https";
2 import fs from "fs";
3 import app from "./app.mjs";
4
5 const PORT = 3000;
6
7 const server = https.createServer({
8   key: fs.readFileSync('keys/privatekey.pem'),
9   cert: fs.readFileSync('keys/certificate.pem')
10 },app)
11
12 server.listen(PORT);

```

2.4 Integrating with MongoDB

- To connect to MongoDB, we need to save our connection string as an environment variable.

We create a file called .env and save the connection string within



```

File Edit Selection View Go Run Terminal Help ← → 🔍 BACKEND [Administrator]
EXPLORER ...
BACKEND
  > keys
  > node_modules
  .env
  JS app.mjs
  {} package-lock.json
  {} package.json
  JS server.mjs
.env > ...
1 ATLAS_URI="mongodb+srv://yusparuk:2JYyxfn72vYCV2K1@cluster00.dcd4dot.mongodb.net/?retryWrites=true&w=majority"
2

```

- Next we create a folder called db with our connection file within.
- We also install new packages using the terminal.

This file will connect our app to the database.

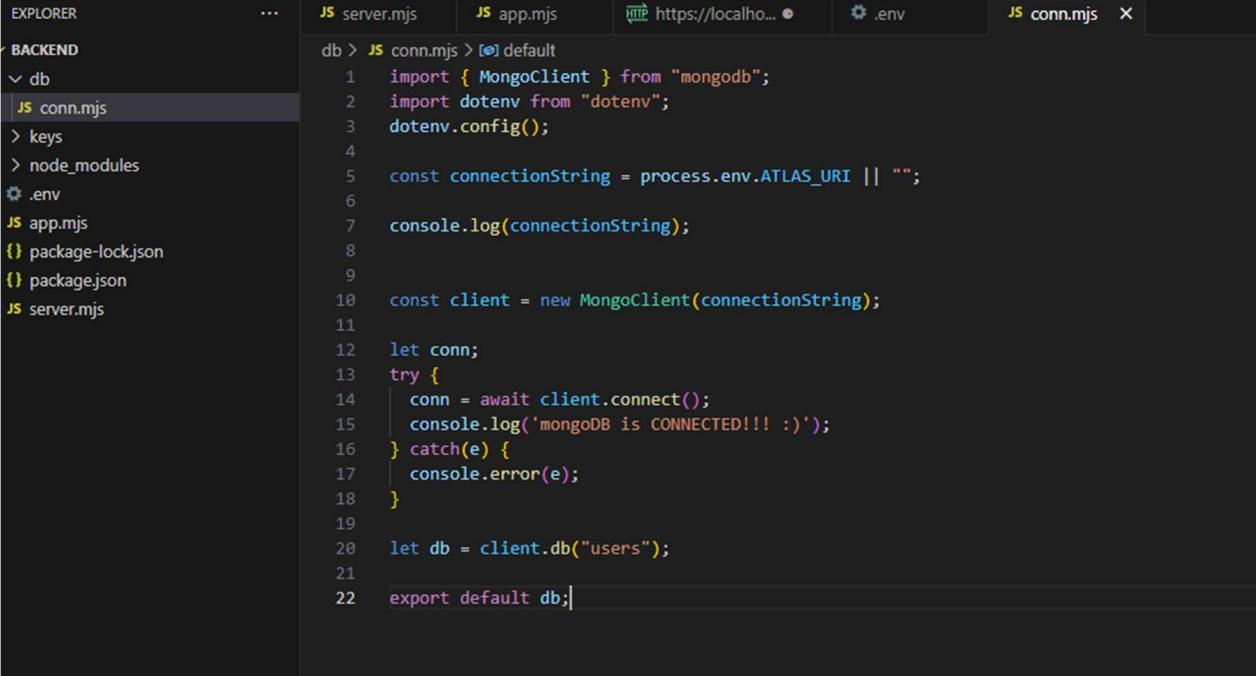
The screenshot shows a terminal window within a code editor interface. The terminal tab is active, displaying the following command and its execution:

```
PS C:\Users\lab_services_student\Desktop\APDS\BACKEND> npm i mongodb mongoose dotenv
added 22 packages, and audited 119 packages in 6s
18 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities
PS C:\Users\lab_services_student\Desktop\APDS\BACKEND>
```

The terminal window has tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, PORTS, and POSTMAN CONSOLE. The TERMINAL tab is selected. The status bar at the bottom right indicates "Ln 1, Col 1".

We can now update the connection file to connect us to the database.

We use MongoClient to connect us to the database and dotenv to load the environment variables.



```
db > JS conn.mjs > [o] default
1 import { MongoClient } from "mongodb";
2 import dotenv from "dotenv";
3 dotenv.config();
4
5 const connectionString = process.env.ATLAS_URI || "";
6
7 console.log(connectionString);
8
9
10 const client = new MongoClient(connectionString);
11
12 let conn;
13 try {
14   conn = await client.connect();
15   console.log('MongoDB is CONNECTED!!! :)');
16 } catch(e) {
17   console.error(e);
18 }
19
20 let db = client.db("users");
21
22 export default db;
```

We are now moving on to Routing, we use routing to route different endpoints to functions. In this case we can create a routes folder and a post.mjs file.

Here we created the route “/” to get all the posts.
And “/upload” to create a post.

```
BACKEND > routes > JS post.mjs > ...
1 import express from "express";
2 import db from "../db/conn.mjs";
3 import { ObjectId } from "mongodb";
4
5 const router = express.Router();
6
7 // Get all the records.
8 router.get("/", async (req, res) => {
9   let collection = await db.collection("posts");
10  let results = await collection.find({}).toArray();
11  res.send(results).status(200);
12 });
13
14 // Create a new record.
15 router.post("/upload", async (req, res) => {
16   let newDocument = {
17     user: req.body.user,
18     content: req.body.content,
19     image: req.body.image
20   };
21   let collection = await db.collection("posts");
22   let result = await collection.insertOne(newDocument);
23   res.send(result).status(204);
24 });
25
```

Our post takes 3 inputs, a username, content and a picture which will be saved as string in the db.

Now we can update our server.mjs

We are including cors to allow our server to accept requests from other endpoints, here we have everything set to "*" which means all.

Then we import our posts from the post.mjs file and we allow our app to use the routes.

```
BACKEND > js server.mjs > ...
1  import https from "https";
2  import http from "http";
3  import fs from "fs";
4  import posts from "./routes/post.mjs";
5  import users from "./routes/user.mjs";
6  import express from "express"
7  import cors from "cors"
8
9  const PORT = 3001;
10 const app = express();
11
12 const options = {
13   key: fs.readFileSync('keys/privatekey.pem'),
14   cert: fs.readFileSync('keys/certificate.pem')
15 }
16
17 app.use(cors());
18 app.use(express.json());
19
20 app.use((req,res,next)=>
21 {
22   res.setHeader('Access-Control-Allow-Origin', '*');
23   res.setHeader('Access-Control-Allow-Headers', '*');
24   res.setHeader('Access-Control-Allow-Methods', '*');
25   next();
26 })
27
28 app.use("/post", posts);
29 app.route("/post", posts);
30 app.use("/user", users);
31 app.route("/user", users);
32
33 let server = https.createServer(options,app)
34 console.log(PORT)
35 server.listen(PORT);
```

Now we can test creating a new post using a POST request.

```

POST https://localhost:3001/post/upload
{
  "user": "Dentistkiller",
  "content": "testing",
  "image": "/9j/4AAQSkZJRgABAQEAYABgAAD/2wBDAAMCAgICAgMDAwMEA+MEB0gFBQQEBQoHbVYIDAoMDAsKCwsNDhIQQ04Rdg...LEBYQERMFUVDa8XGBYUGBIUFRT/...
}
    
```

Body

```

{
  "acknowledged": true,
  "insertedId": "669502d985dfc531d562b5a5"
}
    
```

Status: 200 OK Time: 414 ms Size: 394 B

And we can test if it updated by performing a GET request.
Note for now input a random string to test for the image.

```

GET https://localhost:3001/post
{
  "id": "6694f6b9e3f80157c4198f59",
  "user": "Dentistkiller",
  "content": "testing",
  "image": "/9j/4AAQSkZJRgABAQEAYABgAAD/2wBDAAMCAgICAgMDAwMEA+MEB0gFBQQEBQoHbVYIDAoMDAsKCwsNDhIQQ04Rdg...LEBYQERMFUVDa8XGBYUGBIUFRT/...
}
    
```

Body

```

{
  "id": "6694f6b9e3f80157c4198f59",
  "user": "Dentistkiller",
  "content": "testing",
  "image": "/9j/4AAQSkZJRgABAQEAYABgAAD/2wBDAAMCAgICAgMDAwMEA+MEB0gFBQQEBQoHbVYIDAoMDAsKCwsNDhIQQ04Rdg...LEBYQERMFUVDa8XGBYUGBIUFRT/...
}
    
```

Status: 200 OK Time: 605 ms Size: 94.09 KB

Since we have first 2 endpoints working, we can now update our post.mjs with other operations, like edit (PATCH), and DELETE, as well as to find a specific record by its ID.

```
BACKEND > routes > JS post.mjs > ...
20
27  //Update a record by id
28  router.patch("/:id", async (req, res) => {
29    const query = { _id: new ObjectId(req.params.id) };
30    const updates = {
31      $set: {
32        name: req.body.name,
33        comment: req.body.comment
34      }
35    };
36
37    let collection = await db.collection("posts");
38    let result = await collection.updateOne(query, updates);
39
40    res.send(result).status(200);
41  });
42
43 // Gets a single record by id
44 router.get("/:id", async (req, res) => {
45   let collection = await db.collection("posts");
46   let query = { _id: new ObjectId(req.params.id) };
47   let result = await collection.findOne(query);
48
49   if (!result) res.send("Not found").status(404);
50   else res.send(result).status(200);
51 });
52
53 // Delete a record
54 router.delete("/:id", async (req, res) => {
55   const query = { _id: new ObjectId(req.params.id) };
56
57   const collection = db.collection("posts");
58   let result = await collection.deleteOne(query);
59
60   res.send(result).status(200);
61 });
62
63 export default router;
```

Testing the Patch method.

The screenshot shows a POSTMAN interface with the following details:

- Method:** PATCH
- URL:** https://localhost:3001/post/669502d985dfc531d562b5a5
- Body:** JSON (selected)


```

1  [
2    ...
3      "user": "Dentistkiller",
4      "content": "testing",
5      "image": "no more image"
6  ]

```
- Response Body:** JSON (selected)


```

1  [
2      "acknowledged": true,
3      "modifiedCount": 1,
4      "upsertedId": null,
5      "upsertedCount": 0,
6      "matchedCount": 1
7  ]

```

Testing to get specific updated records.

The screenshot shows a POSTMAN interface with the following details:

- Method:** GET
- URL:** https://localhost:3001/post/669502d985dfc531d562b5a5
- Body:** (empty)
- Response Body:** JSON (selected)


```

1  [
2      "_id": "669502d985dfc531d562b5a5",
3      "user": "Dentistkiller",
4      "content": "testing",
5      "image": "/9j/4AAQSkZJRgABAQEAYAgA2/2wBDAAMCAgMCAgMDAwMEAwMEBQgFBQQEBQoHbwYIDAOmDAsKCwsNDhIQDQ4RDgsLEBYQERMUFRUVDA8XGBYUGBIUFRT/
6          2wDQAMEBAUFBQkUDQsNFQQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBT/wAACAO4BzDwASIAhEBxEB/8QAhAAAQUBAQEBAQFAAAAAAAECAwOFBgcICQo
7          8QAtRAAAGEDAwTEAwUFBAQAAAF9AQIDAQQRBR1HMUEGE1FhByJxFDK8kaETI0KxwRVsFAKm2JyggkKFhcYGrO1JiccoK5o0NTY30Dk6Q0RFRkdISUpTVFVVV1hZwmNkZwZnaG1qc3R1dn4eXqDhIWGh41JipKT
8          KztLW2t71suLdxIXCx8jyELT1NxW19jZ2uhI4+T1SuFoSerx8vP09fb3+Pn6/8QAhwEAwAEBAQFQAQFQAQFQAQFQAQFQAQFQAQFQgICtCQ6L/
9          8QAtREAAgCBAQD0BACFBAQD0AQJ3AAECAxEEBSEx8hJBUDQdhcRM1MeIFEKRobhBCSzluAVynLRChYkNOE18rcyGRomJygpKJU2Nzg50KNERUZHE1KU1RVVldYhVpjZGmZ2hpanN0dXZ3eh16goEhYaHiImK
10         mqsR08tb23uLm0wsPExcbHyMnK0PjPU1dbX2Nna4uPk5ebn6Onq8vP09fb3+Pn6/9oADAMBAAIRAXeAPd04TX7ppiehvS1kat5BevezetFItl/EKAD+TUFxGj
11         +10c9qAcijnvRQAUDxcmi1igA7njFFFH44oAKKPxzQAbttG7dRSNQatHPakWloAoe9FwoU0lRRRQAUUUUAHpbjo47de9H4zo/DfAB/EKN21jr/CaP4RQAjNuow1pGoAWikVttL3UAFFHPbp3o47de9ABRR
12         .0%Pxz7054U10cc0bSsSS0LAEfEAR/cjPABRROAfE1-E3a1591174198804AUfU0Qx81Lw/DEFHRLQXK248804Ulb718364301SNR77tuoyYwvR8R00AULHUL0av80E045B/CaP4b5ymg/loc/

```

Testing Delete method.

The screenshot shows the Postman application interface. At the top, there is a header bar with the URL `https://localhost:3001/post/669502d985dfc531d562b5a5`. Below this, the main interface has a "DELETE" button and the same URL in the address bar. The "Body" tab is selected, showing the following JSON response:

```
1 {  
2   "acknowledged": true,  
3   "deletedCount": 1  
4 }
```

Below the body, there is a note: "This request does not have a response body".

The screenshot shows the Postman application interface. At the top, there is a header bar with the URL <https://localhost:3001/post/669502d985dfc531d562b5a5>. Below this, a dropdown menu shows "GET" selected. The main workspace has tabs for "Params", "Authorization", "Headers (7)", "Body", "Pre-request Script", "Tests", and "Settings". The "Body" tab is active, showing options: "none" (selected), "form-data", "x-www-form-urlencoded", "raw", "binary", and "GraphQL". Below these tabs, there are buttons for "Body", "Cookies", "Headers (10)", and "Test Results", with "Body" being the active tab. Under the "Body" tab, there are buttons for "Pretty" (selected), "Raw", "Preview", "HTML", and a copy icon. The main content area displays the response: "1 Not found".

We can see its deleted since we cant find a record with the id anymore.

Moving on to our User Routes.

We can start by creating a user.mjs file in the routes folder,
And at the terminal we can install our required packages.

The screenshot shows the Visual Studio Code interface. On the left is the Explorer sidebar with a tree view of the project structure. The 'routes' folder contains 'fruit.mjs' and 'user.mjs'. Other files like 'conn.mjs', 'keys', 'node_modules', '.env', 'app.mjs', 'loadenvironment.mjs', 'package-lock.json', 'package.json', and 'server.mjs' are also listed. The 'TERMINAL' tab is active at the bottom, showing the output of an npm install command. It displays:

```
added 66 packages, and audited 185 packages in 5s
21 packages are looking for funding
  run `npm fund` for details
  found 0 vulnerabilities
  PS C:\Users\lab_services_student\Desktop\APDS\BACKEND> npm i bcrypt jsonwebtoken
```

Below the terminal, another terminal window shows the command:

```
PS C:\Users\lab_services_student\Desktop\APDS\BACKEND> npm i express brute
added 1 package, and audited 186 packages in 3s
21 packages are looking for funding
  run `npm fund` for details
```

We are using:

- Jsonwebtoken(JWT), a secure token authentication system.
- Bcrypt, which is a powerful encryption tool.
- Express Brute, for bruteforce prevention.

In creating our signup method, we use bcrypt to hash the password.

```
routes > JS user.mjs > ...
1   import express from "express";
2   import db from "../db/conn.mjs";
3   import { ObjectId } from "mongodb";
4   import bcrypt from "bcrypt";
5   import jwt from "jsonwebtoken";
6   import ExpressBrute from "express-brute";
7
8   const router = express.Router();
9
10  var store = new ExpressBrute.MemoryStore();
11  var bruteforce = new ExpressBrute(store);
12
13  //sign up
14  router.post("/signup", async (req, res) => {
15      const password = bcrypt.hash(req.body.password, 10)
16      let newDocument = {
17          name: req.body.name,
18          password: (await password).toString()
19      };
20      let collection = await db.collection("users");
21      let result = await collection.insertOne(newDocument);
22      console.log(password);
23      res.send(result).status(204);
24  });
}
```

We can also add our login method.

We are using brute force. Prevent to prevent brute force attacks

Once a user is successfully logged in they get assigned a token encoded with their credentials.

```
25 //login
26 router.post("/login",bruteforce.prevent, async (req, res) => {
27   const { name, password } = req.body;
28   console.log(name + " " + password)
29
30   try {
31     const collection = await db.collection("users");
32     const user = await collection.findOne({ name });
33
34     if (!user) {
35       return res.status(401).json({ message: "Authentication failed" });
36     }
37
38     // Compare the provided password with the hashed password in the database
39     const passwordMatch = await bcrypt.compare(password, user.password);
40
41     if (!passwordMatch) {
42       return res.status(401).json({ message: "Authentication failed" });
43     }
44     else{
45       // Authentication successful
46       const token = jwt.sign({username:req.body.username,password:req.body.password}, "this_secret_should_be_longer_than_it_is", {expiresIn:"1h"})
47       res.status(200).json({ message: "Authentication successful", token: token, name: req.body.name });
48       console.log("your new token is", token)
49     }
50   } catch (error) {
51     console.error("Login error:", error);
52     res.status(500).json({ message: "Login failed" });
53   }
54 }
55 );
56
57 export default router
```

We can now also update the server file to allow for user routes.

```
js server.mjs > ⚙ app.use() callback
1 import https from "https";
2 import http from "http";
3 import fs from "fs";
4 import fruits from "./routes/fruit.mjs";
5 import users from "./routes/user.mjs";
6 import express from "express"
7 import cors from "cors"
8
9 const PORT = 3000;
10 const app = express();
11
12 const options = {
13   key: fs.readFileSync('keys/privatekey.pem'),
14   cert: fs.readFileSync('keys/certificate.pem')
15 }
16
17 app.use(cors());
18 app.use(express.json());
19
20 app.use((req,res,next)=>
21 {
22   res.setHeader('Access-Control-Allow-Origin', '*');
23   res.setHeader('Access-Control-Allow-Headers', '*');
24   res.setHeader('Access-Control-Allow-Methods', '*');
25   next();
26 })
27
28 app.use("/fruit", fruits);
29 app.route("/fruit", fruits);
30 app.use("/user", users);
31 app.route("/user", users);
32
33 let server = https.createServer(options,app)
34
35 server.listen(PORT);
```

Testing our login and registration methods.

The screenshot shows the Postman interface for testing a POST request to `https://localhost:3000/user/signup/`. The request body is set to raw JSON:

```
1 {
2   "name": "Dentistkiller",
3   "password": "friedgreentomatoes"
4 }
```

The response body is also in raw JSON:

```
1 [
2   {
3     "acknowledged": true,
4     "insertedId": "66927082c64d05d958dba1b9"
5   }
]
```

At the bottom, the terminal window shows the MongoDB connection status:

```
mongoDB is CONNECTED!!! :)
```

```
Promise { '$2b$10$Ao5EcniI4QBjxpFqHb58huowJgLy0GxJ41QWBnpPzail9Rl08fIia'
```

On successful login you can see the generated token.

The screenshot shows the Postman interface. The URL is `https://localhost:3000/user/login/`. The method is `POST`. The `Body` tab is selected, showing the following JSON payload:

```

1  [
2    "name": "Dentistkiller",
3    "password": "friedgreenatoes"
4  ]

```

The `Headers` tab shows `(9)` headers. The `Body` tab shows `raw` JSON content. The response status is `200 OK` with a time of `175 ms`. The response body is:

```

1  [
2    "message": "Authentication successful"
3  ]

```

The bottom panel shows the terminal output:

```
[nodemon] starting `node server.mjs`  
mongodb is CONNECTED!!! :)  
your new token is eyJhbGciOiJIUzI1NiIsInR5C16IkpxVCJ9.eyJwYXNzd29yZC16ImZyaWVzZ3JlZW5ob21hdG9lcysImIhdCI6MTcyMDg3MzQzOSwiZXhwIjoxNzIwODc3MDM5fQ.cMDh6Lthe6xen7qMlPui0PEX7AjBog7iD4Xv2-p6c
```

Now that we have our token we need to use it to verify if someone's session is valid, to do this we need to create a class to check if a user is authenticated.

The screenshot shows the VS Code editor with the `check-auth.js` file open. The code defines a function `checkauth` that takes `req`, `res`, and `next` as parameters. It uses the `jwt.verify` method to decode the token and passes control to the next handler if successful. If there is an error, it sends a `401` response with a message indicating the token is invalid.

```

JS check-auth.js > checkauth
1 import jwt from "jsonwebtoken"
2
3 const checkauth=(req,res,next)=>
4 {
5   try{
6     const token = req.headers.authorization.split(" ")[1];
7     jwt.verify(token,"this_secret_should_be_longer_than_it_is")
8     next(); //passes control to the next handler
9   }
10  catch(error)
11  {
12    res.status(401).json([
13      {
14        message: "token invalid"
15      }
16    ]);
17  };
18
19 export default checkauth

```

We can now update our post.mjs routes to include the checkauth when a request is being made.

```
15 // Create a new record.
16 router.post("/upload", checkauth, async (req, res) => {
17   let newDocument = {
18     user: req.body.user,
19     content: req.body.content,
20     image: req.body.image
21   };
22   let collection = await db.collection("posts");
23   let result = await collection.insertOne(newDocument);
24   res.send(result).status(204);
25 });
26
27 //Update a record by id
28 router.patch("/:id", checkauth, async (req, res) => {
29   const query = { _id: new ObjectId(req.params.id) };
30   const updates = {
31     $set: {
32       name: req.body.name,
33       comment: req.body.comment
34     }
35   };
36
37   let collection = await db.collection("posts");
38   let result = await collection.updateOne(query, updates);
39
40   res.send(result).status(200);
41 });
42
```

Now to test in postman we need to include the token generated from logging as a header in your request, we select authorization -> Bearer token.

The screenshot shows a Postman interface with the following details:

- Method:** POST
- URL:** https://localhost:3001/post/upload
- Authorization Type:** Bearer Token
- Token:** eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9eyJwYXNzd29yZC16ImZyaWVkZ3JIZW50b21hdG9lcylsImh0dC16MTcyMTA0MjE0MSwiZxhwjoaNzIxMDQ1NzQxfQ.pedr69mk7NTSRTAIFs1ZeMwZFX1StfkE0eoNIWg8LA
- Body:** (Pretty) JSON response:

```
1  [
2   {
3     "acknowledged": true,
4     "insertedId": "6695051d304ea22ba36be2a8"
5   }
]
```

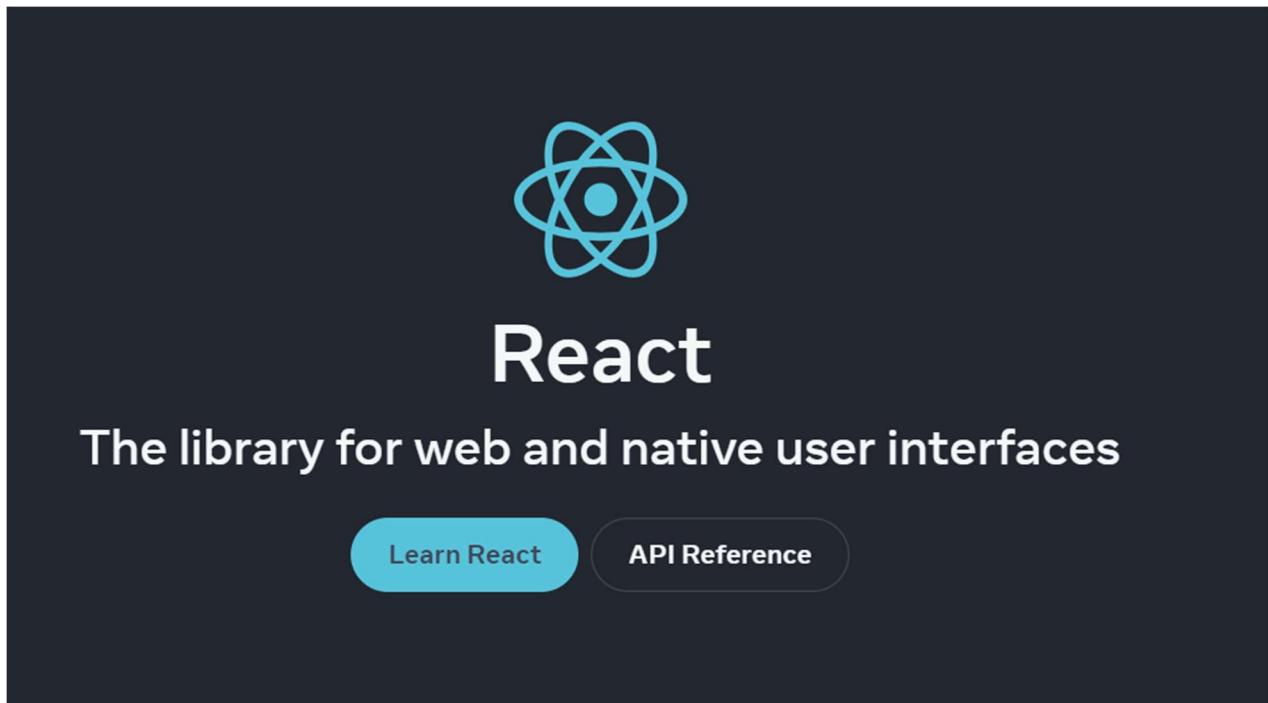
You should now have a fully functioning backend application.

3 Frontend

3.1 First React Application

The frontend development we use is React – the “R” from MERN . <https://react.dev/> [Accessed 13 July 2024].

Learn more by following their docs. <https://react.dev/learn/start-a-new-react-project> [Accessed 13 July 2024].



Introduction to React

React is a widely adopted JavaScript library developed by Facebook for creating user interfaces, especially for single-page applications. It enables developers to build interactive and efficient UIs through a component-based architecture. Here is a detailed overview of the fundamental concepts of React:

1 Components

Components are the fundamental building blocks of a React application. They are self-contained units that define parts of the user interface. React supports two main types of components:

Functional Components: These are JavaScript functions that accept inputs (known as props) and return React elements. They are generally used for components that do not require internal state management or lifecycle methods.

Class Components: These are ES6 classes that extend React.Component and include a render method which returns React elements. Class components can manage internal state and utilise lifecycle methods to handle various stages of a component's existence.

2 JSX (*JavaScript XML*)

JSX is a syntax extension for JavaScript that allows developers to write HTML-like structures within JavaScript code. JSX enhances readability and writing efficiency, as it merges the concerns of HTML and JavaScript into a single, coherent structure. This syntax is compiled into standard JavaScript function calls that describe the user interface.

3 *Props*

Props, short for "properties," are immutable attributes used to pass data and event handlers from parent components to child components. Props enable dynamic rendering of components based on the data provided and facilitate component reusability and composition.

4 *State*

State represents internal data that affects how a component behaves and renders. Unlike props, state is mutable and managed within the component itself. It can be updated in response to user actions or other events, which triggers re-rendering to reflect changes in the user interface.

5 *Lifecycle Methods*

Lifecycle methods are special functions in class components that are invoked at various stages of a component's lifecycle. These methods allow developers to execute code at specific points, such as when a component mounts, updates, or unmounts, enabling the management of side effects and component behaviour.

6 *Hooks*

Hooks are functions introduced in React 16.8 that enable functional components to use state and lifecycle features, which were previously only available in class components. Essential hooks include useState for managing state, useEffect for handling side effects, and useContext for accessing context values.

7 *Context API*

The Context API is a feature that allows the sharing of global data across the component tree without the need to pass props manually through every level. It is used for managing global state or settings, such as user authentication status or application theme.

8 React Router

React Router is a library used for implementing routing in React applications. It facilitates the definition of different routes and navigation between components or pages, enabling the creation of complex, multi-view applications.

9 Virtual DOM

The Virtual DOM is a lightweight, in-memory representation of the actual DOM. React uses the Virtual DOM to efficiently manage and apply updates to the real DOM by comparing the current and previous states and only making necessary changes. This approach enhances performance and ensures that updates are executed in the most efficient manner.

https://www.w3schools.com/react/react_intro.asp [Accessed 10 July 2024]

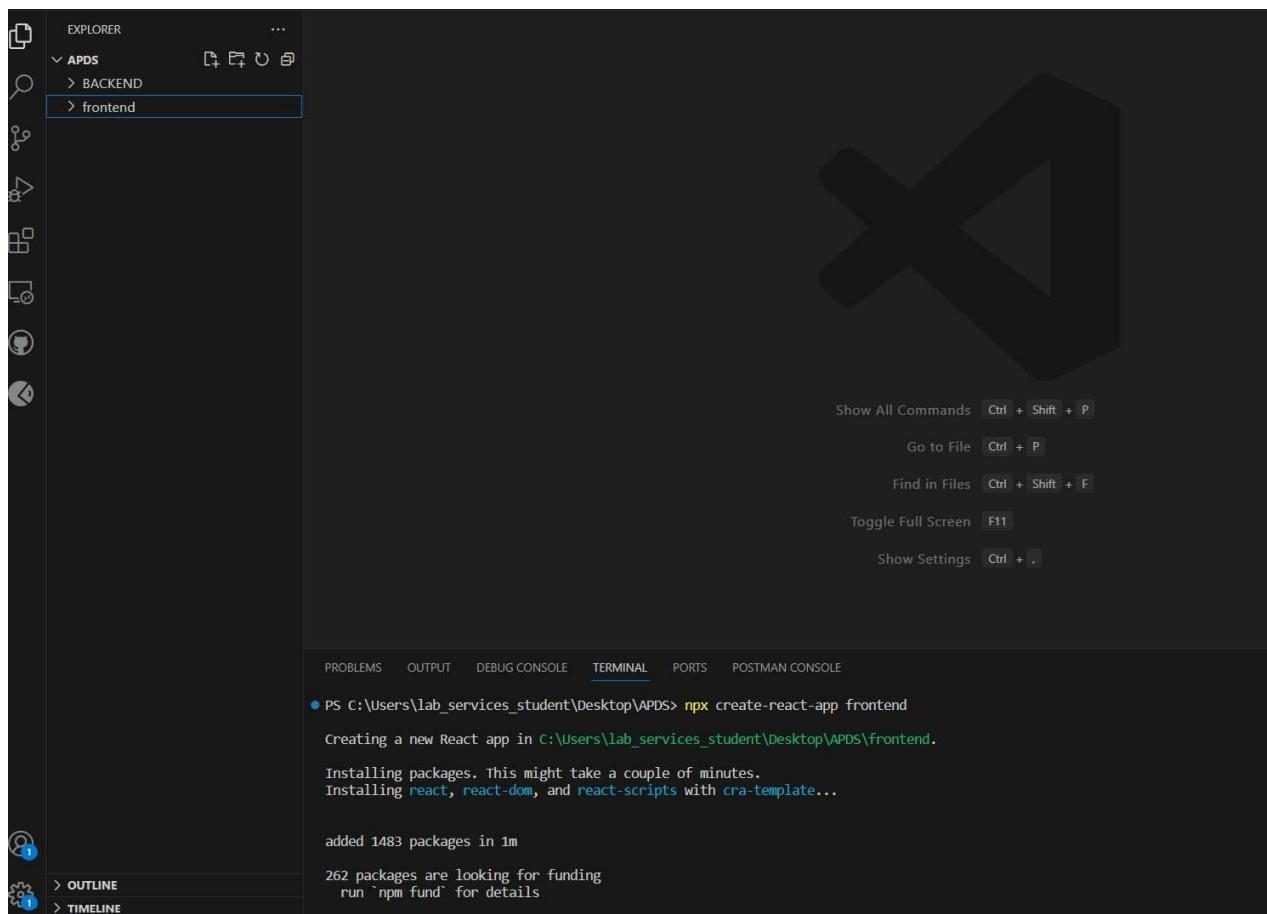
Run the following command inside a terminal to create a new application/project called “frontend”.

Note you must be in a folder that is generally accessible to you, I created an APDS folder containing both frontend and backend folders.

- `npx create-react-app frontend`

This creates a “barebones” application for your project in the “frontend” folder and installs the required packages in a folder called “node_modules”. Accept the default for any questions.

NOTE node_modules can be safely deleted and recreated – useful to reduce the size of the project for copying and moving.

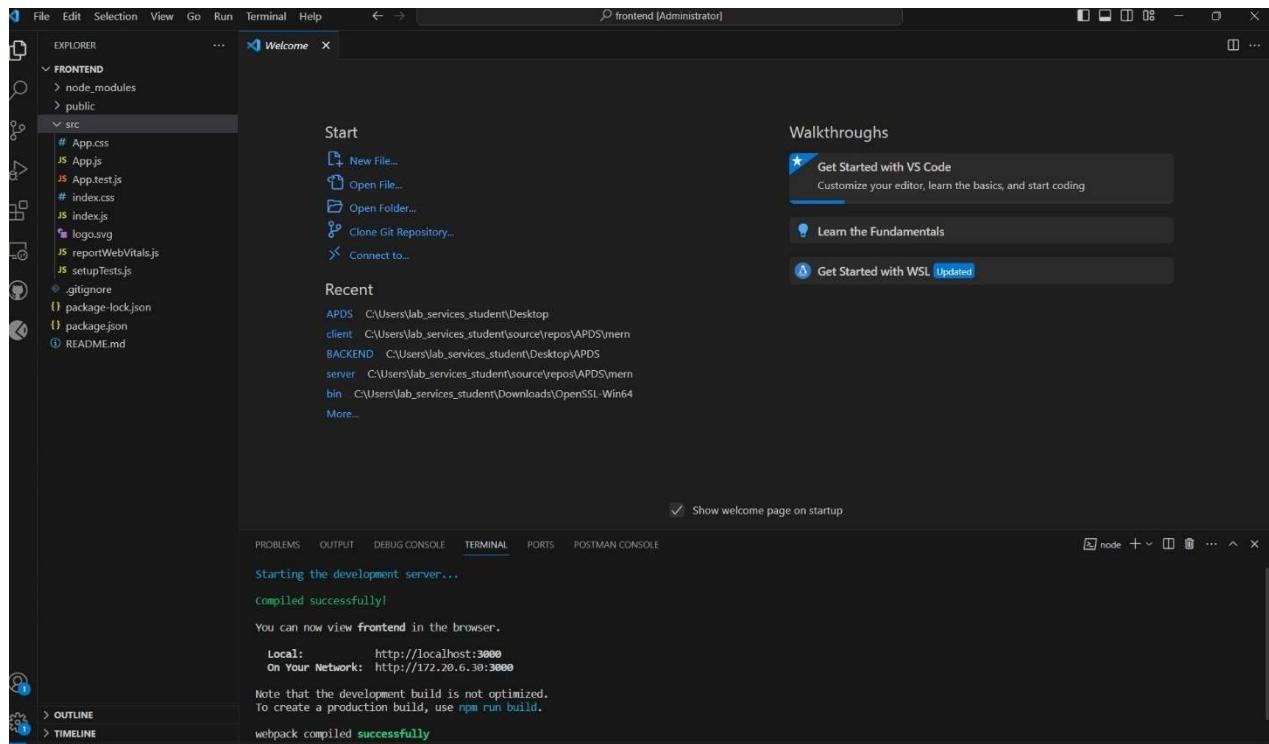


The screenshot shows the Visual Studio Code (VS Code) interface. On the left is the Explorer sidebar, which displays a folder structure with 'APDS' expanded, showing 'BACKEND' and 'frontend'. The 'frontend' folder is selected and highlighted with a blue border. The main workspace is dark-themed with a large, semi-transparent 'X' logo in the center. At the bottom, the Terminal tab is active, showing the command line output of the 'create-react-app' command:

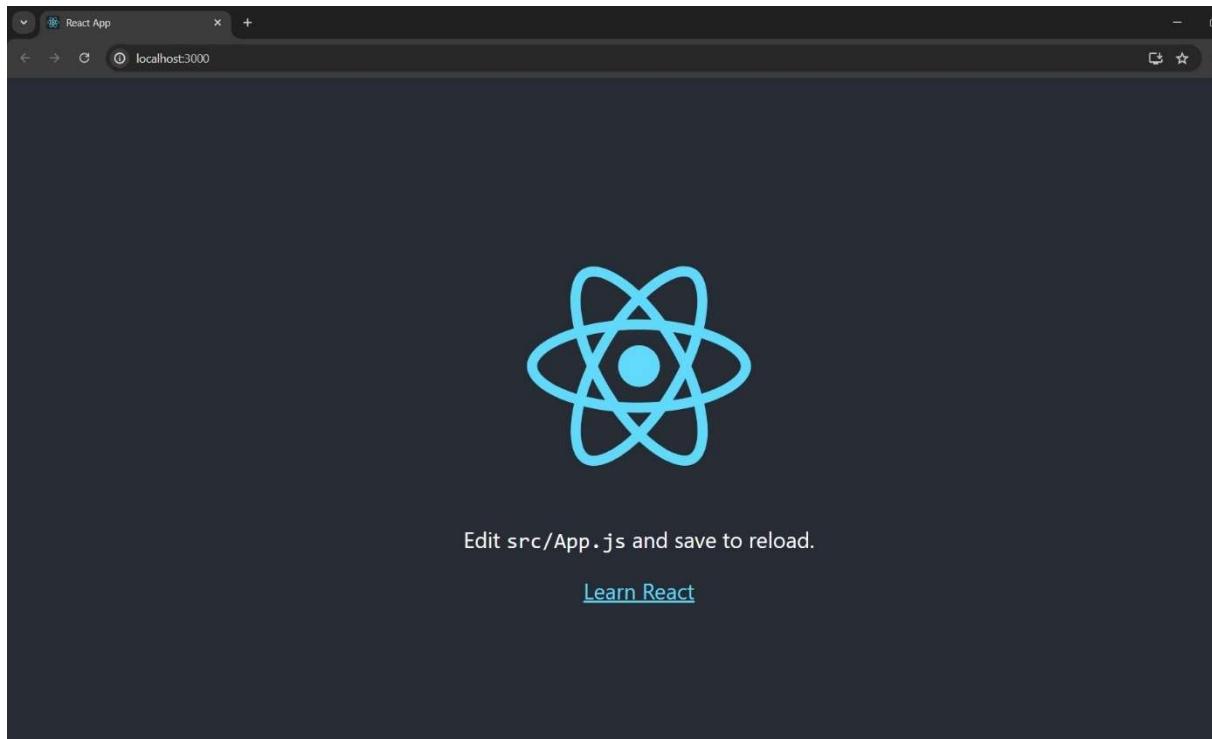
```
PS C:\Users\lab_services_student\Desktop\APDS> npx create-react-app frontend
Creating a new React app in C:\Users\lab_services_student\Desktop\APDS\frontend.
Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts with cra-template...
added 1483 packages in 1m
262 packages are looking for funding
  run `npm fund` for details
```

Once the folder is created, we can open the folder in Visual Studio.

We can navigate to the frontend colder in the terminal by using “cd frontend” and then we run “npm start” to run our react app.

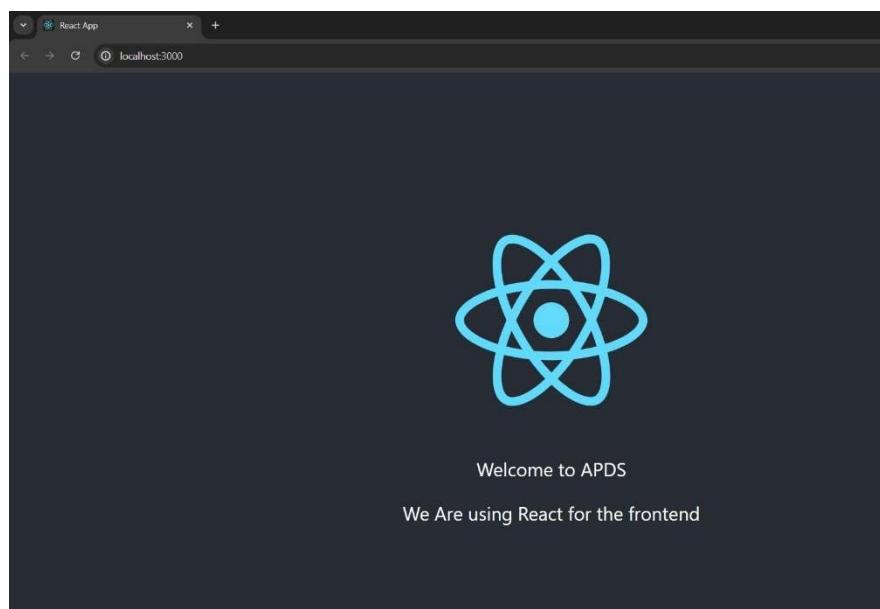


What our browser will look like.



Now let's edit our app.js to display something else.

```
JS App.js    X  
src > JS App.js > ...  
1 import logo from './logo.svg';  
2 import './App.css';  
3  
4 function App() {  
5     return (  
6         <div className="App">  
7             <header className="App-header">  
8                 <img src={logo} className="App-logo" alt="logo" />  
9                 <p>  
10                    Welcome to APDS  
11                </p>  
12                <a href="#">  
13                    We Are using React for the frontend  
14                </a>  
15            </header>  
16        </div>  
17    );  
18}  
19  
20 export default App;  
21
```



Run the following command to install the router, this will connect the backend api to the front

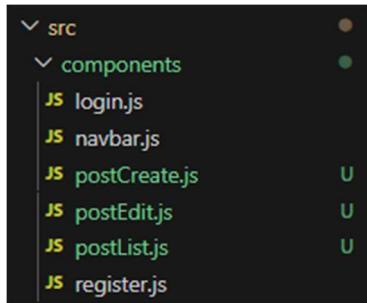
end.

```
● PS C:\Users\lab_services_student\Desktop\APDS\frontend> npm i react-router-dom
added 3 packages, and audited 1549 packages in 5s
262 packages are looking for funding
  run `npm fund` for details
8 vulnerabilities (2 moderate, 6 high)

To address all issues (including breaking changes), run:
  npm audit fix --force

Run `npm audit` for details.
```

Now we can start creating our frontend by creating our components.
We will create a folder and within that create files for all our components.



The provided React code sets up routing for the application, including paths for viewing the post list, editing a post, creating a new post , user registration, and user login, along with a navbar that is displayed on all pages.

```
frontend > src > JS App.js > [o] App
1  import './App.css';
2  import React from "react";
3  // We use Route in order to define the different routes of our application
4  import { Route, Routes } from "react-router-dom";
5  // We import all the components we need in our app
6  import Navbar from "./components/navbar";
7  import PostList from "./components/postList";
8  import EditPost from "./components/postEdit";
9  import CreatePost from "./components/postCreate";
10 import Register from "./components/register";
11 import Login from "./components/login";
12
13 const App = () => {
14   return [
15     <div>
16       <Navbar />
17       <Routes>
18         <Route exact path="/" element={<PostList />} />
19         <Route path="/edit/:id" element={<EditPost />} />
20         <Route path="/create" element={<CreatePost />} />
21         <Route path="/register" element={<Register />} />
22         <Route path="/login" element={<Login />} />
23       </Routes>
24     </div>
25   ];
26 };
27
28 export default App;
29
```

We can now update our defaults “/” page with is the postList.js
 This function returns the HTML used to design the UI, creating the display.

```
frontend > src > components > js postList.js > ...
1 import React, { useEffect, useState } from "react";
2 import { Link } from "react-router-dom";
3
4 export default function PostList() {
5   return (
6     <body>
7       <div className="container">
8         <h3 className="header">APDS notice Board</h3>
9         <table className="table table-striped" style={{ marginTop: 20 }}>
10          <thead>
11            <tr>
12              <th>User</th>
13              <th>Caption</th>
14              <th>Image</th>
15              <th>Actions</th> /* Added column for actions */
16            </tr>
17          </thead>
18        </table>
19      </div>
20    </body>
21  );
22}
```

The navbar.js also needs to be updated to allow navigation between the different pages.

```
frontend > src > components > js navbar.js > Navbar
1 import React from "react";
2 import logo from "../logo.svg"
3 // We import bootstrap to make our application look better.
4 import "bootstrap/dist/css/bootstrap.css";
5 // We import NavLink to utilize the react router.
6 import { NavLink } from "react-router-dom";
7 // Here, we display our Navbar
8 export default function Navbar() {
9   return (
10     <div>
11       <nav className="navbar navbar-expand-lg navbar-light bg-light">
12         <NavLink className="navbar-brand" to="/">
13           <img style={{ width: 25 + '%' }} src={logo}></img>
14         </NavLink>
15         <div className="navbar" id="navbarSupportedContent">
16           <ul className="navbar-nav ml-auto">
17             <NavLink className="nav-link" to="/">
18               List
19             </NavLink>
20             <NavLink className="nav-link" to="/create">
21               Create Post
22             </NavLink>
23             <NavLink className="nav-link" to="/register">
24               Register
25             </NavLink>
26             <NavLink className="nav-link" to="/login">
27               Login
28             </NavLink>
29           </ul>
30         </div>
31       </nav>
32     </div>
33   );
34 }
```

This is what the app should look like at this point.

The screenshot shows a web browser window with the URL 'localhost:3000'. The page title is 'APDS notice Board'. At the top, there is a navigation bar with links for 'List', 'Add Fruit', 'Register', and 'Login'. Below the title, there is a table with four columns: 'User', 'Caption', 'Image', and 'Actions'. The table is currently empty, displaying only its header row.

User	Caption	Image	Actions
------	---------	-------	---------

Now to add functionality to this design.

This code fetches a list of posts from a database and displays them in a table with an option to delete each post, using hooks `useEffect` for data fetching and `useState` for state management.

```
frontend > src > components > JS postList.js > ...
1 import React, { useEffect, useState } from "react";
2 import { Link } from "react-router-dom";
3 import './App.css';
4
5 const Post = (props) => (
6   <tr>
7     |   <td>{props.post.user}</td>
8     |   <td>{props.post.content}</td>
9     <td>
10       |   {props.post.image && (
11         |     <img
12           |       src={`data:image/jpeg;base64,${props.post.image}`} // Convert base64 string to image
13           |       alt="Post Image"
14           |       style={{ maxWidth: '100px', maxHeight: '100px', objectFit: 'cover' }} // Ensure the image fits within the size limits
15         |     />
16       )
17     </td>
18     <td>
19       |   <button className="btn btn-link"
20         |     onClick={() => {
21           |       props.deletePost(props.post._id);
22         }}
23       >
24         |       Delete
25       </button>
26     </td>
27   </tr>
28 );
29
30 export default function PostList() {
31   const [posts, setPosts] = useState([]);
32
33   // This method fetches the posts from the database.
34   useEffect(() => {
35     async function getPosts() {
36       const response = await fetch(`https://localhost:3001/post/`);
37
38       if (!response.ok) {
39         const message = `An error occurred: ${response.statusText}`;
40         window.alert(message);
41         return;
42       }
43
44       const posts = await response.json();
45       setPosts(posts);
46     }
47
48     getPosts();
49
50     return;
51   }, [posts.length]);
```

```
frontend > src > components > JS postList.js > ...
51     ), [posts.length]);
52
53     // This method will delete a post
54     async function deletePost(id) {
55         const token = localStorage.getItem("jwt");
56         await fetch(`https://localhost:3001/post/${id}`, {
57             method: "DELETE",
58             headers: {
59                 "Authorization": `Bearer ${token}`,
60             },
61         });
62
63         const newPosts = posts.filter((el) => el._id !== id);
64         setPosts(newPosts);
65     }
66
67     // This method will map out the posts on the table
68     function PostList() {
69         return posts.map((post) => {
70             return (
71                 <Post
72                     post={post}
73                     deletePost={() => deletePost(post._id)}
74                     key={post._id}
75                 />
76             );
77         );
78     }
79
80     return (
81         <body>
82             <div className="container">
83                 <h3 className="header">APDS notice Board</h3>
84                 <table className="table table-striped" style={{ marginTop: 20 }}>
85                     <thead>
86                         <tr>
87                             <th>User</th>
88                             <th>Caption</th>
89                             <th>Image</th>
90                             <th>Actions</th> /* Added column for actions */
91                         </tr>
92                     </thead>
93                     <tbody>{PostList()}</tbody>
94                 </table>
95             </div>
96         </body>
97     );
98 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE COMMENTS
webpack compiled with 1 warning

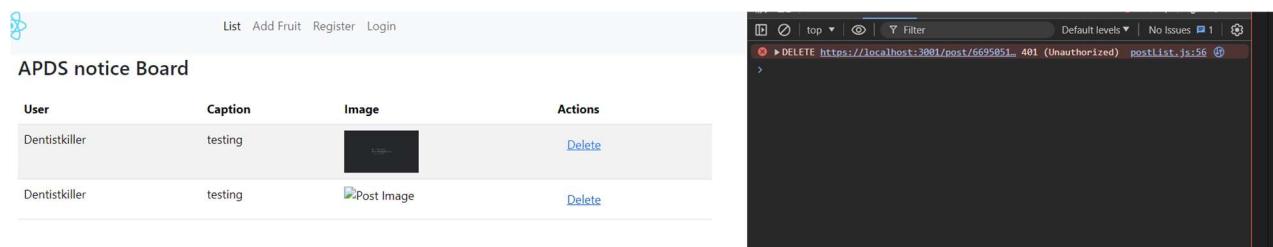
When accessing the website you can now see the site update with data from the database.



The screenshot shows a web browser window with the URL 'localhost:3000'. The page title is 'APDS notice Board'. At the top, there is a navigation bar with links: 'List', 'Add Fruit', 'Register', and 'Login'. Below the navigation bar, there is a small blue icon of a person. The main content area displays a table with four columns: 'User', 'Caption', 'Image', and 'Actions'. There are two rows of data:

User	Caption	Image	Actions
Dentistkiller	testing		Delete
Dentistkiller	testing		Delete

If a delete is attempted, since the user is not logged in, it will return as unauthorized. We use the developer tools in your selected browser to help us test in React.



The screenshot shows a browser window with the same 'APDS notice Board' application. To the right of the application, the browser's developer tools are open, specifically the Network tab. A single entry is listed:

- Method: DELETE
- URL: <https://localhost:3001/post/6695051>
- Status: 401 (Unauthorized)
- Message: postlist.js:56

In order to authorize a user, we need to start with updating the signup.js file.

```
src > components > JS register.js > .onSubmit
1 import React, { useState } from "react";
2 import { useNavigate } from "react-router";
3 export default function Register() {
4   const [form, setForm] = useState([
5     name: "",
6     password: "",
7   ]);
8   const navigate = useNavigate();
9   // These methods will update the state properties.
10  function updateForm(value) {
11    return setForm((prev) => {
12      return { ...prev, ...value };
13    });
14  }
15  // This function will handle the submission.
16  async function onSubmit(e) [
17    e.preventDefault();
18
19    // When a post request is sent to the create url, we'll create a new user in the database.
20    const newPerson = { ...form };
21
22    await fetch("http://localhost:3001/user/signup", {
23      method: "POST",
24      headers: {
25        "Content-Type": "application/json",
26      },
27      body: JSON.stringify(newPerson),
28    })
29    .catch(error => {
30      window.alert(error);
31      return;
32    });
33
34    setForm({ name: "", password: ""});
35    navigate("/");
36  ]
]
```

```
37 // This following section will display the form that takes the input from the user.
38 return (
39   <div>
40     <h3>Register</h3>
41     <form onSubmit={onSubmit}>
42       <div className="form-group">
43         <label htmlFor="name">Name</label>
44         <input
45           type="text"
46           className="form-control"
47           id="name"
48           value={form.name}
49           onChange={(e) => updateForm({ name: e.target.value })}
50         />
51       </div>
52       <div className="form-group">
53         <label htmlFor="password">Password</label>
54         <input
55           type="text"
56           className="form-control"
57           id="password"
58           value={form.position}
59           onChange={(e) => updateForm({ position: e.target.value })}
60         />
61       </div>
62       <div className="form-group">
63         <input
64           type="submit"
65           value="Create person"
66           className="btn btn-primary"
67         />
68       </div>
69     </form>
70   </div>
71 );
72 }
73 }
```

We will also need to update the login.js file.

```

frontend > src > components > # login.js ...
1 import React, { useState } from "react";
2 import { useNavigate } from "react-router";
3 export default function Login() {
4   const [form, setForm] = useState({
5     name: "",
6     password: "",
7   });
8   const navigate = useNavigate();
9
10  function updateForm(value) {
11    return setForm((prev) => {
12      return { ...prev, ...value };
13    });
14  }
15  // This function will handle the submission.
16  async function onSubmit(e) {
17    e.preventDefault();
18
19    // Passes the form data to the API on our backend.
20    const newPerson = { ...form };
21
22    const response = await fetch("https://localhost:3001/user/login", {
23      method: "POST",
24      headers: {
25        "Content-Type": "application/json",
26      },
27      body: JSON.stringify(newPerson),
28    })
29    .catch(error => {
30      window.alert(error);
31      return;
32    });
33
34    const data = await response.json();
35    const { token, name } = data;
36    console.log(name + " " + token)
37
38    // Save the JWT to localStorage
39    localStorage.setItem("jwt", token);
40
41    // Optionally, save the username if needed
42    localStorage.setItem("name", name);
43
44    setForm({ name: "", password: "" });
45    navigate("/");
46  }
47  // HTML for the login page.
48  return (
49    <div>
50      <h3>Login</h3>
51      <form onSubmit={onSubmit}>
52        <div className="form-group">
53          <label htmlFor="name">Name</label>
54          <input
55            type="text"
56            className="form-control"
57            id="name"
58            value={form.name}
59            onChange={(e) => updateForm({ name: e.target.value })}
60          />
61        </div>
62        <div className="form-group">
63          <label htmlFor="password">Password</label>
64          <input
65            type="text"
66            className="form-control"
67            id="password"
68            value={form.password}
69            onChange={(e) => updateForm({ password: e.target.value })}
70          />
71        </div>
72
73        <div className="form-group">
74          <input
75            type="submit"
76            value="Login"
77            className="btn btn-primary"
78          />
79        </div>
80      </form>
81    </div>
82  );
83}

```

Our login and registration should now look something like this.

The screenshot shows a browser window titled "React App" with the URL "localhost:3000/register". The page has a light gray header with a small logo on the left and navigation links: "List", "Add Fruit", "Register", and "Login". Below the header is a section titled "Register". It contains two input fields: one for "Name" with the value "NewTestUser" and another for "Password" with the value "P@ssword123". At the bottom of this section is a blue button labeled "Create person".

A successful registration should display in the developer console.

The screenshot shows a developer console with a log entry for a "fetch" request. The request path is "register.js:22", the status is 200, the response size is 394 B, and the time taken is 236 ms. The response body is a JSON object with a single key "token" containing a long string of characters.

```
your new token is eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJwYXNzd29yZCI6IjEyMzQ1NiIiLSImhdCI6MTcyMDk1NTI2NiwiZXhwIjoxNzIwOTU4ODY2fQ.8-610xKLwqpuDSIA2twYLncpnrvj08Sp5b1wMEsrAw
```

When you login it should generate the token. To ensure the token is generated check the log in the sever after pressing login.

Now on to creating the posts.
Update postCreate.js like so.

Notice how in the response header we now have to include the token in order to perform the checkauth function and validate whether the user is authenticated.

```
return (
  <div className="container">
    <h3 className="header">Create New Post</h3>
    <form onSubmit={onSubmit} className="form">
      <div className="form-group">
        <label htmlFor="user">User</label>
        <input
          type="text"
          className="form-control"
          id="user"
          value={form.user}
          disabled // Make the user field read-only
        />
      </div>
      <div className="form-group">
        <label htmlFor="content">Content</label>
        <input
          type="text"
          className="form-control"
          id="content"
          value={form.content}
          onChange={(e) => updateForm({ content: e.target.value })}
        />
      </div>
      <div className="form-group">
        <label htmlFor="image">Image</label>
        <input
          type="file"
          className="form-control"
          id="image"
          accept="image/*"
          onChange={handleImageChange}
        />
      </div>
      <div className="form-group">
        <input
          type="submit"
          value="Create Post"
          className="btn btn-primary"
        />
      </div>
    </form>
  </div>
);
```

```

frontend > src > components > postCreate.js > CreatePost
  1 import React, { useState, useEffect } from "react";
  2 import { useNavigate } from "react-router-dom";
  3 import "./App.css";
  4
  5 export default function CreatePost() {
  6   const [form, setForm] = useState({
  7     user: "",
  8     content: "",
  9     image: ""
 10   });
 11   const navigate = useNavigate();
 12
 13   // Retrieve the user from localStorage when the component mounts
 14   useEffect(() => {
 15     const savedUser = localStorage.getItem("name"); // Make sure the user is saved in localStorage as a string
 16     if (savedUser) {
 17       setForm((prev) => ({
 18         ...prev,
 19         user: savedUser,
 20       }));
 21     } else {
 22       // Redirect to login or another page IF there is no user
 23       navigate("/login"); // Redirect to login if user data is missing
 24     }
 25   }, [navigate]);
 26
 27   function updateForm(value) {
 28     return setForm((prev) => {
 29       return { ...prev, ...value };
 30     });
 31   }
 32
 33   // Function to handle image file change
 34   async function handleImageChange(e) {
 35     const file = e.target.files[0];
 36     if (file) {
 37       try {
 38         const reader = new FileReader();
 39         reader.onloadend = () => {
 40           const base64String = reader.result.split(",")[1]; // Remove the 'data:image/*;base64,' part
 41           updateForm({ image: base64String });
 42         };
 43         reader.readAsDataURL(file);
 44       } catch (error) {
 45         window.alert("Error reading image file.");
 46       }
 47     }
 48   }
 49
 50   // Function to handle form submission
 51   async function onSubmit(e) {
 52     e.preventDefault();
 53
 54     const token = localStorage.getItem("jwt");
 55
 56     const newPost = {
 57       user: form.user,
 58       content: form.content,
 59       image: form.image
 60     };
 61
 62     try {
 63       const response = await fetch("https://localhost:3001/post/upload", { // Changed to http for local development
 64         method: "POST",
 65         headers: {
 66           "Content-Type": "application/json",
 67           "Authorization": `Bearer ${token}`,
 68         },
 69         body: JSON.stringify(newPost),
 70       });
 71
 72       if (!response.ok) {
 73         throw new Error("Network response was not ok");
 74       }
 75
 76       const result = await response.json();
 77       console.log("Post created:", result);
 78       //setForm({ user: form.user, content: "", image: "" }); // Reset form but keep user
 79       //navigate("/");
 80     } catch (error) {
 81       window.alert(error);
 82     }
 83   }
 84 }

```

4 Further Development

- The user interface in this guide has been left deliberately simple to teach you the core concepts. Explore ways to enhance this by using alternate HTML, CSS and relook at Angular Material <https://react.dev/learn/describing-the-ui> [Accessed 11 July 2024].
- Use Regex to help setting and maintaining a strong password policy. https://www.w3schools.com/s/s_regex.asp [Accessed 11 July 2024].
- Clean up your debugging console.log commands or wrap these in a method in which they can be turned on or off.
- Have a look at the DOM sanitizer to sanitize (secure) your output <https://angular.io/guide/security> Have you considered the other aspects in this link? [Accessed 11 July 2024.]
- Do not re-invent the wheel! Research additional packages to assist in security e.g., Prevent brute force attacks <https://www.npmjs.com/package/express-brute> [Accessed 11 July 2024].
- General express security <https://www.npmjs.com/package/helmet> [Accessed 11 July 2024].
- Server logging <https://www.npmjs.com/package/morgan> [Accessed 11 July 2024].