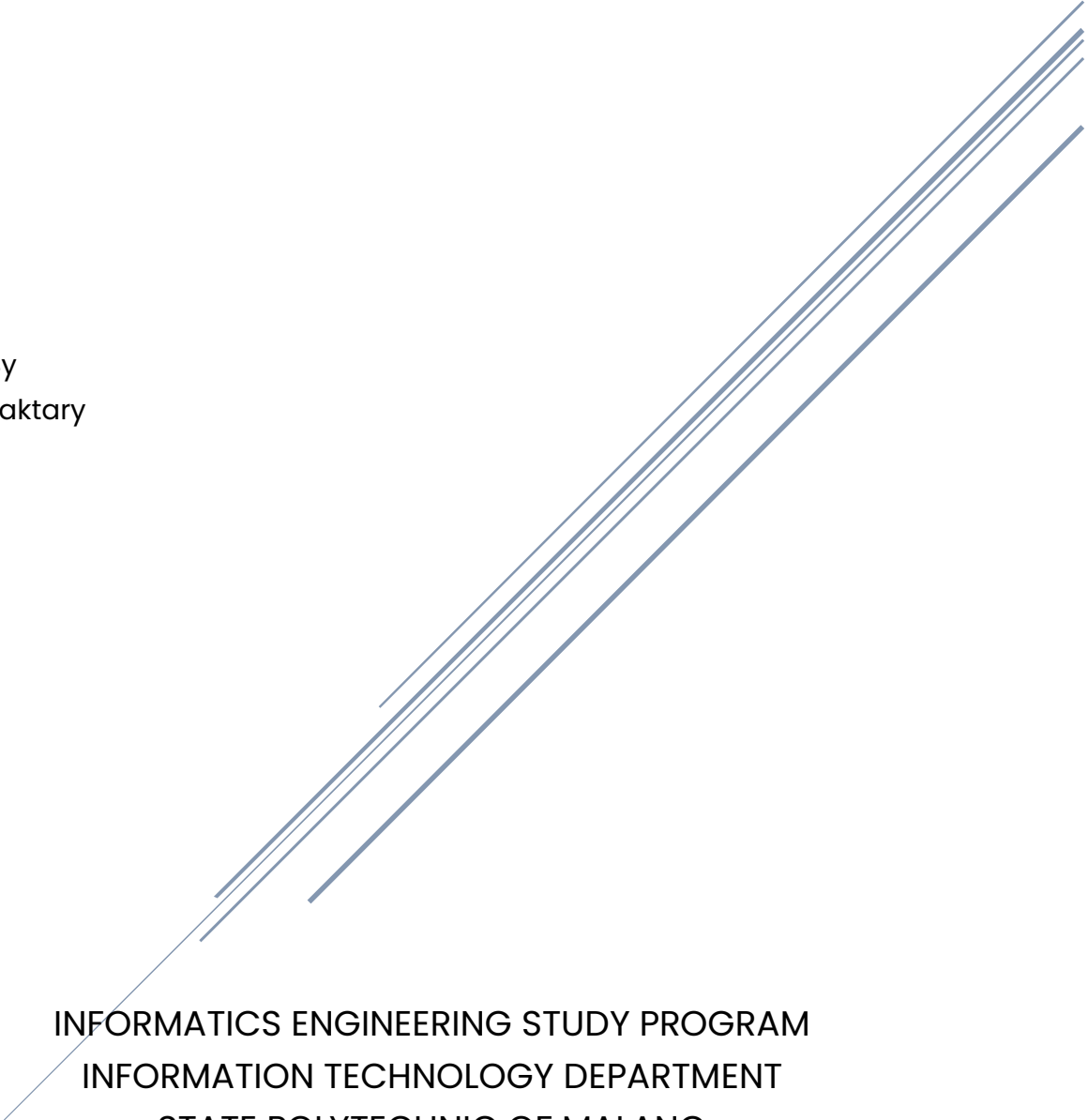


# GUIDE A01

## Introduction to NodeJS and MongoDB

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 .....	2
Software required .....	2
NPM Packages .....	2
Resource.....	3
Task Description.....	3
Installing Software.....	3
Installing Visual Studio Code .....	3
Installing NodeJS.....	6
Installing Postman.....	11
Configuring MongoDB.....	12
Creating a New Project.....	16
Start Coding.....	20
Running The API Application.....	22
Testing The API Application.....	23
Via Console and Browser.....	23
Using Postman.....	24
Running The API Test File .....	26
Creating The Web Interface.....	27
Running and Testing The Web Interface.....	30
Results .....	32

# Introduction to NodeJS and MongoDB

## Objectives

1. Students understand how to create NodeJS projects and install dependencies.
2. Students understand the way to connect MongoDB to a NodeJS application.
3. Students understand the basics of creating a web interface using NodeJS and EJS (Embedded JavaScript templates) as a view engine.

This document aims to be an introductory document for the RESTful API NodeJS web programming learning material. Students are expected to learn how to create new NodeJS projects and configure them with the necessary dependencies. Students are also expected to understand the way to create a database on MongoDB's cloud infrastructure and connect the database to the NodeJS project. Students should be able to create a web interface for the application. This web interface will contain a welcome page. Students should use the concept of lay outing to manage the web interface which will contain the welcome page and other pages.

## 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.

Note: NodeJS, Visual Studio Code, and Postman installation will be explained in later sections of this guide.

## 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.
- mongoose-slug-generator: Automatically generates slugs based on a Mongoose schema field.

## Resource

- Documents: Guide A01
- Tests: `api/testA01.test.js`, `web/testA01.test.js`, `web/images`
- Supplements: `.env.example`, `gitignore`, `main.ejs`, `main.css`

## Task Description

Students can install a new NodeJS project with the necessary dependencies and connect the application to MongoDB. Students will install the necessary software required to complete this task such as VSCode, NodeJS, and Postman. After the installation is completed, the students will create a new NodeJS project and configure it with the necessary dependencies. The project will contain a simple testing api route and a route for the welcome web interface.

## Installing Software

If VSCode, NodeJS, and Postman are installed in your operating system then skip to [Configure MongoDB](#) section.

### Installing Visual Studio Code

Visual Studio Code (VSCode) is a free and open-source code editor created by Microsoft. This software will allow for writing code and interacting with it using its integrated terminal. For this guide document, VSCode will be utilized for creating and editing NodeJS projects.

To set up VSCode, complete the installation process using the following steps:

1. Go to the official VSCode website, [link](#).
2. Download the installation file for your operating system.

Click the download button to download the software. This will download the installation file for VSCode. Choose a directory and then start downloading.

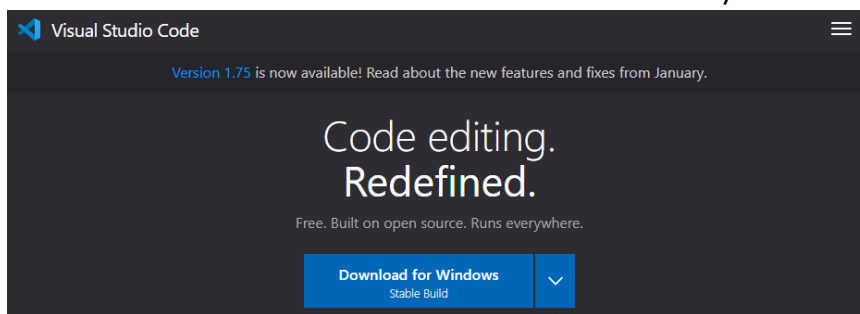


Figure 1 Visual Studio Code Official Website

3. Run the installation file.

After reading the license agreement, choose the option "I accept the agreement" to continue the installation process.

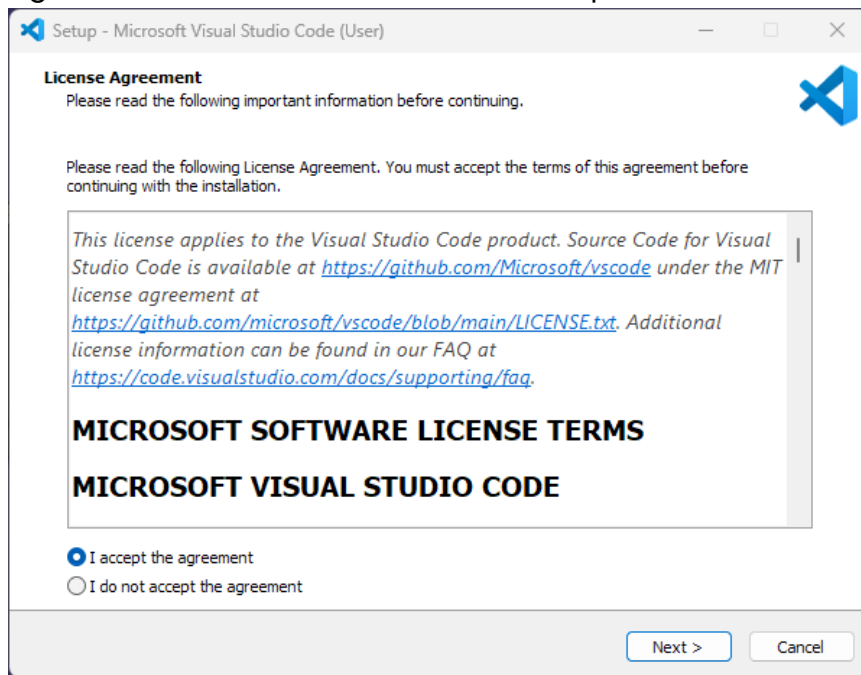


Figure 2 VSCode License Agreement

4. Follow the instructions provided by the installer to complete the installation.

Choose the option "Add to PATH" to allow for running VSCode from the command line or terminal by simply typing "code ."

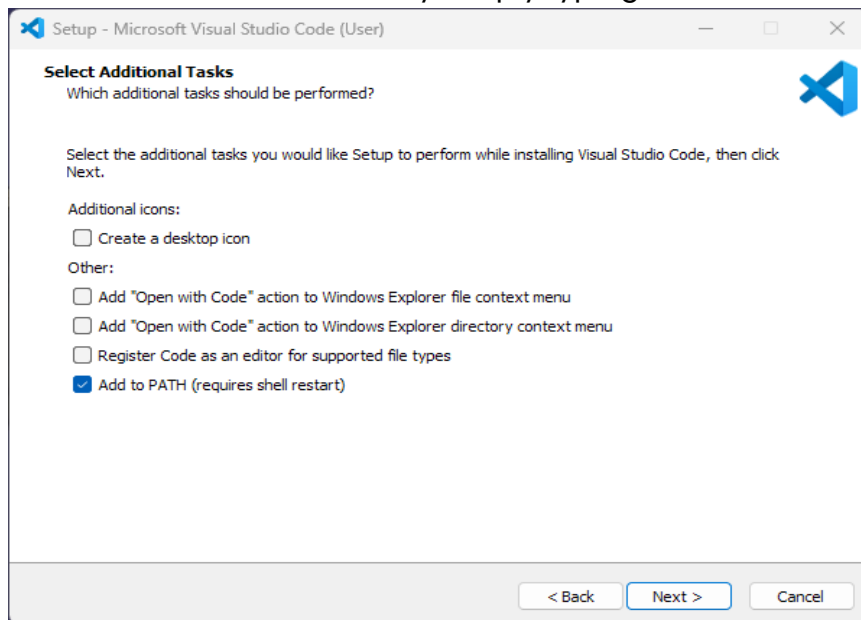


Figure 3 VSCode Installation Additional Tasks

The final setup screen is a summary of the installation. Click “install”.

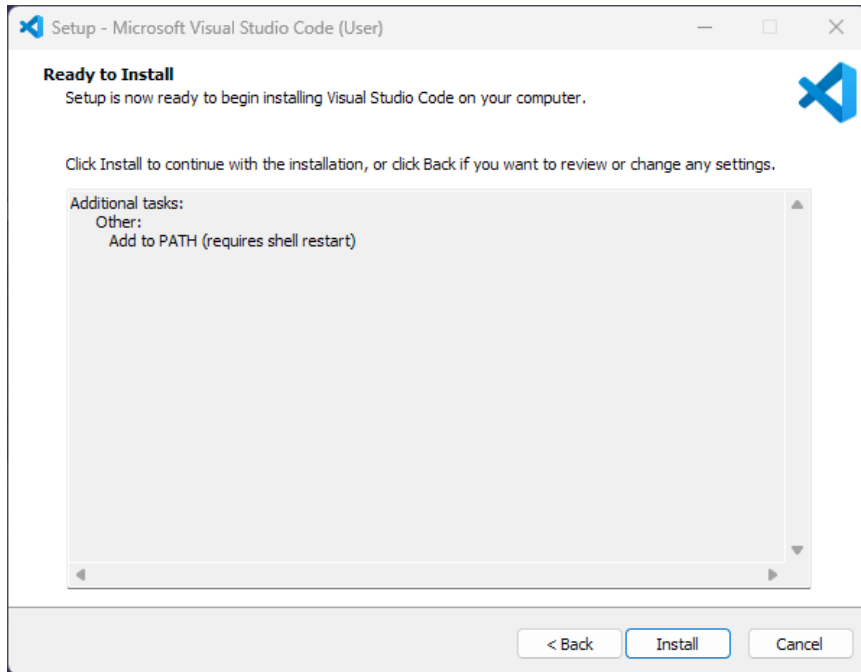


Figure 4 VSCode Installation Summary

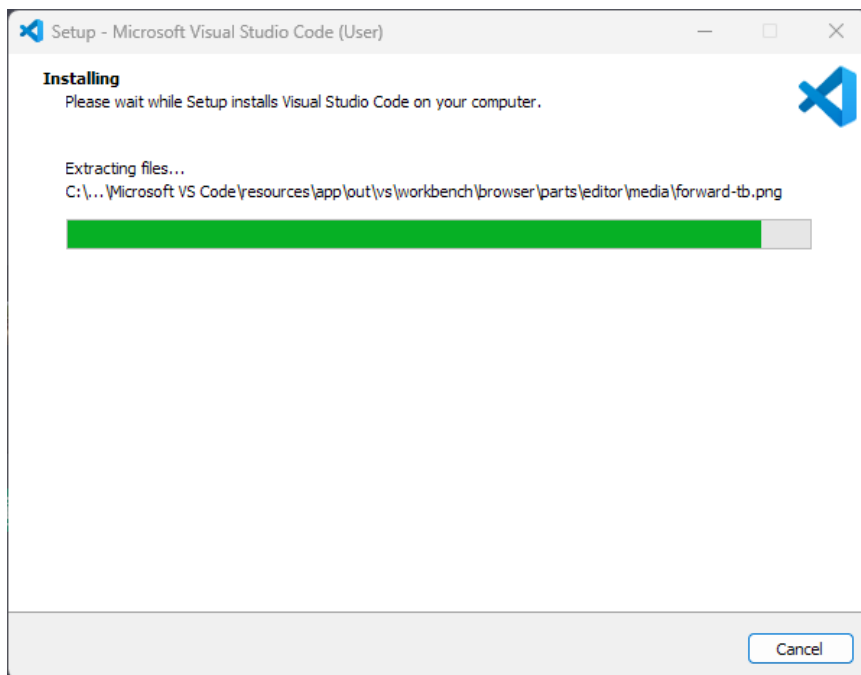



Figure 5 VSCode Installation Process

5. Once the installation is complete, open VSCode.

To open VSCode, simply find it in the application by pressing the Windows logo  on the keyboard or the taskbar. VSCode can be opened from the terminal

by typing the command "code .". Click  $\square$  + r and then type "cmd" to open the Command Prompt terminal. Once it is open, type "code ." to open VSCode. This will be used later to open the project folder.

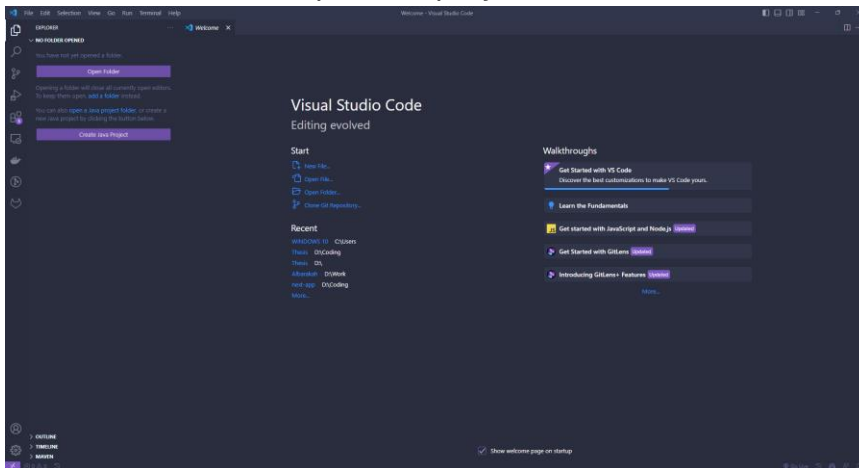


Figure 6 VSCode After Installation

## Installing NodeJS

NodeJS is a cross-platform, open-source, server-side JavaScript runtime environment that enables the execution of JavaScript code outside of a web browser. For this guide document, NodeJS will be used for a simple RESTful API application.

To set up NodeJS, students complete the installation process using the following steps:

1. Go to the official NodeJS website, [link](#).
2. Download the installation file for your operating system.

Click the download button to download the software. This will download the installation file for NodeJS. Choose LTS "Long-term support" option then choose a directory and then start downloading

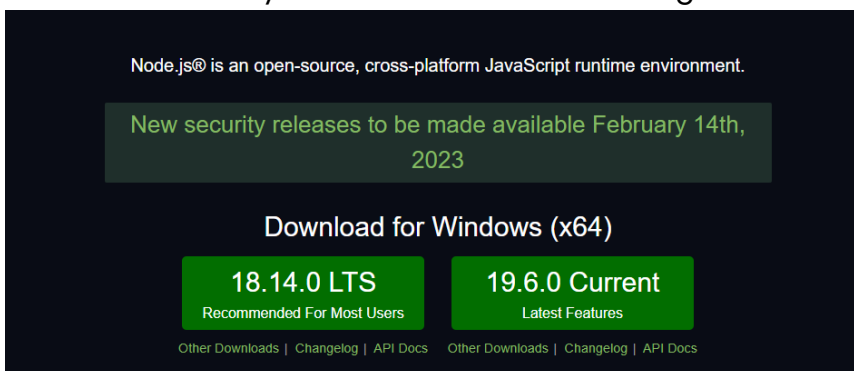
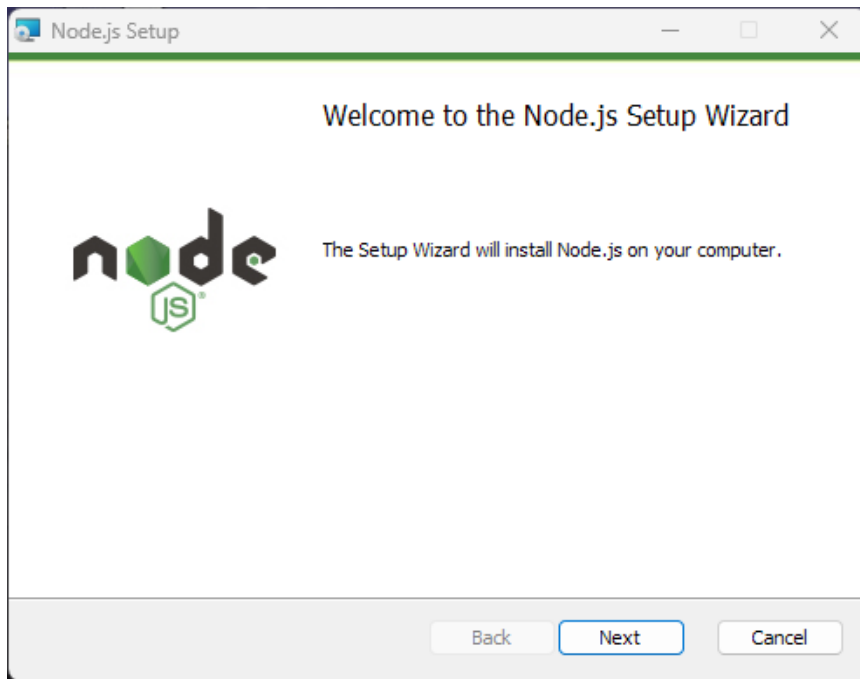


Figure 7 NodeJS Official Website



3. Run the installation file.



4. Follow the instructions provided by the installer to complete the installation process.

After reading the license agreement, choose the option "I accept the terms in the License Agreement" to continue the installation process.

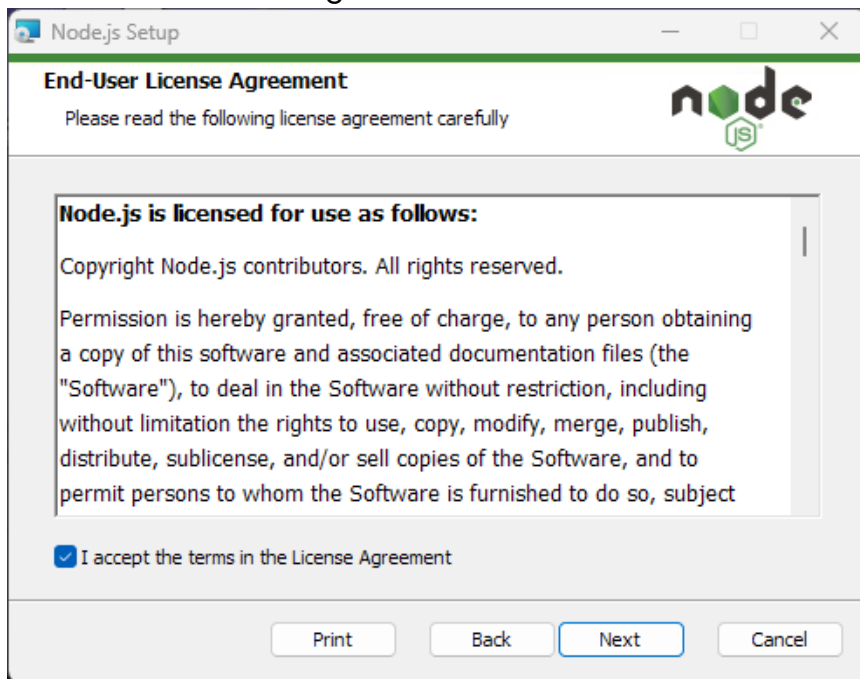


Figure 8 NodeJS Installation License Agreement

Choose a directory for the NodeJS software files, it is recommended to store them in the "C" drive. Click "Next" to continue.

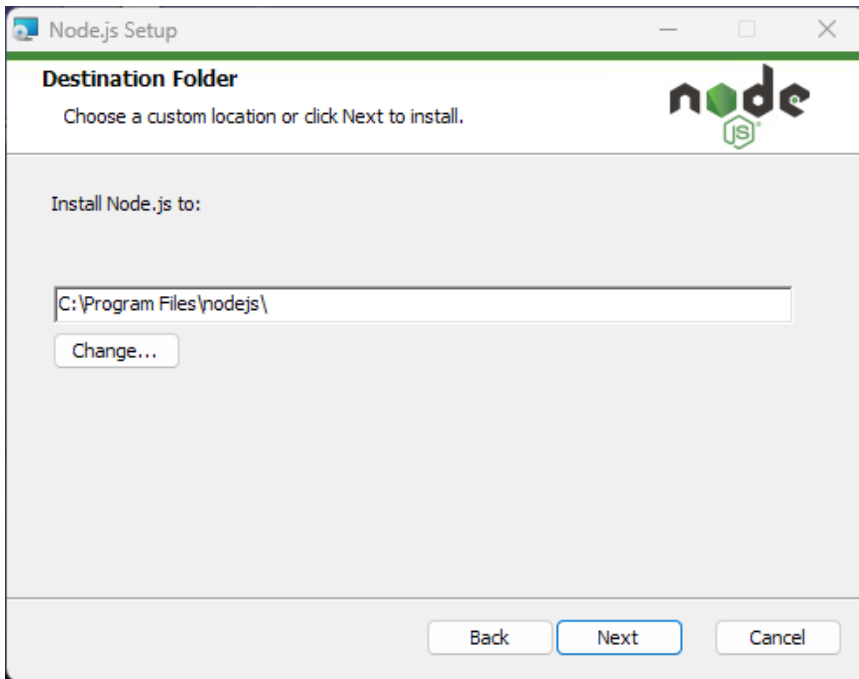


Figure 9 NodeJS Installation File Directory

This a summary of additional options for custom setup, it is recommended to continue with the default setup, click "Next" to continue.

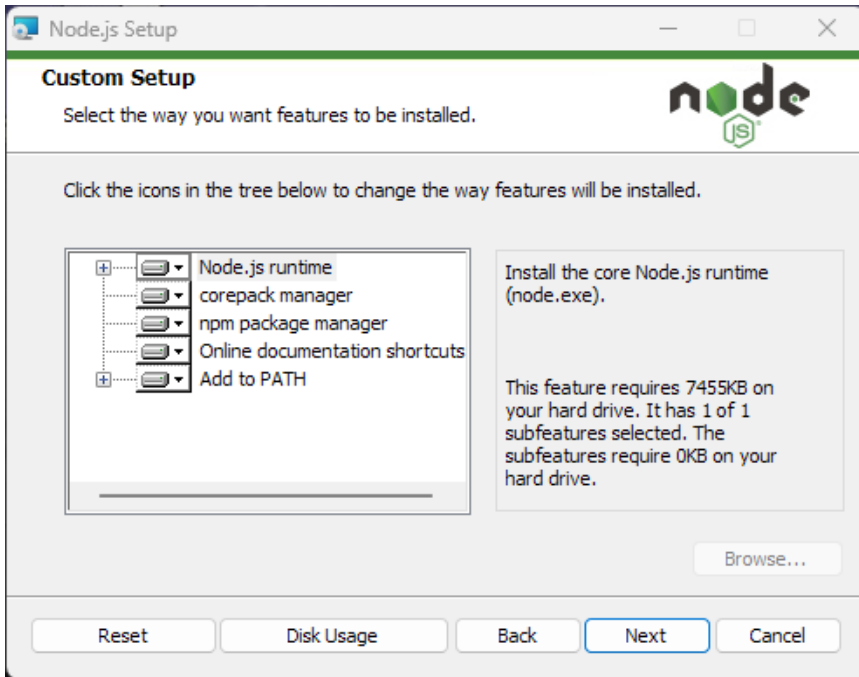


Figure 10 NodeJS Installation Additional Options

For this guide document, it is not necessary to install “Chocolatey”, leave the option empty and click “Next” to continue.

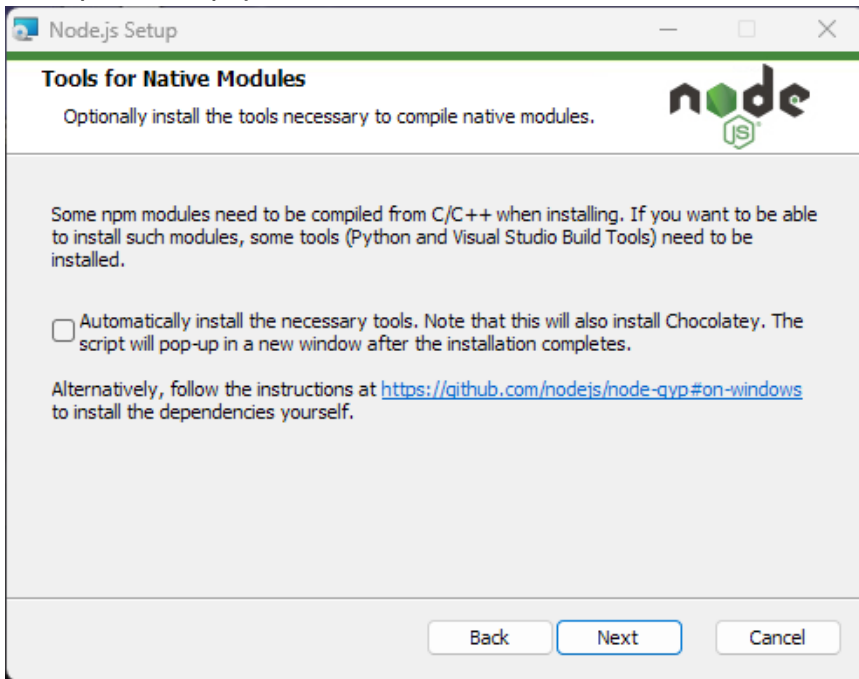


Figure 11 NodeJS Installation Additional Tools

Click “Install” to start the installation.

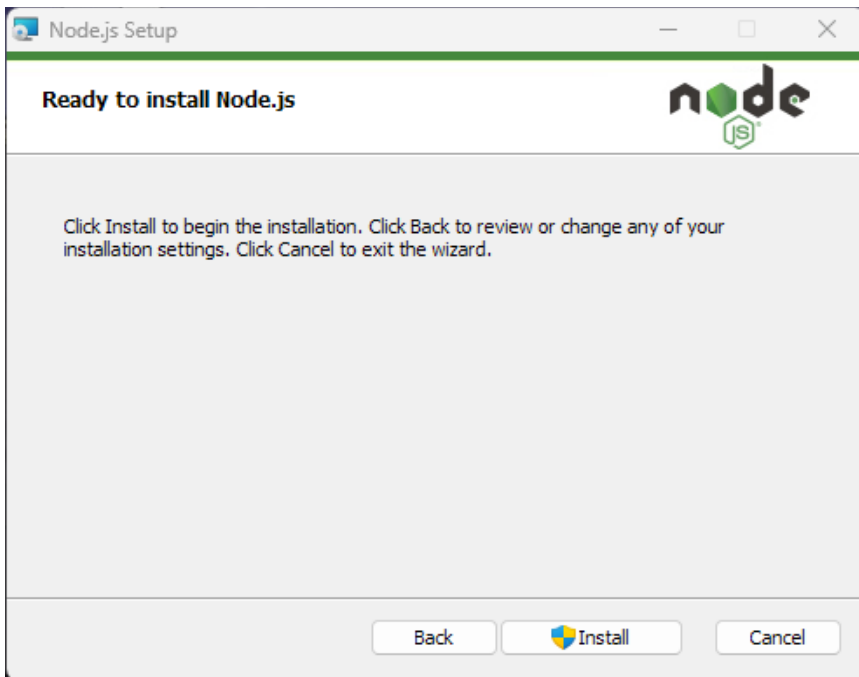


Figure 12 NodeJS Installation Final Step

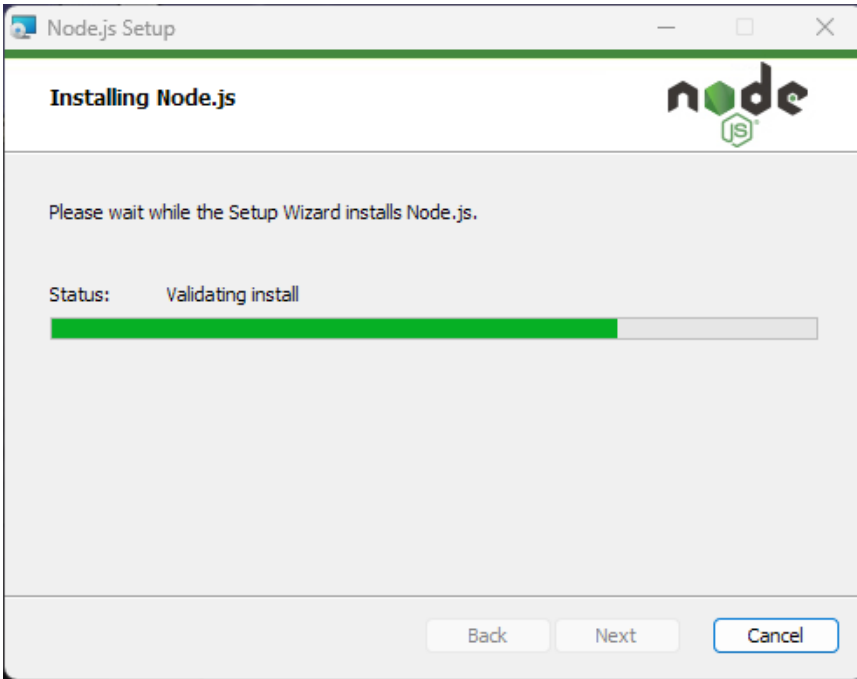


Figure 13 NodeJS Installation Process

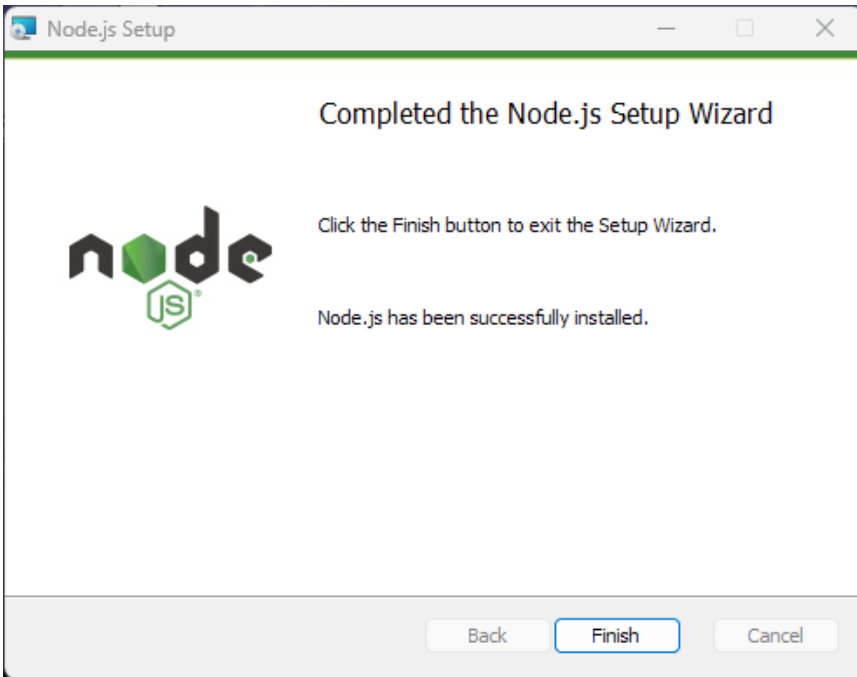


Figure 14 NodeJS Installation Completed

5. Once the installation is complete, open a command prompt or terminal window and type "node -v" to verify that NodeJS has been installed correctly.

```
Command Prompt
Microsoft Windows [Version 10.0.25284.1000]
(c) Microsoft Corporation. All rights reserved.

C:\Users\WINDOWS 10>npm -v
9.3.0

C:\Users\WINDOWS 10>node -v
v18.14.0
```

Figure 15 Verify NodeJS Installation

## Installing Postman

Postman is a popular API testing tool that simplifies the process of sending requests, debugging, and automating API development. For this guide document, Postman will be used for testing the APIs created in the NodeJS application.

To set up Postman, students need to go to the official website, download the software, and complete the installation process using the following steps:

1. Go to the official Postman website, [link](#).
2. Click on the "Download" button and choose the version for your operating system.

### Download Postman

Download the app to get started using the Postman API Platform today. Or, if you prefer a browser experience, you can try the web version of Postman.

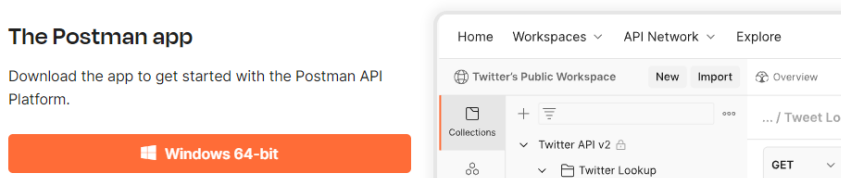


Figure 16 Postman Official Website

3. Run the installation file.
4. Follow the instructions provided by the installer to complete the installation process.

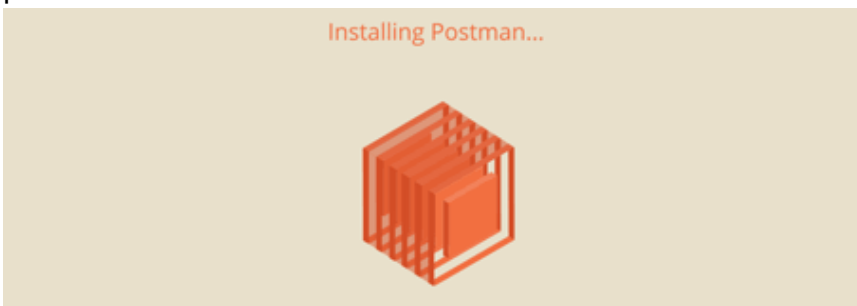


Figure 17 Postman Installation Process

5. Once the installation is complete, open Postman.

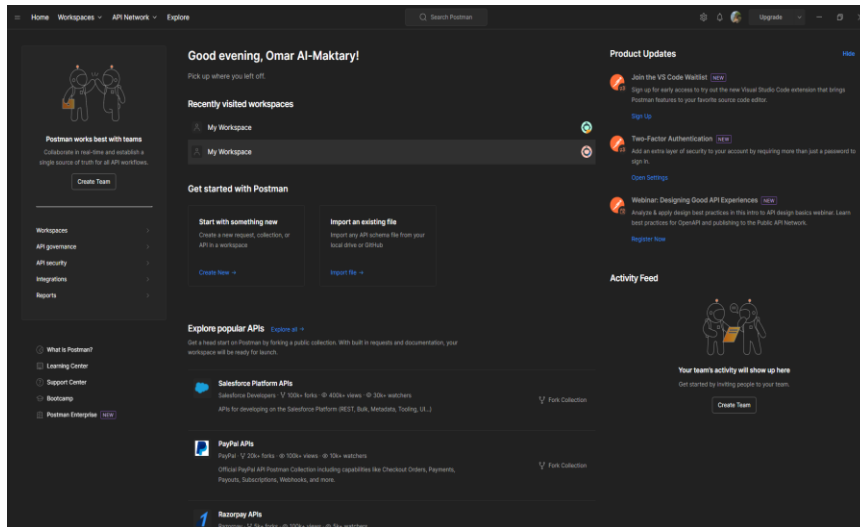


Figure 18 Postman After Installation

## Configuring MongoDB

MongoDB is a popular open-source document-oriented database that uses a flexible data model to store data in a JSON-like format. It is designed to be scalable and developer-friendly. For this guide document, MongoDB Atlas will be used as the database option.

MongoDB Atlas is a fully managed cloud-based version of MongoDB that provides automatic scaling, backup and recovery, and security features. It allows for easy deployment, operation, and scale of MongoDB databases in the cloud, and is a popular choice for many developers and organizations.

To set up a MongoDB database on MongoDB Atlas, adhere to the subsequent steps:

1. Go to MongoDB's official website, [link](#).
2. Sign up for a new account and verify the registered email.
3. After the verification, MongoDB will redirect the page to the "Welcome to Atlas!" page, fill in the questions and continue.

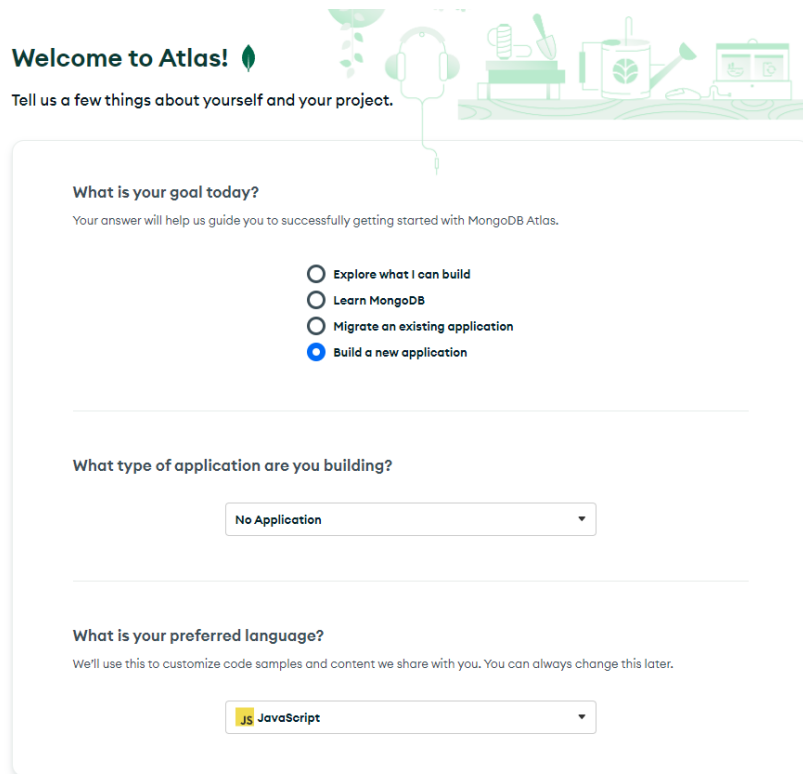


Figure 19 Welcome to MongoDB Atlas Options

4. Choose a database plan. It is recommended to choose the free option for learning purposes. Choose the “Shared” option and click “Create”.

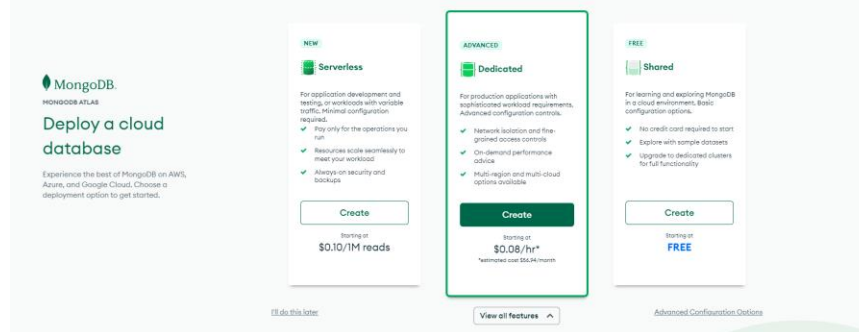


Figure 20 MongoDB Atlas Database Plan

5. Configure the shared cluster for the database. It is recommended to keep the default options and continue the process.

For the cloud provider and region choose the most suitable for the location, in this case, the cloud provider will be AWS (Amazon Web Services) and the region should be Singapore. The name of the cluster can be modified as well. Click “Create Cluster” after reviewing the configuration.

## Create a Shared Cluster

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](#).

For learning and exploring MongoDB in a sandbox environment. Basic configuration controls. No credit card required to start. Upgrade to dedicated clusters for full functionality. Explore with sample datasets. Limit of one free cluster per project.

### Cloud Provider & Region AWS, Singapore (ap-southeast-1) ^

★ Recommended region ⓘ  Dedicated tier region ⓘ

NORTH AMERICA	EUROPE	AUSTRALIA
Oregon (us-west-2) ★	Stockholm (eu-north-1) ★	Sydney (ap-southeast-2) ★
N. Virginia (us-east-1) ★	Frankfurt (eu-central-1)	<b>ASIA</b>
Ohio (us-east-2) ★ <input checked="" type="checkbox"/>	Paris (eu-west-3) ★	Seoul (ap-northeast-2) ★
N. California (us-west-1) <input checked="" type="checkbox"/>	Ireland (eu-west-1) ★	Tokyo (ap-northeast-1) ★
Montreal (ca-central-1) ★ <input checked="" type="checkbox"/>	London (eu-west-2) ★ <input checked="" type="checkbox"/>	Mumbai (ap-south-1) ★
<b>SOUTH AMERICA</b>	Milan (eu-south-1) ★ <input checked="" type="checkbox"/>	<input checked="" type="checkbox"/> Singapore (ap-southeast-1) ★
Sao Paulo (sa-east-1) ★	<b>MIDDLE EAST</b>	Hong Kong (ap-east-1) ★
	Bahrain (me-south-1) ★	Jakarta (ap-southeast-3) ★ <input checked="" type="checkbox"/>
	<b>AFRICA</b>	<input checked="" type="checkbox"/> Osaka (ap-northeast-3) ★ <input checked="" type="checkbox"/>
	Cape Town (af-south-1) ★	

Cluster Tier M0 Sandbox (Shared RAM, 512 MB Storage)

Additional Settings MongoDB 5.0, No Backup

Cluster Name Cluster0 ^

One time only: once your cluster is created, you won't be able to change its name.

Cluster names can only contain ASCII letters, numbers, and hyphens.

Figure 21 Configure MongoDB Shared Cluster



## 6. Configure the security quickstart tab.

Add a username and a password. These credentials are important to access the database from the NodeJS project. It is advised to save them in the browser or a note file. Click "Create User" and continue.

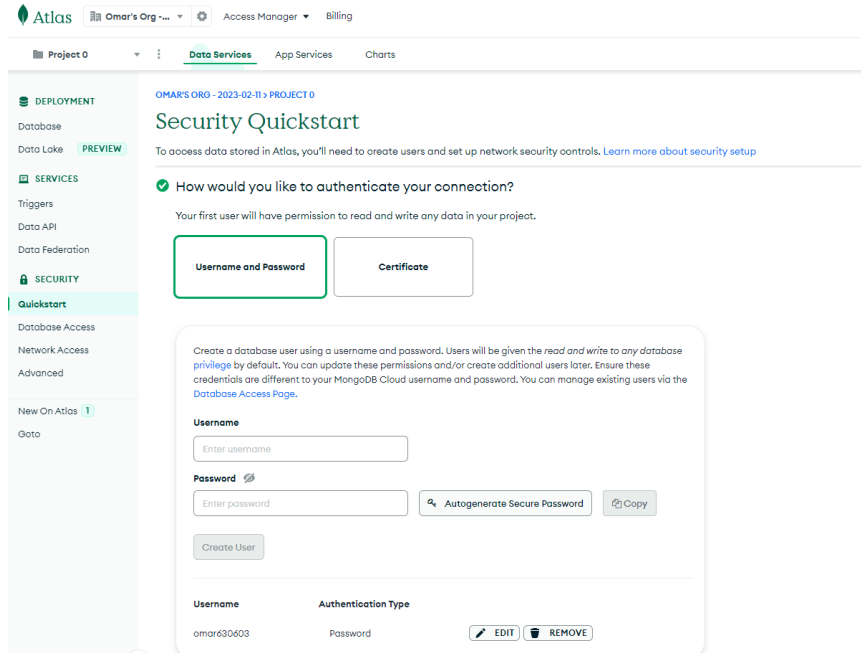


Figure 22 MongoDB Cluster Username and Password Configuration

To reach the cluster, add an IP address by selecting "Add My Current IP". When the process is finished, click "Finish and Close" to hide the Quickstart option and go to the Database. For access from any location, modify the IP address to "0.0.0.0/0". This is beneficial if the computer's IP changes due to Wi-Fi. The "Network Access" tab can be used to make changes to the IPs.

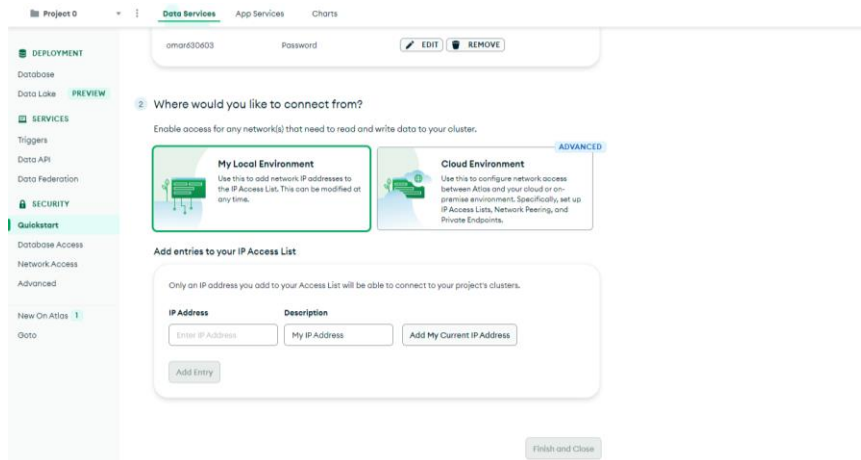


Figure 23 MongoDB IP Address Configuration

7. Wait for the configuration of the database to finish and then click on “Connect” to get the access link.

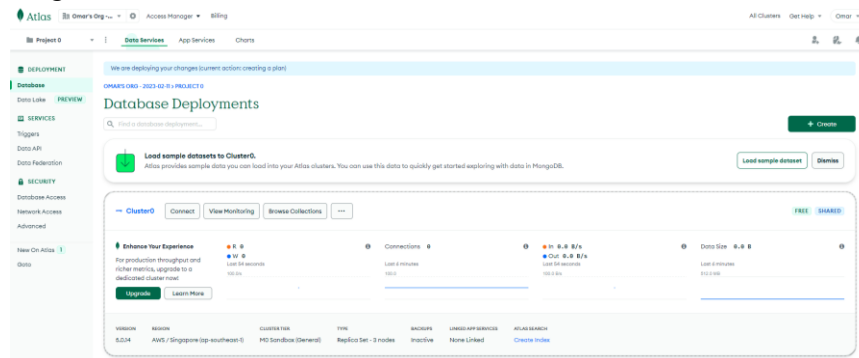


Figure 24 MongoDB Database Panel

After clicking “Connect”, choose the “Connect your application” option and then select “Node.js” as the driver and “4.1 or later” as the version. Copy the link and save it for the project. The link should resemble the following link:

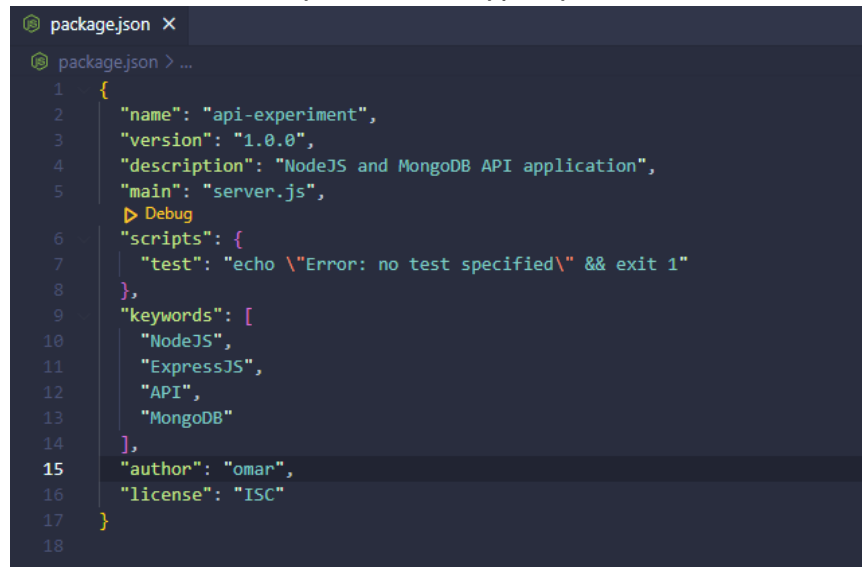
```
mongodb+srv://<username>:<password>@<cluster_name>.incnimg.mongodb.net/<database_name>?retryWrites=true&w=majority
```

## Creating a New Project

In this section, students should create a new folder named “api-experiment” and then open the folder in VSCode from the “File” tab. To Create the NodeJS project follow these steps:

1. Create a new folder named “api-experiment”.
2. Open the folder in VSCode.
3. Open the VSCode terminal from the Terminal tab by clicking “New Terminal”.
4. Type “npm init” to initialize a Node project and create a new “package.json” file. In the terminal, answer the following questions as follows:
  - a. package name: (api-experiment): click “Enter” to keep the name of the package the same as the folder name.
  - b. version: (1.0.0): click “Enter” to continue.
  - c. description: add a description for the project. Example: “NodeJS and MongoDB API application”.
  - d. entry point: (index.js): change this to “server.js” by typing it in the terminal.
  - e. test command: click “Enter” to continue.
  - f. git repository: click “Enter” to continue.

- g. keywords: add keywords for the project for example "NodeJS, ExpressJS, API, MongoDB".
- h. author: add your name
- i. license: (ISC): click "Enter" to continue.
- j. Finally, npm will generate a new "package.json" file and in the terminal, it will show a summary of the file, type "yes" to continue.



```
package.json X
package.json > ...
1  {
2    "name": "api-experiment",
3    "version": "1.0.0",
4    "description": "NodeJS and MongoDB API application",
5    "main": "server.js",
6    "scripts": {
7      "test": "echo \"Error: no test specified\" && exit 1"
8    },
9    "keywords": [
10     "NodeJS",
11     "ExpressJS",
12     "API",
13     "MongoDB"
14   ],
15   "author": "omar",
16   "license": "ISC"
17 }
18
```

Figure 25 "package.json" File Configuration

## 5. Install dependencies by following these steps:

There are dependencies and there are development dependencies. A dependency is an object that contains the library, which is required for a project's environment and functionalities. A development dependency is a dependency that is required for development purposes only.

- a. In this guide, development dependencies are:
  - i. nodemon: Reload the application after each change in the code.
  - ii. cross-env: Allows developers to set environment variables.
  - iii. jest: Executing tests.
  - iv. supertest: Send HTTP requests.
  - v. jest-expect-message: Jest with custom error messages.
  - vi. jest-image-snapshot: Adds image snapshot testing to Jest.
  - vii. puppeteer: Headless Chrome or Chromium browser.
  - viii. supertest: Makes HTTP requests to a node app.

To install these packages, type the following command:

```
npm i nodemon cross-env jest supertest jest-expect-message jest-image-snapshot puppeteer supertest --save-dev
```

“npm”: a package manager for installing the above dependencies.

“i”: is short for install.

“--save”: to write the dependencies in the “package.json” file.

“--dev”: to specify that those are development dependencies.

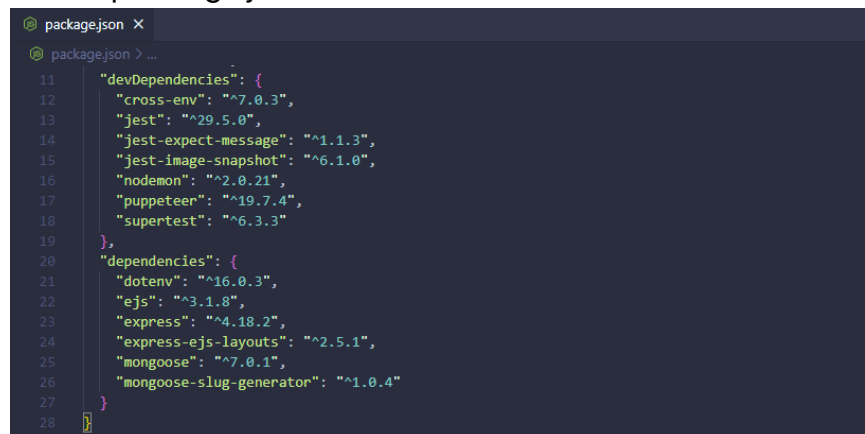
- b. In this guide, production dependencies are:
- i. dotenv: Manage environment variables.
  - ii. express: Node lightweight framework.
  - iii. ejs: Embedded JavaScript templating.
  - iv. express-ejs-layouts: Layout support for EJS in Express.
  - v. mongoose: MongoDB object modeling library for NodeJS.
  - vi. mongoose-slug-generator: Generates slugs based on a Mongoose schema field

To install these packages, type the following command:

```
npm i dotenv express ejs express-ejs-layouts mongoose-slug-generator mongoose --save
```

- c. Once the installation is completed, there will be a folder name “node\_modules” which will contain all the dependencies files and a file named “package-lock.json” which will keep information about the dependencies like version and repositories. For this experiment, students must not change those files.

It should be noted that the names of the dependencies are listed in the “package.json” file.



```
11  "devDependencies": {
12    "cross-env": "^7.0.3",
13    "jest": "^29.5.0",
14    "jest-expect-message": "^1.1.3",
15    "jest-image-snapshot": "^6.1.0",
16    "nodemon": "^2.0.21",
17    "puppeteer": "^19.7.4",
18    "supertest": "^6.3.3"
19  },
20  "dependencies": {
21    "dotenv": "^16.0.3",
22    "ejs": "^3.1.8",
23    "express": "^4.18.2",
24    "express-ejs-layouts": "^2.5.1",
25    "mongoose": "^7.0.1",
26    "mongoose-slug-generator": "^1.0.4"
27  }
28 }
```

Figure 26 Dependencies Installation

6. Copy the supplement “.env.example” and “gitignore” files to the base directory of the project.

The file “gitignore” is used to prevent specific files or folders, like “node\_modules”, from being committed when using version control like git. This file is helpful to not push huge files and secret files such as “.env” which contains values like the URL to the database.

7. Create a new file named “.env”
8. Copy the content of the file “.env.example” to the file “.env”
9. Change the values of the variables inside “.env” to the correct values according to the instructions in the “.env.example” file.

The variables in the “.env” file are:

- a. MONGODB\_URI: Change the <username>, <password>, and <cluster\_name> according to the link provided by MongoDB Atlas in the [previous step](#). Change the <database\_name> to “api-experiment”
- b. MONGODB\_URI\_TEST: Fill in with the same value of the previous variable but the <database\_name-test> should be “api-experiment-test”. This will ensure tests will run in a different database.
- c. PORT: Use port 8080.

The file structure of the project should look like this:

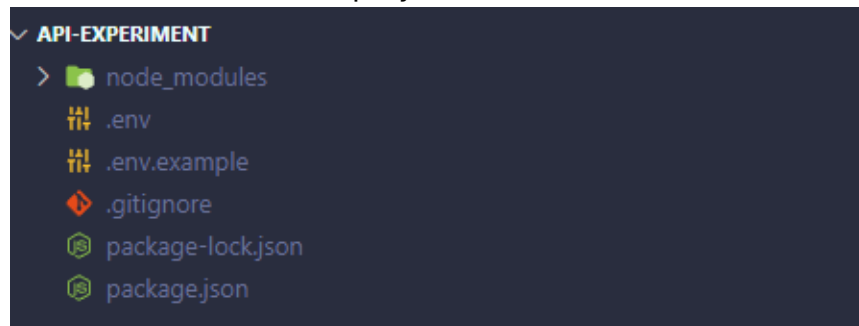


Figure 27 File Structure

## Start Coding

1. Create two files "app.js" and "server.js" in the base directory of the project.
  - a. "app.js" file: It sets up the Express application, defines the routes and middleware, and starts the application's service.
  - b. "server.js" file: Typically, the main entry point of the application. This file uses the app.js file to run the application based on the predefined requirements.
2. In the "app.js" file add the following code:

```
const express = require("express");
const app = express();
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.get("/api/v1/test", (req, res) => {
  if (req.body.message) {
    res.status(200).json({ message: req.body.message });
  } else {
    res.status(200).json({ alive: "True" });
  }
});
module.exports = app;
```

Figure 28 "app.js" File Code

### Code Explanation:

- a. The code creates a web server using the ExpressJS library by creating an instance of the library in constant named "app".
- b. It sets up a middleware function to parse incoming JSON data.
- c. The extended: false option means that the parser will only handle key-value string or array data and not any other data types in the request body.
- d. A route is set up for the root URL path ("/api/v1/test").
- e. If the incoming request has a "message" property in its body, the server responds with a JSON object containing that message and a status code of 200.
- f. When a GET request is made to this route with no request body with property "message", the server responds with a JSON object containing "alive": "True".
- g. The application instance is exported for use in other parts of the codebase.

3. In the "server.js" file add the following code:

```
const mongoose = require("mongoose");
const app = require("./app");
require("dotenv").config();
const PORT = process.env.PORT || 5000;
mongoose
  .connect(process.env.MONGODB_URI)
  .then(() => {
    app.listen(PORT, console.log(`Server started on port ${PORT}`));
  })
  .catch((error) => {
    console.log(error);
  });
```

Figure 29 "server.js" File Code

#### Code Explanation:

- a. The Mongoose library is imported in the first line of code.
  - b. The second line of code imports and saves the instance of the Express.js application from the "./app" file.
  - c. The third line of code uses the dotenv library to load environment variables from the ".env" file.
  - d. The server's port number is configured in the fourth line of code, either using an environment variable or a default value of 5000.
  - e. The Mongoose library is used in the next few lines to connect to a MongoDB database. The connection string for the database is stored in the environment variable "MONGODB\_URI".
  - f. If the connection is successful, the web server begins listening on the designated port, and a confirmation message will be in the console.
  - g. The error is recorded in the console if it happens when connecting to the database.
4. Finally, update the "package.json" file by changing the value of the scripts to be as the following value and add the jest configuration to make sure that the tests run as intended:

```

"scripts": {
  "start": "node server.js",
  "dev": "nodemon server.js",
  "api-testA01": "cross-env NODE_ENV=test jest -i tests/api/testA01.test.js --
testTimeout=20000",
  "web-testA01": "cross-env NODE_ENV=test jest -i tests/web/testA01.test.js --
testTimeout=20000",
  "api-testA02": "cross-env NODE_ENV=test jest -i tests/api/testA02.test.js --
testTimeout=20000",
  "web-testA02": "cross-env NODE_ENV=test jest -i tests/web/testA02.test.js --
testTimeout=20000",
  "api-testA03": "cross-env NODE_ENV=test jest -i tests/api/testA03.test.js --
testTimeout=20000",
  "web-testA03": "cross-env NODE_ENV=test jest -i tests/web/testA03.test.js --
testTimeout=20000",
  "api-testA04": "cross-env NODE_ENV=test jest -i tests/api/testA04.test.js --
testTimeout=20000",
  "web-testA04": "cross-env NODE_ENV=test jest -i tests/web/testA04.test.js --
testTimeout=20000",
  "api-testA05": "cross-env NODE_ENV=test jest -i tests/api/testA05.test.js --
testTimeout=20000",
  "web-testA05": "cross-env NODE_ENV=test jest -i tests/web/testA05.test.js --
testTimeout=20000",
  "testAA": "cross-env NODE_ENV=test jest --testTimeout=40000 --silent --
detectOpenHandles"
},
"jest": {
  "setupFilesAfterEnv": [
    "jest-expect-message"
  ],
  "noStackTrace": true,
  "silent": false
},
}

```

Figure 30 the "package.json" File Scripts

## Running The API Application

To run the application, use the following command:

For production purposes, the command "npm run start" should be used in the terminal to execute the code "node server.js" which will run the "server.js" file using NodeJS to start the application as shown in the scripts added in the "package.json" file above.



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.

### Via Console and Browser

After running the application, the console should show a message stating that the application is running on port 8080 similar to the following:

```
PS C:\Users\WINDOWS 10\Desktop\api-experiment> npm run dev
> api-experiment@1.0.0 dev
> nodemon server.js

[nodemon] 2.0.20
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node server.js`
(node:27572) [MONGODB] DeprecationWarning: Mongoose: the `strictQuery` option will be switched back to `false` by default in Mongoose 7. Use `mongoose.set('strictQuery', false);` if you want to prepare for this change. Or use `mongoose.set('strictQuery', true);` to suppress this warning.
(Use `node --trace-deprecation ...` to show where the warning was created)
"Server started on port 8080"
```

Figure 31 Console Message After Running The Application

There is a warning about mongoose's strict query mode, here is an explanation for the warning message:

When using Mongoose, the strict mode for query conditions may be disabled with `mongoose.set("strictQuery", false)`. By default, Mongoose applies strict mode to query conditions, which restricts the use of fields in queries to those that are declared in the schema. Mongoose will issue a `StrictModeError` if a query condition contains a field that is not specified in the schema. When strict mode for query conditions is set to false using `mongoose.set("strictQuery", false)`, Mongoose will not enforce it, enabling the usage of fields that are not declared in the schema. This is a risk nevertheless since it can allow searches to accidentally match or alter more documents than they should. As a result, it is typically advised to leave strict mode on and specify every field in the schema.

To disable this warning add the following code `"mongoose.set('strictQuery', true);"` to the `"server.js"` file after the first line.

In the browser, open the following link <http://localhost:8080/api/v1/test>. It should display the following message `"{"alive": "True"}"`. Similar to the following figure:

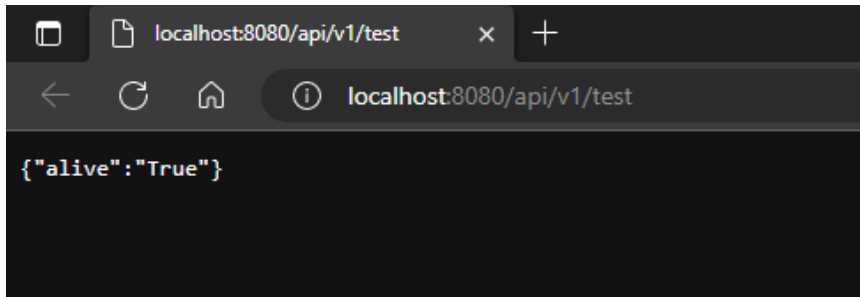


Figure 32 Browser After Running The Application

## Using Postman

To test results from this guide and the next guides on Postman, follow these steps:

1. Open a workspace or create a new one from the Workspaces tab in the top navbar.
2. Create a new environment from the Environments tab on the left sidebar and fill in the following values:

	VARIABLE	TYPE ⓘ	INITIAL VALUE ⓘ	CURRENT VALUE
<input checked="" type="checkbox"/>	protocol	default ▾	http://	http://
<input checked="" type="checkbox"/>	host	default ▾	localhost	localhost
<input checked="" type="checkbox"/>	port	default ▾	:8080	:8080
<input checked="" type="checkbox"/>	version	default ▾	/api/v1	/api/v1

Figure 33 Postman Environment Values

3. Create a new collection from the Collections tab.
4. In the created collection, create a folder named `"api-experiment"`.
5. In the created folder create a GET request with the name `"GET /api/v1/test"`.
6. 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 GET request as the following:  
`"{{protocol}}{{host}}{{port}}{{version}}/test"`



Figure 34 Postman Request Configuration Bar

7. Click “Send” and wait for the response. Postman should show results as the following if everything is working correctly:

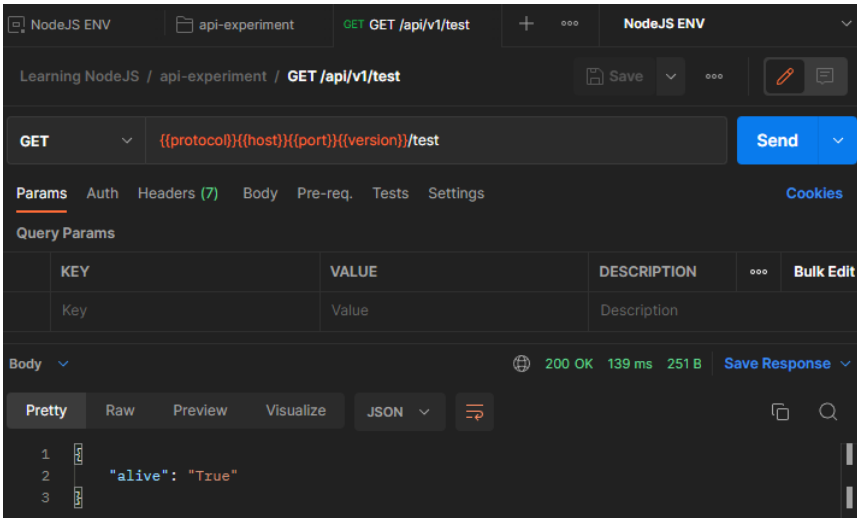


Figure 35 Postman Request Results

Notice that the results are in JSON format because the express JSON functionality is used in the application.

8. Test the request body by adding the property “message” in the x-www-form-urlencoded tab in the body section. This should show the message sent:

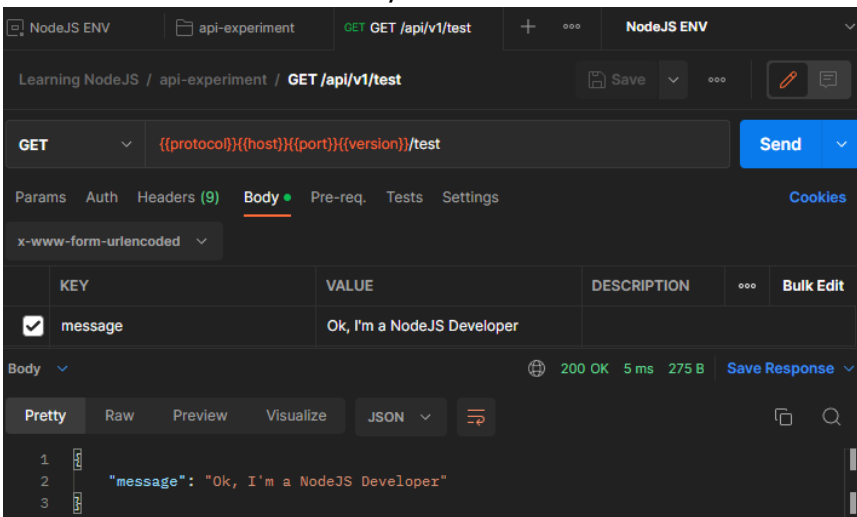


Figure 36 Postman Request Results With “message” Property

## 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. Make a folder called "tests" in the main directory. Inside this folder, create two folders called "api" and "web". Copy the file "testA01.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-testA01" 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 ~/Desktop/api-experiment
$ npm run api-testA01

> api-experiment@1.0.0 api-testA01
> cross-env NODE_ENV=test jest -i tests/api/testA01.test.js --testTimeout=20000

console.log
  Database connected successfully

    at log (tests/api/testA01.test.js:18:15)

PASS tests/api/testA01.test.js
  Testing application configuration
    ✓ Should have the necessary development packages (10 ms)
    ✓ should have the necessary production packages (5 ms)
    ✓ should have the right name and packages (1 ms)
    ✓ should have the right environment variables (5 ms)
    ✓ should have the right database connection (1 ms)
    ✓ should be using json format and express framework (2 ms)
  Testing GET /api/v1/test
    ✓ should return alive (75 ms)
    ✓ should return the same message (39 ms)

Test Suites: 1 passed, 1 total
Tests:      8 passed, 8 total
Snapshots:  0 total
Time:       3.449 s, estimated 6 s
Ran all test suites matching /tests\\api\\testA01.test.js/i.
```

Figure 37 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:

```
OMAR@LAPTOP-MHSUC5AB MINGW64 ~/Desktop/api-experiment
$ npm run api-testA01

> api-experiment@1.0.0 api-testA01
> cross-env NODE_ENV=test jest -i tests/api/testA01.test.js --testTimeout=20000

console.log
Database connected successfully

    at log (tests/api/testA01.test.js:18:15)

FAIL tests/api/testA01.test.js
  Testing application configuration
    ✓ Should have the necessary development packages (11 ms)
    ✓ should have the necessary production packages (5 ms)
    ✓ should have the right name and packages (2 ms)
    ✓ should have the right environment variables (5 ms)
    ✓ should have the right database connection (1 ms)
    ✓ should be using json format and express framework (4 ms)
  Testing GET /api/v1/test
    ✓ should return alive (52 ms)
    ✗ should return the same message (43 ms)

  ● Testing GET /api/v1/test > should return the same message

    The status code should be 200, but it is "201", change the status code in the function that handles the GET /api/v1/test route

    expect(received).toBe(expected) // Object.is equality

    Expected: 200
    Received: 201

Test Suites: 1 failed, 1 total
Tests:       1 failed, 7 passed, 8 total
Snapshots:  0 total
Time:        4.254 s, estimated 9 s
Ran all test suites matching /tests\\api\\testA01.test.js/i.
```

Figure 38 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 this application will be created. The same basic endpoint explained in the previous sections will be implemented in a web page that can receive a message attribute through the app URL and display it on the index page.

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

1. Update the file "app.js". Add the following code to the file to set up a view engine using EJS and configure the application page layout directory.

Add the following lines to the first lines of the page. These lines import the libraries for the view engine and lay-outing.

```
const path = require("path");
const ejsLayouts = require("express-ejs-layouts");
```

Figure 39 "app.js" Updates for The Web Interface 1

Add the following lines after this line "app.use(express.urlencoded({ extended: false }));":

These lines are for configuring the application's view engine and specifying the directory of the view file and the layout file which will be used for rendering the index page for the application. The function is used to respond to any request for the entry point of the application.

```

app.use(express.static(path.join(__dirname, "web")));
app.use(ejsLayouts);
app.set("view engine", "ejs");
app.set("views", path.join(__dirname, "web", "views"));
app.set("layout", path.join(__dirname, "web", "layouts", "main"));
app.get("/", (req, res) => {
  if (req.query.message) {
    return res.render("index", {
      title: "API-Experiment | Home",
      message: req.query.message,
    });
  } else {
    return res.render("index", {
      title: "API-Experiment | Home",
    });
  }
});

```

Figure 40 "app.js" Updates for The Web Interface 2

Add the following lines at the end of the application before the last line in which the app instance is exported by the model object.

These lines are used to respond to any request which has not been listed in the application as a URL. This function will render the error page with a status of 404 indicating that the page or the request was not found.

The function contains the statement "render". This function takes 2 parameters which are the name of the view file and the data that will be sent to it. These data later will be converted to an HTML file which can be viewed through a web browser.

Notice the data "title". This data will be used in the layout file to display the title of the web page on the page's tab in the browser.

```

app.use((req, res, next) => {
  const error = { status: 404, message: "NOT FOUND" };
  return res.render("error", { title: "API-Experiment | Error", error });
});

```

Figure 41 "app.js" Updates for The Web Interface 3

2. Create a folder named "web" in the base directory. This folder will contain three folders namely "layouts", "styles", and "views".
3. Copy the file "main.ejs" to the newly created folder "/web/layouts" and copy the file "main.css" to the folder "/web/styles".

4. Create two files namely "index.ejs" and "error.ejs" in the "/web/views" folder.
5. Copy the following code to the "index.ejs" file:

```
<div class="container">
  <h1 class="title">Welcome to my beautiful web page!</h1>
  <% if (typeof message !== 'undefined') { %>
    <div class="alert alert-success">
      <p class="message"><%= message %></p>
    </div>
  <% } else { %>
    <div class="alert alert-warning">
      <p class="message">
        Type a value for the property message next to this page URL
        <br />
        /?message=Hello
      </p>
    </div>
  <% } %>
</div>
<div class="container flex-col-container">
  <h2 class="subtitle">I can show products from my database in MongoDB</h2>
  <p class="description">Click on this button below to go the products page</p>
  <a href="/products" class="btn btn-primary">Products</a>
</div>
```

Figure 42 "/web/views/index.ejs" File Structure

Note that the use of plain JavaScript in an HTML file is possible thanks to the power of EJS which allows for more dynamic applications with less code used.

6. Copy the following code to the "error.ejs" file:

```
<% if (typeof error !== 'undefined') { %>
<div class="alert alert-error">
  <h1 class="title"><%= error.status %></h1>
  <p class="message"><%= error.message %></p>
</div>
<% } %>
```

Figure 43 "/web/views/error.ejs" File Structure

This code will be the template for error messages in the application. Mostly the error of "Not Found" can happen if a request is made to an unknown route.

## 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.

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

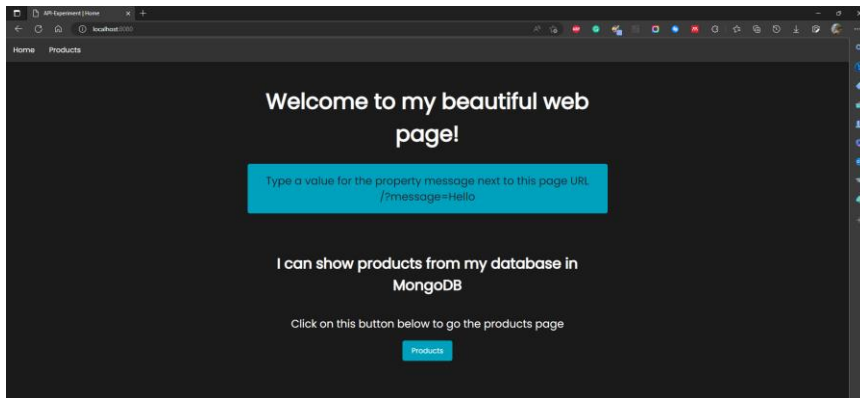


Figure 44 The Welcome Page

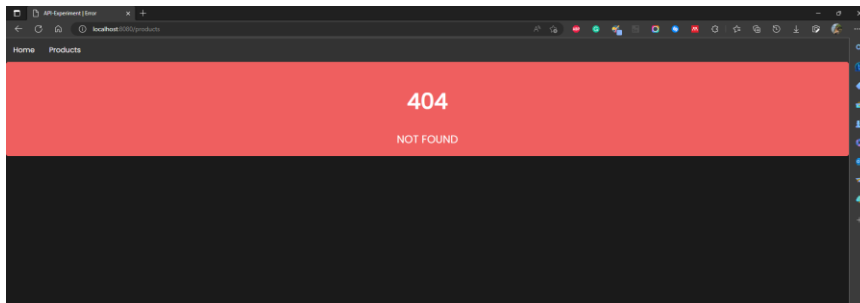


Figure 45 The Error Page

To test the application, copy the file “testA01.test.js” and the folder “images” from the “/tests/web” folder and paste it to the folder “/tests/web” in your project directory. After that, run the command “npm run web-testA01” and notice the results.

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



```

OMAR@LAPTOP-N1SUC5AB MINGW64 ~/Desktop/api-experiment
$ npm run web-testA01

> api-experiment@1.0.0 web-testA01
> cross-env NODE_ENV=test jest -i tests/web/testA01.test.js --testTimeout=20000

PASS tests/web/testA01.test.js (5.149 s)
  Testing the index page title and content
    ✓ should have the right title (809 ms)
    ✓ should have a button with the text 'Products' and url '/products' (56 ms)
    ✓ should have nav bar with 2 links (29 ms)
  Testing the index page for receiving messages
    ✓ should receive a message and display it (81 ms)
    ✓ should have the correct color for the box after receiving a message (38 ms)
  Testing the error 'Not Found' page
    ✓ should have the right title (40 ms)
    ✓ should have a status code of 404 (48 ms)
    ✓ should have a message saying 'NOT FOUND' (48 ms)
  Testing the index page and error 'Not Found' page image snapshots
    ✓ matches the expected styling (392 ms)
    ✓ matches the expected styling (507 ms)

Test Suites: 1 passed, 1 total
Tests:      10 passed, 10 total
Snapshots: 2 passed, 2 total
Time:      5.368 s, estimated 9 s
Ran all test suites matching /tests/web/testA01.test.js/i.

```

Figure 46 Successful Web Test Results

```

OMAR@LAPTOP-N1SUC5AB MINGW64 ~/Desktop/api-experiment
$ npm run web-testA01

> api-experiment@1.0.0 web-testA01
> cross-env NODE_ENV=test jest -i tests/web/testA01.test.js --testTimeout=20000

FAIL tests/web/testA01.test.js (9.834 s)
  Testing the index page title and content
    ✓ should have the right title (2878 ms)
    ✓ should have a button with the text 'Products' and url '/products' (102 ms)
    ✓ should have nav bar with 2 links (57 ms)
  Testing the index page for receiving messages
    ✓ should receive a message and display it (188 ms)
    ✓ should have the correct color for the box after receiving a message (67 ms)
  Testing the error 'Not Found' page
    ✓ should have the right title (63 ms)
    ✓ should have a status code of 404 (78 ms)
    ✓ should have a message saying 'NOT FOUND' (75 ms)
  Testing the index page and error 'Not Found' page image snapshots
    ✗ matches the expected styling (963 ms)
    ✓ matches the expected styling (637 ms)

  ● Testing the index page and error 'Not Found' page image snapshots › matches the expected styling

    The web styling for the index page is not correct check the file "tests/web/images/__diff_output__/index-page-diff.png" to find the difference
    Expected image to match or be a close match to snapshot but was 0.20815531412760416% different from snapshot (1637 differing pixels).
    See diff for details: tests/web/images/__diff_output__/index-page-diff.png

    > 1 snapshot failed.
  Snapshot Summary
  > 1 snapshot failed from 1 test suite. Inspect your code changes or run 'npm run web-testA01 -- -u' to update them.

Test Suites: 1 failed, 1 total
Tests:      1 failed, 9 passed, 10 total
Snapshots: 1 failed, 1 passed, 2 total
Time:      9.822 s, estimated 11 s
Ran all test suites matching /tests/web/testA01.test.js/i.

```

Figure 47 Failed Web Test Results

The terminal displays feedback that compares your project screenshot to the reference image. Differences between the two images are located in the `"/test/web/images/__diff_output__"` folder. If there are no differences, the images will disappear. The different images are useful for identifying UI differences. The reference image is used as a guide for how the webpage should look.

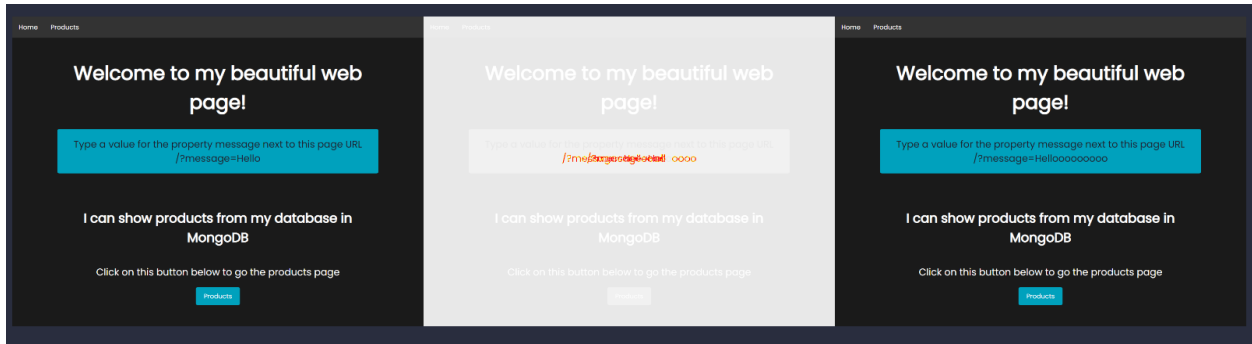


Figure 48 " \_\_diff\_output\_\_ " Image Example

The image on the left is the reference image and the one on the right is the project's current image. The middle image shows the differences in red color indicating that the difference is in the message area.

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 first meeting, where students will be introduced to various software tools and technologies. Specifically, students will learn how to install Visual Studio Code, NodeJS, and Postman, as well as how to create and configure a MongoDB Atlas Cluster. Additionally, students will gain an understanding of how to create a NodeJS project and develop a basic API endpoint with a web interface.