

# Chapter 3

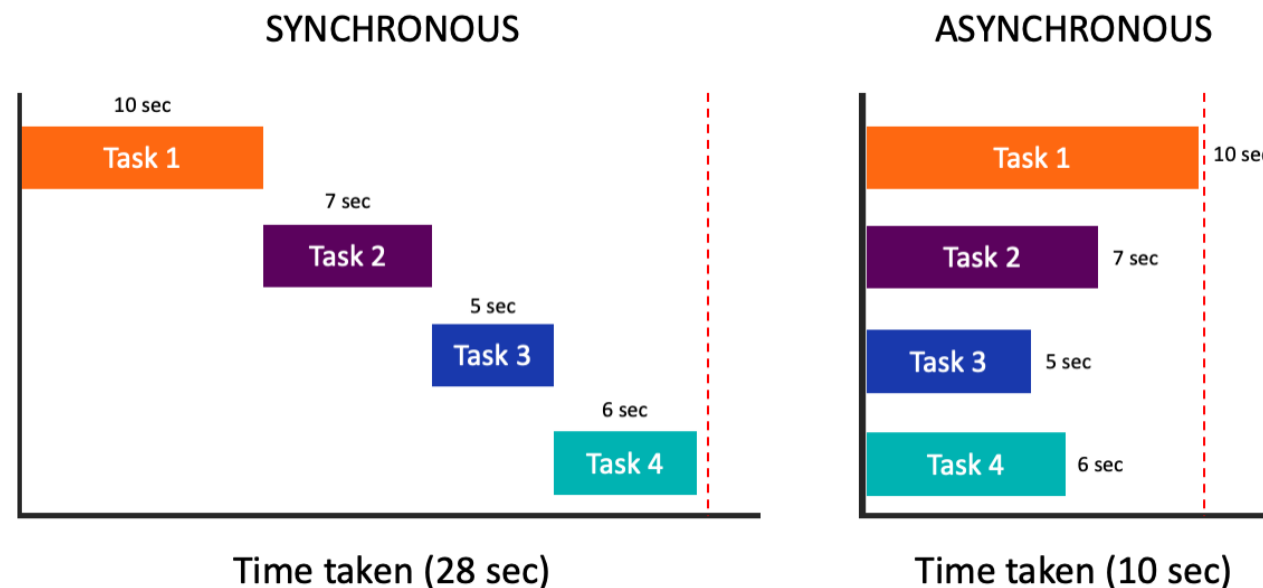
## Backend Application with Nodejs and Express

Instructor : Amonrat Prasitsupparote

# Nodejs

2

- Formerly, JavaScript was designed for a static webpage on a browser only.
- Node.js is an open-source and cross-platform **JavaScript runtime environment**.
  - Allows JavaScript to be no longer required to run in the browser.
  - Using Asynchronous and event driven are the main concept.



# Getting started to Nodejs

3

- Install Nodejs
  - Download at <https://nodejs.org>
- Run Nodejs file
  - `node <FILE_NAME>`

```
const a = 10;  
const b = 20;  
console.log("a*(a+b)=" + a*(a + b));
```

# Asynchronous Operations

- Callback is an asynchronous function
  - Called at the completion of a given task.
  - Equivalent to synchronous code

```
Function_Name (param1, param2, ..., callback)
```

```
const myRequest = (data, callback) => {  
  const response = 10 + data;  
  const error = undefined;  
  const result = callback(error, response);  
  return result;  
};  
const result = myRequest(5, (err, res) => {  
  if (err) return "Got error";  
  else return res;  
});  
console.log("result is " + result);
```

# Asynchronous Operations (Cont.)

- Promise is a way to deal with asynchronous code, without getting stuck in callback hell (complex nested callbacks).
  - Once a promise has been called, it will start in a **pending state**.
  - End in a **resolved state**, or in a **rejected state** (passed to then and catch)

```
New Promise((resolve, reject) => {...})
```

```
const getStd = (id) => {  
  return new Promise((resolve, reject) => {  
    setTimeout(() => {  
      let name = "James"; //undefined  
      if (name) resolve({ id: id, name: name });  
      else reject(new Error("Cannot get the result"));  
    }, 2000);  
  });  
};  
getStd(4)  
  .then((res) => {  
    console.log(res);  
  })  
  .catch((err) => {  
    console.log(err);  
  });
```

# Asynchronous Operations (Cont.)

6

- Async & Await is a modern asynchronous code, built on promises
  - Reduce the complexity of promises
  - When call the async function, prepend await
    - The calling code will stop until the promise is resolved or rejected

```
async Function_Name() {  
  //code  
}
```

OR

```
const Function_Name = async () => {  
  //code  
}
```

```
const checkAuth = (id, pass) => {  
  return new Promise((resolve, reject) => {  
    setTimeout(() => {  
      console.log("User authenticated");  
      resolve({ id: id, pass: pass });  
    }, 1000);  
  });  
};  
  
const getStd = (id) => {  
  return new Promise((resolve, reject) => {  
    setTimeout(() => {  
      console.log("Retrive student data");  
      resolve({ ...id, name: "James" });  
    }, 2000);  
  });  
};  
  
const getResult = async () => {  
  const auth = await checkAuth(2, "mypass");  
  const data = await getStd(auth);  
  console.log(data);  
};  
getResult();
```

# Module

7

- Same as JavaScript libraries.
- There are three types of modules
  - Build-in module or Core module, for example, http, fs, path, etc.
    - Usage: `const VAR_NAME = require("MODULE_NAME");`
  - Custom module or Local module – created locally in your Nodejs application
    - Use `module.exports` or `exports` to expose object, function or variable as a module in Node.js.
  - Third Party Modules – the module that available online using the Node Package Manager (NPM) to install first.
    - It can be installed in the project folder or globally

# RESTful API

8

- API (Application Program Interface) allows two applications to communicate with each other.
- RESTful API is based on representational state transfer (REST)
  - Also referred to as a RESTful web service or REST API
- An architectural style for an application program interface (API) that uses HTTP requests to access and use data.
  - That data can be used to GET (reading), PUT (updating), POST (creating) and DELETE data types.
- Data formats the REST API supports include:
  - application/json
  - application/xml



# HTTP response status codes

9

- Status codes are issued by a server in response to a client's request.
- HTTP response status codes are separated into five categories.
  - The first digit of the status code defines the category of response.
  - *1xx informational response* – the request was received, continuing process.
  - *2xx successful* – the request was successfully received, understood, and accepted.
    - 200 OK – the request succeeded.
  - *3xx redirection* – further action needs to be taken in order to complete the request.
  - *4xx client error* – the request contains bad syntax or cannot be fulfilled.
    - 401 Unauthorized – the authorization has been refused.
    - 404 Not Found – the server can not find the requested resource.
  - *5xx server error* – the server failed to fulfil an apparently valid request.

# Postman

10

- An API tool for testing the RESTful API.

Methods	URLs	Actions
GET	https://api.publicapis.org/categories	List all categories
GET	https://api.publicapis.org/entries/:category	List all entries currently cataloged in the project

GET <https://api.publicapis.org/categories> Send

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

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

Body Cookies Headers (9) Test Results 200 OK 313 ms 1.07 KB Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "count": 51,
3   "categories": [
4     "Animals",
5     "Anime",
6     "Anti-Malware",
7     "Art & Design",
8     "Authentication & Authorization",
9     "Blockchain",
10    "Books",
11    "Business",
12    "Calendar",
13    "Cloud Storage & File Sharing",
14    "Continuous Integration",
15    "Cryptocurrency",
16    "Currency Exchange",
```

GET <https://api.publicapis.org/entries?category=Photography> Send

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

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
<input checked="" type="checkbox"/> category	Photography			
Key	Value	Description		

Body Cookies Headers (9) Test Results 200 OK 318 ms 5.2 KB Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "count": 28,
3   "entries": [
4     {
5       "API": "apilayer screenshotlayer",
6       "Description": "URL 2 Image",
7       "Auth": "",
8       "HTTPS": true,
9       "Cors": "unknown",
10      "Link": "https://screenshotlayer.com",
11      "Category": "Photography"
12    },
13    {
14      "API": "APITemplate.io",
15      "Description": "Dynamically generate images and PDFs from templates with a simple API",
```

# Homework 2

11

- Find out the free API and write the usage instruction in A4 format with the following topics:
  - What is the purpose of API?
  - How to call API?
  - Describe the required parameters.
  - Describe the optional parameters.
  - Show an example of API calls through Postman including the description.
- Submit the PDF file at <https://bit.ly/465F3tV>
- Due Date: Sep 4

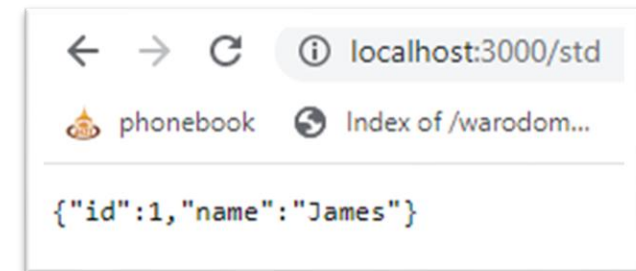
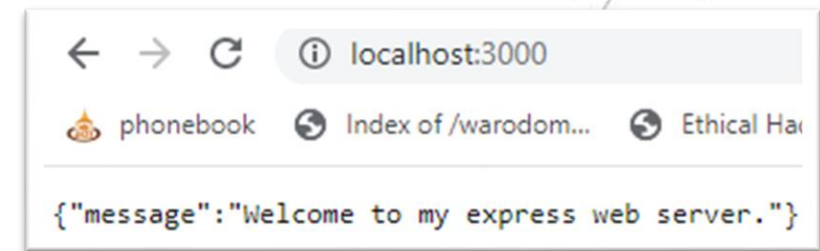
# Express

12

- Express is Nodejs framework that designed for building web applications and APIs by using JavaScript.
- Installation – \$ npm install express

```
const express = require("express");
const app = express();
app.get("/", (req, res) => {
  res.send({ message: "Welcome to my express web server." });
});
app.get("/std", (req, res) => {
  res.send({ id: 1, name: "James" });
});

const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}.`);
});
```



# nodemon

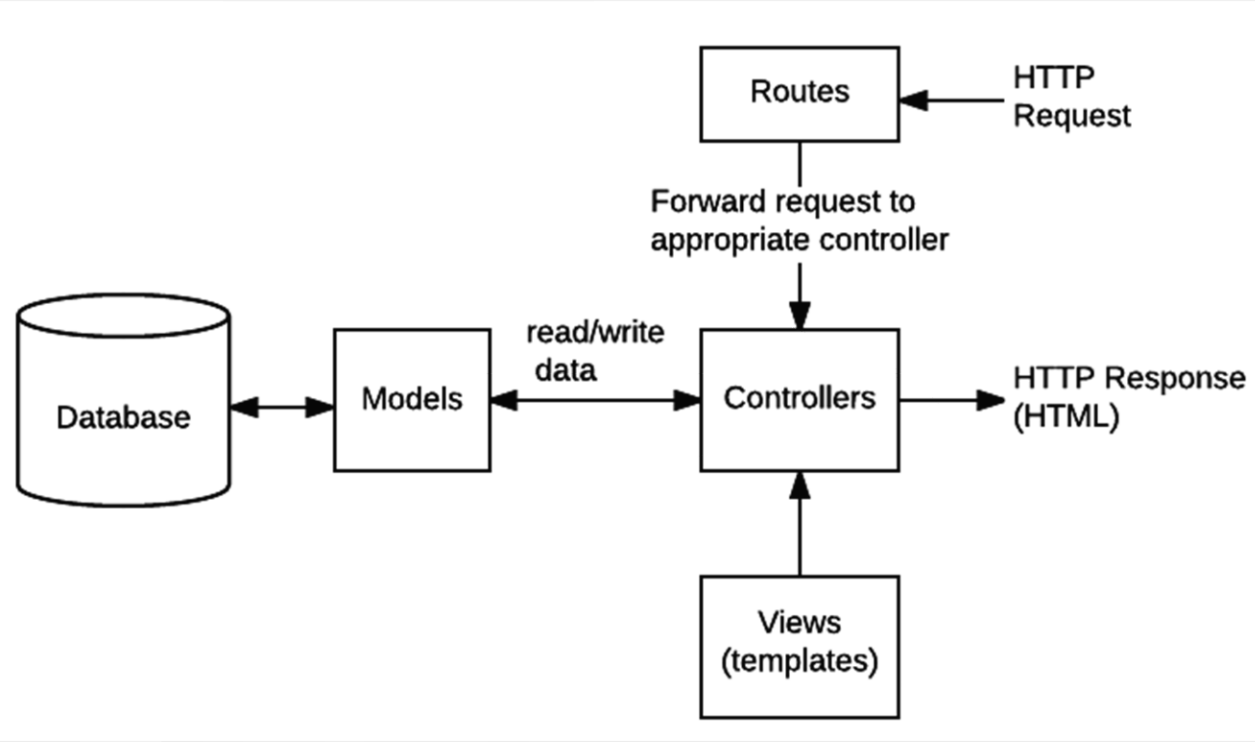
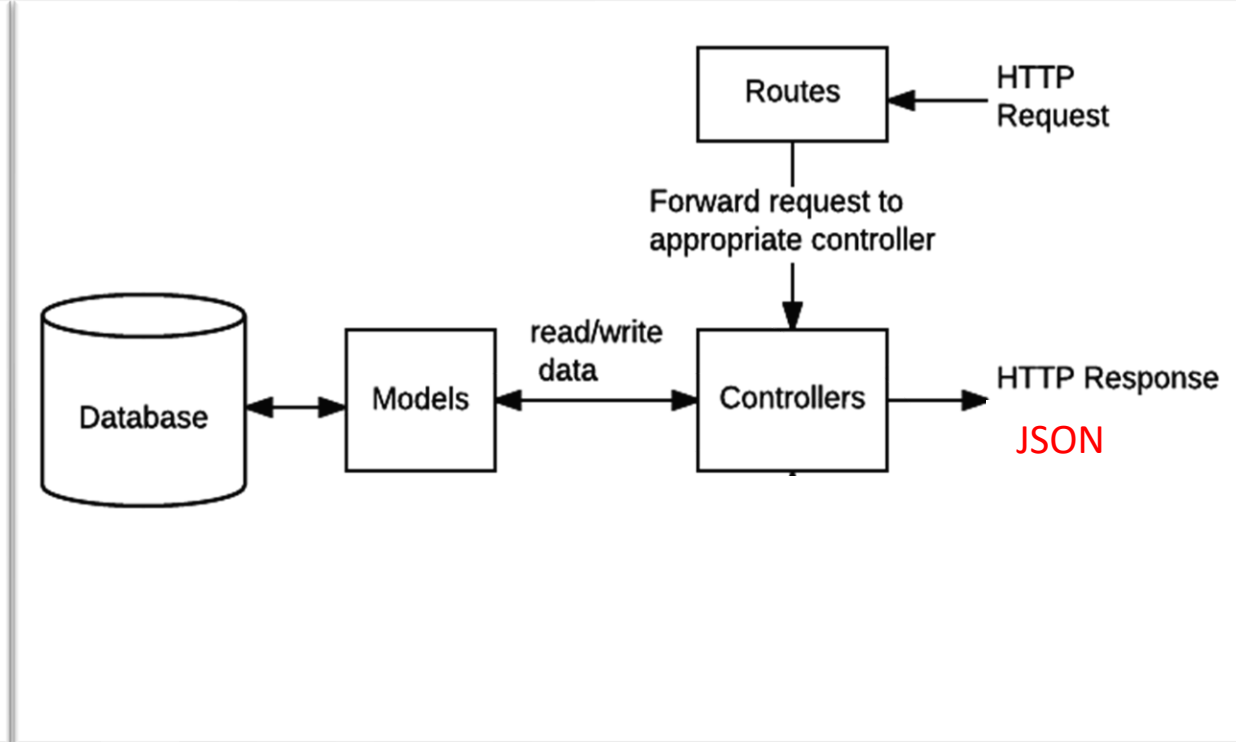
13

- Any change to the express file will not appear until restart the server (run file again).
- nodemon is a tool to solve this problem
- Installation – recommended dev dependency
  - Installed as globally – `$ npm install -g nodemon`
  - Installed as a development dependency – `$ npm install --save-dev nodemon`
- Usage
  - `$ npx nodemon Nodejs_FILE.js`
  - Update the scripts section in package.json file
    - Add “dev”: “nodemon Nodejs\_FILE.js”
    - Run `$ npm run dev`

# MVC

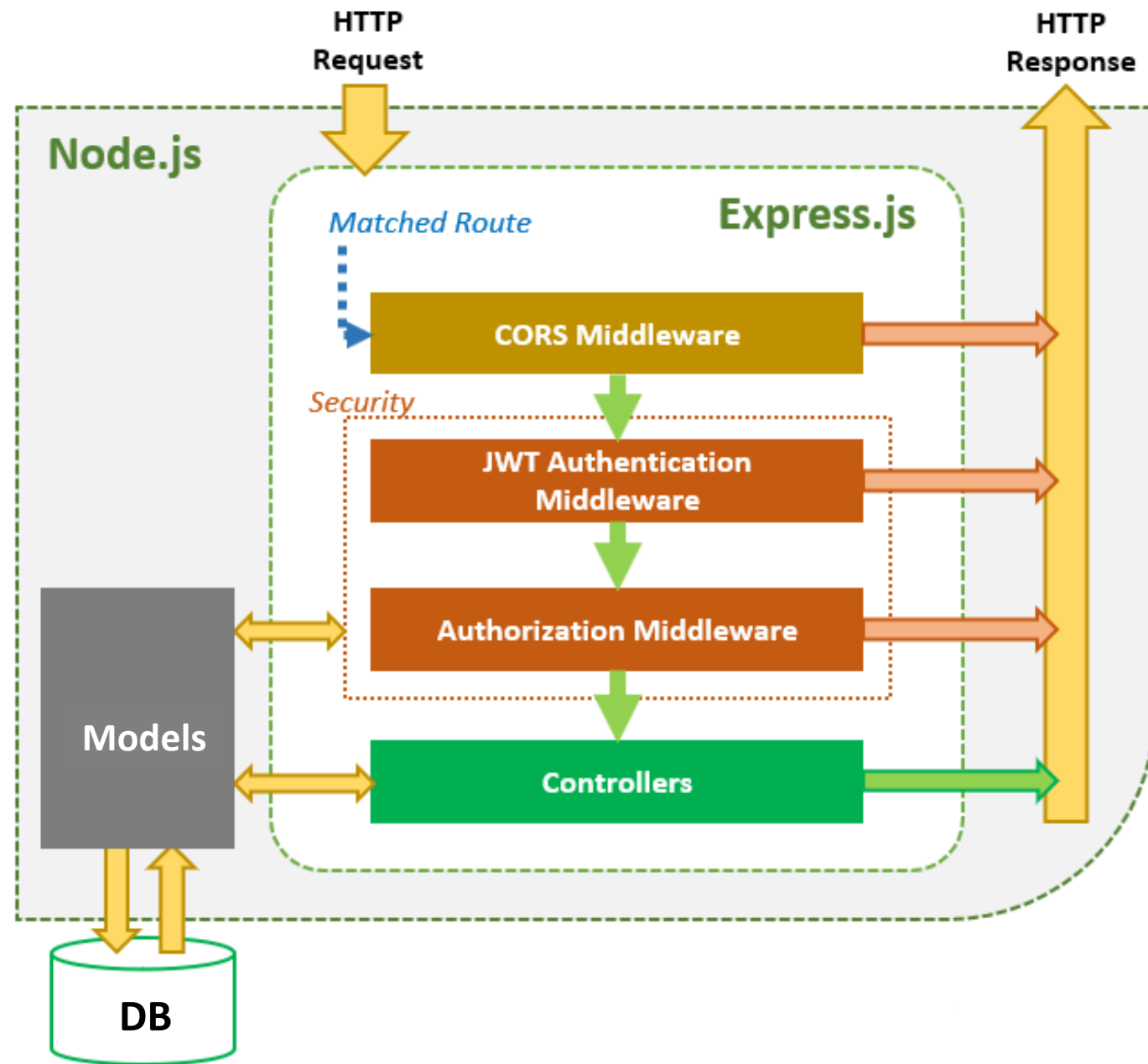
14

- MVC stands for Model, View, Controller.
- It is an architectural pattern.
- Three main logical components:
  - Model is the database interface that lets you interact with the database.
  - View used by the controllers to render the data into plain HTML.
    - It needs a Template Engine for the rendering process.
    - EJS is the famous Template Engine that works with Express.
  - Controller handles the HTTP Request and returns a response.

**MVC****RESTful API**

# Middleware

- Middleware is a type of computer software that provides services to software applications.





# RESTful API (Backend App) for Manage Accounts App

# Functions Listing

18

- Check the exist username
- Upload an image
- Create a new account
- Login
- List all users
- Modify the specific user
- Remove the specific user

# Production version

19

- Application
  - CoC (PSU Network): <http://172.26.117.16:5000>
  - Public: <https://manage-account-frontend.vercel.app>
- API
  - CoC (PSU Network): <http://172.26.117.16:3000>
  - Public: <https://manage-account-api.vercel.app>
  - Problem issue in the upload file due to Vercel is serverless.

# Create a node project

- \$ npm init OR Create *package.json* file
- Download DB <https://bit.ly/3BuNx0x>

```
package name: (account-rest)
version: (1.0.0)
description: A RESTful API for authentication and accounts management
entry point: (index.js)
test command:
git repository:
keywords:
author: Amonrat P.
license: (ISC)
About to write to D:\Gdrive\COC\Sem1-2565\977-110 WEB\Source\ch04\account-rest\package.json:
```

```
{
  "name": "account-rest",
  "version": "1.0.0",
  "description": "A RESTful API for authentication and accounts management",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Amonrat P.",
  "license": "ISC"
}
```

Is this OK? (yes) yes

```

  account-rest
  app
  config
    db.config.js
    jwt.config.js
  controllers
    file.controller.js
    user.controller.js
  middleware
    auth.jwt.js
    upload.js
  models
    db.js
    user.model.js
  routes
    file.routes.js
    user.routes.js
  assets\uploads
    FileUpload-1659519409782.jpeg
  > node_modules
    package-lock.json
    package.json
    server.js
```

# Install the necessary modules

```
$ npm install express mysql2 body-parser cors  
jsonwebtoken bcryptjs multer nodemon
```

- CORS (Cross-origin resource sharing) is an HTTP-header based mechanism that allows a server to indicate any origins (domain, scheme, or port) other than its own permit loading resources.
  - Restrict access only specific URL
    - Access-Control-Allow-Origin: <https://foo.example>
  - Allow any origin
    - Access-Control-Allow-Origin: \*
- JSON Web Token (JWT) is securely transmitting information between parties as a JSON object.
  - Using a public/private key pair to sign a token

# Install the necessary modules (Cont.)

22

- bcrypt.js is used to encrypt a plain text password with a hash algorithm.
  - It is one-way encryption and cannot decrypt the cipher to plain text.
  - Salt is random data that is used as an additional input to a one-way function
  - To hash a password

```
const saltRounds = 10;  
const salt = bcrypt.genSaltSync(saltRounds);  
const hash = bcrypt.hashSync(myPlaintextPassword, salt);
```

- Modify *package.json*

```
"scripts": {  
  "test": "echo \"Error: no test specified\" && exit 1",  
  "dev": "nodemon server.js",  
  "start": "node server.js"  
},
```

```
"dependencies": {  
  "bcryptjs": "^2.4.3",  
  "body-parser": "^1.20.0",  
  "cors": "^2.8.5",  
  "express": "^4.18.1",  
  "jsonwebtoken": "^8.5.1",  
  "multer": "^1.4.5-lts.1",  
  "mysql2": "^2.3.3",  
  "nodemon": "^2.0.19"  
}
```

# Setup Express web service

23

- Create *server.js* at the root folder
  - For execute the REST API
- Create *app* folder at the root folder
  - Keep all application files

```
const express = require("express");
const bodyParser = require("body-parser");
const cors = require("cors");
const app = express();

global.__basedir = __dirname;
var corsOptions = {
  origin: "*"
};

app.use(cors(corsOptions));
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }));
app.get("/", (req, res) => {
  res.json({ message: "Welcome to my REST." });
});

const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}.`);
});
```

# Config DB & JWT

- Create *config* folder under *app* folder to store the configuration files.
- Create *db.config.js* under *config* folder for the DB configuration.

```
module.exports = {  
  HOST: "localhost",  
  USER: "root",  
  PASSWORD: "",  
  DB: "testdbapi"  
};
```

- Create *jwt.config.js* under *config* folder for keep the secret word.

```
module.exports = { secret: "my-secret-key" };
```



# Define DB connection

- Create *model* folder under *app* folder to store the model files.
- Create *db.js* under *models* folder for DB connection.

```
const mysql = require("mysql2");
const dbConfig = require("../config/db.config.js");
// Create a connection to the database
const connection = mysql.createConnection({
  host: dbConfig.HOST,
  user: dbConfig.USER,
  password: dbConfig.PASSWORD,
  database: dbConfig.DB
});
// open the MySQL connection
connection.connect(error => {
  if (error) console.log("MySQL connection: " + error);
  else console.log("Successfully connected to the database.");
});
module.exports = connection;
```

# Define Middleware

- Create *middleware* folder under *app* folder.
- Create *auth.jwt.js* under *middleware* folder for verify token.

```
const jwt = require("jsonwebtoken");
const scKey = require("../config/jwt.config");
const verifyToken = (req, res, next) => {
  const token = req.headers["x-access-token"];
  if (!token) {
    return res.status(403).send({ message: "No token provided." });
  }
  jwt.verify(token, scKey.secret, (err, decoded) => {
    if (err) {
      return res.status(401).send({ message: "Unauthorized." });
    }
    req.id = decoded.id;
    next();
  });
};
module.exports = verifyToken;
```

# Define Middleware (Cont.)

27

- Create *upload.js* under *middleware* folder for upload file with *Multer*.
  - `util.promisify()` makes the exported middleware object can be used with `async-await`
- Create *assets* folder and *uploads* folder at the root for keep the upload files.

```
const util = require("util");
const multer = require("multer");
const storage = multer.diskStorage({
  destination: (req, file, cb) => {
    cb(null, __basedir + "/assets/uploads/");
  },
  filename: (req, file, cb) => {
    const extArray = file.mimetype.split("/");
    const extension = extArray[extArray.length - 1];
    const newFileName = `FileUpload-${Date.now()}.${extension}`;
    cb(null, newFileName);
  },
});
const uploadFile = multer({ storage: storage }).single("singlefile");
const uploadFileMiddleware = util.promisify(uploadFile);
module.exports = uploadFileMiddleware;
```

# Define Routes

28

- Create *routes* folder under *app* folder.
- Create *file.routes.js* under *routes* folder.
- Create *user.routes.js* under *routes* folder.
- Add routes to *server.js*

```
// server.js
require("../app/routes/file.routes")(app);
require("../app/routes/user.routes")(app);
```

```
// file.routes.js
module.exports = app => {
  const file_controller = require("../controllers/file.controller");
  var router = require("express").Router();
  router.post("/upload", file_controller.upload);
  app.use("/api/file", router);
};
```

```
// user.routes.js
const authJwt = require("../middleware/authJwt");
module.exports = (app) => {
  const user_controller = require("../controllers/user.controller");
  var router = require("express").Router();
  router.post("/signup", user_controller.createNew);
  router.get("/:us", user_controller.validateUsername);
  router.post("/login", user_controller.login);
  router.get("/", authJwt, user_controller.getAllUsers);
  router.put("/:id", authJwt, user_controller.updateUser);
  router.delete("/:id", authJwt, user_controller.deleteUser);
  app.use("/api/auth", router);
};
```

# Define the model & controller

29


- Create *user.model.js* under *models* folder.
- Create *controllers* folder under *app* folder.
- Create *file.controller.js* under *controllers* folder
- Create *user.controller.js* under *controllers* folder


```
// file.controller.js
const uploadFile = require("../middleware/upload");
const upload = async (req, res) => {
  try {
    await uploadFile(req, res);
    if (req.file === undefined) {
      return res.status(400).send({ message: "Please upload a file!" });
    }
    res.status(200).send({
      message: "Uploaded the file successfully: " + req.file.filename,
      uploadFileName: req.file.filename,
    });
  } catch (err) {
    res.status(500).send({
      message: "Could not upload the file:" + err,
    });
  }
};
module.exports = { upload };
```

# Check Exist Username

30

Method	URLs	Description
GET	<i>DOMAIN/api/auth/:us</i>	Check the exist username


 Manage Acc App





Your Fullname  
Amonrat P.

Your Email  
amonrat.pr@phuket.psu.ac.th

Your Username  
jare

 This username is NOT Available.


 Manage Acc App



Your Fullname  
Amonrat P.

Your Email  
amonrat.pr@phuket.psu.ac.th

Your Username  
jare2

 This username is Available.

# Test Check Exist Username By Postman

GET ▼ http://172.26.117.16:3000/api/auth/v22

Params Authorization Headers (7) Body Pre-req

Query Params

KEY
Key

Body Cookies Headers (8) Test Results

Pretty Raw Preview Visualize JSON ▼

```

1  {
2    "message": "Not found v22",
3    "valid": true
4  }
```


GET ▼ http://172.26.117.16:3000/api/auth/v23

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

Query Params

KEY	VALUE
Key	Value

Body Cookies Headers (8) Test Results

Pretty Raw Preview Visualize JSON ▼ 

```

1  {
2    "record": {
3      "id": 31,
4      "fullname": "Test Ver23",
5      "email": "v23@q.qq",
6      "username": "v23",
7      "password": "$2a$10$bg2jY0jKF9mMi0TlIr0PpuzpkaEuS/0l/zNJTYzNd9tgcJcBAE2j2",
8      "img": "FileUpload-1658085529952.png"
9    },
10   "valid": false
11 }
```

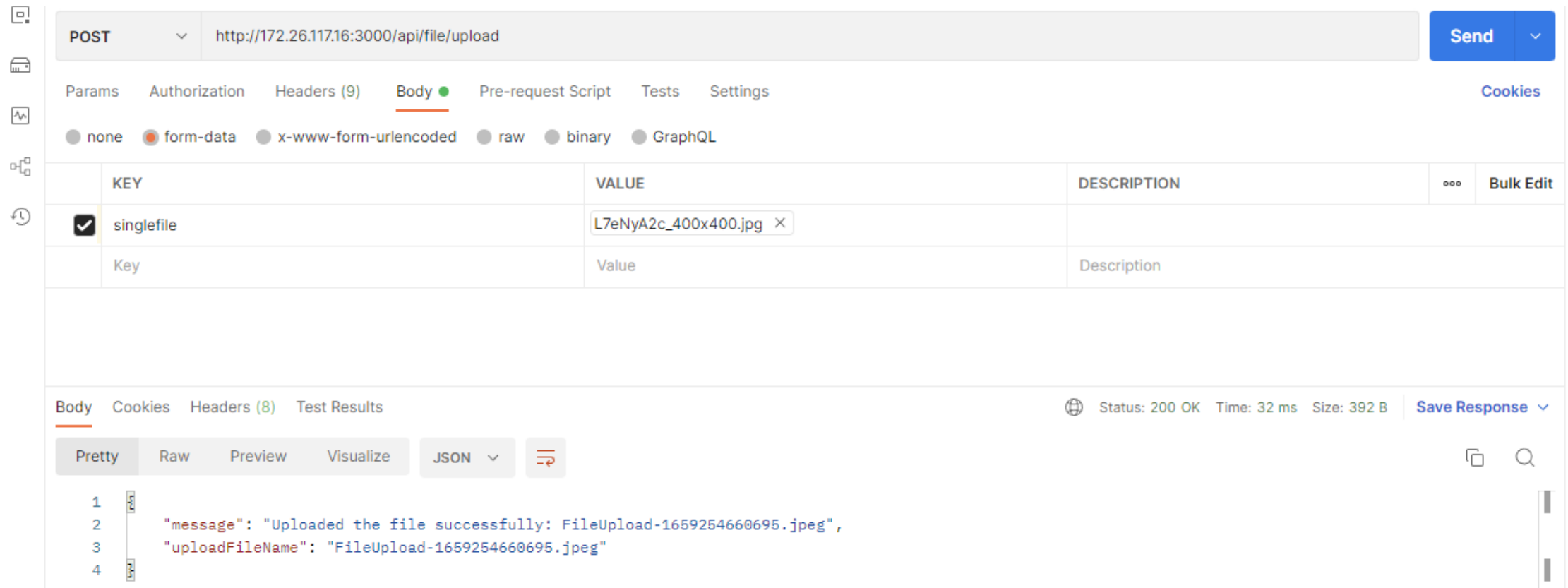
# Upload an image

32

Method	URLs	Description
POST	<i>DOMAIN/api/file/upload</i>	Upload a single file to the server Required body: <ul style="list-style-type: none"><li>• singlefile</li></ul>



# Test Upload an image By Postman



The screenshot displays the Postman application interface for testing an API endpoint. The top section shows a **POST** request to `http://172.26.117.16:3000/api/file/upload`. The **Body** tab is selected, showing a **form-data** type with a single key-value pair: **singlefile** with the value `L7eNyA2c_400x400.jpg`. Below this, a table lists the request details: **KEY**, **VALUE**, and **DESCRIPTION**. The **Body** tab is also selected in the bottom section, showing the response in **JSON** format. The response status is **200 OK** with a time of **32 ms** and a size of **392 B**. The response body contains a JSON object with a success message and the uploaded file name.

**POST** `http://172.26.117.16:3000/api/file/upload` **Send**

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings Cookies

☐ none ☒ form-data ☐ x-www-form-urlencoded ☐ raw ☐ binary ☐ GraphQL

	KEY	VALUE	DESCRIPTION	...	Bulk Edit
<input checked="" type="checkbox"/>	singlefile	L7eNyA2c_400x400.jpg			
	Key	Value	Description		

**Body** Cookies Headers (8) Test Results **Status: 200 OK** Time: 32 ms Size: 392 B **Save Response**

Pretty Raw Preview Visualize JSON

```
1 {
2   "message": "Uploaded the file successfully: FileUpload-1659254660695.jpeg",
3   "uploadFileName": "FileUpload-1659254660695.jpeg"
4 }
```

# Download an image

34

Method	URLs	Description
GET	<i>DOMAIN/api/file/:name</i>	Download a specific file

GET localhost:3000/api/file/FileUpload-1666673650984.jpeg Send

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

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

Body Cookies Headers (12) Test Results




Status: 200 OK Time: 47 ms Size: 25.25 KB Save Response



# Create New Account

Method	URLs	Description
POST	<i>DOMAIN/api/auth/signup</i>	Add a new user Required body: <ul style="list-style-type: none"><li>• fullname</li><li>• email</li><li>• username</li><li>• password</li><li>• img &lt;optional&gt;</li></ul>

Manage Acc App

35

# Test Create New Account By Postman

The image shows the Postman application interface. At the top, a POST request is configured to the URL `http://172.26.117.16:3000/api/auth/signup`. The 'Body' tab is selected, and the request body is a JSON object with the following fields: `fullname`, `email`, `username`, and `password`. The 'Send' button is visible in the top right corner.

Below the request editor, the 'Body' tab of the response is selected. The response is a JSON object with the following fields: `id`, `fullname`, `email`, `username`, `password`, and `accessToken`. The status is 200 OK, the time is 187 ms, and the size is 601 B. The 'Save Response' button is visible in the top right corner of the response section.

**Request Body:**

```
1 {
2   "fullname": "Amonrat Prasitsupparote",
3   "email": "amonrat.pr@phuket.psu.ac.th",
4   "username": "myus",
5   "password": "myps"
6 }
```

**Response Body:**

```
1 {
2   "id": 40,
3   "fullname": "Amonrat Prasitsupparote",
4   "email": "amonrat.pr@phuket.psu.ac.th",
5   "username": "myus",
6   "password": "$2a$10$rDzpkI8wltfbxZv0u0eQweCUTWd95yw4gMiNHpmtVPcvtJjeimJB2",
7   "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6NDAsIm1hdCI6MTY1OTI1MjY2MCwiZXhwIjoxNjU5MjU5ODYwYwQ.
8     D9DFWAzq-vFn7XG7MCBFjL5LrxmTMSHlpuWuEj1MK58"
9 }
```

# Login

Method	URLs	Description
POST	<i>DOMAIN/api/auth/login</i>	User authentication Required body: <ul style="list-style-type: none"><li>• username</li><li>• password</li></ul>

☰ Manage Acc App



Your Username

Your Password



SUBMIT

Not registered? [Create an Account](#)

37

# Test Login By Postman

The screenshot displays the Postman application interface for a POST request to `http://172.26.117.16:3000/api/auth/login`. The request body is a JSON object with `username: "myus"` and `password: "mysps"`. The response status is 200 OK, and the response body is a JSON object containing user details and an access token.

**Request Details:**

- Method: POST
- URL: `http://172.26.117.16:3000/api/auth/login`
- Body Type: JSON
- Body Content:

```
1 {
2   "username": "myus",
3   "password": "mysps"
4 }
```

**Response Details:**

- Status: 200 OK
- Time: 155 ms
- Size: 601 B
- Body Type: JSON
- Body Content:















```
1 {
2   "id": 40,
3   "fullname": "Amonrat Prasitsupparote",
4   "email": "amonrat.pr@phuket.psu.ac.th",
5   "username": "myus",
6   "password": "$2a$10$rDzpkI8wltfbxZv0uOeQweCUTWd95yw4gMiNHpmtVPcvtJjeimJB2",
7   "img": null,
8   "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpYXQiOiJlZ2NTkyNTI4NDksImV4cCI6MTY1OTI2MDA0OX0.Hv5QFqrPy8ohFEW8vDPoAvcibDnp9YskzmmVAL0cKbQ"
9 }
```

# List All Users

Method	URLs	Description
GET	<i>DOMAIN/api/auth</i>	List all users Required JWT <ul style="list-style-type: none"><li>Use "<b>x-access-token</b>" in header</li></ul>

Manage Acc App

Hello f2

ID	FullName	Email	Actions
1	test fullname2	test email2	 
2	test2 fullname	test2 email	 
6	f2	f2@f.ff	 
7	f3	f3@f.ff	 
8	f4	f4@f.ff	 
10	f6 Edit	f6@f.ff666	 
11	f7edit	f7@dd.ee	 

Records per page: 7

1-7 of 30 |< < > >|

# Test List All Users By Postman

The screenshot shows the Postman interface for a GET request to `http://172.26.117.16:3000/api/auth`. The request is configured with the following headers:

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> x-access-token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpYXQiOiJlbnR5NTI...	
Key	Value	Description

The response is displayed in the Body tab, showing a JSON array of two user objects. The status is 200 OK, with a response time of 18 ms and a size of 5.29 KB.

```

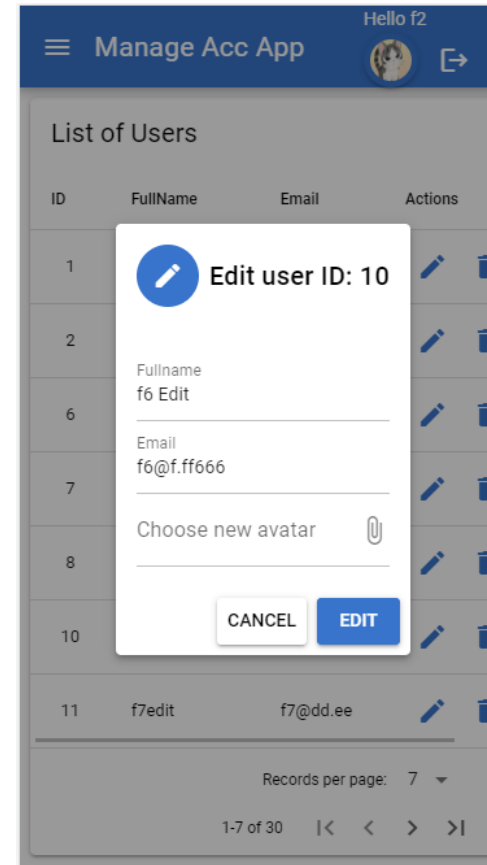
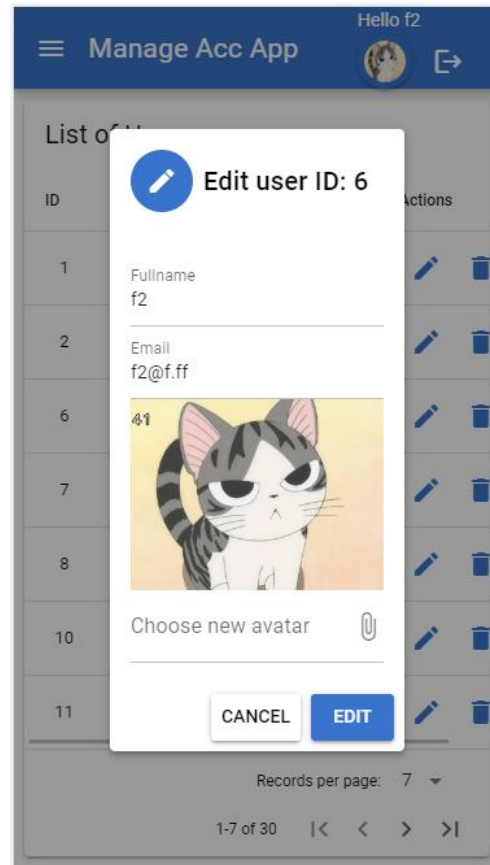
1  [
2    {
3      "id": 1,
4      "fullname": "test fullname2",
5      "email": "test email2",
6      "username": "test username",
7      "password": "$2a$08$T0B0i/96KE90jAYPOhpsN.vJGVPMfFw.FbxljzuQkkN4ZK3YauRLq",
8      "img": "FileUpload-1658154816561.png"
9    },
10   {
11     "id": 2,
12     "fullname": "test2 fullname",
13     "email": "test2 email",
14     "username": "test2 username",
  
```



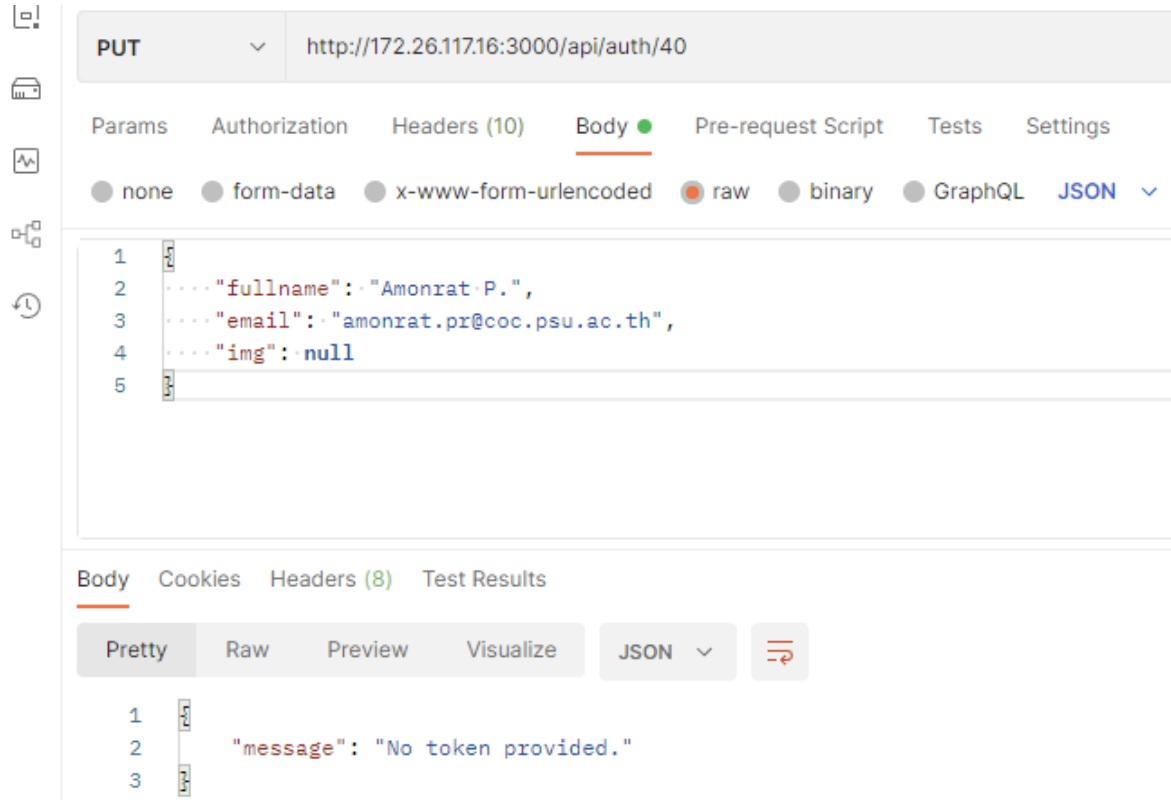
# Modify Specific User

41

Method	URLs	Description
PUT	<i>DOMAIN/api/auth/:id</i>	Update the specific user by id Required JWT <ul style="list-style-type: none"><li>Use "<b>x-access-token</b>" in header</li></ul>



# Test Modify Specific User By Postman



PUT ▼ http://172.26.117.16:3000/api/auth/40

Params Authorization Headers (10) **Body** Pre-request Script Tests Settings

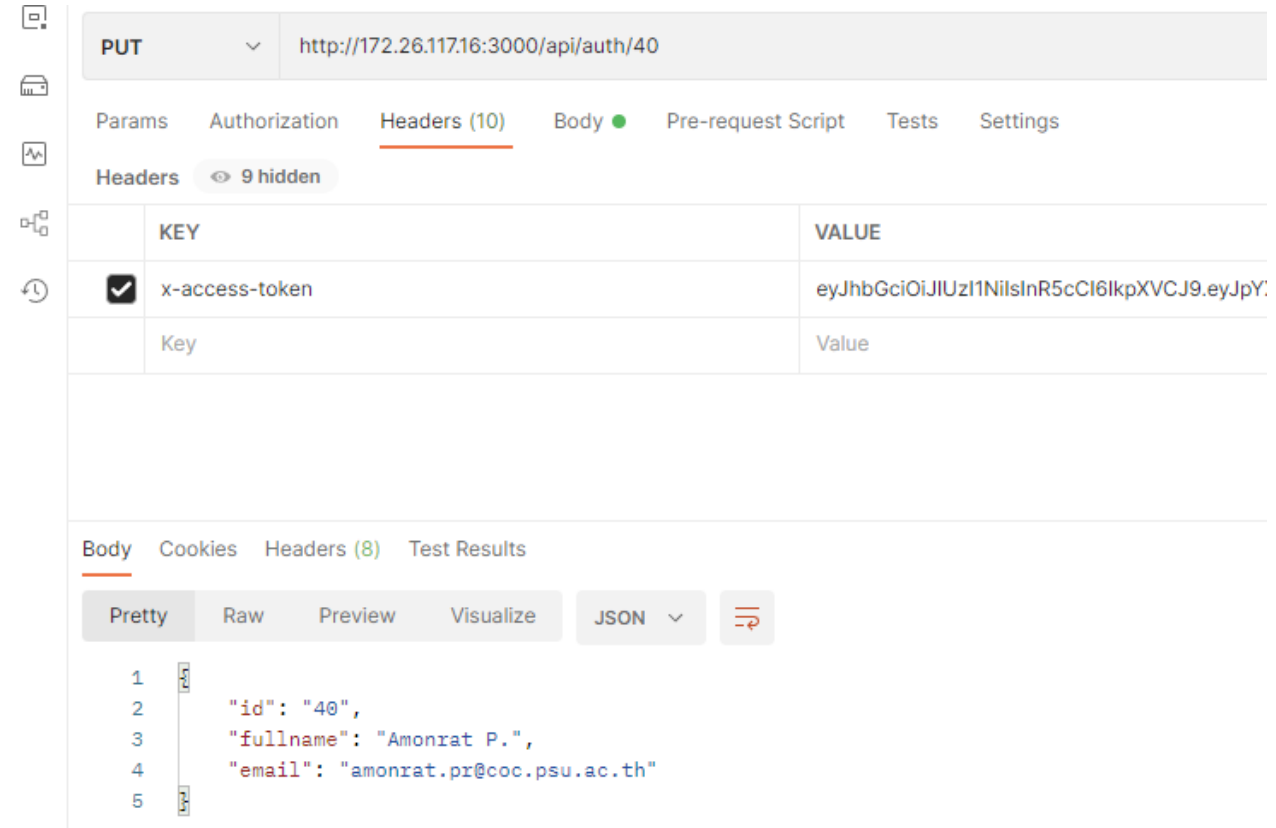
● none ● form-data ● x-www-form-urlencoded ● raw ● binary ● GraphQL **JSON** ▼

```
1 {
2   "fullname": "Amonrat P.",
3   "email": "amonrat.pr@coc.psu.ac.th",
4   "img": null
5 }
```

Body Cookies Headers (8) Test Results

Pretty Raw Preview Visualize **JSON** ▼

```
1 {
2   "message": "No token provided."
3 }
```



PUT ▼ http://172.26.117.16:3000/api/auth/40

Params Authorization **Headers (10)** Body Pre-request Script Tests Settings

Headers 9 hidden

	KEY	VALUE
<input checked="" type="checkbox"/>	x-access-token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpY...
	Key	Value

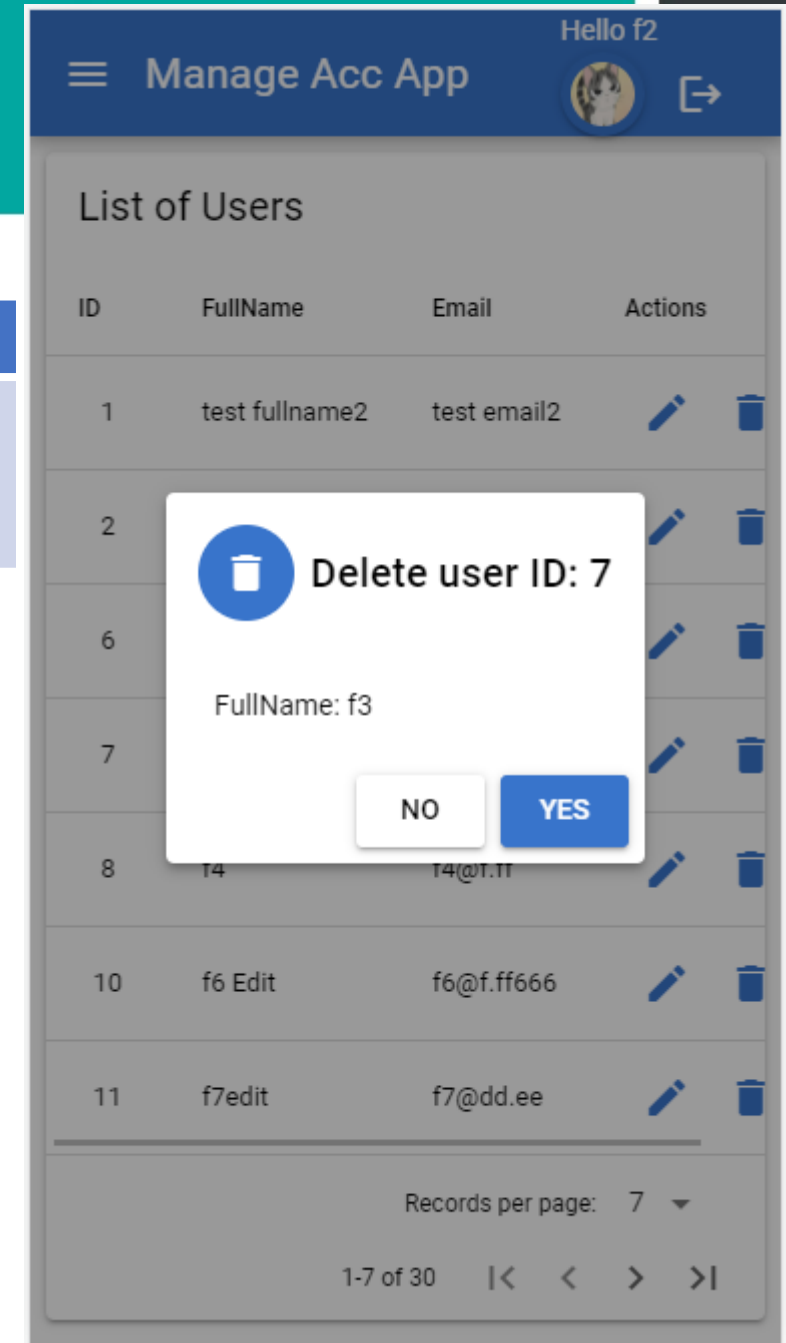
Body Cookies Headers (8) Test Results

Pretty Raw Preview Visualize **JSON** ▼

```
1 {
2   "id": "40",
3   "fullname": "Amonrat P.",
4   "email": "amonrat.pr@coc.psu.ac.th"
5 }
```

# Remove Specific User

Method	URLs	Description
DELETE	<i>DOMAIN/api/auth/:id</i>	Remove the specific user by id Required JWT <ul style="list-style-type: none"><li>Use "<b>x-access-token</b>" in header</li></ul>



# Test Remove Specific User By Postman

The screenshot displays the Postman interface for a DELETE request. The URL is `http://172.26.117.16:3000/api/auth/40`. The **Headers** tab is selected, showing 8 headers, with 7 hidden. One header is visible: `x-access-token` with a value `eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpYXQiOiJlbnR5NTI...`. The **Body** tab is also active, showing a JSON response: `{ "id": "40" }`. The status is 200 OK, time is 29 ms, and size is 277 B.

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> x-access-token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpYXQiOiJlbnR5NTI...	
Key	Value	Description

```
1 {
2   "id": "40"
3 }
```

# Deploy to Public Server

# Deploy to Vercel

46

- Vercel is a platform for frontend developers
  - Install Vercel-cli
  - Create Vercel account
- Git is a distributed version control system
  - git-scm at <https://git-scm.com/download>
  - Create Github account

```
$ npm i -g vercel
```

# Vercel Configuration

47

- Create *vercel.json* under the root directory of your project.
  - version  $\geq 2$
  - name [optional] – string name for the deployment
    - A maximum length of 52 characters
    - Only lower case alphanumeric characters or hyphens are allowed
    - Cannot begin or end with a hyphen, or contain multiple consecutive hyphens
- See more detail at <https://vercel.com/docs/concepts/projects/project-configuration#project/>

```
{  
  "version": 2,  
  "name": "ANY_STRING",  
  "builds": [{ "src": "server.js", "use": "@vercel/node" }],  
  "routes": [{ "src": "/(.*)", "dest": "/server.js" }]  
}
```

# Free Database Server

48

- FREEDB.TECH (<https://freedb.tech> )
  - 1 MySQL Database with 50 MB Storage
  - Limited Queries
  - Need login before the usage, waiting for update service > 10 mins
- PlanetScale (<https://planetscale.com> )
  - NO phpMyAdmin
  - Required SSL connection
  - Connection String > create password
  - Connect with: General
  - 1 Database



# MySQL Workbench

49

## Setup New Connection

Connection Name:

Connection Method:

Parameters ☒ SSL ☐ Advanced

Hostname:  Port:  Name or IP address of the MySQL host.  
TCP/IP port.

Username:  Name of the user to connect as.

## Setup New Connection

Connection Name:

Connection Method:

Parameters ☒ SSL ☐ Advanced

Use SSL:  Turns on SSL encryption. If not available, the connection will fail.

# Test MySQL connection from localhost

50

- PlanetScale

- Connect with: Node.js
- Install module dotenv
- Create a file at the root directory, named **".env"**, for localhost only.
- Create file **".gitignore"**
- Modify *models/db.js*

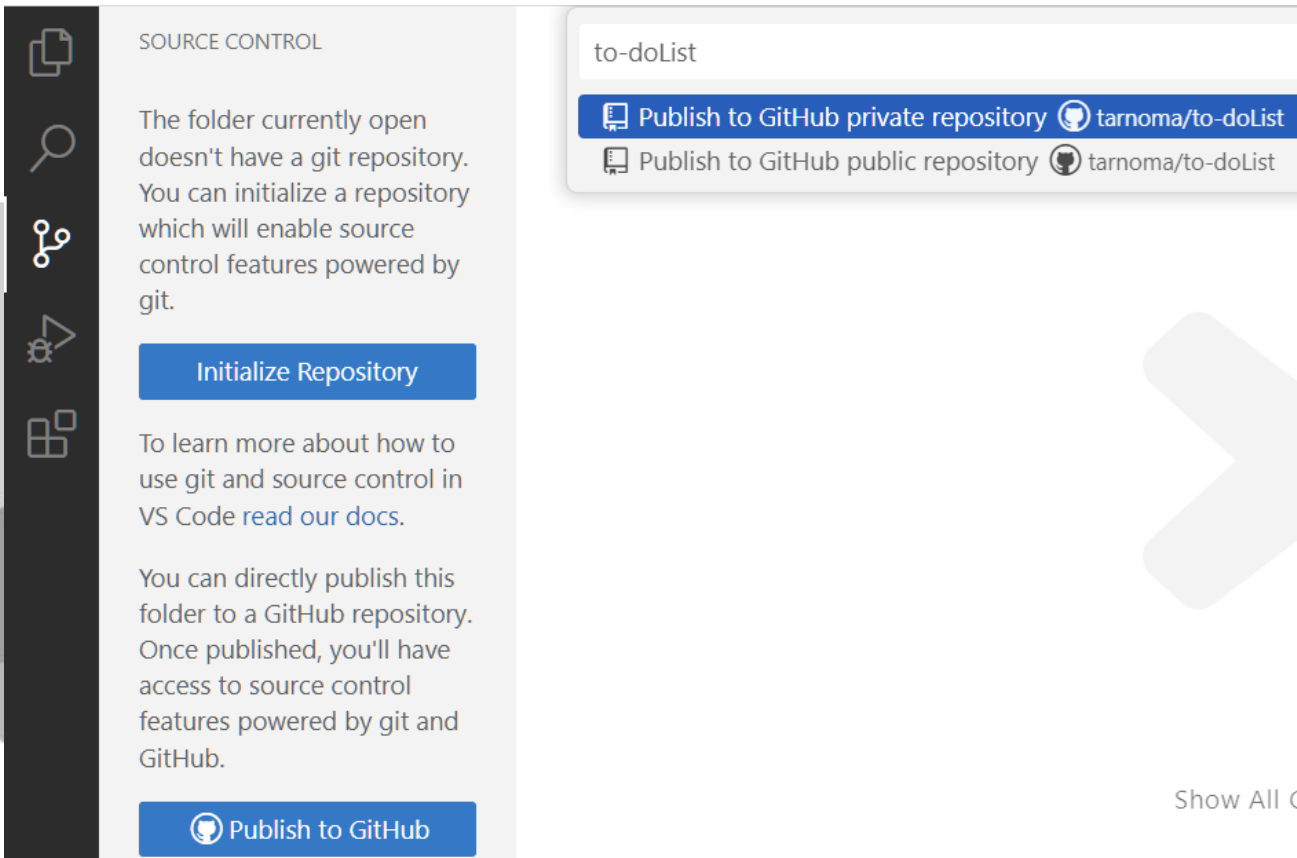
```
$ npm install dotenv
```

```
// db.js
const mysql = require("mysql2");
require("dotenv").config()
const connection =
mysql.createConnection(process.env.DATABASE_URL)
```

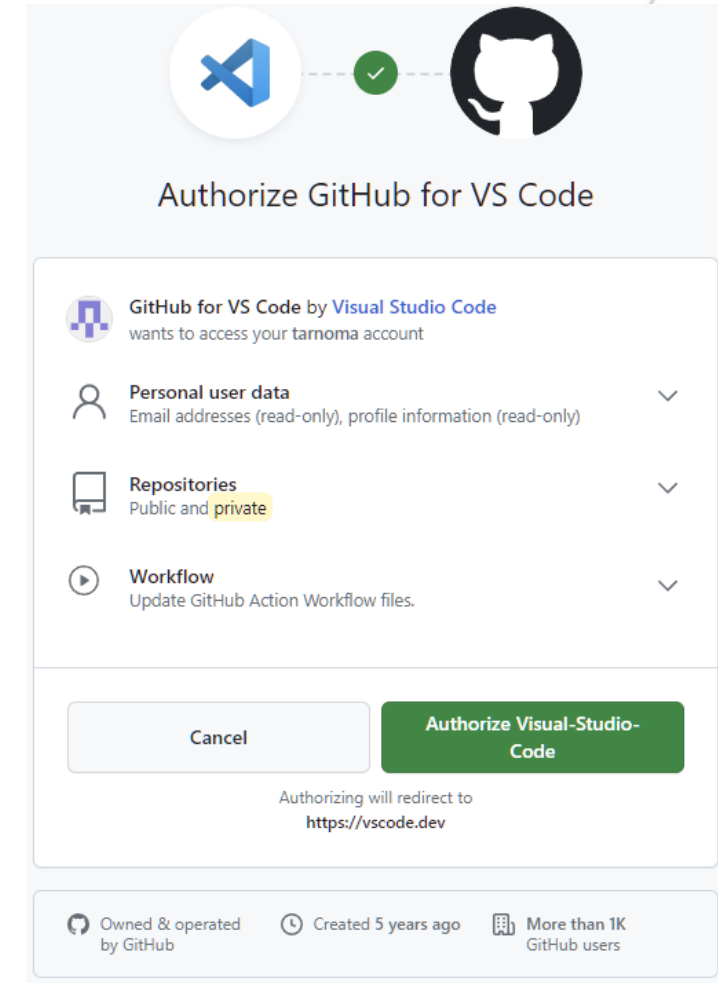
# Upload files to Github

51

- Create *.gitignore* file at root directory



The screenshot shows the VS Code Source Control panel. On the left, a sidebar contains icons for Explorer, Search, Source Control, Run and Debug, and Extensions. The main area is titled "SOURCE CONTROL" and contains the following text: "The folder currently open doesn't have a git repository. You can initialize a repository which will enable source control features powered by git." Below this text is a blue button labeled "Initialize Repository". Further down, it says "To learn more about how to use git and source control in VS Code read our docs." and "You can directly publish this folder to a GitHub repository. Once published, you'll have access to source control features powered by git and GitHub." At the bottom is a blue button labeled "Publish to GitHub". A context menu is open over the "Publish to GitHub" button, showing two options: "Publish to GitHub private repository" and "Publish to GitHub public repository", both associated with the user "tarnoma/to-doList".



The screenshot shows the "Authorize GitHub for VS Code" dialog box. At the top, it features the VS Code logo, a green checkmark, and the GitHub logo. Below the logos is the title "Authorize GitHub for VS Code". The main content area lists the permissions that GitHub for VS Code by Visual Studio Code wants to access: "Personal user data" (Email addresses (read-only), profile information (read-only)), "Repositories" (Public and private), and "Workflow" (Update GitHub Action Workflow files). Each item has a dropdown arrow to its right. At the bottom, there are two buttons: "Cancel" and "Authorize Visual-Studio-Code". Below the buttons, it says "Authorizing will redirect to https://vscode.dev". At the very bottom, there is a footer section with three items: "Owned & operated by GitHub", "Created 5 years ago", and "More than 1K GitHub users".

# Upload files to Github (manually)

52

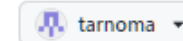
- Create a new repository

```
$ git init
$ git add .
$ git commit -m
  "YOUR_COMMIT_WORD"
$ git branch -M main
$ git remote add origin
  "GIT_REPOSITORY"
$ git push -u origin main
```

## Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner \*



Repository name \*

electric\_system\_api

✓ electric\_system\_api is available.

Great repository names are short and memorable. Need inspiration? How about [solid-adventure](#)?

Description (optional)

☒ Public

Anyone on the internet can see this repository. You choose who can commit.

☐ Private

You choose who can see and commit to this repository.

Initialize this repository with:

☐ Add a README file

This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore

.gitignore template: None

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license

License: None

A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

You are creating a public repository in your personal account.


Create repository


# Create a new project on Vercel


53

## Import Git Repository

Select a Git provider to import an existing project from a Git Repository.

 Continue with GitHub

 Continue with GitLab

 Continue with Bitbucket

[Manage Login Connections](#)

[Import Third-Party Git Repository](#) →

## Import Git Repository

 tarnoma


 Search...



electric\_system\_api · 1m ago

Import

## Import Git Repository

 tarnoma

 tarnoma

+ Add GitHub Account

☰ Switch Git Provider

# Create a new project on Vercel (Cont.)

54

- Environment variables
  - Only value, without single quote.

**Configure Project**

Project Name  
electric-system-api

Framework Preset  
Other

Root Directory  
./ Edit

> Build and Output Settings

> Environment Variables

Deploy

**Configure Project**

Project Name  
daily-expense4-vercel-api

Framework Preset  
Other

Root Directory  
./ Edit

> Build and Output Settings

Environment Variables

Name	Value (Will Be Encrypted)	
DATABASE_URL	I9JU23NF394R6HH	Add

TIP: Paste a .env above to populate the form [Learn more about Environment Variables](#)

Deploy



# Config Environment Variable on Vercel

55

## Environment Variables

In order to provide your Deployment with Environment Variables at Build and Runtime, you may enter them right here, for the Environment of your choice. [Learn more](#)

A new Deployment is required for your changes to take effect.

Key	Value
<input type="text" value="DATABASE_URL"/>	<input type="text" value="xxxxxx"/>  
<a href="#">+ Add another</a>	

Environment

☒ Production

☒ Preview [Select custom branch](#)

☒ Development

[↓ Import](#) TIP: Paste a .env above to populate the form [Save](#)

# Free Hosting

56

- [cyclic.sh](https://cyclic.sh)
- [digitalocean.com](https://digitalocean.com)
- [railway.app](https://railway.app)
- [render.com](https://render.com)
- [adaptable.io](https://adaptable.io)
- [fly.io](https://fly.io)



# Homework 3

57

- Deploy your Manage Accounts API to any hosting that can be accessed anywhere.
- After that fill your URL at <https://bit.ly/46fTBHx>
- Due Date: XXXX

# References

58

- “Vue.js - The Progressive JavaScript Framework | Vue.js.”  
<https://vuejs.org/guide/introduction.html> (accessed Jul. 19, 2022).
- “Quasar Framework - Build high-performance VueJS user interfaces in record time,”  
*Quasar Framework*. <https://quasar.dev/> (accessed Jul. 19, 2022).
- Node.js, “Documentation,” *Node.js*. <https://nodejs.org/en/docs/> (accessed Jul. 19, 2022).
- “HTML Tutorial.” <https://www.w3schools.com/html/default.asp> (accessed Jul. 19, 2022).
- “HTML: HyperText Markup Language | MDN.” <https://developer.mozilla.org/en-US/docs/Web/HTML> (accessed Jul. 19, 2022).
- “CSS Tutorial.” <https://www.w3schools.com/css/default.asp> (accessed Jul. 19, 2022).