

# (Workshop 5) Introduction to backend

🕒 Created	@December 18, 2024 11:12 PM
👤 Last edited by	Ⓚ Kai Xun
🕒 Last edited time	@February 14, 2025 6:14 PM

## ▼ 📖 Important Content

This is the exercise folder we will be using for our workshop later

[cyc-workshop-introduction-to-backend \(v2\).zip](#)

## ▼ 📖 (Recap) What are HTTP request? (~1min)

**source:**

[HTTP Requests Defined: What They Are & How They Work - Sematext](#)

## Hypertext Transfer Protocol (HTTP)

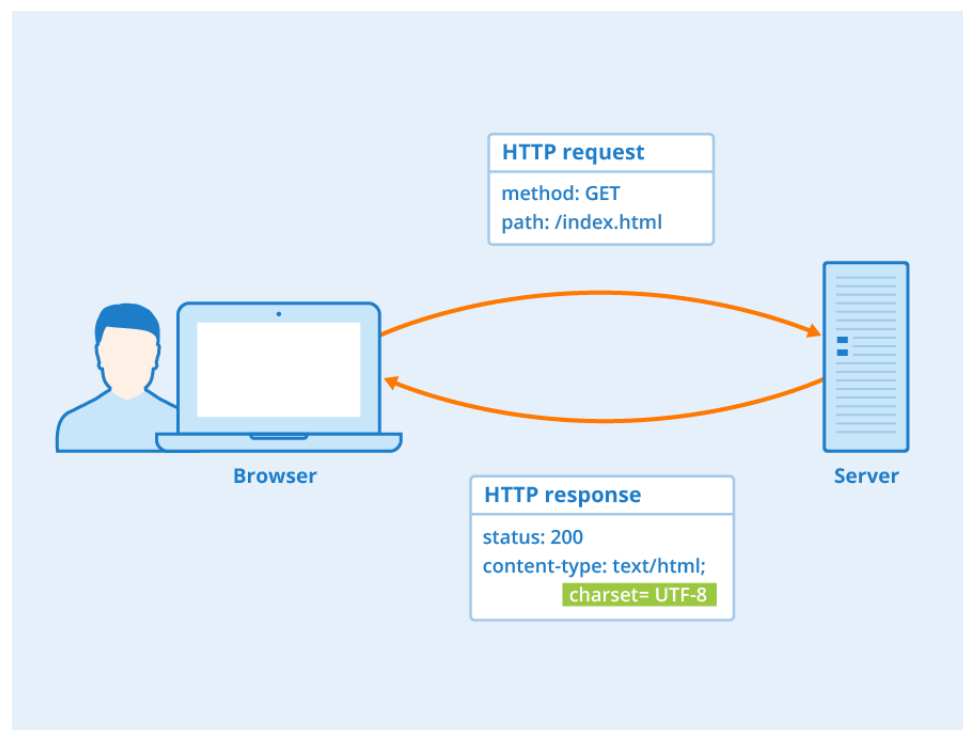
HTTP is a command language that enables communication between a client (you) and a server (who you want to talk to). The client makes a request, the

server validates the request, processes the required function, and returns a response containing a header and message body.

## HTTP request Methods

HTTP supports several main request methods, each serving a specific purpose:

1. **GET** - Retrieves data from the server.
2. **POST** - Submits new data to the server.
3. **DELETE** - Removes data from the server.
4. **PUT/PATCH** - Updates existing data on the server.

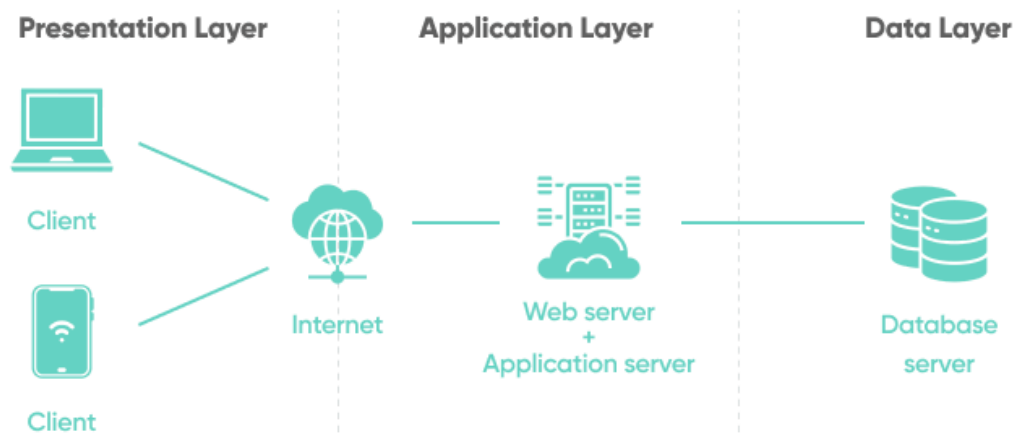


Source: <https://tech.jotform.com/understanding-http-headers-f240f215f37b>

## ▼ What is a Backend? (~4mins)

**Backend programming languages** (as known as Backend) acts as the brains and central nervous system of your application. They handle all the logic and functions that you need for your application to work. While the **Frontend** handles the user interface (what users see and interact with), the **Backend** ensures that:

- User inputs are processed correctly.
- Data is securely stored and retrieved.
- Critical application logic runs efficiently.



## Why are backend important?

1. **Data Management** - Backend handles the storage, retrieval and management of the application data through the database. For example if you log in to a website, your credentials are sent to the backend to be verified and compared against stored data.
2. **Security** - protects sensitive user information (e.g. passwords, credit card details) through encryption and secure APIs, ensuring data integrity and privacy
3. **Business Logic** - all rules and operations that are defines how your application works is usually executed in the backend
4. **Integration** - backend is can be used to connect your applications to external services (think DBS/OCBC card payments, email services, etc.)

## Types of Backend languages

There are 100s of backend languages for you to pick and choose from, each with their unique features to suit your needs. Some note-worthy backend programming languages include:

- Python
- Java
- JavaScript
- Express



We will be using **NodeJS** and **Express**, two beginner friendly JavaScript language frameworks to handle our APIs

## Variables In Backend

Similar to what was covered during our JavaScript workshop, variables are also used in backend languages like Express and Nodejs to store data values that can be referenced and manipulated.

To understand more about the different types of variables used in the backend, you can refer to this notion page here:

✨ (Workshop 2) [Enhancing Websites](#)

## ▼ 🧰 Tools to make backend development easier (~5-10mins)

### ▼ 🖼️ NodeJS and Express

**Node.js** is a tool that lets you run JavaScript on the server, making it possible to build fast and efficient backend systems. It's lightweight and great for real-time applications like chat apps.

**Express.js** is a simple framework built on top of Node.js that helps you create web servers and APIs easily. It provides features to handle routes, requests, and responses, making backend development quicker and more beginner-friendly.

Together, Node.js and Express allow you to build powerful backend systems using JavaScript.

**npm (Node Package Manager)** is the default package manager for Node.js. It allows developers to install, manage, and share packages (libraries or

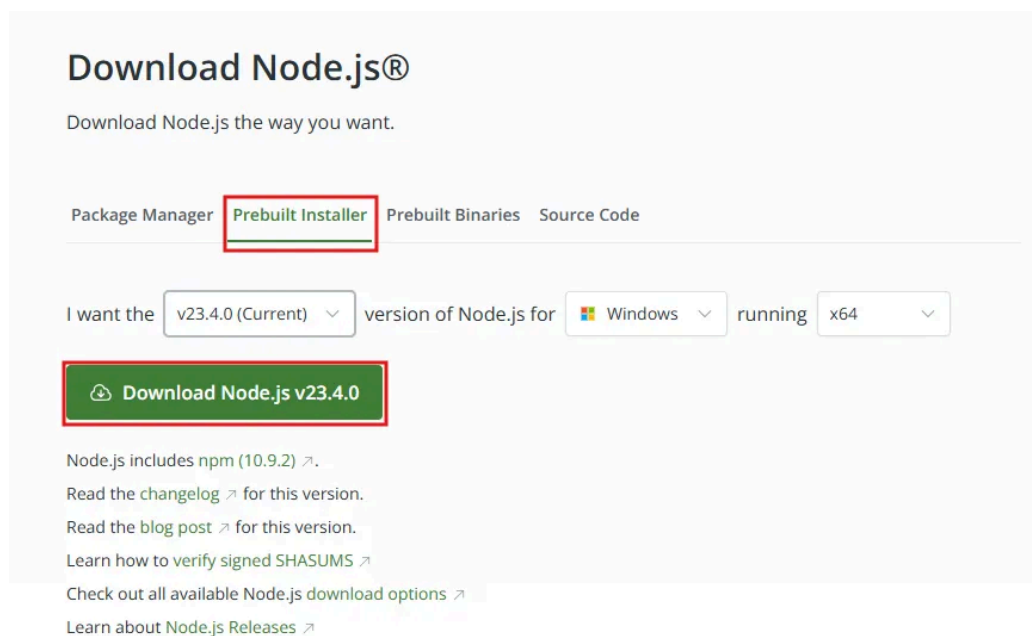
modules) created by the community. With npm, you can easily integrate external tools, frameworks, and utilities into your projects, speeding up development and reducing the need to write code from scratch.

- **npm** comes packaged together when you install Node.js on your computers. There are many different package managers for the different backends languages out there. some nonedible ones include **pip** (for python), **composer** (for PHP) or **Maven/Gradle** (for Java)

## ▼ Installation process

To start you will need install Node.js on your computer

1. Go to Node.js installation page  
(<https://nodejs.org/en/download/prebuilt-installer>)
2. In pre-built installer section, click on the download button as shown in the screenshot below



3. Follow the installation instructions and click on **install** at the end
  - a. To test for the successful installation of Node.js, open you command prompt and enter this command to get to NodeJS version installed on your computer:

```
node -v
```

Now let's install express on your computer

1. Create a new folder in your desired location

```
mkdir backend
```

2. initialize your backend application

```
npm init -y
```

3. install express via this command:

```
npm install express
```

The following files will be created after running all the commands as mentioned above.



4. Install `nodemon` to help automatically restart your servers when file changes are detected


```
npm install --save-dev nodemon
```

```
// To run - `npx nodemon app.js`
```

And voila ✨✨✨, you have successfully installed Node.js and express!!

## ▼ Creating your first Backend Function (~20-30mins)

Now let create your first backend function. You will be able to test your functions from both your **web browser** and **Postman**. If you have not installed postman on your computer, you can follow this guide created by my fellow instructor here:

 [\(Workshop 4\) Introduction to API](#)

It's located under the "**resources to install/download**" section.

## ▼ (beginner) Setting up and creating a simple GET functions

1. create a new file name **app.js**
2. Within app.js, we will be initializing our backend on app.js and adding two simple GET functions add the following code in

```
'use strict';
// app.js file
const express = require('express');
const app = express();
const PORT = 3000;

// define the route functions in your backend

// to test if your backend is reachable on the internet
app.get('/', (req, res) => {
  res.send(
    "Hello World"
  );
});

// Function to return hello message
function getHelloMessage() {
  return `

# 


```

```

}

// (Beginner) A simple route to call you function
app.get('/hello', (req, res) => {
  res.send(
    getHelloMessage()
  );
});

// this part of the code here lets you know that the backend server is runni
app.listen(PORT, () => {
  console.log(
    `Server is listening at http://localhost:${PORT}`
  );
});

```

3. run the follow command to start up your backend instance on localhost:

```
npx nodemon app.js
```

4. go to your web browser and enter the following link(s) separately

URL 1: <http://localhost:3000/>

URL 2: <http://localhost:3000/hello>

## ▼ 🏃 (intermediate) Creating a GET functions that takes in parameters

Parameters are variables (string, int, bool) that you can pass to your backend function to perform a specific task

1. in your app.js add the follow code

```

// (Intermediate) a backend function that takes in variables
function getHelloUserMessage(params) {
  return `<h1 style="color: blue;">Hello ${params.name}</h1>`;
}

```



```
}

// (Intermediate) A simple route to call your function
app.get('/hello/:name', (req, res) => {
  res.send(
    getHelloUserMessage({ name: req.params.name })
  );
});
```



Using functions outside of your routes helps to make your code readability to you and other developers and it allows for your to reuse the function if needed

2. Save your file changes by doing a `ctrl + s` and your changes will be reflected. If you have start your backend server, you may start it up again by running

```
npx nodemon app.js
```

3. go to your web browser and enter the following link

```
http://localhost:3000/hello/tom
```

You can try changing your tom to your own name!

## Now you try!

Here's are some backend questions for you to try it out on your own.

### ▼ 🏃 TASK #1 : Convert Temperature (from Celsius to Fahrenheit)



Create an API endpoint that converts **Celsius to Fahrenheit**.

### Instructions:

- Create a **GET** route at `/convert/:celsius`.
- Retrieve `celsius` from `req.params`.
- Convert it to **Fahrenheit** using the formula:

$$\text{Fahrenheit} = (\text{Celsius} * 9/5) + 32$$

- Return both **Celsius** and **Fahrenheit** values as JSON.

### Expected Request

GET /convert/25

### Expected Response

```
{
  "celsius": 25,
  "fahrenheit": 77
}
```

### Hints

- Use `req.params.celsius` to access the user input.
- Ensure `celsius` is a **number** and not any other data type.
- Return a **JSON response** with both Celsius and Fahrenheit.

## ▼ TASK #2 : Reverse a String



Create an API endpoint that **reverses a given string**.

Instructions:

- Create a **GET** route at `/reverse/:word`.
- Retrieve `word` from `req.params`.
- Reverse the word using JavaScript.
- Return both **original** and **reversed** versions as JSON

### Expected Request

```
GET /reverse/hello
```

### Expected Response

```
{
  "original": "hello",
  "reversed": "olleh"
}
```

### Hints

- Use `req.params.word` to access the user input.
- Use `.split("").reverse().join("")` to **reverse the string**.
- Return a **JSON response** with both values.

## ▼ Introduction to Database (DB) (~30-45mins)

### ▼ What are databases?

A **database** is a structured collection of data. They store essential information like user details or transaction records while enabled backend servers to **retrieve, update and manage** data efficiently.



Think about the games you play or the apps you used. How do they keep your information? How do they update whenever you play or post something online? → **Databases!**

There are tons of database engines and services to choose from but we will be covering **Supabase** SQL today!

## Basic SQL Commands (CRUD Operations)

SQL (Structured Query Language) is used to interact with databases. Here are the fundamental CRUD operations:

### CREATE - Adding Data

```
-- Insert a single row
INSERT INTO table_name
VALUES ('value1', 'value2');

-- Insert multiple rows
INSERT INTO table_name (column1, column2)
VALUES
  ('value1', 'value2'),
  ('value3', 'value4');
```

### READ - Retrieving Data

```
-- Select all columns
SELECT * FROM table_name;

-- Select specific columns
SELECT column1, column2 FROM table_name;

-- Select with conditions
```

```
SELECT * FROM table_name  
WHERE column1 = 'value';
```

## UPDATE - Modifying Data

```
-- Update records  
UPDATE table_name  
SET column1 = 'new_value'  
WHERE condition;  
  
-- Update multiple columns  
UPDATE table_name  
SET  
    column1 = 'new_value1',  
    column2 = 'new_value2'  
WHERE condition;
```

## DELETE - Removing Data

```
-- Delete specific records  
DELETE FROM table_name  
WHERE condition;  
  
-- Delete all records  
DELETE FROM table_name;
```

💡 Always use WHERE clauses with UPDATE and DELETE to avoid unintended changes to your data!

## ▼ What is Supabase?


Supabase is an open-source Backend as a Service (BaaS) platform that provides developers with a suite of tools to build scalable applications swiftly. At its core, each Supabase project includes a dedicated PostgreSQL database, renowned for its robustness and reliability.


### Key Features:

- **Instant APIs:** Supabase auto-generates RESTful APIs for your database tables, enabling immediate data interaction without developing your APIs.
- **Authentication:** Integrated user authentication and authorisation simplify user management and security.
- **Realtime Capabilities:** Applications can subscribe and react to database changes as they occur, facilitating dynamic user experiences.
- **Storage:** Supabase offers scalable storage solutions for managing and serving large files, such as images and videos.

Supabase | The Open Source Firebase Alternative

Build production-grade applications with a Postgres database, Authentication, instant APIs, Realtime, Functions, Storage and Vector embeddings. Start for

 <https://supabase.com/>


  
**Build in a weekend**  
**Scale to millions**


## ▼ Setting up with Supabase

We will be create a temporary email you can use for this demo. Head over to the follow website

Temp Mail - Disposable Temporary Email

Keep spam out of your mail and stay safe - just use a disposable temporary email address! Protect your personal email address from spam with Temp-mail

 <https://temp-mail.org/en/>





if you are comfortable you may use your own personal emails to create a Supabase account as it can be super helpful if you are doing any other projects

Once you have emails set, head over to Supabase to create a account

#### Supabase

By continuing, you agree to Supabase's Terms of Service and Privacy Policy, and to receive periodic emails with updates.

 <https://supabase.com/dashboard/sign-up>



you won't need to create a database in your Supabase account do give me a try after the workshop!

#### Video tutorial

[attachment:6ddd1750-a8a1-4fc5-a400-88845164a6a5:supbase\\_account\\_creation.mp4](#)

### ▼ Running your applications with Supabase

Now let's running out application with Supabase. First head over to the

`Introduction-to-sql` folder. cd into `back-end (exercise)` and run `npm install`

Before we jump into running the application lets discuss what the code structure means

```

'use strict';
const { createClient } = require('@supabase/supabase-js');
require('dotenv').config();

// Express Setup
const express = require('express');
const cors = require('cors');
const app = express();
const PORT = 5000;

app.use(cors());
app.use(express.json());

// Supabase Configuration
const supabaseUrl = 'https://cswjtvfmczcepiugldbe.supabase.co';
const supabaseKey = process.env.SUPABASE_KEY;
const supabase = createClient(supabaseUrl, supabaseKey);

console.log("✅ Connected to Supabase");

/* =====
  ♦ AUTHENTICATION ROUTES
  ===== */

/**
 * ✅ GET: Check if a user is logged in
 */
app.get('/auth/user', async (req, res) => {
  const { data: { user }, error } = await supabase.auth.getUser();
  if (error) return res.status(500).json({ error: error.message });
  res.json({ user });
});

/**
 * ✅ POST: User Signup

```



```

*/
app.post('/auth/signup', async (req, res) => {
  const { email, password } = req.body;
  const { data, error } = await supabase.auth.signUp({ email, password });

  if (error) return res.status(400).json({ error: error.message });
  res.json({ message: "User signed up successfully", data });
});

/**
 * ✅ POST: User Login
 */
app.post('/auth/login', async (req, res) => {
  const { email, password } = req.body;
  const { data, error } = await supabase.auth.signInWithPassword({ email,

  if (error) return res.status(400).json({ error: error.message });
  res.json({ message: "User logged in successfully", data });
});

/**
 * ✅ POST: User Logout
 */
app.post('/auth/logout', async (req, res) => {
  await supabase.auth.signOut();
  res.json({ message: "User logged out successfully" });
});

/* =====
 *   ♦ MENU ROUTES
 * ===== */

//... more can be found in your app.js file

```

This is a food ordering application with a menu item list and the ordering feature. This application also include user authentication that is handle via Supabase.

To access Supabase, you just need 3 lines of code:

```
// Supabase Configuration
const { createClient } = require('@supabase/supabase-js');
const supabaseUrl = 'https://your-supabase-url.supabase.co';
const supabaseKey = process.env.SUPABASE_KEY;
const supabase = createClient(supabaseUrl, supabaseKey);
```

- **supabaseUrl** tells your backend server where your online database is located at
- **supabaseKey** serves as the way to access your database

you will need to install the Supabase JavaScript package library by running

```
npm install @supabase/supabase-js
```


You may have notice **supabaseKey** is defined as a `process.env.SUPABASE_KEY` , this is mainly due to the fact that keys are considered as sensitive information we need to protect in our application. They are like passwords to the applications we use.



In order to store sensitive information, a environment variable file ( `.env` ) is used to safely store such information within. JavaScript offers a special package for the servers to read from the `.env` file to access specific services. All you need to do is run `npm install dotenv` and add the follow line in.


```
require('dotenv').config();
//example usage of process.env
const supabaseKey = process.env.SUPABASE_KEY;
```

Each features has it own functions (with each routes) to handle different parts of your features. (e.g. menu)

```

/**
 *  GET: Fetch all menu items
 */
app.get('/menu', async (req, res) => {
  const { data, error } = await supabase.from('menu_items').select('*');
  if (error) return res.status(500).json({ error: error.message });
  res.json(data);
});

/**
 *  TASK 1: Add a New Menu Item (POST /menu)
 * -----
 * Instructions:
 * - Accept `name`, `description`, and `price` from `req.body`.
 * - Insert the new menu item into the `menu_items` table.
 * - Return a success message with the added item.
 */
app.post('/menu', async (req, res) => {
  //  Participants must implement this!
});

/**
 *  DELETE: Remove a menu item
 */
app.delete('/menu/:id', async (req, res) => {
  const { id } = req.params;

  const { error } = await supabase
    .from('menu_items')
    .delete()
    .eq('id', id);

  if (error) return res.status(500).json({ error: error.message });
});

```

```
res.json({ message: "Menu item deleted successfully" });
});
```

## Now you try!

Here's are some backend questions (with Supabase) for you to try it out on your own.

### ▼ 🏃 TASK #1: add a new menu item (POST /menu) (~5 mins)



**Goal:** Create an API endpoint that allows users to add new menu items to the database.

#### Instructions:

- Build a `POST /menu` API route in `app.js`.
- Accept `name`, `description`, and `price` from the request body.
- Insert the new menu item into the `menu_items` table.
- Return a success message with the added item.

```
/**
 * 🎯 TASK 1: Add a New Menu Item (POST /menu)
 * -----
 * Instructions:
 * - Accept `name`, `description`, and `price` from `req.body`.
 * - Insert the new menu item into the `menu_items` table.
 * - Return a success message with the added item.
 */
app.post('/menu', async (req, res) => {
  // 🚧 Participants must implement this!
});
```

#### Expected Request

```
POST /menu
{
  "name": "Cheese Burger",
  "description": "Juicy beef patty with cheese",
  "price": 5.99
}
```

### Expected Response

```
{
  "message": "Food item added successfully!",
  "item": {
    "name": "Cheese Burger",
    "description": "Juicy beef patty with cheese",
    "price": 5.99
  }
}
```

### Hints:

- Use `req.body` to access the incoming data.
- Check for missing fields and return an error if any are missing.
- Use `supabase.from('menu_items').insert([...])` to add an item.
- Return a JSON response with the newly added menu item.

## ▼ TASK #2: Fetch Order History for a User (GET /orders/user/:user\_id) (~5 mins)



**Goal:** Create an API endpoint that retrieves a user's order history, including the ordered items and their details.

### Instructions:

- Build a `GET /orders/user/:user_id` API route in `app.js`.

- Retrieve all orders that belong to the given `user_id`.
- Use a **JOIN query** to fetch **order items** and their associated **menu item details**.
- Return a structured JSON response showing the user's **order history**.

```
/**
 * 🎯 TASK 2: Fetch Order History for a User (GET /orders/user/:user_id)
 * -----
 * Instructions:
 * - Retrieve all orders for the given `user_id`.
 * - Use a JOIN query to fetch order items & menu item details.
 * - Return a JSON response with the order history.
 */
app.get('/orders/user/:user_id', async (req, res) => {
  // ⚠️ Participants must implement this!
});
```

### Expected Request

```
// example user_id
GET /orders/user/123e4567-e89b-12d3-a456-426614174000
```

### Expected Response

```
[
  {
    "id": 1,
    "total_amount": 12.99,
    "order_date": "2024-02-07T12:00:00Z",
    "order_items": [
      { "name": "Cheese Burger", "quantity": 2, "price": 5.99 }
    ]
  }
]
```

### Hints:

- Use `req.params.user_id` to access the user's ID.
- Use Supabase JOIN queries to fetch order items & menu item details.
- Only return orders related to the given `user_id`.
- If no orders exist, return an empty array.


## ▼ TASK #3: Delete an Order (DELETE /orders/:id) (~10 mins)



**Goal:** Create an API endpoint that allows users to delete an order along with its associated order items.

### Instructions:

- Build a `DELETE /orders/:id` API route in `app.js`.
- Delete **all** `order_items` related to the given `order_id`.
- Delete the **order itself** from the `orders` table.
- Return a **success message**.

```
/**
 *  TASK 3: Delete an Order (DELETE /orders/:id)
 * -----
 * Instructions:
 * - Delete all `order_items` related to the order.
 * - Delete the `order` entry from the `orders` table.
 * - Return a success message.
 */
app.delete('/orders/:id', async (req, res) => {
  //  Participants must implement this!
});
```

### Expected Request

```
// example order_id
DELETE /orders/1
```

### Expected Response

```
{
  "message": "Order deleted successfully!"
}
```

### Hints


- Use `req.params.id` to access the order ID.
- First delete from `order_items`, then from orders.
- Ensure you handle errors gracefully (e.g., if the order doesn't exist).
- Return a success message after deletion.

## ▼ Useful Resources

We have come to the end of our workshop!! Here are some useful resources to learn more about backend!


#### Quick Node/Express.Js Project Setup Guide.

This guide is for developers returning to Express.js with essential tools already installed. It serves as a quick refresher on foundational...

 <https://medium.com/@naveednadaf/quick-node-express-js-project-setup-guide-88cd4d9a7af3>

#### Folder structure for a Node JS project - GeeksforGeeks


A Computer Science portal for geeks. It contains well written, well thought and well explained computer science and programming articles, quizzes and practice/competitive

 <https://www.geeksforgeeks.org/folder-structure-for-a-node-js-project/>






## W3Schools.com


W3Schools offers free online tutorials, references and exercises in all the major languages of the web. Covering popular subjects like HTML, CSS, JavaScript, Python, SQL,  [https://www.w3schools.com/MySQL/mysql\\_sql.asp](https://www.w3schools.com/MySQL/mysql_sql.asp)



## Supabase | The Open Source Firebase Alternative


Build production-grade applications with a Postgres database, Authentication, instant APIs, Realtime, Functions, Storage and Vector embeddings. Start for free.

 <https://supabase.com/>

 **Build in a weekend**  
**Scale to millions**

## HOW TO STRUCTURE YOUR BACKEND CODE IN NODE.JS.

When developing a Node.js application with Express.js, it's essential to maintain a well-structured codebase to ensure scalability...

 <https://medium.com/@datasciencenexus/how-to-structure-your-backend-code-in-node-js-c33a31bf458>



## npm: nodemon

Simple monitor script for use during development of a Node.js app.. Latest version: 3.1.9, last published: 2 months ago. Start using nodemon in your project by running `npm i

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

