

Learning Reflection

CSC105 Web Application Development (2/2022)

Back-End Development: Back-End Web Development Fundamentals

Due Date: Wednesday, 10 May 2023, 12:00 PM.

Instruction: Type your answers on this document and upload it as a .pdf file to CSCMS.

1. In your own words, explain what you learned from the previous lecture (April 27th, 2023) about the fundamentals of back-end web development.

Backend development: (often called the “server-side” development), The backend normally consists of an application, server and database. When you interact with a website by entering information, the information is saved in a database on a server. The results are then supplied to you as frontend code to be displayed on the site. **The reason that we not connect frontend to database directly** because if you connect your database directly from the frontend, you are exposing all your database credentials to the browser, and anyone can look up the code in the console and take it.

The Request URL is the link that you sent to request the data from server.


The Request Method is the method that Identify the action to be performed for a given resource from sever.

In this class as P.Mixko said in doc, we will focus on 5 HTTP methods

- 1.GET - Retrieve data from the server
- 2.PUT - Handles updates by replacing the entire entity
- 3.POST - Create data
- 4.PATCH - Only updates the fields that you give it
- 5.DELETE - Delete data

The Status Code is the HTTP response status codes indicate whether a specific HTTP request has been successfully completed. Responses that we focused in this class are grouped in three classes:

Successful responses (200 – 299) 

Client error responses (400 – 499) 

Server error responses (500 – 599) 

JSON is a text-based standard for encoding structured data that is based on JavaScript object syntax. It is often used for data transmission in web applications.

"node (file.js)" too run file you want to run.

Express JS is a Node.js framework for creating APIs that allow for communication via HTTP requests and responses.

"npm init" to get data in the package.json file that shows the information of the project that you have filled in the command line.

A file called 'package.json' contains descriptive and functional metadata about a project, such as its name, version, and dependencies. The file contains information that the npm package manager can use to identify the project and manage dependencies.

"npm install express" top install the 'ExpressJS' framework to your project.

"const express = require('express');" This line is you are importing 'express' framework to your project using require("express") and assigning it to the [express] variable.

"const app = express();" [App] is an instance of the Express application, which is created using the express() function.

Get method

The .get() method is a function provided by the app object, which sets up a route for HTTP GET (HTTP Methods) requests to a specific URL.

The second argument to the .get() method is a callback function, which will be called when the server receives a GET request for the specified URL. The callback function takes two arguments, req and res, which represent the request and response objects for the current HTTP request.

The .listen() method is a function provided by the app object, which starts the server and listens for incoming HTTP requests on a specified port number. The port variable is the number of the port on which the server listens. Also should be at the bottom of the code.

"npm install mysql2" is to install the mysql to your project.

"const mysql = require('mysql2');" assigning it to the [mysql] variable.


This is the database configuration of your database.

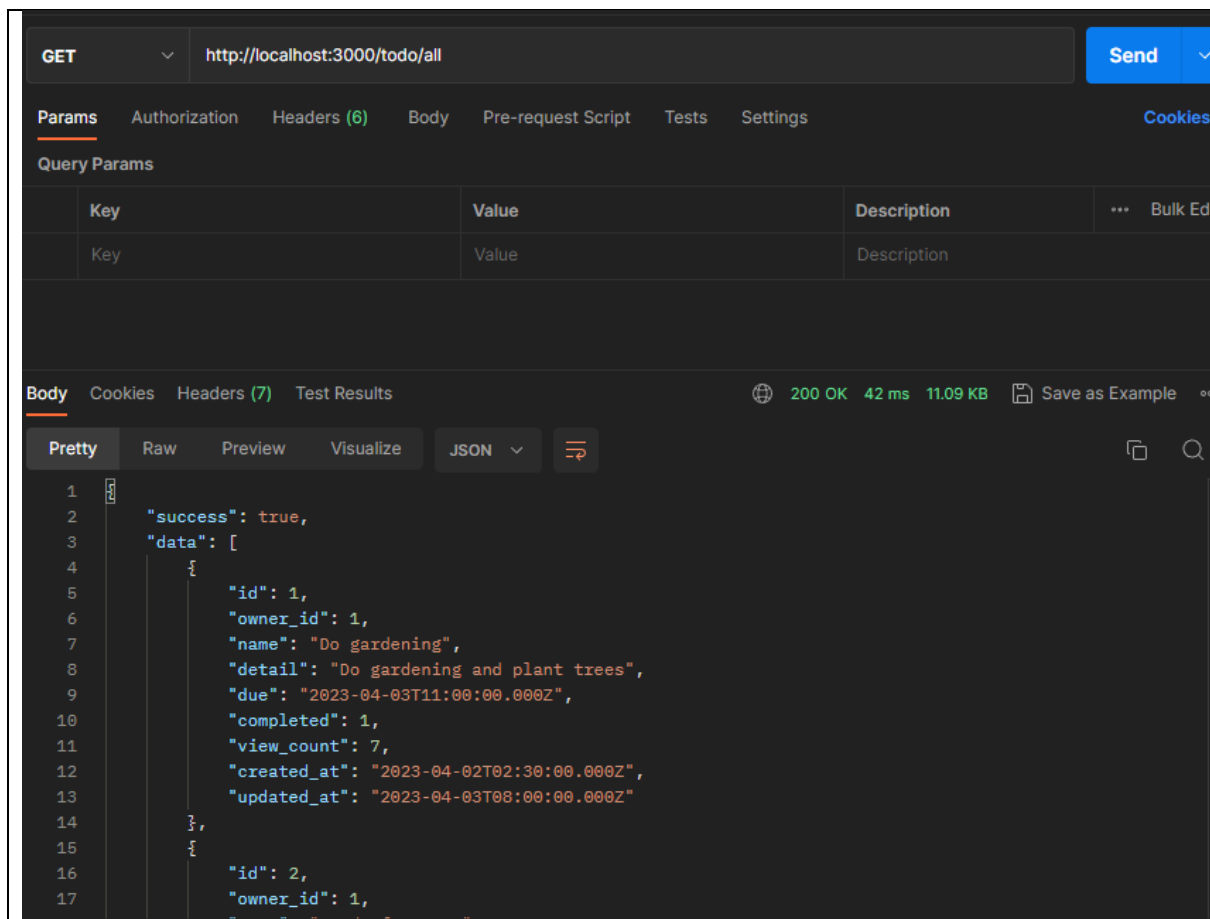
```
const connection = mysql.createConnection({
  host: "server2.mixkoap.com",
  port: "7777",
  user: "user",
  password: "password",
  database: "csc105-workshop",
});
```

```
// Connect to database
connection.connect();
```

In this code is get data from database if it is true.

```
app.get("/todo/all", (req, res) => {
  connection.query("SELECT * FROM items", (err, rows) => {
    // Check if cannot find the data in the database then return error
    if (err) {
      res.json({
        success: false,
        data: null,
        error: err.message,
      });
    } else {
      // Return data to the client if success
      return res.json({
        success: true,
        data: rows,
        error: null,
      });
    }
  });
});
```





(This one get all data in todo/all) ^

Query method

We will use the `.query()` method to insert your SQL query.

The second argument to the `.query()` method is a callback function `(err, rows)`, which will be called when the MySQL send the response. The callback function takes two arguments, `err` and `rows`, which represent the error of the database and rows objects for response data from MySQL database.

If we found an error, we will response the error message to the client.

else, we will return row which is data from the items table from the database.

Use Postman program to test API easier.

Localhost is 127.0.0.1.

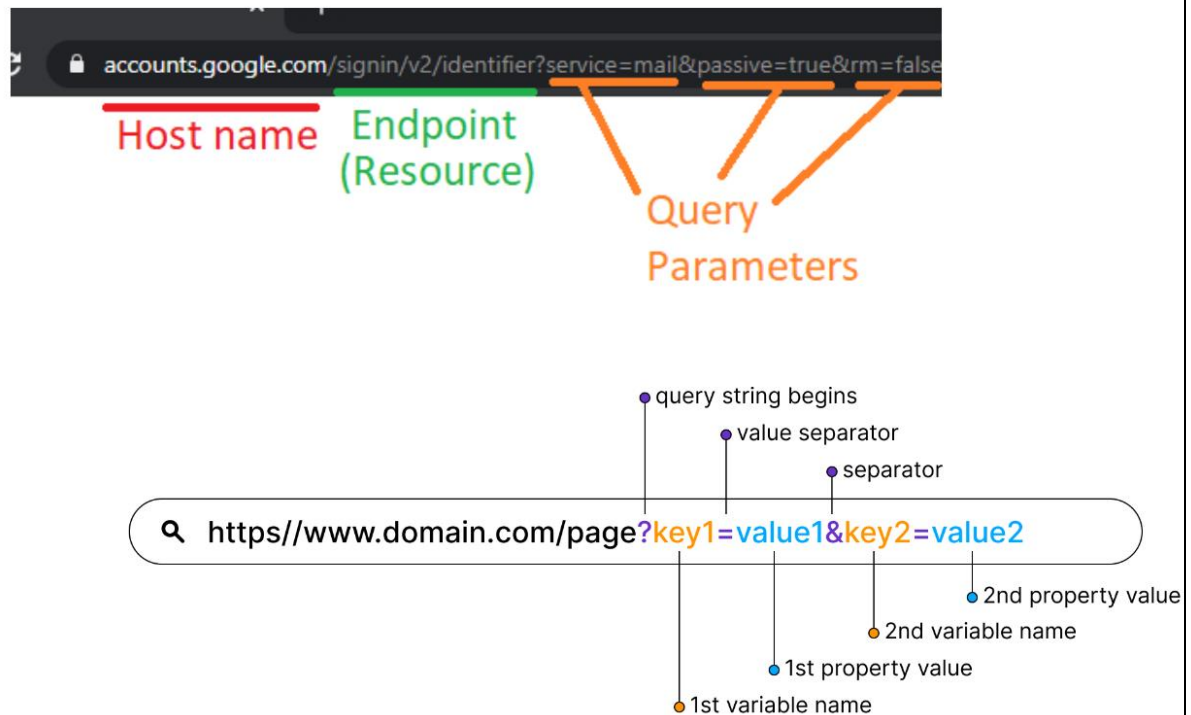
P.Mixko explain very clearly.

In some part, it is very hard to explain code without code.

That is why I will cap the screen instead.

Example of query parameter URL

9. Create an endpoint that queries a todo-list by `id` using `parameters` from URL.



This is how query parameter works in URL 🖱️

The screenshot shows a code editor and the Postman API client. In the code editor, the following JavaScript code is shown:

```
index.js > app.get("/todo") callback > connection.query() callback
51 app.get("/todo", (req, res) => {
52   // Assign the params as a variable
53   // https://medium.com/@joseph.pyram/9-parts-of-a-url-that-y
54   const todoId = req.query.todo_id; = 1
55 }
```

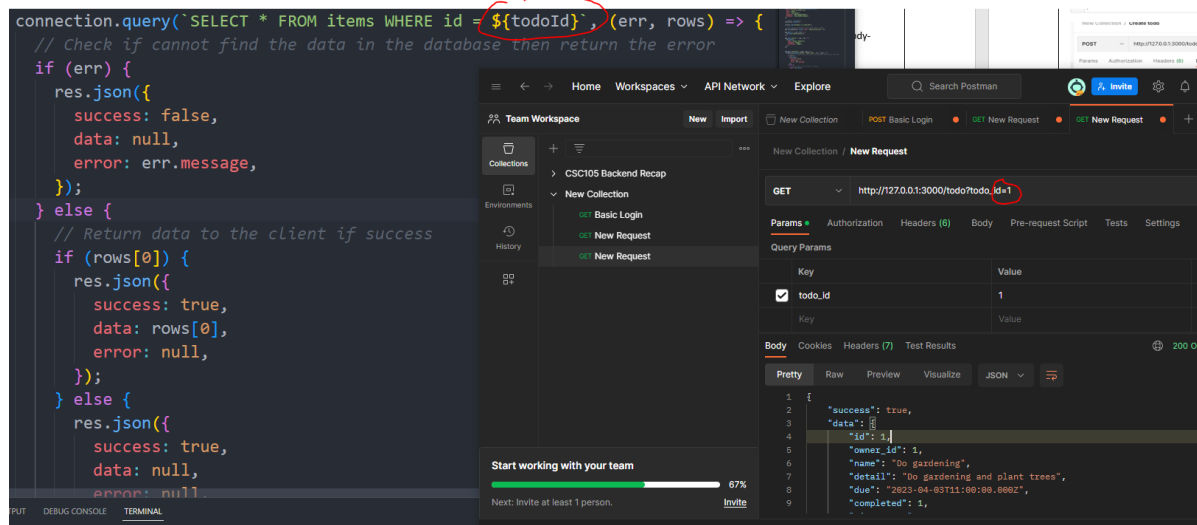
Handwritten annotations in the code editor:

- A green circle around `app.get("/todo",` is labeled **Endpoint** in green.
- A blue circle around `req.query.todo_id` is labeled **query string** in blue.
- A red circle around `todo_id` is labeled **variable name** in red.

Below the code, the Postman interface is shown. The URL bar contains `http://127.0.0.1:3000/todo?todo_id=1`. Handwritten annotations in Postman:

- A blue arrow points from the **query string** label to the `?todo_id=1` part of the URL.
- A red arrow points from the **variable name** label to the `todo_id` part of the URL.
- A green arrow points from the **Endpoint** label to the `/todo` part of the URL.

This one we query to get data from database. This one output will be only id 1 because we select id 1.



Post method

Before writing a request, we need to install the library 'body-parser' first.

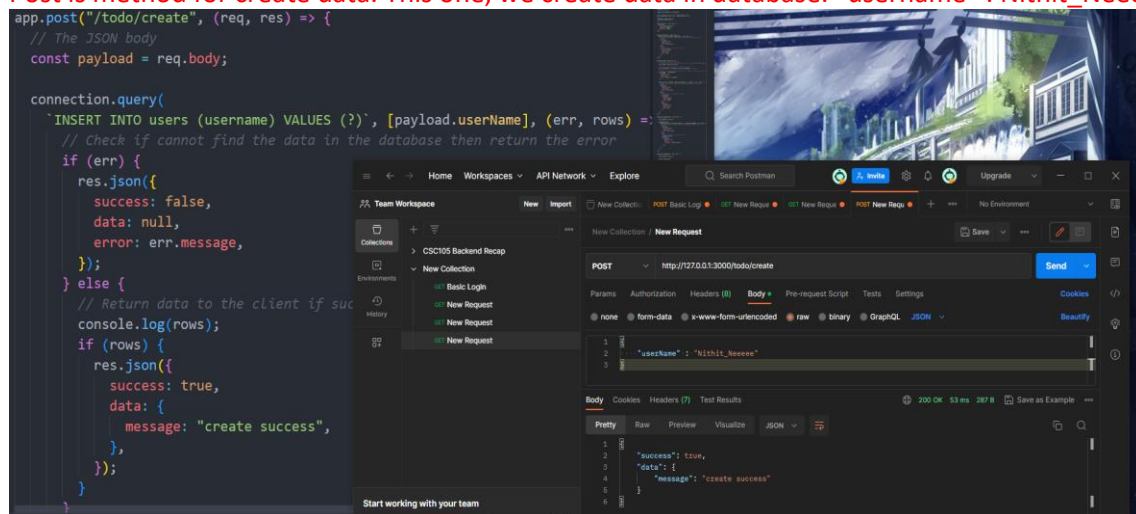
"npm i body-parser"

Because Express JS cannot read the JSON body request directly, So we need to install 'body-parser' first.

```
“const bodyParser = require("body-parser");”
```

```
// parse various different custom JSON types as JSON
app.use(bodyParser.json({ type: "application/json" }));
```

Post is method for create data. This one, we create data in database. “username” : Nithit_Neeeee”



For more clearly look at P.Mixko doc that I will cap screen and put in this doc.

- Add the `JSON` body

`http://127.0.0.1:3000/todo/create`

```

1 {
2   "userName": "Apisit Maneerat"
3 }

```

NEW COLLECTION / Create todo

POST `http://127.0.0.1:3000/todo/create`

Params Authorization Headers (8) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```

1 {
2   "userName": "Apisit Maneerat"
3 }

```

- Click `Send`

POST `http://127.0.0.1:3000/todo/create` Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```

1 {
2   "userName": "Apisit Maneerat"
3 }

```

Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize JSON

```

1 {
2   "success": true,
3   "data": {
4     "message": "create success"
5   }
6 }

```

Status: 200 OK Time: 77 ms Size: 287 B Save Response

- The response from your backend application.
- In the Database

Playground

```

1 select *
2 from users where username = 'Apisit Maneerat';

```

Output

id	username	created_at	updated_at
1	5 Apisit Maneerat	2023-04-26 18:11:58	2023-04-26 18:11:58

- For more detail

```

98
99 app.post("/todo/create", (req, res) => {
100   // The JSON body
101   const payload = req.body;
102 }

```

Search Postman

Working locally in Scratch Pad. Switch to a Workspace

Import Overview New Collection GET all todos GET Get todo by id POST Create todo

New Collection / Create todo

POST `http://127.0.0.1:3000/todo/create`

Params Authorization Headers (8) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```

1 {
2   "userName": "Apisit Maneerat"
3 }

```

```

const payload = req.body;

connection.query(
  `INSERT INTO users (username) VALUES (?)`, payload.userName, (err, rows) => {
    // Check if cannot find the data in the database then return the error
    if (err) {
      return res.json({

```

Patch method

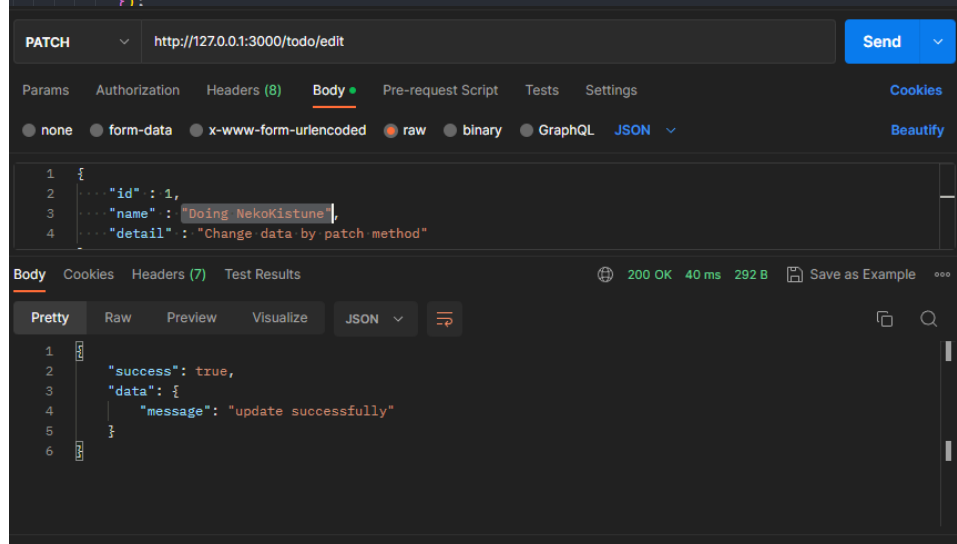
Create an endpoint that edit the data in the Database.

Edit data in database. For this one I change name to "Doing NekoKitsune"

```
app.patch("/todo/edit", (req, res) => {
  // The JSON body
  const payload = req.body;

  console.log(payload);

  connection.query(
    "UPDATE items SET name = ?, detail = ? WHERE id = ?", [payload.name, payload.detail, payload.id],
    (err, rows) => {
      // Check if cannot find the data in the database then return the error
      if (err) {
        res.json({
          success: false,
          data: null,
          error: err.message,
        });
      } else {
        // Return data to the client if success
        if (rows) {
          res.json({
            success: true,
            data: {
              message: "update successfully",
            },
          },
        );
        }
      }
    }
  );
});
```



```
{
  "success": true,
  "data": [
    {
      "id": 1,
      "owner_id": 1,
      "name": "Doing NekoKistune",
      "detail": "Change data by patch method",
      "due": "2023-04-03T11:00:00.000Z",
      "completed": 1,
      "view_count": 7,
      "created_at": "2023-04-02T02:30:00.000Z",
      "updated_at": "2023-04-03T08:00:00.000Z"
    },
    {
      "id": 2,
      "owner_id": 1,
      "name": "Study for exam",
      "detail": "No pain, no gain!"
    }
  ]
}
```

Delete method

Create an endpoint that delete the data from the links table in the database.

Delete data in database.

```
app.delete("/todo/delete", (req, res) => {
  // Assign the params as a variable
  const id = req.query.id;
  const todoId = req.query.todo_id;

  connection.query(
    `DELETE FROM links where id = ? AND todo_id = ?`, [id, todoId],
    (err, rows) => {
      // Check if cannot find the data in the database then return the error
      if (err) {
        res.json({
          success: false,
          data: null,
          error: err.message,
        });
      } else {
        if (rows) {
          res.json({
            success: true,
            data: {
              message: "delete successfully",
            },
          });
        }
      }
    }
  );
});
```

New Collection / New Request

Save

...

...

...

DELETE

http://127.0.0.1:3000/todo/delete?id=1&todo_id=1

Send

Params

Authorization

Headers (8)

Body

Pre-request Script

Tests

Settings

Cookies

Query Params

	Key	Value	Description	...	Bulk Edit
<input checked="" type="checkbox"/>	id	1			
<input checked="" type="checkbox"/>	todo_id	1			

Body Cookies Headers (7) Test Results

200 OK

36 ms

292 B

Save as Example

...

Pretty

Raw

Preview

Visualize

JSON

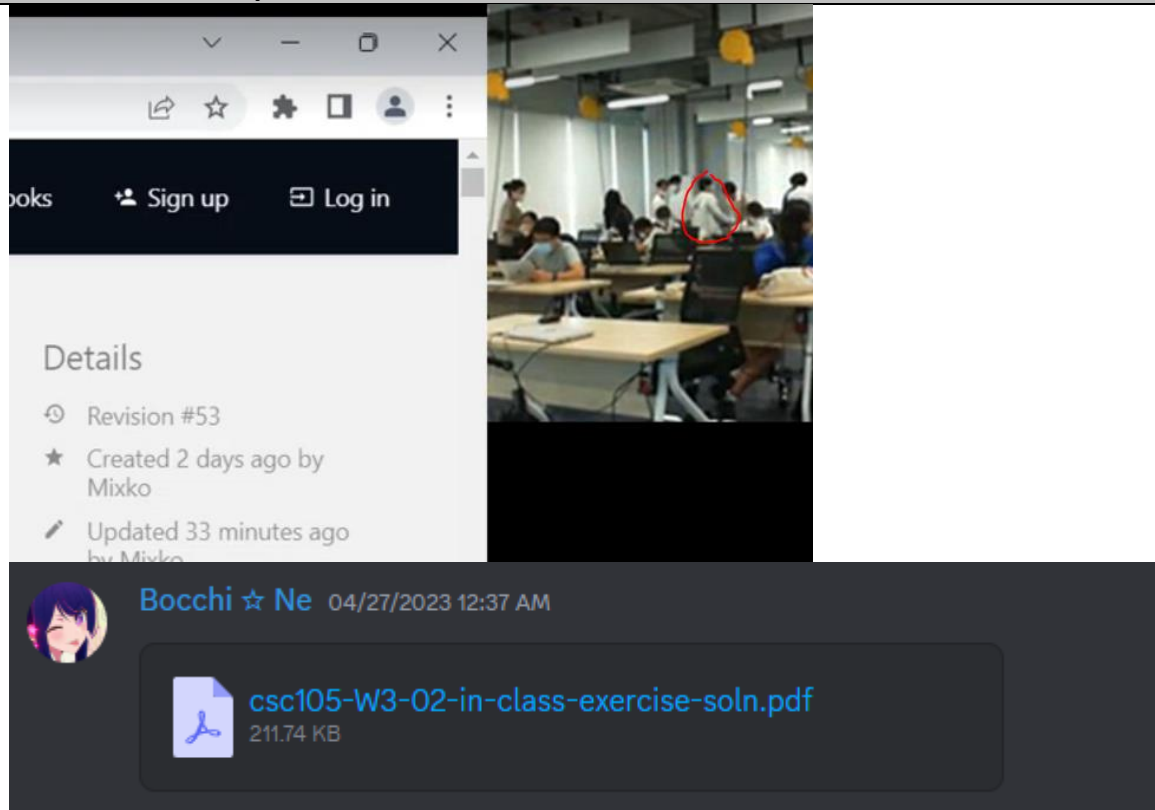
...

...

...

```
1  {
2    "success": true,
3    "data": {
4      "message": "delete successfully"
5    }
6  }
```


2. Provide evidence to demonstrate that you attended the previous lecture, such as code screenshots, pictures, etc.



Done on that day and send to myself another device at home before shutting notebook down.

I plan to continue when arrived at home.

```
File Edit Selection View Go Run ... index.js - A13o2_LearningReflection_Work - Visual Studio Code
EXPLORER
A13O2_LEARNINGREFLECT...
  node_modules
  index.js
  package-lock.json
  package.json
index.js
1 const express = require("express");
2 const app = express();
3 const mysql = require("mysql2");
4 const bodyParser = require("body-parser");
5 const port = 3000;
6
7 const connection = mysql.createConnection({
8   host: "server2.bsthun.com",
9   port: "6105",
10  user: "lab_12eyur",
11  password: "DRL1EsqIgnmfEn8c",
12  database: "lab_todo01_124aiqt",
13 });
14
15 // Connect to database
16 connection.connect();
17
18 app.delete("/todo/delete") callback > connection.query("DELETE FROM links where id = ? AND todo_id = ?") callback
19
20 Database is connected
21 Example app listening on port 3000
22 { id: 1, name: 'Do gardening', detail: 'Do gardening and plant trees' }
23 PS C:\Users\xXNeo\Desktop\CSC105_Code\CSC105_Code_Backend\A13o2_LearningReflection_Work> node index.js
24 Database is connected
25 Example app listening on port 3000
```

Criteria Rubric:

5	4	3	2	1
Student uses complete thoughts to analyze specific details that are directly related to the topic and includes multiple detailed examples from the topic to support answers.	Student uses complete thoughts to explain specific details that are directly related to the topic and includes a detailed example from the topic to support answers.	Student describes specific details that are related to the topic and includes an example to support answers.	Student describes details that are somewhat related to the topic OR Student uses a list of words/phrases to answer questions.	Student demonstrates a lack of understanding concepts and/or task.