University of Colombo School of Computing

SCS 2208 - Rapid Application Development

Lab Sheet 08 - Node.js + Express.js

What is Express.js ?

Express is a web application framework for **Node.js** that allows you to spin up robust APIs and web servers in a much easier and cleaner way.

Starting with Express.js

1. Check the prerequisites.
    - A local development environment for Node.js.

2. Initialize a New Project:
    - Create a new project folder on your computer.

    - Open a terminal or command prompt and navigate to the project folder.

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19044.3086]
(c) Microsoft Corporation. All rights reserved.

C:\Users\UCSC\OneDrive\Desktop\Projects>mkdir express-first-app

C:\Users\UCSC\OneDrive\Desktop\Projects>cd express-first-app

C:\Users\UCSC\OneDrive\Desktop\Projects\express-first-app>
```

    - Run the following command to initialize a new Node.js project and create a **package.json** file using **npm init -y** command.

```
C:\Users\UCSC\OneDrive\Desktop\Projects\express-first-app>npm init -y
```

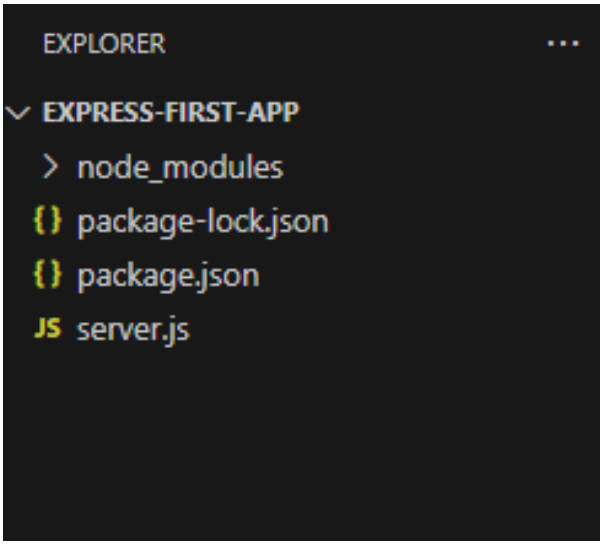
```
C:\Users\UCSC\OneDrive\Desktop\Projects\express-first-app>npm init -y
Wrote to C:\Users\UCSC\OneDrive\Desktop\Projects\express-first-app\package.json:

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

3. Install Express.js.

```
C:\Users\UCSC\OneDrive\Desktop\Projects\express-first-app>npm install express
```

4. Create an Express Application:
   - Create a new JavaScript file (e.g., **app.js** or **index.js** or **server.js**) in your project folder.

   - Open the JavaScript file in a code editor.



5. Import and Configure Express.
    - In your JavaScript file, require and initialize Express.

```
JS server.js
JS server.js > ...
1   const express = require('express');
2
3   const app = express();
4
```

6. Create Routes.
    - Define routes using the **app.get()**, **app.post()**, **app.put()**, **app.delete()**, and other HTTP methods provided by Express.

    - For example, to create a route that responds to GET requests at the root URL (/), you can use:

```
4
5   app.get('/', (req, res) => {
6     res.send('Successful response.');
7   });
```

7. Start the Server:
    - Add the following code at the end of your JavaScript file to start the server listening on a specific port (e.g., 3000):

```
9   const port = 3000;
10  app.listen(port, () => {
11    console.log(`Server is listening at http://localhost:${port}`);
12  });
```

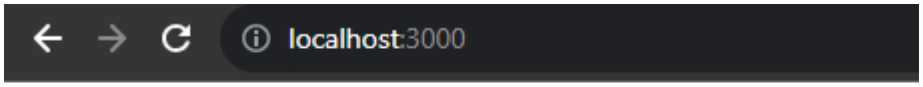
8. Run the Application.
    - Following code should be your final code.

```
JS server.js
JS server.js > ...
1   const express = require('express');
2
3   const app = express();
4
5   app.get('/', (req, res) => {
6     res.send('Successful response.');
7   });
8
9   const port = 3000;
10  app.listen(port, () => {
11    console.log(`Server is listening at http://localhost:${port}`);
12  });
13
```

    - Run your Express application by executing the **node server.js** command.

```
PS C:\Users\UCSC\OneDrive\Desktop\Projects\express-first-app> node server.js
Example app is listening on port 3000.
```

    - Open a web browser and go to **http://localhost:3000** to see your application in action.



Successful response.

Activities

1. Create a route that takes a name parameter and displays a personalized greeting message. For example, accessing **'/greet/John'** should display **"Hello, John!"**.

2. Build a user registration form that accepts name, email, and password. Display the submitted data.
   Hint: Use body-parser.

3. Create a basic RESTful API that returns a list of books when accessed at the **'/api/books'** route.
   Sample Array: [{ id: 1, title: 'Book 1' }, { id: 2, title: 'Book 2' }, { id: 3, title: 'Book 3' }];

4. Implement a basic authentication system that checks for a username and password in the request query parameters and displays a welcome message.
   Hint: Create an endpoint that expects username and password query parameters in the URL. Validate these parameters by checking them against predefined values

5. Implement error handling middleware that displays a custom error page when a route is not found (404).
   Hint: Use app.use()

6. Create a route that accepts query parameters for two numbers and a math operation (add, subtract, multiply, or divide). Return the result of the operation.
   Hint: Create a route that listens for GET requests with query parameters for num1, num2, and operation, then perform the specified math operation and send back the result.