
iX
Embrace Opportunity

Software Engineering Day 7

REST APIs & ExpressJS

Today’s Overview

- Introduction to Node.js.
- Setting up an Express server.
- Express Routes,
- CRUD
- Controllers
- Introduction to Postman

- Cheat Sheet:
  - iX Cheat Sheet
  - iX Cheat Sheet - Day 7

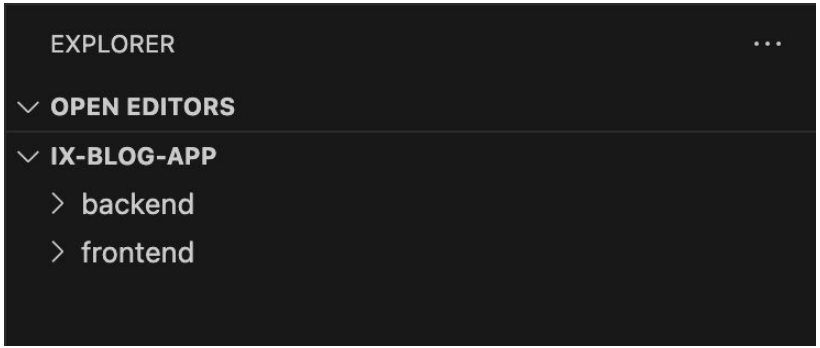
ExpressJS

Backend server in NodeJS

Backend setup

Express JS

- Create a new directory adjacent to your front end app called "backend":



- cd into you backend directory and run the following command. Press enter to skip through the set up prompts using the default set up.

```sh
npm init
```

Install Express

Express JS

- Run the following command to install Express JS.

```sh
npm install express --save
```

Notice the entry under dependencies in the package.json file.

Setup basic express server

Express JS

- Create a *src/* directory and create a *src/index.js* file in your backend work space directory and add the following boilerplate ExpressJS code:

```js
const express = require('express')
const app = express()
const port = 3000

app.get('/', (req, res) => {
  res.send('Hello World!')
})

app.listen(port, () => {
  console.log(`Example app listening on port ${port}`)
})
```

Run express server

Express JS

- Run the following command to run:

```sh
node .
```

Go to http://localhost:3000 in your browser and you should see "Hello World!" text on the screen.

Run express server with nodemon

Express JS

- Install nodemon to reload the server if a code change is made:

```sh
npm install --save-dev
```

- Update your scripts in *package.json* to run nodemon with dev keyword.

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

Run express server with nodemon

Express JS

- Re-run server with nodemon:

```sh
npm run dev
```

Routes

Express Routes

Basic routing in express

Express JS

- Routing refers to determining how an application responds to a client request to a particular endpoint, which is a URI (or path) and a specific HTTP request method (GET, POST, and so on).
- Each route can have one or more handler functions, which are executed when the route is matched.
- Route definition takes the following structure:

```js
app.METHOD(PATH, HANDLER);
```

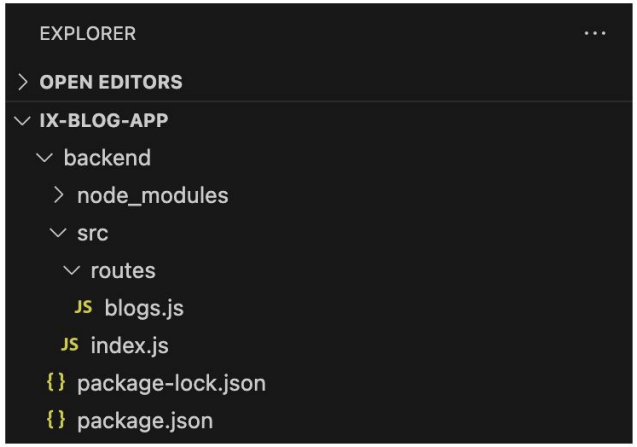
Where

- *app is an instance of express.*
- *METHOD is an HTTP request method, in lowercase.*
- *PATH is a path on the server.*
- *HANDLER is the function executed when the route is matched.*

Setup api routes

Express JS

- In *backend/src/* create a routes directory and add a file called *blogs.js*.



Setup api routes

Express JS

- Add the following boilerplate router code to *src/routes/blogs.js*:

```js
const express = require("express");
const router = express.Router();

router.get("/", (req, res) => {
  res.send("Return all blogs!");
});

module.exports = router;
```

Setup api routes

Express JS

- Update src/index.js with the following code:

```js
const express = require("express");
const app = express();
const port = 3000;

app.use(express.json());

app.use("/api/blogs", require("./routes/blogs"));

app.listen(port, () => {
  console.log(`IX blogging app listening on port ${port}`);
});
```

CRUD

Create, Read, Update and Delete

CRUD

Express JS

CRUD stands for **Create, Read, Update and Delete**. CRUD operations are foundational to most web and software applications, as they allow for the management and manipulation of data. This concept is widely used in programming and database management to ensure that software can interact with and manipulate the data layer within an application effectively.

CRUD routes

Express JS

- **Create**
  - **HTTP Method:** POST
  - **Description:** Adds a new resource.
  - **Express Route Example:**

```js
router.post("/", (req, res) => {
  res.status(200).json({ message: "Create new blog!" });
});
```

CRUD routes

Express JS

- **Read**
  - **HTTP Method:** GET
  - **Description:** Get all resources or get resource by ID.
  - **Express Route Example:**

```js
// GET ALL
router.get("/", (req, res) => {
  res.status(200).json({ message: "Return all blogs!" });
});

// GET BY ID
router.get("/:id", (req, res) => {
  res.status(200).json({ message: "Return blog by ID!" });
});
```

CRUD routes

Express JS

- **Update**
  - **HTTP Method:** PUT/PATCH
  - **Description:** Updates a resource by ID.
  - **Express Route Example:**

```js
router.put("/:id", (req, res) => {
  res.status(200).json({ message: "Update blog by ID!" });
});
```

CRUD routes

Express JS

- **Delete**
  - **HTTP Method:** DELETE
  - **Description:** Deletes a resource by ID.
  - **Express Route Example:**

```js
router.delete("/:id", (req, res) => {
  res.status(200).json({ message: "Delete blog by ID!" });
});
```

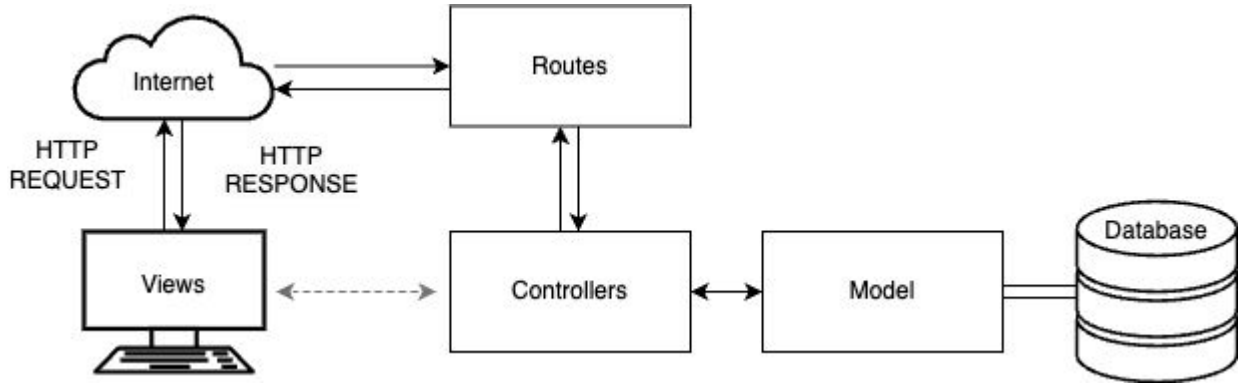
Controllers

Controllers in express and MVC

MVC

Express JS

- MVC or Model View Controller, is a **software architectural pattern** used for developing user interfaces. It divides the application into three interconnected components, allowing for efficient code organization and separation of concerns.



**Some server-side rendered frameworks like nextJS and Laravel the controller returns the populated view.*

MVC

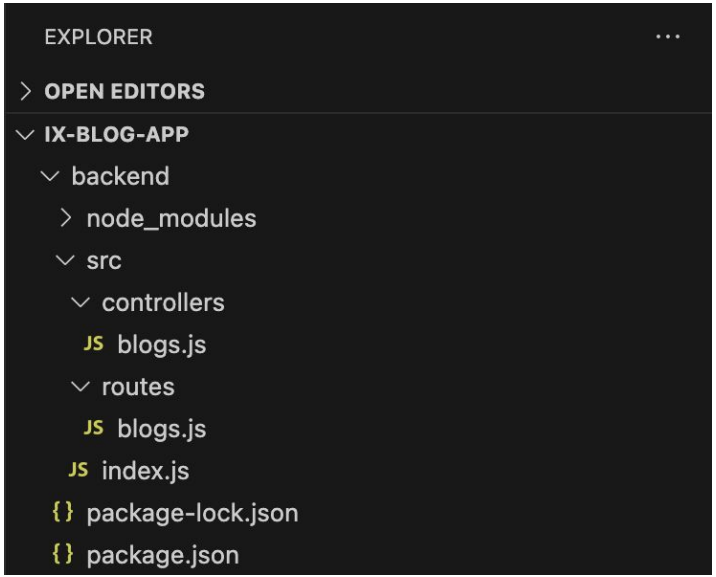
Express JS

- **Model:** Represents the data of the application. It directly manages the data of the application, responding to requests for information about its state (usually from the view) or instructions to change state (usually from the controller).
- **View:** Represents the UI of the application. It displays data (the model) to the user and sends user commands (events) to the controller. The view is responsible for rendering the data provided by the model in a format suitable for interaction, typically through a user interface.
- **Controller:** Acts as an interface between Model and View components. It processes all the business logic and incoming requests, manipulates data using the Model component, and interacts with the Views to render the final output. The controller receives input, optionally validates it, and then passes the input to the model.

Setting up controllers in express

Express JS

- Controllers are the Controller part of the MVC software architectural pattern.
- In *backend/src/* create a controllers directory and add a file called *blogs.js*



Setting up controllers in express

- Move routes code to *src/controllers/blogs.js:*

```js
const createBlog = (req, res) => {
  res.status(200).json({ message: "Create new blog!", data: [] });
};

const getBlogs = (req, res) => {
  res.status(200).json({ message: "Return all blogs!", data: [] });
};

const getBlog = (req, res) => {
  res.status(200).json({ message: "Return blog by ID!", data: [] });
};

const updateBlog = (req, res) => {
  res.status(200).json({ message: "Update blog by ID!", data: [] });
};

const deleteBlog = (req, res) => {
  res.status(200).json({ message: "Delete blog by ID!", data: [] });
};

module.exports = {
  getBlogs,
  getBlog,
  createBlog,
  updateBlog,
  deleteBlog,
};
```

Setting up controllers in express

- Update *src/routes/index.js* with the following code:

```js
const express = require("express");
const router = express.Router();
const {
  createBlog,
  getBlogs,
  getBlog,
  updateBlog,
  deleteBlog,
} = require("../controllers/blogs");

router.post("/", (req, res) => {
  createBlog(req, res);
});

router.get("/", (req, res) => {
  getBlogs(req, res);
});

router.get("/:id", (req, res) => {
  getBlog(req, res);
});

router.put("/:id", (req, res) => {
  updateBlog(req, res);
});

router.delete("/:id", (req, res) => {
  deleteBlog(req, res);
});

module.exports = router;
```

Postman

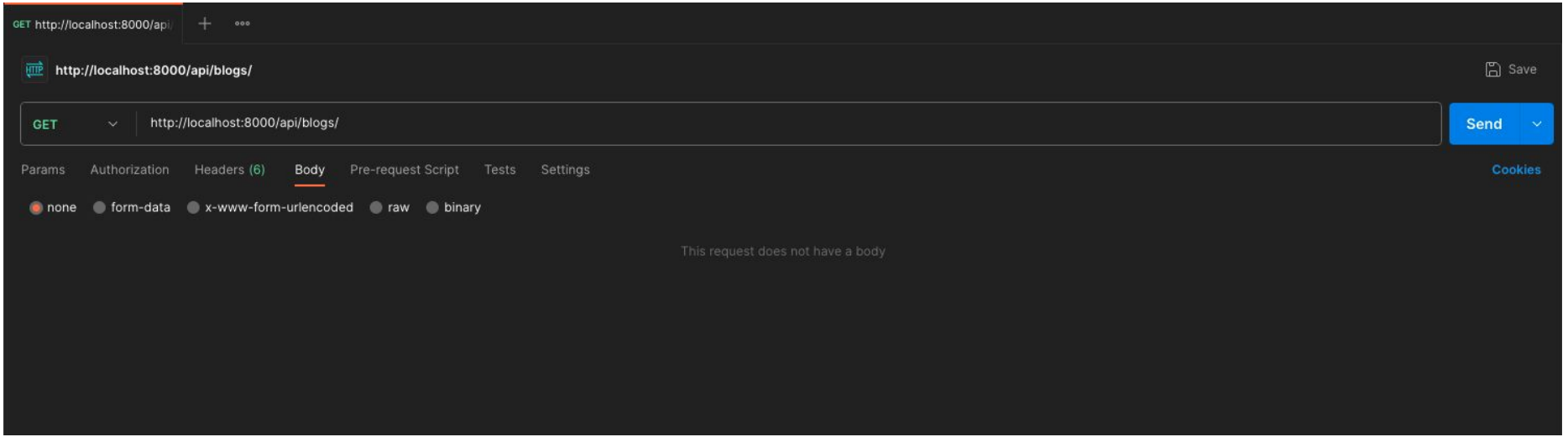
Controllers in express and MVC

Postman

- Download and install Postman.
- Postman is a popular software development tool used for testing API services. Postman provides a friendly GUI for sending requests to web servers and viewing responses, making it an invaluable tool for modern software development and testing.

Postman

- Test get blogs posts endpoint
  - Method: GET
  - URL: http://localhost:8000/api/blogs



Postman

- Expected Result:

Exercise

Exercise:

- Create CRUD routes and controllers for blogs and categories to fetch static data from the backend server.
- Use Postman to test each api route:
  - Create
  - Read
  - Update
  - Delete
- Each response should align with tutorial results.
- Push changes to project Github repository.

Homework

Apply what we have learned

Homework

- Create express backend server.
- Create routes and controllers to fetch static data from the backend server.
- Use postman to test the backend server.
- Push the updates to capstone GitHub mono repo.

Pre-Work:

- Install Mongo DB server and Mongo DB Compass on local machine.

Next Class

React Lifecycle Methods, Hooks, Context API and Routes