Building Web Services With Express

CS5610 : Web Development

Joydeep Mitra

Pre-class Activity

- Fork the repository https://github.com/CSE-316-Software-Development/learn-express
- Follow instructions in README.md to install dependencies.
- Create a new branch with today's date.
- Push all activities to the branch.

Quick Recap

- Why do we need a web server?
  - A dynamic server allows us to send custom content for a request based on inputs provided by a user.
- One way to build a web server is to use Node.js
  - provides a runtime environment to run JavaScript outside the browser.
  - has a single thread event loop architecture.
  - allows asynchronous program execution (promises and async/await).

What is Express?

- Express is a web framework built on top of Node. It is used to:
  - Write handlers for HTTP requests (routes).
  - Integrate with view rendering engines.
- Express is minimalist and flexible.
  - Small core; enough to build web applications.
  - Handles I/O effectively.
  - Offers numerous middleware modules that can be used off-the-shelf.

Key Concepts

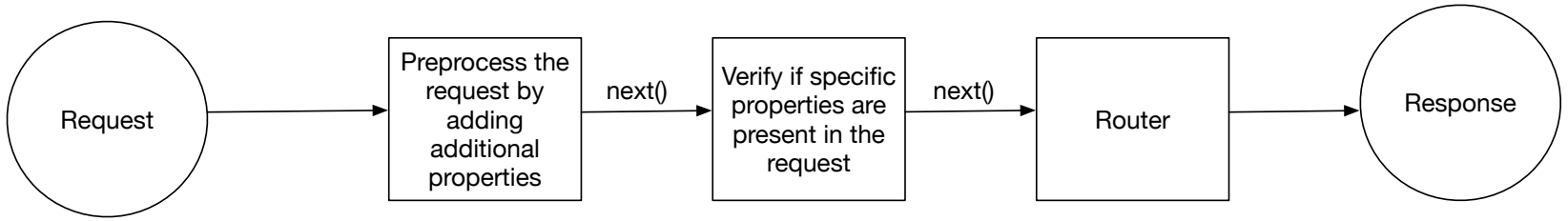
- Express allows us to
  - Define Routes
  - Use middleware
  - Write middleware

What are Routes?

- Routing is the process of determining how a server responds to a client's request.
- A route is defined as `app.METHOD(URI, HANDLER)`
  - **app** is the express object.
  - METHOD is an HTTP verb (GET, POST, PUT, DELETE) sent with the HTTP client request.
  - URI is the identifier for the HANDLER.
  - HANDLER is the function be executed for the HTTP request.
    - Processes incoming HTTP request.
    - Sends an appropriate HTTP response back to the client.

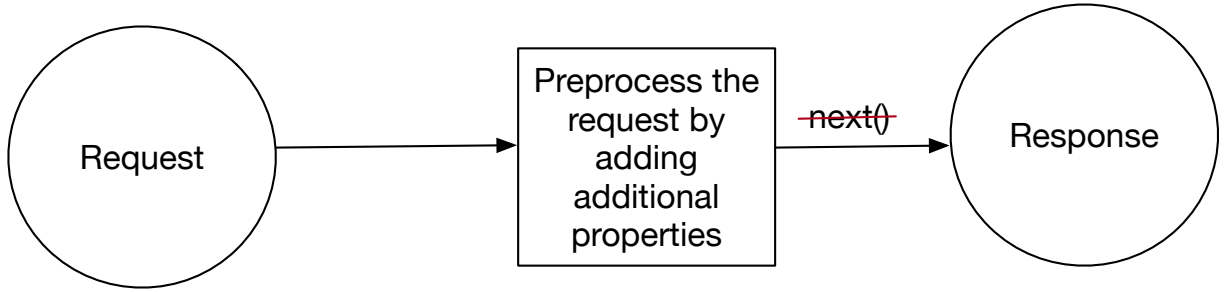
What is Middleware?

- Middleware functions are used to pre-process incoming requests and outgoing responses.
- They are defined as functions that take *request*, *response*, and a *next* handler.
- The *next* handler when invoked triggers the succeeding middleware function in the stack.



Middleware Functions

- If during preprocessing we encounter an error, we can return from the middleware without reaching the router.



Express Middleware Functions

- A few popular examples:
  - **cors**. Used to specify cross-origin request policy. Preprocess incoming request and verify if its origin property is allowed access. https://expressjs.com/en/resources/middleware/cors.html
  - **express.json**. Used to preprocess incoming request and parse associated json. https://expressjs.com/en/5x/api.html
  - Full list of express middleware: https://expressjs.com/en/resources/middleware.html
- We can also define our own middleware.

Using Middleware

- A middleware can be used
  - for all routes in the application.
    ```
    app.use((req, res, next) => { logic })
    ```
  - or for specific routes in the application.
    ```
    app.use(PATH,(req, res, next) => { logic })
    ```

Example

- Suppose we have a server, which stores user information in a file (see *data/users.json*)
- The code in *server.js* defines REST services to query and modify *data/users.json.*
- In *server.js* observe the following:
  - The HTTP routes and their corresponding handlers.
  - How are the requests processed?
  - What do the handlers return?
  - Are there any middleware functions?
  - What is their purpose?
- The component in *client/src/components* sends HTTP requests to this server using a library called **axios**.

For You to Do

- Express endpoints accept input parameters via the route pattern “route/:input”. E.g.,
  - express.get(‘home/:name’, (req, res) => {

    req.params.name // input parameters are in the params property

    })
- The client has a feature that allows us to search for specific users with their username and see their email.
- This feature is not working because of a missing HTTP service. Extend *quiz/server.js* with the necessary service.
  - Use the URI `/read/username/:name`

Express Router

- **express.Router()** is a mini-application within an Express application.
  - helps group routes and middleware together
  - provides a more organized and modular way to arrange our web services.
- **Purpose**: improve code maintainability and scalability by organizing routes and related logic into separate modules.
- **Usage**:
  1. instantiate a router using **express.Router()**
  2. then define routes and middleware on it using methods like .get(), .post(), .use(), etc.
  3. These routes can then be mounted onto the main Express application using **app.use()**.

Example

- The service in *router-ex-server.js* demonstrates an example of how to use express.router(). Observe the following?
  - The *birds.js* script.
  - How *app.use()* is used to modularize the *router-ex-server.js*?
  - Send an HTTP GET request from the browser to the endpoints:
    - localhost:8001/birds
    - localhost:8001/birds/about

For You to Do

- The code in *quiz/server.js* which you extended with an additional service is monolithic.
- We want to modularize it using *express.router()* as follows:
  - The services to read all users and a specific user must be within the */read* URI.
  - The services to add new users must be within the */write* URI.
  - Complete the code in *quiz/readUsers.js*, *quiz/writeUsers.js,* and *quiz/server4.js* to this end.

Points to Note

- Express.js is a framework that makes it convenient to write REST-based web services.
  - We can define endpoints corresponding to HTTP methods.
  - We can write and use middleware to handle requests and responses.
- On the front-end side we write glue code (e.g., Axios) to connect with our web services.
- Express has support for integrating a back-end database. More on that later!

Additional Reading

- https://expressjs.com/en/guide/routing.html
- https://expressjs.com/en/guide/writing-middleware.html
- https://expressjs.com/en/guide/using-middleware.html