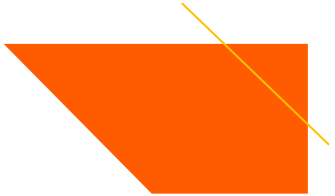
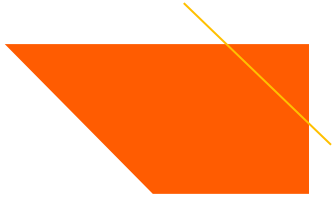
Web Application Development

COSI 152A



Main Point Preview

- In this lesson, you will learn how to build routes and middleware functions with Express.js.
- You use middleware functions to work with Express.js in analyzing request body contents.
- At the end of the lesson, you will learn about MVC and see how routes can be rewritten to use controllers in your application.



Routes in Express.js

Routes in Express.js

- A route definition starts with the app object, followed by a lowercase HTTP method and its arguments:
  - the route path and callback function
- A route handling POST requests to the /contact path should look like:

```
app.post("/contact", (req, res) => {
  res.send("Contact information submitted successfully.");
});
```

Handle requests with the Express.js post method.

Parameterized Routes

- Express.js lets you write routes with parameters in the path.
  - parameters are a way of capturing data through the request
  - route parameters have a colon (:) before the parameter and can exist anywhere in the path.
- An example of a route with parameters:

```
app.get("/items/:vegetable", (req, res) => {
 res.send(req.params.vegetable);
});
```

Respond with path parameters.

Building Routes with Express.js

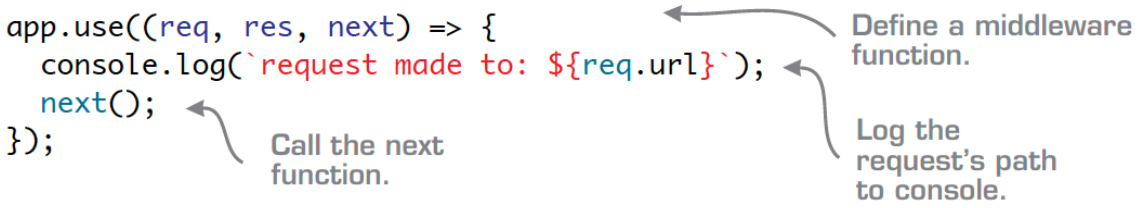
- Initialize a new project called express_routes.
- Install Express.js and add the code to require and instantiate the Express.js module.
- Then create a route with parameters and respond with that parameter.

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

app.get("/items/:vegetable", (req, res) => {
  let veg = req.params.vegetable;
  res.send(`This is the page for ${veg}`);
});

app.listen(port, () => {
  console.log(`Server running on port: ${port}`);
});
```

Add a route to get URL parameters.

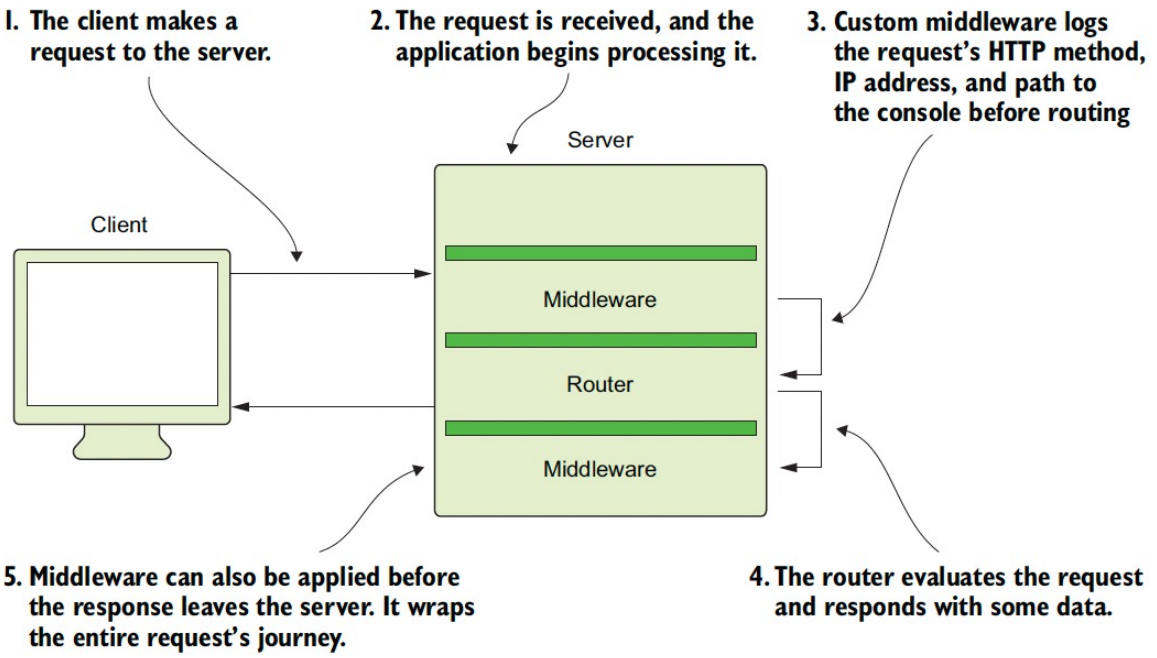
Building Routes with Express.js

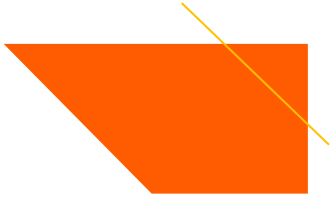
- Log the path of every request by adding a log message to every route or by creating the middleware function like this:



- This code defines a middleware function with an additional next argument
  - It logs the request's path to the terminal console and then calls the next function to continue the chain in the request-response cycle.

The role of middleware functions





Analyzing Request Data

- You have two main ways to get data from the user:
  - Through the request body in a POST request
  - Through the request's query string in the URL

Request Data through POST

- Express.js makes retrieving the request body easy with the body attribute.
  - add express.json and express.urlencoded to app instance to analyze incoming request bodies.
- Use req.body to log posted data to the console.

```
app.use(
  express.urlencoded({
    extended: false
  })
);
app.use(express.json());

app.post("/", (req, res) => {
  console.log(req.body);
  console.log(req.query);
  res.send("POST Successful!");
});
```

Tell your Express.js application to parse URL-encoded data.

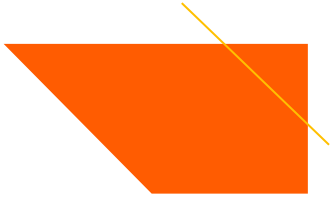
Create a new post route for the home page.

Log the request's body.

Request Data through POST

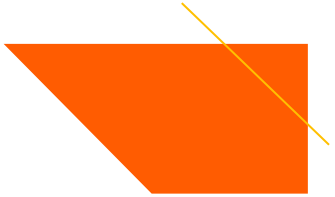
- Test the code by submitting a POST request to http:// localhost:3000, using the curl command:

```
curl --data "first_name=Hadi&last_name=Mohammadi" http://localhost:3000
```



Request Data through URL

- Another way to collect data is through the URL query string.
- Express.js lets you collect values stored at the end of your URL's path, following a question mark (?)
  - these values are called query strings
  - they are often used for tracking user activity on a site and storing temporary information about a user's visited pages.



Request Data through URL

- Examine the following sample URL:

  `http://localhost:3000/?cart=3&pagesVisited=4&utmcode=837623`

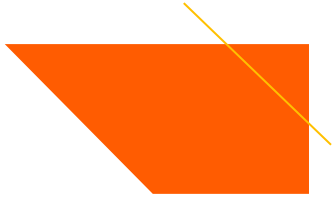
- This URL might be passing information about:
  - number of items in a user's shopping cart
  - number of pages they've visited
  - a marketing code to let the site owners know how this user found your app in the first place.

Request Data through URL

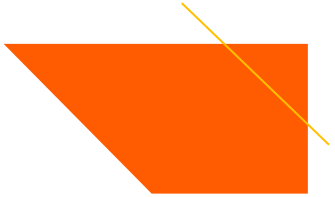
- To see the query strings on the server:
  - add `console.log(req.query);` to your middleware function in main.js
- Now try visiting the same URL.

  `http://localhost:3000/?cart=3&pagesVisited=4&utmcode=837623`

- You should see `{ cart: "3", pagesVisited: "4", utmcode: "837623" }` logged to your server's console window.

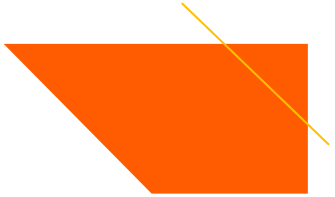


MVC Architecture



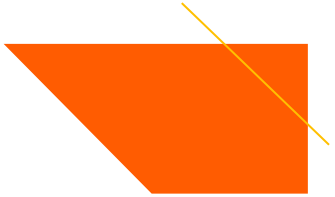
MVC Architecture

- Express.js's request-response cycle is based on an application architecture known as MVC.
- MVC architecture focuses on three main parts of your application's functionality:
  - Models
  - Views
  - Controllers



MVC Architecture

- **Models:** Classes that represent object-oriented data in your application and database.
- **Views:** Rendered displays of data from your application.
- **Controllers:** The glue between views and models.
  - controllers perform most of the logic when a request is received.
  - they determine how request body data should be processed
  - controllers also determine how to involve the models and views

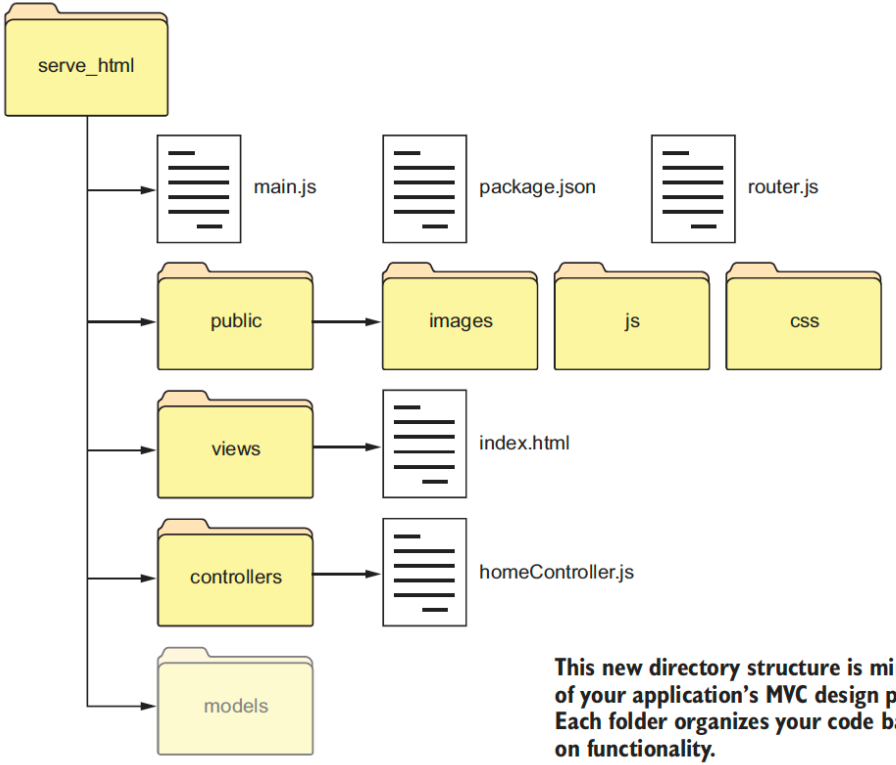


Moving to MVC design pattern

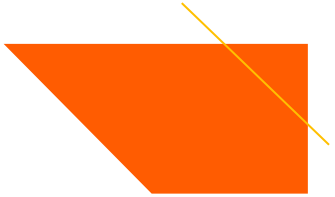
- To follow the MVC design pattern, move your callback functions to separate modules that reflect the purposes of those functions.
  - Callback functions related to user account creation, deletion, or changes, for example, go in a file called usersController.js within the controllers folder.
  - Callback functions for routes that render the home page or other informational pages can go in homeController.js by convention.

MVC based project structure

- The file structure that your application will look like on MVC base:



This new directory structure is mindful of your application's MVC design pattern. Each folder organizes your code based on functionality.



Restructuring

- To restructure your express_routes application to adhere to this structure, follow these steps:
  1. Create a controllers folder within your project folder.
  2. Create a homeController.js file within controllers.
  3. Require your home controller file into your application by adding the following to the top of main.js:

```
const homeController = require("./controllers/homeController");
```

Restructuring

4. Move your route callback functions to the homeController and add them to that module's exports object. Your route to respond with a vegetable parameter, for example, can move to your home controller:

```
exports.sendReqParam = (req, res) => {
  let veg = req.params.vegetable;
  res.send(`This is the page for ${veg}`);
};
```

Create a function to handle route-specific requests.

In homeController.js, you assign exports.sendReqParam to the callback function.

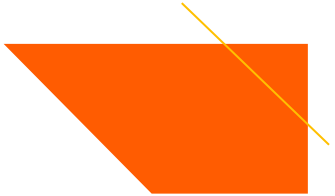
Restructuring

5. Back in main.js, change the route to look like:

```
app.get("/items/:vegetable", homeController.sendReqParam);
```

Handle GET requests to "/items/:vegetable".

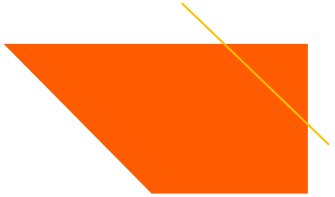
When a request is made to this path, the function assigned to sendReqParam in the home controller is executed.



Restructuring

6. Apply this structure to the rest of your routes and continue to use the controller modules to store the routes' callback function. You can move your request-logging middleware to a function in the home controller referenced as logRequestPaths, for example.
7. Restart your Node.js application and see that the routes still work.

Your Express.js application is now taking on a new form with MVC in mind!



Thank You!