COSI 152A

Web Application Development

Express.js

Main Point Preview

- It is time to take your application to a more robust and professional level by using a web framework and dynamic content.
- In this lesson you will get familiar with Express.js and how to configure a new Node.js application.
- You will get an overview of how a web framework helps you develop an application.

Web Framework

- A web framework is a predefined application structure and a library of development tools.
  - it makes building a web application easier and more consistent.
- A web framework in Node.js is a module or library that provides a structured and organized way to build web applications and services
  - easily build and customize your application.
  - no need to build certain features from scratch

Node.js Web Frameworks

- There are several web frameworks and libraries available for Node.js that help developers build web applications efficiently.
- Here are some of the commonly used Node.js web frameworks:
  - Express.js
  - Koa.js
  - Total.js
  - Hapi.js
  - Sails.js

Express.js

- Express.js increases development speed and provides a stable structure on which to build applications.
- Like Node.js, Express.js offers tools that are open-source and managed by a large online community.

Why Express.js ?

- It is the most used framework in the Node.js community
  - ensures that you find the support you need
  - newer frameworks
- It provides methods and modules to assist with handling requests, serving static and dynamic content, connecting databases, and keeping track of user activity.
- It is used by new and professional Node.js developers alike, so if you feel overwhelmed at any time, know that thousands of others can help you overcome your development obstacles.

Other Node.js web frameworks

Node.js frameworks to know

Node.js frameworks	Description
Koa.js	Designed by developers who built Express.js with a focus on a library of methods not offered in Express.js (http://koajs.com/)
Hapi.js	Designed with a similar architecture to Express.js and a focus on writing less code (https://hapijs.com/)
Sails.js	Built on top of Express.js, offering more structure, as well as a larger library and less opportunity for customization (https://sailsjs.com/)
Total.js	Built on the core HTTP module and acclaimed for its high-performance request handling and responses (https://www.totaljs.com/)

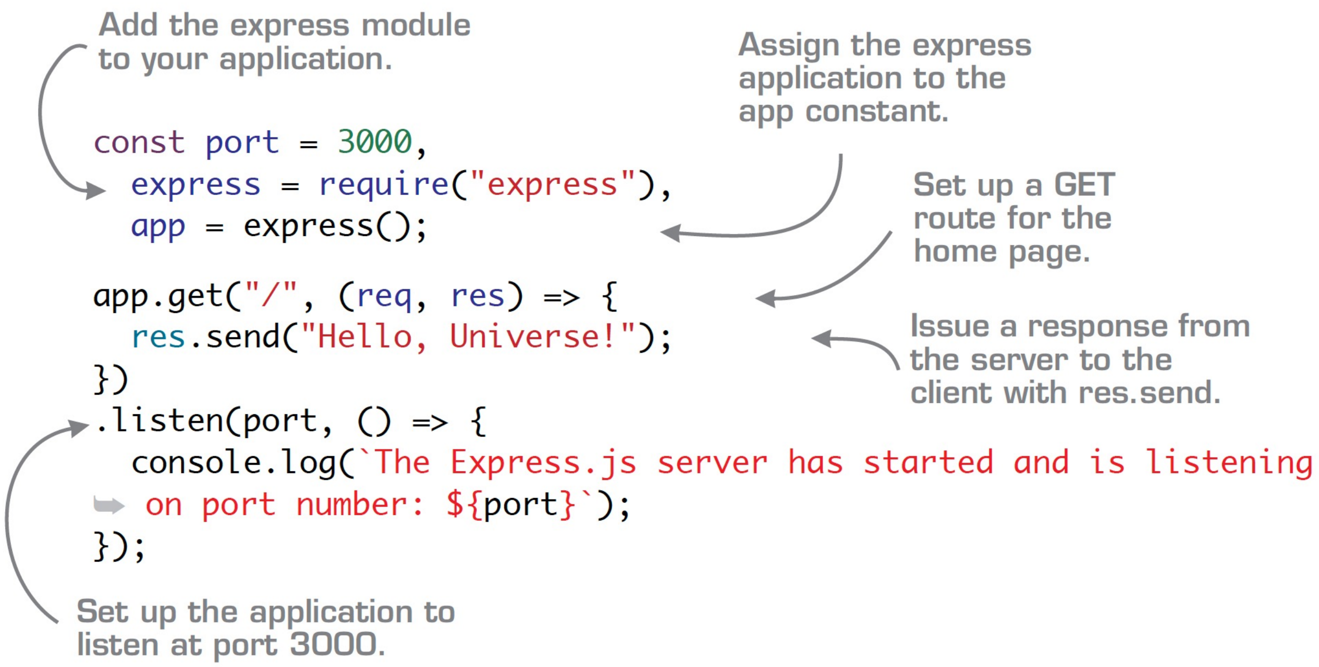
Building First Express Application

- Initialize your application by creating a new project directory
  - first_express_project
- Download and install express.js by running the following command:

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

Building First Express Application

- Create main.js file with the following code

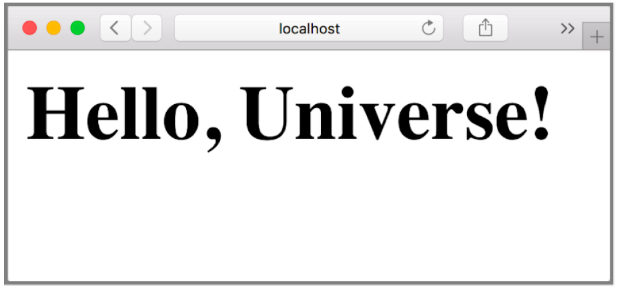

Add the express module to your application.
Assign the express application to the app constant.
Set up a GET route for the home page.
Issue a response from the server to the client with res.send.
Set up the application to listen at port 3000.

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

app.get("/", (req, res) => {
  res.send("Hello, Universe!");
})
.listen(port, () => {
  console.log(`The Express.js server has started and is listening
➥ on port number: ${port}`);
});
```

Run Your Application!

- Run `node main` in your terminal
- Open a browser tab and visit localhost:3000


localhost

Hello, Universe!

Congratulations, your first express web server is responding!!!!!!

nodemon Package

- To see your application server code changes in effect, every time you need to restart the server in terminal.
- You can use nodemon package to start your application the first time and automatically restart it when application files change.
  - To install nodemon globally, run:

```
npm i nodemon -g
```

  - You may need to run command in terminal as an administrator.

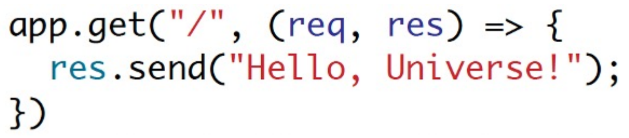
```
sudo npm i nodemon -g
```

  - To run your project using nodemon package:

```
nodemon main.js
```

Working Around Express.js

- Express.js provides a way to listen for requests to specific URLs and respond by using a callback function.



```
app.get("/", (req, res) => {
  res.send("Hello, Universe!");
})
```

Working Around Express.js

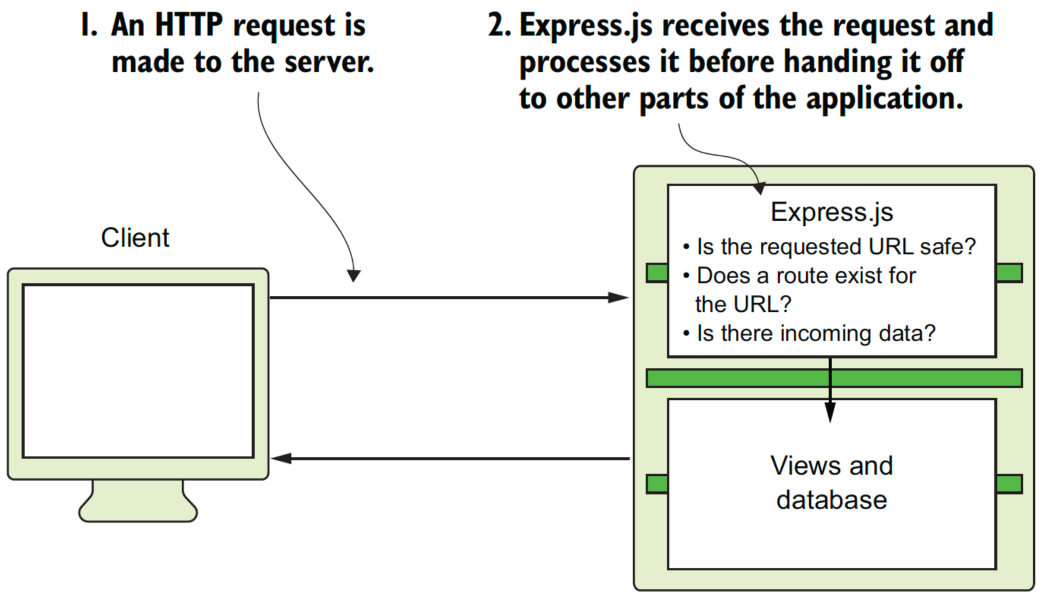
- Express.js operates through functions considered to be middleware because they sit between the HTTP requests and the application's route handlers (the callback functions).
- Middleware is a general term applied to code that assists in listening for, analyzing, filtering, and handling HTTP communication before data interacts with application logic.

```
app.get("/", (req, res, next) => {
    console.log(`Received a request`);
    next();
  },
  (req, res) => {
    res.send("Hello, everyone!");
  });
```

```
app.use((req, res, next) => {
    console.log(`Received a request`);
    next();
  });
app.get("/", (req, res) => {
    res.send("Hello, everyone!");
  });
```

- You can think of middleware as being like a post office.

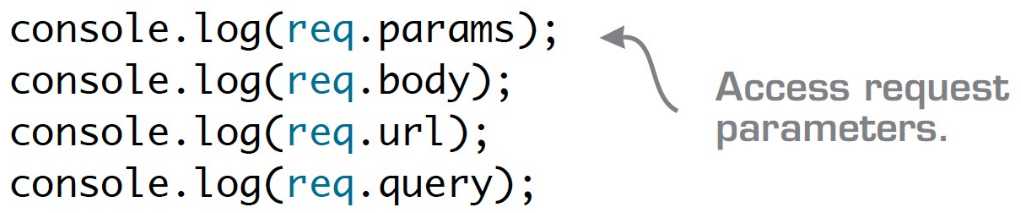
Working Around Express.js


1. An HTTP request is made to the server.
2. Express.js receives the request and processes it before handing it off to other parts of the application.
Client
Express.js
• Is the requested URL safe?
• Does a route exist for the URL?
• Is there incoming data?
Views and database

Express.js stands between the HTTP requests and your application code.

Working Around Express.js

- You get the same request and response objects, containing a lot of rich information about the sender and its contents.
- Express.js provides simpler ways to pull and log data from the request body.
- Add the following code to your GET route handler in main.js.



```
console.log(req.params);
console.log(req.body);
console.log(req.url);
console.log(req.query);
```

Thank You!