Module 4: Fullstack Review; About npm; Building a backend with Express.js; RESTful Routing

Edgardo Molina, PhD | Head Instructor

CUNY Tech Prep 2018-2019

Overview

- Fullstack Review
- About `npm`
- Building a backend with Express.js
- RESTful Routing

Fullstack Review: Frontend vs Backend

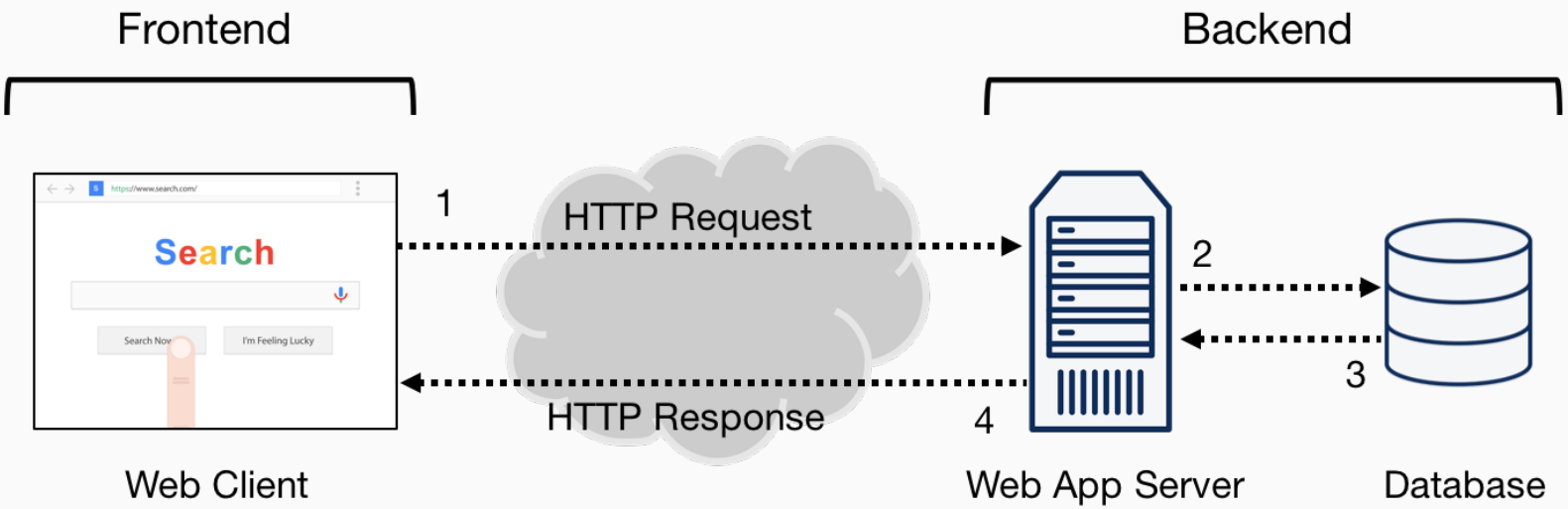

Figure 1: Frontend and Backend

The Frontend

What is it for:

- controls what the user *sees* and *interacts* with
- calls the backend to *send* and *receive* data

The Frontend

Frontend Technologies

- **Only Choice**: HTML, CSS, JavaScript
- Libraries and Frameworks
    - React.js, Angular.js, Vue.js, jQuery

The Backend

What is it for:

- Sending data to clients
- Receiving data from clients
- Processing data for the clients
- *Connecting* clients!

The Backend

Backend Technologies

- Most programming languages: JavaScript, Java, Ruby, Python, C#, etc
- Frameworks
    - Express.js, Ruby on Rails, Django, Spring

Frontend vs Backend performance concerns

Frontend

- the code runs on the client hardware (web browsers, mobile, etc)
- the code runs separately for each user
- has to be *responsive* to the user events and render the screen *quickly*

Backend

- the code runs on the server
- the code handles multiple requests and users from one instance
- has to be *responsive* to many users simultaneously, processing and transmitting data *quickly*

About npm

What is `npm`?

NPM: Node Package Manager (https://www.npmjs.com/)

`npm` is initially installed with Node.js.

We use `npm` to:

- install JavaScript libraries and tools
- create JavaScript projects/packages
- publish our own packages

Installing packages

Global tools

We can install globally available tools using the `-g` flag

i.e. `npm install -g create-react-app`

Updating `npm`

```
npm install -g npm
```

Creating projects

We use `npm init` to create a Node.js/JavaScript project.

This creates a `package.json` file in our directory

`package.json` is used to:

- list our project package *dependencies*
- add *scripts* for our project
- manage how packages are updated

package.json

When we download someone else's node.js project, we usually run

```
npm install
```

This command looks for the `package.json` file to install the necessary dependencies. These are stored within the project in the `node_modules/` directory. We can then run the project, this is typically done with:

```
npm start
```

This is only a convention, the `start` script must be defined in the `package.json` file.

Creating projects

Adding dependencies to your project

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

To install as a dependency

```
npm install --save-dev sequelize-cli
```

To install a dependency for development purposes only. (This is not installed in production)

Publishing packages

You can publish packages if you want to share your code with the community.

For more details see:
https://docs.npmjs.com/getting-started/publishing-npm-packages

Git and package.json

Add to git:

- `package.json`
- `package-lock.json`

Ignore from git:

- `node_modules/`

  This can always be recreated from the package.json files.

.gitignore

Always use a `.gitignore` file

You can find templates here: https://github.com/github/gitignore

Building a backend with Express.js

Outline

Building an App in Express.js from Scratch

- Introduction to the Express.js Framework
    - Routing
    - Route Parameters
    - Query Parameters
    - Body Parameters
- RESTful Routing (GET, POST, PUT, DELETE)

Introduction to the Express.js Framework

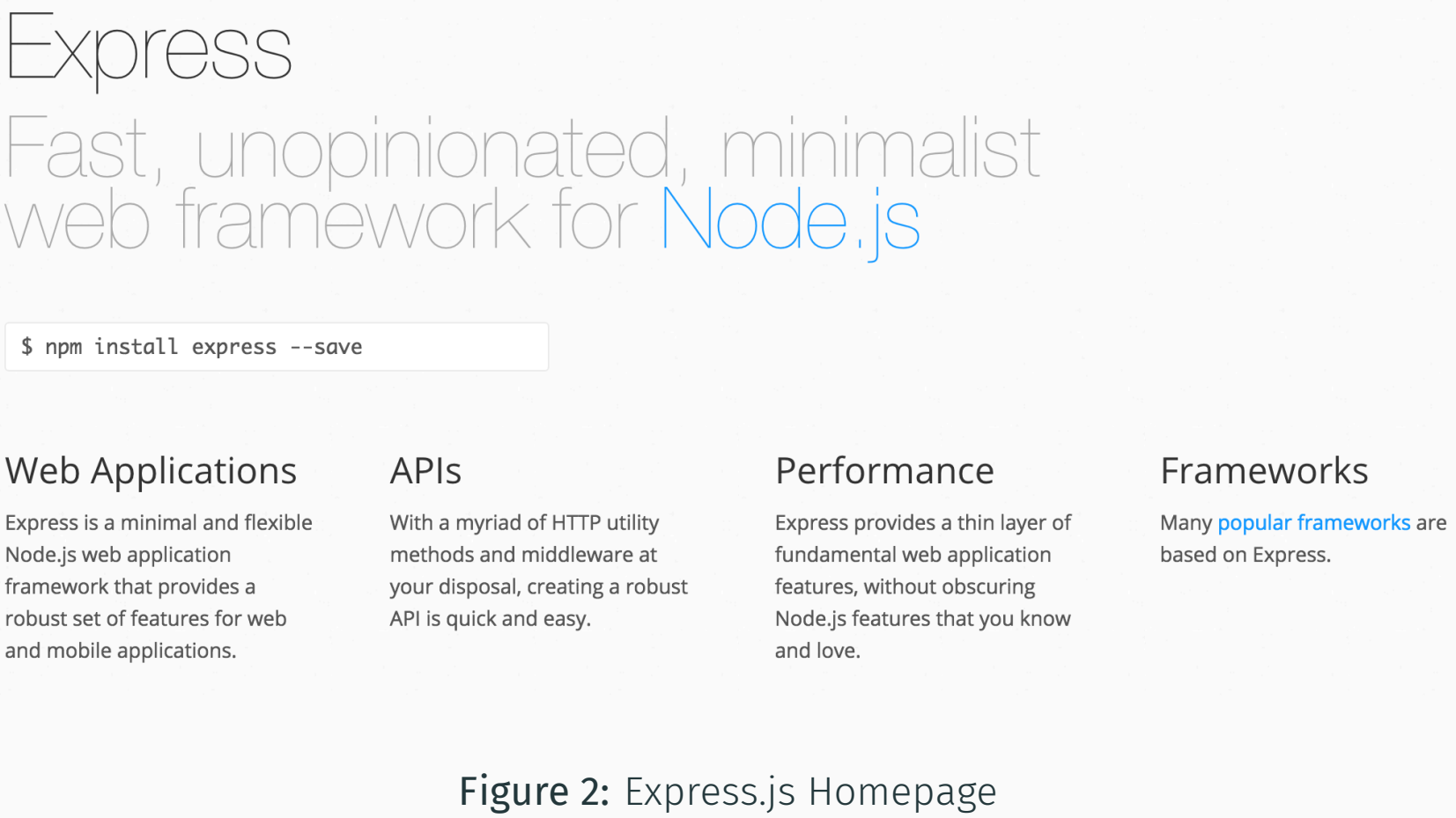

Figure 2: Express.js Homepage

What is Express.js?

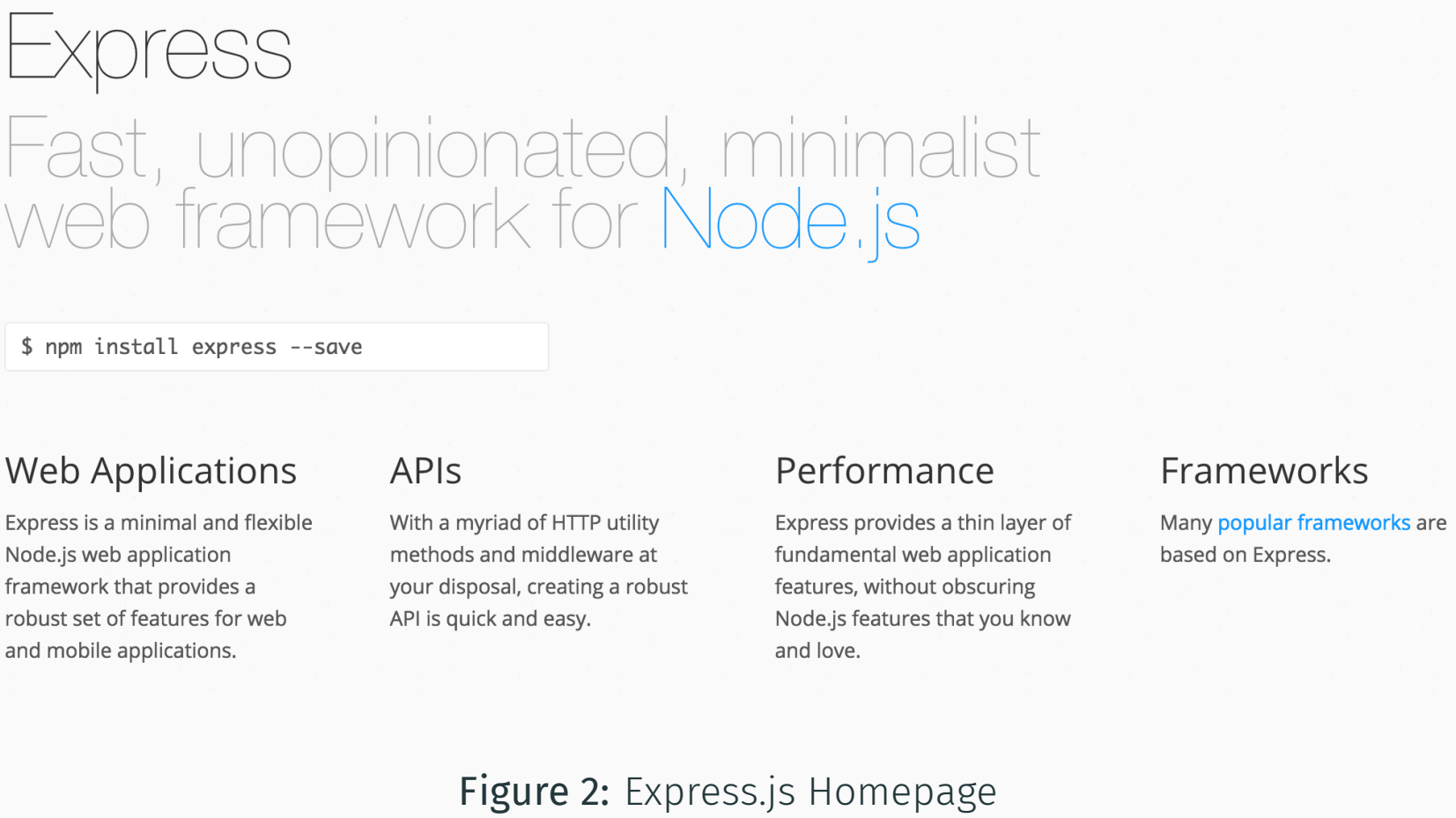

Figure 2: Express.js Homepage

What does it all mean!?

What does Express.js provide us?

Express provides some important features:

- An HTTP server that listens on a specific port
    - Provides access to the HTTP Request and Response
- A URL Router
    - Maps URL paths to our backend code
- An interface (API) to use and write our own middleware and plugins

Express.js does not:

It **does not** provide:

- A database
- A testing framework
- A file structure

Express.js does not:

It **does not** provide:

- A database
- A testing framework
- A file structure

It is lean, mean, and unopinionated!

Express Resources

Express documentation (all on one page)

- https://expressjs.com/en/4x/api.html

Starter Tutorials to look at

- Hello World: https://expressjs.com/en/starter/hello-world.html
- Basic Routing: https://expressjs.com/en/starter/basic-routing.html

Express Resources

Detailed Guides

- Routing in depth: https://expressjs.com/en/guide/routing.html

Live code: Build the Hello World app

Live code: Build the Zip Code API

Route Parameters / Dynamic Routes

`http://localhost:8000/zip/10016`

The path in this URL is: `/zip/10016` In express we can match this path with a handler such as:

```
app.get('/zip/:zipCode', (req, res) => {
    const zip = req.params.zipCode;
    // handle zip...
    res.json(results);
});
```

`:zipCode` is a **route parameter** and can be accessed in the request as `req.params.zipCode`.

Read more here: https://expressjs.com/en/guide/routing.html

Query Parameters

`http://localhost:8000/zip/10016?sort=desc&sort_by=city_name`

Everything after the `?` are **query parameters**. Query parameters are **NOT** part of the path. In this case the query parameters are:

`?sort=desc&sort_by=city_name`

```
app.get('/zip/:zipCode', (req, res) => {
    const sort = req.query.sort;
    const cityName = req.query.city_name;
    // handle zip...
    res.json(results);
});
```

Access query parameters from the request like this: `req.query.sort`

Read more here: https://expressjs.com/en/4x/api.html#req.query

Body Parameters / Form Parameters

Body parameters are sent as part of the request body. Typically for non-GET requests, like in forms that use POST method.

```
app.post('/login', (req, res) => {
    const username = req.body.username;
    const password = req.body.password;
    // handle zip...
    res.json(results);
});
```

We access the body parameters from the request object, such as `req.body.myData`.

Read more here: https://expressjs.com/en/4x/api.html#req.body

RESTful Routing (GET, POST, PUT, DELETE)

What is CRUD?

- **CRUD** represents the four basic functions of working with data or resources
    - (C)reate
    - (R)etrieve
    - (U)pdate
    - (D)elete
- Many applications require some or all users to perform these operations

What is RESTful Routing? (BEST PRACTICE)

- **REST** – REpresentational State Transfer
- We use the concept of *Resources*
- We want to allow CRUD operations on the resources through HTTP
- Make use of the HTTP verbs for these operations
- Make consistent and “pretty” URL’s

CRUD to REST mapping

- Create –> POST
- Retrieve –> GET
- Update –> PUT
- Delete –> DELETE

RESTful route design

HTTP Verb	Path	Controller#Action	Used for
GET	/photos	photos#index	display a list of all photos
GET	/photos/new	photos#new	return an HTML form for creating a new photo
POST	/photos	photos#create	create a new photo
GET	/photos/:id	photos#show	display a specific photo
GET	/photos/:id/edit	photos#edit	return an HTML form for editing a photo
PATCH/PUT	/photos/:id	photos#update	update a specific photo
DELETE	/photos/:id	photos#destroy	delete a specific photo

Figure 3: RESTful routes example