Module 3: Building an App from Scratch; MVC Architecture; RESTful Routing; Intro to Testing

Edgardo Molina, PhD | Lead Instructor

CUNY Tech Prep 2016-2017

Outline

Building an App in Express.js from Scratch

- Introduction to the Express.js Framework
- The Model-View-Controller (MVC) Architecture
- RESTful Routing (GET, POST, PUT, DELETE)
- Intro to Testing with Mocha/Chai

Introduction to the Express.js Framework

What is Express.js?

What is Express.js?

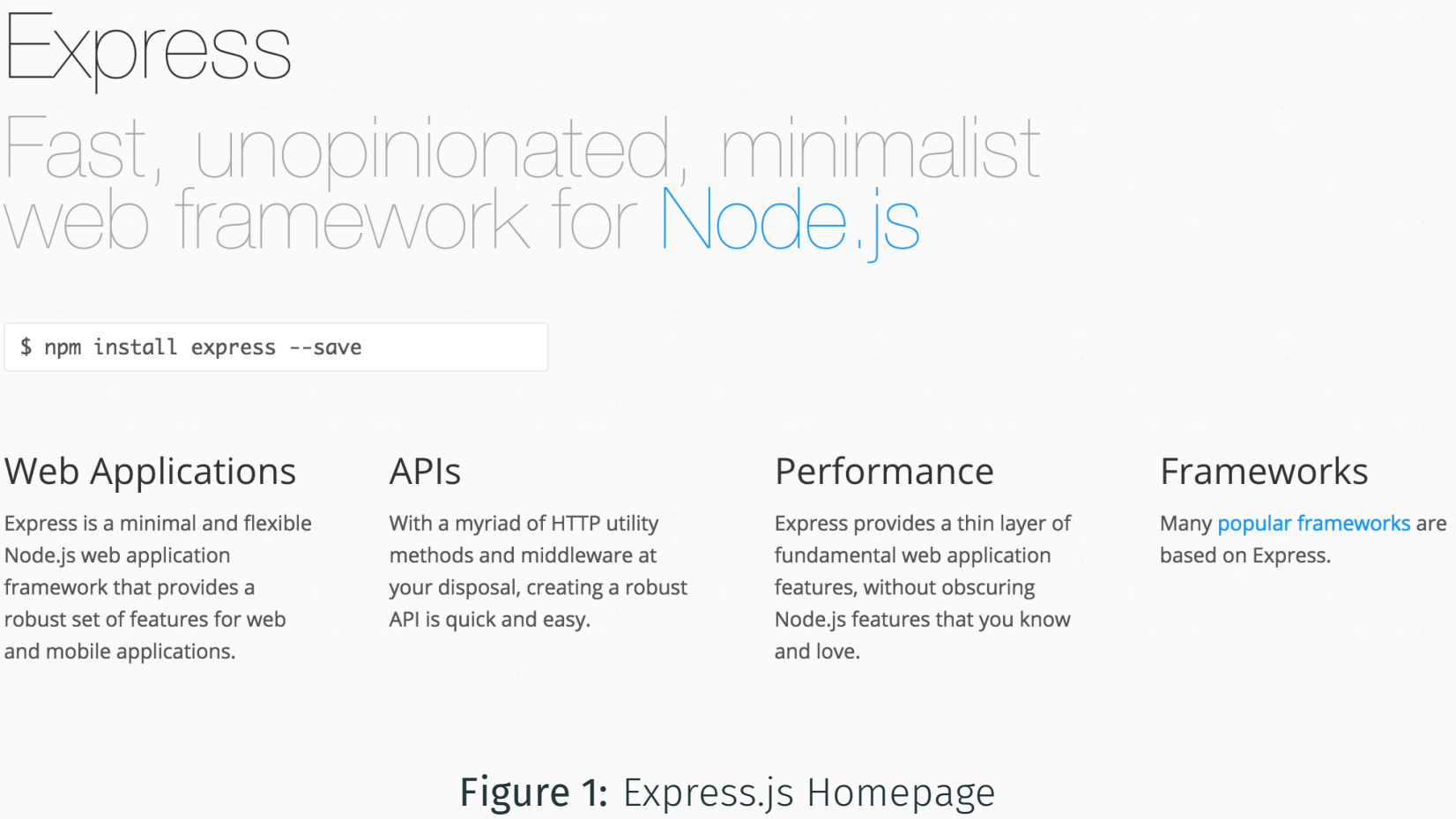

Figure 1: Express.js Homepage

What is Express.js?

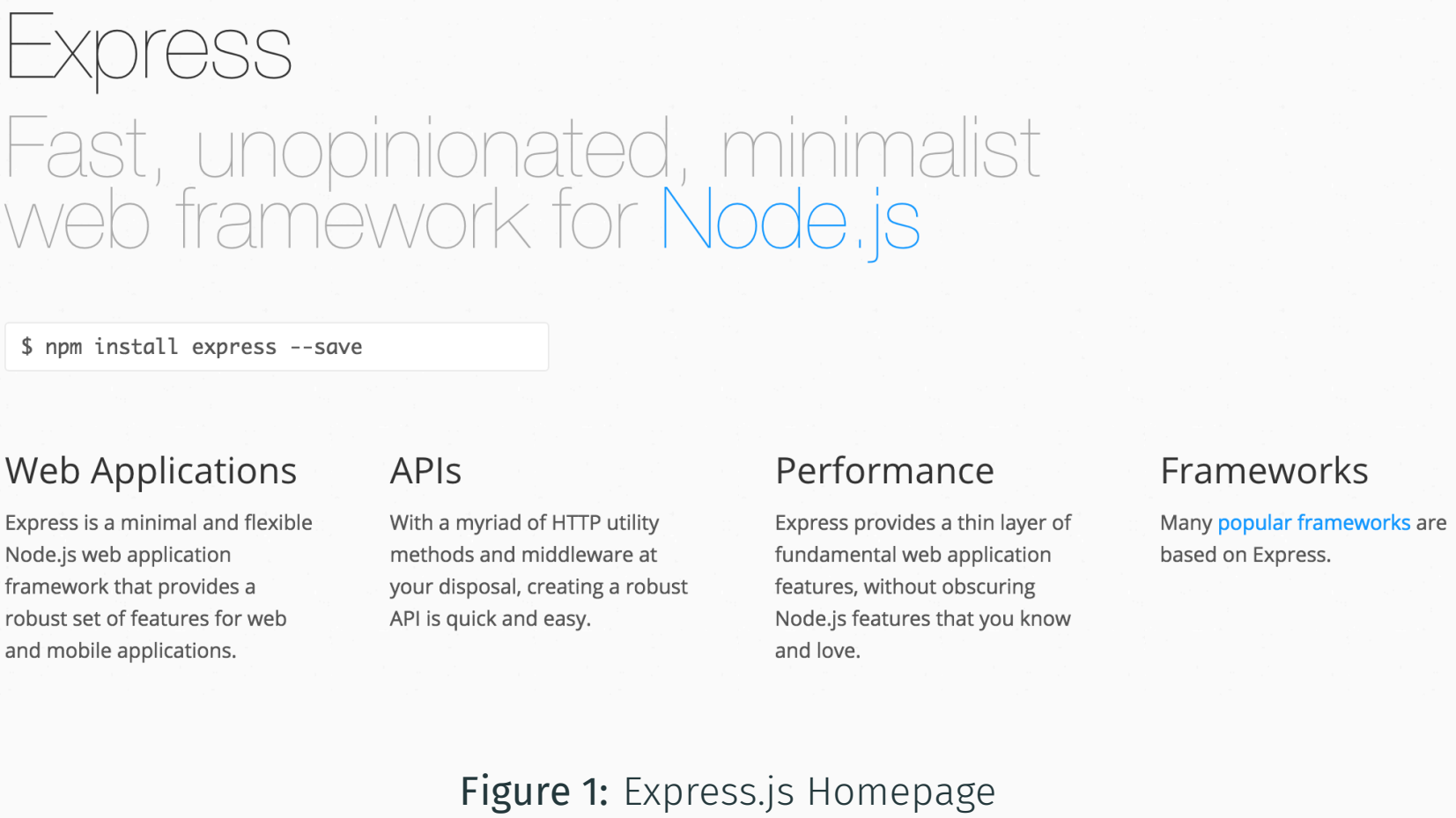

Figure 1: Express.js Homepage

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
- A URL Router
- An interface (API) to use and write our own plugins and middleware

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`

Let’s see some code!

Live Code! We will do the following:

- Create a new Node.js App with NPM
- Install Express.js
- Create a simple Web App and learn
    - Route Matching
    - Route Parameters
    - Query Parameters

Code is here: (ctp-lecture-code/module03/01)

The Model-View-Controller (MVC) Architecture

What is MVC?

(M)odels, (V)iews, and (C)ontrollers

MVC is a software pattern that defines how we should structure and layout our application code.

Many modern web frameworks have adopted the use of the MVC pattern.

Models

Models ***represent our data entities****, and provide a mechanism to* ***store the data*** *to persistent storage (databases, filesystem, services, etc), and to* ***retrieve the data****.*

Models

> *Models* ***represent our data entities****, and provide a mechanism to* ***store the data*** *to persistent storage (databases, filesystem, services, etc), and to* ***retrieve the data****.*

This is where we model the data that our web app is working with, such as: *Articles, Movies, Songs, Books, Cars, Users, Admins,* etc.

It is an interface to our database/persistence layer, (although there does not have to be one).

Views

Views ***generate our applications output****. Primarily concerned with the* ***presentation*** *and* ***display*** *of our data.*

Views

> *Views* ***generate our applications output****. Primarily concerned with the* ***presentation*** *and* ***display*** *of our data.*

This is where the output presentation is handled. Minimal to no logic should exist here.

We should treat it as if it were a “fill in the blanks” template.

Controllers

Controllers contain ***action methods*** *that* ***receive HTTP requests****, the action method processes any input (route, query, and body parameters), it calls models and services as needed, and finally uses a view to* ***provide output for the HTTP response****.*

Controllers

> *Controllers contain* ***action methods*** *that* ***receive HTTP requests****, the action method processes any input (route, query, and body parameters), it calls models and services as needed, and finally uses a view to* ***provide output for the HTTP response****.*

This is where we map our desired URL route space to specific *action* functions (callbacks) in Express.

The actions manage the lifecycle of http request and response.

Models, Views, and Controllers

MVC is only a pattern. It is up to the programmer to use the pattern effectively to layer their application.

Models, Views, and Controllers

MVC is only a pattern. It is up to the programmer to use the pattern effectively to layer their application.

For example:

- Views should not include db queries or business logic code
- Controllers should not directly talk to the database and it should not directly generate HTML.
- Models should not be concerned with html output or business logic rules*

Models, Views, and Controllers

MVC is only a pattern. It is up to the programmer to use the pattern effectively to layer their application.

For example:

- Views should not include db queries or business logic code
- Controllers should not directly talk to the database and it should not directly generate HTML.
- Models should not be concerned with html output or business logic rules*

** These are the “minimum” application layers. Your business needs may require that you add additional layers between these 3 as your apps become more complex.

What is business logic?

- Authorization and Access rules
    - Who is allowed to access the data?
    - Who is allowed to modify the data?
- What input is needed?
- Which actions are allowed?

Let’s see some code!

Live Code! We will do the following:

- Let’s install nodemon!
- Use `express.Router()` to create a Controller

Code is here: (ctp-lecture-code/module03/02)

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
- Think of our Post and User model in the CTP Microblog.

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 2: RESTful routes example

Intro to Testing with Mocha/Chai

Testing

Why do we test?

Testing

Why do we test?

Many reasons! We'll cover the details next lecture.

Mocha / Chai

Mocha is a testing framework

- `https://mochajs.org/`
- It handles running and reporting test outcomes

Chai is an assertion library

- `http://chaijs.com/`
- Provides us 3 different styles of writing tests
- Each has it’s own merits.

How does testing influence our App code

If it is too complex to test, then it should be broken up

Convention, avoid objects and functions that do too much. This applies to Models, Views, and Controllers.

TDD - Test Driven Development

BDD - Behavior Driven Development

Let’s see some code!

Live Code! We will do the following:

- Install mocha, chai, and chai-http
- Add a test task to `packages.json`
- Create pending tests for our RESTful controller
- Implement a test and an action
- Run the tests

Code is here: (ctp-lecture-code/module03/03)