NODE.JS

Part II

NEED FOR A TEMPLATE

- ❑If you want to add specific handling for different HTTP verbs (e.g. GET, POST, DELETE, etc.)
- ❑Separately handle requests at different URL paths ("routes")
- ❑ Serve static files, or use templates to dynamically create the response,
- ❑ Node won't be of much use on its own.
- ❑You will either need to write the code yourself, or you can avoid reinventing the wheel and use a web framework!

APPLICATION FRAMEWORKS

- provides frozen spots
  - overall architecture
  - How the components interact
- allows to concentrate in hot spots to extend the behaviour of the framework
  - Hot spots are the functions written for the application
- A framework is not suitable for a problem when …

WEB APPLICATION FRAMEWORKS

- An application framework that is designed to support development of web applications that generally includes
  - Database support
  - Templating framework for generating dynamic web content
  - HTTP session management with middleware support
  - Built-in testing framework
- It can also support internationalization, security and privacy
- Consistent look and feel and consistent with database

WEB FRAMEWORKS EXAMPLES

- Ruby on Rails
- Play
- ASP.NET
- Django
- Symfony
- Spring
- Vue.js
- Angular js

EXPRESS

- Express is the most popular *Node* web framework, and is the underlying library for a number of other popular Node web frameworks.
- It provides mechanisms to:
- Write handlers for requests with different HTTP verbs at different URL paths (routes).
- Integrate with "view" rendering engines in order to generate responses by inserting data into templates.
- Set common web application settings like the port to use for connecting, and the location of templates that are used for rendering the response.
- Add additional request processing "middleware" at any point within the request handling pipeline.

EXPRESS FEATURES

- *Express* itself is fairly minimalist
- Developers have created compatible middleware packages to address almost any web development problem
- There are libraries to work with cookies, sessions, user logins, URL parameters, POST data, security headers, and *many* more.
- You can find a list of middleware packages maintained by the Express team at Express Middleware (along with a list of some popular 3rd party packages).

IS EXPRESS OPINIONATED

- Opinionated frameworks are those with opinions about the "right way" to handle any particular task. They often support rapid development *in a particular domain* (solving problems of a particular type) because the right way to do anything is usually well-understood and well-documented

- Unopinionated frameworks, by contrast, have far fewer restrictions on the best way to glue components together to achieve a goal, or even what components should be used.

EXPRESS OPINION

- Express is unopinionated.
- You can insert almost any compatible middleware you like into the request handling chain, in almost any order you like.
- You can structure the app in one file or multiple files, using any directory structure.
- You may sometimes feel that you have too many choices

EXPRESS FEATURES

- ❑Express provides methods to specify what function is called for a particular HTTP verb (GET, POST, SET, etc.) and URL pattern ("Route")
- ❑ Provides methods to specify what template ("view") engine is used
- ❑ Where template files are located
- ❑ What template to use to render a response
- ❑You can use Express middleware to add support for cookies, sessions, and users, getting POST/GET parameters, etc.
- ❑You can use any database mechanism supported by Node (Express does not define any database-related behavior).

EXPRESS APP

- This object, which is traditionally named app, has methods for
- routing HTTP requests
- configuring middleware,
- rendering HTML views,
- registering a template engine, and
- modifying application settings that control how the application behaves (e.g. the environment mode, whether route definitions are case sensitive, etc.)
- res.json() to send a JSON response or res.sendFile() to send a file

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

app.get("/", function (req, res) {
  res.send("Hello World!");
});

app.listen(port, function () {
  console.log(`Example app listening on
port ${port}!`);
});
```

ROUTER

A common convention for Node and Express is to use error-first callbacks. In this convention, the first value in your *callback functions* is an error value, while subsequent arguments contain success data.

Routes allow you to match particular patterns of characters in a URL, and extract some values from the URL and pass them as parameters to the route handler

The node:path module provides utilities for working with file and directory paths

The path.join() method joins all given path segments together using the platform-specific separator as a delimiter, then normalizes the resulting path.

Zero-length path segments are ignored. If the joined path string is a zero-length string then '.' will be returned, representing the current working directory.

```
path.join('/foo', 'bar', 'baz/asdf', 'quux', '..');

// Returns: '/foo/bar/baz/asdf'
```

REQUEST HANDLING

- The *Express application* object also provides methods to define route handlers for all the other HTTP verbs, which are mostly used in exactly the same way:
- checkout(), copy(), **delete()**, **get()**, head(), lock(), merge(), mkactivity(), mkcol(), notify(), options( )patch(), **post()**, purge(), **put()**, report(), search(), subscribe(), trace(), unlock(), unsubscribe()
- There is a special routing method, app.all(), which will be called in response to any HTTP method.
- This is used for loading middleware functions at a particular path for all request methods.

```
app.all("/secret", function (req, res, next) {
         console.log("Accessing the secret section…");
          next(); // pass control to the next handler });
```

ROUTING THROUGH EXPRESS

```
const wiki = require("./wiki.js");

app.use("/wiki", wiki);
```

```
// Home page route
router.get("/", function (req, res) {
  res.send("Wiki home page");
});
```

- ❑Often it is useful to group route handlers for a particular part of a site together and access them using a common route-prefix
- ❑a site with a Wiki might have all wiki-related routes in one file and have them accessed with a route prefix of */wiki/*
- ❑ In *Express* this is achieved by using the express.Router object.

https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction

ALL ABOUT ERRORS

- ❑ Errors are handled by one or more special middleware functions that have four arguments, instead of the usual three: (err, req, res, next)

```
app.use(function (err, req, res, next) {
     console.error(err.stack);
     res.status(500).send("Something broke!"); });
```

These can return any content required, but must be called after all other app.use() and routes calls so that they are the last middleware in the request handling process!