What Is Express

Express is a web framework for Node.js. Web applications share common patterns, so using a framework is often a good idea. You will find that you can develop faster and write applications on top of stable, tested code.

Some other web frameworks that you may be familiar with are

- Ruby on Rails (Ruby)
- Sinatra (Ruby)
- Django (Python)
- Zend (PHP)
- CodeIgniter (PHP)

Why Use Express?

Express is a lightweight framework, which means that it does not make too many assumptions but gives you enough to avoid reinventing the wheel.

Some of the things you can do with Express include

- JSON-based APIs
- Single-page web applications
- Real-time web applications

Some reasons for using a framework like Express include

- It takes less time to create applications using a framework.

- Common patterns, like routing and view layers, are accounted for in a framework like Express, meaning you do not have to write code for this.

- A framework like Express is actively used, maintained, and tested. The stability of the code can be expected.

Frameworks like Express are not appropriate for everything, though. If you are creating a command-line script, you certainly do not want to use something like Express.

Express Was Inspired by Sinatra

Sinatra is a popular lightweight web framework for Ruby. Express supports template engines, routing, and passing data to views in much the same way that Sinatra does in Ruby.

If you have used Sinatra, you will feel right at home in Express. If you have not used Sinatra, most developers like the simplicity of its design.

Installing Express

You can install Express via npm:

```
npm install -g express
```

Creating a Basic Express Site

Now that you have installed Express, you are ready to get a basic site up and running.

Follow these steps to install a basic Express site:

1. Open your terminal and generate a skeleton Express site by running the following command

```
express express_example
```

2. Express politely reminds you to install the dependencies needed to run Express. So, make sure that you install the dependencies:

```
cd express_example && npm install
```

3. Start the application by running

```
node app.js
or
set DEBUG=myapp & npm start
```

4. Open your web browser of choice and browse to http://127.0.0.1:3000. You see a basic website served from Express

Exploring Express

If you look in the folder for the example Express site that you just created, you see the following structure:

- app.js
- node_modules
- package.json
- public
- routes
- views

app.js

app.js is the application file used to start the application. It contains configuration information for the application.

node_modules

node_modules holds any node modules that are defined in package.json and have been installed.

package.json

package.json gives information on the application, including the dependencies that should be installed for it to run.

Public

The public folder serves the application to the web. You will find stylesheets, JavaScripts, and images in this folder. You will not find any application logic in this folder; this is a common pattern to ensure web application security.

Routes

In simple terms, a route defines the pages that an application should respond to. For example, if you want to have an About page in your application, you need to set up an 'about' route. The routes folder holds these declarations.

Views

The views folder defines the layouts for the application.

Folder Structure Is Optional

The Express generator creates a suggested layout for an Express project. This is just a suggestion, so if your application has specific requirements or your personal preference is different, you can structure Express projects however you want. If you are getting started with Express, it is recommended that you use the structure from the generator.