In order to follow this tutorial you will need the latest version of [Node js](https://nodejs.org/en/), Typescript, and [Typings](https://github.com/typings/typings) installed. I recommend installing node js via [Homebrew](http://brew.sh/) if you are on OSX. Once node is installed, both Typings and Typescript can be installed via [NPM](https://www.npmjs.com/).

Use the following commands to do so:

npm install -g typescript

npm install -g typings

Setting up a Typescript project

Now that the prerequisites are installed we can begin setting up the Typescript project. Open up a terminal, create, and 'cd' into a directory called *express-ts*.

mkdir express-ts

cd express-ts

Now that we are in the created directory we can initialise the Typescript project. To do this we will use the tsc and typings commandline executables. These were installed via Typescript and Typings respectively.

Firstly we want to create a tsconfig.json file. This can be done with tsc.

tsc --init

Running the above command creates the tsconfig.json in the current directory. It also adds some useful boilerplate code to the file. Looking inside the newly-generated tsconfig.json you should see the following:

{

{

"compileOnSave": false,

"compilerOptions": {

"outDir": *"dist"*,

"baseUrl": *"src"*,

"sourceMap": false,

"declaration": false,

"moduleResolution": *"node"*,

"emitDecoratorMetadata": true,

"experimentalDecorators": true,

"target": *"es5"*,

"typeRoots": [

*"node\_modules/@types"*

],

"lib": [

*"es2016"*,

*"dom"*

]

}

}

Now that the tsconfig is correctly set up we will add some Typescript Definition files or .d.ts files via Typings. These files are used to give the compiler knowledge of the application.

Run the following to create a typings.json file:

typings init

As we will be using Express and ES2015 syntax we need to install the es6-shim, node, and express typings. Run the following commands to do so:

typings install dt~node --save --global

typings install dt~es6-shim --save –global

typings install dt~express --save --global

typings install dt~serve-static --save --global

typings install dt~express-serve-static-core --save –global

typings install dt~mime --save --global

The above commands will update the typings.json file which was generated by the typings init command and place the Typings dependencies in the typings folder.

Now we can install the application depencies via NPM. In the application root folder run the following:

npm init –y

npm install --save express

This creates a package.json file and installs Express as a dependency.

Finally let's create the application files. Make a folder called app and add a server.ts file. The application will be a greeter. So let's also add a WelcomeController. Inside the app folder create another folder called controllers and a welcomeController.ts and index.ts file.

mkdir app && cd app

touch server.ts

mkdir controllers && cd controllers

touch index.ts welcomeController.ts

The application should now have the following structure:

.

├── app

│ ├── controllers

│ │ ├── index.ts

│ │ └── welcomeController.ts

│ └── server.ts

├── node\_modules

├── package.json

├── tsconfig.json

├── typings

└── typings.json

## Creating an express app

We will be creating a greeting app which will have one route that takes a name parameter and then greets that name.

Open a text editor inside the application folder. If you are using Visual Studio Code you can open it inside the folder by running the code command with a folder argument.

code .

Firstly, let's take a look at the welcomeController.ts file. This file will handle the welcome routes. To do this we need it to export an Express router object. I have added code comments to the snippet below which explains how this file should work:

/\* app/controllers/welcomeController.ts \*/

// Import only what we need from express

import { Router, Request, Response } from 'express';

// Assign router to the express.Router() instance

const router: Router = Router();

// The / here corresponds to the route that the WelcomeController

// is mounted on in the server.ts file.

// In this case it's /welcome

router.get('/', (req: Request, res: Response) => {

// Reply with a hello world when no name param is provided

res.send('Hello, World!');

});

router.get('/:name', (req: Request, res: Response) => {

// Extract the name from the request parameters

let { name } = req.params;

// Greet the given name

res.send(`Hello, ${name}`);

});

// Export the express.Router() instance to be used by server.ts

export const WelcomeController: Router = router;

Now that the WelcomeController is ready to be used, let's export it from the controllers folder. As you may know, index files act as folder entry points. Thanks to this we can access exports from the index.ts file we created in the controllers folder. Add the following to that file:

/\* app/controllers/index.ts \*/

export \* from './welcomeController';

Note: as good practice you should never add application logic inside an index file.

Now that the WelcomeController is being correctly exported, let's make use of it inside the server.ts file.

/\* app/server.ts \*/

// Import everything from express and assign it to the express variable

import \* as express from 'express';

// Import WelcomeController from controllers entry point

import {WelcomeController} from './controllers';

// Create a new express application instance

const app: express.Application = express();

// The port the express app will listen on

const port: number = process.env.PORT || 3000;

// Mount the WelcomeController at the /welcome route

app.use('/welcome', WelcomeController);

// Serve the application at the given port

app.listen(port, () => {

// Success callback

console.log(`Listening at http://localhost:${port}/`);

});

Now that we have finished writing the application, let's transpile it to javascript. In the root of the application run the following:

tsc

Alternatively you may run the following to tell the Typescript compiler to run everytime it detects a filesystem change.

tsc --watch

These commands will tell the Typescript compiler to build the application based on the tsconfig.json. If everything has been successfull you will see a newly created build directory. This is where the Typescript compiler has placed the generated .js files based of the optional outDir parameter in the tsconfig.

We can now run the app with the following command:

node build/server.js

\*\*Config webpack

npm install --save-dev webpack

npm install ts-loader –-save-dev

npm i json-stringify-safe –save

npm i --save lodash

\*webpack.config.js

**const** path = require('path')

module.exports = {

mode : 'production',

entry :'./src/server.ts' ,

target : 'node',

output : {

filename : 'study.[name].js',

path : path.resolve(\_\_dirname, 'dist')

},

resolve : {

extensions : [ '.ts', '.js', '.json' ]

},

module : {

rules : [ {

test : /\.html$/,

loader : 'html'

}, {

test : /\.ts$/,

loader : 'ts-loader'

} ]

}

}