Express Framework

WIF2003 WEB PROGRAMMING

Objectives

- What are framework and library?
- What is Express?
- Why are we using Express?
- Create web applications from scratch using Express
  - Create main app.js file to define routes
  - Write routes containing parameters
  - Create package.json file
- Create web applications using Express Generator

What is a library?

- A library on the other hand is a collection of functionality that you can call.
- We have used a few libraries, such as jQuery and Bootstrap.
- If you want to use a library, you can use one method or more methods, just like how we include jQuery
  - It's up to us which parts of it we use

What is a framework? (1)

- Frameworks take all of the **common tasks** that we do in every application (e.g. setup work, basic things that every app needs) by **prepackaging** the common tasks
- We can use framework and get started on new apps **without** having to do all the **basic groundwork** every single time.
  - Save our time and effort
- With framework (e.g. Express, Matlab), we give up a little bit of control, the framework have **made some decisions** for us
  - But the framework does not to replace any sort of creativity

What is a framework? (2)

- There are frameworks that help you to:
  - make video games,
  - make mobile apps,
  - make web applications
    - E.g. Django for Python, Rails for Ruby
    - E.g. Express.js, NestJs and Sails.js for Node.js)
- **Lightweight Framework:** You need less configurations to make the application work. It is like up and running.
  - E.g. Spring for Java, Express for Node.js
- **Heavyweight Framework :** You need to make a lot of changes before making the application works completely.
  - E.g. Rails for Ruby

Framework vs Library

- The most important **difference**, and in fact the defining difference between a library and a framework is **Inversion of Control.**
- What does this mean?
  - **Library:** When you call a library, you are in control.
  - But with **a framework**, the control is inverted:
    - The framework calls you
    - This is called the Hollywood Principle: *Don't call Us, We'll call You.*
- All the **control flow** is already in the framework, and there is just a bunch of predefined white spots that you can fill out with your code

What is Express.JS?

- A Web Framework to develop Web Applications very easily and quickly in Node JS Platform.
- Express JS Official Website:
  - http://expressjs.com/
- Express is **a minimal** and **flexible** Node.js web application framework that provides **a robust set of features** for web and mobile applications.

Why are we using Express?

- One of the most widely use web framework for Node.js
  - There are a lot of tutorials and big community of developers are using this
- Express is a **lightweight** framework, much lighter as compared to other frameworks
- Fast, flexible, minimalist, web framework for Node.js
- By using Express, we are able to :
  - **focus on** writing the **application codes**, and
  - **don't have to focus** on all the **basic work** that every app requires to create a web application

Express JS Features

- Light-weight Web Application Framework
- It Supports Routings
- It supports Template Engines
- It supports File Uploading
- Develop SPA(Singe Page Web Applications)
- Develop Real-time Applications

Using middleware

- Express is a **routing** and **middleware** web framework that has **minimal functionality** of its own
- An Express application is essentially a series of middleware function calls
- Middleware functions are functions that have access to:
  - the request object (req),
  - the response object (res),
  - and the next middleware function in the application's *request-response cycle*

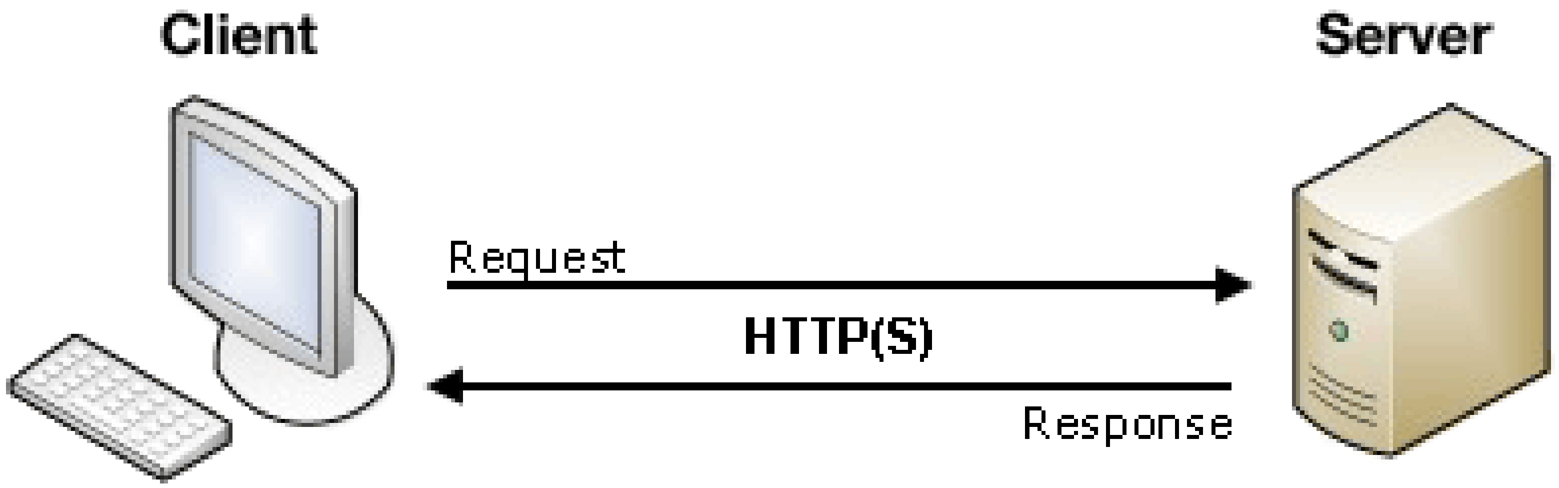
Middleware functions

- Middleware functions can perform the following tasks:
  - Execute any code.
  - Make changes to the request and the response objects.
  - End the request-response cycle.
  - Call the next middleware in the stack.
- Reference: https://expressjs.com/en/guide/writing-middleware.html

Request and Response objects

Request and Reponse

- Client-Server Model



HTML Form

```
<form action="http://www.foo.com" method="POST">
  <div>
    <label for="say">What greeting do you want to
say?</label>
    <input name="say" id="say" value="Hi" />
  </div>
  <div>
<div>
    <button>Send my greetings</button>
  </div>
</form>
```

HTTP Request Methods

- **GET:** The GET method requests a representation of the specified resource. Requests using GET should only retrieve data.
- **POST:** The POST method submits an entity to the specified resource, often causing a change in state or side effects on the server.
- **PUT:** The PUT method replaces all current representations of the target resource with the request payload.
- **DELETE:** The DELETE method deletes the specified resource.
- **Reference:** https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods

GET	POST
GET is an array of variables passed to the current script via the **URL parameters**.	POST is an array of variables passed to the current script via the **HTTP POST method**.
GET requests can be **cached**.	POST requests are **never** cached.
Information sent is **visible to everyone** (all variable names and values are displayed in the URL).	Information sent is **invisible to others** (all names/values are embedded within the body of the HTTP request).
It is **possible to bookmark** the page	It is **not possible to bookmark** the page.
Has **limits** on the amount of information to send. The limitation is about 2000 characters.	Has **no limits** on the amount of information to send.
GET may be used for sending **non-sensitive data**. Should NEVER be used for sending passwords or other sensitive information!	POST may be used for sending **sensitive and non-sensitive data**. Support **advanced functionality** such as support for multi-part binary input while uploading files to server.

Create web applications using Express

ExpressJS help us to …

- Start up a server to listen for requests
- Parse incoming requests
- Match those requests to particular routes
- Write our http response and associated content

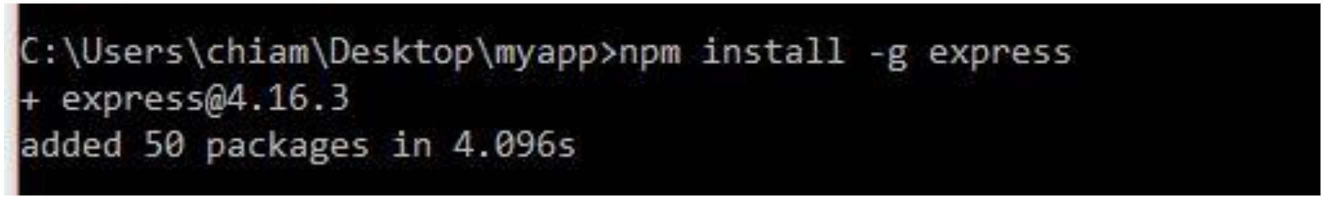
Express JS Setup

- Express JS does not come with as Node JS Default modules.
- We need to install it manually.
- To install Express JS globally, execute this command:

  `npm install -g express` **OR**

  `npm install express`

  - "express" means Express JS Module
  - "-g" means install Express JS Module globally.

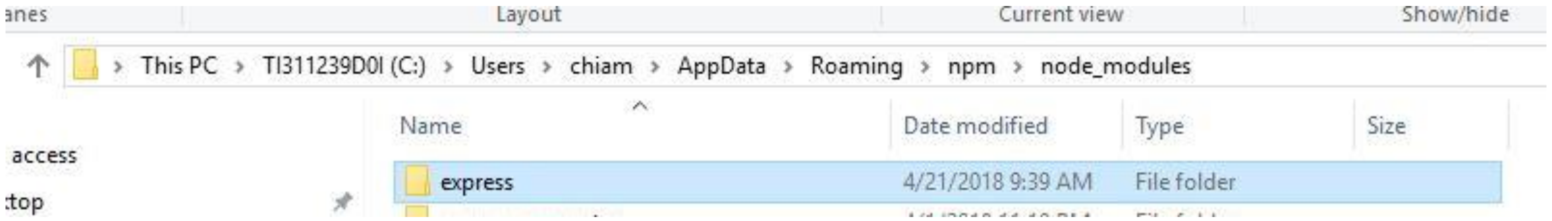


```
C:\Users\chiam\Desktop\myapp>npm install -g express
+ express@4.16.3
added 50 packages in 4.096s
```

Express JS Setup

To verify installation:

- If it is installed successfully, we can find a new **folder** at your operating system, e.g. Windows: `C:\Users\[Windows_UserName]\AppData\Roaming\npm\node_modules\express`



This PC › TI311239D0I (C:) › Users › chiam › AppData › Roaming › npm › node_modules

Name	Date modified	Type	Size
express	4/21/2018 9:39 AM	File folder	

Create a simple Express App

Steps to create a simple Express App:

1. Create a new project directory:

```
$ mkdir FirstExpressApp
```

2. Change directory:

```
$ cd FirstExpressApp
```

3. Use the `npm init` command to create a package.json file for your application (to store metadata about a package or project)

```
$ npm init
```

4. Create a new main file named `app.js`

```
$ fsutil file createnew app.js 0
```

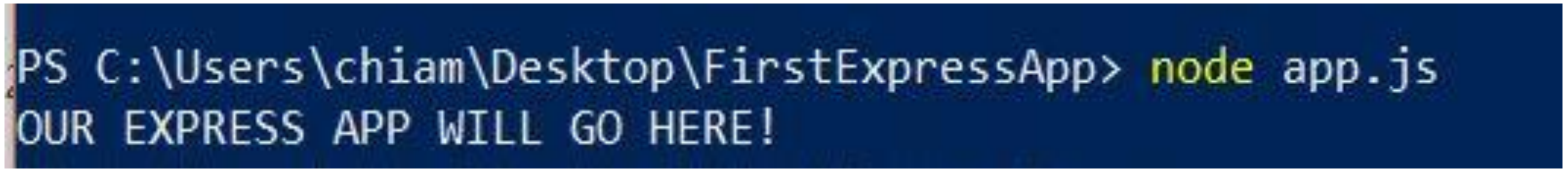
Create a simple Express App

5. Open the app.js file and add the JS code:

```
console.log("OUR EXPRESS APP WILL GO HERE!");
```

6. Run the app.js file to test the file:

```
$ node app.js
```



```
PS C:\Users\chiam\Desktop\FirstExpressApp> node app.js
OUR EXPRESS APP WILL GO HERE!
```

7. Install Express in the project directory

```
$ npm install express
```

Create a simple Express App: Define routes

8. Open app.js file again, *remove* the JS code and edit the file to:

- require content of Express in our application, and
- execute express as a function and save it to a variable called `app`

```
const express = require("express");

const app = express();
```

9. Run the file at console (nothing should happen)

```
$ node app.js
```

Create a simple Express App: Define routes

10. Edit app.js file to define routes

- Example syntax to define a route

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

- The first parameter '/' is the **root** url or path
- The second parameter is the **callback function**
  - The callback function takes two different object arguments: **request** and **response**

Create a simple Express App: Define routes

- The callback function takes **two different object arguments**: request (`req`) and response (`res`)
  - `req` are objects that contains all the information about the request that was made that trigger this route
  - `res` are objects that contains all the information about what we are going to respond with
- res.send – responding with some text

Create a simple Express App: Start a server and listens to requests

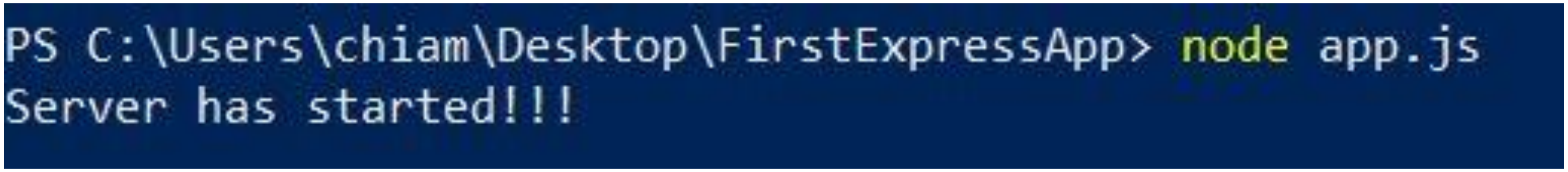
- **11. Edit app.js file to write the code to tells Express app to start a server and listens on port 3000 for connections (requests).**

```
app.listen(3000, function(){
    console.log('Server has started!!!');
});
```

- Run the `app.js` file:

```
 $ node app.js
```

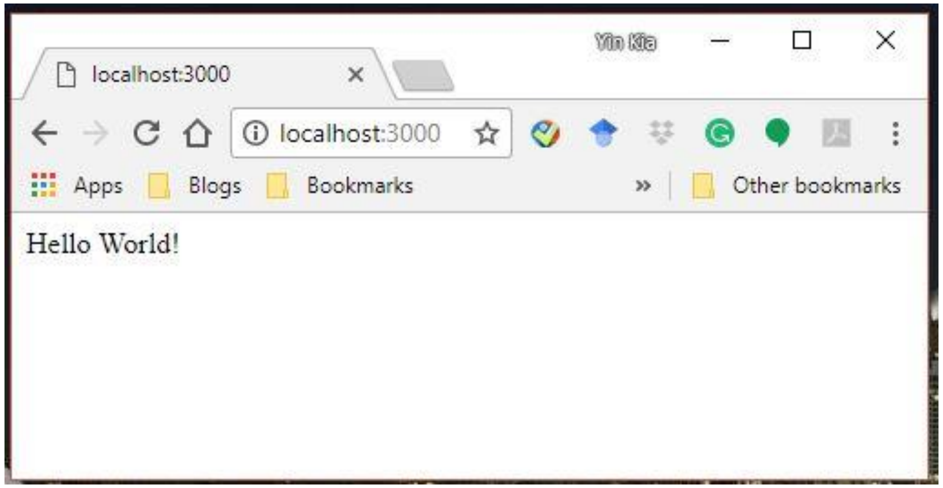
- The web server is up and running now:



```
PS C:\Users\chiam\Desktop\FirstExpressApp> node app.js
Server has started!!!
```

Create a simple Express App: Start a server and listens to requests

- Access Express JS web application using `http://localhost:3000/` from our browser.
- The app responds with "Hello World!" for requests to the root URL (/) or route.
- Every time **you make any changes** to the app.js file, you need to **restart** the web server (stop the server using Ctrl-C)



Hello World!

App.js

```
const express = require('express')
const app = express()
const port = 3000

// respond with "hello world" when a GET request is made to the homepage
app.get('/', (req, res) => {
  res.send('Hello World!')
})

app.listen(port, () => {
  console.log(`Example app listening on port ${port}`)
})
```

This app starts a server and listens on port 3000 for connections. The app responds with "Hello World!" for requests to the root URL (/) or ***route***. For every other path, it will respond with a **404 Not Found**.

Example of Post and Get routes

```
// GET method route
app.get('/', (req, res) => {
  res.send('GET request to the homepage')
 })

 // POST method route
 app.post('/', (req, res) => {
  res.send('POST request to the homepage')
 })
```

Route matcher &
Route parameters

Route paths based on strings

- This route path will match requests to the root route (home page), /

```
app.get('/', (req, res) => {
  res.send('root')
})
```

- This route path will match requests to /about.

```
app.get('/about', (req, res) => {
  res.send('about')
})
```

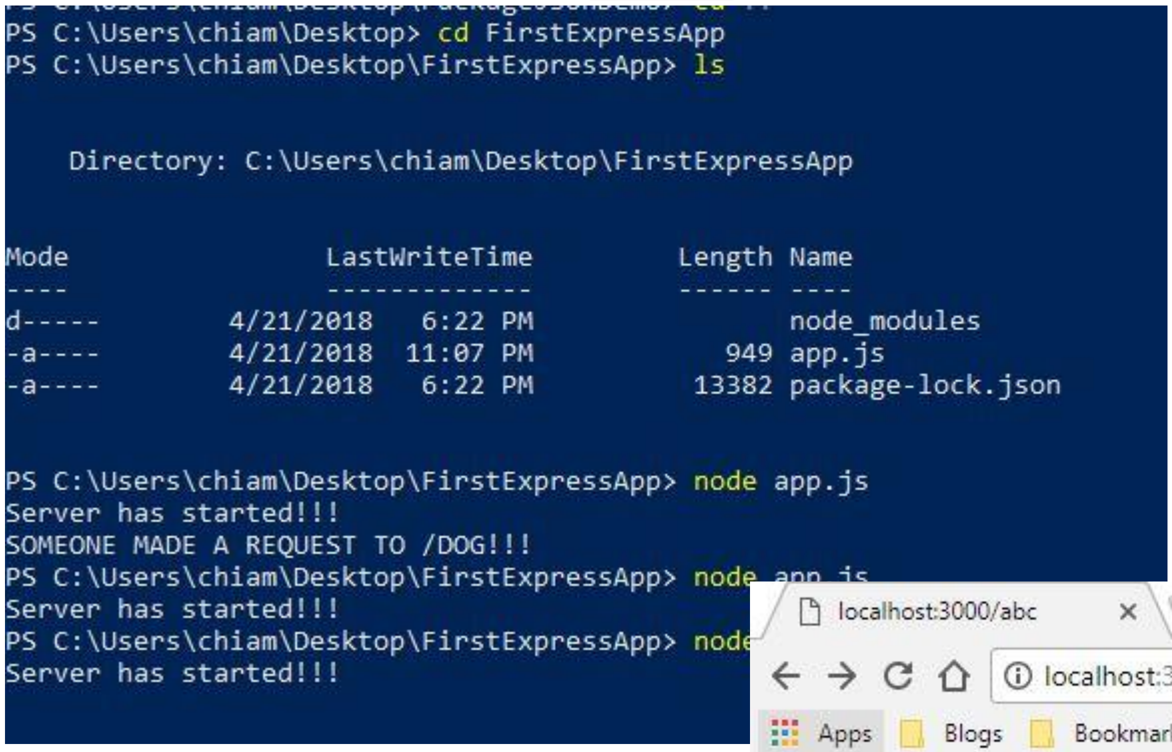
Show the '*' route matcher

- Open the app.js file and add the route matcher:

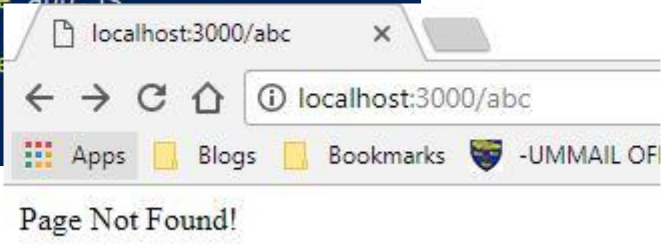
```
app.get('*', (req, res) => {
     res.send(`Page Not Found!`)
})
```

- This is especially useful if you want to have some sort of **error message** or **area of web page** that you show a user anytime they try and **access a route** that **is not defined**
  - You can show some sort of message or some HTML template that says "Page Not Found"

Show the ‘*’ route matcher



```
PS C:\Users\chiam\Desktop> cd FirstExpressApp
PS C:\Users\chiam\Desktop\FirstExpressApp> ls


    Directory: C:\Users\chiam\Desktop\FirstExpressApp


Mode                LastWriteTime         Length Name
----                -------------         ------ ----
d-----        4/21/2018   6:22 PM                node_modules
-a----        4/21/2018  11:07 PM            949 app.js
-a----        4/21/2018   6:22 PM          13382 package-lock.json


PS C:\Users\chiam\Desktop\FirstExpressApp> node app.js
Server has started!!!
SOMEONE MADE A REQUEST TO /DOG!!!
PS C:\Users\chiam\Desktop\FirstExpressApp> node app.js
Server has started!!!
PS C:\Users\chiam\Desktop\FirstExpressApp> node
Server has started!!!
```



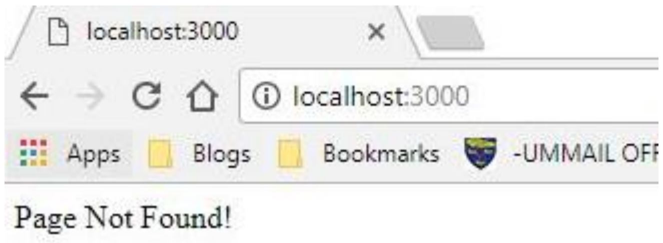
```
localhost:3000/abc
localhost:3000/abc
Apps  Blogs  Bookmarks  -UMMAIL OF
```

Page Not Found!

`/abc` route does not exist

Show the '*' route matcher

- If you move the "*" route matcher code to the beginning of route, you will find that when you access the root url (http://localhost:3000/) the browser still return "**Page Not Found**"



```
Page Not Found!
```

- It is because, if one of the callback functions is running, the **HTTP request has been handled** and it never moves on to other routes.

Route parameters

- Visit www.reddit.com:
  - https://www.reddit.com/**r**/soccer/
  - https://www.reddit.com/**r**/Music/
  - https://www.reddit.com/**r**/movies/
- When you visit each page and click one of the post, you will get all of the comments that correspond to the post that has that title
  - Example: https://www.reddit.com/r/soccer/comments/1k97lmx/blue_flares_lit_outside_anfield_after_an_everton/

Define a route pattern using route parameters

- In `app.js`, rather than define a separate route for every single page:

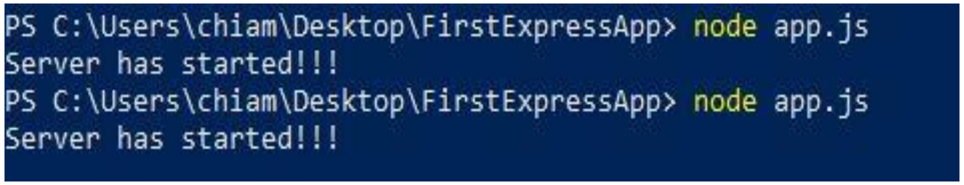
```
app.get("/r/soccer");
app.get("/r/Music");
app.get("/r/movies");
```

- We can define a **pattern** to listen for a get request using **route parameters**, add ':' in front of every parameter

```
app.get('/r/:subreddit', (req, res) => {
    const { subreddit } = req.params;
    res.send(`<h1>Browsing the ${subreddit} subreddit</h1>`)
})
```

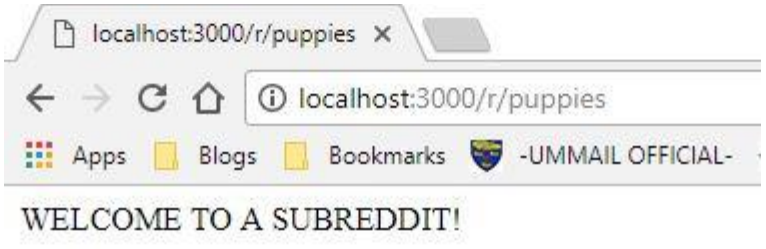
Define A Route Pattern using Route Parameters (Screenshot)

- Restart the web server:



```
PS C:\Users\chiam\Desktop\FirstExpressApp> node app.js
Server has started!!!
PS C:\Users\chiam\Desktop\FirstExpressApp> node app.js
Server has started!!!
```

- Go to http://localhost:3000/r/puppies, we will get a message "**WELCOME TO SUBREDDIT!**"



WELCOME TO A SUBREDDIT!

Define A Route Pattern using Route Parameters

- Rather than define a separate route for every single title:

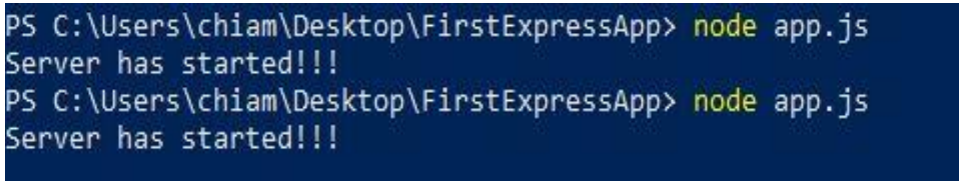
```
app.("/r/soccer/comments/8dvncz/daily_discussion_20180421/")
```

- we could define a pattern to listen for a get request using **route parameters/variables**

```
app.get('/r/:subreddit/:postId', (req, res) => {
    const { subreddit, postId } = req.params;
    res.send(`<h1>Viewing Post ID: ${postId} on
the ${subreddit} subreddit</h1>`)
})
```

Define A Route Pattern using Route Parameters (Screenshot)

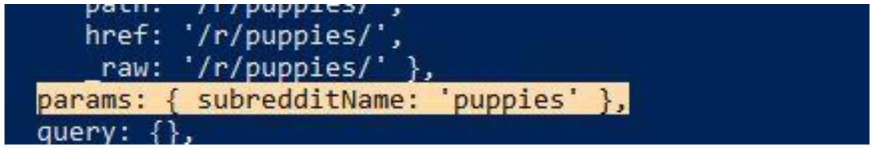
- Restart the web server to view the changes:



```
PS C:\Users\chiam\Desktop\FirstExpressApp> node app.js
Server has started!!!
PS C:\Users\chiam\Desktop\FirstExpressApp> node app.js
Server has started!!!
```

Request parameters

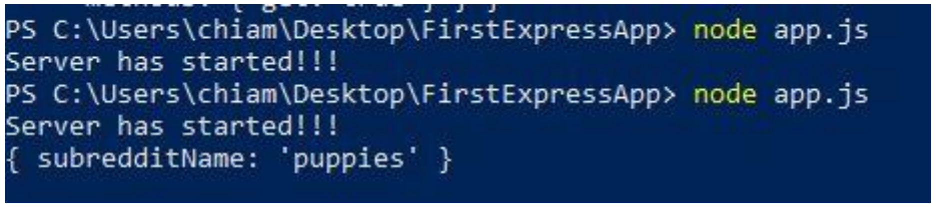
- If we add a statement `console.log(req);` in the callback function, we will get a lot of information coming from that request.
- We can look for **request parameters**. For example, we will get '`puppies`' if we visit http://localhost:3000/r/puppies :



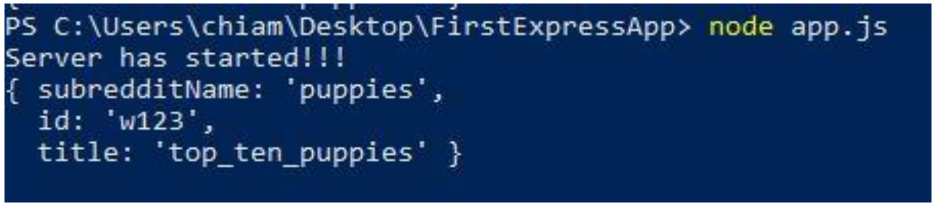
```
    href: '/r/puppies/',
     raw: '/r/puppies/' },
params: { subredditName: 'puppies' },
query: {},
```

Request parameters

- Modify the statement to `console.log(req.params)` in the callback function to get the request parameters information only
- Example, visit http://localhost:3000/r/puppies and http://localhost:3000/r/puppies/comments/w123/top_ten_puppies , we will get the following parameters at console:



```
PS C:\Users\chiam\Desktop\FirstExpressApp> node app.js
Server has started!!!
PS C:\Users\chiam\Desktop\FirstExpressApp> node app.js
Server has started!!!
{ subredditName: 'puppies' }
```



```
PS C:\Users\chiam\Desktop\FirstExpressApp> node app.js
Server has started!!!
{ subredditName: 'puppies',
  id: 'w123',
  title: 'top_ten_puppies' }
```

Define request parameters

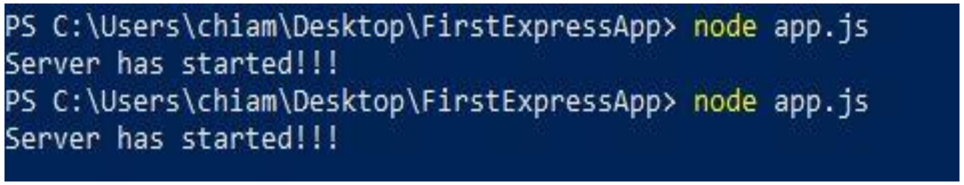
- In app.js file, **modify** the callback function using **request parameters**:

```
app.get("/r/:subredditName", (req, res) => {
    const subreddit = req.params.subredditName;
    res.send("WELCOME TO THE " +
subreddit.toUpperCase() + " SUBREDDIT!");
});
```

- Now we have a **dynamic** web page

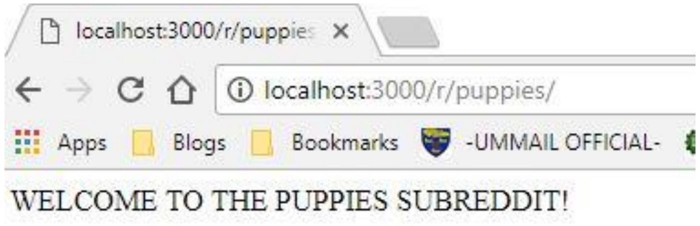
Define request parameters (Screenshot)

- Restart the web server:



```
PS C:\Users\chiam\Desktop\FirstExpressApp> node app.js
Server has started!!!
PS C:\Users\chiam\Desktop\FirstExpressApp> node app.js
Server has started!!!
```

- Go to `http://localhost:3000/r/puppies`, we will get a message "**WELCOME TO THE PUPPIES SUBREDDIT!**"



WELCOME TO THE PUPPIES SUBREDDIT!

package.json file

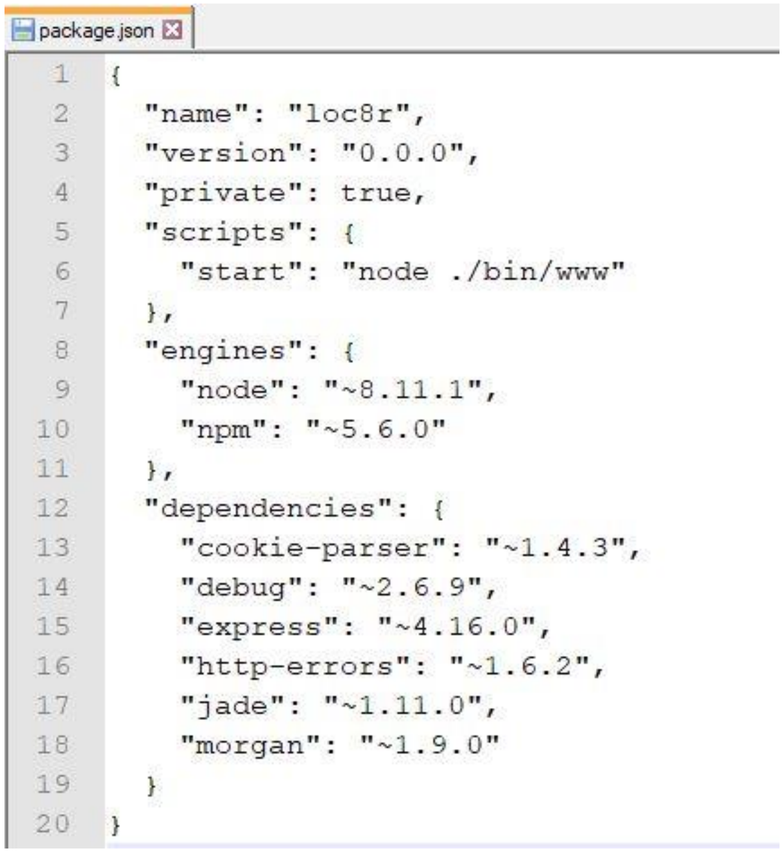
References:

- https://docs.npmjs.com/cli/v11/configuring-npm/package-json
- https://nodesource.com/blog/the-basics-of-package-json

What is package.json file?

- All npm packages contain a file, usually in the `project root`, called `package.json` - this file holds various **metadata** relevant to the project.
- This file is used to give information to npm that allows it to **identify the project** as well as **handle** the project's **dependencies**.
- Also contain other metadata such as a *project description*, the *version* of the project in a particular distribution, *license* information, even *configuration data* - all of which can be vital to both **npm** and to the **end users** of the package.

Sample package.json file



```
{
  "name": "loc8r",
  "version": "0.0.0",
  "private": true,
  "scripts": {
    "start": "node ./bin/www"
  },
  "engines": {
    "node": "~8.11.1",
    "npm": "~5.6.0"
  },
  "dependencies": {
    "cookie-parser": "~1.4.3",
    "debug": "~2.6.9",
    "express": "~4.16.0",
    "http-errors": "~1.6.2",
    "jade": "~1.11.0",
    "morgan": "~1.9.0"
  }
}
```

In `package.json`, the most important aspect is the "**dependencies**". It contains **a list of the packages** and the **version number** of each package that's needed in order for this **application to run**.

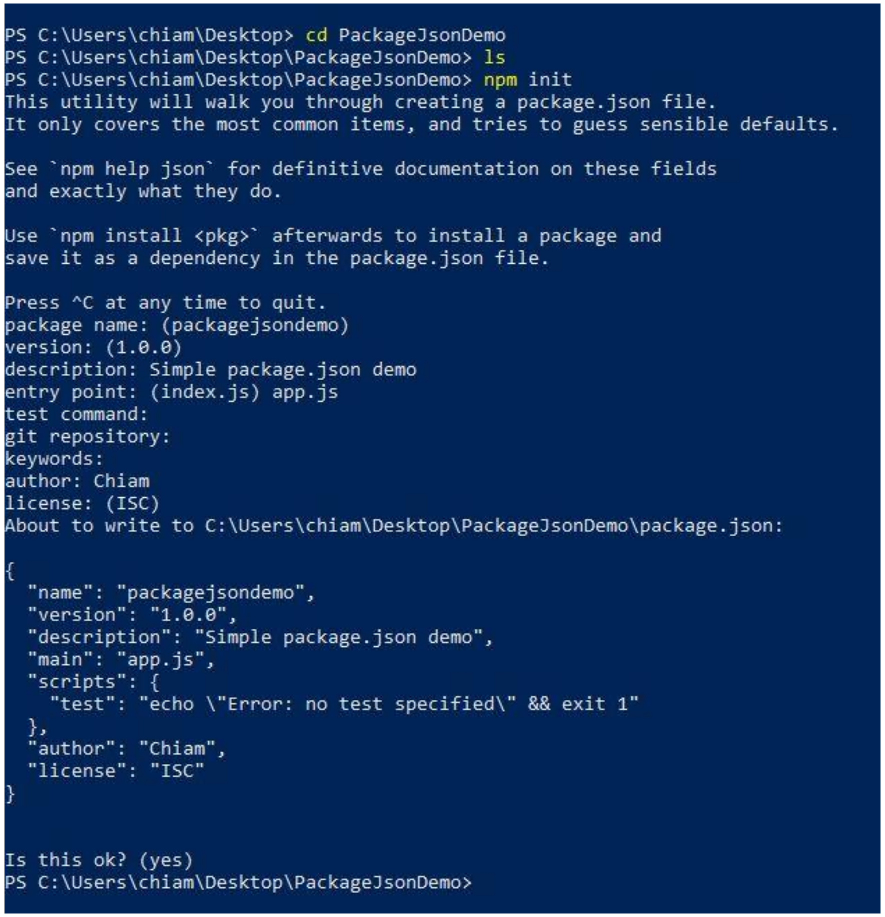
Create a new package.json using `npm init` command

- Create a `package.json` file using this command:

  `$npm init`

- This utility will walk you through creating a `package.json` file.
- It only covers the most common items, and tries to guess sensible defaults.
- When we install a package with `$npm install` and we add on the `--save` at the end
  - It will take the package name and version in automatically save it into our `package.json`

Create a new package.json using `npm init` command (Demo)



```
PS C:\Users\chiam\Desktop> cd PackageJsonDemo
PS C:\Users\chiam\Desktop\PackageJsonDemo> ls
PS C:\Users\chiam\Desktop\PackageJsonDemo> npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help json` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (packagejsondemo)
version: (1.0.0)
description: Simple package.json demo
entry point: (index.js) app.js
test command:
git repository:
keywords:
author: Chiam
license: (ISC)
About to write to C:\Users\chiam\Desktop\PackageJsonDemo\package.json:

{
  "name": "packagejsondemo",
  "version": "1.0.0",
  "description": "Simple package.json demo",
  "main": "app.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Chiam",
  "license": "ISC"
}


Is this ok? (yes)
PS C:\Users\chiam\Desktop\PackageJsonDemo>
```

Summary of steps to create a new Express project from scratch

1. Create a project directory: `$mkdir newdirectory`
2. Change to the new directory: `$cd newdirectory`
3. Create new package.json: `$npm init`
4. Create a new main app.js file:

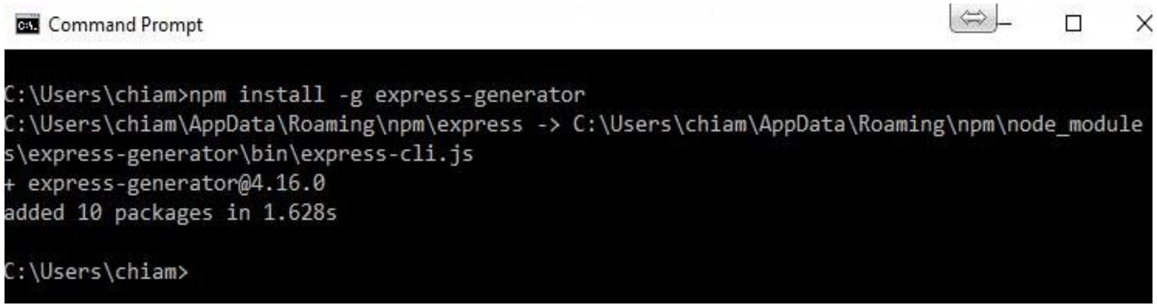
   ```
   $fsutil file createnew app.js 0
   ```
5. Install express: `$npm install express`
6. Edit main app.js file to:
   - Require express and define **const app** to execute express as a function
   - Define app.listen to start the server at a particular port
   - Define routes to handle HTTP requests and responses
7. Start the web server to test the routes on browser: `$node app.js`

App.js

Create web applications using Express Generator

Express Generator Setup

- Like Express JS, **Express Generator** is also a Node JS Module. It is used to **quick start** and develop Express JS applications very easily.
- https://expressjs.com/en/starter/generator.html
- **To install Express JS globally**
  Open command prompt and execute this command: `npm install -g express-generator`

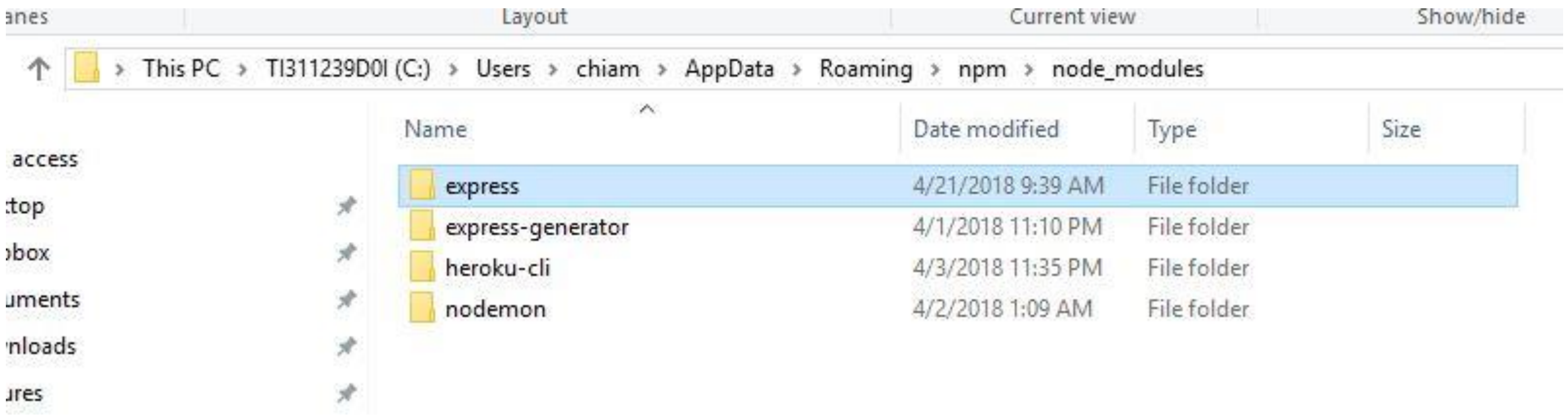


```
C:\Users\chiam>npm install -g express-generator
C:\Users\chiam\AppData\Roaming\npm\express -> C:\Users\chiam\AppData\Roaming\npm\node_module
s\express-generator\bin\express-cli.js
+ express-generator@4.16.0
added 10 packages in 1.628s

C:\Users\chiam>
```

Express Generator Setup

To verify installation:

- If it is installed successfully, we can find a new **folder** at `C:\Users\[Windows_UserName]\AppData\Roaming\npm\node_modules\express-generator`


anes
Layout
Current view
Show/hide
This PC › TI311239D0I (C:) › Users › chiam › AppData › Roaming › npm › node_modules
access
top
box
uments
nloads
ures

Name	Date modified	Type	Size
express	4/21/2018 9:39 AM	File folder	
express-generator	4/1/2018 11:10 PM	File folder	
heroku-cli	4/3/2018 11:35 PM	File folder	
nodemon	4/2/2018 1:09 AM	File folder	

Develop A Simple Express JS Web Application

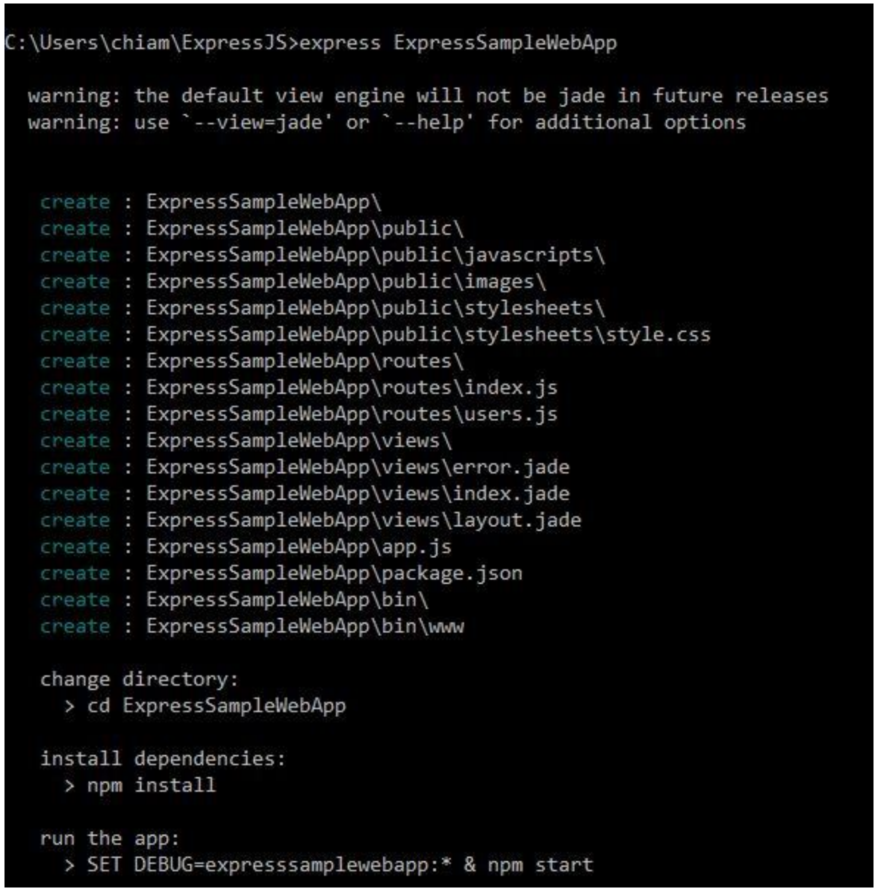
- Develop simple Express JS Web Application using Express Generator Module
- Open command prompt in our local FileSystem and execute the “express” command.
- **“express” command syntax:**

```
express <Your-ExpressJS-Application-Name>
```

Example:

```
express ExpressSampleWebApp
```

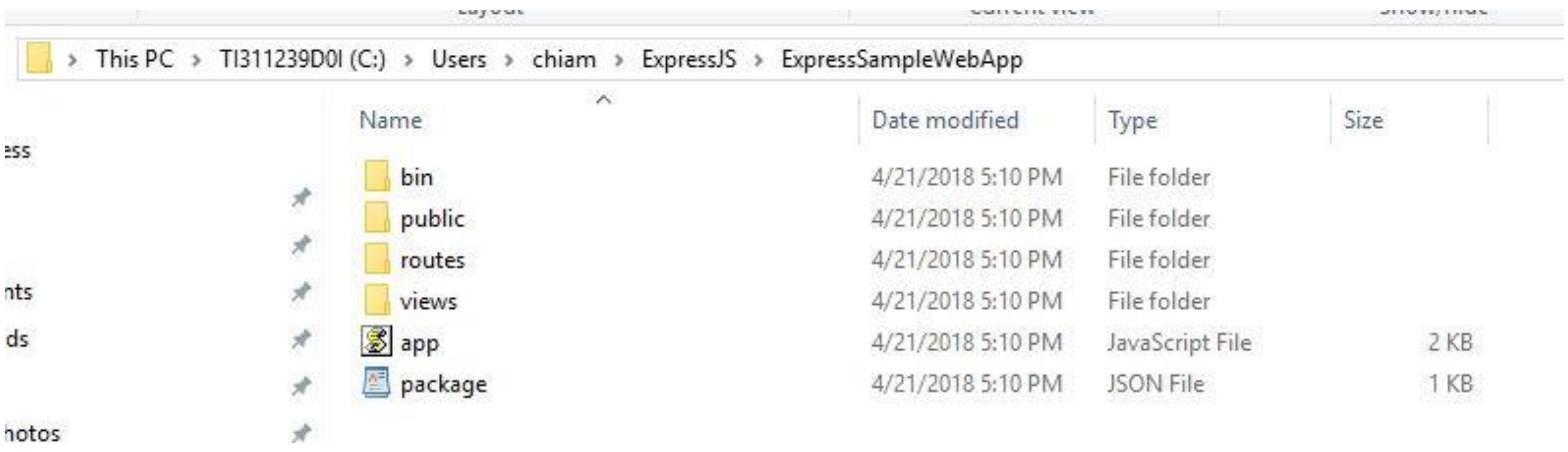
Develop A Simple Express JS Web Application



```
C:\Users\chiam\ExpressJS>express ExpressSampleWebApp

  warning: the default view engine will not be jade in future releases
  warning: use `--view=jade' or `--help' for additional options


   create : ExpressSampleWebApp\
   create : ExpressSampleWebApp\public\
   create : ExpressSampleWebApp\public\javascripts\
   create : ExpressSampleWebApp\public\images\
   create : ExpressSampleWebApp\public\stylesheets\
   create : ExpressSampleWebApp\public\stylesheets\style.css
   create : ExpressSampleWebApp\routes\
   create : ExpressSampleWebApp\routes\index.js
   create : ExpressSampleWebApp\routes\users.js
   create : ExpressSampleWebApp\views\
   create : ExpressSampleWebApp\views\error.jade
   create : ExpressSampleWebApp\views\index.jade
   create : ExpressSampleWebApp\views\layout.jade
   create : ExpressSampleWebApp\app.js
   create : ExpressSampleWebApp\package.json
   create : ExpressSampleWebApp\bin\
   create : ExpressSampleWebApp\bin\www

   change directory:
     > cd ExpressSampleWebApp

   install dependencies:
     > npm install

   run the app:
     > SET DEBUG=expresssamplewebapp:* & npm start
```

Develop A Simple Express JS Web Application

- Now if we access our application folder , we can see the following content.



This PC › TI311239D0I (C:) › Users › chiam › ExpressJS › ExpressSampleWebApp

Name	Date modified	Type	Size
bin	4/21/2018 5:10 PM	File folder	
public	4/21/2018 5:10 PM	File folder	
routes	4/21/2018 5:10 PM	File folder	
views	4/21/2018 5:10 PM	File folder	
app	4/21/2018 5:10 PM	JavaScript File	2 KB
package	4/21/2018 5:10 PM	JSON File	1 KB

Develop A Simple Express JS Web Application

- Change directory to your Express project directory

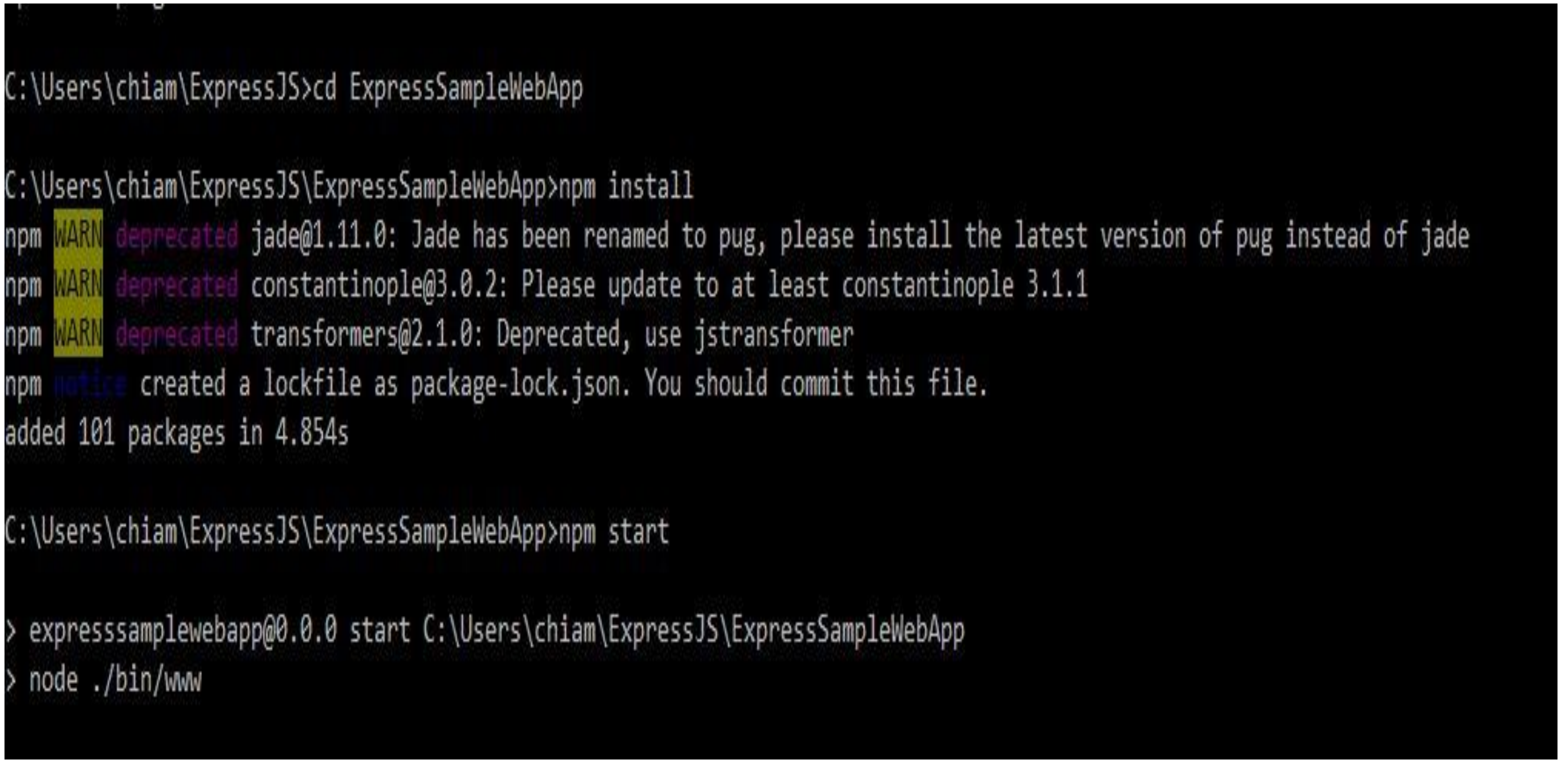
  Execute command:

  ```
  cd  <Your-ExpressJS-Application-Name>
  ```
- To install (build) dependencies of our application:

  Execute command: `npm install`
- To start our Express application:

  Execute command: `npm start`

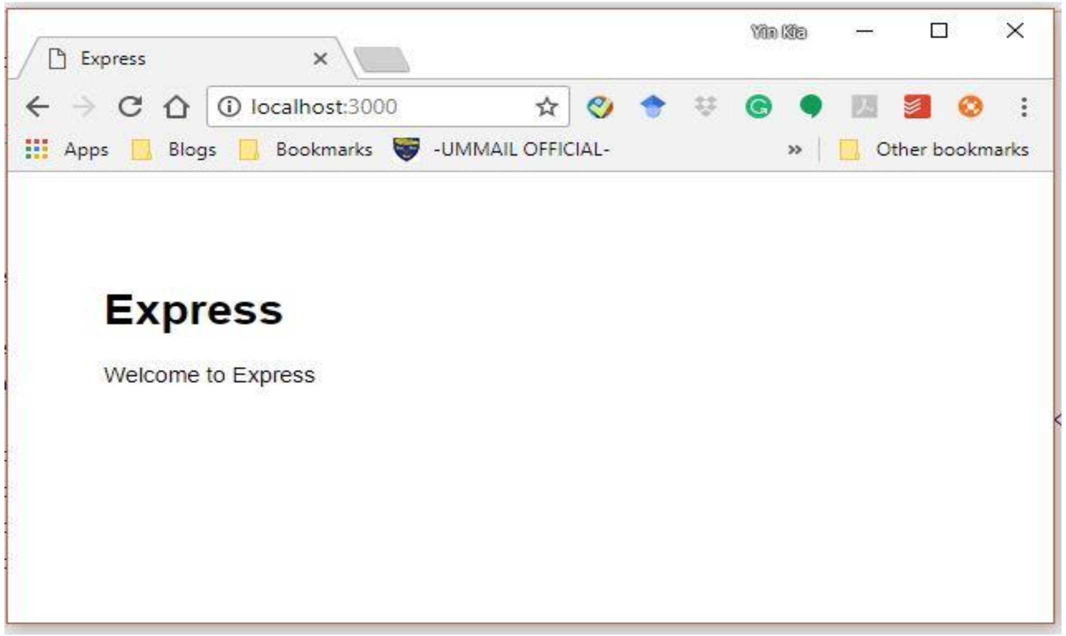
Develop A Simple Express JS Web Application



```
C:\Users\chiam\ExpressJS>cd ExpressSampleWebApp

C:\Users\chiam\ExpressJS\ExpressSampleWebApp>npm install
npm WARN deprecated jade@1.11.0: Jade has been renamed to pug, please install the latest version of pug instead of jade
npm WARN deprecated constantinople@3.0.2: Please update to at least constantinople 3.1.1
npm WARN deprecated transformers@2.1.0: Deprecated, use jstransformer
npm notice created a lockfile as package-lock.json. You should commit this file.
added 101 packages in 4.854s

C:\Users\chiam\ExpressJS\ExpressSampleWebApp>npm start

> expresssamplewebapp@0.0.0 start C:\Users\chiam\ExpressJS\ExpressSampleWebApp
> node ./bin/www
```

Develop A Simple Express JS Web Application

- Access Express JS Sample Web Application using `http://localhost:3000/` from our browser.



Express

Welcome to Express

Summary

In this lecture, we have covered

- An introduction to Express framework
- Create web applications from scratch using Express
- Create web applications using Express Generator