Tagged - Milestone One Testing and Report

Contents

Purpose

The purpose of this document is to detail the development and test results for the first Milestone, "Create Home Splash Page".

Development Notes

Development Environment Setup

Node & Webpack Setup

Node.js is responsible for hosting all of the modules required to run and develop web applications including the Node Package Manager (NPM). Webpack is responsible for bundling all of your project into a single bundle.js file which is placed inside the body tags of your index.html document and will load in all the components that form the web application.

Babel-loader is a webpack specific version of Babel, which is used to convert (parse) ES6 and JSX amongst other things into CommonJS (ES5). Webpack-dev-server will be used to auto-compile when changes to the source code are made and will also force refresh the browser to streamline the development process.

The HTML assembler, file and css loaders are used to process the "app" folder's contents and generate the deployable website in the "build" folder.

The process of setting up a testing and development environment for Tagged required the following steps to be performed in the order listed :

- Download the latest stable release of Node.js as a compiled binary and install
- Create the root project folder with "build" and "app" as sub-directories.
- Create the following directory structure inside "app" -> /app/heading/images
- Through Terminal (command prompt), navigate to the project root folder and use NPM to do the following :
    - **Initialize the Project :** npm init
    - **Install Webpack :** npm install webpack --save-dev
        - *--save* lists the package in package.json
        - *-dev* moves it to the developer section in package.json "for dev use only, not a dependency"
        - Open Package.json and add this to Scripts :

```
"scripts":
{
        "build": "webpack",
        "start": "webpack-dev-server --devtool eval --progress",
        "dev": "npm run build && npm run start"},
}
```

-
    - **Install Babel-Loader :** npm install babel-loader@5.x --save-dev
        - @5.x forces the latest 5.x version **6.x is not fully supported yet**
    - **Install Jquery Library :** npm install jquery --save
    - **Install Autoprefixer & Postcss :** npm install postcss-loader autoprefixer -D
    - **Install CSS & File Loaders :**
      npm i css-loader style-loader file-loader url-loader -D
    - **Install HTML Assembler :** npm install html-webpack-plugin -D

Preparing for Webpack-dev-server

The next step is to prepare the computer for webpack-dev-server to be installed and because we are running a Windows system it will require the following to be taken care of before we can install the package with NPM :

When installing webpack-dev-server you need to install Python 2.7.3
http://www.python.org/download/releases/2.7.3#download

Then on windows 10 add a System Environment Variable to the Path section. Use the "Search Windows" charm in the taskbar to search for Environment Variable. In **system** (not user) add :

The folder where python.exe is located but not to the actual executable file.
Should look something like "c:/Python27/" (just add it to the end of the long list of PATH variables putting a semi-colon ";" before your new entry and then press Apply/Ok or similar.

If you have Visual Studio 2015 but did a "typical" install (which does **not** install c++ stuff), you can install the c++ stuff by going File > New Project and trying to start a c++ project. You will get options to install things. Make sure you have a couple gb on your hard disk. Or install VS 2015 Community and make sure you include the C++ stuff in a custom installation.

https://github.com/nodejs/node-gyp/issues/629#issuecomment-151018292

Webpack-dev-server Setup

Now that the prep work is completed it's time to install and configure the web server so open up command prompt to the root directory of the project and do the following :

- npm install webpack-dev-server --save
- npm install ip -D

At this point, the packages used for detecting the Local IP and running the dev server are installed. The next task is to create the **webpack.config.js** file inside the root directory of your project and add the following :

```
var path = require('path');
var webpack = require('webpack');
var ip = require('ip');
var htmlgenerator = require('html-webpack-plugin');
var autoprefixer = require('autoprefixer');

var ROOT_PATH = path.resolve(__dirname);
var APP_PATH = path.resolve(ROOT_PATH, 'app');
var BUILD_PATH = path.resolve(ROOT_PATH, 'build');
var LOCAL_IP = ip.address();
var MY_PORT = 8080;

module.exports =
{
	entry: './app/index.jsx',
	output:
	{
		path: BUILD_PATH,
		filename: 'bundle.js'
	},
	module:
	{
		loaders:
		[
			{
				test: /\.jsx?$/, // scan for js and jsx files only
				loader: 'babel',
				include: APP_PATH
			},
			{
				test: /\.css$/, // scan for css files only
				loader: 'style!css!postcss-loader',
				include: APP_PATH
			},
			{
				test: /\.png$/,
				loader: 'url?limit=100000&mimetype=image/png',
				include: APP_PATH
			},
			{
				test: /\.jpg$/,
				loader: 'file',
				include: APP_PATH
			},
			{
					test: /\.json$/,
					loader: 'file',
					include: APP_PATH
				}
			]
		},
		devServer:
		{
			historyApiFallback: true,
			hot: true,
			inline: true,
			progress: true,
			host: LOCAL_IP,
			port: MY_PORT
		},
		postcss: function ()
		{
	    return [autoprefixer({browsers: ['> 1%']})];
	},
		plugins:
		[
			new webpack.HotModuleReplacementPlugin(),
			new webpack.NoErrorsPlugin(),
			new htmlgenerator({
				title: 'Tagged',
				filename: 'index.html',
				template: APP_PATH+'/index_template.html',
			})
		]
};
```

Run the Server

To start the web server Two (2) more things need to be done. Firstly the following documents need to be created inside rootDirectory/app/ :

Entry File - index.jsx

```
import './default.css';
```

Main CSS File - default.css

```
*
{
	text-decoration: none;
	padding: 0px;
	margin: 0px;

}
```

The index.html page will be automatically generated and will include code for adding bundle.js (generated by webpack). In web applications, favicons are used on mobile devices as "app icons" when a shortcut is established to your web application on the device Home screen. Generating the favicon and coding it to be picked up by the browser will be covered in the development stage of the report.

Finally we can run the server by opening the terminal to the root directory of the project and entering the following commands to control the server operation :

- **npm run dev :** This starts the server which can be accessed in a browser at your local ip address followed by the port e.g 192.168.1.2:8080
- **Control+C** : This shortcut command will stop the server

The page should currently be blank and is ready for setting up React.

React Setup

Now the server is working and the jsx parser is installed it's time to get the React and ReactDOM packages installed. In the Terminal, navigate to the root directory of the web application and enter the following commands :

- **Install React and ReactDOM :** npm i react react-dom --save

Now React and ReactDOM are installed the development of the web application can begin.

Development of Web Application

The development of Tagged follows several steps beginning with the customisation of the index.html template file.

Index Template File

Inside the "app" sub-directory within the project, create a html file named, "index_template.html" and enter the following code :

```
<!DOCTYPE html>
<html{% if(o.htmlWebpackPlugin.files.manifest) { %} manifest="{% =
o.htmlWebpackPlugin.files.manifest %}"{% } %}>
	<head>
		<meta charset="UTF-8"/>
		<title>{% =o.htmlWebpackPlugin.options.title || 'Add title prop in webpack
config' %}</title>

	{% if (o.htmlWebpackPlugin.files.favicon) { %}
		<!-- Favicons are added --><link rel="shortcut icon" href="{%
=o.htmlWebpackPlugin.files.favicon %}">
		{% } %}
	<!-- Check if viewed on a Mobile -->
		<meta name="viewport" content="width=device-width, initial-scale=1">

	</head>
	<body>
		{% for (var chunk in o.htmlWebpackPlugin.files.chunks) {
		%}<script type="text/javascript" src="{%
=o.htmlWebpackPlugin.files.chunks[chunk].entry %}"></script>{%
		} %}
	</body>
</html>
```

The template file is referenced inside the webpack.config.js file and is used by the html-webpack-plugin to generate our code. The {% %} symbols are used for concatenation and should always have a space between the "%" sign and the code they are wrapping.

Entry File

The next step is to flesh out the **Entry** file, "index.jsx" as shown below :

```
//	"const" is a global variable declaration with a frozen value
//	"let" is a local variable declaration
//	if a variable needs to be dynamic use let instead of var
//	This avoids IIFE wrapping but requires an ES6 transformer (babel)

//	import files used for all pages
import './default.css';

import AndroidIcon from './heading/images/androidicon.png';
import MyManifest from './heading/images/manifest.json';
console.log(MyManifest);

//	Libraries file?name=[name].[ext]!
import React from 'react';
import ReactDOM from 'react-dom';
import Helmet from 'react-helmet'; // usage = <Helmet props=""/>
```

```
//	Component import area -- will need to add routing here later
import HeaderDiv from './heading/headerdiv.jsx';

//	Generates head tags and places the HeaderDiv inside the container
class DocuHeadTags extends React.Component
{
	render()
	{
		return (
			<div className="header">
				<HeaderDiv/>
				<Helmet
					link={[
						{"rel": "icon", "sizes": "192x192", "href":
AndroidIcon},
						{"rel": "manifest", "href": MyManifest}
					]}
					meta={[
						{"name": "mobile-web-app-capable", "content":
"yes"}
					]}
				/>
			</div>
		);
	}
}

//	Declare a new div element to hold the react content
function main()
{
	let container_div = document.createElement('div');
	container_div.setAttribute('class', 'container');

	document.body.appendChild(container_div);

	ReactDOM.render(<DocuHeadTags/>, container_div);
}

//	Run the main function
main();
```

The "main" function will be used to generate a holding div for our React web application. It is bad practice to insert the react code directly into the document.body node as other 3rd party browser plugins etc may manipulate this element breaking the code.

The <HeaderDiv/> component is being imported from a file that hasn't been created yet.

Header Div Component

The following code will need to be added to the HeaderDiv component jsx file which will be created inside a sub-directory below "app" e.g. root/app/heading/headerdiv.jsx

```
import React from 'react';

import taglogo from './images/taggedLogo.png';
import './heading.css';

export default class HeaderDiv extends React.Component
{
        render()
        {
                return (
                        <div className="headBar">
                                <div className="logoPanel">
                                        <img className="tagLogo" src={taglogo} alt="Tagged
Logo"/>
                                </div>
                                <div className="loginPanel">
                                        <form className="authForm">
                                                <span className="formTitle">My
Account</span>
                                                <input className="logFormInput" type="text"
name="username" placeholder="Username"/>
                                                <input className="logFormInput"
type="password" name="password" placeholder="Password"/>
                                                <div className="formBtns">
                                                        <button className="logFormInput"
name="register">Register</button>
                                                        <button className="logFormInput"
name="login">Login</button>
                                                </div>
                                        </form>
                                </div>
                        </div>
                );
        }
}
```

CSS Files

Both the index.jsx **Entry** file and the headerdiv.jsx **Component** are referencing several files including Two (2) Cascading Style Sheets, JSON Manifest document for managing the smartphone Meta Data and several images.

The Default.CSS file previously created should be extended as shown below :

```
*
{
	text-decoration: none;
	padding: 0px;
	margin: 0px;

}

/*CSS for sections that appear more than once in the page*/
p
{
	font-size: 12pt;
	color: #000000;
}

.container
{
	font-family: Verdana, Arial, Helvetica, sans-serif;
	display: block;
}
```

The Heading.CSS file should be created in the following directory : root/app/heading/heading.css and will need the following style information added to it.

```
/*This styling is for the heading component of Tagged
Colours are listed :
logo blue = #007DF1
Ice blue = #D3E6F7
Navy = #000C4D

Header spans the full width of the window adding a Logo Blue */
.header
{
	display: flex;
	justify-content: center;
	background-color: #007DF1;
}

/*Head_bar pushes the logo_panel and login_panel divs apart
A 2.5% gap each side reduces accidental softkey presses for mobile viewers*/
.headBar
{
	display: flex;
	justify-content: space-between;
	max-width: 95%;
	width: 980px;
	height: 100px;
}

/*Controls Auto Scaling and placement of Tagged Logo*/
```

```
.logoPanel
{
	display: flex;
	justify-content: center;
	align-items: center;
	background: rgb(211, 230, 247);
	flex-basis: 81%;
	margin-right: 5px;
}

.tagLogo
{
	max-height: 80px;
	max-width: 177.717px;
	width: 80%;
}

/*Controls styling of the login_panel*/
.loginPanel
{
	display: flex;
	flex-direction: column;
	justify-content: center;
	align-items: center;
	background: rgb(211, 230, 247);
	min-width: 150px;
}

.authForm
{
	display: flex;
	flex-direction: column;
	align-items: center;
	justify-content: center;
	width: 80%;
	height: 92%;
}

.formTitle
{
	font-weight: bold;
	height: 1em;
	line-height: 0.7em;
	text-align: center;
	width: 100%;
}

.logFormInput
{
	box-sizing: border-box;
	margin: 2px 0px;
}

.authForm > input
{
	text-align: center;
	width: 100%;
}

.formBtns
{
	display: flex;
	flex-direction: row;
	justify-content: space-between;
	width: 100%;
}

.formBtns > button
{
	background: radial-gradient(rgba(255, 255, 255, 1), rgba(255, 255, 255, 0));
	height: 1.6em;
	line-height: 0.4em;
	font-size: 0.8em;
	width: 45%;
	border: solid;
	border-color: rgba(67, 138, 201, 0.25);
	border-width: 1px;
	border-radius: 0.25em;
	cursor: pointer;
}

.formBtns > button:hover
{
	background: rgba(67, 138, 201, 1);
	color: rgba(255, 255, 255, 1);
}
```

The Tagged logo, Favicon and Manifest code can be found at :

https://github.com/JaxCavalera/Tagged/tree/Milestone1

Testing

The scripts were parsed by several compilers to ensure that they were free of syntax errors and adhered to best practices as outlined in the following React style guide :

https://github.com/airbnb/javascript/tree/master/react#airbnb-reactjsx-style-guide

The primary goals in the design of the foundational structure to the component and template were as follows :

- Ensure the website would load all content adjusting for mobile and desktop displays appropriately
- Avoid horizontal scrolling
- Ensure the logo and favicon displayed correctly on both mobile home screen and desktop browser windows
- Establish a clear directory structure for the development of Milestone2 which involves setting up the postgreSQL database and using node.js to communicate between the React front end.

Ports were forwarded from the local ip through the router and directed to a public domain using a DNS service called duckDNS.org. This allowed for the testing on both desktop and mobile simultaneously taking advantage of the Hot-Reloading of components made available through Webpack-Dev-Server.