**IS 312 Web Design: TypeScript (TS) for Modern Web Application**

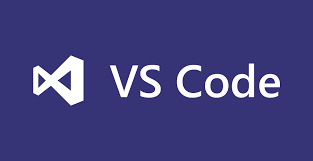
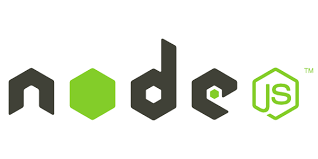
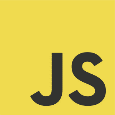
**HOP10 Stand-Alone Web Application, Part 1**

5/30/2020, Developed by Amrutha Vaidyanathan, Class of 2020

9/23/2020, Revised by Kim Nguyen, Class of 2021

School of Technology & Computing (STC)

City University of Seattle (CityU)

**Before You Start**

* The directory path shown in screenshots may be different from yours.
* Some steps are not explained in the tutorial**.** If you are not sure what to do:
  1. Consult the resources listed below.
  2. If you cannot solve the problem after a few tries, ask a TA for help.

**Learning Outcomes**

Students will be able to:

* How Typescript can be used for developing a web application

**Resources**

* Freeman, A. (2019). [*Essential TypeScript: From Beginner to Pro*](https://login.proxy.cityu.edu/sso/skillport?context=147707). Apress. (ISBN 9781484249789)   
  Chapter 15: Creating a Stand-Alone Web Application, Part 1

**Activities**

* Preparation
* Creating a Web Application
* Bundler
* Development Web Server
* Data Model & Data Source
* DOM API
* Deployment

**Preparation**

1. Open the VS Code, git clone <URL>(to download this Github repository to your local machine)

1. Change directory to the corresponding folder of each week. For example: Your work for module 1 should be stored under Module 1 folder; your work for module 2 should be stored under Module 2, and so on:

* cd “Module 8”

1. Now, follow the instructions provided in each folder to complete your Hands-on Practice

**Creating a Web Application**

1. Inside the **Module 08** folder, we are going to create the web application.   
   Open the VS code terminal and check if you are in the correct directory which is Module10(type **pwd** to check it)In the VS code terminal type the following commands. The below command is to create **package.json** file  
     
   >>> npm init --yes

1. Find out which version of typescript is installed in your system  
   >>> tsc –version  
   In my system it is 3.9.3
2. The below command will create the **node\_modules** and **package-lock.json**  
     
   **>>>** npm install --save-dev [typescript@3.9.3](mailto:typescript@3.9.3)  
     
   **Note:** If you have a different version of typescript replace the 3.9.3 with it**.**Any import statements can be resolved using the JavaScript generated by the TypeScript compiler or by the packages installed in the node\_modules folder.
3. Under the **Module10** folder create a new file **tsconfig.json** and add the below code to it.  
   A screen shot of a computer

   Description automatically generated  
     
   The above code configuration tells the compiler to target the ES2018 version of JavaScript, to find the code files in the src folder, and to put the generated files in the dist folder.
4. Under **Module 08**, create a folder called **src** and inside the **src folder** create a file called **index.ts**. Type the below line to the **index.ts.**
5. Let’s compile and run to see the output.  
   >>> tsc  
   >>> node dist/index.js  
     
   You will see the below output.  
     
   The tsc will create a folder called as **dist** which will have the **index.js.**

**Bundler**

A bundler helps web developers manage a group of tools that solve related problems. You can think of it as a “meta-tool,” where each sub-tool is managed by a plugin.

The most widely used bundler is webpack, and it forms a key part in the toolchains used by React, Angular, and Vue.js. Webpack can be complex to work with, but it is supported by a wide range of add-on packages that allow development toolchains to be created for just about any type of project.

1. Type the following command in the VS Code terminal. Run these commands one by one.   
     
   >>> npm install --save-dev [webpack@4.31.0](mailto:webpack@4.31.0)  
   >>> npm install --save-dev [webpack-cli@3.3.2](mailto:webpack-cli@3.3.2)  
   >>> npm install --save-dev [ts-loader@6.0.0](mailto:ts-loader@6.0.0)  
     
   These will add the webpack package to the project/Module8. You can find these files under **node\_modules.** The **webpack** package contains the main bundler features, and the **webpack-cli** package adds command-line support. The **ts-loader** package adds support for compiling TypeScript files and feeding the compiled code into the bundle created by webpack.
2. Let’s configure the webpack. Add a file named **webpack.config.js** to the **Module8** folder with the below contents.  
   A screen shot of a smart phone

   Description automatically generated  
     
   This entry and output settings tell webpack to start with the **src/index.ts** file when resolving the application's dependencies and to give the bundle file the name **bundle.js**. The other settings configure webpack to use the **ts-loader** package to process files with the ts file extension.
3. Run the below command to run webpack and create the bundle file.  
   >>> npx webpack  
   A screenshot of a cell phone

   Description automatically generated  
     
   Under the **dist** folder you will be able to see the **bundle.js**
4. Execute the code in the bundle  
   >>> node dist/bundle.js  
     
   You will see the output as **Web App** in the terminal.

**Development Web Server**

A web server is required to deliver the bundle file to the browser so it can be executed. The Webpack Dev Server (WDS) is an HTTP server that is integrated into webpack and includes support for triggering automatic browser reloads when a code file changes, and a new bundle file is produced.

1. Run the command in the VS Code terminal to install the WDS package.  
   >>> npm install --save-dev [webpack-dev-server@3.3.1](mailto:webpack-dev-server@3.3.1)
2. Under the **webpack.config.js**, add the highlighted portion.  
   **A screen shot of a computer

   Description automatically generated**  
     
   The added portion is to set up the basic configuration for WDS. It tells the WDS to look for any file that is not a bundle in a folder named assets and to listen for HTTP requests on port 4500.
3. Provide WDS with an HTML file that can be used to respond to browsers. Create a folder **assets** under the **Module8** folder. Inside the **assets**, create a file **index.html** with the following contents.  
   A screen shot of a social media post

   Description automatically generated  
     
   When the browser receives the HTML file, it will process the contents and encounter the script element, which will trigger an HTTP request for the bundle.js file, which contains the application's JavaScript code.
4. Start the server by running the code in the Module8 path.  
   >>> npx webpack-dev-server  
   You will see the **Compiled successfully** message after the code has compiled.  
     
   Go to <http://localhost:4500/> to check the output.  
     
   A screenshot of a cell phone

   Description automatically generated
5. Right click on the page and select Inspect. At the bottom of the page the select Console to display the result of the **index.ts** file.  
     
   A screenshot of a cell phone

   Description automatically generated  
     
   A screenshot of a cell phone

   Description automatically generated

**Data Model & Data Source**

A data model refers to the logical inter-relationships and data flow between different data elements involved in the information world. It also documents the way data is stored and retrieved.

1. Under **Module10’s src** folder create a new folder called **data**. In the **data** folder create a file called **entities.ts** with the following code.  
   A black sign with white text

   Description automatically generated  
     
   The Product type is exported so that it can be used outside of the code file.  
   Data Source: Data Source is a name given to the connection set up to a database.  
     
   Let’s create a class that provides access to some local test data. We will be defining an abstract class that provides the basic features and create concrete implementations for each data source.
2. Under **src’s data** folder, create a new file called **abstractDataSource.ts** with the following code.  
     
   A screenshot of text

   Description automatically generated  
   The AbstractDataSource class uses the JavaScript Promise features to fetch data in the background and uses the async/await keywords to express the code that depends on those operations. The class invokes the abstract loadProducts method in the constructor, and the getProducts methods wait for the background operation to produce data before returning any responses.   
     
   Let’s implement the data source class that uses local test data.
3. Under **src’s data** folder, create a new file called **localDataSource.ts** with the following code.  
     
   This class uses the Promise.resolve method to create a Promise that immediately produces a response and allows test data to be easily used.   
     
   A screen shot of a computer

   Description automatically generatedLet’s check if the basic features of the data model are working.
4. Under **src** folder, make changes to the **index.ts** file.  
   A screen shot of a smart phone

   Description automatically generated  
     
   In the above code we are getting the name of the product and calculating the length of the array which has the product list.
5. In the terminal the server has to be running. If it is not running type   
     
   >>> npx webpack-dev-server  
     
   You should be able to see the Compiled successfully message.  
     
   Go to the <http://localhost:4500/> and refresh the page. You should see the Web Placeholder displayed. Do Right click à Inspect à Console tab to check the output of the **index.ts** file.  
     
   A screenshot of a cell phone

   Description automatically generated

**DOM API**

The Document Object Model (DOM) is an application programming interface (API) for HTML and XML documents. It defines the logical structure of documents and the way a document is accessed and manipulated.

Browsers provide the Domain Object Model (DOM) API to allow applications to interact with the HTML document displayed to the user, generate content dynamically, and respond to user interaction.

1. Under **src** folder create a file **domDisplay.ts** with the following contents.  
   A screenshot of a cell phone

   Description automatically generated  
     
   This class will produce an HTML element. The DomDisplay class defines a getContent method whose result is an HTMLElement object, which is the type used by the DOM API to represent an HTML element. The getContent method creates an H3 element and uses a template string to set its content. The data values used in the template string are provided through a property named props.
2. In the **src**’s **index.ts,** replace the existing code with the below.  
   A screenshot of a cell phone

   Description automatically generated  
     
   The above code has to wait for two tasks to be completed before it can display any content. The browser has to complete processing the HTML document contained in the index.html file before the DOM API can be used to manipulate its contents.   
     
   It also has to wait for the data source to obtain its data. The LocalDataSource class uses local test data that is immediately available, but there may be a delay when the data is retrieved from a web service.
3. In the terminal the server has to be running. If it is not running type   
   >>> npx webpack-dev-server  
     
   You should be able to see the Compiled successfully message   
     
   Go to the <http://localhost:4500/> and refresh the page to check the output.  
     
   A picture containing table

   Description automatically generated  
     
   When the tasks mentioned in the step 2 are completed, the placeholder element in the index.html file is removed and replaced with the HTMLElement object obtained by creating a DomDisplay object and calling its getContent method.

**Deployment**

**Push your work to GitHub**

Run the following commands to push your work to the GitHub repository:

Open the terminal from the VSCode by hitting the control + ~ key and type the following command:

**>>> git add .**

**>>> git commit -m “Submission for Module 8– YOUR NAME”**

**>>> git push origin master**