CSCI 492 Final Project Report
Bairon J Vasquez

Project: NodeJS Template Generator

This project attends to create a simple NodeJS template project to help any NodeJS developer get started with their unique application. When the execute is run, it takes multiple parameters but only two of those parameters are required. Those two parameters are -Path which specifies where will the project be created and -ProjectName which specifies the name of the project. This name will also be the name of the root folder of the project. Every parameter takes a string as a value. The following are all the parameters the project accepts:

1. -Path: specifies where will the project be created
2. -ProjectName: specifies the name of the project. This name will also be the name of the root folder of the project
3. -Description: Gives a description to the project. This allows the developer to briefly explain the purpose of this project
4. Author: Sets the creator's name of this project
5. GitURL: Sets the GitHub repository where this project will be maintained

Almost of the parameters given to the executable can be viewed in a file generate named "package.json". This file is responsible for linking dependencies and other requirements for the NodeJS project. The project is able to give a user a simple give on what options are available and how to use them. This can be done by run the executable and passing "-help" to it.

The folders and files are generated recursively using a defined type named "folder" of type 'a. The two main functions responsible for this are "createFolderStructure" and "build". A main function calls "createFolderStructure" with an array of folder and file structure, passing each structure to build so that it is created.

The content of all file (except for package.json) are inside a file named “helperlibrary.ml”, all this file does it to return the content of a specify file.

The biggest challenge of this project was to generate the package.json. The package.json file is being generated by a file named “singlejson.ml”. This file uses a defined type json of type ‘a to consider some of the variation this project needs to create the package.json file. The challenge was to properly define when indentation needed to increase, when commas need to be inserted, and inserting the values coming from the parameters to the JSON file.

The first challenge I was able to solve was the indentation. All I needed to do was to pass an integer of how many times I needed to insert the \t before the string. Since everything was done recursively, I only needed to increase the integer as needed but never decrease because when the recursive call returns, the integer would have the correct indentation number.

The second challenge was when to insert the commas. I decided to have an initial defined structure for the JSON file where all key and value pairs will be inserted first and then the properties that are also objects. This way I can place a comma right every key and value pair except the last key and value pair. Then, for every property object, I insert a common first and then the object itself.

Finally, because most of the parameters are optional, I decided that the Arg.parse was a good choice since it allowed be to work with references type seamlessly. That way I was able to generate the properties of the JSON even if the respective parameters were not provided.