May 2019

Making the Bio-CS Bridge Accessible to High School Teachers and Students

Christian Cedron
Worcester Polytechnic Institute

Making the Bio-CS Bridge Accessible to High School Teachers and Students

An Interdisciplinary Qualifying Project
completed at Worcester Polytechnic Institute

By:
Christian Cedron
May 13, 2019

Professor Carolina Ruiz, Project Advisor
Department of Computer Science

Professor Elizabeth F. Ryder, Project Advisor
Department of Biology and Biotechnology

Professor Robert J. Gegear, Project Advisor
Department of Biology and Biotechnology

Abstract

There is a need to include more interdisciplinary work into the high school curriculum. This project aims to support the Bio-CS Bridge project being developed at WPI by collecting data on local high schools and producing material for teachers and students to learn how to use Angular, a key component of the Bio-CS Bridge.

Authorship Page:

Christian Cedron worked alone on this project, and therefore contributed all work in regards to research, preparing the deliverables and writing the paper. The advisors provided key insight and assistance throughout the project, which was implemented by Christian.

Acknowledgements

I would like to thank my three advisors, Carolina Ruiz, Elizabeth Ryder, and Robert Gegear. Without them this project would not have been possible. I would also like to extend our appreciation to Jen Hardy and Jo vanderSpek from Worcester Technical High School, Jennifer Field and Ronald Cochran from Nipmuc Regional High School and Shari Weaver of the WPI STEM Education Center. They provided key advice for the teaching material to assure it would work well in the classroom. Finally I would like to recognize the key role that Theresa Bruckerhoff plays for the whole of the Bio-CS Bridge project.

Table of Contents

1 Introduction

High school teachers are tasked with preparing and educating the next generation regardless of whether they will enter the workforce or continue their scholastic progress. According to the 2016 Massachusetts Science and Technology/Engineering Curriculum Framework[1], there needs to be an integration of Science and Engineering practices in the classroom, along with application of science topics in an engineering context. This point clearly describes the need for cross-disciplinary education in classrooms regardless of age groups. For this reason, the Bio-CS bridge ("the Bridge") was developed. The Bridge is an innovative approach to high school curricula that provides both teachers and students with a system to tackle biological questions using computational methods.

This Bridge has been in development at WPI for the past two years under a grant from the National Science Foundation (NSF)[2]. It is currently being deployed in a handful of classrooms in Massachusetts to test its effectiveness and assure that the curricula is suitable for as many students as possible.

The goal of this IQP was to first identify which high schools in Central and Eastern Massachusetts teach both biology and computer science courses, and then to create materials that would guide teachers and students through some of the technical aspects of the Bridge. A thorough survey of high schools in Central Massachusetts was conducted by combing through the Massachusetts Department of Education high school database and selecting schools whose websites provide information from which it can be inferred that they offer both biology and computer science classes. Once that was done, a set of video tutorials was made to teach the use of Angular, a TypeScript based web application framework, and one of the technologies presented within the Bridge.

2 Background

2.1 Web Development

When developing a website there are many technologies that can be used. However, the most common by far are HTML, JavaScript, and CSS. All three of these work together to display the content web developers want. The Bridge project is no different. In order to make the main project websites, the team behind the Bridge used HTML, CSS and Angular. TypeScript is an object-oriented extension of JavaScript. Angular is a TypeScript based framework that expands the existing tool and allows for a wider range of design freedom while maintaining simplicity for the developer.

Each tool plays a different role and it is important to learn the differences in order to use them correctly. HTML stands for Hyper Text Markup Language and it is the meat of a website; it provides the content, be it an image, some text, a table, or any other form of content. It does this through a series of ".html" files that web browsers can load. This is accompanied by CSS or Cascading Style Sheets. CSS provides the aesthetics of a website, or the style. The cascading portion comes from the fact that multiple style sheets can be loaded at once in order to simplify page design. JavaScript provides the functionality of the site. It is a scripting language whose scripts can be embedded with the HTML and with it the developer can manipulate how the site reacts to user input. TypeScript provides additional support to web development including type-checking and enhanced scalability. In our case, rather than using raw JavaScript or TypeScript, Angular was used. Angular expands the basic functionality provided by JavaScript while also simplifying the ties between the website and the server behind it. Angular further simplifies the development process by assisting with the hosting of the website.

2.2 The Bio-CS Bridge Computational Infrastructure

There are three main components in use by the Bridge project: the Bio-CS Bridge website[3], the Beecology project website[4], and the Beecology Bee Identification web app[5]. The Bio-CS bridge site hosts the information about using, joining and development of the Bio-CS Bridge. Meanwhile both Beecology sites are a working example of what the Bridge can achieve. The Beecology project is a working effort to "recruit[ing] citizen scientists from across the New England region to digitally collect and submit ecological data on native pollinator species using our freely available web apps."[3] At the website for it, you can find all the information needed to begin collecting information about local pollinators and submit it to the database for further analysis. The tool used to gather and submit this information is the Beecology Bee Identification web app. This tool leverages Angular's ability to facilitate a connection between a site and a server to allow the users to record data about the pollinators they find in the wild and later submit it to the central database.

3 Methodology

The central goal of the project was to develop a set of materials that teachers could use in the classroom in order to facilitate the students' understanding of what the Bridge does and how to use it. Additionally, we needed to also identify potential future users of the Bridge and its accompanying materials. This section describes the methodology that we used to tackle these goals.

3.1 Identifying Potential High School Adopters

We analyzed the surrounding area in Massachusetts to identify high schools that taught both biology and computer science as these high schools could potentially benefit from the Bridge. Additionally, we wanted to construct a complete list of teachers who taught biology, environmental sciences or computer science, along with their contact information.
The listing of courses was also important because it allowed us to take note of any classes offered that seemed to be best suited to implement the Bridge into the curriculum.This process was completed using the school websites and directories.

In order to gather this data we first navigated to the official Massachusetts Department of Education listing for all primary and secondary schools. Then from here we navigated to the entry for every public high school in each city in each county. Once at the site we searched for the schools' program of studies (PoS) and teacher directory. Neither of these were guaranteed to exist but both were common and contained all the needed information readily available when there did exist. Once we had either or both documents, we searched the program of studies for any biology and computer science classes offered, and the teacher directory for the teachers of any courses we did find. In the event that a program of studies could not be found we would search the directory for any teachers

listed under Biology or Computer science. Finally we compiled a document containing each school, the highest level course they taught in both biology and computer science, a list of all the teachers under each department, and contact info for each teacher.

Another major use of the data we collected was to assess the viability of the Bridge in the state. There were concerns about whether the number of high schools that offer computer science courses across Massachusetts was too low. When developing the methodology we made sure to include some way to quantify the proportion of schools that offered any level of computer science. With this we can see if there is enough of an existing base in the schools today.

3.2 Development of Educational Material

The next step in facilitating the adoption of the Bridge is to make sure that both teachers and students find using and learning about the Bridge simple and intuitive. To do this we needed to create material on how to use the Bridge and its respective elements. The goal for the team was to make a few tutorials that would help teachers and students download, install, and get started with the Bridge, and that would serve as examples for future tutorials to come. We decided that the best place to start would be with the JavaScript framework: Angular.

There are tutorials already available online on what Angular does and how to use it. However good tutorials are scarce and even good tutorials may be either too general or too advanced. For this reason the tutorials we made were focused on being simple to follow, flowing well from one to the next in a logical order, and containing information pertinent to the Bridge with as little accessory information as possible. Furthermore, the tutorials had to not only explain how Angular works, but also how Angular works in the context of the Bridge. These goals were kept in mind during the writing

and production of the tutorials to make sure they were at a level that was both engaging and understandable to a high-schooler whose knowledge of both computer science and biology may be limited.

To achieve these goals we made sure that ties were frequently made back to the Beecology website. This allowed the students to have a visible output to refer to at any time. We also made sure to keep the videos short and as limited in scope as possible. That was intended to not bore students or make the material seem more complex than it really is.

The tutorials were split into two segments: the video, and the accompanying documentation. The videos are a visual representation of the target information, through this students who learned by watching information being used would learn. The videos are also meant to be watched in their entirety and used to provide context for when certain skills and knowledge are to be used. The document on the other had provided a textual representation of the information. This allows for faster and more versatile referencing of the material, it also allows a medium for us to provide any students with any code or references used in the video. This idea was adapted from many popular online food recipe sites. These site are usually structured such that a YouTube video is embedded at the top of the page and below it is a block of text where you can find the recipe being cooked in the video.

Making a tutorial involved several steps. Firstly, the scope of the video was defined. If the scope was deemed to be a coding skill, we then searched in the Beecology code for an implementation of that skill and used it in the tutorial. In order to decide what we wanted to make into a tutorial we looked into the examples we wish we had had when learning Angular, and what we thought to be the key skills that a beginner should learn. Occasionally the identified examples in the Beecology code were used in a context that would require too much information; when this happened we simplified the

example to include only what was necessary to explain the skill being taught without oversimplying or rendering the example irrelevant for the Bridge. This does not mean that the examples would have no required background however. If there were parts of an example that we expected the student to learn about, we made sure to note them and, when needed, also prepare separate videos to teach the expected background in isolation. Next, we broke down the example into key points. We isolated each key part of the example and then each key point was broken down into components. These components would later become the stepping stones that the tutorial would move through in order to teach the information or skill that the tutorial was meant to address. A draft video would be recorded, making sure to hit all points we felt were key and attempting to remove any superfluous information. The video would then be presented to members of the Bio-CS Bridge group, including the project advisors and participating high school teachers, for review and critique. Next we would re-record the video taking into account the points provided. These two steps would continue until a respectable video was produced. At this point the document would be written. In this document ("the recipe") we included: the expected background, all links and documents used in the video, links to any explanation of the expected background, and a breakdown and explanation of all key points covered in the video. Following this, another cycle of presentation and review would occur until the document was deemed satisfactory. The videos were recorded using Open Broadcaster Software and then edited using Adobe Premiere Pro. WPI's Global Lab recording studio was used; this space provided us with high quality recording hardware and software. The recipes were written using Microsoft Office Word and then exported to PDF. The video editing was broken into three main parts: first the raw video and audio would be cut together and trimmed to produce a fluid video, then the audio was touched up to adjust volume and remove as

much background noise and echo as possible, and finally written text was added on the screen in the video to provide the viewer with some additional information.

4 Results

4.1 Identifying Potential High School Adopters

The original scope of the project included all of Massachusetts and in turn all of the high schools therein, however, we quickly identified that this was not a realistic scale due to the sheer number of schools in the entire state and the difficulty in finding relevant information on their websites. To address this, the scale was reduced to only include Worcester county and Middlesex county. These two counties combined had approximately 120 high schools and 600 teachers.

The full table can be found online at: https://drive.google.com/open?id=1E5mEeDzAKl2gAl49zzePRUK34K-KCyxID-JkyVpwjZ8. Within you can find all of the publicly available information about every teacher that seemed relevant to the study.

Our methodology did provide for adequate information, albeit with notable flaws. Most notably is the lack of consistency in the documentation available for each school. Most schools did have a program of studies (PoS) and some sort of directory of teachers; however not all PoSs and directories were up to date, and the directories were not always sorted by department or were even not sorted at all. Another issue encountered was the method by which teachers could be contacted. We quickly ran into directories that would only provide extension numbers for the teachers and some that would only provide an in-site portal to email the teachers. This was an issue as it makes sending an email to many teachers at once complicated. A further issue we encountered had to do with the different layouts for the PoS. Some schools had the computer science courses under the math department, others had it under the science department, others had a separate department for computer science while finally, some schools included computer science as an art course. With all these different placements for

the computer science courses we did spend more time than expected going though all the PoSs to make sure the school did not offer any computer science courses and not that we simply could not find them. Finally, there were some cities, like the city or Worcester, that do not provide information about the courses being offered or the teachers at each school. For this city we listed all the teachers we could find.

Even with all these issues the methodology did prove to be sufficient and well founded. Taking into consideration that: the PoSs usually provide a complete listing of the offered courses, most websites are simple to navigate and search, and all public schools are listed in the Massachusetts DoE website, we feel comfortable that what we gathered is representative of the vast majority of high schools in the two target counties.

Another major use of the data we collected was to assess the viability of the Bridge in the state. In particular, we wanted to determine whether the number of high schools that offer of computer science courses across Massachusetts was too low. One of the factors taken into account in our survey of high schools was the highest level of biology and computer science offered.This was to make sure that there was some course being offered at the school that could make use of the Bridge. This was later determined to not be a perfect metric, AP courses were deemed to not be the best candidates for the Bridge as they had quite restrictive requirements that would not allow for enough deviation from the AP test material to be able to incorporate the Bridge. The one exception to this is AP computer science principles. This course is meant to be “language agnostic” meaning that the exam is not based on any one programming language, but rather on the general theory behind Computer Science. With this in mind, an application like the Bridge would provide a solid foundation for projects and we believe it will enhance the learning experience.

With the data collected, analysis provided some insight as to the viability of the project. In Worcester county 22 schools did not have either a Biology or a Computer science courses out of approximately 60 schools studied and in Middlesex county only 5 out of approximately 55 schools studied. One caveat to this is that schools may appear to not have a Biology or Computer Science course, when in fact they do. This occurs when the information pertaining to the school could not be found; this is especially notable in the case of Middlesex county as every school listed as not having a CS or Biology course was missing the PoS.

These numbers were considered more than acceptable for viability and therefore we moved on with the production of materials for the Bridge.

4.2 Development of Educational Material

Our methodology allowed us to produce a set of tutorials that provide enough information for the students to use the Bridge. Furthermore, by providing the information in both a textual and visual formats we made sure to include two of the major learning styles; and if the student follows along with the examples, the two formats may reinforce learning. The methodology also allowed us to consolidate the information, providing only what needs to be taught and not having the teachers spend class time going over information that the students may not need for their use of the Bridge.

The three topics we decided to expand on were setting up a development environment, exploring the structure of an Angular project, and adding an element to an Angular project.

The "Setting up an environment" tutorial includes downloading and installing the required software to write execute Angular code. We decided to use VisualStudio Code as our IDE of choice, it is a free, multiplatform environment that can open CSS, HTML and Angular files making it perfect

for this application. Appendix A contains the textual documentation ('recipe') for this tutorial.

The "Explanation of an Angular project" tutorial includes opening each file type and folder of an Angular project. A regular Angular project includes four file types, the HTML file, the CSS file, and two angular files, one is a testing file and the other is the main file. We do not cover the testing as it seemed out of scope and not a file that the students would be using when working with the Bridge. These four file types are used time and time again throughout an angular project in order to maintain a modular format. Appendix B contains the textual documentation ('recipe') for this tutorial.

Finally, the "Adding an element to an Angular project" tutorial was the most technical step, which included writing CSS and HTML for a new webpage as well as adding it to a routing table and the angular loading script. Appendix C contains the textual documentation ('recipe') for this tutorial.

The tutorial ‘Recipes’ were written with care to include all of the relevant content provided in the videos, but they occasionally had to do so in an order different from the video due to stylistic differences in the two media. We felt this did not lower the quality of the material presented and therefore left it as is.

There are of course issues with the methodology we chose to follow. Firstly, the designation of scope and breakdown of key points did take significant time to do. This means that a lot of time was spent in a way that does not directly produce tangible results. Furthermore, the inclusion of a video means that access to high quality audio and video recording/editing tools was needed. This may pose a barrier for future creation of tutorials. Finally, and arguably most importantly, by choosing to only include information that is needed for students to use the Bridge and not much additional background information, students may have a lack of

understanding on how to use Angular for other tasks. We decided that this was not much of a concern for us, as students that choose to continue on a computer science path can learn these skills along with any other projects they encounter later on, building upon the knowledge they learned from these tutorials.

5 Conclusion and Future Work

Our project goals were met by collecting data on the high schools in Massachusetts in order to determine viability of the Bridge project within the state; And by producing teaching materials for the Angular web application framework in the context of the Bridge.

It would be useful in future work to expand the amount of data collected on high schools that offer both biology and computer science courses, as of now there is only data on two Massachusetts counties This data collection may be expanded to more Massachusetts counties or even some counties in other areas of New England.

Future work needs to include also making the teaching material produced during this IQP available online. As of now there is no place to put the tutorials for teachers and students to access them. We recommend a dedicated section of the Bio-CS Bridge website, where each individual tutorial can have its page with a directory style page available for users to find what they want to watch. Furthermore, there are also more tutorials that can be made. Some topics we would recommend writing tutorials for include starting a modular Angular project, how to connect an Angular project to a database, using the browser cache to store information and how to set up an Angular project for hosting from a server.

References

[1] Massachusetts Department of Elementary and Secondary Education (2016, Apr.) “2016 Massachusetts Science and Technology/Engineering Curriculum Framework” [on-line],pp. 19-20, 66-94. Available: http://www.doe.mass.edu/frameworks/scitech/2016-04.pdf [May 6, 2019]

[2] E. F. Ryder, R. J. Gegear, C. Ruiz, S. Weaver. “Building Educational Bridges Between Computer Science and Biology Through Transdisciplinary Teamwork and Modular Curriculum Design.” NSF Award #1742446 Internet: https://www.nsf.gov/awardsearch/showAward?AWD_ID=1742446&HistoricalAwards=false, Jul. 26, 2017[May 6, 2019]

[3] “The Bio-CS Bridge.” Internet: https://biocsbridge.wpi.edu/website/home, [Apr. 23, 2019]

[4] “The Beecology Project.” Internet: https://beecology.wpi.edu/website/, [May 1, 2019]

[5] “The Beecology Project Web App” Internet: https://beecology.wpi.edu/webapp/app/home, [May 3, 2019]

Appendix A.

Tutorial 1: Setting Up The Environment

Tutorial 1's video and recipe can be found at:
https://drive.google.com/drive/folders/1FmRp5LjkJbhWH607PygDbMh5KDV9IjbT
For convenience, the recipe for this tutorial is also included below.

Expected Background:

- Basic Command Line Knowledge
  - See links on the right if you need a tutorial

Pre-Requisites:

- Functioning Internet connection
- Administrator Privileges
- Enough disk space to install the programs (amount may change with updates)

Links:

Windows Command Line Commands
Unix Command Line Commands
Visual Studio Code Download
NodeJS Download
Angular NPM Page

Setting up Virtual Studio (VS) Code:

What is it?

Visual Studio Code is an Integrated Development Environment (IDE). This is where we will be writing and executing our code from. This tool allows us to develop from within it with minimal use of external tools. (In the case of this tutorial series, none.)

Setup:

1. Download the installer from the link provided above.
   a. First find your operating system (and bit version if applicable)
   b. Different Installer types: (windows only)

Type	Requires Admin?	Installs for
User Installer	No	Current user only
System Installer	Yes	All Users
.zip	No	Not an installer. Unpack to use VS Code from a flash-drive or disk without installing. NOTE: Won't update automatically.

2. Install
   a. Launch Install wizard
   b. Read and accept the license agreement
   c. Set correct install location (Default is ok)
   d. Continue until you have five tick boxes, select the ones you want.
      i. I recommend all of them (see below)

Name	**What it does**
Create Desktop Icon	Makes a VS Code icon on your desktop
Add "Open With Code" for file	Lets you right click a file and get an option to open it with VS Code
Add "Open With Code" for folder	Same as above but for Folders
Register VS Code for filetypes	Lets the computer know it can use VS Code to open a bunch of different file types
Add to PATH	Adds VS Code to the OS PATH variable. **YOU NEED THIS ONE!!**

e. Install
f. Close the wizard when installation is complete

Setting up NodeJS:

What is it?

NodeJS is a JavaScript engine on which Angular runs. It is a common backbone for web development.

Setup:

1. Download the installer from the link provided.
    a. It should auto-detect your operating system.
    b. Select the version you want
        i. I recommend the LTS (Long term support)

Name:	**Pros:**	**Cons:**
LTS	Stable Easy to find information on	May not have some super new features
Current	New features and changes added constantly	May not be supported by add-ons May be hard to find Documentation Potentially unstable

2. Install
    a. Launch Install wizard
    b. Read and accept the license agreement
    c. Set correct install location (Default is ok)
    d. Keep defaults in the rest of the installer
        i. Keep clicking next
    e. Click install
        i. Requires Administrator privileges
    f. Close the wizard when installation is complete

Setting up Angular:

What is it?

Angular is the platform that we will use to build our website/web application. It is a powerful tool that allows programmers to make complex websites simpler by using the features and libraries built into Angular.

Setup:

1. Open a new command line console
2. Run in your command line console:

```
npm install -g @angular/cli
```

3. Wait for it to run

NOTE: Restart your machine at this point!

Verify Everything installed correctly

1. VS Code
    a. You should be able to run it with no errors or issues.
    b. If you get a PATH error, make sure you rebooted your machine and that you installed it with the "Add to PATH" option selected.

2. NodeJS
    a. Run
    ```
    node -v
    ```
    b. It should return whatever version number you downloaded
        i. LTS is 10.15.3 as of April 2019
    c. If you can't run this, make sure you rebooted your machine and that you followed all the install instructions correctly.
3. Angular
    a. Run
    ```
    ng --version
    ```
    b. Should return some ASCII art and a handful of version numbers.
    c. Verify that the version of "Angular CLI" is correct
        i. See Angular's NPM page for latest version number

Making a new Angular Project:

1. Open a terminal where you want you project folder to be
2. Run

```
ng new <appName>
```

3. I use the name my-first-app
4. Click enter to accept defaults whenever prompted for an option
    a. Should be twice at the start only
5. Wait a while, this can take several minutes
6. Once it is done, you will have a folder that contains all the angular files and structure that will allow your program to function as an angular project

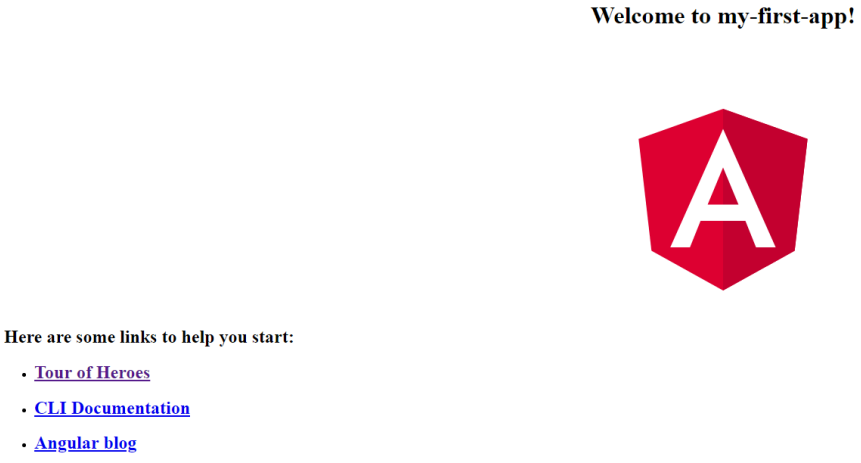
Serving the Angular Project:

7. From within the new folder run

```
ng serve [--open]
```

    a. The –-open parameter automatically opens your browser to the website when the site is ready. URL is "localhost:4200"
    b. The –-open parameter is optional

8. The website should look like this



Modifying the Project:

1. Open the project in VS Code
2. Navigate to the "./src/app/" folder
3. Open app.component.html
    a. It should look like this (below)

```
<!--The content below is only a placeholder and can be replaced.-->
<div style="text-align:center">
  <h1>
    Welcome to {{ title }}!
  </h1>
  <img width="300" alt="Angular Logo" src="
</div>
<h2>Here are some links to help you start: </h2>
<ul>
  <li>
    <h2><a target="_blank" rel="noopener" href="https://angular.io/tutorial">Tour of Heroes</a></h2>
  </li>
  <li>
    <h2><a target="_blank" rel="noopener" href="https://angular.io/cli">CLI Documentation</a></h2>
  </li>
  <li>
    <h2><a target="_blank" rel="noopener" href="https://blog.angular.io/">Angular blog</a></h2>
  </li>
</ul>
```

    b. This is almost entirely standard HTML, except for {{ title }}
    c. The {{ title }} notation tells angular to search in the app.component.ts and find the title variable
    d. By looking at the site that opened when we served the project we know that {{ title }} should be (in my case) "my-first-app"
4. Open app.component.ts
    a. It should look like this (below)

```
1   import { Component } from '@angular/core';
2
3   @Component({
4     selector: 'app-root',
5     templateUrl: './app.component.html',
6     styleUrls: ['./app.component.css']
7   })
8   export class AppComponent {
9     title = 'my-first-app';
10  }
```

b. The templateUrl points to the HTML file that this angular project should load
c. The styleUrls is an array of all the css files we want this project to load
d. In the AppComponent class we find that variable "title" we were looking for

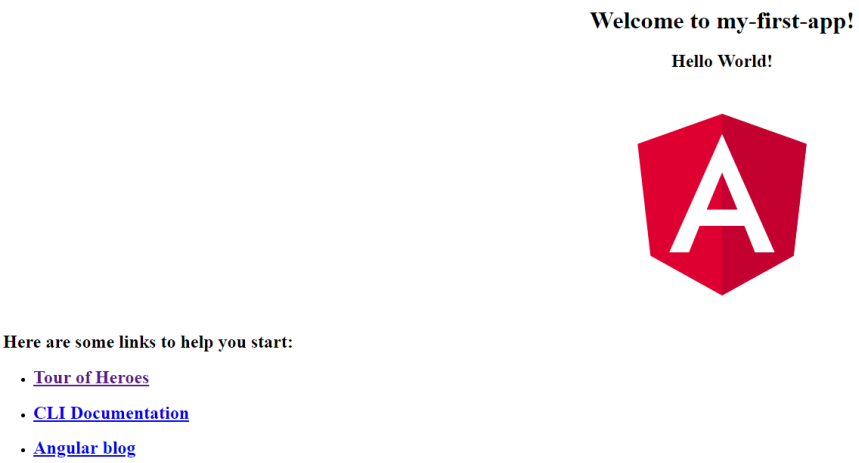
5. We can now modify the code to add a subtitle
   a. Add a variable called subtitle to the App Component class. The class should now look like this

```
export class AppComponent {
  title = 'my-first-app';
  subtitle = 'Hello World'
}
```

   b. We can now reference it in the HTML file
   c. In that file, add a level 2 header under the level 1 header and have it call the subtitle variable you made. You can add extra text like in the header or not like in my example. The top of the HTML should now look something like this

```
<!--The content below is only a plac
<div style="text-align:center">
  <h1>
    Welcome to {{ title }}!
  </h1>
  <h2>
    {{ subtitle }}
  </h2>
```

   d. When you save both files, the terminal that you had serving the webpage should update and your browser should quickly reflect the changes you made.
   e. In my case my website now looks like this (below)



You have now successfully modified the Angular project. Feel free to poke around the files and see what you can do. The CSS file is empty, but the syntax isn't affected so feel free to play with that. Add more variable, add more content in the HTML. In a later tutorial we will go through and talk about what each file does and how we can expand this structure to a fully modular system capable of implementing a website with multiple web pages.

Appendix B.

Tutorial 2: Angular Modular Design

Tutorial 2's video and recipe can be found at:
https://drive.google.com/drive/folders/1FmRp5LjkJbhWH607PygDbMh5KDV9IjbT
For convenience, the recipe for this tutorial is also included below.

Expected Background:

- Knowledge of HTML
- Knowledge of CSS

Links:

Bio-CS Bridge Code

Downloading the Angular code from the Bio-CS Bridge website:

- Navigate to the link above
- Download the compressed folder
- Unpack the compressed folder.

Partial Structure of our modular Project

- <Project Name>
  - src
    - app
      - <ComponentName>
        - componentname.component.css
        - componentname.component.html
        - componentname.component.spec.ts
        - componentname.component.ts
      - <ComponentName>
        - componentname.component.css
        - componentname.component.html
        - componentname.component.spec.ts
        - componentname.component.ts
      - <ComponentName>
        - componentname.component.css
        - componentname.component.html
        - componentname.component.spec.ts
        - componentname.component.ts
      - .....
      - app.component.css
      - app.component.html
      - app.component.spec.ts
      - app.component.ts
      - app-routing.module.ts
      - app.module.ts
      - material.module.ts

The structure is made up of two major portions:

- The yellow portion contains all the "Components", these components are all the major articles of our website. This includes each individual page, the header and the footer. The yellow portion is what the user of the site will be interacting with.

- The green portion is the "Machinery" of the website. These files tell Angular what needs to be loaded and how to load them.

Components:

- The four major files and their contents are:
  - .css
    - Standard CSS, this will outline the way that the component will be styled. From colors, to text size, even to margin width.
  - .html
    - Standard HTML, this file describes the body of the component. With the HTML we describe the content that we want to be displayed within this component.
  - .spec.ts
    - This file is used for testing the component and make sure all the components are playing nice. We will not be working with it.
  - .ts
    - Standing for TypeScript, these files are the brain of the Angular component.
    - This controls the loading of each part of the component.
    - This will link the HTML and CSS files to this component.

Machinery:

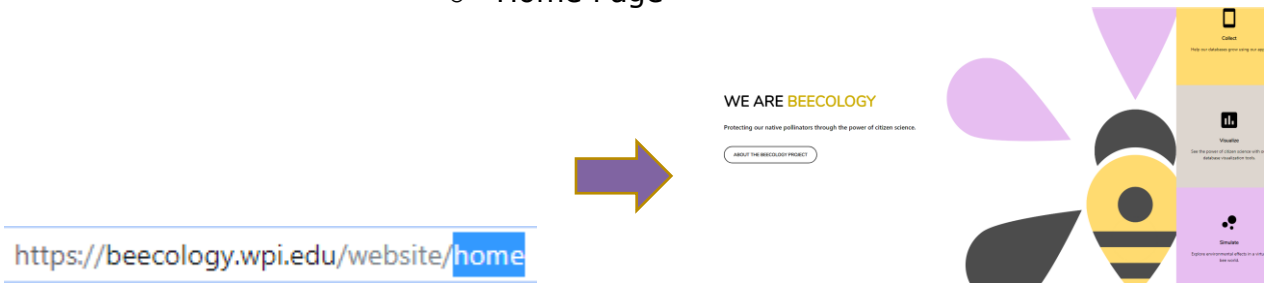
- The core Angular files still follow the rolls described above.
- They work on the whole project not just individual components.
- The major files are:
  - app.component.*
    - All the app.component files are leftovers from when the project was started. They served the roles described above.
  - App-routing.module.ts
    - Allows us to link from one webpage to another through the use of buttons on the different components.

```
import { FaqPageComponent } from './faq-page/faq-page.component';
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';

import { HomePageComponent } from './home-page/home-page.component';
import { TeamPageComponent } from './team-page/team-page.component';
import { ParticipatePageComponent } from './participate-page/participate-page.component';
import { LearnPageComponent } from './learn-page/learn-page.component';
import { NewsPageComponent } from './news-page/news-page.component';
import { BioCsBridgePageComponent } from './bio-cs-bridge-page/bio-cs-bridge-page.component';

const routes: Routes = [
  { path: '', redirectTo: 'home', pathMatch: 'full' },
  { path: 'home', component: HomePageComponent },
  { path: 'news', component: NewsPageComponent },
  { path: 'team', component: TeamPageComponent },
  { path: 'participate', component: ParticipatePageComponent },
  { path: 'learn', component: LearnPageComponent },
  { path: 'faq', component: FaqPageComponent},
  { path: 'biocsbridge', component: BioCsBridgePageComponent }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

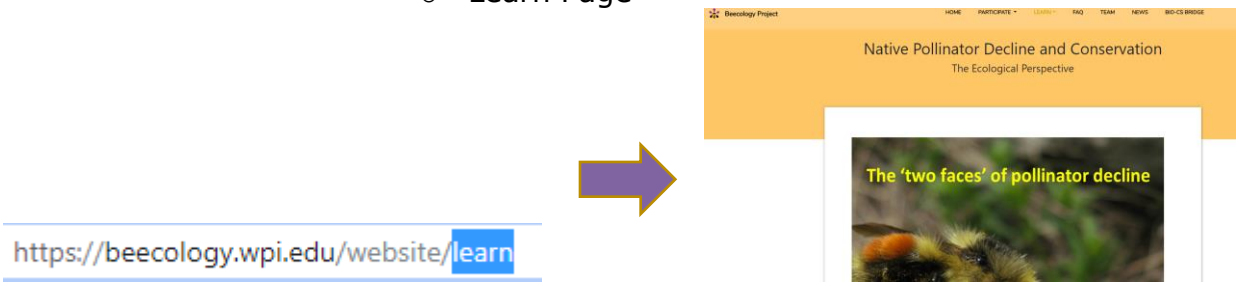
- This code snippet breaks into three parts
  - Import Block
    - Gives names to all the modules/components we will be using or linking to in the website
  - Routing Block
    - Sets rules for what webpage will be loaded depending on what is at the end of the URL
    - We can use this to move from one page to another by redirecting to a different URL when we click a button
  - Class Definition
    - Tells Angular what other Angular files we will need to make this file work
  - Examples

In the examples, the highlighted word is the key that can be found in the routing block.

- Home Page



- Learn Page



- app.module.ts
  - The brain of the whole Angular project. This loads every component and specifies how and when to load each of them.
  - Like the routing module it is broken into the same three sections:
    - Import Block
      - Loads every component/module that will be used
    - Body
      - Specifies where and when to load each component
    - Class Definition
      - In this case, its empty. However, it serves the same role.
- material.module.ts
  - Gives a library of materials that we use to stylize the whole project. This file loads every material we may need into a name, so we can use it later.
  - Once again, follows the same structure of import block, body, class definition
    - The body here does the same as the app.module.ts body and loads the proper materials at the right time.
    - The class definition is still empty

Appendix C.

Tutorial 3: Adding a Component to an Angular Project

Tutorial 3's video and recipe can be found at:
https://drive.google.com/drive/folders/1FmRp5LjkJbhWH607PygDbMh5KDV9IjbT
For convenience, the recipe for this tutorial is also included below.

Expected Background:

- Basic Command Line Knowledge
  - See links if you need a tutorial

Pre-Requisites:

- Having completed our Angular Tutorial 1
- Beecology site code

Links:

Flowers Code and Pictures
Windows Command Line Commands
Unix Command Line Commands

We will be making a new web page as an example of how we can easily expand the existing modular framework that exists in the Beecology website. To do this we will be updating our Angular dependencies, creating a new module, and then populating it with our content. For the sake of this tutorial, the webpage we make will contain information about three flowers, but the content can be anything you want.
In the links, the folder you can download includes the images we will use as well as the completed module. The images and text included therein was taken from Wiki-commons, if you would like to see the authors or photographers search the flower species name in Wikipedia and select the main page for that flower.

Installing Dependencies

Before we can begin, we must make sure all the packages needed to run the Beecology code are installed and up to date. To do that, follow these simple steps.

1. With the ModularWebsite code (i.e., the Angular code you downloaded from the Bio-CS Bridge website) open in VSCode, open a terminal (either with ctrl+shift+~ or by clicking Terminal -> new Terminal)
2. Run :

```
npm install
```

3. Wait for the dependencies to finish downloading and installing

Creating a new Component

Next, we will use the built in Angular commands to create and update the files so that we can start work on building our page.

1. In the same terminal, run:

```
ng generate component <componentName> --module app
```

This command breaks down into the following parts

Part	
ng	An angular command
generate component	Us asking angular to generate a component
<componentName>	The name of the component, I replace this with flowers-page
--module app	The module we want this added to. We need this because of the way the source code is set up.

2. Open up the four files it made and app.module.ts
    a. The four new files in the new folder should be mostly empty. We will ignore the .spec.ts file. The CSS should be completely empty. The HTML will contain a tiny amount of starter code so we can see it working later. The .component.ts has the basic code.
    b. The app.module.ts should have a new include, and a line added to declarations. This makes sure that angular will load our new page when we try to bring it up.

Expanding the routing table

Before we can bring up the page we need a way to get to it. In order to do that we will add our page to the routing table.

1. Open app-routing.module.ts
2. Add the following import to the import block
    a. Note: I use flower-page, if you called it something else, change that. Also, type all of this continuously on one line.

```
import { FlowersPageComponent } from './flowers-page/flowers-
    page.component';
```

3. Add a comma to the end of the last entry in the routing table.
4. Add the following to the routing table

```
{ path: 'flowers', component: FlowersPageComponent }
```

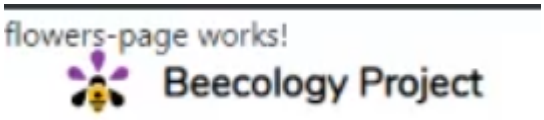
5. Your app.routing.module.ts should look something like this now (below)

```
import { FaqPageComponent } from './faq-page/faq-page.component';
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';

import { HomePageComponent } from './home-page/home-page.component';
import { TeamPageComponent } from './team-page/team-page.component';
import { ParticipatePageComponent } from './participate-page/participate-page.component';
import { LearnPageComponent } from './learn-page/learn-page.component';
import { NewsPageComponent } from './news-page/news-page.component';
import { BioCsBridgePageComponent } from './bio-cs-bridge-page/bio-cs-bridge-page.component';
import { FlowersPageComponent } from './flowers-page/flowers-page.component';

const routes: Routes = [
  { path: '', redirectTo: 'home', pathMatch: 'full' },
  { path: 'home', component: HomePageComponent },
  { path: 'news', component: NewsPageComponent },
  { path: 'team', component: TeamPageComponent },
  { path: 'participate', component: ParticipatePageComponent },
  { path: 'learn', component: LearnPageComponent },
  { path: 'faq', component: FaqPageComponent},
  { path: 'biocsbridge', component: BioCsBridgePageComponent },
  { path: 'flowers', component: FlowersPageComponent}
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

6. To verify that it worked, save the files and open a terminal.
7. Run

```
ng serve
```

8. Navigate to: localhost:4200/flowers
9. In the top left you should see flowers-page works!
    a. If you don't, check to make sure you saved the routing file
    b. Make sure not to mouse over the top banner, it will hide it
10. The top left should look like this



Setting up the CSS

To make sure that we keep an even look across all of the pages, we can copy over the CSS from another webpage on the Beecology website. In our case we will use the learn –page:

1. Open learn-page.component.css
2. Copy all of the contents to flowers-page.component.css
3. At the bottom of the flowers-page css file add

```
.title {
        font-size: 3em;
}
```

    a. This will become useful in a bit.

Setting up the HTML

With the aesthetics down, we can add content. To do this we will again borrow from the learn page. We will use its HTML to make sure we get the same layout, and the nice header.

1. First to add a title copy the first four lines from the learn-page.component.html
    a. This is the top banner we see when we go there.
2. Modify the header so it reads: "Flower Preference of Bombus Impatiens"
3. Remove all the text from the subheader, but leave the line there.
    a. This helps with the spacing later.

```
<div class="banner" id="page-title">
    <div class="row header justify-content-center">Flower Preference of Bombus Impatiens</div>
    <div class="row subheader justify-content-center"></div>
</div>
```

Importing Images

Before we can add the bulk of the content, we need the images to be imported.

1. Download and unzip the file containing the images and text included above.
2. On the sidebar of VSCode, right click on the assets folder and select "Open in explorer"
3. Copy the Flowers folder you unzipped into assets
4. The folder hierarchy should be
    a. Assets ->Flowers->3 flower images
5. You can now reference these images in the HTML

Expanding the HTML

Now with the images we need we can add the bulk of the content. Again we will be borrowing from the learn page for the general layout.

1. Open the flowers-page.component.html
2. Open the downloaded flowers-page.component.html
3. Copy over the rest of the file (you already have the header).

Originally this came from the learn page, layout gives us image slides with a header and a description below. To understand all the modifications and why we made them, please view the video.

4. Save your files
5. Open localhost:4200/flowers
6. You should now see three slides with three images and text.

Congratulations, you have now learned how to set up an environment, how to navigate through our modular angular structure, and how to add your own component. The content in this tutorial was arbitrary and simply to keep in theme with the rest of the bee site. Feel free to add any content and any images, play with the CSS and see what happens.