Capstone Project One: Overview

Project Steps

We have broken down the Capstone Project into easy-to-follow steps. Each step of the capstone contains a link with instructions for that step. Here's a quick overview of what you'll do for each step of your capstone project:

1. Step One: Initial Project Ideas
    a. You'll pick up to 3 project ideas to propose to your mentor and the Springboard community. You'll also explore some potential APIs.
2. Step Two: Project Proposal
    a. For this step, you'll write a proposal for the site you want to build. This will help your mentor better understand your chosen capstone project idea.
3. Step Three: Schema Design and API Selection
    a. After your mentor approves of your capstone project proposal, you'll figure out the database design of your application and which API you'll be using.
4. Step Four: Coding User Flows
    a. Once you've figured out what you're building, you'll write the code to implement it. It's important to think about what you want a user's experience to be like as they navigate your site.

5. Step Five: Polishing Your Application
    a. Once you have the core functionality implemented, you'll focus on additional UI enhancements and styling for your application.
6. Step Six: Documenting and Submission
    a. You've done a lot of work so now it's time to show your mentor your progress! Create a README in markdown, make sure your GitHub is organized, and submit your finalized project.

Overview

For your first Capstone Project, you'll build a database-driven website off an external API of your choice. Your finished capstone will be an integral part of your portfolio; it will demonstrate to potential employers everything you've learned from this course.

We want you to work on a challenging project that will incorporate all of the back-end skills you've been developing and some of your front-end skills from the last section. The goal of this project isn't to create something that's never been done before. You could potentially create a website similar to one that already exists, or use a popular API. That being said, we do encourage you to be creative when building your site. You're free to choose any API you'd like to use and we encourage you to tap into your imagination throughout the project.

There is a term in software development called CRUD - Create, Read, Update, Delete. This refers to all of the basic operations that a relational database performs. Your website should have more functionality than simple CRUD.

Examples

There are thousands of free, publically available APIs. If you love cars, you can pick from dozens of automotive APIs to build something that will reflect your passion. If you're more into history, look into an API that lists the nobility of Europe. If you love sports, build a site about India's top cricketers or your local football league.

Let's give you an example of what a site could look like. Say you choose an API like The Movie Database, your site could have a landing page saying "Welcome To MyMovieDB"

and a separate page that displays a sortable list of all the movies in the API. This would be CRUD.

You could implement various filtering methods - to filter based on an actor, a director, the year the movie was released, etc. When you click on the record associated with the movie, you could redirect a user to a separate page that displays all of the data associated with that movie.

Now let's talk about bells and whistles. If you were to implement ONE feature like creating sharable lists of your favorite movies, finding and playing a trailer for the movie on-page, or a simple "recommendation system" that would recommend new movies based on similarities to movies you liked, this would go beyond CRUD. A simple "recommendation system" would be along the lines of, if you like Big Daddy with Adam Sandler, recommending other Adam Sandler comedies from the 90s or recommending movies his co-stars like Steve Buscemi starred in. This does not mean creating a complicated system from scratch like Netflix.

It is better to pick a project that errs on the side of simple and boring than a complex project with a million moving parts you can get stuck in.

Here is an example of a previous project.

Guidelines

1. You will use the following technologies in this project: Python/Flask, PostgreSQL, SQLAlchemy, Heroku, Jinja, RESTful APIs, JavaScript, HTML, CSS. Depending on your idea, you might end up using WTForms and other technologies discussed in the course.
2. Every step of the project has submissions. This will alert your mentor to evaluate your work. Pay attention to the instructions so you submit the right thing. You will submit the link to your GitHub repo several times, this is for your mentor's convenience. Your URL on GitHub is static and will not change.
3. The first two steps require mentor approval to proceed, but after that, you are free to continue working on the project after you submit your work. For instance, you don't need your mentor to approve your database schema before you start working on your site. Likewise, you don't need your mentor to approve the first iteration of your site before you start polishing it.
4. If you get stuck, there is a wealth of resources at your disposal. The course contains all of the material you will need to complete this project, but a

well-phrased Google search might yield you an immediate solution to your problem. Don't forget that your Slack community, TAs, and your mentor there to help you out.

5. Make sure you use a **free API** and **deploy your project on Heroku**, so everyone can see your work!