Museumary Project

Motivation

For our web application we are creating a website which allows users to view works of art, artists, museums and types of art from our currently limited catalog. We chose to catalog art works as we all have a deep appreciation for the finer things in life. We felt that creating a website that is centered around something we were passionate about would result in an overall higher quality application. Beautiful works inspires beautiful code. Further, we created this application in order to familiarize ourselves with a number of software engineering practices, tools, and procedures. A few of us in the group were relatively unfamiliar with tools such as git and practices such as continuous integration. By implementing these tools and practices throughout the development process, we hope to strengthen our individual skills as software developers while learning to cooperate in a larger scale group project than we are accustomed to.

Use Cases

Currently, our website is limited such that a user can only browse through our collection of models. A user can select any model type such as 'Works' or 'Artist' and will brought to a page with all the instances of that model. Further, a user can also select any instance of a model and will be brought to a page displaying more information on that instance. For example, from the 'Works' page a user can select 'Street in Auvers-sur-Oise'. The user will be brought to a separate page with an image of the work along with information such as the work's artist, creation date, type of art, and location. From the instance page, the user can also select any of the data that corresponds to a model in our application (artist, type of art, location). If the user clicks on the artist, then the user will be linked to the instance page of the for that particular work.

RESTful API

The main principle of a REST-ful API is to conform to a standardized architecture that makes use of common HTTP requests. HTTP functions include GET, POST, DELETE, and PUT requests. In following a uniform HTTP REST-ful standard interface, it is easier for other developers and web applications to communicate with the API. These common functions have defined behaviors that make interacting with REST-ful APIs less ambiguous than interacting with APIs that don't follow the same architectural style. Resources provided and managed by these methods are identified by a URI, or a Uniform Resource Identifier. Multiple URIs may refer to the same resource in a REST-ful service depending on the implementation.

The POST method is commonly utilized to *create* new resources. The PUT method is commonly used to update resources, but may also be used to create resources depending on the provided parameter. The DELETE method is used to delete a resource identified by a URI.

The GET method returns a resource response usually formatted as a JSON or XML text. Our web app currently only defines the behavior of GET requests to the API since we don't have any implementation for users to modify or delete resources in the database. In the future the front-end of our application will make GET requests to the API which returns the data of the requested resource formatted as a JSON file. We documented the behavior of GET requests for the entire list of instances of our four models as well as requests for single instances of our models.

Models

Our Web Application will be centered around 4 core models: the Artwork, the Artist, the Venue, and the Art Type. Each model has multiple attributes that relate to one another and yet still represent a unique characteristic that describes an aspect of the craft.

The Artwork - the piece of art that a person has created. Its importance to the craft doesn't need to be explained as how can the entire concept of art exist if nothing is created?
Attributes:

- **artwork_id**: The unique identifier of the artwork
- **name:** The name of the artwork
- **artist:** The creator of the artwork.
- **type:** The medium of the artwork
- **data:** The time it took to create the artwork
- **culture:** The source of culture of the artwork
- **location:** The current location (Museum) of the artwork

Non-Model and Identifier Attributes:

- **data**
    - the time period and year [beginning, end] that it took to complete the artwork. As each new artwork is a relatively unique piece, there is no set amount of time that it takes to complete a single work. Sometimes production is fast, and sometimes it takes years and years to finish. This information can show an artist's dedication to his craft and how much effort needs to be put in each new creation.
- **culture**
    - this attribute represents the culture of the artwork. Every society has their own customs and beliefs and each piece of artwork created is representative of those things. It's can give a person hints of what and why that piece of art was created.

The Artist - the creator of the artwork. Everything that's created has to have been created by someone and the artist is that person. The creator of the artwork is of utmost importance as the artwork wouldn't have been able to exist if the artist did not create it. It is also important for the general recognizability of the piece. Famous artworks are usually created by famous people (even if they are not famous at the time).
Attributes:

- **artist_id:** The unique identifier of the artist
- **name:** The name of the artist

- **birth:** The year the artist was born
- **death:** The year the artist died (if applicable)
- **birthplace:** The city and country where the artist was born
- **deathplace:** The city and country where the artist died
- **artwork_ids:** A list of artwork_ids that the artist created
- **culture:** The culture that the artist came from

Non-Model and Identifying Attributes:

- **birth and death**
    - The years that the artist was born and died are one of the biggest influences in an artist's life. Whether it was a turbulent or peaceful time period can influence the artworks the artist will create. It also shows which art periods the artist has lived through in his or her life.
- **birthplace**
    - While the birth year tells when the artist was born, the birthplace is where he or she came from. Every society has different customs and beliefs and where a person was born is one of the most important factors in their upbringing. As art is just a form of expression, the birthplace of an artist also influences their what they create.
- **deathplace**
    - Opposite to the birth place, the death place signifies where the artist died. While it cannot represent an artist's entire life, it can show the location where he or she was before death. This is important to the life of an artist as the experiences of each artist is different and that experience is conveyed to whatever they are creating.

The Venue - the building that is hosting the artwork. Any art needs a proper audience so that it could be enjoyed. Excluding private collections, hosts are typically museums or galleries.
Attributes:

- **venue_id:** The unique identifier of the venue
- **name:** The name of the venue
- **country:** The country that the venue is located in
- **city:** The city of in the country the venue is located
- **work_ids:** A list of artwork_ids that are currently being hosted by the venue
- **address:** The address of the artwork including street name, city, country, and zip

Non-Model and Identifying Attributes:

- **country and city**
    - The world is divided by countries and each one has its own culture and differences. The the country that a venue is in can largely determine the rules it has and its subculture
- **address**
    - The address of the venue tells people where it's actually located on a map. That way if people want to go there or research it on their own, they know which venue to go to and look up.

The Art Type - the category of the artwork. This is the classification of art based on chosen attributes that tries to encompass the groups that most artworks fall into. The type of art is its art

form. Because there are so many aspects of art and it exists in all forms, how the artwork was created is really relevant. While a painting and a sculpture are both categorized as art, they cannot be any more different.

Attributes:

- **type_id:** The unique identifier of the art form
- **type name:** The name of the category that the art form describes
- **mediums:** The materials used to typically create the art form
- **notable_examples:** The artwork_ids of notable examples of the art form
- **notable_artists:** The artist_ids of notable examples of the art form

Non-Model and Identifying Attributes:

- **mediums**
    - Because the concept of art is so broad, artwork can be created in many different ways. The artwork can be a painting, a drawing, or a sculpture and still be thought of as an expression of art. The medium attribute is our way of classifying what materials were involved in the creation of that piece. It's not all encompassing and there's a lot more items involved, it's a good place to get a basic structure down,

While each model is different from the others, they each share two key characteristics:

- **identifier**
    - The unique key for the instance of that model in our database. Even though the data in each model typically has a different name from others in its group, there is a possibility of overlap between names as many things have been rescued throughout history. Adding a unique id to each item will let us bypass potential overlaps entirely.
- **name**
    - The name of the item inside the model is generally most important part. While a unique identifier is good for record keeping and distinguishing between other items it is only related to our database and not others. The name is how people will recognize the difference between each instance.

Tools

The tools we are using for our project are Apiary, Bootstrap, React, Flask, GitHub, Trello, Jinja, PlanItPoker, and Slack. Our Apiary currently contains the preliminary design and documentation for our future REST-ful API. As described above, we have so far only described the behavior of GET requests for data on our models and for single instances of each model. Bootstrap was used as a design template that enabled us to get a basic working web design. Jinja is a templating language we used primarily in our HTML files to simplify the design of our webpages. With Jinja, we created a base template for each of our HTML files which contains the code block for our navigation bar and our header CSS files. Using a base template allows each new HTML file to simply extend the base template to automatically load in the common elements across all the pages. This makes it simpler and more efficient for making changes to features that exist on every page since it involves only changing the base template rather than requiring the modification of every single HTML file. Jinja also provides the use of expressions in HTML files such as flask's url_for( ) and for loop constructs which we will use in the future

for populating our model pages. This will cut down on the amount of code that is simply copied and modified over and over.

We are using Flask as a web microframework for back-end routing of our html templates to our web server host. Flask maps our web URLs to our HTML files using the render_template( ) function. In order to use our Flask framework, we had to change our file structure to conform to Flask's standards. Flask automatically looks for HTML files in the "templates" folder when rendering them to the web server. Similarly, all CSS, JavaScript, and other static images must be located in the "static" folder.

When working in a team it is important to consider version control to ensure there are no conflicts in the application, good issue tracking to ensure that the team knows what features still need to be implemented or have already been implemented, and a good platform for communication so that the team stays connected and understands the current state of the project.

We are using GitHub for version control and Trello is being used for issue tracking. Trello allows issues to be grouped in separate lists so that it is easier to see what needs to get done for each phase of the current project. For example, the issues that need to be fixed by the team members working on the backend files can be kept separate from the issues the frontend team members are working on. This makes organization substantially easier when there are a large amount of issues to be resolved. We are using Slack for group communication and as a means to quickly share links. Slack provides an excellent way to pin important links the team needs access to and the use of multiple channels helps keep discussion relevant to the team members that are currently working on the same part of the project. We used PlanItPoker as a planning tool to estimate how long each user story would take. In the future we can use these estimations to compare to new tasks to improve our time estimations to completion.

Hosting

We are hosting our website on Google Cloud Platform using their App Engine service. Google App Engine offers both standard and flexible environments for running applications. The standard environment is more lightweight than the flexible environment, but it has more constraints on the programming language version and frameworks that can be used. The standard environment makes use of a "sandbox" environment that ensures that apps cannot perform actions that would be able to interfere with the performance and scalability of other apps. This means that an app using the standard environment is not able to write data to the local file system or make other arbitrary network connections, and instead must use services provided by the App Engine to store data and communicate over the network.

We chose to use the flexible app environment because it allows for the use of Python 3, whereas the standard environment only allows the use of Python 2.7. We also chose to use the flexible environment because the standard environment raises exceptions when attempting to import a Python module which does not work within the sandbox restrictions. The flexible environment allows for the freedom to use any framework or library of our choosing.

In order for the App Engine to host our site, we created an app.yaml file which specifies runtime settings. Google also needs to be provided with a requirements.txt file that lists the dependencies

the engine needs to install each time the application is run(e.g. flask==0.12.2 gunicorn==19.7.1). The runtime starts the application using the specified *entrypoint* in the file. Our application is using Gunicorn which is a Web Service Gateway Interface (WSGI). When the app is deployed, it is provided with a default [PROJECT_ID].appspot.com address which can then be mapped to a new custom domain mapping which we have set to museumary.me.