Project Report

Basic Information

Project Name: FitHub

Team Canvas group: A13

Team Members, Email, and Github username:

Kaylee N Trevino, kntrevi1@utexas.edu, kntrevi1

Christopher G Saenz, schrodingersfridge@utexas.edu, ChriSaenz

Michelle Wen, michellewen@utexas.edu, michellewen3

Andrew Wu, andy22wu@utexas.edu, wuhoo16

Github repo link: https://github.com/UT-SWLab/TeamA13

Deployed app link: https://fithub-291802.uc.r.appspot.com/

Motivation and Users

The preliminary idea for our IMDB project stemmed from the fact that all four team members were interested in fitness and we wanted to find a way to aggregate many different types of online resources to help people of all ages and activity levels. After doing some research online, we were surprised to find that due to how prevalent desk jobs are in modern society, over 60-85% of people around the world live sedentary lifestyles that is attributed to around 2 million deaths every year. In fact, the World Health Organization states that "sedentary lifestyles increase all causes of mortality, double the risk of cardiovascular diseases, diabetes, and obesity, and increase the risks of colon cancer, high blood pressure, osteoporosis, lipid disorders, depression and anxiety."

Our motivation for creating Fithub was because we believe that everyone around the world should have both the resources and opportunity to improve their fitness and strive for a more active lifestyle. Considering the dangers of a sedentary lifestyle, we want to provide detailed information about many different types of exercises, equipment, and popular Youtube channels that will allow our users to incorporate regular exercise into their routine. With the ongoing COVID-19 pandemic, we are aware that everyone around the world may now have limited access to public gyms and face major challenges to finding and staying consistent to a workout routine. As a result, we believe our comprehensive website database will help athletes and fitness enthusiasts of all ages and experience levels who want to push themselves to the next level.

Requirements

In Phase 1, we created user stories that would satisfy the wants of the user. Our first user story demonstrates a user’s want to easily navigate through the pages and to the different models. This want is satisfied through our implementation of a NavBar. Secondly, we used a user story to satisfy a user’s want to view the ratings of equipment. We displayed information on equipment instance pages so that users can know which sellers are trusted by other users and thus can feel more trusting towards them as well. Thirdly, anticipating that users would want to directly access YouTube videos and channels through links on our pages, we embedded such links in our channel instance pages. Fourthly, we had a user story to satisfy any user’s want to learn more about the team who developed the website or want to learn more about the tools used to develop the website. On the about page, we clearly stated information about the team and the tools used for the website’s development. Finally, our last user story was created to allow users to view information about different models, such as the prices of workout equipment. This was done so that all models have appropriate information that allows you to gain information about each instance and thus know more about what you are interested in. For example, by learning about the prices of workout equipment, one can learn about which equipment they are interested in based on what they can afford.

In Phase 1, our first user story’s estimated and completion time was 1 hour. Our second user story was expected to take about 1 hour; however gathering the information took about 2 hours. Our third user story was expected to take about 5 hours, but figuring this out took about 6 hours. For our fourth user story, the about page was expected to take about 2 hours and took about 3 hours instead. Lastly, our fifth user story was expected to take about 1 hour but took about 3 hours.

In Phase 2, we added six user stories to satisfy desires we users will have. Firstly, one user story was created to link exercises to muscles which those exercises workout. By providing information about the muscles which exercises work out, the user can then find exercises that will workout the area they intend to for that workout session. Secondly, we wanted to improve the design and styling of our website so that users felt more inclined to use it and use it often. Thirdly, we created a user story to satisfy user's desires to navigate through model pages and not be overwhelmed by having all instances presented on the single model page. This was satisfied through pagination on all three model pages. Fourthly, we anticipated users would want to see how the three models are related and more specifically, which instances of one model are related to instances of other models. This was completed through tagging and by presenting tables on each instance page that shows how the current instance is related to other instances from other models. For our fifth user story we wanted to satisfy users' desires to view multimedia, more specifically images, on each of the model and instance pages that were clean and non-broken images. Users would not want to be presented with images that do not show any image or do not show it clearly. Lastly, it seems that users would want to have more information than previously presented for each of the YouTube channel instance pages; this means including information about channels that are related to the YouTube channel being presented and the subscriber, views, and video count. Users would want this information to feel more comfortable about which channels they are drawn to.

In Phase 2, our first user story was expected to take about 10 hours, but was a bit harder than expected and thus took 15 hours. For our second user story, the improvement of styling, which was applied to all model pages and instance pages, was expected to take about 4 hours, but instead took about 3 hours. Our third user story was expected to take about 3 hours to complete and took about 3.5 hours to complete. Fourthly, as expected, our fourth user story took about 10 hours to finish. For our fifth user story, cleaning the data was expected to take about 4 hours for each model, and ended up taking about 6 hours for each model. Finally, for our last user story, although expected to take about 3 hours, this took about 4 hours.

For Phase 3, we also had six user stories. Our first user story was designed to allow users to check if certain exercises will require specialized equipment; this was achieved through the linking instances together and displaying user-friendly information such as description, price, vendor location, and more. Secondly, our next user story was created so that users could securely and easily navigate throughout the website. This was achieved by writing sufficient unit tests for CRUD operations on the backend and writing Selenium tests for the frontend interface that the user interacts with. Our third user story focused on the searching functionality of our website where the user is able to search the website with real-time suggestions and updated cards. That way, the user can easily see the search results in one-line descriptions or detailed cards. Our fourth user story is focused on allowing the user to perform multiple operations on the cards, such stacking the filtering, searching, and/or sorting functionalities. This gives the user greater flexibility when looking for desired instances. For our fifth user story, we expected users to want to sort in ascending or descending order on different attributes for each of the model pages. For example, the user should be able to sort by name, category, and equipment on the exercises model page and sort by name, subscriber count, view count, and video count on the channels model page. Our sixth user story focused on the filtering functionality added to our website. We want the user to be able to find specific instances that they’re interested in out of a large database of exercises, equipment, and channels. We added filtering to support all 3 model pages and allowed the user to filter on multiple attributes.

In Phase 3, our first user story was expected to take about 15 hours, but took about 12 hours. Our second user story was expected to take 8 hours and ended up taking around 10 hours. Our third user story was expected to take 14 hours to complete, and, instead, took 15 hours. Our fourth user story was expected to take around 7 hours, but ended up taking around 9 hours. Our fifth user story was estimated to be completed around 4 hours, and we ended up spending less time at around 3 hours. Our sixth user story was estimated to be completed in 6 hours, but ended up taking 7 hours instead.

For Phase 2 of our project, you can see the use case diagram for our website in Figure 1. The actor in our use case diagram is the user or fitness enthusiast. Their goal is to find out more information about exercises, exercise equipment, and Youtube channels, along with specific details and descriptions about each model. The user is able to view several pages such as the home page, exercises model page, equipment model page, Youtube channels model page, and about page. Users are able to access each of the model pages from the home page and, within the 3 model pages, the user can click on specific cards that lead them to an instance page that goes more in depth about the information they are seeking. You can also see that because each of the model instance pages provides links to other instances of other models, the diagram reflects this as model instance pages extend other model instance pages.

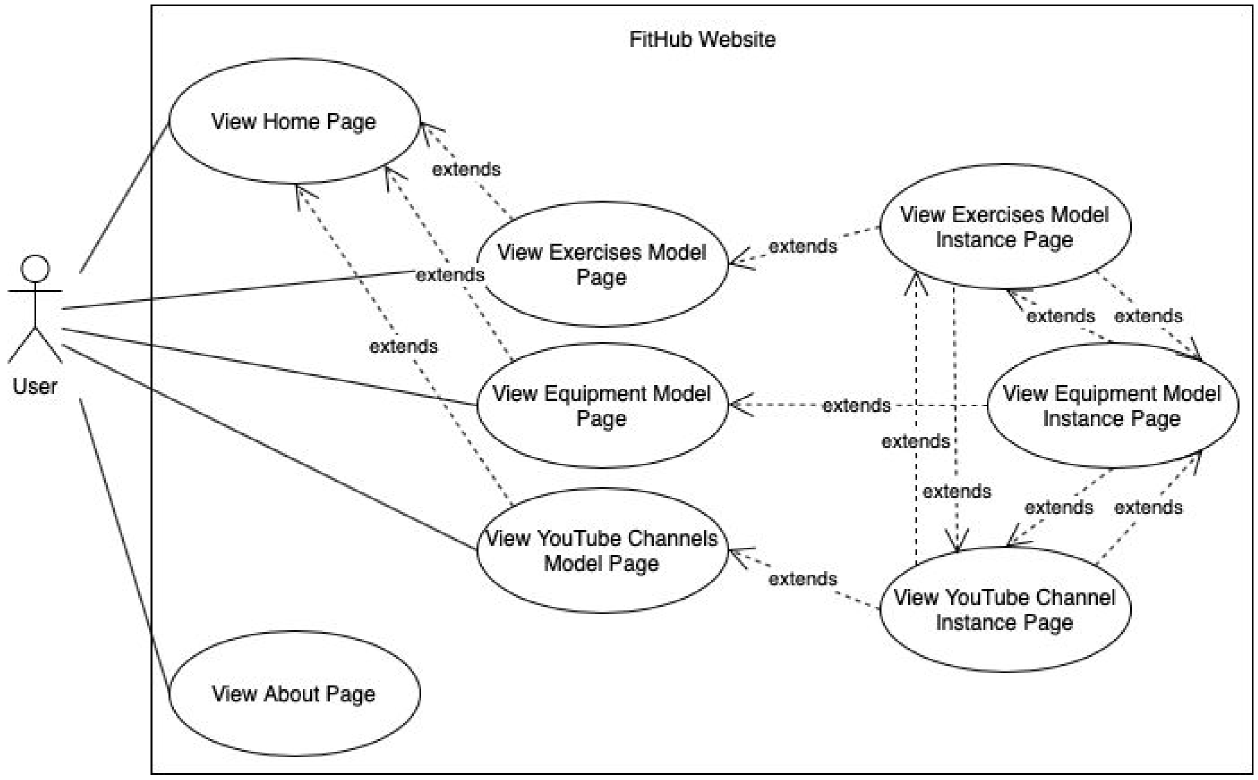
FitHub Website
User
View Home Page
View Exercises Model Page
View Equipment Model Page
View YouTube Channels Model Page
View About Page
View Exercises Model Instance Page
View Equipment Model Instance Page
View YouTube Channel Instance Page
extends

Figure 1: Use case diagram for FitHub Project in Phase 2.

In Phase 3, the filtering options of searching, sorting, and filtering were added to the functionality of the site for each of the model pages. For this reason, the use case displayed in Figure 2 extends Figure 1 by additionally having the use cases of searching, sorting, and filtering. Furthermore, the filtering operation use case is a generalization of these operations and, as indicated through the extends keyword, can be accessed from each of the three model pages.

Note that in Phase 4, because no additional functionality was added and only refactoring of code was completed, the use case diagram is the same in both Phase 3 and 4.

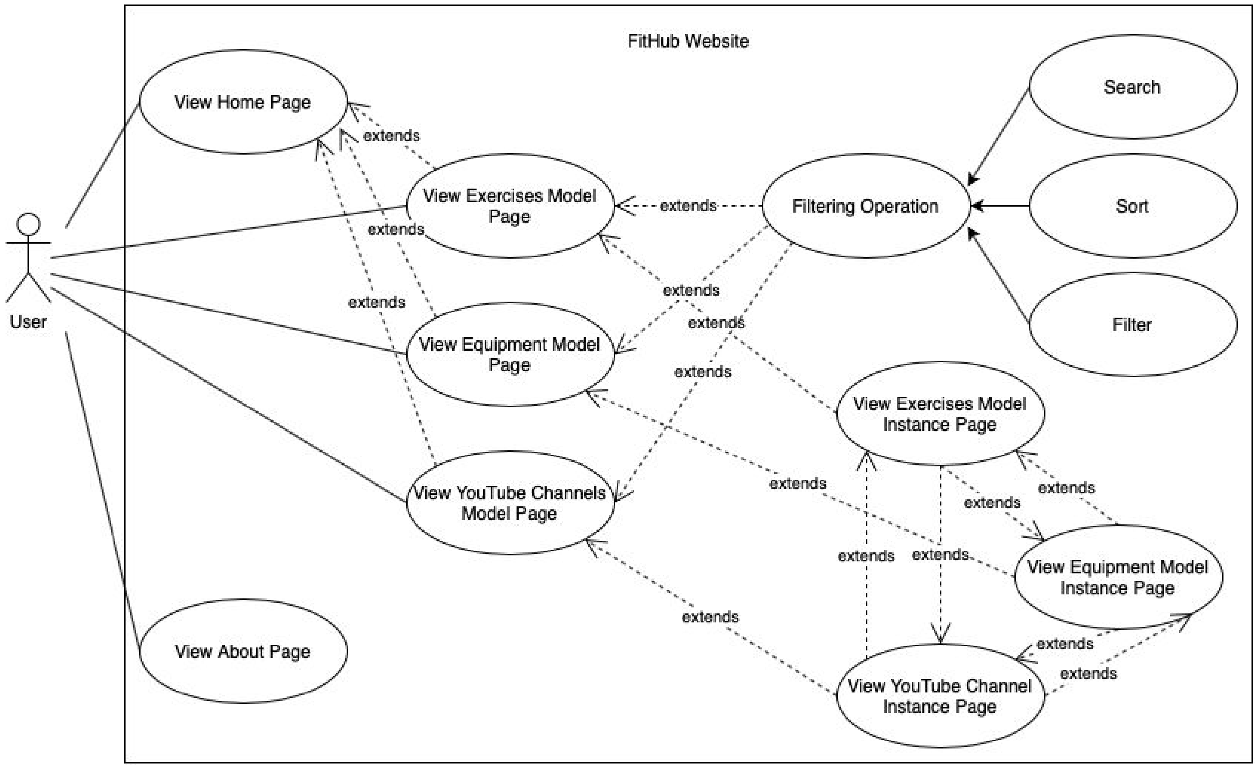
FitHub Website
User
View Home Page
View Exercises Model Page
View Equipment Model Page
View YouTube Channels Model Page
View About Page
Filtering Operation
Search
Sort
Filter
View Exercises Model Instance Page
View Equipment Model Instance Page
View YouTube Channel Instance Page
extends

Figure 2: Use case diagram for FitHub Project in Phase 3 and 4.

Design

The data we pulled from our APIs was organized into objects of each model type (Equipment, Exercise, Channel) via Python helper functions, which both handled the API calls as well as the processing of the JSON data received from the API calls. Each of these objects include a convert to dictionary method that helps to easily convert the object into a format that can be easily stored in our mongoDB database as key-value pairs for each attribute. The objects are put into their respective collection type, which matches up exactly with the model type. These 3 collections, named ‘exercises’, ‘equipments’, and ‘channels’, are then retrieved into global arrays when the Flask server is deployed for the first time. Some of the objects we pulled from the APIs were missing images or information, and their respective instances would not display correctly in their instance pages or their model page cards. To solve this issue, we made blacklists to aid in identifying and ignoring these instances.

Our website begins with a homepage, displaying a welcome message and a carousel. Each item in the carousel consists of an image, the name of a model, and a button link to the corresponding model page. At the top of the homepage is a navbar with quick links to all three model pages and the website’s About page, as well as a logo icon that links back to the homepage. This navbar is defined in the base HTML template and consistently appears on every page of our website.

Our model pages all have the same overall design - a 3x3 grid of cards listing the initial information of each model instance along with a picture describing the instance. Originally, each card had a button at the bottom that the user could click to reach the instance’s respective page, but after some discussion we decided to make the entire card clickable instead, on the grounds of user accessibility and polished design. At the bottom of each page is a pagination bar that allows the user to jump to further pages of the model they're looking at. The backend logic that handles the pagination is dynamic and changes based on the length of the object array routed to the model page to support the different number of instances of each of the model types.

Our instances pages also use similar designs to each other. At the top of each instance page is a title and a carousel showing various images related to the instance. Below this carousel is all the detailed attributes of the instance. After the instance's information are three Bootstrap tables showcasing related instances of each model. We use the category property of each model object to find a list of related objects, which is then passed as a parameter to the instance page. Then, this list is split by model type and displayed as grids with links to the listed instance pages.

Figure 2 shows a UML class diagram detailing the classes used to represent model instances. Our 3 model classes are Exercise, Equipment, and Channel. Each of the classes have specific fields and attributes used to identify the instances.

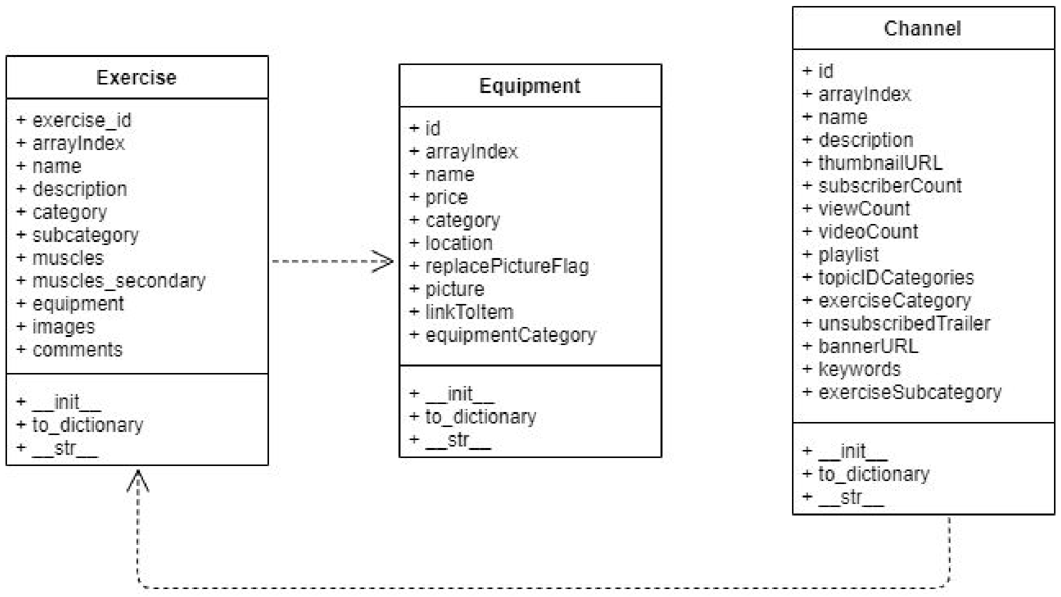

Figure 2: Class diagram for FitHub Project.

Testing

Since all of the backend infrastructure for this web app project was done in Python, we used the unittest framework to test all of the Python methods in our main.py that communicate with our remote mongo database. This required setup prior to the TestMain to connect to a new testing remote database with collections that would be setup parallel to our actual 'phase2Database' and avoid corrupting our official database contents. Our unit testing file includes 7 test methods that cover all of the CRUD operations that are used in our backend including creating documents for each of the 3 model collections, loading documents from each of the collections, and cleaning the database by deleting the collections. Note that since our backend and web page implementation never changes the contents of our database after initialization, we do not need a test for updating documents in the database. In addition, our calls to the initialization methods also verifies that our API helper methods are working as expected and that our API keys are not expired. The verification of the database and array lengths, unique ids, and arrayIndices verify that we have enough instances for each type of model and are storing unique instances in the order that we expect them to be in.

For the frontend and GUI, we used Selenium to test multiple components of our website. We tested each of our three model pages separately, our instance pages, and our home and about page. Within each of these we also tested components such as the carousel, the navigation bar, links to other pages in our Bootstrap tables, attributes on the model cards and instance pages, empty attributes, pagination, bootstrap components such as cards, and more. We attempted to test many components, attributes, links, etc. in a way that would give us a better coverage of the website when testing and would ensure that when code is changed, we would have consistent notification if a bug is introduced.

We also performed tests on the search bar, sorting, and filtering methods developed in phase three of this project. For the search bar, we tested both the dropdown that is updated in real time as the user is typing, as well as the results of the search bar after typing has been completed and the user would like to see the updated cards for a specific search on a specific instance. For the search bar, we also tested invalid searches and ensured the results were as expected. Additionally, when testing the filtering we tested by various attributes and considered a subset of the checkboxes. We then verified that upon submitting the form, the cards are updated appropriately. Finally, when testing sorting, we tested every combination of sorting attributes both in order of descending and ascending attributes. With all of these tests, we verified that these three functions worked well both independently and together. Finally, to sum up our tests for phase three, we also tested our ability to reset the three functions applied so that the original cards could be considered again. We were able to use our tests from phase three for phase four after refactoring.

Models

Our three models are workout exercises, workout equipment, and Youtube fitness channels. With the exercise model having 100 instances and the equipment and fitness channel models having 40+ instances, it was important for us to gather a large amount of useful and rich data. Gathering large amounts of useful data also helped us link the model pages by common attributes and suggest related exercises, equipment, and channels for each of the unique model instance pages. All of the data used for the three models came from a variety of APIs which allowed us to display rich multimedia on our model and instance pages.

Our exercises model mainly used data pulled from the wger REST API which included several hundred exercises with information like exercise category, primary muscles, secondary muscles, equipment, images, and comments. We used a subset of that data by filtering for exercises in English with complete information on description, category, and equipment. The Google Custom Search JSON API was also used to grab related images for exercises that did not have an existing image. Using these 2 APIs, we were able to display a variety of information about the exercises along with several photos to visually describe how the exercise works.

Our equipment model pulled fitness equipment data from the eBay Browse API. The data returned includes the equipment price, category, seller location, image, and a link to the item on eBay which we displayed on the model and instance pages.

Our Youtube channel model pulled data using the Youtube Data V3 API and, more specifically, used the channels, search, playlists, and video API operations. This data included the channel description, number of subscribers, number of total views, number of videos, and video links. Using this data, our channel model pages has a variety of information including images and embedded videos for the user to click on.

Our Exercises model page relied on the wger REST API (https://wger.de/en/software/api) and the Google Custom Search API (https://developers.google.com/custom-search/v1/overview). Our Equipment model page relied on the eBay Browse API (https://developer.ebay.com/api-docs/buy/browse/overview.html). Finally, our Youtube channel model page relied on the Youtube Data V3 API (https://developers.google.com/youtube/v3/getting-started). We mainly relied on documentation from the APIs directly, since they were sufficient in giving us the information we needed to extract useful information for our website.

Tools, Softwares, and Frameworks

In phase 1, we used GitHub for code collaboration and merging. We pushed code to our own branch and then merged our working code to the ‘dev’ branch. We also used issue tracking to fix problems in our project and project boards to keep track of user stories. We used Slack as our main form of communication along with channels within Slack that were dedicated to a certain topic. For our project, we used the Flask web framework to easily build our website, Bootstrap4 as a CSS framework to easily style and design our website, and Google App Engine to quickly deploy our website. We also used MongoDB to store and easily access the large amounts of data pulled from the APIs.

In phase 2, we used all tools, software, and frameworks from phase 1 with the addition of tools used for testing. To test our project, we used Selenium and Python Unit Testing. Selenium was used to test multiple components of our website frontend GUI including the three model pages, instance pages, home page, and about page. Within these components, we tested the content, validity of the pages, carousel, navigation bar, attributes, linking, and pagination. We used Python unittest, a unit testing framework, to verify that the Python backend methods that communicate with our MongoDB database are working as intended.

In phase 3 and 4, we continued to use all tools mentioned previously from phase 1 and 2. The tools and software we used in previous phases were sufficient in providing us with what we needed to continue making improvements to our website. Throughout all phases, we also used Draw.io to assist in creating UML diagrams needed for the technical report and Phase 4’s design report.

Sources

World Health Organization News (https://www.who.int/news/item/04-04-2020-physical-inactivity-a-leading-cause-of-disease-and-disability-warns-who) was used during initial research on the specific detrimental health impacts that sedentary lifestyles have when we were deciding on our website's overall purpose.

Bootstrap4 W3Schools Tutorial (https://www.w3schools.com/bootstrap4/default.asp) was referenced for using Bootstrap4 as a CSS framework. More specifically, we focused on containers, grids, pagination, tables, images, cards, carousel, and navbar from the tutorial.

MongoDB PyMongo Tutorial (https://api.mongodb.com/python/current/tutorial.html) was used as a refresher on the syntax to making a connection to our remote database and write queries to access our different documents and collections.

Selenium WebDriver Documentation (https://selenium-python.readthedocs.io/api.html#module-selenium.webdriver.remote.webelement) was used for testing the frontend GUI using Selenium.

LocalStorage Javascript Blog **(**https://www.sitepoint.com/quick-tip-persist-checkbox-checked-state-after-page-reload/) was used to learn more about how to add key-value pairs, access, and clear data to the browser's local storage. We used local storage to help us persist checkbox state, sort selections, and search queries to match up with our modified array state.

Stack Overflow: How to get the value of the childnodes using getElementsById (https://stackoverflow.com/questions/22488705/how-to-get-the-value-of-the-childnodes-using-getelementsbyid) was used to add dynamic Javascript code to find the actively displayed select

option for a select HTML drop-down menu. This helped us add alerts if the user tries to click either sort button without selecting a sorting attribute first.

Explore Flask: Advanced patterns for views and routing (https://exploreflask.com/en/latest/views.html) was used to debug our broken routes when refactoring to support multiple clients. This helped us debug and implement the passing of the currentArray between the frontend HTML and the backend Flask view functions.

W3Schools: HTML <meta> Tag (https://www.w3schools.com/tags/tag_meta.asp) was used for learning to work with meta tags in html and learning to access data passed from flask to html file and access that data through javascript.

W3Schools: JavaScript HTML DOM Node Lists (https://www.w3schools.com/js/js_htmldom_nodelist.asp) was used for learning to work with listNodes of html elements in javascript & jquery.

Google Cloud: Uploading Objects (https://cloud.google.com/storage/docs/uploading-objects) was used for learning how to upload large video files to a bucket in order to display on our website and deploy our website using Google App Engine

W3Schools: CSS Reference (https://www.w3schools.com/cssref/) was used as a reference for CSS styling used on the website.

Youtube: Build A Responsive Bootstrap Website A Full Screen Image Slider using Bootstrap 4, HTML5 & CSS3 (https://www.youtube.com/watch?v=Thw33qJ5DXo&t=3937s&ab_channel=DrewRyan) was used to build the frontend homepage design of our website.

Reflection

Phase 1

In this first phase, we learned how to reference and pull from multiple APIs and easily retrieve data from different sources. It was really interesting to see how Youtube and Google captured complex resources such as a Youtube channel and represented the data in a JSON format. After reading documentation, our whole team gained invaluable experience with how to access data in a RESTFUL manner for developer use cases. Throughout the development of the different parts of our project, we also got more and more comfortable with using Github branches, storyboard, and issue tracker for project management. In addition, we were able to apply the basics of HTML, CSS, and Bootstrap learned in class to a real-world use case and design the different pages on a website with the end user in mind. The majority of our team has never worked with frontend development before, so directly using Bootstrap elements to simplify the web app development and layout was a very helpful skill that we picked up during this phase. Last but not least, we also learned a lot about the details of MongoDB and how to use PyMongo to access and store large datasets to a remote database.

Although we learned a lot, we also had many struggles during this phase. For example, a few of our members were unfamiliar with the Github workflow, causing code integration to be extremely inefficient. In addition, not enough local testing was done before pushing to the public repo. We also had trouble with formatting the model grids nicely, especially with centering alignment. Also, some of the image URLs pulled from the Google Search API were broken, resulting in images not showing for both exercises and equipment models. Additionally, when moving to the next picture on the carousel, the picture sometimes shows up at the bottom of the screen then moves up to its position; this means the carousel still works, but we want to improve on this in the next phase of this project. Finally, calling data from APIs took way too long, around 45 seconds to pull over 200 exercises; therefore, we needed to use MongoDB database to store data so the APIs are not called during deployment. However, the data and the amount of data we stored in the database still needs to be improved in the future.

Alternatively, we also had many things that went well this phase. Fortunately, the splash page development as a whole went quite smoothly as the navbar was easy to work with. The creation of the channel instance pages also went well, and the data could be easily found in the Youtube API. We were also glad that our images for each team member in the About page worked great after cropping. They very nicely fit on our cards for each member, and we were very happy with how that page was finally completed visually. We were also fortunate that our Google Custom Search API worked well for easily searching images needed for the exercises and equipment page. This allowed us to easily get a large amount of multimedia required for the instances of the three different models. Finally, we also were glad with our idea to use a ‘dev’ branch and individual branches for each person. These individual branches are updated with the code each person is currently working on, the dev branch has the most updated finalized code, and the main branch holds the code that is due for each phase.

Phase 2

In this second phase, we learned how to further work with the API’s we previously mentioned to aggregate even more information. We learned more about how to pull data from the new ebay API, store the data in a remote mongoDB database, and then read the data from the database in order to dynamically populate each of our three instance pages. Our team used MongoDB and thus learned how to work with a NoSQL database, where the structure of the data was extremely flexible and could be designed to match the attributes of our backend objects using key-value pairs. We also gained experience with using a Python driver pymongo to easily capitalize on the powerful operations supported by the mongo database such as easy queries with filtered keywords to help create our cross-model links. We also learned how to use pagination as well as how to make each of our model pages more attractive through the use of bootstrap and CSS styling. Finally, we learned how to use testing to ensure that our website continues to work the way we have intended, especially as we make small tweaks and our codebase grows larger and larger. We not only learned how to test the backend python methods with unittest, but also how to use Selenium to simulate the role of the user and test that navigation and links throughout the webpage are working as expected. For this particular phase, we did not use Mocha since we currently have no interactive Javascript components in our webpage. All in all, adding in testing showed up the importance of automating the process of testing basic units of our code so that it is easy to find bugs right as they occur.

For the future of this project, we are hoping to add more test cases to better ensure that our website works as intended. We believe that our main.py is getting extremely large and dividing it into modular style would really help our overall project organization and make it easier to make smaller, targeted test files. In addition, from this phase we were met with a lot of extra work since all of our APIs return direct image URLs and therefore we have no control over the reliability of the image URL. This issue was especially obvious for the equipment model page, since ebay sellers often remove their image from the ebay website, which causes new broken images even after we have already manually blacklisted and cleaned broken links. In phase 3, we can look into a long-term solution of this issue by using an image scraper to store the image files statically in our project directory. Third of all, we are hoping to continue to make this site much more user-friendly by adding features such as a search bar and filtering options on all 3 model pages. In addition, we need to improve the image quality and resize the pictures for some of the equipment instances, which Christopher unfortunately did not have time to get to this phase. Last but not least, we are hoping to continue improving our Github skills and familiarity; we have become more comfortable with Github as a team and are hoping to continue improving in order to reduce integration time and prevent the time-consuming task of resolving merge conflicts.

Although we faced many roadblocks, there are also many tasks that went well for our team during phase 2. For example, working with the API's went smoothly, and we were able to effectively make multiple calls without many issues. In particular, taking initiative on registering for the ebay developer account and providing all of the documentation and resources for Christopher made switching over to using the ebay API pretty streamlined. Secondly, our transition from using static data to dynamic data was completed very easily and we did not have any major issues when integrating this change into our code. Third of all, we were able to implement pagination in a way that nicely complemented the pages we had already built and is dynamic based on the number of instances for each model type. Something that also went much better this phase was that team members were more active in the Slack channel and Kaylee did a great job of being a team leader and taking initiative for all team meetings. Last but not least, we already had clear method definitions for interacting with our database, so testing the backend was very straightforward.

Phase 3

In this third phase, our team learned more about how to work with JavaScript when designing the filtering, sorting, and searching functionalities. With our searching functionality, we learned how to update the data being presented in real-time, as the user is typing their search input, as well as how to update the cards after the search has been completed and entered. For sorting, we also learned how to sort on various types of categories, such as price, number of videos, number of views, and more. We decided not to reinvent the wheel and used the sorted Python method and lambda function to be able to easily sort on any of our object's attributes. Additionally, for filtering we learned how to apply multiple filters to our model's instances. By reading the MongoDB documentation, we realized that querying each of our model collections with certain keywords could easily achieve the filtering functionality that we wanted. Finally, we learned how to stack searching, sorting, and filtering, and apply all functionalities to a group of instances at once. We also learned how to better use CSS to improve the design of our website. This included changing almost all of the default styling of the element such as color, font weight, border styles, letter spacing, and text transforms. We also added dynamic CSS to several elements such as hovering and shadows on the model instance cards and a stretching search bar. The option to filter on the model pages is also shown on a dynamic side navigation panel that was styled using CSS.

While learning many things, we also struggled with some things in this phase. For example, we struggled with stacking the three functions of searching, stacking, and sorting; because many people were working on different parts of these functions, merging the result had some difficulties and required all members of the team to be finished with their responsibilities before handling cross-interactions. Another struggle we encountered was redundancy of code; this is something we hope to improve on in the next phase as there are some aspects of our code that can be condensed. Thirdly, we struggled with maintaining the search in both real-time and after the search has been submitted; this code is something that can also be condensed in the future. Additionally, we also struggled with using CSS to dynamically style the pages and making the colors and layout look in a way that is appealing to the user. We also had a small issue where the video on the bottom of our homepage autoplays and loops for most of our team, but failed to work for Kaylee’s personal setup. We dismissed it as an outdated browser version, but are still a bit unsure on how compatible our video HTML element should be with different browsers that the user may use. Last but not least, we struggled a great deal with handling specific edge cases regarding localStorage, especially when our server is restarted. This is because the browser’s storage is independent of the state of our object arrays in our backend server, and therefore caused inconsistencies between the checkbox/sort/search state and the actual displayed cards.

We also felt that many things worked well in this phase. Firstly, we felt that our ability to use GitHub has improved and thus made our collaboration on this project easier. Secondly, we felt that our designs went well as they are easy for the user to understand and are also appealing to the eye. Thirdly we felt that our addition of a video on the homepage went well, as the implementation and integration of the video was smooth and the video itself is very fitting for our workout website to engage the visitor. Fourthly, we thought that the design of our exercises model page was very eye-catching and the GIFs really help to clearly visualize how to perform each of our exercises versus just an image. Finally, our collaboration as a team went very well as we helped each other on issues we were stuck with. Instead of strictly sticking to our own responsibilities, we worked on each other's parts too and did lots of pair programming.

Phase 4

In the fourth phase, we learned that our website had some stability issues that randomly and occasionally caused an internal server error when navigating to the model pages. We believe this was caused by how we structured and coded the website's backend that didn't allow for multiple users to filter, sort, and search at the same time. We were extremely pressed for time during the end of phase 3, and didn't thoroughly test our website after deployment. This causes us to overlook the differences between local host testing and the final deployed site. We learned how to fix this issue by slowing down before jumping into the code implementation and planning out the design patterns that we wanted to use to migrate our logic from the server-side to the client side. We learned a great deal about how to efficiently communicate between the frontend and backend, especially using detailed Flask routing rules and custom ListConverters that helped us dynamically update and pass an array of our objects between our HTML pages and our backend Python methods. This helped us learn a great deal about routing and view functions, as well as the advantages and disadvantages of trying to do complex logic with Javascript instead of communicating data to the server. Furthermore, we also learned to be much more thorough in our testing In this phase, we also explored different design patterns and learned how to apply them to our project. More specifically, we learned how to implement the Facade pattern which decoupled the clients from the server side and allowed for a more simplified interface. In this phase, we also learned how to refactor the code to improve our current designs. We learned how to better debug our project, since we had some issues with the search bar implementation as well as the localStorage persistence for different use cases. Finally, we learned how to work better as a team and did more paired programming. Kaylee and Andy benefited a great deal when doing the final refactoring since we caught each other's mistakes very often and were able to pull up references of old working code if we ran into blockers.

In this phase, we struggled with several issues with our website. The main issue we struggled with was a stability issue that caused an internal server error, as well as a major design flaw from early on in phase 3 that caused non-deterministic crashes that were almost impossible to debug. This issue was surprising to us since the website previously worked for all of us after deployment, but many bugs showed up when the TA was grading our website. Our biggest struggle was that we stored previous states, global flags, and objectArrays on the server side, which is very flawed logic when multiple people are using our website simultaneously. We also struggled with the time constraint in trying to fix issues with our website a few days before this phase was due. This is because we didn’t realize how detrimental the issue was to the functionality of our project before our project was graded. Most of our issues stemmed from not being able to handle multiple clients on our website, and lots of our longer functionality was very difficult to migrate over to the frontend since most of our team is much more comfortable with Python compared to JavaScript. Furthermore, there were also some caching and old artifact issues when using Google App Engine to deploy. Even though we uploaded the most recent files with all working code, GAE still tried to service our old static files, and an old javascript file version that broke our website many times without much that we could do to fix the issue. Our workaround was to get another teammate who had not deployed before to pull from our master branch and deploy in a fresh state. Last but not least, many times during testing where the functions of model pages worked for some people, and not for others. This was often confusing and we could definitely have benefitted from better automated testing and more frequent browser refreshing to wipe the cache and localStorage.

Despite our struggles, we had several things work well for us. We were able to implement the Facade design pattern into our project with ease. We were also able to refactor our project in several different ways without running into major problems. Our team worked well together during this phase as some of us spent a large amount of time pair programming and working together. Communication between team members was always efficient, and people were always eager to help one another with implementing ideas or understanding confusing concepts.