Item Catalog: Getting Started

PDF Download

Project Description

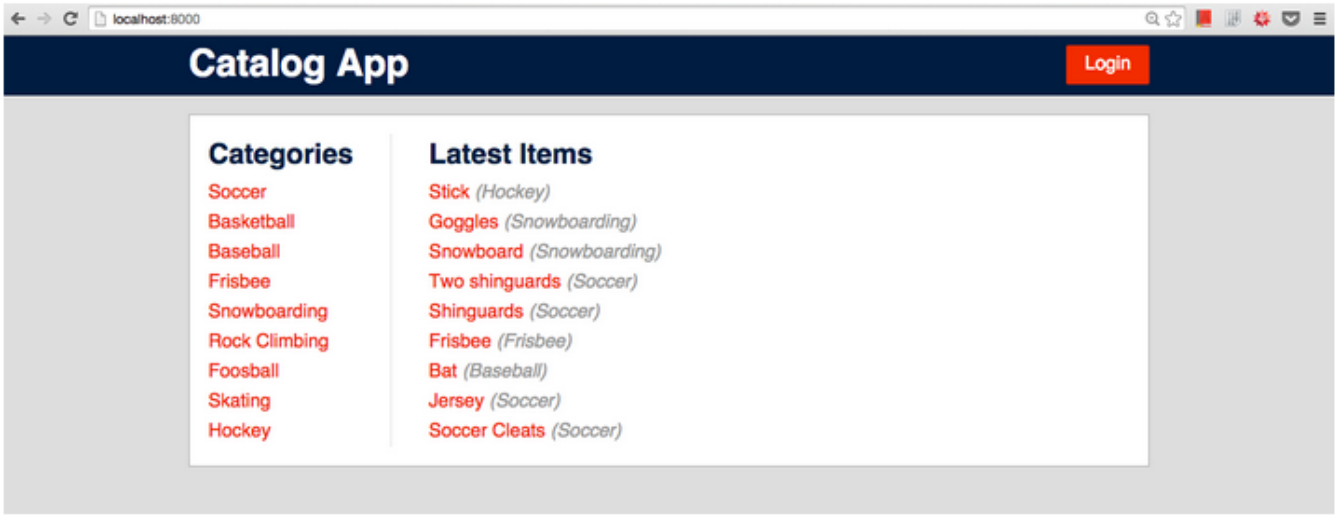
In this project, you will be developing a web application that provides a list of items within a variety of categories and integrate third party user registration and authentication. Authenticated users should have the ability to post, edit, and delete their own items.

You will be creating this project essentially from scratch, no templates have been provided for you. This means that you have free reign over the HTML, the CSS, and the files that include the application itself utilizing Flask.

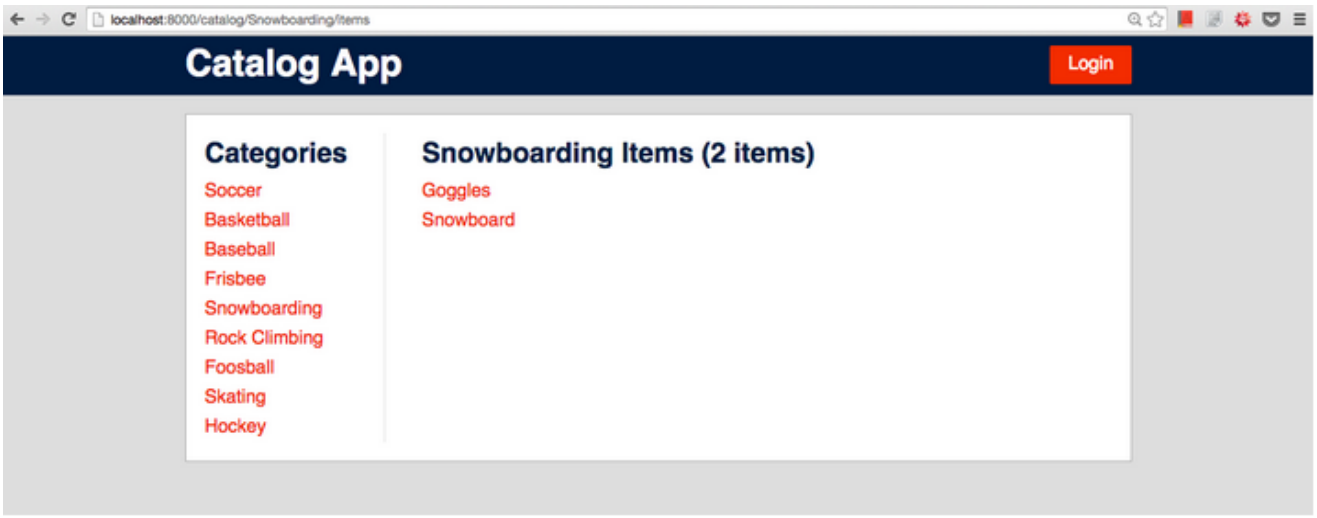
Project Display Example

Note: The screenshots on this page are just examples of one implementation of the minimal functionality. You are encouraged to redesign and strive for even better solutions.

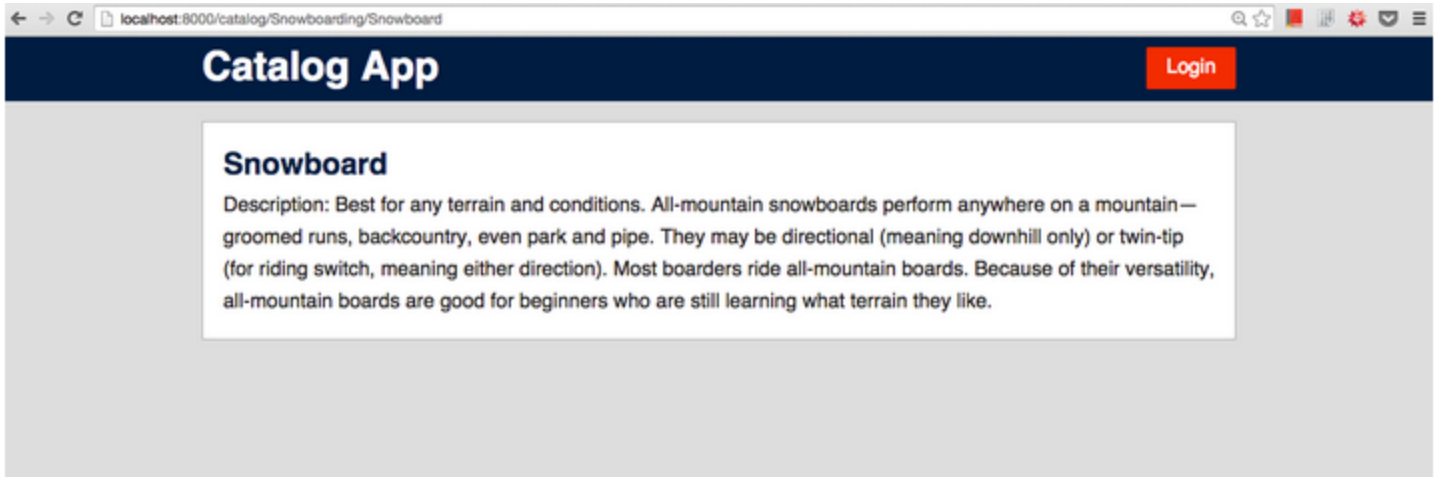
In this sample project, the homepage displays all current categories with the latest added items.



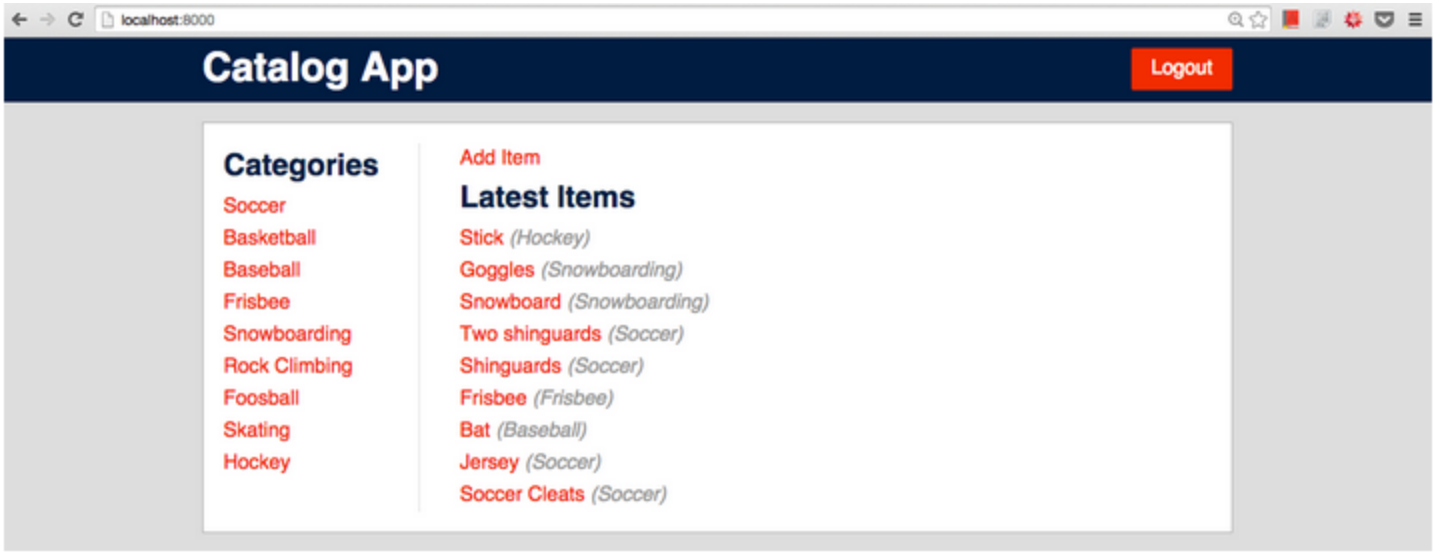
Selecting a specific category shows you all the items available for that category.



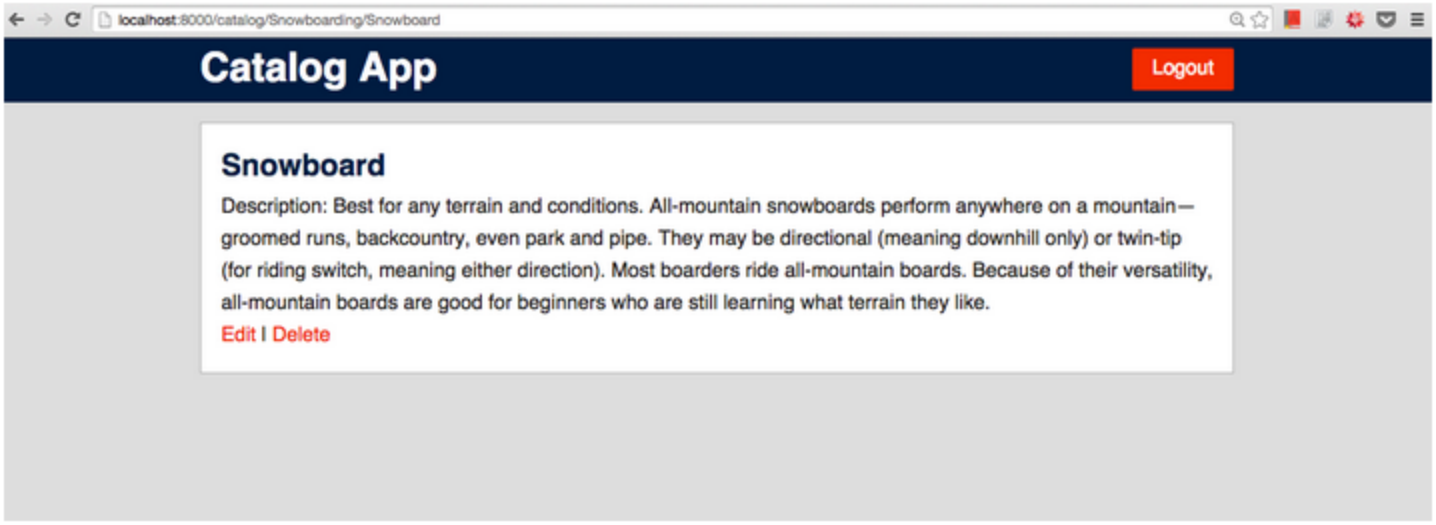
Selecting a specific item shows you specific information about that item.



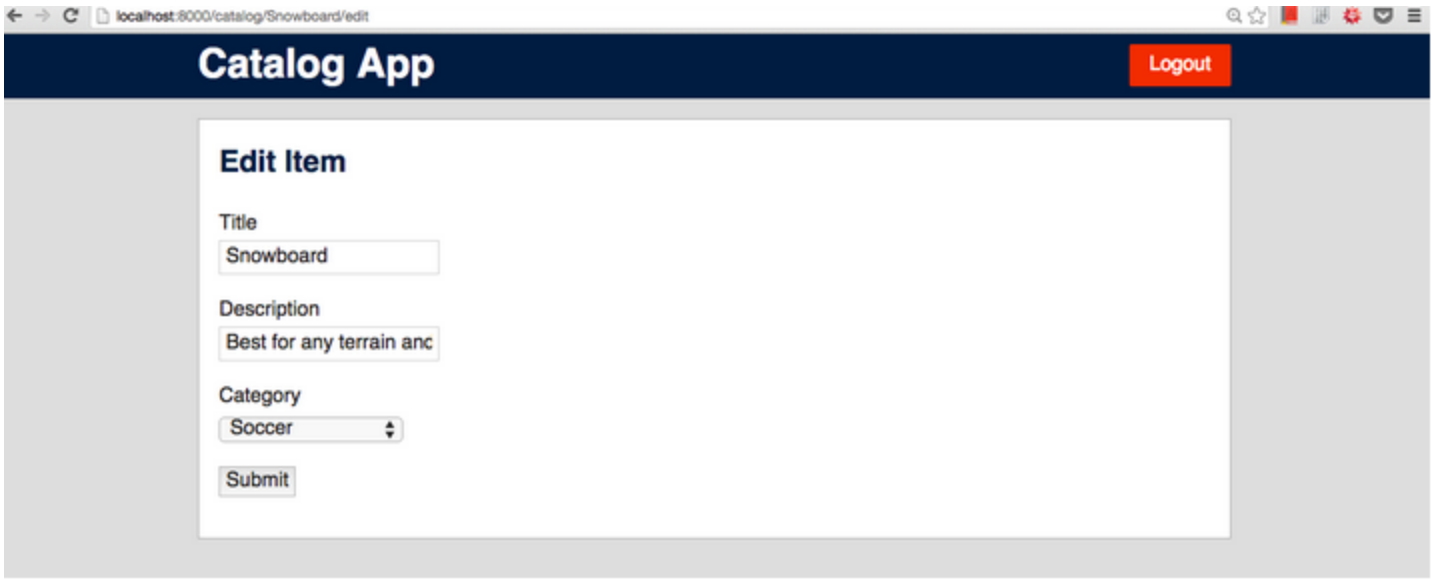
After logging in, a user has the ability to add, update, or delete item information.



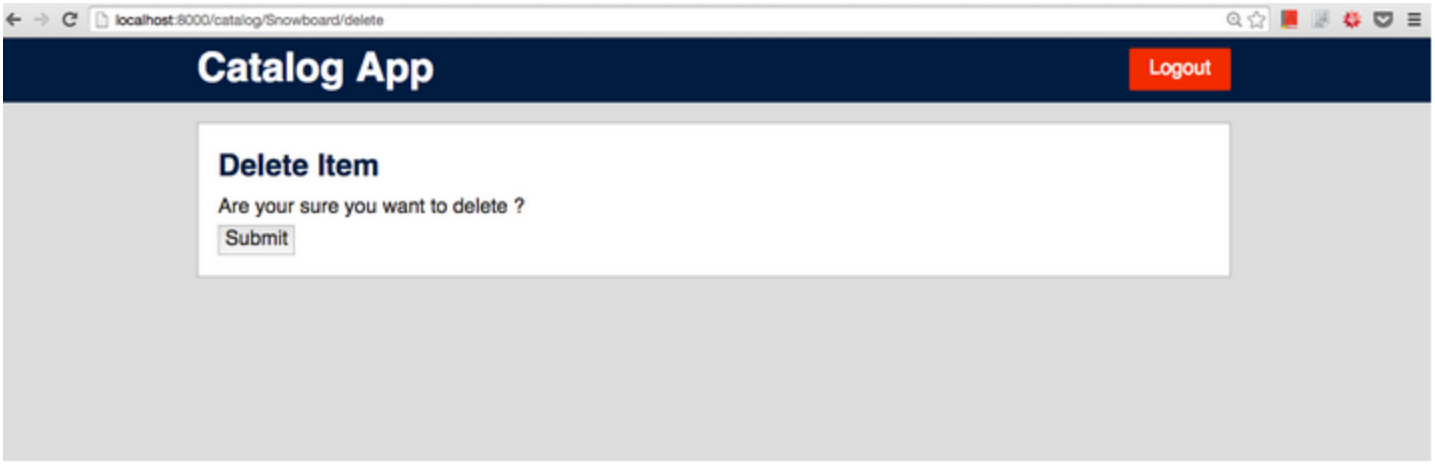
http://localhost:8000/ (logged in)



http://localhost:8000/catalog/Snowboarding/Snowboard (logged in)



http://localhost:8000/catalog/Snowboard/edit (logged in)



http://localhost:8000/catalog/Snowboard/delete (logged in)

The application should provide a JSON endpoint at the very least.

http://localhost:8000/catalog.json

Getting Started

Before we begin coding, there are several steps that you should take to make sure that you have everything downloaded in order to run your future web application.

1. Install Vagrant and VirtualBox if you have not done so already. Instructions on how to do so can be found on the websites as well as in the course materials..
2. Clone the fullstack-nanodegree-vm repository. There is a catalog folder provided for you, but no files have been included. If a catalog folder does not exist, simply create your own inside of the vagrant folder.
3. Launch the Vagrant VM (by typing `vagrant up` in the directory `fullstack/vagrant` from the terminal). You can find further instructions on how to do so here.
4. Write the Flask application locally in the `/vagrant/catalog` directory (which will automatically be synced to `/vagrant/catalog` within the VM). Name it application.py.
5. Run your application within the VM by typing `python /vagrant/catalog/application.py` into the Terminal. If you named the file from step 4 as something other than `application.py`, in the above command substitute in the file name on your computer.
6. Access and test your application by visiting http://localhost:8000 locally on your browser.

Now that we've set up the directories and downloaded the software, some of you may be wondering, what do I do now? Do I start with the front end? Do I start with application.py? Do I make my database? Now might be a good time to revisit Lesson 4 of Full Stack Foundations, where we covered these issues.

Whether you start on the front end or the back end is up to you. Some people prefer seeing the layout before thinking about the data they want to present, whereas others enjoy thinking about the structure and organization of their data and the Flask application before beginning on the front end portion of their project.

There are four parts that you will need to complete:

- the HTML (structure of the pages)
- the CSS (the style of the pages)
- the Flask Application (to put it online)
    - it must include authentication/authorization to allow users to login before making changes
- the database (to store and organize the information)

As long as you hit these four parts, it doesn't matter where you begin, whether you begin with the HTML/CSS or Flask and the database. If you're looking for a little bit more guidance, this is what Lorenzo, the Full Stack Foundations instructor had to say:

"Personally, I usually start with the database layout so that the database is modelling the information the way I want. Then I go ahead and add the backend, the Flask code, the Python code, and then I move on to the frontend where I then receive feedback on the frontend where I use the feedback to make it more stylish and elegant and presentable with everything else already in place. This is just me though, it varies from developer to developer."

Additional Functionality

In addition to the basic functions listed above, this project has many opportunities to go above and beyond what is required. Some ways to achieve exceeds specifications are to include the following requirements:

- **API Endpoints**: Research and implement additional API endpoints, such as RSS, Atom, or XML. While we do require at the bare minimum a JSON endpoint, we encourage you to research and implement a different endpoint to see what else is out there.
- **CRUD: Read**: Add an item image field that is read from the database and displayed on the page. This project is being built from scratch meaning that the information that you include and the layout of the page are entirely up to you. Add pictures for a more vibrant web application!
- **CRUD: Create**: Update the new item form to correctly process the inclusion of item images. If you included additional information to include images, there should be a way to include those images when entering in new items into the database.
- **CRUD: Update:** Update the edit/update item form to correctly process the inclusion of item images. Again, to stay consistent with the inclusion of images, items that already exist should have the option of changing the image as well.
- **CRUD: Delete:** Research and implement this function using POST requests and nonces to prevent cross-site request forgeries (CSRF).

- **Comments**: Comments are not just a way for you to keep track of what you're writing in terms of code, but also a great way to help other developers who may be reading your code. While comment preferences may differ from team to team, the general idea is that good comments cover the main purpose of the code, mention inputs and outputs, etc. Check out the comments section of PEP-8 and the Google Python Style Guide to get a better idea of good comments.

These are some points present on the rubric on how to exceed specifications, but there are many other ways to create a project that goes above and beyond to capture the attentions of future employers. Because this project is created from scratch, feel free to display the website you believe is best. Have fun with the design and the layout. Go through documentation to see if there are any other features that will cause your website to be more dynamic and efficient. The sky is truly the limit on this project.

To Submit

Once you have finished your project, go to this link here. If you have a Github account (which we recommend), connect with Github to get started. If you do not have a Github account, follow the instructions here for Mac OS X 10.0 or later, here for Windows 7, 8, or 8.1, or here for anything else. These links will help you create a Github account to submit your project.

If you run into any trouble, send us an e-mail at fullstack-project@udacity.com, and we will be more than happy to help you.