Shopify Backend Developer Intern Challenge 2022

Philip George

GitHub repo can be found here

Table of Contents

Application Architecture

Tech Stack

The application consists of a python frontend command line interface, a Flask backend server, and a NoSQL MongoDB database. **The frontend and backend also maintain a cache of product IDs, to ensure that ID duplication checks can be executed without extraneous API calls to the server.**

Database

1. The MongoDB database is hosted on a free Atlas cluster in an M0 sandbox setup with 3 nodes
2. The ‘**inventory’** database contains a single collection **‘product’** according to the requirements of the challenge
3. The Salted Challenge Response Authentication Mechanism (SCRAM) authentication mechanism is used, consisting of a username and password. The password is supplied to the intended user at Shopify via a pdf attached in the application form.
4. The product collection has a list of products containing the following fields
    a. ObjectID: Automatically assigned by MongoDB and is not exposed to the end user for modification.
    b. ProductID: Unique product ID that is taken as input from the end-user. However, it can be changed later.
    c. Product Name
    d. Product Description
    e. Product Quantity

Backend Server

1. The backend server uses the **Flask** application framework in Python.
2. It exposes 5 main API endpoints and/or functionalities which are enumerated later
3. The **pymongo[srv]** library is used to interact with the database
4. It also uses the **Pandas** library to enable easy creation of .csv files from product data.

API Endpoints

1. ‘/product/<id>’ [GET]: Queries database and returns products as a list of python dictionaries if <id> = 'all', else returns a single dictionary whose product_id matches <id>
2. ‘/product’ [POST]: Inserts a single product into database with fields and values received in the POST request JSON

3. '/product<id>' [PATCH]: Updates one or more fields of the product specified by product_id = <id>; returns "Nothing to update" if all new values are same as previous values.
4. '/product<id>' [DELETE]: Deletes the product specified by product_id = <id> from the database
5. '/product/export' [GET]: Returns all products in a comma delimited text payload, which is then saved to a CSV by the frontend

Frontend

1. The frontend uses **Python**; the user can interact with the program using console input.
2. If the user wants to export product data to .csv, the program writes to the **'project/path/downloads'** folder, which the user can then access. The **Pandas** library is not used for this, as in a real scenario, a browser would likely not have access to the **Pandas** library.

Software Architecture

This application largely follows a hybrid of the **three-tier** and **model-view-controller** software architectures.

Three-Tier Architecture

1) Presentation Tier

The presentation tier is the command line user interface in the python input console. The files in the '**src/frontend/**' folder are part of this tier.

- **a) '/app.py' :** Entry point for the application. Creates an InvCLI() object.
- **b) '/view'**
  - **i) '/view/inv_cli.py/':** Contains the InvCLI class with the main user menus
- **c) '/frontend_utils/'**
  - **i) '/frontend_utils/constants.py/':** Contains frequently used constants
  - **ii) '/frontend_utils/frontend_utils.py/':** Contains helper functions

2) Application Tier

The application tier contains the functional logic in the **'src/backend'** directory. Briefly, these are the modules

- **a) 'backend/server.py':** Main entry point to initialize and run the Flask server. Initializes database connection and creates an InvController() object, passing the database object to it. The controller in turn passes this object to the InvModel() which stores this as a class attribute.
- **b) '/backend_utils':**
  - **i) '/backend_utils/backend_utils.py':** Contains helper functions

c) **'/backend/controller/controller.py':** The functions in server.py that are linked to API endpoints delegate the actual implementation of the logic to the controller which handles the specifics of the request. It manipulates the model whenever a change to the application state is triggered.

d) **'/backend/model/model.py':** Contains the current state of the database and is the only module that interacts directly with the database.

   i) **Product ID Cache:** as mentioned earlier, the model maintains a cache of product IDs in order to run product ID duplication checks without having to query the database.

3) Data Tier

The data tier consists of the MongoDB collection 'product' within the 'inventory' database. The application tier is linked with the data tier only through the model.

MVC Architecture

The application also **loosely** follows the MVC architecture.

1) View

Logically, the view corresponds to the presentation layer as defined above. The view is the InvCLI() class that displays menus to the user and communicates with the controller through the API in server.py. All updates to the view are made via JSON responses from the server.py; the logic for these responses is embodied in the controller.py file, therefore it resembles an MVC setup.

2) Model

The model has been explained above in the Application Tier. The model is manipulated by the controller, and by extension, the linked database. The model is a representation of the database at the application tier.

3) Controller

The controller has been explained above in the Application Tier. It receives user input from the frontend View through API requests, and then determines the logic to formulate a response; in the process it manipulates the model to access the actual database.

Project Management

Tools and IDEs

1) **PyCharm** as python IDE
2) **MongoDB Compass** for debugging the database interactions
3) **MongoDB Atlas** for hosting the database
4) **Git** for Version Control