Carlos Eguiluz Rosas (cee2130)
Matthew Kersey (mlk2194)
Justin Kim (jyk2149)
Yiqiao Liu (yl4629)

Django Unchained: T3 First Iteration

Part 1:

Link to GitHub Repository: https://github.com/MattKersey/DjangoUnchained

Part 2:

User stories copied from revised proposal:

*For the second and third user stories, we haven't fully implemented the functionalities yet. So for iteration 1, we chose to focus on the first and fourth stories while also capturing some of the needs from the second and third stories that will be fully revisited in the next iteration. The detailed test plans of all four stories are described in our proposal.

- Logistics for Small Business: As a small business owner, I want my staff and I to keep track of my store's inventory, view purchases and history of inventory changes, and edit individual prices, quantities, and descriptions so that I can ensure my customers that I have their desired products in stock.
    - My conditions of satisfaction are
        - All employees have access to view the store's inventory but some employees have more access than others.
            - Ex. The ability to edit prices should only be granted to administrators like me (owner) and the manager.
        - When I edit the price for some product, I only want the price change to affect all future purchases, not those from the past which may have had different prices.
        - If I edit the description for some item (even by 1 whitespace), then the new description should be different from the previous one.
        - If I add/remove certain products, then I want to know when, how much, and (optional) why. All these changes should appear in my history.
            - Ex. Y units of X-product were purchased or restocked. I should be able to find and view the details.
- Individual vs Bulk Orders: As a holiday (tour) representative, I want my tours to be purchased in bulk (i.e. wholesale/group) so that my customers can purchase more tours at a reasonable price.
    - My conditions of satisfaction are
        - Tours can be marked for individual, bulk, or both.
            - If both, then the price of a tour in a group order must be less than that in its individual order.

        - A customer can only purchase a group tour if they meet the minimum quantity requirement. Meaning, if a customer wants to purchase N tours in a single, then N has to be >= the minimum quantity I defined for the tour to be considered bulk.
        - If there are X number of tours left, then any purchase (either individual or bulk) cannot surpass X.
- Composed Parts: As a chef, I want to know which dishes are plausible and how many I can cook given the quantity of food left in the pantry so that I don't exceed the kitchen's limits and offer our customers dishes that cannot be made.
    - My conditions of satisfaction are
        - Clear indicators of which dishes can and cannot be made given the constraints.
        - If I want to prepare some set of plausible dishes, then I want to know the maximum number of each dish I can possibly make without breaking any constraints.
        - Updates on pantry and dishes after every order since we cannot accurately predict how many dishes will be served on a given day.
        - If a customer wishes to add/remove an item in a dish (ex. extra cheese, no onions), then I would like for my waiters to check if the request can be made and how it would affect the number of plausible dishes before agreeing to it.
- CRUD Operations: As a cashier operator at a fast-fashion store, I want to find all of last week's merchandise, remove most of it from our online store, mark certain items for clearance, and "drop" (i.e add) this week's merchandise so that my customers are fully aware our new and recently discounted items.
    - My conditions of satisfaction are
        - A searchable and sorting page for cashier operators to filter, find, and sort items according to some user-inputted criteria.
        - Clear indicators that items were dropped, edited (quantity and/or price), and added.
            - Dropped: The item itself should no longer appear on the webpage, but some record should be maintained for admin purposes.
            - Edited: The item should still appear on the webpage but the price and/or quantity must be different from before.
            - Added: New item should appear on webpage.

Carlos Eguiluz Rosas (cee2130)
Matthew Kersey (mlk2194)
Justin Kim (jyk2149)
Yiqiao Liu (yl4629)

Acceptance Testing Plan:

We carried out the following acceptance-testing plan to guarantee that PyMarket is able to serve its users to the fullest extent. For each valid and invalid input, our app reacts exactly as described in the plan.

1. Create a Vendor, a Manager, and an Employee tied to a single store populated with several items. Verify that each user has the ability to see all information tied to the items and modify stock. Verify that Vendor and Manager can change the price and add and remove items, but Employee can not.
2. Create an item with high stock, and then create a transaction with that item. After this, change the price on the item as a Manager or Vendor. Repeat this process several times. Verify that the item history reflects the multiple prices on the item, not just the single last value.
3. Create an item with high stock. Create transactions with the item to decrease and increase the stock by varying amounts. Verify that stock is accurate according to the number of items in each transaction and that the date and time of each transaction is accurate.
4. Create an item with high stock, and then create a transaction with that item. After this, change the description on the item as a Manager or Vendor. Repeat this process several times. Verify that the item history reflects the multiple descriptions on the item, not just the single last value.
5. Create new items across a span of minutes, recording when each item is added. Verify that we are able to see the accurate times of creation reflected in the item history for each one.
6. Create a new item and add it to a store. Verify that store users are able to see it. As a Manager or Vendor, delete the item. Verify that store users can no longer see it, but it still is present in the Django Admin console in the form of an entry in the item_history.
7. Attempt to create a transaction that reduces an item's stock to below 0. Verify that the transaction is rejected.
8. Create a new item and add it to a store. Verify that store users are able to see it. As a Manager or Vendor, edit the item's description and price. Verify that store users including Employees now only see the modified item.

For many of these, it will be best for us to run through the inputs as if we are actual users. That being said, we could also streamline the process by automating calls to our API and verifying that the information returned by our endpoints is accurate and reflects the intentions behind the calls.

Carlos Eguiluz Rosas (cee2130)
Matthew Kersey (mlk2194)
Justin Kim (jyk2149)
Yiqiao Liu (yl4629)

Part 3:

Submit the link to the folder(s) within your GitHub repository containing all the test cases that are automatically invoked by your unit testing tool.

- Backend: https://github.com/MattKersey/DjangoUnchained/tree/main/backend/api/tests
- Frontend: https://github.com/MattKersey/DjangoUnchained/blob/main/frontend/src/App.test.js

Submit the link to the file(s) in your repository that configure the build tool and/or package manager and the automated unit testing tool.

- Production (GitHub) Workflows:
    - Build Workflow: https://github.com/MattKersey/DjangoUnchained/blob/main/.github/workflows/build.yml
    - Report Workflow: https://github.com/MattKersey/DjangoUnchained/blob/main/.github/workflows/report.yml
- Development Workflow (Builds and Reports):
    - Pynt: https://github.com/MattKersey/DjangoUnchained/blob/main/build.py
- Backend:
    - Tox: https://github.com/MattKersey/DjangoUnchained/blob/main/backend/tox.ini
- Frontend
    - Calling Test: https://github.com/MattKersey/DjangoUnchained/blob/398bc25d1274eadae5083d4a0a13a3bbb52a9e15/frontend/package.json#L24
    - Babel: https://github.com/MattKersey/DjangoUnchained/blob/398bc25d1274eadae5083d4a0a13a3bbb52a9e15/frontend/babel.config.json
    - Parcel: https://github.com/MattKersey/DjangoUnchained/blob/f4f11951e15230b9089cda60bb10f4f87e7f1861/frontend/package.json#L22-L24

Carlos Eguiluz Rosas (cee2130)
Matthew Kersey (mlk2194)
Justin Kim (jyk2149)
Yiqiao Liu (yl4629)

Part 4:

Note: Backend → Flake8 and Frontend → ESLint

Examples of Good Reports:

Back End Lint Report:
https://github.com/MattKersey/DjangoUnchained/blob/main/reports/backend/GOOD_Bug_report.txt

Back End Bug Report:
https://github.com/MattKersey/DjangoUnchained/blob/main/reports/backend/GOOD_Bug_report.txt

Front End Lint Report

https://github.com/MattKersey/DjangoUnchained/blob/main/reports/frontend/GOOD_Bug_report.html

Front End Bug Report

https://github.com/MattKersey/DjangoUnchained/blob/main/reports/frontend/GOOD_Lint_report.json

(If clean, no files will show)

Examples of Bad (Less Clean) Reports:

BackEnd Bug and Lint Report

https://github.com/MattKersey/DjangoUnchained/blob/main/reports/backend/BAD_Bug_report.txt

https://github.com/MattKersey/DjangoUnchained/blob/main/reports/backend/BAD_Lint_report.txt

FrontEnd Bug and Lint Report

https://github.com/MattKersey/DjangoUnchained/blob/main/reports/frontend/BAD_Bug_report.html

https://github.com/MattKersey/DjangoUnchained/blob/main/reports/frontend/BAD_Lint_report.json