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

Part 1:

Revise your user stories, from the first iteration and/or project proposal, to reflect what is now fully implemented, tested and working. **Do not include anything that is not actually evident in your GitHub repository and/or you will not be able to show in your final demo.**

- 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 <u>future</u> 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.
- 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 sorting functionality for the item-history page where each field is sortable.
  - 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.

Part 2

Write a test plan that explains the ***equivalence partitions*** and ***boundary conditions*** necessary to unit-test each of the major subroutines in your system (methods or functions, excluding constructors, getters/setters, helpers, etc.) and then implement your plan. Associate the names of your specific test case(s) with the corresponding equivalence partitions and boundaries (if applicable). Your test suite should include test cases from both ***valid*** and ***invalid*** equivalence partitions, and just below, at, and just above each equivalence class boundary (or inside vs. outside the equivalence class when boundary analysis does not apply).

Link to Backend test suite:

https://github.com/MattKersey/DjangoUnchained/tree/main/backend/api/tests

Link to Frontend test suite:

https://github.com/MattKersey/DjangoUnchained/blob/main/frontend/src/App.test.js

Backend Test Plan for **Main** Subroutines:

User - add_store
• Attempt to create store with valid conditions
• Attempt to create store with invalid conditions
  ○ Invalid or missing data
  ○ User to set as owner/vendor does not exist
	Test_add_store
	Test_add_store_bad_user
	Test_add_store_too_long_name
	Test_add_store_no_name
	Test_add_store_no_address
	Test_add_store_no_category
	Test_add_store_bad_category

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

User - remove_store ● Attempt to detach a store from a user with valid conditions ● Attempt to detach a store from a user with invalid conditions ○ User attempting operation lacks proper clearance ○ User/store/association between the two does not exist	Test_remove_store Test_remove_store_employee_auth Test_remove_store_bad_user Test_remove_store_bad_store Test_remove_store_bad_association Test_remove_store_bad_role
User - delete_store ● Delete a store with valid conditions ● Delete a store with invalid conditions ○ User/store/association does not exist ○ User attempting operation lacks proper clearance	Test_delete_store Test_delete_store_bad_user Test_delete_store_employee_auth Test_delete_store_manager_auth Test_delete_store_bad_store Test_delete_store_bad_association Test_delete_store_bad_role
User - change_role ● Change the role of someone when you have/don't have exiting association to store ○ "Don't have" covers invalid and invalid store ● Change the role of another user that has existing/non-existing association to the store ● Change the role of someone based on your role	Test_change_role_no_association Test_change_role_pre_association Test_change_role_bad_user Test_change_role_bad_store Test_change_role_bad_role
User - add_user_to_store ● Add an employee/manager/vendor to some valid store ○ Test for duplicate emails (unique identifier) ○ Test for missing fields ● Add a user (regardless of role) to an invalid store	Test_employee_add_user Test_manager_add_user Test_vendor_add_user Test_add_user_to_invalid_store Test_invalid_user_adds_user Test_add_user_with_missing_fields Test_adds_user_duplicate_email
User - register ● Test invalid/valid user ○ Invalid meaning that a user by the email exists in application	Test_register_good Test_register_bad
User - list	test_list_user

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

• List of all users in application	
User - retrieve • Retrieve detail of one user in application ○ Valid and invalid (doesn't exist)	Test_retrieve_user Test_retrieve_user_does_not_exist
User - update • Update user with valid information • Attempt to update nonexistent user • Attempt to update user with an email address attached to another user	Test_update_user Test_update_user_does_not_exist Test_update_user_email_already_used
User - current_user • Test getting user based on OAuth credentials passed to endpoint	Test_current_user

Store - retrieve • Retrieve a store's detail based on existing/non-existing association (meaning the user is connected to the store by some role)	Test_retrieve_store_assoc Test_retrieve_store_no_association

Store - update
• Update an invalid/valid store
  ○ Unique store name
  ○ Name too long
• Update a store based on user's authentication
	Test_update_store
	Test_update_store_bad
	Test_update_store_employee_auth
	Test_update_store_manager_auth
	Test_update_store_too_long_name

Store - purchase_items
• Purchase an invalid/valid list of items
  ○ No item
  ○ Regular and bulk items
  ○ Non-existent items
  ○ Items from another store
• Purchase within or more than remaining stock
	Test_purchase_items
	Test_purchase_items_bulk
	Test_purchase_items_bad_store
	Test_purchase_items_over_stock
	Test_purchase_items_bad_item

Store - create_checkout_session • Checkout items from invalid/valid store	Test_purchase_items_wrong_store Test_create_checkout_session Test_create_checkout_session_bulk

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

• Checkout items
  ○ Regular and bulk items
  ○ No items	

Store - add_item • Add an invalid/valid item • Add such item to invalid/valid store • Add an item based on authentication	Test_add_item Test_add_item_bad_store Test_add_item_employee_auth Test_add_item_manager_auth
Store - remove_item • Remove an invalid/valid item • Remove such item to invalid/valid store • Remove an item based on authentication	Test_remove_item Test_remove_item_bad_item Test_remove_item_bad_store Test_remove_item_employee_auth Test_remove_item_manager_auth
Store - delete_item • Delete an invalid/valid item • Delete such item to invalid/valid store • Delete an item based on authentication	Test_delete_item Test_delete_item_bad_item Test_delete_item_bad_store Test_delete_item_employee_auth Test_delete_item_manager_auth

Item - retrieve • Retrieve list of items from invalid/valid store	Test_retrieve_item Test_retrieve_item_bad
Item - create • Create an invalid/valid item • Create an item and associate it with a bad store • Create item with different authentications	Test_create_item Test_create_item_bad_store Test_create_item_employee_auth Test_create_item_manager_auth Test_create_item_bad_item
Item - update • Update an invalid/valid item • Try to update when no updates were made • Update item with different authentications	Test_update_item Test_update_item_no_update Test_update_item_employee_auth Test_update_item_manager_auth Test_update_item_invalid_prices Test_update_item_bad_item

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

OAuth - redirect ● Use the OAuth redirection endpoint as a Vendor and Employee to verify that tokens are properly assigned	Test_OAuth_redirect Test_OAuth_redirect_employee
OAuth - access ● Access a dummy endpoint with both valid and invalid credentials	Test_bad_access Test_good_access
OAuth - update_token_scope ● Update the scopes attached to a user's token after they have had a role change ○ Test for changing to all three roles for a given store	Test_update_user_scopes_employee Test_update_user_scopes_manager Test_update_user_scopes_vendor

Scopes ● Get the custom OAuth scopes dynamically generated for the PyMarket's list of stores	Test_get_all_scopes Test_get_available_scopes Test_get_default_scopes

User Model ● Create valid users with multiple privilege levels ● Attempt to create users with invalid properties ○ Bad or previously used email address ○ No password	Test_duplicate_email Test_invalid_email Test_no_email Test_staffuser Test_superuser Test_superuser_no_password Test_user
User Manager Model ● Attempt to create a user through the user manager with a non-existent email address	Test_value_error
Item Model ● Create valid items with both default and specified values ● Attempt to create items with invalid inputs	Test_item Test_price_default Test_invalid_name Test_invalid_stock Test_no_price

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

○ Bad name format ○ Negative stock ○ Non-existent price ○ Invalid bulk vs individual behavior	Test_no_bulk_attributes Test_bulk_price_higher_than_price Test_invalid_price
Store Model ● Create a valid store with no/invalid/valid items ○ Label store with existing and non-existing categories (expect fail when entering non-existing categories) ● Associate the store to the user via and verify that the store exists	Test_store Test_user_with_store Test_invalid_item_to_store Test_non_other_category Test_no_other_category
Association Model ● Associate user to store ○ Valid user, valid store ○ Valid user, invalid store ○ Invalid user, valid store ○ Invalid user, invalid store ● Duplicate Association (expect to fail as this is a foreign key relationship)	Test_association Test_association_invalid_user Test_association_invalid_store Test_assocaition_invalid_combo Test_association_duplicate

Frontend test plan - test cases name:

● App ● Login page by default ● Expect login component within default screen	Get Login Screen by default Test Login Component
Shop ● Add an item in shopping page ● Expect “add-an-item” button in page	Test Shop Page Test Add an Item Button
Add-Item-Form ● Once the user clicks “add-an-item” button, the form should appear	Test AddItemForm

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

Add-Shop-Form ● Once the user clicks “add-a-store” button, the form should appear	Test submit AddShopForm
Change-Add-Shop-Form ● User can change the name of the store on the shop adding page	Test Change AddShopForm
Store-Card ● The store card should appear within the user page that list of all associated stores to the user ● Clicking the store should redirect to store page with all the item details	Test StoreCard Test Click StoreCard
User-Page ● The user page should list all associated stored ● Once the user clicks on the button to add a store, the form should appear ● The user should also be able to close the form and cancel submission ● The user should be able to create a new store on the user page	Test UserPage Test UserPage Open Modal Test UserPage Close Modal Test UserPage Submit Modal
Error ● The error page should appear when the user enters some invalid url	Test Error Page
Navbar ● Tests the navbar on top of every page	Test Navigation Bar

Part 3:

Link to coverage reports:

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

Frontend:
https://github.com/MattKersey/DjangoUnchained/blob/main/reports/frontend/coverage.txt

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

Codecov Report (main): https://codecov.io/gh/MattKersey/DjangoUnchained/tree/main

We're following our Codecov Report which reports a branch coverage of **96.78%**. We weren't able to achieve 100% coverage because our coverage tool is also capturing Django configuration-related files that aren't directly relate to the (core) functionalities of our API and webpage (ex. backend/wsgi.py, backend/apps.py ← if we have multiple Django applications besides our API).

Part 4:

For this project, we used GitHub Actions:

Link to workflow configurations:

https://github.com/MattKersey/DjangoUnchained/tree/main/.github/workflows

Link to CI Reports:

https://github.com/MattKersey/DjangoUnchained/actions

We have two workflows: one for creating the builds and another for generating the reports. The one more relevant to this part is the "Build Workflow".