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

# Django Unchained: T1 Project Proposal

Part 1:

1. **What will your project do?**

   Our project, PyMarket, aims to simplify business inventory management specifically optimized for small business owners. This means tailoring the experience so that one owner can set up, monitor, and adapt their business with one single tool.

2. **Who will be the users?**

   PyMarket will have several groups of users. **Vendors** will be able to perform actions such as listing their inventories and setting prices. **Shoppers** (general users) will be able to view shop inventories and make purchases. In addition, we will have **admins** who are able to manage information for other users.

3. **How will you demo the project?**

   For vendors and admins, we can utilize Djanjo's native admin console to showcase a sample of inventory (along with their prices) and publish a change in prices from either the console or via an endpoint. For shoppers, we can demo a small webpage where the user is presented with items to purchase and create a sample of inventory updates that will be reflected on the client side.

4. **What kind of data do you plan to store?**

   Since we have two large categories of vendors, we will store different information for each type of user:

   **Vendor**:

   > Information about the vendors (Name, log-in credentials, card/paypal info, business address, shipping information) and their inventories (name, amount left, price, VAT).

   **User:**

   > Information about the shoppers (Name, log-in credentials, payment credentials, purchase history) and the products they bought (name, amount, price).

5. **What public api will you be using?**

   We will be using a couple of Data Validation APIs to validate business and customer information such as email, street address, VAT numbers for products, etc. Also, in order

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

to make it easier for our businesses to set up their catalog, we will be using public image api's to get stock images for their products.

---

Part 2:

>>> 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 and quantities 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 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.

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

>>> 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.

---

Part 3:

1. Testing for the first user story (small business owner):
   - Let an employee user of a small business view the store's inventory: pass
   - Let an employee user of a small business edit an item's price: fail
   - Let an administrator of a small business edit an item's price: pass
   - Let an administrator of a small business edit an item's price and check a past purchase price. If the price changes, fail. Else, pass.
   - Let an administrator of a small business edit an item's price and make a purchase of that item. If the price reflects the changed price, pass. Else, fail.
   - Add an amount of an item in the inventory. Check if that change and time of the change are reflected in the history. If yes, pass. Else, fail.
   - Delete an amount of an item in the inventory. Check if that change and time of the change are reflected in the history. If yes, pass. Else, fail.
2. Testing for the second user story (holiday representative):
   - Let a representative mark a tour as bulk (and define the minimum quantity): pass
   - Let a representative mark the price of the individual tour lower than that of the bulk one: fail
   - Let a customer purchase an amount of a tour marked as bulk, while the amount is lower than the minimum quantity: fail
   - Let a customer purchase an amount of a tour, while the amount is larger than the amount left for that tour: fail
3. Testing for the third user story (chef):

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

- Let a customer request a dish or an item. Check if the app gives the correct feedback based on the stock. If yes, pass. Else, fail.
- Let a customer request a dish that the kitchen is able to offer. Check if the corresponding stock changes after the request. If yes, pass. Else, fail.
- Let a chef/waiter input a dish/single item. Check if the app gives the correct amount of this dish that the stock is able to support. If yes, pass. Else, fail.

---

Part 4:

**Framework:** Django (Backend) + React (Frontend)

Django is a Python-based web framework. React is a JavaScript library that will help us create a simple and easy-to-use frontend for our application.

**IDE:** VS Code

VS Code is a common IDE with which we are all familiar. With all of us using it, collaboration should be easy.

**Build Tool:** Pynt + Virtualenv + Travis (Backend) npm + Parcel + Babel (Frontend)

Pynt takes the place of a makefile and allows us to specify a set of build tasks. Virtual environments will allow us to have a greater level of control over packages. Travis is a CI tool that will help us confirm the success of the build and install dependencies in our virtual environment. npm acts as a package manager and allows us to start builds of our frontend. Parcel bundles our application. Babel translates modern JavaScript used by React so that legacy browsers can understand it.

**Style Checker:** Black (Backend) StandardJS (Frontend)

Black will both check and format our code to ensure that it meets their standards. StandardJS is easy to install and use and will take care of checking our JavaScript style.

**Unit Testing:** tox + PyTest (Backend) Jest (Frontend)

PyTest will allow us to write unit tests for our Django code, and tox will let us specify conditions for those tests (see tox link). Jest is a testing framework that works well with React and Babel.

**Coverage:** Coverage.Py (Backend) Jest (Frontend)

Coverage.py is easy to use and provides detailed information about Python code coverage including lines missed. Jest (which is doing testing for our frontend) also supports code coverage checking for React.

**Bug Finder:** PyChecker (Backend) JSHint (Frontend)

PyChecker will allow us to find bugs quickly and easily in our Django code. JSHint will work similarly with our React code.

**Data Store:** AWS RDS free tier running PostgreSQL

AWS RDS is easy to set up and will allow us to have a common database. PostgreSQL integrates nicely with Django and is straightforward to use.