# Design Document

Feature Name: Django-based Inventory Management system with custom User ­Model

Start Date: 29/07/2021

# Overview

The inventory management system was created to satisfy the requirements of managing the inventories and warehousing problems. The built system was written in Python using Django framework and incorporating the solutions for several use cases related to inventory management. The solution is currently running on a local machine (localhost) with the idea of hosting it into a cloud platform. The MVP should be optimized before implemented into production.

## Purpose & Goal

The purpose of this solution is to address the issues of managing several components with future enhancement in mind, namely:-

1. Stakeholders (multiple users of the system)
2. Business requirements (management of the products and warehouses)
3. Business management (streamlining the process flows of inventory management)

For above-mentioned purpose, I will be overriding default Django user model to add few attributes such as user group for access level. The remaining features will be developed on Django default with the assistance of Javascript and Crispy forms libraries.

The goal is to let the system to register, authenticate, validate and logout the user according to their respective permissions accordingly with the thinking that the similar design should be transferrable into Blockchain Indy and Fabric development.

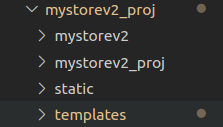
## Motivation

Why this approach? What use case does it support? What is the expected outcome? Screen shots?

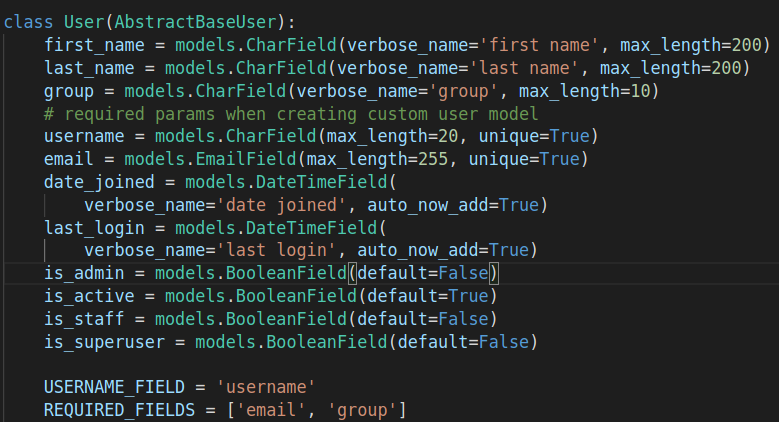
The system built on Django native may incur some limitations in the future especially the scalability and future ehancement, hence, the overiding of Django Default User Model, the uses of Class-based Views (CBV) and Role-based Access Control (RBAC) framework are implemented with the followings in mind:-

1. Dynamic Implementation.
2. Catered for larger use cases in future.
3. Proven security framework
4. Codes simplicity.
5. Adhere to DRY principle of Python.

Python – Django was chosen due to its simplicity, modular, open-source and vast libraries support. Reference of codes are easily available and shorten the maintenance time.



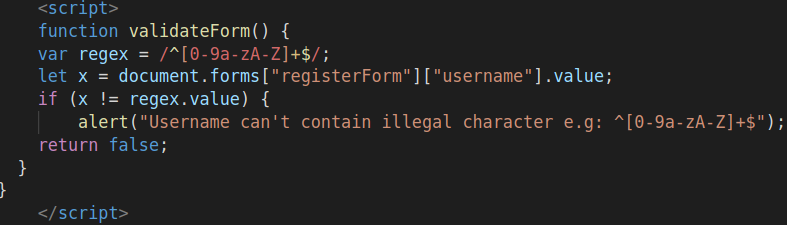
*App-based implementation making the future development easier e.g app “mystorev2”.*

**

*Overriding default Django user model will allow greater flexibility on user requirements.*

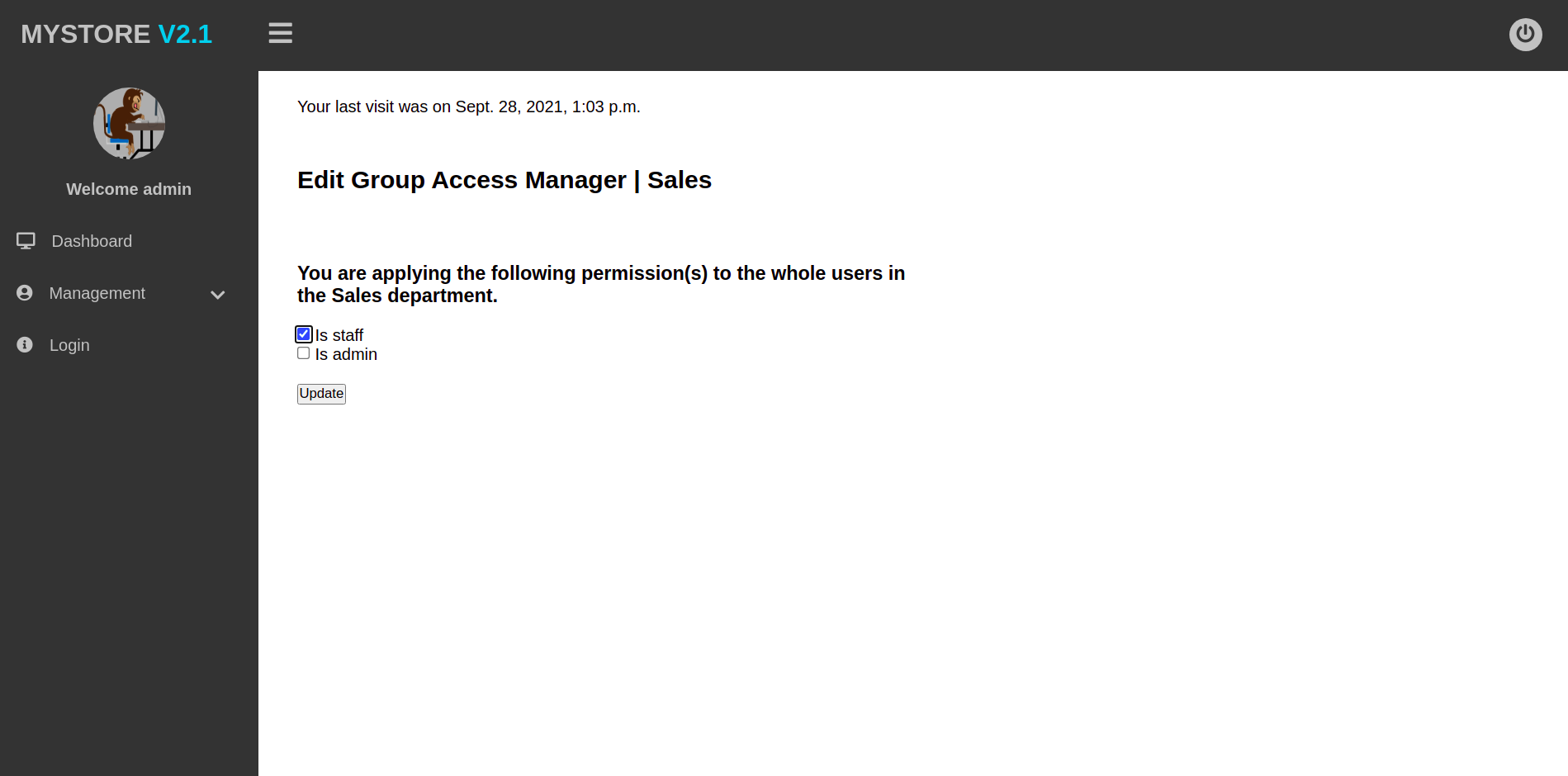
Due to the small size of users at the moment and priority over releasing the MVP to the customer, The developer have decided to maintain the uses of Sqlite as primary database. Future needs for scalability would require migration of the database into a better database to cater for larger user base such as SQL and Posgres.

Nonetheless, the developer have also overide the the default User model to avoid difficulty in scalability in the future. Overiding the existing Django user model allowing the developer to continuously expanding the model attributes to cater for different use cases as the needs arise.

*Extending the control to front-end to ensure data integrity*

On top of the data integrity protection by using Django models and forms, Javascript library is used to establish extra layer of protection to protect the database integrity that may jeopardize future migration on scalability.

**Role-Based Access Control**

*RBAC: Applying group-wide permissions*

Motivation & Method Summary:

On top of the overiding method of user model, the implementation was also designed to use CBV instead of Functional-Based View (FBV) as to conform with industry practice at the same time of elimination of redundancies in codes by utilizing the existing libraries. This is also in adherence od DRY (Dont-Repeat-Yourself) principle in Python which will ensure a much efficient system operation. CBV also is also helps in future maintenance which cuts down significant lines of codes and making the codes more organized and easier to maintain.

On the implementation based on RBAC methodology instead of ACL (Access Control List);

For most business applications, RBAC is superior to ACL in terms of security and administrative overhead. ACL is better suited for implementing security at the individual user level and for low-level data, while RBAC better serves a company-wide security system with an overseeing administrator. An ACL can, for example, grant write access to a specific file, but it cannot determine how a user might change the file.

## Guide-level explanation

Python:

* Required packages are mentioned in “requirements.txt” which can be found in Github.

Django:

* Required installations for developments can be perform by running (on top of Python installations)

**$** python -m pip install Django

Javascript:

* No requirement on installation for development other than described above. JavaScript code runs entirely within the browser. As modern browsers handle their updates automatically, there is no setup needed in order to run JavaScript.

Process description:

* After a user has filled the registration form and then submit, a module in this feature checked for whitespaces, illegal characters, encrypt user password, and then store all info as plaintext except for password which is converted to hash before storing into the database. Store and retrieve info from Sqlite. Information should be accessible by all.
* Based on the business requirements, an administrator should be approving new user for security purpose. The user information will be retrieved through “Management Access > Dashboard” where modification and access can be granted accordingly

(*e.g. is\_staff = True , is\_authenticated =True etc.*)

* Approved user should be able to log in after approval according to their permission/ access level. For example:

a. Unlogged user is unable access everything.

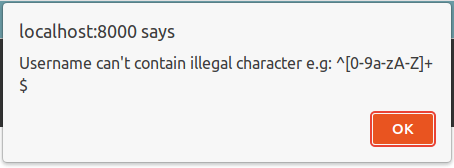
b. Logged user but not a staff (which whill be granted by the admin) is unable to access the administrator component.

c. Logged user and a staff is able to manipulate / make changes to the inventory.

d. Logged user and an admin is able to perform changes and as well as granting access to newly registered users.

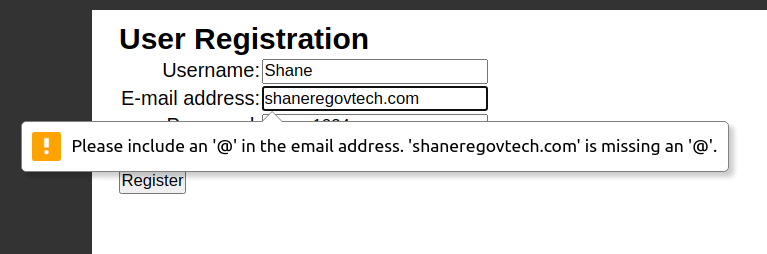
List of Errors Dialog:

* Front-End: Web
* Msg1 = “Illegal Characters”



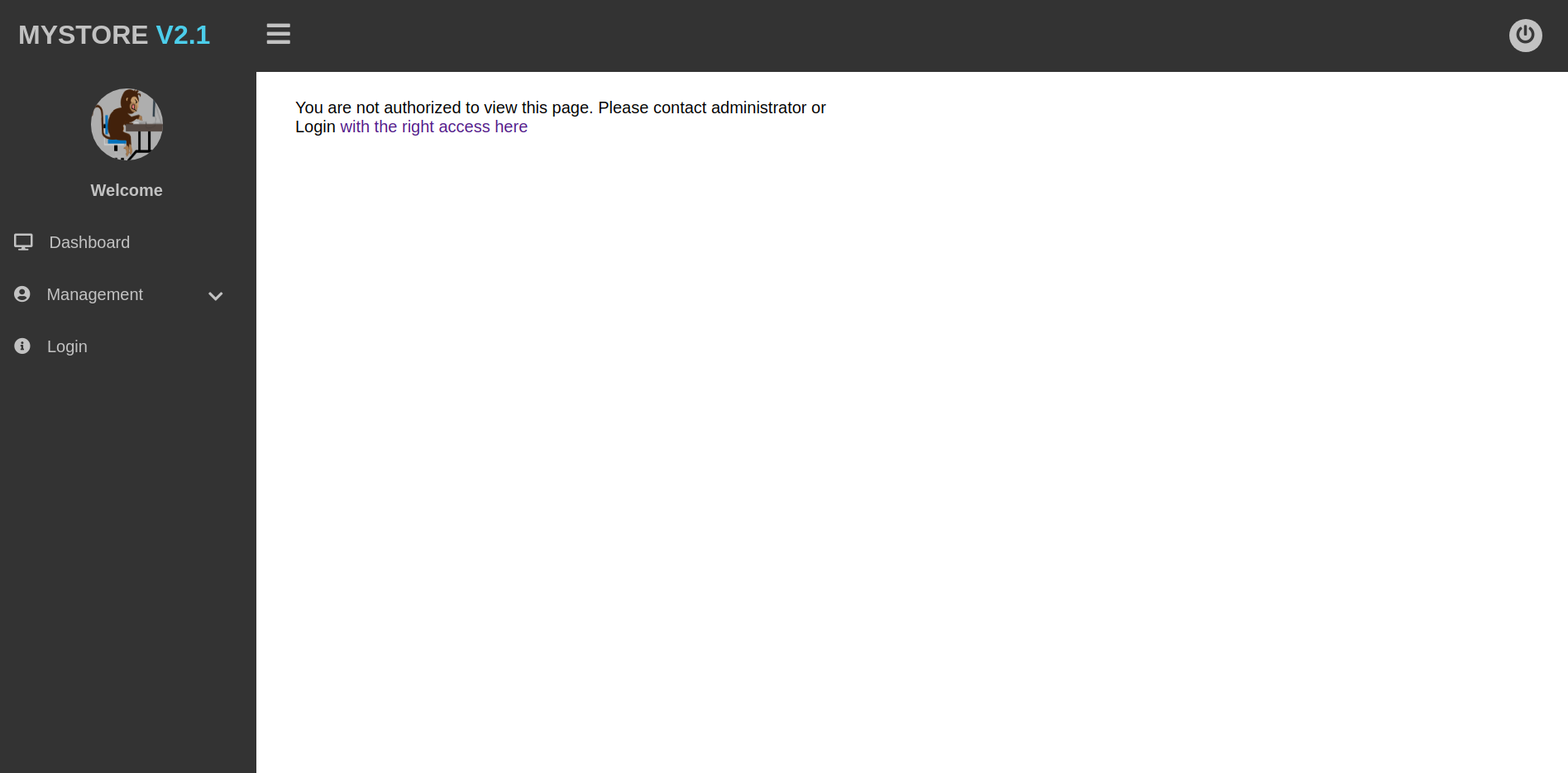
*Registration: Error shown when user input illegal characters in Username field.*

* Msg2 = “Data format is inaccurate”



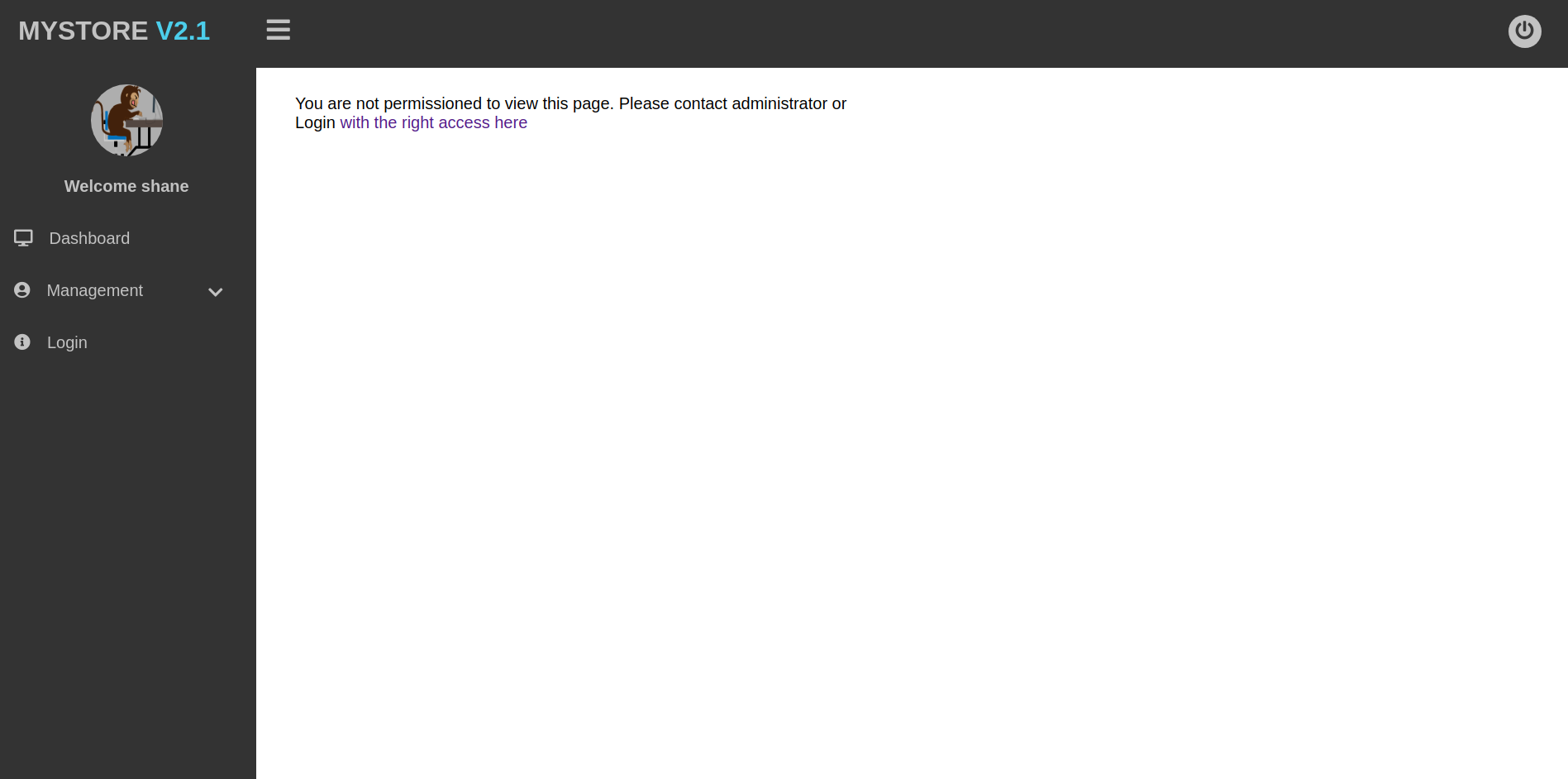
*Registration: Error shown when user missed the format for email field.*

* Msg3 = “Unauthorized access”



*Dashboard: Error shown when logged user does not have “is\_staff” approval.*

* + Msg4 = “Unauthorized access” / Error 403 equivalent in Django



*Management: Error shown when logged in user does not have sufficient permission.*

Back-End

* To be implemented
* Impacts & summary:

a. Ensuring data integrity to be protected at all times.

b. Ensuring data privacy to be protected by unauthorized personnel.

c. Ensuring a better control over data manipulation.

Summary:

Multiple level of protections allow greater control and security over the system which will help with the implementation on the Hyperledger Indy wallet. All the controls will allow misinput of important data such as wallet address.

## Design alternatives:

List pros and cons for each. Prefered choice.

There are alternative design solutions for the architecture. For greater form of flexibility, a RESTful API which can be implemented by using either Flask or Django Rest Framework (DRF) along with many others.

While all threes are great solutions to be used each and every one of them comes with a set of pros and cons that needs to be carefully considered before implementation. For example, DRF would be highly compatible and provide a modular, customizable and straightforward integrations with Mystorev2 as the system is built on Django. The framework is also highly popular due to its reusability of its code hence making the code structure is highly efficient and easier for developers to add more functionalities.

However, implementation of DRF should be thread with cautions as it may consume project development speed due to its reusable modules. The compatibility checks on previous against newer releases added the weight unto its nature to be slower as compared to its alternatives.

The other alternative framework such as Flask is a micro web framework written in Python, easier to be setup and great for minimalistic design web applications. Flask is also flexible and simple to setup hence, speeding up the process of development and allows a greater understanding of the developers. Nonetheless, Flask’s method of handling multiple requests in in turn is taking a toll on its performance especially with high-load enterprise software.

Preferably, the developer think Flask is better suited to be the alternative design solutions for MyStore due to its simplicity and shorter project time. This will allow a fully efficient development process and the advantages of Flask outweighs the other which makes it a preferrable choice.

## Drawbacks

Request for comments (RFC) - Non for now

## Accomplishments and Performance

* Should include after module unit testing or integration..