# Grocery Store V2

Problem definition

Modern Application Development - II

Last Updated: 23-06-16

# Frameworks to be used

- Flask for API
- VueJS for UI
- VueJS Advanced with CLI (only if required, not necessary)
- Jinja2 templates if required
    - Not to be used for UI
- Bootstrap, if required (No other CSS framework is allowed)
- SQLite for database (No other database is allowed)
- Redis for caching
- Redis and Celery for batch jobs
- It should be possible to run all the demos on the student's computer, which should either be a Linux based system or should be able to simulate the same. You can use WSL for Windows OS.

# Grocery Store

- It is a multi-user app (one required admin, store manager and other users)
- Used for buying grocery
- User can buy many products for one or multiple sections
- Store manager can add products and request addition of new section/category
- Each section/category will have
  - ID
  - Name etc.
- Each product will have
  - ID
  - Name
  - Manufacture/Expiry date
  - Rate per unit etc. (Rs/Kg, Rs/Litre)
- Every category can have a number of products
- System will automatically show the latest products added

Terminology

- Inventory
- Section/Category - List of products, amount etc.
- Product - Name, Price etc.
- Dynamic Pricing (optional) - Product prices can go up/down depending upon the season/demand

# Similar Products in the Market:

1. BigBasket
    - Web, IOS and Android
2. Blinkit
    - Web, IOS and Android

Refer to the wireframe given below;
GroceryStoreApp

- These are meant for exploring the idea and inspiration
- Don’t copy, get inspired

# Core Functionality

- This will be graded
- Base requirements:
  - User signup and login (using RBAC)
  - Mandatory Admin Login (using RBAC)
  - Store Manager Signup and Login (using RBAC)
  - Category and Product Management
  - Search for Category/Product
  - Buy products from one or multiple Categories
- Backend Jobs
  - Export Jobs
  - Reporting Jobs
  - Alert Jobs
- Backend Performance

# Core - User Signup and Login

- Form for username and password (both login and signup)
- Use Flask Security or JWT based Token Based Authentication only
- Suitable model for user

# Core - Admin Login

- Form for username and password (can be same as other roles)
- Use Flask Security or JWT based Token Based Authentication only
- Suitable model for user

Note: The application should have only one admin.

# Core - Store Manager Signup and Login (Using RBAC)

- Form for username and password (can be same as other roles)
- Add an admin user whenever a new database is created
- The app should clearly differentiate among various roles
- Every new store manager sign up should be approved by the admin
    - The request should automatically go to the admin’s dashboard for approval

# Core - Section/Category Management (Only for Admin)

- Create a new section/category
  - Storage should handle multiple languages - usually UTF-8 encoding is sufficient for this
- Edit a section/category
  - Change title or description
- Remove a section/category
  - With a confirmation from the admin
- Approve requests from store managers to add new categories, edit or delete existing categories

# Core - Product management (Only for Store Manager)

- Create a new product
  - Storage should handle multiple languages - usually UTF-8 encoding is sufficient for this
- Edit a product
  - Change title, description, category or image
- Remove a product
  - With a confirmation
- Allocate section/category while creating products
- Request admin to add new section/categories or edit/remove existing ones

# Core - Search for products

- Ability to search products based on section/category
- Ability to search products based on price, manufacture date etc.

# Core - Shopping Cart for User

- Ability to add multiple products in a cart (all products may or may not belong to same category)

# Core - Buy Products

- Display all the products available for a given category to the users
- Ability to buy multiple products from one or multiple sections.
- Ability to show out of stock for the products that are not available.
- Ability to show the total amount to be paid for the transaction.

# Core - Daily Reminder Jobs

- Scheduled Job - Daily reminders on Google Chat using webhook or SMS or Email
  - In the evening, every day (you can choose time of your choice)
  - Check if the user has not visited/bought anything
  - If yes, then send the alert asking them to visit/buy

# Core - Scheduled Job - Monthly Activity Report

- Scheduled Job - Monthly Activity Report
  - Come Up with a monthly progress report in HTML (email)
  - The activity report can consist of orders made by a user in a given month, total expenditure, etc.
  - On the first day of the month
    - Start a job
    - Create a report
    - Send it as email

# Core - User Triggered Async Job - Export as CSV (Only for Store Managers)

- User Triggered Async Job - Export as CSV
  - Come up with an export CSV format for products
  - This export is meant to download the product details (name, stock remaining, description, price, number of units sold, etc.)
  - Have a dashboard where the store manager can export
  - Trigger a batch job, send an alert once done

# Core - Performance and Caching

- Add caching where required to increase the performance
- Add cache expiry
- API Performance

# Recommended (graded)

- Well designed PDF reports (User can choose between HTML and PDF reports)
- Single Responsive UI for both Mobile and Desktop
  - Unified UI that works across devices
  - Add to desktop feature

# Optional

- Styling and Aesthetics
- Product Reviews and predicting popularity of a product based on ratings and reviews

# Evaluation

- Report (not more than 2 pages) describing models and overall system design
  - Include as PDF inside submission folder
- All code to be submitted on portal
- A brief (2-3 minute) video explaining how you approached the problem, what you have implemented, and any extra features
  - This will be viewed during or before the viva, so should be a clear explanation of your work
- Viva: after the video explanation, you are required to give a demo of your work, and answer any questions
  - This includes making changes as requested and running the code for a live demo
  - Other questions that may be unrelated to the project itself but are relevant for the course

# Instructions

- This is a live document and will be updated with more details and FAQs (possibly including suggested wireframes, but not specific implementation details) as we proceed.
- We will freeze the problem statement on or before 25th June, beyond which any modifications to the statement will be communicated via proper announcements.
- The project has to be submitted as a single zip file.
- Please refer to the guidelines document available on the portal.