Software Requirements Specification

Ez Shopping

1.0.0

Ryan Arnold; Janki Parmar

March. 22, 2025

Table of Contents

Revision History

1 Introduction
- 1.1 Purpose
- 1.2 Document Conventions
- 1.3 References

2 Overall Description
- 2.1 Product Perspective
- 2.2 Product Features
- 2.3 User Classes and Characteristics
- 2.4 Operating Environment
- 2.5 Design and Implementation Constraints
- 2.6 Assumptions and Dependencies

3 System Features
- 3.1 Authentication Features
- 3.2 Frontend Features
- 3.3 Backend Features
- 3.4 Database Features
- 3.5 Interface Features

4 External Interface Requirements
- 4.1 User Interfaces
- 4.2 Hardware Interfaces
- 4.3 Software Interfaces
- 4.4 Communications Interfaces

5 Other Nonfunctional Requirements
- 5.1 Performance Requirements
- 5.2 Safety Requirements
- 5.3 Security Requirements

6 Key Resource Requirements

Revision History

Name	Date	Reason For Changes	Version
Ryan Arnold	01/22/2025	Initial Commit and upload of project.	0.1.0

Janki Parmar	01/22/2025	First cut at login services.	0.2.0
Ryan Arnold	01/23/2025	First functional CICD pipeline, using Github Actions.	0.3.0
Janki Parmar	01/25/2025	First Frontend demo using Streamlit.	0.4.0
Ryan Arnold	02/28/2025	Added prototype API for FrontendBackendInterface	0.5.0
Janki Parmar	02/01/2025	First Release that has functional Login Service.	0.6.0
Ryan Arnold	02/02/2025	Backend Integration of Kroger Locations API	0.7.0
Ryan Arnold	02/03/2025	Second Version of Frontend Login Service.	0.7.1
Ryan Arnold	02/05/2025	Backend Integration of Kroger Products API	0.8.0
Ryan Arnold	02/09/2025	First cut at integrating MongoDB client	0.9.0
Ryan Arnold	02/15/2025	Fully functional backend integration of Kroger API	1.0.0
Janki Parmar	02/16/2025	Working version of frontend dashboard	1.1.0
Ryan Arnold	03/06/2025	Frontend feature upgrades	1.1.1
Ryan Arnold	03/11/2025	Full integration of all frontend features.	1.2.0
Ryan Arnold	03/12/2025	Full integration/bridge between frontend and backend	1.3.0
Janki Parmar	03/13/2025	Full integration of image object detection pipeline	1.4.0
Ryan Arnold	03/21/2025	backend Kroger integration bug fixes. Add support for multiple image formats. Final CICD release pipeline fixes	1.5.0

1 Introduction

1.1 Purpose

This document outlines the software requirements and specifications of the “EzShopping” app, a Find Fresh Innovations LLC. product. At the time of writing, the latest release of the software is 1.5.0.

1.2 Document Conventions

In this document, every listed requirement is intended to have its own priority. This document will use bulleted lists for requirement and feature lists to improve readability. Requirement numbering

is based on the scheme provided in a requirements spreadsheet to be included in the final set of deliverables. For example, REQ-1 corresponds to *RequirementID*: 1 in the spreadsheet tab "Requirements". Note: requirements may not always be ordered logically, meaning dependent features may not be ordered after the independent feature(s) they depend on.

1.3 References

Kroger Developer Documentation	*https://developer.kroger.com/documentation/partner*
Streamlit Documentation	*https://docs.streamlit.io/*
ngrok Documentation	*https://ngrok.com/docs/*
Docker Documentation	*https://docs.docker.com/*

2 Overall Description

2.1 Product Perspective

Our application is used to automate grocery shopping by allowing users to upload images of common, household grocery products; have a machine learning algorithm identify those objects; then subsequently add the identified products to an authenticated Krogger shopping cart instance on Kroger.com. Our software product is an original design, and self-contained.

2.2 Product Features

- Frontend Features
  - The frontend has a caching mechanism, so that users can run the application from where they left off, without having to restart.
  - The frontend directs users to a "Dashboard" page, which hosts three subpages:
    - upload + add-to-cart tab
    - product history tab
    - tab that externally links to Kroger cart page
  - Image uploader supports the following image formats: *.png, *.jpg, *.tiff
  - The "add to cart" widget interacts with the backend to automatically add a grocery item to an authenticated user's Kroger cart.
  - A widget that enables the user to specify the quantity of items to add to the cart
  - A service to render uploaded images upon upload.
  - History service, so that users can view a history of all their products, and be able to "re-add" to their shopping carts. This combines elements of the frontend, backend, and database linkages.
- Authentication Features
  - The frontend has a login service. This allows users to create accounts or login with existing credentials.
  - Support for Oauth2 authentication with Kroger API.
  - Gateway to redirect traffic from Kroger authentication services.
- Database Features
  - A Mongo Database is used to store user data, and encrypts passwords.
  - Database to store locations response data to enable faster lookups.
  - Database to store product response data to enable faster lookups.
  - Database to store raw image data, as well as classification labels for visualization and to reduce added overhead from object detection pipeline.
  - Database server support using Docker and docker-compose cli.
- Backend Features
  - An Image recognition pipeline to automatically detect and classify grocery items in user-supplied images.
  - Kroger Locations API backend REST integration.
  - Kroger Products API backend REST integration.
  - Filters that request products based on lowest price and promotional prices.
  - Kroger Cart API backend REST integration.

- ➢ Interface Features
    - An interface to bridge the frontend and backend in a way that supports streamlined integration of the two pieces of software while independent development occurs.
    - Common set of Python dataclasses for easier, language-enabled interactions with pymongo clients.
    - backend logging and error-handling service.

2.3 User Classes and Characteristics

- **`FrontendBackendInterface`**: API that abstracts implementation details of common functions:
    - `add_to_cart()`
    - `classify_image()`
    - `get_cheapest_product_nearby()`
    - `get_nearest_kroger`
- **`ProductCard:`** used to represent product data records.
- **`KrogerLocationCard:`** used to represent location data on nearby Kroger locations.
- **`ProductImageRecord:`** used to represent uploaded image data and classification labels.
- **`UserData:`** used to store usernames, passwords, and API tokens.
- **`ProductHistory:`** used to collect uploaded information during a past time window.

2.4 Operating Environment

The software is operating system agnostic, and will work on most modern platforms. The software does require a version of Python 3.10 or greater. The host machine must have virtual machine support to run docker. We assume that developers wanting to deploy the app can install docker, ngrok, and Python-Poetry on their development machines. We depend on Github actions to run CICD pipelines. Pytest is the framework we employ for our unit testing pipeline. Finally, developers must create accounts with Kroger Developers and Ngrok to gain access to API access tokens before deploying the application.

2.5 Design and Implementation Constraints

The most prominent limitation in our software is that it currently supports localhost mode only. Tools we use to deploy the application include: Streamlit frontend library, ngrok for gateway tunneling, docker and docker-compose for database servers, MongoDB as the database schema, bash shell, JSON for configuration files, and the Python3 programming language

2.6 Assumptions and Dependencies

We assume that the Kroger grocery store website is acceptable to most users, and that the users are willing to create a Kroger account that they can use to authenticate with. Third party commercial resources include Ngrok and Kroger Developer API. For the MVP of this product, we assumed that localhost mode is acceptable. Notable Python dependencies include and are not limited to: *Poetry, Pillow, Yolo11, Torch, Streamlit, requests, and pymongo.*

3 System Features

3.1 Authentication Features

3.1.1 Description and Priority

A user of our application must be able to personalize their data, and therefore have their own account. Also, in order to use Kroger API services, client and customer tokens are required, so the goal of this feature is to be able to manage the authentication data per-user and securely. The Priority is **HIGH.** The risk is high, as the authentication mechanisms are complex and proper trafficking services are usually paid services.

3.1.2 Stimulus/Response Sequences

Users of our application must be willing to associate their Kroger account with the application. Users should be able to create a new account with our services, and their passwords should be secured using encryption algorithms. Once users authorize this app with their Kroger accounts, our application should support Kroger's *Oauth2* authentication methodology, which requires traffic redirection and an active gateway.

3.1.3 Functional Requirements

REQ-9: Create a homepage with a login portal and the option to create a new account. If invalid credentials are entered, the user should be warned, and if appropriate, directed to create an account.

REQ-20: The application must properly perform a handshake with Kroger's API via their *Oauth2* authentication mechanism. This includes ingesting the response access token and being able to redirect traffic through an appropriate gateway service.

REQ-21: Customer data must be protected and secured by using a sha256 hash encryption algorithm.

3.2 Frontend Features

3.2.1 Description and Priority

The application frontend is hosted through Python Streamlit, and is the primary interaction point with users and customers. The priority is **HIGH.** The risk is low, as we are confident in our selection of a frontend framework and its ease-of-use.

3.2.2 Stimulus/Response Sequences

The frontend shall feature two primary pages: a Home/login page and a dashboard page. Within the dashboard, there will be three subpages, or streamlit tabs. These subpages include: “upload and add to cart”, “History”, and “My Kroger Cart”. The *upload and add* page should allow the user to upload an image of grocery products using a button and display the image uploaded to the user. Once uploaded, a backend image detection pipeline runs, and displays to the user what the model detected. Once processed, the user should be able to select a button that “adds to cart”, supplemented with a widget to allow the user to set an item quantity. The *history* page should show the user an itemized history of all products they uploaded within a time window. The time window is presented as a radio menu. Once selected, entries showing the image uploaded, classification labels, metadata, and all associated product+location data should be displayed. The user can then “re-add” the record to their cart if they wish. Finally, the “My Kroger Cart” page should direct the user to their authenticated cart instance.

3.2.3 Functional Requirements

REQ-7: Create a fully functional end-to-end frontend that can work in tandem with the backend.

REQ-8: Use the Python *Streamlit* library as the primary frontend framework.

REQ-9: Create a homepage with a login portal and the option to create a new account. If an invalid username or password are entered, the user should be warned and redirected appropriately. The user should, at any point, be able to logout.

REQ-11: Create a page that allows users to “Add to Cart” and a corresponding link to the user’s authenticated cart page.

REQ-24: The frontend should support a caching mechanism such that they can return to the last spot they left off at when they were last logged in.

REQ-25: The user should be able to view a history of their product queries, and be able to re-add items to their cart for quicker results.

REQ-27: The user should be able to upload multiple image formats (*.png, *.jpg, *.tiff), and unable to select unsupported formats from their filesystem.

3.3 Backend Features

3.3.1 Description and Priority

The backend for our application will mainly consist of a machine-learning, image-object detection pipeline, and a library that integrates with Kroger: Locations, Products, and Cart REST APIs. The priority is **HIGH**. The risk is low, since we are confident we will interact with Kroger, and their API is well-documented. The same applies to Ultralytics YOLO v11.

3.3.2 Stimulus/Response Sequences

The image classification pipeline shall take image data as input, and produce object labels based on what it detects in the image. If the labels do not intersect with a label subset, filtered to grocery items, then an empty list is returned. If the image being analyzed already exists in a database, use the previously predicted labels to reduce overhead. For our Kroger backend integration, we must have a mechanism to retrieve client API keys, and customer access tokens. For any REST request, we must first check to see if the requested product data already exists in our databases, to minimize API calls to Kroger.

3.3.3 Functional Requirements

REQ-1: Backend must include a fully functional image object recognition pipeline.

REQ-3: REST-integrated library must be designed to request grocery product data and PUT data through the following Kroger APIs: Locations, Products, Cart.

REQ-4: REST-integrated library must select products based on the lowest products available at the nearest location.

REQ-5: REST-integrated library can filter-out items that are out-of-stock.

REQ-14: Establish a backend logging system to better debug errors.

REQ-17: Products interface can discriminate between market and promotional prices.

3.4 Database Features

3.4.1 Description and Priority

To be able to lookup historical data and access output efficiently, we require our app to have a set of databases. Data storage pertains to: User data, uploaded image and classification data, Kroger locations data, and Kroger products data. The priority is **MEDIUM.** The risk is medium, since all users/developers may not be able to run Docker properly.

3.4.2 Stimulus/Response Sequences

The databases will be designed using MongoDB. We will host separate collections that can be linked to each other. User data, including authentication tokens, should be stored in a user database collection. When a user uploads an image, it gets classified by our backend object detection pipeline. The classification results and the raw image data should be stored in its own database, and linked to the user who uploaded it. Once uploaded, the object should be searched in Kroger's product API, as well as which Kroger is nearest to the user. The corresponding response data should be stored in separate collections, and linked via the unique location ID. Finally The user may request a history of their uploads. Images should be linked to the user who uploaded them; the detected objects should be linked to products via search term; and product data should be linked to Kroger locations via location ID.

3.4.3 Functional Requirements

REQ-2: Store image uploads and classification results to a unique MongoDB collection.

REQ-6: REST response data from Kroger API must be stored in a database to enable caching and limit API calls (and therefore reduces costs)

REQ-22: User data should be securely stored (using encryption) to a MongoDB collection.

REQ-23: Databases should support linking, such that images, products, and the user requesting them can be associated and queried together

3.5 Interface Features

3.5.1 Description and Priority

In order for development of independent features to progress asynchronously, we require interfaces to abstract implementation details, where they are not explicitly needed. This includes our FrontendBackendInterface. We also designed interfaces to store and retrieve data from all our databases in a Python language supported fashion. This makes it simpler to integrate data items to the frontend and backend, as they pertain to our databases. The priority is **LOW.** The risk is low, since the interfaces do not interfere with core business logic.

3.5.2 Stimulus/Response Sequences

The FrontendBackendInterface abstracts high-level dependent features including: adding to cart, classifying an image, storing an image, finding the cheapest product nearby, and finding the nearest Kroger locations. This way, frontend development can continue, using these methods, while backend features and improvements are being developed. Having dataclasses for user data, product data, image data, and history records makes it easier to develop with databases, since these classes abstract database schemas in-language.

3.5.3 Functional Requirements

REQ-10: Create a Frontend-Backend interface to bridge the backend to the frontend to decouple code for more streamlined, asynchronous development of independent features.

REQ-13: Design a common set of dataclasses for result records that can be marshalled to our databases.

REQ-16: Establish a unit testing workflow, particularly to validate items in the test plan and ensure that all features can integrate together in the same environment.

4 External Interface Requirements

4.1 User Interfaces

Our primary interface exposed to the users is our *Streamlit* frontend. The Home/Login Page provides text fields for their username and password, where the password field supports password hidden visibility. A “login” and “Create account” buttons are provided. We also provide buttons that link our Github source code and documentation site. Immediately after the user chooses to login, the user is asked to authenticate their Kroger account with a button. If successful, the Kroger authentication service redirects the user back to the application. Upon successful login, the user is directed to a “Dashboard” page. This page has tabs that function like sub-pages. There is a tab for *Upload and Add to Cart* that has an image uploader widget, a “Add to cart” button, and a quantity ticker widget. The next tab is a *History* tab, that will show upload records within a user-selected time window. The time window options are presented in a radio menu. Once a time window is selected, the frontend will display tiles of all the user’s uploaded images, product data, classification labels, and other relevant metadata. The user is also presented the option to “re-add” to cart, along with quantity widgets, for every tile presented. Finally, the user is presented a *My Cart* tab, that will direct the user to their authenticated Kroger cart instance, via a link button.

4.2 Hardware Interfaces

Currently, our application only supports localhost mode. To deploy database servers, we use Docker containers, which act like virtual machines on the hardware.

4.3 Software Interfaces

Our software will run on most operating systems and platforms that are modern. This includes: Ubuntu Linux, MacOS, and Windows. Our software connects to Mongo Databases using the *pymongo* client library. Data includes user data, product response data, location response data, raw image data, and image-object classification labels. Grocery product data is requested through Kroger Developer endpoints, provided in their public APIs for: *Locations, Products, and Cart.* The databases are served using Docker and the docker-compose cli tool. Our network trafficking gateway is hosted by ngrok, which must be installed by a system package manager. The gateway deployment entrypoint is provided through a bash shell script. Image classification is possible by using the Ultralytics YOLO v11, pre-trained classification model. Finally, we use the *Python Streamlit* module to host our frontend application.

4.4 Communications Interfaces

We employ ngrok to create a network trafficking gateway, which creates network tunnels (binds addresses and ports) on the host machine to a public network trafficking gateway. This facilitates traffic between Kroger authentication services and our hosted application. The gateway abides by HTTP and HTTPS communication standards

5 Other Nonfunctional Requirements

5.1 Performance Requirements

REST requests shall not exceed 60s. An explicit timeout argument is set to enforce this requirement. Image classification should not exceed several seconds. Adding a product to a user's cart shall not exceed several seconds.

5.2 Safety Requirements

N/A.

5.3 Security Requirements

User passwords must be encrypted using sha256 hash encryption algorithms. Kroger account access must be manually authorized by all users before the program collects customer access tokens for further usage.

6 Key Resource Requirements

Major Project Activities	Skill/Expertise Required	Internal Resource	External Resource
Sprint and Meeting Management	Managerial	Ryan Arnold: Part-time	None
Frontend Development	Frontend Development	Ryan Arnold: Part-time Janki Parmar: Part-time	None
Interface Development	Full Stack Experience	Ryan Arnold: Part-time	None
Backend Development	Backend Experience	Ryan Arnold: Part-time Janki Parmar: Part-time	None
Database Development	Mongo Database Experience	Ryan Arnold: Part-time	None
Documentation	Documentation Website Generator Experience	Ryan Arnold: Part-time	None