Module 15 | Project Submission and Documentation Report

CEN3031 Group 12 - Bag Boys

Project Title: *BreadBasket*

Github URL: *https://github.com/JacksonSchreiber/BreadBasket*

Team Members: *Ahmed Eltabbakh, Kyle Miller, Jackson Schreiber, Yaroslav Voryk*

Table of Contents

Section 1 | Project Description

1. Project Description

a) Overview

BreadBasket is a consumer-facing web-based application designed to help families and individuals combat the rising cost of groceries across the United States. Our group pursued a mission of developing a software product that would help families and individuals locate the lowest priced items near them from a common group of goods (a “breadbasket”) based on geographic input in the form of a zip code.

b) Challenge Statement Solution

Our solution to the challenge was to create an application that was easily accessible to everyday shoppers and laymen who do not necessarily know how to run complex software products. By focusing on accessibility and our mission from our overview above, we believe that this consumer application can positively impact society and combat issues related to hunger, poverty, and inflation.

c) Feature Functionality Highlight 1 | Home Page

When users first visit BreadBasket, they will see our clean home page with information about what our app does - comparing grocery prices across multiple stores. If they're not logged in, they'll see a "Get Started" button. For logged-in users, they can enter their ZIP code to start comparing prices.

d) Feature Functionality Highlight 2 | Login/Registration

Users can create accounts or login with: Email and password. After logging in, users have full access to all features. The logins are stored in a backend database using SQLite.

e) Feature Functionality Highlight 3 | Price Comparison

After entering a ZIP code, users can:

- View prices for common grocery items across 5 stores (Kroger, Publix, Aldi, Walmart, WholeFoods)
- Filter items by category or search term
- Sort prices by store
- See which store has the best price for each item (highlighted)
- View unit prices when available
- Expand or collapse category sections
- Add items to cart with a single click

f) Feature Functionality Highlight 4 | Shopping Cart

The shopping cart feature lets users:

- Add items from any store
- Adjust quantities
- See total price by store
- Clear cart or remove individual items

g) Feature Functionality Highlight 5 | Smart Shopping Assistant "Bready"

Our AI assistant "Bready" helps users with:

- Recommending grocery items
- Creating shopping lists based on recipes
- Answering questions about products
- Adding items directly to cart
- Providing meal planning suggestions

h) Feature Functionality Highlight 6 | Account Settings

Users can:

- Update profile information
- Change password
- Set shopping preferences
- View past orders

i) Feature Functionality Highlight 7 | Admin Features

Admins can:

- View and manage user accounts
- Promote/demote other admins
- Review contact form submissions
- Track usage statistics

j) Architectural Pattern

BreadBasket implements the Client-Server with Model-View-Controller (MVC) pattern for its architecture. The client is the user interface and the server is the backend that processes requests, communicates with data sources, and returns results.

k) System Context Model

Figure 1 is a simplified diagram (in text form) that shows how all components interact with each other and with external resources. The arrows indicate the flow of data or requests.

l) Use Case Model

See Figure 2 for our use case model. The “server” actor functions as an abstraction of all the parties that directly or indirectly participate in the API calls made from the front end to the backend (i.e. the grocery stores). This external dependence is a part of our risk assessment and multiple backup programming modes were implemented to prevent lack of access. Of note, as our project progressed our basic client server model evolved into something that resembles a microservice architecture.

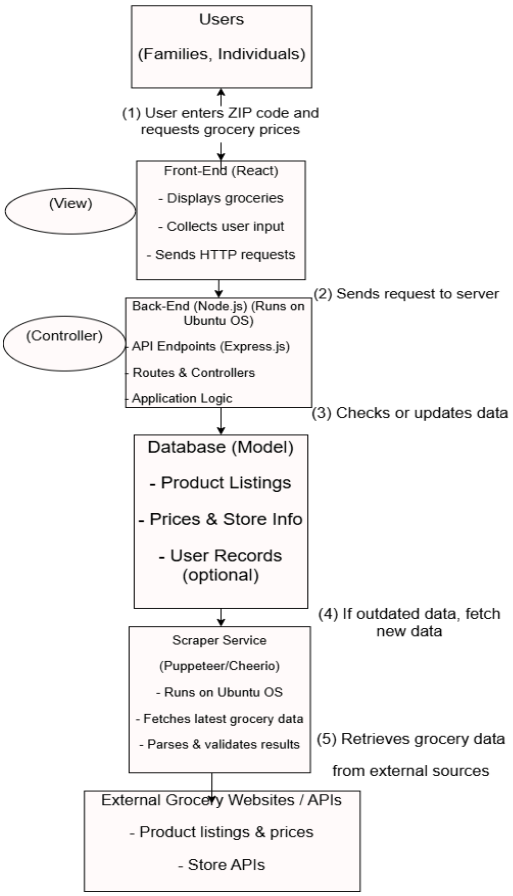

Figure 1 - System Context Model

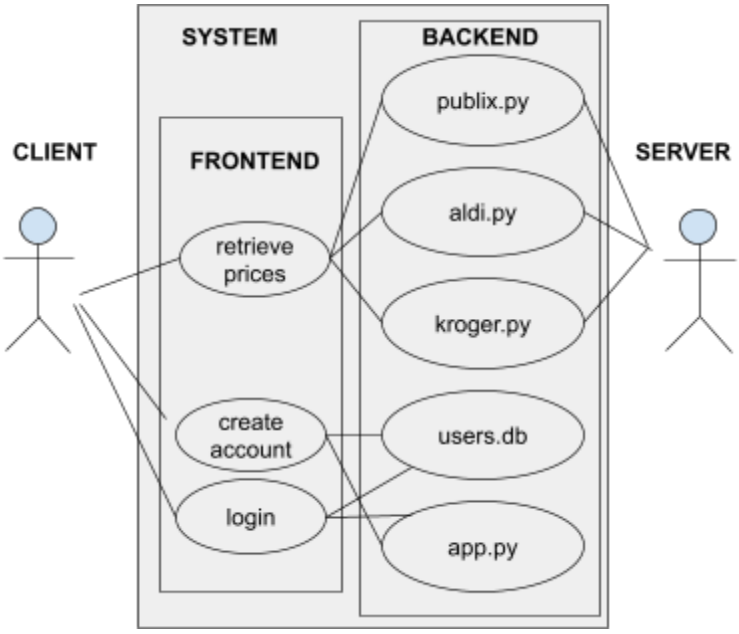

Figure 2 - Use Case Model

Section 2 | Code Management

2. Code Management

a) Code Management

Code Management was performed primarily through GitHub and with application of Agile principles including regular scrums. We had a very active Discord server over the course of this semester-long project and communicated frequently. This communication helped us to continuously deploy new code and get the whole team involved in end-to-end testing.

b) Test Plan

- Front-end
    - Node package manager testing performed on front end to validate REACT
    - Jest Document Object Model for assertions on DOM form
- Back-end
    - In addition to try catch blocks, password hashing through werkzeug.security, server testing with flask library
    - SQLite validation performed by that library and caching inserted as failsafe
- Harness (end-to-end)
    - End to end testing was performed by the whole development team to validate functionality (in future sprints would like to add a better test suite here)
    - dotenv library applied to ensure safety of API keys and limit public exposure

c) Static Code Analysis

As most of our web app is javascript based and the backend fairly straightforward, we used npm audit for static linting and ran flask in a development environment with all warning turned on. Results below:

```
PS Z:\CODE\UF SPRING 2025\CEN3031\BreadBasket\frontend>npm audit
# npm audit report

http-proxy-middleware  <=2.0.8
Severity: moderate
http-proxy-middleware allows fixRequestBody to proceed even if bodyParser has failed - https:/
/github.com/advisories/GHSA-9gqv-wp59-fq42
http-proxy-middleware can call writeBody twice because "else if" is not used - https://github.
com/advisories/GHSA-4www-5p9h-95mh
fix available via `npm audit fix`
node_modules/http-proxy-middleware

nth-check  <2.0.1
Severity: high
Inefficient Regular Expression Complexity in nth-check - https://github.com/advisories/GHSA-rp
65-9cf3-cjxr
fix available via `npm audit fix --force`
Will install react-scripts@3.0.1, which is a breaking change
node_modules/svgo/node_modules/nth-check
  css-select  <=3.1.0
  Depends on vulnerable versions of nth-check
  node_modules/svgo/node_modules/css-select
    svgo  1.0.0 - 1.3.2
    Depends on vulnerable versions of css-select
    node_modules/svgo
      @svgr/plugin-svgo  <=5.5.0
      Depends on vulnerable versions of svgo
      node_modules/@svgr/plugin-svgo
        @svgr/webpack  4.0.0 - 5.5.0
        Depends on vulnerable versions of @svgr/plugin-svgo
        node_modules/@svgr/webpack
          react-scripts  >=2.1.4
          Depends on vulnerable versions of @svgr/webpack
          Depends on vulnerable versions of resolve-url-loader
          node_modules/react-scripts

postcss  <8.4.31
Severity: moderate
PostCSS line return parsing error - https://github.com/advisories/GHSA-7fh5-64p2-3v2j
fix available via `npm audit fix --force`
Will install react-scripts@3.0.1, which is a breaking change
node_modules/resolve-url-loader/node_modules/postcss
  resolve-url-loader  0.0.1-experiment-postcss || 3.0.0-alpha.1 - 4.0.0
  Depends on vulnerable versions of postcss
  node_modules/resolve-url-loader

9 vulnerabilities (3 moderate, 6 high)

To address issues that do not require attention, run:
  npm audit fix
```

Figure 3 - Static Code Analysis Report

Section 3 | Technical Details

3. Technical Details

a) Technical Details 1 | Backend

- Flask-based RESTful API handling user authentication and data processing
- SQLite database for user data, contact submissions, and admin management
- Web scraping implementation (playwright) for store price collection:
- Kroger API integration in kroger.py
- Publix scraper using BeautifulSoup and requests in publix_scraper.py
- Aldi scraper implementation in aldi_scraper.py
- JWT-based authentication with role-based access control

b) Technical Details 2 | Frontend

- Built with React.js for a responsive application
- State management using React hooks (useState, useEffect) and local storage
- React Router for navigation between different views (Home, Results, Contact, Admin, Login)
- CSS for styling with responsive design considerations
- API integration with backend services using fetch API
- AI Assistant (Bready)
    - OpenAI API integration for language processing
    - Custom prompt engineering to create the "Bready" assistant
    - Function calling implementation for cart management
    - Context-aware conversation history tracking

c) Installation Instructions

1. ```git clone https://github.com/JacksonSchreiber/BreadBasket.git```
2. ```cd backend && pip install -r requirements.txt```
3. ```cd ../frontend && npm install```
4. ~/BreadBasket/frontend ```npm run start```
5. *Open up four separate terminals and change directory to ~/BreadBasket/backend*
6. ALL TERMINALS: ```python3 ```

7. List of servers to start (one in each shell, app.py first): ```app.py```, ```aldi_scraper.py```, ```publix_scraper.py```, ```kroger.py```

Make sure to read below on API Keys before starting app

d) Login and Access Credentials / API Keys

- Admin Username: asdfasdfasd, Admin Password: asdfasdfasd
- Create a file title ".env" in both the backend directory and the frontend directory.
- Create or update backend file "api_secrets.py"

```
# OpenAI API Configuration
OPENAI_API_KEY=sk-proj-vEUfrh-ieYNcUdcNEbO0IDwIYIXXah9tSAusV0CSzbvn3W3OX
UsayJglp-YP6LWHAamgh0WfPoT3BlbkFJyHknwsVuV9H0axw341j0DAUpOsRYeguZx1U_cVm
OQZqN_TNdDcdiKee5Cb_1_LGi9mSDQ5vskA

FLASK_ENV=development
FLASK_APP=app.py
FLASK_DEBUG=1

# Database Configuration
DATABASE_URL=sqlite:///shopping_assistant.db
```

BreadBasket/backend/.env

```
REACT_APP_API_URL=http://127.0.0.1:5000/
REACT_APP_OPENAI_API_KEY=sk-proj-vEUfrh-ieYNcUdcNEbO0IDwIYIXXah9tSAusV0C
Szbvn3W3OXUsayJglp-YP6LWHAamgh0WfPoT3BlbkFJyHknwsVuV9H0axw341j0DAUpOsRYe
guZx1U_cVmOQZqN_TNdDcdiKee5Cb_1_LGi9mSDQ5vskA
```

BreadBasket/frontend/.env

```
# Kroger API Creds:
CLIENT_ID =
"breadbasket-243261243034246a79762e693372424b5a70524e2f5771706d454854756
4385547777445674f2e3847396f39366a34764d6c533555614a77492e564f36158428656
38524082"
CLIENT_SECRET = "dXN8ZnQJ02vsck7HuLVKOk6D06ZDMvzLTxlX6W2Y"
```

BreadBasket/backend/api_secrets.py

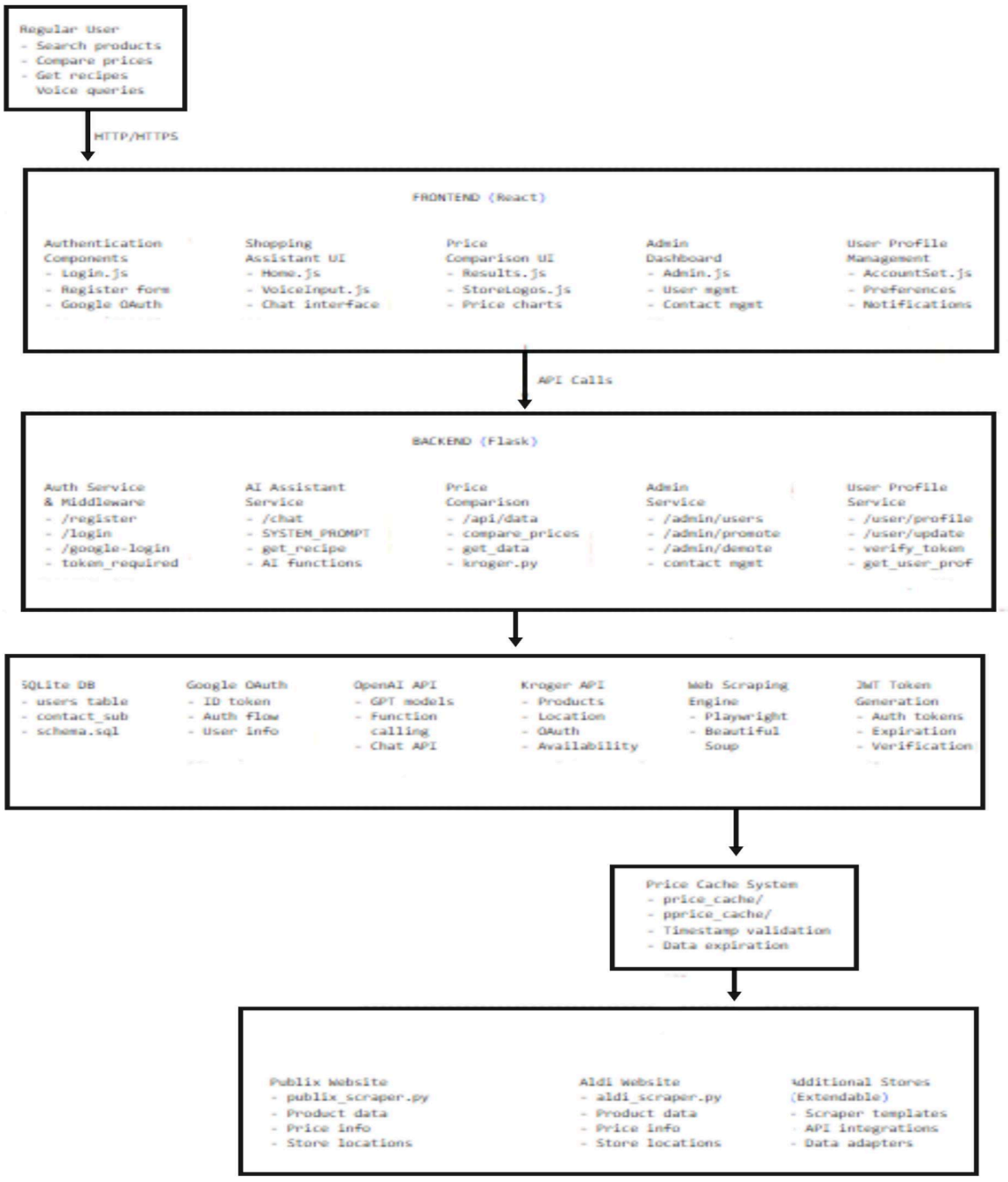

Figure 4 - End-to-End Technical Overview

Section 4 | Risk Management

4. Risk Management and Software Quality Attributes

a) Risk Management

Our risk management plan was two fold - focusing on end user misuse or malintent and server side actors (ie the grocery stores) potentially changing end points or API keys. Our full risk management plan is appended at the end of our report.

b) Software Quality Attributes

- Usability
    - Our frontend was designed to be very user friendly and easy to navigate. Given that the servers are up and running, front end usability is great.
- Functionality
    - Our application is fairly straightforward and pulls prices as requested.
- Performance
    - Initially a problem with some scraping methods, the inclusion of a caching library has resulted in our prices being generated quickly.
- Reliability
    - Our reliability and error / exception recovery are handled well in the frontend, however we could improve in the backend (server crashing)
- Efficiency
    - While caching helps, more optimization is needed for full deployment.
- Flexibility
    - The frontend is flexible in that there are no “dead-end” routes for the user.
- Security
    - Our environment, api_secrets, and user hashing guarantees security.
- Interoperability
    - On desktop the app is easy to deploy, mobile requires more work.
- Testability
    - Frontend and Backend are good, end-to-end could improve.
- Availability
    - Publicly available on GitHub now!

References & Additional Docs

4. References & Additional Docs

a) References

- https://www.ibm.com/docs/en/rational-soft-arch/9.6.1?topic=diagrams-use-case
  - IBM Use Case Model and Diagrams
- https://platform.openai.com/docs
  - OpenAI. *OpenAI API Documentation*.
- https://www.reactjs.org/docs/getting-started.html
  - ReactJS. *React Documentation*.
- https://www.flask.palletsprojects.com/
  - Flask. *Flask Documentation*
- https://www.crummy.com/software/BeautifulSoup/bs4/doc/
  - BeautifulSoup. *Web Scraping Documentation*.
- https://playwright.dev/
  - Playwright. *Browser Automation Framework*.
- https://www.sqlite.org/index.html
  - SQLite. *SQLite Home Page*.
- https://pypi.org/project/python-dotenv/
  - dotenv. *Managing Environment Variables in Python*.
- https://www.cleanpng.com/

  Logos

b) Additional Docs (Risk Management Plan)

Risk	**Probability**	**Impact**	**Mitigation Strategy**
Website Blocking Scrapers	High	Catastrophic	Use rotating IPs, user-agent switching, and prioritize APIs
API Limitations	Medium	Serious	Implement caching and use rate- limiting

			strategies
Data Accuracy Issues	High	Serious	Validate extracted data with multiple sources
Performance Issues	Medium	Serious	Optimize scraper and use asynchronous
Compliance & Legal Issues	Medium	Catastrophic	Check terms of Service and seek store permissions
Requirement Changes	Medium	Serious	Maintain clear project scope and update changes

Website Blocking Scrapers	High	Catastrophic	Use rotating IPs, user-agent switching, and prioritize APIs
API Limitations	Medium	Serious	Implement caching and use rate- limiting strategies
Data Accuracy Issues	High	Serious	Validate extracted data with multiple sources
Performance Issues	Medium	Serious	Optimize scraper and use asynchronous
Compliance & Legal Issues	Medium	Catastrophic	Check terms of Service and seek store permissions
Requirement Changes	Medium	Serious	Maintain clear project scope and update