SmartKart System Design

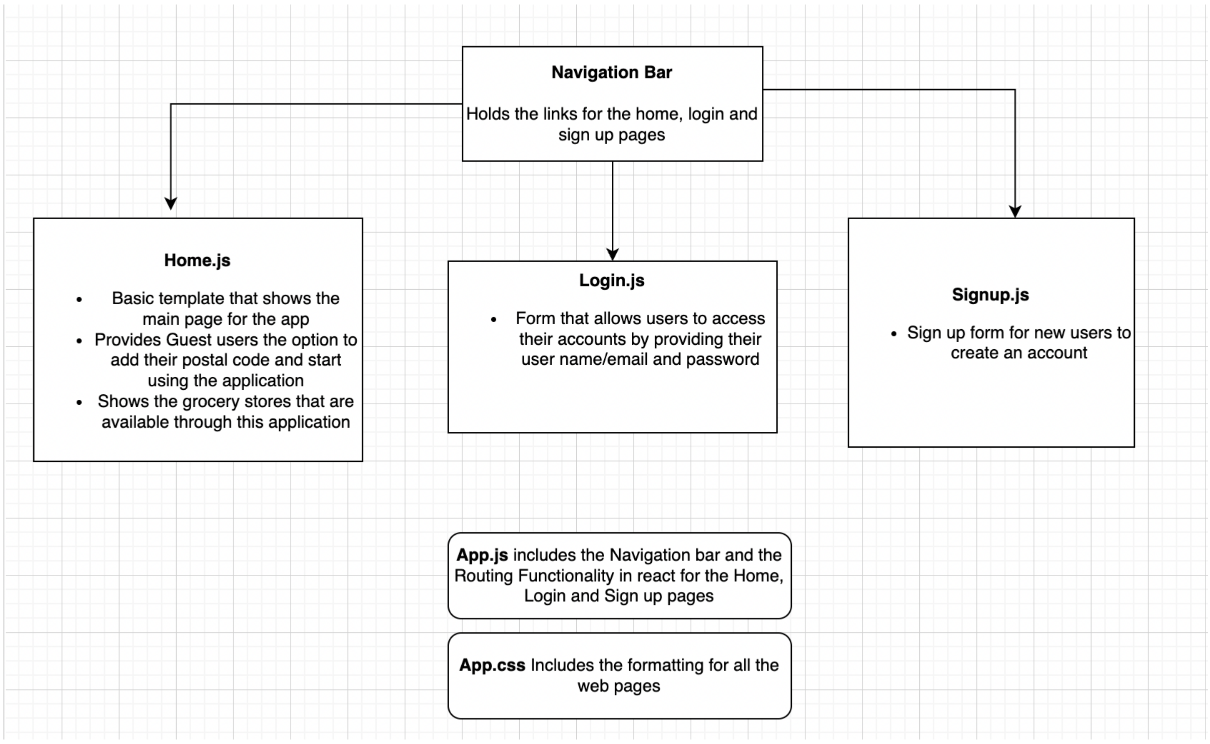
Activision

Contents

Architecture Description

https://www.ibm.com/cloud/learn/three-tier-architecture
Our architecture follows the design of the Three-tiered architecture. We have separated our design into the Presentation tier, Application tier, and Data tier. Our Presentation tier consists of our front end and uses JavaScript, HTML, and CSS which displays the SmartKart web app. Our Application tier contains our backend and business logic (scrappers, database manipulation, etc) and uses Python, Flash, and SQLAlchemy. Finally, our Data tier contains our PostgreSQL databases, namely the Item database and User databases, which is how we can manipulate item and user data and utilize them for the function of the app.

Front End Diagram



Navigation Bar
Holds the links for the home, login and sign up pages
Home.js
Basic template that shows the main page for the app
Provides Guest users the option to add their postal code and start using the application
Shows the grocery stores that are available through this application
Login.js
Form that allows users to access their accounts by providing their user name/email and password
Signup.js
Sign up form for new users to create an account
App.js includes the Navigation bar and the Routing Functionality in react for the Home, Login and Sign up pages
App.css Includes the formatting for all the web pages

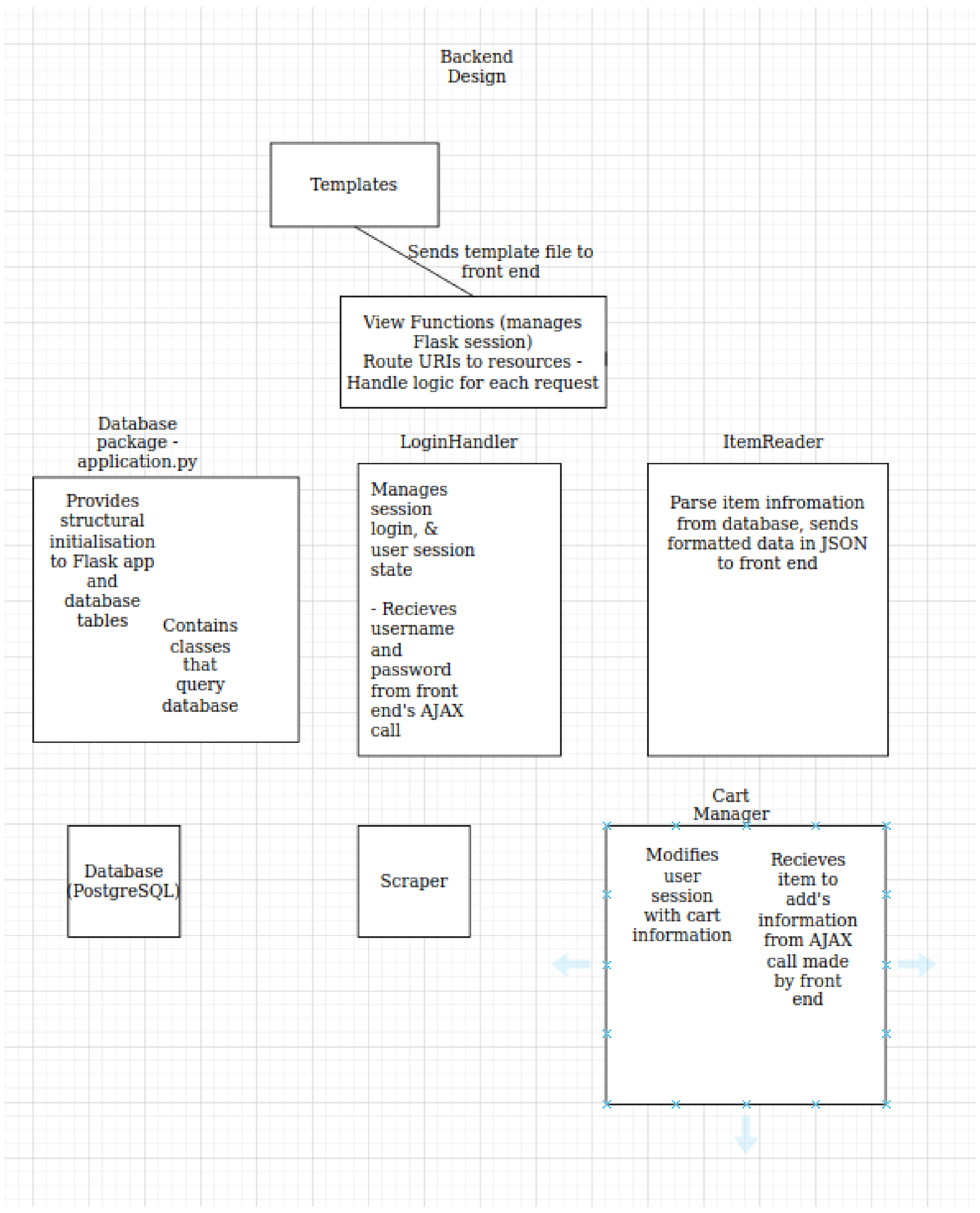
Start Shopping (Items.js)

- Includes the Flyers, Category Options, Items matched after search and Items frequently bought together with your searched database

Checkout (Pickup.js)

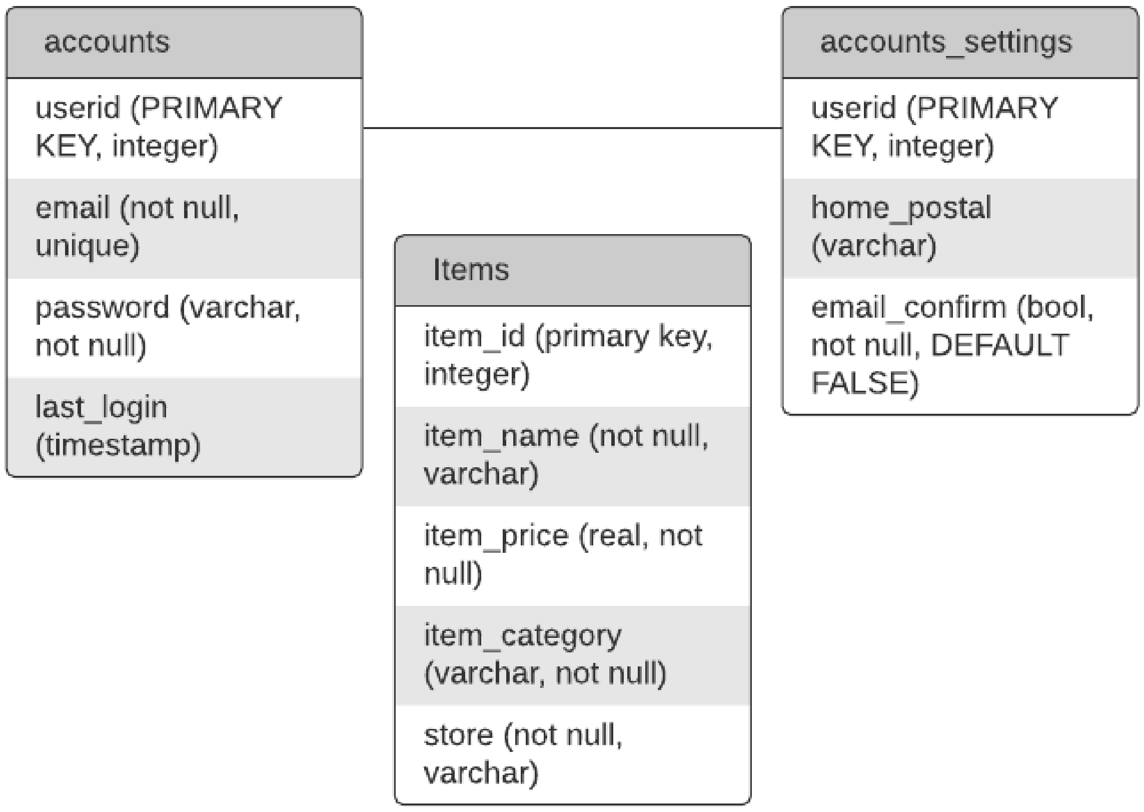
- Includes buttons for user to select their method of picking up the groceries.

Backend Diagram



Backend Design
Templates
Sends template file to front end
View Functions (manages Flask session) Route URIs to resources - Handle logic for each request
Database package - application.py
Provides structural initialisation to Flask app and database tables
Contains classes that query database
LoginHandler
Manages session login, & user session state
- Recieves username and password from front end's AJAX call
ItemReader
Parse item infromation from database, sends formatted data in JSON to front end
Cart Manager
Modifies user session with cart information
Recieves item to add's information from AJAX call made by front end
Database (PostgreSQL)
Scraper

Schema Design


accounts
userid (PRIMARY KEY, integer)
email (not null, unique)
password (varchar, not null)
last_login (timestamp)
accounts_settings
userid (PRIMARY KEY, integer)
home_postal (varchar)
email_confirm (bool, not null, DEFAULT FALSE)
Items
item_id (primary key, integer)
item_name (not null, varchar)
item_price (real, not null)
item_category (varchar, not null)
store (not null, varchar)

Accounts Table

The accounts table will be used to store basic account login info.

Userid: unique identifier that will be generated per user. Will be used to connect *accounts* table to other user-related tables in the database (such as accounts_settings)
Email: email of registered user, used to login to webapp. The email must be unique per user, which will prevent duplicate sign-ups with the same email
Password: password for logging into webapp
Last_login: timestamp of last user login

Accounts Settings Table

The accounts_settings table stores various settings and details of user accounts. The reasoning for separating the accounts_settings is to allow modification and addition of various user settings without any interaction with basic account info for logging in.

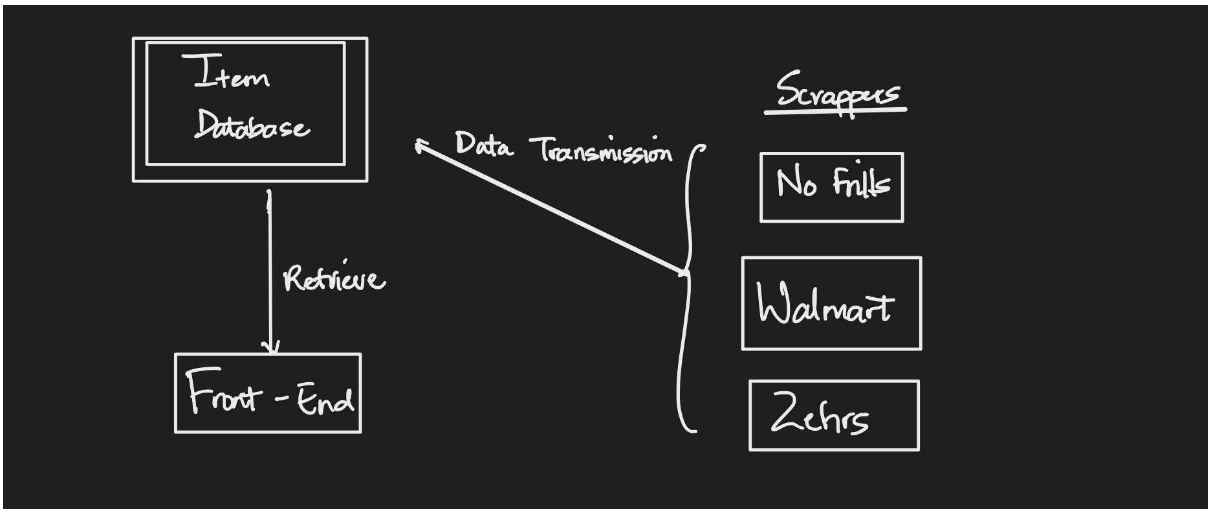
Userid: unique identifier that will be generated per user. Will be used to connect *accounts* table to other user-related tables in the database (such as *accounts*)
Home_postal: Postal code of user.
Email_confirm: True of False value indicating whether the user has confirmed their email or not.

Item Table:

The item table is designed to store information regarding a certain grocery item.
The fields of the table include an id, name, price, category and the store that is selling it at the price stored. So, there will be multiple entries of say, milk, with information specific to different grocery stores that sell milk. This design allows for easy comparison among prices from the various stores, retrieval of items by category, filtering by prices or stores etc
.
item_id: unique identifier for various grocery items. This field is the primary key of the table, and will therefore be by definition unique and not null
Item_name: name of grocery item
item_price: price of item in CAD
Item_category: category of item (e.g. fruits and vegetables, meats, etc.)
Store: name of store that contains this item

Scraper Return Value

Scraper will take category and subcategory of item as argument, and return a list of (item_name, item_price, item_unit_price). This is subject to change as scraper develops, and functionality to download thumbnails from scraped websites is added.



<u>CRC Cards</u>

Scraper.py

Class Name: GetNoFrills

Responsibility	Collaborators
Scrape items from NoFrills	None

Class Name: SendNoFrills

Responsibility	Collaborators
Send results from NoFrills scraper to database using pscyopg2	GetNoFrills

Class Name: GetLoblaws

Responsibility	Collaborators
Scrape items from Loblaws	None

Class Name: SendLoblaws

Responsibility	Collaborators
Send results from Loblaws scraper to database using pscyopg2	GetLoblaws