Boundary Breakers - System Design Document

CSCC01 - Fall 2021

Members: Chang Liu, Haowen Rui, Hayden Mak Long Hei, Hrithik Kumar Advani, Johnson Su, Nimra Maqbool, Raymond Ho

Table of Contents

CRC Cards	2-5
System Interaction Description	6
System Architecture	7
System Decomposition	8

CRC Cards

Database table schema for ‘company’:

id	integer
username	varchar(20)
password	varchar(80)
email	varchar(80)
manager_name	varchar(80)
store_name	varchar(80)
store_location	varchar(80)
logo	varchar(100)
map_of_store	varchar(100)

Database table schema for ‘companyProfile’:

cid	integer
description	varchar(512)
avg_review	integer(0<=value<=5)
open_time	varchar(12)
close_time	varchar(12)
contact_phone	integer
contact_email	varchar(80)
website	varchar(80)

Welcome_Screen		
Responsibilities: Enable users to login as a Company or a User		
Parent class: none	Sub class: none	Collaborations: main

login_screen		
Responsibilities: Enables users to login using the Username and Password. If the company doesn't have an account, it redirects to the company signup page. After successful login, it will redirect to the homepage.		
Parent class: none	Sub class: none	Collaborations: welcome_screen company_signup_screen NavBar

main		
Responsibilities: Run the front-end app and initiate the users to the welcome page		
Parent class: none	Sub class: none	Collaborations: Welcome_Screen

NavBar		
Responsibilities: Build the bottom navigation bar that shows the users the 4 primary functions in this app.		
Parent class: none	Sub class: none	Collaborations: StoreScreen, ShoppingListScreen, MapScreen, UserScreen

StoreScreen		
Responsibilities: Demonstrate to the users all available stores and a short list of items in that store		
Parent class: none	Sub class: none	Collaborations: none

ShoppingListScreen		
Responsibilities: Show the user the goods he or she has added to the shopping list Display corresponding store that sells the goods		
Parent class: none	Sub class: none	Collaborations: none

MapScreen		
Responsibilities: Show the user the shortest way to get all his goods either by path, price, or store		
Parent class: none	Sub class: none	Collaborations: none

UserScreen		
Responsibilities: Display the user profile, settings, and preferences when using the app		
Parent class: none	Sub class: none	Collaborations: ProfilePage

ProfilePage		
Responsibilities: Display specifics of the user profile Contains a logout button that leads to the logout page		
Parent class: none	Sub class: none	Collaborations: CompanyLogout

CompanyLogout		
Responsibilities: Displays a logout screen and lead the user back to the initial welcome screen Provide a link to welcome page		
Parent class: none	Sub class: none	Collaborations: Welcome_Screen

company_signup_screen		
Responsibilities: Displays a sign up screen Ensures that all fields are valid Lead the user to the login screen after they successfully registered		
Parent class: none	Sub class: none	Collaborations: login_screen

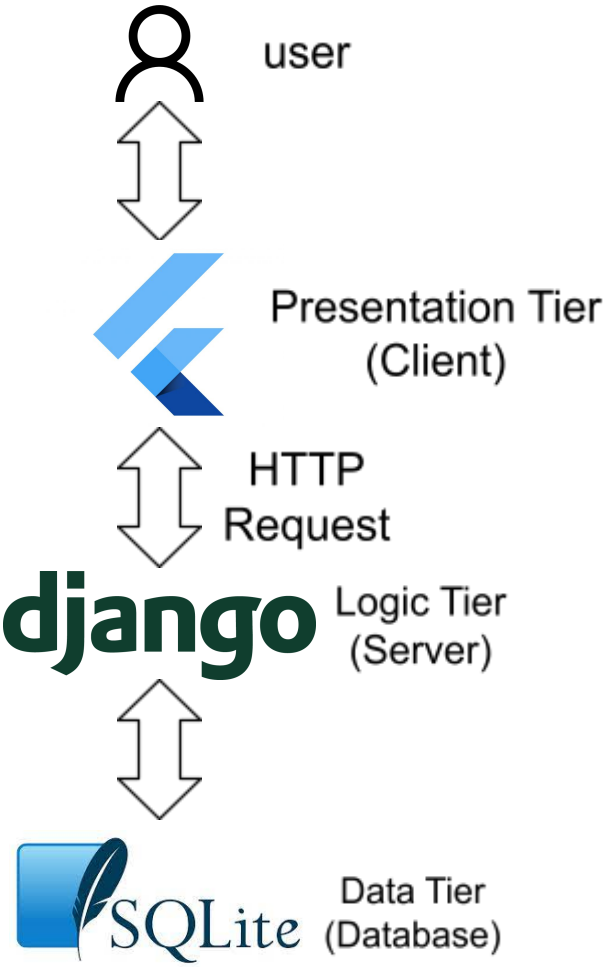
System Interaction Description

The user must have a working browser or mobile device with internet access. Through the device, the user is able to open the flutter app and can choose whether they are the company or customer. Once they have chosen their role, they can register and sign in. As the company, they will be able to add items that their store sells as well as their store layout to the app. Every company will have a profile where the store location, store contact and store manager information can be added or modified . As the customer, they will be able to add items to the shopping list and find out which stores contain the items on their shopping list, as well as a route to get all the items from the store.

To run the application locally, our program assumes the user has installed Flutter and has also set up a device with Flutter. The program also assumes the user has installed Python, Java, Django and Pillow. After installing required software, clone the Git repo to your device and navigate to the code folder. For frontend, cd into smart_grocery_map and run “flutter run -d chrome --web-port=3001 .\lib\main.dart”. This starts the front end. For Django, cd into backend and run “python manage.py runserver”, which starts Django. The frontend can be accessed at “http://localhost:3001/” while the admin panel of Django can be accessed at “http://localhost:8000/admin”.

System Architecture

The system architecture we are using is the three tier architecture (https://www.ibm.com/in-en/cloud/learn/three-tier-architecture). Our presentation tier uses Flutter and communicates through http requests to the Django server which is our logic tier. Django connects to the SQLite database automatically through the settings.py file and accesses our data tier, an SQLite database, and then the server returns a response back to the client.



reference to this architecture:

https://www.ibm.com/in-en/cloud/learn/three-tier-architecture

System Decomposition

The user will access the app through their device that is connected to Flutter. The app will send api requests to the Django server which include creating an account, logging in and viewing user profiles. Django receives the requests through the routing endpoints in place and will handle the request through the views.py file. The views.py file then uses the models in models.py to query the SQLite database which stores all the app's information.

If the user provides invalid input and attempts to make a request to the server, the server will respond with an appropriate error code. The server performs checks to make sure any invalid data does not get stored in the database. An example of this is if the user created a new account with missing fields which are required or using a username that already exists, the server will identify this and not store the data into the database while also returning an error code. The app also does some checks on user input, like making sure all fields are filled in the sign up form before sending a request to the server.