# CSE 546 — You shop, we save!

Group no. 21
*Sahil Santosh Patil*
*Prabhudev Terakanambi Rajendra*
*Kasturi Chandraprakash Adep*

## 1. Introduction

People are always looking to save money and time without compromising on their needs while shopping, but the prices of products vary across stores making it difficult to do a thorough analysis before making the purchase. To tackle this issue in general and grocery shopping in particular, we present a website that allows users to compare prices of grocery items along with shopping lists. The idea is to find the best value for the list of items from the nearby supermarkets so that the user can either go in person to the store or order it online.

Aim: To build a web application for shopping list comparison using GCP PaaS services which tackles some major flaws in state-of-art price comparing solutions and adds new features to it.

## 2. Background

We all love to shop online, be it clothes, gadgets, home decor etc, but none of us wants the hassle of going through every website, gather the data, consolidate it, compare it to find the cheapest price for all the items that we want. To be truthful, sometimes the savings that we make might not even be worth all that effort that we put. But, what if we have an application or a website that does all this for us. All we have to do is give our current location, select the distance within which we want the shops to be, select the list of items from shops of our preferences and we will get the best deals available arranged in the increasing order of price, and then? we start saving!

Wondering where this is useful?

Any form of eCommerce websites or local stores with item information available online. The approach that we have used in our application can be enhanced to get unified information about any item from any shop of any category.

Current solutions:

There might already be applications whose intentions are somewhat on the similar lines. Some examples of such applications are ‘Basket’ mobile app and ‘MyGroceryDeals.com’. Basket app is useful for grocery comparison by items. It is available on Android and iOS as of now. MyGroceryDeals is a website which helps users to find the best deals/offers available in stores. We found that there is a lot of potential in developing such applications as it benefits a lot of users daily. However, after doing some research about these applications and looking at the user feedback, we realized that these applications can be improved in certain areas. Moreover, we thought of implementing some additional features that might prove useful. Few of these new features were designed based on user feedback.

The current solutions have the following drawbacks:

- They provide comparison for a single item at a time
- They don't provide comparison data based user's entire shopping list
- Users cannot limit the search radius of stores around their location
- Some of the solutions are available only on mobile platforms

## 3. Design and Implementation

### A. Application Architecture and Flow:

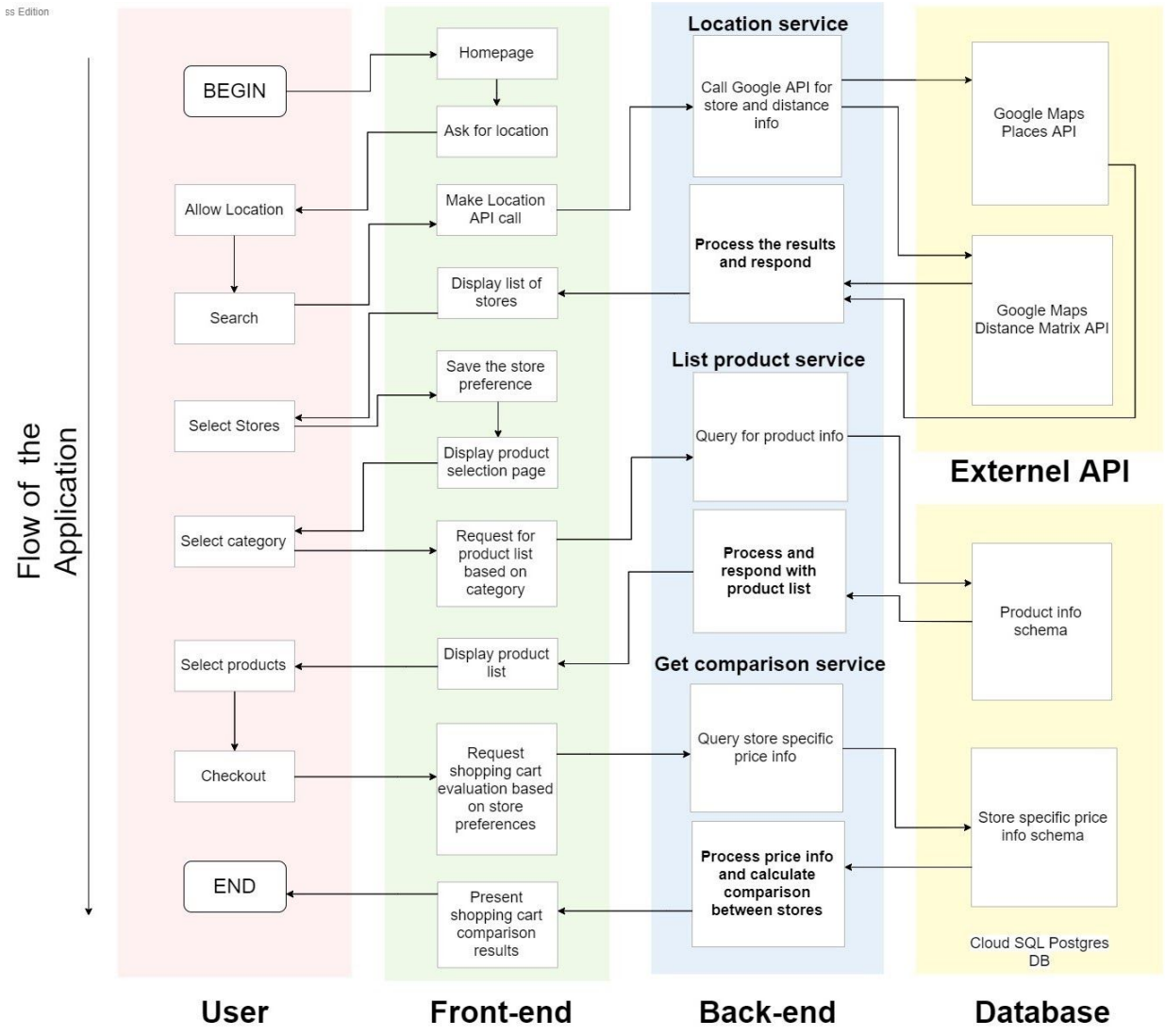


Figure 1 : Architecture and flow

Major Components:

- Front-end App:

  On the front-end we use a React app to build the functionality. The design is inspired from Walmart. com. The journey is divided into 3 steps.

  Step 1 - User allows location to be detected, edits maximum radius for search area and clicks on search. This uses the Location Service API explained in the following section. This API returns a list of stores and their details (name, address, distance from user's location and ETA) in the response. All options are displayed to the user. User then selects stores they would like to use in price comparison.

  Step 2 - On the next step, a cart is displayed that is empty by default. A left-hand menu presents all product categories. The user may select any option from the list. On selection of an option, the Product Details API is called which takes category and selected stores as parameters of the request. The response contains a list of specific products for that category and are displayed to the user. They may select the quantity of the product as required and repeat this step for adding their entire shopping list to the cart.

  Step 3- This is the checkout step and displays the final result of which store offers the cheapest option for the user's shopping list. The cart details from the previous step are sent as a request to the Checkout API. The API calculates the prices based on user selected quantity of products and returns all details to be displayed on the output screen. All items and their prices are displayed on the final screen for each store user selects in step 1. The cheapest store is listed first and is highlighted with the total cost.

- Back-end App:

  The Back-end is developed using the Django REST framework in Python 3.8. It is deployed on a separate Google app engine than the front-end. The objective behind this is to make the application components loosely coupled. This ensures individual scalability.

  In this, three major services (REST APIs) are hosted.

  i. Location Service: It gets location from front-end and uses Google Maps Places API to get nearby stores within user-specified range. After that it will use the Google Maps Matrix API to calculate distance and ETA from the user's location.
  ii. Item information service: It takes the item category and the list of selected stores and returns all the items in that particular category which are available in the selected stores along with each item's information.
  iii. Comparison Service: It takes the user's shopping list, performs comparison based on the total cost and returns the comparison data.

- Database:

  We are using Postgres 11 as the database in this application. This database is used to store information about different products in different supermarkets. The back-end services will query this database for generating results. For high availability and low latency we have used Google Cloud SQL service to host our database.

- External APIs:

  Google Places API and Google Distance Matrix API are used for location service in the back-end. The Places API provides nearby places based on the given location and range. The Distance Matrix API provides distance and ETA between two locations.

**B. Cloud services used:**

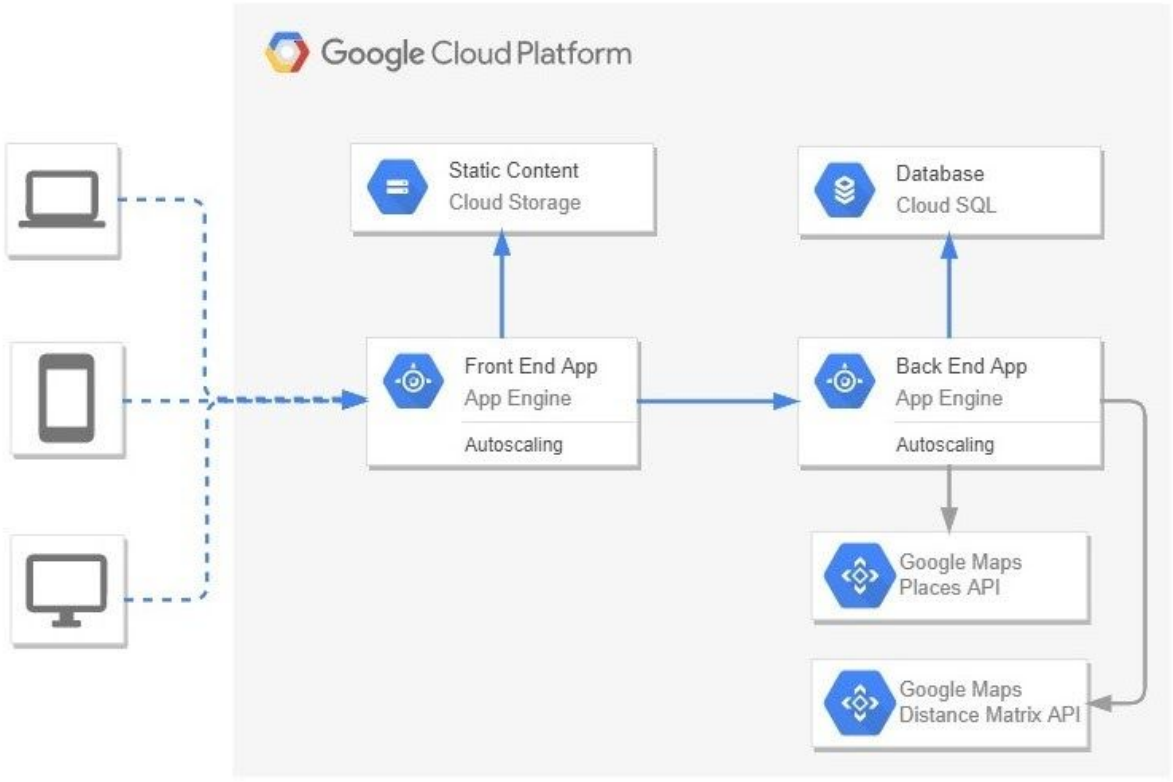


Figure 2 : Cloud Services Used

- Google App Engine:
  GAE is used to deploy both front-end and back-end applications separately. The front-end app engine makes REST calls to the back-end application hosted on another GAE

- Google Cloud SQL:
  Cloud SQL is used to host the application's postgres 11 database. The GAE hosting the back-end communicates with Cloud SQL for querying the data.

- Google Cloud Storage:
  Cloud Storage is used to store static data/files of front-end application. Also, it is used to store app related files of the application deployed in the back-end.

## C. Role of Google App Engine:

The front-end app is a React web app deployed on the Google App Engine. The back-end app is a python django web application which is deployed on a separate GAE.
The reasons to use GAE for deploying applications in this project are -

- Hosting applications is very easy.
- No deployment configurations required except specifying the runtime.
- Node.js is supported and readily available for use.
- No infrastructure setup required.
- Autoscaling is managed internally by GAE.
- Systematic application error logging mechanism.
- GAE provides application usage statistics along with visualization of certain metrics.
- Pay as per usage: You have to pay as per the usage of resources as the application grows.

As our app is now deployed on GAE, users with access to the internet can use it from anywhere in the world.

## D. Autoscaling:

Autoscaling is the ability of an application to scale in/out automatically depending on the application load so as to serve all the requests seamlessly.

The definition of load may vary from application to application. With respect to this application, load can be defined as the number of requests the website is getting at a particular point of time. Also, depending on the user input the time to serve each request may vary as the input varies. So, the nature of the input parameters can also be considered in deciding that a request is lightweight/heavy.

Considering the above two parameters, whenever the load on the website increases/decreases, GAE manages to scale out/in the resources so as to meet the new requirements as per current load. This kind of autoscaling is provided by GAE out of the box. Therefore the application developer doesn't have to deal with implementing autoscaling for the applications deployed on GAE. This is one of the most important reasons for choosing GAE.

## E. How does this application solve the problem? How is it different from others?

With the support from GAE PaaS services our web application helps the users to gather the data from the shops of their choice, consolidates it, compares it to find the cheapest price for all the items that the user wants, all that in a matter of a few clicks.

Our application also supports some features which the state-of-the-art applications don't. like, selecting the radius within which the user wants his/her shop to be in, price comparison for the entire shopping list, ability to compare as many shops as they want side by side, provides a database support so that we don't have to scrape the data from the websites every time a new user uses the web application.

## 4. Testing and evaluation

We have tested the application against different constraints for all options available to the user.

The filtering of stores based on distance with boundary conditions.

Exploratory analysis was done for combinations of product selections and shopping lists.

Apart from regular application testing, we performed load testing to evaluate the autoscaling response of our application. Following are the details of these tests.

As a part of stress testing to witness autoscaling we used Apache Bench, a tool for benchmarking hppt web servers.

We ran the below command.

**ab -n 100000  -c 1000 <sample request url for the website>**

This runs 100000 requests to perform the benchmarking session and does 1000 requests concurrently. The results of the tests are shown below

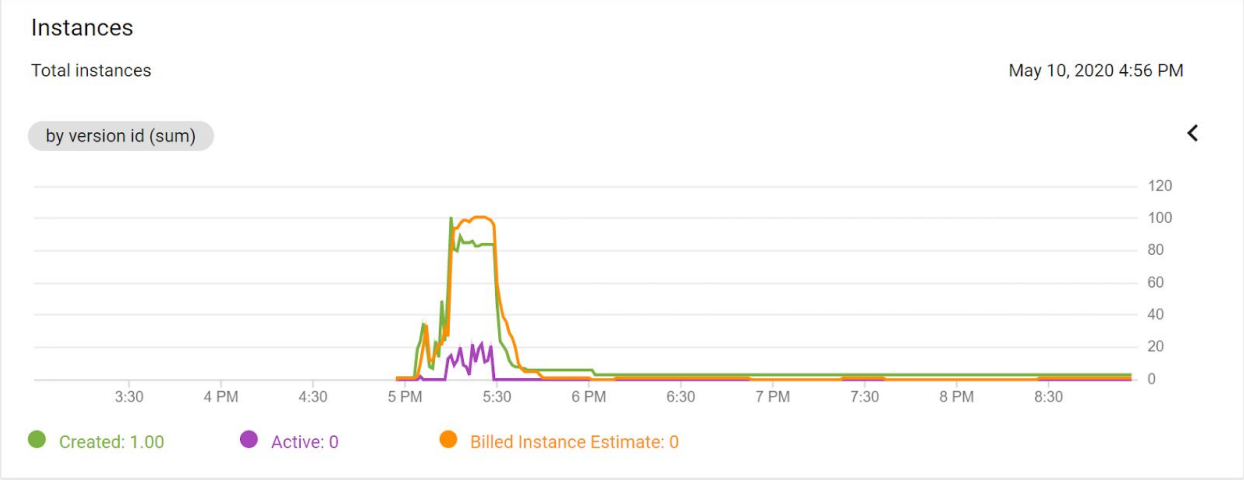


Figure 3 : Instance scaling during stress test

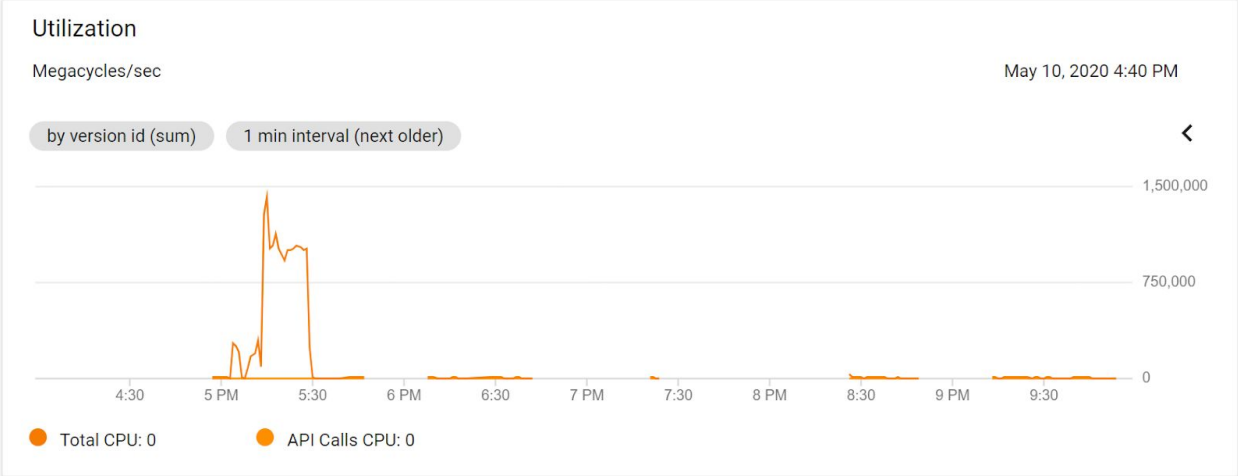


Figure 4 : Utilization during stress test

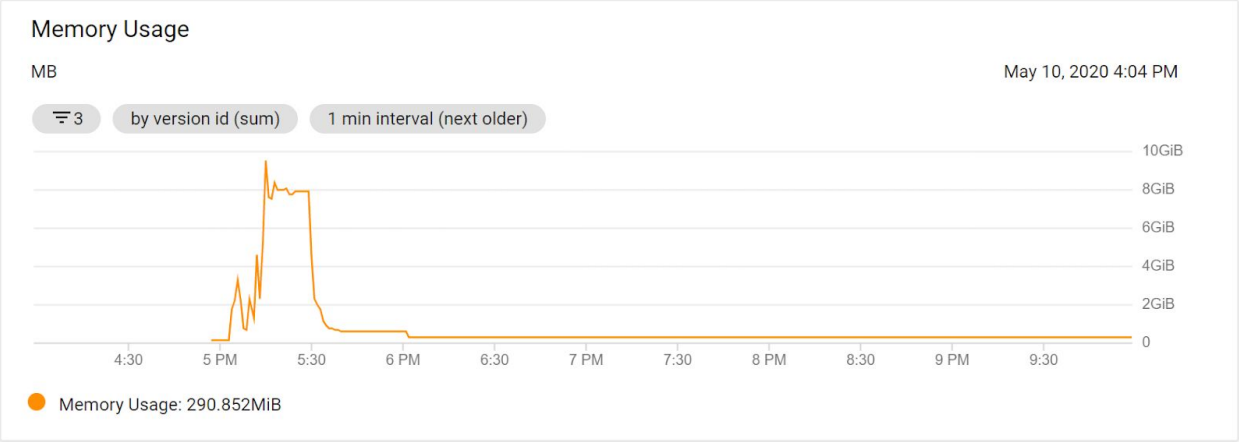


Figure 5 : Memory usage during stress test

## 5. Code

1. React web app - front-end (price_comparator)

   This folder contains all the code for the frontend app. Start by extracting the zip file.

   Requirements - Node.js (latest version), npm (latest version)

   Running the code:

   Open command prompt/terminal and change directory to price_comparator.

   To run locally -

   *npm install*

   *npm start*

   To deploy on GAE -

   1. Setup gcloud command in your system using this official guide.
   2. Open cmd/terminal in price_comparator folder.
   3. Run *npm run build* to create a production build of the application.
   4. Run *gcloud app deploy* to push the code to the project selected while setting up Cloud SDK.

   Currently this app is deployed and can be accessed at -

   https://poetic-result-276707.uc.r.appspot.com/

   Project structure -

   - package.json - Contains all the dependencies and commands required to run this project.
   - build - Contains compiled production build of the app with all the JavaScript combined.
   - yaml.app - Required to notify GAE of the type of runtime (nodeJS) required to run this app.

    - Additionally this file contains information about the url to be loaded. GAE will use the production build folder created in steps mentioned above.
- src/

  /components - Contains all the React Components used across the web app.

  /constants - Lists all the Action Types

  /containers - Contains containers used in conjunction with the components.

  /reducers - All the reducers to access data from Redux Store

2. Back-end Python project:

   Requirements- Python 3.8, Google cloud SDK for python. Required python libraries are listed in Requirements.txt which is also used by GAE to install the libraries while running the application

   Running the backend application on GAE:

   - Extract the zip
   - Navigate to 'restapi' directory which is the root python django project
   - This directory contains the app.yaml file which contains necessary information needed by GAE to set up the application.
   - Using the google cloud SDK shell, login to your google cloud account. Select the google cloud project within which you want to deploy the application.
   - Now in project root directory execute the following command to deploy the application:

     *gcloud app deploy*

   - Assuming that you've already deployed the front-end application, the website will be ready now. You can access it using the GAE project's URL.

   Description of important files in the back-end application:

   - app.yaml - This file is needed by GAE to run the app. It contains the runtime specifications.
   - appengine_config.py - this file tells GAE to include libraries from the /lib directory (if any)
   - main.py - this is the main entrypoint to the application. This is the file that GAE will look for when deployed.
   - /restapi/urls.py - all the supported url patterns are mapped to the respective functions
   - /restapi/settings.py - this file contains django specific configurations for the application
   - /Location/views.py - this the file where all the REST API methods are actually written

## 6. Conclusions

To summarize, in this assignment we have developed an end-to-end application that compares price of products across different supermarket chains. And as a result, recommends the cheapest store to shop from based on the user's shopping list.

Additional features that are provided as part of this project are - users can enter the maximum radius of the search area to get stores. And the prices are compared using the items and quantity selected and not the brand.

The application we have built successfully proves the concept proposed as part of this project. However, there are features that can be added to improve the sophistication of the idea, such as-

1. Options to create user accounts to save history.
2. Filters and search options to locate products.
3. Feature to redirect users to store websites after checkout.
4. Print and email shopping lists.

## 7. References


[1] Walmart website structure - https://www.walmart.com/

[2] Cloud SDK Guide - https://cloud.google.com/sdk/

[3] Medium: Deploying a React App to Google Cloud Platform using App Engine - https://medium.com/better-programming/deploy-a-react-app-to-google-cloud-platform-using-google-app-engine-3f74fbd537ec

[4] Medium: Deploying React App to Google App Engine - https://medium.com/tech-tajawal/deploying-react-app-to-google-app-engine-a6ea0d5af132

[5] Basket.com - http://basket.com/

[6] Google Play reviews for Basket.com - https://play.google.com/store/apps/details?id=com.basketsavings.basket&hl=en

[7] Mygrocerydeals.com - https://mygrocerydeals.com/