Cloud Computing and Big Data Project - Car Finder

Amrit Parimi (ap4142) Atul Balaji (ab5246) Saravanan Thanu (st3523) Vikram Kumar (vk2503)

***Abstract*— In this project, we developed a car finder and recommendation system using AWS services. This application aims to help potential car buyers find their desired car based on various preferences they enter such as make, model and price range, and also provides recommendations based on the cars they have liked. The recommendations are provided using the K-Nearest Neighbors (KNN) algorithm, by using various features of the cars they have previously liked.**

Video Link: `https://youtu.be/KeXr3tD9wG8`
Our presentation can be found here.
The code for the application can be found here

I. INTRODUCTION

Selecting the right car to purchase is not an easy decision with the enormous number of choices available and a multitude of features to be considered. Through this project, we intend to make this process easier by creating an application to give recommendations for users based on the cars they previously expressed interest in. Users will also have the option to search for cars based on multiple filters on features like make, model, price and style. The key functionalities of this application are described below.

A. Key functionalities

- **Search** - Users can search for cars by selecting one of more fields out of make, model, price, vehicle style and transmission type.
- **Car Description** - Users have the ability to view the image and detailed description of any car.
- **Like** - When looking at the description page of a car, users can like the car, which will make the car appear in their wishlist.
- **Recommendations** - In the home page of the application, we display cars recommended to the user, based on the cars they have liked previously.

B. AWS services

- Cognito
- CloudFront
- S3
- API Gateway
- Lambda
- DynamoDB
- OpenSearch
- CodePipeline

C. Dataset

For making this application, we used the Car Features and MSRP dataset from Kaggle, which was obtained by aggregating data from car websites such as Edmunds. This dataset contains various details about many different car models, such as make, model, price, style, transmission, fuel type, horsepower, number of cylinders, mileage and popularity. It contains a total of 928 distinct car models, which forms the basis of our application.

The car images were scraped from Wikipedia by using the given car make and model.

II. OVERALL ARCHITECTURE

Figure 1 shows the overall architecture of our system. The frontend code for the application is hosted using CloudFront, into which the code is deployed through AWS CodePipeline from Github. We use AWS Cognito to handle user login and manage user identities.

The various endpoints and functionalities of the application are handled via API Gateway. It connects with Lambda functions which are responsible for providing search results, enabling the like function, provide recommendations, rendering car description, etc. In order to perform these functions, they interact with DynamoDB, OpenSearch and S3 in order to store and retrieve data.

The various data stores being used are described below.

A. DynamoDB

- *cars-table-unique*: This table stores the details of all the different cars and has carID as the primary key. It also has a column called *knn* which contains the list of carIDs of similar cars for a given car, which were determined using the K-Nearest Neighbors algorithm, for the purpose of recommendations.

carID	city mpg	Driven_Wheels	Engine Cylinders	Engine Fuel Type	Engine HP	highway M...	knn	Make	Market Category	Model
C10045	24	front wheel drive	4.0	regular unleaded	160.0	30	[C12418','...	Acura	Hatchback,Luxury,...	RSX
C13169	18	all wheel drive	6.0	premium unleaded (...	305.0	26	[C13258','...	Acura	Luxury,Performance	TL
C13225	18	front wheel drive	5.0	regular unleaded	176.0	24	[C12841','...	Acura	Luxury	Vigor
C13200	12	rear wheel drive	12.0	premium unleaded (...	565.0	18	[C13227','...	Aston Martin	Exotic,High-Perfor...	V12 Vantag...
C13227	13	rear wheel drive	12.0	premium unleaded (...	490.0	18	[C13199','...	Aston Martin	Exotic,High-Perfor...	Virage
C12615	13	rear wheel drive	12.0	premium unleaded (...	540.0	19	[C13227','...	Aston Martin	Exotic,High-Perfor...	DB9 GT
C13199	11	rear wheel drive	12.0	premium unleaded (...	460.0	18	[C13227','...	Aston Martin	Exotic,High-Perfor...	V12 Vanquish
C12614	11	rear wheel drive	12.0	premium unleaded (...	420.0	18	[C13199','...	Aston Martin	Exotic,Factory Tun...	DB7
C12970	20	front wheel drive	4.0	premium unleaded (...	200.0	29	[C12902','...	Audi	Crossover,Luxury	Q3
C13062	18	all wheel drive	6.0	premium unleaded (...	333.0	26	[C13194','...	Audi	Factory Tuner,Luxu...	S5
C13194	22	all wheel drive	4.0	premium unleaded (...	265.0	31	[C13062','...	Audi	Factory Tuner,Luxu...	TT S
C12451	20	all wheel drive	4.0	premium unleaded (...	220.0	27	[C12429','...	Audi	Luxury	allroad
C13061	18	all wheel drive	6.0	premium unleaded (...	333.0	28	[C13067','...	Audi	Factory Tuner,Luxu...	S4
C12430	22	all wheel drive	4.0	premium unleaded (...	220.0	31	[C12429','...	Audi	Luxury	A5
C12431	24	all wheel drive	6.0	diesel	240.0	38	[C12433','...	Audi	Diesel,Luxury	A6
C13146	12	all wheel drive	12.0	flex-fuel (premium ...	631.0	19	[C12567','...	Bentley	Exotic,Flex Fuel,Fac...	Supersport...

Fig. 2. DyanmoDB cars table

- *likes-db*: This table stores the cars liked by all users. It uses userID as the partition key and carID as the sort key which together uniquely determine a row. This table is used to get populate the wishlist for a user.

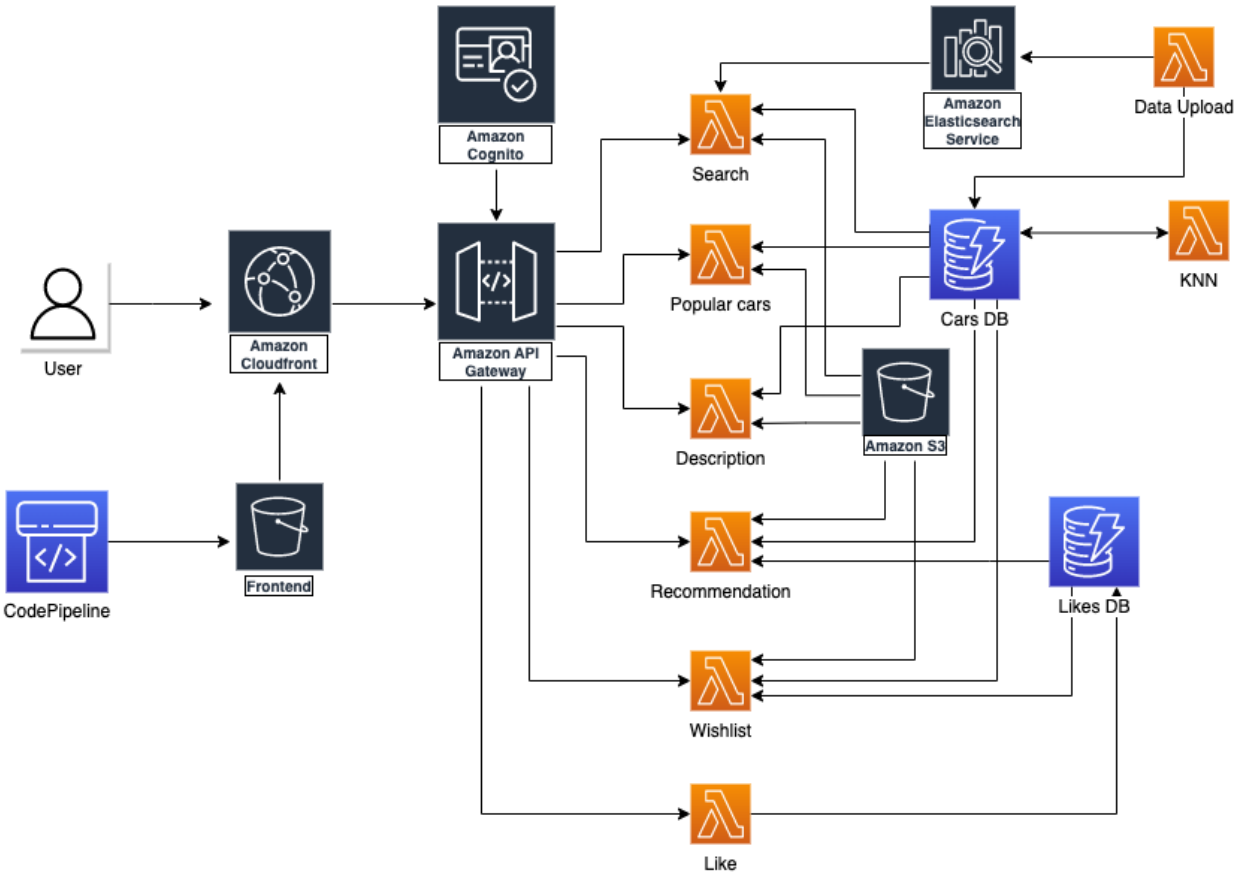

Fig. 1. Overall Architecture of the system

B. Elasticsearch

- *cars-3*: This domain stores the following fields:
  - carID
  - Make
  - Model
  - MSRP
  - Vehicle Style
  - Transmission Type

  The field *MSRP* is an integer and is used to perform range based search for price, while the other fields are strings.

C. S3

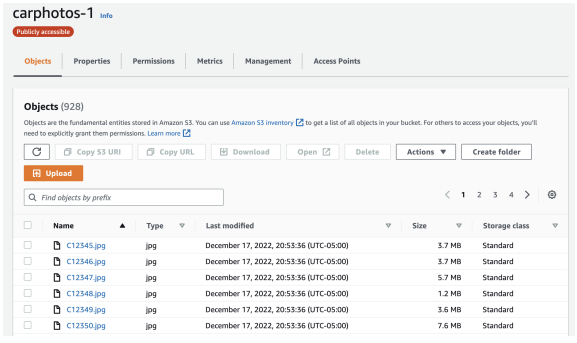

Fig. 3. S3 bucket of car images

- *carphotos-1*: This bucket stores the car photos for all unique car models, with the name of each photo being the carID, so that the right image can be retrieved easily.
- *project-frontend-car*: This bucket stores the frontend code, which is updated via CodePipeline from a GitHub repository.

III. KEY DESIGN DETAILS

This section describes the detailed working of different functionalities and the design components that were used in their implementation.

A. Search

Users can search for cars using the dropdown fields in the home page of the application. Here they can select one or more fields from make, model, price range, vehicle style and transmission type. Once they make the selections, these details are passed using API Gateway to the *GET /search* API which takes in the input as a query string containing the fields that are being searched for along with their selected values. This API triggers the *cars-search* Lambda function.

This Lambda function parses the query string and performs a search on the Elasticsearch domain using the given query. For each match obtained, it then looks up the DynamoDB table of car details and returns the list of results to the API. The search results thus obtained are rendered in the frontend.

B. Car Description

When the user clicks on a particular car, they are directed to the car description page for the corresponding carID. This request is served using the API *GET /description* which takes as input a query string of the carID. This API then triggers the Lambda function *car-desc* which takes the carID from the query string and looks it up in the DynamoDB table. It returns all the attributes for the item found, which is then rendered in the frontend.

C. Like

When the user views a particular car, they have the option to click on the like button in the description page. On clicking this button, the API *GET /like* is called, along with the carID as the query string. This triggers the *put-like* Lambda function. This functions takes in the given carID as input and parses the userID from the current session and adds the userID and carID to the *likes-db* table.

D. Wishlist

When the user clicks on the wishlist button, to see the cars they have liked, the API *GET /wishlist* is called, which triggers the *wishlistPopulate* Lambda function. This function gets the userID from the current session and looks it up in the *likes-db* table. It then looks up the carIDs found in this process in the cars table and returns the results to the API, which is used to populate the wishlist in the frontend.

E. Popular Cars

The list of popular cars are loaded using the *GET /popular* API which triggers the *popular-cars* Lambda function. This function retrieves the most popular cars from each make based on the popularity column in our dataset.

F. Recommendation

The recommendations are loaded using the *GET /recommendation* API which triggers the *recommendation-LambdaFunction*.

IV. GENERATING RECOMMENDATIONS

```
class KNeighborsClassifier:
    def __init__(self, k=5, dist_metric=dist):
        self.k = k
        self.dist_metric = dist_metric
    def fit(self, X_train, y_train):
        self.X_train = X_train
        self.y_train = y_train
    def predict(self, X_test):
        neighbors = []
        for i,x in enumerate(X_test):
            distances = self.dist_metric(x, self.X_train)
            dy = zip(distances, self.y_train)
            y_sorted = [y for _, y in sorted(dy, reverse=True)]
            y_sorted.remove(self.y_train[i])
            neighbors.append(y_sorted[:self.k])
        return neighbors
```

Fig. 4. KNN implementation

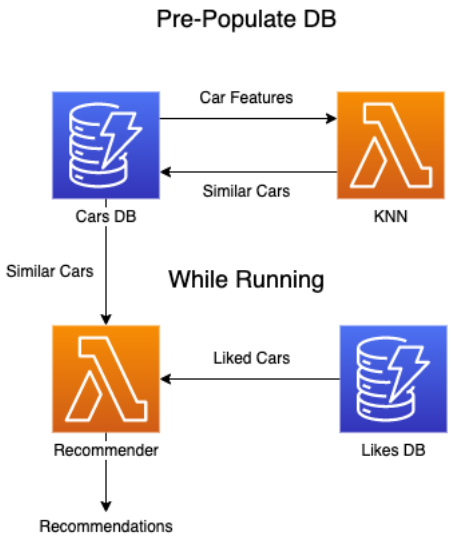

Fig. 5. Recommnedation Design

At the beginning, the *KNN* Lambda Function collects the car details of all the cars from the Cars Database in DynamoDB. This data is then used to find the k-nearest neighbors using the algorithm shown in Fig.4. The algorithm uses a similarity metric to find the similarity between any two cars based on a total of 11 attributes including numerical as well as categorical fields. The categorical fields used are market Category, Driven Wheels, Model, Number of Doors, Vehicle Style, Make, Vehicle Size, Transmission Type, and the numerical fields used are highway MPG, Engine HP, Number of Doors.

For each car, the function calculates the similarity and determines the top 5 similar cars which are then added to the Cars Database as a new column. When the user logs in, the Recommendation API is called, which triggers the *Recommender* Lambda Function. This function retrieves the cars liked by the user from the *likes-db* table and then collects the similar cars by querying the new column added to the Cars Database. The function then randomly chooses a subset of the similar cars and sends the details as recommendations to the user.

V. CODE STRUCTURE

This section describes the structure and the functioning of the code used to implement both the frontend and backend components of our application.

A. Frontend code

The frontend code contains the following html pages:

- *login.html*
- *index.html*
- *search.html*
- *car.html*
- *wishlist.html*

The following javascript files are used:

- *main.js*
- *dropdown.js*

B. Backend code

The following Lambda functions are being used to implement the various backend functionalities:

- *data-upload*: This lambda function reads the dataset of car details from a CSV file and writes the data to the DynamoDB table *cars-table-unique* and the Elasticsearch domain *cars-3*.
- *popular-cars*: This Lambda Function queries the cars table and returns the most popular cars based on the popularity column of the cars table.
- *cars-desc*: This Lambda function takes the carID as input and looks up the car details in the *cars-table-unique* table and returns the details for the given car.
- *cars-search*: This Lambda function takes a query string with given fields and values as input and returns the details of cars that match the criteria using Elasticsearch and DynamoDB.
- *put-like*: This Lambda function takes the userID and the carID as input and adds an entry to the likes-db if the user likes the car and removes the entry from likes-db if the user dislikes the car.
- *wishlistPopulate*: This Lambda function queries the likes-db and returns all the cars that are liked by the user.
- *KNN-LambdaFunction*: This Lambda function determines the similar cars for each car in the database and stores it as a new column in the cars table.
- *recommendation-LambdaFunction*: This Lambda function returns the list of recommendations based on the cars liked by the user using KNN.

VI. RESULTS

In this section, we illustrate the working of different functionalities of the application and show the results in the frontend for the various APIs discussed above.

A. Search

When the user clicks on the search button, the search query is passed to the search API which triggers the *cars-search* lambda function, which returns the cars that match the given criteria. The search result page is rendered as shown below.

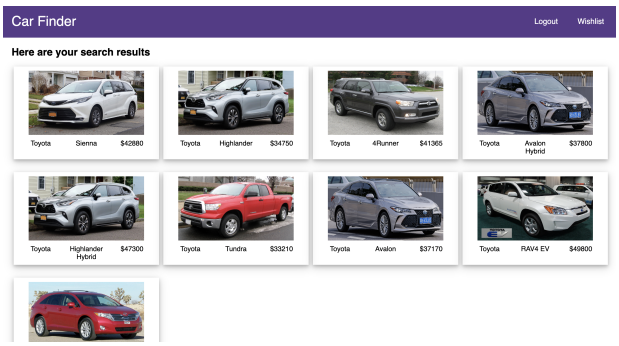

Fig. 6. Search results page

B. Wishlist

When the user clicks on the Wishlist link, the Wishlist API returns the cars liked by the user and the page is rendered as shown below.

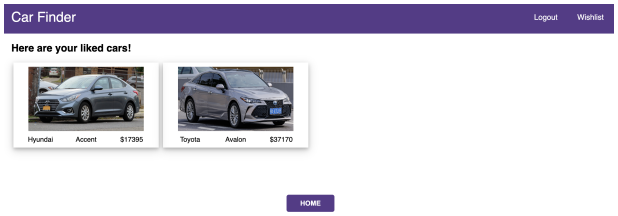

Fig. 7. Wishlist page

C. Car Description

When the user clicks on a particular car, the car details are displayed and the user has the option to like or unlike the car. The car description page is rendered as shown below.

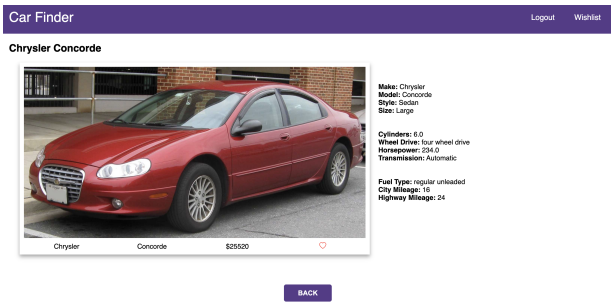

Fig. 8. Car description page

D. Recommendations

When the user logs in and has liked some cars, the recommendations are displayed on the home page which is rendered as shown below.

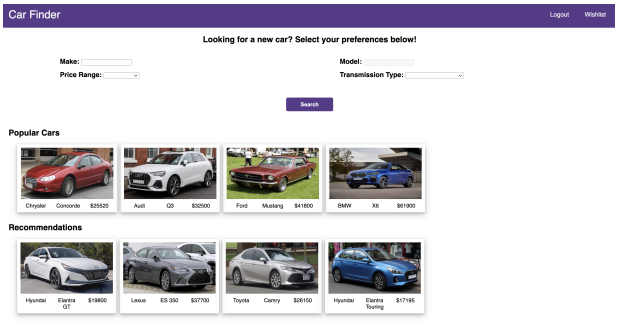

Fig. 9. Recommendations in Home Page

VII. CONCLUSION

In this project we built a car finder and recommendation application using AWS services such as Cognito, CloudFront, DynamoDB, OpenSearch, S3, API Gateway, Lambda, and CodePipeline. The working of our application was based on an event driven and scalable design. These backend components are supported by the frontend which was designed in a user friendly manner.

The application contains various important features that provide the user with the functionalities necessary to find cars they want to buy. The search functionality, implemented using OpenSearch enables the user to quickly narrow down their options according to their preferences. The like functionality and wishlists provide users with a way to save cars they like for future reference. Our recommendation system provides users with tailored options based on their history of likes, which makes it easier for them to find cars they are more likely to buy.

VIII. LINKS

Video Link: `https://youtu.be/KeXr3tD9wG8`
Our presentation can be found `https://docs.google.com/presentation/d/1W8qMwPMu4VEaASX73vc3v7jnyM3xQuqt7qWHJ8OD0DQ/edit?usp=sharing`.
Github Link: `https://github.com/saranthn/Car-Recommendation.git`