Auto GovSurplus

Search and track automobile auction lots across several listings

Assignment 3

ECE1779 Introduction to Cloud Computing

Team Members

Denis Noskov • 1007140666
Sheran Cardoza • 1001070066
Amarpreet Singh • 1002513764

Date of Submission

April 21st, 2021

1. Project Description

The last decade saw a large growth in used car marketplace as the supply of used vehicles continues to increase coupled with the effect of vehicles having a longer life line on the road. According to Statista Research Department, the retail sales of used car dealers in Canada amounted to over 12 billion CAD in 2019 and have have doubled since 2012[1]. The largest contributor to the industry however has been the growing presence of online marketplaces, with platforms like eBay and Autotrader providing not only a large flow of vehicle listings, but also enabling transparency in pricing and an opportunity for consumers to purchase a used vehicle directly at cost instead of paying additional fees to dealers.

In spite of the rising trends in the industry, the main sources of procurement of used vehicles remain pre-owned car dealerships and online marketplaces. Our application aims to change that by providing its users with a whole new channel of purchasing used vehicles by delivering the vehicles sold at government auctions in a user-friendly manner. Two of the largest government auction platforms in Canada at the moment are GCSurplus and Govdeals. GCSurplus is owned and operated by the Canadian Government and serves as a platform for government agencies and offices to auction off their surplus inventory, which includes service vehicles. Govdeals is owned by Liquidity Services, a company that operates multiple auction websites, though this one is intended only for public institutions such as universities. Because both platforms list equipment in an auction format, the inventory listed is not very structured and the main audience for these services remain resellers or users that are knowledgeable of the auctioning process. Auto GovSurplus intends to bring these services closer to all users by providing a product that allows them to easily browse their vehicles of choice by distinct categories like make and model and adding additional functionalities that will enable them to keep track of different listings.

2. Webapp User Flow

2.1. User Access

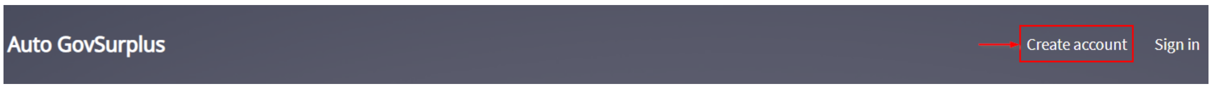
The webapp can be accessed via the following URL:
https://c3qau7i0c9.execute-api.us-east-1.amazonaws.com/dev/

- The URL takes users directly to the home page, which lists all the available auction listings

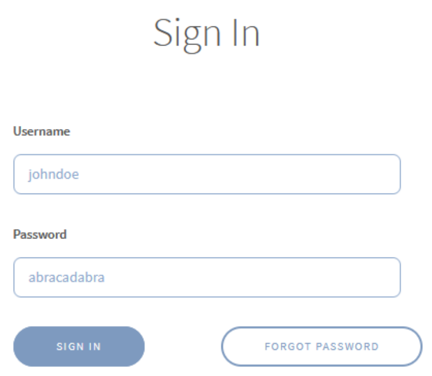
[1] https://www.statista.com/statistics/431937/retail-sales-of-used-car-dealers-in-canada/

User authentication:

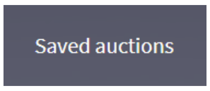
- Users can create new accounts directly from the webapp without requiring admin intervention.



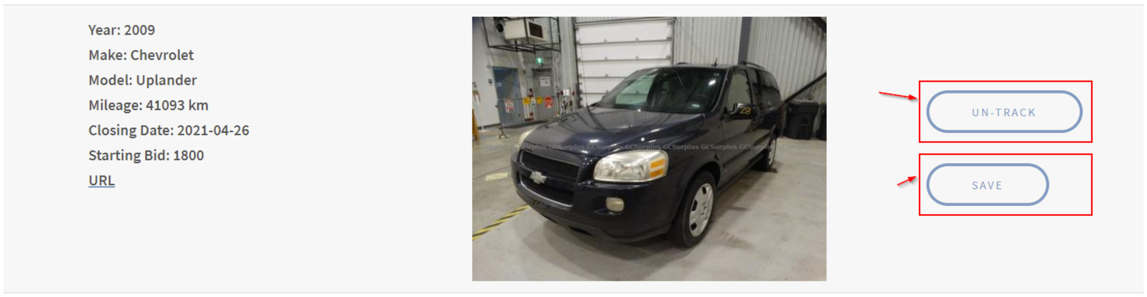
- The user can log in by clicking on the "Sign In" link in the navigation tab



- Upon signing in, two buttons become visible next to each listing.
    - The save button allows the user to add a listing to its saved list, which will make this listing visible in the "Saved auctions" tab in the top navigation bar. To remove a listing from the list, user needs to click the same button which should now read "un-save".

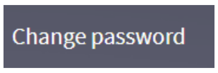


    - The track button allows the user to receive email notifications about that listing every 24 hours until the lot is sold, with the up-to-date highest bid. Once the item is sold, the user will automatically stop receiving notifications, The other option is to click the "un-track" button.



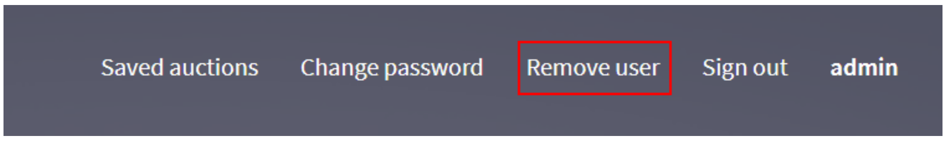
Year: 2009
Make: Chevrolet
Model: Uplander
Mileage: 41093 km
Closing Date: 2021-04-26
Starting Bid: 1800
URL

- When a user is signed in, they also have the ability to change their password by clicking on the "Change password" button in the navigation tab



Administrator functionalities:

- Admin login credentials:
    - Username: *admin*
    - Password: *12345*
- The Admin has the additional ability to remove existing users via “Remove user” button in the navigation tab:



Backend access: in order to access the back-end functionalities, you can do so via the following AWS student account

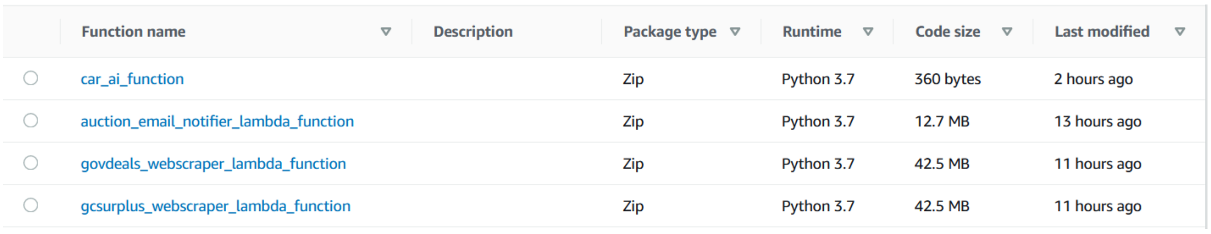
- Username: denis.noskov@mail.utoronto.ca
- Password: EYF5*%QNu7

2.2. Features

Backend:

- Web Scraping: the list of latest car listings is updated every 24 hrs via a script that parses auction listings and is invoked via a lambda function. See section 1.3.1 for details.
- Car AI model: An AI model is used to extract insights such as vehicle type and color from auction listings, which is then presented as filters to the user. This provides more organic filtering options whereby a user can, for instance, search for “silver pickup trucks”. See 1.3.2 for details.
- Email notification: whenever a user adds a listing to its tracked list, they begin to receive email notifications every 24 hours with an update regarding the item’s latest highest bid, until that item is sold and is marked as not active in the application. See 1.3.3 for details.

All 4 functions associated with these backend processes can be accessed via the AWS account provided above, by going to the Lambda menu. Given that the AWS student account does not provide the ability to schedule recurring events, the Web-scraping and Car AI functions must be invoked manually, which can be done by pressing the “Test” button. The assumption followed is that in the production environment, the scheduling function would be available, allowing to run these processes every 24 hours.

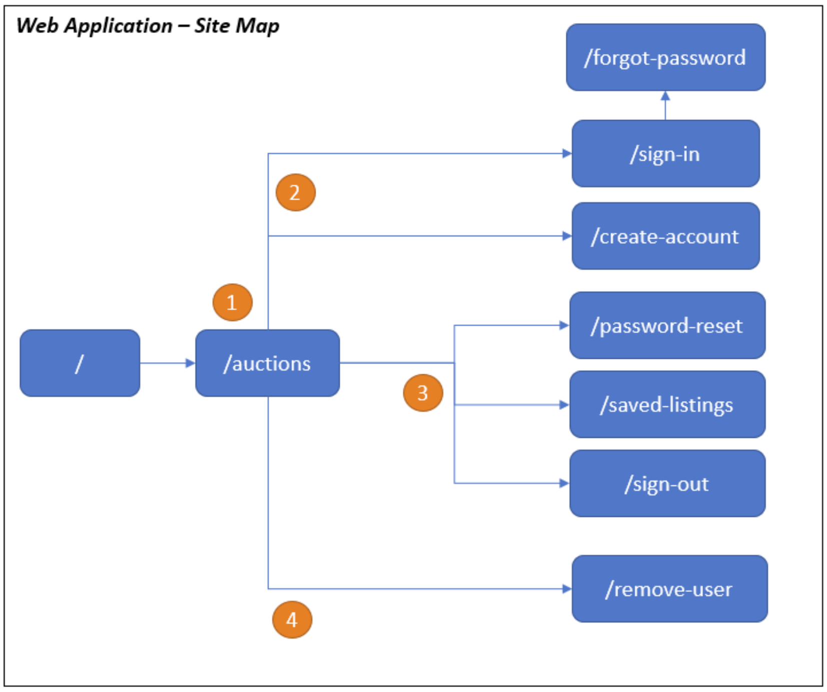


	Function name	Description	Package type	Runtime	Code size	Last modified
○	car_ai_function		Zip	Python 3.7	360 bytes	2 hours ago
○	auction_email_notifier_lambda_function		Zip	Python 3.7	12.7 MB	13 hours ago
○	govdeals_webscraper_lambda_function		Zip	Python 3.7	42.5 MB	11 hours ago
○	gcsurplus_webscraper_lambda_function		Zip	Python 3.7	42.5 MB	11 hours ago

Frontend:

- Account registration: users are able to register for an account, which enables them to access additional features of the website such as saving lots as well as tracking lots they are interested in, in order to be notified of their status and latest highest bid via email.
- Sign in and password reset: after having created an account, users are able to sign into the website and change their password either once they are logged in, or request a password reset in case they have forgotten it.
- Search automobile listings via filters based on manufacturer, car type, color, purchase year, and mileage.
- Add lots to tracked and saved lists: users have the ability to mark any of the active car listings accessible in the auctions page as "Tracked" or "Saved". When lots are marked as tracked, they are included in a notification email sent out to the users every 24 hrs with the latest status changes pertaining to each lot, including the latest highest bid. The lots marked as saved can be accessed by the user directly via the "saved lots" tab in the navigation bar at the top.

2.3. Site Map



3. Webapp Architecture

3.1. Background Processes

3.1.1. Website Scraper

Web scraping is made up of two distinct back-end processes for GCSurplus and Govdeals websites. The scraping is done using Python Beautifulsoup library, which is executed on the specific portions of the websites that pertain to vehicles listings. The information is gathered into a Pandas dataframe and after iterating over all available urls, extracted into the DynamoDB table. The pages of the website are accessed with a roughly 5-10 second delay to avoid straining the hosting server.

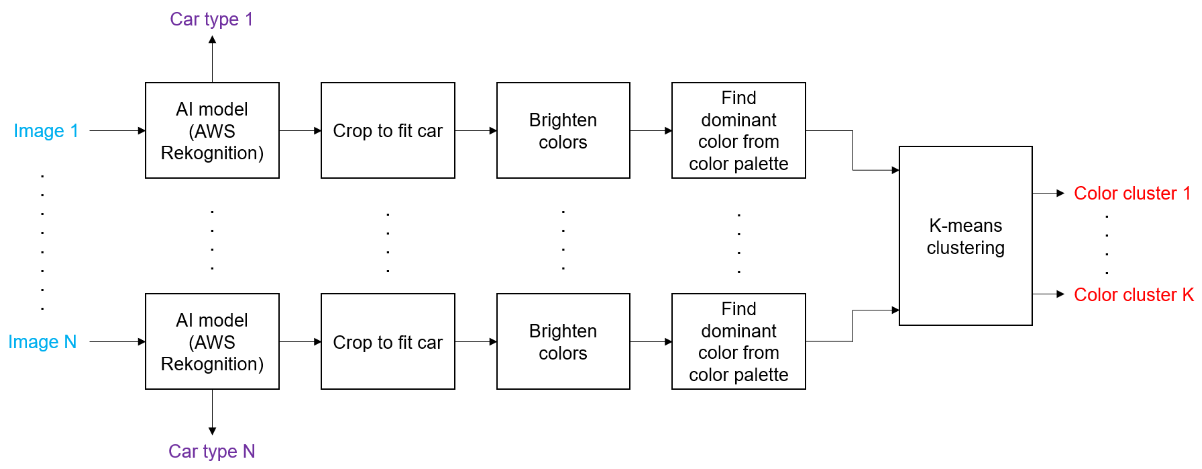
Parsing scripts are run on two separate lambda functions, which are scheduled to be invoked every 24 hours. In addition to scraping the latest listings, each function also updates the status of previous listings in the table by checking whether the listings marked as “active” should be changed to “sold” status based on the closing date of the lot.

Websites scraped:

- GCSurplus:
  - URL: https://www.gcsurplus.ca/
  - Categories:
    - Cars and light trucks
- Govdeals:
  - URL: https://www.govdeals.com/
  - Categories:
    - All terrain vehicles
    - SUVs
    - Trucks, light duty under 1 ton
    - Vans

3.1.2. AI Insights

We use a dedicated background process to run an AI model to extract insights from the auction lots. For each lot, the automobile type and color are deduced based on the image provided with the listing. The automobile type is obtained by feeding the image to Amazon's AWS Rekognition service, which is a trained neural net that can identify labels such as 'Sedan', 'Pickup Truck', 'Van', and other types of vehicles. The neural net also outputs a bounding box encapsulating the vehicle, which we use to crop the vehicle from the background. The cropped image is brightened to accentuate its colors, after which a color palette is extracted using Python's PIL library. The color palette indicates the dominant color in the cropped image which we use as an indicator of the car's color. Related colors are then binned into color categories using a K-means clustering algorithm provided by sklearn library. Finally, we use this deduced information about the automobile type and color to allow users to filter vehicle listings by type and color. For instance, the user can choose to view listings of “blue sedans” without having to specify a specific manufacturer.



The AI model is defined as a lambda function that gets invoked by the web-scraping script once it completes scraping. Upon invocation, an EC2 instance is booted up to run the AI model and update database entries with extracted insights, and on completion the EC2 instance is automatically stopped.

3.1.3. Email Notifier

The email notifier can send an email to every registered user every 24 hours. The email contains the active auctions that the user selected for tracking through the UI. No email notification is sent to a user if they are not tracking any auctions, or if all of the auctions that they were tracking have become inactive (meaning the auctions have passed their closing date). The email body contains an auctions table where each row represents an auction, and each column represents a feature of an auction (like closing date or mileage in the context of automobiles). For user convenience, the last column of the auctions table is the url of the website where the auction exists. The user can conveniently use this url to navigate to the origin website of the auction and perform any actions necessary (like bidding).

3.2. Database

The database for the application is built using AWS DynamoDB services and is made up of two main tables:

1. “**gcsurplus**”: this table contains information about the car listings that are parsed from the auction websites. The lot_id is taken from the auctions and is used as a key to identify and link each listing to the images stored in S3 bucket. This table is automatically updated whenever a parsing script is updated. In addition to information coming from auction websites, fields “car_type” and “color” are updated from the car_ai script that analyzes images of lots that were parsed.

JSON Schema:
```
{
     “TableName”: “gcsurplus”,
     “Item”: {
          “lot_id”:        “type”: “string”,
          “closing_date”: “type”: “string”,
          “lot_status”:    “type”: “string”,
          “make”:          “type”: “string”,
          “mileage”:       “type”: “string”,
          “model”:         “type”: “string”,
          “url”:           “type”: “string”,
          “year”:          “type”: “string”,
          “car_type”:      “type”: “string”,
          “color”:         “type”: “string”,
          “current_bid”:   “type”: “number”,
          “starting_bid”:  “type”: “number”
     }
}
```

2. “**users**”: this table holds users’ data and is used for registration and when accessing the application. The table stores users’ information such as email, which is used for notification purposes and their password, which is encrypted using bcrypt password hashing function based on the Blowfish block cypher. In addition to that, it also contains personalized data regarding users’ tracked lots, which are the listings they wish to be notified about while they’re active and saved lots, which are listings that they can access directly in the “saved auctions" tab.

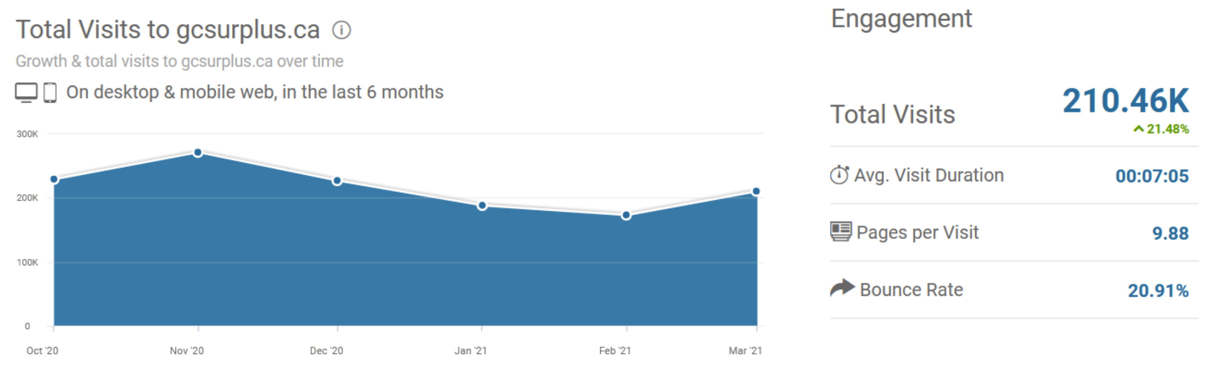
JSON Schema:
```
{
     “TableName”: “users”,
     “Items”: {
          “username”:      “type”: “string”,
          “email”:         “type”: “string”,
          “password”:      “type”: “string”,
          “tracked_lots”:  “type”: “StringSet”,
          “saved_lots”:    “type”: “StringSet”
     }
}
```

4. Cost Model

4.1. App Usage Analysis

The intended user-base for the application is to capture the traffic flow of current Canadian audience for gcsurplus and govdeals that browses those websites for vehicles. The plan is to also capture the organic traffic that goes through those websites, so in addition to understanding the monthly visits and proportion of visits intended for viewing car listings, we are also going to assess the percentage of traffic coming through search results. All of this information can be obtained through https://www.similarweb.com.

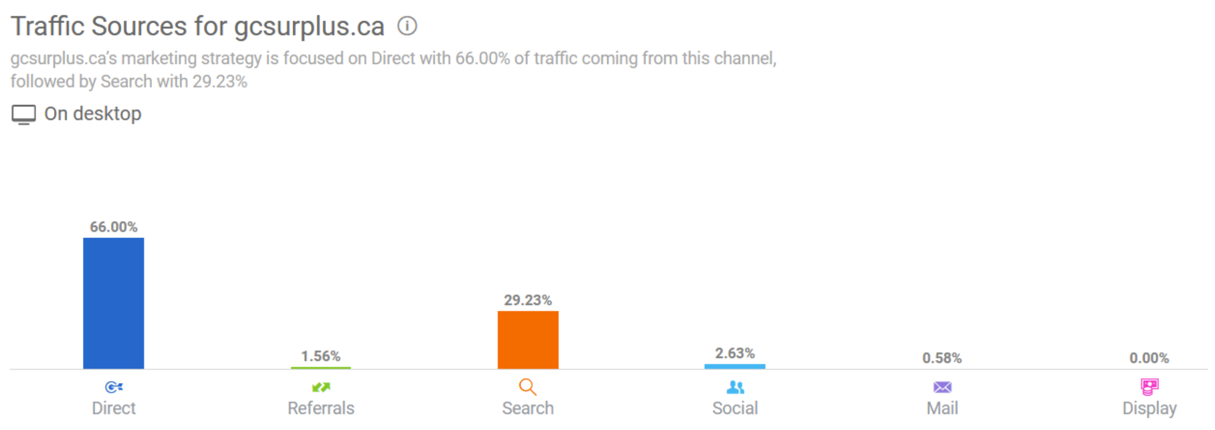
GCSurplus traffic = 8,120 (200,000 monthly visits * 14% of visits intended for cars category * 29% of organic traffic)


Total Visits to gcsurplus.ca
Growth & total visits to gcsurplus.ca over time
On desktop & mobile web, in the last 6 months
300K
200K
100K
0
Oct '20
Nov '20
Dec '20
Jan '21
Feb '21
Mar '21

Engagement

Total Visits	210.46K ^21.48%
Avg. Visit Duration	00:07:05
Pages per Visit	9.88
Bounce Rate	20.91%

Source: https://www.similarweb.com

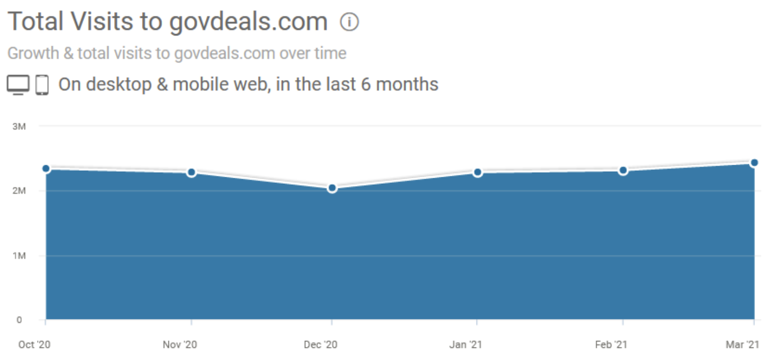

Traffic Sources for gcsurplus.ca

gcsurplus.ca's marketing strategy is focused on Direct with 66.00% of traffic coming from this channel, followed by Search with 29.23%

On desktop

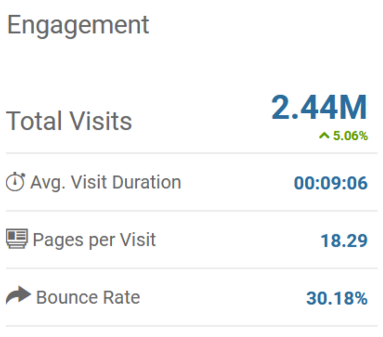
Source: https://www.similarweb.com

Govdeals traffic = 469 (2,400,000 monthly visits * 1.48% of traffic coming from Canada * 6% intended for cars category * 22% of organic traffic)


Total Visits to govdeals.com ⓘ

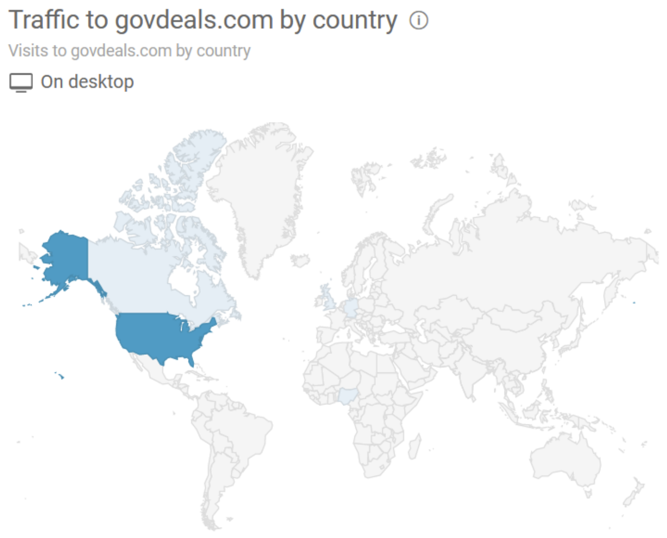
Growth & total visits to govdeals.com over time

On desktop & mobile web, in the last 6 months


Engagement

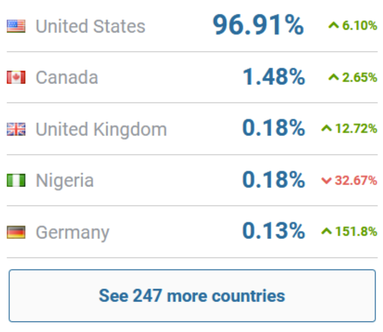
Total Visits	2.44M ^5.06%
Avg. Visit Duration	00:09:06
Pages per Visit	18.29
Bounce Rate	30.18%

Source: https://www.similarweb.com


Traffic to govdeals.com by country ⓘ

Visits to govdeals.com by country

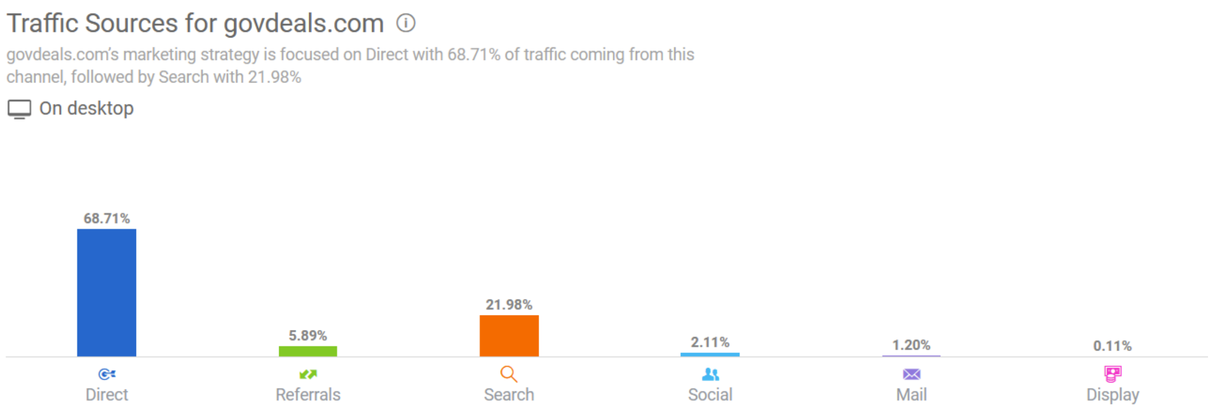
On desktop



United States	96.91%	^6.10%
Canada	1.48%	^2.65%
United Kingdom	0.18%	^12.72%
Nigeria	0.18%	˅32.67%
Germany	0.13%	^151.8%

See 247 more countries

Source: https://www.similarweb.com


Traffic Sources for govdeals.com ⓘ

govdeals.com's marketing strategy is focused on Direct with 68.71% of traffic coming from this channel, followed by Search with 21.98%

On desktop

Source: https://www.similarweb.com

All together, the expected traffic for the app in the first few months of its launch will be around 8,589 users per month. For the purpose of cost analysis however, we are going to present 3 different scenarios of users falling into a group of 10, 1,000 and 1,000,000 people.

4.2 Cost Model

Throughout the cost analysis, the services associated to the following processes will have a fixed cost, regardless of the number of users and will therefore follow a fixed cost model:

- Website scraping for GCSurplus and Govdeals (see 3.1.1. for details)
- The car AI model (see 3.1.2. for details)

Assumption: an assumption applied to these processes is that the number of listings parsed every day based on current scraping criteria will average around 60 per day. Because auction listings typically change week to week, their overall number remains fairly consistent week to week.

4.2.1 Lambda function

AWS Lambda Pricing: https://aws.amazon.com/lambda/pricing/

Fixed Cost

Daily cost formula = Price per 1ms for a given memory size * Billed Duration

Lambda Function	Memory Size (MB)	Billed Duration (ms)	Daily cost (usd)
GCSurplus scraper	128	14min * 60,000 = 840,000	0.0000000021 * 840,000 = 0.001764
Govdeals scraper	128	14min * 60,000 = 840,000	0.0000000021 * 840,000 = 0.001764
Car AI	128	800	0.0000000021 * 800 = 0.00000168
Total:			0.00352968
Total Cost after 6 months (usd):			**0.635342**

- *Assumption 1*: average time to run the scraping functions for 60 listings is around 14 minutes

- *Assumption 2*: the time required to run car AI function is around 800ms
- *Assumption 3*: both web scraping functions will be scheduled to be invoked once every 24 hours
- *Assumption 4*: the cost of ec2 instance associated with the Car AI function will not be accounted toward the total cost as the instance is only active for under a minute, hence the cost is negligible

Variable Cost

Due to the nature of the website, the variable cost estimation assumes a somewhat constant traffic flow. The auctions typically follow a weekly process and new lots are usually listed week to week, hence users of this app will be expected to exhibit repetitive behavior week to week. For that reason, the web activity is going to be estimated on a daily basis and averaged across users.

The billed duration will be estimated using an approximation of average user behavior of the app. The table below indicates the actual billed duration of various pages of the website for a single user. A typical user is expected to follow the following pattern:

Main page → Sign-in → Saved lots → Sign-out

The assumption regarding the app is that the auction page is going to be paginated with a default of 25 listings per page and that the average user will visit 2 pages. This comes to a total average billed duration of 71ms per user per day.

Web Page	Billed Duration (ms)
Main page - auctions	18
Saved lots	5
Registration	50
Sign-in + post-sign in	20
Change password	5
Sign-out	10

In addition to that, each user is expected to visit the registration page at least once to create an account and may have to change their password, hence an additional 55ms will be added over the entire period of 6 months for the number of users.

<u>Cost formula</u> =

[(Cost per invocation for a given memory size * Billed Duration/day) * Number of days + (Cost per invocation for a given memory size * Fixed billed Duration/user)]
* Number of users

Cost of Web Application:

Billed Duration (ms) / user / day	Cost / day (usd)	Fixed cost / user	**Total Cost after 6 months (usd)**		
			10 users	1,000 users	1,000,000 users
71	0.0000000083 * 71 = 5.893E-07	0.0000000083 * 55 = 4.565E-07	**0.001065305**	**0.1065305**	**106.5305**

- *Assumption 1*: expected average memory size for the application throughout its usage will remain at 512MB
- *Assumption 2*: we limit the number of auctions we load on the home page via pagination to 25 listings per page
- *Assumption 3*: every user will register for an account within 6 months

The email notification function is expected to run every 24hours and its cost is going to vary with the number of users signed up for notification. The billed duration per user has been extracted doing actual testing and for the cost projection, we will assume that all users will sign up for notifications.

Daily cost formula = (Cost per invocation for a given memory size * Billed Duration/day) * Number of days * Number of users

Cost of Email Notification:

Billed Duration (ms) / day / user	Cost per Day (usd) / user	**Cost after 6 months (usd)**		
		10 users	1000 users	1,000,000 users
1,600	1,600 * 0.0000000021	**0.006048**	**0.6048**	**604.8**

- *Assumption 1*: all users will sign up for notifications
- *Assumption 2*: email notification function will be scheduled to be invoked once every 24 hours
- *Assumption 3*: expected average memory size for the application throughout its usage will remain at 128MB

4.2.2 S3 bucket

AWS Lambda Pricing: https://aws.amazon.com/s3/pricing/

Storage pricing	
First 50 TB / Month	$0.023 per GB
Next 450 TB / Month	$0.022 per GB
Over 500 TB / Month	$0.021 per GB

The S3 costs are going to be considered as fixed since their storage size is not going to be impacted by the traffic flow of the app and will therefore have a fixed cost over the 6 months period.

The bucket holding zipped files for the web scraping lambda functions will be holding 2 files at a time. The two zip files total 90MB = 0.09GB / month.

The bucket allocated to the web application does not seem to be having any activity, hence will not have any cost associated with it.

The bucket allocated to holding images of scraped auction listings will have a variable cost. On any given week, the expectation as stated previously is to have 60 active listings, resulting in 60 images to be held in S3 bucket. In order to attain some savings, the images of auctions that are no longer active are going to be deleted to free-up space, although their information will be kept for future analysis in the DynamoDB table.

Each image is expected to be around 100KB in size (based on the data that has been gathered over the past 2 weeks). With that in mind, the expected storage over any given month is 50 images * 100KB = 5MB = 0.00071GB / month.

S3 Bucket	Storage Size (GB)	Total Cost (usd)
Bucket holding zipped files for lambda functions	0.09	0.023
Webapp (zappa)	0	0
Scraped Images bucket	0.00071	0.023
Total Cost after 6 months (usd):		**0.046**

- *Assumption 1*: only active auction images are stored in the bucket, per week each scraper holds 50 images.

4.2.3 API Gateway

HTTP APIs Pricing: https://aws.amazon.com/api-gateway/pricing/

Number of Requests (per month) - First 300 million → Price (per million) - $1.00

The pricing for the HTTP API associated with the web application is measured in terms of API calls. After simulating the typical expected user behavior outlined in the lambda function variable cost model, the expected number of daily API Calls per user is 12.

Cost formula = Number of API Calls / day / user * Number of days * Number of users

API Gateway	API Calls per day per user	**Cost after 6 months (usd)**		
		10 users	1000 users	1,000,000 users
Webapp	12	12 * (6 * 30) * 10 requests = 21,600 requests → $**1.00**	12 * (6 * 30) * 1,000 requests = 2,160,000 requests → $**3.00**	6 * (6 * 30) * 1000000 requests = 2,160,000,000 requests → $**2,160**

- *Assumption 1*: each user will be making 12 API calls per day on average, according to the user behavior outlined earlier

4.2.2 DynamoDB

DynamoDB Pricing: https://aws.amazon.com/dynamodb/pricing

The two tables stored in DynamoDB will be priced according to on-demand and provisioned capacity mode based on their usage rate.
Pricing for on-demand capacity mode:

Type	**Cost**
Data storage cost	$0.25 per GB-month
Write request units	$1.25 per million write request units
Read request units	$0.25 per million read request units

Auctions table: two lambda functions to run every 24 hours, totalling approximately 60 writes and 10 reads - on any given day, there are around 60 active auctions that are parsed and updated in the DynamoDB table with about ⅙ typically set to “sold” status. Total writes per month comes up to 1,600 and 300 reads. Since this table does not have consistent traffic, it makes sense to use the on-demand capacity pricing mode:

	Monthly writes	**Monthly reads**
	1,600	300
Monthly Cost	0.002	0.000075
Cost after 6 months	0.012	0.00045
Data storage monthly cost (usd)	0.25	
Total cost after 6 months (usd)	**1.51245**	

- *Assumption 1*: monthly writes and reads are estimated according to the assumption of expected flow of 60 auctions per day during web scraping

Users table: this table is utilized by the application for user authentication purposes as well as for storing account specific preferences associated with auction lots that users have marked as tracked or saved.

The reads will happen every time a user loads the auction page and access saved listings. Since the expected behavior is for users to load two auction pages and one saved page per day, the expectation is (3 / 86,400 seconds) reads per second per user.

The write occurs whenever a user adds a lot to a tracked or saved list or accessed the application. The expected daily behavior will be to add 1 lot to the tracked and saved list, with a total of (2 / 86,400 seconds) writes per second per user.

Capacity calculator = Number of writes / reads per day per user * Number of users

	10 users	1,000 users	1,000,000 users
Read capacity units	(3 / 86,400 seconds) * 10 → 1	(3 / 86,400 seconds) * 1,000 → 1	(3 / 86,400 seconds) * 1,000,000 → 35
Write capacity units	(2 / 86,400 seconds) * 10 → 1	(2 / 86,400 seconds) * 10 → 1	(2 / 86,400 seconds) * 10 → 24
Provisioned capacity Cost based on the read and write units (usd)	0.59	0.59	13.35
Fixed data storage monthly cost (usd)	0.25	0.25	0.25

Total cost after 6 months (usd)	**5.04**	**5.04**	**161.7**

- *Assumption 1*: Provisioned capacity Cost is taken from the capacity calculator that is available in the DynamoDB capacity calculator

Cost Summary - 6 months

AWS Service	**Fixed cost**	**Cost according to number of users**		
		10 users	1,000 users	1,000,000 users
Lambda	0.635342	0.001065305	0.1065305	106.5305
S3	0.046	-	-	-
API Gateway	-	1	3	2,160
DynamoDB	1.51245	5.04	5.04	161.7
	2.193792	6.041065305	8.1465305	2,428.2305
TOTAL:		**8.23 usd**	**10.34 usd**	**2,430.42 usd**

References

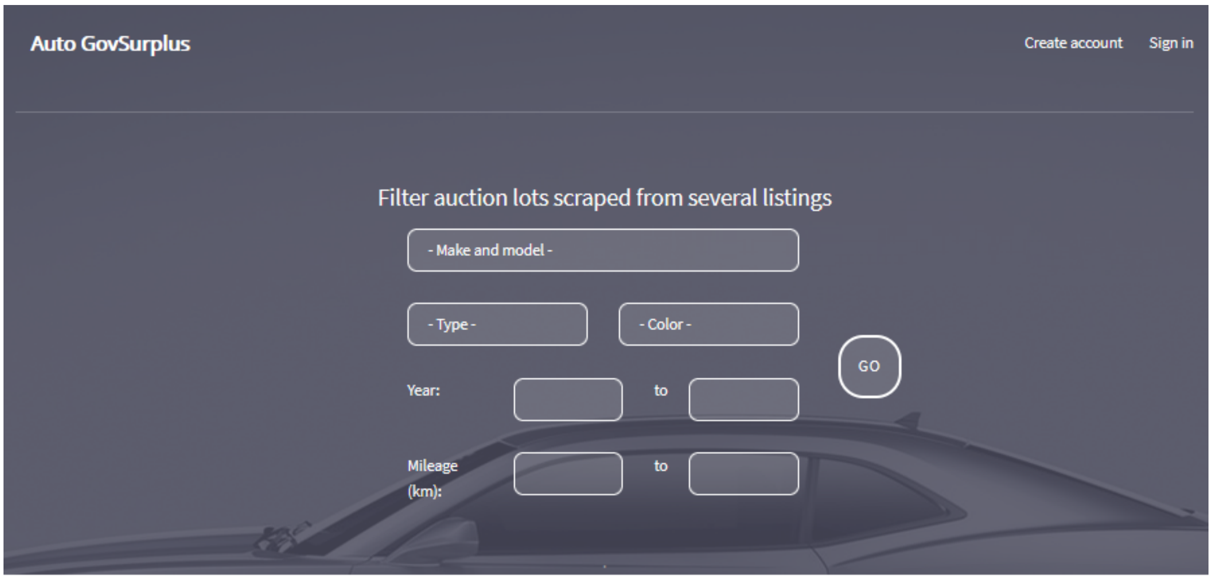
- Homepage wallpaper: https://images.freecreatives.com/wp-content/uploads/2015/03/Car-showroom-website-background.jpg

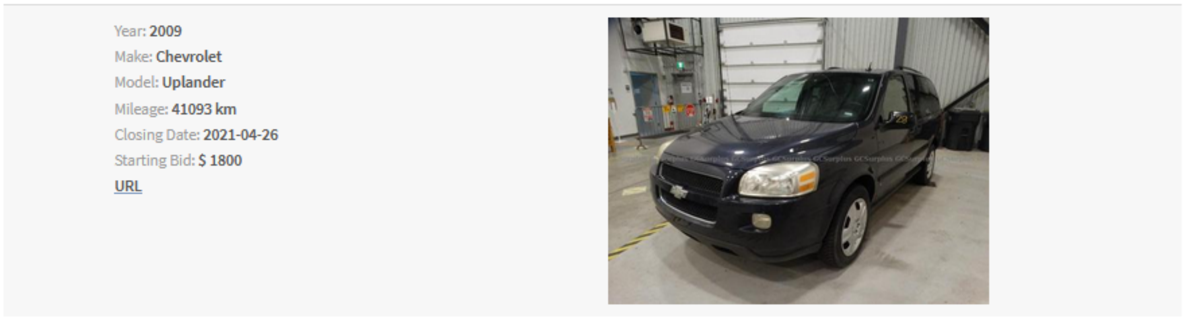
- CSS style sheets and fonts: https://templated.co/projection

Appendix:

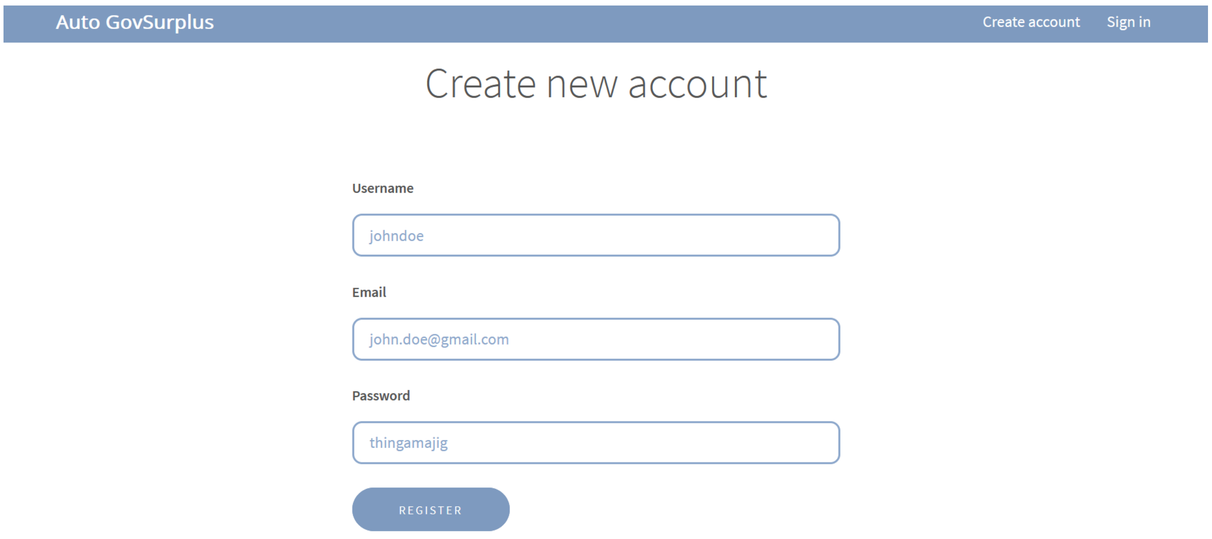
Appendix A: Webapp Screenshots

Main page:




Year: 2009
Make: Chevrolet
Model: Uplander
Mileage: 41093 km
Closing Date: 2021-04-26
Starting Bid: $ 1800
URL

Account registration:



Create new account

Username

johndoe

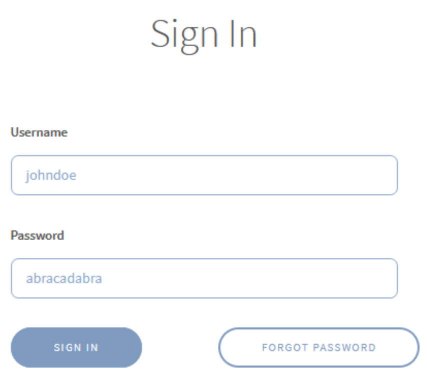
Email

john.doe@gmail.com

Password

thingamajig

Sign-in and Forgot Password pages:



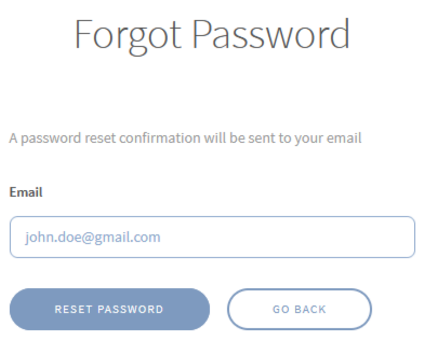
Sign In

Username

johndoe

Password

abracadabra



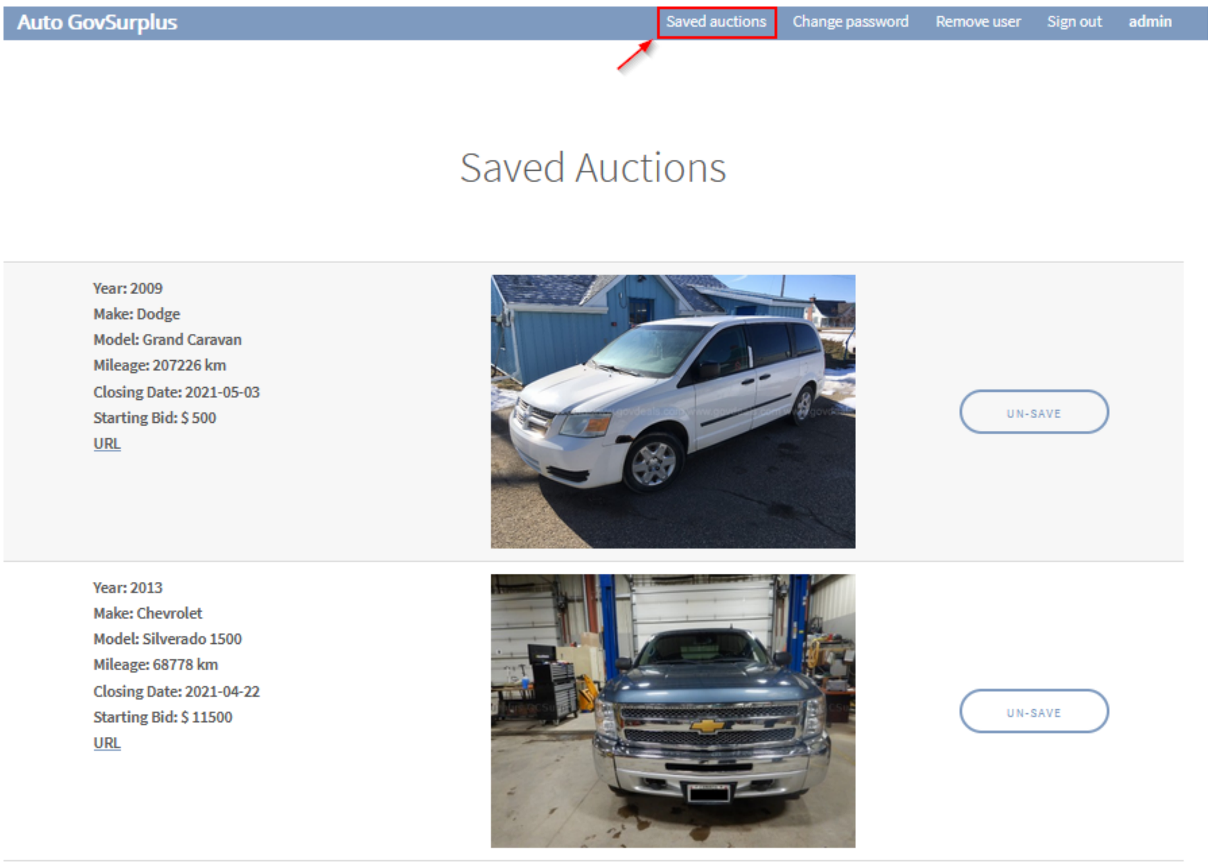
Forgot Password

A password reset confirmation will be sent to your email

Email

john.doe@gmail.com

Saved Auctions:



Saved Auctions

Year: 2009
Make: Dodge
Model: Grand Caravan
Mileage: 207226 km
Closing Date: 2021-05-03
Starting Bid: $ 500
URL

Year: 2013
Make: Chevrolet
Model: Silverado 1500
Mileage: 68778 km
Closing Date: 2021-04-22
Starting Bid: $ 11500
URL