Writing 4 - Technical Design Document

Names: Abdullah Almaayoof, Saif Alzaabi, Dakharai Murray

Elevator Pitch

“The global online car buying market was valued at $237.97 billion in 2020, and is projected to reach $722.79 billion by 2030.” This means that more cars will be listed for sale online, and more car sale websites will be created. The car-buying process is a tedious and time-consuming process. Car buyers are likely to be viewing many websites to compare all of the possible options. What if there was a tool that would recognize what car the user is looking for, scrape the web for all similar cars, and list the best options to buy based on the user's preferences?

Carcow is a google chrome extension that will make every car buyer’s life easier. When the user is viewisng a car listing they are interested in, the extension will recognize the car being viewed. Then it would use that information to compile a list of all the similar cars on the web using our dynamic scraping algorithm. Carcow will then evaluate the cars on the list based on the user’s input preferences, and produce a score of 10 for each car. Finally, a list of the cars with the highest scores will be shown to the user.

What differentiates Carcow from other car search engines is that it will prioritize showing the best cars based on the user's preferences. While other websites, such as autotempest.com, do a standard search across the most popular car sales websites and

output a list. Our rating algorithm will take in user preferences for its input: how much they value price, low mileage, car Trim, color, etc. Then our algorithm will use those preferences to rank each car for each individual. So one person's 7/10 car would be another 9/10 dream car.

Many car sales websites may be incentivized by dealerships to show overpriced cars at the top of the search results. But that wouldn't be a problem for Carcow users since the algorithm is able to scrape the entire website and extract the best deals.

Another selling point of our program is that the user will be able to explore all the best deals without having to leave the page they are currently on. Searching for cars online will typically leave users with tens of open tabs on different websites, and a real difficulty when comparing them. Carcow will do all the work in the background and present the results to the user on the chrome extension popup, and the only time they need to go to a new page is when they find the car they are looking for.

Our target customers will be individuals looking to buy second-hand cars online. Typically, the users' ages will range from 22 to 40 (Millenials), who make up the majority of people who look for cars to buy online. Our tool would work for users around the globe as long as they are on an English-speaking car listing site, which is another selling point of Carcow.

The Dynamic scraping algorithm would use a model that is set to recognize a car listing website apart from any other kind of website. It will then be able to recognize key data points within each listing page to be able to extract important information such as make, model, mileage, and vin. This algorithm would take a large portion of our focus

during this project because it will go into being able to activate the extension when a user is on a car page, extracting car data from the browser, and being able to scrape other websites. This algorithm would allow us to extract car data from an infinite number of car websites, given the websites follow the general car listing format (1 page per car, general car data, contact information, VIN). Any changes to a website's HTML page tree or HTML class names will not affect the success of this algorithm when scraping.

In conclusion, searching for cars online can be difficult and time-consuming. Cracow will help consumers save a lot of time and money when searching for cars online. The dynamic scraping algorithm will search through the major car listing websites for a large number of cars that match the user’s preferred car, then it will output the best cars for the user, all while remaining on the same page.

Product Specifications

User Stories

As an upcoming college student, who will be commuting to campus daily. I'm not good with cars, so I can't differentiate between a good and a bad deal. I would like to find the cheapest possible car to reliably get me from home to school.

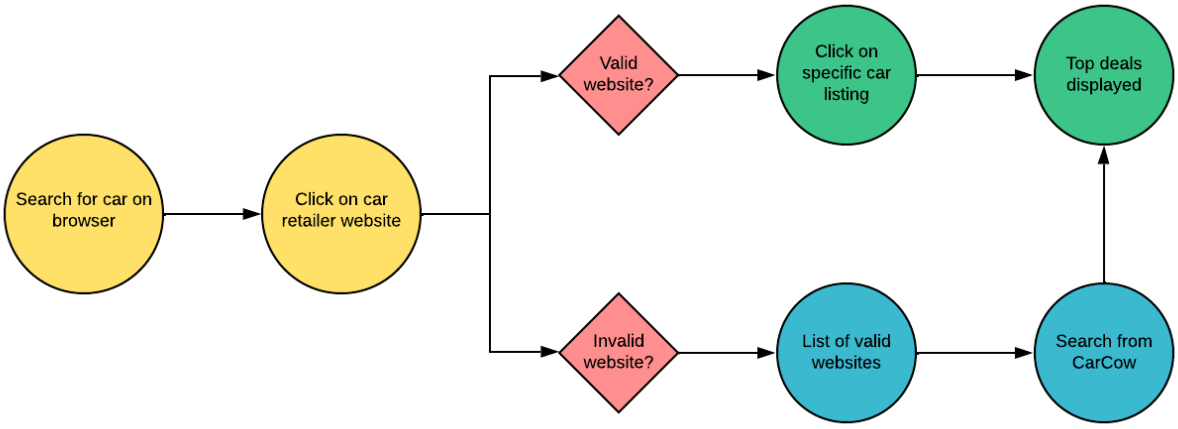
As a standard employee, my truck recently broke down on me and had to be scrapped, so I'm looking for a replacement that won't break the bank and still be the truck model I like.

As a busy business man, I'm looking for a S class Mercedes with very specific preferences, however I don't want to spend a lot of time online or at dealerships.

As a small business owner, I need a van with low mileage to deliver my inventory, however I do not want to spend too much time or money getting that van. The van needs to be low in mileage, and close to where I am located.

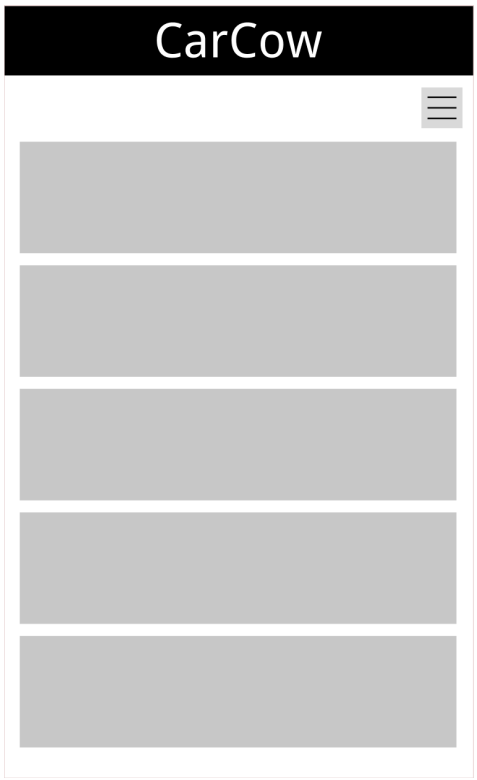
As a father of a 16 year old boy, I would like to find a sports car for my son when he gets his driver's license. I am open to exploring many options and would like to find the best deal of each option I consider before buying.

Flow Diagram

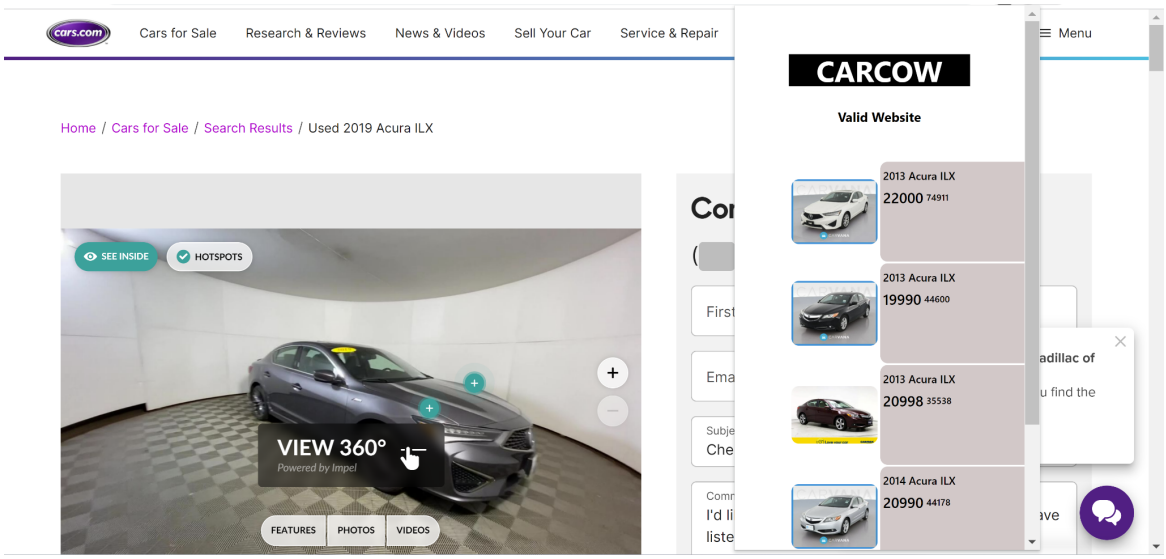


Mockups/WireFrames

Mockups from Figma

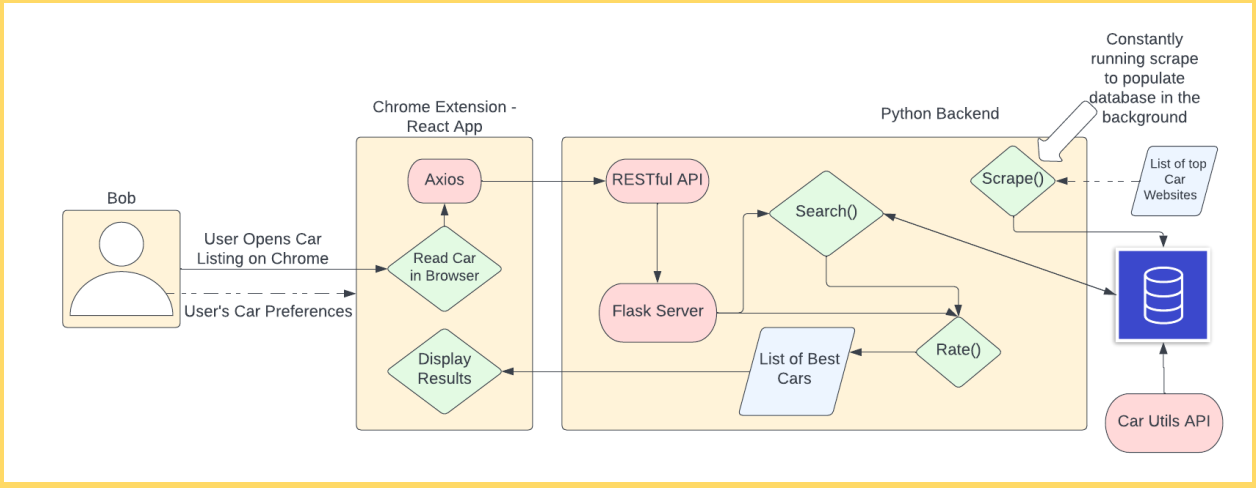


Actual current User Interface


cars.com
Cars for Sale
Research & Reviews
News & Videos
Sell Your Car
Service & Repair
Menu
Home / Cars for Sale / Search Results / Used 2019 Acura ILX
SEE INSIDE
HOTSPOTS
VIEW 360°
Powered by Impel
FEATURES
PHOTOS
VIDEOS
CARCOW
Valid Website
2013 Acura ILX
22000 74911
2013 Acura ILX
19990 44600
2013 Acura ILX
20998 35538
2014 Acura ILX
20990 44178

Technical specifications

Architecture/System Diagrams



Description

The program architecture diagram above details all the processes in effect when a user searches for a car using CarCow. The user first connects to a valid car listing website. The program reads the current page and sends its data to our flask server using Axios and RESTful API. The flask server will start the calls to all of our algorithms and return any outputs back to the extension using RESTful API. Once car data in the browser is processed within the flask server, the function search() is called. Search looks for all cars in the database with the same make, model as the car listing and within a certain radius of it. The database is constantly being filled with scraped data from the top car listing websites to maintain the current available cars in the market, as well the market price of each car using Car Utils API. Once the searched data is retrieved from the

database, it is sent to the rating algorithm Rate(). This algorithm also takes in the user's car preferences from the flask server to process the list of best cars specific to that user. Both Scrape() and Rate() are broken down into more detail in the Algorithms section. The output list of top deals is then sent to the user in the chrome extension.

External APIs and Frameworks

- **Beautiful Soup**
    - Goals
        - Scrape 5 websites for car listing data, and scrape single car listing of which car the user is viewing
    - Description
        - The Beautiful Soup API is used in our extension.py and scrapeV1_6.py files in the code. It connects to the scraping functionality and is called when a valid website URL is found and when we need to scrape from all the retailer websites. A car listing URL is passed as a parameter into one of our singleCarDataX functions, where X corresponds to one of the 5 websites we want to scrape data from.
    - Endpoints Used:
        - page = requests.get(url) ← Gives us the raw HTML
        - Beautiful Soup(page.content, 'html.parser') ← Parses the raw HTML to allow us to find elements within it
- **RESTful API & Axios Framework**
    - Goal
        - Facilitate communication between the chrome extension and server
    - Description
        - We make use of the RESTful API and Axios Framework to send requests from our App.jsx file and to GET responses from our

server.py file. The RESTful API allows us to have two components, the client (our Chrome extension) and the server (our Python server), that we can pass information between, as Axios requests, and facilitate the necessary communication between our users and the algorithms we use to scrape for and rate car listings. Once this data collection and rating is completed, processed data is passed back from server to client (via the Axios Framework) for display on the user's screen using the render code at the bottom of App.jsx.

    - Endpoints Used
        - GET/query(active web page URL)/ {fetchURL}/axios request/ getURL(this is where we get our car data)/{url}/axios GET/render

- **Selenium**
    - Goal
        - Automated web site navigation to the data we need
    - Description
        - We make use of Selenium in our rateV1.py, extension.py, and our scrapeV1_6.py files. It connects to our web scraping functionality and is called when we need to interact with the web pages that we scrape information from (specifically when we need to click buttons on the page or input data into a form). The webpage URL is a parameter generated by the scraping function and passed into this API for navigation by searching for specific div classes.
    - Endpoints Used:

        - GET/{url}/{div class}

- **Car Utils**
    - Goal
        - Get market value for each scraped car.
    - Description
        - This API is used in our rateV1.py file in the code. It connects to the rating functionality and is called when the market value of the car is needed. The parameters VIN and mileage are generated by the scraping function and passed into this API to get the result.
        - We use Car Utils for the car evaluation, as this is how we obtain the market value of the car that we then use in the rating algorithm.
    - Endpoints Used:
        - GET /car-utils.p.rapidapi.com/marketvalue/headers/{vin,mileage}

- **Amazon Web Services (AWS) RDS Database**
    - Goal:
        - Maintain a cloud database, which we use to store the information of all scraped cars.
    - Description:
        - This framework is used in both server.py and database.py. It connects to the insertions/extractions of the database functionality. All relevant data about each car is used as the parameter to pass

into the AWS database and is generated by both the scraping and rating functions.

- Using the AWS RDS database ensures that when we activate our extension to search for car listings, this job can be accomplished very quickly and provide results to our users in a timely manner, as the data is already scraped and collected, so it would just need to be retrieved.

- Endpoints Used:
  - SQL SELECT & INSERT

Algorithms

Dynamic Scraping Algorithm

Goal

The Dynamic Scraping Algorithm will allow us to scrape most car listing websites without looking into the html path of the elements in the listing. This will allow us to access a large number of websites car listing data without having to hard code an scraping function for each website.

Description

The algorithm will work by parsing the html of each site and using regular expressions to identify key elements to store about each car. The first step will be to navigate the homepage to get a search results page, so it will look for make and model select fields as well as a search button. The second step will be to navigate the search results page and identify the html element which contains each car, which is where most of the car data is. In that html element, the algorithm will parse text to find any recognizable, make, model, trim. Then it will look for any text which matches the price, mileage, or vin format. This will result in an algorithm that will scrape car data dynamically without relying on a predetermined html parse tree.

Rating Algorithm

Goal

Carcow will be able to evaluate each scraped car and rank them using The Rating Algorithm. The user will input their preferences into the extension which will affect how each car is rated.

Description

The rating score will take into account 5 categories for car evaluation : Price, Mileage, Trim, Color, and Distance. Each category will generate a rating for each car, which will be out a maximum of 1. The rating will be multiplied by the ratio of the input user preferences on the sliders in the extension. For example, if the user has an equal preference for all categories then each rating will be multiplied by 1⁄5 then added together to get the final rating out of 1.

The price rating is based on the ratio of the market price of that car to the listed price of the car, the higher the ratio the better the price rating. We will get the market price from Car Utils API. The Mileage rating is based on a miles per year ratio for the car inverted, so the higher the inverted ratio is, the better the mileage rating. The Trim is just a binary score where the listing trim matches the trim desired by the user makes the trim rating positive, if it doesn't then it's a zero. The color rating is based on the listed color of each vehicle, and how close they are to the desired color of the user. The distance rating is just the ratio of the user's desired range to the listed car's location, the closer the car is to the user, the higher the rating. Each of these ratings will lead to a personalized list of top cars based on the user's preferences.