https://github.com/53Dude/TheDrag.git

Daniel Lazcano, daniellazcanoplus@gmail.com, https://github.com/DanielLazcano
Ethan Santoni-Colvin, ethan.santonicolvin@gmail.com, https://github.com/ethansantonicolvin
Frank Le,fle734449@gmail.com, https://github.com/fle734449
Guy Sexton, 53dude@gmail.com, https://github.com/53Dude
Jonathan Walsh, jdwalsh79@gmail.com, https://github.com/jdw4867
Kishan Dayananda, kishdaya@gmail.com, https://github.com/kish314

Group: Morning - 4 / Pit Crew

The Drag

Motivation and Users:

TheDrag is a website built for users who want a consolidated place that holds all the automobile information they would need. Whether it's a college student looking prices and dealerships for their first car purchase, speed-demon car enthusiasts looking to find the next hot rod on the market, or a conserved environmentalist who needs to get to where they but wants to find the least carbon emissive vehicle that they can get; TheDrag serves to bring all that info right into your web-surfing device.

User Stories:

Our own:

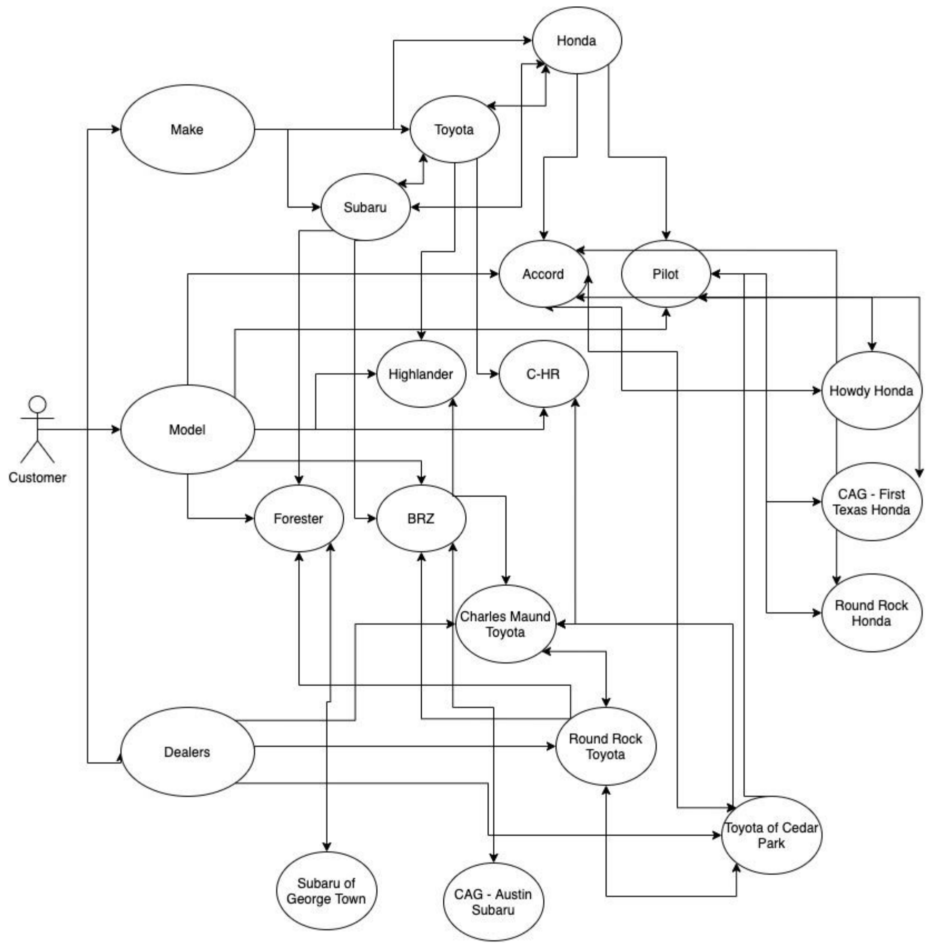
1. I want a list of current manufacturers that are sold around my area.
   Estimated Time: 2 hours
   Actual Time: 4 hours
2. I'm a die hard Subaru fan, I need to know which models are made by Subaru.
   Estimated time: 2 hours
   Actual Time: 2 hours
3. As an undergraduate college student with no income, I want a way to find a cheap car, so I can drive to school.
   Estimated Time: 3 hours
   Actual Time: 3 hours
4. As a commuter to work, I want to know the highway mpg of certain models.
   Estimated Time: 4 hours
   Actual Time: 2 hours

5. I’m in the market for a car. More than just knowing the price, I want to know what the car will look like before I buy it.
   Estimated Time: 2 hours
   Actual Time: 3 hours

<u>Customer Team:</u>

1. As a concerned user of who is making my product, I would like to see the developer team of the product.
   Estimated Time: 2 hours
   Actual Time: 2 hours
2. I want to be clear about the purpose of the website.
   Estimated Time: 1 hours
   Actual Time: 1 hour
3. As a user with bad eyesight, I want to be able to use the website easily for someone with my disability.
   Estimated Time: 2 hour
   Actual Time: 2 hours
4. As someone who browses the internet every day, I would like a clean and polished UI. Assuming this customer can tell if a product is worth it based on how the developer team may look.
   Estimated time: 3 hour
   Actual Time: 3 hours
5. I would like to know the closest locations of dealerships selling a certain model.
   Estimated Time: 3 hours
   Actual Time: 3 hours

Use Case Diagram:

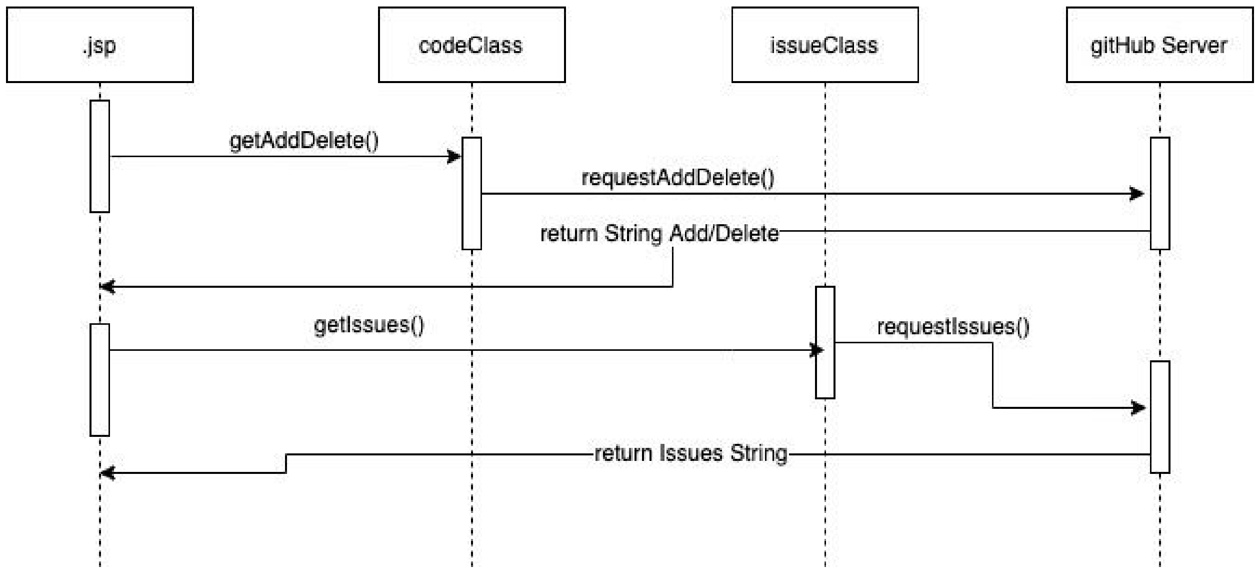


Design:

Our design was essentially giving users the options of choosing to browse by make, model, and dealership of a car. Clicking “browse by make” will take you to a list of makes available. Clicking a specific make shows a list of model instances to click, as well as other available makes on the bottom and dealerships associated with that make. Clicking a specific model on that page takes you to information about that model such as an image, price range, manufacturer, and the dealerships that possess these models. One can access this list of models directly by first clicking on “browse by model” at the top on the navbar. Clicking “browse by dealerships” up top takes you to a page listing dealerships nearby. Clicking a single instance of these takes you to a detailed description of the dealership, along with an image and address. It also reveals all the available models that are sold at that dealership. So basically, make, model and dealership are all linked to each other in some way where you can access any instance from another instance.

We derived our About page statistics dynamically from GitHub by making calls to two different Java classes in our jsp file, CodeStats.java and IssueStats.java. CodeStats pulls GitHub’s information on number of commits, while IssueStats retrieves the number of issues assigned from our GitHub. If the request to the GitHub API is denied, it will print an error message and still show the about page, otherwise it will print the information. If GitHub statistics doesn't show up, wait for a while and retry, this issue is due to the GitHub API, not our code.

The 17 static pages that we have created for Phase 1 and described in the above UML diagram are About, Accord, BRZ, Charles Maund Toyota, C_HR, Dealers, Forester, Highlander, Honda, Makes, Models, Pilot, Round Rock Toyota, Sitemap(Homepage), Subaru, Toyota, and Toyota of Cedar Park.



Testing:

No formal testing was done in this phase.

Models:

Our three models are the make of a car, the model, and the dealerships. When accessing makes of car, you have options to choose from that then take you to a bunch of different models of that car. You can also access these models by clicking browse by model as well and clicking browse by dealerships will take you to a list of dealerships, each linked to an instance of a specific car model. We scraped data for makes, models, and dealerships from various APIs along with images as well. These sources are listed below:

API 1: https://apidocs.marketcheck.com/?version=latest
API 2: https://vpic.nhtsa.dot.gov/api/
API 3: Youtube

API 4: https://www.fueleconomy.gov/feg/ws/index.shtml
API 5: https://pricedigests.com/api/
API 6: https://api.carmd.com/member/docs#image
API 7: Twitter
API 8: http://www.dougdemuro.com/dougscore
API 9: https://www.cars.com/

The three instances of Make that we scraped data for are Honda, Toyota and Subaru. On their respective static page you will see a JSON file that has been outputted to the page. This JSON includes data on all the different car models associated with that make.We collected this data by running MakeServlet.java which goes to the NHTSA API and scrapes the Make data and writes it into a JSON file.

The three instances of Car Models that we scraped are Toyota C-HR, Subaru BRZ, and Honda Pilot. On their respective static page you will see a JSON file that has been outputted to the page. This JSON includes data on price, mileage, dealerships its sold at, attributes of the car, etc. We scraped this data with web scraper which is detailed in our tools section.

The three instances of Dealerships that we scraped are Charles Maund Toyota, Round Rock Toyota, and Toyota of Cedar Park. On their respective static page you will see a JSON file that has been outputted to the page. This JSON includes information on cars that are sold at that location. For now, we only collected the information of one car sold at each location. In the future the JSON will hold all cars sold at that location. We scraped this data with web scraper which is detailed in our tools section.

Since we were required to make static pages for phase one, all instances of our models have static images representing the instance. In the future we plan to dynamically get the multimedia images by scraping google images and google maps.

Tools, Software, Frameworks:

The Pit Crew (The development team behind TheDrag) utilized the following technical tools to complete Phase I:

Eclipse - The main IDE everyone used to code the Java, CSS, HTML, JSP, XML, etc. files and to run the website on the localhost through Google App Engine integration.

Google AppEngine - The PaaS used to host and do webby stuff.

Adobe DreamWeaver - The Front-End team used this visual to code editor for streamlined editing/development of Bootstrap, CSS, and HTML files.

Bootstrap - A bootstrap design was found online and implemented into our front-end web design.

Maven - Used as our project builder and to quickly add third party libraries for us to use in our project

JSON Simple by Google - Used to encode the JSON file from CSV.

OKHTTP by Square - A request-response API that allows us to collect data on our instances through making a request to other APIs.

Web Scraper by webscraper.io - A chrome extension that allows us to scrape data from websites easily. We can choose elements we want to scrape and running the app allows us to download the scraped data as a CSV file which we can convert to a JSON file ourselves programmatically. We plan to only use this tool for Phase I, going forward we will program our own web scraper.

Reflection:

Our team had good communication going throughout phase 1 via our slack channel. We all made significant efforts to meet up whenever there was a lot of work to be done and were good at letting everyone know when one of us couldn't make it for some reason. We were tremendous at pair programming and it made it much easier to troubleshoot. Front-end and back-end teams worked very well with each other to make sure that everyone was on the same so errors would be minimized in the long run. The team just had great synergy.

Some struggles our team faced were some members having subscriptions to more advanced software to get front-end work done, which is also associated with differences in expertise. But we got around that with free trials and student memberships. Other struggles included our schedules not lining up at times and some people would inevitably have to be left out. We did a good job of tuning in remotely if busy, through hangouts or facetime, but sometimes it was just unavoidable.

We learned what it was like to work in a more team-dependent environment where roles had to be delegated and trust had to be developed for others to get their work done. We learned the necessity of JSPs despite our hatred towards them. We better understood servlets and the calls necessary to make them work, and how to encode JSON files to gather data from them.