Oregon State University
College of Engineering

CS Capstone Final Document

June 11, 2019

Auction Hunter

Prepared for

Ryan Kalb _____ _____
Signature *Date*

Prepared by

Group 4
Auction Hunter

Alexander Hull _____ _____
Signature *Date*

Alexander Jacobson _____ _____
Signature *Date*

Yufei Zeng _____ _____
Signature *Date*

Abstract

This design document outlines the components of our Auction Hunter development project. The elements, relationships, and design concerns will be discussed for each. This will serve as a road map for the remainder of this project.

CONTENTS

1 INTRODUCTION

Auction Hunter was a project requested by Ryan Kalb, our client. It is meant to improve and exhance a user's experience when searching for salvaged car auctions. If someone totals a vehicle, the car will sometimes be auctioned by insurance companies. Some of these are perfectly fixable but insurance companies opt to sell them due to exorbitant repair costs charged by body shops.

When implemented, this tool could be used to maximize profits when selling spare parts or rebuilding vehicles. It could also be used by the public to make sure they are getting a good deal. Action Hunter will help users gather data on these cars quickly and conveniently.

1.1 Members

- Alexander Hull - Worked on database, value calculation, and web scraping.
- Alexander Jacobson - Worked on front end development, and managed a server for the project
- Yufei Zeng - Worked on researching and implementing Web Scraping.
- Ryan Kalb - Established requirements and provided feedback.

Oregon State University
College of Engineering

CS Capstone Requirements Document

April 19, 2019

Auction Hunter

Prepared for

Ryan Kalb _____ _____
Signature *Date*

Prepared by

Group 4
Auction Hunter

Alexander Hull _____ _____
Signature *Date*

Alexander Jacobson _____ _____
Signature *Date*

Yufei Zeng _____ _____
Signature *Date*

Abstract

This document encompasses the features that our proposed product will implement. These features are those requested by our client, and agreed upon by the members of group 4.

CONTENTS

1 Change Log

Section	Original	New
Definitions	Mentioned Teslas specifically	Removed all mentions of Teslas. Replaced general ”vehicles”
System Requirements	Mentioned Teslas specifically	Removed all mentions of Teslas. Replaced general ”vehicles”
User stories	Mentioned Teslas specifically	Removed all mentions of Teslas. Removed mention of automatically bidding functionality. Removed machine learning functionality. Removed ”scrape from IAAI” and replace with ”extendable to scrape from other sources”
Functional requirements	Query Copart and IAAI API’s	Replace API with web scraping. Added Damage information, start code, key presence, and link to original posting. Removed live auction tracking. Remove automatic auction bidding. Removed machine learning component.
Performance requirements	Automatic bidding	Removed automatic bidding perfomrance requirement. Removed automatic bidding requirement and notifications. Removed Copart.
Testing	Section detailing how we will test the accuracy of value prediction by comparing to listed value.	Removed section.
Continuous integration	Section on building a continuous integration system that tests the project each time a commit is made to master branch.	Removed continuous integration section.

2 Introduction

2.1 Purpose

The purpose of Auction Hunter is to find the highest value car auctions for wrecked/damaged vehicles on auction website. Our product will parse through a collection of auctions, evaluate their value, and display results to a user through a website UI.

2.2 Overview

There are three primary components to our proposed product. The first part skims over car auctions using a web crawler and collects all relevant data, then saves it. The second part parses through all that saved data and performs estimations on value, which can be compared to the current asking price. The third component displays this information to the user through a website. A user will be able to search for particular characteristics to help narrow down the available auctions. They will also ultimately be the final say over which auction they choose. Images of each auction will be displayed to the user, which enables them to make a final judgment call. A stretch goal would be to use machine learning to eliminate or promote a subset of auctions based on the images provided.

2.3 Definitions

Web Crawler - a program that systematically collects information off of the web. This can be used to compile information or data without requiring a human to manually navigate to each web page.

Website UI - A privately or publicly hosted website which will use an internet browser to display information to a user. It serves as a way for humans to easily interpret information that is being collected and calculated upon in the back-end.

Wrecked Car Auctions - Insurance companies will commonly create auctions to sell cars that have crashed or been badly damaged. The insurance company will pay out an insurance holder, or replace their car. Vehicles are commonly auctioned because insurance companies avoid repairing them due to high or variable repair costs.

Highest Value Car - The cars being sold at auction have some intrinsic value. This could be the grand total of each individual part after labor has been accounted for. It could also be the total price of the car after necessary repairs have been performed. Since the cars are sold at auction, this price is more variable based on bidders' actions. An auction would be worth bidding on if the total value is greater than the current asking price. Since there are so many auctions, the best would be displayed first.

3 System Requirements

A publicly hosted Auction Hunter website that displays to the user the projected value of each auction. There will also be a way for the user to sort the list of auctions to refine their search. When a user finds a high value action, they can obtain additional information such as images and a link to the original auction listing.

This website will utilize a number of back-end scripts which crawl through wrecked car auctions to pull all relevant data. Once the data from each auction is collected, we can predict relative values for each auction and add it to a database to be displayed to the user.

To better judge the effectiveness of our algorithms, we can compare the value that our scripts assign to each car to the final sale price. We can also manually evaluate our predictive algorithms to verify it is assigning value correctly. We will plot any relevant performance data to present the effectiveness of our algorithms.

3.1 User Stories

- As a user, I want to be able to access Auction Hunter from anywhere.
- As a user, I want a user interface that is easy to use.
- As a user, I want to sort the car list by selecting price range and present or absence of VIN.
- To save as much money as possible, I want to make sure that I am getting a good deal on wrecked car auctions, especially for newer vehicles.
- As a user, I want to be able to set alerts when there is a car I want up for auction
- As a mechanic, I want to have proof that Auction Hunter is accurate in it's value predictions so I can buy parts while saving money. I also want proof that it works faster than manually evaluating auctions.
- Auction Hunter should be able to scrape Auction data from IAAI and be extendable to scrape other sources.

3.2 Functional Requirements

At its core Auction Hunter is a website that allows users access to information it has stored in its database. The website should allow for the following functions to be preformed.

- User access controls to only allow signed-up users access to the Auction Hunter
  - Allow users to sign up to Auction Hunter
  - Let users sign in using their email and password
  - Reset their password if a user forgets it
- Account page to manage user preferences
  - Lets user change email and password
  - View notifications Auction Hunter has sent them about auctions
  - Specify parameters about what types of auctions they want alerts about
- Details page that display all information about a specific auction
  - Pictures of car
  - Estimated value of car calculated by Auction Hunter
  - All other data that Auction Hunter can find on auction websites

Auction Hunter also needs a way of gathering the auction data to power its website. This is done via web scraping scripts that run at automated times.

- Scrape IAAI website for basic auction data
  - Vin
  - Damage information
  - Start Code
  - Presence of key
  - Link to original posting
  - Airbag and crash data
  - Photos of vehicle and damage
  - Options and specifics of the car
- Scrape auction websites for when auctions are happening in the future

3.3 Performance requirements

Response Time: All of the response time are measured in end user environment(the browser rendering the response)

- Below operations must respond within 5 seconds.
  - Access action hunter homepage
  - Sign up
  - Log in account
  - Log out account
  - ”Viewing the details of car”
  - Set the values of notification for coming cars
  - Sort the list of auctions by price, value, model, VIN, etc

Workload: The table below estimates that for a user scenario how many requests per day.

Ref No	Scenario	Pages	Daily Total(estimate)
1	Visit homepage	Portal	100
2	Sort car list	Portal	100
3	Set notification	Login,Notification	10
4	Scrape auction data	Portal	80
5	View notification	Login,Notification	10
6	View bid	Login,Bid	5
7	View car details	Portal,CarDetails	90

3.4 User Testing

As a stretch goal, we could get some users unaffiliated with this project to try out our final website to provide feedback.

4 SCHEDULE

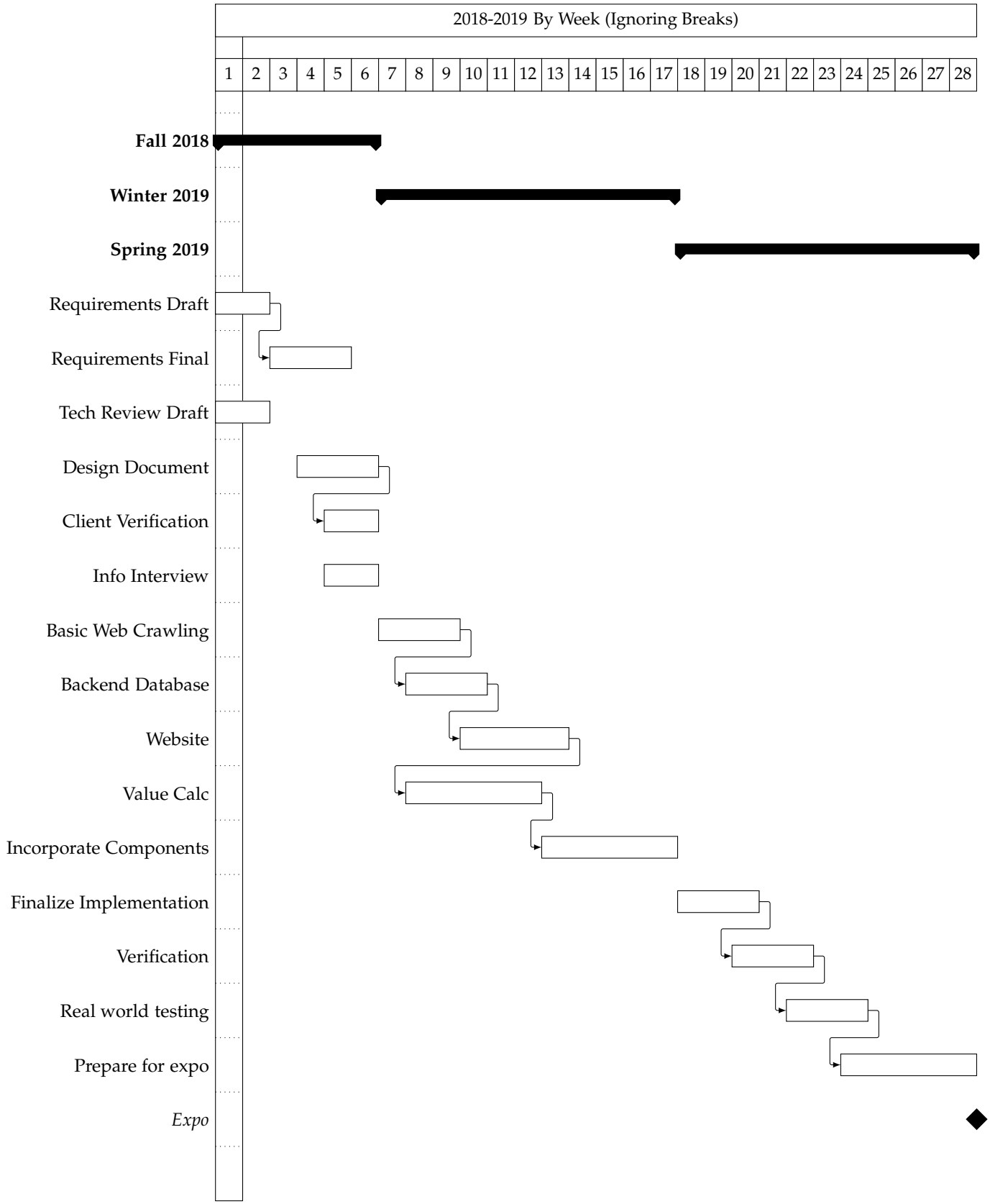
2018-2019 By Week (Ignoring Breaks)
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
Fall 2018
Winter 2019
Spring 2019
Requirements Draft
Requirements Final
Tech Review Draft
Design Document
Client Verification
Info Interview
Basic Web Crawling
Backend Database
Website
Value Calc
Incorporate Components
Finalize Implementation
Verification
Real world testing
Prepare for expo
Expo

Fig. 1. Example of salvage car to be auctioned

Oregon State University
College of Engineering

CS Capstone Design Document

April 19, 2019

Auction Hunter

Prepared for

Ryan Kalb _____ _____
Signature *Date*

Prepared by

Group 4
Auction Hunter

Alexander Hull _____ _____
Signature *Date*

Alexander Jacobson _____ _____
Signature *Date*

Yufei Zeng _____ _____
Signature *Date*

Abstract

This design document outlines the components of our Auction Hunter development project. The elements, relationships, and design concerns will be discussed for each. This will serve as a road map for the remainder of this project.

CONTENTS

1 CHANGE LOG

Section	Original	New
Title Page	Custom Title Page	Used default CS capstone title page formatting.
2.3 - Context	Mentioned Teslas specifically	Removed all mentions of Teslas. Replaced general ”vehicles”

2 INTRODUCTION

2.1 Purpose

The purpose of this document is to outline the design choices for each component of our proposed product.

2.2 Scope

Auction hunter will pull data from one or many salvaged car auctions, store and process the data, then intelligently display evaluations through a web interface.

2.3 Context

Every month hundreds of crashed vehicles get put up for auction by insurance companies in the United States and abroad. Quantifying the potential value of a wrecked car is tedious; each must be manually evaluated. Auction Hunter will help users gather information about the value of salvage vehicles, and provide recommendations. Auction Hunter seeks to watch as many auctions as publicly available and create pricing trends across different types of crashes. By applying previously gathered data to new auctions, a prediction can be made on the expected return on investment of these wrecked cars. The program will iterate through auctions, gather data and images, then populate a UI with relevant information. This will allow the users of Auction Hunter to better understand the real value of a used car and even assist in buying one for a fair price.

2.4 Definitions

- Web Crawler - a program that systematically collects information off of the web. This can be used to compile information or data without requiring a human to manually navigate to each web page.
- Website UI - A privately or publicly hosted website which will use an internet browser to display information to a user. It serves as a way for humans to easily interpret information that is being collected and calculated upon in the back-end.
- Wrecked Car Auctions - Insurance companies will commonly create auctions to sell cars that have crashed or been badly damaged. The insurance company will pay out an insurance holder, or replace their car. Vehicles are commonly auctioned because insurance companies avoid repairing them due to high or variable repair costs.
- Highest Value Car - The cars being sold at auction have some intrinsic value. This could be the grand total of each individual part after labor has been accounted for. It could also be the total price of the car after necessary repairs have been performed. Since the cars are sold at auction, this price is more variable based on bidders' actions. An auction would be worth bidding on if the total value is greater than the current asking price. Since there are so many auctions, the best would be displayed first.

3 STAKEHOLDERS

Our project stakeholders are Ryan Kalb, Oregon State University, our CS Capstone group 4. Ryan Kalb first proposed the product and basic design, so he is the primary project stakeholder.

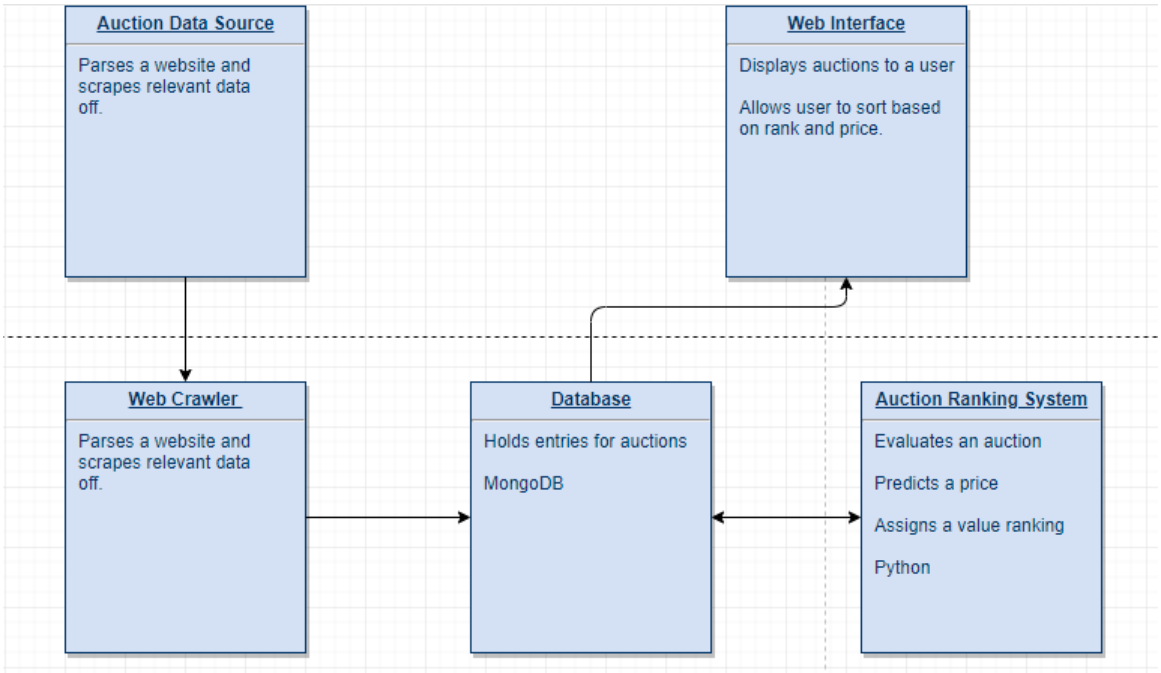

Fig. 1. Flow Design

4 Design Components

4.1 Database Viewpoint

4.1.1 Design Description

Our database will be implemented with the MongoDB database program. [1] Each entry in the database will contain information on car from the auction website. Every relevant piece of information that is available on the website will be added to the database. Later, we can search though the database with a similar call:

```
db.collection.find( { price: { $gt: 10000 } } )
```

Fig. 2. MongoDB 'find' example

This will return a structure with all the database entries with a price greater than $10,000.

4.1.2 Design Elements

There are two main elements to the Database design. The first is the database populator. The second is the database searcher. There is no need for the database to be updated after it has been populated, unless new entries are added for new auctions. However, our implementation could eventually extend to track auctions over time. In this case, we will add a list of variables such as price. This allows for static parameters to remain in the same database entry. For price, a list of price could be added which contains all updates to the price.

4.1.3 Design Relationships

The database populator will need to be closely coupled with the web crawler. The information scraped by the web crawler will need to be reconstructed into a document, then sent to the database with a MongoDB call.

```
db.inventory.insertOne(
   { item: "Tesla Model S", price: 12500, tags: ["bumper damage"] }
)
```

Fig. 3. MongoDB 'insert' example

The database searcher will need to be coupled with the auction ranking system as well as the web interface. Once all the information has been scraped off the auction websites, the local MongoDB database is the only thing the auction ranking system will need to have access to. The website will also access the database information, but might also link directly to the original website or images.

4.1.4 Design Rationale

Modularity will make this project much easier to implement. By separating the populator and the searcher, we are able to split up the project into logical pieces. Since the web crawler only has access to the populator, and the auction ranking system only has access to the searcher, we enforce a strict process flow.

4.1.5 Design Concerns

One concern is how we will implement a database that tracks the lifecycle of an auction. As stated earlier, there could be a list structure for dynamic parameters such as price to track over time. However, this would also require that the web crawler continually scrapes the website every time interval. This poses another issue where changes on the website could happen within minutes, especially when an auction is about to close. To continually check the website for updates, comparing the current parameters to the ones stored in the database, adds complexity to this problem. This issue might cause this implementation to exit the scope of this design.

4.2 Website Viewpoint

4.2.1 Design Description

The only and biggest portion of Auction Hunter that users will interact with is the website front end. Its job is to display the results of the web crawler and damage estimator. It also has to allow users to update and manage their preferences such as login information and auction notifications. The front end will actually be made up of two parts: client side and server side. Auction Hunter will use HTML/CSS and JavaScript to designed and build the client side code. The server side will be written in Python to receive the API calls from the client side. The client side code is what will make the site look nice, and be functional while the server side only has to respond to REST API calls and return the appropriate structured data to the client side for it to display to the user. When a user navigates to Auction Hunter in their web browser the client side code gets downloaded to their computer or mobile device and then is rendered in the browser. The client side code will then make requests to the server for different parts of data stored in the Auction Hunter database. The Python server side code receives those requests and the queries the database and returns the appropriate results. The server side code is in charge of making sure the client is authorized to see the data and is not attempting to do something malicious like delete data that doesn't belong to that specific user.

4.2.2 Design Elements

There are several pieces to the front end: the client JavaScript, the client HTML/CSS styling, and the server side Python backend. The HTML/CSS styling is what will make the site look nice and be easy to use. While the JavaScript and Python will work together to make the site functional.

4.2.3 Design Relationships

The client side code will be served by a webserver hosted on the dedicated server Auction Hunter is using for hosting. The webserver will also act as a gateway to the Python server side code which will in turn make database queries to the mongoDB database also hosting on the dedicated server.

4.2.4 Design Rationale

The reason for splitting up the design and the functionality is so that the Auction Hunter team can work on those pieces independently. Once we get a basic working layout we can focus mostly on functionality. Once the core functionally is finished we can go back and update the styling to make it look nicer and be easier to use. JavaScript is required any time your doing something in a web browser as its the only scripting language that runs natively in all major browsers. Python is being used for the backend because the rest of Auction Hunter will be written in the same thing and the Auction Hunter team members are most familiar with it.

4.3 Web Crawler Viewpoint

4.3.1 Design Description

For collecting cars data, we need to find a method to collect the photo, bidding information, and winning bid amount of cars, and which are listed on eBay Motors, ADESA, Auction Auto Mall, Dashub, A better bid, Salvage bid, Smart Auction, OVE.com, Auto Trader, and Insurance Auctions USA Inc. Auction Hunter will allow users to integrate this information in one place. The potential solution is using website crawler technology to collect the relative web images and data from the websites that were mentioned before.

4.3.2 Design Elements

Web crawler is a kind of script which will follow a special set of rules to automatically gather information from a website. Web crawling is also called web scraping, or web spidering. It is used for web classification in World Wide internet. All kinds of search engines use web crawler to provide valuable searching results. In fact, web crawler collects some special HTML images or text contents or some hyperlinks from other website, and view them in an appropriate way. The scraping mainly includes two steps: The first one is that looking for and downloading web pages systematically. The second one is that extracting information from web pages we got in advance.

4.3.3 Design Relationships

The data web crawler extracted will be reconstructed in some ways, and stored in database for future use.

4.3.4 Design Rationale

Scrapy is a twisted-based Python frame work for web crawling .It can be used to parse or extract HTML,XML,JS and CSV style documents. The main function of it includes: Requests manager, Selectors and Pipelines. For pipelines, once we got the data from website, we are able to pass them through different pipelines. Such as storing the scraped data,

cleaning the HTML data, and validating data. For selectors, it is Scrapys own mechanism, and it is used to extract data. Selector works by selecting the selected parts of HTML files which specified by the expressions of XPath and CSS.For requests manger, it is responsible for downloading pages, and it will work behind the event at the same time. That is the reason why Scrapy costs less time than other web crawler. Scrapy is compatibly with some complicated crawl scenes. For example, for Auction hunter, we need to crawl many pages from different website. Using Scrapy will be a good choice.

4.4 Web Hosting Viewpoint

4.4.1 Design Description

Auction Hunter is going to be hosted on a dedicated server located in one of the team member's houses. This server hosts several virtual machines (VM) one of which will be dedicated to hosting Auction Hunter. Auction Hunter will run on the latest Ubuntu LTS release for familiarity to all of the Auction Hunter team members. Inside that Ubuntu VM will be several Docker containers for the various parts of the website. Each part of Auction Hunter will be put into a Docker container and linked together using docker compose to form a complete working product.

4.4.2 Design Elements

The hosting consists of VMware's ESXi as the hypervisor running on the bare metal server with a Ubuntu LTS VM. In the VM Auction Hunter will use Docker to containerize each part of the system. There will be several containers including one for the front end, one for the backend, the web crawler, and the database.

4.4.3 Design Relationships

All code written for Auction Hunter will need a place to run. The dedicated hosting provided by one the team members will run every part of Auction Hunter, from the continuous integration system to the front end web server. All of these will live in Docker containers or as their own VM on the server. Each of the containers will be linked using Docker's networking features with the Nginx web server being the only end user facing portion. The web crawler will also access the internet to gather data but it won't be directly accessible by any of the users of Auction Hunter.

4.4.4 Design Rationale

The main reason for using dedicated hosting is it gives the Auction Hunter team the most control over how the different portions of the project interact. It also is the only completely free option, as the server is already online and in use.

4.5 Auction Ranking System Viewpoint

4.5.1 Design Description

The auction ranking system is what will provide a best prediction for value, and which auction a user should focus on. Without this component, this project is simply a compilation and reorganization of auctions. The primary purpose is to generate a more realistic price prediction than the one listed on the website. The difference between the prediction and the listed price can allow a user to pick the highest value auctions. Additionally, some factors may matter more for a certain user, so tuning the value prediction based off those factors will provide a more realistic evaluation for that user.

4.5.2 Design Elements

The Auction Ranking System will be implemented in python, and will require pulling data from our generated database. For each parameter stored in the database, a predicted value will be modified on the fly. For example, a single deployed airbag may subtract $500 from the predicted price. The precise numbers will have to be tuned. A good starting point would be to attempt to predict a price as accurately as possible to the listed price based on all information from the website. Next, the tuning would be changed to favor certain parameters over others. The next element would be to send this information to the web view. This could be performed by adding a new entry to the database, or create a new database with relevant prediction information. An id would have to be created for both databases to be able to match entries between the two tables.

4.5.3 Design Relationships

The auction ranking system will need interface primarily with the database created by the web crawler. This is where all the data comes from to feed into the algorithm. The second relationship is between the auction ranking system and the web interface. Tentatively, the auction ranking system will need to write to the database with its prediction values and rankings, to later be displayed by the website module.

4.5.4 Design Rationale

The auction ranking system will be written in Python because it interfaces the best with the other components. We use data to feed into the algorithm entirely from the database to make our implementation more modular and simplistic. Our stretch goal will be to track the entire life cycle of the auction, and compare our initial prediction to the final sale price.

4.5.5 Testing

To test our auction ranking system, we will perform value predictions on a set of auctions. Then, the predicted value can be compared to the asking price. The standard deviation can be found, which will give an indication of the accuracy. Predictions that are far from the asking price can be investigated further. If our auction ranking system is able to without flaw predict the same value as the asking price, there isn't much value to our product. Therefore, certain parameters can be weighted more heavily, and we can run the same value predictions over again and compare the old vs new predicted values. Some manual evaluation of the prediction will be necessary, which will help improve our algorithm. Our client Ryan Kalb should be capable of manually evaluating the price of an auction and then compare to our value.

4.5.6 Design Concerns

A concern is how to send the result of the auction ranking system to the web interface. There are a number of ways to perform this, but they all break the clean modularity and sequence flow in place.

4.6 Damage Estimation Viewpoint

4.6.1 Design Description

As the bonus part of Auction Hunter, the purposes of developing damage estimation tool include that many consumers do not know the rough budget of repairing cost, and the cost of traditional manual screening method by machinist is too expensive and too ineffective. We hope to develop an EXUSCRC like accompanying tool on Auction Hunter to estimate the repair cost.

4.6.2 Design Relationships

The data damage estimation tool obtained will be reconstructed in some ways, and stored in database for future use.

4.6.3 Design Rationale

On EXUSCRC site, user is asked to select the vehicle type by clicking a drop-down list first, because the vehicle type will determine paint cost, scratches repair cost, and dents repair cost. The second step is to select damage part by clicking areas on an image which shows damaged positions(front bumper, hood, right/left front bumper corner, right/left fender, roof, trunk, rear bumper, right/left rear bumper corner, right/left rear corner bumper, right/left quarter panel, right/left front door, right/left rear door ,right/left front wheels, right/left rear wheels). Different damaged part of car will cause different cost. After selecting repair areas, the last step is that user is asked to answer if there are holes or dents existing on each damaged area, and their size. The above three steps can be done by anyone without car-repair knowledge in less than 2 minutes. After user finish the estimating, a repair estimate cost report will show immediately. In addition, the copy of image with damaged areas and cost report will be stored and attached to the corresponding car's data on Auction Hunter site as reference for other customers. Everyone is free to modify damaged areas and revise cost report for the specified car. It means everyone who visit our site is a potential contributor for damage estimation tool.

4.6.4 Design Concerns

The damage estimation tool did not count frame damage, however this kind of damage will significant increase repair cost. This tool only supports to estimate minor collision repair cost, and it cannot be used to estimate some invisible damage, like mechanical damage or salvage.

4.7 Auction Data Source Viewpoint

4.7.1 Design Description

The auction data source is where the auctions will be sourced from, and what the web crawler will be scraping data from. To create an effective tool, the maximum amount of information must be gathered from the website. Without sufficient information, we will have to rely more heavily on the users decision making. Considering this, IAAI.com [2] appears to be the best option

4.7.2 Design Relationships

The choice of website will directly affect our implementation of the web crawler script. A stretch goal is to extend the functionality of the web crawler to other websites to be able to compile auction listings from multiple sources.

4.7.3 Design Rationale

IAAI has much more information than its counterparts, such as Copart [3]. There is information on where all damage takes place, as well as airbag status. It also has all data organized in plain text in one place, which will make web crawling much easier to implement.

4.7.4 Design Concerns

Using multiple data sources could pose a number of issues. One issue is since there is different information available from each website, it would be hard to make a 1:1 comparison between entries from two different sources. In addition, there are likely auctions that are posted on multiple websites.

4.8 Collaboration Tools Viewpoint

4.8.1 Design Description

The code repository can be seen like a file archive. The purpose of using code repository is to share code, and make a better software. A good code repository allows us easily version and share code with other developers,and a good collaboration software will raise our working efficiency.

4.8.2 Design Elements

We are going to use GitHub, Slack, and Discord to be our code repository and communication tool during this whole design cycle.

4.8.3 Design Rationale

The reason we are using GitHub to be our code repository is that When we are working on building Auction Hunter, we will produce many functions in progress. So we need to create a branch for the project, and any change we make wont affect the master branch. We can feel free to commit changes on the branch.In addition, GitHub allows developer to track the changes done, we can easily find who made the change and when he made the change. And we are able to control the versions of project. If the newest version cannot be compiled, we can rollback to older version. Also, the function SCM of GitHub allows us to store the code in the same file without conflict. The reason we are using Discord to be our communication tool is that it supports text chat, voice function, and file sharing at the same time. Slack is used to communicate and recieve announcement from our client.

4.9 Continuous Integration System Viewpoint

4.9.1 Design Description

The continuous integration system is a piece of software that allows for Auction Hunter to be tested and validated as it is being build. Every time code is pushed to the Git repository, in Auction Hunter's case, Jenkins will download it and run automated testing on it. Depending on the desire's of the developers it can also automatically deploy the code into the development or production environments. As team members start writing code, the Auction Hunter team will enforce a strict regiment of automatic tests. These tests will be created as features are added to the project to make sure any new additions don't break the whole system from functioning. Since there are many different parts of Auction Hunter each will have its own Jenkins project to contain the tests. Jenkins will be linked to our git repository so that all of the tests run whenever a developer creates a pull request. After a pull request is successful, Jenkins will then deploy the code to our development server to see the changes live. After manually testing the development version, developers can then create a pull request to the master branch which, after a successful pull request, will deploy the new changes to the production environment.

4.9.2 Design Relationships

Deploying Jenkins onto the dedicated server is the main piece of the continuous integration system. It will need to be connected to the git repository where the Auction Hunter code is stored. It will also need to be able to talk to the Docker daemon that manages the Docker containers. In the git repository developers will need to create tests for Jenkins to run. Using Jenkin's built in Configurator, the Auction Hunter team will need to set up when and how the tests and developments are run.

4.9.3 Design Rationale

Jenkins was chosen because continuous integration systems are very similar in their feature set and the team member of Auction Hunter are already familiar with this one. Setting up a continuous integration system is a complicated task and is different for every project, despite having some knowledge of how Jenkins work it will still be a learning process to get everything correct. Having some familiarity will save the team time at the beginning of the project so they can focus on coding and not messing around with tools.

REFERENCES

[1] MongoDB, “What is mongodb,” https://www.mongodb.com/what-is-mongodb, 2018, accessed: 2018-11-9.

[2] IAAI, “Faq,” https://www.iaai.com/Support/SupportFaq.aspx, 2018, accessed: 2018-11-9.

[3] C. Inc., “copart,” https://www.copart.com/, 2018, accessed: 2018-11-9.

Auction Hunter

Technology Review Draft 1

CS 461 - Fall 2018

Alexander Hull

I. Abstract

The components of our Auction Hunter program are: auction data source, auction database, and value calculation algorithms. We plan on acquiring real auction data from popular websites, storing them, performing value calculation, then displaying our results intelligently to the user. There are a plethora of tools that would be capable of completing this task, and this document will serve to discuss and compare options for each task.

CONTENTS

II. INTRODUCTION

To intelligently select the right technology for the job, a number of factors must be considered: team combined experience, ease of use, compatibility with other tools, and supported features. The three components of our project that this document will cover are: Database tool, source of auction data, and auction ranking system. Our project needs to be able to pull data from any number of auction websites, then store that information in a database. Next, we need to perform calculations to determine which auction has the best value.

III. DATABASE MANAGEMENT SYSTEM

We will need to store the data acquired in a database. At its core, a database could be any organization of data. However, there are plenty of tools that abstract away the innards of the database and provide a logical interface. These pre-made tools will make our lives much easier, but not every database management system is treated equally.

A. MongoDB

MongoDB is a free, open source, and cross-platform database program. It is classified as a document-oriented, NoSQL database. Document-oriented means that the data is stored in a format such as XML or JSON. NoSQL means it doesn't follow the conventional relational database functionality. A document-oriented database will likely be the best fit for our means, because it closely matches the format that we will be pulling data from. HTML is easily converted to JSON or

XML. An example of what a JSON file might look like is available in Figure 1. Additionally, there isn't a necessity to have a relational database because we only have a large collection of independent elements. MongoDB will be easy to learn and use because it is compatible with C++, Java, Python, and C#. This means whatever language we choose to interface with, our data will be easily integrated into our database. After checking some of the documentation, it looks incredibly easy to get started with MongoDB. If using python, all that is required is a MongoDB install, import, then call MongoClient(). There is sample code for each language with snippets for simple tasks available on the MongoDB website. This could easily be added to our web crawling script to easily built up our database. It is open source so there is a plethora of documentation and other open source programs that use MongoDB. [1]

```
{"car": {
  "vin": "1234567",
  "color": "Blue",
  "model": "ModelX"
}}
```

Fig. 1. JSON example

B. Couch DB

CouchDB is another open source, document-oriented database program. It is built to interface with JSON documents, which will be an advantage when capturing data from auctions. Unlike MongoDB, it takes a bit more work to interact with the database. To add to retrieve data, a program must make http requests (get, post, head, put). This means that any program that can create an HTTP request can interact with the database. However, it doesn't have some of the easy to use implementations built for specific languages. In terms of adding data, it is fairy intuitive . All a user would have to do is create a JSON structure for an entry, then make a POST request to the database. "Futon" is a built in administrative interface for displaying the database, which would be useful for development. It could be easier to develop and test our program if we can see the data being entered. [2]

C. MySQL

MySQL is a widely used and well documented SQL database. Like MongoDB, it is easily integrated into languages such as python. MySQL has the advantage of using relational features such as keys. It seems that it would be easy to make an SQL call that simply selects all the cars that are white from our database. Finding a use for relationships might be difficult, because each car auction is an individual component. It could be useful if we eventually pull data from a wide array of sources, and we need to associate each auction with the original source. However, we don't necessary need to use the relational features that MySQL provides for it to fulfill all our requirements. As a consideration, the databases class at OSU uses MySQL to teach students. Therefore, our team would likely have the most combined experience working with SQL-like databases. [3]

IV. Auction Data Source

Each auction could be accessed conventionally with an internet browser, but we want to pull all the relevant information out automatically to avoid manually sifting through auctions. We will have to determine which website would provide the highest quantity of robust data.

A. IAAI

This website features IAA auctions for used and salvaged vehicles. We are focusing on Teslas, and there is a wide selection of auctions from around the United States. It is very easy to narrow your search based on a variety of parameters, which can be added to the URL. One great part about this website is it breaks down exactly what isn't working with the vehicle. If there is front end damage and the vehicle won't start, that will be specified on the auction page. It also displays which airbags were deployed, which greatly affects the value of the car. Figure 2 shows some of the information it provides. If we were to over simplify our backend algorithms, we could simply check the number of airbags deployed, and focus on vehicles that had only one deployed. This could give a user a rough estimate for the value of the auction. Ideally, our algorithm will be much more complicated than that, and the more data we have as input the better. Some other information that we can pull from each auction is: VIN, body style, make, model, damaged parts, condition, and lots of pictures of the vehicle. The website also has a current bid price, and time until bidding ends. Theoretically we could track the value of each vehicle, and if we find a vehicle is undervalued and reaching the final bid, we could automatically bid. Looking into the source of each page, it seems easy enough to filter out the relevant information, as it is displayed in the HTML file in an organized way, with relevant class names for each section. [4]

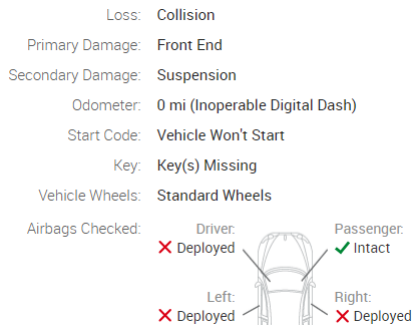

Fig. 2. Damage information on IAAI

B. Copart

Copart is similar to IAAI, with slightly less damage information displayed. Other than that, it has all the similar search and filter features. The layout of the source is a bit more simple, which could make this a good website to start out with, as a proof of concept. There are uniform and high quality pictures of the vehicle that we could pull from the website. There is information about the current bid price, and the final sale date if it has been announced. In terms of the damage information, it has a text entry of the primary and secondary damage. The lack of airbag information is a large disadvantage. There are two similar models of Teslas listed, and the description of damage is the same but the actual damage is very different. One of the Teslas lists side damage, and has a big dent in the side. The other Tesla lists front end damage, and the entire front of the car is demolished. Unless a user evaluated the pictures for themselves, or a very advanced machine learning algorithm was used, these two auctions would be evaluated the same in the database. This website could be a good proof of concept website, and further development could combine multiple websites into one database. The one problem is we could find ourselves comparing lots of information to little, when comparing postings from different websites. [5]

C. Auto Auction Mall

Auto Auction Mall is similar to Copart because there is a somewhat limited amount of information. There is slightly less available information on this website. It simply displays where any primary damage is, and if the car starts or not. One advantage of this simplicity is it seems that there are only a few options for the car's status. With Copart, there was a text entry box where someone entering the information could be a bit more descriptive. Although there is more information available, it could be harder for a program to evaluate if it has to understand what the status means. A disadvantage of this website is it seems to rely heavily on JavaScript, meaning the layout of the HTML file is a bit harder to navigate, and could be difficult to pull information from. [6]

V. Auction Ranking System Language

Our value calculation could be performed with any turing complete programming language, but it is important to consider its compatibility with the other components. Ease of use and team experience are also important considerations.

A. Python

Python has the benefit of easily integrating with many database options. For example, MySQL and MongoDB both have packages which you can include to make working with them very easy. Our group is all familiar working with python, so it should be easy for us to collaborate. Another benefit of using python is there are a number of easy packages to use for the web crawling. Whichever language we end up using for web crawling, we will likely end up using for the backend calculations. One language to interact with the whole backend process will be much easier to work with. Performance isn't much of a consideration for our purposes because we are working with relatively small sets of data, and will likely be bottlenecked by the speed of network connectivity rather than the program itself.

B. C++

C++ is a favorable option because our team has a sufficient amount of combined experience working with th elanguage. It is moderately simple to get C++ to interface with our database system. However, it is nowhere near as easy when compared to python. C++ will be a good choice of language if we find ourselves doing some more heavy calculations on the data provided. Once the program has pulled all the relevant information out of the database, it will be easy in C++ to perform value calculations.

C. Java

Java is somewhere in the middle between python and C++ when it comes to interfacing with a database such as MongoDB. Our group has the smallest combined experience in Java, but one member has comparable experience in Java as with C++. Java serves as a middle ground between C++ and python, and could be a compromise if some of our group prefers python and some prefer C++.

VI. Conclusion

After evaluating these possible technologies, there are clear advantages. MongoDB seems to be the easiest to use with Python. Getting it started is incredible simple, and the document oriented database is just what we need for lots of unrelated

entries. The auction ranking language seems best suited for Python. Python is easy to use, but it also is great at interfacing with other programs, such as MongoDB or a web crawler. Additionally, our group has lots of experience working with Python. For the auction data source, IAAI seems to be the best pick. Although it is similar to Copart in terms of layout and design, it has much better damage information. Our algorithm is severely handicapped if it only has a couple parameters to base the evaluation on. Before a final decision is made, additional research, group approval, and cursory testing will be necessary.

REFERENCES

[1] MongoDB, "What is mongodb," https://www.mongodb.com/what-is-mongodb, 2018, accessed: 2018-11-9.

[2] A. S. Foundation, "Data where you need it," http://couchdb.apache.org/, 2017, accessed: 2018-11-9.

[3] Oracle, "mysql," https://www.oracle.com/technetwork/database/mysql/index.html, 2018, accessed: 2018-11-9.

[4] IAAI, "Faq," https://www.iaai.com/Support/SupportFaq.aspx, 2018, accessed: 2018-11-9.

[5] C. Inc., "copart," https://www.copart.com/, 2018, accessed: 2018-11-9.

[6] A. A. Mall, "Auto auction mall," https://www.autoauctionmall.com/, 2018, accessed: 2018-11-9.

Oregon State University
College of Engineering

CS Capstone Technology Review

December 7, 2018

Auction Hunter

Prepared for

Ryan Kalb

Prepared by

Group 4
Auction Hunter
Alexander Jacobson

Abstract

Auction Hunter is designed to be a website the collects information about salvage car auctions to help users find the right purchase. It tracks past auctions and alerts users of up coming auctions that have cars that meet their criteria. Auction Hunter estimates the value of the car based on base auctions and photos of the damage. Using machine learning and access the auction's photos Auction Hunter tries to estimate the cost of the damage to help users make an educated purchase.

CONTENTS

1 Introduction

1.1 Auction Hunter Background

Auction Hunter has two main pieces that work independently of each other: the web crawler / data processing portion and the web front end. The web crawler continually searches auction sites for data and inputs into a shared database. The web front end then allows Auction Hunter users to interact with its data and set alerts for auctions. This technology review will discuss programming language options for the website, which hosting solution Auction Hunter will use, and what continuous integration software will be used to tie the front and the back-end together.

1.2 Programming Languages

Auction Hunter does not need to use the same programming language for the front and backend because web technologies and trends tend to be outliers from general programming trends. Front end technologies also have different sized stacks ranging from basic HTML generation all the way through full stack deployments with front end and backend components linked together. Within a given language there are also various frameworks Auction Hunter could use to generate a front end. For example, JavaScript has React as well as Angular.

1.3 Hosting Options

The main trade-offs between hosting options revolve around ease of use versus level of control and adaptability. There are hundreds of different options when it comes to hosting a web service and most of them strive for lowest cost and ease of deployment. These two factors are always at odds with control and adaptability because the easier it is to deploy the less control the provider is going to give you. For example, the easiest solution is to pay a provider to host static HTML or even PHP scripts. From start to functioning website is only the time it takes to enter credit card details. The issue with simplicity is that all advanced features are completely abstracted away from the user. Auction Hunter needs to have automatic jobs run constantly which is just not possible with turn-key solutions.

1.4 Continuous Integration Systems

Auction Hunter is a complex project which will require a myriad of different configurations and operations to be performed. A continuous integration system is used to aid developers in maintaining a large code base with lots of different options. The system's job is to take code from a repository and get it running for development, testing, or even final production deployments. Continuous integration systems can also be extended to include automated testing and other more advanced features to help developers spend more time coding and less time trying to get the code to run.

2 Website Programming Language

2.1 Criteria

Picking the programming language is one of the first steps when starting a new project. Auction Hunter has pretty simple criteria for its front-end language. First and foremost, it must be either already known to the developers working on the project or at least straightforward for them to learn. The speed of development and relevancy to the type of project are the next most important. Given the limited amount of time and number of features Auction Hunter requires, faster to write is better than faster to run. Since this project will be used by less than a thousand people (best case) it does not need to be optimized for speed, it needs to be just be working and stable. This eliminates most statically typed languages as they require too much code and memory management to be viable for rapid development.

2.2 JavaScript

JavaScript is an obvious choice being the only language that runs natively in the browser, meaning that any other programming language will either need to be transpiled into JavaScript or supplemented with it. JavaScript is dynamically typed and most developers have at one point written some basic scripts in it. JavaScript is easy to get started with, but can be difficult after getting into more advanced areas due to the looser syntax when compared to other dynamically typed languages [1]. The two main web frameworks for JavaScript are AngularJS and React. Angular is a full suite of software tools including a compiler and package manager while React allows the developer to use whatever tools they want. They are similar with the main trade-off being an all-in-one solution versus a more piecemeal approach.

2.3 Python

Python is dynamically typed and is extremely easy to learn. All the team members of Auction Hunter have already learned Python so it would be a great choice for this project. The downside of Python is that it cannot run natively in a browser and thus must still rely on JavaScript to do client-side operations. One common approach is to use Python or some server-side language to handle API calls to the database and use JavaScript to help render the website on the client side. This is most likely the approach that the Auction Hunter team will take as it allows the use of Python for rapid and powerful development while making it easy to write and manage the user interface in JavaScript. Python's two main web frameworks are Django, which is usually used for larger scale projects, and Pyramid, a more compact but not quite as all-encompassing framework.

2.4 PHP

PHP is a server-side language used since nearly the beginning of the web. Like Python it suffers from the problem of not being able to run natively in the browser and relies on sending JavaScript files to the client to run. Do to PHPs age it has countless web frameworks all of with have different styles and features. Although much effort has been spent to remedy this, PHP still has a reputation of being difficult and confusing to use because of the inconsistency of its language design [2]. The benefit however, is PHP is the go-to standard if you are trying to get a simple site working quickly. It runs on every hosting provider and its basics are simple to learn. Only one of the Auction Hunter developers has used PHP before and therefore is not as great of an option as Python.

3 Hosting Solution

3.1 Criteria

Auction Hunter is a website and thus needs to be hosted on an always online server accessible from the internet. The main criteria for hosting, in order of importance, are cost, flexibility, easy of use, and scalability. The different ends of the hosting spectrum go from shared hosting, which is the cheapest and easiest to use but the least flexible, all the way to self-hosting which can be the most expensive and most difficult to use but offers the greatest flexibility.

3.2 Shared Hosting

Shared hosting is the simplest way to host a website, but it only allows static web pages with simple backend infrastructure. Shared hosting would not be a great option for Auction Hunter because it needs to do more than just serve basic web pages. It requires scripts to run in the background and crawl websites for data without any user

intervention. While shared hosting is the easiest to setup and use, it is too simplified for the more complicated needs of Auction Hunter.

3.3 Cloud Hosting

Amazon's AWS platform and Googles Cloud Hosting are the two front runners of cloud hosting. Both would allow Auction Hunter tons of flexibility and a relatively small monetary cost. However, one of the group members already has their own dedicated server online 24/7 and can host Auction Hunter for free while it is in development. In the longer term, a cloud provider will be a better option as the project will not rely on one single person to pay the bill. In the future, after most of the development has been completed, the customer for Auction Hunter may wish to deploy their own version of Auction Hunter on a cloud provider. In this case AWS is a better fit because it would require only minimal amounts of configuration changes to get Auction Hunter running on it. Google's platform runs Python projects directly but it requires the project to be in a specific format [3]. This means it would require some code changes to get Auction Hunter running on Google's platform.

3.4 Self-Hosting

Self-hosting is the most complicated to setup and maintain, but it is also completely cost free if Auction Hunter can piggyback on a server that is already always online. Self-hosting offers even more flexibility then any cloud offering, but it would not be cost effective to setup a server just for Auction Hunter. The Auction Hunter team has access to an always online server and since the option is available, Auction Hunter will most likely go the self-hosting route because it is free when compared to a monthly cost for using a cloud provider.

4 Continuous Integration Systems

4.1 Criteria

Most continuous integration systems are very similar to one another and choosing one is more dependent on the programming languages it is compatible with rather than any other factor. JavaScript and Python are both interpreted languages meaning they are compiled as they are run instead of ahead of time. Since we are using an interpreted language that does not require a discrete compilation step, picking which piece of continuous integration software to use is mostly a matter of personal preference and language / code repository compatibility. What language and what repository Auction Hunter uses will drive the choice of continuous integration software.

4.2 Jenkins

Jenkins is one of the most common choices with continuous integration software, because of its vast compatibility and impressive flexibility. Jenkins has an extensive array of plugins that expand its compatibility even further. Jenkins' main draws are that it is completely free and very easy to setup and use. Several of the Auction Hunter team members have also used Jenkins before which would reduce the time to setup and maintain it.

4.3 Travis CI

Travis CI is very similar to Jenkins in its compatibility and ease of use. Travis CI's plugin library is smaller than Jenkins' is, but Travis CI has compatibility with most products and technologies that the Auction Hunter team is aware of. One notable benefit of Travis CI over Jenkins is its seamless integration with Github. If the Auction Hunter team chooses to host the project's code on Github then Travis CI would be a good option because of how easy it is to use Travis CI with Github.

4.4 Gitlab

Jenkins and Travis CI are both compatible with Github and Bitbucket, but GitLab has its own continuous integration software built in and therefore is only compatible with itself. Any of these options would work for Auction Hunter as they all support Python and JavaScript. However, since Auction Hunter will already need to host one of the continuous integration systems it might be easier to just host an instance of Gitlab which has a code repository and a continuous integration system rolled into one.

5 Conclusion

The first choice that needs to be made for Auction Hunter is what programming languages the team will use which will drive a lot of the other technology choices. Things like the hosting solution and build system depend greatly on what languages they need to support. A hybrid solution of Python and JavaScript seems to be the best solution for the front-end which will allow the Auction Hunter team to use the same language for the front and back ends. This means that self-hosting will be the best option due to its affordability and flexibility. This will also allow the Auction Hunter team the ability to host their own code repository, continuous integration system, and any other tools that might be needed as development progresses.

REFERENCES

[1] The good and the bad of javascript full stack development. [Online]. Available: https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-javascript-full-stack-development/

[2] Advantages and disadvantages of php frameworks. [Online]. Available: https://medium.com/@mindfiresolutions.usa/advantages-and-disadvantages-of-php-frameworks-c046d50754e5

[3] Structuring web services in app engine. [Online]. Available: https://cloud.google.com/appengine/docs/standard/python3/configuration-files

CS CAPSTONE TECHNOLOGY REVIEW
NOVEMBER 9, 2018
Auction Hunter
Group 4
Yufei Zeng

Abstract

This document breaks the whole project into several pieces tech reviews. It will mainly talk about the crawler: coding language/framework, code repository / collaboration software, and damage estimation engine.

1 Introduction

If someone totals a vehicle, they are in a serious accident and the vehicle is so badly damaged that it is not worth repairing. At that time, the car will be auctioned by insurance auctions.Then dealers and insurance companies will buy and sell the vehicle. However, as a ordinary consumers, it is hard to know if a car is worth to bid, or how to bid at appropriate prices

The main function of Auction hunter is automatically collecting information on a variety of auction sites. So for the current work, the first task is to look for a reliable web crawler to crawl car's data from eBay Motors, ADESA, OVE, and so on.

In addition, during the process of development, we need to establish a code repository to allows us easily version and share code with other developers.

For damage estimation engine,it is a bonus part for Auction hunter, the involved technologies are too advanced for us. For this part, we would prefer to study rather than prove in the real application.

2 Web Crawler

For collecting car's data, we need to find a method to collect cars photo, bidding information, and winning bid amount which are listed on eBay Motors, ADESA, Auction Auto Mall, Dashub, A better bid, Salvage bid, Smart Auction, OVE.com, Auto Trader, and Insurance Auctions USA Inc. Integrating these information to one place. The Potential solution include that using website crawler technology to collect the relative web images and data from eBayMotors, ADESA, Auction Auto Mall, Dashub, A better bid, Salvage bid, Smart Auction,OVE.com, Auto Trader, and Insurance Auctions USA Inc. Web crawler is a kind of script which will follow a special set of rules to automatically gather information from website. Web crawler also called web scraping, or web spidering. It is used for web classification in World Wide internet. All kinds of search engine use web crawler to provide valuable searching results. In fact, web crawler collects some special HTML image or text contents or some hyperlinks from other website, and view them as an appropriate way. The scraping mainly includes two steps: The first one is that looking for and downloading web pages systematically. The second one is that extracting information from web pages we got in advance.

2.1 Scrapy

Scrapy is a twisted-based Python frame work for web crawling .It can be used to parse or extract HTML,XML,JS and CSV style documents. The main function of it includes: Requests manager, Selectors and Pipelines. For pipelines, once we got the data from website, we are able to pass them through different pipelines. Such as storing the scraped data, cleaning the HTML data, and validating data. For selectors, it is Scrapy's own mechanism, and it is used to extract data. Selector works by selecting the selected parts of HTML files which specified by the expressions of XPath and CSS.[1] For requests manger, it is responsible for downloading pages, and it will work behind the event at the same time. That is the reason why Scrapy costs less time than other web crawler. Scrapy is compatibly with some complicated crawl scenes. For example, for Auction hunter, we need to crawl many pages from different website. Using Scrapy will be a good choice.

2.2 BeautifulSoup

BeautifulSoup is one of most popular Python web scraping library. It is able to construct a Python object which based on HTML code's structure. BeautifulSoup provides several easy way to navigate, search, and modify a parse tree.It will automatically converts the form of incoming documents from Unicode to UTF-8.[2] The advantage of BeautifulSoup includes it performs to be compatible with broken or non-standard HTML and XML. So it means that it provides better compatibility. However, comparing with lxml, BeautifulSoup by itself earns a lower scores on the efficiency. It cannot perform well when parsing some extremely large HTML and XML files.

2.3 lxml

The most important feature of lxml is that it is very fast. In addition, lxml supports standards-compliant XML and broken HTML. it includes a module called soupparser which lets it to be able to fall back on the

functionality of BeautifulSoup. It means when users try to solve some broken HMTL or XML, lxml allows them to switch back to parse data from a broken
HTML by using soupparser.[3]

2.4 Conclusion
For Action Hunter, it focus on helping users to gather data about these cars more quicker and more convenient. It provides a series of information to help customers make purchasing decisions. I think Scrapy fits well.BeautifulSoup cannot deal with web side which has a bunch of items, and lxml has less compatible. Scrapy is one of most famous Python scraping libraries. It will help us to crawl a whole domain, and ignore the content type of page. It saves a lots of time for us to crawl the car's data from other web side.

3 Code Repository / Collaboration Software
The code repository can be seen like a file archive. The purpose of using code repository is to share code, and make a better software. A good code repository allows us easily version and share code with other developers & A good collaboration software will raise our working efficiency.

3.1 GitHub
We are able to store many projects in GitHub repositories, such as some open source projects. For GitHub has good help section and guide document for user who is new to GitHub.
For example, when I try to create a branch for Auction Hunter, and I don't know the flow of GitHub. I can simply search GitHub Flow within GitHub. The guide of flow shows that branching is one of most important concept of Git. When we are working on a project, we will got many features in progress. So we need to create a branch for the project, and any change we make won't affect the master branch. We can feel free to commit changes on the branch.[4] In addition, GitHub allows developer to track the changes done, we can easily find who made the change and when he made the change. And we are able to control the versions of project. If the newest version cannot be compiled, we can rollback to older version. Also, the function SCM of GitHub allows us to store the code in the same file without conflict.
The disadvantage of GitHub is that it is disagreeable towards layman. The use of GitHub is more inclined to programmer. However, all of my group members have been working with GitHub for several years.

3.2 Slack
The advantage of slack includes integration, better search feature, accurate online indication, and abundant messaging. User is able to create custom integration by slack. Slack is one of most convenience collaboration software we are using.

4 Damage Estimation Engine
As the bonus part of Auction Hunter, the purpose of developing AI damage estimation engine is that the cost of manual screening is too expensive and too ineffective, and it might make consumers pay more for the vehicle.It is necessary to finding a way to determine the value of salvage vehicles.Consumers need a tool to help them to automatically bid the vehicle. It is necessary to find a reliable tool to estimate damage images of car. However, due to the lack of relative application, estimating the damage of car is well above current level technology, so we will only focus on study for this part.

4.1 **The IBM Watson Visual Recognition service**
The IBM Watson Visual Recognition service analyzes a variety of photos by using learning algorithms. Such as it can be used to estimate vehicle damage. The flow of this service is that user sent the images of car after an accident to public cloud, then server side will take the photos for analysis. Finally, Watson Visual Recognition service will classify the photos, such as a broken windshield, or a broken bonnet, lastly, the results will be returned to customers.[5] It is kind of a new technology, and I had never been seen it before. But it is a open source project,and its code can be found on IBM website. I will suggest a future testing with my group.

4.2 TensorFlow
TensorFlow is using deep learning to estimate the damaged images of the car.

4.3 Datasets of damaged cars

Developers are able to train their model with cars' photos after an accident by accessing specific datasets such as Kaggle UK Car Accidents and Data.gov. and UIUC Image Database for Car Detection. Also those datasets would help us to judge which damage estimation engine is more reliable.

Reference

[1] Scrapy https://doc.scrapy.org/en/1.5/topics/selectors.html
[2] BeutifulSoup https://www.crummy.com/software/BeautifulSoup/
[3] lxml https://lxml.de/
[4] GitHub Help https://help.github.com/
[5] D. Scott(Nov 9, 2018), IBM, https://developer.ibm.com/patterns/classify-vehicle-damage-images/

5 Weekly Blog Posts

5.1 Alex Hull

5.1.1 Fall

Week 4: I have learned a bit more about using LaTeX. I started using overleaf which is significantly easier for quick drafting and viewing changes immediately. I learned in a different class how to use a Makefile for LaTeX as well. I finished my problem statement, and I'm currently working on finishing the group problem statement. I plan to complete the group problem statement tonight. I also want to look into other forms of communication so my group can collaborate on a project at the same time while working remotely.

Week 5: Our team has drafted up the majority of our requirements document. We are now waiting to hear back from our client to get some feedback and/or approval. We plan to complete the requirements document this weekend, after we hear back from our client.

Week 6: Our group recently finished the requirements document. I think we collaborated well for this last assignment and things went well. Our group also got together and assigned portions of the project for the tech review. I plan to complete the tech review today and then enjoy my weekend.

Week 7: I am working on applying all feedback from the tech review peer review. I am nearly complete, I just need to make a few final changes. I am struggling to find relevant and useful citations for my tech review. There aren't too many unbiased discussion comparing the technologies I'm working with. If I go to the MongoDB website, I can only find positive commentary about their product. I plan to finish up my tech review today.

Week 8: Our group has not had very much work to do for our project the last week. One thing I have been working on is getting in better contact with our TA. I have contacted Kirsten Winters with this issue and she has met with the TA. We are running into a problem where it is difficult to get into contact with our TA. We haven't had anything graded yet and we want to make sure that we have met with our TA to go over anything that he ends up grading. I plan to wait a couple more days to give our TA a chance to respond to my message, and a chance to grade our papers. If I find that neither of these are completed, I will have to contact the TA and Kirsten again until it is resolved.

Week 9: We still haven't heard back from our TA, although Kirsten Winters has communicated with us on his behalf. We plan to begin working on the design document soon. Since we are all celebrating the holiday weekend, we likely won't putting a lot of work into it until the Sunday. Thankfully there are large individual components that we can all think about before starting.

5.1.2 Winter

Week 1 Our group has begun to get back into the swing of things. We are organizing a time for us to meet with our TA, and made sure that everyone can make the required class time. One problem I have is I may have a scheduling conflict with our required class time. I am hoping that there isn't a conflict, and I'm continually checking my other event to make sure it doesn't conflict. Our group plans to meet on a more regular basis with our TA than we were used to the previous term. We are starting to think about implementation and how to delegate sub tasks.

Week 2 We have met with out TA, and once with just our group. We have set up a server for us to develop and test code on, and organized out GitHub repo a bit so we all know where to keep our code. We have roughly divided up the tasks for who will manage each part of the project. Once we feel a good prototype is finished, we can reevaluate and plan for future development. We are not entirely sure of deadlines that we are planning on meeting. We plan to all work

individually on our components. We will be working on the poster and elevator speech in the coming weekend.

Week 3 I have personally finished a good starter code for our database component of our project. I created a pull request to be viewed by my teammates. Hopefully this will be useful for them to base their own development off of. One issue I had this week was I forgot to go to our meeting with our TA. I was able to communicate with my team over slack to get all the details from the meeting. I plan to continue perfecting my database component, to make to easier to combine with the other components. Currently, there is a main() function which tests out the other functions that I implemented. I will need to eventually remove the main function so that other python files can call my functions.

Week 4 Made pull request for starting database code. Completed the elevator speech session with group. I plan to create a more concrete plan in terms of implementation due dates. Hopefully this will allow us to avoid procrastination.

Week 5 We have been individually working on our components and have agreed to begin tying together the pieces in the coming week. Our group will plan on piecing together an alpha version of our project by the end of week 6. We are presumably working on midterms at this time so finding time to work could be difficult.

Week 6 Our group has all made individual progress on our components. We completed the poster review, and now have some good notes of pieces to change. We plan to put all the pieces together to create an alpha version over the weekend. Since the progress report has been pushed back a week, it allows us to spend more time putting together the alpha version.

Week 7 We are collectively working on getting our alpha version up and running. We plan to meet this weekend to finalize our alpha and work on the progress report that is due on Monday. I suspect that this will be a multiple day endeavor.

Week 8 We were able to get together as a group and connect the individual components that we have been working on. We are able to pull information off the websites, store it in the database, then display through a web interface. Now, improving each step of the process will be relatively easy, and also simple to test. Haven't had any major problems this week. One 'problem' is a lot of my initial work on the database side ended up being pointless because our scraper, mongodb and web side all interface with each other by default. However, the work I did should make it easier to add some additional complex queries or updates. We plan to improve the UI, and increase the amount of information we provide on each car. I am also planning on creating some rudimentary value prediction algorithms to start displaying and being able to sort on.

Week 9 One problem that we had was that I was sick so I wasn't able to attend the TA meeting. We plan to work on the progress report and our project in the coming week. I personally have less work than normal in the coming week so I should be able to allocate a lot of time to our project.

Week 10 We have begun to work on the data acquisition and auction ranking. Once we put that together we should have a complete beta. No major problems to speak of, other than balancing this project and other classes. We plan to work a lot on the project over the weekend, then work on the progress report and video once we have reached a beta point on the project.

5.1.3 Spring

Week 1 I am starting to work the web scraping, because I think it is one of the more difficult components, and it is a precursor to my database work. Previously we were scraping a single page and we were only getting data from the preview. I'm working on making it so the scraper actually follows the links and goes through all the pages. This will make our data a lot more robust and we will have more of it. I plan to organize a schedule for us to finish up the project

before the code freeze at our TA meeting today.

Week 2 I have been working on enhancing our scraper to also get pricing information from the website. This has been especially difficult because instead of getting all our info from one web page, we actually have to follow links and back out. I also found that the VIN information has been masked on the web page, which makes scraping it difficult. I think they did this to prevent what I am doing. Another problem is the website uses javascript to handle the next page button, instead of a simple link. This is also very difficult to scrape and follow the link. I plan to finish up this task, then make sure our project is ready to go. We will also have to provide instructions to build and run.

Week 3 We submitted the code review, and I starting work on getting the revisions emailed to our client for verification. I just finished the confirmation of the EXPO details assignment.

Week 4 We are currently working on our poster. I was able to keep notes from the poster review, so I am starting with applying those changes. I am still trying to get our client to verify our design document and requirements document. I attended one of the required speakers last Friday, so I only have to worry about the poster and expo for the coming weeks.

Week 5 We have finished submitting our poster. We had to make some last minute changes to make sure there was a picture of us and there were captions. I am still having difficulty getting our client to verify our design and requirement documents. It seems that he is very busy, and also might be disappointed that we didn't finish the stretch goals that we had established at the beginning. In hindsight, we had a lot of lofty goals for our project that were unrealistic. We also didn't establish an open thread of communication with our client, so we were unaware of his plans for the project, and he was unaware of our progress. We plan to talk more with our client to see if we can get this resolved, and if there is any additional work we can put into the project that he would like to see.

Week 6 Our group is trying to meet to discuss what we are going to present for expo. We are discussing whether we are going to have a monitor displaying the front end, or possibly the back end components. I still need to meet with Kirsten to ask her if we are allowed to show anything at expo that wasn't done by us. Since we have an active pull request for an update to our front-end by somebody unaffiliated with our group.

5.2 Alexander Jacobson

5.2.1 Fall

Week 4 This week I worked on combining the individual problem statements into a group version. We haven't had many problems, our client responds promptly, and we are able to stay pretty organized by using slack and discord to group voice chat. Our plan for next week is to work on our requirements document and talk to our client about our problem statements.

Week 5 This week I worked on the requirements document with the team. We haven't been meeting in person, mostly just on discord which has been working well for us. Our client has been a little busy so we haven't gotten as much feedback as expected but I'm confident we understand what our client wants and can articulate it onto the page. Plan for next week is to finish the requirements document and design who is going to take which parts for the tech review.

Week 6 This week my group figured out our requirements for our project. This was mostly painless except for getting everyone working on the same latex document can be a pain. We submitted our draft for the specification buts for our final one our stakeholder suggested some changes which we will make. We also had the tech assignment this week which was hard because we had to come up with 9 different things for people to research. Once we got that all sorted out it wasn't much of an issue to write. Next week is our final version of the tech paper is due so I will need to work on

that. We also have a midterm sometime so need to figure out what that's about.

Week 7 We didn't do a lot as a group this week because we are focusing on our tech reviews which the final version is due on Friday night. So no group progress or problems to report. It doesn't look like we have anything due till the 27th, but that assignment looks quite large, so we will need to meet as a group and plan out how to tackle it. Once we distribute some of the work, we can meet again at the end of the week for a progress update.

Week 8 We finished the tech review final version this week. I had broken my collarbone a couple of weeks back so it has been difficult to type, so I turned my tech review in late. Not sure what we as a team are going to do next week, but I am researching potential tools and designs to use for our implementation phase of the project.

Week 9 We worked on design document by taking all of our individual parts from our respective tech reviews and compiling them together. It turns out that we probably should have coordinated a little better on our tech reviews because some of our individual choices didn't make sense when we put them together with the rest of the project. We are working on the video and final progress update this weekend and next week.

5.2.2 Winter

Week 1 This I setup our development server and build tools so that we could get started developing as soon as possible. We still need to do some more organisation and timelining exactly when each part needs to be done, but so far so good. My group needs to spend some more time planning our term out, but we can resolve that this weekend.

Week 2 This week we worked on planning as a group to figure out what we need to do individually. I then worked on setting up a prototype of what the front end website will look like, and begun building the tool chain to run it. I've also started working on automating the build and deploy tools so that we can focus on developing and not hitting the deploy button a million times. Next week we are going to work on the poster and our elevator speech.

Week 3 This week I worked on getting the build system working. So I can now run two commands that my website back-end and front-end both get built and deployed on a local computer. This isn't quite working for a full production deployment, but that should be soon to come. This also works on the server I have hosted. I can now access an example version of the website remotely hosted off the server. Next week I am going to be working on getting the skeleton of the front-end in place to then build out.

Week 4 This past week I finished with the skeleton of the front-end and now I am going to work on getting each page of the site laid out with fake data to make sure it looks nice and will support all the features needed. There are quite a few different pages, login screen (w/ error messages), home page, detail page, account page, etc...

Week 5 Got most of the individual pages laid out and filled with temp data. Haven't finished them all so that will be a bit of work this week too. After that I am going to work on getting the back-end REST API setup so I can start reading data from the database into the front-end. I probably won't get any individual pages working, just the whole consuming pipeline working.

Week 6 Got the REST API working on the back-end. Having trouble creating a standard front-end consumer pipeline though. I don't have any of the individual pages working, but once I have the pipeline in place I can will get some of the basic page data into the home page and maybe a details page.

Week 7 This past week I got the home page working. So it lists basic data and links to an image of the vehicle. I have a standard way of receiving data from the back-end, but I realized that I am also going to to have to send some data as well from the front-end back to the back-end. So next week is getting sending from the front-end to the back-end working and also getting the details page populated with correct data from the back-end.

Week 8 Sending data is turning out to be kind of difficult with authentication and everything, but I have something hacked together working now. I didn't get the details pages working yet, that will be for next week. I also going to soon need to start integrating with the crawler and the database more directly. We need to be able to get as much data from the external sites as possible, so I can display it all in the front-end details page.

Week 9 Got the details pages worked out fairly well. It seems like we have more information from the crawler progress so that's good. Now I need to work on the user account page some more as well as making sure I can interact with the site effectively (i.e. clicking on a row gives you a detail of the car, the pictures load all correctly, malformed data is handled correctly, etc.).

Week 10 Got the user account page working (mostly) and so users can login and set alerts for types of cars. Nothing is going to send the alert yet, but that's almost done. I have a mostly feature complete front-end and the database has most of the data we will need. Will probably try and clean it up in the future to make it so we don't have strange corner cases appearing in the details page.

5.2.3 Spring

Week 1 Right now we are in the final push to clean everything up and write the build scripts to make it easy to run. This week I mostly focused on packaging my code for "production" vs. "development". This means changing a bunch of flags in my normal scripts and finding a fully fledged hosting solution rather than just the built in web server that Django has. Most of my issues have revolved around getting the new web server working. Next week we are going to work on getting out clients approval on everything and fixing any issues or bugs that he finds.

Week 2 Worked on fixing all the bugs so demo runs without a problem. Still need to write the directions for running the code. Not really any problems other than just lack of time. Never enough hours in the day (yes planning ahead would help alleviate this). We are going to finish everything up for the demo this weekend then work on the document revision next week before its due next Friday I believe.

Week 3 This week I worked on documentation for the code, and getting approval for our modified requirements document. We had some trouble getting the whole team getting everyone in the same place (virtual or otherwise), but eventually got it organized. Next week we have the poster due which we are going to work on, before Friday.

Week 4 This week I worked on the poster with the team. I accidentally missed the meeting where we were going to finalize it, but I reconnected with them later and we finished it all up before it was due. Not sure what else I need to work on for this. I have been adding little features to our site over time to make it better for expo, but other than that unsure what to improve.

Week 5 This week we got our poster done expo is coming up. Still some tweaking and other things we need to do for the project before expo. We also need to get our client to verify our stuff. I've been messaging him, but he hasn't taken a look at our stuff yet. He doesn't seem super please with what we got, but also I wish we got more done too. Next week is figuring out expo stuff and client verification.

Week 6 Not much progress this week. Haven't really talked to the team either, we have been setting up meetings and all bailing on them. Not much else other than picking who is going to be where and when for expo, and what exactly we want to show off.

5.3 Yufei Zeng

5.3.1 Fall

Week 4 We were working on requirements document. And my role is to write the performance requirement.The problem is that We dont know how web crawler will work at this time. We were going to meet next week for Tech Review.

Week 5 I and my group was working on group problem statement this week.We exchange the idea for our project from each one's paper.One of our group member has a bone fracture. So we were using discord to discuss. The problem is that We are still waiting for the response from client for user story part. It is hard to distinguish the project with other common used-car side. We need some more notable features to attract users. We are going to meet again at Sunday for requirements document. We are going to see TA next week for next step.

Week 6 I met my group this Wednesday. We have assigned work for each person for Tech Review. Alex Hull is responsible for database choice, auction ranking system and where to pull auction data form. Alex Jacobson is responsible for website: coding language / framework, hosting solution and build system. I'm responsible for crawler: coding language / framework, damage estimation engine, and code repository / collaboration software. The problem we met is that we don't know if the assigned work fits each other, and if they can finish their work well. So we might change the position later. We are going to meet again on next week Monday through discord to talk about more project.

Week 7 I have done a Tech review peer review this week during class activity. During this event, we exchange ideas each other. I found many good ideas, and I have learned much about how to write a Tech review follow IEEE format. My project group didn't meet this week. Because all of work we are doing is individual. I don't know their progress. I don't know if my the techonology in my tech review fits the project. I look forward next meeting. I will meet with my group next Sunday maybe for design document. And I will continue to know more about the contents on Tech review.

Week 8 We were working on Tech review last week, and we have assigned the task to everyone. Alex J is responsible for website: coding language / framework and hosting solution. Alex H is responsible for database choice, auction ranking system, and where to pull auction data from. I'm responsible for crawler: coding language / framework, damage estimation engine, and code repository / collaboration software.We don't know if the assigned work fits everyone well. So we still need some time to think about this.We are going to meet and work on Design document draft next week.

Week 9 I'm working on my individual work for Design Document: Draft 1. I'm trying to add some proper introduction and conclusion for them. So my group can combine them to one document.I think one part of my individual work(damage estimation engine) which is not suitable for me very well. I'm trying to talk this with my teammates. But all good.We are going to meet with our client through Slack next week for verification.

5.3.2 Winter

Week 1 We all went back to work this week. I'm working on crawler and damage estimation engine for Auction Hunter. Client askes our team to add a track progress and ticketing system. We are going to talk about this during next meeting. We are going to meet with TA every Wednesday 11.59 AM.

Week 2 The first meeting has been completed. We have assigned the task to everybody. I'm responsible for web crawler, Alex Hull is responsible for database, and Alex Jac is responsible for interface. In addition, we have met TA this week, and talk to him about our project. We have to make a detailed schedule for our project. So we can finish each part on time. We are going to meet each Tuesday 6pm, and meet with TA each Wednesday 12pm to talk about the progress.

Week 3 We are still working on Auction Hunter implement this week. We are using github to share and store the code, and I'm responsible for web crawler part.Web crawler is totally new thing for me, and I'm going to use Scrapy open

source tool as the framework. I still need some time to be familiar with it. We are going to have group meeting every Tuesday at 6pm to exchange our research and TA meeting every Wednesday at12pm to talk about project's progress.

Week 4 This week we have completed Elevator Pitch and Poster. In addition, I'm working on my web crawler part for our project Auction Hunter. The open source web crawler I'm using is Scrapy which are written in Python language. I still need some time to familiar with this tool.We are going to meet with other group next Friday for Poster Critique Session. And we are going to meet with our TA to talk about our progress next Wednesday 12am.

Week 5 We have met the other group for poster critique today. We are still working on project implementing. I'm responsible for web crawler. The prototype of Auction Hunter need to be done on Feb 18. We are on a tight schedule.We are going to have group meeting next Tuesday 6pm, and meet with TA next Wednesday 12am.

Week 6 I'm still working on my individual part web crawler. I have been able to scrape some basic elements like the names and images of car from a specific website. The next step is to think about what kind of datas we actually needed. Then download these datas to local, and classify or sort them by different scheme. We need some time to put all things together. We are going to have a group meeting next Tuesday 6pm, and meet with TA on Wednesday 12am.

Week 7 I'm still working on my part web crawler. I have been able to scrape some basic elements(bidding time, car's images, car's informations) from the specific page, and my spiders can automatically loads to next page of the site. I'm trying to figure out how to download the image through url to local.We are trying to piece together everything.We are going to have a group meeting next Tuesday and demo our project to our TA next Wednesday.

Week 8 We are working on improvement our scraping logic. We have been able to intergrade the scraping data into our mongo database this week, but we need more valuable data. We need to figure out how to reconsctruct the scraping data, and make them to be more valuable. We are going to demo our alpha version to our TA next Wednesday, and we have a group meeting on next Tuesday.

Week 9 We are working on interface and improvement scraping for our auction hunter. We have been able to scrape the necessary elements from specifc website like IAAI, and store the data into our database. We are still looking for a method for implementing damage estimate. We are going to meet with TA next Wednesday, and have a group meeting next Tuesday.

Week 10 I'm still working on improvement web crawler. I have been able to scrape some basic elements(bidding time, car's images, car's informations) from the specific page, and my spiders can automatically loads to next page of the site. I'm going to create several spiders for different sources of auction. In addition, the scraped datas have been able to store to our mango database automatically. My teammates are working on interface. We are trying to make the interface being user-friendly, and show everything we got on interface.We are going to have a group meeting and finish our final project video next Tuesday.

5.3.3 Spring

Week 1 I'm still working on improvement web crawler. I have been able to scrape some basic elements(bidding time, car's images, car's informations) from the specific page, and my spiders can automatically loads to next page of the site. I'm going to create more spiders for different source sites.We are trying to make the interface being user-friendly, and show everything we got on interface. I'm going to attend Charles Shaffer: 4/8-Monday/Presentation at 12:30-KEC 1007.We are going to have a group meeting next Wednesday and meet with our TA next Thursday 2pm.

Week 2 We are working on interface and improvement scraping for our auction hunter. We have been able to scrape the necessary elements from specifc website like IAAI, and store the data into our Mango database. But we still need

to scrape price and current bid information(which is stored in different page), and develop several spiders for different source sites.We are going to meet with TA next Thursday, and upload our final project to github before code freeze.

Week 3 We were working on code freeze this week, and we had uploaded the whole working project to Github with readme. We are working on revising requirement doc. Our client still didn't send back verification of documentation ye. We are going to submit our most current document revisions first, and add the verification later. Although we have done for code freeze, we still are going to improve our project. We are going to have a group meeting next week through slack and meet with our TA on next Thursday.

Week 4 After meeting with our TA, we realize that our back-end works well, but there are still a lot of works need to be done in back-end. Such as alerting part, sorting, user preferences, and user account management. We are going to keep working on our project, and we got about 2 weeks before expo. We are going to meet with our TA next Thursday 2pm.

Week 5 We were working on our front-end last week. We were trying to add sorting, alerting part, user preferences, and user account management for it.We try to scrap a large number of data from a specifc sites each time. But the target sites' database are not allowed us to visit so much data. We are trying to solve this problem. We are going to keep working on our project, and we got about 2 weeks before expo. We are going to meet with our TA next Thursday 2pm.

Week 6 We are working on front-end. We have talked to our TA about if we can use an existing front-end from co-worker, and we need to ask Kirsten. We are still trying to add more new functions for our project, such as alerting part, user preferences, and user account management. These are not a requirement, but we just want to make it. We are going to meet with our TA on Thursday 2pm before Expo.

6 Final Poster

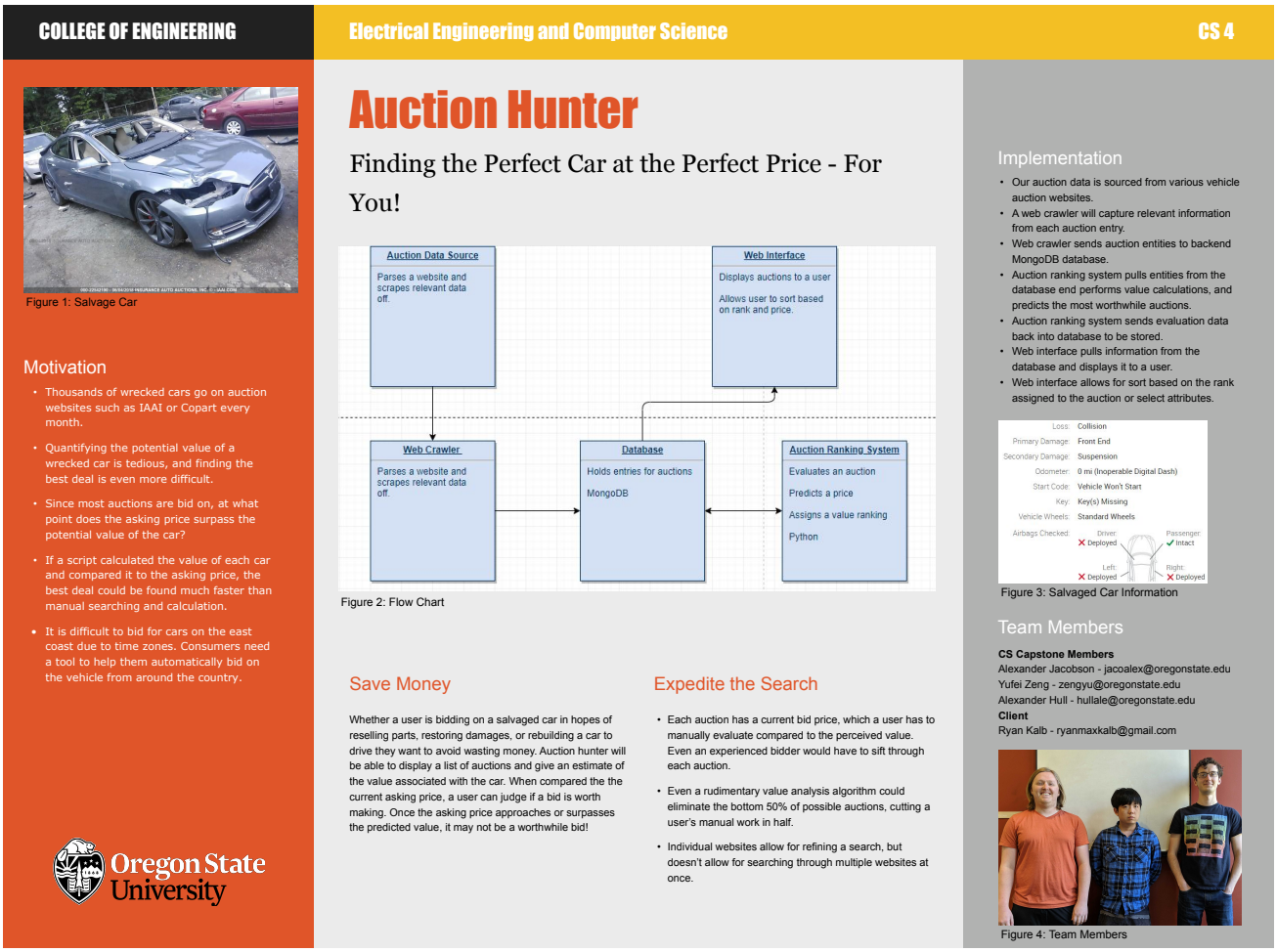

Fig. 1. Poster

7 Project Documentation

7.1 Project Layout

There are three key pieces to Auction Hunter: front-end, back-end, and scraper. The front end is what gets displayed to the user. This was written using React and JavaScript. This connects to the backend via REST endpoints exposed by the backend. This backend is a combination of a MongoDB database and a Python library called Django. The last piece is the scraper, also written in Python, scrapes the front ends of Auction Sites to gather data. For more detail look at the following diagram.

7.2 Installation Guide

The most up to date installation guide can be found in the README file in our Github Repo. The pieces of software need to run Auction Hunter are Python + Django, + Scrapy, NodeJS + React, and MongoDB. Auction Hunter should run on nearly any platform as Python and NodeJS have been ported to nearly everything, however, Auction Hunter

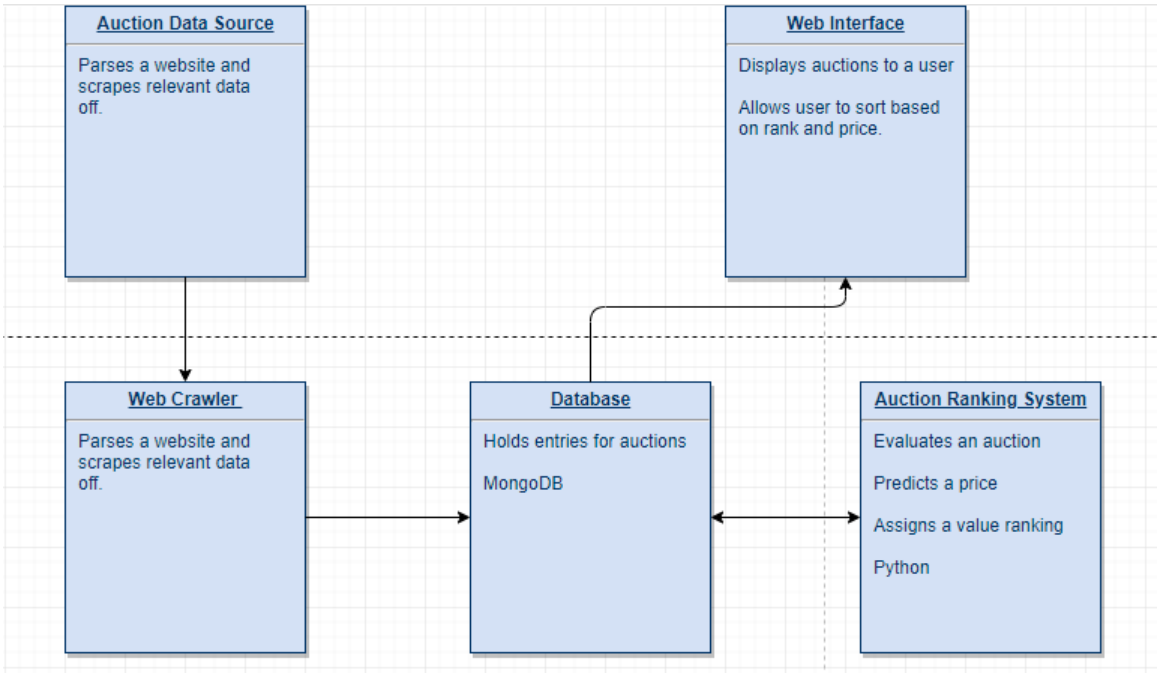

Fig. 2. Flow Design

has only been tested and deployed on Ubuntu and Windows, so other OSes may have bugs. https://github.com/AuctionHunter/AuctionHunter

8 Recommended Resources

- Django Documentation: https://docs.djangoproject.com/en/2.2/
- Django Rest Framework Documentation: https://www.django-rest-framework.org/
- React Documentation: https://reactjs.org/docs/getting-started.html
- Scrapy Documentation: https://docs.scrapy.org/en/latest/intro/tutorial.html
- MongoDB Documentation: https://docs.mongodb.com/manual/

9 Conclusions and Reflections

9.1 Alexander Hull

I was able to learn more about how to work with MongoDB with python. I also learned the basics of web scraping using Scrapy. I learned that there is a lot more to putting together a large project than just coding it. Establishing requirements and technologies is important. Its also a good idea to check in with the current status and the requirements along the way, to make sure the client is aware of the trajectory, and how it matches with the requirements.

In regard to project work, I realized that maintaining a thread of communication between the team members and the client. Having everyone up to date is crucial to be able to collaborate effectively. It's difficult to be in a situation where everyone is self managing. Its a lot easier to have one person oversee the status of the project to keep things moving.

If I were to repeat this project, I would make sure to lot get too lofty with out goals, and also communicate more effectively with my client.

9.2 Alexander Jacobson

With this project I took a bit of what I already knew and expanded on it. I was familiar with Django and JavaScript frameworks, but for this project I taught myself React which is an extensive and quite large JavaScript web framework.

Working together on a team can be quite challenging especially when learning new technologies and the most important thing I learned was to set reasonable and actionable goals each week of the project. This, I have found, keeps the project from overwhelming the team as well as making sure everyone is working on something that pushes the team towards completing the requirements. Its very easy for team members to all start working on different parts that might not come together to meet the initial goals.

If I were to do something different, I think it would be to have set more milestones along the way to make sure we all were making progress.

9.3 Yufei Zeng

Through this project, I learned a lot from both technical information and non-technical information. I learned the basics of web scraping base on Python frame. I was able to scrape the basic elements from any specific website and store them to local or cloud. In addition, I learned the basic of how web crawler Scrapy and MongoDB interact.

Keeping good communication with the team members is essential to promote the progress of project. Weekly group meeting and weekly meeting with TA ensures that everyone will contribute something new for the project.

If I were to do something different for this project, I will communicate more with client, and satisfy client's requirements as more as possible.