CarHub: Final Report

Group 1
Ashley Nelson
Brian Reber
Jamie Kujawa
Josh Peters

Version	Date	Author	Change
1.0	11/28/12		Initial Document
1.1	11/29/12	Brian Reber	Update formatting
1.2	12/2/12	Jamie Kujawa	Added content/Added individual responsibilities
1.3	12/2/12	Brian Reber	Added individual responsibilities, and frameworks
1.4	12/6/12	Ashley Nelson	Added individual responsibilities and definitions
1.5	12/6/12	Brian Reber	Added dependencies, data decomposition
1.6	12/6/12	Ashley Nelson	Added design goals, design issue #1, references, maintenance description, overall module decomposition
1.7	12/6/12	Jamie Kujawa	Added module descriptions. Added UI description.
1.8	12/7/12	Josh Peters	Added design issue #2, and individual responsibilities.

1. Introduction

The purpose of this project is to demonstrate the team's user interface design skills in the form of an automotive web application. This will be a simple, easy-to-use web site that users will be able to interact with to seamlessly manage and find the latest information about their vehicles.

1.1 Purpose

The purpose of this document is to allow a client to have a detailed overview of the project that is being created. The intention of this document is to answer many of the design questions regarding the content that is represented.

1.2 Scope

The scope of this project is one semester consisting of roughly 16 weeks.

The deliverables are as follows:

- Secure user login
- Able to choose multiple cars per user
- Upload receipts/maintenance records
- Gas mileage tracker
    - Shows the previously logged fuel ups and odometer readings
    - Shows average gas mileage for car and most recent fuel ups
    - Shows average gas price paid
    - Track amount spent on vehicle (per year, per month, etc)
    - Shows total amount spent
    - Shows records of recently logged expenses
    - Shows expense breakdown by category (maintenance, gas, detailing)
- Estimate cost of gas for trip
    - Trip Planner
- Alert user to recalls and news related to saved vehicles
- Display user data in the form of graphs and charts
- Nearby gas price information
- Reminders for oil changes and other recurring services

1.3 Definitions, Acronyms, Abbreviations

Term	Description
Google App Engine	Cloud computing platform for hosting databases and web applications
HTML	HyperText Markup Language - Language used for interfaces to

	be displayed in a web browser
Django	Template language used in HTML files to allow for communication between server and views

1.4 Design Goals (based on Deliverables, Functional, Non-Functional, User-Interface Requirements)

1. Reliability
    a. Hosted on Google App Engine
2. Extensibility
    a. Clear separation of modules so it is easy to integrate new ones
3. Response Time
    a. Tried to avoid redirects wherever possible
    b. Use of caches
4. Responsive Design
    a. User interface reformats to accommodate window size
5. Security
    a. Google Authentication for login
    b. Data stored by Google

2. References

- The Python NDB API - https://developers.google.com/appengine/docs/python/ndb/
- Python Documentation - http://docs.python.org/2.7/
- Google News Search API - https://developers.google.com/news-search/
- Google Charts - https://developers.google.com/chart/
- Twitter Bootstrap Framework - http://twitter.github.com/bootstrap/index.html
- MyGasFeed - http://www.mygasfeed.com/
- Google Maps - https://developers.google.com/maps/documentation/javascript/tutorial
- Datatables - http://datatables.net/

3. User Interface Description

The user interface consisted of several HTML pages. Within these pages the user is able to perform various tasks. The goal for this user interface was to make everything very intuitive. Some of the pages that users are able to view are considered to be static meaning that they have nothing to do with who is logged into the site. The other set of pages requires the user to have an account to user.

Once the user is logged in they will be presented with a garage page which consists of a few buttons and table that lists the cars that they own. When a user clicks on a car they are able

to perform more actions such as inputting maintenance records, track expenses and view a news feed bases on their current car. A user can add a new vehicle at any time. The expense manager and maintenance record pages are HTML tables that show all records. To populate these tables the user will have the option to add a record which will then bring up a form. The news feed once again contains several rows of information auto populated based on the car of the current user.

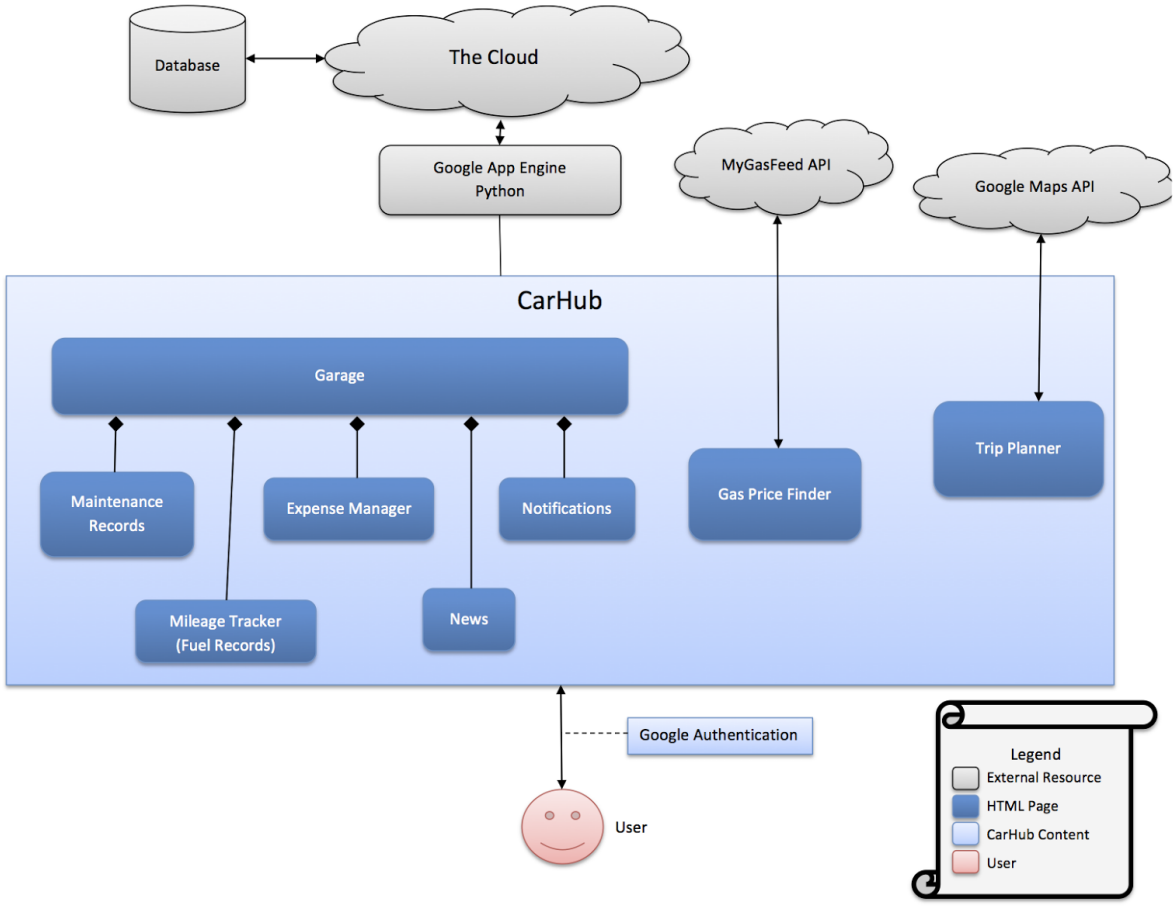
The gas price page of this website is static. A user will have the option to enter a zip code or to use their current location to search for nearby prices. A table will be populated with the results. The trip planner page is also static. A user will be presented with a form input of their current location and destination. A map and directions will be generated upon pressing a submit button.

The entire website has a navigation bar at the top and a footer at the bottom. The navigation bar contains links to many of the items described above.

4. Decomposition Description

4.1 Module Decomposition

The diagram below shows the pre-existing technologies that we utilized in gray at the top, as well as the modules that we designed ourselves which are colored blue and located in the CarHub box. There are two categories within the modules that we developed - the modules that require a user account in order to gain access to them, and the modules that are available to every visitor of the site, regardless of membership. The modules that appear under the Garage module are those that require a user account, because their content is specific to the user's selected vehicle. The Gas Price Finder and Trip Planner, which appear to the right, are in the category available to all users. The user connects to our modules via Google Authentication, which is displayed at the bottom of the diagram.



4.1.1 Maintenance Records

This module allows users to view the records for all reported maintenance that has been performed on their vehicle. The records are split up into different tables depending on their categories, and the tables are listed in alphabetical order by category. Users can add, edit, and delete records from the maintenance page. This module connects to the later mentioned Notifications module to allow for either one-time or recurring reminders for selected maintenance categories.

4.1.2 Expense Manager

This module allows a user to track their expenses and view all of them in one place. The expense manager deals with all user expenses. A table is generated with each row containing a different expense. The table will be sortable and allow for an expense to be updated or deleted. The user will be able to use this page to track all expenses including maintenance and fuel record expenditures. This page will also allow users to log expenses that don't fall under the other two categories.

4.1.3 Nearby Gas Prices

This module allowed users to search for nearby gas prices. Using data powered by myGasFeed.com a table is populated with a list of nearby gas prices. With this information a user is able to view the information about the stations as well as pull up a map of the location.

4.1.4 Trip Planner

This module focuses on letting the user plan a trip. Using the Google Maps API this page allows for input of two locations and will then generate a map with the route and directions from the starting location to the destination location. The price of a trip can then be calculated based on the values populated.

4.1.5 News Feed

This module allows a user to view recent news articles regarding the selected vehicle from their garage. This module uses data returned from the Google News Search API to populate a page of information based on a query of the current car's make and model.

4.1.6 Notifications

This module deals with notifications similar to those of Facebook. A user will be able to opt for a notification to remind them of an event such as scheduled maintenance or scheduled service. A notification tab will then be populated with a list of reminders. Notifications are specific to a vehicle and can be edited. They are directly connected to the maintenance records and are updated and checked whenever a new record is added.

4.1.7 Fuel Records

This module allows the user to view all of the inputted fuel records as well as allow a user to input a new one. A visual chart display of information will be populated with the information. A user can add and delete a record at any time.

4.2 Concurrent Process

There are two main processes that can be running concurrently on our website. The first is the main frontend, which is what the user interacts with. The other process is the vehicle data fetcher that retrieves data from cars.com every night.

4.2.1 Main Web Frontend Description

The majority of our application is based on the interaction between the user and the web frontend. This is the main process that is run while the application is in use.

4.2.2 Cars.com Data Fetcher Description

We built a module that fetches vehicle information (makes, models and years) from Cars.com, and stores this information in our database for use when a user chooses to add a vehicle to their account. This module runs in the background at midnight every day. This runs in the background, and should not affect user experience at all. The user doesn't need to know, and

shouldn't care that this process is running.

4.3 Data Decomposition

4.3.1 BaseVehicle Description

This represents a raw vehicle that is retrieved from cars.com. It consists of a make, a model, and a list of years the vehicle has been made. It is used as a templaye for users to add their own vehicles to their garage.

4.3.2 UserVehicle Description

This represents a vehicle that has been added to a user's garage. It contains a make, model, year, color and license plates. It represents information relevant about a specific vehicle they own. All expenses are tied to a specific UserVehicle.

4.3.3 BaseExpense Description

This represents an expense that a user made on a specific vehicle. It contains a reference to the UserVehicle that it is for, the date of the purchase, the category of the expense, the location of the expense, a description, a total cost and an image link.

4.3.4 MaintenanceRecord Description

A MaintenanceRecord extends BaseExpense, meaning that it contains all of the same fields as a BaseExpense. It also contains one additional field representing the odometer reading at the time of the maintenance.

4.3.5 FuelRecord Description

A FuelRecord extends BaseExpense, meaning that it contains all of the same fields as a BaseExpense. It also contains a field representing the miles / gallon of the vehicle since the last fuel up, the starting odometer reading, the ending odometer reading, the number of gallons of fuel added to the vehicle, the price per gallon and the fuel grade used to fill up with.

4.3.6 UserExpenseCategory Description

A UserExpenseCategory is a class that is used to define categories for expenses for a specific user. They contain an owner, and the name of the category.

4.3.7 MaintenanceCategory Description

A MaintenanceCategory is a class that is used to define categories for maintenance specific records on a per user basis. They contain an owner, and the name of the category.

4.3.8 Notification Description

A Notification is a class that is used to define notifications for a user. They contain an owner, a vehicle, a notification category, whether it is recurring or not, whether the recurrence is date-

based or mile based, the date for the notification, the mileage for the notification, how many days/miles before to notify, and the recurrence for miles or months.

4.4 Overall System Specification

CarHub is constructed so that the user can navigate to each page which is an individual module described above. Each module will allow the user to either interact with the datastore on Google App Engine to store their information and view it, or to get information from an external source and view it (Google Maps, myGasFeed). Any modules that interact with the datastore will use object models described in the Data Decomposition section to represent the data. The static modules will retrieve data from other sources using the APIs that are associated with those outside sources to display the data to the user.

Each module contains an HTML page that uses Django templating to get the datastore objects. Most pages also use javascript and jQuery to make the pages dynamic. The Google App Engine has our python files running on it as URL handlers to load the correct module with the correct data retrieved from datastore. Each page is styled using css and Twitter Bootstrap template.

5. Dependency Description

5.1 Inter-Module Dependencies

Our modules do not have any dependencies on other modules. All modules are views that allow the user to interact with data in different ways. Each module is a separate view, which handles the processing required to perform its specific task.

5.2 Inter-Process Dependencies

N/A

5.3 Data Dependencies

5.2.1 BaseExpense and UserVehicle

The BaseExpense class, and all classes that extend it have a dependency on the UserVehicle class. Based on how our system is set up, a UserVehicle class is necessary in order to have a BaseExpense. If a UserVehicle is deleted, all the BaseExpenses need to be deleted as well since they are unable to stand alone.

5.2.2 Notification and UserVehicle

A Notification object is tied to a specific UserVehicle. If a UserVehicle object doesn't exist, and Notification can't be created for it. Also, when a UserVehicle is deleted, the Notifications corresponding to that UserVehicle must be deleted as well.

6. Interface Description

6.1 Module Interface

All of our modules are websites that are interacted with by visiting a specific URL. The modules don't really interact with each other, but instead just use the same data model and display different information in different ways.

6.2 Process Interface

The two processes that are used are initiated by visiting specific URLs. The processes don't have any interface other than just the web browser or cron job sending a GET request to the proper URL which in turn starts the correct process.

7. Design Rationale

7.1 Notification Implementation

7.1.1 Description

It was difficult to decide the best way of displaying upcoming maintenance notifications to a user. We knew that the notifications should appear in the Notification dropdown menu, but we were unsure of how to determine when a user has viewed them.

7.1.2 Factors affecting Issue

We did not want to have to redirect to the current page when it came to handling the action of the user viewing their notifications, so we could not change the property of notifications that affects whether or not they are displayed immediately after the user views them. We also wanted to make sure the users still had access to their notifications once they viewed them, so we could not simply remove them from the dropdown menu once they were viewed, but we still had to distinguish the viewed and unviewed states from each other.

7.1.3 Alternatives and their pros and cons

- If the notification dropdown is clicked, use JavaScript to store value in hidden input field to let handler know that notifications were viewed.
    - Pros:

        - User would not have to do anything besides view notifications in order to make the notifications disappear
    - Cons:
        - Would need to check input field in every handler for every possible web address within site
- Make notifications into hyperlinks that the user must click in order to acknowledge they were viewed. Notification will redirect to notifications page.
    - Pros:
        - The redirecting is purposeful because it would not redirect to the page the user was already on
    - Cons:
        - User is forced to go to notifications page in order to clear notification indicator
- Refresh page when notification dropdown menu is expanded so notification properties can be set to not display for the rest of the day immediately after viewing.
    - Pros:
        - User is able to remain on same page after viewing notifications
        - Easy to code
    - Cons:
        - User has to experience a redirect when they are not even changing pages

7.1.4 Resolution of Issue

We chose to go with the second option of requiring a user to click on a notification in order to mark it as viewed. This is because it was manageable to code and also gave the user more purpose with the redirecting that needed to occur.

7.2 Record Categories Implementation

7.2.1 Description

One of the main goals of the site it to have the user's data be very easy to find and also group together based on the type of information. This is why we have the expense manager, maintenance page, and fuel records page. We wanted to be able to have categories of expenses that would help to distinguish maintenance for say and oil change from maintenance that like a new paint job.

7.2.2 Factors affecting Issue

We have already split record types into three overarching categories based on the type of record stored. Now the goal is to allow you to group records even within the expense object type. We want these categories to be customizable per user, since each user has different things they want to use CarHub for.

7.2.3 Alternatives and their pros and cons

- Only have the main three record types to group the records by
    - Pros
        - The most simple solution, less confusing
        - User can still enter a description to help them remember the context of an expense
    - Cons
        - Much less ability to group items together
        - Not able to match up notifications well with maintenance records logged. (Ex: notification to have oil change, you log an oil change record of a maintenance object)
- Allow for each record type to have an additional category within it and have the user be able to add and remove their own categories.
    - Pros
        - This gives the user the most power. They decide how specific and comlex they want their records to be.
        - Scalable, this allows someone to store records that can be grouped in three object types , but many more.
    - Cons
        - Harder to implement, needs to have more object models, have a page to edit and delete user categories.
        - It may not be necessary. It could just make it more complex than it has to be.

7.2.4 Resolution of Issue

We ended up having specific categories that the user can add, edit, and delete that can be associated with a base expense and a maintenance record. We don't let the user choose the category for a fuel record because a fuel record does not need to be able to be grouped by types of fuel records. For base expense records there is a set of categories that we have as default categories and the user can add categories also. The maintenance records also have a specific set of categories with some default ones that we have provided and the user can add their own categories as well. We chose to give more power to the user and give them that extra customization and ability to have more usable data.

8. Traceability

No	**Use Case/Non-functional Description**	**Subsystem/Module/classes that handles it**
1	Adding a maintenance record	Maintenance Records, Notifications

2	Viewing a vehicle's charts	Charts, Maintenance Records, Expense Manager, Mileage Tracker
3	Viewing expense manager	Expense Manager, Maintenance Records, Mileage Tracker
4	Planning a trip	Trip Planner, Mileage Tracker
5	Finding nearby gas prices	Gas Price Finder
6	Adding a notification	Notifications, Maintenance Records, Mileage Tracker

9. Language/Framework/Tools Used

When we started working on this project, we chose to use Google App Engine to host and run our code and database. One of our team members had previous experience using AppEngine, and recommended it because of the all-in-one hosting that Google provides. We originally started to work on the project using the Java AppEngine SDK, using JSPs and Java Servlets for our endpoints. We chose this because of our team's familiarity with the Java language.

After our jumpstart presentation, we decided to try out AppEngine's Python SDK, because we were interested in trying out a new language and framework. We were very pleased with this choice, as it provided a very easy to understand project layout, and was easy to figure out.

Frameworks/Libraries used:

- Google AppEngine Python SDK
- Django templating system for Python
- jQuery
- Twitter Bootstrap for the theming
- Python "Next DB" (NDB) library for automatic caching of datastore entities
- Google Charts API

10. Individual Responsibilities

10.1 Ashley Nelson

My responsibilities for this project included three different pages/features. The first page I worked on was the recent news page. It utilizes the Google News API to get a list of the eight most recent and relevant news articles pertaining to the currently selected vehicle in the user's account. This feature can play an important role in making sure users are aware of any recalls or other security concerns involving their vehicles. In order to view an article of interest, the user can click on its link and it will open in a new tab.

The second feature is the maintenance records page, which displays the user's inputted maintenance records in tables separated by category. I worked on the UI for this page as well as the app engine queries for maintenance records and categories.

The last and largest part that I implemented was the notifications feature. This allows users to set notifications to remind themselves of upcoming maintenance that they need to perform on their vehicles. The notifications refresh once a day or whenever a new maintenance record is added, and users have the option of adding and deleting them on a separate page.

10.2 Brian Reber

I was responsible for setting up the project framework, including getting the Eclipse project up and running, the AppEngine project set up, the domain name resolving to the correct place, and other general setup tasks. I was also responsible for the user authentication to the website, which we decided would be handled using the Google AppEngine account authentication system that uses Google accounts to authenticate with. In addition, I set up the "Add a Vehicle" page, which involved getting a list of vehicle makes, models and years. I setup a cron job to automatically fetch this data from cars.com every day at midnight, and store it in our datastore.

I also worked on setting up the page with charts about a specific vehicle. This involved setting up an AJAX API endpoint in our Python code that allowed the page to fetch the relevant information about a vehicle that would be displayed in charts.

I also setup an admin page that allows administrators to delete entities, flush the cache, manually reload car data from Cars.com, and a few other tasks. In addition, a basic settings page was setup to allow a user to change information about themselves. Right now, this just displays the location that allows them to change their profile picture.

10.3 Jamie Kujawa

My contributions to this project consisted of two static pages that could be accessed whether or not a user was logged into the website. The first page was a gas price finder page which used the service from myGasFeed.com to display information about the nearby gas prices. Users have the option to search by a zip code or use their current location. This page had several options to help the user narrow down their search. Also included in this page is a google map that displays the location of the gas station. By clicking on an element in the table of gas prices a modal dialog will appear with a google map that has a pin on the gas station.

The second page that I was responsible for was the trip planner page. The overall concept of this page was to allow the user to plan a route from point A to point B and get directions. Also included on this page is a cost calculator where a user will be able to input the average MPG of their car and the estimated price per gallon. The user will then be able to view the estimated price or their trip.

Towards the end of the project I focused on creating the homepage to make our application have a better visual appeal upon first visit. This page listed all of the features of the application and added some screenshots to give the user a glimpse at what the website looks like.

10.4 Josh Peters

I was responsible for the Expense manager and the Gas Mileage Tracking page. Because I was designing these pages I also helped to design the objects that represent the data and to create a page that will allow the user to add records a vehicle. I also was responsible for the categories feature that is involved in many of the record types.

The Expense Manager page is a page that loads all the expenses that a user logs including BaseExpense, MaintenanceRecord, and FuelRecord. This page allows you to view, add, edit, and delete records. This also would keep track of the total amount spent by the user on that vehicle.

The Gas Mileage Tracker page shows the user all the FuelRecords they have stored. This page computes the average miles per gallon (mpg) based on what the user has logged. This page also allows the user to view, add, edit, and delete these records.

I made a page that allows you to add all the different record types. This page displays differently in order to tailor to the type of record that you are adding. This page also allows the user to add a new category right on the page as they assign it to a record.

I also added a portion of the Settings page that will allow the user to view, add, edit, and delete the categories that the user has created.