**Fuse User Requirements - DRAFT**

**PHASE 1**

* **Quick Start Set-up**

**Load app to phone**

* + 1. Android & Apple App stores

When the user downloads the app from either store, they will need to approve the use of their location services (GPS), calendar of your choice, contacts, car data including diagnostic information, mileage, usage (miles per hour, short stops, fuel levels) by the app.

The EULA will be included in the box with the Fuse Key. Instructions should say something to the effect that by plugging in the device, they are agreeing to the EULA.

**Login or Sign-up**

Once the app is installed and opened, a Quick Start process should appear. The user should be offered both a “Sign-up” button and a “Log-in” (for existing users).

If they choose “Log-in,” they should be given a login screen with their email/username and password fields. A “Forgot Password” feature and a “Remember Login” checkbox at this point.

Once they submit their username and password, they should be dropped into the Main Menu Interface & Menu (section 2 below).

*Note from Phil: Let’s talk about authorization, OAuth, and the Fuse/Kynetx model here. It’s a little different and we may need to subtly change the sign up process to ensure user’s understand that. Much like linking an app to Dropbox, the user authorizes the Fuse app to link to their Fuse personal cloud. We don’t need to telegraph all that, but it changes things slightly.*

If the user chooses *Sign Up*, they should be taken into a Getting Started Wizard where they will enter the following:

* + - * **Email** (validate format in field)
      * **Password** (suggest 8 characters, at least one non-alpha.)
      * **Name** (First & Last separate fields)
      * **Would you like to add photo?** If yes, allow them to choose one from their phone gallery, or their contact list.

Once they click “Submit”, Users should be given a Success Message:

**“Congratulations! You’ve now created a Fuse account.**

**Please confirm your email address in the validation email sent to your inbox.”**

Offer an “Okay” button and allow the user to drop into the app and handle the email validation later. Leverage as much of the Squaretag code here as is appropriate.

Once in the app, present users with the next step:

**Add a Car to Account.**

* + - 1. **Nick Name for Car**

Allow the user to enter their own Nick Name for their car. **Link to Fuse Key**

Next, display user instructions on how to link to Fuse Key (like those that exist). Let’s also offer a link out to any manufacturer information on where the fuse key will plug in based on year/make/model. Let’s also include some photo examples of where the key might be (the four most likely places).

Display success message on app once scan is complete.

**“Success! Your car is now linked to your Fuse App.**

**Keep in mind, it will take about 30 mins of driving for data to appear in your Fuse App.**

**Happy Driving!”**

All other needed info. (like the API keys) should be generated for the user under the covers.

* + - 1. **VIN**

Give users the option to either hand-enter their VIN, or scan it in. Once submitted, let’s validate format on this field using a third party VIN database (http://developer.edmunds.com/).

Per Chris’ note, other devices are reading the VIN from the ODBII device. We need to push Carvoyant to add the VIN to their script. Once they’ve added it, we want to ask the user to validate it. This may not be a phase 1 feature, however.

* + - 1. **Year/Make/Model**

Once the VIN is entered, the app should fill in the Y/M/M for the user.

* + - 1. **Are you the Primary Driver?** Yes/No
      2. **If not, who is?** (Give option to open their contact list and select a name.) Pull the name and photo into the app upon selection.

At this point, the user clicks “Submit” and the account is created.

**Link Additional Cars to App**

Once the initial car is linked, give the user the following options:

**Would you like to add another car?**

**Yes / Not now, thanks.**

If they choose “Yes,” drop them into the car adding form and allow them to add another car nick name, link to the Fuse key, primary driver, etc. (same steps as above). If they choose “Not now,” drop them into the main app interface.

**Send Welcome Email**

Once the account is created and car/s linked, send user a welcome email:

**“Welcome to Fuse!**

**Congratulations on signing up for Fuse – the car management and maximization system. With Fuse you can link to as many cars as you’d like on your own smartphone app, keeping track of a single car, or group of drivers and vehicles.**

**Next Steps?**

Add dynamic content here if:

1) User has not added a car

**Add your car and primary driver to your app.**

**Simply open your Fuse app on your smartphone and add your car with the Getting Started Wizard. The app will also walk you through installing and linking to your Fuse Key.**

2) Linked to their Fuse Key.

**Don’t forget to link to your <insert car year/make/model> to your Fuse key. To do so, simply <Insert instructions here> If you have any trouble, please contact us at: support@joinfuse.com**

3) Then show everyone the generic content below:

**Fill up your gas tank!** – Don’t forget to record your costs in Fuse with every fill up. Over time you’ll be able to track trends and get tips on how to lower your cost per gallon.

**Add Maintenance Dates & Tasks –** With Fuse it’s easy to keep track of oil changes, tire rotations and more. Simply add a maintenance task to your calendar and fuse will remind you, then give you options for service providers, stores, even promotions to accomplish your task.

**Add more cars & drivers to your app** – With Fuse you can manage an entire fleet of cars from total miles driven to keeping track of your teen. Link all the cars in your life to your app in just a few easy steps.

**Have Questions?** Contact us at: [XXXX@joinfuse.com](mailto:XXXX@joinfuse.com) or access our FAQs at <include path to page>.

Happy driving,

The Fuse Team

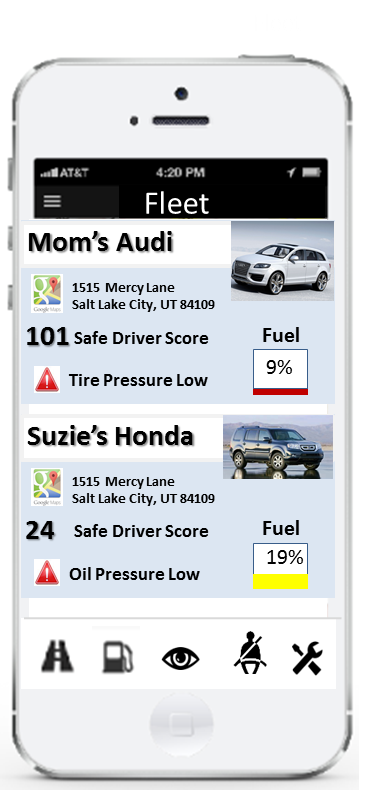
1. **Main Interface & Menu**

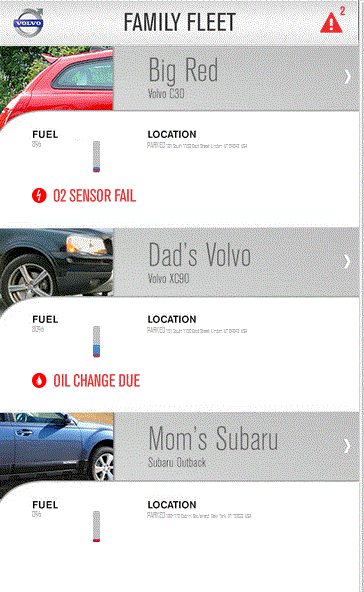
The primary interface of the app should be a quick view list of all cars linked to the user’s app. This list view should include the car’s name and photo (YMM if there’s enough room) as well as the current location (with a link out to a map), the current safe driver score (linkable to score detail), the most recent alert, if any (link to list of alerts), and fuel level (link to fuelsmart). If the user only has one car, only the stats for that single car should be listed.

If the user clicks on the map button or linkable location text, they should drop into a map view where that car appears on the map with an icon or photo of the car (if present). If there is no photo a default car icon in a different color (assigned by the app) should appear for each car. This color should be consistent throughout the app for that vehicle (where relevant). When the user taps the individual icon in the detail-view, the basic vital stats view (like we have now, but let’s add the name and pic of the primary driver) should open.

The user should be able to toggle back to the fleet list view by clicking on the fleet icon in the upper-right. When in the fleet view, no icon is present in the upper right hand corner. To look at individual data, the user will need to select a car off the list.

Below are a couple examples of possible Quick List layouts for the GUI.



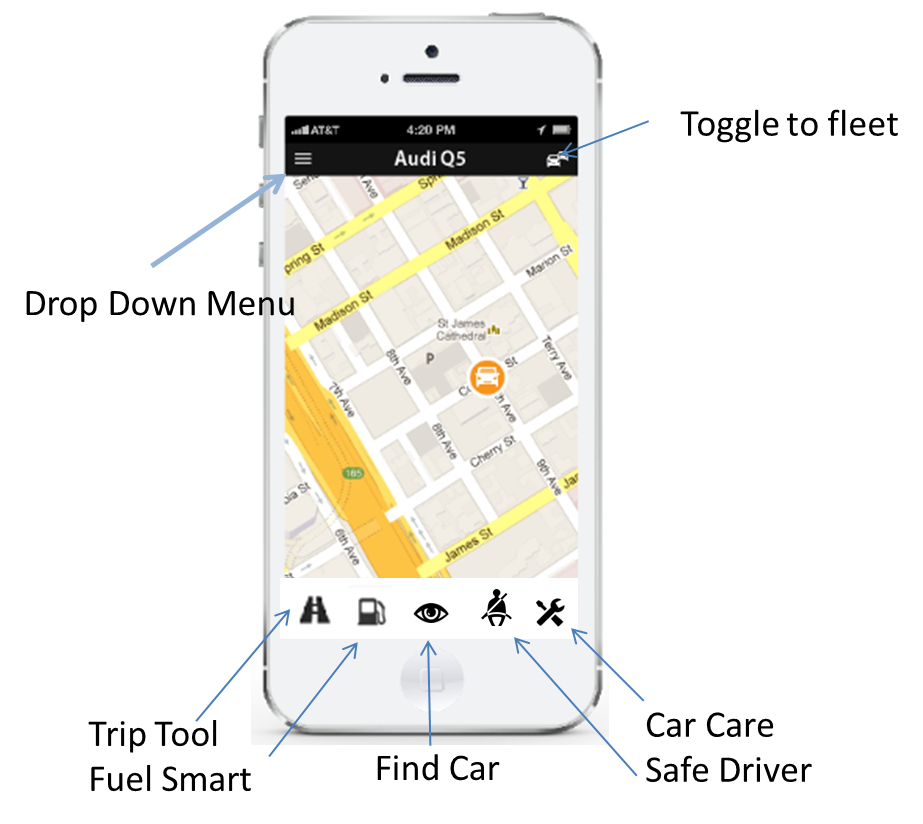


The persistent bottom navigation should include icons for Trip Tool, Fuel Smart, Find Car, Safe Driver, and Car Care.

**Single Car vs. Multi-Car Navigation:**

Throughout the app, when there are multiple cars linked, open to a multi-car list, then offer a drill down into a single-car view. If the user only has one car linked to the app, do not require them to go through the multi-car views. Instead drop them directly into the single-car screen for that feature.

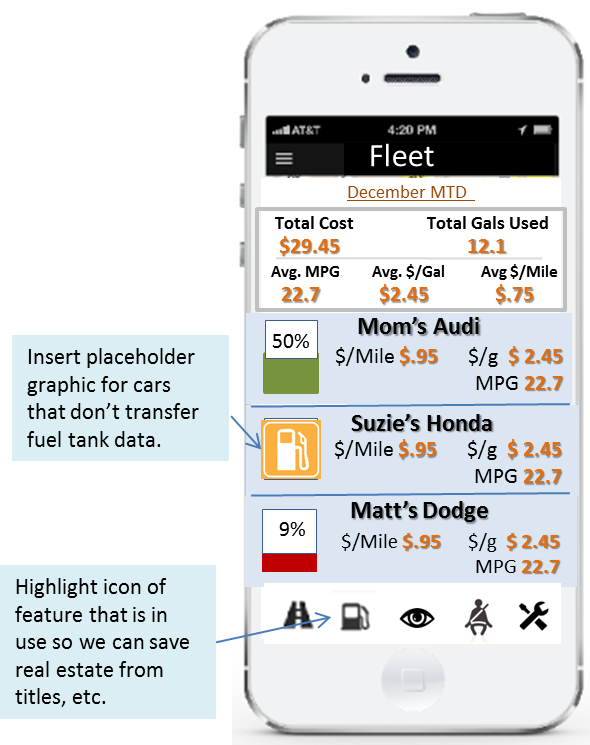
This should include the fleet vs. single car toggle at the top of the screen. If there is only car linked, the fleet icon should never show.



1. **Fuel Smart**

When the user chooses the Fuel Smart tool, they should initially see a rolled up view of current stats for all cars within a default date range (like month to date). The user should be able to change the date range, but clicking on that date, opening a calendar and selecting a new range.

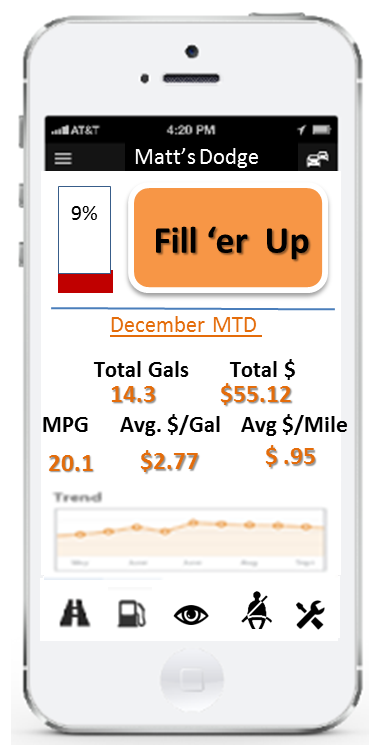
Below the roll up, the user should see a list of the individual cars that make up the total. The list should include the Averages: Average MPG, Avg. Cost per Gal, Avg. Cost per Mile and Totals: Total Fuel Cost, Total Fuel used for each car within that date range, along with a current Gas Tank fill icon and percentage. Tanks below 10% should show red, those between 20 and 10%, yellow, and everything above 20% should be green. For those cars that do not allow the device to read their fuel levels, insert a placeholder graphic instead of the actual percentage icon.



Anywhere within the Fuel Smart feature, users should be able to arrow key through one month increments. The current month should be the default view.

When the user clicks on an individual car in the list, they should then be taken to a detail screen that gives them more data. This data should include a trend chart that shows their MPG, Cost/Gal, and Cost/Mile on a line graph for the selected date range.

Below that could be a list of general tips for saving gas (e.g. “Make sure your tire pressure is correct”).



If the “Fill ‘er Up” button is selected, the user should also be able to enter information about a fill-up. The following data entry fields should be offered:

**# of Gals** (Required)

**Price per Gal** (Required)

**Location** (optional). Automatically fill where possible.

Odometer readings should be pulled from the Fuse Key. The field should also be editable in case the odometer reading can’t be pulled from the key, or it’s wrong, or the user simply prefers to enter a different reading.

If the user’s Fuse Key records fuel amounts, it should fill in the # of Gals. If the key does not, the user should be offered a data entry field. Once the required fields are entered, the system should total the amount the user spent.

Once at the station, the app should use the smart phone’s GPS an Google Places to locate the station and list the name. A drop-down box should list any other stations nearby (1 mile radius) in case the GPS is not dead on.

For those cars that share fuel level data, the app should record exactly how much gas was put in the tank and auto-enter that amount for averaging purposes.

***Note from Phil:*** *This will be easy/doable now if Carvoyant gives us the “fuel level changed” event I’ve asked for. Otherwise, we’ll push to v2.*

This amount will be recorded in a FIFO alogrithm for pricing purposes and updated after the car is started. The icon showing the tank level should adjust accordingly.

Once the user clicks “Submit,” the fill-up should be date and time stamped and pushed to the Car calendar.

1. **Trip Tool**

For Tip Tools, we should leverage the existing functionality, adding the following:

**Aggregate view for all Trips**

When the user opens the Trip/Miles tool, like the Gas feature, they should see a rolled up view of their fleet by a date range. (Default to current month-to-date). As in the Gas section, they should be able to change the month being viewed at will.



Once the user selects an individual car, they should be offered the list of trips for that date range. Arrows on either side of the date range should allow them to move through lists of trips associated with the previous months. A calendar view should also be available, with selection and previous month features.

**Totals:**

Add a totals section for that car at the top of the screen.

**Trip Names & Categories**

By clicking on the Arrow button, the user should be able to see the detail of the trip (like it works now) with the ability to name the trip.

In addition, the detail screen should offer a “Category” label with a drop down of pre-added categories:

Business

Medical

Moving

Charitable

Other

If the user chooses “Other,” a data entry field should open and allow the user to enter a new category for the list on the fly. A feature to manage this list should be offered in the Drop Down Settings menu.

Like the Trip Name, when a user chooses a category for a trip, that should remain the default for that trip until they change it. Categories should be visually different from Trip Names so they are easy to see (bold, color, etc. Danny will need to decide what looks best).

**Export Trip Data**

The Trips tool should offer a mileage export feature that allows the user to choose a date range, all cars or an individual car, then export that data to a .csv file and email it to a designated receiver (themselves, their CPA).

* + - * The data exported should include:
        1. Dates
        2. Times
        3. Stop/start locations
        4. Miles for each leg
        5. Total miles for date range
        6. Total Cost
        7. Individual Trip Cost
        8. Trip name
        9. Category
        10. Primary driver
        11. Vehicle

1. **Find Car**

Some typical uses of this feature could be when the car is in a big parking lot (like at the airport) and the user needs to find it, when anyone else is driving (like their teen), or when the car is suspected to be stolen.

If Google Maps offers it, allow the user to select a car to locate on the map view, then click a “Path” button to find their car.

Once selected, a highlighted path should appear on the map from where their phone is at the time to the car’s location.

From there any Google maps features (like navigation—walking, driving, etc. or directions) should be available to the user. Once the user is done, they should choose “Exit” from the Google navigation or directions and return to the main map view.



1. **Safe Driver**

When the user enters the Safe Driver feature, they should be shown the list of cars along with the name of the primary driver associated, a Safe Driver Score, and a settings button for each.

**Safe Driver Score:**

The user can then click on the score of a car/driver combo to be taken to a complete history screen of all relevant activities. 

**Safe Driver History:**

All activity should be logged in the Safe Driver History. Users should be able to access activities and points that were added or subtracted at any time.

**Point System:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Safe Driver Criteria** | **Points** | **Increment** | **Notes** |
| **Subtract points** |  |  |  |
| Hard Stops | -10 | Each |  |
| Fast Starts (acceleration) | -10 | Each |  |
| Night driving | -10 | Each | Track using official sunset/sunrise info – can we find and pull in? |
| Rush hour driving | -10 | Each | Need to pull this from somewhere. Rush hours are different in different areas. |
| Driving more than 3 hours w/o stop | -10 | Each |  |
| Driving more than 30 miles a day. | -10 | Daily |  |
| *Can we track when air bags are deployed?* | *-100* |  | *Thoughts?* |
| Geo fence |  | Each | Notify, but no points are decremented |
| High speed |  | Each | Notify, but no points are decremented |
| **Add Points** |  |  |  |
| Soft stop | 10 | Daily |  |
| Steady acceleration | 10 | Daily |  |
| Driving less than 30 miles | 10 | Daily |  |
| *Filled tank before ¼ empty* | *10* | *Each* | *What do you think about this one?* |
| Day light driving | 10 | Daily | Track using official sunset/sunrise info – can we find and pull in? |
| Driving below speed threshold |  | Daily | Notify, but no points are incremented |
| Stayed in/out of the Geo Fence |  | Daily | Notify, but no points are incremented |

**Settings:**

Users also have the ability to receive notifications on the following self-set parameters:

**Geo-fences:**

The user should be able to enter an address and radius for each geo-fence. If Google Maps allows it, show the fence the user has designated on a map. If the map looks correct, they can choose to get notified if the driver enters or exits the fence (toggle). If a geo-fence is breached, the user will receive a notification in the way they designate (see additional notification settings in Drop Down Menu>Preferences).

Clicking back into that Geo-fence at any time should allow the user to edit or delete that record.

Geo-fencing should not be part of the points system (safety score). This will allow multiple drivers can have multiple fences for the same primary driver without skewing the points economy.

**Speed Threshold:**

Users should also be able to set up MPH thresholds for a primary driver/car combo. This would be a finite number (90 MPH) and not be related to the posted speed limit. If the speed limit threshold is exceeded, send the user a notification in they way they designate in the app.

Like Geo-fencing this threshold should not be part of the points system. This will allow multiple users to set thresholds for the same driver (e.g. two parents for a single teen) without skewing the points system (and without letting the teen know).

**

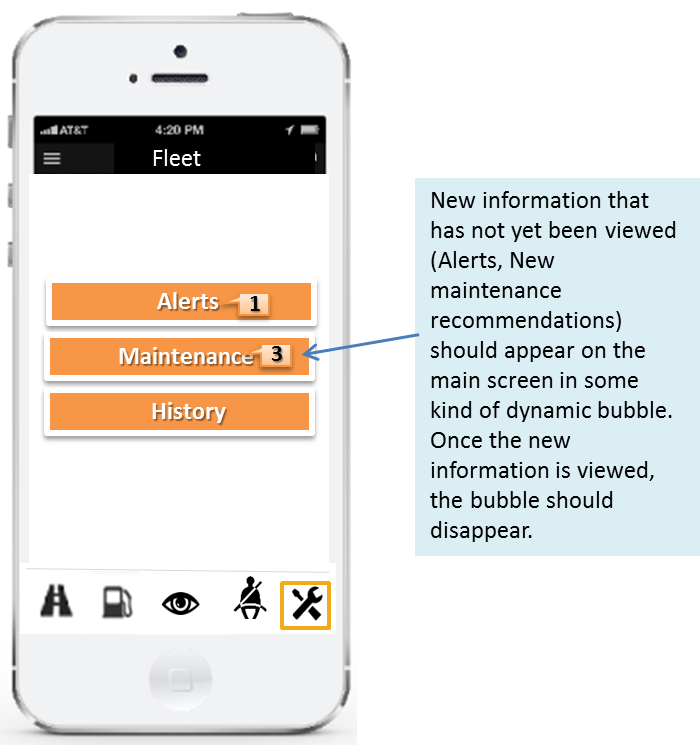
1. **Car Care**

When the user enters the Maintenance app, it will default to the Fleet view and offer the ability to go into three different features:

1) Alerts

2) Maintenance

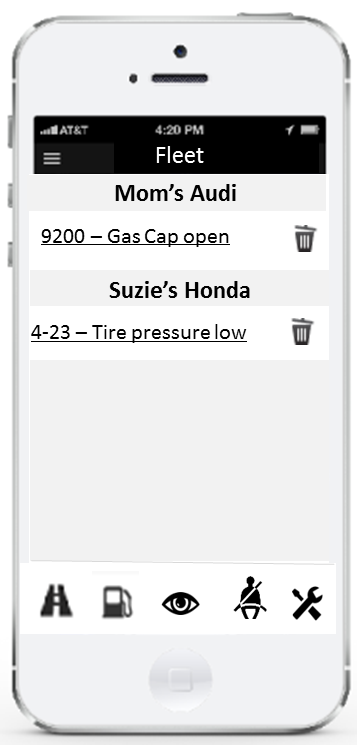
3) History



Alerts and Maintenance buttons should include dynamic bubbles that appear if there are new records to view. Once the user views those records, the bubbles should disappear until new information appears.

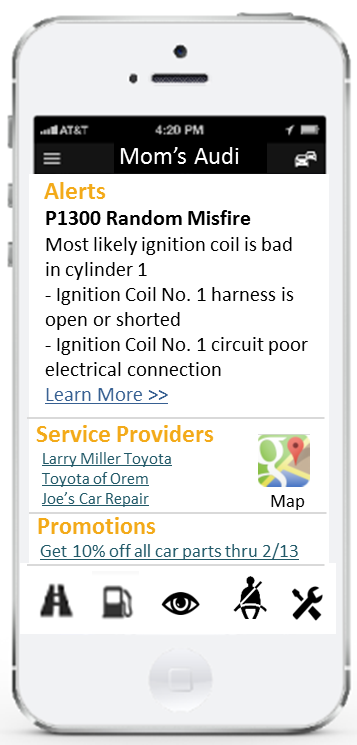
**1) Alerts**

Once the user chooses “Alerts” from the Maintenance menu, they should see a list of cars and any associated alerts. They should then be able to choose a specific car and drop into the Alert Detail record for that car.



**Alert Detail:**

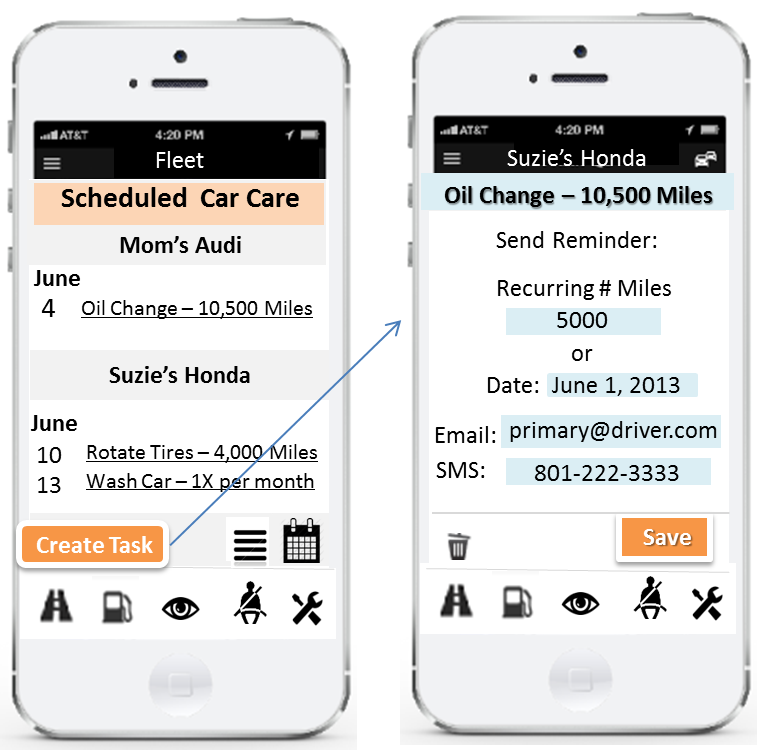
Once in the detail of the alert, the user should see the code, the summary description of the code, any solutions we have access to, then a link to the online user’s manual (Edmunds or other API required).

**

**2) Maintenance**

**Driver enters their own maintenance reminders:**

Inside the Scheduled Car Care feature, the user should have the option to create their own maintenance reminders. Once they click the “Create” button, they should be offered a detail data entry screen. There they should be able to enter the name of the task and then choose either an odometer increment or a date for the reminder.



Once saved, the maintenance reminder should be available in the Settings menu for editing or deletion. If the user chooses that item in Settings, they should be given the detail screen again, where the parameters can be changed, or the record deleted.

When a reminder event occurs (odometer or date), the new reminder should appear in the recommended tasks list where it can be treated like other new tasks (deleted or scheduled).

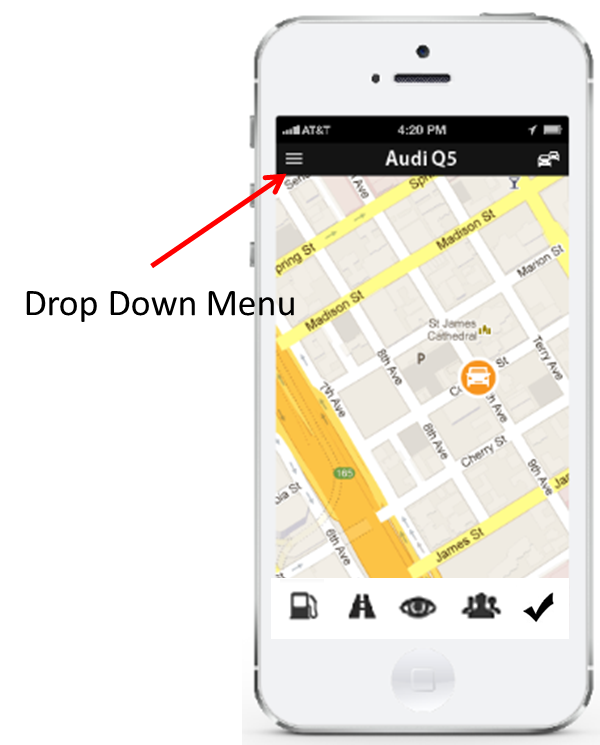
**3. Maintenance History**

If the user chooses to enter the History feature, they should see their entire fleet for the previous month. A date range link at the top should allow them to change to previous months. Below that, they should see the list of both scheduled and completed maintenance tasks. Completed tasks should be listed in a different color than scheduled. If the user clicks into a task, they should be able to see the saved detail.

****

1. **Drop Down Menu:**

A drop down menu that includes the app settings and other tools should be available in the upper left hand corner of the screen.



The Menu should include:

**Profile**

Username

Password (with reset functionality)

Email

Photo

Phone

**Preferences**

Push Notifications (On, Off)

The app should default to the phone notifications set in the OS. For example if the app is loaded on an iPhone, it should check the settings on the phone and use those for any push notifications (same with Android). Users should also be able to turn all notifications off/on from this area.

Email notifications (On, Off)

SMS notifications (On, Off)

***Question??*** *\*Wondering if these should over ride the maintenance & Safe Driver reminder settings?*

Share your data patterns anonymously? (Yes, No)

**Alerts**

Scheduled Maintenance List (This is where users should be able to edit or delete recurring maintenance tasks that they set up themselves. Should offer list by car).

**Car List**

Where users will go to delete, add, edit cars linked to app. Allow user to “unlink” a key.

**Trip Categories**

Add/edit/delete pre-defined categories

**Carpool**

To set up a new carpool, the user selects “Create New Pool” and names the new pool. The app should look for a contact list on the phone, then ask the user‘s permission to access it. If the user says “Okay” the list should open and the user should be able to use all the search and selection features of the list. If they say, “No,” they should be able to add the data by hand. If added from a contact list, the app should pull in any mobile phone and address information, then allow the user to edit/add if needed.

As the user adds destinations and people to their route, their photo, name and address should show in order for that pool. A drag-and-drop reorder feature should be available for the list in case a user wants to quickly reorder.

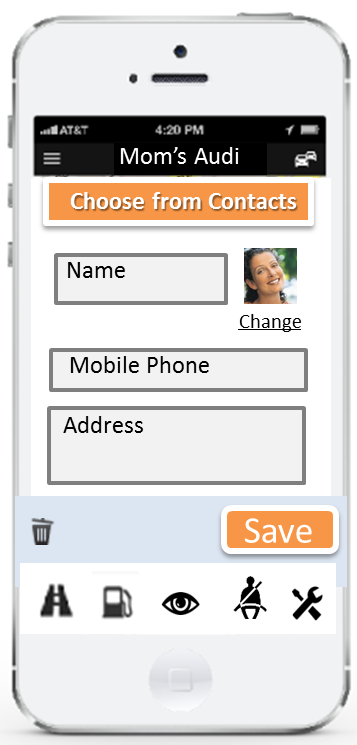
Once they’re happy with the list and the order, they should be able to click “Save.”

Once submitted, a success message should appear.

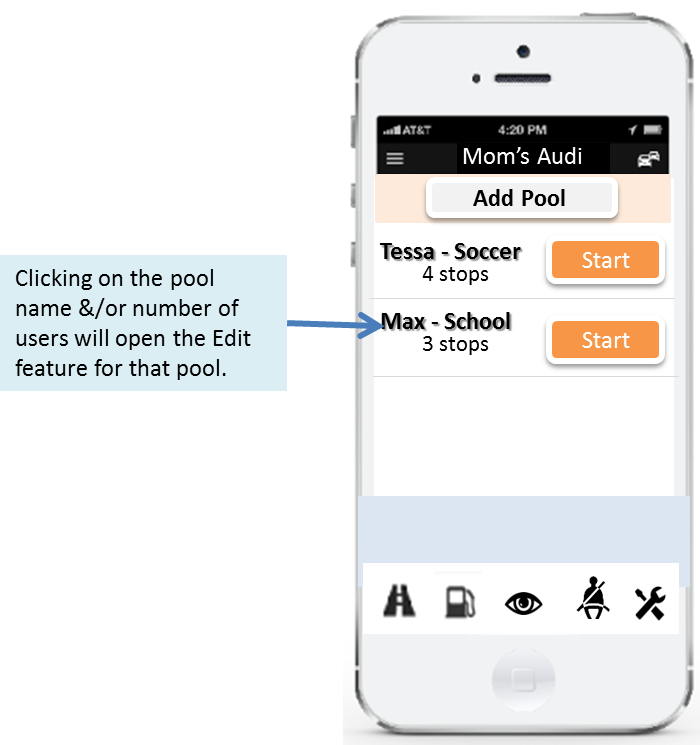
Edit and Delete features should be available when the user chooses a specific pool by clicking on the name. Within the detail view for that specific pool, they should be able to move the order of the stops by dragging and dropping them. Users should also be able to add to the list then see the new route on a map. The trash can in the upper right, if clicked, will delete the whole pool (once an “Are you sure you want to delete this list?” dialogue box pops up and the user says “Yes”).



If the user wants to Add, Edit or Delete a person (stop) inside a specific pool, the simply need to click on them in the list and a detail screen should open. That screen should allow the user to populate the Add Stop form from their Contacts list, or type in/edit the data themselves. *Make sure this will turn to landscape view for data entry for users with a slide out qwerty keyboard.* The user can also delete the stop by clicking the trash can icon, then Okaying the delete. Once the stop is done, they click “Save” and are taken back into the pool detail view with the list of stops and the map.



When the user is ready to drive a carpool, they can simply open the tool, then select the “Start” button next to the name of the pool.

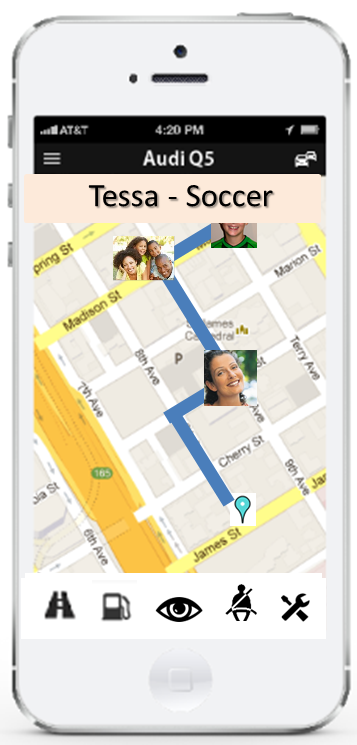


At that point, the app should then display the map on the screen with the route diagrammed (only if Google Maps does this for us) and begin sending notifications to everyone on the list. The user should be given a notification.

“*Car Pool Alerts have been sent to the <insert name> car pool.”*

Notification to people in car pool list could read:

*“Car Pool Alert: <insert name of driver> has left their house.”*



**Car Calendar**

This menu choice should open the designated third-party calendar that the user selected to record any/all events, tasks, or history.

**Fuse Store**

Link to an online store where the user can quickly buy more keys or any other products/services we may want to offer.

**Fuse Support**

Offer the user any support information (linked) like emails or FAQs. As additional support services are added, this screen will adapt.

**Logout**

Give user a button that allows them to logout at any time.

**Stretch Goals for PHASE 1:**

**Car Care:**

**Third-party or manufacturer-suggested maintenance reminders (if Edmunds data is available):**

When the user enters the maintenance feature, they should see a list of recommended maintenance tasks for all cars linked to their phone.

They can then choose which tasks they want to keep and which to discard.



If they choose to keep a task, they can change the date, or see service providers and promotions and send the info. to their phone/electronic calendar. If they save the task, links to provider and promotions lists should be transferred to the event/appt in the calendar so they can access it on the day of the service.

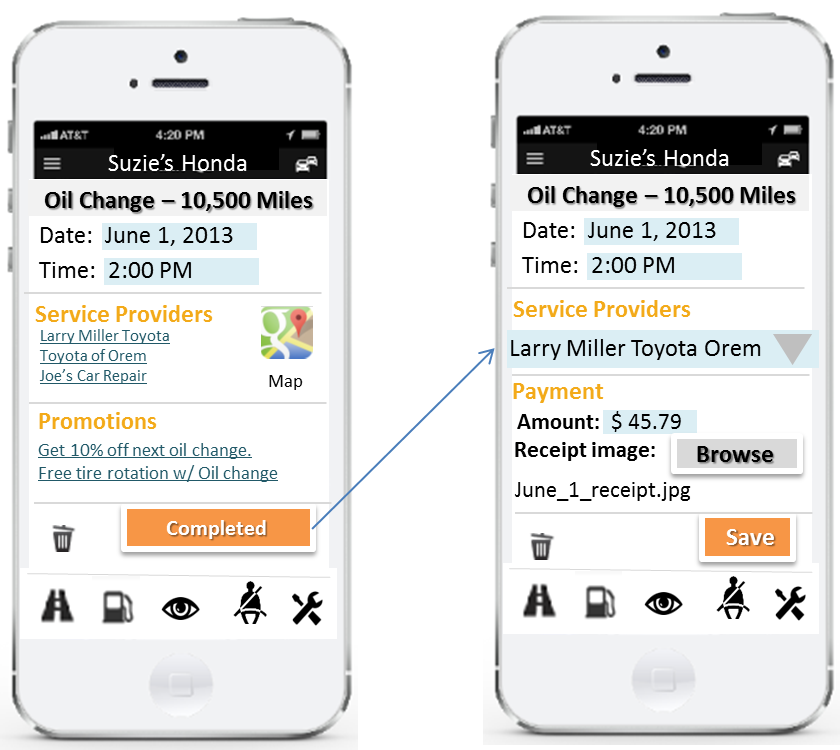
Once in the calendar, they should be able to use all the features from that calendar, like inviting others to the event, changing the times reminders are sent, etc.

*?? How closely can we link these calendars? If they change the details of the event in the calendar or delete it altogether, can it update the data in the app so the list view is accurate?*

Once the task is “scheduled” it will clear from the recommended list and be available in the Scheduled Car Care.



As the user goes to complete the maintenance task, they should be able to open it (by clicking on the item in the scheduled list). This should offer the adapted detail screen where they can click into the service provider link for directions, etc., or open promos and coupons beforehand.



Once the service is complete, the user should be able to record that it was completed, by whom, and record the amount. The app should use the most accurate method to offer the location where the service was done, or offer an “Other” option with a dynamic data entry field (like the current app). Users should also have the ability to add the photograph of the receipt to the task record.

Once the details for completion are filled in and the user clicks “Save,” the record should move out of the Scheduled Car Care list and into Car Care History on the App.

The user also has the option to delete the task altogether. If they choose to do so, the task/appt. should also delete from the calendar view.

The Fuse Key should send odometer data to the app so it can keep track of mileage related tasks in the future (e.g. oil changes, tire rotations).

**Appendix A**

**Available Data From Fuse Device**

|  |
| --- |
| Diagnostic Trouble Codes |
| Battery Voltage |
| Vehicle Reported Odometer |
| GPS Location |
| Heading |
| Engine Speed |
| Percentage of Fuel Remaining (on some cars) |
| Rate of Fuel Consumption |
| Engine Temperature |

In addition we see every trip (GPS data from engine on to engine off)

We get some of the data points (battery, GPS, Heading, DTC) even when the car is off.

**Hard Stops and starts?**

**Do we get cruise control use?**