# PATH

This is the Path Section. In short we are setting the expectations of what to look for with API.

## How does the API works

It will listen for a URL on a certain port to come through. Once received it will break the URL down and direct it to the correct function.

I.E. [http://wheelz.com:8080/get/drivers/](http://wheelz.com:8080/get/drivers)

The API will trim the URL removing (<http://wheelz.com:8080>) leaving (/get/drivers/).

Then it will take each word inside “/” and use them as a map.

Calls get function with an parameter of drivers. I.E. we are using JSON as a format.

[{  
 "DriverID": 1,  
 "first": "Rachel",  
 "last": "Murphy",  
 "email": "rmurphy0@census.gov",  
 "AddressID": 42047,  
 "picture": "http://dummyimage.com/107x214.png/dddddd/000000",  
 "MemberID": 359769,  
 "BackgroundID": 686700,  
 "car": "http://dummyimage.com/245x204.png/dddddd/000000",  
 "date": "2015/08/26",  
 "time": "3:34 AM"  
}, {  
 "DriverID": 2,  
 "first": "Gloria",  
 "last": "Anderson",  
 "email": "ganderson1@google.nl",  
 "AddressID": 908928,  
 "picture": "http://dummyimage.com/192x155.jpg/5fa2dd/ffffff",  
 "MemberID": 487013,  
 "BackgroundID": 562181,  
 "car": "http://dummyimage.com/159x182.png/cc0000/ffffff",  
 "date": "2015/07/05",  
 "time": "2:08 PM"  
}, {  
 "DriverID": 3,  
 "first": "Eric",  
 "last": "Dunn",  
 "email": "edunn2@walmart.com",  
 "AddressID": 554973,  
 "picture": "http://dummyimage.com/168x181.png/dddddd/000000",  
 "MemberID": 678581,  
 "BackgroundID": 828724,  
 "car": "http://dummyimage.com/226x173.png/ff4444/ffffff",  
 "date": "2015/11/05",  
 "time": "12:18 AM"  
}]

## Give each section a name and definition.

1. API
   1. Anything that the API receives or sends.
2. Internal
   1. The tool or tools we use.
3. SMS
   1. Messages that have been sent.
4. Website
   1. Rider and Driver UI for signing up, updating profiles, Tutorials, FAQ, etc.

### 

### 

### Features

We want to break each feature down into an example here. This makes for a much easier design and for a better understanding of anyone looking into the project as to what we are looking to accomplish with this feature.

#### Website

1. Sign Up
   1. Drivers
      1. The request is called by the website that a Driver is signing up. The needed information is then stored in the Driver table. A background request is sent via email and a the request is stored in the background table. The Driver’s address is then stored in the Address Table.
   2. Riders
      1. The request is called by the website that a Rider is signing up. The needed information is then stored in the Rider table. The address is stored in the Address table.
2. Profile
   1. Driver
      1. The request is called to view a Driver’s profile. The (first or pseudo name), Driver & Car pictures, rating, and reviews are sent. A increment is done and a log of who viewed the Driver’s profile is stored. If someone who is not logged in or recognized then the IP address will be logged.
      2. The request is called to update a Driver’s profile. Checks which columns to edit and sends the query.
   2. Rider
      1. The request is called to view a Rider’s profile. API checks to see if this is a Driver.

* If it is not a driver
  + Requested information is denied.
* If it is a Driver
  + The (first or pseudo name) & picture are sent. A increment is made and the DriverID is logged.
    1. The request is called to update a Driver’s profile. Checks which columns to edit and sends the query. A log is stored with IP address.

1. Feedback
   1. Driver
      1. A request to place feedback on a driver has been received. Checks if rating is included. Stores data inside feedback table. The preferred method of communication for the Driver is then checked and used to send a notification that feedback has been placed.
      2. A request to view a Driver’s feedback has been made. Pulls the Driver’s feedback using the DriverID. A counter is incremented and a log of (who is logged in or their IP address) is stored.
   2. Rider
      1. A request to place feedback on a rider has been received. Stores it inside the feedback table. An notice is then sent to the company.
      2. A request to view a Rider’s feedback has been made. Pulls the Rider’s feedback using the RiderID.
   3. Company
      1. A request to place feedback on the company has been made. Stores this feedback inside the feedback table. Then a notice is sent to the company. A log is then stored with the (Who is logged in or their IP address) that submitted the feedback.
      2. A request to view feedback on the company has been made. Replies with the a list of the feedback, status, and if any replies have been done. A log is created for the view.
2. Contact Us
   1. A request to send a message to the company has been made. Check how the message has been sent.
      1. Call
         1. Places this data into the Call queue.
         2. Creates a log.
         3. Checks the Available queue.
            1. If empty then checks the OnCall queue.

If empty then sends a message back to the website stating no one is taking calls. Send a notice to the company.

If someone is on call then send a message stating everyone is on a call and we will call them as soon as possible. Send a notice to the company.

* + - * 1. If not empty then send a message back to the website stating someone will be calling them soon. Send a notice to the company.
    1. SMS
       1. Places this data into the SMS queue.
          1. Checks if someone is in the Available queue.

If it is empty, checks if someone is in the OnCall queue.

If empty then sends a message to the website stating no one is available. Send a message to the company.

If not empty, send a message to the website saying someone will be with them soon. Send a message to the company.

If not empty, then send a message to the website saying someone will be with them soon. Send a message to the company.

* + 1. Chat
       1. Before this queue is available it will check to see if someone is in the Available queue.
          1. If not then it will check if someone is in the OnCall queue.

If not then it will allow someone to send a request.

If so then it will allow request.

* + - * 1. If true then it will allow requests.
      1. Places this data into the Chat queue.
         1. Once accepted Chats will be handled through a different API.
    1. Email
       1. Places this data into the Email queue.
       2. A cron job will run every 10 minutes and check this queue.
          1. If job finds an email it will run a script to send the email.

#### SMS

1. Request
   1. Riders
      1. Pick Up
         * Request to pick up.
           1. Places Rider into the queue and sends a message back asking for the address.
           2. Checks the address if it is already stored.

If not then gets geolocation from google API. Grabs the new AddressID and updates the Rider in queue.

If it is then grabs AddressID and updates the Rider in queue.

* + 1. Favorites
       - Rider has a list of addresses that they can save and add titles to.
       - If they have this setup, they just send an SMS with that title.

Request received and places the Rider into the queue.

* + 1. Queue
       - Get a Driver
         1. Checks for a Driver in the queue in that zip.

If not available, check the DriverIn queue.

If there are not any Drivers available then send an SMS to all Drivers.

If there are drivers available, send an SMS to the Rider that there are Drivers in the area, however they are all busy. The moment one is available we will notify you.

If Driver is available send an SMS to the Driver with profile and geoLocation and move the driver to the Possible queue.

* + - * 1. If Driver accepts then take the Driver & Rider out of the queue and create a transaction with their IDs.
  1. Drivers
     1. Ready
        + Just starting
          1. Places the Driver in the queue without a zip.
          2. Sends the Driver an SMS asking which zip.
          3. Updates the DriverID with zip.
        + Just finishing a Transaction
          1. Moves the driver from OnRide queue to Waiting queue
        + Just finishing Break
          1. Moves Driver from Break queue to Waiting queue
     2. Accept
        + Accepting a Rider.
          1. Takes the Driver out of the Waiting queue
          2. Places the Driver in the OnRide queue.
     3. Change
        + Updates the DriverID with a new zip.
     4. Deny
        + The Driver denies the offer for the Rider.
     5. Break
        + Moves the Driver to the Break queue.
     6. Leave

Takes the Driver out of the queue.

1. Transaction
   1. Rider
      1. Cancel
         * Cancels the transaction.
           1. Receives the SMS.
           2. Moves the Driver back into the queue
           3. Updates the Transaction to Canceled.
      2. How Much
         * Updates the TransactionID with Rider Paid
      3. Feedback
         * Feedback stored in the Feedback Table
      4. Rating
         * FeedbackID updated.
   2. Driver
      1. Accident
         * Tells us that the Driver been in an accident.
           1. Sends a notice to the Company.
           2. Sends an email to the Company.
           3. Takes the Driver off of the transaction.
           4. Suspends the Driver for further review.
           5. Sends an SMS to the Rider Explaining.
           6. Starts the request for a Driver over.
      2. Here
         * Updates the TransactionID.
      3. Picked Up
         * Updates the TransactionID
      4. Arrived
         * Updates TransactionID
           1. Arrived to get Rider
           2. Arrived to drop Rider
      5. Feedback
         * Provides Feedback about the Rider.
2. Feedback
   1. Rider
      1. Stores Feedback about the Transaction.
      2. Stores Rating for that Driver.
   2. Driver

Stores the Feedback about the Transaction

1. Profile
   1. Rider
      1. Update
         * Updates the profile.
           1. Receive an SMS (Update Pone Number 5555555555)
           2. The API will break up each word and consider options for them. If a word is misspelt it will check the database for those words. In this case pone. It looks through the database for functions starting with P. It finds 3. It will provide all take all through and look for the next letter. If it cannot find a po so it will see if the other three options have an o in there. If only one match comes up it will take that function.
           3. Update function is called.
           4. Found a match for Phone which is the function for changing a phone number.
           5. It will then remove all character ( ) - and spaces.
           6. Being that the next parameter is Number and it does not have 10 digits it will remove it from the list. The next option would be the 5’s. It matches all 10 digits.
           7. Sends an SMS back (If you would like to change your Phone Number to 555-555-5555 then please provide your password.
           8. Once the password has been received it will update the profile.
   2. Driver
      1. Update
         * Updates the profile.
      2. Feedback
         * Provides the oldest unread feedback.

#### Internal

1. Analytics

Statistics

1. Update
   1. Riders
      1. Bad feedback that does not belong.
      2. Suspend abusive Riders.
      3. Unsuspend Riders
   2. Drivers
      1. Bad feedback that does not belong.
      2. Suspend poor quality Drivers.
      3. Unsuspend Drivers
   3. Users
      1. Reset Passwords
      2. Suspend User access
      3. Change permissions
2. Commands
   1. Add SMS functions
   2. Edit SMS functions
   3. Suspend SMS functions
   4. Remove SMS functions
   5. Read SMS functions
3. Logs
   1. Search logs based on
      1. Times
      2. Events
      3. Viewer
      4. Receiver
4. Review
   1. The feedback will be provided as a queue so you can log in and view the feedback that has come in period. If you feel it is not appropriate or want to schedule some time with a Driver or a Rider to talk more about the feedback then you do so here.
   2. Feedback
      1. Rider
         1. Review feedback posted by a Driver.
         2. Suggest a schedule to talk with the Driver about this feedback.
      2. Driver
         1. Review feedback posted by a Rider.
         2. Suggest a schedule to talk with the Rider about this feedback.
      3. Company
         1. Review feedback posted by anyone.

Suggest a schedule to talk with them about this feedback.

1. Support
   1. About
      1. This is based on any type of support. Experiences with the UI, SMS, Driver, anything. These are a list of all the queues. It gives times waited, the name. It also gives missed communications.
   2. Chat
   3. Email
   4. SMS
   5. Phone

#### API

The interaction of everything listed above. Instead of re-listing all of the features I am leaving this section open to only methods that are not listed above. The API will be broken into three programs. SMS, Web, and Internal listeners. The reason for this is because I want all three jobs to be running without any bottlenecks.

1. Logging
   1. This is a very important part of any API or system in general.
   2. It needs to log every action
   3. Time
   4. Date
   5. User
   6. Function
   7. Output
   8. Status
2. Receiving SMS
   1. Every SMS received will be broken down into an Array for each word. The first word will be the command. Everything after that will be the a parameter or argument.
      1. If the Command is not found then it will move to the next word until it finds a word.
      2. If it does not find one or one that is similar then it will reply with (Sorry I do not understand)
   2. Every SMS will be logged with each phone number that is sending the messages. If a number becomes aggressive then it will be marked and will not be received, but still logged.
      1. 5 minutes
      2. 10 minutes
      3. 30 minutes
      4. 60 minutes
      5. Banned.
   3. Numbers that continue to harass even after being banned will be manually looked over before requesting a block from the SMS API.
   4. Queues
      1. Waiting
         * Waiting on a Rider
      2. OnRide
         * Is in the middle of a transaction
      3. Break
         * Not taking Riders, but expected to come back.
      4. Rider
         * Rider waiting on a Driver.
      5. SMS (Support)
3. Receiving Web Requests.
   1. Queues (Support)
      1. Though this queue will be maintained by the Internal it is added to by the Web.
         * SMS
         * Chat
         * Email

Calling

## Timelines

This is broken up into weeks so to possibly get an idea of how each feature will be added and program created.

### Week 1

1. Create outline for project management
   1. Monday
2. Setup the Databases
   1. Tuesday
3. Create Basic API outline for all three programs
   1. Tuesday and part of Wednesday
4. Create UI for Tickets, Logging, and Debugging
   1. Late Wednesday to Thursday
5. Test
   1. Thursday to Mid Friday
6. Correct any bugs
   1. Late Friday Possibly Saturday

### Week 2

1. Finish correcting bugs
   1. Monday midday
2. Implement SMS into API
   1. Late Monday thru Friday
3. If possible start testing for SMS
   1. Late Friday

### Week 3

1. Finish SMS if not done.
   1. Monday
2. Finish SMS testing
   1. Late Monday .
3. Correct any bugs.
   1. Tuesday
4. Implement Website into the API
   1. Tuesday thru Thursday
5. Test Website
   1. Friday

### Week 4

* Correct any bugs or errors.
  1. Monday
* Finish Internal UI to add the remaining features.
  1. Tuesday thru Wednesday
* Test UI
  1. Late Wednesday
* Correct Any bugs.
  1. Thursday evening

Priority

This section will mainly be managed by Andy. He will decide which portions and features are more important than others.

1. SMS
   1. Queue
      1. Riders
      2. Drivers
      3. SMS Support
   2. Riders
      1. Queue
      2. Commands
         1. Profile
         2. Requests
         3. Feedback
   3. Drivers
      1. Queue
      2. Commands
         1. Profile
         2. Requests
         3. Feedback
   4. Support
      1. Commands
         1. SMS Support
         2. Basic Q&A
2. Website
   1. Profiles
      1. Creation
      2. Editing
      3. Deleting
   2. Support (Queues)
      1. SMS/Chat/Email/Phone
         1. Request
         2. Check

Feedback

1. Internal
   1. Logging
      1. Read
      2. Archive
      3. Search
   2. Tickets
      1. Read
      2. Archive
      3. Search
   3. Queue Monitoring
      1. SMS/Chat/Phone
         1. Waiting
         2. Being Helped
         3. Helped
         4. Archived
      2. Email
         1. Waiting
         2. Sent
         3. Replied
         4. Waiting For Response.
         5. Archived
      3. Riders
         1. Waiting
         2. OnRide
         3. Archived
      4. Drivers
         1. Waiting
         2. OnRide
         3. Break

Archived

* 1. Feedback Review
     1. Riders
        1. Unread
        2. Read
        3. Replied
        4. Waiting
        5. Archived
     2. Drivers
        1. Unread
        2. Read
        3. Replied
        4. Waiting
        5. Archived
     3. Company
        1. Unread
        2. Read
        3. Replied
        4. Waiting
        5. Archived
  2. Messages
     1. Can only be read inside this tool.
     2. Unread
     3. Read

Archived

## Databases

This will provide a better sense of how they will be setup and what to expect inside them. There are several databases and with that being said I have created a spreadsheet for easier overview of the databases and how they look. Please reference this spreadsheet. Here is a list of the database for reference and Relations.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Riders | Drivers | Commands | Address | Transaction |
| Address | Address | User |  | Rider |
|  | Background |  |  | Driver |
|  |  |  |  | Address |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Favorites | SMS | Queues | Messages | Background |
| Address | Rider | Rider | User | Driver |
| Rider | Driver | Driver |  |  |
|  | Command | Address |  |  |
|  | Transaction |  |  |  |
|  | Address |  |  |  |

|  |  |  |
| --- | --- | --- |
| Feedback | Log | User |
| Rider | All IDs are linked |  |
| Driver |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

## Queries

I will be listing queries needed here for calling in the database. This will make classes easier to create for the database connections.

* Create
  + Check if exists first
    - If exists then update
    - If does not exists and return false
  + Insert row
  + Insert rows
* Read
  + Check if exists
    - If does not exist return false
    - If does exist read and return true
  + Read Table
  + Read Row
  + Read Columns
* Update
  + Check if exists first
    - If does not exist then return false
    - If does exist then update and return true.
* Delete
  + Notice: This should not be used often, mainly for the queues.
  + Check if exists first
    - If does not exist then return false.
    - If does exist then delete and return true
  + Delete row
  + Delete rows