BLINKAH Prototype Testing Report 2
ENG EC 464 Senior Design

Team 2:

Ritam Das
Ram Bukkarayasamudram
Will Pine
Ayush Upneja
Parker Van Roy

Summary

The prototype test covered the following topics:

- **All new license plate OCR (Ram Bukkarayasamudram + William Pine)**
- Diagram Updates
- **Lane Detection (Ritam Das + William Pine)**
- **Further Jetson Nano Hardware (Parker Van Roy)**
- **Django API (William Pine + Ritam Das + Parker Van Roy)**
- **Dashboard (Ayush Upneja)**
- Timeline
- Updated deliverable progress

Each team member spearheaded one of the bolded topics, and the other updates were discussed during standup meetings and agreed upon as a team effort.

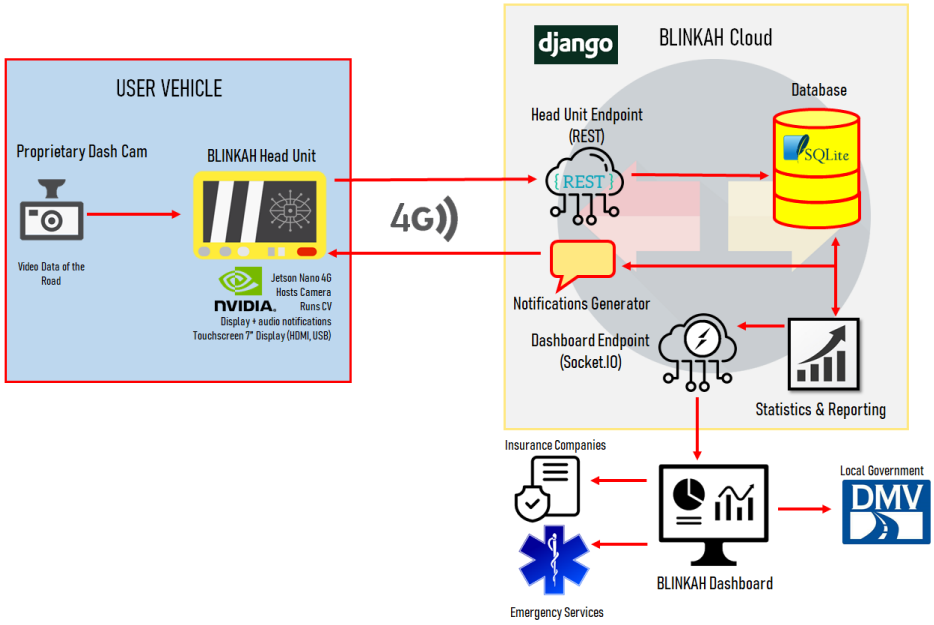
Results

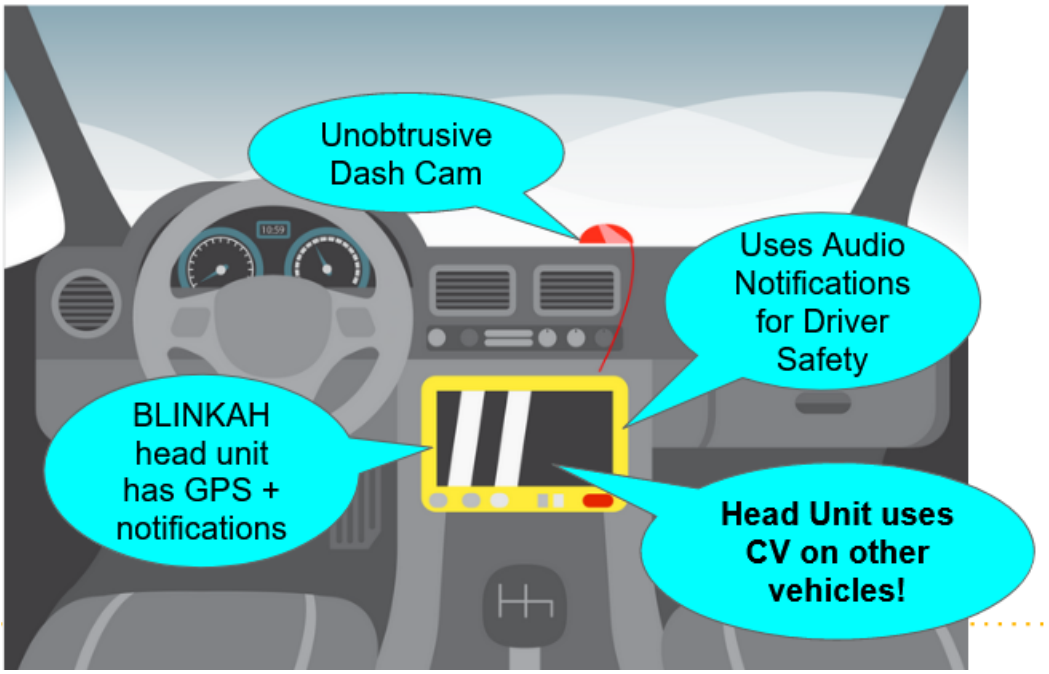
Updated MVP Deliverable Progress

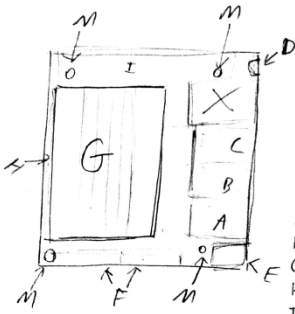
- Working hardware prototype
    - BLINKAH hardware unit - camera + ML algorithm
    - Data upload to BLINKAH cloud
- Software algorithm
    - Cloud data processing of road analysis
    - > 2/3% erratic driver identification accuracy
- Dashboard + Backend API
    - API endpoint: output relevant data to entities
    - Secure database of driver history records

Diagram Updates

Some of our diagrams were updated to reflect our newest expectations.



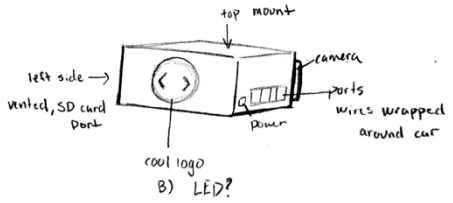


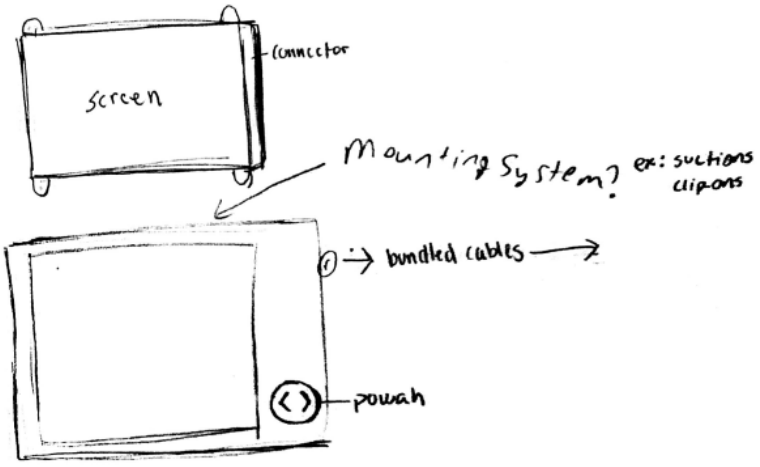


to beany

X = not used

A = HDMI
B = monitor USB + cellular USB
C = wifi + open USB
D = micro USB
E = power
F = camera (ribbon cable)
G = heatsink (don't be next to plastic!)
H = SD card (removeable)
I = GPIO pins internal usage (speaker, power)
M = mounting holes





License Plate OCR (Ram)

For our second prototype, we realized that there were several shortcomings in our license plate OCR tool we were originally using. The original google colab based algorithm was unable to detect any text several times from the license plates, used too much memory, and was space inefficient. As a result, we decided to switch to OpenALPR, another open sourced tool used for license recognition. For the prototype, we tested the license plate OCR using OpenALPR, data presentation, and accuracy. We ran a set of 26 sample images through our OpenALPR based python test script and had the user input which to choose. Our test results were able to gather text from most of the license plates from sample photos collected, have a confidence range showing its most accurate predictions, and store data. However, this was heavily reliant on the quality of the image since some lower quality photos resulted in the algorithm not detecting them. The new license plate OCR is more portable for the hardware component, and we gathered more accurate license plate results in text format, and we were able to present it properly in a JSON file. Future plans on this portion involves sending the data to our server, and testing it on videos or potentially breaking the videos down and running it within frames. Overall, the test results match the test plan expectations, and there's been good progress.

Sample results from test

```
ram@ram-Lenovo-YOGA-900-13ISK:~/Desktop/ML/test_scripts$ python3 main.py
Enter a value from 1 to 26
11
[{'license_plate': 'FJR8741', 'confidence': 87.5443}, {'license_plate': '1FJR8741', 'confidence': 85.6506}, {'license_plate': 'F
JRB741', 'confidence': 85.2504}, {'license_plate': '1FJRB741', 'confidence': 83.3567}, {'license_plate': 'JR8741', 'confidence':
 80.0286}, {'license_plate': 'PJR8741', 'confidence': 79.5713}, {'license_plate': 'FJ8741', 'confidence': 79.1202}, {'license_pl
ate': 'FJRO741', 'confidence': 78.9353}, {'license_plate': 'FJRG741', 'confidence': 78.5756}, {'license_plate': 'FJB8741', 'conf
idence': 78.1785}]
ram@ram-Lenovo-YOGA-900-13ISK:~/Desktop/ML/test_scripts$
```

{} Reports.json ×

{} Reports.json > ...

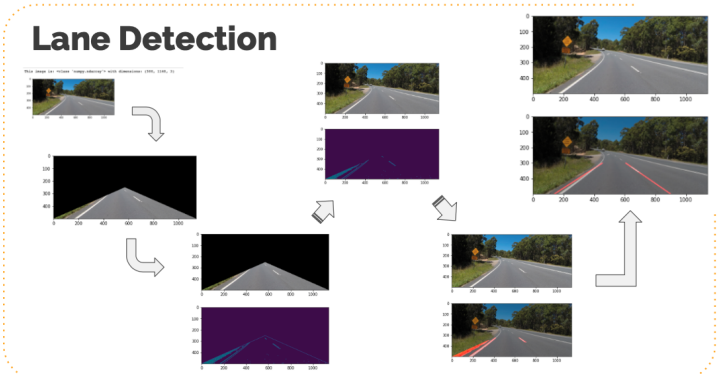
```
1  [{"license_plate": "FJR8741", "confidence": 87.5443}, {"license_plate": "1FJR8741", "confidence":
```

Saves values in Reports in JSON format

```
ram@ram-Lenovo-YOGA-900-13ISK:~/Desktop/ML/test_scripts$ python3 main.py
Enter a value from 1 to 26
1
[{'license_plate': 'AM74043', 'confidence': 92.3589}, {'license_plate': 'AH74043', 'confidence': 84.9314}, {'license_plate': 'AM
74043', 'confidence': 84.598}, {'license_plate': 'A74043', 'confidence': 84.3962}, {'license_plate': 'AM74Q43', 'confidence': 84
.0058}, {'license_plate': 'AM74D43', 'confidence': 83.535}, {'license_plate': 'AN74043', 'confidence': 83.249}, {'license_plate'
: 'AB74043', 'confidence': 82.7792}, {'license_plate': 'AM74U43', 'confidence': 81.9293}, {'license_plate': 'AM74B43', 'confiden
ce': 81.7965}]
ram@ram-Lenovo-YOGA-900-13ISK:~/Desktop/ML/test_scripts$
```

Another example

Lane Detection (Ritam Das)

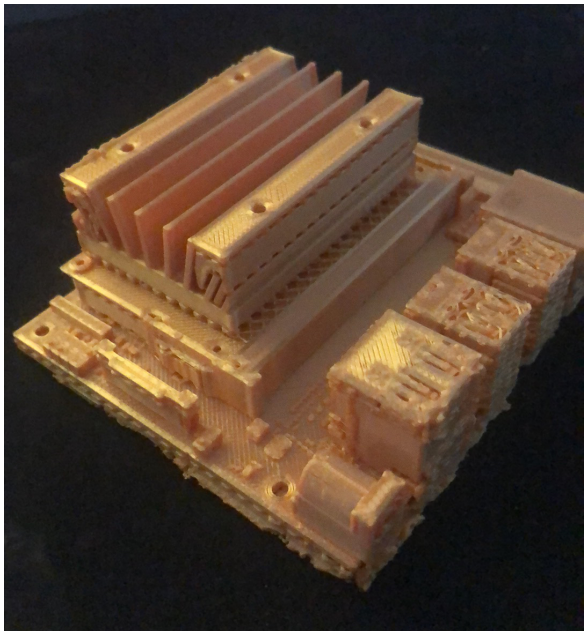


Using a medium article for guidance, we took a simple image as input data and processed it to detect the lane within which the vehicle is. We found a representative line for both the left and right lanes and rendered those representations on the final image as a red overlay. Using opencv, numpy, and matplotlib we were able to do the necessary processing, transformation, and rendering of the image data. All of this was developed in a jupyter notebook. Initially, we loaded the image into memory and cropped the image to a region of interest. We then ran edge detection on the cropped image by converting to grayscale, analyzing intensity values within the image, and seeing where the most rapid change was/where a pixel is a mismatch in color to all of its neighbors. We then ran Canny edge detection and removed the edges formed by our region of interest triangle. Lines were then generated from edge pixels using Hough transformations. These Hough lines were rendered and overlaid on the original image. Finally, we grouped the left and right line groups to create a single linear representation of each line group (essentially extending the lines into the horizon. In theory this lane detection should be applicable to video using moviepy, but I ran into issues and fell short of a breakthrough. We will address this as a team when waterfalling our ML algorithm.

Jetson Nano Configuration (Parker Van Roy)

As far as the hardware, the main concern currently is working with the limited memory (4G). We attempted to connect our 4G LTE modem however ran into problems with the modems not supporting linux (false marketing). In redirection the sprint was focused on CAD modeling for the Head Unit. We needed a 3D print for reference when creating custom enclosure for the BLINKAH head unit. Performing hardware enclosure development, measurement, and testing with actual Jetson nano board was too risky, so 3D print

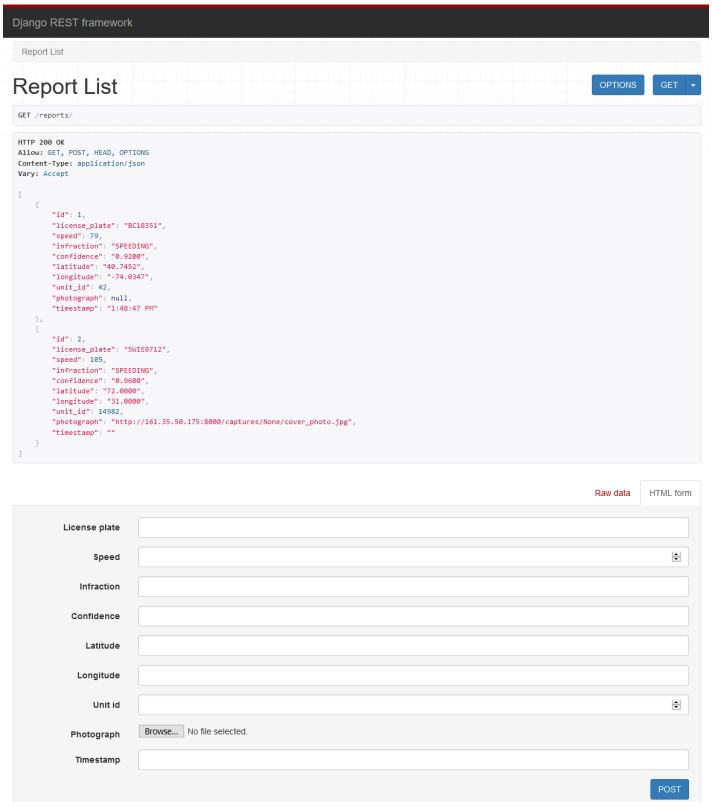
was made from official CAD files. The print is ABS in Silk Gold PLA.

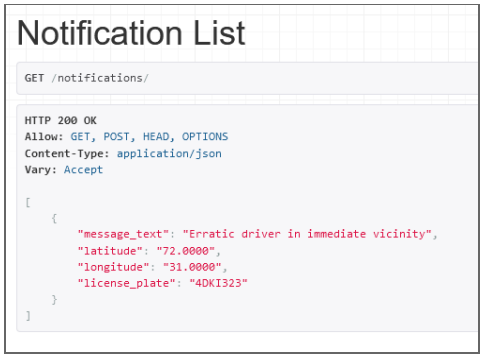


Django API (Will, Parker, Ritam)

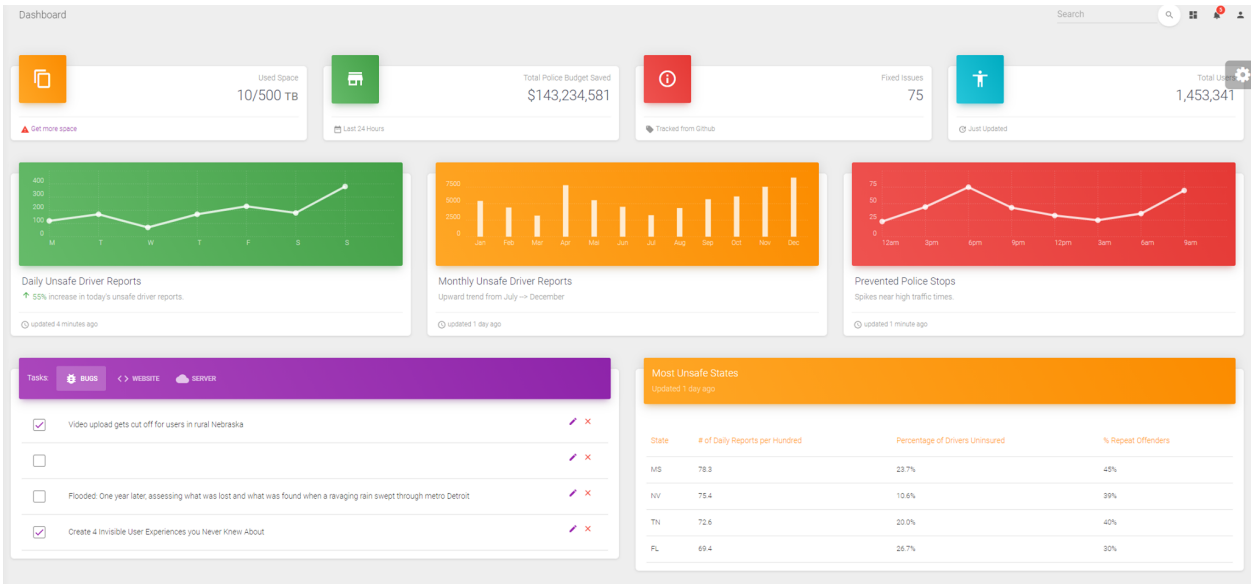
A Django REST API server was set up to act as the backbone of the project, communicating with both the head units and the dashboard. A REST API was chosen due to its standardization in providing, accessing, and changing data, and Django was used as a framework as it makes serialization (the process of converting & querying tabular data into JSON format) and database schema definitions very straightforward. We chose to use SQLite as our database software, as it easily integrates with Django and is lightweight enough for our MVP.

Django was installed and configured using Python's virtualenv module to manage dependencies, and makes use of the Django REST Framework plugin for creating REST endpoints that tie in with Django's schema definitions and database handling. I came up with a database schema to hold reports (uplinked from head unit) and notifications (downlinked to head unit), and defined its models and serializers, used in REST framework. Finally, I wrote the endpoint specification, which is used by both the head units, for reporting and polling, and by the dashboard, for information collection, display, and analysis.

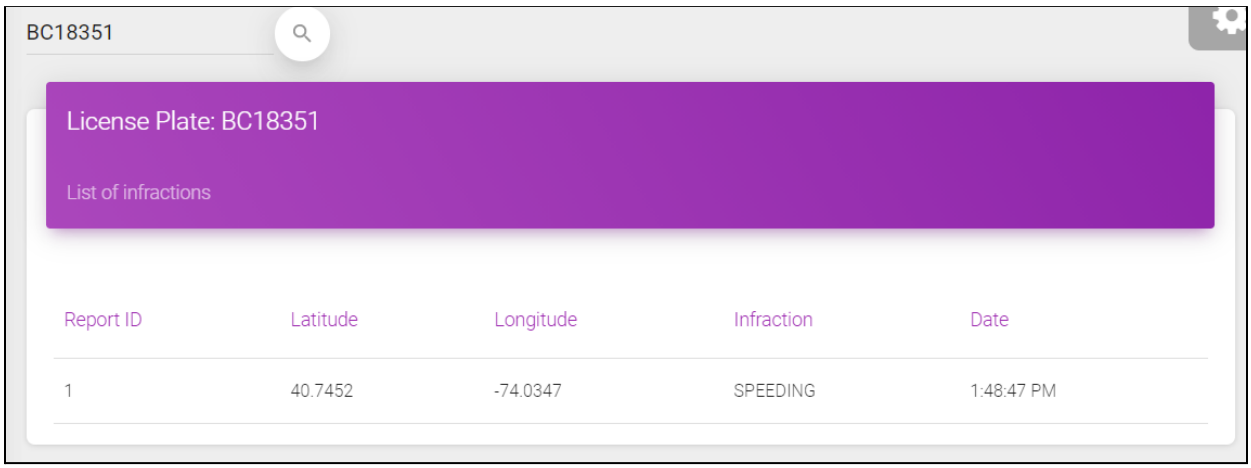


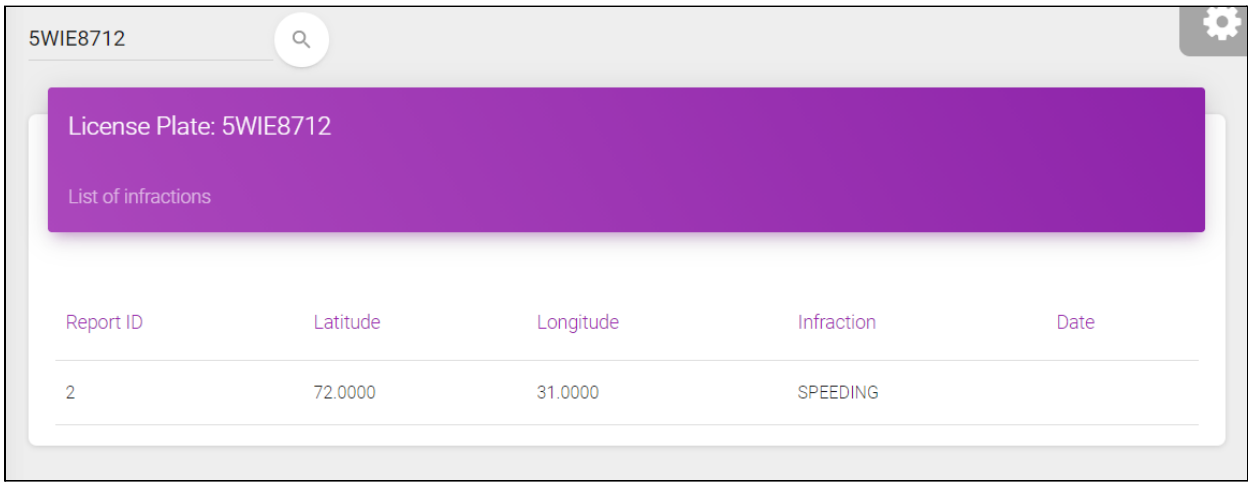


Dashboard (Ayush Upneja)



For the second prototype of the frontend visualization, we kept the original data dashboard, and focused heavily on getting the end to end testing working. Thus, we kept the dynamic graphs and focused on the license plate lookup feature as that is the current core feature of our platform. We are able to currently pull directly from the backend, inputting any existing license plate and receiving the relevant information. Here are two pictures of us polling license plates directly.





Here we are able to display the Report ID, the Latitude, Longitude, Infraction, and Date of Report. This license plate lookup is seamless, and almost instantaneous. We will be iterating on this functionality much more in the future, now that we have the end to end fully connecting from back to front.

Timeline

Overall, we are making steady progress as we shifted into complete agile development. There is a bit of slowdown in terms of ML progress due to lack of experience, so it may be beneficial to temporarily waterfall that side of things to make sure it's up to speed with the rest of the project.

Discussion

We received praise from Professor Pisano on our progress so far, however, he did share concerns of getting everything to work together. He praised our individual abilities and tasks, but reminded us that they need to work as one. Here are our MVP goals with the components that have shown progress highlighted:

- Working hardware prototype
    - **BLINKAH hardware unit - camera + ML algorithm**
    - Data upload to BLINKAH cloud
- Software algorithm
    - **Cloud data processing** of road analysis
    - > 2/3% erratic driver identification accuracy
- Dashboard + Backend API
    - **API endpoint: output relevant data to entities**
    - **Secure database of driver history records**

We have isolated applications with very little done in terms of BLINKAH as a whole. It is time to focus on bringing loose ends together.