BLINKAH Prototype Testing Report
ENG EC 463 Senior Design

Team 2:

Ritam Das
Ram Bukkarayasamudram
Will Pine
Ayush Upneja
Parker Van Roy

__Summary__

The prototype test covered the following topics:

- Redefined MVP Deliverables
- Diagram Updates
- **License Plate OCR (William Pine)**
- **Jetson Nano Configuration (Ram Bukkarayasamudram)**
- **Django API (Ritam Das)**
- **CI/CD Pipeline (Parker Van Roy)**
- **Dashboard (Ayush Upneja)**
- Timeline

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__

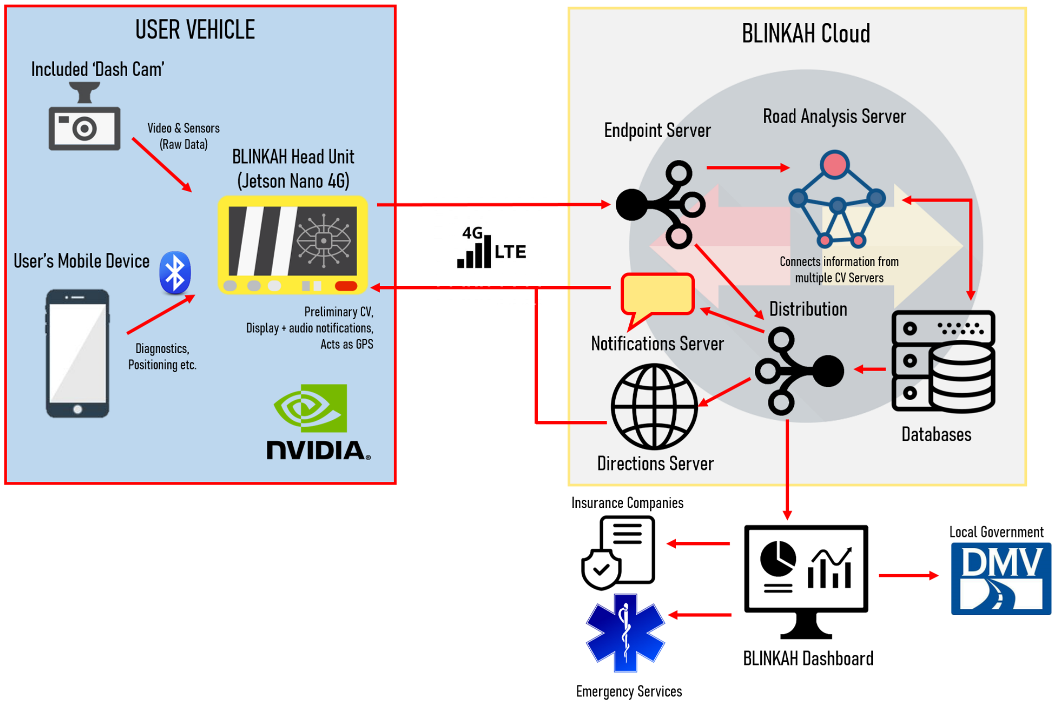
Redefined MVP Deliverables

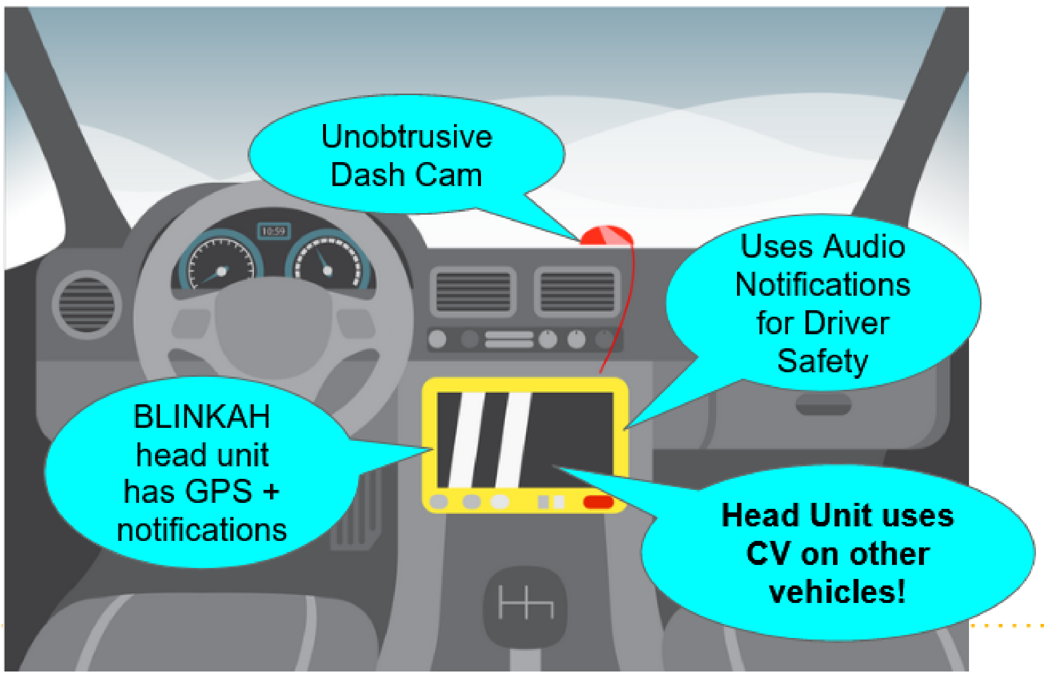
The following was decided upon as adjusted MVP deliverables.

- 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

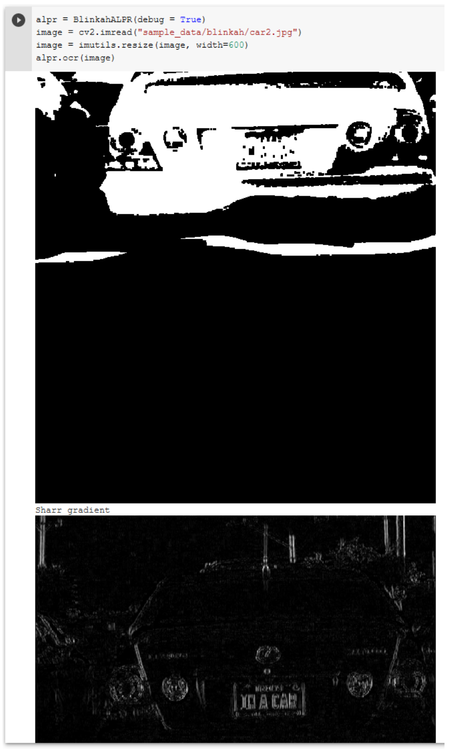
Diagram Updates

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



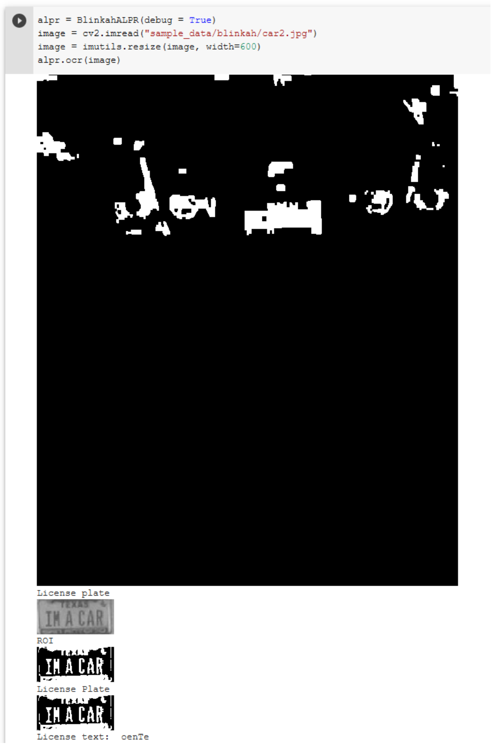

Unobtrusive Dash Cam
Uses Audio Notifications for Driver Safety
BLINKAH head unit has GPS + notifications
Head Unit uses CV on other vehicles!

License Plate OCR (William Pine)



```
alpr = BlinkahALPR(debug = True)
image = cv2.imread("sample_data/blinkah/car2.jpg")
image = imutils.resize(image, width=600)
alpr.ocr(image)
```

```
Sharr gradient
```



```
alpr = BlinkahALPR(debug = True)
image = cv2.imread("sample_data/blinkah/car2.jpg")
image = imutils.resize(image, width=600)
alpr.ocr(image)
```

```
License plate
ROI
License Plate
License text:  oenTe
```

For this prototype test, we exhibited our progress on our OCR algorithm for recognizing/classifying US license plates. We are prototyping the algorithm in Google Colab, making use of Python, OpenCV, and the Tesseract library. In our demonstration, we ran multiple still frames of dashcam footage against our algorithm, which was able to successfully identify and isolate license plates in almost all of the photos. However, our OCR algorithm still needs a decent amount of fine-tuning and iteration, as it is not displaying the correct letters and digits of the license plates used as input.

Jetson Nano Configuration (Ram Bukkarayasamudram)

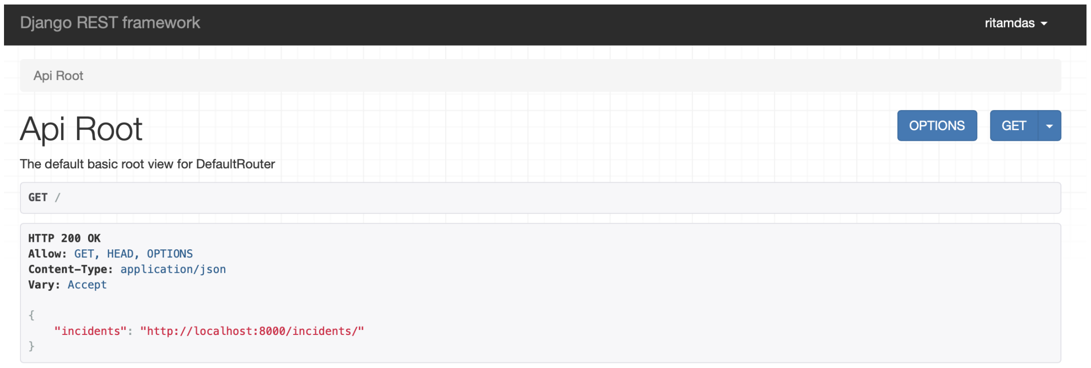
For the set up of the hardware portion, we decided to prototype using a Jetson Nano. The Jetson Nano is the ideal hardware unit for our project because it has a lot of RAM and easy IoT functionality which are both important for our specific project. High processing power allows us to use it effectively for handling our ML and CV work and easy IoT will allow us to set it up appropriately for real-time, updated data handling.
For the actual demonstration, we had it configured to a sample monitor, keyboard, mouse, and a working webcam. We were able to host a server on the Jetson Nano, and we went to this server to run a sample image classification module. The webcam was set up, and allowed us to demonstrate a simple thumbs-up or thumbs-down machine learning algorithm based on whether the camera detected a thumbs-up or thumbs-down. We trained this data with several images beforehand to get a reasonable accuracy for the demo. Our demonstration showed: complete configured Jetson Nano, communication with internet (hosting server), and sample CV processing.

In terms of how it will be of use to our actual product, we plan to use the jetson nanos to process most of our ML and CV data. Therefore, we demonstrated that we could have a sample server hosted and run some basic ML code on it. The challenge will be trying to set up two of these Jetson Nano configurations in order to communicate with more entities.

Django API (Ritam Das)

A simple REST API was built using the Django REST Framework. A REST API was chosen due to its standardization in providing, accessing, and changing data. Django made serialization, the process of converting & querying tabular data into JSON format, much easier from a developer perspective. Our implementation allows an admin backdoor into the database, which BLINKAH plans to have multiple of for different purposes, so this process must be quick and efficient.

Django was installed and configured with the correct virtual environments, created the API app, and admin credentials. A model was created in the database for the Django ORM to manage. With this and the installation of the rest framework, the model is ready to be serialized. Serialization involved importing a serializer and creating a class to link our model with its serializer. The URLs to visualize the data were wired up and BLINKAH's backdoor into the database was created. Upon completion, admins have the ability to get, post, and delete data within the DB. The challenge comes later in the development process as we try to piece everything together.



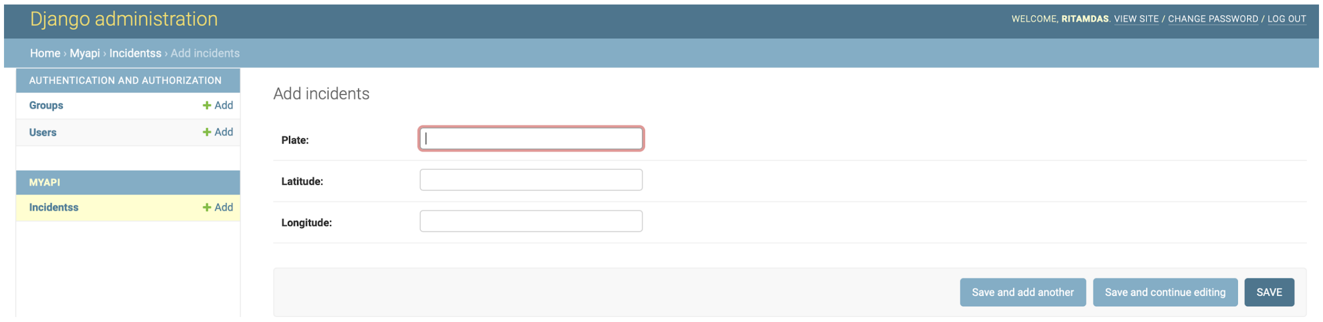
Api Root

The default basic root view for DefaultRouter

```
GET /
```

```
HTTP 200 OK
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "incidents": "http://localhost:8000/incidents/"
}
```

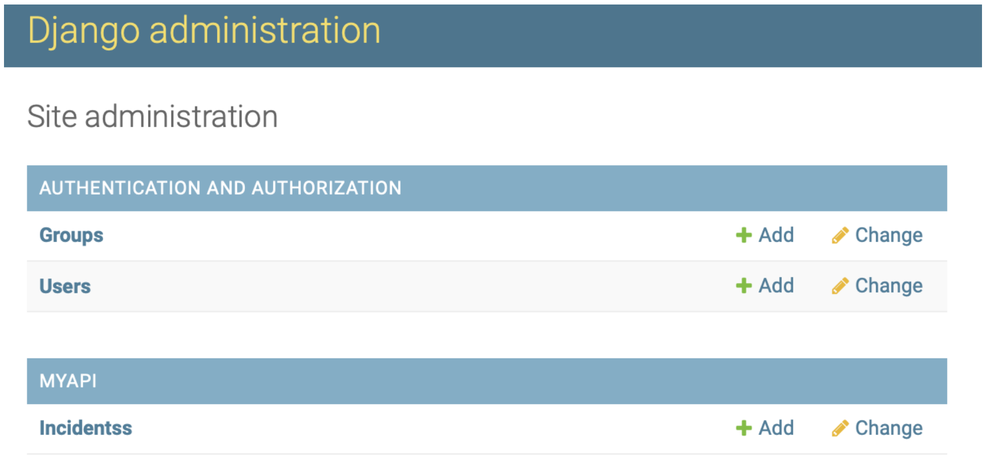

Django administration
WELCOME, RITAMDAS. VIEW SITE / CHANGE PASSWORD / LOG OUT
Home › Myapi › Incidentss › Add incidents
AUTHENTICATION AND AUTHORIZATION
Groups + Add
Users + Add
MYAPI
Incidentss + Add
Save and add another
Save and continue editing
SAVE

Add incidents

Plate:

Latitude:

Longitude:


Django administration

Site administration

AUTHENTICATION AND AUTHORIZATION		
Groups	+ Add	Change
Users	+ Add	Change

MYAPI		
Incidentss	+ Add	Change

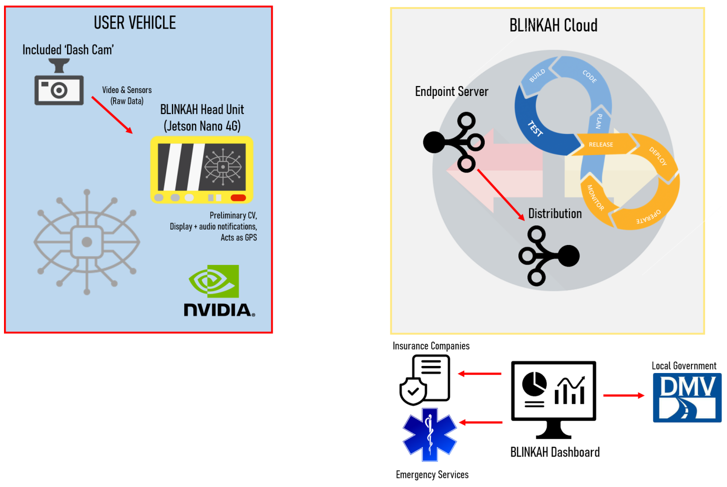
CI/CD Pipeline (Parker Van Roy)

Based on the updated requirements and system overview, we have determined a CI/CD approach that should function well for our system:

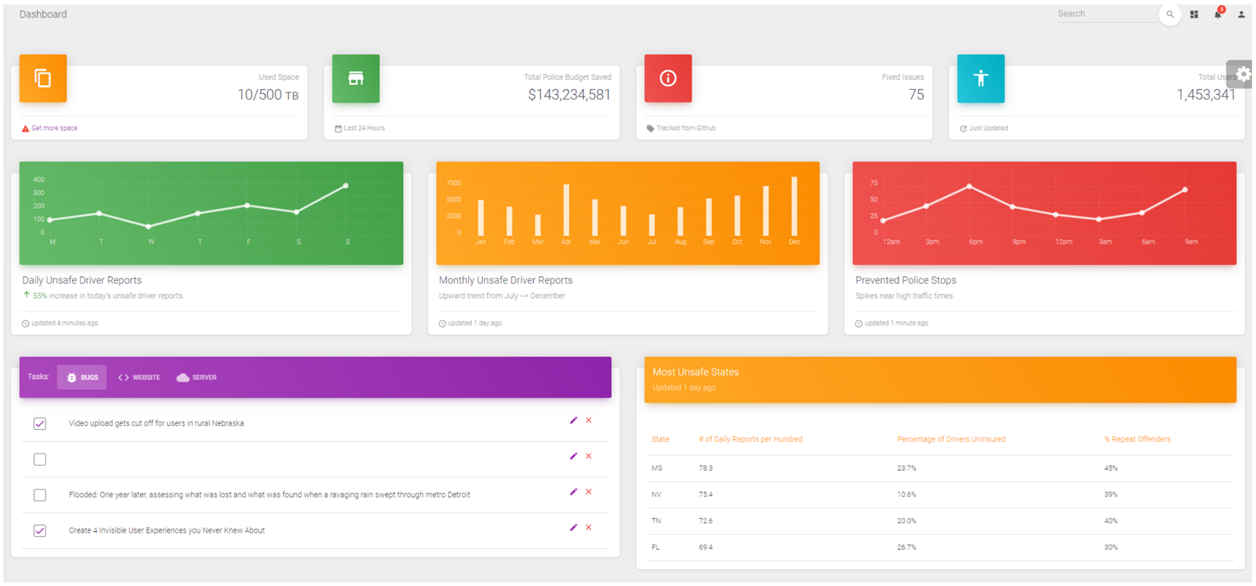
The main CI tool in use will be CircleCI. CircleCI is a leading CI tool with convenient integrations - GitHub Actions, POSTMAN, and Elastic Beanstalk. The approach is to create an E2E test for the BLINKAH cloud and backfill internal servers with functionality.

The process is approximately like this for committing to the BLINKAH cloud:

1. CircleCI is run based on GitHub Actions from a PR
2. CircleCI w/ Postman runs an E2E test with high code coverage tests on upstream servers
    a. CircleCI can also function for ARM64 compilation to test Jetson builds
3. A PR is code reviewed, rebased, and possibly accepted
4. A release candidate is selected from multiple PRs
5. AWS Elastic Beanstalk spawns EC2 instances based on releases
6. A manual visit to the BLINKAH dashboard can analyze for any inconsistencies



Dashboard (Ayush Upneja)



Used Space
10/500 TB
Get more space

Total Police Budget Saved
$143,234,581
Last 24 Hours

Fixed Issues
75
Tracked from Github

Total Users
1,453,341
Just Updated

Daily Unsafe Driver Reports
55% increase in today's unsafe driver reports.
updated 4 minutes ago

Monthly Unsafe Driver Reports
Upward trend from July --> December
updated 1 day ago

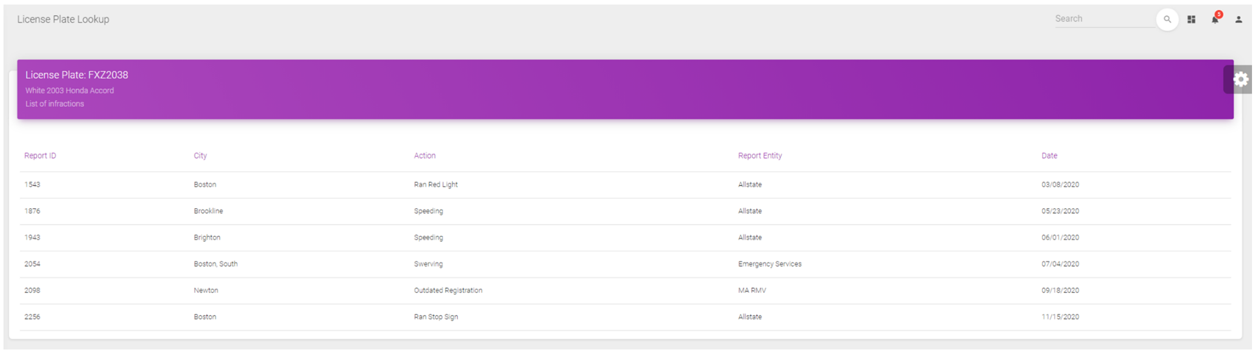
Prevented Police Stops
Spikes near high traffic times.
updated 1 minute ago

Tasks: BUGS WEBSITE SERVER

✓	Video upload gets cut off for users in rural Nebraska
☐	
☐	Flooded: One year later, assessing what was lost and what was found when a ravaging rain swept through metro Detroit
✓	Create 4 Invisible User Experiences you Never Knew About

Most Unsafe States
Updated 1 day ago

State	# of Daily Reports per Hundred	Percentage of Drivers Uninsured	% Repeat Offenders
MS	78.3	23.7%	45%
NV	75.4	10.6%	39%
TN	72.6	20.0%	40%
FL	69.4	26.7%	30%



License Plate: FXZ2038
White 2003 Honda Accord
List of infractions

Report ID	City	Action	Report Entity	Date
1543	Boston	Ran Red Light	Allstate	03/08/2020
1876	Brookline	Speeding	Allstate	05/23/2020
1943	Brighton	Speeding	Allstate	06/01/2020
2054	Boston, South	Swerving	Emergency Services	07/04/2020
2098	Newton	Outdated Registration	MA RMV	09/18/2020
2256	Boston	Ran Stop Sign	Allstate	11/15/2020

For the frontend visualization, we wrote a dashboard in React. This dashboard is designed to be a frontend for the admin side of things from BLINKAH's perspective. The graphs are built with proper React modules that simply need to read the input from the backend, once we decide how we're going to set up the connection between the front and the back. Several important components that we decided to include in our first iteration of the admin dashboard are:

- Database storage capacity status
- Total police budget saved
- Fixed github issues
- Total users
- Daily unsafe driver reports (graph)
- Monthly unsafe driver reports (graph)
- Prevented police stops (graph)
- Tasks visualization

- Most unsafe states broken down with # of daily reports per hundred, percentage of uninsured drivers, and % repeat offenders in table format
- License plate lookup functionality in table format
    - Report ID
    - City
    - Action
    - Report Entity
    - Date

Timeline

Overall, we are making steady progress towards our goal of completely agile development. We will cover this topic further in our discussion.

Discussion

We received praise from Professor Hirsch on our progress so far. Here are our MVP goals with the components that have shown progress bolded:

- 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 can also show our system diagram to date. Note that there has been healthy work done. As our goal is to lay the groundwork to transition to agile methodologies completely, we have isolated applications with very little done on the BLINKAH cloud that will become our focus in the future.