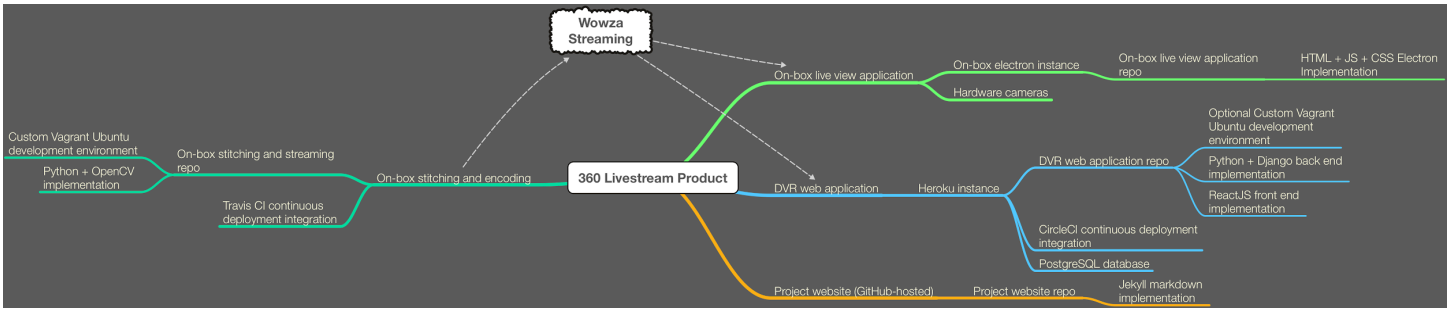
360LifeStream Design Doc - Sprint 5

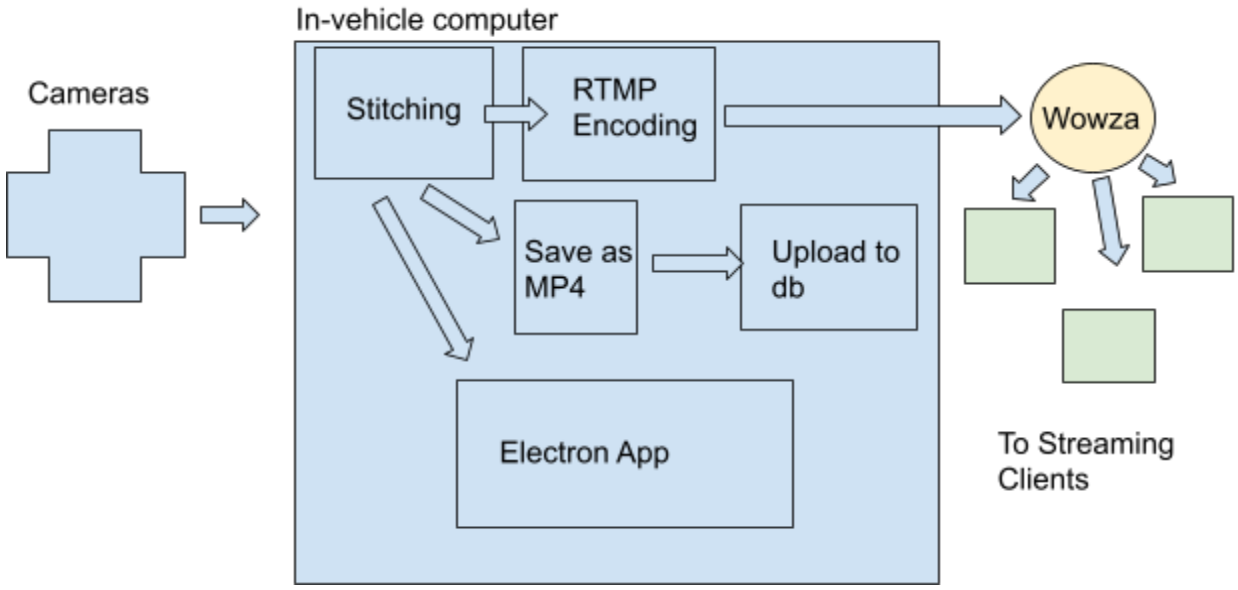
Authors: Luke Fernandez, Dong Lee, Spencer Lewis, Jonathan Witten

Overall Project Architecture

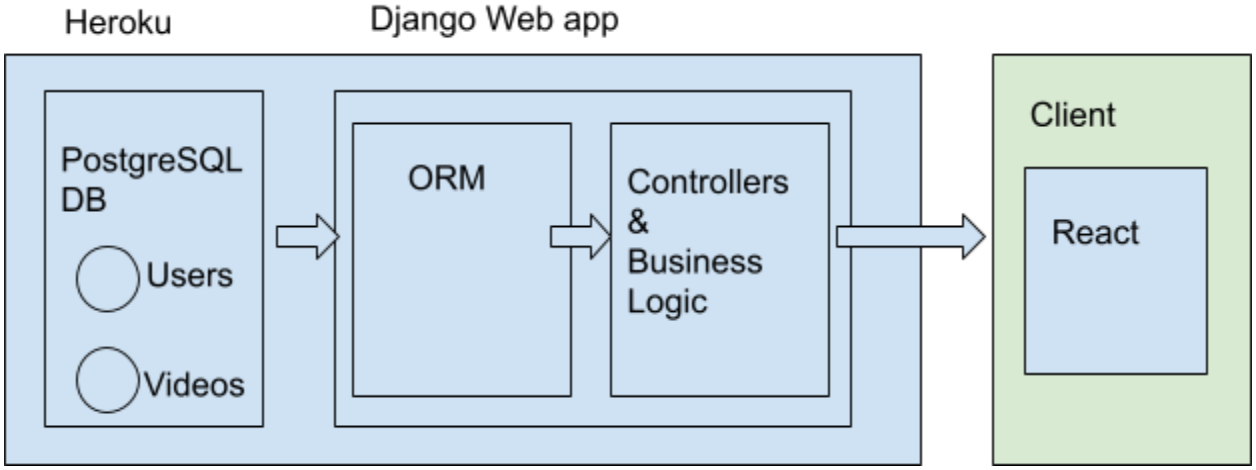
The below diagram illustrates our current understanding of the total project architecture. It's big, and we have taken that into account during sprint planning to make sure we are on track toward reaching our goal.



Box Architecture



Web App Architecture



Decomposition

Modules

- Device
    - Camera Stitching
        - Video coming in from each of four cameras must be stitched together to create one 360 degree image
        - Designed so that input comes from four cameras, is stitched on the device into one extended image, and then that image is encoded into a workable format for streaming.
        - The camera stitching platform can take in both camera streams and local videos, as well as deal with local images.
    - RTMP Encoding
        - We use ffmpeg to stream stitched frames to our Wowza Streaming Engine instance using RTMP
    - Wowza streaming
        - We are hosting an instance of Wowza Streaming Engine on an AWS instance
        - Wowza is a third party service that will allow viewing of our live streams on several distributed devices. It requires an RTMP stream and can then output a controllable 360 degree video to be viewed on the web app
    - Device file storage and upload

        - As videos are recorded, they must be saved as .mp4, and when the recording is done, they must be uploaded to our database so the can be viewed on the web app
    - Electron Frontend
        - An Electron frontend will allow a touch controllable interface for an officer to see the current video stream and view locally stored recordings
    - Ansible provision playbook to setup desktop environment
        - This is a repeatable installation script to install the dependencies required on the desktop to run our stitching and video file writing code
    - Continuous Integration
        - All of our device repos including stitching and the electron app are linked to TravisCI to build and deploy our projects.
- Web app
    - Database
        - A PostgreSQL database will store user and video information so that users can login and view live streams or videos to which they have access
    - ORM Layer
        - This will allow us to access our database information in the form of Python objects that will be easy to use and control
    - Controllers and Business Logic
        - Handling HTTP requests (including AJAX), database calls, and business logic, these backend pieces will help connect the correct models and information to the client interface
    - React Frontend
        - A JavaScript React front end is used to build a responsive, asynchronous frontend
    - S3 File Storage and Retrieval
        - Video files are stored in AWS S3 buckets
    - Player
        - We are using the bitmovin player (as justified in our design decision section) in order to support live 360 degree video playback.
    - Platform
        - The webapp is hosted and built on Heroku.
    - Continuous Integration
        - The webapp is linked to CircleCI to build and deploy the project.

Detailed Module Definitions

- Stitching
    - The stitching package of the application is responsible for the following things:
        - Grabbing multiple camera streams from input

        - Applying distortion correction to camera stream frames
        - Stitching corrected frames together
        - Sending data output to ffmpeg for streaming to the streaming engine mentioned below.
    - The stitching package, today, has the capability to:
        - Stitch from 2 cameras
        - Stitch from 2 corrected cameras
        - Stitch from 2 videos
        - Stitch from 2 corrected videos
        - Stitch from 4 videos
        - Stitch from 4 corrected videos
        - Stitch and stream 2 videos
        - Stitch and stream 2 corrected videos
        - Validate streams for setup
    - The stitching package also has the ability to calibrate cameras based on a set of test images
    - Correcting the radial distortion of incoming frames is computation-intensive, so we have decided to leverage some sort of threading
- RTMP and Wowza Streaming Engine
    - We use ffmpeg to stream stitched frames to our Wowza Streaming Engine instance with RTMP
    - A Wowza Streaming Engine virtual machine image is hosted on a t2.small Elastic Compute Cloud Amazon Web Services instance.
    - Port 1935 is open to receive incoming RTMP streams
    - Guides referenced:
        - http://www.wowza.com/resources/WowzaStreamingEngineforEC2_UsersGuide.pdf
        - https://www.wowza.com/forums/content.php?22-quick-start-guide
- Achieving Low Latency
    - In order to achieve low latency live streaming, the following techniques must be used
    - RTMP and WebRTC are the only two protocols for streaming video data. Apple HLS is sent over HTTP and cannot be used for true low latency streaming.
    - The Wowza Streaming Engine **StreamType** for the **live** application must be configured to **live-lowlatency**
    - Other concerns are audience size and geographic scope, but we are not ready to act on those at this time until demand requires it. Our Wowza Streaming Engine server is hosted in Virginia, which is geographically as close as we can get to our development location of Chapel Hill, North Carolina when hosting on AWS.
    - Guides referenced:
        - https://www.wowza.com/forums/content.php?81-How-to-achieve-the-lowest-latency-from-capture-to-playback
        - https://www.wowza.com/blog/what-is-low-latency-and-who-needs-it

- Streaming to the Web App
    - Wowza Streaming Engine is configured to stream using the MPEG-DASH format, a format that is supported by the Bitmovin player we are using. Embedded on our web app, the Bitmovin player allows view and control of 360 degree video.
    - The streaming engine takes the incoming RTMP stream and converts it to MPEG-DASH by way of creating an MPEG-DASH Media Presentation Description. This XML documents provides information to the player of how to serve the media.
    - Guides referenced:
        - https://www.wowza.com/forums/content.php?715-How-to-use-Bitmovin-Bitdash-players-with-Wowza-Streaming-Engine-(MPEG-DASH)
- Electron App
    - Built using React for declarative UI.
    - Declarative state mutation with the Redux library.
    - Webpack module bundler is used to create development/production builds of the application.
    - Guides referenced:
        - http://redux.js.org/docs/basics/
        - http://webpack.github.io/docs/tutorials/getting-started/
- Video storage and retrieval
    - Storing video files in the database would be slow and clunky, so we are only storing their URLs in our DBMS
    - Video files are kept in Amazon S3
    - When a user uploads a video from the desktop app, it is sent to the web app which is the only component of the service that knows about the existence of S3 in order to reduce coupling.
    - Additionally, since keys are needed to upload and retrieve from S3, it is better that only the server-side of the web app communicate with S3 as opposed to all clients.
- Ansible provision playbook to set up desktop environment
    - Our stitching and video writing code that needs to run on the desktop requires a lot of dependencies. OpenCV, FFMPEG, and video codecs, to name a few, are all required in order for the desktop client to function.
    - In order to ensure that all of the computers that will sit in police cars can all work, we have created this package of provisioning scripts to install and setup the required dependencies on a given Linux Ubuntu OS.
    - It has one setup shell script to manage filesystem setup tasks and one script for installing and building FFMPEG from source and its associated dependencies.
- Web app 360 degree live video
    - To support 360 degree live video streaming, we currently imbed an instance of the Bitmovin player on the single page app. When the app is in live mode, it looks for an MPEG-DASH (Dynamic Adaptive Streaming over HTTP) Media Presentation Description.

    - The MPEG-DASH Media Presentation Description, or .mpd file), is an XML document that contains information about the media segments, their relationships, and how the player knows which to play and in what order.
    - The .mpd file is serviced by our instance of the Wowza Streaming Engine
    - JavaScript code is used to configure the player on the frontend, and it must be configured to support 360 degree video
    - Reference this guide to configure the player:
        - https://bitmovin.com/tutorials/vr-360-video-encoding-playout/
- Web App Frontend
    - The app is a single page with navigation bar at the top and a player below
    - The navigation bar allows switching between live and dvr playback modes
        - Selecting one will unload the current video source from the player and load the desired video source
        - Selecting DVR will also change the display of a table below the player from hidden to visible
        - The table shows the different videos in the production database and clicking one will unload the current video source from the player and load the new video source
    - Jquery is used to hand click events video player configuration
    - Standard html and css are used to create the structure and style of the page
    - Template rendering
        - Django renders the html file when a user navigates to the root url of our web app
        - When Django renders it, it passes in the list of videos from the database
        - The template can interact with those video objects on the front end, and that feature is used to populate the table that provides information about the videos
        - The video data is also used to set an attribute on table rows that allows jquery to pick up the video url on click so that it can set the correct video source in the player
- Continuous Deployment/Integration
    - The device projects such as the stitcher and electron desktop app are linked to TravisCI to test, build, and deploy the projects on merges to the master git branch.
    - The webapp is linked to CircleCI to test, build, and deploy the project on merges to the master git branch.
        - Because Django does not deploy static files by default for deployment reasons, our circle.yml file that configures our CircleCI deployment has been configured to run a Django command line script to deploy the static files.
- Web app platform
    - The web app is hosted on served on Heroku.

    - Heroku allows extremely easy and painless deployment at the expense of fine grained control that one might enjoy hosting on something like Amazon EC2. More information can be found in the design decisions portion of this document.
    - The primary configuration that must be performed is configuring environment variables to allow access to S3. Information on how to do that can be found in our document that describes how to access S3.

Data

Our database will be centered on the relationships and fields of the users (officers and police leadership) that can record and access recordings and live streams online, as well as the videos themselves.

- Users
    - We will have a users table to encapsulate information about the officer and keep a unique primary key for each user
    - Users will be related to each other hierarchically through another table that shows who leads whom. This will allow quick lookups of who a officer on the site leads and therefore what they can stream and what videos they can view
- Videos
    - The videos table will store recorded videos with fields such as time and date of recording, a foreign key for the officer logged in, and a flag that signifies an officer thinks the video is important
    - It contains a url field that points to the location in the S3 bucket where the file is stored and can be retrieved.
    - A VideoCase table will associate videos with open cases so that they can quickly be found and referenced in the pursuit of criminal justice.
    - There will be a table VideoViews that will contain a foreign key user id and a foreign key video id to track who has viewed each recording.

Detailed Data Definitions

Database System

- Our tables will exist in a PostgreSQL database

Schema

- The schema diagram can be viewed here

Design Decisions

- From C++ to Python

    - We originally planned to do stitching and encoding on the in-vehicle device using C++. By using C++ we hoped to maximize performance by minimizing live-streaming latency. However, in order to maximize developer productivity and avoid the pitfalls of premature optimization, we have decided to write that code in Python.
- Using an off-the-shelf 360 live streaming service
    - We have decided to use Wowza, a service that allows streaming and viewing of 360 degree video content. It requires us only to send our stitched frames over RTMP. We will not perform further encoding ourselves in order to save developer time/effort and improve our ability to iterate and prototype quickly.
    - Additional support for Wowza after sprint 4:
        - Because our Wowza Streaming Engine App went down a couple of times, we looked into two alternatives: Bitmovin's cloud encoding and Wowza Cloud.
        - Wowza Cloud only works with basic HTML5 players, and does not support MPEG-DASH needed to live-stream 360 content on the Bitmovin player that we justify support for below.
        - Bitmovin's cloud encoding does not support live streaming in the same capacity as Wowza, but is instead better for playback of pre-recorded video files.
        - Additionally, we created a document showing how to SSH into the AWS Wowza instance so that the app can be restarted and it will stay working.
- Hosting Wowza on AWS
    - Wowza has preconfigured virtual machine images for four different cloud computing providers: AWS, Google Cloud Engine, Microsoft Azure, and Rackspace. We opted to use one of those offerings so that we would benefit from that preconfigured image. Out of those offerings, prices were similar, so we opted for AWS which has substantial documentation from Wowza supporting its use.
- Bitmovin player
    - To perform 360 degree video playback, we opted to embed an existing player into our web app rather than writing our own. We chose to do this to avoid reinventing the wheel and to be able to focus our development efforts elsewhere. We looked at three players, the Wowza player, the JW player, and the Bitmovin player. The Wowza player is not fully featured as a 360 degree player, although it does very easily integrate with the Wowza Streaming Engine we are already using. The JW player is only a prototype, so we do not have the confidence in it we would like. The Bitmovin player does fully support 360 degree video playback over MPEG-DASH, a format supported by Wowza Streaming Engine, and it was our choice.
- Amazon S3 for video file storage
    - Storing files on a DBMS offers some benefits such as the ACID consistency in various situations including a rollback would ensure that our database and files are always in sync

    - However, there are also many drawbacks to that. Performing a `SELECT` on the file column would always involve disk reads and bringing data into memory unnecessarily. Additionally, database backups and restores will take longer
    - Storing the files in S3, the AWS Simple Storage Service does not impact our file retrieval performance and also handles storage without database performance degradation. Therefore, it was our choice.
- Heroku
    - Heroku provides an additional level of abstraction for hosting our web app over something like an Amazon EC2 instance.
    - The primary benefit is ease of use. It is extremely easy to start and deploy to, and to connect to continuous delivery/integration systems. One does not need to deal with a lot of infrastructure concerns to get it up and running, and there are free tiers.
    - The drawback is control. With the ease of deployment we do not have the fine control over hardware, OS, etc. that we might otherwise have. Additionally, for high volume applications we may be charged a premium.
    - These drawbacks were seen as inconsequential and to take them into account was deemed premature optimization by the team. We optimized for ease of development.