Assist MD - Product Requirements Document

Team name: High Voltage Society
Team lead: Brian Humphreys
Team Scribe: David Roster
Andrew Laux
Ram Malyala
Siddharth Malik

Table of Contents:

Assist - MD: Introduction **4**
GOALS: 6

System Architecture Overview **7**

User interaction and design **8**

Project Milestones and Objectives Outlined **9**
High-Level User Stories and Acceptance Criteria: 9
iOS App for Data Streaming - 9
S3 Triggered Lambda Function facilitating EC2 - 9
Lambda SSH and Darknet Execution - 10
Lambda AWS SDK Prediction Upload to S3 - 10
Web App Request for S3 Resources - 10
Locally Train a YoloV2 Model - 11
Historical Surgeries Tab - 11
Live Surgeries Tab - 11
Lambda Function to Extrapolate Useful Data - 11
Downloadability of Compiled Data - 12
Local Server to Access S3 Resources - 12
Local Server to Perform Darknet Inferences - 12
Local Server Inference Upload - 13
Lower-Level Use Cases - User Stories Broken Down: 13
Build Simple iOS Platform that allows for video and image capture 13
Send Visual Data from HVStream to Amazon S3 14
Create Lambda Function Triggered by S3 Object Creation 14
Lambda Function to Prepare URL data for EC2 Commands 14
Integrate Simple-SSH in to Lambda and Create SSH Object 15

AWS EC2 instance properly calls darknet and generates a prediction after being called from AWS Lambda 15
If videos are available in S3 buckets, add to a surgery card to Historical View 15
Make videos expandable upon clicking 16
Live upload to our Web Application using HVStream 16
Lambda function using Panda to compile Time Series 16
Be able to to make video inferences using a Yolov2 model on EC2 17
Calls to darknet on EC2 should utilize GPU acceleration for improved performance 17
Be able to train a darknet CNN model. 17
Be able to make inference calls to a YOLOv2 model trained on preliminary dataset. 18
Be able to run tests in Xcode to test HVStream application. 18
Lambda AWS SDK Prediction Upload to S3 . 19
Lambda SSH and Darknet Image/Video Execution . 19
Lambda Function triggered by Web App to download PowerPoint of Surgery Stats 19
Send live video data from HVStream to Amazon S3 19
Continuously query S3 to get newly uploaded files 20
Execute darknet on image file and generate prediction file on local server 20
Execute darknet on video file and generate prediction file on local server 20
Upload prediction file on local server to aws S3 bucket 21

UML Diagrams 22
React Web App UML 22
Stand-in Server UML 23
iOS UML Diagram 24
AWS Server Interactions with iOS App and Web App: Overall System UML 25

Sequencing Diagrams 26
Image Inference Sequencing 26
Live Video Inference Sequencing 27
Boto Server Sequencing 28

Assist-MD Technical Implementation - Prototyping 29
Github: 29
iOS interfacing with HVStream: 29
Amazon Web Services (AWS) Backend: 30
Image Recognition Component - YOLOv2 Network Structure: 31
Front End Assist-MD Web App - ReactJS: 32

Appendices 33
What Assist-MD Does Not Do 33
Technologically: 33

Surgically: 33
Technologies Employed 33

Assist MD - Product Requirements Document

Assist - MD: Introduction

"The practice of Medicine, and in particular Surgery, is a continually evolving science. We speak of surgery as "being performed," as though it were a symphony. Technologies and techniques continue to emerge that improve outcomes, health results, and patient experiences.
I discern several critical values in the use of Assist MD algorithms."
- Dr. Brian Humphreys Sr.

Arthrex is a company which provides services, products, and technical innovation for medical professionals. Their enterprise research team has asked our group to implement a machine learning framework that is capable of processing and classifying video from their 4k surgical camera as well as any other cameras present in the operating environment. Specifically, they would like software that can, among other things, recognize and track medical personnel, recognize and track surgical tools.

Towards this end, we will gather test data which will be images of tools and medical personnel that we can use to train a computer neural network. The goal is to develop software that will be able to make real-time inference calls on images from operating room video and relay that information to medical personnel. Accurate processing of this data will allow us to solve a range of problems facing medical personal which he hope to explore. It will also allow us to generate post-surgery reports that will streamline important data collection and booking keeping in the operating environment that will allow personnel to focus on important tasks at hand.

There are many perspectives from which a product such as Assist-MD can be found vital, many of which mentioned in a written review by Dr. Humphreys. Dr. Humphreys believes Assist-MD can be found critical in the medical community in the following ways: teaching surgery, reducing surgical times, tracking key elements of surgery, use in critical surgery, evaluating new surgical techniques, interfacing with robotic surgeries, malpractice legal defense.

Medical Malpractice is a huge issue in the medical community which is the leading cause of death behind heart disease and cancer. To prove malpractice occurred during a surgery, an accuser must prove that a doctor provided inadequate care and an injury occurred with damaging consequences. With the implementation of Assist MD, both malpractice and defense attorneys can use footage to better support their claims. The footage will provide a basis for both party's claims because a doctor's past surgical history can be presented as evidence.

As medical class's size get more substantial, students are unable to get as much one on one time with the professor. Teachers are unable to give enough attention and time to students so

this lack of intimate learning pushes students to study techniques and procedures on their own. In an article titled “Teaching Surgery to Medical Students” by W Brian Sweeney, he states the disadvantage medical students currently have in today’s medical schools: “It is very clear that given the tremendous increase in medical knowledge, surgical technology, and intricate operative procedures, teaching surgery to medical students during this relatively short exposure has become an immense challenge.” Our increasing knowledge of the medical field has made it harder for students to keep up, but that is where Assist MD can make up the difference. Assist MD will provide more quantifiable metrics that students and professionals can judge their skills off and understand where they need to improve in. By providing increased analytics, Assist MD can help up and coming professionals build better technique and procedures for when they operate on a person in their future careers. By improving technical approach, Assist MD will also improve surgical times for students and medical practitioners. This extra time for students will allow teachers to spend other precious allotments reviewing concepts. Current doctors who see improved surgical times due to Assist MD can focus their time on other important matters of theirs.

The innovative features of Assist MD will see transformations in how we view surgery by developing new surgical techniques, advancements when performing crucial surgeries, and interfacing with robotics in the operating room. Currently, medical technology is advancing at a fast rate but proper documentation of new surgical techniques is releasing at a much slower rate. Assist MD’s live feed of a surgery where proper documentation is done at real time will provide enough evidence to release newer techniques to the industry than previously before. These new techniques can then be passed around to medical offices and be in practice quicker for critical surgeries where they could have a tremendous effect. Another positive of improving surgical times and tagging keys elements of a medical procedure is robotics surgeries already implement surgical cameras when conducting surgerys. By adding Assist MD to robotic surgeries, they will have lower risk of human error and produce a much safer outcome for the patient.

Assumptions that will be made are that the hospital has cameras in the surgery room as well as an account for our product. Doctors/medical staff must turn on each of the cameras and the cameras must be functional and positioned correctly to capture each feed. The timing of the cameras should synced. Another assumption is that surgical tools not on the table at the end of the surgery were used for the surgery. If tools are removed for any other reason (dropped by accident, misplaced, moved by a nurse etc.) then there will be an error as our service will assume those tools were used for the surgery.

Assist-MD will utilize video feeds that route to AWS S3 buckets and operate algorithms on said buckets in order to provide a live display of identified surgery equipment. This will be done by routing our S3 buckets to Sagemaker and running our algorithms on the data sent. We will have AI Models that are trained to recognize the instruments, and tag them as such. The output of the application will then be displayed on a web application with a slight delay to account for data transfer and processing times. At the end of a usage run, the program will generate an outline of the operation, showing tools, procedures, and holding the video for later use. Our algorithms can then be expanded to include personnel if time permits us, and potentially even identify people.

GOALS:

High Voltage Society is looking to create a product which will accurately identify medical instruments from a video feed, with the task being done as close to real time as possible. One of our main objectives is to first be able to upload the visual data to AWS servers. Once this is done, we can easily train Darknet algorithms to tune the weight file for our purposes. Once the weights are sufficiently tuned, we can upload this to a darknet repository cloned on AWS servers and execute object inferences using this constructed apparatus.

Our ultimate objective is to reach the state in which the Machine Learning models can differentiate between tools with a high accuracy. At this point, we will have a strong proof of concept and can begin streamlining and perhaps adding more features. One such feature would be to add the ability to download data extrapolated from a time series about the surgery in PowerPoint format with added pictures including bounding boxes of statistically interesting points in the procedure and their corresponding information. This information generator will be accessible at the front end of Assist-MD service on each surgery card where the surgery video contents are also displayed. This will help surgeons review details about the surgery they performed and help prevent any complications with information that may arise.

However, we wish to go above and beyond the requirements, and as such we have elected to add additional objectives to our project. One of these objectives is the aforementioned Surgery room personnel identification once the pipeline structure has been constructed and reinforced with machine learning model training. This data will be compiled on the same time series to which the instrument inference are mapped. Interesting data such as moments where the most and least medical staff are in the room will be included in the generated data file from the front end. We wish to start by simply identifying the surgeon and staff in the operating room. Then perhaps look into being able to identify certain people using pre developed frameworks and libraries. However, this option is dependent on ethical considerations. Completion of AI identification and statistical data points download objectives would result in the project itself being technically completed.

System Architecture Overview

The following UML design depicts where the flow of data starts and ends. Data is collected in real time from surgical cameras oriented to view capture visual data of medical instruments on a table. In order to simply the prototype, we have fashioned an iOS app to capture and send the data to AWS instead of using industrial cameras that require connection to a network. The iOS interface will simply stream live video to AWS where the data will get parsed into frames and feed into our YOLOv2 network. Once the network makes an inference on the individual pieces of data incoming from the stream, it will then connect the inferred data to a Live Video tab on the Web Interface which will provide the front end with a live visualization of instrument inferences in the form of bounding boxes being made on each frame of the surgical video data. In another tab that denotes 'Historical Inferences,' any and all videos of entire surgeries with visual inferences will be available to view and to generate PowerPoint files on their respective statistical extrapolations.

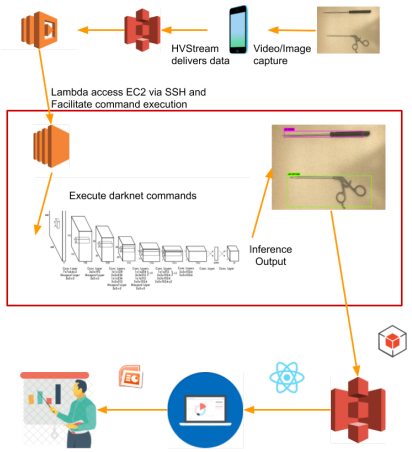


Diagram Legend:

Instrument Coverage: iPhone will be fixed on the treys where the surgeons place the instruments which are not in use. The process of elimination will be used to figure out what instrument the doctor is using

AWS S3: S3 buckets will be used twice here; once to stream unprocessed frames into and twice to place processed images in which the front end will query.

AWS Lambda: One lambda function will be triggered by the creation of an object in the unprocessed prefix of the S3 bucket. Whenever triggered, the function will create a Simple SSH object instance that will be used to facilitate command execution in a remote server.

AWS EC2: The remote server we are using is an EC2 instance that is configured with CUDA to run programs using GPU resources. YOLOv2 inferences are ran on this computing instance .

UI for Data Metrics: Once inferences have been made, the visual data and extrapolated will then be compiled into a time series and sent to the web interface to medical personnel usage.

PowerPoint Compilation: This is where the user will have the option to compile the main data points (i.e. maximum and minimum number of medical personnel involved at any given time, the instrument used most frequently, the instrument used least frequently) into a PowerPoint for post-surgical evaluation and teaching.

User interaction and design

The starting point of user interaction in this project is the start of the video feed capture. Once the iPhone cameras are all turned on, they will immediately initiate video feed streaming into AWS to be processed. All the clocks of the cameras are synced and data is sent with time stamps for the videos to be synced on a time series automatically. Assuming the user is a doctor working at a hospital that has an active AWS account the user may have to:

1. Create an iOS account that securely retains AWS credentials
2. Choose which account to stream video to
3. Give the stream a relatable title
4. Orient the camera towards the appropriate scene

Upon logging in, the user is prompted with a minimalist, monotonic screen to enter their credentials. When granted access, the user dashboard will appear. The initially opened tab will display aggregated company metrics which include(#3 & #4 features are assuming that we reach the point in the project when it is possible to make these inferences):

1. Aggregated number of company surgery streams that have been processed and stored in their library
2. Aggregated number of total instruments used overall video streams with a break down of how many of each kind of instrument
3. Average number of medical personnel in each surgery
4. Average number of people who exit and enter the Operating Room in each surgery

The user will also be presented with a tab that will route them to a page where they can search video streams by some affiliated attributes. The full video feed (initially ordered chronologically according to upload) will appear as a column of bars that display the video metrics. Once clicked on, the bar expands into half a page and displays the video stream in a time-navigable format. Each video displays the bounding boxes around the spaces where instruments were detected.

A search bar is available on the column view of the video search page to make queries of videos based on the following attributes of each video stream:

1. The video title
2. Names of Instruments captured in the video
3. Number of Instruments captured in the video or number of personnel involved and captured in the video (will be applicable if project becomes mature enough).

In the expanded video view that appears when a video bar is clicked, the user is also presented with the option to download a compiled version of the triple video stream capture which will be viewable in a PowerPoint. This will be viewed as a minimalist button in the bottom right of the full card view.

Project Milestones and Objectives Outlined

High-Level User Stories and Acceptance Criteria:

Every User story will display a live github link. If the User story has been completed, it will refer to the last Github link that pertains to the corresponding story. If the User Story is not yet completed it will link to the latest Github commit. If the story has not been started, it will contain no link in the given field. Github links in User Story section may overlap with the links in the Use Case section as the latest work done on the User Story could indeed be the completion of that Use Case. If there was no code and only AWS configuration involved, the field will denote this with 'AWS Configured.'

1. iOS App for Data Streaming -

   As a surgeon, I can capture video feed on the instrument coverage camera to be sent to AWS so that it will be available to be trained by the instrument inference algorithms.

   Acceptance: Data becomes available in a specified S3 bucket in an unprocessed prefix.

   Github Link: https://github.com/brianhumphreys/Assist-MD/commit/34ac2951183864aaec2a55504896902744969ec7

   Trello Card: https://trello.com/c/zee2sG4C

2. S3 Triggered Lambda Function facilitating EC2 -

   As the back end triggering service, whenever the S3 Bucket creates an object file that is in .mov or .jpg format, a Lambda Function will be Triggered in order to facilitate darknet execution.

   Acceptance: Cloud Watch begins to log data of the Lambda execution.

   Github Link: AWS Configured

   Trello Card: https://trello.com/c/HIf1ZBxM

3. Lambda SSH and Darknet Execution -

   As a Lambda Function, if I am triggered, I will ssh into a (running) EC2 instance and perform an inference using darknet algorithms on the image that cause the activation.

   Acceptance: An output image or video (depending on input) named predictions.png/mp4 is generated in the darknet root directory.

   Github Link:
   https://github.com/brianhumphreys/Assist-MD/commit/185fd0149517160f43faf24bf226bf9b54daf14a

   Trello Card: https://trello.com/c/X2fHhp5C

4. Lambda AWS SDK Prediction Upload to S3 -

   As a Lambda Function, once darknet has executed, I need to upload the generated image/video to the specified S3 bucket so the front end web app can have access to it

   Acceptance: An inferred image/video appears in the S3 bucket of prefix ‘processed.’

   Github Link:
   https://github.com/brianhumphreys/Assist-MD/commit/299ccb86891b0dd12863305800829f9f81bcfa10

   Trello Card: https://trello.com/c/0CEzP5cr

5. Web App Request for S3 Resources -

   As the front end React Web App with AWS credentials available, I can query the appropriate public S3 bucket for available objects in order to build image/ video URLs to display inferences.

   Acceptance: Images/Videos appear on the web app and the page does not remain blank.

   Github Link:
   https://github.com/brianhumphreys/Assist-MD/commit/ce78ce0806a2b78df0d4ae2103252ba7db5829de

   Trello Card: https://trello.com/c/EwGsGfpX

6. Locally Train a YoloV2 Model -

   Tag user generated images of tools. Format annotations so that the data can be used to train a CNN that uses darknet. The model can then be used in our system to recognize tools in images video.

   Acceptance: Obtain a useable model that makes correct inferences.

   Github Link: https://github.com/brianhumphreys/Assist-MD/tree/feature/andrew/LocalDarknetLinux/extra

   Trello card: https://trello.com/c/WT4eFtK1

7. Historical Surgeries Tab -

   As a Doctor/user, I can view image/video inferences of surgeries that have already occured in the Historical tab of the web app in order to review interesting statistical information

   Acceptance: The full card view displays important statistical information about the surgery and reasonably correct.

   Github Link: https://github.com/brianhumphreys/Assist-MD/commit/5173b3bfbd66be8de373ebbcdf0051d1034452e8

   Trello Card: https://trello.com/c/gKwqPiqx/68-historical-surgeries-tab

8. Live Surgeries Tab -

   As a Doctor/user, I can view image/video inferences of live surgeries in the Live tab of the web app in order to review real time object inferences frame by frame.

   Acceptance: A file is generated on your local machine in PowerPoint format and is populated with information such as most and least used instruments and time series charts.

   Github Link: Ready for Development

   Trello Card: https://trello.com/c/WsKHZcMD/69-live-surgeries-tab

9. Lambda Function to Extrapolate Useful Data -

   As a Lambda function, whenever an .mov object is created in a specified S3 bucket of prefix ‘processed,’ I will use the processed generated data to extrapolate interesting

information just as the most used instrument and the least used instrument and place the new data into the bucket.

Acceptance: A text file has been generated and place in the bucket of the proper prefix with data summarizing the surgery compilation.

Github Link: Ready for Development

Trello Card: https://trello.com/c/sdyFwHpk/75-lambda-function-to-extrapolate-useful-data

10. Downloadability of Compiled Data -

    As a Doctor/user, I can choose to download compiled data of existing surgeries in order to review the significant portions of the surgery offline

    Acceptance: A file is generated on your local machine in PowerPoint format and is populated with information such as most and least used instruments and time series charts.

    Github Link: Ready for Development

    Trello Card: https://trello.com/c/JWvNbjLJ/70-downloadability-of-compiled-data

11. Local Server to Access S3 Resources -

    As the stand-in server, whenever a new .jpg or .mov file is uploaded to the s3 bucket, it will be downloaded to the local machine so that darknet image classification can be performed on the file in case the AWS EC2 instance fails.

    Acceptance: Whenever a file is uploaded to the s3 bucket, it is downloaded to the local server.

    Github Link: https://github.com/brianhumphreys/Assist-MD/commit/76d9702fa183c79e467af95e0df396f4991b412f

    Trello Card: https://trello.com/c/obXNmOQB/71-local-server-to-access-s3-resources

12. Local Server to Perform Darknet Inferences -

    As the stand-in server, whenever a new .jpg or .mov file is downloaded from the s3 bucket to the local machine, darknet inference will be executed so that the file can be classified in case the ec2 instance fails.

Acceptance: Whenever a file is downloaded from the s3 bucket to the local machine, darknet inference is executed, resulting in an output file with the correct classification of the original file.

Github Link:
https://github.com/brianhumphreys/Assist-MD/commit/76d9702fa183c79e467af95e0df396f4991b412f

Trello Card:
https://trello.com/c/pCijPkNj/72-local-server-to-perform-darknet-inferences-on-video-images

13. Local Server Inference Upload -

As the stand-in server, whenever a .jpg or .mov file has been classified on the local machine, the resulting output file will be uploaded to the s3 bucket so that it can be accessed by the web app.

Acceptance: Whenever an output file is generated by the darknet execution, it is uploaded and appears in the correct s3 bucket.

Github Link:
https://github.com/brianhumphreys/Assist-MD/commit/76d9702fa183c79e467af95e0df396f4991b412f

Trello Card: https://trello.com/c/p4VAZ7tE/73-local-server-inference-upload

Lower-Level Use Cases - User Stories Broken Down:

1. Build Simple iOS Platform that allows for video and image capture

Actors: HVStream App, User
Precondition: User has an iPhone (only applicable in prototype)
Postcondition: Data is captured and stored in phone with a title chosen by user
Flow of Events:
1. User opens app on iPhone
2. User records a video or picture
3. Video/Image is stored on mobile device and user names the file

Alternative Paths: If memory on mobile device is full, user is notified.
Trello: https://trello.com/c/AydpirK6/15-send-video-feed-data-in-bulk-to-aws
GitHub:https://github.com/brianhumphreys/Assist-MD/commit/ee63bde023287bcddbc26a08e3f61cc9cfa1860b

2. Send Visual Data from HVStream to Amazon S3

   Actors: HVStream App, User, AWS S3 servers
   Precondition: User has opened the app
   Postcondition: Data is uploaded to S3 server and is available for manipulation
   Flow of Events:
   4. User selects their server. Application updates all variables
   5. User selects video or picture
   6. Application sends data to the selected server with HVStream library

   Alternative Paths: Specific server is offline. Data will fail to send.
   Trello: https://trello.com/c/gZIr8XMz/76-send-visual-data-from-hvstream-to-amazon-s3
   GitHub:https://github.com/brianhumphreys/Assist-MD/commit/34ac2951183864aaec2a55504896902744969ec7

3. Create Lambda Function Triggered by S3 Object Creation

   Actors: AWS S3 Servers, AWS Lambda
   Precondition: User has successfully sent a video/image file to the S3 prefix, unprocessed
   Postcondition: Cloud Watch begins logging Lambda function execution
   Flow of Events:
   1. File is uploaded to S3 bucket with prefix ‘unprocessed.’
   2. AWS servers create an Event, ‘objectCreated’
   3. This event spins up a lambda function to be executed

   Alternative Paths: If Lambda is not triggered, User is notified through Cloud Watch
   Trello:
   https://trello.com/c/5YvGQCZj/30-configure-api-gateway-to-handle-root-get-requests-and-to-return-either-the-video-or-a-link-to-the-video
   Github:
   https://github.com/brianhumphreys/Assist-MD/commit/ce78ce0806a2b78df0d4ae2103252ba7db5829de

4. Lambda Function to Prepare URL data for EC2 Commands

   Actors: AWS Lambda
   Precondition: Function has been triggered and file resources have been passed in
   Postcondition: URL is successfully constructed and Cloud watch reflects this
   Flow of Events:
   1. Function builds URL for the file created in the S3 bucket
   2. URL of created object is constructed from imported data

   Alternative Paths: If EC2 is not running or if darknet fails to make inference, Cloud Watch will notify user
   Trello:
   https://trello.com/c/77BVjIFg/77-lambda-function-to-prepare-url-data-for-ec2-commands
   Github:
   https://github.com/brianhumphreys/Assist-MD/commit/bf6eb0704e6d91f4584465be5aed0b4ce518e9cc

5. Integrate Simple-SSH in to Lambda and Create SSH Object

   Actors: AWS Lambda
   Precondition: Function has been triggered and and URL for file is constructed
   Postcondition: The Lamdba function gains acces to EC2 instance through SSH
   Flow of Events:
   1. Simple-ssh is imported and object is created with host name and permissions
   2. Execution commands are specified in object execution methods
   3. The SSH begins execution and specified tasks are automatically carried out

   Alternative Paths: If SSH fails to gain access to the EC2 instance then Cloud Watch notifies user
   Trello: https://trello.com/c/X2fHhp5C
   Github:
   https://github.com/brianhumphreys/Assist-MD/commit/bf6eb0704e6d91f4584465be5aed0b4ce518e9cc

6. AWS EC2 instance properly calls darknet and generates a prediction after being called from AWS Lambda

   Actors: AWS Lambda , AWS EC2, AWS S3.
   Precondition: AWS lambda code is active and EC2 instance has been created.
   Postcondition: predictions file is generated and put in the processed S3 bucket.
   Flow of Event:
   1. Lambda is triggered and begins to run commands on EC2 instance.
   2. EC2 runs darknet and yoloV2 with our weights file on the image.
   3. The output is generated from darknet and placed in the processed bucket.

   Alternative Paths:
   1. Lambda is configured incorrectly, and no output will be generated.
   2. Weights file is incorrect, and the output will have no identifications.
   3. Submitted file too unclear, yolo will have no way to interpret the file.

   Trello: https://trello.com/c/HIf1ZBxM
   Github:
   https://github.com/brianhumphreys/Assist-MD/commit/dd14fb8533c7f8c3b8ab7e1eb43dc5a5c775f18f

7. If videos are available in S3 buckets, add to a surgery card to Historical View

   Actors: Assist-MD Web App, User, React
   Precondition: Request from AWS has been received
   Postcondition: Surgery Card components are loaded with video thumbnails
   Flow of Event:
   1. If the Assist-MD app video state updates, create a surgery card component for each video that is included in the app state
   2. If there are too many surgery cards to fit on the page, app will incorporate scroll component

   Alternative Paths: If there are no videos in the app state, no surgery cards will appear on the page

Trello:
https://trello.com/c/N26hqSsn/81-if-videos-are-available-in-s3-buckets-add-to-a-surgery-card-to-historical-view
Github:
https://github.com/brianhumphreys/Assist-MD/commit/5173b3bfbd66be8de373ebbcdf0051d1034452e8

8. Make videos expandable upon clicking

Actors: Assist-MD Web App, User, React
Precondition: There are live surgery cards existent on the Assist-MD site and user is logged in
Postcondition: Surgery card expands to full view and video thumbnail extends to a full video window
Flow of Event:
    1. Upon clicking a Surgery card, surgery metrics are accessed and passed into the Full View component
Alternative Paths: If no metrics or only faulty metrics are available, display nothing
Trello: https://trello.com/c/WEvKdTPQ
Github:https://github.com/brianhumphreys/Assist-MD/commit/f7ea7990536e86b6ab518b705453df11e30f3e0e

9. Live upload to our Web Application using HVStream

Actors: Assist-MD Web App, User, HVStream
Precondition: HVStream is open and upload button has been clicked
Postcondition: The web app displays a live stream from the HVStream device
Flow of Event:
    1. HVStream uploads to S3 and lambda function activates
    2. Lambda function runs darknet analysis and posts the output to our web app.
Alternative Paths: No output. Potential lambda function failure or upload failure.
Trello:https://trello.com/c/aBp6yrwy/35-live-upload-to-our-web-application-using-hvstream
Github:https://github.com/brianhumphreys/Assist-MD/commit/ea8d46a2ce1f94c81f4c163197aa788a8b96b77f

10. Lambda function using Panda to compile Time Series

Actors: AWS Lambda, AWS S3
Precondition: The new object being created in the processed prefix.
Postcondition: The text file gets uploaded to S3 processed prefix.
Flow of Events:
1. A object gets uploaded to processed prefix.
2. Aws Server creates an object created event.
3.That object created event triggers the lambda execution.
4. lambda uses panda to extract minimum and max instruments in time domain.
5. Outputs a text file and puts it in the pre processed.

Alternative Paths:
1. If csv file is not present with a corresponding file, wait 5 seconds and trigger again.
2. if still no file, cloud watch will notify user.

Trello: https://trello.com/c/yIJhIUXO
Github: Ready for Development

11. Be able to to make video inferences using a Yolov2 model on EC2

Actors: AWS EC2, Darknet, YoloV2 model
Preconditions: EC2 must have openCV library installed locally. Darknet must be compiled on EC2 to utilize openCV. Model must be able to recognize tools correctly.
Postcondition: Program should output a predictions.mov with bounding boxes around tools in the video.
Flow of Events:

1. Invoke command ./darknet detector demo
2. Specify paths to .cfg .weights .data files.
3. Specify path to .mov file.
4. Darknet generates predictions.mov
5. Inspect .mov for acceptance

Alternate Paths: If openCV is not configured on EC2 darknet will report an error.
If the paths to necessary files are not correct darknet will indicate that they are missing.
If the model was not trained properly the bounding boxes will be drawn incorrect.
Trello: https://trello.com/c/KEoJyqL0
Github: AWS Code

12. Calls to darknet on EC2 should utilize GPU acceleration for improved performance

Actors: AWS EC2 instance, Darknet
Preconditions: EC2 instance type needs to include access to nvidia GPU cores. Must have cuda installed on the instance with cduNN library present as well. Darknet executable must be built with the features enabled.
Postcondition: Darknet was compiled with the correct flags set without error. be able to make inferences in milliseconds as reported by darknet.
Flow of Events:

1. Set flags in Darknet makefile to allow compilatoin with GPU and OpenCV
2. Remake the program and run the inference

Trello:: https://trello.com/c/fm2g4DeT
Github:

13. Be able to train a darknet CNN model.

Preconditions: Have a valid set of tagged images as data. Have darkent built with GPU acceleration. Have properly configured .data and .cfg files.
Postconditions: Have an avg-loss value of <0.1. .Weights files exist for each 100 iterations.
Flow of events:

1. Initialize training with specified data set.
2. Darknet makes reports at each iteration.End training before apprx. 4000 iterations or when avg-loss does not decrease.

Alternate Paths: If the data was improperly annotated darknet will fail and report an error. If darknet was not built with GPU acceleration iteration will take too long and training to a useable avg-loss will not be feasible. If the data set is of poor quality or nonsensical avg-loss will show no signs of steady reduction that we expect. If the data annotations are improperly formatted the training will fail to execute.
Trello:https://trello.com/c/WT4eFtK1/43-be-able-to-train-a-yolov2-darknet-cnn-using-preliminary-data-set

14. Be able to make inference calls to a YOLOv2 model trained on preliminary dataset.

Actors: Test images, YoloV2 .weights, darknet
Preconditions: Have darknet installed. Have a weights file that has been trained through a sufficient number of iterations.
Postconditions: Darknet reports correct inference with a high degree of confidence. Darknet outputs predictions.JPG. Visual inspections of predictions shows that boundary boxes were drawn correctly around objects we are looking for (surgical tools). Boundary boxes should not be drawn around objects it is not trained to recognize.
Event flow:

1. Select image from test set that was not used in training.
2. Execute command ./darknet detector test (…)
3. Darknet reports inference with confidence level.
4. Darknet correctly recognizes objects we trained it to identify.
5. Darknet does not recognize objects it was not trained to identify.

Alternate path: If darknet was not built the executable will not be runnable. If the training was insufficient or the object is too obscured darknet will fail to recognize.
Trello: https://trello.com/c/hWP6y4tp

15. Be able to run tests in Xcode to test HVStream application.

Actors: HVStream, Xcode
Precondition: Xcode is running and HVStream application is loaded.
Postcondition: All tests are run and return successfully
Flow of Event:

1. The testing conditions are activated and all testing functions run one at a time.
2. Xcode returns if all conditions were properly executed or not.

Alternative Paths:

1. Test failed. Something is wrong with the code. Fix immediately

Trello: https://trello.com/c/VxAYnzAn

16. Lambda AWS SDK Prediction Upload to S3 .

Actors: AWS Lambda, AWS S3, Darknet, YoloV2, AWS EC2
Precondition: AWS Lambda is running and unprocessed bucket is triggered.
Postcondition: Prediction is uploaded to processed bucket
Flow of Event:

1. Lambda runs the processing commands through EC2
2. EC2 runs darknet and Yolo, depositing final prediction in proper S3 bucket.

Alternative Paths:

1. No output. Lambda code possibly incorrect, or trigger not set.

Trello: https://trello.com/c/0CEzP5cr/65-lambda-aws-sdk-prediction-upload-to-s3

17. Lambda SSH and Darknet Image/Video Execution .

Actors:AWS Lambda, AWS EC2, AWS S3, Darknet, Yolo
Precondition: AWS Lambda is running and unprocessed bucket is triggered.
Postcondition: Darknet will run and generate a final prediction.
Flow of Event:

1. Lambda runs the processing commands through EC2
2. EC2 runs darknet, which triggers Yolo and begins executing.
3. Darknet generates a final output file and places it in a position such that Lambda can access it.

Alternative Paths:

1. No output. Lambda code possibly incorrect, or trigger not set.

Trello: https://trello.com/c/nlKUouBb/64-lambda-ssh-and-darknet-image-video-execution

18. Lambda Function triggered by Web App to download PowerPoint of Surgery Stats

Actors: User, AWS S3 Servers, AWS Lambda, Web App
Precondition: Surgery cards exist on Web App and User presses the download data button
Postcondition: PowerPoint appears in download folder of User's machine
Flow of Events:

1. User selects to download data
2. Lambda function is triggered and PPTX parses .txt file passed in
3. Parsed information is feed into PowerPoint on with python-pptx
4. Web download of PowerPoint occurs

Alternative Paths:

1. If web download fails, lambda will attempt once more. If it fails again, Cloud Watch will notify user
2. If .txt file is not present or is corrupted for corresponding video file, Cloud Watch will notify user.

Trello: https://trello.com/c/TGhBKY6h
Github: Ready for Development

19. Send live video data from HVStream to Amazon S3

Actors: HVStream App, User, AWS S3 Servers

Precondition: User has opened the app
Postcondition: Data is live streamed and uploaded to the S3 servers, ready for real time manipulation by the Assist-MD algorithms.
Flow of Events:

5. User selects their server and clicks live stream button
6. User begins live streaming.
7. Stream is uploaded directly to AWS servers.

Alternative Paths:

3. Specific Server is offline. Data will fail to send and app will stop the stream.

Trello:

1. https://trello.com/c/9HYVxrRh

Github: Ready for Development

20. Continuously query S3 to get newly uploaded files

Actors: boto script, AWS S3
Precondition: The specified S3 bucket exists
Postcondition: If new file is created, download it, if not, keep querying
Flow of Event:

1. Query bucket for newly created file
2. If the file exists, use AWS-SDK CP command to download it locally

Alternative Paths:

1. If file exists and does not download, Boto will retry three times and return error

Trello: https://trello.com/c/obXNmOQB
Github: https://github.com/brianhumphreys/Assist-MD/commit/76d9702fa183c79e467af95e0df396f4991b412f

21. Execute darknet on image file and generate prediction file on local server

Actors: boto script, Darknet, YoloV2
Precondition: A .jpg file is downloaded to the local server
Postcondition: A prediction .png output file is generated on the server
Flow of Event:

1. Once the file has successfully download, darknet will take over and perform inference on the file
2. Once the output is generated, AWS SDK will copy the output to the processed prefix

Alternative Paths: If darknet or AWS SDK fails, a catch block will be implemented to handle these errors and the system will be notified of the fault
Trello: https://trello.com/c/OKsJ8fJ8
Github: Ready for Development

22. Execute darknet on video file and generate prediction file on local server

Actors: boto script, Darknet, YoloV2
Precondition: A .mov file is downloaded to the local server
Postcondition: A prediction .mov output file is generated on the server

Flow of Event:

3. Once the file has successfully download, darknet will take over and perform inference on the file
4. Once the output is generated, AWS SDK will copy the output to the processed prefix

Alternative Paths: If darknet or AWS SDK fails, a catch block will be implemented to handle these errors and the system will be notified of the fault
Trello: https://trello.com/c/pCijPkNj
Github:
https://github.com/brianhumphreys/Assist-MD/commit/76d9702fa183c79e467af95e0df396f4991b412f

23. Upload prediction file on local server to aws S3 bucket

Actors: boto script, AWS S3
Precondition: An output .png or .mov file exists on the local server
Postcondition: The output file is in the correct S3 bucket under the “processed” prefix
Flow of Event:

1. Once an output file is generated by darknet, it is present on the local server and uploaded to the the S3 bucket under the “processed” prefix
2. The prediction file is available in the correct aws S3 bucket under the “processed” prefix.

Alternative Paths:

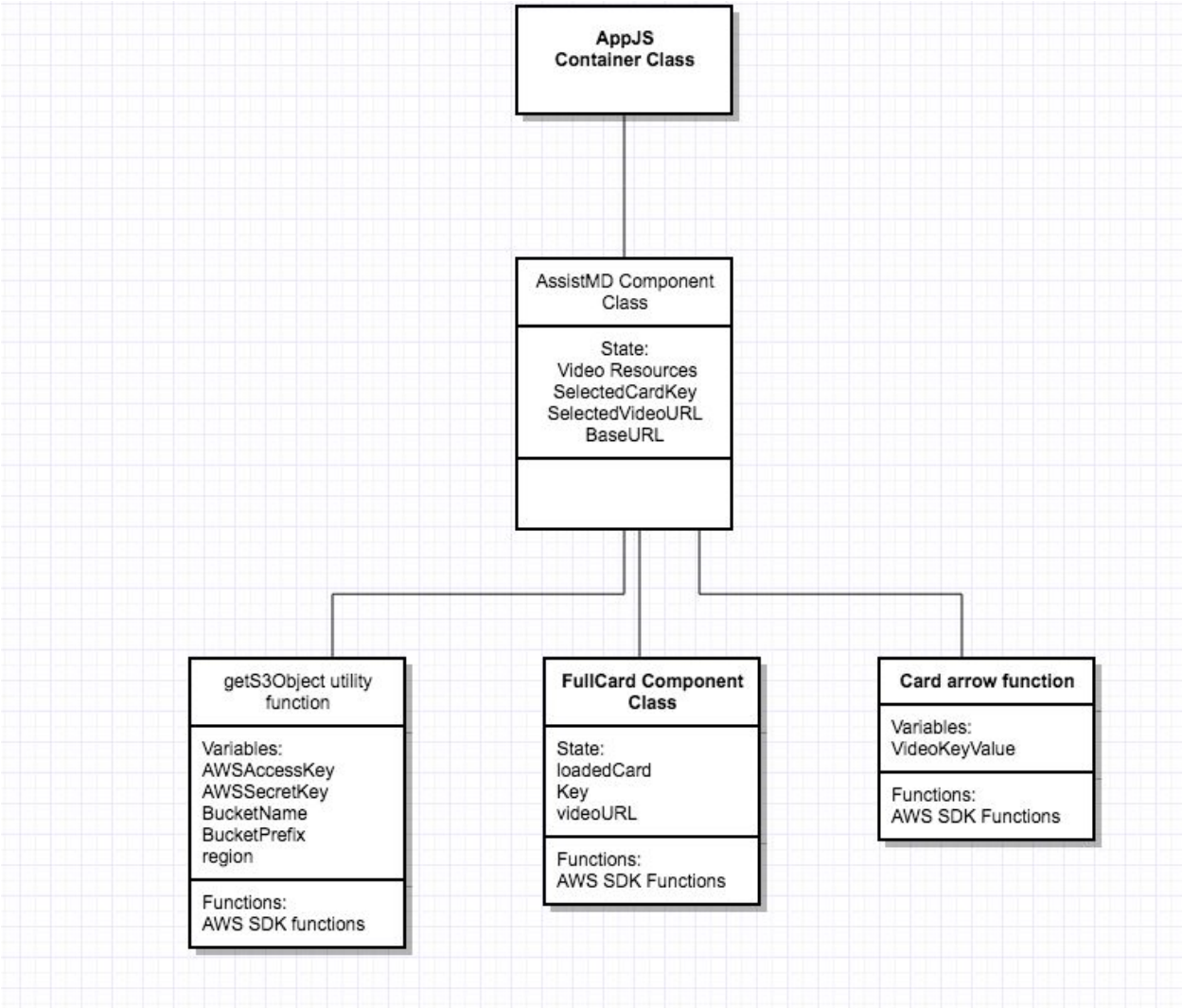
1. If the prediction file does not exist on the local server an error is output to the console.
2. If the upload of the file to the correct bucket or prefix fails, an error is output to the console.

Trello:: https://trello.com/c/p4VAZ7tE
Github:
https://github.com/brianhumphreys/Assist-MD/commit/76d9702fa183c79e467af95e0df396f4991b412f

UML Diagrams

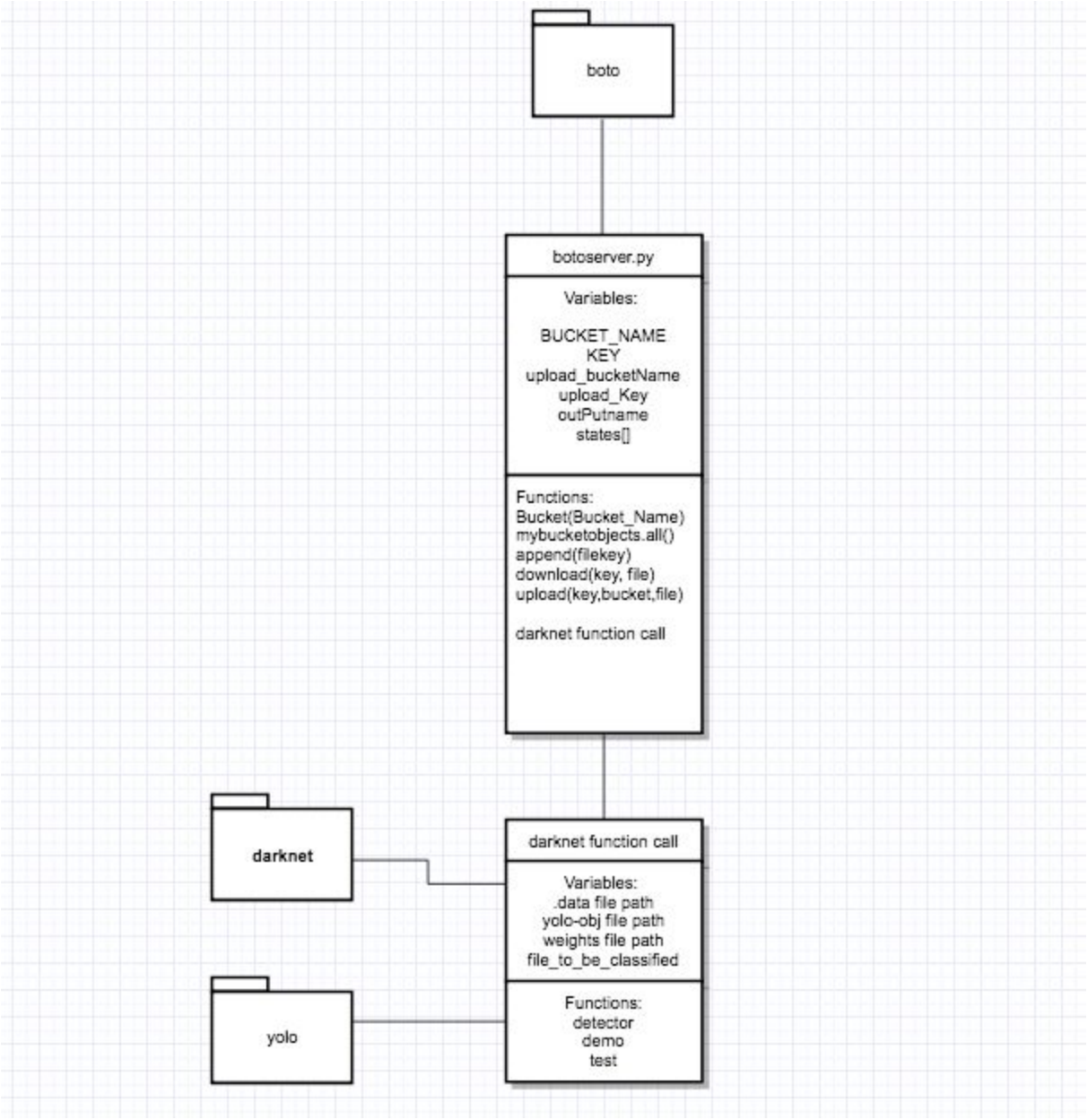
React Web App UML

Our AppJS file is are overarching class that we used to structure the webpage Assist MD. AppJS is in charge of rendering each of AssistMD Component Classes with the states: “video resources”, “SelectedCardKey”, “SelectedVideoURL”, and “BaseURL”. The Assist MD utilizes the *getS3Object Utility Function* to pull in the videos/images from our S3 buckets. Assist MD renders both the *FullCard* which is a *Component Class* and the *Card Arrow* which is a *Function.* They display each video’s title and statistics of each one as well as being responsible for formatting the webpage.
Web App UML diagram:



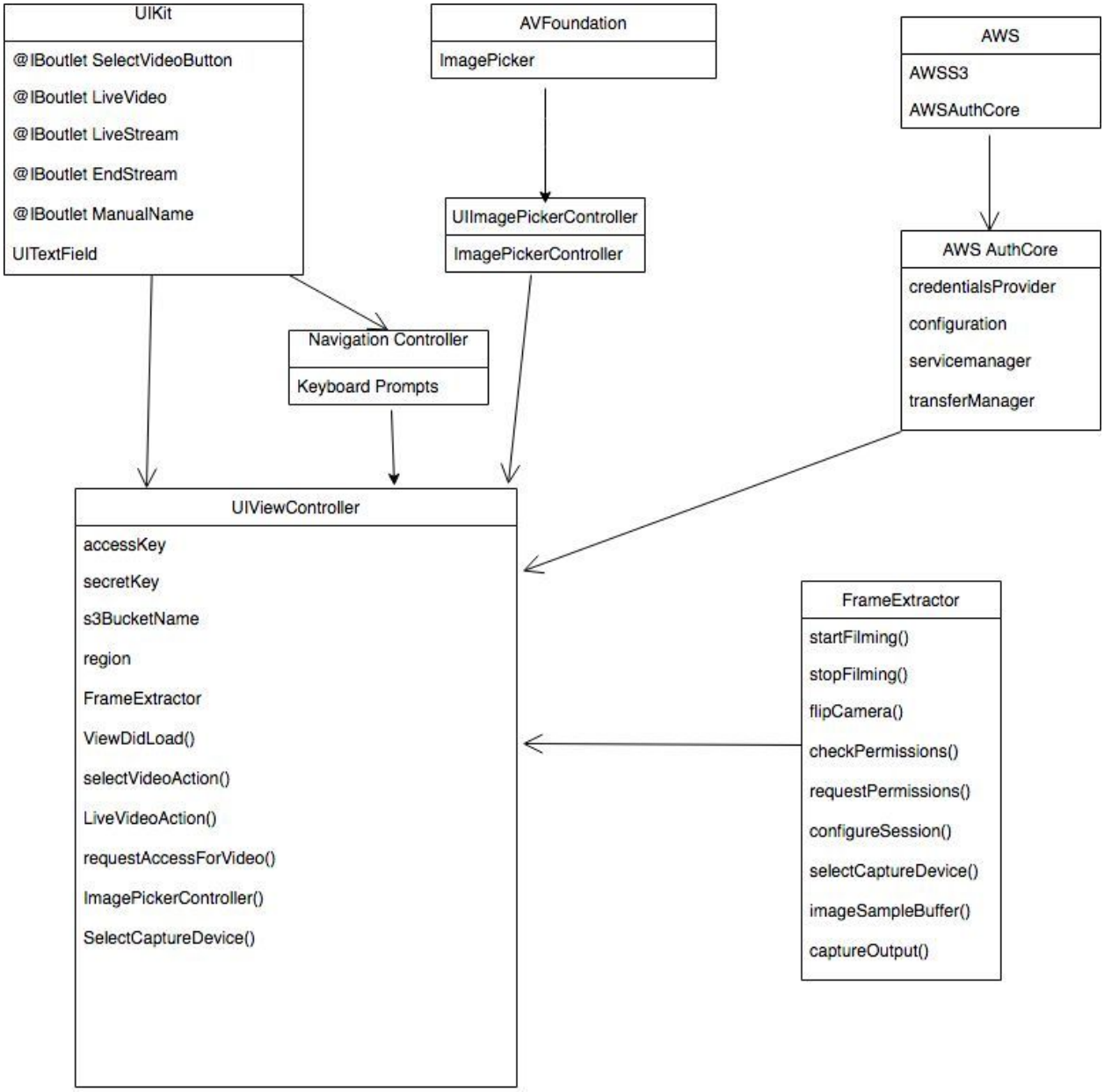
Stand-in Server UML

The stand-in server is used as a substitute for the aws ec2 instance in case the instance fails. The server downloads files from aws bucket, classifies them with a call to darknet, and then uploads them to processed aws bucket. The server has variables corresponding to the aws bucket names, keys, file names, and a list of states which keeps track of files downloaded to the local machine. Functions include uploading, downloading files and accessing buckets. There is also a darknet function call which makes use of the darknet api and yolo to do the classification.



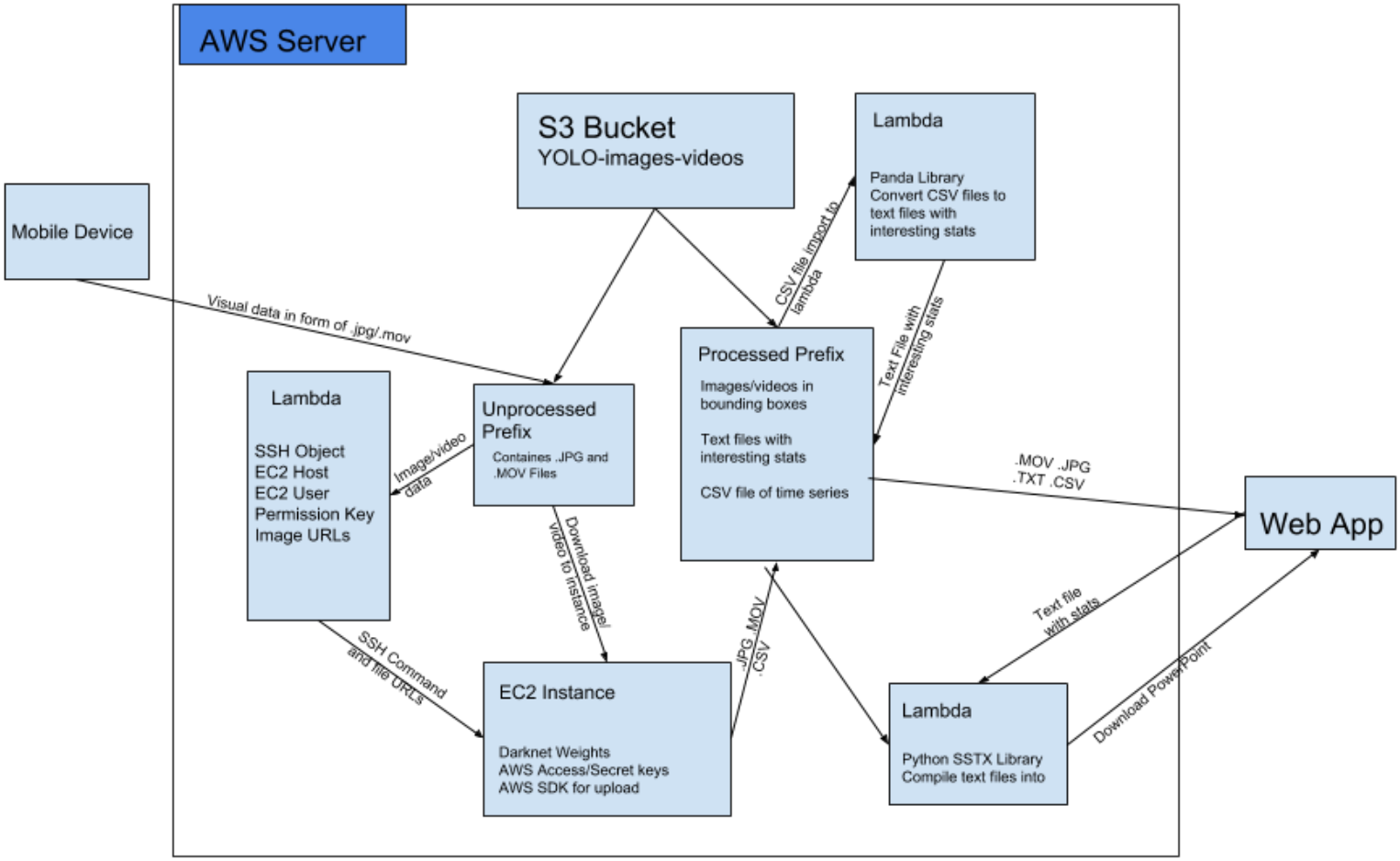
iOS UML Diagram

UIViewController controls the entire app, and all the variables and classes are imported and created in this class. The AWS classes are imported to the viewController, and their functions are used in UIViewController to access AWS services. UIKit contains the basic buttons and fields that all touchscreen applications have, and UIViewController implements the variables listed under UIKit. AVFoundation contains the image and video handling services that we use as containers to send to AWS. The FrameExtractor class contains our prototype live services.



AWS Server Interactions with iOS App and Web App: Overall System UML

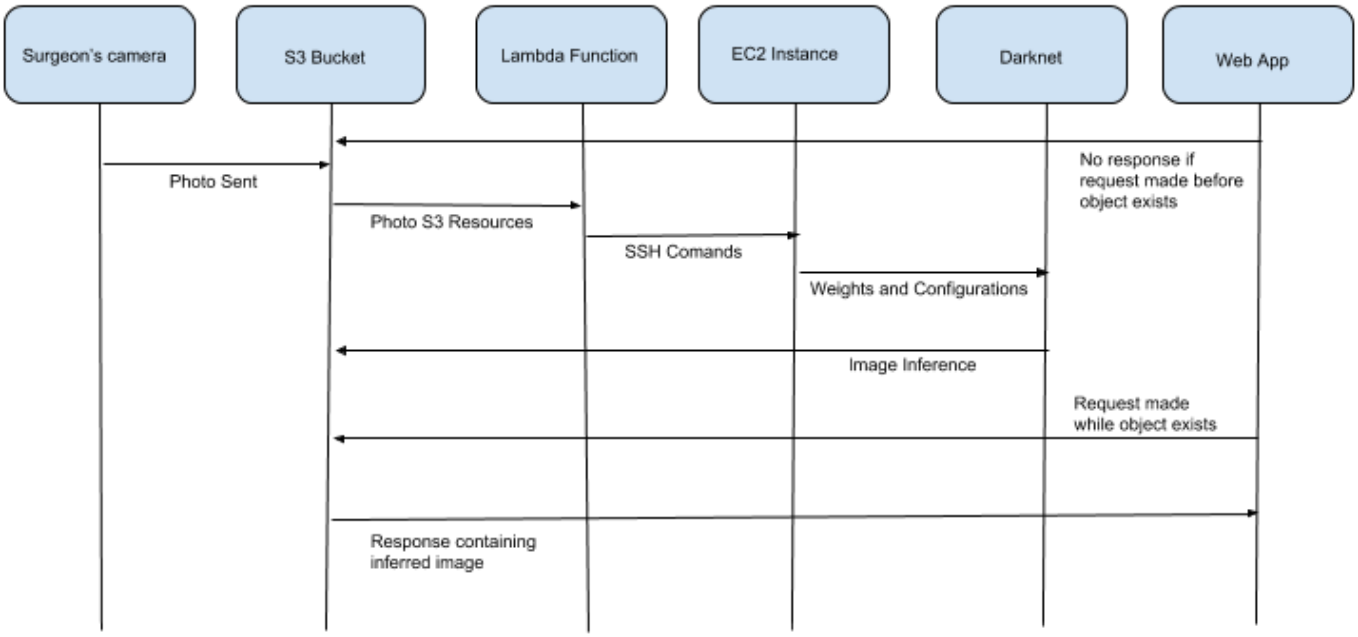
The following UML diagram Is slightly different from the previous in its content and design. The blue boxes within the AWS Server box represent AWS Services. The information inside each box Is variables contained in the service or information that is stored within them. The arrows denote what information is being passed between two services. The Picture also shows how the AWS server frame work reacts with the two front end components whose break downs were described in detail above.



Sequencing Diagrams

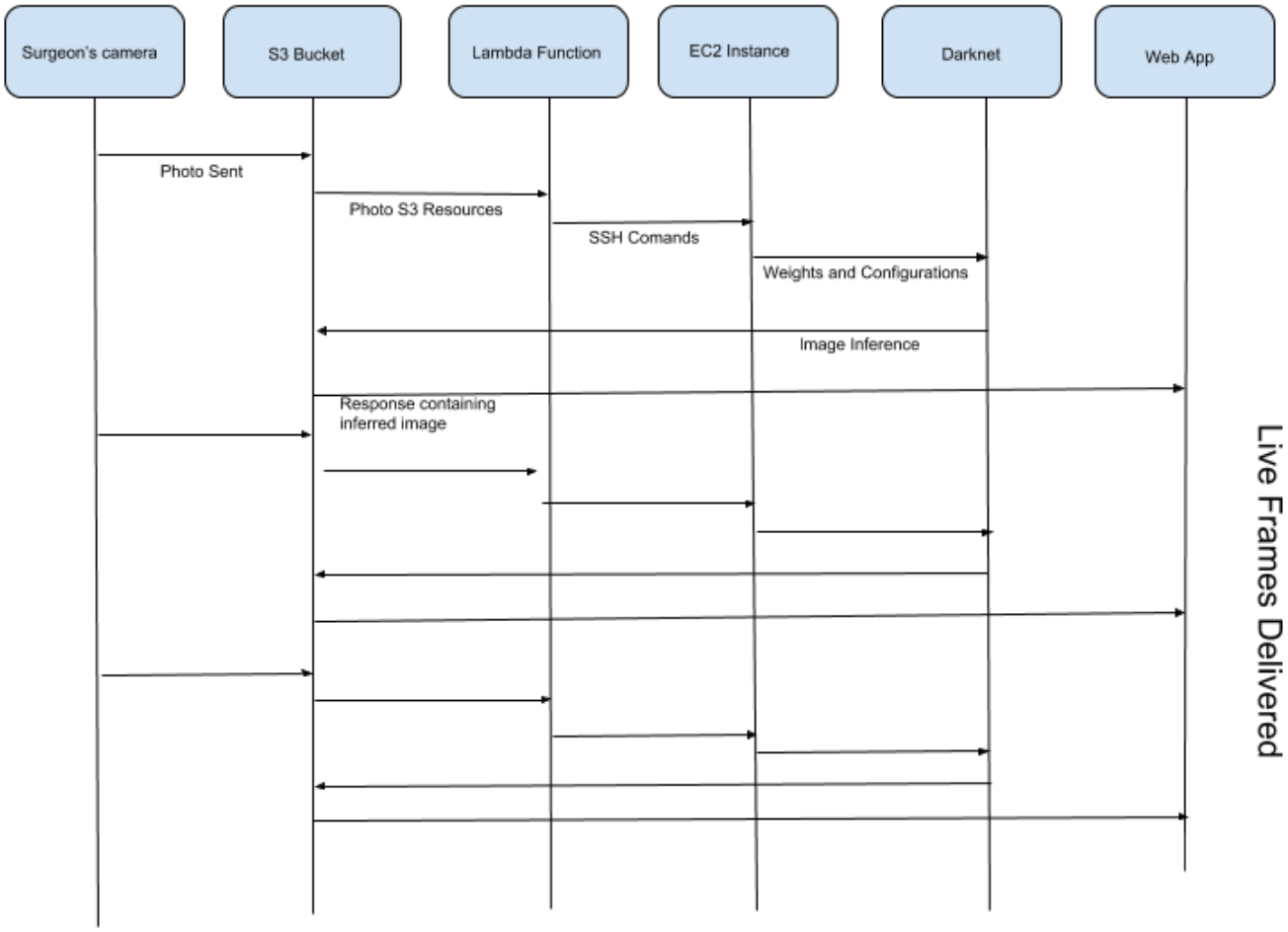
Image Inference Sequencing

The following time series sequencing chart depicts the flow of information from front end to back end components when an image is sent to the server to be requested. The image will start in the iOS hardware and end up on a web app hosted on S3.



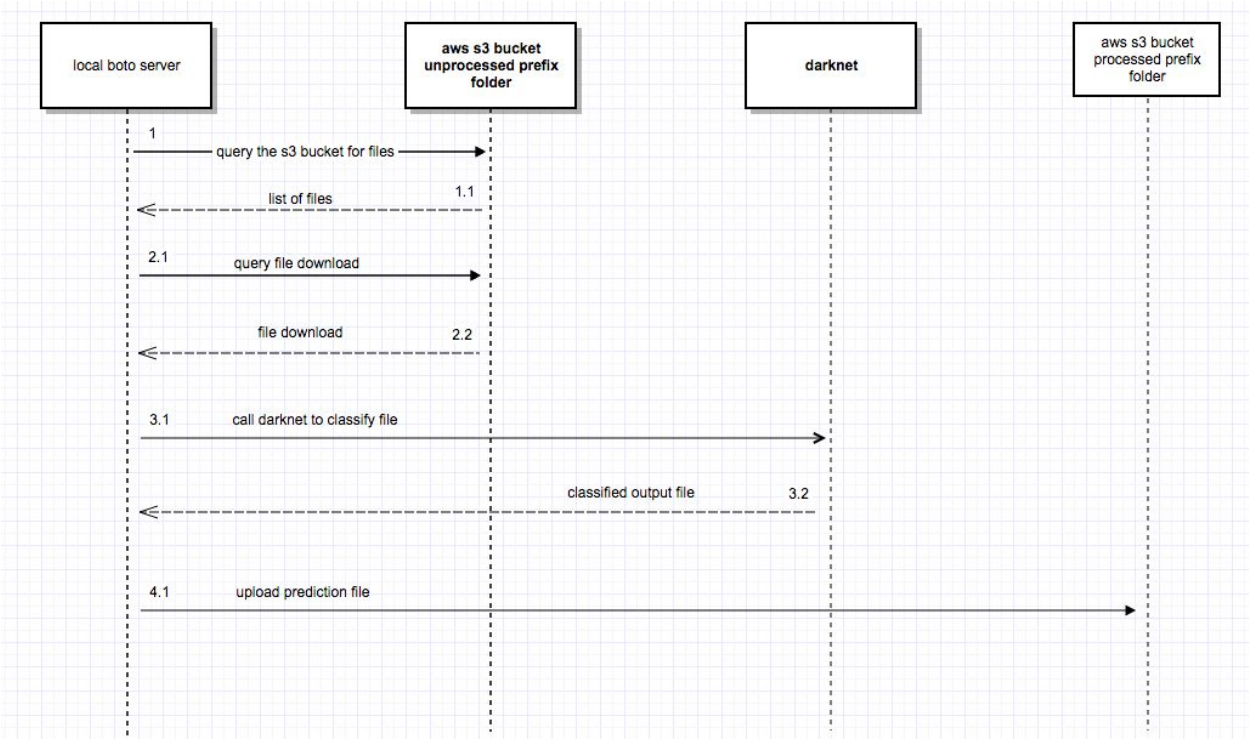
Live Video Inference Sequencing

The following sequencing diagram is very similar to image inferences with the exception that the front end web app receives video inferences frame by frame instead of just one frame.



Boto Server Sequencing

The following sequencing chart shows how the stand-in server interacts with aws s3 buckets and darknet. The server queries the unprocessed s3 bucket folder for files. The file is downloaded to the server, then the server calls darknet on the file. The output prediction file from darknet is then uploaded to the processed prefix folder.



Assist-MD Technical Implementation - Prototyping

Github:

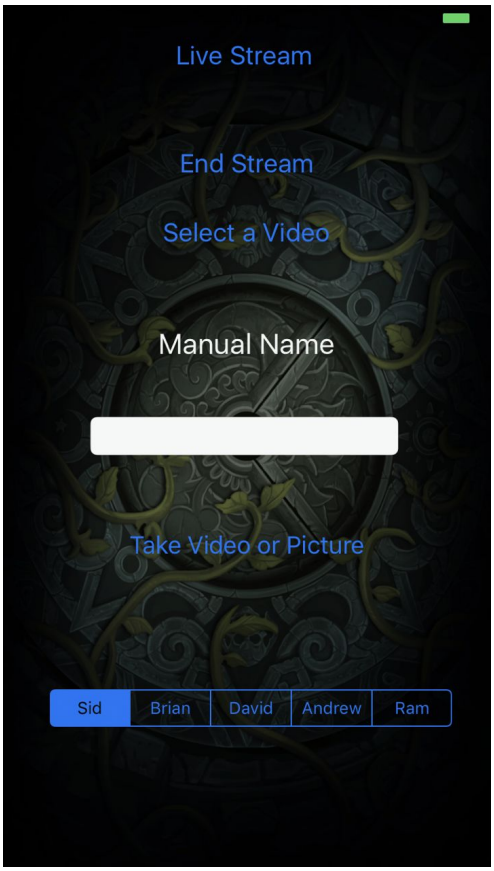
Our Github currently features the prototype of the Assist-MD Web App and the iOS App that records and streams video data to an AWS S3 bucket. It also features a deployable lambda function that incorporates Serverless Framework to update AWS services. As a backup, incase AWS falters in some way, we have built a stand in server that performs the basic functionality that the EC2 instance performs while listening to the S3 server and wathing for new objects to download and infer. Testing is also included.

https://github.com/brianhumphreys/Assist-MD

iOS interfacing with HVStream:

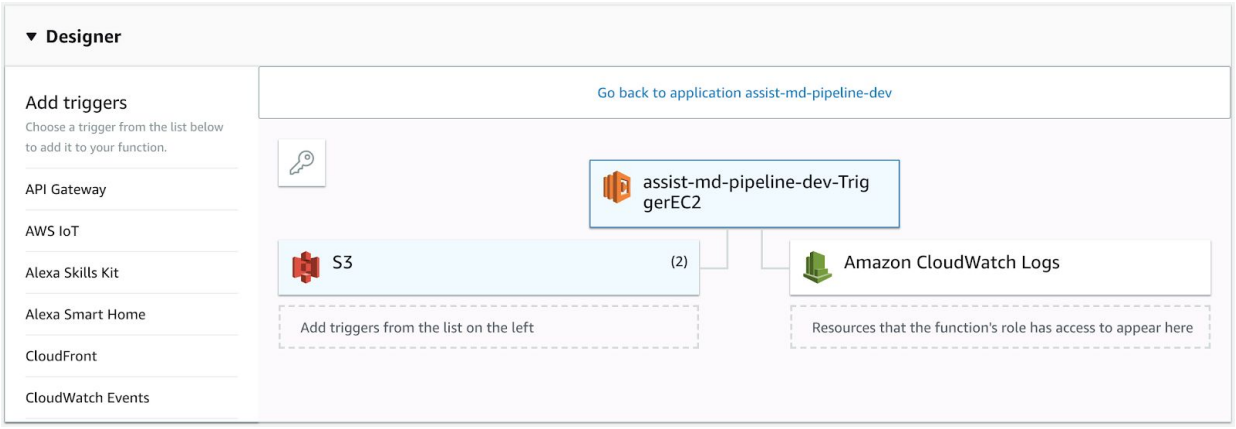
Direct and quick video upload system to aws for use with iphone. Three different cameras can be at play at a time on the same AWS account for surgical scene inferences. The data captured by the iPhone cameras will be streamed into an S3 bucket, fed into our YOLOv2 network to be processed and inferred upon, sent back to S3 and then streamed from Assist-MD Web App.

We utilize swift and integrate the AWS-SDK for the iOS to connect directly with our resources on the AWS website, specifically our S3 resources. The app can name and upload both video and photos to any users specific S3 bucket, as long as we know their access and security keys, along with their bucket name and region. These can all be changed instantly by the click of a button as shown in the image to the right at the bottom of the screen where options are displayed for AWS accounts available to stream to. We are then ready to begin processing the data with our YOLOv2 network.



Amazon Web Services (AWS) Backend:

The entry point of the AWS backend is the streaming of data from the iOS app to an S3 Bucket. The Lambda function shown below is triggered by the creation a new object (i.e. whenever a file is uploaded to it) in the S3 bucket specified in the Function Tree displayed in the left child of the function



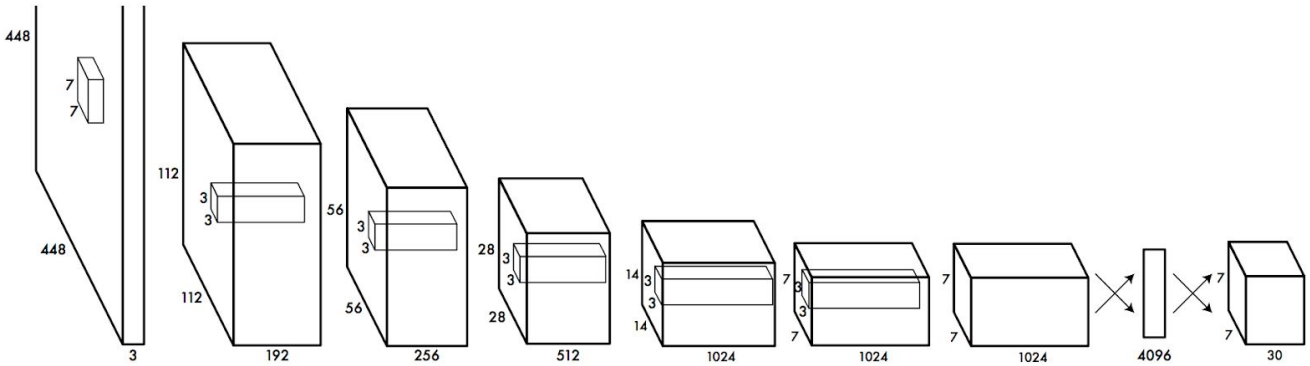
When the function is triggered, it receives data resources that describe in detail the object that was created. This data is used to build a URL that can access the public file on S3. When executed, the code below creates an SSH object and uses it to securely connect to a specified host (not shown for security reason). In our case, an EC2 instance acts as a host and is used as a vessel to execute image inferences, which the lambda function facilitates when the SSH object calls its exec() method. The file is downloaded to the server, inferred upon by the darknet command specified in the second .exec() method and then re-uploaded to the S3 bucket.

runcommand.js

```
var SSH = require('simple-ssh');
var fs = require('fs');

exports.handler = (event, context, callback) => {

    const bucket = event.Records[0].s3.bucket.name;
    const key = decodeURIComponent(event.Records[0].s3.object.key.replace(/\+/g, ' '));
    // const region = event.Records[0].awsRegion;
    console.log("bucket: ", bucket);
    console.log("key: ", key);

    /* -- create SSH object wit the credentials that you need to connect to your EC2 instance -- */
    var ssh = new SSH({
        host: 'NICE TRY SUCKERS',
        user: 'ec2-user',
        key: fs.readFileSync("ec2spin.pem")
    });
    ssh.exec('aws s3 cp s3://' + bucket + '/' + key + ' ~/assist-md/' + key, {
        out: function(stdout) {

            console.log(stdout);
            console.log('now launching command');

        }
    }).exec('cd darknet && ./darknet detector test obj.data Assist/yolo-obj.cfg Assist/yolo-obj_1400.weights ../assist-md/' + key, {
        out: function(stdout) {

            console.log(stdout);
            console.log('now launching command');

        }
    }).exec('aws s3 cp ~/darknet/predictions.png s3://yolo-images-random/processed/annotated.jpg', {
        out: function(stdout) {

            console.log(stdout);
            console.log('now launching command');

        }
    }).start();


}
```

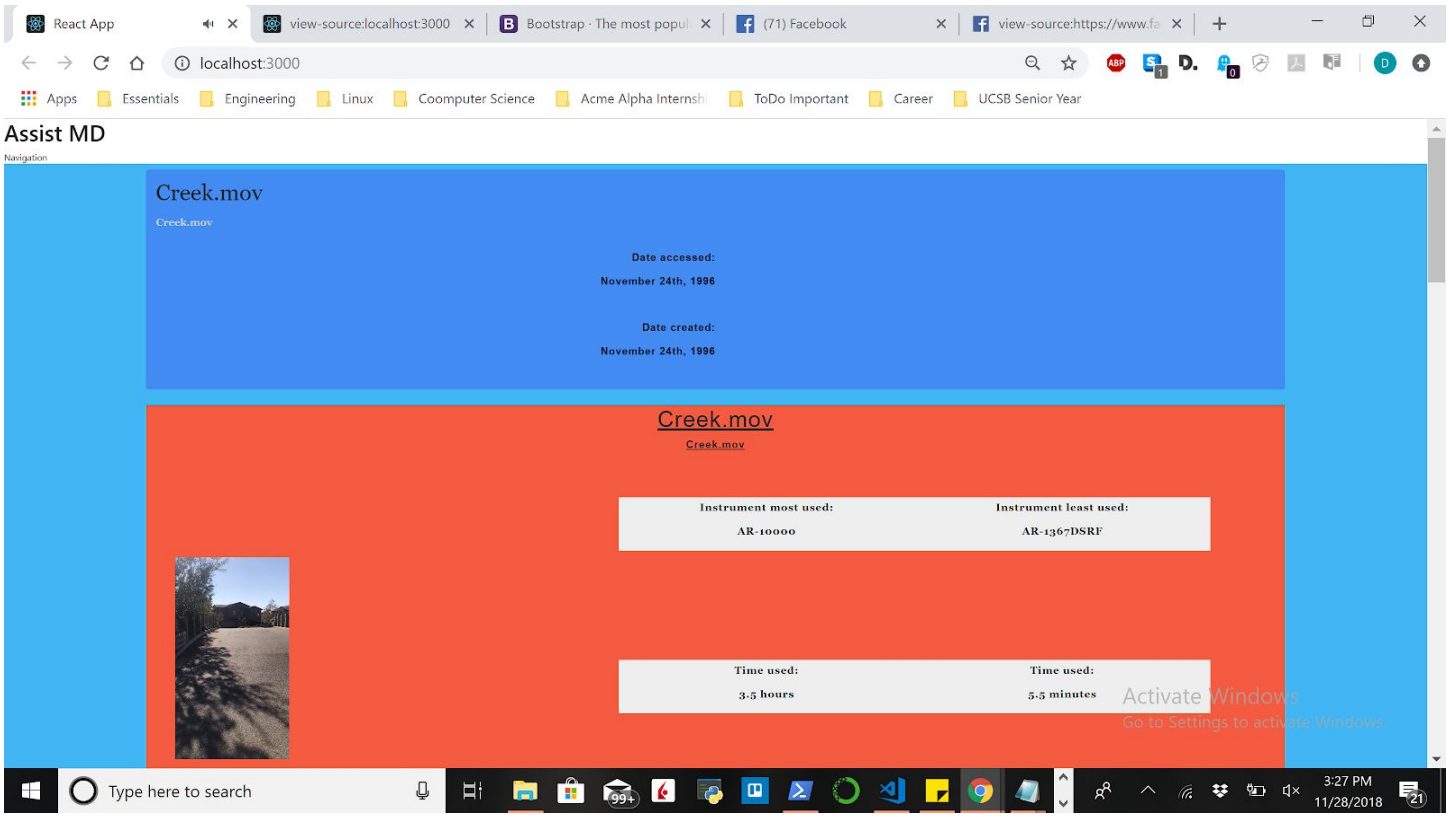
Image Recognition Component - YOLOv2 Network Structure:

For the image recognition portion of Assist-MD that will make inferences of the medical equipment and personnel, the High Voltage Society has decided to implement an instance of Joseph Redmon's YOLOv2 network to make multiple object inferences in a single picture. The below figure describes the structure of the YOLO network which implements some layers of convolution.



Front End Assist-MD Web App - ReactJS:

React-JS Web Interface - The app prototype has been built using standard ReactJS formatting with component and container folder structure. There are three individual page components used in the rendering of the page shown below. The blue rectangle is an object called a ‘Surgery Card’ and when clicked, the red box know as the ‘Full Card’ is shown beneath it. Inversely, the Full Card will disappear when clicked again. The video is displayed in the Full Card.



Appendices

What Assist-MD Does Not Do

Technologically:

Assist-MD (AMD) has many implications attached to it upon reading the Requirements sheet. One might think upon learning that AMD tracks medical personnel, that we will be building profiles of individuals. All data received about a medical personnels facial structure, eye color, build, etc. will be discarded. In a sense, every surgery will build its own environment of personnel characteristics and instrument usage.

The algorithm will not be able to directly make instrument inferences of instruments while they are in use. It is assumed that that the camera making instrument inferences will not be oriented in the surgeons perspective but at the instruments not in use. From this data, process of elimination will be used to figure out the instrument in use.

Surgically:

Although AMD will provide metrics of each surgery, no high-level analysis will be made on the surgery. As an example, AMD will be able to figure out the instruments that were used the most and the least and will also be able to know how many people were in the room at any given time, but it will not be able to give the surgeon advice on what instruments to use in the future or how to improve his performance. These high-level analyses will be left up to human intelligence for now.

Technologies Employed

AWS Lambda - A snippet of code that is spun up through AWS by an event trigger. The code takes event data as input and returns the desired output before the function instance is deleted from the servers and then saved as an image.

AWS S3 - Amazon Simple Storage Service is storage for the Internet. It is designed to make web-scale computing easier for developers. Amazon S3 has a simple web services interface that you can use to store and retrieve any amount of data, at any time, from anywhere on the web.

AWS Cloudfront - Amazon CloudFront is a web service that speeds up distribution of your static and dynamic web content, such as .html, .css, .js, and image files, to your users. CloudFront delivers your content through a worldwide network of data centers called edge locations. When a user requests content that you're serving with CloudFront, the user is routed to the edge location that provides the lowest latency (time delay), so that content is delivered with the best possible performance.

AWS MediaConvert - AWS Elemental MediaConvert is a file-based video transcoding service with broadcast-grade features. It allows you to easily create video-on-demand (VOD) content for broadcast and multiscreen delivery at scale. The service combines advanced video and audio capabilities with a simple web services interface and pay-as-you-go pricing. With AWS Elemental MediaConvert, you can focus on delivering compelling media experiences without having to worry about the complexity of building and operating your own video processing infrastructure.

AWS MediaLive - AWS Elemental MediaLive is a broadcast-grade live video processing service. It lets you create high-quality video streams for delivery to broadcast televisions and internet-connected multiscreen devices, like connected TVs, tablets, smart phones, and set-top boxes. The service works by encoding your live video streams in real-time, taking a larger-sized live video source and compressing it into smaller versions for distribution to your viewers. With AWS Elemental MediaLive, you can easily set up streams for both live events and 24x7 channels with advanced broadcasting features, high availability, and pay-as-you-go pricing. AWS Elemental MediaLive lets you focus on creating compelling live video experiences for your viewers without the complexity of building and operating broadcast-grade video processing infrastructure.

AWS EC2 - Amazon Elastic Compute Cloud (Amazon EC2) provides scalable computing capacity in the Amazon Web Services (AWS) cloud. Using Amazon EC2 eliminates your need to invest in hardware up front, so you can develop and deploy applications faster. You can use Amazon EC2 to launch as many or as few virtual servers as you need, configure security and networking, and manage storage. Amazon EC2 enables you to scale up or down to handle changes in requirements or spikes in popularity, reducing your need to forecast traffic.

CUDA - is a parallel computing platform and programming model that makes using a GPU for general purpose computing simple and elegant. The developer still programs in

the familiar C, C++, Fortran, or an ever expanding list of supported languages, and incorporates extensions of these languages in the form of a few basic keywords.

Docker - Docker is a tool designed to make it easier to create, deploy, and run applications by using containers. Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and ship it all out as one package. By doing so, thanks to the container, the developer can rest assured that the application will run on any other Linux machine regardless of any customized settings that machine might have that could differ from the machine used for writing and testing the code.

OpenCV - OpenCV (Open Source Computer Vision Library) is a powerful open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products.

ReactJS - In computing, React (also known as React.js or ReactJS) is a JavaScript library[2] for building user interfaces. It is maintained by Facebook and a community of individual developers and companies. React can be used as a base in the development of single-page or mobile applications. Complex React applications usually require the use of additional libraries for state management, routing, and interaction with an API.

SSH - also known as Secure Shell or Secure Socket Shell, is a network protocol that gives users, particularly system administrators, a secure way to access a computer over an unsecured network. SSH also refers to the suite of utilities that implement the SSH protocol. Secure Shell provides strong authentication and encrypted data communications between two computers connecting over an open network such as the internet. SSH is widely used by network administrators for managing systems and applications remotely, allowing them to log into another computer over a network, execute commands and move files from one computer to another.

Swift - Swift is a programming language that was created to be extremely safe and fast. It is utilized in various mobile and cloud platforms, including all of apple's products and software, which is where we will be utilizing it.

Tensorflow - TensorFlow™ is an open source software library for high-performance numerical computation. Its flexible architecture allows easy deployment of computation across a variety of platforms (CPUs, GPUs, TPUs), and from desktops to clusters of servers to mobile and edge devices.

YOLOv2 Algorithm - Most other computer vision algorithms repurpose classifiers or localizers to perform detection. They apply the model to an image at multiple locations and scales. High scoring regions of the image are considered detections. YOLO use a totally different approach. It applies a single neural network to the full image. This

network divides the image into regions and predicts bounding boxes and probabilities for each region. These bounding boxes are weighted by the predicted probabilities.

Simple-SSH - A wrapper for the ssh2 client module by Brian White which makes it easier to run a sequence of commands over SSH.

Python-pptx - A typical use would be generating a customized PowerPoint presentation from database content, downloadable by clicking a link in a web application. Several developers have used it to automate production of presentation-ready engineering status reports based on information held in their work management system. It could also be used for making bulk updates to a library of presentations or simply to automate the production of a slide or two that would be tedious to get right by hand.

Pandas - In computer programming, pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series. It is free software released under the three-clause BSD license. The name is derived from the term "panel data", an econometrics term for data sets that include observations over multiple time periods for the same individuals.