CSCE 606 - Software Engineering
Texas Fighting Aggies Platform
Project Report

Members
Pedro Henrique Villar de Figueiredo, Jesse Phipps, Ryan Kafka, Xintong Wu, Siddhant Thakur

Presentation & Demo Video
https://youtu.be/Z7m5cVmvchs

Pivotal Tracker
https://www.pivotaltracker.com/n/projects/2598148

GitHub
https://github.com/jessefphipps/fighting-aggies-platform

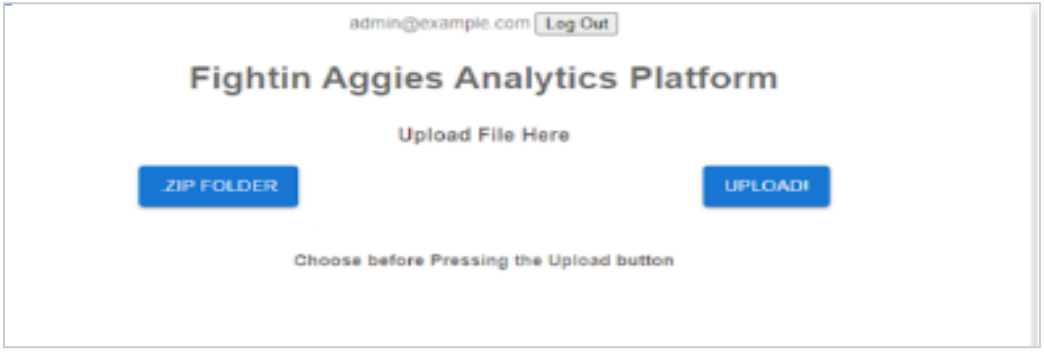
Heroku
https://fightin-aggies.herokuapp.com/

Summary
The Texas A&M Football program has a wealth of data associated with their players and performance. This data ranges from wearable monitors tracking their health and activities to video footage of practices and games. The program needs to streamline this pipeline of processing data, generating, and reporting the results of the models developed from the plethora of data to create actionable insights for their coaching staff and players.

The Texas Fighting Aggies platform focuses on using a computer vision model to analyze game and practice plays to identify players who tend to make mistakes while running their routes in a given play. The Vision model then sends the raw data to be analyzed by the backend and generates insights that can be further used by the coaches and the players to improve the team's performance. Our system also allows for the coaches to get in touch with the raw data to build their own statistical measures and graphs based on it.

User Stories
Iteration 1
- Feature [1pt] - User can (must) log in to their own personal account
    - As an A&M Athletics Member
    - I want to setup a login page
    - So that I can maintain the privacy of the data uploaded

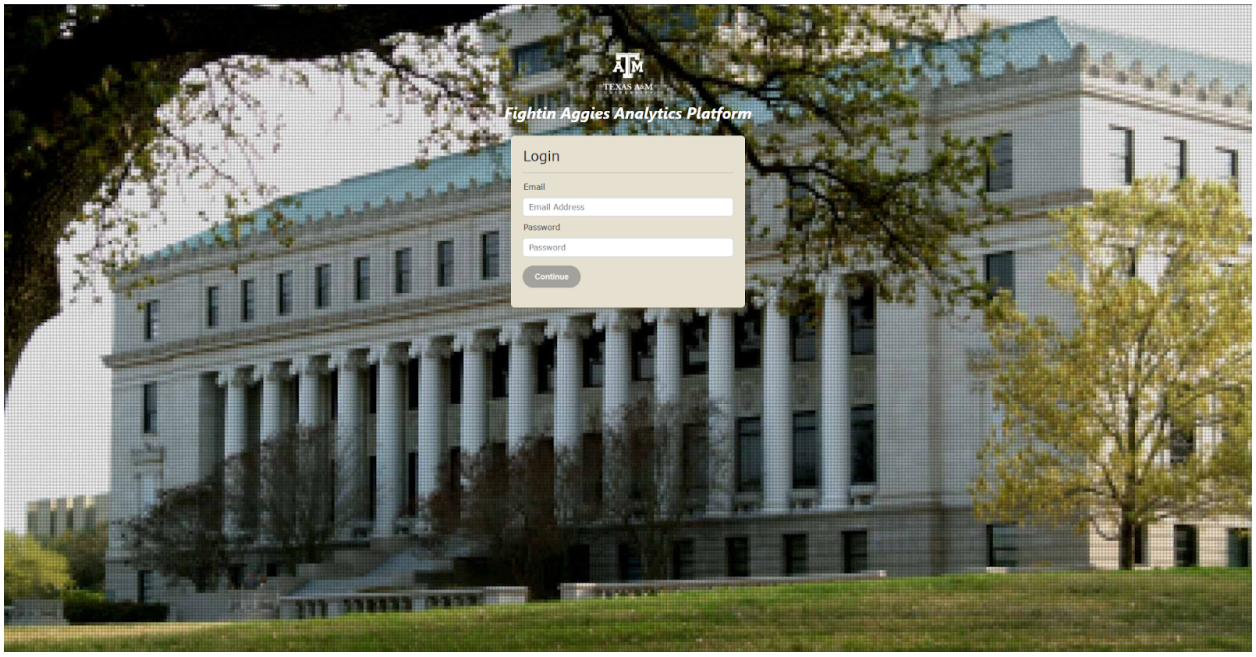
- Feature [3pt] - User can upload video files
    - As an A&M Sports Analyst
    - I want to upload video files of football practices / games
    - So that I can get a performance report of my players

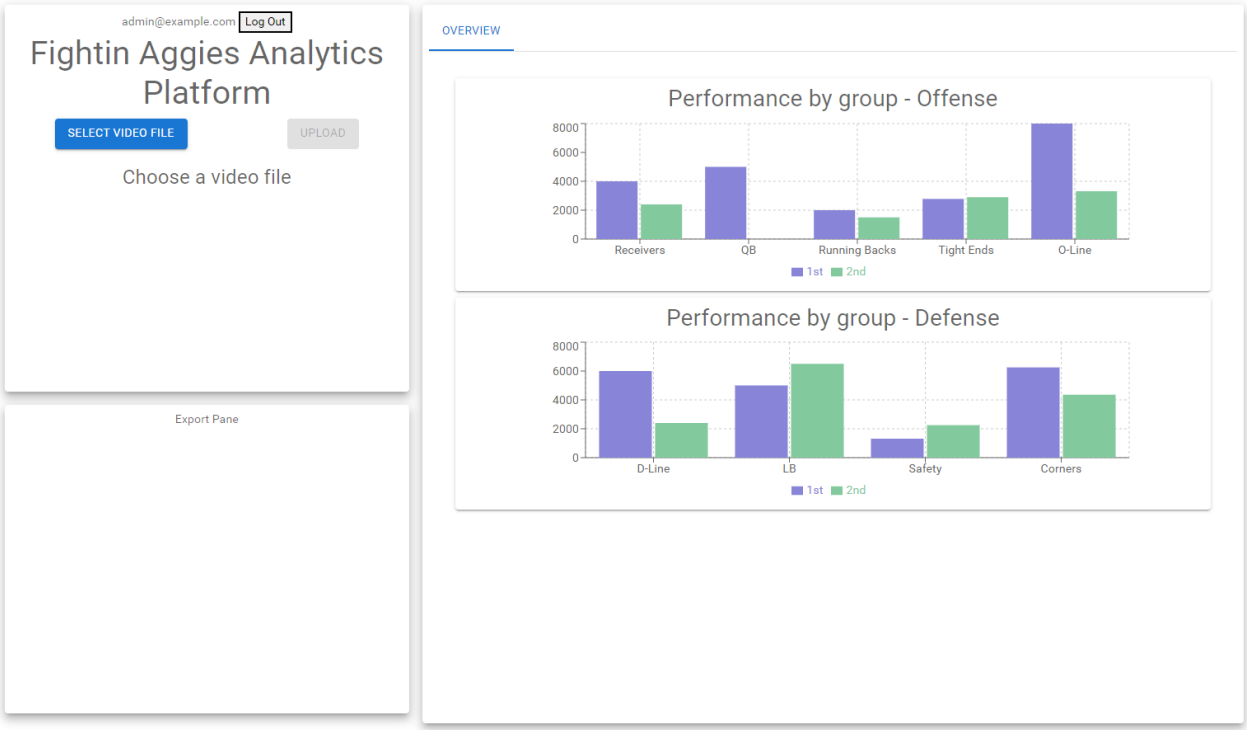


Implementation - We successfully implemented the log in and uploading video functionalities in our local machines, but we were facing problems while deploying these changes to Heroku.

Iteration 2

- Feature [1pt] - User can log in and out of the application
    - As an A&M Athletics Member
    - I want to setup a login session manager
    - So that I can maintain the privacy of the data uploaded





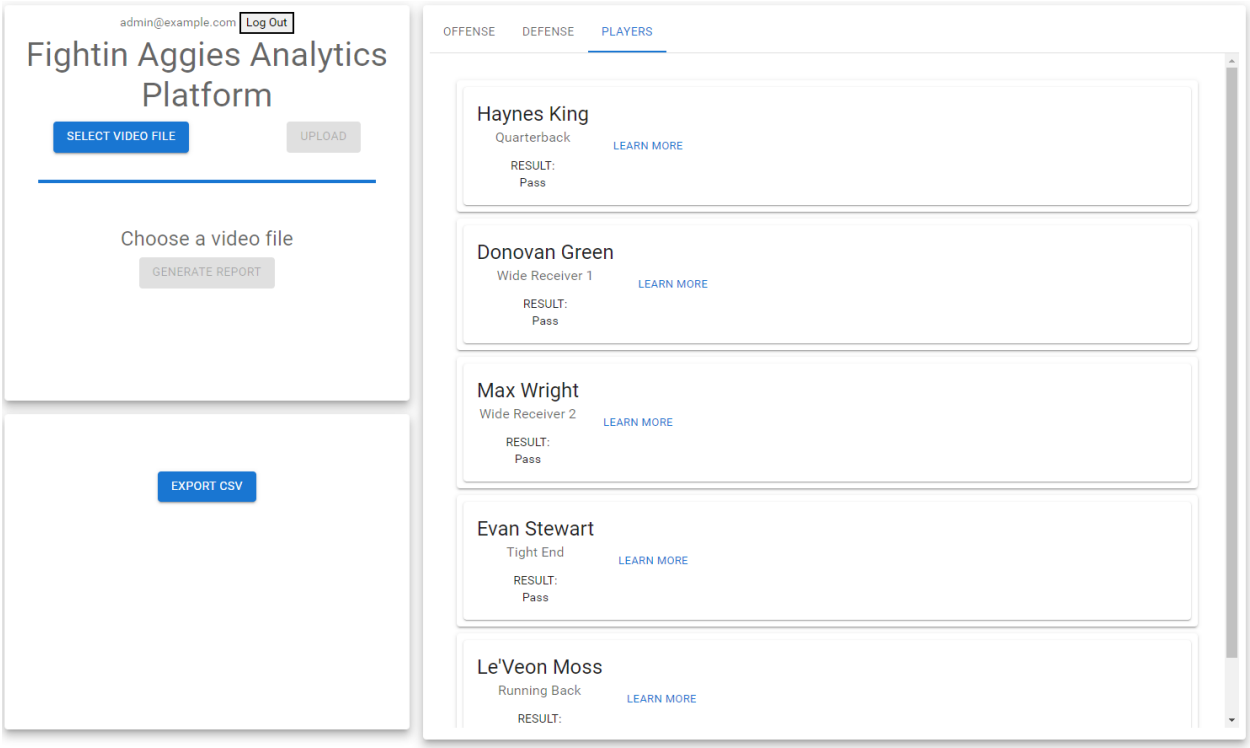
Implementation - Building on our previous iteration, we focused on setting up a collaborative environment on AWS using Cloud9. This helped us set up a common package, and we didn't face any issue regarding installing and working with the dependencies. We implemented the iteration 1 user stories in this environment as well as added on to it by implementing a Session Management approach to log in users on this platform. We successfully deployed and demonstrated the platform on Heroku.

Iteration 3

- Feature [3pts] - User can see the analysis done by the model on the dashboard
    - As a Texas A&M analytics staff member/coach
    - I want to see the insights generated by the video data
    - So that I can help the players understand how they are performing

Implementation - After setting up a working environment and deployment, we focused on generating analysis over the data we receive from the computer vision model. We focused on charts and key metrics that can be useful for the coaches and players to improve the team's performance.

Iteration 4



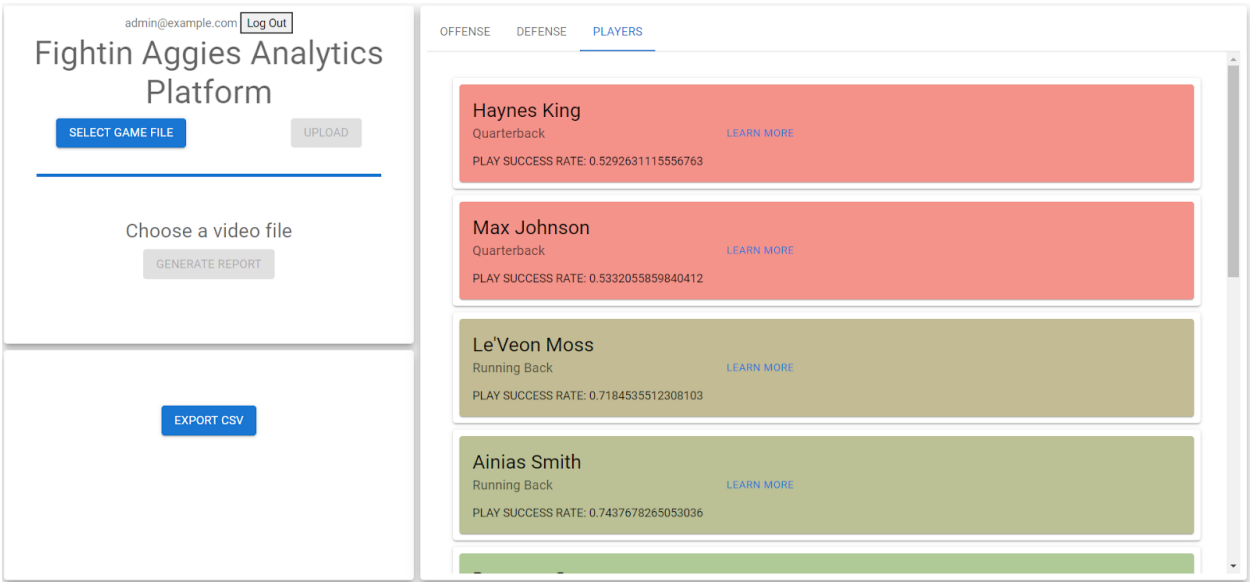
- Feature [1pt] - User can export raw data report from video as a CSV file
    - As a Texas A&M Data Analyst
    - I want to have easy access to the performance report data in an interactable format ('.CSV')
    - So that I can easily view, manipulate, and perform my own analyses of the data in a user-friendly way

- Feature [3pts] - User can view per-play data containing information of multiple players given by API
    - As a Texas A&M Data Analyst
    - I want to gain access to per-play report information of my players

- So that I may attribute a performance rating and give a report card to my players based on the data provided

Implementation - Generating the system's graphs and charts, we realized the need of implementing a functionality which gives the coaches and the staffs access to the raw data coming from the Vision model. Therefore, in this iteration, we implemented an Export button allowing the user to export the raw data in csv format. This also allowed the user to view player information for a single play.

Iteration 5



- Feature [1pt] - User can see summarized findings compiled from hundreds of plays
  - As a Texas A&M Athletics Data Analyst
  - I want to compile performance results across many plays
  - So that I can view summarized data describing the effectiveness of plays and players

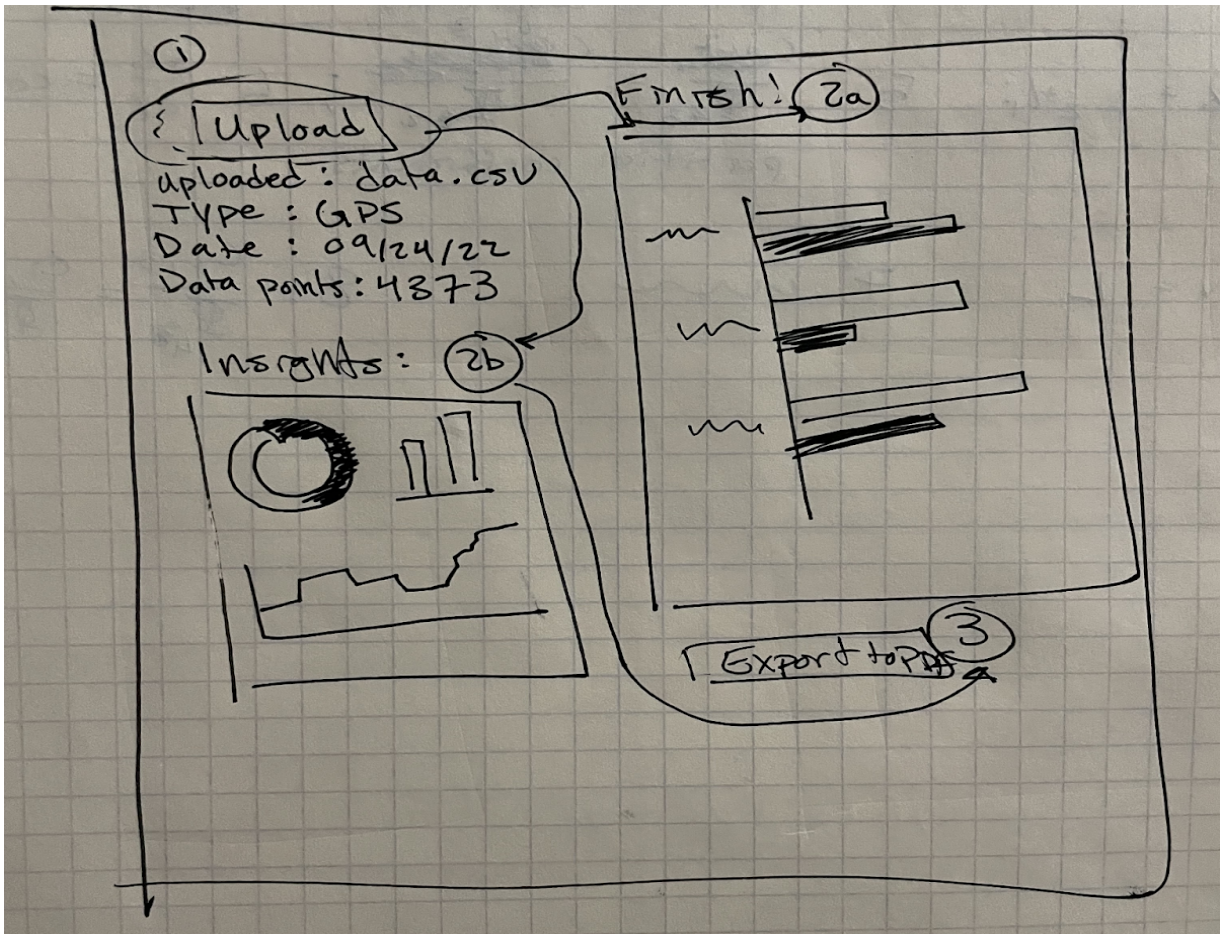
- Feature [1pt] - User can sort relevant player data
  - As a Texas A&M Athletics Data Analyst
  - I want to be able to sort the data I am viewing
  - So that I can easily view the performance information that is the most informative

- Feature [1pt] - User can view color coded player data
  - As a Texas A&M Data Analyst
  - I want my data to be color coded to indicate when there is a particularly low performance rating
  - So that I can quickly detect weaknesses in the respective category

- Feature [1pt] - User can export results as an excel file containing the summarized data displayed on the dashboard
    - As a Texas A&M Athletics Data Analyst
    - I want to be able to export the performance information into a formatted excel file
    - So that I can easily view, edit, and share the data outside of the web application

Implementation - Allowing the user to create their own graphs is certainly a great feature, however, there may be a case where the user might want to interact with the graphs shown in the platform's dashboard. To take care of this, we implemented a color coding scheme based on the raw data as well as sorting the data to the user's needs. Since the raw data being generated was only for a single play in the last iteration, we focused on devising the backend to handle multiple plays and export it in a csv format ready to be used as a dataframe.

Lo-Fi UI Mockup



Roles for project

Iteration Number	0	1	2	3	4	5
Date	09/23	10/08	10/21	11/04	11/18	12/02

Scrum Master	Jesse Phipps	Ryan Kafka	Xintong Wu	Siddhant Thakur	Pedro Henrique Villar de Figueiredo	Jesse Phipps
Project Owner	Siddhant Thakur	Pedro Henrique Villar De Figueiredo	Jesse Phipps	Xintong Wu	Ryan Kafka	Siddhant Thakur
Others	Pedro Henrique Villar De Figueiredo	Jesse Phipps	Pedro Henrique Villar De Figueiredo	Jesse Phipps	Jesse Phipps	Ryan Kafka
	Ryan Kafka	Siddhant Thakur	Ryan Kafka	Pedro Henrique Villar De Figueiredo	Siddhant Thakur	Pedro Henrique Villar De Figueiredo
	Xintong Wu	Xintong Wu	Siddhant Thakur	Ryan Kafka	Xintong Wu	Xintong Wu

Customer Meeting Dates

Iteration 1 - Friday, October 14, 2022 at 3pm

Demo - Log-in and Upload functionalities demonstrated (Local Machine)

Iteration 2 - Friday, October 28, 2022 at 1.45pm

Demo - Session Management functionalities demonstrated (Heroku), Setting up of Collaborative Work Environment

Iteration 3 - Wednesday, November 9, 2022 at 10am

Demo - Analytical Panel implemented with graphs (structuring)

Iteration 4 - Monday, November 28, 2022 at 3.45pm

Demo - Exporting data and Per-Play report functionalities available

Iteration 5 - Tuesday, December 6, 2022 at 2:20pm (informal meeting in class)

Demo - Summarizing multiple plays, sorting and color coding data functionalities demonstrated

BDD/TDD Process and Benefits

We followed a Test Driven Development process for most of our project cycle, since writing tests for a specific functionality allowed us to better understand what that functionality really does and how it can really be broken down by the user. We then implemented the given feature part by part satisfying every test and making sure it passed, so as to create a stable platform. This ensures that we are correctly accomplishing our tasks for the given functionality which leads to better software quality and design.

Configuration Management

For configuration management of our project, we used Git as version control. We realized the importance of branching off our work based on functionalities being added to the project, and we never faced any merge conflicts which we might have faced had we not used Git as the version control. We used multiple branches for different components of our project, and after finishing working on them, we merged and deleted those branches to maintain a clear and simple working directory. We just had to have spikes in the starting few iterations to get everything regarding the project clear, so that the implementation would be easier for us to do.

Dependencies are managed by the Gemfile and yarn.lock found in the root directory of the project. Yarn was the primary package management system, although some other installs outside of yarn will need to take place to get the project up and running as described in the "getting started" section.

Issues with deployment

In between our idea implementation and deployment, we faced a problem where we had dependency issues with the app working on different platforms. Even though our individual functionalities that we were supposed to implement were working, we were not able to deploy it on Heroku and it just created a mess. In the end, we had to preserve our functionalities, deploy a new skeleton of the application and create a collaborative environment which solved the major dependency issues. An important note to future teams is to first deploy your application, even though it might only contain a static page, and then start working on individual functionalities.

Issues with Cloud9 and Github

During our working phase, we did not face any issues with Github. However, we ran into out-of-memory issues on Cloud9, but we switched to a better version with more memory and the process was smooth after that.

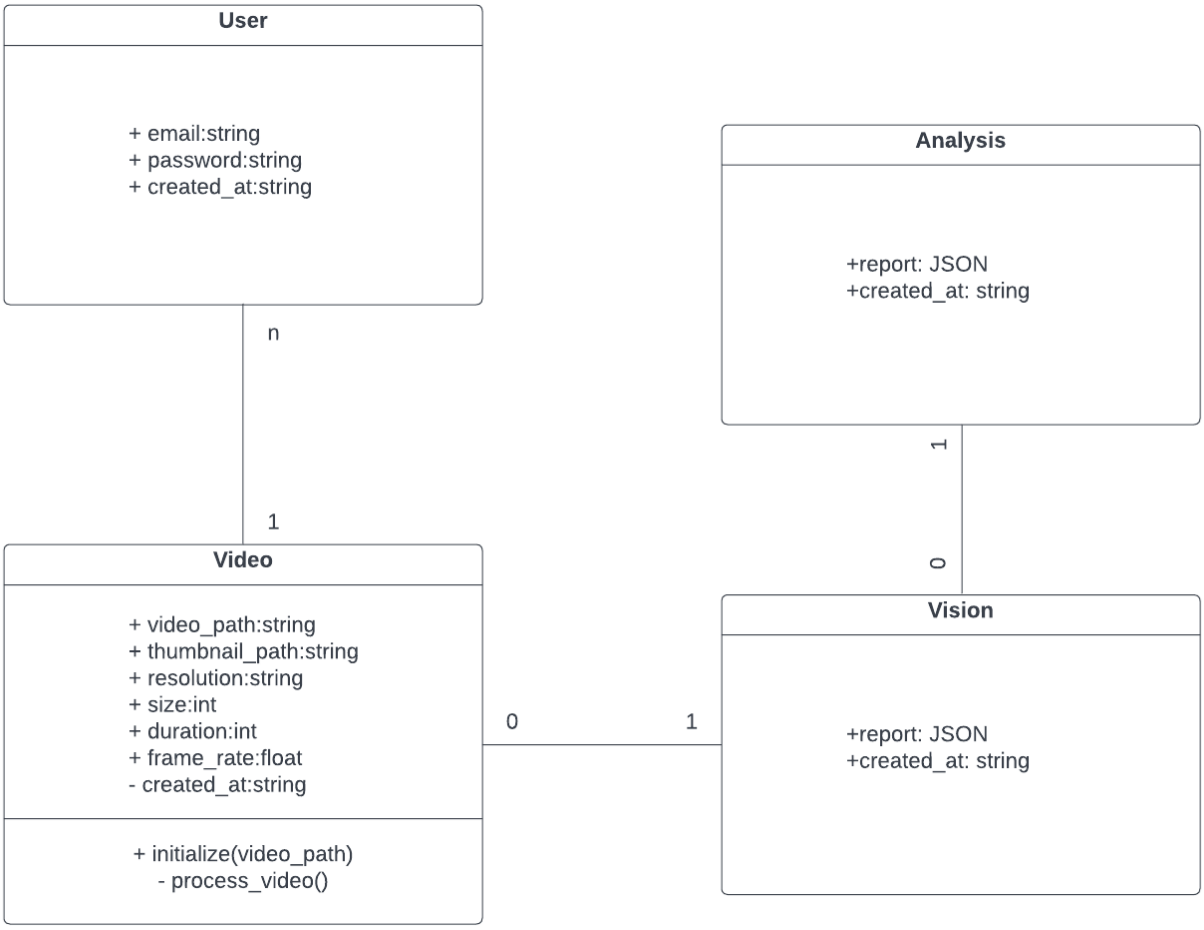
Solution Design

We incrementally improved our solution design through the five project iterations. Our proof-of-concept defines four main entities that are essential to our user stories: *User*, *Video*, *Vision*, and *Analysis*. We provide a short description of each entity followed by their relationship within our system illustrated by a class diagram.

User: Holds user information needed to enable the login/logout stories. In the future, the User entity may be used to allow personalized interfaces for different types of users such as players and the general public.
Video: Entity that stores video attributes. It includes metadata for each uploaded video that is displayed in the dashboard.

Vision: Stores raw data outputted by the Vision API. It is attributed to an existing Video in the system. It is also available in the dashboard via the “Export CSV” button.

Analysis: Contains front-end visualization descriptors for a video. It is a JSON-based format descriptor based on the raw data stored in the Vision entity. The front-end dashboard interprets the descriptor and displays the corresponding visualization in the dashboard.



Repository Contents

We divide the contents of the repository that holds our solution into 3 sections: frontend, backend, and tests. File structuring follows modern Ruby On Rails practices. The outline of each section can be found below.

Frontend

As previously mentioned, the frontend of this project was implemented with ReactJS. The main files coordinating frontend rendering are app/javascript/components/App.jsx, app/javascript/components/Login.jsx, and app/javascript/pages/home/home.jsx. These files can be described as follows:

App.jsx

- Holds the recoil root for state management. This is seen by the <RecoilRoot> tag.
- Manages routing between login page and the dashboard and vice versa. This is done via React Router.

Login.jsx

- This is the React component that renders the login page.
- It not only holds frontend elements including the text fields for username and password, but also the logic for our authentication client Firebase.
- The initial state for the page is stored on render in the INITIAL_STATE variable.

Home.jsx

- The React component that renders the dashboard.
- This functional component holds several aspects:
    - The logic for sending backend requests for file uploads and analysis generation.
    - The layout frame that holds the following components:
        - Logout button
        - File upload component
        - Export component
        - Results component

Each component that is contained within Home.jsx can be found in app/javascript/components, where each main component has its own folder. All components are rather straight forward and follow typical ReactJS structure except for the results components.

The logic behind results components rendering is as follows: (1) the generateReport function in home.jsx sends a backend request to generate a JSON formatted object that represents a single report. This JSON object is assigned to the recoil results atom, and the Results component is then re-rendered due to the state change. (2) The JSON object is sent to the Results component, where the component separates each block that represents a tab/figure list and then sends the figure blocks to the results_handler. (3) This handler parses the figure JSON block into one of the components found in app/javascript/components/Results/results_component.

Backend

We follow the MVC pattern for our solution. Given that the views are written in ReactJS (see frontend above), in this section we focus on the model and controller components that make up the backend. More specifically, our project implements both the model and controller for the *Video*, *Vision*, and *Analysis* entities.

Models

We enforce the relationships between the entities via Rails' Active Record *associations*. In our project, the Video entity is created when a user uploads a video. During its initialization, we use FFMPEG to retrieve its metadata (e.g. resolution, duration, etc.) and also to create a thumbnail of the video, stored in our server next to the uploaded video. Following our solution design, a Video entity can be associated with up to one Vision entity; which then can be associated with up to one Analysis entity. We added cascade delete rules to enforce the above composition scheme.

Since our proof-of-concept focuses on the web platform, we do not implement a working computer-vision-based system to evaluate football plays. Instead, we show our work through a dummy module called *Vision*. It takes an uploaded video ID as input, and randomly generates raw data that would otherwise be inferred from a computer-vision system. The Vision model stores such a report as a JSON attribute of the Vision entity in our database.

The *Analysis* model is initialized with a video ID, and produces front-end descriptors based on the video's raw computer-vision data (stored in a corresponding *Vision* object). If the raw vision object for the requested video ID is not present in the database, the initializer generates it before proceeding. The front-end descriptors are JSON-based, being interpreted by the front-end system as part of our application's dashboard.

Controllers

We implement the controllers following RESTful API best practices, including returning an appropriate request code depending on the operation or error. Even though we only use the CREATE endpoint for most of the controllers in our proof of concept, our backend implements all CRUD operations (including tests) for all our controllers.

We make use of three controllers, Videos, Visions and Analyses. The Video controller allows us to manage the CRUD operations over selecting and uploading video to the platform, while the Visions and Analyses controller allow us to manage the raw data and the analysis generated (respectively) from the uploaded video. We then use these controllers in our front-end to develop graphs and export raw data for the users. For technical reference for future teams, we have listed the endpoints of the application below, allowing them to further develop on this project.

URI Pattern	Request	Use

/	GET	Landing Login Page
/dashboard	GET	Analytical Panel Page
/api/v1/videos/index	GET	List of videos present in the database
/api/v1/videos/create	POST	Create a video instance in the database
/api/v1/videos/show	GET	Shows a video when ID is provided
/api/v1/videos/destroy	DELETE	Delete video identified by the URI
/api/v1/visions/index	GET	List of raw data from the Vision API in the database
/api/v1/visions/create	POST	Create an instance, in this case raw data, from the Vision API in the database
/api/v1/visions/show	GET	Shows the raw data from the Vision API when ID is provided
/api/v1/visions/destroy	DELETE	Delete raw data from the Vision API identified by the URI
/api/v1/analyses/index	GET	List of analysis generated for the graphs in the database
/api/v1/analyses/create	POST	Create an analysis in the database when the video is uploaded
/api/v1/analyses/show	GET	Shows an analysis from the video when the video ID is provided
/api/v1/analyses/destroy	DELETE	Delete the analysis generated of a video identified by the URI

Evaluation and Testing of Code

We have used Cucumber and Minitests to test our front-end and back-end development. Cucumber was used for UI tests on both login and dashboard pages. It was also used for end-to-end testing on the dashboard (front-end + back-end, to see how the user will interact with the system). Minitest was specifically used for the backend Video testing.

Tests for running the app on the browser (Environment testing)

- Environment should be driven by a Selenium webdriver and use Chrome.

Tests for UI Login: Cucumber

- **[bad path 1]** whether the user has provides invalid password → wrong password
- **[bad path 2]** whether the user provides invalid email address → user not found
- **[bad path 3]** whether the user provides a bad email format → improper email format
- **[correct path 1]** when the user clicks the logout button → logged out from session & redirected to login page
- **[correct path 2]** whether the user enters correct email and password combination → redirected to dashboard

Tests for End-to-End File Upload (UI + backend): Cucumber

- **[bad path 1]** when the user logs in to the dashboard → upload button is disabled by default
- **[bad path 2]** when the user uploads a file with incompatible file type → user gets negative feedback: incompatible upload type
- **[bad path 3]** when the user uploads a file which is corrupt → user gets negative feedback: incompatible upload file
- **[correct path 1]** when the user uploads file successfully → user gets positive feedback: file uploaded successfully

Tests for report generation (UI): Cucumber

- **[generate button disabled 1]** when the user first accesses the dashboard, the report generation button should be disabled
- **[generate button disabled 2]** when the user has selected a file but not successfully completed an upload, the button should be disabled
- **[success]** when the user successfully uploads a file and all other tests pass, the report is generated and the results pane is populated

Tests for Video Model: Minitests

- **[bad path 1]** when there's no video uploaded → Uploaded file is not a video
- **[bad path 2]** when the format for the video is incorrect → Uploaded file is not a video
- **[bad path 3]** when the video file path is not present → Video file does not exist
- **[bad path 4]** when the video file is corrupted→ Cannot read the video file
- **[correct path 1]** when the video file path is correct for existing and valid sample video
    - **[bad path 1]** when the file does not exist → Video file does not exist
    - **[bad path 2]** when the recorded video path does not match the uploaded video path → Recorded video path is not the same as in db
    - **[bad path 3]** when the thumbnail of the recorded video does not match the thumbnail of the uploaded video → Recorded thumbnail path is not the same as in db

        - **[bad path 4]** when there is no thumbnail generated of the previous upload → Generated thumbnail file does not exist
        - **[bad path 5]** where there is a resolution problem with the recorded video → Recorded resolution does not match sample video
        - **[bad path 6]** where there is a duration problem with the recorded video → Recorded duration does not match sample video
        - **[bad path 7]** where there is a frame problem with the recorded video → Recorded frame does not match sample video
        - **[bad path 8]** where there is a frame rate problem with the recorded video → Recorded frame rate does not match sample video

Tests for Video Controller: Minitests

- index
    - **[correct path 1]** controller is able to fetch index → 200, Success
    - **[bad path 1]** checks if it returns all videos present in DB → Videos coming from index request don't match db's videos
- show
    - **[bad path 1]** if no id present in get request → 400, No ID was provided
    - **[bad path 2]** if id is invalid (negative or not in DB) → 400, Video with requested ID not found in DB
    - **[bad path 3]** if id is valid, but it doesn't match the query → Requested video does not match with query ID
    - **[correct path 1]** if id is valid → 200, Success
- create
    - **[bad path 1]** uploading video with no data → 400, No video provided
    - **[bad path 2]** uploading video with incorrect file format → 400
    - **[bad path 3]** uploading video with corrupted data → 400
    - **[correct path 1]** uploading video with everything correct → 201, video json details
- destroy
    - **[bad path 1]** deleting video with no id → 400, No ID was provided
    - **[bad path 2]** deleting video with invalid id (negative or not in DB) → 400,Video with requested ID not found in DB
    - **[bad path 3]** deleting a video with valid id, but it doesn't match the query → Video was not deleted from DB
    - **[correct path 1]** deleting a video with valid id → 200, Success

Tests for Vision Controller: Minitests

- index
    - **[correct path 1]** controller is able to fetch index → 200, Success

        - **[bad path 1]** checks if it returns all raw vision data present in DB → Raw vision data coming from index request don't match db's raw vision data
- show
    - **[bad path 1]** if no id present in get request → 400, No ID was provided
    - **[bad path 2]** if id is invalid (negative or not in DB) → 400, Raw vision data with requested ID not found in DB
    - **[bad path 3]** if id is valid, but it doesn't match the query → Requested raw vision data does not match with query ID
    - **[correct path 1]** if id is valid → 200, Success
- create
    - **[bad path 1]** generating raw vision data with no data → 400, No raw data received
    - **[bad path 2]** generating raw vision data with video id not present in the database → 400
    - **[bad path 3]** generating two raw vision data for same video id → 400
    - **[correct path 1]** generating raw vision data with correct video id→ 201, raw vision data json details
- destroy
    - **[bad path 1]** deleting raw vision data with no id → 400, No ID was provided
    - **[bad path 2]** deleting raw vision data with invalid id (negative or not in DB) → 400, Raw Vision data with requested ID not found in DB
    - **[correct path 1]** deleting a video with valid id also destroys raw vision data→ 200, Success

Tests for Analysis Controller: Minitests

- index
    - **[correct path 1]** controller is able to fetch index → 200, Success
    - **[bad path 1]** checks if it returns all analyses present in DB → Analyses coming from index request don't match db's analyses
- show
    - **[bad path 1]** if no id present in get request → 400, No ID was provided
    - **[bad path 2]** if id is invalid (negative or not in DB) → 400, Analysis with requested ID not found in DB
    - **[bad path 3]** if id is valid, but it doesn't match the query → Requested analysis does not match with query ID
    - **[correct path 1]** if id is valid → 200, Success
- create
    - **[bad path 1]** generating analysis with no data → 400, No analysis done
    - **[bad path 2]** generating analysis with video id not present in the database → 400
    - **[bad path 3]** generating two analyses for same video id → 400

    - **[bad path 4]** generating analysis with correct video id but null raw vision data → 400
    - **[correct path 1]** generating analysis with correct video id and correct raw vision data→ 201, analysis json details
- destroy
    - **[bad path 1]** deleting analysis with no id → 400, No ID was provided
    - **[bad path 2]** deleting analysis with invalid id (negative or not in DB) → 400, Analysis with requested ID not found in DB
    - **[correct path 1]** deleting a video with valid id also destroys analysis → 200, Success