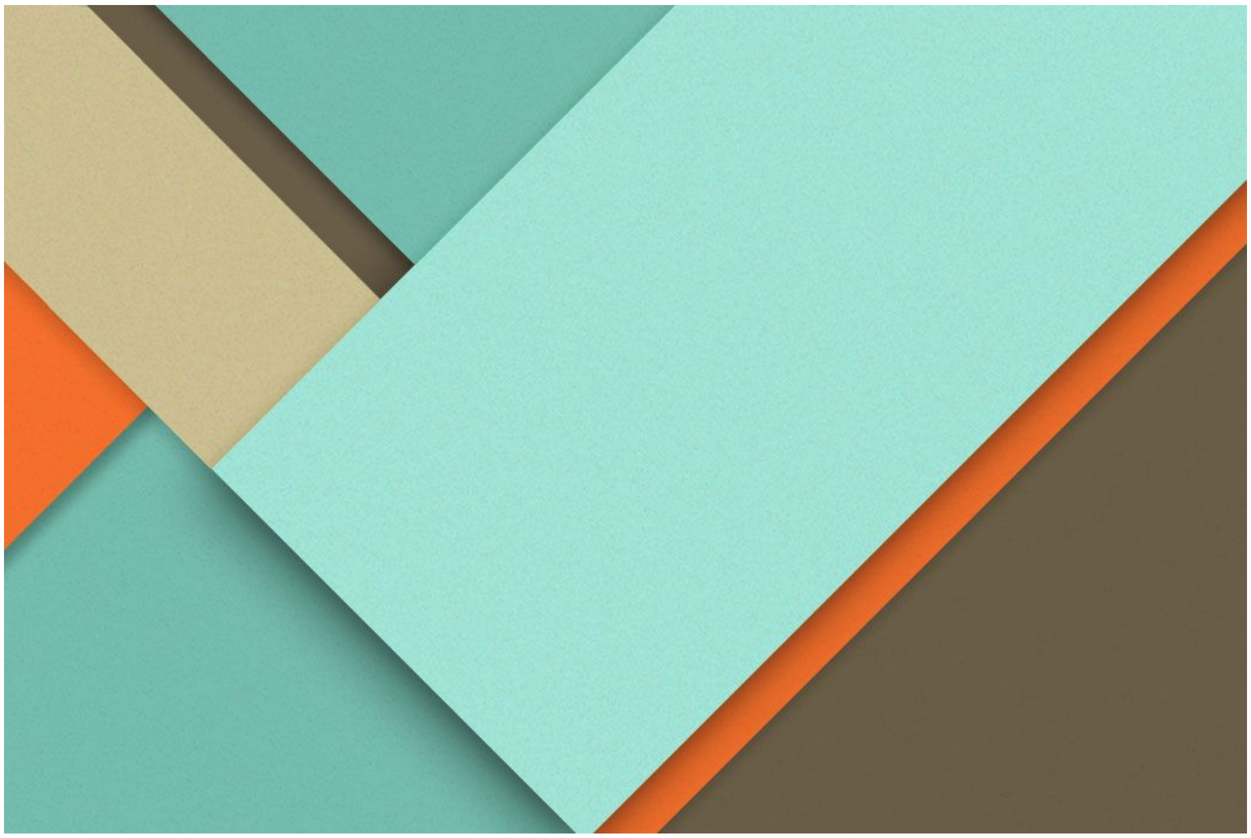


# AIRAT Final Report

12/10/2019

Team 8
-Ryan Cerveny
-Apurva Patel
-Jonathan Murphy
-Takao Shibamoto

## Project Overview

### Problem Statement

Requirement tracing is the process of creating logical links between individual requirements within a requirement hierarchy which consists of both “high level” and “low level” requirements. The creation of these links between high and low level requirements is an essential part of all projects carried out by Collins Aerospace. Due to the safety critical nature of systems developed by Collins Aerospace, accurate and robust requirement tracing is a necessity in order to ensure that all necessary features are recognized, that no unnecessary features are included, and that we can link the reasoning for including a component to a higher feature. Currently, requirement tracing at Collins is a manual process. This is problematic for their business because this process is very time consuming and leaves room for human error, which can not be tolerated in the very sensitive and safety critical systems that Collins Aerospace engineers and builds. The goal of our senior design project was to develop a tool to be used by engineers at Collins Aerospace that automates the requirement tracing process.

The goals of the project can be split into two main areas. One area of the project was the development of a web application that provides a user friendly interface for Collins engineers to start the requirement tracing process and view the current completion status or results of other requirement traces. The second area of the project was to research machine learning techniques, specifically forms of natural language processing, in order to accurately classify requirements as either “good links”, meaning that a high level requirement and a low level requirement are closely related and should be linked and “bad links”, meaning the two requirements are unrelated.

## Project Design

### Web App

#### Involved Softwares

- Ubuntu 18.04 EC2 on AWS
- Python 3.7
- Django 2.2
- Django Rest Framework
- Django-Filters
- ReactJs
- React Bootstrap
- Redux
- Sqlite
- Shell scripts
- Gitlab CI/CD

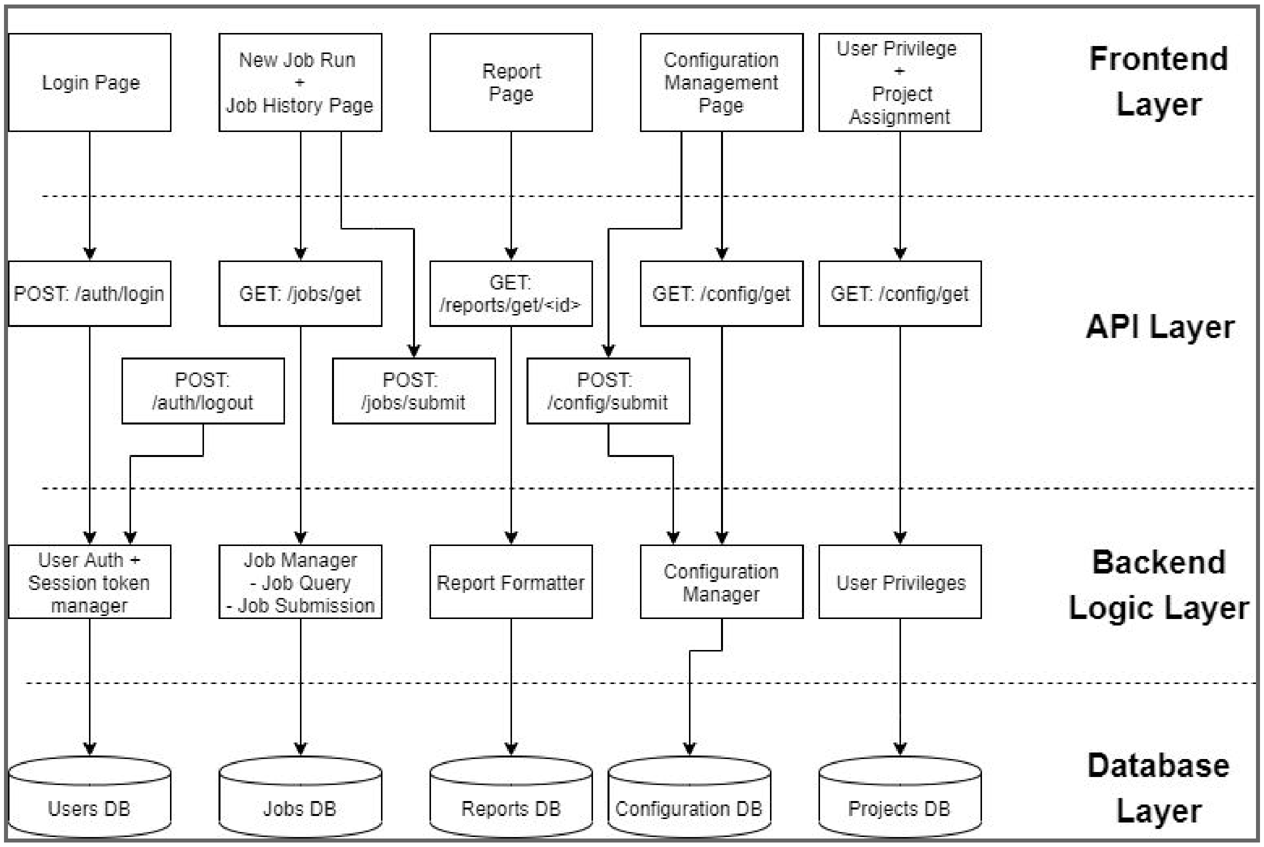
#### Database Design

The structure of data which will be given to us from Collins is best suited for a relational database. Below are the tables which will be discussed further in the implementation section of the report:

- Users
  - Stores information about each user
- Roles
  - Defines role codes for user table (user, admin, superuser)
- Project_permission
  - Links users to projects they have access to
- Project
  - Stores existing projects
- Job
  - Stores all created jobs
- Configuration
  - Stores configuration definitions
- Config_val
  - Stores key/val for configurations
- Report
  - Defines a report

- Requirement
  - Stores the requirements within a report
- Link_existing
  - Stores all the existing links between requirements
- Link_rec
  - Stores all the recommendations for a requirement links

## Block Diagram



### Authentication Trackway

The authentication trackway was designed to allow Collins to modify it to apply to the company security standards. The web app will check the username and password upon login against the database, then will proceed to use the username as a session ID which later could be enhanced to Collins' security approach, or discarded for another approach.

### Configuration Trackway

Configurations are sets of key-value pairs which may be linked to a project to modify the way Collins' analysis process behaves. Users may create/modify a configuration on the configuration page and send it to the backend to be stored in the database. Once stored, it may be seen on the run page when starting new jobs.

### Job Trackway

When a user runs an analysis for a project on the front end, the request will be sent through the job api on the back end. On this endpoint, the job will first be inserted into the jobs database with a status of “In Progress”. Then, a thread will be started on the backend. This thread is the extension point for Collins, which will be discussed later in the report.

### Report Trackway

Once an analysis has been run, the user may download an excel report via the reports page. On the reports page, the user can search for the desired report, and click on the download link. This will send a request to the back end which will pull the report from the database and format it into an excel document and return it to the user as a download.

### User Privilege Trackway

There are three different user types for this webapp: standard user, admin, and super user. Super users and admins have access to a permissions page which allows them to modify access levels of other users. Super users may add admins to projects, and admins may add standard users to projects.

## Integration Point

AIRAT was split into two components. The web app, and the analysis tool which was built by Collins. Once both are complete they need to be integrated together. To account for this, we created a threading process which allows collins to call their code from the run endpoint without bogging down the backend, and leaving it flexible for the clients needs. The inner workings of the integration point will be discussed in the implementation section of the report.

## Machine Learning Research

During the first semester of our project, we researched and experimented with a variety of different machine learning techniques including Word2Vec, Doc2Vec, GloVe, Topic Modeling, and Swarm Intelligence in search of a technique that could accurately classify requirements. These approaches proved to be unsuccessful either due to insufficient accuracy results (Word2Vec, Doc2Vec, and GloVe) or our contacts at Collins suggesting that we abandon the approach and search for other alternatives (Topic Modeling and Swarm Intelligence).

Over the summer between our first and second semesters of working on the project, Collins decided to put together a small engineering team internally in order to research other feasible approaches for solving the requirement classification problem. They came to the conclusion that the best course of action would to classify the requirements using two different models, an implementation of gradient boosted trees using XGBoost and an implementation of Long Short-Term Memory. A brief explanation of  both of these approaches is provided below. The results of each classifier would then be optimized using a technique known as Model Stacking, which involves developing an algorithm in order to combine the output results from multiple models in order to produce the optimal result. Though the majority of our team was focused on developing the web app portion of the project, our client suggested that we dedicate at least one member of the team to continue researching potential machine learning solutions based on the techniques mentioned above and share our findings. We researched different ways to implement classifiers using XGBoost and LSTM and reported to Collins when feasible alternatives for their current implementations were discovered. The approaches proposed by the Collins Aerospace team and the alternate approaches suggested by our team are detailed in the "Machine Learning Implementations" section.

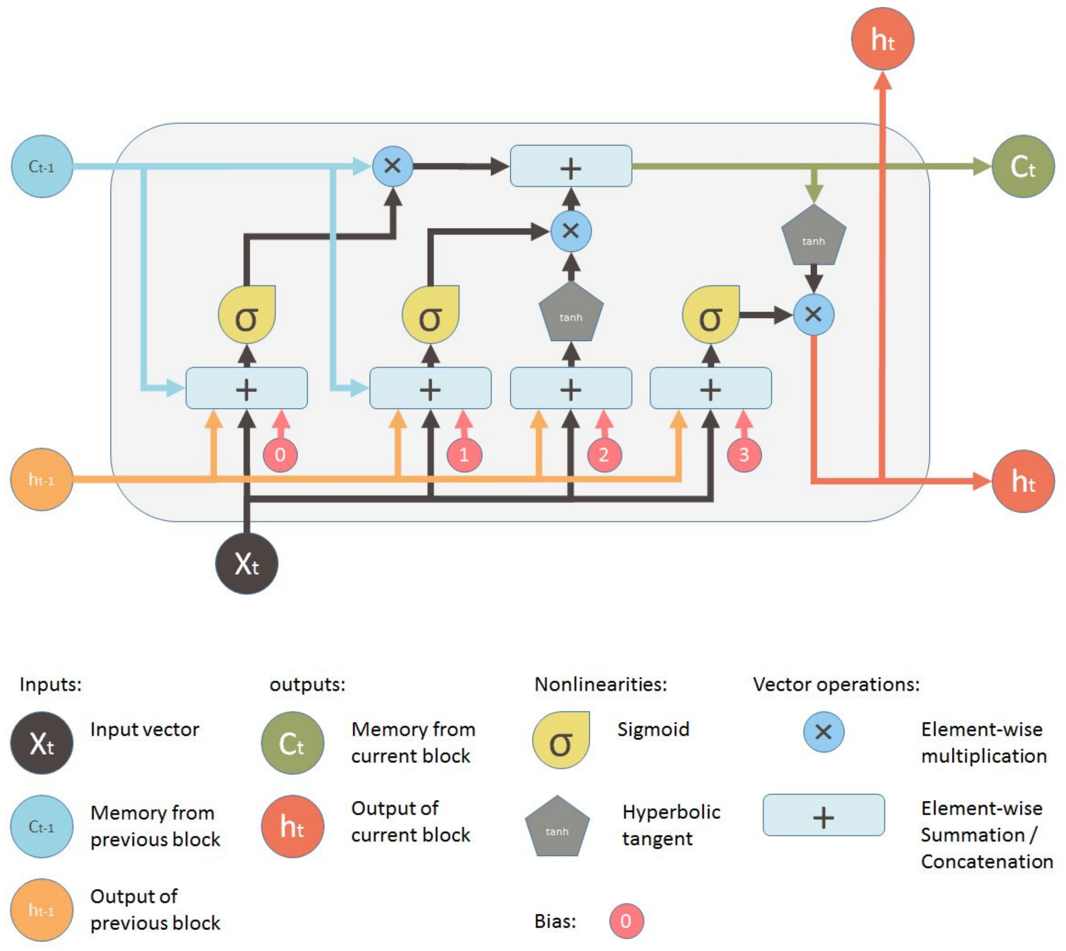
### XGBoost

XGBoost is an implementation of gradient boosted trees in which features are passed to a decision tree in order to classify data. The idea of gradient boosting is to combine the results of multiple weaker classifiers, and use these results to algorithmically develop a much stronger classification. XGBoost is highly regarded as one of the most accurate and powerful available classifiers, and is known to improve both model performance and computational speed due to its implementation based on boosted tree algorithms. XGBoost also provides compatibility with the eli5 library, which provides the capability to create analysis reports describing exactly how the classifier makes its decisions. This information can be used to modify the implementation accordingly to improve accuracy, making XGBoost a viable step toward the solution to the requirement tracing problem.

## LSTM

Long Short-Term Memory (LSTM) is a form of Recurrent Neural Network (RNN). RNNs allow information to be passed from one step of the network to the next, effectively creating a loop in the network. The problem with traditional RNNs is that if the gap from the relevant information to help identify context and the current input is large, the RNNs become unable to learn to connect the information. This is known as the problem of "long term dependencies". LSTM is an implementation of RNN that eliminates this problem, making it effective in identifying context and sentiment. Like traditional RNNs, LSTMs have a chain like structure of repeating modules. LSTM Networks have been used successfully in a wide variety of machine learning applications. Most importantly to us, it has been known to be effective in document classification. Thus, we have determined that an implementation of LSTM is a viable approach to exploring a solution to the requirement tracing problem.

Below is a diagram of an LSTM Network. The input ($X_t$) would be a vector representation of the next word in the requirement. $H_{t-1}$ is the output produced from the previous LSTM unit. $C_{t-1}$ is the "memory" of the previous LSTM unit. The memory and output of the current LSTM unit are calculated and represented as $h_t$ and $C_t$. The primary difference between an LSTM Network and a traditional RNN is the introduction of the concept of memory being passed between modules. The influence that the memory has on the overall classification is controlled within each module through a series of pointwise operations on the vectors.



## Implementation Details

### Web App Implementation

#### AWS

The frontend and backend run on separate EC2 instances. They are kind of like virtual machine instances that run Ubuntu 18.04.

Frontend:  The box runs a nginx web server and it serves the built version of the react frontend.  Nginx is chosen because of its high ability to handle static files.

Backend:  The box simply runs the django server in the background using nohup.  Simple scripts are written to make run/kill of the background process easier.

#### Continuous Integration / Continuous Delivery (CI/CD)

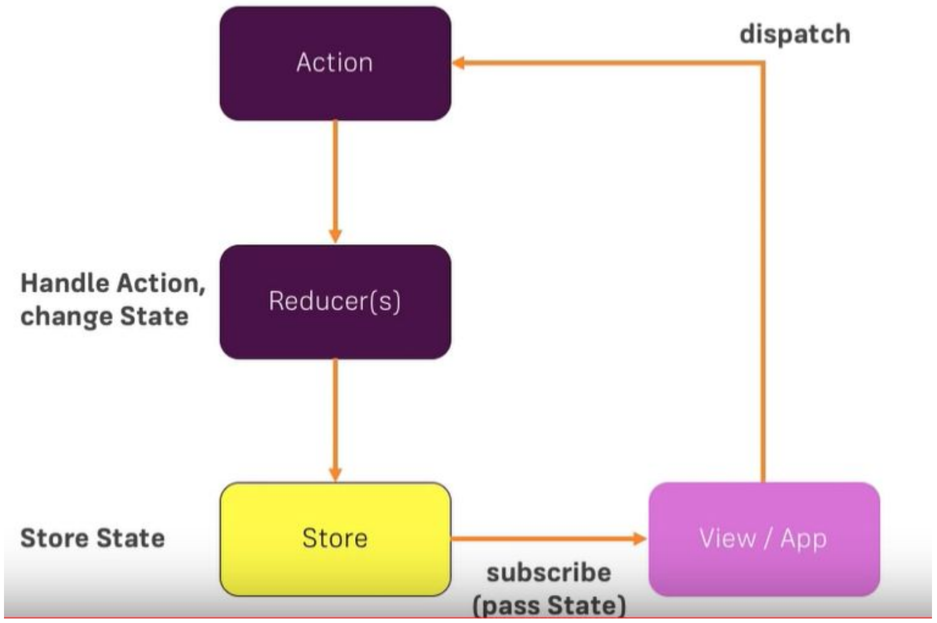
CI/CD pipeline consists of 3 steps.  Test, Build and Deploy.  Each step just creates a one-time docker container and all the operations are run on the container which runs on Gitlab.  The test step runs unit/integration tests.  The build step checks if the app can be built.  The deploy step deploys the code on AWS.  A different pipeline is setup for backend and frontend. Those are separated by the branch.  All the CI/CD operations are stored in .gitlab-ci.yaml file.

Frontend Deployment:  copy the built version of react frontend directly to /var/www/html folder on frontend EC2 instance using scp command.

Backend Deployment: The read-only repository is set up inside the backend EC2 instance. The deploy step ssh into the backend EC2 instance and runs git pull, kill the django server running in the background, and rerun the server.

## Frontend Implementation Details

Frontend state is shared on the component level as well as global level.  The global state is stored on Redux, an independent node library.  It works like a database on a frontend, making handling of asynchronous actions and cross-component data sharing easy.  Reducers are created for projects, configurations, users, and jobs.  Each reducer handles actions specific for each category.



The view layer is written in HTML and CSS. We utilized react-bootstrap package to make the development faster. Most of CSS is handled by bootstrap.

Everytime an updated data is needed, any of the components can fire up Redux asynchronous action, which queries the backend REST API.

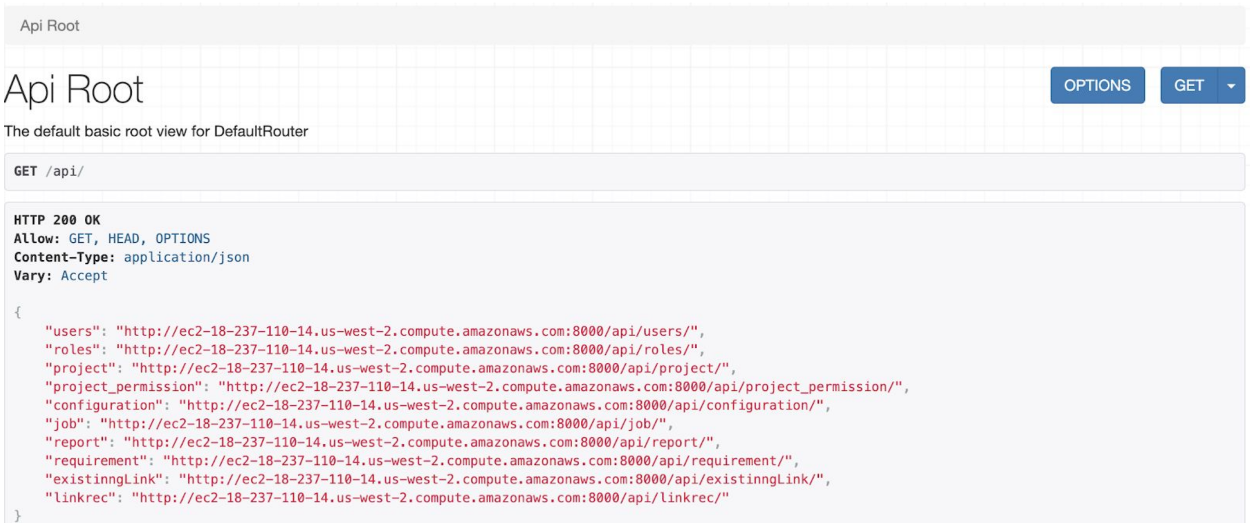
For the runner component, a PULL request will be sent to the run API endpoint, creating the job on the backend.  The backend will then update the job history, so if the user goes to the history page, the frontend would pull new history from the backend and displaying.

The API details are abstracted away using Redux actions (this is one of the reasons we chose Redux), so on the local development, we just use dummy json data without worrying about the backend.  The frontend is completely independent from the backend.

## Backend Endpoints

The backend service is build in Django Version 2.0. The API's are as follows:

API List : <address> : 8000/api



Api Root

The default basic root view for DefaultRouter

```
GET /api/
```

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

{
    "users": "http://ec2-18-237-110-14.us-west-2.compute.amazonaws.com:8000/api/users/",
    "roles": "http://ec2-18-237-110-14.us-west-2.compute.amazonaws.com:8000/api/roles/",
    "project": "http://ec2-18-237-110-14.us-west-2.compute.amazonaws.com:8000/api/project/",
    "project_permission": "http://ec2-18-237-110-14.us-west-2.compute.amazonaws.com:8000/api/project_permission/",
    "configuration": "http://ec2-18-237-110-14.us-west-2.compute.amazonaws.com:8000/api/configuration/",
    "job": "http://ec2-18-237-110-14.us-west-2.compute.amazonaws.com:8000/api/job/",
    "report": "http://ec2-18-237-110-14.us-west-2.compute.amazonaws.com:8000/api/report/",
    "requirement": "http://ec2-18-237-110-14.us-west-2.compute.amazonaws.com:8000/api/requirement/",
    "existinngLink": "http://ec2-18-237-110-14.us-west-2.compute.amazonaws.com:8000/api/existinngLink/",
    "linkrec": "http://ec2-18-237-110-14.us-west-2.compute.amazonaws.com:8000/api/linkrec/"
}
```

1. Users : /api/users/

   E-mail and the username are the unique values. The api should be able to get, post, delete and filter the data. Filtering is based on first name, last name, email, and privilege

2. Roles:

   Roles are the unique values. The api should be able to get and post the roles.

3. Project

   Project name should be unique. The api should be able to get and post any project.

4. Project_permission

   Permission to view, change, and add a project would be functionalized in the api. The api allows the get and post operation.

5. Configuration

   Configuration api add configuration for the selected project. It takes in the project id, user id, and the configuration values. The api allows, get and post operation

6. Job

   Job api creates the job. The allowed operations are get, post, delete and filtering based on project id, config id, priority, start time, and status of the job.

7. Report

   Create a report link along with the information like user who created the job as well as project name.

8. requirement

   Requirement api saves the requirement created by the client's requirement json generated by the Client's AI module

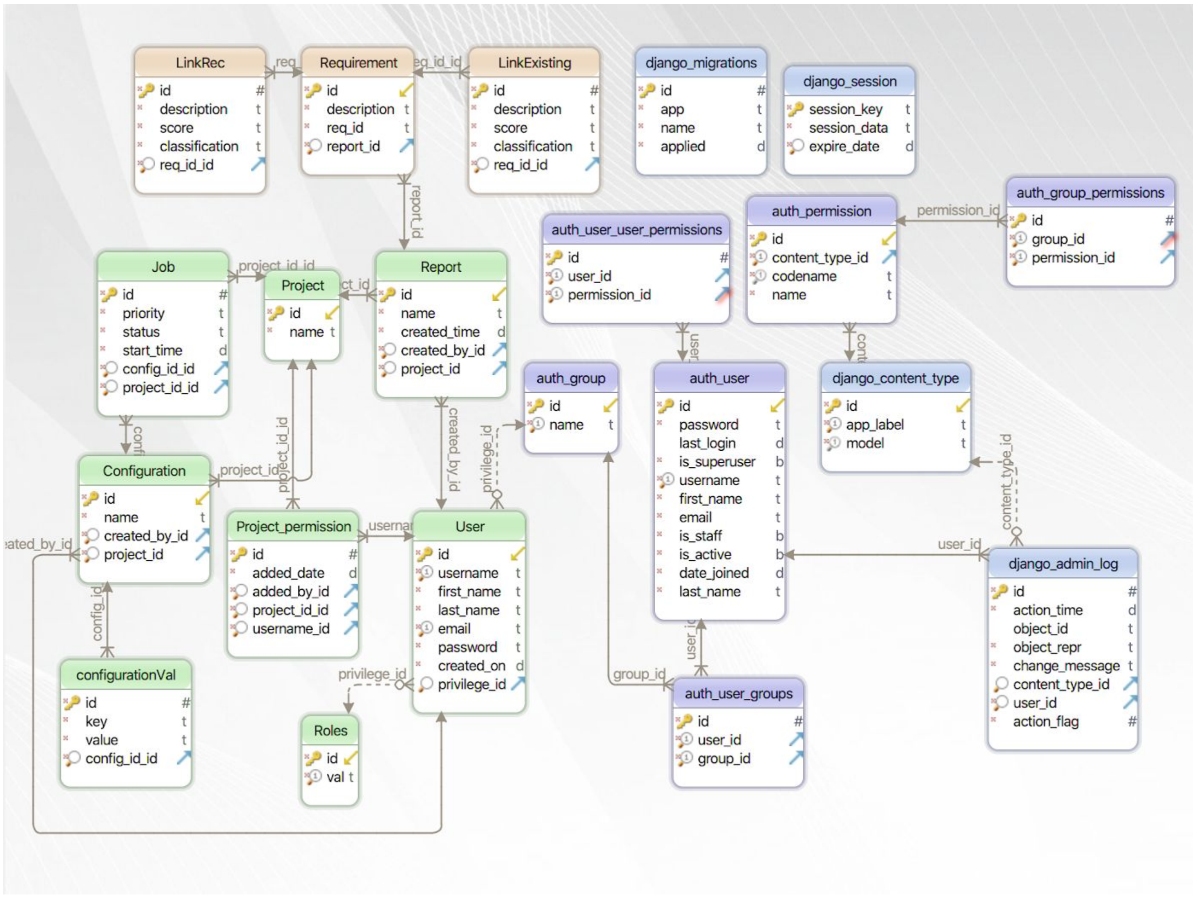
9. existingLink:

   existingLink api saves the existingLink created by the client's existing json generated by the Client's AI module

10. linkRec:

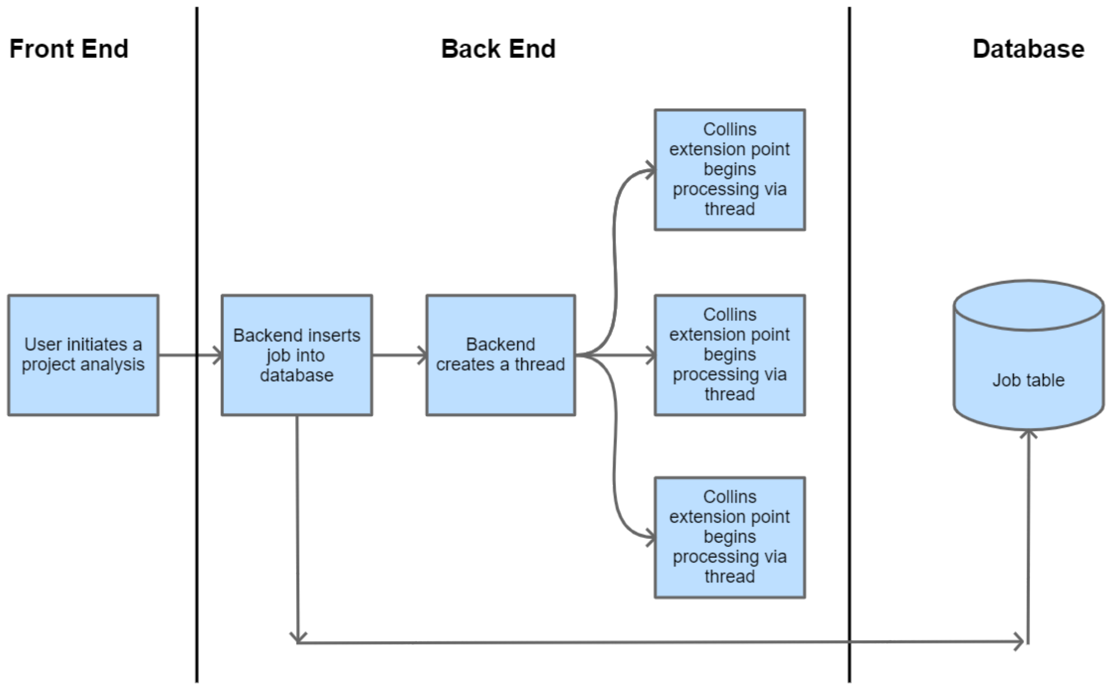
    linkRec api saves the existing requirement created obtained from Client's AI Module.

## Database Structure



## Integration Point

The integration point is a vital component of the project as it allows Collins to connect their code to the webapp. The process follows the diagram below. First, the user initiates a project analysis on the front end. The run request is sent to the backend which continues by inserting a job into the database. Once the job is instantiated, the backend generates a thread. Within the thread, Collins will write code to call their process while the backend can continue handling requests.


Front End
Back End
Database
User initiates a project analysis
Backend inserts job into database
Backend creates a thread
Collins extension point begins processing via thread
Collins extension point begins processing via thread
Collins extension point begins processing via thread
Job table

## Machine Learning Implementations

### Current Implementations

In the current implementation of XGBoost, two important features from the high level and low level requirements are extracted; a Bag of Words and the Term Frequency-Inverse Document Frequency (TF-IDF). A bag of words is the representation of a collection of words (such as a requirement) that disregards sentence structure but retains memory of the frequency that each word is used. TF-IDF is a numerical representation of the importance of a word within a document based on the frequency it is used in a single sentence compared to the frequency it is used in the training corpus. These two pieces of information are fed into a decision tree, which is able to classify the high level requirement and low level requirement as either a good or bad link. A diagram of the approach suggested by Collins is shown below.

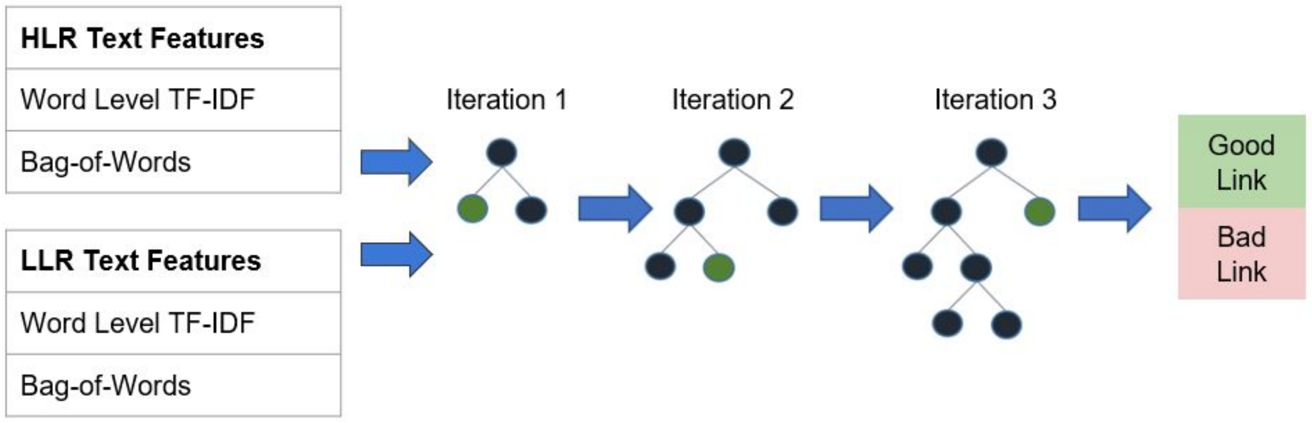


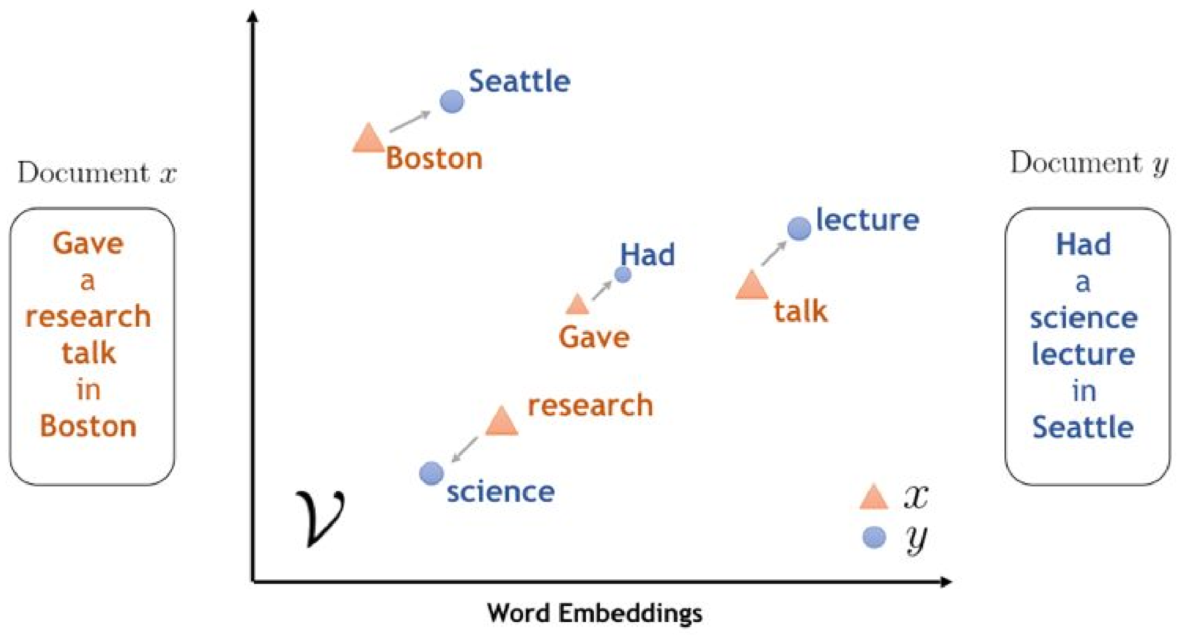
Diagram provided by Collins Aerospace

Currently, Collins does not have a working implementation of LSTM that they have accepted. Thus, they have been unable to develop a Model Stacking algorithm in order to combine the results of the XGBoost and LSTM classifiers.

## Suggested Alternatives

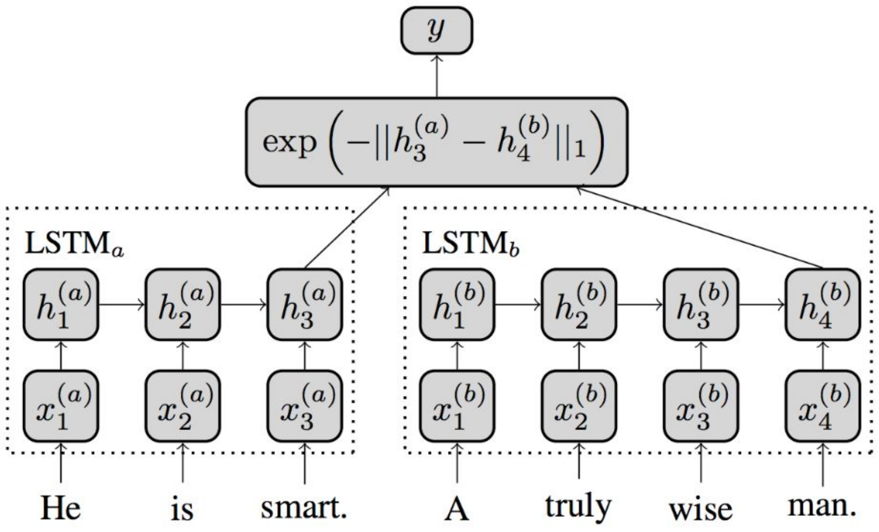
After researching a number of alternative implementations involving XGBoost or LSTM, two stood out as the most promising candidates: An XGBoost implementation to boost the results of Word2Vec, and an LSTM implementation based on Siamese Networks. Both of these approaches yielded a promising classification accuracy, but it must be noted that Collins' requirement data access was restricted to our team due to security restrictions. To work around this, models were trained using Google News data and classification of Quora questions as either "related" or "unrelated" as a proxy for requirement link classification.

The alternate implementation of XGBoost relies on boosting the results produced by Word2Vec/Word Mover's Distance. This approach embeds each word of a document (in this case, a Quora question) into a vector, and calculates the similarity between two questions by determining the minimum distance that one question needs to travel within the vector space in order to reach another question's vector representation. An example of this can be seen in the diagram below.



The Word2Vec model extracts a variety of features including Cosine Distance, Manhattan Distance, Jaccard Distance, etc. in order to make its classification based on these distances. Then, XGBoost sequentially adds new models on top of the existing Word2Vec model in order to improve the results until the underlying algorithm in the XGBoost model determines that the results have been optimized as much as possible.

Based on the research completed, an LSTM implementation relying on Siamese Networks was suggested as a potential approach. As suggested by their name, Siamese Networks involve implementing two or more identical networks. Again, the model is trained using the pre-trained Google News word embeddings For the suggested approach, two identical LSTM networks were implemented. LSTM is executed on each of the questions in the pair during comparison as described in the diagram below.



$y$
$\exp\left(-\|h_3^{(a)} - h_4^{(b)}\|_1\right)$
LSTM$_a$
LSTM$_b$
$h_1^{(a)}$ $h_2^{(a)}$ $h_3^{(a)}$
$x_1^{(a)}$ $x_2^{(a)}$ $x_3^{(a)}$
$h_1^{(b)}$ $h_2^{(b)}$ $h_3^{(b)}$ $h_4^{(b)}$
$x_1^{(b)}$ $x_2^{(b)}$ $x_3^{(b)}$ $x_4^{(b)}$
He is smart.
A truly wise man.

After executing LSTM on each of the questions in the pair, the resulting vector outputs of the LSTM networks hold the "semantic meaning" of each question. These results are compared using LSTM's defined comparison function in order to determine the similarity of the two questions.

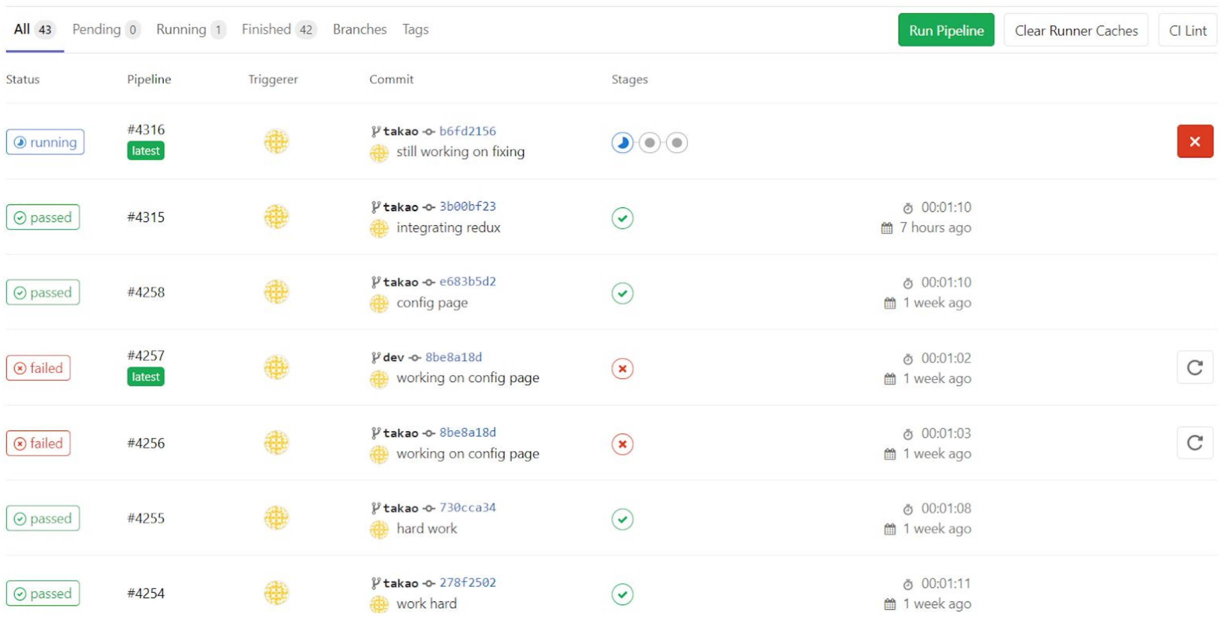
### Machine Learning Results

The implementation of XGBoost developed by the Collins engineering team yielded a classification accuracy of 82%. The suggested alternative XGBoost implementation that relies on boosting Word2Vec results yielded a slightly lower classification accuracy of 76%. The accuracy of the suggested alternative has not been tested on Collins' data due the security protocol mentioned previously. Thus, a fair conclusion about the feasibility of this approach cannot be made until engineers at Collins test it against their requirement data. The suggested approach of implementing LSTM with a Siamese Network yielded a classification accuracy of 78%, but also must be tested with Collins requirement data in order to make a fair conclusion about the results.

## Testing Process and Results

### CI/CD Pipeline

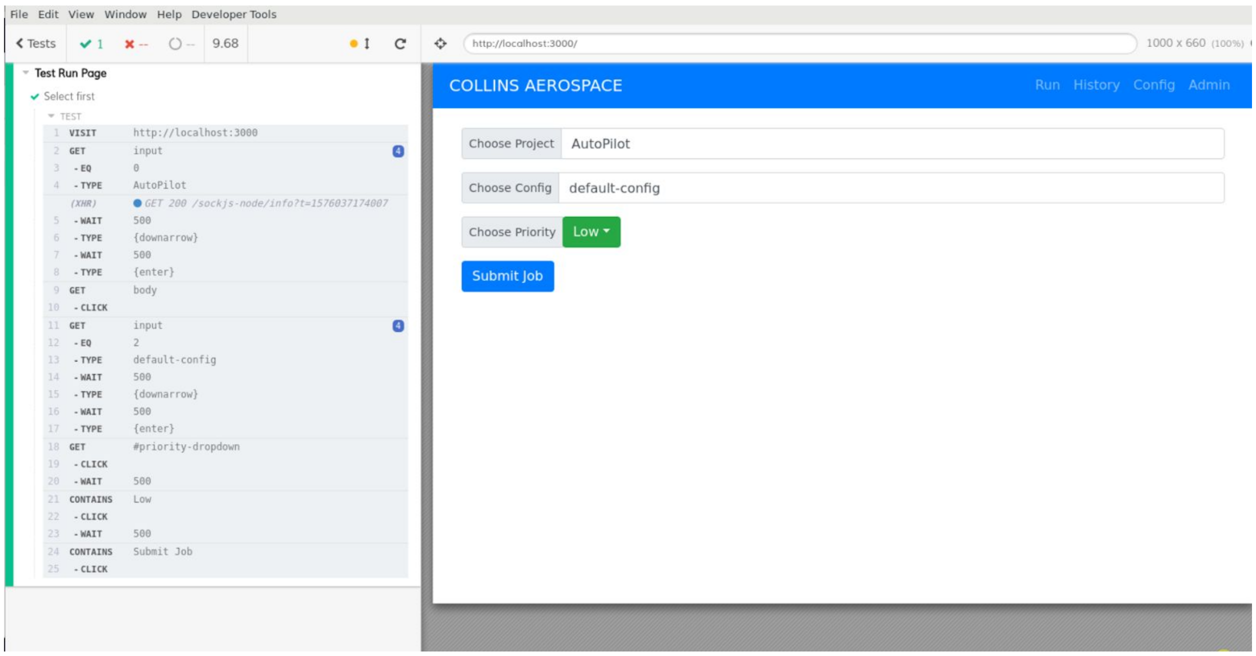
The picture below is how a developer sees the pipeline progress on Gitlab CI/CD.  There are three states for each job: running, passed, and failed.  The developer will receive the result for each job and it's clear what needs to be fixed before the new feature or bug fix can be deployed.


All 43 | Pending 0 | Running 1 | Finished 42 | Branches | Tags — Run Pipeline | Clear Runner Caches | CI Lint

| Status | Pipeline | Triggerer | Commit | Stages | | |
|---|---|---|---|---|---|---|
| running | #4316 latest | | takao -o- b6fd2156 still working on fixing | | | × |
| passed | #4315 | | takao -o- 3b00bf23 integrating redux | ✓ | 00:01:10 7 hours ago | |
| passed | #4258 | | takao -o- e683b5d2 config page | ✓ | 00:01:10 1 week ago | |
| failed | #4257 latest | | dev -o- 8be8a18d working on config page | × | 00:01:02 1 week ago | |
| failed | #4256 | | takao -o- 8be8a18d working on config page | × | 00:01:03 1 week ago | |
| passed | #4255 | | takao -o- 730cca34 hard work | ✓ | 00:01:08 1 week ago | |
| passed | #4254 | | takao -o- 278f2502 work hard | ✓ | 00:01:11 1 week ago | |

## System Test With cypress.io

The user behavior is tested using cypress.io testing framework. As shown in the picture, behaviors such as visiting the URL, typing in the input boxes, selecting the dropdown, and pressing the button are emulated. Tester can easily see the testing progress in real-time and it's very visual and easy to tell what went wrong. This test is currently run on local machine but it will eventually be included in the CI/CD pipeline.



Test Run Page
✓ Select first
TEST

```
VISIT      http://localhost:3000
GET        input
 - EQ      0
 - TYPE    AutoPilot
(XHR)      GET 200 /sockjs-node/info?t=1576037174007
 - WAIT    500
 - TYPE    {downarrow}
 - WAIT    500
 - TYPE    {enter}
GET        body
 - CLICK
GET        input
 - EQ      2
 - TYPE    default-config
 - WAIT    500
 - TYPE    {downarrow}
 - WAIT    500
 - TYPE    {enter}
GET        #priority-dropdown
 - CLICK
 - WAIT    500
CONTAINS   Low
 - CLICK
 - WAIT    500
CONTAINS   Submit Job
 - CLICK
```

COLLINS AEROSPACE — Run | History | Config | Admin

| | |
|---|---|
| Choose Project | AutoPilot |
| Choose Config | default-config |
| Choose Priority | Low |

Submit Job

## API TEST With Postman

After creating all of the endpoints for our REST API, Postman was used as a tool for ensuring that the REST API endpoints were active and stable. After hitting each endpoint, we confirmed that all endpoints are working as expected.

# Operation Manual

## Installation and Setup

First, an Ubuntu 18.04 server must be created, and the repo pushed to the server.

### Backend Setup

The backend service includes Django - version 2.2. Libraries includes Django Rest Framework and Django Filters. For the installation of the backend service python3.7 is crucial and required. Moreover pip also needs to be installed in order to install the libraries.

Installation guide for python3.7 : https://www.python.org/downloads/

A file called requirement.txt is provided to the Client that contains the library list. The client needs to run the following command in order to install the libraries on the environment:

"pip install -r requirements.txt"

Once the installation of python3.7 and required libraries are successful. Client would be able to run the backend service. However, before the backend service is started, the models needs to be migrated. Thus, following commands should be run in order:

1) Navigate to the project folder where manage.py file is located
2) Run: python3.7 manage.py makemigrations
3) Run: python3.7 manage.py migrate
4) Run: python3.7 manage.py runserver

The runserver will start the service on port 8080.

### Frontend Setup

Run the following commands

- npm install (install packages)
- npm start (start dev server)

### Frontend Deployment on custom server

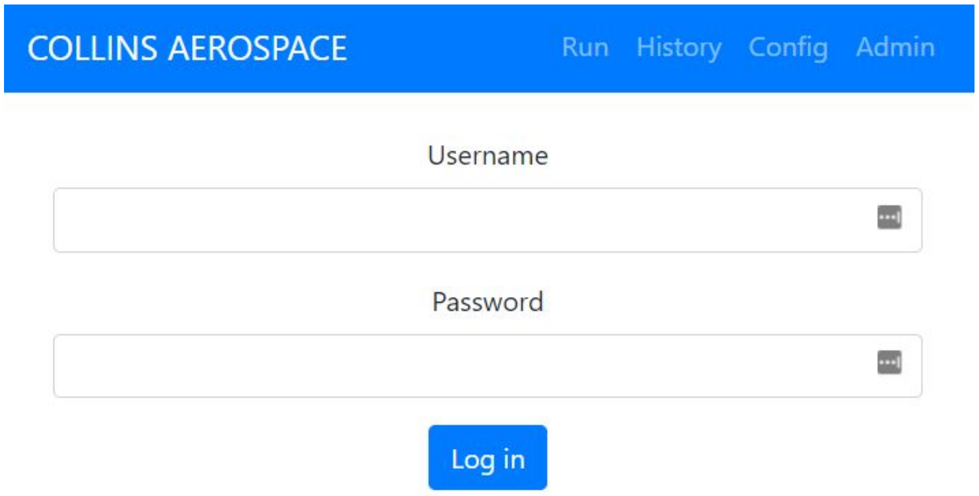
Change FRONTEND_SERVER variable on the gitlab repository or you could deploy to customer server manually. npm run build - will pack all the jsx files into plain javascript, html, css files using webpack. This built version can be simply put on a web server. HOWEVER, since react-router is used to display multiple pages, all request to url other than the root needs to be redirected to index.html. This settings can be done by modifying /etc/nginx/sites-available/default to the following.

Change: try_files $uri $uri/ =404;

To: try_files $uri $uri/ /index.html;

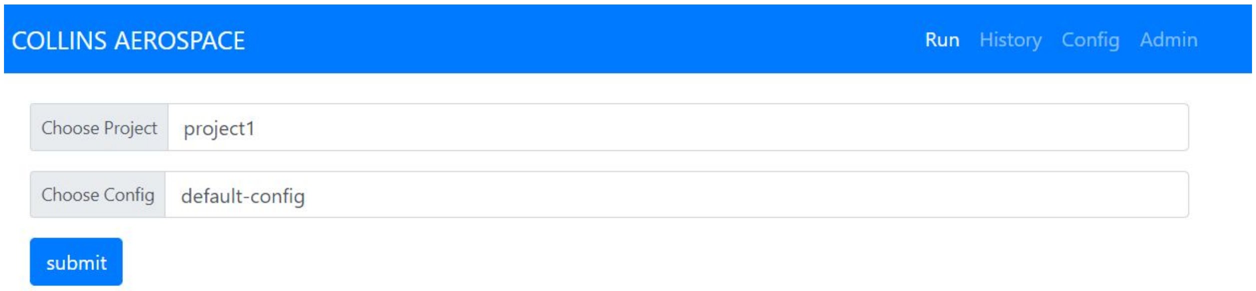
## User Guide

### Login Page



The user will log in via the login page above

### Run Page

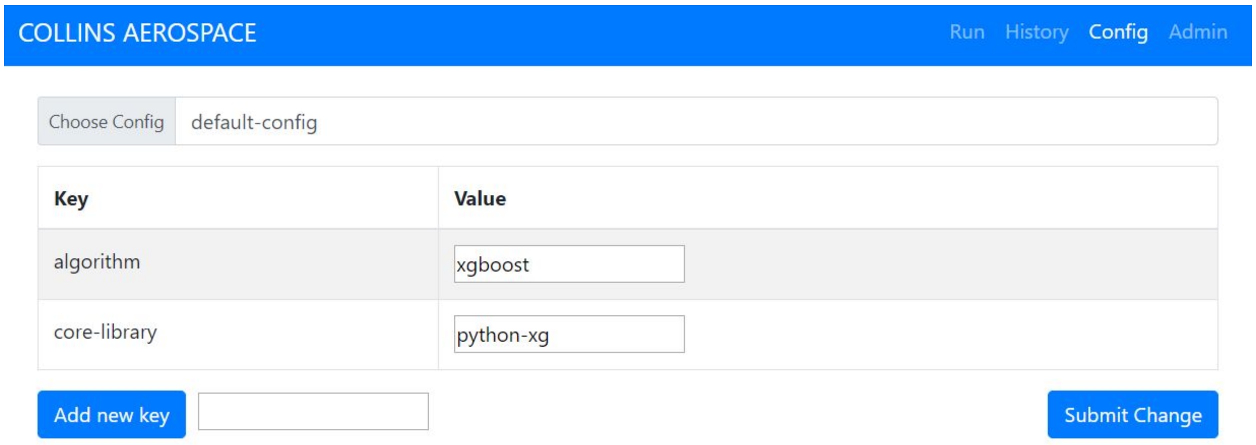


Choose Project: project1

Choose Config: default-config

The run page is where users initiate the analysis process. The user will use the input boxes to select a project and configuration which they have access to. Once they have the appropriate specifications, they will click submit, and the run process will be initiated.

## Configuration Page



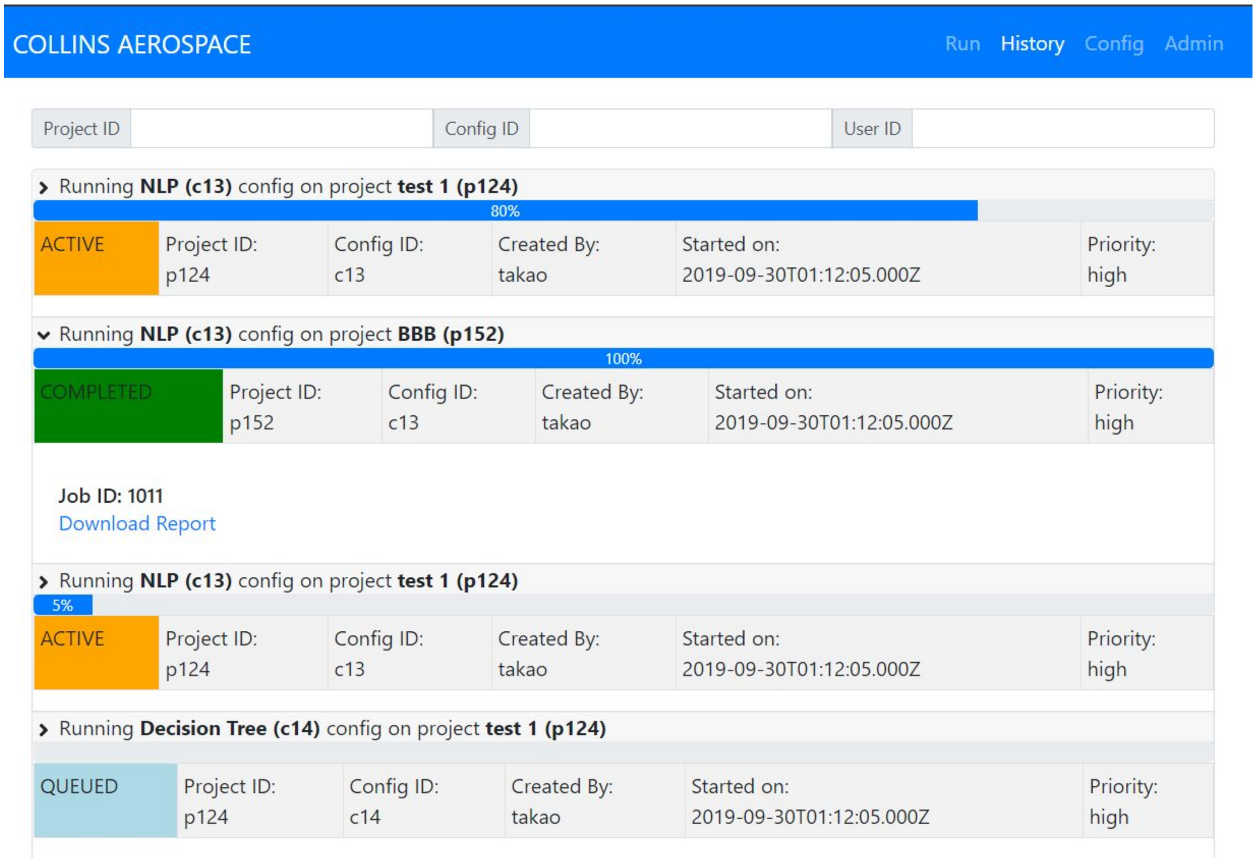
Choose Config: default-config

| Key | Value |
|---|---|
| algorithm | xgboost |
| core-library | python-xg |

In the previously seen run page, the second input box allowed for the user to choose a configuration to run the analysis with. If the user would like to create one of these configurations, they will navigate to the config page. These are specified as a list of key value pairs. The steps of creating this are as follows:

1. Specify the config you would like to create/modify with the choose config input
2. To edit an existing value, modify the value text box with the appropriate key
3. To add a new key:
    a. Enter the key name in the box next to “Add new key”
    b. Click “Add new key”
    c. In the value box next to the new key, add the appropriate value
4. Once the user has finished, click submit config to create the configuration

## History Page



› Running **NLP (c13)** config on project **test 1 (p124)**

80%

| | | | | | |
|---|---|---|---|---|---|
| ACTIVE | Project ID: p124 | Config ID: c13 | Created By: takao | Started on: 2019-09-30T01:12:05.000Z | Priority: high |

⌄ Running **NLP (c13)** config on project **BBB (p152)**

100%

| | | | | | |
|---|---|---|---|---|---|
| COMPLETED | Project ID: p152 | Config ID: c13 | Created By: takao | Started on: 2019-09-30T01:12:05.000Z | Priority: high |

Job ID: 1011
Download Report

› Running **NLP (c13)** config on project **test 1 (p124)**

5%

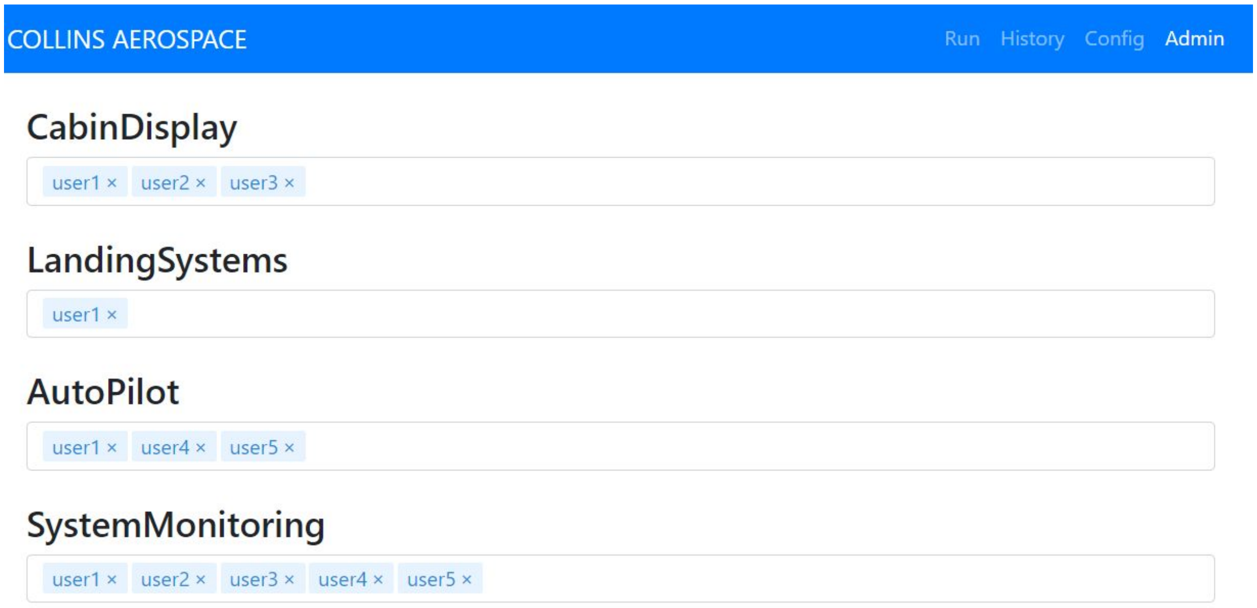
| | | | | | |
|---|---|---|---|---|---|
| ACTIVE | Project ID: p124 | Config ID: c13 | Created By: takao | Started on: 2019-09-30T01:12:05.000Z | Priority: high |

› Running **Decision Tree (c14)** config on project **test 1 (p124)**

| | | | | | |
|---|---|---|---|---|---|
| QUEUED | Project ID: p124 | Config ID: c14 | Created By: takao | Started on: 2019-09-30T01:12:05.000Z | Priority: high |

After a user has started a job by using the run page, they may navigate to the history page to see information about that job. The two most important things to see here are the job status, and the download report link. When the report is completed, there will be a link present which will download an excel report to view the results.

## Admin Page



CabinDisplay

user1 × user2 × user3 ×

LandingSystems

user1 ×

AutoPilot

user1 × user4 × user5 ×

SystemMonitoring

user1 × user2 × user3 × user4 × user5 ×

Finally, the admin page is available to any user who is a superuser or an admin. The admin page allows the user to see all of the projects they have access to. For these projects, they may specify which users have access to which projects by simply adding the user to the appropriate lane under the project name.

## Extending the Code

The context of this project includes integrating the machine learning tool created by Collins into the web app. This will be done with an API extension on the ML tool side. Within the backend of the web app, there is a python file called run.py. When a user initiates an analysis from the web app, this file is called and two things will be done. First, a job will be inserted into the database with a status of "In Progress". Then, a thread will be kicked off for calling the Collins machine learning tool.

For Collins to integrate the tool into the webapp, run.py must be modified to initiate the analysis process, and then update the job status to "Complete" when the analysis is finished.