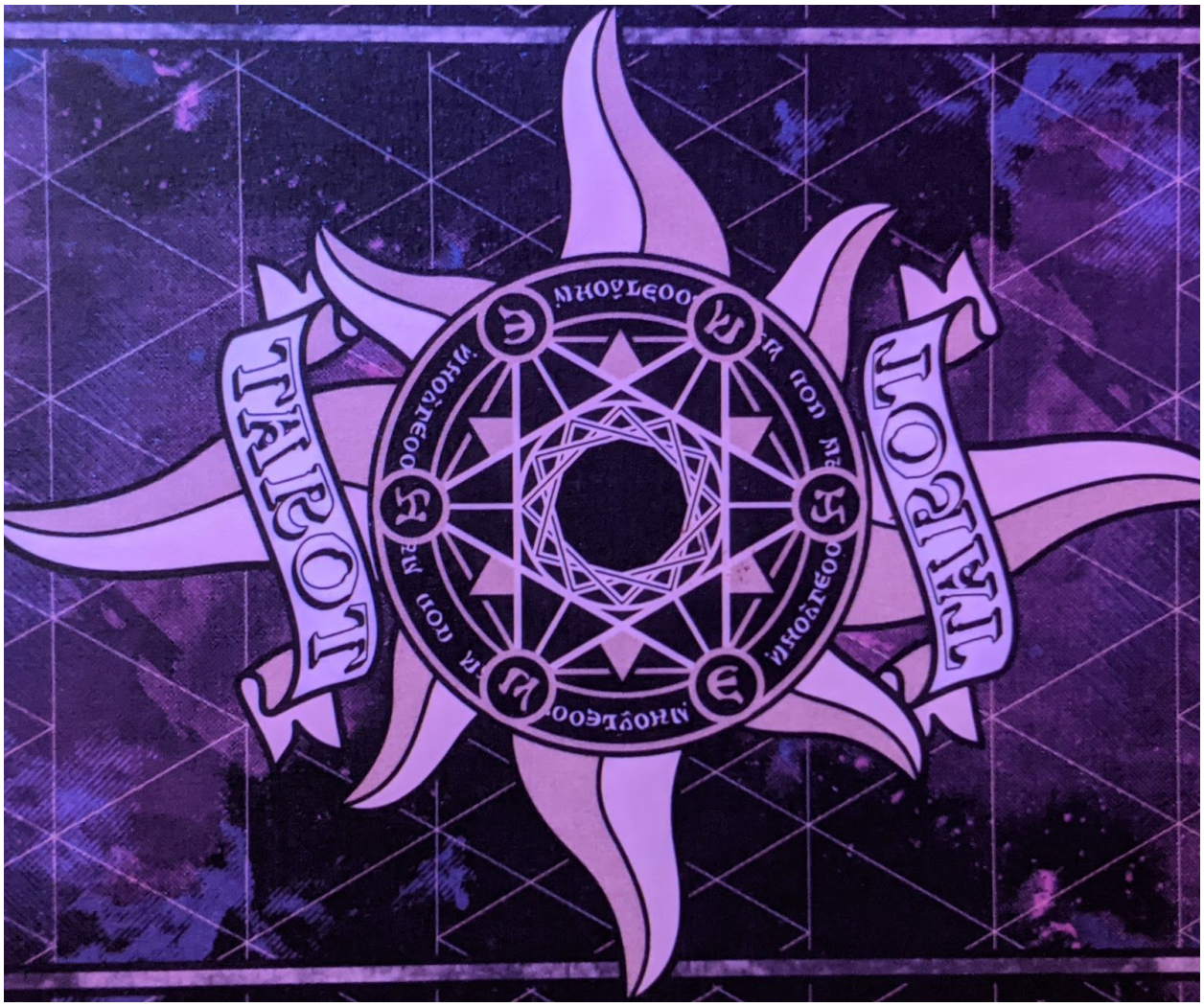


TAROTBOT

12.05.2020

CHRISTOPHER RICO
MSDS 434

PROJECT OVERVIEW

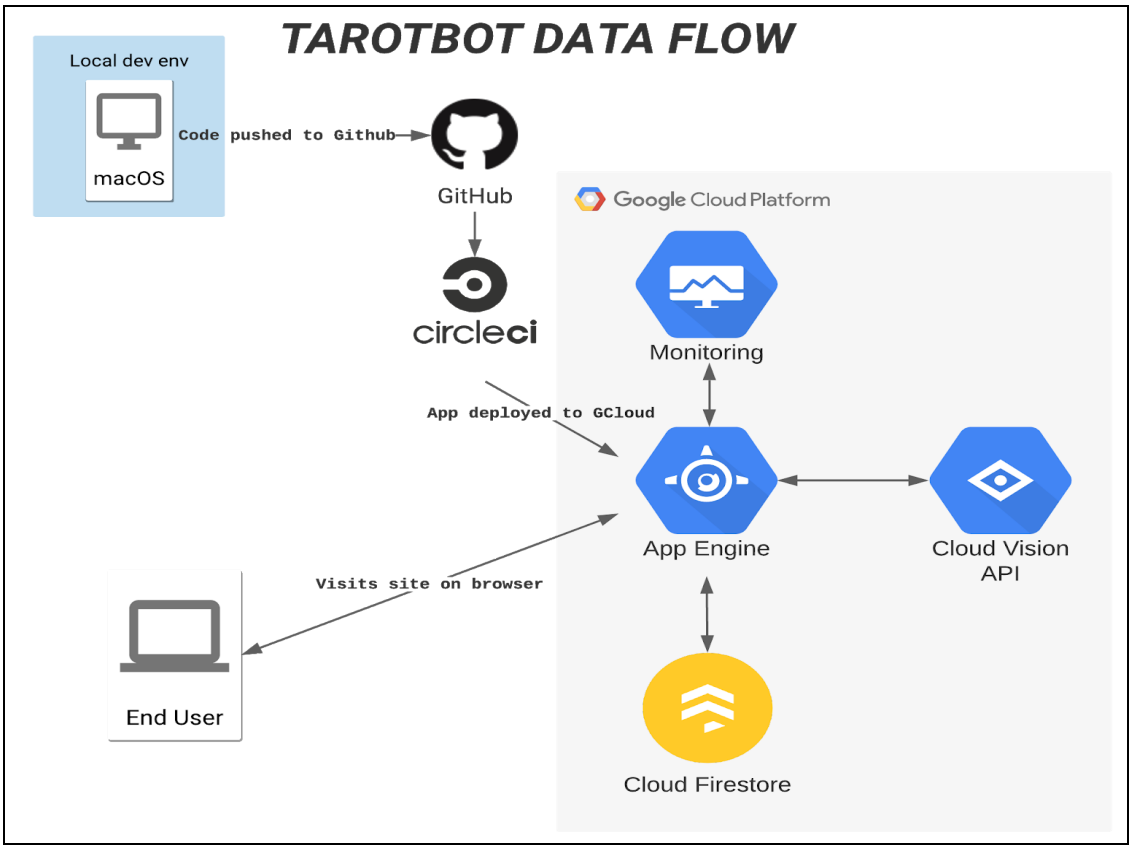
Our group was hired by Zoltar, Inc., to develop a proof of concept system that would detect and classify tarot cards from an image submitted via website. The solution was developed using Google AutoML Vision to train a custom object detection and multi-class classification model. Using AutoML to build and deploy the model allowed for rapid development and integration of computer vision capability within a basic Flask app.

TOOLS USED

- **Pipenv:** Pipenv was used to create a virtual development environment within a Unix shell running on my local macOS machine.
- **Flask:** Flask operates as the central point of data flow within the app -- it serves out an interactive website where users can submit a photo and see the results of the machine learning inference.
- **Git/Github:** Development branch on Git stored commits to the dev build of the app. Once a working version was developed and tested via Pytest, the dev branch was merged with the main branch and pushed to GitHub.
- **CircleCI:** CircleCI was used to incorporate continuous integration into the project. Anytime code was pushed to the main branch of the GitHub repository, CircleCI was triggered to build and test the newest version of the app before deploying to Google App Engine.
- **Google App Engine:** GAE was used as the server to serve out the Flask app, which is a basic website that accepts user input of a photo, connects to the AutoML Vision API to get an inference, then stores said inference in Firestore. When users view the results page of the app, it retrieves past stored inferences from Firestore.
- **Google AutoML Vision:** This is the service that was used to train and evaluate the object detection and classification model, as well as deploy the completed model to the cloud so that the Flask app could connect to it via the Vision API client libraries.
- **Google Firestore:** Firestore is used as a NoSQL database to store tarot inference results. The Flask app stores individual results to Firestore as they are returned, and also requests all existing inference results to display to the user.
- **Google Cloud Monitoring:** Cloud monitoring provides logging of any errors that may occur. Cloud monitoring was also used to set uptime checks, and send alerts in case the app was unreachable for more than one minute.

DATA FLOW

Figure 1 below shows the entire dataflow pipeline, beginning with a user submitting a photo within the webpage form and ending with the results of the image inference displayed in text on a webpage after being stored in Firestore.
Further shown is the development/deployment pipeline, from a local development environment, to GitHub to testing on CircleCI to production deployment in Google App Engine.



RECOMMENDATIONS

The project team recommends the following project improvements, which were outside the scope of this project:

- Train a model to detect and classify all 78 clases of tarot card instead of just the 22 major arcana, as well as relative positions of the cards.
- Integrate Firebase into the project to allow for mobile users to sign in and see their results via the web browser.
- Integrate AJAX or JQuery into the webpage to display AutoML inference as soon as it is returned from the Vision API.
- Set up a QC environment to perform system testing within Google Cloud before build is pushed to GAE production environment.

APPENDIX A

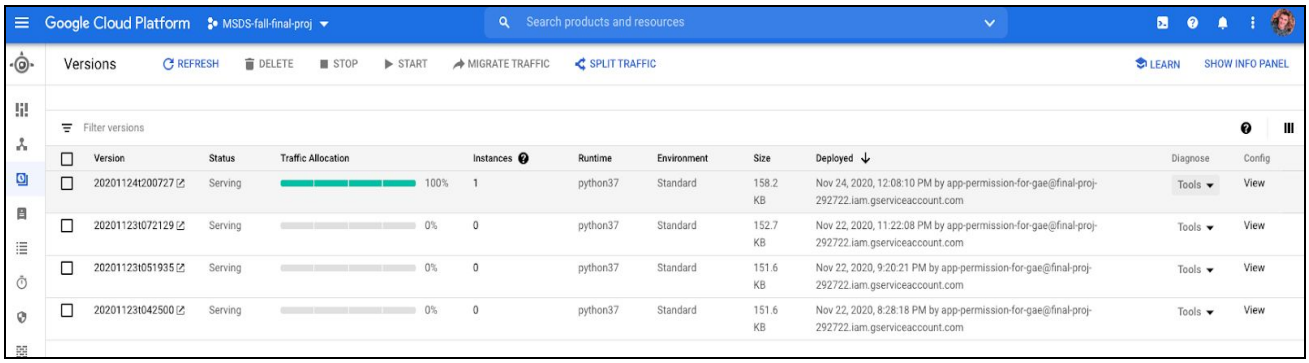
Filename	Directory	Description
Pipfile(.lock)	./	These detail pipenv requirements
GCPkeys.json	./	Google Cloud Platform service credentials.
Config.yml	./.circleci	CircleCI build, test, and deploy instructions.
App.yaml	./app	GAE app deployment configuration
GCPkeys.json	./app	Google Cloud Platform service credentials.
main.py	./app	This is the main program. Flask app that sets up routes to serve a website. Integrated with Vision API helper and Firebase helper.
main_test.py	./app	Unit tests to check that server, AutoML model, and firebase integrations are all online.
image_handler.py	./app	Code to integrate Flask app with AutoML Vision API. Sends image data to model, then translates model inference results into plain text.
firestore_handler.py	./app	Code to integrate Flask app with with Firestore. Set/Get methods store model inference results and retrieve all past inference results.
config.py	./app	Basic configuration for Flask app in dev environment.
index.html	./app/templates	Homepage for app, with form to submit photo
results.html	./app/templates	Results page which shows all previous AutoML inference results from Firestore..
noresults.html	./app/templates	Page that displays when no results are found in Firestore.
404.html	./app/templates	Page to display a pretty 404 error
500.html	./app/templates	Page to display a pretty 500 error.

APPENDIX B

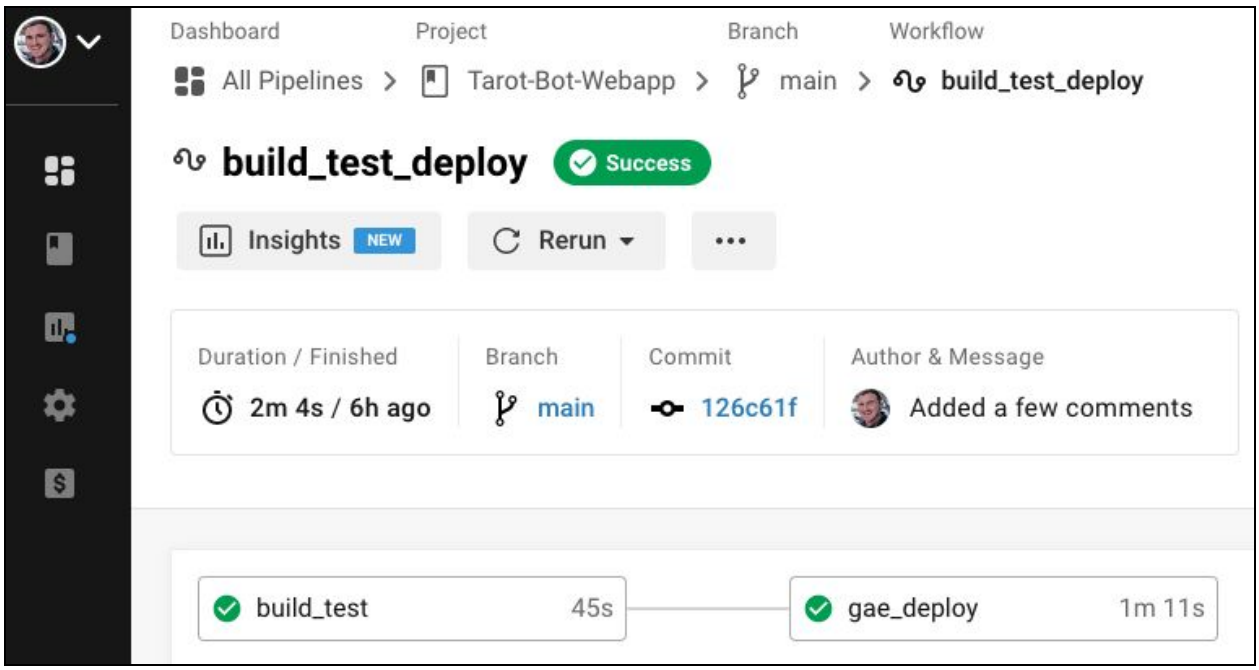
Local dev environment

```
[06:23:43 (tb-wa-M-VRbLgj) christophrico@Hackintosh tb-wa ±|dev ×|→ ls
GCPkeys.json    Pipfile         Pipfile.lock    README.md       app             assets
[06:23:45 (tb-wa-M-VRbLgj) christophrico@Hackintosh tb-wa ±|dev ×|→ ls ./app
GCPkeys.json            config.py               image_handler.py        main_test.py            templates
app.yaml                firestore_handler.py    main.py                 static
[06:23:50 (tb-wa-M-VRbLgj) christophrico@Hackintosh tb-wa ±|dev ×|→ python3 ./app/main.py
 * Serving Flask app "main" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 390-994-615
```

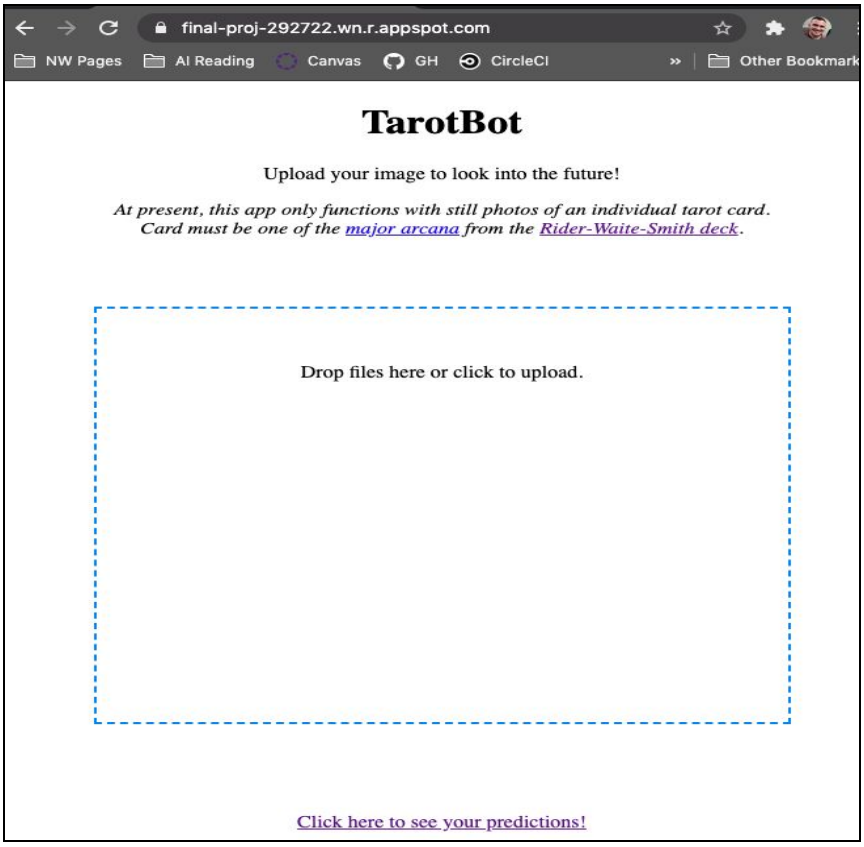
GAE production environment



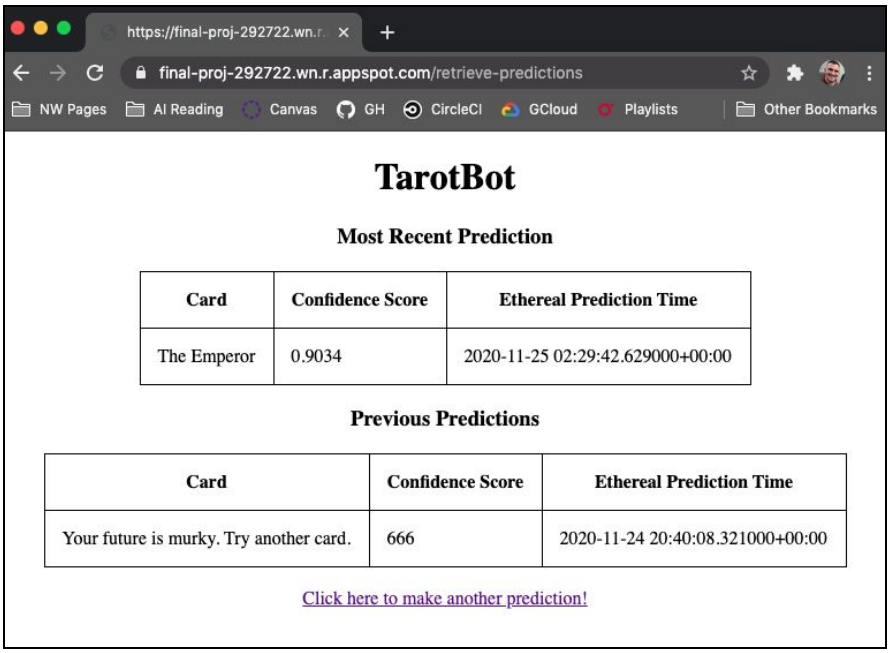
CircleCI successful build



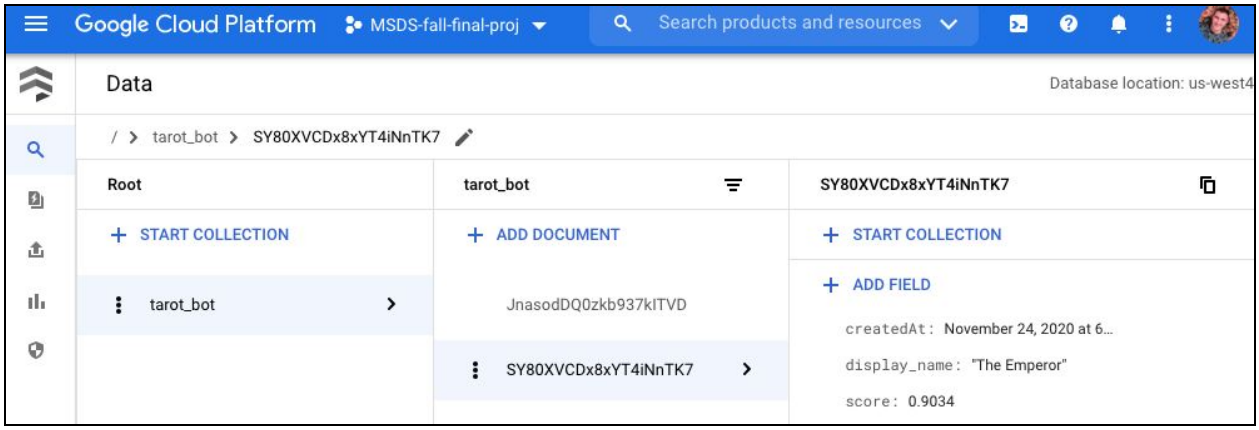
Submission page



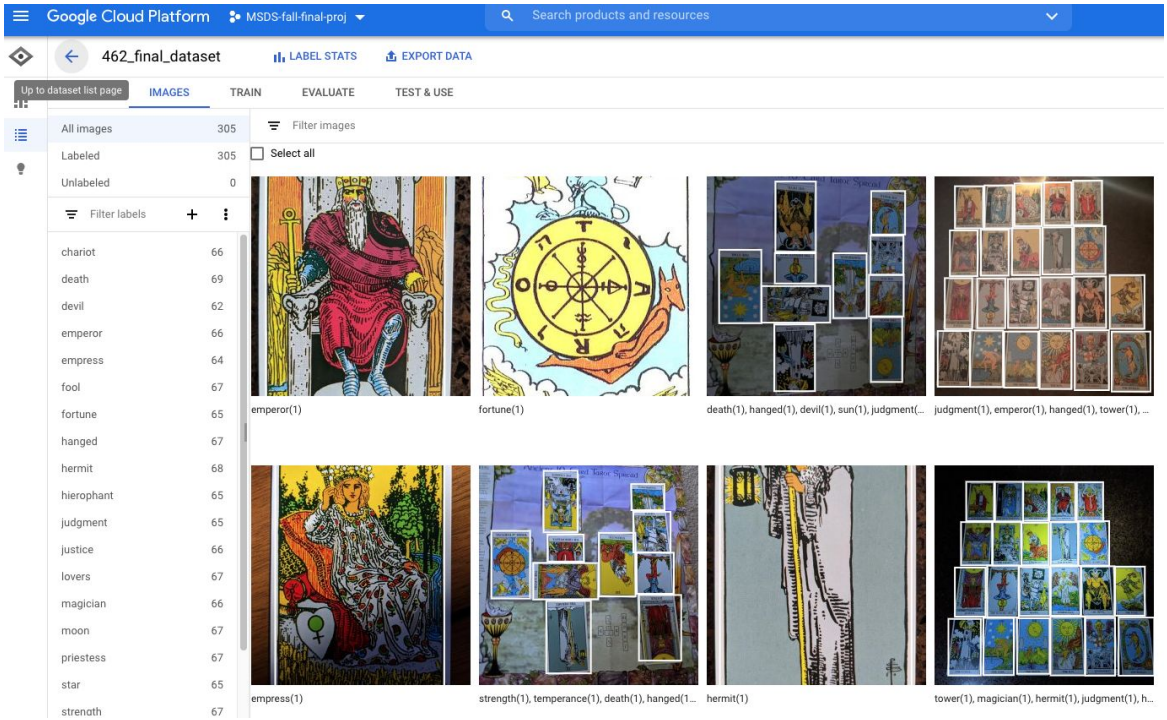
Results page

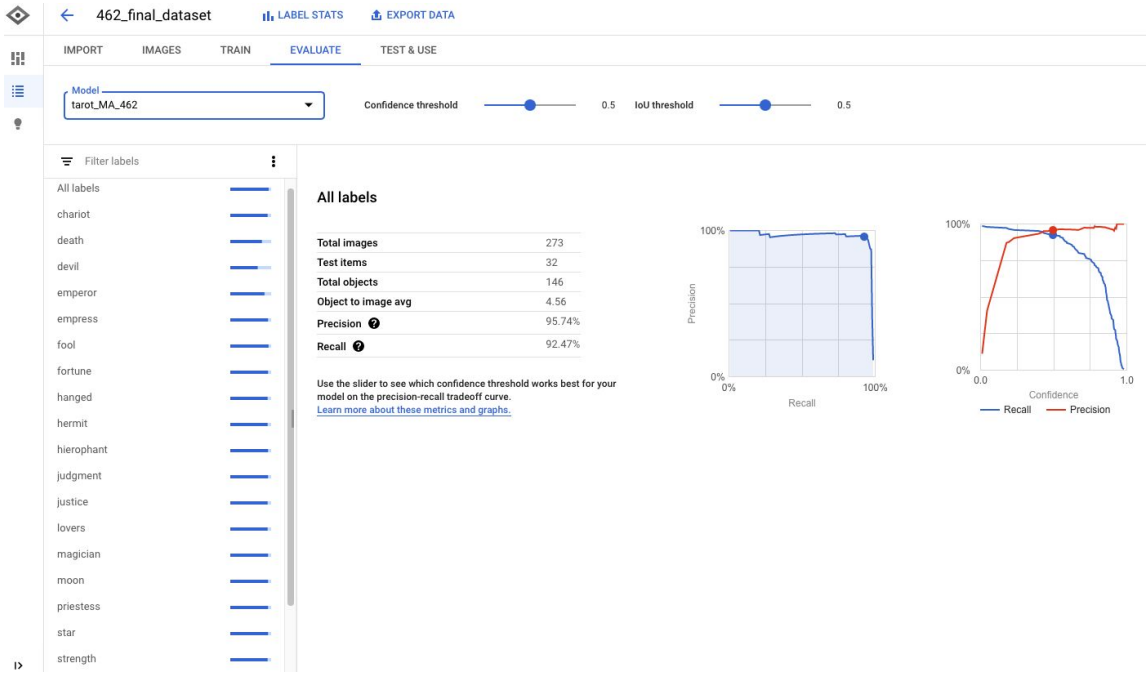


Firebase stored data

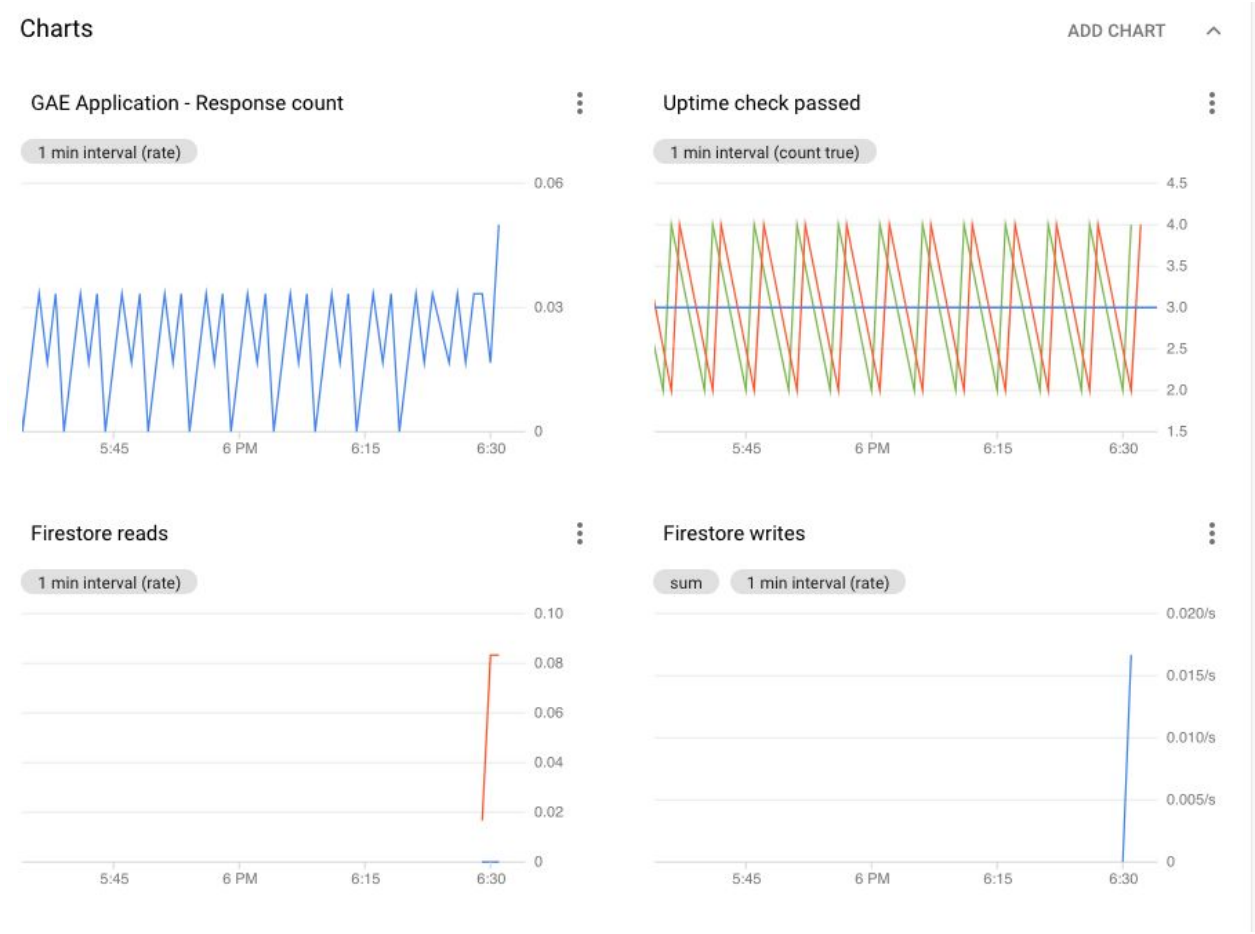


Cloud vision model training results





Monitoring dashboard



Uptime alert setup