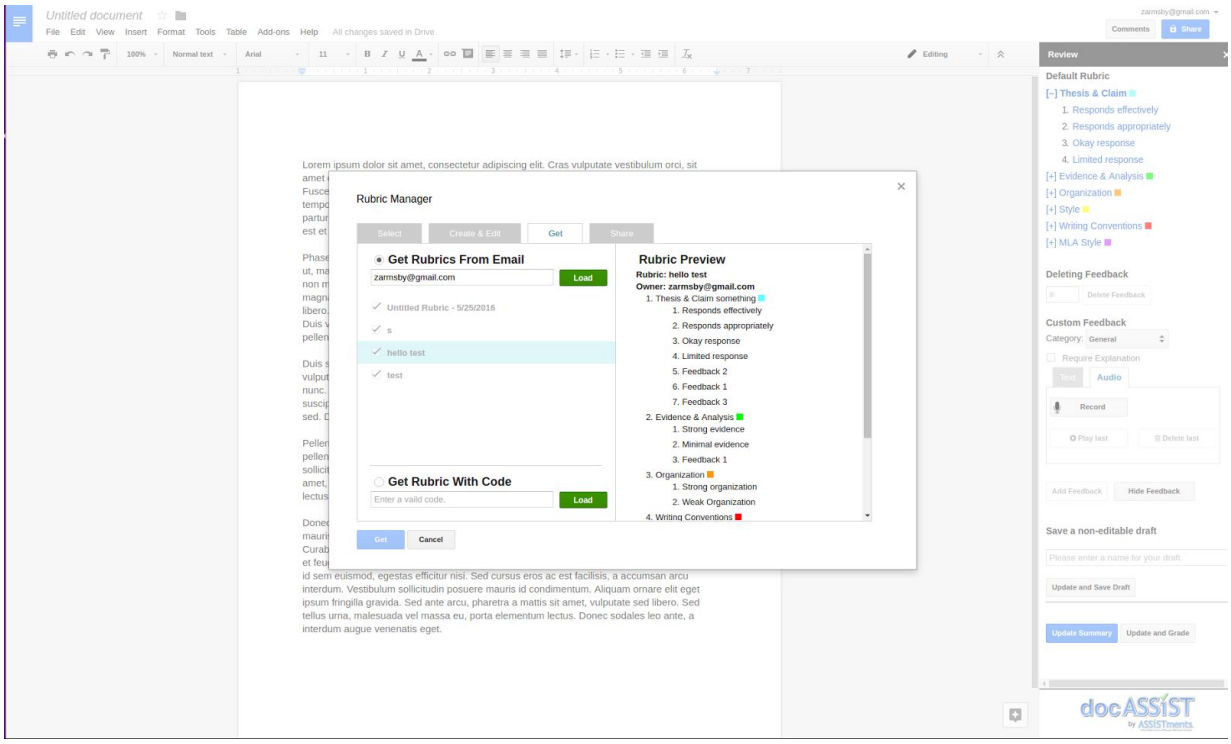
Continued Developing of docASSIST: A Google Doc Add-on



By Zachary Armsby

docASSIST by ASSISTments
A Free Public Service of Worcester Polytechnic Institute

Continued Developing of docASSIST: A Google Doc Add-on

An Interactive Qualifying Project Report Submitted to the Faculty of the

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the Degree of Bachelor of Science

By

_____

Zachary Armsby

Submitted on

Approved:

_____

Professor Neil T. Heffernan, Advisor

_____

Cristina Heffernan, Advisor

Table of Contents

List of Figures

Abstract

docASSIST is a tool created by WPI students as MQP (Major Qualifying Project) and IQP (Interdisciplinary Qualifying Project) projects with advisement and mentorship from the WPI assistments team. The purpose of docASSIST is to help teachers give feedback to students faster and more efficiently. Currently many teachers are overworked and take a long time to give feedback to students. docASSIST solves this problem by giving teachers tools to quickly evaluate assignments using rubrics and user defined feedback options for rubric categories. The purpose of this IQP was to continue developing docASSIST. Areas of development included the development workflow of the docASSIST project, the rubric manager, and streamlining many common processes to maintain and continue development.

Acknowledgments

I would like to acknowledge all of the previous developers of the docASSIST project, Nick McMahon, Sam La, Jean Marc Touma, Christian Roberts, and Zi Wang, for all of their work on docASSIST to get it to where it is today. Also to thank our advisors Christia and Neil Heffernan for continuing the docASSIST project and for all of their advice and guidance while developing docASSIST. Additionally I would like to thank the Assistments Lab and especially Andrew Burnett for their feedback and help in moving docASSIST forward. Also I would like to acknowledge Gianluca Tarquinio for working with me on docASSIST (worked during the same time but reporting separately). As well as Cory Tapply and Trevor Valcourt for continuing to work on docASSIST.

Authorship

This paper was written by Zachary Armsby using Jean Marc Touma and Christian Roberts paper as an example.

Introduction

docASSIST is not a new project, it has been worked on by at least five different developers and is over a year old. In that time the project was never set up to for real development. Changes were piled on top of each other until most of the code was barely understandable and a significant amount of redundant functions existed. This was the result of the development setup and ecosystem used by apps script projects and having developers focus only on select areas of the project and then leaving when their project was finished. This was the first part of docASSIST that I decided to improve by adding source control, refactoring the code, and defining new best practices for apps script development.

Following the introduction of a better development environment I focused on the development of the rubric manager and support for sharing rubrics. Rubrics are a central concept to docASSIST, allowing teachers to quickly grade based on predefined metrics and feedbacks. This necessitated the need for a better interface for creation, editing, and sharing of rubrics. In order to implement the rubric manager many changes had to be made to the existing ui prototype and the backend had to modified significantly to support the sharing of rubrics between users.

Development Workflow

Git Integration

When I began working on docASSIST I had experience with javascript in web development and java for class work, but I had no experience with google's apps script ecosystem or with server backends. It was very difficult to understand code written by previous developers in apps script until several concepts about apps script were understood. Apps script is very different from most languages, in that it is a variation of javascript completely managed by google that can only run on their infrastructure. Keeping with google's idea of everything on the cloud, all apps script projects exist only on the cloud and were not easily accessible in any other way. The editing, storage, testing, and running of an apps script project was completely dependent on google's cloud ecosystem. The apps script ecosystem is at the complete mercy of google and any new features that are desired need to be implemented by google. This also meant that several key development tools have been omitted from the apps script ecosystem because google has not implemented them yet/if they ever will.

One glaring problem was that there was no version control when I began working on docASSIST. All of the code for the project was saved in the cloud, a google doc with a bounded apps script project in a shared folder. Whenever anyone made any changes to the project it would update everyone's version of the code base after new code was saved. This meant that if two people were working on the code base at any one time and making changes everything would work as expected until one person saved their new changes to the code. At that point the other person would lose all of the work they had just done and be synced with the person who just saved, losing minutes to hours of work. Having everyone try to edit the code simultaneously was not effective and often led to loss of work, dropping productivity. Another flaw in this system was that without version control it was impossible to have multiple versions of the code simultaneously existing for the same project. This made developing new features in parallel almost impossible, with conflicting development processes running in tandem interfering with each other. There was only one version of the code and all development had to done in that one version. The absence of version control led to chaotic development.

After encountering these challenges initially I decided it was necessary to integrate git into the docASSIST project. Without it development could only move at a snail's pace and would constantly be held back artificially. First it was necessary to chose a source control system to integrate docASSIST with. Git with a GitHub remote hosted repository was chosen as the source control method because of the widespread use of git and GitHub for computer science projects and my familiarity with git. Additionally GitHub's hosting of free private repositories for students made it a more attractive choice for an academic project.

Once the git repository was set up it was necessary to associate it with a google apps script project on google drive. docASSIST is written using google apps script, google's cloud hosted version of javascript. There are no real differences in terms of the language, but there is more added functionality accessible through google services. Apps script is intimately tied to

google’s own core infrastructure and cannot be run locally. This is a problem for using a local repository and not having access to the google drive file system to use source control. To solve this it was necessary to use the google drive api to upload an apps script file to a user's google drive anytime a change was made. This was implemented using a third party tool called gapps which took care of interfacing with the google drive api. However it was tedious and annoying to have to upload the project to google drive every time a change was made. To solve this a task runner, Gulp, was used to simplify the process of uploading code to just one command. Both gapps and Gulp were written in and required nodejs to run. Due to the dependence on the nodejs ecosystem npm, the nodejs package manager, was used to manage the dependencies for this project and install all necessary software for new developers. To recap, at this point all development took place locally on the developer's machine in a git repository, not the google cloud, they have installed software with npm, and are uploading code when they make changes with gapps being called from a gulp task runner to simplify the process for the developer. They would then test their changes by running the uploaded code.

Refactor

After going through the code base and introducing version control I had seen most of the code written by previous developers. After seeing the code it was apparent that a refactor was necessary for development to continue in the future. Past development of the project was done completely in google’s cloud apps script environment without many common development tools like source control. This caused a weird development style to evolve for the docASSIST project to counter many of the shortcomings of the gapps development ecosystem.

A result of not being able to edit code simultaneously or have parallel development was that code was often repeated in many different locations. There were even exact copies of sections of code in the same files only a few lines away from each other. The addition of source control would prevent the code duplications in the future by allowing development to occur more seamlessly across multiple developers, but it was necessary to go through all of the code and remove any duplicated sections to prevent unpredictable errors when only one section would be changed. There was no easy way to detect duplicate sections of code across the entire code base so every file had to be checked manually for duplications and fixed. By going through manually it became apparent that there was a lot of duplicated functionality or redundant code that had been forgotten about after new methods were developed or development had moved to other areas.

Additionally the google apps script environment would not allow separate files for style (css), function (js/gs), and form(html) forcing previous developers to put completely different code functionalities inline in gs and js files. This caused files to become extremely long, confusing, and hard to read and understand. To fix this it was necessary to combine files by embedding script and style code inside the html code. This was achieved using script and style tags in html and by using a task runner, gulp, to compile the file together before uploading to google drive. After the development of the file embedding task it was then integrated with the upload task, that uploaded the code to google drive so it could be tested easily in google apps

script environment. This made it so all of the separate files would be automatically embedded in the main files on upload.

Another problem with the code base was that there was no consistent style from function to function or line to line. This made it difficult to read and modify existing code, slowing down development unnecessarily. To solve this problem manually would have been ineffective and cost too much time to be worth implementing. But while investing how to use task runners I came across code beautifiers, tasks that modify code structure to fit a desired code style. This would make the code style consistent, allow for future reformations , and take minimal time to implement since the beautifiers were already written and the project already had a task running system in place.

Rubric Manager

A central component to docASSIST is the rubric manager. The docASSIST application is based on the idea of rubrics and feedback based on rubrics to speed up the process of giving feedback to students. In order to give fast, effective, and consistent feedback rubrics need to be well thought out and well managed by the user in order to get the full effect of docASSIST. In order to better facilitate the process of creating and managing rubrics the rubric manager was created to provide a strong interface for users to understand and interact with rubrics. When I began work on the rubric manager it only created and edited rubrics with a clunky user interface that needed to be reworked. During my time working on the project a new user interface and rubric sharing was added to the rubric manager.

Back End

In addition to the front end of docASSIST, the program handling interaction with the user, a backend program is needed to store user information in a database to persist information through multiple user interactions with the docASSIST app. When I began work on docASSIST only basic rubric information was stored by the backend in the database, such as the rubric id and user email. This only allowed for the retrieval of rubrics by specific users and limited the ability to expand the functionality of the rubric manager to include more advanced features such as sharing rubrics. Rubric sharing was a high priority item to add to docASSIST, two methods of sharing the rubrics were requested; email sharing and code sharing. Email sharing would work by having a person share the rubric by giving people their email address to find the rubrics they had made. While code sharing would share a rubric using a generated code that a user could give to other users to get the rubric. Another requirement was that it would be necessary to have the user control which of their rubrics would be made public or could even be accessible from any form of sharing. In addition it was necessary that users could change the sharing settings of their rubrics at any time to give them control over who could see and use their rubrics at any given point in time.

In order to implement shared rubrics it would be necessary to store more information about the user. To keep track of the rubrics that a user gotten/added through sharing it would be necessary to store a new user object in addition to rubric objects. This would allow for storage of what rubrics that user owned, rubrics that had been email shared, and which rubrics had been code shared. This user object was then associated with a user based on their email account, information that the front end of docASSIST could provide from google services. Then it was necessary to implement a way to determine if a rubric was shareable or even accessible at any time by any of the sharing methods. In order to accomplish this the rubric object had to be modified to store permissions about how the rubric could be used. The rubric object being stored in the database was modified to keep track if email sharing and if code sharing was enabled. In addition to changing the information stored and the objects used to store the

information, the rest api calls to the backend had to be changed to service new requests for shared rubric information and to store the new data.

The process of migrating to the new data structure on the back end provided several challenges. The first challenge was modifying the backend in a way service to the user would not be interrupted in any significant way. This constraint proved to be very difficult. Prior to this development there had been no other significant modifications of the backend and there was no well defined way from previous developers to roll out an update smoothly. After substantial edits to the backend it was not obvious that the modifications would work and could cause all requests for rubrics to fail, resulting in a complete halt to user activity until we could patch the problem. This behavior was not acceptable for a production application. To solve this problem a development version of the backend was created in a separate backend. This was allowed us to test our changes and work out time intensive bugs in the the development environment without affecting real users.

While a separate development environment allowed us to test the new backend separately it did not address how to migrate all of the user data to the new storage scheme. In order migrate the data we created a function that would only be run once to convert all of the user data to the new scheme. After it was used it was then removed from the system to prevent accidental usage (repeated use could easily exceed the data quota). This converted all of the previous user data to the new format on our test environment. While this worked in the development environment it would not work for in the production environment since the backend would have to be updated in order for the conversion function to be present and to not have new data be accessible without another update after the conversion. Additionally in order to reach the new backend the front had to change how it was requesting data, requiring a concurrent front end update. With all of the problems of updating the production environment live it was decided to have both the production and development environment running and then transfer all of the current data from the production environment to the development environment, then update the front end to use the development environment's backend. At that point as users received the front end update they would begin using the development environment backend and stop using the production environment. After all of the users received the update the production environment became the new development environment and the old development environment became the new production environment, effectively swapping the environments. The only side effect of this operation was that from the time of the transfer of user data to the development environment to when they got the update, any new data or modifications to rubric data would be lost once the recieved the update. To minimize the impact the update it occurred during non-peak hours and only took a few minutes.

Rubric Category UI

When I began work on docASSIST the user interface or ui for the rubric manager was not well defined and needed to be updated. While I was working on the backend Gianluca had began work on updating the ui. After finishing development on the backend I began to assist Gianluca with the rubric manager ui. While Gianluca worked on adding functionality to the rubric

manager's interface I rewrote the underlying structure of the page by rewriting the html and associated styles. Originally most of the elements had been added without any well defined structure or organization to assist with the layout of the elements on the page. Excessive amounts of code dealing with style had been used to make structure. This made the original layout difficult to modify, did not handle page sizing changes well, and resulted in different user experiences on different tabs in the rubric manager. This was fixed by creating a uniform structure and style that was applied to all of the tabs in rubric manager, simplifying and improving the user experience.

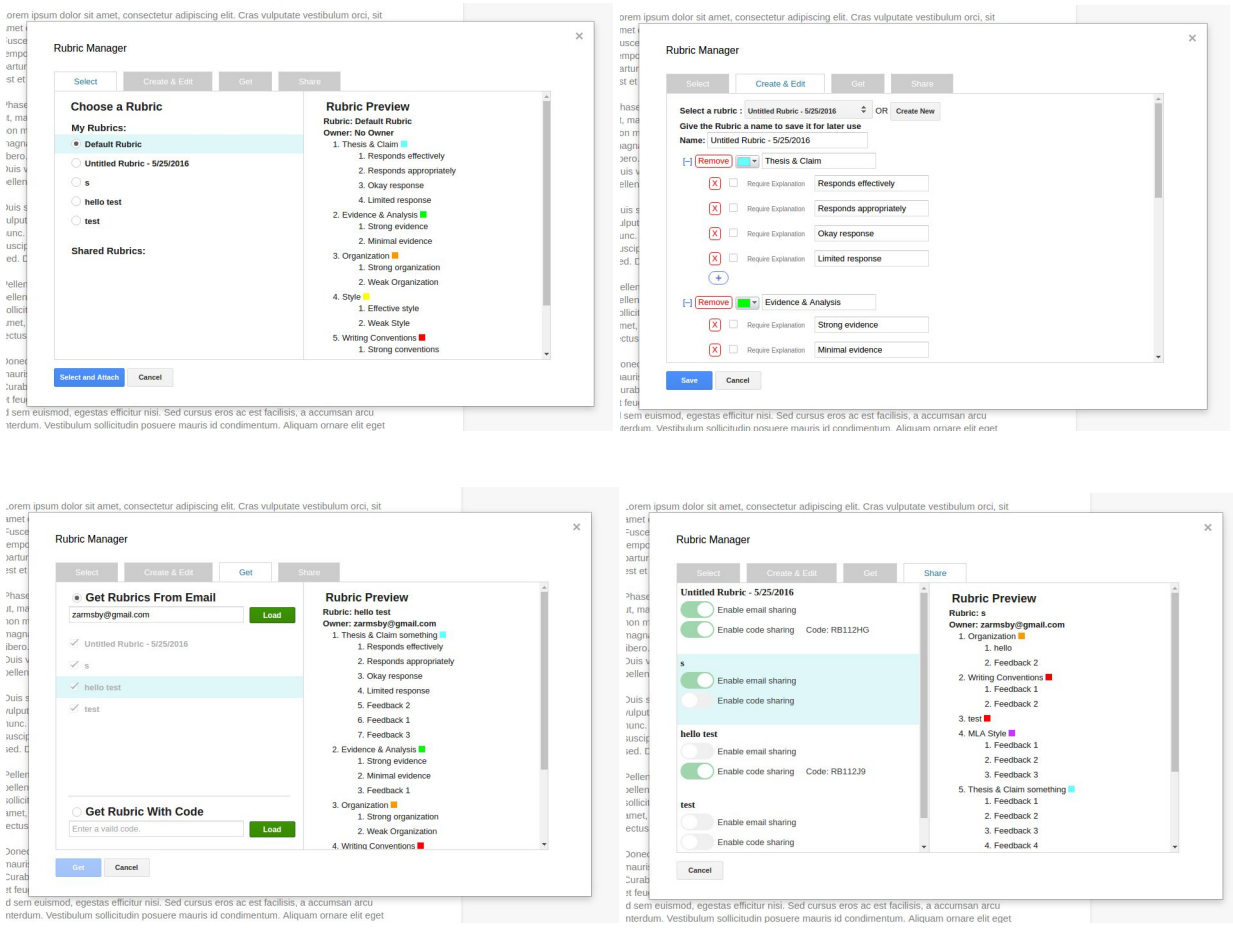


Figure 1 Rubric manager panels

In addition to improvements in the regular user interface a request was made to have drag and drop of rubric categories and feedback for creating and editing rubrics. This was the final area of development during my IQP and is not implemented in docASSIST yet. It still needs more development and testing on more platforms before it is ready for deployment. However significant progress was made in the development of the feature and a mostly working model was built. Currently it has dragging and dropping functionality and the ability to to add and delete new items in. Also it has the transitions necessary to move between different states with

smooth animation started by app and handled by the browser to increase performance. All that remains to be implemented are basic controls and more rigorous testing on more browsers.

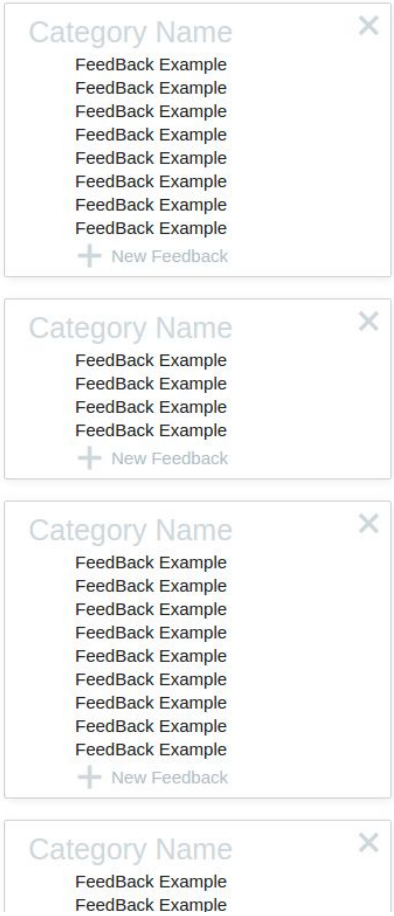


Figure 2 Experimental drag and drop interface for rubric manager

Updating and Testing Systems

Installing docASSIST Apps Script Tools

The docASSIST front end application repository is on Github and has extensive instructions on installing the tools necessary for development, a link is provided in the appendix to the repository.

Tester Application

When creating new features it is important to test them rigorously before releasing them to all of the users. When developers personally test features they are not able to find all of the bugs in the system because they have very fixed mindset on how they think everything will be used. So it is important to have other, non-developers use the application first before releasing to all users. In order to do that with the google ecosystem it it necessary to create an additional private application that runs the new code with the new features. This new private application is then shared with trusted testers. In our case it has been shared with our advisors and other users that want to give feedback on new features.

Steps to deploy a tester application:

1. Go to the google webstore by clicking on apps, then web store

Figure 3 Google web store navigation

2. Click on the gear in the right corner and select developer dashboard

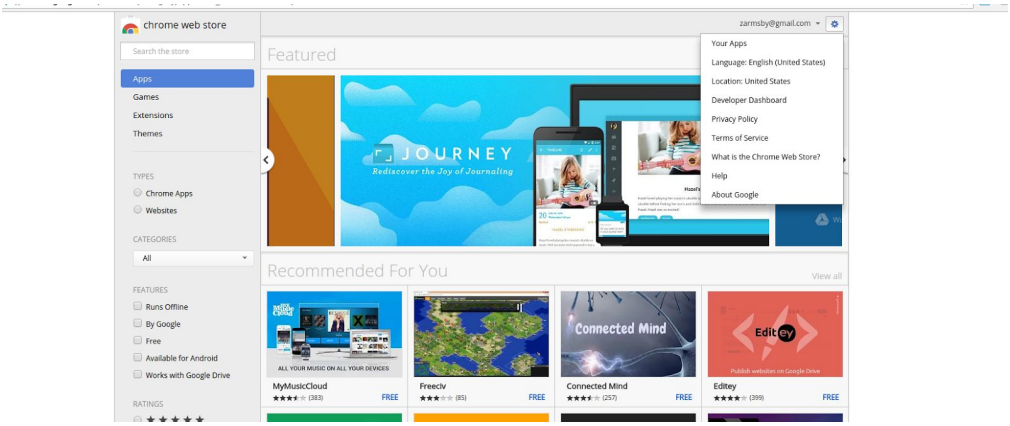


Figure 4 Developer dashboard navigation

3. Setup your developer profile (set name and pay one time fee to allow publishing) and add testers by typing in email accounts

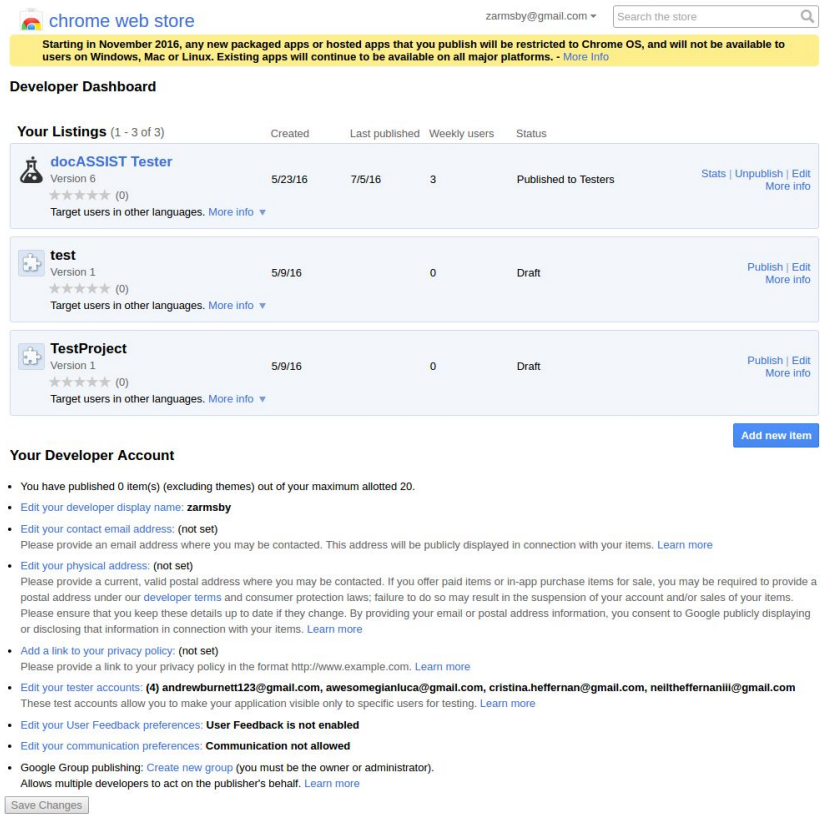


Figure 5 Developer dashboard

4. From the test apps script project that you have been uploading to every time you make a change (or another apps script if you would like, you just need the desired code in an apps script) go to publish > deploy as web add-on

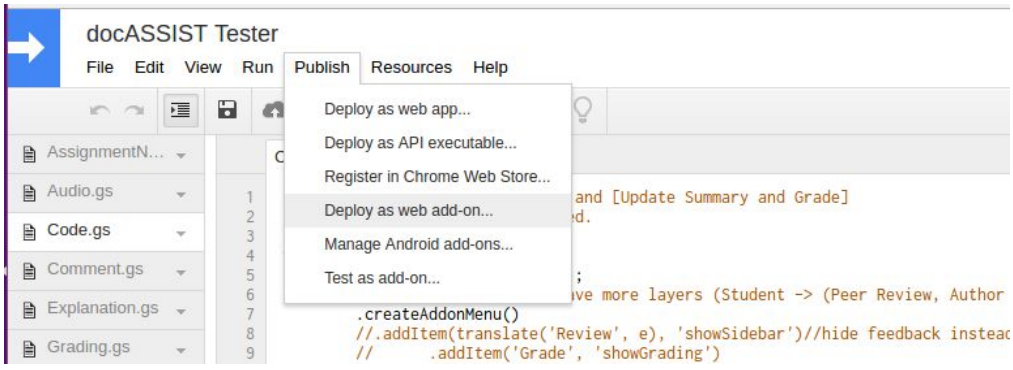


Figure 6 Apps script editor, publish tab

5. Create a webstore draft and do not select publish in google apps marketplace

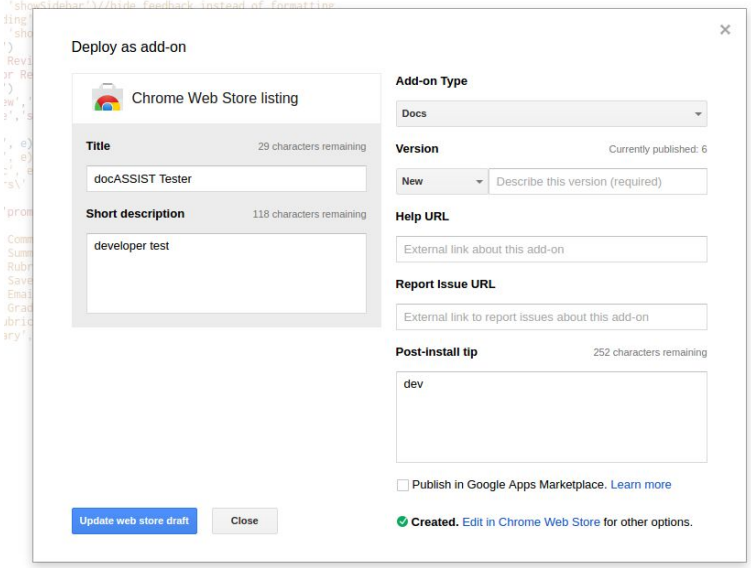


Figure 7 Deploy as web add-on dialog

6. Then back in the developer dashboard you will see a draft of the application as pending, to deploy to only testers you need to edit the draft

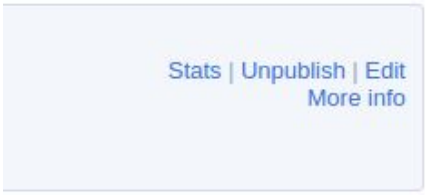


Figure 8 Developer dashboard, web store draft controls

7. Once editing the draft at the end of the dialog set the visibility option to private so only your list of testers can use it and then publish changes

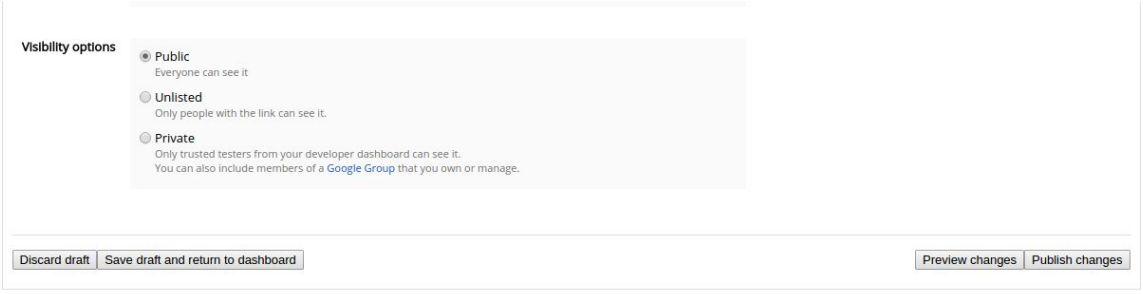


Figure 9 Webstore draft edit page, visibility options displayed

8. Then send links to the install page for your users and explain what features you are testing.

Backend Management

docASSIST uses the google cloud platform to host the backend and database. To edit the backend it is necessary to use the cloud platform command line utility or use the web based google cloud console. To edit the source code and deploy the backend it is recommended to setup IDE integration with the google cloud platform for deploying to specific projects. The IDE integration process is covered in depth in google's documentation.

The important areas most relevant for docASSIST are the following:

1. The main project dashboard to view key statistics

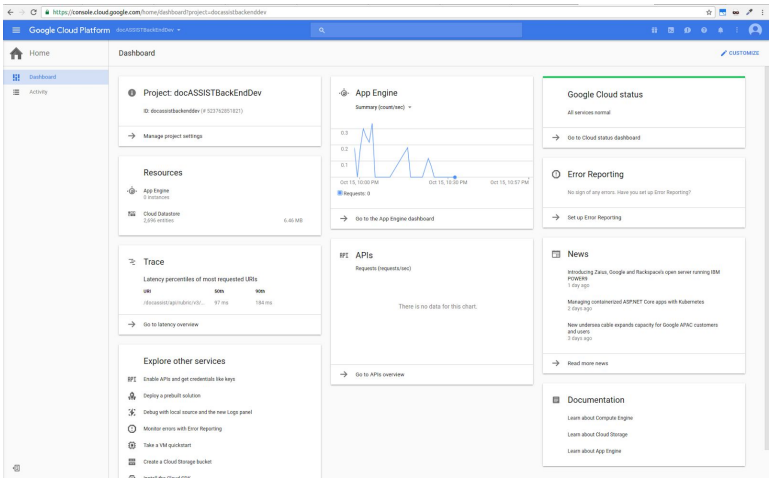


Figure 10 Google cloud console dashboard

2. The navigation panel to reach other important areas

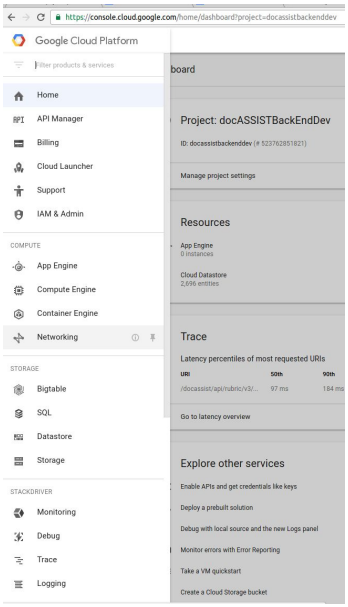


Figure 11 Navigation pane

3. The app engine dashboard to monitor the application

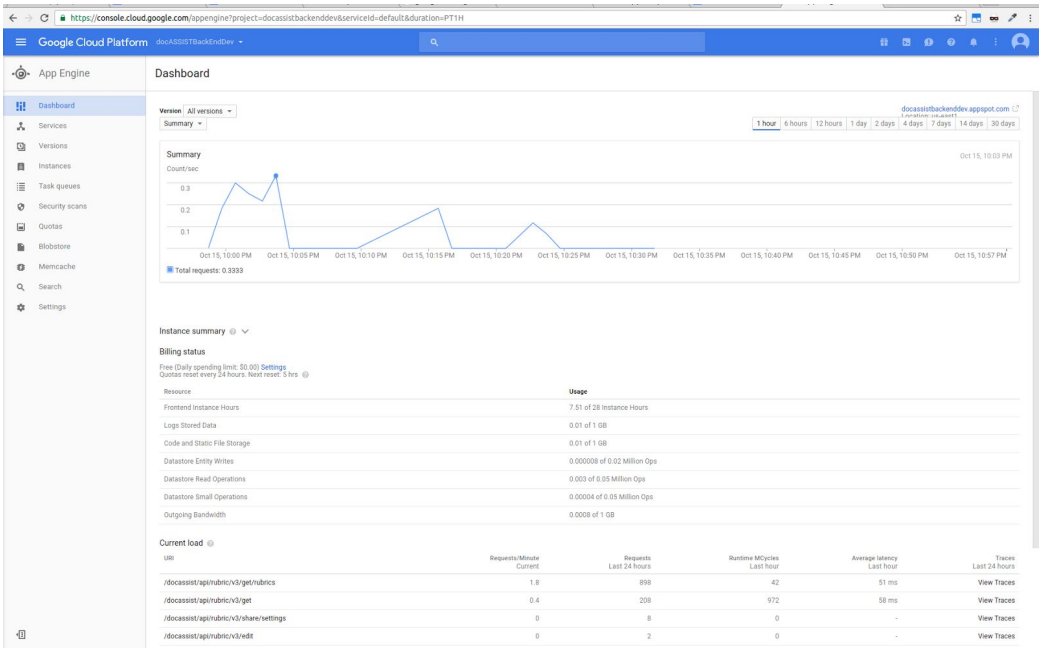


Figure 12 Google app engine dashboard

4. The app engine version and traffic routing manager to manage what version of the backend is currently being served

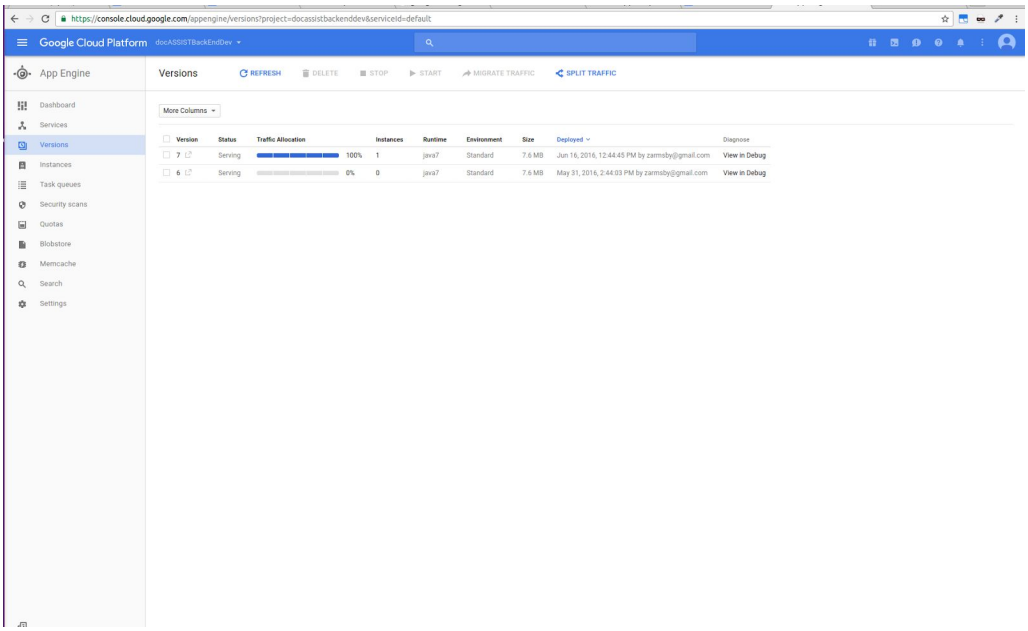


Figure 13 Version and traffic routing control page

5. The app engine quota page to monitor how much data the project is using (this is more important as more users join or complete database updates are necessary)

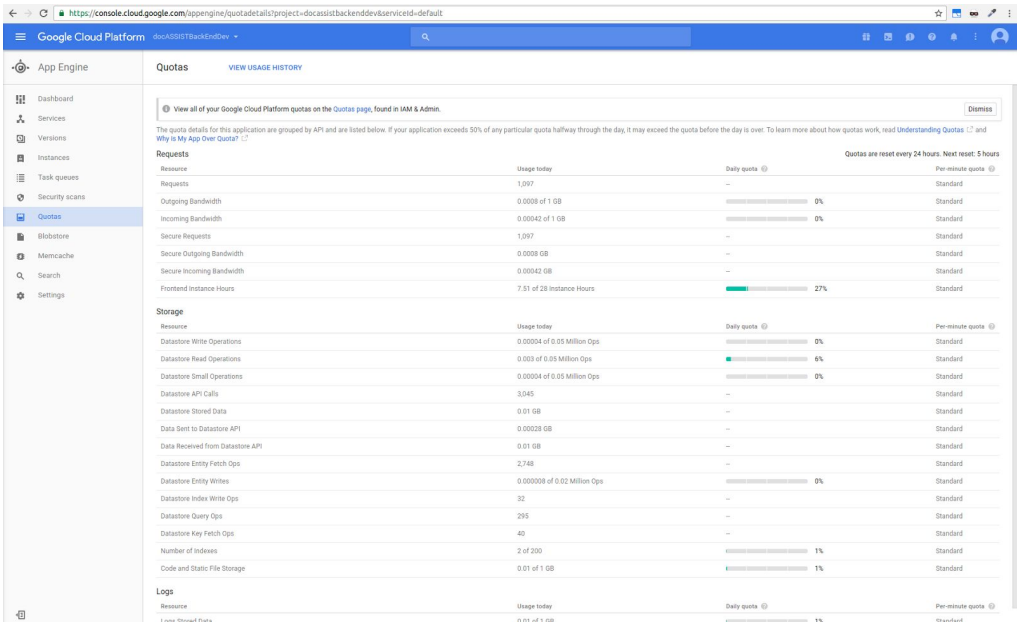


Figure 14 Google app engine data quota page

6. The datastore viewer to view database entries

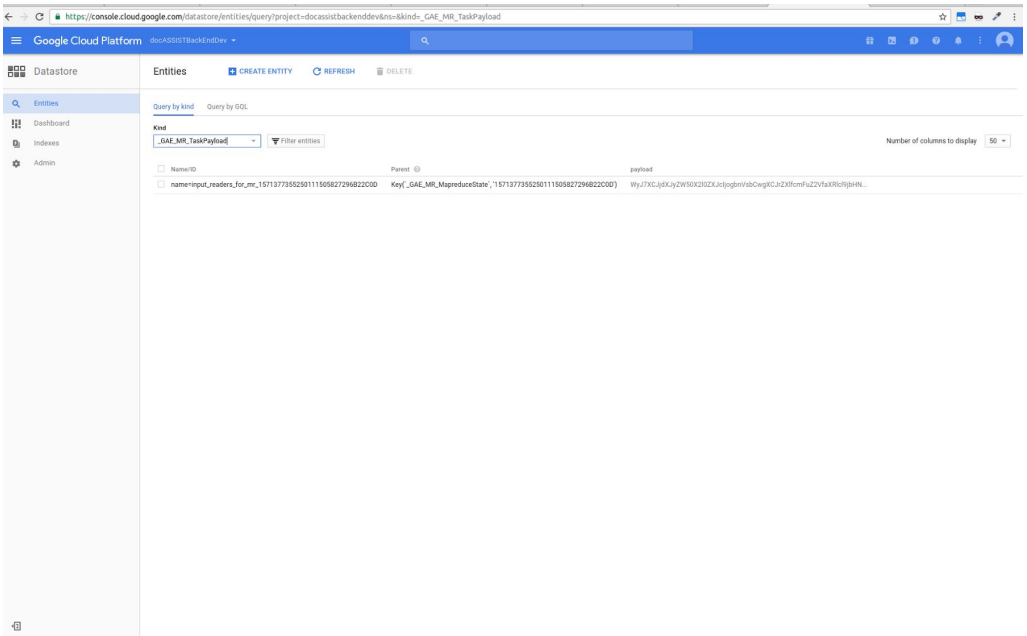


Figure 15 Datastore viewer page

7. The log viewer to help debug failing requests to the backend and see what errors are occurring

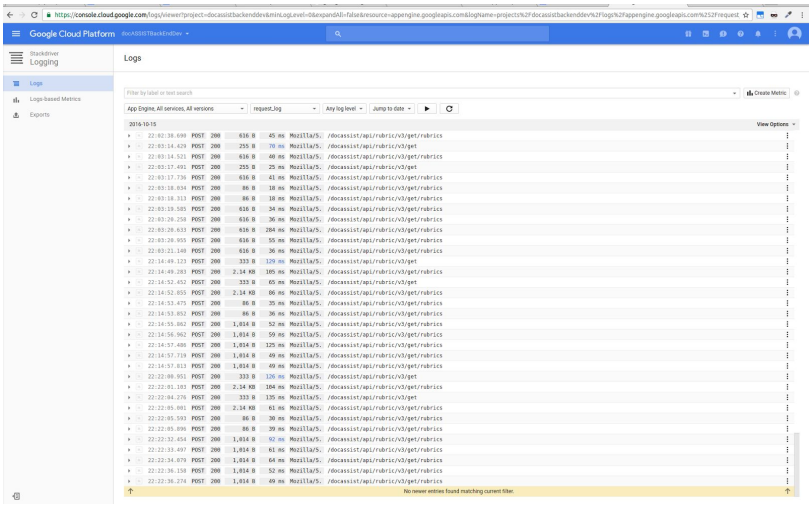


Figure 16 Log viewer

Release Process

As changes are made to docASSIST the application needs to be updated for all users. The steps for updating the application are described below.

1. Finalize code and check for bugs
2. Use a tester application (described above/previously) to have a few other users check for bugs
3. Log into the assistments teacher profile and copy the code (textual copy for the script editor for every file) from your tester project to the official docASSIST application (the “Add-On Bounded Doc” file). This step is unavoidable due to the way google has structured the docASSIST ecosystem if we do not want to have all of the users reinstall the application.
4. Create a new version under file > manage versions to serve as a fallback point
5. Under publish > publish as a web add-on update the web store draft as a new version and do not check publish in google apps market place (shown in Figure 6-7)
6. Go to the developers dashboard (shown in Figures 3-5)
7. Edit the draft of the application (shown in Figure 8)
8. Publish changes (make sure the visibility is set to public, shown in Figure 9

User Interaction School Visit

After releasing the rubric manager we wanted to get feedback from teachers on what they thought about docASSIST, to learn what features the would like to see in future development and to see what parts of the application the like or dislike. Thanks to Andrew Burnett we visited Shrewsbury high school to demo docASSIST to their english department. While we were there Andrew demoed docASSIST while Gianluca and I observed the teachers interacting with the app and answered any questions the teachers had. After going through the demo with the teachers we saw the problems they ran into while trying to use the application and were surprised by what they seem to prioritize for new features. One of the top requests was for an indicator for when certain tasks had completed in the app, so that the teacher could know that an event had occurred and finished (this has since been implemented and updated for all users). Another request was for more support for in depth statistical tools for grades and for tools to help teachers perform their own analysis. The most surprising request for us was for mobile support so students could use the school provided tablets with docASSIST (this is unfortunately not in our control, google would need to allow this). Other requests that we anticipated going into the meeting included more permissions and a stronger idea of roles for different people of the same document and for an easy to use versioning system to track changes and monitor feedback fulfillment.

Conclusion and Future Development

After working on docASSIST I am happy with the progress that was made. During my time on docASSIST the development work flow was substantially improved and streamlined by integrating source control into the project and refactoring the code base. The rubric manager was implemented with a clean user interface and fully supported shared rubrics. Finally new methods for testing and updating the front and back end of the application were developed to prevent bugs from occurring and to increase quality control.

Then we received feedback from teachers on how to improve docASSIST and what new features they would like to see in the future. From their suggestions I urge future developers to implement new grading and statistical analysis tools, more concrete and visible roles for users on documents, and to integrate versioning and feedback tracking into the application. But I am worried that future extensions of docASSIST may prove to be very difficult. Many of the desired additions to the app require a fundamental change to the way that docASSIST would operate internally and externally. At this point so many new concepts need to be introduced that I would recommend a rewriting of the docASSIST project if it is to progress past it's current state. With a version two of docASSIST many of the pitfalls of the first version could be avoided and core ideas about how docASSIST functions could be changed to better match its new requirements. Additionally all of the work in the first version of docASSIST could be ported to a new version of the project. There is still a lot of work to do on docASSIST and I am confident that it will continue to grow.

Appendix

Docassist Website

https://sites.google.com/site/assistmentsfeedbacktool/

Github Repositories

https://github.com/docASSIST will be the future location of the docASSIST repositories

Docassist Frontend

https://github.com/zarmsby/docASSISTAppsScriptProject (must be member to view to request access email zarmsby@gmail.com)

Docassist Backend

https://github.com/nmcmahon1215/docASSIST (must be a member to view, my work is in the zdev branch)