# Gbiv Project Plan

## Division of Labor

| Team Member | Primary Responsibilities | Secondary Responsibilities |
|---|---|---|
| Sam Heilenbach | Testing and Integration | Image Processing |
| Cheyanne Kester | Frontend Development | Frontend Styling, System Design |
| Ian McConachie | Documentation | Web Interface Styling, Image Processing |
| Scott Wallace | System Design | Jira management, Backend Development |
| Austin Warren | Image Processing | User Documentation |
| Catherine Raj | UI Design | Frontend Development |

## Project Timeline

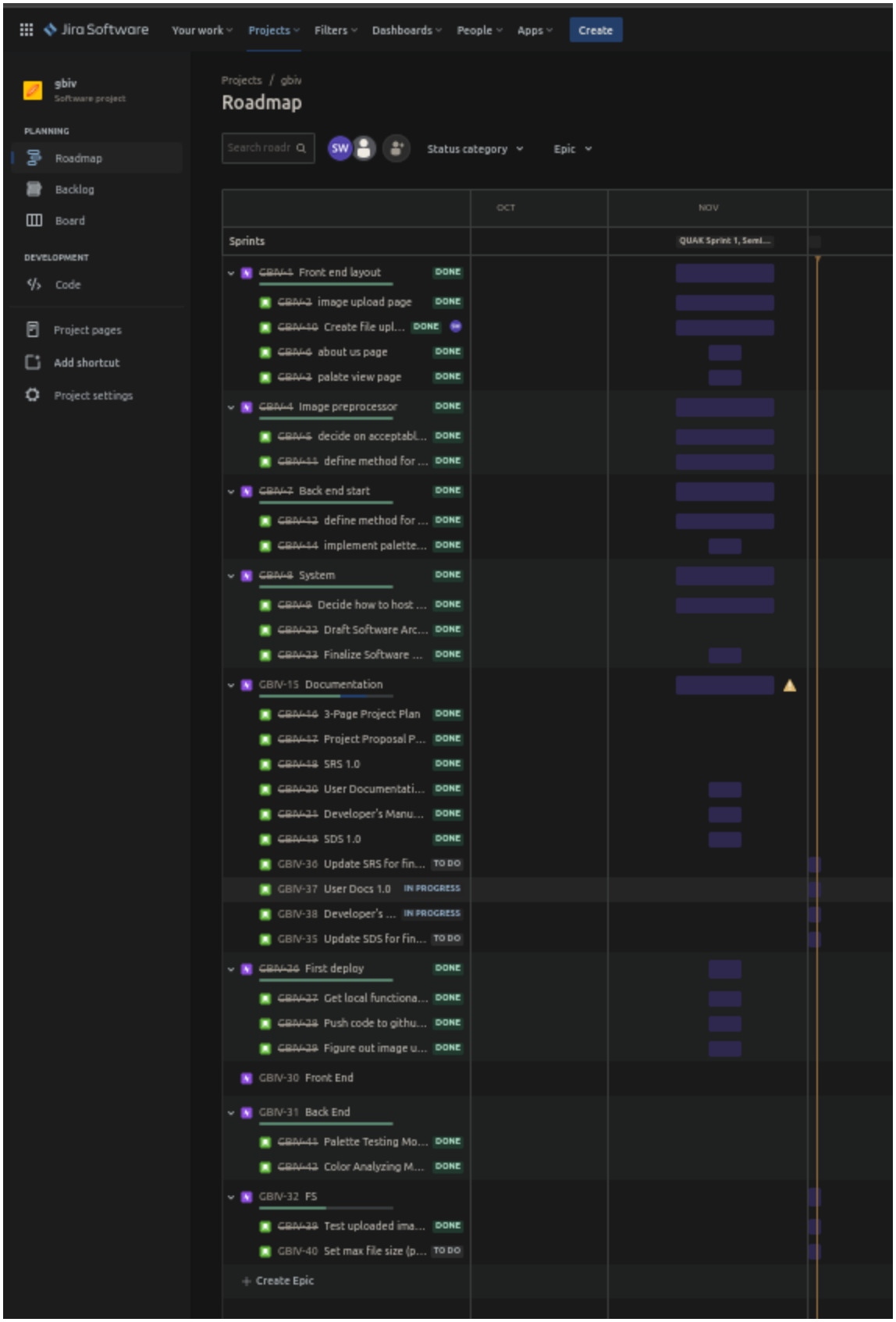
| Week | Task Outline | Issues encountered |
|---|---|---|
| 1 | Implement a basic upload page using flask and jinja templating. Begin to think about how to host apps.<br><br>Begin development of palette generation module.<br><br>Start SRS, SDS, write project proposal. | Where to store temporary files on the web? Firebase deployment is overcomplicated, other options?<br>Learning about the color systems.<br><br>Getting our ideas straight for what the application will offer. |
| 2 | Create additional pages (about us, color theory, and sample palates) and consolidate with a nav bar into one website. Solved temporary storage and | How to sync code changes? Broken directory links and imports. |

| | | |
|---|---|---|
| | firebase issues at the same time using python anywhere which allows an intuitive WSGI web hosting framework with a browser accessible and unix cabaple system. Files stored on remote machines can be accessed via the website's UI or via a bash terminal and came prepackaged with a virtual environment to contain our project dependencies.<br><br>Develop more color generation functionality in the palette generation module.<br><br>Finish SDS and SRS. | |
| 3 | Integrated our git repository with our remote machine and created a remote branch to consolidate necessary local changes with the code we were developing. Fixed broken links and implemented error handling for unusable input.<br><br>Develop a testing module for the palette generation module.<br><br>Start user docs and developer’s manual. | A user1 can hypothetically delete another user2's image prematurely given the way we implemented a routine to clear our uploads folder.<br><br>Reacquainting ourselves with pytest. |
| 4 | Aforementioned routine has been set to run infrequently and is triggered only when users are not operating the upload page.<br>All pages get their finishing touches, had to redo the About Us page<br><br>Finalize the palette testing module. Determine the most pertinent future testing developments for | Getting user image to output was a challenge. General formatting final touches for the palettes. |

| | | |
|---|---|---|
| | documentation.<br><br>Finish user docs and developer's manual. | |

## Task Management

To manage the tasks involved in completing this project, we used Jira as a team management system. We used Jira to coordinate most of our planning along with git to synchronize our code changes. Our Jira workflow most closely resembled that of SCRUM.


Jira Software
Your work
Projects
Filters
Dashboards
People
Apps
Create
gbiv
Software project
PLANNING
Roadmap
Backlog
Board
DEVELOPMENT
Code
Project pages
Add shortcut
Project settings
Projects / gbiv
Roadmap
Search roadr
SW
Status category
Epic
OCT
NOV
Sprints
QUAK Sprint 1, Semi...
CBAV-1 Front end layout DONE
CBAV-2 image upload page DONE
CBAV-10 Create file upl... DONE
CBAV-6 about us page DONE
CBAV-3 palate view page DONE
CBAV-4 Image preprocessor DONE
CBAV-5 decide on acceptabl... DONE
CBAV-11 define method for ... DONE
CBAV-7 Back end start DONE
CBAV-12 define method for ... DONE
CBAV-14 implement palette... DONE
CBAV-8 System DONE
CBAV-9 Decide how to host ... DONE
CBAV-32 Draft Software Arc... DONE
CBAV-33 Finalize Software ... DONE
GBIV-15 Documentation
CBAV-16 3-Page Project Plan DONE
CBAV-17 Project Proposal P... DONE
CBAV-18 SRS 1.0 DONE
CBAV-20 User Documentati... DONE
CBAV-21 Developer's Manu... DONE
CBAV-19 SDS 1.0 DONE
GBIV-36 Update SRS for fin... TO DO
GBIV-37 User Docs 1.0 IN PROGRESS
GBIV-38 Developer's ... IN PROGRESS
GBIV-35 Update SDS for fin... TO DO
CBAV-26 First deploy DONE
CBAV-27 Get local functiona... DONE
CBAV-28 Push code to githu... DONE
CBAV-29 Figure out image u... DONE
GBIV-30 Front End
GBIV-31 Back End
CBAV-41 Palette Testing Mo... DONE
CBAV-42 Color Analyzing M... DONE
GBIV-32 FS
CBAV-39 Test uploaded ima... DONE
GBIV-40 Set max file size (p... TO DO
Create Epic

# Testing and Integration

It is hard to overstate the importance of incorporating testing into the implementation process. In many junior development efforts, it is unfortunately common to rely mostly on manual testing to ensure the quality and functionality of a system. The limitations of this are evident; it is unlikely that a development team can maintain the pace of manual necessary for a rigorous analysis of the system's robustness.

Given the time constraints presented by this project in conjunction with our development team's limited knowledge of automated testing techniques, it was not feasible to develop an exhaustive testing suite of the site's functionality. That said, it was still crucial for overall confidence in the system to ensure we tested that the functionalities provided to us by the python libraries we utilized (colorthief & colorsys) were working as expected in the context of our system. To do this and test some of the primary utilities of the palette_generation python module that drives the logic of our palette selection, we developed a test_palettes module that employed the pytest library to create some automated tests (or sanity checks) to ensure that the system functions as expected.

In terms of testing the python libraries that we employed for color extraction (colorthief) and switching between color representation systems (e.g., hex, rgb, hsl) (colorsys), we noticed early on in development that there was a minor bug in the get_color function in the colorthief library that leads to it extracting a slightly altered hex/rgb/hsl color than what is the modal color in the image provided to it. Despite this, our site still provides the correct color-theory-derived palettes for a hue that is indistinguishable from the exact modal color of the picture.

Integration of the system followed that of a prototyping mentality, so each iteration that was pushed to our main development repository was considered working and valid code. Given our emphasis on minimalist functionality, the user basically has one direct input with the website other than interacting with the nav bar which are links to static html code. Proper functionality was ensured by restricting file types to .png, .jpg, and .jpeg. This is not checked in the html document and is instead checked by our backend using the python import `from werkzeug.utils import secure_filename`.  The functionality of this library is somewhat beyond the scope of our current understanding but its general use is to ensure that invalid or malicious files cannot sneak past our backend by having seemingly valid filenames with actual invalid file/filename content. In that case, the basic operation of our website was proof enough of its functionality. That being said, we did encounter an issue where a user could hypothetically delete other users' upload during the upload process. This was solved somewhat trivially and is mentioned in our project timeline.