Gbiv System Design Specification

Release 1.2

Dux D-zine

Dec 01, 2022

TABLE OF CONTENTS

CHAPTER

ONE

SYSTEM OVERVIEW

Gbiv is a web application with the goal of making color theory more accessible to the world and inspiring the designer in all of us. Its primary use case is that users can upload images and Gbiv will extract the dominant color and use it to suggest related colors and palettes. In addition to this, Gbiv shows users example palettes for design inspiration.

Like most web applications, our system is divided into frontend and backend. On the frontend we have a “frontend manager” module which employs the Flask framework and python code to build the general structure for our user-facing website. The site itself utilizes HTML, CSS, and Jinja for styling and extra functionality in terms of user interface. We use the common page system and employ 4 pages: a main page for users to upload images, a page to see example palettes, an informative page about color theory, and an about page describing the project.

On the backend we have a similar “manager” module which also uses Flask and ties together the front end framework with scripts and a database in the backend. Beyond the manager we have the color analyzer script (also written in python) which handles all the color extraction and related color calculation.

These two sections of the system are tied together using the Flask framework as described above. Gbiv is hosted online using pythonanywhere and the source code is maintained through GitHub’s version control system. Documentation for Gbiv was created using reStructuredText, Sphinx documentation generator, and MS Visio. For team management, we used Jira with Google Drive supporting shared project plan documents. For more information there are several documents describing Gbiv: the SRS, the SDS (this document), the user documentation, and the developer’s manual.

CHAPTER

TWO

SOFTWARE ARCHITECTURE

The divide between frontend and backend was a guiding force in designing our software architecture. On one hand, it was important to maintain modularity within the two sides for ease of development. On the other, it was key that the two sides communicate fluidly so that the entire system could function properly. This led us to creating a "manager" module for both sides and using Flask as our framework on both sides.

The diagram below (Fig. 2.1) shows our architecture visually. Note that there is a third section of the application shown which is dubbed the "User Space." This is not part of our implementation, but is an essential component of our system because Gbiv is nothing without its users.

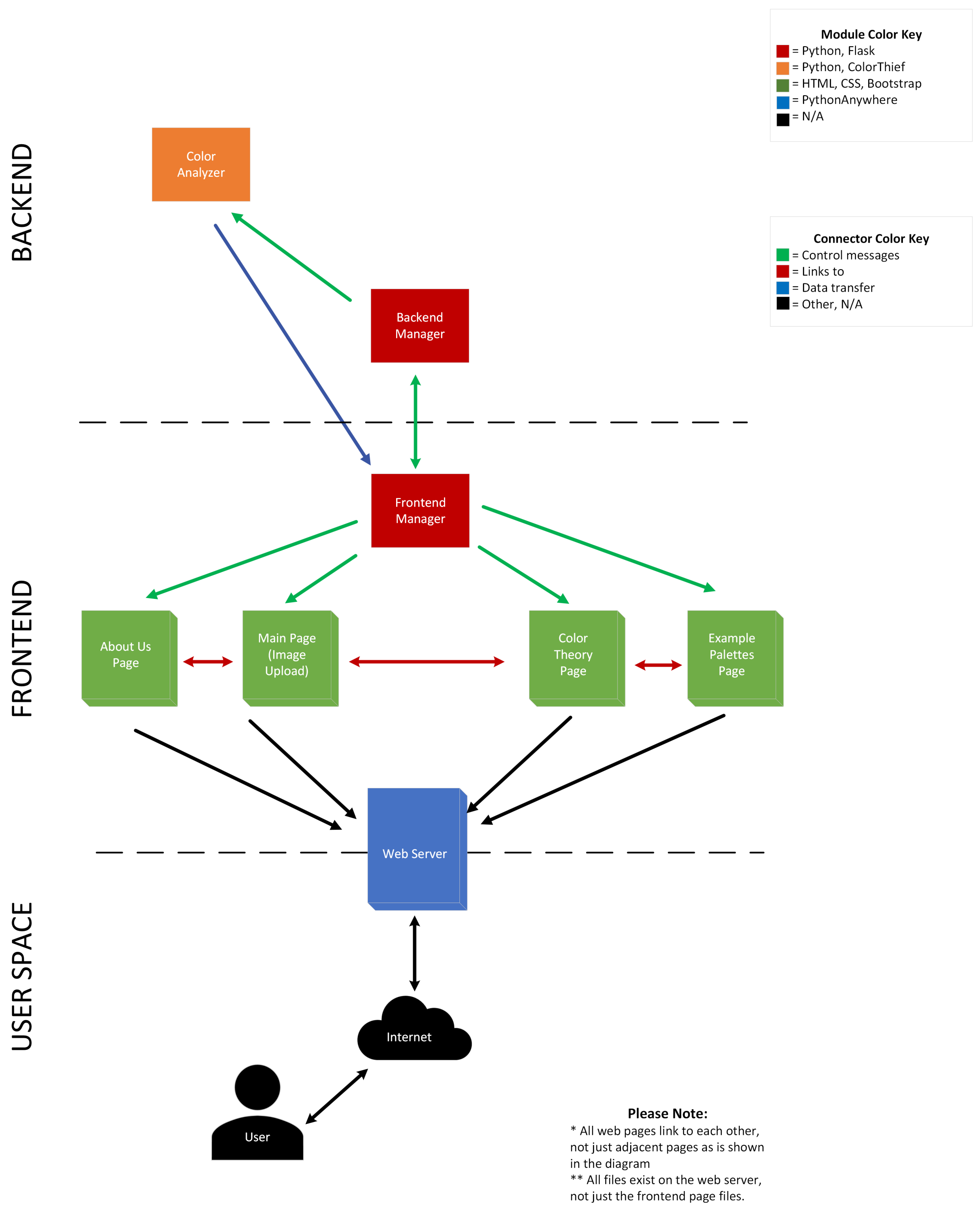

Fig. 2.1: Gbiv Software Architecture

The model above shows both components of the system and their interactions. We can further elaborate our architecture by focusing on each of these dimensions in particular. First, we can examine the individual components using the following table:

Table 2.1: Software Architecture Modules and Sub-Modules

Module/Sub-Module	**Category**	**Functionality**
Color Analyzer	Backend	Extracts dominant color from images, finds related colors, generates relevant palettes.
Backend Manager	Backend	Communicates with frontend and sends control messages to backend modules.
Frontend Manager	Frontend	Communicates with backend and sends control messages to frontend modules.
About Us Page	Frontend	Tells users about the team and the project.
Main Page (Image Upload)	Frontend	Allows users to upload image files and view dominant color, related colors, and relevant palettes.
Color Theory Page	Frontend	Gives users more information on the basics of color theory so that they understand what Gbiv is providing.
Popular Palettes Page	Frontend	Displays popular palettes in the Gbiv database to give users ideas and inspiration for their design projects.
Web Server	Frontend / User Space	Bridges the gap between the application and the user space. Hosts Gbiv files and provides infrastructure for the application to be accessed via web browsers.
Internet	User Space	The network infrastructure that allows users to access the application.
User	User Space	Anyone and everyone who has an interest in design and/or color theory.

Looking at each component in isolation is one way of viewing a system, but equally important to the application is the interactions between these components. The software architecture design shows all communication between modules, but we can highlight a few of the key interactions to better understand how Gbiv functions.

First, we can look at the transferring of control messages between the frontend and backend manager modules. These messages are formatted to comply with Flask's interface–which is the framework used in both manager modules. Having consistent implementation and technology in these "bridge" modules allows communication without complicated translation or additional frameworks.

Another key interaction we can profile is the web server's interface with both the front end and user space. Web hosting is what allows us to easily provide the functionality of Gbiv to end-users. We decided to use pythonanywhere for our web hosting because it allowed us to outsource server side logic and networking while maintaining the unique design of our web page. Web hosting is a great way to reach users because it provides an interface that is easily accessible and familiar to the majority of target users.

CHAPTER

THREE

SOFTWARE MODULES

Below are in depth descriptions of the design of each individual software module. It should be noted that not every component of the software architecture (see Fig. 2.1) is included in this list because not every component is part of the system we implemented. In particular, we do not describe users (for more information on users see the SRS), the Internet, or our web server (for which we used an OOTBS/COTS option).

3.1 Frontend Manager

The purpose of the frontend manager module is to help bind together separate components of the system. In particular, it is one half of the duo that is responsible for bridging the front and the back end (for more information on the other component of that connection see Section 3.3). Because the frontend manager acts primarily as a middle ground for passing information and control messages, it does not have much internal structure. However, we can still represent this module in a static fashion using the diagram below:

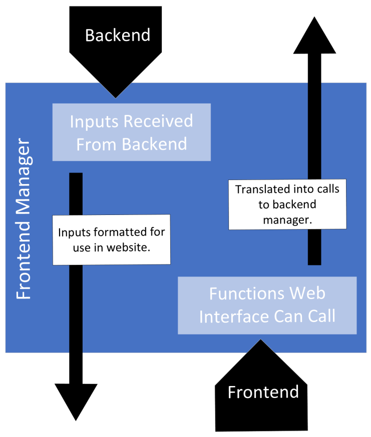
Backend
Frontend Manager
Inputs Received From Backend
Inputs formatted for use in website.
Translated into calls to backend manager.
Functions Web Interface Can Call
Frontend

Fig. 3.1: Frontend Manager Static Model

The frontend manager acts as a sort of "ambassador" between the frontend and the backend which means it accepts inputs and outputs from both sides. On the backend side, this module gives outputs to the backend manager and receives inputs from the color analyzer. The outputs that are given to the backend manager are control messages that indicate what the user is "asking for," which results in the proper data being generated. The inputs from the backend are different types of data intended to be delivered to end-users. Because this module is implemented using the same framework and technologies as the backend, the data can be transferred directly from where it is generated to the frontend manager.

In terms of interfacing with the frontend, we can again divide the interactions into inputs and outputs. Inputs from the frontend originate from user interaction which is translated into function calls in the frontend manager. Outputs to the frontend carry information provided by the backend modules, but formatted in a way that works with web display. A dynamic diagram can be used to show these interfaces in a more streamlined way:

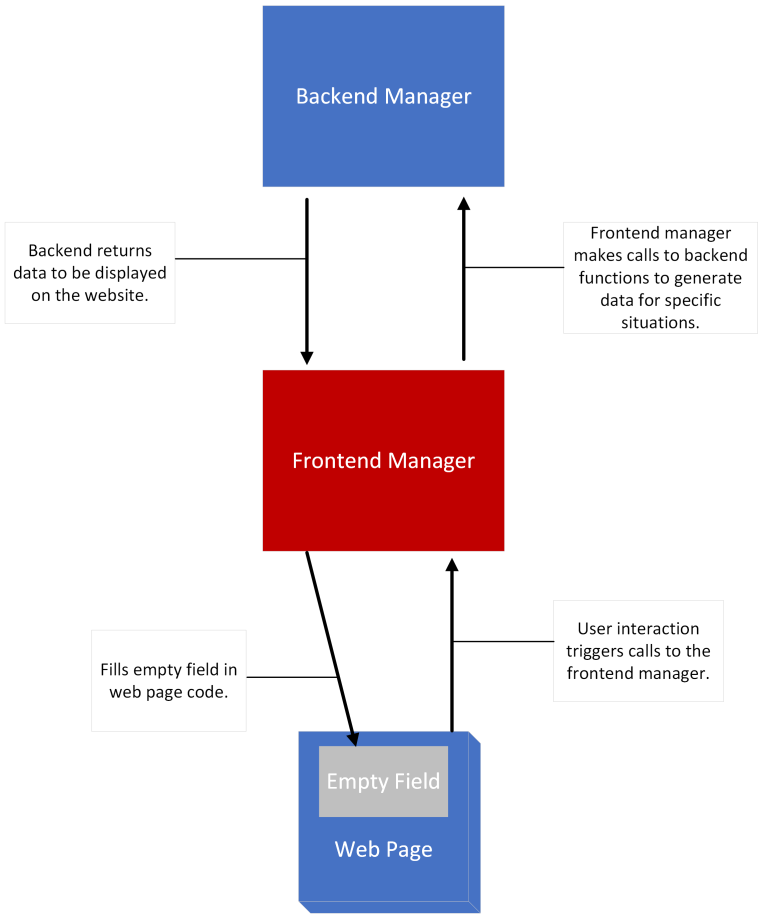
Backend Manager
Backend returns data to be displayed on the website.
Frontend manager makes calls to backend functions to generate data for specific situations.
Frontend Manager
Fills empty field in web page code.
User interaction triggers calls to the frontend manager.
Empty Field
Web Page

Fig. 3.2: Frontend Manager Dynamic Model

The design rationale behind the frontend manager prioritizes system communication and cohesion. We could have designed Gbiv so that frontend components interfaced directly with backend modules, but by establishing a central place for inputs from and outputs to user-facing modules, we simplified our implementation significantly.

3.2 Web Interface

The Web Interface module is essential to the system because it defines the user experience in our application. The design of our website follows the traditional multi-page model seen on many popular websites with a navbar at the top to switch between the pages. Each page has a different functionality and displays different information to the user. The general layout of each page is shown visually in the static diagram below:

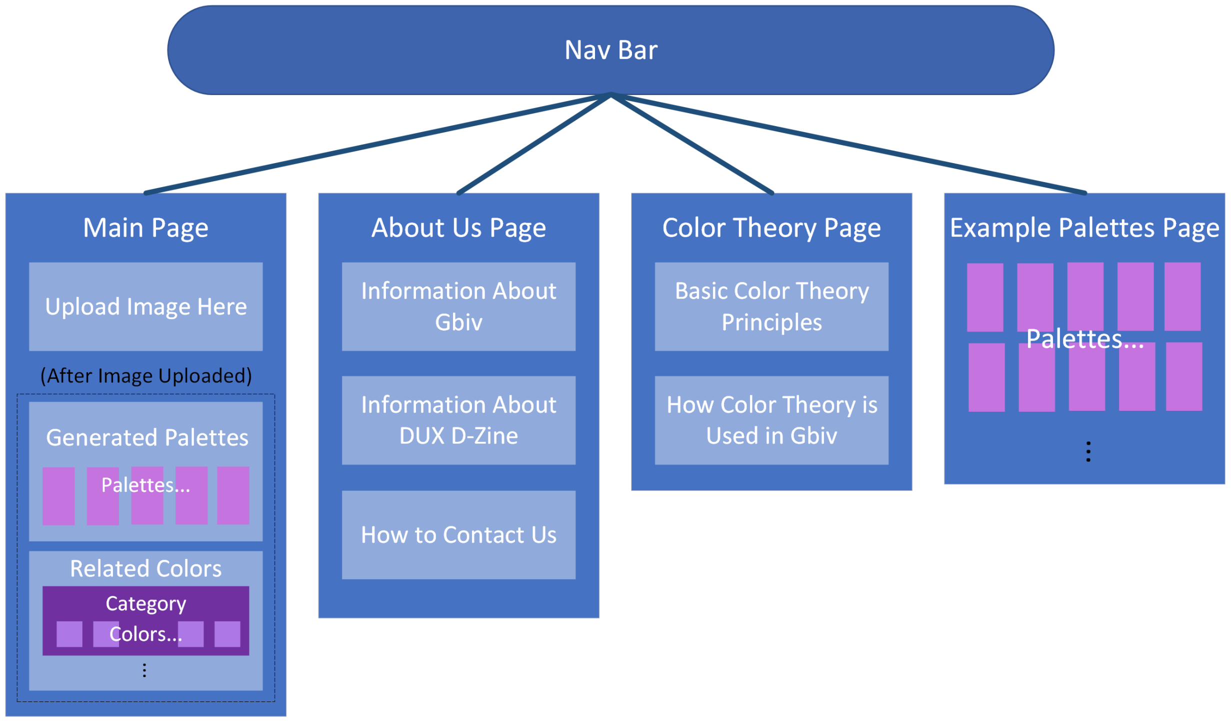

Fig. 3.3: Web Interface Static Model

There are two main interactions that the web interface has with other components in the system. First, it interacts extensively with the users through Internet by way of the server it is hosted on. The web interface is how the user requests the services that Gbiv provides and receives the information that is generated in response to these requests. Because Gbiv is web hosted, the web interface needs the pythonanywhere server in order to properly interface with users.

The second interaction that is key to keep the web interface functioning is the communication between each page and the Frontend Manager module. The Frontend Manager connects the user-facing display with the backend functionality and allows for dynamic pages that respond to user input. These two types of interaction are elaborated on in the dynamic model below:

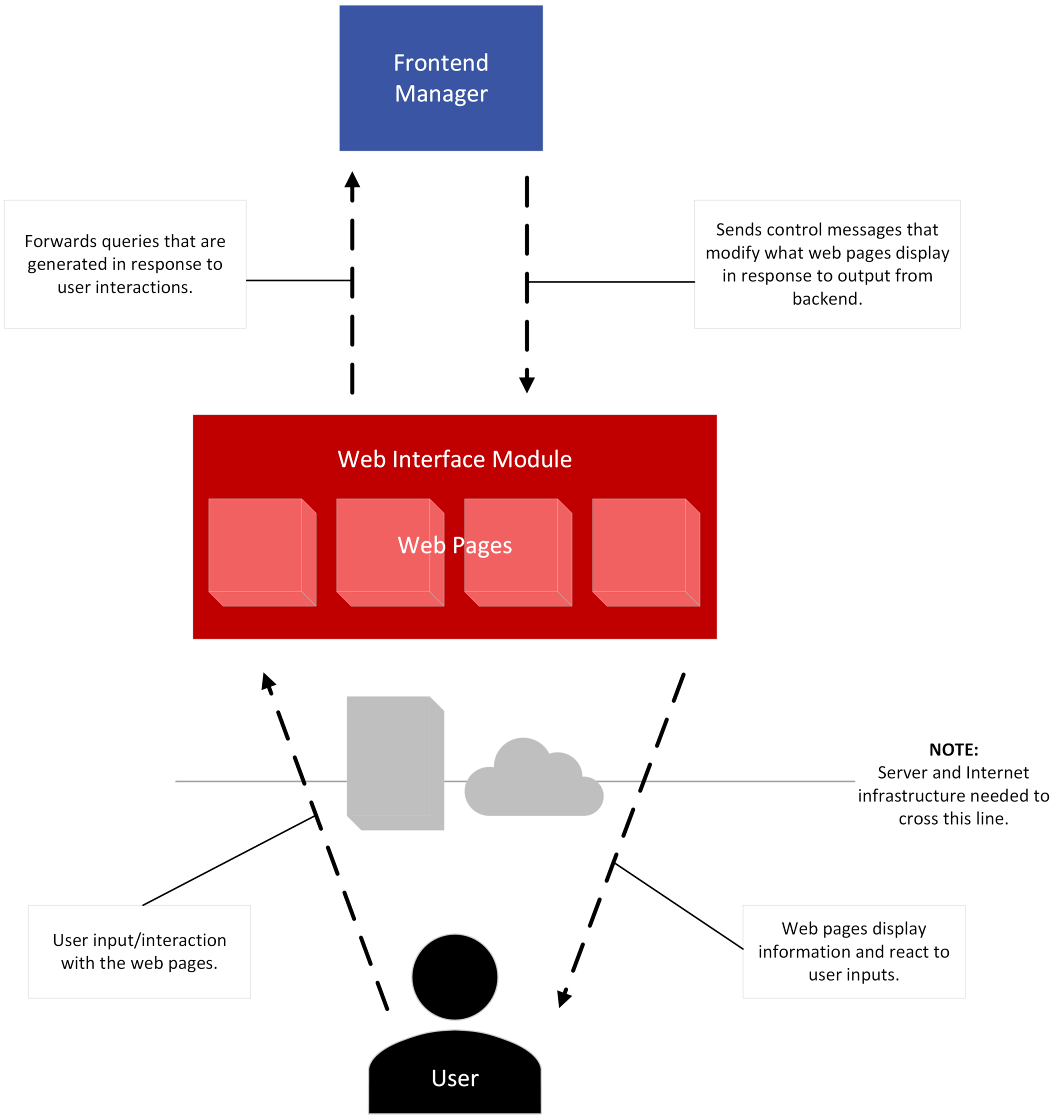
Frontend Manager
Forwards queries that are generated in response to user interactions.
Sends control messages that modify what web pages display in response to output from backend.
Web Interface Module
Web Pages
NOTE:
Server and Internet infrastructure needed to cross this line.
User input/interaction with the web pages.
Web pages display information and react to user inputs.
User

Fig. 3.4: Web Interface Dynamic Model

The website was designed in this way for two primary reasons: (1) the multi-page website is familiar and therefore easily navigable for our users and (2) having separate pages increases modularity. The former point is important for Gbiv because our target users are a very wide demographic, so we want an intuitive and accessible user interface. The latter point has key advantages when it comes to the development of the system. In particular, modularity allows for easier delegation of tasks and for more efficient and focused debugging when problems arise.

The web interface module can be divided into sub-modules based on separate pages on the site. Below we have a brief description of each page's functionality and structure.

3.2.1 Main Page (Upload an Image)

This is the page where users can upload an image to have its dominant color extracted and related colors and palettes generated for that dominant color. For more information on the dynamics of this use case see Fig. 4.1. At first the page will only have a skeleton with blank palettes and color blocks, but after the user uploads a valid image, those blocks will be populated with the generated colors and the user's uploaded image will be displayed.

3.2.2 Example Palettes Page

This page of our website shows a variety of example palettes so that users can get ideas and inspiration for their own color palettes.

3.2.3 Color Theory Page

This part of the website is purely informational. It will provide users with basic knowledge of color theory and show how the principles of this discipline have been applied in Gbiv to generate new colors after an image has been uploaded.

3.2.4 About Us Page

Like the color theory page, the "About Us" page has little to do with the dynamics of Gbiv, rather it exists to provide background to the users. Information about the project and the team are important because it gives users an avenue for contacting the team to report bugs or to become a contributor themselves if we make this system open source in the future.

3.3 Backend Manager

The functionality of the backend manager is very similar to that of the frontend manager (see Section 3.1) in that it is middle ground for communication throughout the system. It is a vital part of the overall framework of Gbiv because without it the connection between the front and backend would be much more complex and vulnerable to bugs. Like the other "manager" module, this component is mostly defined by its interaction with other modules. However, we can still make a basic static diagram that shows the structure through which information flows:

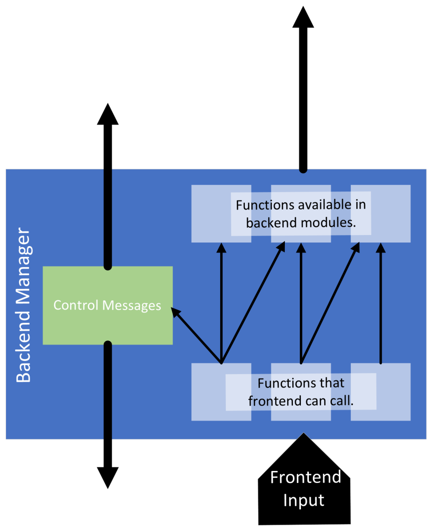
Backend Manager
Control Messages
Functions available in backend modules.
Functions that frontend can call.
Frontend Input

Fig. 3.5: Backend Manager Static Model

The backend manager has both inputs and outputs from the front and backend. On the frontend, the inputs come in the form of requests for data and/or computation that requires backend modules. The outputs to the frontend are entirely control messages because the backend modules that manage computation and data retrieval can return directly to the frontend manager.

Outputs to the backend come in the form of function calls to either the color analyzer or database interpretor modules. In addition to these function calls, control messages may be passed along to the backend for special cases such as error handling and application updates. To show all of these inputs and outputs in a concise manner we can build a dynamic model for the backend manager:

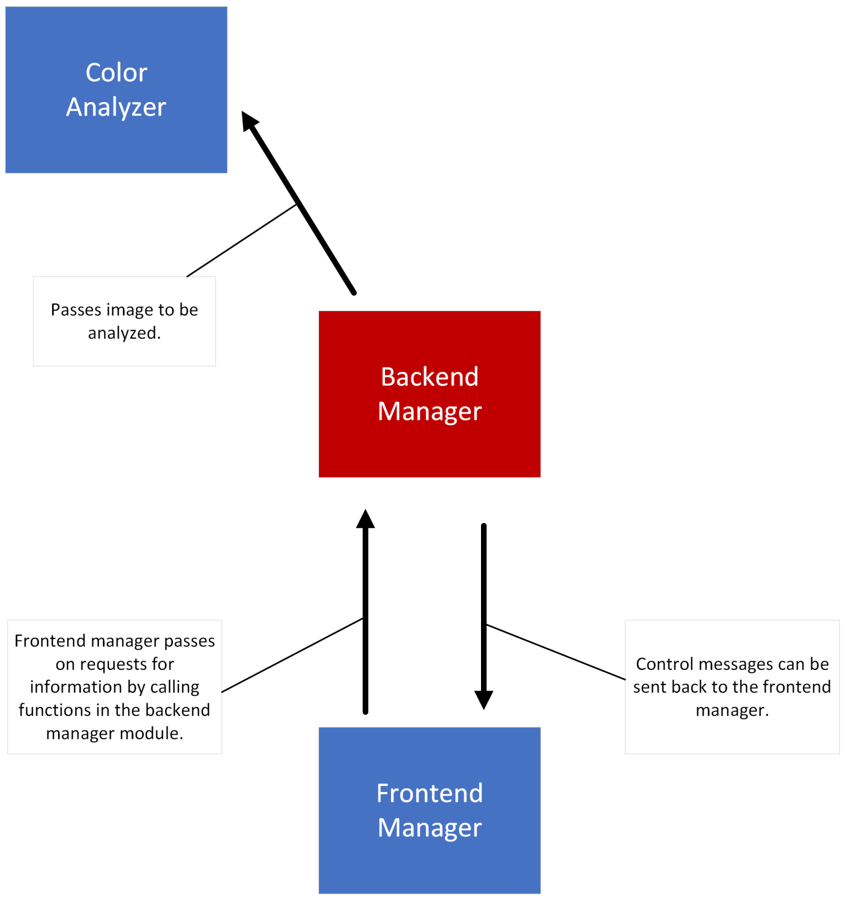
Color Analyzer
Passes image to be analyzed.
Backend Manager
Frontend manager passes on requests for information by calling functions in the backend manager module.
Control messages can be sent back to the frontend manager.
Frontend Manager

Fig. 3.6: Backend Manager Dynamic Model

This module was designed with ease of communication as the main goal. By establishing a central module where communication from the frontend to the backend passes, we are able to reduce the structural complexity of the system and do more with less function calls. Furthermore, by having the color analyzer return directly to the frontend, we avoided the need for extensive data processing and reformatting.

3.4 Color Analysis

The primary function of this module is to the color analysis and generation that happens after a user has uploaded a photo. This module is made up of several sub-modules (divided by functionality) which are further divided into sub-sub modules. The static model below gives a visual picture of how the color analysis module is structured.

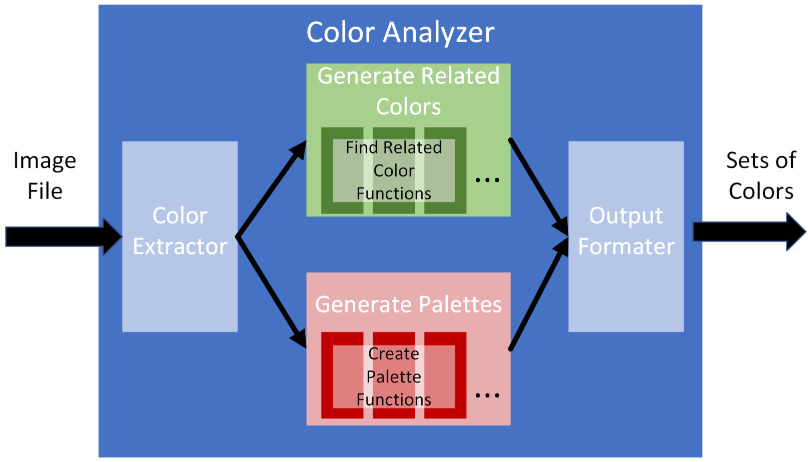

Fig. 3.7: Color Analysis Module Static Model

As the above model shows, all of the work with color manipulation and analysis is done within the module. This makes for a high level of cohesion that allows for a weak coupling with other modules in the system. In fact, the color analysis module only has to take inputs from a single module which is the “Backend Manager.” The backend manager passes an image in the form of a .png, .jpg, or .jpeg file and this module returns several sets of color codes as lists of hex code strings. All outputs of the color analyzer go directly to the frontend manager. The inter-module interactions of this part of the system are further specified in the dynamic model below.

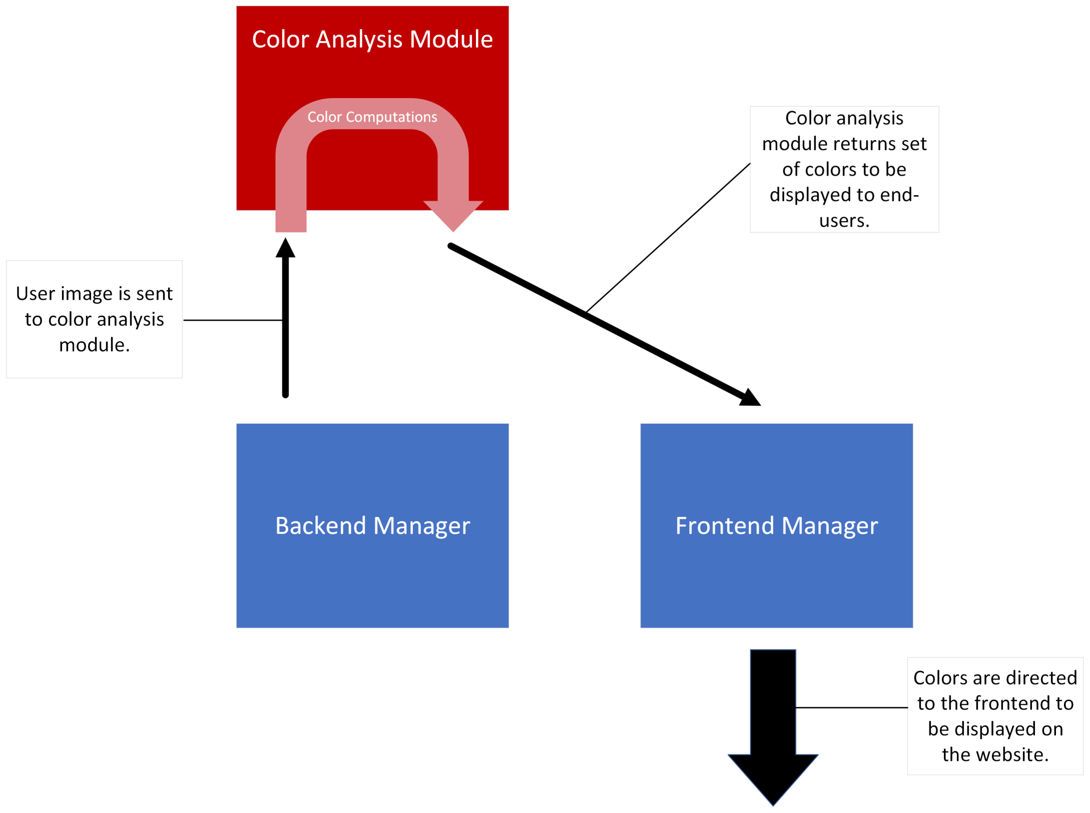
Color Analysis Module
Color Computations
Color analysis module returns set of colors to be displayed to end-users.
User image is sent to color analysis module.
Backend Manager
Frontend Manager
Colors are directed to the frontend to be displayed on the website.

Fig. 3.8: Color Analysis Module Dynamic Model

This module was designed with a high degree modularity in mind. By separating the color analysis process into two parts, we are able to define two classes of sub-functions that share common features: palette generator functions and related color finder functions. This allows for code re-use and also source code that is easier to read and interpret. We also designed this module to have simple data types as both inputs and outputs. This allows easier integration with the rest of the system and fits well into our chosen framework (Flask).

CHAPTER

FOUR

DYNAMIC MODELS OF OPERATIONAL SCENARIOS

There is a single use case for Gbiv which is described in the SRS. However, it should be noted that our application has functionality beyond this in the form of providing information. The information given on the sample palettes, color theory, and about us pages is essential, but requires little to no interfacing with the user and between modules, so they are not described below.

Our use case describes how users can upload an image and have Gbiv extract the dominant color and make color suggestions in response to that dominant color. The diagram below (Fig. 4.1) shows the operational scenario visually in terms of user interaction, as well as interaction within the system.

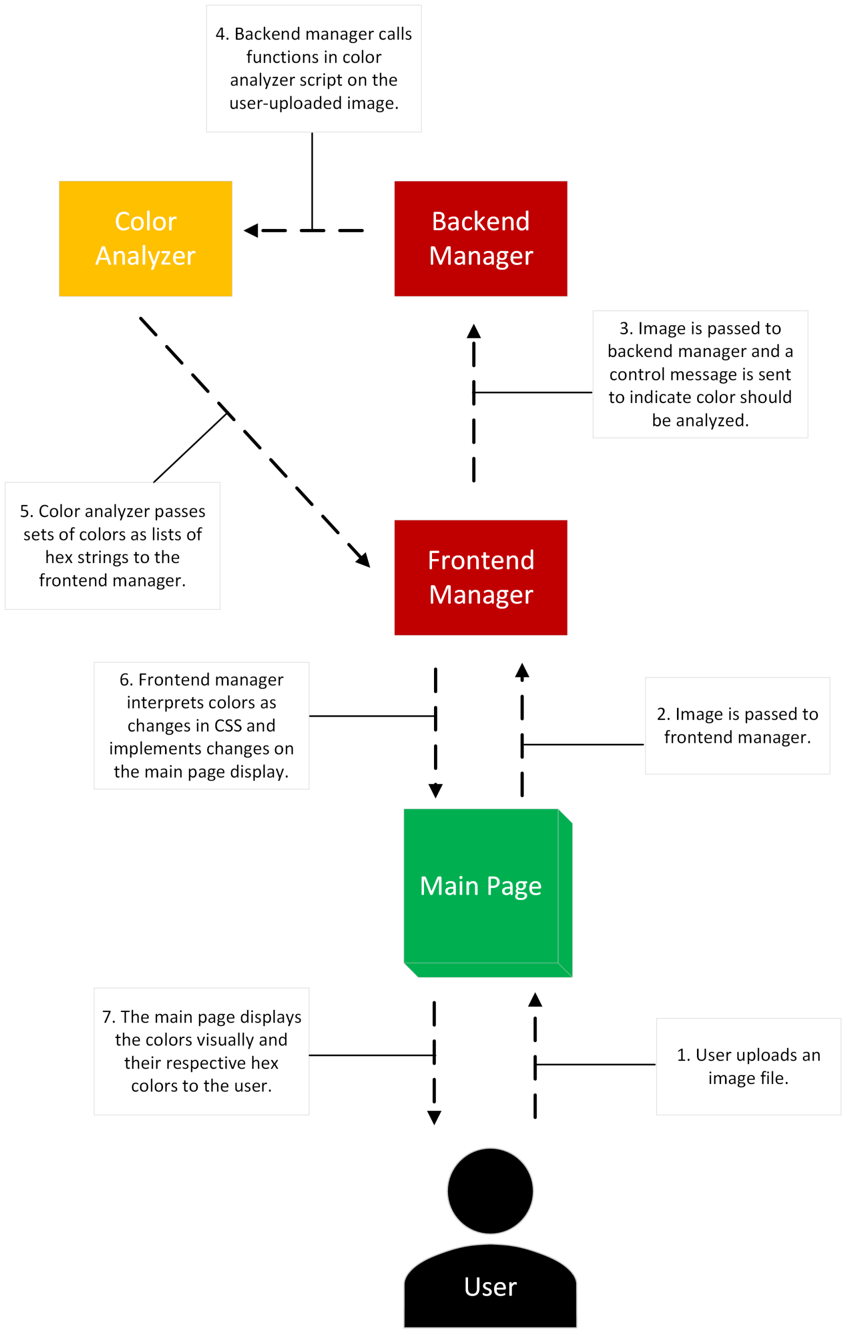
4. Backend manager calls functions in color analyzer script on the user-uploaded image.
Color Analyzer
Backend Manager
3. Image is passed to backend manager and a control message is sent to indicate color should be analyzed.
5. Color analyzer passes sets of colors as lists of hex strings to the frontend manager.
Frontend Manager
6. Frontend manager interprets colors as changes in CSS and implements changes on the main page display.
2. Image is passed to frontend manager.
Main Page
7. The main page displays the colors visually and their respective hex colors to the user.
1. User uploads an image file.
User

Fig. 4.1: Use Case #1: Uploading an Image for Color Analysis

CHAPTER

FIVE

ACKNOWLEDGMENTS

The format of this SDS document was originally created by Professor Juan Flores. Reference to a completed SDS document was provided by Ronny Fuentes, Kyra Novitzky, Jack Sanders, Stephanie Schofield, Callista West with their Fetch project SDS.