Gbiv System Design Specification

Release 0.5

Dux D-zine

Nov 16, 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 popular palettes that have been viewed often on the site.

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 Bootstrap 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 popular 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. The scripts that contain functions that are called by Flask are divided into two python files: the color analyzer and the database interpretor. The former handles all the color extraction and related color calculation while the latter provides an interface with the database that stores popular palettes. The database is implemented using MongoDB and is accessed using Mongo's provided python library.

These two sections of the system are tied together using the Flask framework as described above. Gbiv is hosted online using Firebase and the source code is maintained through GitHub provided version control. 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 use Flask as our framework throughout.

The diagram below 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 design because ultimately Gbiv is built for the users.

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**
Palette Database	Backend	Stores popular palettes that many users have viewed.
Database Interpretor	Backend	Interfaces with database to pass queries from the system to the DB and to transfer data from the DB to the rest of the application.
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.

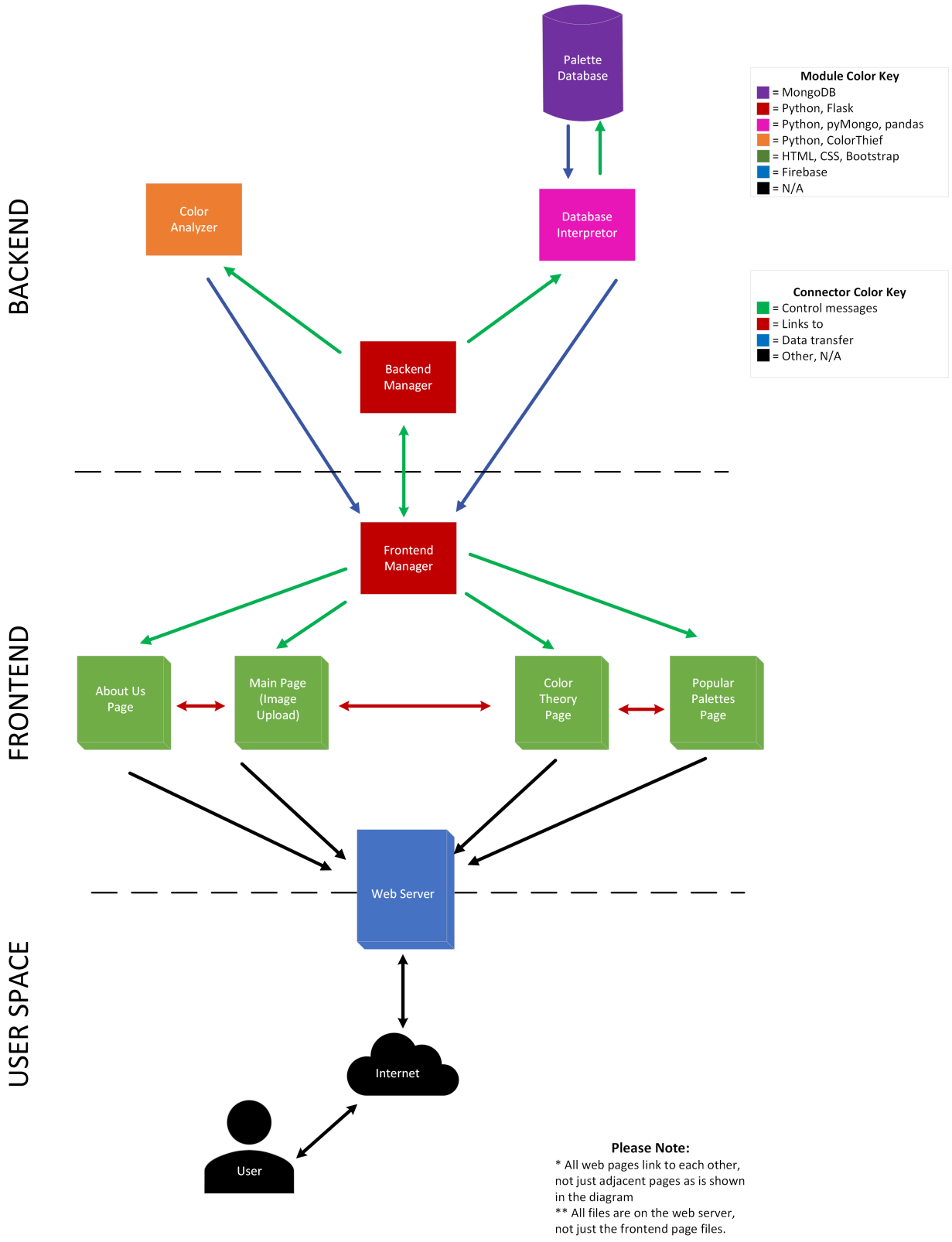

Fig. 2.1: Gbiv Software Architecture

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 will function.

First, we can look at the transferring of control messages between the frontend and backend manager modules. These messages are formatted with the Flask python framework which is also used in implementing both of these key modules. Having consistent implementation and technology in these "bridge" modules allows communication without complicated translation or an additional framework in the mix.

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 Firebase 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 the target users.

CHAPTER

THREE

SOFTWARE MODULES

3.1 Frontend Manager

3.2 Backend Manager

3.3 Web Interface

3.3.1 Main Page (Upload an Image)

3.3.2 Popular Palettes Page

3.3.3 Color Theory Page

3.3.4 About Us Page

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.

As the above model shows, essentially 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 interact with 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. The inter-module interactions of this part of the system are further specified in the dynamic model below.

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).

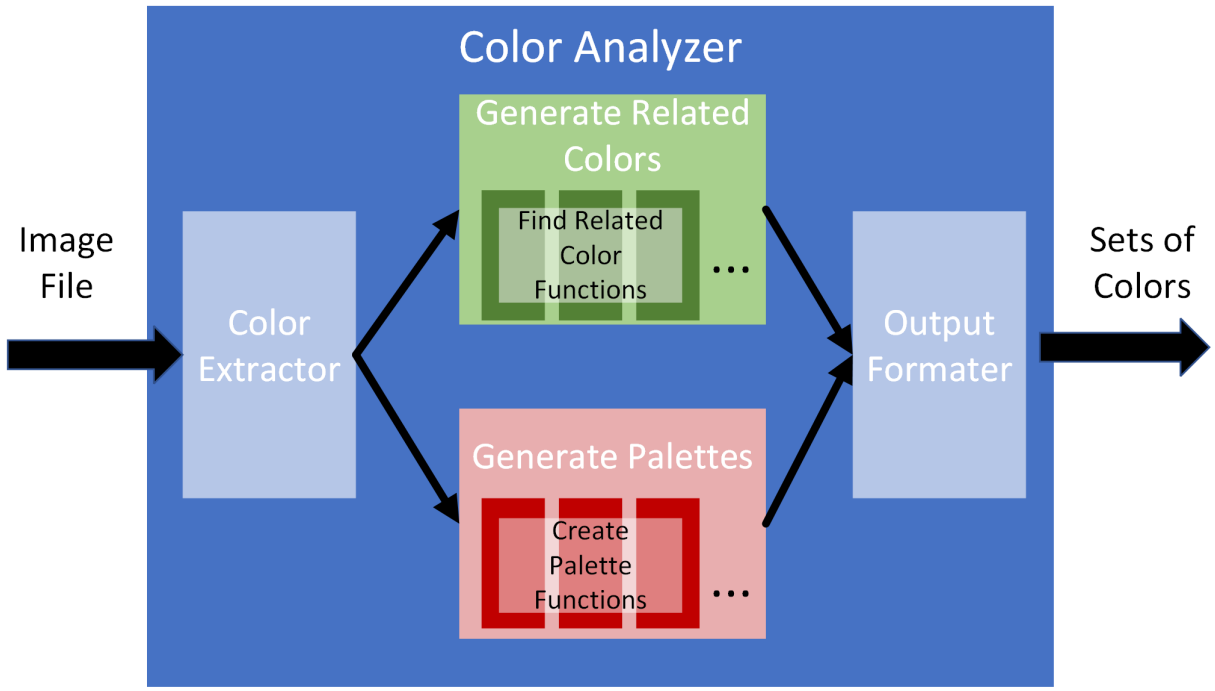

Fig. 3.1: Color Analysis Module Static Model

3.5 Palette Database

The purpose of the Palette Database module will be to store popular palettes that many visitors to Gbiv have looked at. This will allow users to view a range of different color combinations and get inspiration for their design projects. Because there is only one collection of elements in the database for this project, the design of the database itself is somewhat straightforward. The static model below shows the layout visually:

The technology we will be using to implement our database (MongoDB) comes with a library that allows for efficient interfacing through python. Because of this built in advantage, we have designed the system so that the database has one module with which it communicates, the "Database Interpretor." This interface consists of a single type of input and a single type of output. When a user visits the "Popular Palettes" page, the frontend will query that backend which will reach the palette DB as a request to view the collection of palettes. When this query happens, the database will pass the collection on to the database interpretor module in a format that allows for easy movement to the end-users. Below we have included a dynamic model to demonstrate this interface.

The design choices for the palette database module were made with the goal of simplicity. By keeping the number of collections to a minimum and formatting all data entries identically, the organization and movement of Gbiv's data can be straightforward and efficient. This prevents database accessing from being a bottleneck for performance, as well as reduces the need for more modules for data formatting.

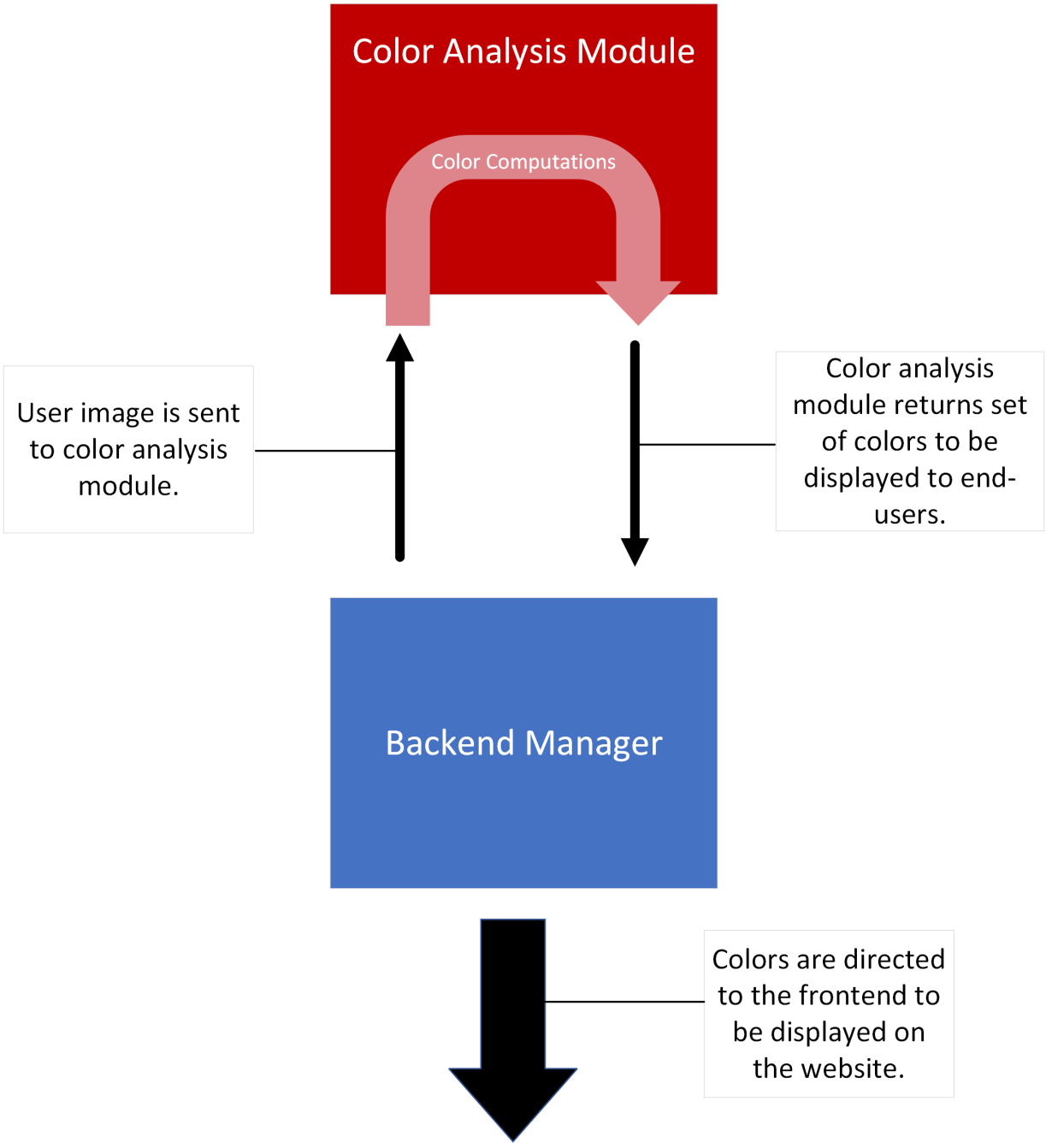

Fig. 3.2: Color Analysis Module Dynamic Model

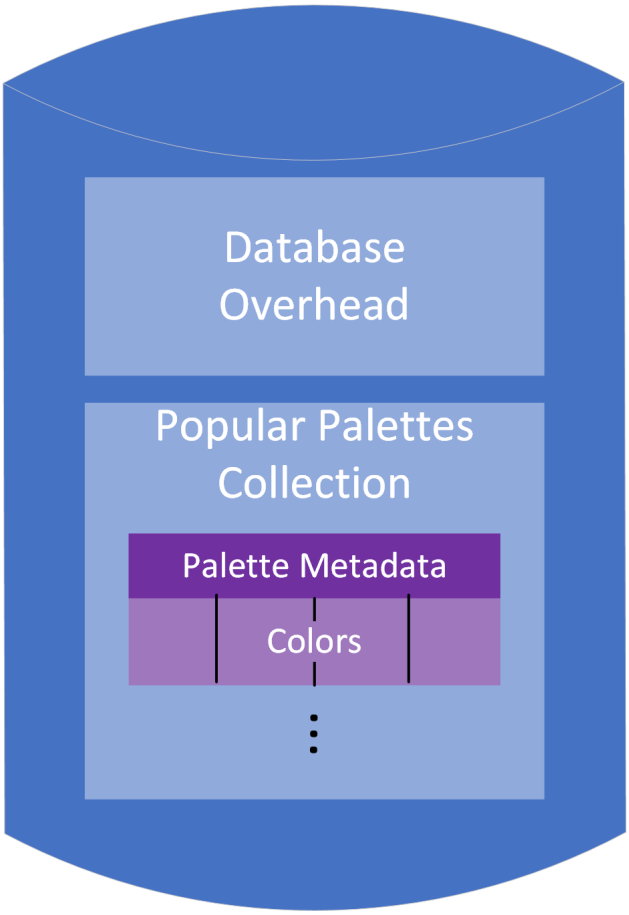

Fig. 3.3: Palette Database Static Model

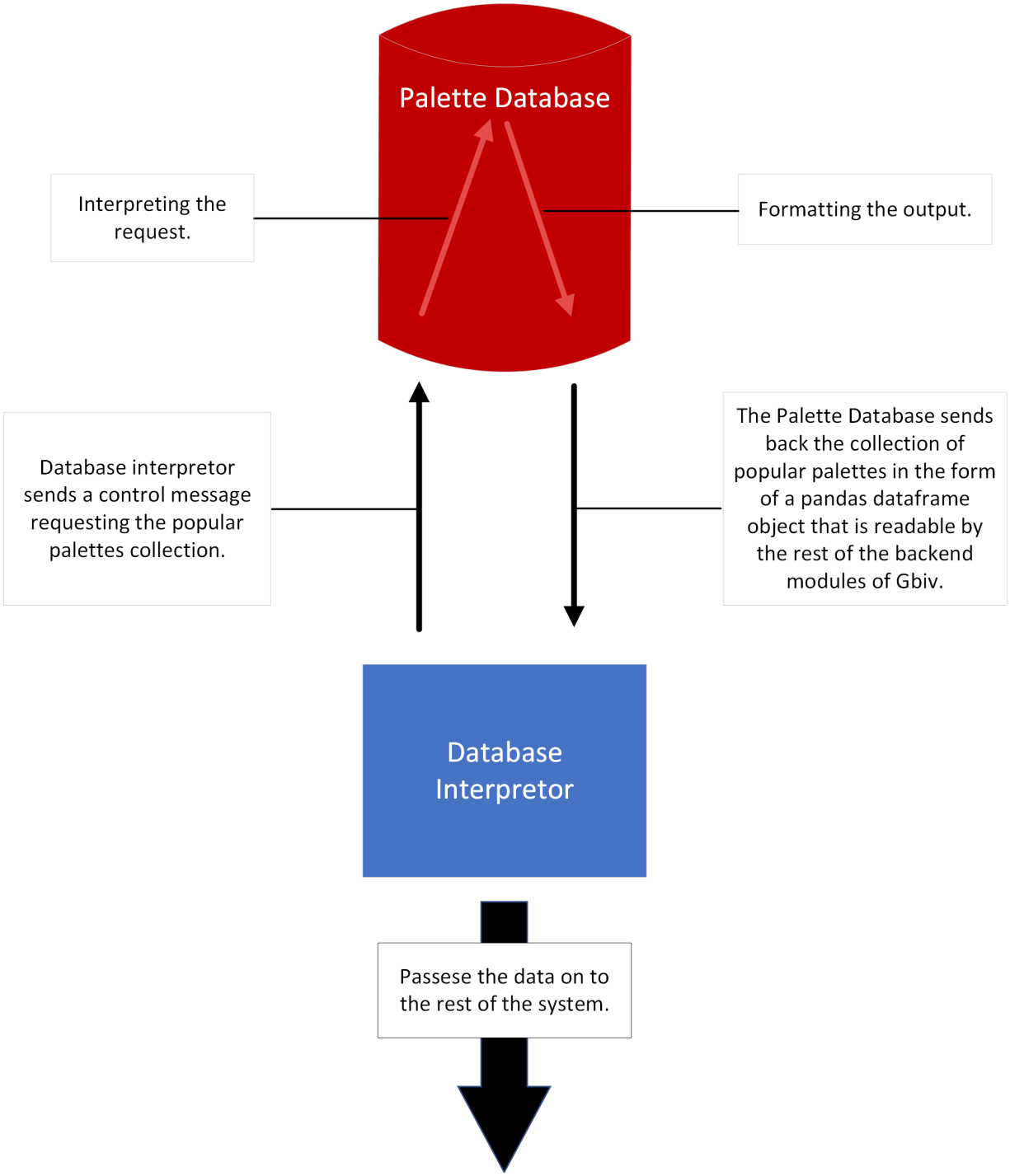

Fig. 3.4: Palette Database Dynamic Model

3.6 Database Interpretor

CHAPTER

FOUR

DYNAMIC MODELS OF OPERATIONAL SCENARIOS

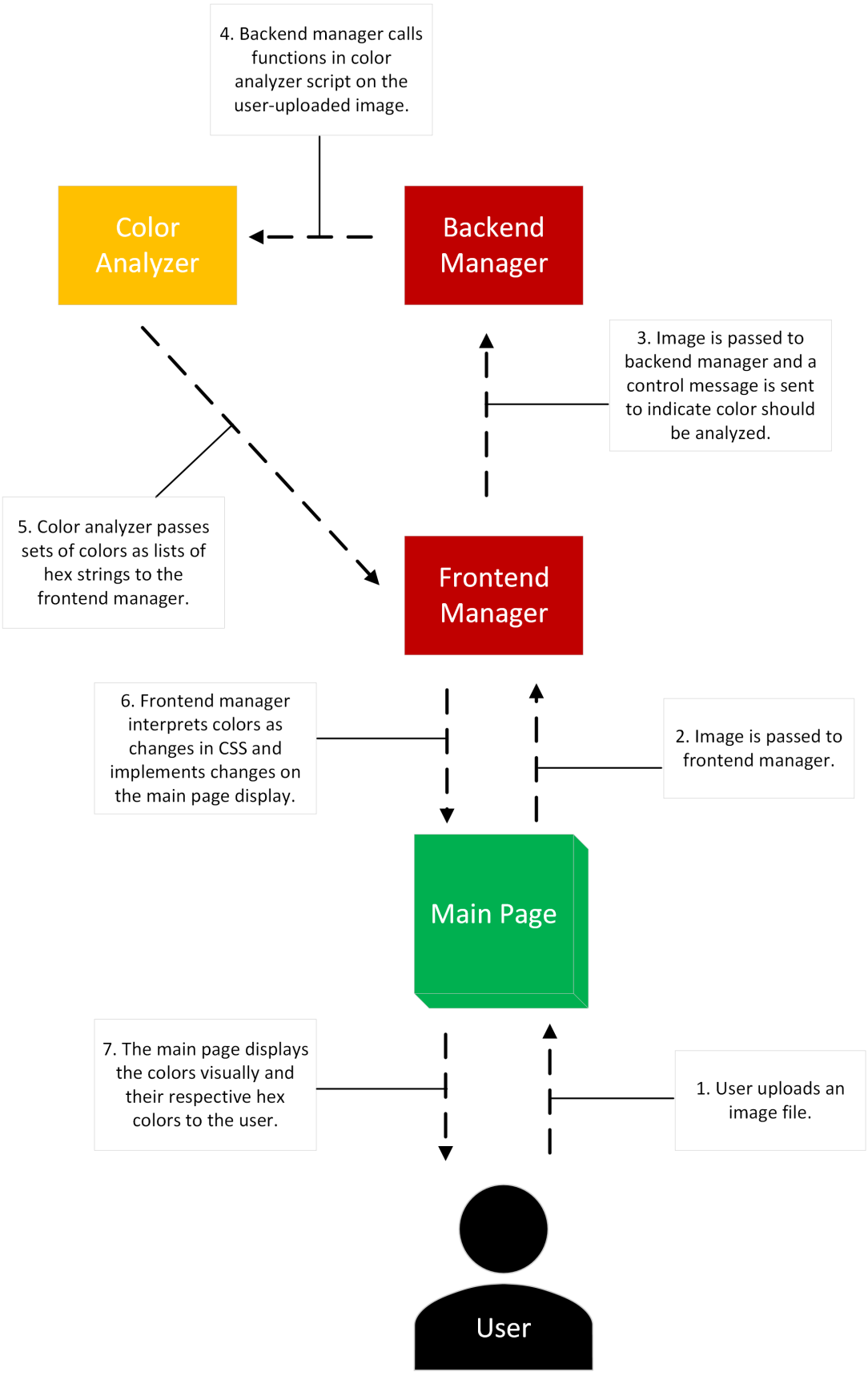

Fig. 4.1: Use Case #1: Uploading an Image for Color Analysis

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.