Software Design Document
Team Bird's iView

February 5th, 2021

Version 1.0

Project Sponsor: David Plemmons

Project Mentor: Sambashiva Reddy Kethireddy

Team Members:

Jonas Dunham Jordahl

Jordan Colebank

Chenhao Li

Tyler Riese

This document will brief readers on the architectural overview, as well as implementation details, that will define how the pEEp website functions.

Table of Contents

Introduction

More than 50 million people spend more than $40 billion every year on bird-related equipment and travel. Birding activities can bring people not only spiritual pleasure, but bring huge economic effects. “Birding in the United States: A Demographic and Economic Analysis” (U.S. Fish and Wildlife Service), an addendum to the 2011 National Survey, shows that bird watchers spend nearly $41 billion annually on trips and equipment. Local community economies benefit from the $14.9 billion that bird watchers spend on food, lodging and transportation. In 2011, 666,000 jobs were created as a result of bird watching expenditures. In September 2017, a presentation at the American Birding Expo found that, among other events, the Space Coast Birding Festival generated nearly $1.3 million over 5 days.

According to the above data, we can find that the bird-watching industry has a huge market. Additionally, the bird-watching industry can bring great spiritual enjoyment to people by connecting with nature. There are about 10,000 species of birds living in countless habitats on seven continents, with different sizes, different color patterns of feathers, and even different songs and actions. Birding is also good for your health as it provides great pleasure to observe a wide variety of birds. To find rare species, you may take a hike outdoors.

More importantly, birding also allows you to make friends. You may meet people who have the same hobby of watching a particular kind of bird and become friends with them. Nowadays, with the internet you can make friends with people who live on the other side of the world.

Our client, David Plemmons, is an avid bird watcher and enthusiast. He has been working hard over the last three years to bring his vision to fruition. His product, the pEEp Smart Feeder, aims to get people more involved with the birds around them by providing an educational platform and vibrant community for people to connect and grow with other like-minded individuals. Here are the features he wants us to provide for his software:

- **Provide a space where users can upload photos of birds taken by pEEp**
  Because our client’s product——pEEp Smart Feeder can take pictures when feeding birds, he wants his users can see the pictures of the birds that pEEp taken and he wants users can know more about the species of birds, so it is necessary to have a space to store the pictures of birds.
- **Provide a discussion forum where users can share their pictures with each other**
  Our client David hopes our users can have a space to share their photos of birds.

We think this is a great feature, which allows people to feel the joy of making like-minded friends on the Internet, as well as learn new knowledge.

- **Provide the AI module for users to know the species of the birds**
  This is the core function of our software. With the popularity of artificial intelligence, our customer hopes that his website can have the function to help users identify birds, so as to better popularize the knowledge of birds and attract more users.

To solve our client's requirements, we will develop an online website application called bird's iView web to satisfy our users with these functions above. We will help our users who buy the pEEp feeder upload their pictures of birds, and then our website will provide them with a personal space to share the pictures and an AI model to help them to identify the species of birds. Here are the specific features of our application:

- **User account:** our users can create an account on our website, and they can have a personal profile which can be customized by them, by using this account, our users can upload the pictures of the birds they took, and they can also chat with other people in the forum.

- **Chatting forum:** the chatting forum is a space for our users to share their pictures of birds, our users can create a thread to show his pictures and other users can comment on it. We hope that through this function, users can share with each other, learn new bird knowledge, and thus play an educational and social function.

- **Database for pictures store:** the database will help our users store pictures and personal information. We will use MySQL as our database. Users can upload photos to the database and delete photos from the database.

- **AI model:** our user may not know what the specific species of the bird he photos. Therefore, our website will have an artificial intelligence model, which will identify the species of birds. This function will be our core function. Of course, this work is also very difficult, and we hope to improve the success rate of recognition as much as possible.

- **Expert Answer Area:** Our customer hopes that his website can play the role of popular science education, so we plan to have an expert section, so that our users can upload their bird photos to bird experts, who will answer their users about birds, and this function can also help them to better explore bird species.

Implementation Overview

For our project, we will be designing a modern web application that will act as a central community for avid birders, as well as a place where people can come to learn more about birds. We will have a customizable profile system, storage space for users to save their images, tools to interact with the images they've taken, and a community forum where users can connect with each other. This web application will eventually be integrated with the pEEp Smart Feeder, but that integration is beyond the scope of this project.

Our overall structure will include the standard components of most modern web applications: a frontend portal, a backend API, and a database. To stitch all of these components together, we are adopting the Model-View-Controller design pattern (hereafter referred to as MVC), as this design pattern has proven to be extremely effective at breaking down complex web applications into clean, well-defined components.

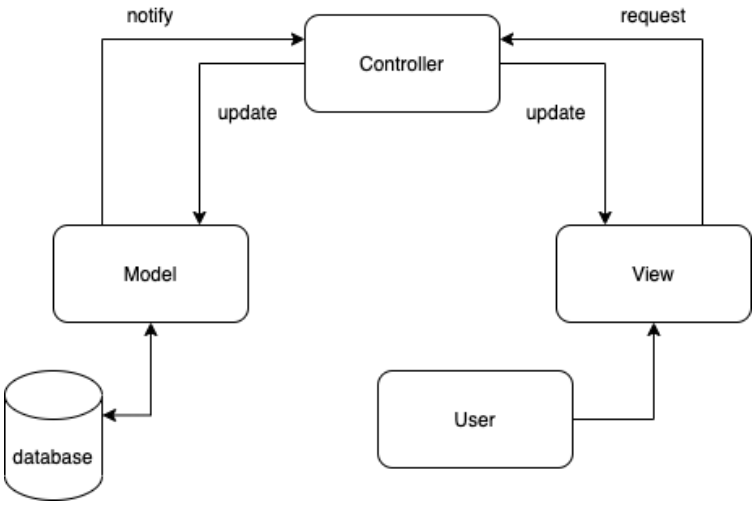
Our dedicated controller will be the main Flask application file, and will be responsible for receiving requests and dispatching them to the appropriate model functions. In this way, all core logic will be abstracted away from the controller and encapsulated within the model, allowing the controller to focus solely on directing requests. The model, which will handle the real meat of each request, will act as the central point for database access and general computation, and will respond to such requests with the relevant data and results.

Upon receiving the results from the model, the controller works to integrate the data into a view, in effect dynamically creating custom web pages based on the results of the call to the model. In this way, we completely separate the model and view components, giving us the desirable decoupling that will allow the model and view to progress at their own rate, (mostly) independent from each other.

To implement this architecture, we will be using Flask and associated libraries to handle all controller and model logic, Jinja2 templating to dynamically create web pages, Keras in our AI section, and MySQL as our database.

Architectural Overview

As stated above, we are going to be using the Model-View-Controller design pattern as a guideline for the architecture of our system. Shown below is a simple diagram of the MVC design pattern which will be further discussed.



This diagram above is composed of 4 main components. The view, controller, model, and database. Within each component are designated responsibilities that allow our application to be fully functional for the user.

The View: The two major responsibilities of the view component is to provide the user with data and to handle the user interactions.

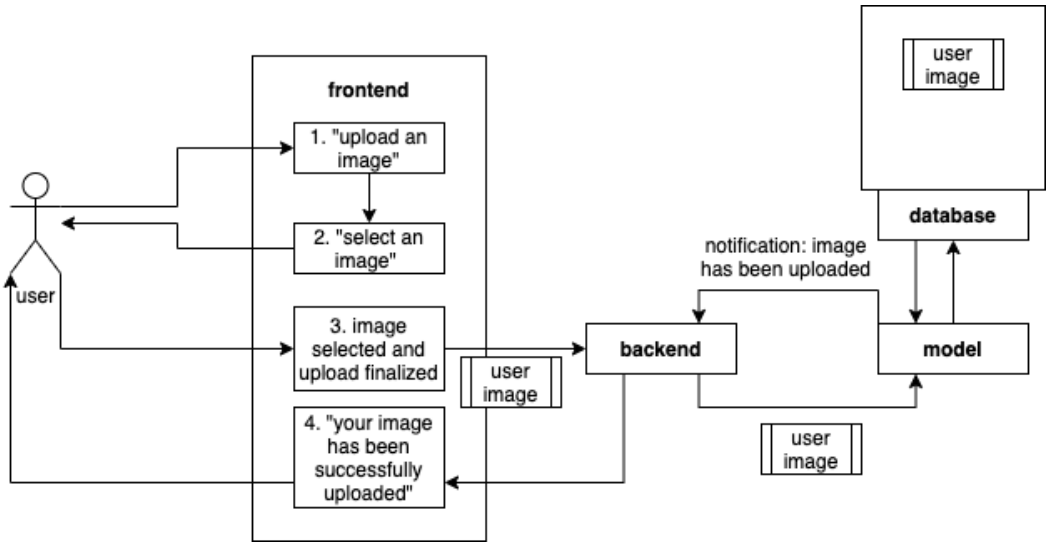
The Controller: The controller plays a big role in the MVC design pattern as it acts as a connection between the model and the view. This component is in charge of handling the requests from the view component, deciding what to do with said requests, and subsequently updating both the model and view components.

The Model: This component solely represents the data of the application. Since the view is in charge of displaying the data to the user, the model is responsible for providing the data to the controller, which then sends the data over to the view.

The Database: In order for data to be provided and stored, a database is crucial. This component (of which is not typically part of the MVC design pattern) is responsible for storing the data.

Now that we acknowledge the main responsibilities of each module, it is important to understand how every part communicates with one another.

For our project, the View would be the frontend element or user-interface of the web application. This provides the environment for the user to interact with each component. Requests are then sent to the backend which notifies the model. The model grabs any data that has been requested or received and sends a notification to the database. Then the data is either stored in the database or the information is extracted from the database and sent back to the controller. The diagram below shows the process of when a user clicks on an “upload an image” button and how each component communicates with one another.



In summary, the key components of the architecture of our application are the view, controller, model, and database. It is essential that all four of these components can effectively communicate with one another, to give the user a fully functioning application.

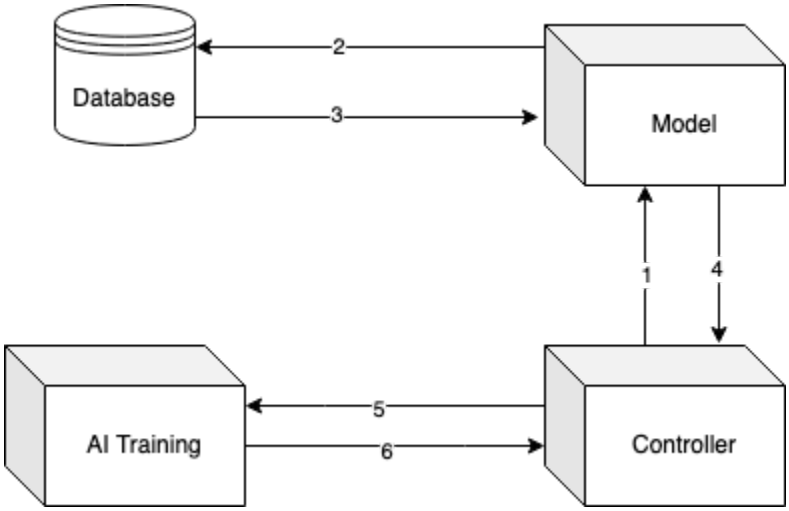
Module and Interface Descriptions

AI Module

The AI module for this system will be split between several components. Namely, the model in our MVC split will be responsible for loading a trained model and running a prediction on user images with this model. This functionality is quite trivial once the trained model is exported to a file.

The main section of the AI model is the training pipeline, which will eventually become fully automated and only require minimal manual verification. The idea here is to separate the training pipeline programs from the main web application in such a way that they can both run independently. To do this, the easiest and most effective option is to spin up an additional server that will listen for requests containing newly verified images that are ready to append to our training set. Every evening, if there are conditions (discussed below) that are sufficient to require retraining, this independent server will train a new model based on the current training set, and send the new model to our main web application via a special request that gets routed to a AI update function in the model.

There are 2 main conditions that will trigger retraining are as follows: new images have put a certain species at or above the minimum image threshold to include in our model, or a species that is currently supported by our model has gained enough new images to require retraining. In either of these cases, a new model will be trained based on the existing training set, and subsequently sent back to the main web application. The visual depiction below illustrates this.



Control Flow

1. The controller asks the model to round up all images that have been verified but not set appended to our training set.
2. The model runs a query on the Images table, asking for the file paths of all images ready to be appended to the training set.
3. The database responds to this query with a list of file paths. After getting a response from the database, the model will package the images together (in multiple packages, if necessary) and send them back to the controller.
4. The controller redirects the packaged images from the model to the AI module.
5. The AI model unpacks the images, appends them to the training set, then sends back a response that signifies whether or not a new model is being trained.

Frontend

The frontend is simply responsible for sending users requests to the backend and then displaying the results of said requests back to the user. Considering that this module is the main point of contact between the user and the backend services, the frontend of our application can be broken down into two main sub-programs.

Authentication

The first and most important sub-program is user-authentication. In order for a user to access any other main subprogram of the frontend, they will need to be an authenticated user. Figure 1.1 gives a clear depiction of how this process is done. Any time an action is executed when a user is not authenticated, a notification will be shown letting the user know that they either need to log in or create an account, as well as a redirect to a place where the user can do so. There will be an entire page dedicated to user sign-up, as well as a link that redirects them to a login box if they already have an account. Once a user has been authenticated and are logged in, they will be able to access the additional subprograms that will be discussed below.

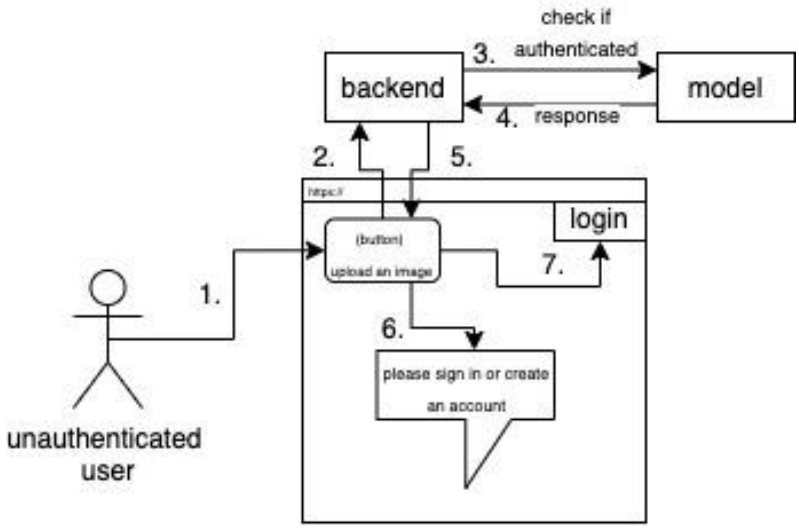


Figure 1.1 - *This diagram demonstrates the process of authentication within the user-interface.*

File Management

Secondly is the file management system. This sub-program is responsible for managing all of the actions in regards to the user's images. There are a handful of sub-programs within the file management, such as uploading images, viewing images, sending images to the AI module, and sharing images. When it comes to uploading image files, the frontend is responsible for providing the environment to upload the images. This would simply consist of a button that says “upload image” and a form that prompts the user to

select images from their local machine. There would be an additional button that would send the request to the backend component of the application. Figure 1.2 below demonstrates the process of uploading and viewing images.

Once an image has been uploaded, there needs to be a designated location for the images to be displayed to the user. There are a variety of ways to display images on a web application, but for now, we can assume the images will be displayed in a carousel on the users home page.

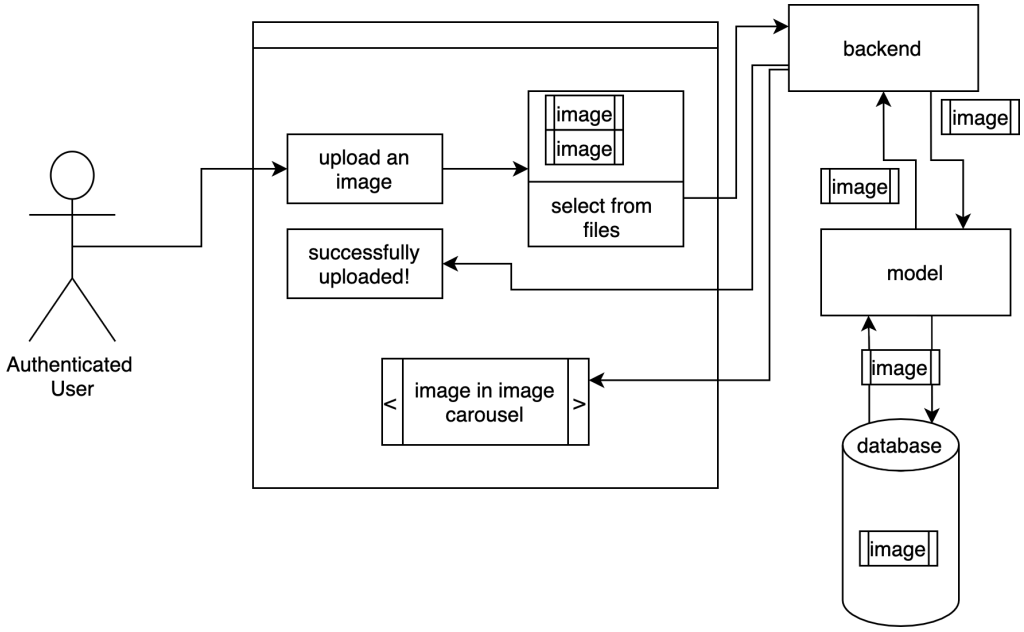


Figure 1.2 *This diagram demonstrates the process of an authenticated user uploading an image and then the image being displayed back to the user.*

Next, there would need to be an option for the user to send their images to the AI classification model. This is similar to uploading an image in the sense that there will be a button provided that says “Classify Image”. In addition to sending the request to the backend component, the frontend will need to retrieve the classification results and display them on the screen to the user.

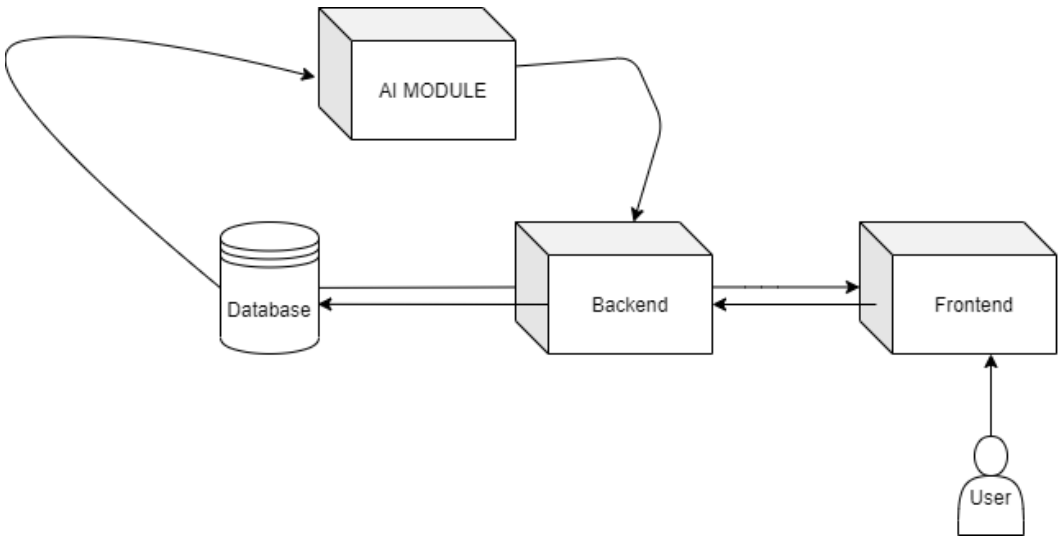
Lastly, the user will be given the option to share their images with the rest of the community. The frontend will need to provide a modal that displays the image being shared, as well as a form box that allows the user to create a caption. Furthermore, the new image post will be displayed on a separate page known as the “forum” page, where other users can view and interact with the shared images.

In summary, the frontend component's main job is to provide the interface for the user by being the main point of contact between the user and the backend services. This can be done in many ways by providing the user with buttons, boxes, modals, image displays, and other forms of data display.

Backend

As the back end, its main job is to connect to our front end and database. Our users create their own accounts in the front end, and through the back end, we transfer the account information created by users to our database for storage. Then when the user wants to modify their account information, our database will pass the user's information through the back end to the front end, so that our users can according to their own needs to modify one or several items of information. At the same time, users can also upload photos of birds, which will also be uploaded from the front end to our database and stored through the backend. Users can view the uploaded photos anytime and anywhere, and they can also delete or modify the photos.
Whenever users take pictures of birds they do not know, our website will provide artificial intelligence recognition function to help users identify these birds by using the obtained data, so as to play the role of education and popular science.

The following diagram will show more visually the roles and functions rehearsed by the back end



For our backend, we will use Flask as our backend framework.The advantage of Flask is that it meets the needs of our site without taking up a lot of space and is lightweight.

We will use the Flask to create two interfaces, a front end, used to connect to us through this interface, the front end data will be uploaded to the backend folder, and then we will use another interface to connect to our database, through this interface, data will be stored in the form of words or pictures in the database we have created.

The way we use for the interface of Flask is GET and POST now. By using these two methods, we can connect our index.html and flask app.py. And we can now upload pictures into our website. For the future, we will make it more concrete. We will create a new folder on our website to display photos uploaded by users, and we will also connect to our database so that we have a better space for storing user information and files.This will be our next interface goal.
Our interface will return the user's file as a PNG or JPG file, which is also the type of image that our user will upload.

Database

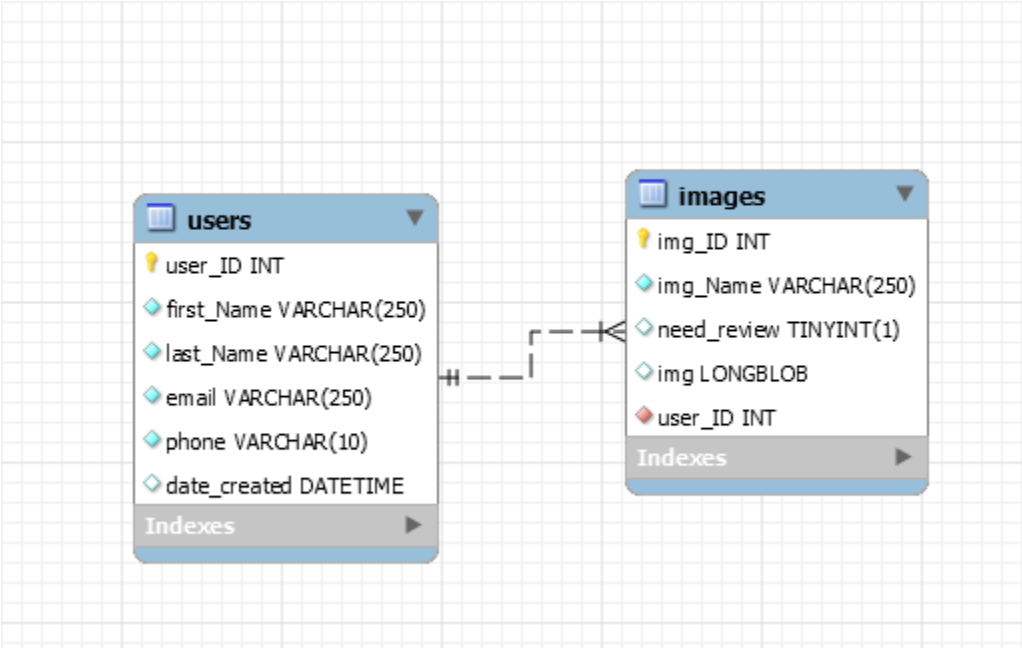


Diagram Breakdown

This is an ER or entity relationship diagram. Demonstrating the relationship between the users table and the images table in our database.

Primary Keys (represented by lightbulb): These include the user_ID and img_ID variables. These represent unique identifiers for users and images. For example, a new

user will receive a unique ID number that is exclusive to their account. Similarly when a user submits an image, the image will also be granted this unique identifier.

Foriegn Keys (represented by the red diamond): In our database we only have one Foriegn Key at this point in time. It is the user_ID being represented in the images table. This is done so that the unique user_ID can be tied to all user submitted images, thus linking the images back to their original uploader.

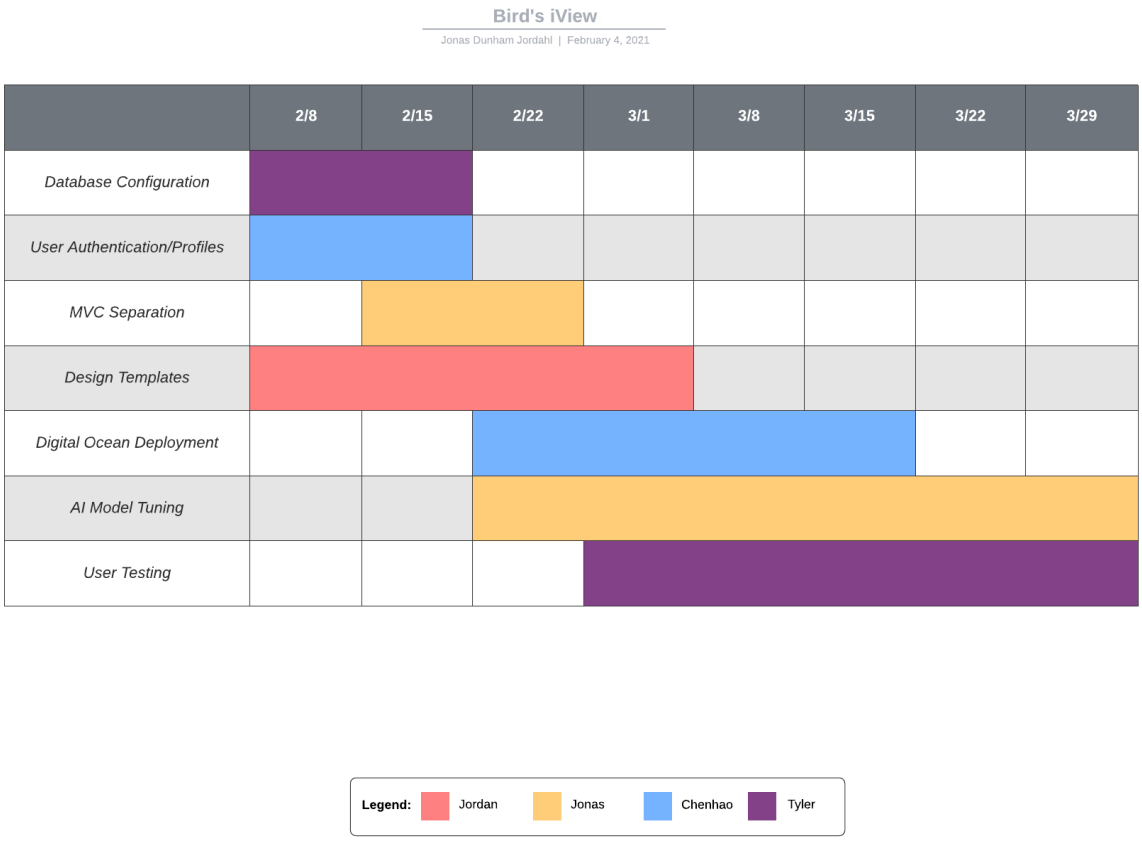
Other stored data (Users): The users table will take in the following information, the user ID, first name, last name, email address, phone number, date of user creation.

Other stored data (Images): The images table will take in the following information, image ID, a true or false if the image needs review or not, the image itself, and the user ID of the user who uploaded the image.

Relationship of the two entities: The line between the two boxes represents the relationship between the two entities. In this case the relationship is “for one user there are many images”. The double dash on the left side of the line represents “one”, while the three pronged end of the right side represents “many”. This relationship exists because one user can upload many images.

Implementation Plan

Our plan to implement the pEEP web application can be seen visually in the Gantt chart below. As can be seen, the majority of the foundation structure is being worked on in parallel by each team member.



Database Configuration

Currently, there are only two tables in our prototyped database: one to hold user information, and another to hold image information. To complete the configuration of our database, we will have to create a script that creates the database, defines it's tabe structure and relationships, and also creates privileged admin users (whose credentials will be used by the model to execute queries). The database will certainly be extended beyond this simple schema as our core functionality grows and the need to store certain data becomes more clear.

User Authentication/Profiles

We will need a form of user authentication so that users can create and access their own customized profiles. Additionally, to alleviate users from having to enter their credentials each time they move to a different page on the site, we will have to employ either sessions or JWT tokens to allow some level of persistent authentication, and to truly make our website seamless. Since we have solved the base level problem of integration between all modules, implementing user authentication is the logical next step so that we can provide a personalized experience to our user base.

MVC Separation

Since we have chosen to adopt the MVC design pattern, we will need to split our backend functionality such that logic, routing, and rendering all become as decoupled as possible from each other. To do this, we will port over the logic and computation that is happening in the controller over to the model, and have the controller rely on the data generated by the model. Additionally, we will continue to separate our views from our controller so that we can simply render our views and provide our templating engine with the necessary variables to dynamically generate customized user experiences.

Design Templates

This is the bulk of the UI work, where we will have to design the templates for each page on the website. This will involve experimenting with different pages and different control flows to find the best combination that will make the user experience feel natural. The overall styling of the templates will be left to a later date (though kept in mind during design), since the look and feel of a page is something that can take many, many hours to perfect.

Digital Ocean Deployment

The current prototype uses the development server that ships with the Flask module. While this development server is ideal for rapid prototyping and is configured for concurrency by default, we will need to switch to a more robust web service that has the reliable servers and resources required for web applications to truly scale. Digital Ocean provides said servers and resources, but there is configuration details that will differ and fine details to iron out to successfully transfer our web application from development to production.

AI Model Tuning

The current model has been neglected, and the current dataset is lacking. This is due to the fact that our entire team has been focusing on initializing and integrating all components. While there is currently a model and training pipeline, the accuracy leaves something to be desired, and the training pipeline could be made more concrete. While the model is tuned, the subprograms that make up the training pipeline will also need to be made concrete. This means that the consistent refinement of training data, and the subsequent retraining, should happen in an orderly fashion, and in such a way that does not disrupt the website. These subprograms might require a separate server to truly perform at scale, but this extension is one which can be put off for now. In addition to the training images sourced from users, the models primary dataset will be obtained via open-source datasets currently available.

User Testing

The most important part of any web application. If we don’t do our part to make an application that people actually want to use, we have failed on all accounts. We will implement user testing as specified, either by using real subjects, automated scripts, or a combination of the two to validate the usability of our site.

Conclusion

Our client, David Plemmons, has a vision of both building a new online community of bird watchers, as well as strengthening the existing community of bird watchers. His current project, the pEEp Smart Feeder, allows people to engage in birdwatching without traveling or buying expensive gear. However, it currently lacks a “community” as there is no online source available to let users of the pEEp bird feeder share and interact with one another. Our team has been tasked to build a web application that will provide a simple and interactive user experience, analyze and store user submitted content and images, whilst still maintaining all the desired functionality wanted by the client. To conquer this task we have divided the web application into three main components that will all be working in conjunction with one another to provide the best possible experience. These include the frontend, backend, and Artificial Intelligence model. In summary, the frontend is what the user will see and interact with, the backend will process and store interactions and data, while the Artificial Intelligence model will analyze user submitted images along with a test group to provide feedback on what type of birds users are watching and interacting with.

Currently, our group has laid the groundwork for the web application. We have a prototype that connects the front end with the backend as well as the database. This was done so that our group knows how everything will fit together and interact with each other. Next, we will be expanding on the prototype, implementing custom frontend pages to make the web application more unique and closer to David’s vision. We’ll also design the backend so the frontend pages will be able to have the functionality required, such as creating user accounts, uploading images, and participating in user forums. Lastly we will continue to test the AI model making sure that it provides the best possible results and slowly transitioning into the analysis of user submitted data and finally linking everything together.

Once we have solved all integration issues, we can move on to both optimization of individual modules as well as application deployment. Optimizations will take the form of UI tweaks, AI model tuning, as well as general backend maintenance and logic optimizations. Deployment will be handled using DigitalOcean, so some additional research will be required to handle server configuration, as well as linking a domain name to our deployment server. Through integration, optimization and product deployment, we will deliver on our requirements to create a website that bird enthusiasts truly love using.