Final Report for Spring Semester 2022

Ryan Lockard and Hasha Drabrek

Art Museum App with Database Services for the Museum Industry:
User Experience Through Art Discovery and Ease of Database Population for Museums

Table of Contents

1 Introduction

1.1 Team Introduction

Hasha Drabek is the report/project manager. Her concentration is Nanoengineering, with research in fabrication and fabrication interfaces. She also has experience in web design, business management, and front end user experience.

Ryan Lockard is the system manager. His concentration is Bioengineering, with research experience in brain computer interfaces and machine learning. Through his internship experience, he has gained experience in designing and developing python applications and managing SQL databases.

Hannah Jiao is the project sponsor. She is a graduate student studying human computer interaction and has a lot of experience in front end development and user experience.

1.2 Time Management

The team has committed 10 hours per week to working on the project outside of class. The team will work on the project during normal lecture hours when allowed, and will stay beyond lecture time to work together in person.

The team will meet both in person and by zoom as necessary, with in person meetings with our project sponsor every other week.

The team will share labs, presentations and reports via Google Docs or Google Slides so that collaboration happens instantly and content is accessible easily. Code is shared via Github. The team also emails and messages regularly to manage meeting times and lab, presentation, and documentation progress.

1.3 Resource Management

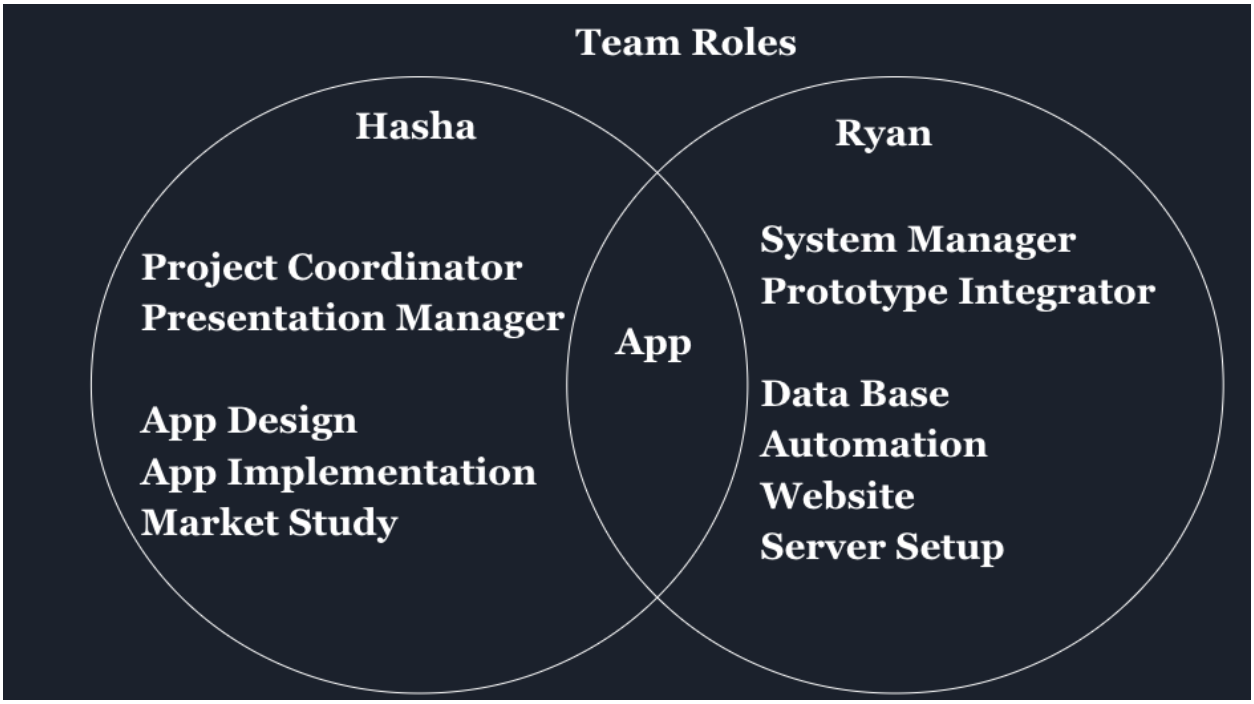


Fig. A visual representation of the breakdown of the team roles and allocation of tasks

Hasha is responsible for the front end development of the application, as well as backend integration, such as connecting databases that house the content that populates the app. She also conducted the market research interviews and market study. She is also responsible for managing the presentations, labs, and final report. She coordinates between the team and the mentor, and makes finishing touches on any content that is integrated from the team into one final product.

Ryan is responsible for managing the website front end and website back end development, including things like hosting and creating the html website itself. He is also responsible for building the software that allows museum employees with little technical knowledge to populate the website with content, this includes things such as dynamic creation of html pages.

1.4 Gantt Charts

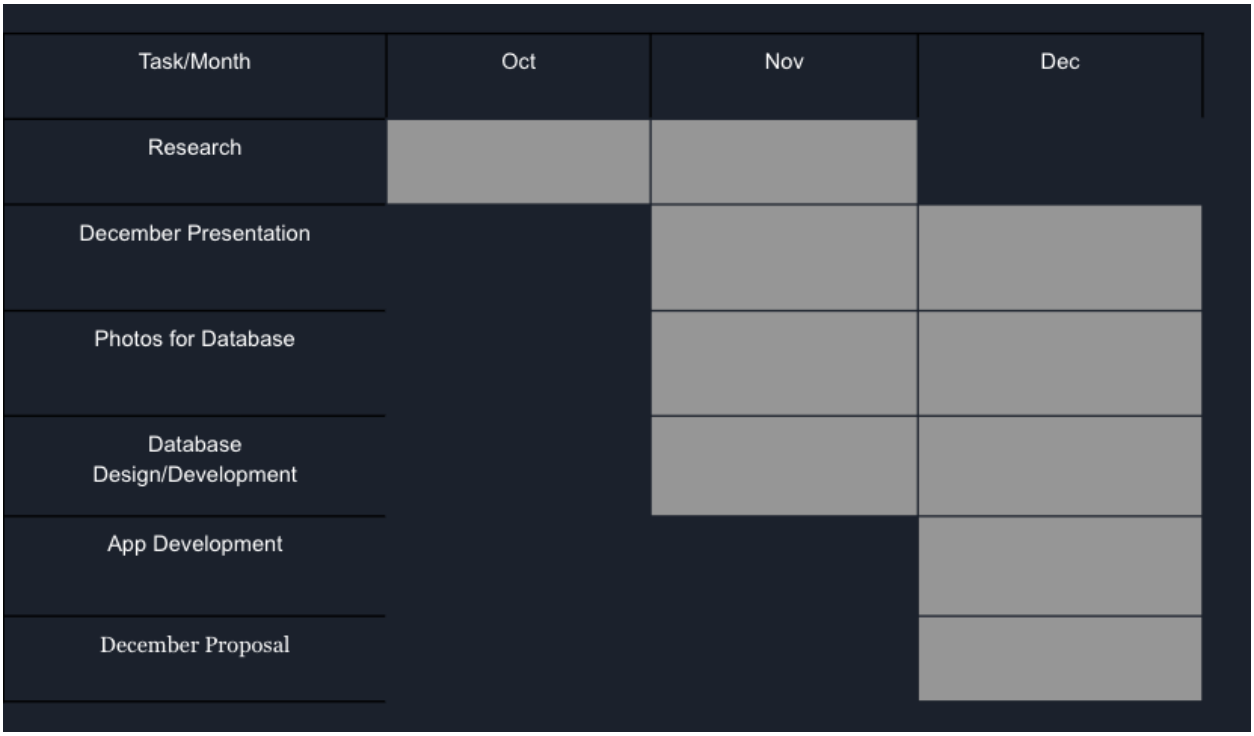


Fig. Fall Gantt Chart.

This was the scheduling we followed for the fall semester. It reflects the AI design we attempted to implement. By the end of December, we had major parts of the user interface completed, and we also had an AI model that had varying levels of success (accuracy of a few percent up to accuracy in the high 90's). Because of inconsistencies in results, we decided to move to a more solid and predictable QR model and refocus our product around databasing services that we discovered is something many museums need.

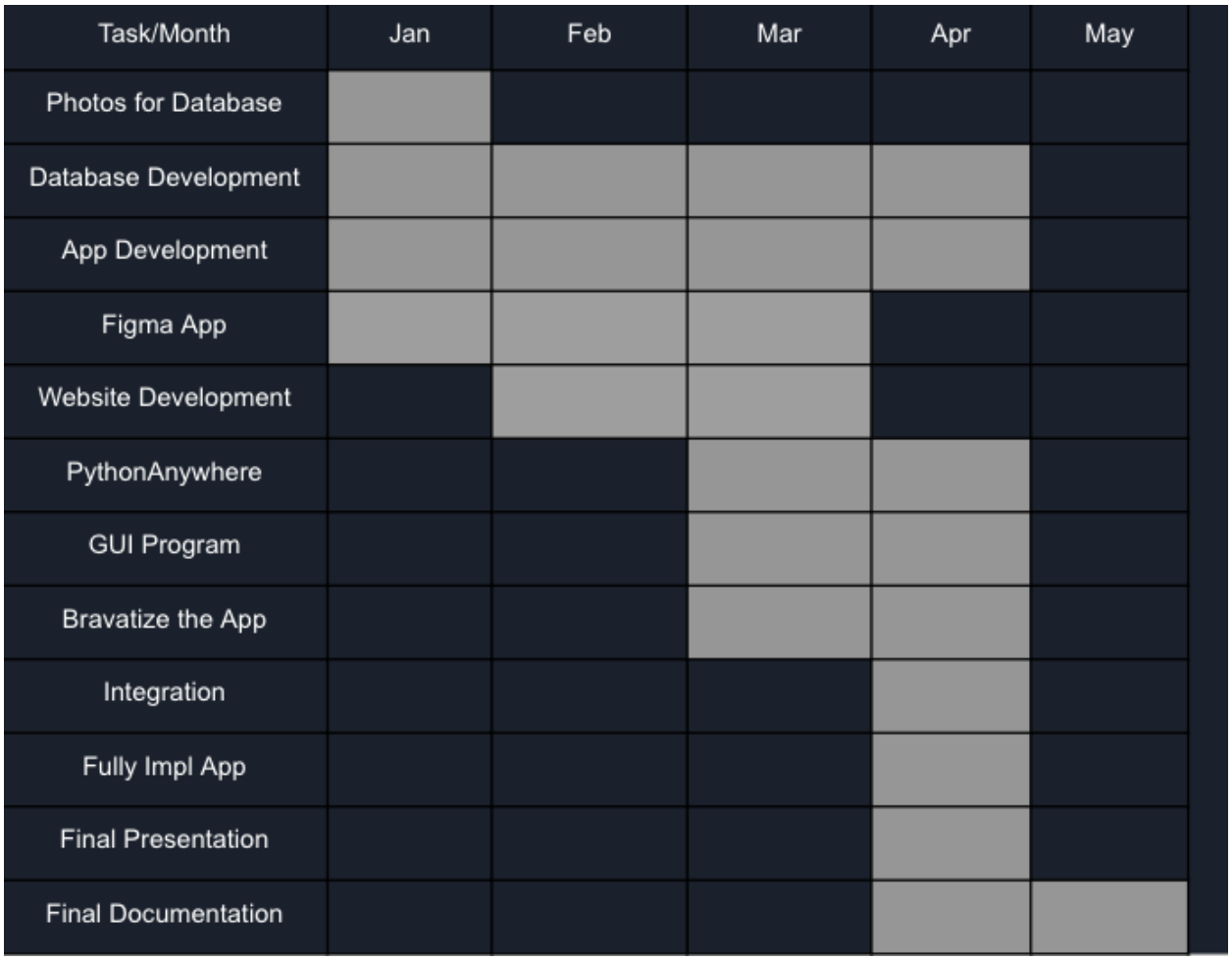


Fig. Spring Gantt chart.

For the spring, the app build faced backend development, and we needed to face the new data-basing services completely. Since Ryan had a lot of experience working with databases in the past, we did not need to learn how to work with databases, so we just started with populating the database with content that we got from the Eskenazi Museum of Art's website. Ryan then moved to developing a website to host the data, and a GUI for museum employees. Hasha spent much of the 2nd semester working on the backend of the app with Bravo Studio, as well as the presentations and documentation.

2 Project Introduction

2.1 Project Description

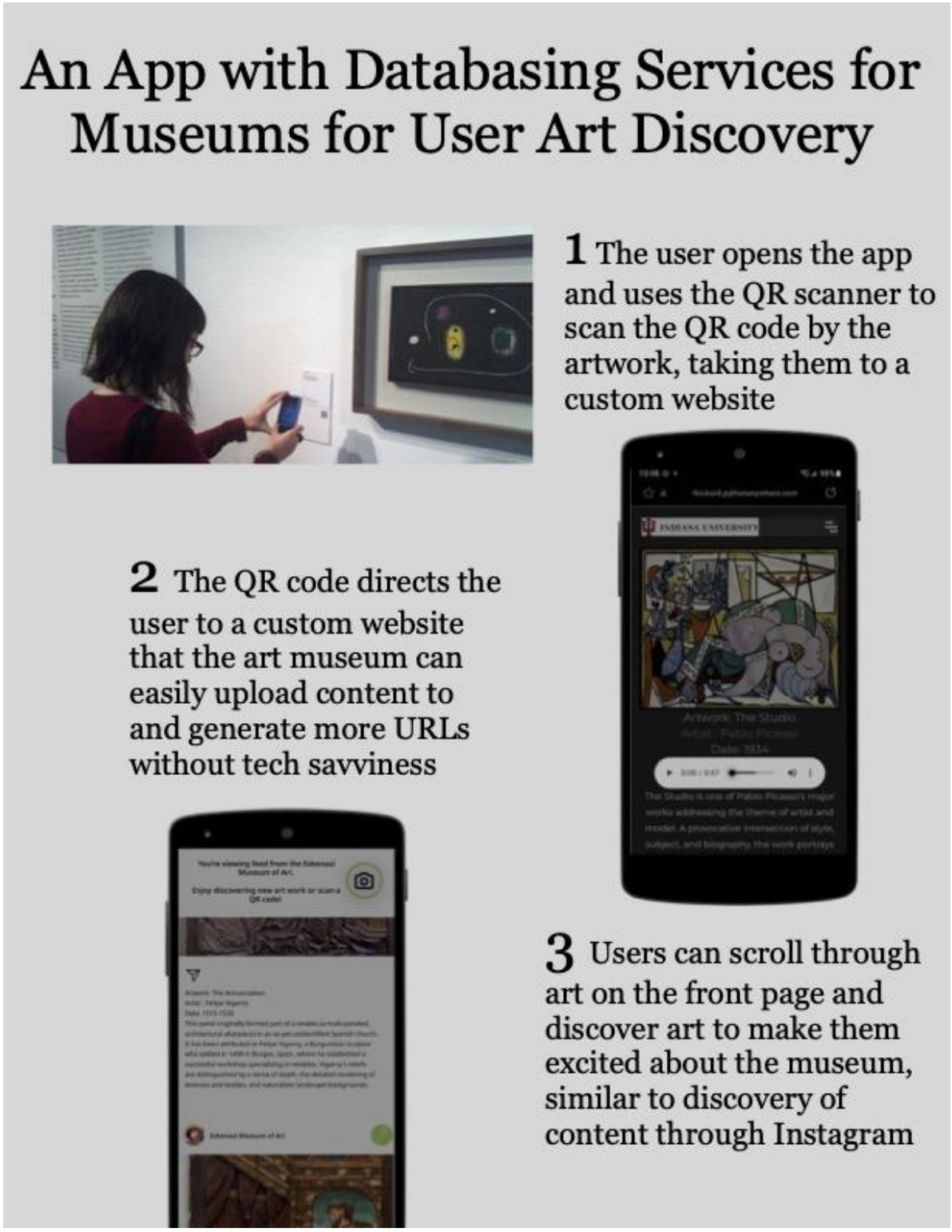


Fig. Information poster about our project. For the purpose of quick information relay about our project for market feedback and lead generation.

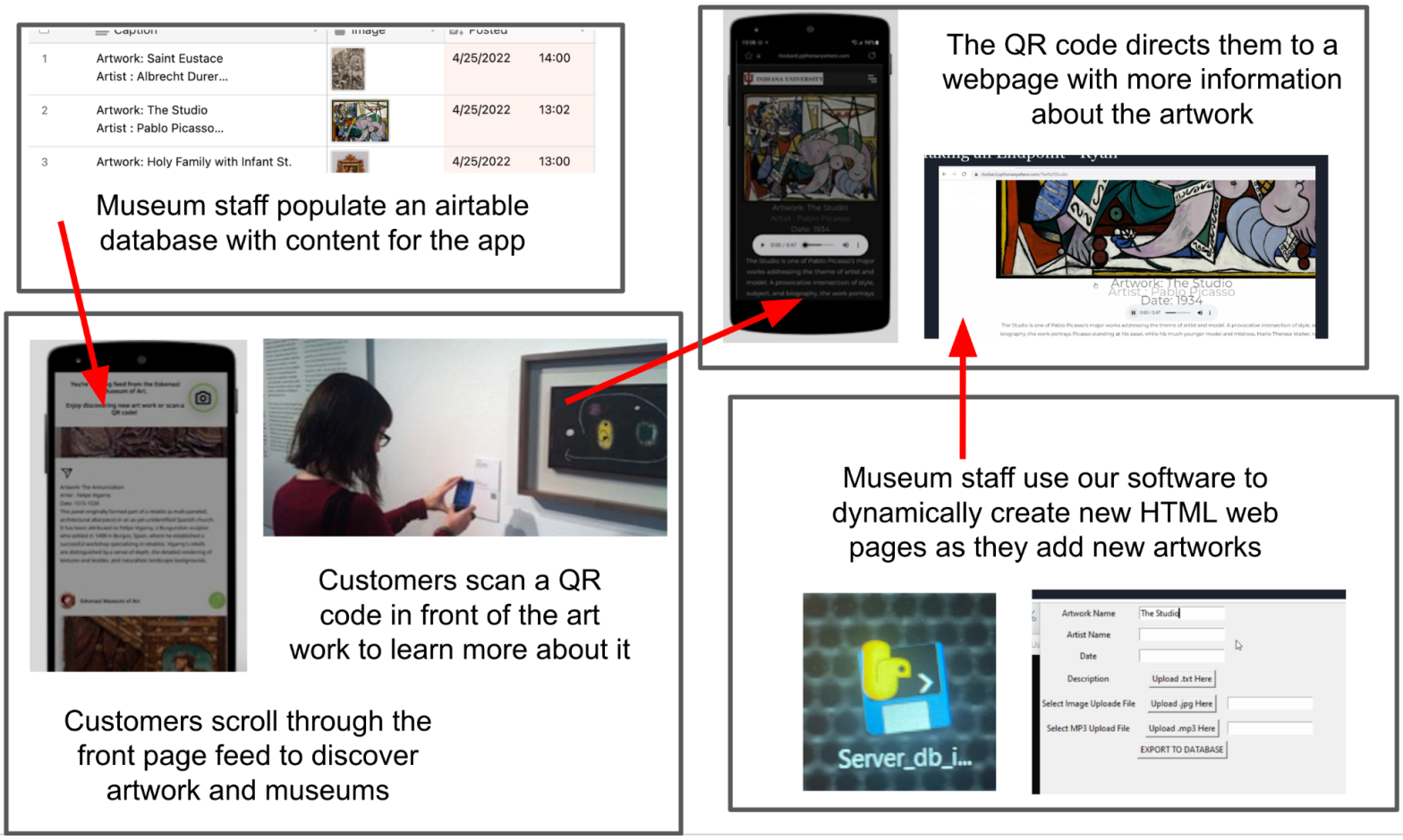


Fig. A slightly more in depth perspective on the types of services and products we made as our project.

Our project is to provide database services for the museum industry that can be used by them and turned into customer discovery, customer retention, increase general customer satisfaction, increased customer knowledge of art, and to generally increase customer excitement about art through an interaction with artwork via a mobile application. The app features a login system for tracking, and a feed of artworks and information about each artwork that is populated from a database the museum can easily populate with content themselves. Our product also features a more B2B style data-basing services, with a software built for museums to dynamically create HTML websites which customers can then use the app to scan QR codes while inside the museum to be directed to the HTML websites that contain more information about each artwork.

Due to the results from the first semester working towards an AI solution, we decided to transition to a more solid QR code solution. This solution must appeal to two separate entities:

business, and museum goers. In that way, we are working on a business to business model, as well as an end user model. Let us first discuss the end user model aspect to our project.

The end user portion of our project is the mobile application. The mobile application features an art discovery element that we kept from the first semester and was something our sponsor, Hannah, felt was very important to the project. This includes an Instagram style content discovery feed, as well as a QR scanner to be able to access the HTML websites we built, as well as additional information about the museum, such as location (map). The QR code scanner is included so that we can tie the 2nd part of our project together.

The QR scanner is intended to be used to access the HTML websites that have more information about each artwork. The HTML website pages can be made by museum staff using our software which dynamically creates new HTML pages as the staff index new artworks into website pages. QR codes can then be created for the HTML pages and put up inside the museum next to each artwork for the customer to scan as they stand in front of the artwork.

2.2 Competitors

There are no direct competitors with our project idea. One similar company to ours is a company called Daily Art. Daily Art is a mobile application that provides its users a picture of the day. The only real connection between our idea and Daily Art is that both are mobile applications that have something to do with art. But the art is not museum specific, and there is no additional information about how to access that art in person at a museum, which is a key element of our app.

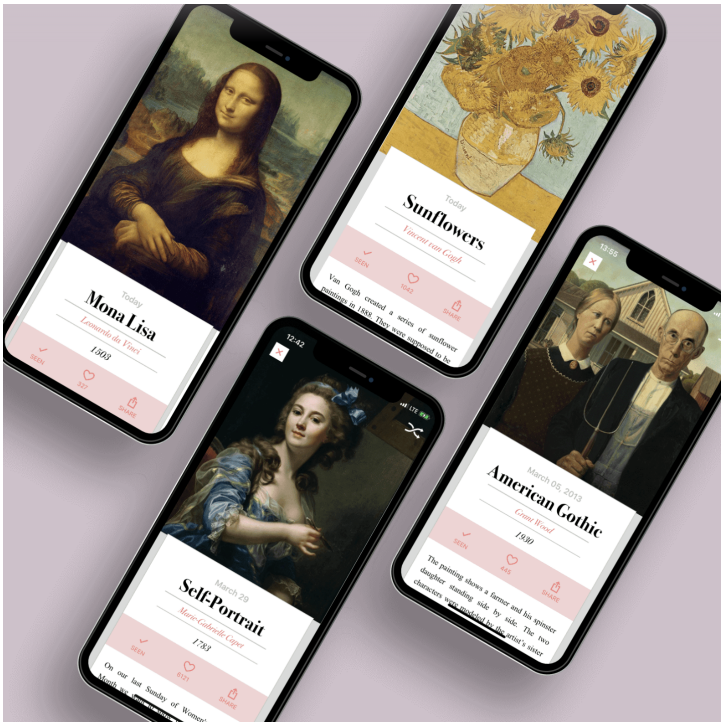


Fig. Examples of Daily Art homepages.

The second kind of competitor we found to our project idea is at the Cleveland Museum of Art called Gallery One. Gallery One is a one of a kind virtual experience that was part of a 350 million dollar renovation that provides its users with a virtual tour of the Cleveland Museum of Art where museum goers can interact with the pieces that interest them. Although our project idea is pretty similar to Gallery One, the main difference is that everything in Gallery One is a virtual experience on an ipad whereas our project is a way to interact with different pieces of art interactively while actually walking around the museum.

Fig. Gallery One, Cleveland Museum of Art.

2.3 Lead Generation, Interviews with Industry Experts

Over the course of 2 months, we interviewed 4 different industry professionals: Melody Barnett, PhD, who is an expert in the museum industry, Maria Domene-Danés, PhD, who is an art interpretation expert, Margaret Graves, PhD, who is a museum industry and interpretation expert, and Faye R. Gleisser, PhD, who is a museum curation expert. We interviewed them ranging from 30 minutes to 2 hours.

Out of the interviews, some key points emerged that we incorporated into our design objectives and business plan. First, our app can provide unique data analytics for museums. Our app can

provide engagement analytics on specific pieces at a museum that the museum can then take those analytics and use them in their own grant applications. This point solidified that the monetization of our product will not come from the individual consumers, but from museums (B2B). Museums have an incentive to buy into our product because they can then go take the analytics and apply for their own funding.

Another key thing to come out of the interviews is the database services for small museums. A centralized database would allow one museum to advertise themselves to a user who is using the app for a different museum. The museum the user is not familiar with can come up in similar search results or recommendations, thus making the user aware about the connections between pieces of art across different museums and perhaps making them more inclined to visit that new museum they just became exposed to.

Another key thing to come out of the interviews is the ability to give a voice to controversial objects. It is not easy to move Civil war monuments, but they already create controversial dialogue on other forms of media. Our app would be able to provide a voice to these monuments and similar sculptures outside of a museum setting where there is no commentary present, let alone the commentary of many voices which our product can provide a platform for.

Another key thing to come out of the interviews is that the average user will probably not be an art historian or someone already very active in the art museum community, but rather, a person who does not regularly search out information on art. This app provides a safe place for the novice art enthusiast to engage and participate in art commentary. It is well documented that people behave strangely in museums, and are hesitant to engage with curators or their resources. Our app could provide a quiet place to engage with the art where experts and curators could answer questions about the art asked by novices on the app. Adding this type of feature would be a great addition to the app.

Another key thing to come out of the interviews is that many voices can be heard in one place. Similar to giving a voice to controversial objects mentioned above, our app can also provide many voices on one piece inside a museum. Traditional curation is limited to 200 words curated by a limited number of individuals, or even one individual. Users can select pieces and contribute to a growing dialogue about the pieces, bringing much more than the 200 word paragraph can provide. Our app provides an update to the paragraphs seen beside works in museums, and brings the technology used in museums, such as the isolating earbud audio experiences many museums implemented in the mid 2000s into the 2020’s.

3 Objectives [priority]

Phone Application

- Camera integration (resort to picture upload) [3]

- Easy to Use [1]
- Compatible with IOS and android devices [4]
- Extendable for different museums [2]

Database Server Interface

- Easy for a non-technical person to use

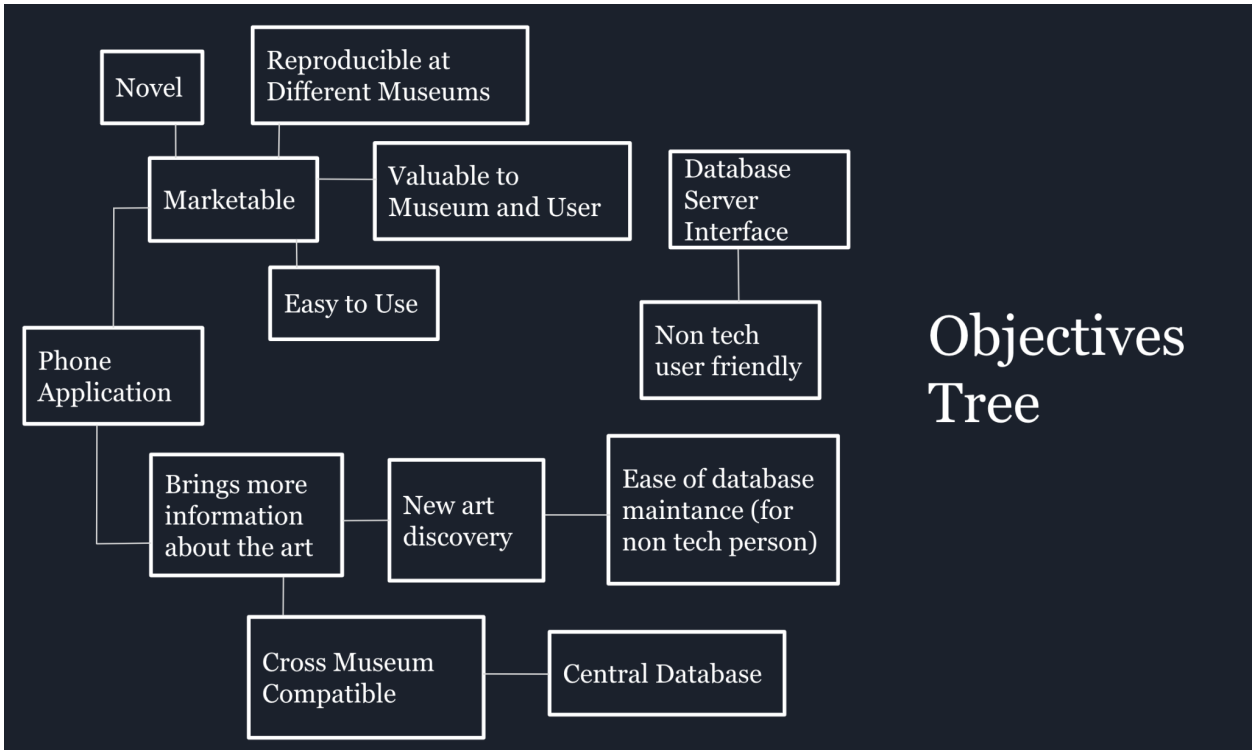


Fig. A visual representation of how our objectives are interconnected.

4 Constraints

Phone Application

- App size less than 50 MB (averages: 38MB for IOS, 15MB for Android)
- Follow Laws about user data
- Safe and secure download

Database Server Interface

- 74,000 Kb of disc space (this is the size of our database html application)
- Windows 8 machine or newer (this also is needed to accommodate the application size)
- Non tech user friendly

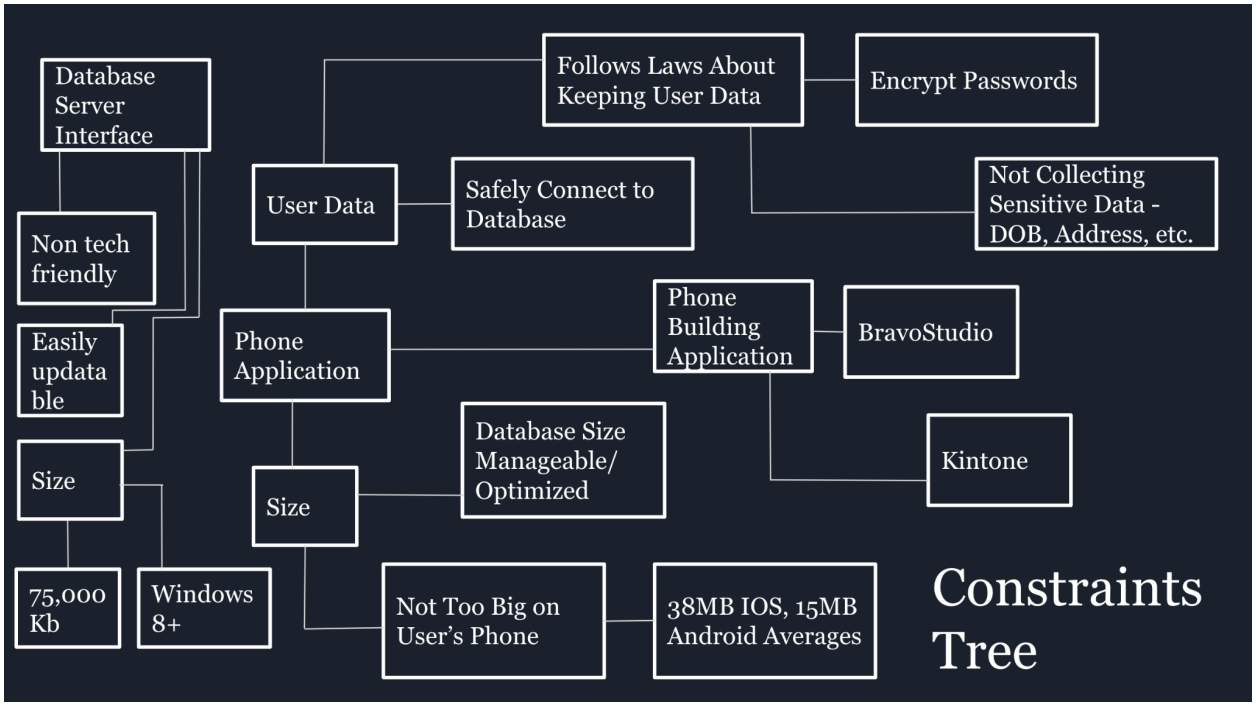


Fig. A visual representation of how our constraints are interconnected.

5 Sustainability

The major sustainability concerns with our product are the power consumption of the software. There is a very minimal physical component: the printed QR codes that are put up next to each artwork. Once these are made, they should be able to be used as long as the artwork is up for display and the website it links to is also up. The QR code can either be printed paper, which is very sustainable and completely biodegradable, or it can be made of a plastic like PVC, which is entirely recyclable.

Since the majority of our product is software, the main concern is the power consumption. To mitigate this, monitoring power consumption as the code is written can be done. Such as using Intel's Software Development Assistant, which allows designers to measure power consumption as they are developing. These can be used when developing the automation software as well as developing the website and database setup.

For the app design, we are using tools and softwares that already exist for the purpose of UI design and implementation: Figma and Bravo Studio. It is assumed that these companies already practice sustainable software practices, and it is something that is not monitorable as we design with them.

We also must be conscious of how much space on the phone the mobile application takes up how much space on the computer the automation software takes up. We must be conscious of the end user's device storage and power usage. To mitigate this, we monitor the size of the applications as we build them. Bravo Studios has application size tracking, and we are staying well under the normal range of application size.

6 Financial Considerations

—	Description	Requirement	Notes
B1	Intended Market Geography	North America, EU. Or other English Speaking Museum Entaties, pre-approve	
B2	Intended Market Demography	B2B aspect: Small to medium sized museums in need of database services and/or increasing public interaction Public aspect: Any museum goer with a smart phone (apple or android)	
B3	Ave List Sales Price - ASP	B2B: $15K per 1,000 pieces (?), ongoing data hosting fees Public aspect: free download with no premium features	
B4	Estimated Annual Volume	# 1 one new contract per month, 1 month turn around	# of units/year

B5	Max Material Cost (Parts)	$ 5000 one time camera costs
		$ X server costs (?)
		$ ongoing labor costs, per diem when traveling to museums
		$ ongoing copywriting costs (might be free if user generated)
			$20/hr, it takes about 1 hour per 100 pieces, about $2000 labor costs for salary and per diem for 1 week for 15,000 pieces
			AI computing services will be much more

B6	Max Assembly & Test Cost	$ N/A	Must be less than ASP
B7	Product Life Target	Evaluate need for updates for app every 6 months or sooner if apparent	Will be adjusted after reliability evaluation
B8	Target Warranty Length	Charge museums for updated database services they request, open warranty for 3 months post contract	Will be adjusted after reliability evaluation
B9	Min Life Cycle Period	Support as long as business is profitable and museum is paying data hosting fees	Will be adjusted after reliability evaluation

Fig. Visualization of financial considerations of our business model.

Since this project is set up as a software project, there is no physical product to be manufactured. The costs come in the form of software development and upkeep of the software. Since we are building the software as part of a class, and no museum or company is paying us for this, this has no cost associated with its development.

Once the app and database need to be hosted at scale, costs begin to become a thought. For example, Amazon Web Services service for hosting apps, costs about $65 a month for an app with 10,000 active users, 2 updates per month, and a size of 100MB. This is a reasonable estimate for an app like ours at scale.

Profitability comes in the form of charging museums for our services, which would include labor charges and hosting charges. We would need to train museum employees on how to use our website software. Even if this is not necessarily necessary from a documentation standpoint, it is our understanding that it is still a good idea to develop the business relationship by demonstrating the services and 'training' in person with the museum professionals. This would be a one time charge of a few thousand dollars for each museum. A museum would then have internal fees to populate the database by themselves, and they would have minimal ongoing charges for app and website maintenance, as well as the hosting costs. This is on the order of magnitude of $1XX-$1XXX per month per museum (based on AWS), depending on how populous their database is and how active their users are. Similarly, hosting of the html site will run about $1X, with specific estimates around $40 (AWS).

7 Ethical Concerns

- Proper Citation of Artwork/Artist.
- To mitigate this, we will specifically cite each work and its associated institution property, make sure that the art we include is able to even be reproduced on our application (not copyrighted or permission explicitly denied).
- Applicable standards are the 17 U.S. Code 106A, a law that protects claim to work.

8 Safety Concerns and Regulations

- Overheating device battery (Deployed app has no infinite loops/bugs that could damage the device).
- To mitigate this we will test code thoroughly especially for things like infinite bugs, we will also test length usage, like leaving the app open for a long period of time. We are also not coding directly in something like Java, we are using tools like Figma and Bravo Studio that we trust have already implemented tests against things like infinite bugs, and are running themselves optimally.
- Maintaining and protecting user data
- We will have to make sure data is stored on secured servers and the stored data is encrypted. Specifically, we are using Firebase's login services, which is part of Google's flagship app building service, for our login data management. This data management service is very sensitive to following laws and regulations and encrypting user login data, and we trust Google's service for this.
- The applicable standards are the 15 U.S. Code 41 et seq. law that protects user data, and privacy laws about user data, and the 18 U.S. Code 2721 et seq. law that governs privacy

and disclosure of gathered personal information. The California consumer Privacy Act of 2018, which says all private data must be encrypted and at rest in encryption, and that security must be ‘reasonable’ to protect this data. The Federal Information Processing Standards, mandated by NIST that also regulates the encryption and collection of user data. Specifically we are mitigating this by using Firebase, which is FIPS 140-3 certified. In Europe, the GDPR, general data protection regulation, manages regulation of user data. We again trust Firebase to be compliant with this regulation, we are trusting the name of Google to have made sure they are compliant with most major world user data regulations.

9 Detailed Design - Mobile Application

9.1 Flow of Execution of Using the Mobile Application

The main flow of execution is that a user will first open up the mobile application. The user will make a login profile that will then be stored in the user database. The purpose for the login profile is for data tracking, as well as for future expansions of the app, such as comment and chat components. The homepage of the app is an infinite scroller, very similar to instagram. The content is populated into the home feed via museum employees uploading image and text content to an airtable database, then the images and text are paired and sent to the user’s homepage.

If they are additionally interested in the artwork, they can select the museum which is appearing as the ‘poster’ of the content, which takes them to a page where they can see map location data about where the museum is located and where they can see the artwork. If they are already in person at the museum, the application also features a QR scanner, so when the user is in the museum and in front of a QR code, they are able to scan the QR code and be taken to the html web page which houses additional content and information about the artwork they are standing in front of.

The only way to access the html web page is via the QR code, or directly typing in the html address. This is strategic to get the user to use the app in order to interact with the additional content.

9.2 Glass Box - Application

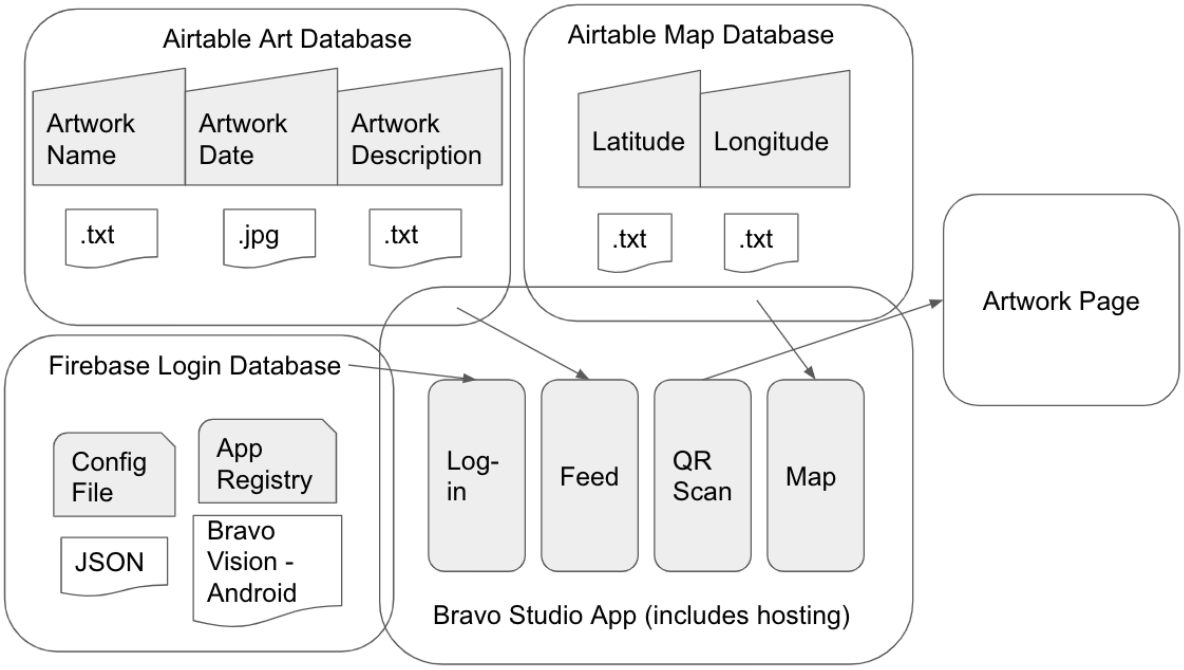


Fig. Glass Box diagram of the mobile application.The glass box shows how the implemented aspects of the app were accomplished.

9.3 Figma

Figma is a graphic editor and prototyping tool which we are using for the purpose of application design. Figma has features such as creating containers, which is an abstraction of the app layer that packages code and dependent components together. These containers can then be bound to actual data, such as API data, in the later development process. Figma also has the ability to extract Apple and Android compatible code, even though we did not use this feature, we think it is important to include this detail here for potential future development. Instead of directly extracting something like the Java source code, we used Bravo Studio, which Figma has a close relationship with. This includes things like being able to add Bravo Studio Tags, which are a text string placed in the layer name of a Figma file. After importing the Figma file into Bravo Studio, the layer with the Tag will turn into a mobile component, or action, in Bravo Vision, which is Bravo Studio's own self hosting service for the purpose of seeing the app fully executed before placing the app on the app store.

We found Figma is comparable to an Adobe design suite, such as Photoshop, in terms of strength of execution/features and learning curve. It is a powerful application design tool, and almost any feature can be designed with it (even if that feature is not executable).

For actually implementing the features of the app, we used Bravo Studio.

https://www.figma.com/

9.4 Bravo Studio

Bravo Studio is a self proclaimed ‘no-code’ mobile app builder, which has extraction to both iOS and Android app stores. While the term ‘no-code’ may seem appealing, there is still a learning curve similar to that of an Adobe suite, like Photoshop, and careful connections, such as API addresses must be taken care of.

The purpose of Bravo is to take the Figma design and turn it into an app with working features that will become real features if exported to a real app store, such as iOS or Android. While Figma may look like a real app, it is really just as implementable as a paint document is. Bravo Studio is necessary to turn the Figma design into a mobile component or action.

https://www.bravostudio.app/

9.5 Figma Design

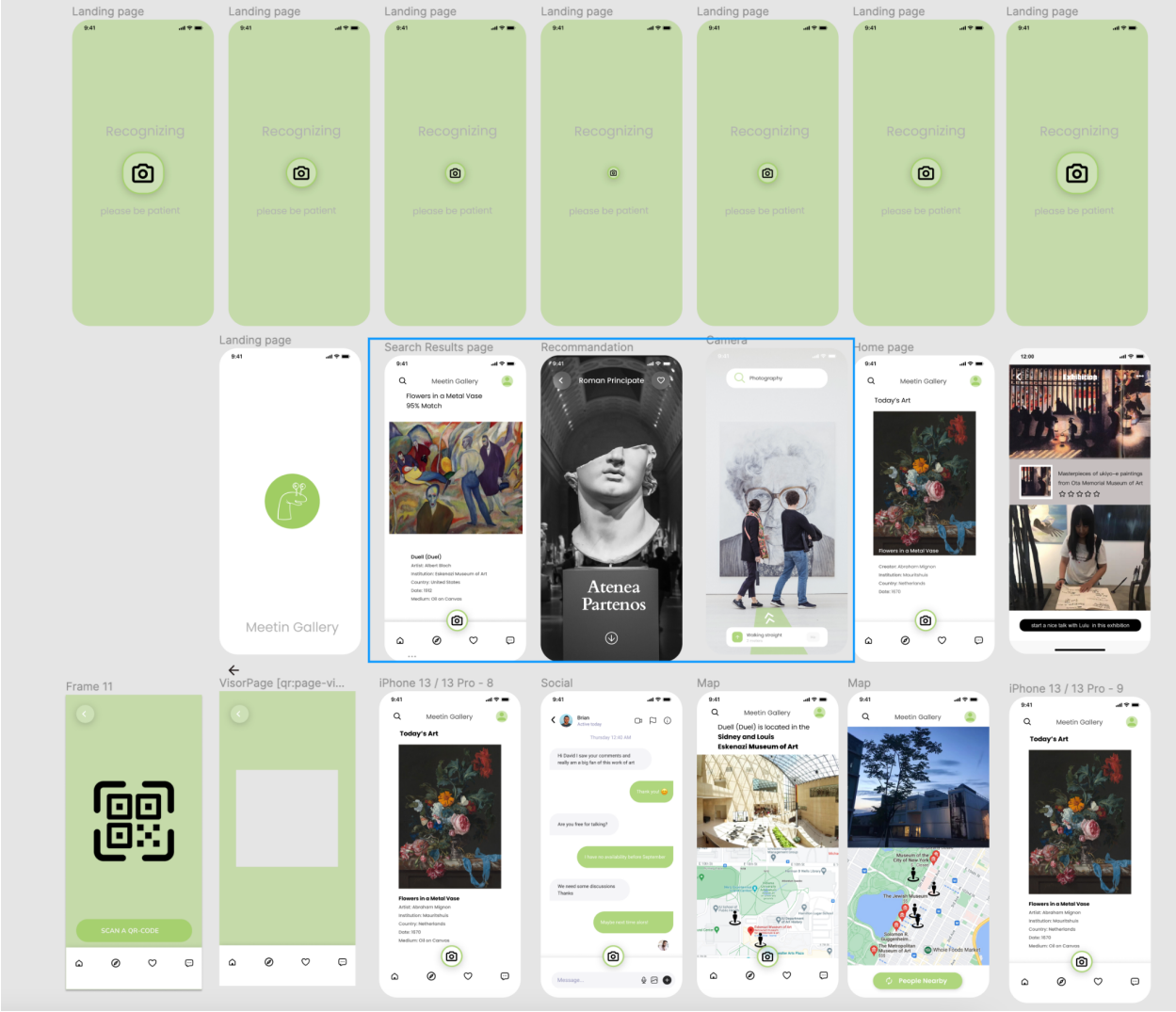


Fig. Initial Figma designs.

We initially developed many Figma designs, with ideas of features like comments, chat, map, location of friends, QR scanner, and home screen feed. We ended up settling on implementing the map, home screen feed, and QR scanner. Future development can lead itself to the comment and chat features.

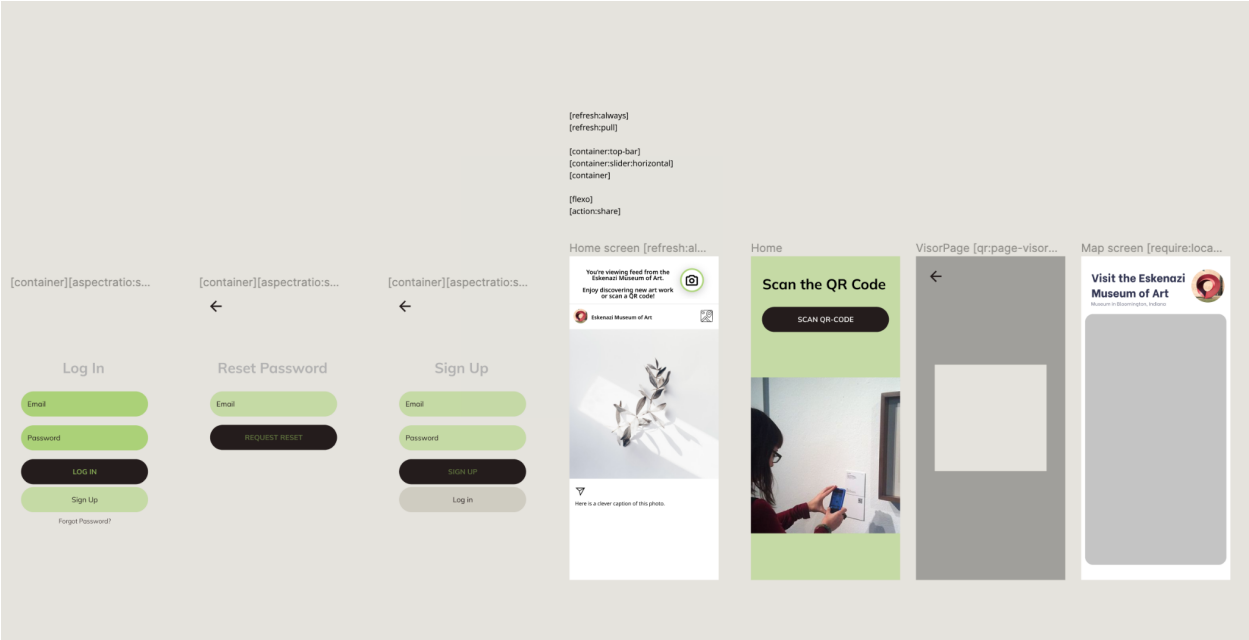
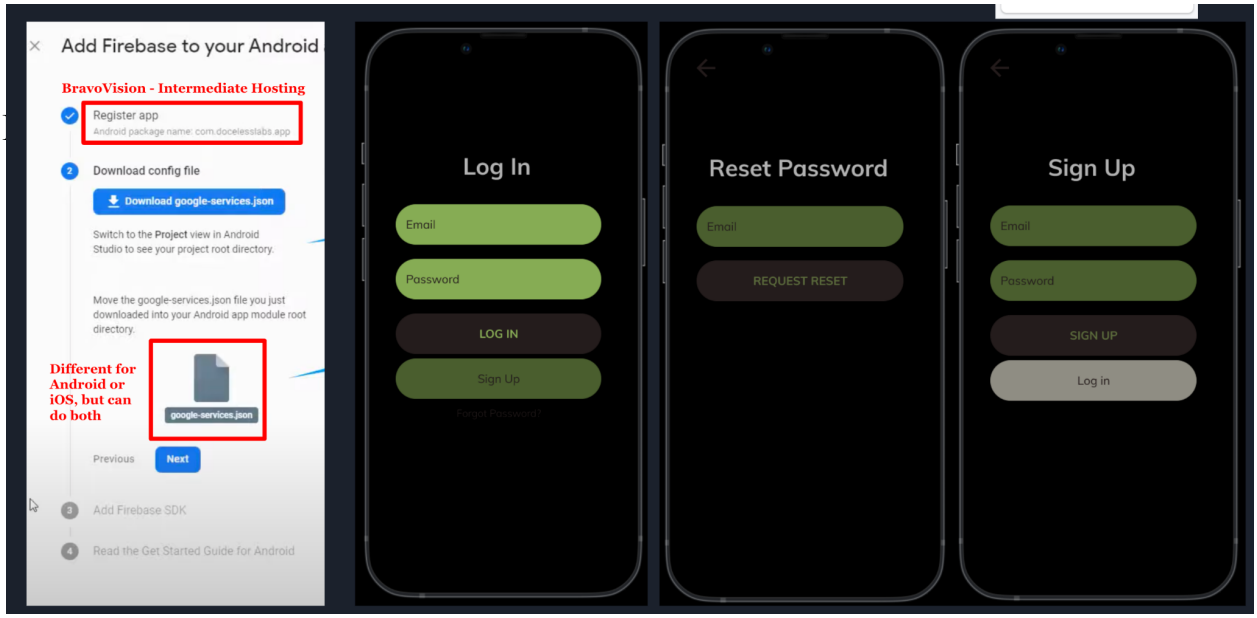


Fig. Figma pages that we imported into Bravo Studio to bind actual data elements to.

Above are the implementable pages including the login system, Instagram like infinite scroller home page, QR code scanner, and map location of the posting museum. Let us take a deeper look into each of these features.

9.6 Login System

The login system was implemented with Firebase, a login service from Google's flagship app building service. We chose them because they are compatible with Bravo Studio, and also provide the security and safety features that are necessary to follow regulations.



First make a Firebase account and have your design exported from Figma into Bravo Studio. Have the appropriate containers and tags for implementing a login system, as well as the correct prototyping navigations in the Figma file, details on this particular feature implementation can be found here:

Key elements of this implementation are having the app ‘registered’ which is ok to just have hosting on Bravo Vision, choose either Android or iOS (you can go back and set it up for both later, but we have just set it up for Android), download the Google Service JSON file, and configure it inside Bravo Studio.

A detailed explanation of this can be found here:
https://docs.bravostudio.app/integrations/user-authentication/app-login-email-and-password

9.7 Homepage Infinite Scroller

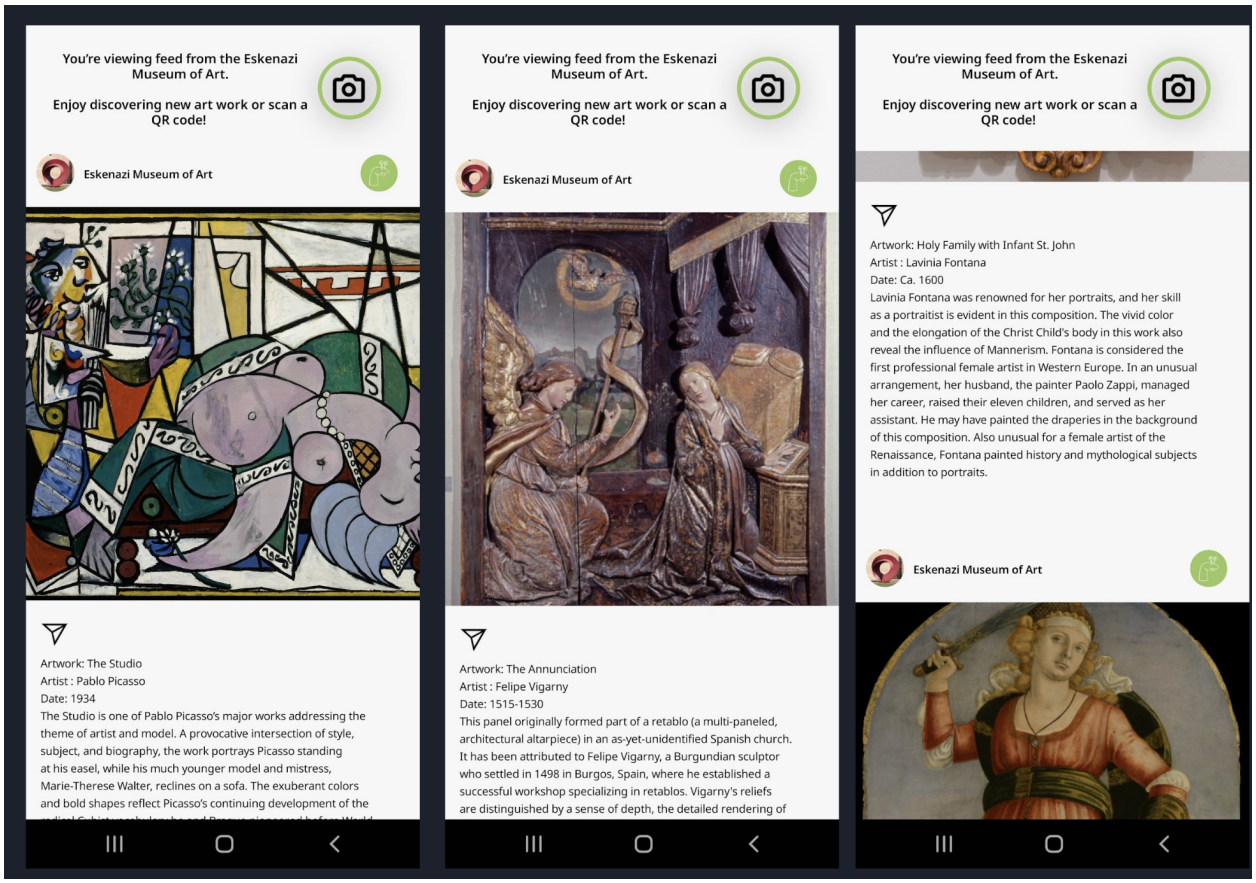
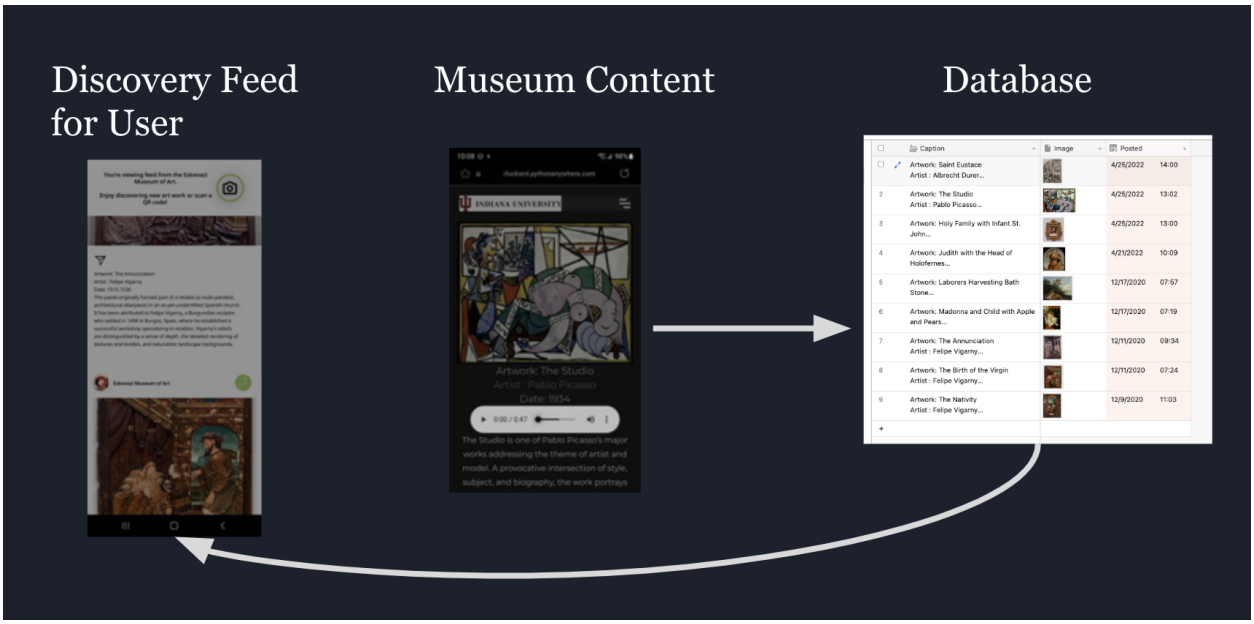


Fig. Example screenshots of the instagram style infinite scroller homepage.

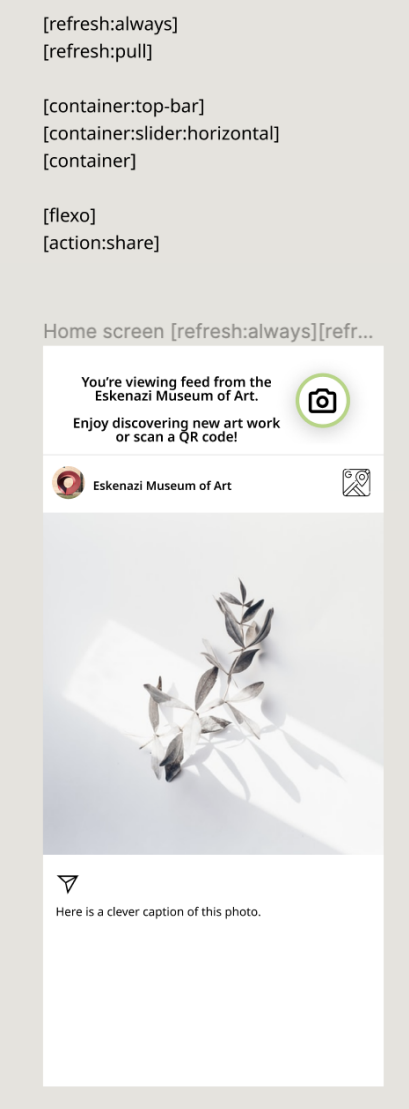
The homepage takes inspiration from Instagram's infinite scroller feature. This is to promote art discovery, which was a core element of our project throughout. We want users to be able to discover art and be excited about art they discover. This scroller feature allows users to discover art on their own, and if they are more interested in that art, they are able to explore the museum where the art is located, and find it on a map. This feature uses an Airtable database to house the content that populates the home screen feed. The museum populates the database with content, and the database is connected to the app via an API. Once the containers with correct tags are set up in the Figma file, specifically: [refresh:always] and [refresh:pull] and are attached to a container in a 2nd layer of another larger container, they can be bound to the API data. The idea of this is seen here:



A detailed tutorial using the necessary steps to setup the Airtable database and API databinding can be found here: https://docs.bravostudio.app/resources/app-cases/bravo-app-kit-foodgram

And we specifically recommend this video tutorial: https://www.youtube.com/watch?v=VM5ipS1uLvM

The container setup in Figma before the API data binding happens in Bravo Studio looks like this:



There is a container for the whole page, and containers for the image and text which will be bound to the data in the Airtable database. The Airtable database looks like this:

	Caption	Image	Posted	
1	Artwork: Saint Eustace Artist : Albrecht Durer...		4/25/2022	14:00
2	Artwork: The Studio Artist : Pablo Picasso...		4/25/2022	13:02
3	Artwork: Holy Family with Infant St. John...		4/25/2022	13:00
4	Artwork: Judith with the Head of Holofernes...		4/21/2022	10:09
5	Artwork: Laborers Harvesting Bath Stone...		12/17/2020	07:57
6	Artwork: Madonna and Child with Apple and Pears...		12/17/2020	07:19
7	Artwork: The Annunciation Artist : Felipe Vigarny...		12/11/2020	09:34
8	Artwork: The Birth of the Virgin Artist : Felipe Vigarny...		12/11/2020	07:24
9	Artwork: The Nativity Artist : Felipe Vigarny...		12/9/2020	11:03
+				

There is specifically text data and image data. Where text is input at .txt, and images are input at .jpg. These two elements can be bound to elements in Bravo Studio so that the elements in the database will populate the location of the bound elements in the app. An implementation with an Airtable database is detailed in the previous two links.

You will specifically need the API documentation to list records: https://airtable.com/appZ4qrafApCxzRNr/api/docs#curl/table:posts:list

The example API url is what you will use when binding data inside Bravo Studio. This is detailed here: https://docs.bravostudio.app/resources/app-cases/bravo-app-kit-foodgram

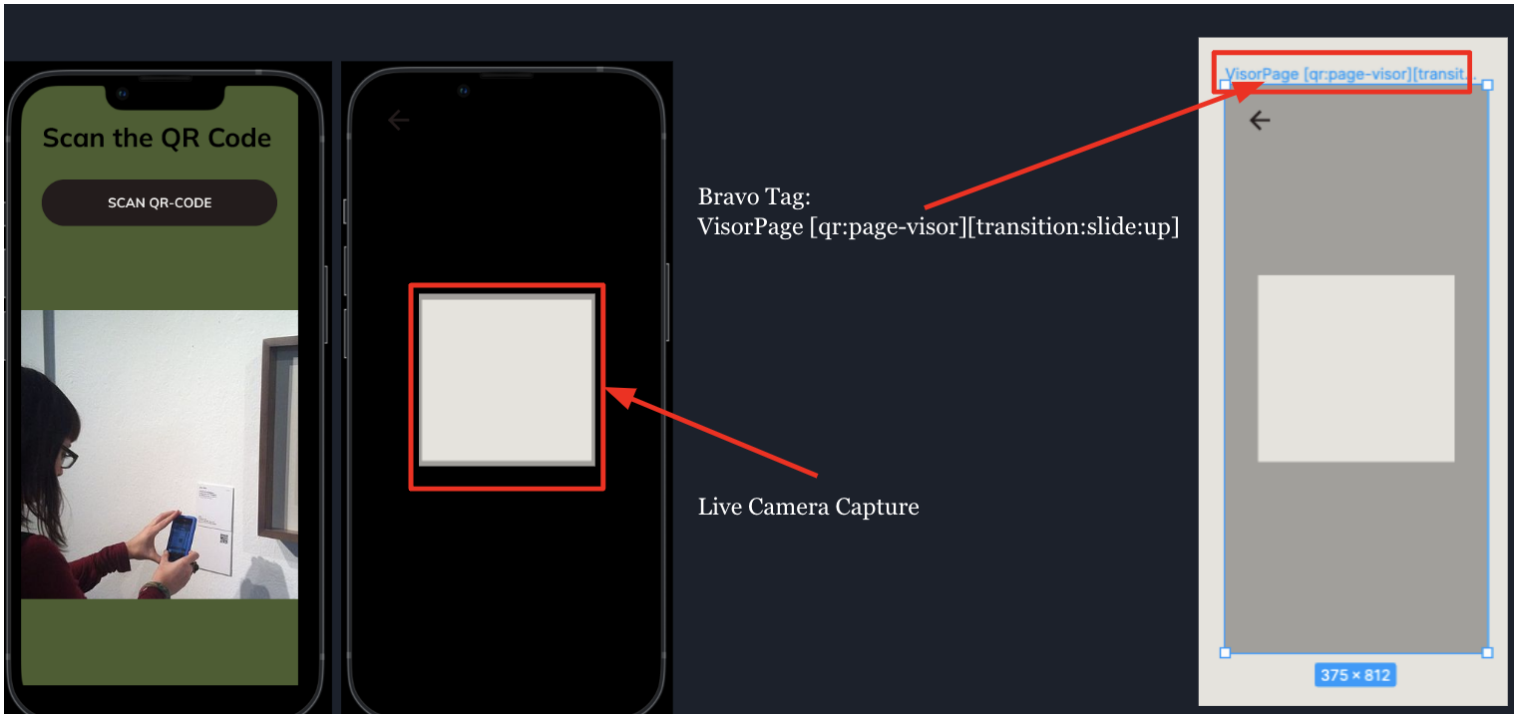
and: https://www.youtube.com/watch?v=VM5ipS1uLvM

9.8 QR Scanner

The QR scanner is a simple package offered by Bravo Studio for use in Figma through the use of a tag. The tag is specifically:VisorPage [qr:page-visor][transition:slide:up], which is connected to a container that is inside another container (such as a rectangle placed inside the more global container that houses the whole app page design). Further information on how to implement the QR scanner can be found here:
https://docs.bravostudio.app/bravo-tags/mobile-actions/scan-qr-code

or you can specifically duplicate this Figma plugin here:
https://www.figma.com/community/file/908302499836222317

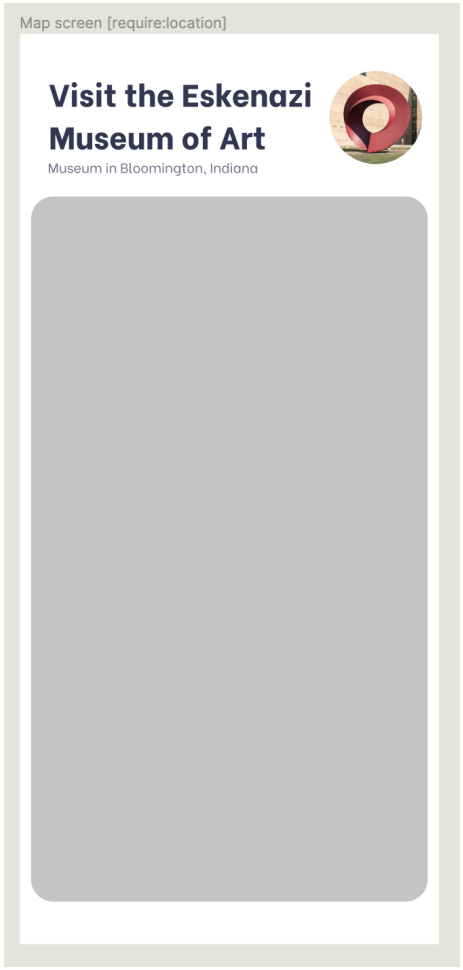
And alter design elements while maintaining the tags and containers.

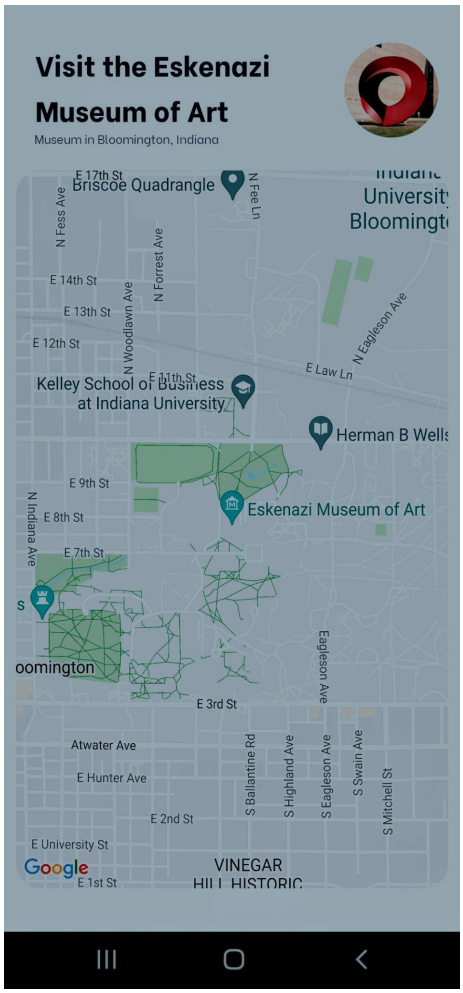


9.9 Map Feature

While scrolling through the feed on the homescreen, if a user sees a piece of art they are extra interested in, they are able to click on the profile of the museum the art is located at. Clicking on this will take them to another page where they see the name of the museum and the location of the museum on Google maps.

Inside Figma it looks like this (left), and once implemented looks like this (right):





On the left, you can see the large gray box is a container that the Bravo Studio map tag, [component:map:<latitude>:<longitude>:<zoom>], is tied to. Latitude and longitude data can be housed in an Airtable database, and then connected via API in a similar way that the homescreen feed is connected, or it can be hard coded in, by adding the actual longitude and latitude coordinates to the tag itself. On the right, once the data is bound or implemented, the screen will be showing the Google Maps location.

10 Detailed Design - Database/Server Management System

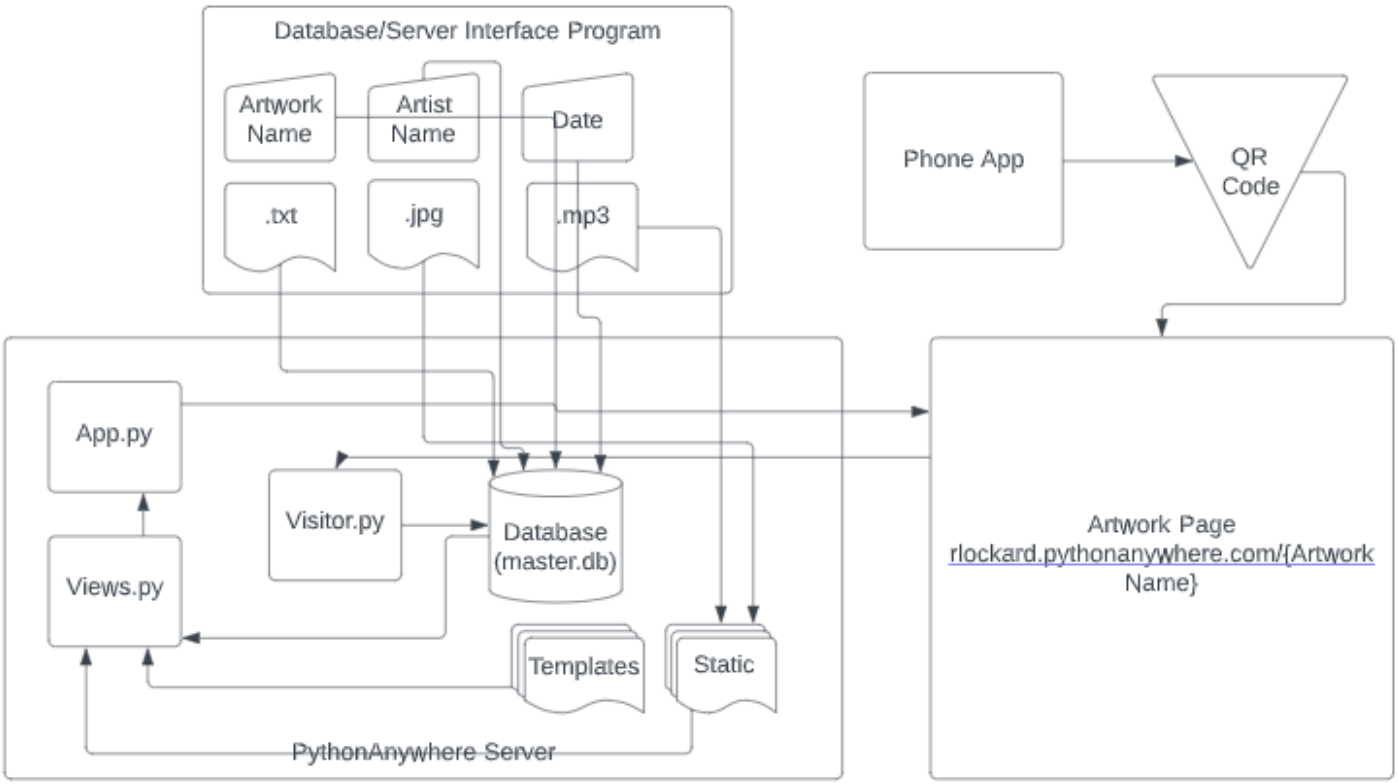


Fig. The Glass Box for the setup of the Database Server Management System and the Website.

10.1 Background and Flow of Execution

Ryan was responsible for 3 of the four parts of the finished prototype. The first part is the database/server interface program. This is a key feature of the prototype. This component allows a non-technical user to add to our system. This is important because someone with no coding skills at all can create new artwork webpages using our system.

The user will input the Artwork Name, Artist Name, and Date into the software. The user will then upload the description of the Artwork as a .txt file to the software. The user will then upload a picture of the artwork as a .jpg and upload an audio file of the description as a .mp3. The user will then click the button “Export to Database”. This will send the information into the database and upload the picture and audio file to the server through an SSH connection.

The second part of Ryan’s portion of the project was the PythonAnywhere Server. The server is what hosts all the files that are running the websites. The main file that is running on the server is the App.py file. This file calls the views.py which is the file that is dynamically creating new

endpoints based off the database entries. The views.py file is taking the information from the database, templates folder, and static folder and using it to tell the App.py file of what it needs to render to the screen. When a user goes to a webpage it calls the visitor.py file. The visitor.py file places the website user data into the database. The last part of Ryan's portion of the project was the website. The website is used to display information to the user about each piece of artwork. The website was designed in Web flow, and the HTML, CSS, and JavaScript code was exported and modified to be used with a python Flask application.

10.2 Tools Used for Implementation

1) Python
   a) Tkinter library (database/server interface GUI)
      https://docs.python.org/3/library/tkinter.html
   b) Flask library (Web framework)
      https://pypi.org/project/Flask/
   c) SQLite3 library (SQL)
      https://docs.python.org/3/library/sqlite3.html
   d) Paramiko library (SSH into server)
      https://pypi.org/project/paramiko/

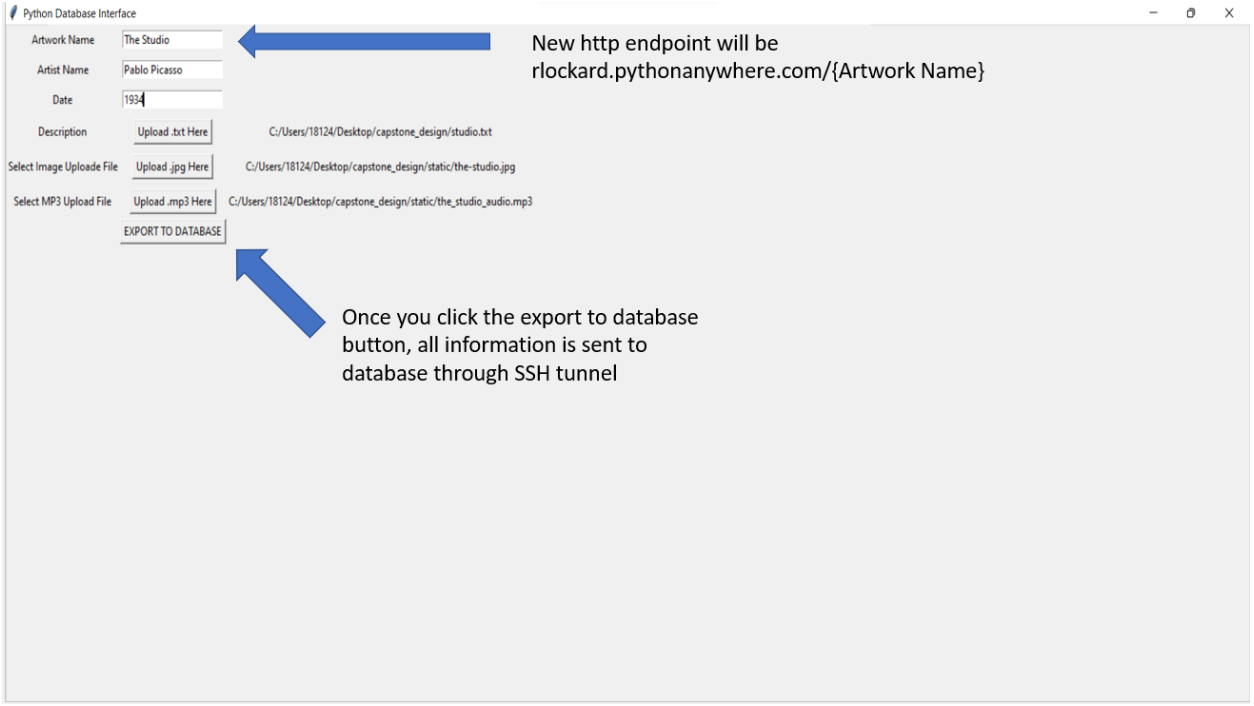
2) Web flow https://webflow.com/
   - Used to generate HTML, CSS, and JavaScript to run website

3) Python Anywhere https://www.pythonanywhere.com/
   - Used to run web application and host files

4) Microsoft Visual Studio Code https://code.visualstudio.com/
   - Text editor used to edit files

5) DB Browser for SQLite https://sqlitebrowser.org/
   - Used to view and edit database

10.3 Database/Server Interface



The user interface to add to our system is a fully packaged python application. The user types in the artwork name, artist name, and date, then uploads a .txt file for the description, a .jpg file for the artwork's picture, and a .mp3 file for the artwork's audio.

Once all the information is correctly inputted, the user will click the export to database button. This button will send all the information to the database and the python anywhere server. The code that is running on the server will then dynamically create a new http endpoint for every item added to the database. This gives our system a way for non-technical people to add to it.

The database/server interface code utilizes three different python libraries: Tkinter, paramiko, and sqlite3. Tkinter library was utilized to build the GUI. The paramiko library was used to be able to send the audio and image uploads to the python anywhere server via SSH. The sqlite3 library was utilized to be able to the send the information to the SQL database.

Pyinstaller was then used to package the python code as a completely packaged executable. The constraints on the database/server executable is that you need 74,000 KB of disc space and a machine that is running windows 8 or newer.

10.4 Python Anywhere Code

The python anywhere server code is used in our system to dynamically build http endpoints. The code on the database will create a new http endpoint for every data entry into the Eskenazi database table. The python anywhere server code in conjugation with the database/server interface executable is the true selling point of our system to a museum.

These two things working together gives non-technically people the ability to create a new page for each piece easily with no code. The python anywhere server code is a flask application. Flask is a microframework that maps python code to http endpoints. Our code is utilizing the Flask function “render_template” to render html documents to the correct http endpoint. For Flask to function properly, there is a directory structure that you must use follow.

All of the html documents for your flask web-app must be in a directory called “templates” that is in the same root directory as your running python file. All the css, javascript, images, audio, etc. must be in a separate directory called “static” for everything to be rendered correctly.

The “render_template” function gave us the ability to only have to create one html template for all of the artworks, instead of having to create an html document for each piece. The app.py file is the main file that is running on the python anywhere server. The view.py file is the file that is dynamically adding http endpoints for every database entry using the render_template function.

The other python file that is being ran on the python anywhere server is the vistor.py file. This file is the file that tracks how many people have visited each one of the web pages. The visitor.py file utilizes the ipaddress python library. This library enables our system to be able to track the devices that are using our system.

10.5 App.py

/home/rlockard/mysite/app.py

```
1  from asyncio import threads
2  from distutils.log import debug
3  from pickle import TRUE
4  from flask import Flask
5  from views import views
6  from waitress import serve
7
8
9
10 app = Flask(__name__, static_folder="static")
11 app.register_blueprint(views, url_prefix="/")
12
13
14 if __name__ == '__main__':
15     app.run()
16
17
18
19
20
```

10.6 Views.py

/home/rlockard/mysite/views.py

```
1  from flask import Blueprint, render_template
2  import sqlite3
3  from sqlite3 import Error
4  import visitor
5
6  from matplotlib import image
7
8  db_connection = sqlite3.connect('master.db', check_same_thread=False)
9  db_cursor = db_connection.cursor()
10
11 views = Blueprint(__name__, "views")
12 @views.before_request
13 def track_visitor():
14     visitor.track_visitor()
15
16 @views.route("/")
17 def home():
18     return render_template("eskenazi.html")
19
20 @views.route("/<page_name>")
21 def pages(page_name):
22     artwork_name = page_name
23     sql = "SELECT * FROM eskenazi WHERE name = ?"
24     db_cursor.execute(sql, (artwork_name,))
25     df = db_cursor.fetchone()
26     artwork_name = df[2]
27     artist_name = df[3]
28     artwork_date = df[4]
29     image_filename = page_name + '.jpg'
30     audio = page_name + '.mp3'
31     artwork_description = df[5]
32     artwork_description = artwork_description.replace('\\','')
33
34     return render_template("the_studio.html", audio_filename=audio, image_file
35
```

** return render_template("the_studio.html", audio_filename=audio, image_filename=image_filename, artwork_name=artwork_name, artist_name=artist_name, date=artwork_date, description=artwork_description) **

10.7 Visitor.py

/home/rlockard/mysite/visitor.py

```
import sys


def track_visitor():
    try:

        print("hello", file=sys.stderr)
        print(request.url, file=sys.stderr)
        print(request.url_root, file=sys.stderr)
        ip_address = request.remote_addr
        requested_url = request.url
        user_agent = request.user_agent.string
        if requested_url == request.url_root:
            return

        db_connection = sqlite3.connect('master.db', check_same_thread=False)
        db_cursor = db_connection.cursor()
        sql = "INSERT into user_info (ip_address, requested_url, user_agent) VALUES (?,?,?)"
        db_cursor.execute(sql, (ip_address, requested_url, user_agent))
        db_connection.commit()
        #db_connection.close()
        if "the_studio" in requested_url:
            db_connection = sqlite3.connect('master.db', check_same_thread=False)
            db_cursor = db_connection.cursor()
            artwork_name = "The Studio"
            sql = "SELECT * FROM page_views WHERE page_name = ?"
            db_cursor.execute(sql, (artwork_name,))
            df = db_cursor.fetchone()
            print(df[2], file=sys.stderr)
            no_page_views = df[2]
            no_page_views = no_page_views + 1
            sql = "UPDATE page_views SET number_of_views = ? WHERE page_name = ?"
            db_cursor.execute(sql, (no_page_views, artwork_name))
            db_connection.commit()
            db_connection.close()

```

11 Detailed Design - Python Anywhere Server

11.1 Background and Flow of Execution

We decided to use python anywhere as our server running our web-application. Python anywhere is an implementation of a PaaS (platform-as-a-service) concept and can be used to deploy and operate python applications. Python anywhere met all our server needs since we decided to run our web-application with python Flask. Python anywhere gave us the ability to be able to SSH into our server which is used to upload the images and audio that users upload with the database/server interface program. The current amount of space on the server is 1 GB, but that can be scaled up by paying for a bigger plan. We currently have the smallest plan that allows for SSH access. This costed the team 5 dollars per month and would be the only on-going expenses

in our prototype. The plan also comes with 2000 CPU seconds per day, but the current design does not use any of that.

11.2 Database

In our prototype, we decided to use an SQL database to host all the text-based data in our system. The reason the team decided to use an SQL database is because that is what they were already familiar with, and it was sufficient for the team's needs. The database is located on the Python anywhere server and can be accessed through an SSH tunnel. The database has 3 main tables. The first table has all the information regarding each piece of art. The second database table is a museum analytics tool to track the user information of the people using the website who would only be the people inside the museum. The last database table is another museum analytics tool to track how many people have viewed the different pages.

1) Eskenazi Table

name	artist	date	description	image_file_name	audio
Filter	Filter	Filter	Filter	Filter	Filter
The Studio	Pablo Picasso	1934	The Studio is one of Pablo Picasso\'s ...	The Studio.jpg	The Studio.mp3
Beth Aleph	Morris Louis	1960	Morris Louis was a member of the ...	Beth Aleph.jgp	Beth Aleph.mp3
Saint Eustace	Albrecht Durer	Ca. ...	Since prints are made as multiples, ...	Saint Eustace.jpg	Saint Eustace.mp3
Holy Family with Infant St. ...	Lavinia Fontana	Ca. ...	Lavinia Fontana was renowned for he...	Holy Family with Infant St. John.jpg	Holy Family with ...
Judith with the Head of ...	Matteo di Giovanni	Ca. ...	This sword-wielding woman has ...	Judith with the Head of Holofernes.jpg	Judith with the ...
The Annunciation	Felipe Vigarny	1515-...	This panel originally formed part of a ...	The Annunciation.jpg	The ...
The Nativity	Felipe Vigarny	1515-...	This panel originally formed part of a ...	The Nativity.jpg	The Nativity.mp3
The Birth of the Virgin	Felipe Vigarny	1515-...	This panel originally formed part of a ...	The Birth of the Virgin.jpg	The Birth of the ...
Homage to the Square: Brigh...	Josef Albers	1958	In 1920, Josef Albers began a course ...	Homage to the Square: Bright ...	Homage to the ...
Portrait of Lisa Bigelow	Alfred Leslie	1964	Alfred Leslie, who pioneered the ...	Portrait of Lisa Bigelow.jpg	Portrait of Lisa ...
Madonna and Child with Appl...	Studio of Bernard va...	1530	Flemish artists were renowned for vi...	Madonna and Child with Apple and ...	Madonna and Chil...
Laborers Harvesting Bath Stone	Benjamin Barker, the...	Ca. ...	This painting focuses on a picturesqu...	Laborers Harvesting Bath Stone.jpg	Laborers ...

2) Page Views Table

page_name	umber_of_view
Filter	Filter
The Birth of the Virgin	0
Saint Eustace	0
The Studio	21
Beth Aleph	0
Judith with the Head of ...	0
Holy Family with Infant St. ...	1
The Annunciation	0
The Nativity	0
Portrait of Lisa Bigelow	0
Madonna and Child with Appl...	0
Laborers Harvesting Bath Stone	0

3) User Info Table

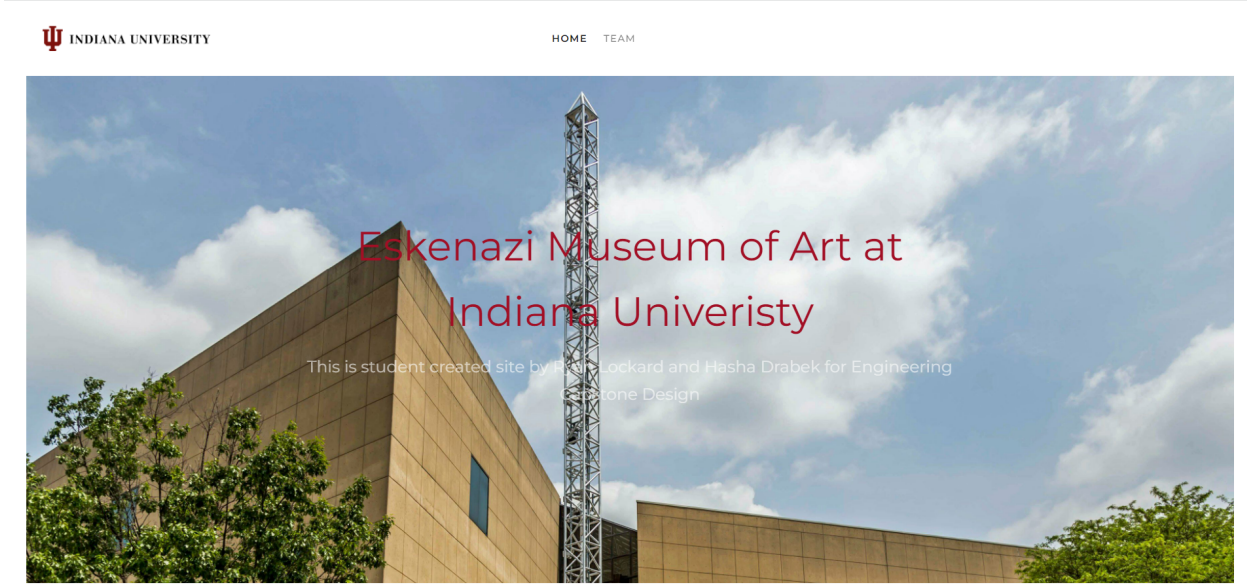
ip_address	requested_url	user_agent
Filter	Filter	Filter
192.168.0.30	http://192.168.0.30:8080/...	Mozilla/5.0 (Windows NT 10.0; Win6...
192.168.1.196	http://192.168.1.196:8080/the_studio	Mozilla/5.0 (Windows NT 10.0; Win6...
192.168.1.196	http://192.168.1.196:8080/the_studio	Mozilla/5.0 (Windows NT 10.0; Win6...
192.168.1.196	http://192.168.1.196:8080/the_studio	Mozilla/5.0 (Windows NT 10.0; Win6...
192.168.1.196	http://192.168.1.196:8080/the_studio	Mozilla/5.0 (Windows NT 10.0; Win6...
192.168.1.196	http://192.168.1.196:8080/the_studio	Mozilla/5.0 (Windows NT 10.0; Win6...
192.168.1.196	http://192.168.1.196:8080/the_studio	Mozilla/5.0 (Windows NT 10.0; Win6...
192.168.1.196	http://192.168.1.196:8080/the_studio	Mozilla/5.0 (Windows NT 10.0; Win6...
10.0.0.66	http://rlockard.pythonanywhere.com/...	Mozilla/5.0 (Windows NT 10.0; Win6...
10.0.0.66	http://rlockard.pythonanywhere.com/...	Mozilla/5.0 (Windows NT 10.0; Win6...
10.0.0.66	http://rlockard.pythonanywhere.com/...	Mozilla/5.0 (Windows NT 10.0; Win6...
10.0.0.66	http://rlockard.pythonanywhere.com/...	Mozilla/5.0 (Windows NT 10.0; Win6...
10.0.0.66	https://rlockard.pythonanywhere.co...	Mozilla/5.0 (Windows NT 10.0; Win6...
10.0.0.66	http://rlockard.pythonanywhere.com/...	Mozilla/5.0 (Windows NT 10.0; Win6...
10.0.0.66	http://rlockard.pythonanywhere.com/...	Mozilla/5.0 (Windows NT 10.0; Win6...
10.0.0.66	https://rlockard.pythonanywhere.co...	Mozilla/5.0 (Windows NT 10.0; Win6...
10.0.0.66	https://rlockard.pythonanywhere.co...	Mozilla/5.0 (Windows NT 10.0; Win6...
10.0.0.66	https://rlockard.pythonanywhere.co...	Mozilla/5.0 (Windows NT 10.0; Win6...
10.0.0.66	https://rlockard.pythonanywhere.co...	Mozilla/5.0 (Windows NT 10.0; Win6...
10.0.0.66	https://rlockard.pythonanywhere.co...	Mozilla/5.0 (Windows NT 10.0; Win6...

11.3 Website

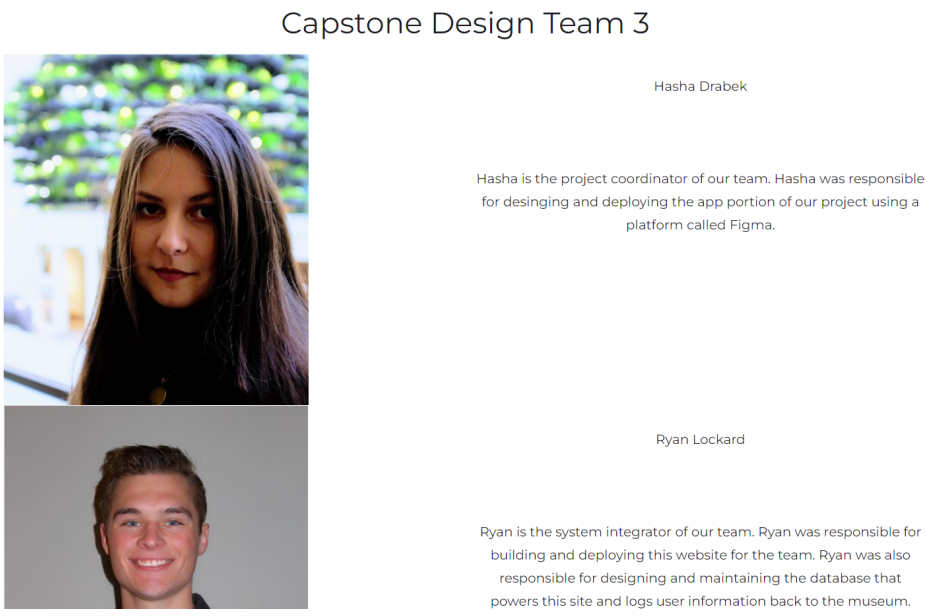
The website was designed using Web flow. Web flow is an online website editing platform that gives you the opportunity to export the HTML, CSS, and JavaScript code running the website. This was a huge help to being able to get the project done on time because no one on the team was very familiar with writing HTML, CSS, and JavaScript to run a website. Once the design of the website was finished, we exported all the code to a zip file and changed the necessary items for the code to function with our python Flask application. All of the html files were placed in

the “templates” folder and all static files (pictures, CSS, audio, JavaScript) were all placed in the “static” folder, so everything would function correctly with the Flask framework.

Home Page



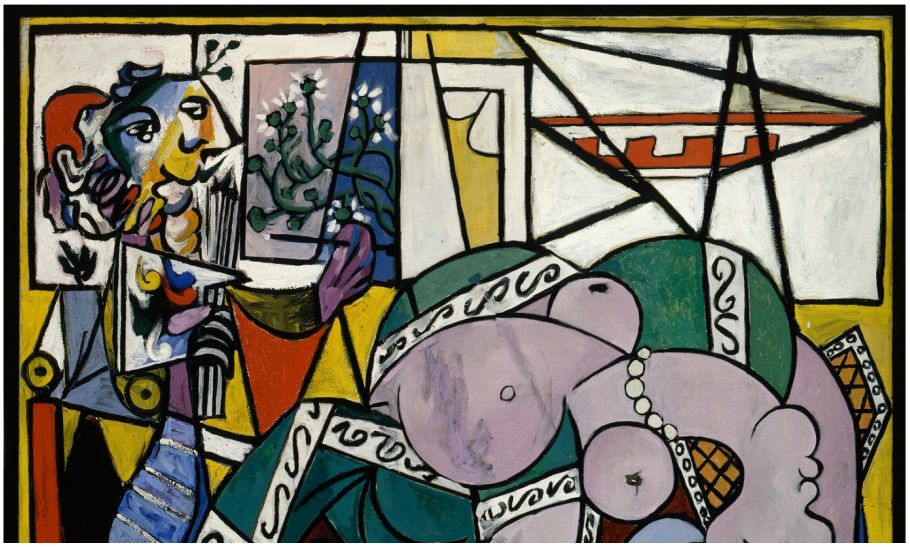
Team Page

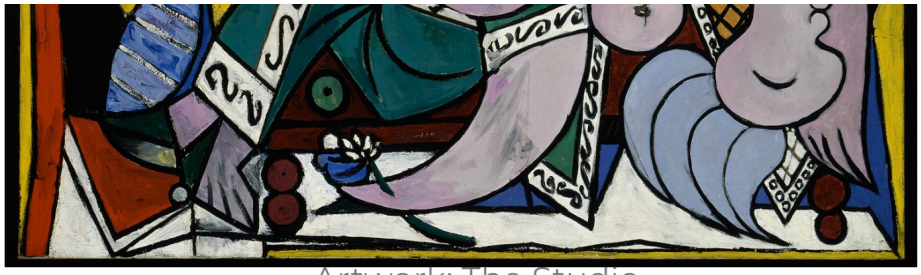


Sample Artwork Page

INDIANA UNIVERSITY

HOME TEAM



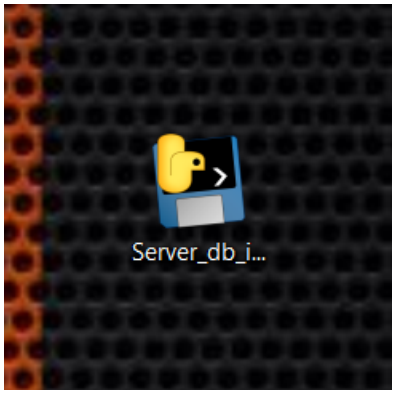


Artwork: The Studio
Artist : Pablo Picasso
Date: 1934

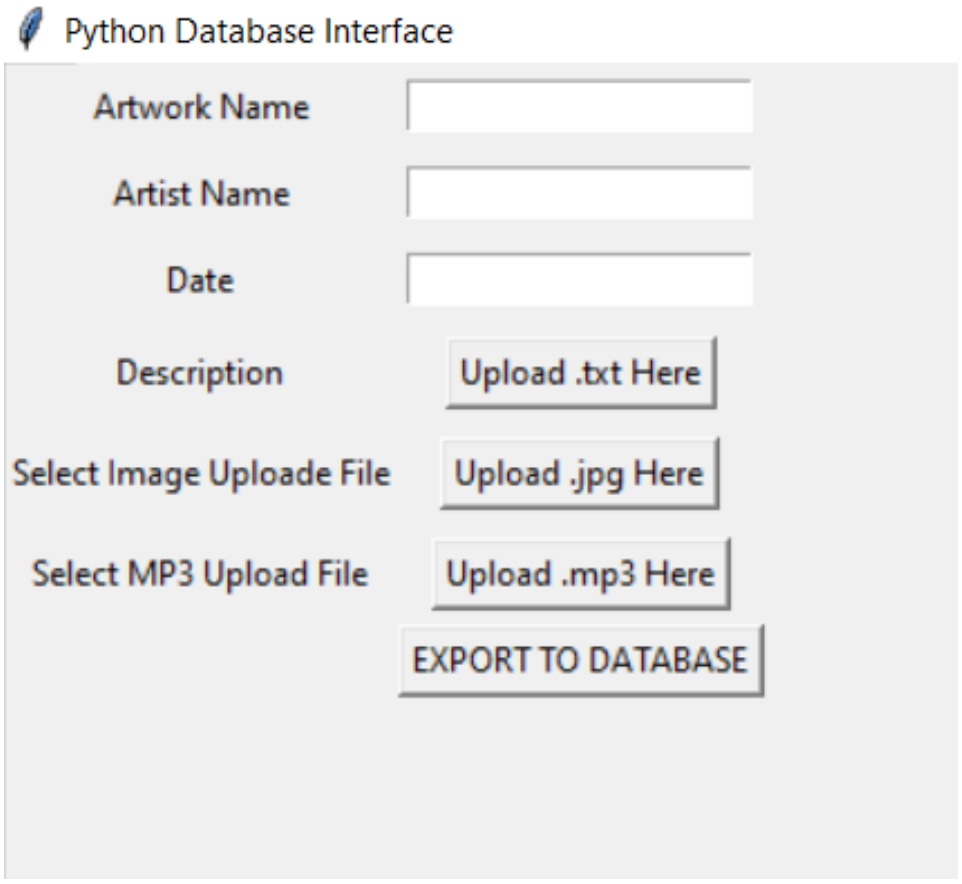
0:00 / 0:47

The Studio is one of Pablo Picasso's major works addressing the theme of artist and model. A provocative intersection of style, subject, and biography, the work portrays Picasso standing at his easel, while his much younger model and mistress, Marie-Therese Walter, reclines on a sofa. The exuberant colors and bold shapes reflect Picasso's continuing development of the radical Cubist vocabulary he and Braque pioneered before World War I. Yet the composition also reinforces traditionally misogynist perceptions about male creativity and female passivity. In this work, the male artist's realm is defined by angular geometries evoking rational intellectualism, while Marie-Therese's form is one of rounded, organic shapes that suggest her creativity is linked to her sexuality.

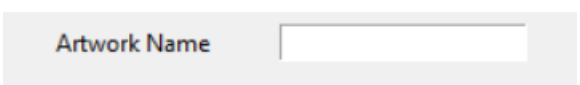
12 User Manual



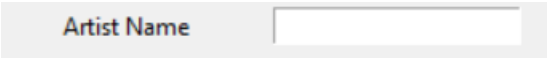
1) Click on the Icon to launch the database server interface program



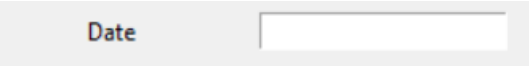
2) Once the program has launched, this should be your screen



3) Type the Artwork Name into this box

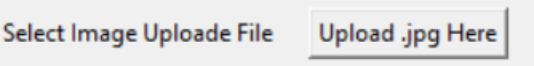


4) Type the Artist Name into this box



5) Type the Date the artwork was published in this box

6) Next click on this button and select a .txt file that contains the description of the artwork



7) Next click on this button and select a .jpg (picture) file for the artwork

8) Next click on this button and select a .mp3 (audio) file for the artwork

9) Make sure all information is correct

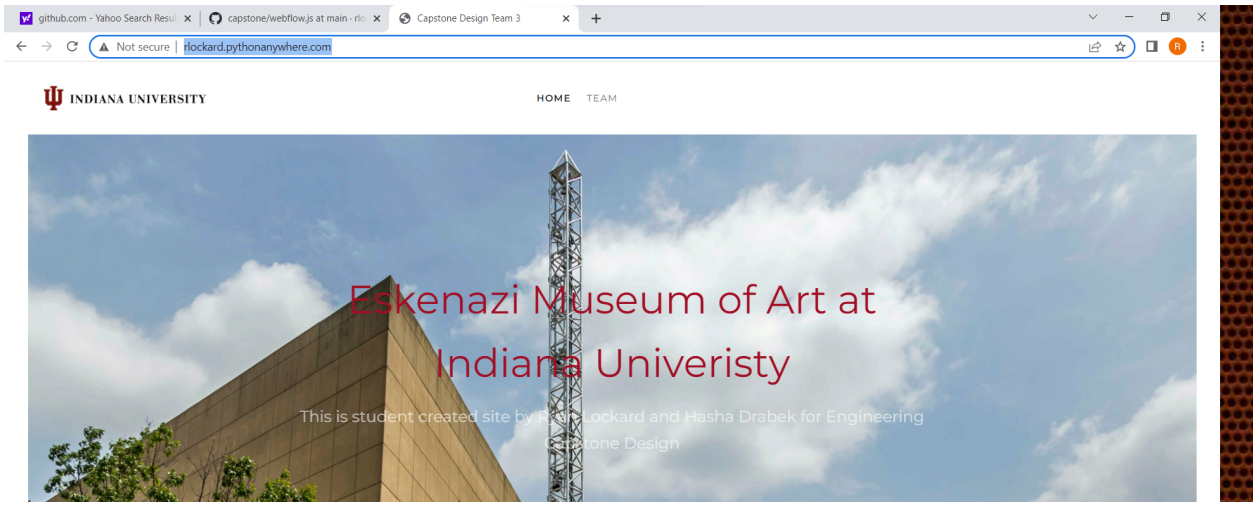
10) After making sure information is correct, click this button to send the information to the database

11) After information is sent to the database, a new http endpoint will be created at rlockard.pythonanywhere.com/{Artwork Name}

13 Steps to Continue the Project

1. Download the code used for the project
   https://github.com/rlockard22/capstone

2. Setup a PythonAnywhere account
3. Upload the code to PythonAnywhere matching the file structure that is on github
4. Tell PythonAnywhere to run the App.py file as a web-application
5. Go to the website to confirm that the code is correctly running. The url will be http://{username}.pythonanywhere.com/ Mine was http://rlockard.pythonanywhere.com/



** It should look like this **

6. Change the server_db_interface.py file to connect to your new PythonAnywhere account

```
ssh = SSHClient()
ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
ssh.connect(hostname="ssh.pythonanywhere.com", username="rlockard", password="Baseball2@")
```

   You will change this line of code to match your new login information from PythonAnywhere

7. After the code has been changed to upload to your new server, you can rebuild the server_db_interface.py file into an executable using the server_db_interface.spec file. To do this, you will need the server_db_interface.py file and the server_db_interface.spec file to be in the same directory. Navigate into the directory with both files in the command line and run the command “pyinstaller -D server_db_interface.spec”. This command will build the python script into a Windows executable.

8. Now try to use the executable to upload to the server. Whatever you name the Artwork Name is what will be used to create the new URL endpoint. http://{username}.pythonanywhere.com/{artwork_name}
9. To continue the app design, all you need is this link, anyone with the link can duplicate the design and import to their own Bravo Studio account: https://www.figma.com/file/J56ML4YxuUzNrM60P71LRZ/gram-test?node-id=0%3A1
10. You will need your own Bravo Studio, Airtable, and Firebase accounts.

14 Future Addition Ideas

Specifically for the app, some ideas we have are to have a better map feature with better location data, there are some ways to include markers and user location, and this would be a good addition to the app. Also as discussed earlier, features like a comment section can be implemented in a similar way to the home feed implementation, with API database connecting, except for just having read data, there would also be write data into the database. The app can also be exported from Bravo Studio into TestFlight, which is the waiting room apps have to sit in before they are accepted into major app stores like iOS or Android. People with a special link from TestFlight are able to download the app and use it on their phone.

Specifically for the databasing services and software we built, we think automating the user tracking would be a good next step. Right now it is all hard coded, so automating it to be friendly for a non tech person would be useful for this project.

Our mentor also still has interest in AI, which we explored in the first semester and have some work of it included in the appendix, as well as VR, which we did not attempt to implement. We think something like revisiting AI and using something like AWS Rekognition Custom Labels might be a next step: https://aws.amazon.com/rekognition/custom-labels-features/

15 Lifelong Learning Objectives

Ryan gained a lot of technical networking skills, CSS, HTML, JavaScript, and improved his knowledge of Python. Hasha gained skills such as learning the Figma software, Bravo Studio software, which included learning about things like containers, tags, and API database connecting. She also learned a lot about UI design, and the business side of developing a service

like this project. Both Hasha and Ryan gained documentation, group coordination, and information sharing skills.

16 Acknowledgements

Hasha and Ryan would like to thank Indiana University for deciding to form the Intelligent Systems Engineering department.

We are beyond excited to join the short list of Engineers to graduate from the Luddy School of Informatics, Computing, and Engineering.

The team completed all the work for the project without the help of any professor, but both team members have ample faculty and staff to thank for getting us ready to start our professional careers.

We would personally like to thank both of our professors, Suha Lassasmeh and Bryce Himebaugh, for their guidance in preparing the team to work in the real world.

17 Appendix (first semester AI attempt)

Art Database

The art database will be used to store all the information about each individual piece of art. The art database will give museums the opportunity to write longer descriptions of each piece of art then the 200 word descriptions that are usually on the wall. The art database will also give museums that do not yet have a database for their art an opportunity to store their information on the cloud. The art database will be managed with an SQL python script. The art database will be read only and will be hosted on Amazon Web Services server.

Database Schema

```
import sqlite3
from sqlite3 import Error
from urllib.request import HTTPDefaultErrorHandler


conn = sqlite3.connect('master.db')
c = conn.cursor()

c.execute('''CREATE TABLE IF NOT EXISTS eskenazi
                    ([id] integer PRIMARY KEY AUTOINCREMENT,
                    [location] VARCHAR(100),
                    [name] VARCHAR(100),
                    [artist] VARCHAR(100),
                    [description] VARCHAR(1000))''')

conn.commit()

```

Table: eskenazi

Filter in any column

	id	location	name	artist	date	description
	Filter	Filter	Filter	Filter	Filter	Filter
1	1	Modern Art	The Studio	Pablo Picasso	1934	The Studio is one of Pablo Picasso\'s major work...
2	2	Modern Art	Portrait of Lisa Bigelow	Alfred Leslie	1964-66	Portrait of Lisa Bigelow depicts Alfred Leslie\'s ...
3	3	Modern Art	Beth Aleph	Morris Louis	1960	Morris Louis was a member of the Washington ...
4	4	European and American Art, Medieval to 1900	Tavern Scene	Cornelis Bega	1664	One of the most important Dutch painters of rusti...
5	5	European and American Art, Medieval to 1900	Allegory of the Four Parts of the World	Francesco Solimena	1738	The Neapolitan clergy, aristocracy, and royal fam...
6	6	European and American Art, Medieval to 1900	The Myth of Deucalion and Pyrrha	Niccolo Giolfino	ca. 1550	This painting depicts a creation tale from Greek ...
7	7	European and American Art, Medieval to 1900	Panel from an Altarpiece: Coronation of the Virgin	Vittore Crivelli	ca. 1497	Vittore Crivelli worked in a late Gothic style that ...
8	8	European and American Art, Medieval to 1900	Madonna and Child	Bernardino Lanino	16th century	Bernardino Lanino studied under the artist ...
9	9	European and American Art, Medieval to 1900	Still Life with Lobster	Pieter de Ring	ca. 1650	An abundance of fruit, seafood, bread, and wine ...
10	10	European and American Art, Medieval to 1900	Copy after the Barberini Holy Family	Workshop of Andrea del Sarto	ca. 1529-30	The richly luminous palette and monumental ...
11	11	Art of Africa, Oceania, and Indigenous Art of the ...	Shield	Maasai peoples, Kenya	collected 1977	A shield was one of the most important objects ...
12	12	Art of Africa, Oceania, and Indigenous Art of the ...	Shield	Telefol peoples, New Guinea	19th or 20th century	Living in the New Guinea central highlands, the ...
13	13	Art of Africa, Oceania, and Indigenous Art of the ...	Painting of a Female Figure	Arnhem Land, Australia	20th century	Most Australian bark paintings are from Arnhem ...

User Database

The user database will be used to store user information. The user database will give museums the opportunity to view analytics based on what pieces of art people are interacting with at the museum. This will be done by a user first creating an account in the application. This will create a new record in the user database identifying that this new user now exists. Whenever a user interacts with a piece of art, a record of the interaction will be created in the user database. The user database will be hosted on an Amazon Web Services server, and will have to be updated after each profile is made and each art interaction is made.

Database Schema

```
import sqlite3
from sqlite3 import Error
from urllib.request import HTTPDefaultErrorHandler


conn = sqlite3.connect('app_users.db')
c = conn.cursor()

c.execute('''CREATE TABLE IF NOT EXISTS users
                    ([user_id] integer PRIMARY KEY AUTOINCREMENT,
                    [first_name] VARCHAR(25),
                    [last_name] VARCHAR(25),
                    [username] VARCHAR(25),
                    [email] VARCHAR(50),
                    [password] VARCHAR(25))''')

c.execute('''CREATE TABLE IF NOT EXISTS interactions
                    ([interaction_id] integer PRIMARY KEY AUTOINCREMENT,
                    [user_id] integer,
                    [art_id] integer,
                    [art_name] VARCHAR(25),
                    [art_museum] VARCHAR(50))''')

conn.commit()
```

Machine Learning Algorithm Design

For the identification portion of our project we planned to use a convolutional neural network. After early testing we realized that just using a CNN was not going to work for our purposes. This was because the output of a CNN was going to be 250 different percentages (because we have 250 different items to identify) that add up to one. This was fine for some of the pieces with the percentage output for the correct class being very close to 1, but for other pieces the outputs were all over the place with the highest output not always being the correct one. To account for this we are going to have to change our machine learning strategy. We are going to have to add another algorithm that compares the input picture to the stock picture for each class from our database to ensure we are identifying each art piece correctly. Our new design for our identification will have a CNN as its first layer. This will output a percentage that a picture belongs to a certain class. The comparison algorithm will be called in order based on the highest to lowest of the percentage output of the CNN. This will hopefully ensure that the comparison algorithm only has to be run a few times (preferably just once) before it identifies the correct piece of art. To train the algorithm we have taken a picture of each piece of art at the Eskenazi Museum of Art. We are using an image augmentation algorithm to produce thousands of augmented copies of each image from the single image we actually took.

Data Augmentation

This algorithm from the tensorflow library will take a single image and produce as many augmented copies of the image as you like. There are many different inputs to the algorithm to tinker with that change how augmented the new image is from the original image.

```python
import tensorflow as tf
from keras.preprocessing.image import ImageDataGenerator
from skimage import io
import os


datagen = ImageDataGenerator(
            rotation_range=10, # 0 to 360 degrees the duplicates will be rotated
            width_shift_range=0.1,
            height_shift_range=0.1,
            shear_range=.05,
            zoom_range=.25,
            fill_mode='constant', cval=256)


i = 0
hello = os.getcwd()
for batch in datagen.flow_from_directory(directory=hello,batch_size=16,
                        target_size=(256,256),color_mode='rgb',save_to_dir='augmented',
                        save_prefix='aug',save_format='jpg'):
    i += 1
    if i > 100:
        break
```

First Model

The first model we built as our prototype was built to identify between 8 different images. The model was a convolutional neural network that took 256x256 sized images as inputs. The model's accuracy was 94.5 percent which is actually really good, but made us realize that just having a CNN was not going to work for identification purposes. In the first model we used an image of a shield, a lobster, a fan, a piece of bamboo, and 4 different pictures from the Catholic Church. The model was very good at distinguishing the shield, lobster, fan, and bamboo. The output for our test image of the correct class was very close to 1, but the four pictures from the Church were a different story. The output of the model for each Church class was very close together and the top output percentage was not the correct class for one of the test images. We are going to go back and take more pictures of each item because this model was built by augmenting a single image for each class. Taking more photos of each class might make our model more accurate and more robust.

```
#Creating the neural network
from tensorflow.keras.models import Sequential

# Initialising the CNN
cnn = Sequential()
from tensorflow.keras.layers import Conv2D, MaxPool2D, Flatten, Dense
# Step 1 - Convolution
cnn.add(Conv2D(filters=64, kernel_size=3, padding="same", activation="relu", input_shape=[256, 256, 3]))
# Step 2 - Pooling

cnn.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2, padding='valid'))

# Adding a second convolutional layer
cnn.add(tf.keras.layers.Conv2D(filters=64, kernel_size=3, padding="same", activation="relu"))
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2, padding='valid'))

# Step 3 - Flattening
cnn.add(Flatten())
# Step 4 - Full Connection
cnn.add(Dense(units=128, activation='relu'))

# Step 5 - Output Layer
cnn.add(Dense(units=8, activation='softmax'))

from tensorflow.keras.optimizers import SGD
sgd = SGD(learning_rate = 0.01,decay  =1e-6,momentum  = 0.9,nesterov  = True) #optimizer,lr = learning rate

cnn.compile(optimizer = sgd, loss = 'categorical_crossentropy', metrics = ['accuracy'])
#you can use adam as well

cnn.fit_generator(training_set,
                  steps_per_epoch = 20,
                  epochs = 30,
                  validation_data = test_set,
                  validation_steps = 10)

from tensorflow.keras.preprocessing import image
```