

# Fractal Factory



Project Paper (Portfolio)

Kenny Mejia, Candice Rivera, Jada Tijani, Brandon Kline, and Eric Stenton  
Professor Rivas  
CMPT\_475N\_200 - CS/IT/IS Capping  
4 December 2019

# **Table of Contents**

|  |           |
|--|-----------|
| <b>Table of Contents</b>                 | <b>2</b>  |
| <b>Summary</b>                           | <b>4</b>  |
| <b>Software Project Analysis</b>         | <b>5</b>  |
| User Requirements                        | 5         |
| Use Case Diagram                         | 6         |
| Use Case Diagram Documentation           | 7         |
| Activity Diagram                         | 14        |
| Activity Diagram Documentation           | 15        |
| Sequence Diagrams                        | 16        |
| Sequence Diagram Documentation           | 17        |
| <b>Software Project Design</b>           | <b>18</b> |
| ER Diagram                               | 18        |
| Database Documentation                   | 19        |
| User Interface                           | 24        |
| Login Page:                              | 24        |
| User Profile Page:                       | 25        |
| Administrator Profile Page:              | 26        |
| Upload Page:                             | 27        |
| Results Page:                            | 28        |
| Purchase Page                            | 29        |
| Coinbase Page:                           | 30        |
| CanvasPop Page:                          | 31        |
| About Page:                              | 32        |
| <b>Infrastructure Specifications</b>     | <b>33</b> |
| IT Requirements                          | 33        |
| Reliability                              | 34        |
| Recoverability                           | 34        |
| Security and Privacy                     | 34        |
| Maintenance                              | 35        |
| <b>Prototype Deployment Instructions</b> | <b>36</b> |
| Github Repository Link                   | 36        |
| Accessing Project in Production          | 36        |
| Nodejs Application:                      | 36        |

|                                 |           |
|---------------------------------|-----------|
| Python API:                     | 36        |
| Database:                       | 36        |
| Testing Functionality           | 37        |
| Registration/Login:             | 37        |
| Uploading Source Code:          | 38        |
| Creating a New Painting:        | 38        |
| Python API:                     | 39        |
| Other Endpoints:                | 39        |
| Credentials                     | 42        |
| CanvasPop:                      | 42        |
| CoinBase:                       | 42        |
| Postgres Database:              | 42        |
| Default Admin Account:          | 42        |
| Facebook:                       | 42        |
| Twitter:                        | 42        |
| Deployment Instructions         | 43        |
| Database:                       | 43        |
| Python API:                     | 44        |
| Nodejs Application:             | 44        |
| <b>Cost Analysis / ROI</b>      | <b>47</b> |
| Cost of Labor                   | 47        |
| Cost of Site                    | 47        |
| ROI                             | 48        |
| <b>Revised Project Plan</b>     | <b>49</b> |
| <b>Ethics Essay</b>             | <b>51</b> |
| <b>Appendix</b>                 | <b>54</b> |
| Database 3NF Proof              | 54        |
| SQL Statements                  | 59        |
| Create Statements:              | 59        |
| Example Data Insert Statements: | 61        |

## **Summary**

Many would argue that coding is an art; when you go to look at a completed program that has followed good programming practices, you usually end up with a file full of text that shares many similarities to a painting. Due to indentation, spacing, line length, and other similar guidelines, patterns begin to form in these blocks of text. These patterns form shapes that at first glance may look unimpressive, but when you think about it, these shapes and patterns were formed because the coder followed good programming practices and standards. We asked whether or not there was a clear relationship between how closely one follows these programming standards and the final patterns and shapes formed by the blocks of code. This idea was originally being researched by Doctor Ron Coleman, a professor at Marist College, School of Computer Science and Mathematics. In his research, he proposed that by taking a line of written code and converting each character to blocks, you would get a final representation that mimicked the style of the original line. The purpose of this conversion was to be able to calculate the fractal dimension of multiple “blocked” lines (also known as a BAM). In fractal geometry, a fractal dimension is a measure of how “complicated” a self-similar figure is. Using this idea of a fractal dimension, we are able to quantify just how complicated these figures can be. Our fractal factory web application aims to take this idea of converting lines of code into blocks to be able to calculate their fractal dimension. We would also calculate the fractal dimension of some paintings that we have acquired through an API provided by the Metropolitan Museum of Art (MET) and find a painting that matches the fractal dimension of the “blocked” lines of code. The fractal dimension of the “blocked” lines of code would be used to ultimately influence the painting from the MET to produce a unique altered version of the original painting. How much or how little the painting gets altered ultimately depends on the original code given as input and the style of its creator.

# Software Project Analysis

## User Requirements

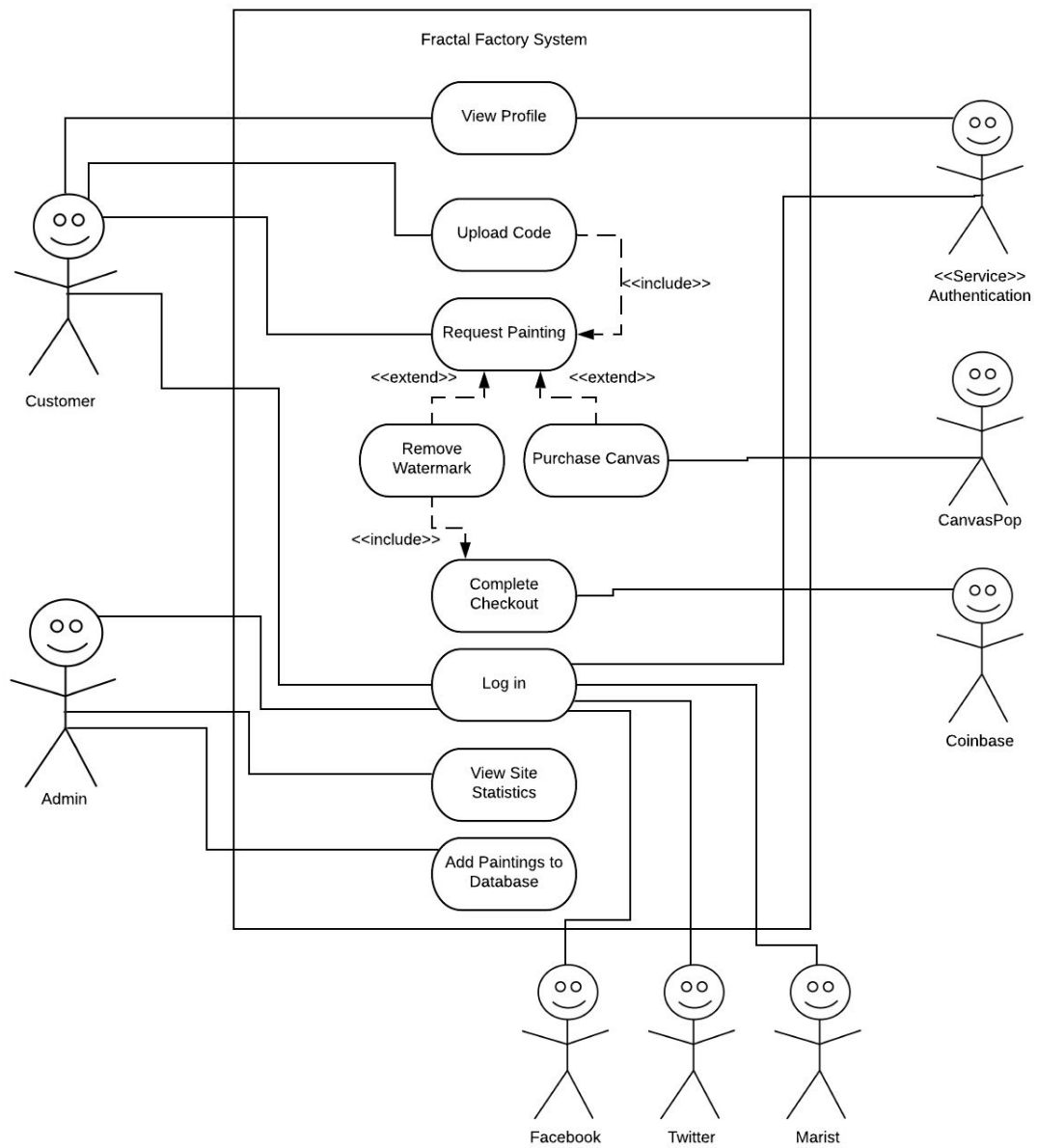
### **Front End:**

- A login page for users, with sign-in supported by Marist, Facebook, and Twitter APIs
- A profile page for each user (simplistic, no plans for a social network)
- The ability to upload code and print stylized renders of said code
- A selection of top 3 paintings for different stylization options, sorted by fractal dimension
- The ability to save previously uploaded code and paintings per user
- The ability to print a full-size image with the Canvas Pop API for a small fee
- Other basic web pages (main page, about us, etc)
- Interactive loading bar while processing to increase immersion

### **Back End:**

- Administrator and User roles/access
- Neural network connected to painting database for code analysis and art production
- Fractal dimension calculator
- Code block converter
- Statistics tracking

## Use Case Diagram



## Use Case Diagram Documentation

|  |                                    |                               |
|--|------------------------------------|-------------------------------|
| <b>Use Case Name:</b> View Profile   | <b>ID:</b> 01                      | <b>Importance Level:</b> High |
| <b>Primary Actor:</b> Customer   | <b>Use Case Type:</b> Detail, real |                               |
| <b>Stakeholders and Interests:</b><br><b>Customer:</b> View their paintings and how many they have.<br><b>Authentication:</b> Validate customer is seeing only information they are allowed to on profile web page. Blocks site statistics from rendering. |                                    |                               |
| <b>Brief Description:</b> Customer is able to see information related to their profile. Currently profile should show their generated paintings.   |                                    |                               |
| <b>Trigger:</b> Customer logs in or navigates to root page while logged in<br><b>Type:</b> External  |                                    |                               |
| <b>Relationships:</b><br><b>Association:</b> Customer, Authentication  |                                    |                               |
| <b>Normal Flow of Events:</b><br>1. Customer gains access to profile page<br>2. Views total number of generated paintings associated with their account<br>3. Views generated paintings associated with their account.<br>The S-1 subflow is performed.    |                                    |                               |
| <b>Subflows:</b><br>S-1: Clicks on generated painting and is brought to purchase page.   |                                    |                               |
| <b>Alternative/Exceptional Flow:</b> None  |                                    |                               |

|   |                                    |                               |
|---|------------------------------------|-------------------------------|
| <b>Use Case Name:</b> Upload Code   | <b>ID:</b> 02                      | <b>Importance Level:</b> High |
| <b>Primary Actor:</b> Customer  | <b>Use Case Type:</b> Detail, real |                               |
| <b>Stakeholders and Interests:</b><br><b>Customer:</b> Uploads source code in the form of raw text or a file.   |                                    |                               |
| <b>Brief Description:</b> Source code can be uploaded to the application either as raw text or within a file. A BAM image is created and the fractal dimension is calculated. |                                    |                               |
| <b>Trigger:</b> Customer clicks ‘Upload Text’ or ‘Upload File’ buttons after filling out the required input field.<br><b>Type:</b> External                                   |                                    |                               |

|   |
|---|
| <b>Relationships:</b><br><b>Association:</b> Customer<br><b>Include:</b> Request Painting   |
| <b>Normal Flow of Events:</b> <ol style="list-style-type: none"> <li>1. Customer provides source code to application.<br/>The S-1 and S-2 subflows are used.</li> <li>2. Source code is saved to machine running Nodejs application.</li> <li>3. BAM image is generated of the source code.</li> <li>4. Fractal dimension of BAM image is calculated.</li> <li>5. Three paintings with similar fractal dimensions are chosen.</li> <li>6. Customer is redirected to results page and given the three paintings to choose from.</li> </ol> |
| <b>Subflows:</b><br>S-1: Customer selects a file of source code from their computer and uploads it to the site.<br>S-2: customer copies or types source code into text field on upload page and uploads it to the site.   |
| <b>Alternative/Exceptional Flow:</b> None   |

|   |                                    |                               |
|---|------------------------------------|-------------------------------|
| <b>Use Case Name:</b> Request Painting  | <b>ID:</b> 03                      | <b>Importance Level:</b> High |
| <b>Primary Actor:</b> Customer  | <b>Use Case Type:</b> Detail, real |                               |
| <b>Stakeholders and Interests:</b><br><b>Customer:</b> Submits source code and chooses a painting to generate a new painting.   |                                    |                               |
| <b>Brief Description:</b> Using source code and a selected painting, the application’s variational autoencoder model can generate a new painting.   |                                    |                               |
| <b>Trigger:</b> Customer chooses a painting on the results page after having uploaded their source code.<br><b>Type:</b> External   |                                    |                               |
| <b>Relationships:</b><br><b>Association:</b> Customer<br><b>Include:</b> Upload Code  |                                    |                               |
| <b>Normal Flow of Events:</b><br>1. Customer chooses one of three paintings on results page.<br>2. BAM image and painting along with their fractal dimensions are sent to python API to serve as inputs to the variational autoencoder model.<br>3. A new painting is outputted from the model and sent back to Nodejs application.<br>4. Customer is redirected to purchase page to view the newly generated painting. |                                    |                               |



|   |
|---|
| <b>Subflows:</b> None                     |
| <b>Alternative/Exceptional Flow:</b> None |

|   |                                    |                               |
|---|------------------------------------|-------------------------------|
| <b>Use Case Name:</b> Remove Watermark  | <b>ID:</b> 04                      | <b>Importance Level:</b> High |
| <b>Primary Actor:</b> Customer  | <b>Use Case Type:</b> Detail, real |                               |
| <b>Stakeholders and Interests:</b><br><b>Customer:</b> Able to remove watermark from their generated painting after payment.  |                                    |                               |
| <b>Brief Description:</b> After payment, watermark is removed from the generated painting.  |                                    |                               |
| <b>Trigger:</b> Customer clicks on the 'Remove Watermark' and 'Pay with Crypto' button.<br><b>Type:</b> External  |                                    |                               |
| <b>Relationships:</b><br><b>Extend:</b> Request Painting<br><b>Include:</b> Complete Checkout   |                                    |                               |
| <b>Normal Flow of Events:</b><br>1. Customer requests watermark to be removed after payment.<br>2. Nodejs application receives request and changes watermark flag on specified painting to false.<br>3. Painting will no longer have watermark added to it upon file request. |                                    |                               |
| <b>Subflows:</b> None   |                                    |                               |
| <b>Alternative/Exceptional Flow:</b> None   |                                    |                               |

|   |                                    |                               |
|---|------------------------------------|-------------------------------|
| <b>Use Case Name:</b> Purchase Canvas   | <b>ID:</b> 05                      | <b>Importance Level:</b> High |
| <b>Primary Actor:</b> Customer  | <b>Use Case Type:</b> Detail, real |                               |
| <b>Stakeholders and Interests:</b><br><b>CanvasPop:</b> CanvasPop API handles purchase, printing, and shipping of generated painting on a variety of materials to the customer. |                                    |                               |
| <b>Brief Description:</b> Using CanvasPop, a customer can order a canvas print of a generated painting.   |                                    |                               |

|  |
|--|
| <b>Trigger:</b> Customer clicks on 'Buy Canvas Print' button.<br><b>Type:</b> External   |
| <b>Relationships:</b><br><b>Association:</b> CanvasPop<br><b>Extend:</b> Request Painting  |
| <b>Normal Flow of Events:</b> <ol style="list-style-type: none"> <li>1. Customer is redirected to the third party CanvasPop application</li> <li>2. CanvasPop receives generated painting image and gives customer options for a custom print.</li> <li>3. Customer is prompted for shipping and payment information.</li> <li>4. Order is complete and customer is redirected back to purchase page.</li> </ol> |
| <b>Subflows:</b> None  |
| <b>Alternative/Exceptional Flow:</b> None  |

|   |                                    |                               |
|---|------------------------------------|-------------------------------|
| <b>Use Case Name:</b> Complete Checkout   | <b>ID:</b> 06                      | <b>Importance Level:</b> High |
| <b>Primary Actor:</b> Customer  | <b>Use Case Type:</b> Detail, real |                               |
| <b>Stakeholders and Interests:</b><br><b>CoinBase:</b> Coinbase handles the checkout process and movement of crypto between wallets.  |                                    |                               |
| <b>Brief Description:</b> The customer is able to pay with crypto currency to remove the watermark from any generated painting.   |                                    |                               |
| <b>Trigger:</b> Customer clicks on the 'Remove Watermark' and 'Pay with Crypto' button.<br><b>Type:</b> External  |                                    |                               |
| <b>Relationships:</b><br><b>Association:</b> Coinbase<br><b>Include:</b> Remove Watermark   |                                    |                               |
| <b>Normal Flow of Events:</b><br>1. Customer clicks on 'Pay with Crypto' button.<br>2. Customer is redirected to the third party site, Coinbase.<br>3. Name and email is provided to Coinbase.<br>4. Customer sends money to the provided wallet address.<br>5. Customer is redirected back to purchase page. |                                    |                               |
| <b>Subflows:</b> None   |                                    |                               |

**Alternative/Exceptional Flow:** None

|   |                                    |                               |
|---|------------------------------------|-------------------------------|
| <b>Use Case Name:</b> Log In  | <b>ID:</b> 07                      | <b>Importance Level:</b> High |
| <b>Primary Actor:</b> Customer  | <b>Use Case Type:</b> Detail, real |                               |
| <b>Stakeholders and Interests:</b><br><b>Customer:</b> Is able to provide credentials to any method of login and gain access to their account.<br><b>Admin:</b> Similar to customer, admin is able to provide credentials to access their account.<br><b>Authentication:</b> Service authenticates the customer using the provided information or third party login method.<br><b>Facebook:</b> Provides the customer the ability to log into the application using their Facebook account.<br><b>Twitter:</b> Provides the customer the ability to log into the application using their Twitter account.<br><b>Marist:</b> Provides the customer the ability to log into the application using their Marist account. |                                    |                               |
| <b>Brief Description:</b> A customer is able to log in to the application to gain access to their profile page using the following: username and password, Facebook, Twitter, or Marist.  |                                    |                               |
| <b>Trigger:</b> Clicking either of the four kinds of ‘login’ buttons on the login page.<br><b>Type:</b> External  |                                    |                               |
| <b>Relationships:</b><br><b>Association:</b> Admin<br><b>Association:</b> Authentication<br><b>Association:</b> Facebook<br><b>Association:</b> Twitter<br><b>Association:</b> Marist   |                                    |                               |
| <b>Normal Flow of Events:</b><br>1. User clicks on either of the four available ‘login’ buttons on the login page.<br>The S-1, S-2, S-3, and S-4 subflow are used.  |                                    |                               |
| <b>Subflows:</b><br>S-1: The user logs in by filling out the username and password field and clicks the ‘login’ button.<br>S-2: The user logs in by clicking the ‘Facebook Login’ button and logs in with a Facebook account.<br>S-3: The user logs in by clicking the ‘Twitter Login’ button and logs in with a Twitter account.<br>S-4: The user logs in by clicking the ‘Marist Login’ button and logs in with a Marist account.   |                                    |                               |

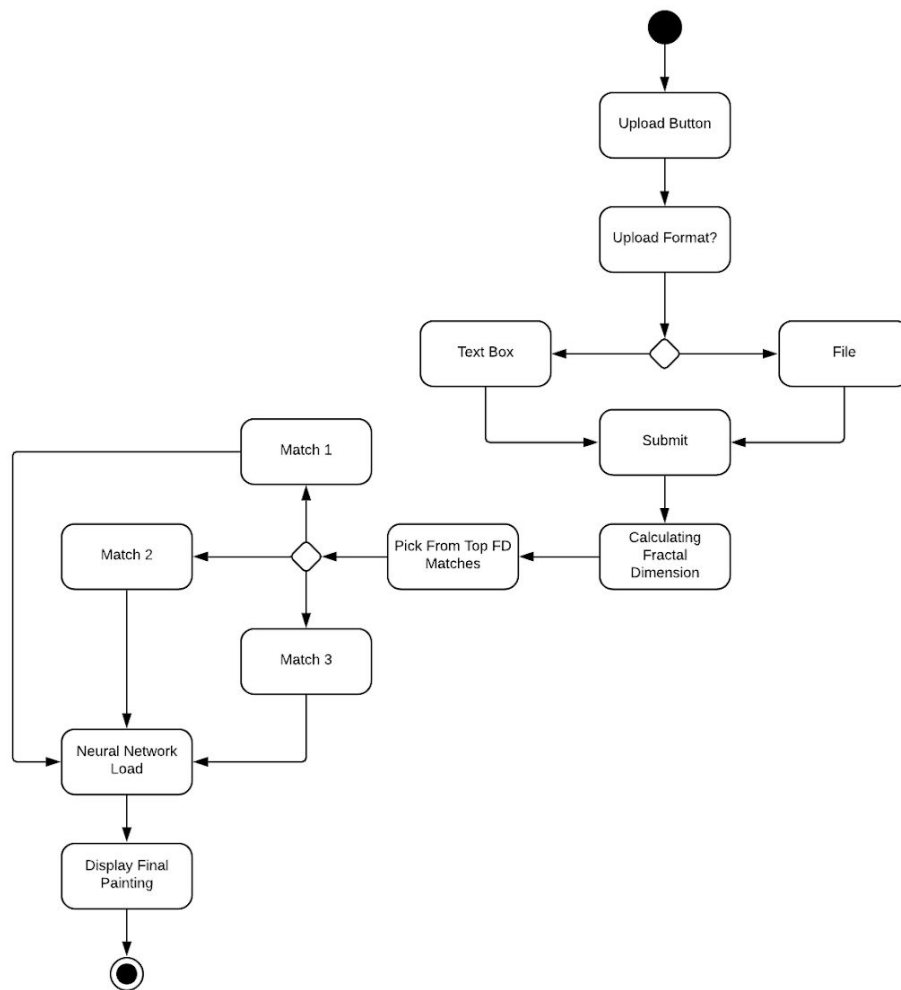
**Alternative/Exceptional Flow:** None

|  |                                    |                               |
|--|------------------------------------|-------------------------------|
| <b>Use Case Name:</b> View Site Statistics   | <b>ID:</b> 08                      | <b>Importance Level:</b> High |
| <b>Primary Actor:</b> Admin  | <b>Use Case Type:</b> Detail, real |                               |
| <b>Stakeholders and Interests:</b><br><b>Admin:</b> Is able to view statistics related to the functioning of the site and carry out actions in response to them.   |                                    |                               |
| <b>Brief Description:</b> An administrator will be able to see various statistics on their profile page from how many submissions there has been to the application to how much space the source file directory is taking up on the machine. |                                    |                               |
| <b>Trigger:</b> An admin travels to their profile page.<br><b>Type:</b> External   |                                    |                               |
| <b>Relationships:</b><br><b>Association:</b> Admin   |                                    |                               |
| <b>Normal Flow of Events:</b><br>1. Admin gains access to profile page<br>2. Heatmap for submission data, table of users, and other various statistics load on the page.<br>The S-1 subflow is performed                                     |                                    |                               |
| <b>Subflows:</b><br>S-1: Admin may interact with user table to either deactivate accounts or make them admins as well.   |                                    |                               |
| <b>Alternative/Exceptional Flow:</b> None  |                                    |                               |

|  |                                    |                               |
|--|------------------------------------|-------------------------------|
| <b>Use Case Name:</b> Add Paintings to Database  | <b>ID:</b> 09                      | <b>Importance Level:</b> High |
| <b>Primary Actor:</b> Admin  | <b>Use Case Type:</b> Detail, real |                               |
| <b>Stakeholders and Interests:</b><br><b>Admin:</b> Has the ability to increase the database’s number of paintings for a better customer experience.   |                                    |                               |
| <b>Brief Description:</b> Paintings can be added to the database from the admin profile by using a form including fields such as the year it was painted, the painter, the title of the piece, and the |                                    |                               |

|  |
|--|
| image file itself.   |
| <b>Trigger:</b> Admin fills out required form fields and clicks 'submit' button.<br><b>Type:</b> External  |
| <b>Relationships:</b><br><b>Association:</b> Admin   |
| <b>Normal Flow of Events:</b> <ol style="list-style-type: none"> <li>1. Admin gains access to profile page</li> <li>2. Admin fills out required fields for painting metadata and supplies image file of painting.</li> <li>3. Admin hits 'submit' button.</li> <li>4. Application receives painting and saves it.</li> <li>5. Painting is sent to python API to have its fractal dimension calculated.</li> <li>6. Painting metadata and file location is updated in database</li> </ol> |
| <b>Subflows:</b> None  |
| <b>Alternative/Exceptional Flow:</b> None  |

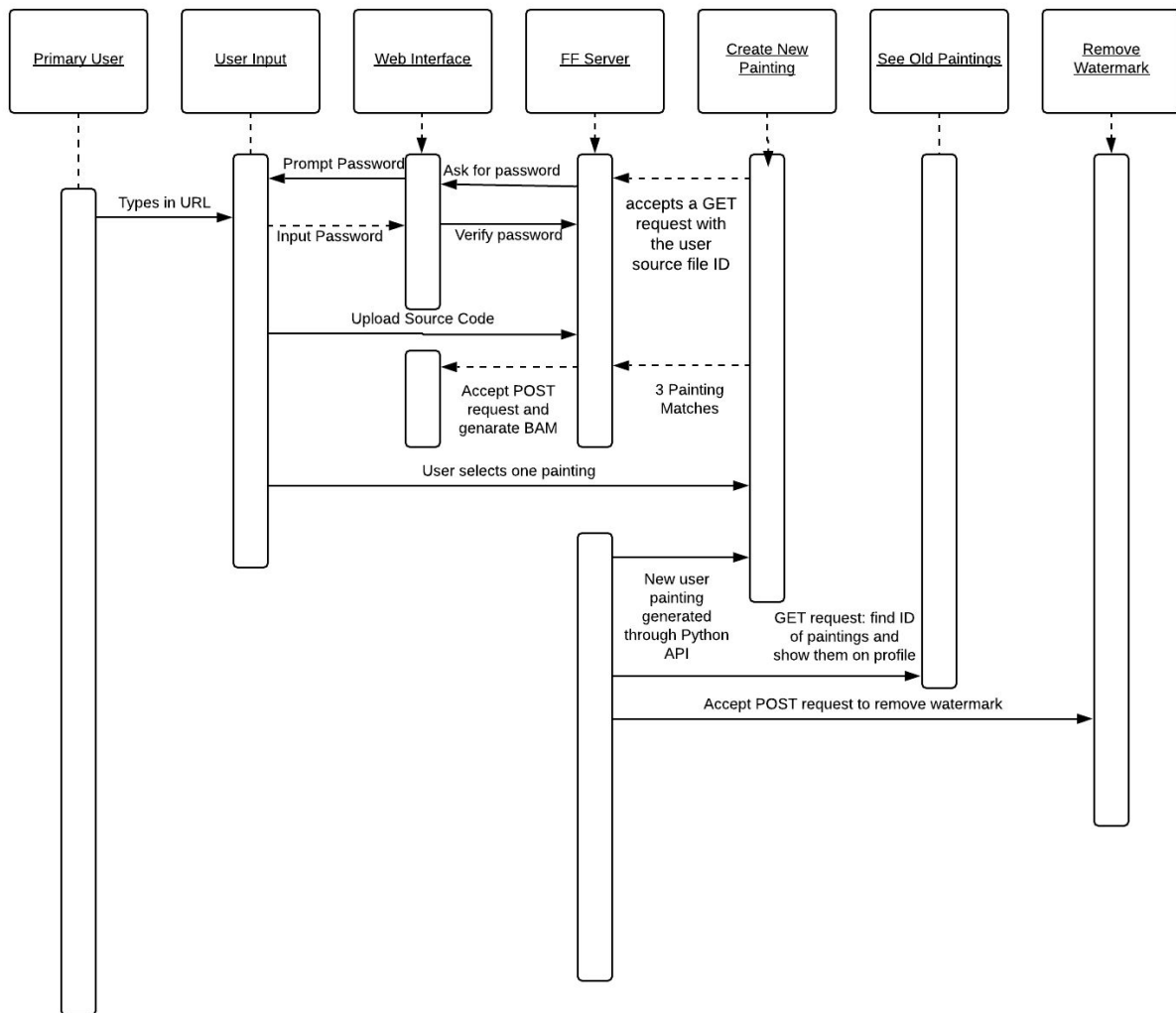
## Activity Diagram



## Activity Diagram Documentation

- User clicks the “Lets Upload” button
  - Navigates to the upload page
- Decides which format they want to upload
  - Text Box
    - Submit button specifically for the text box was implemented
  - File
    - Submit button specifically for file uploads implemented
      - File size limit 1 MB
- Fractal Dimension calculation takes place
  - To the user only a loading screen will appear
- The user is allowed to choose from paintings that match the uploaded code's fractal dimension
  - Three paintings is the limit for how many appear
- Neural network is called
  - To the user only a loading screen will appear
  - May take longer than the fractal dimension calculation
- The final painting is presented to the user

## Sequence Diagrams



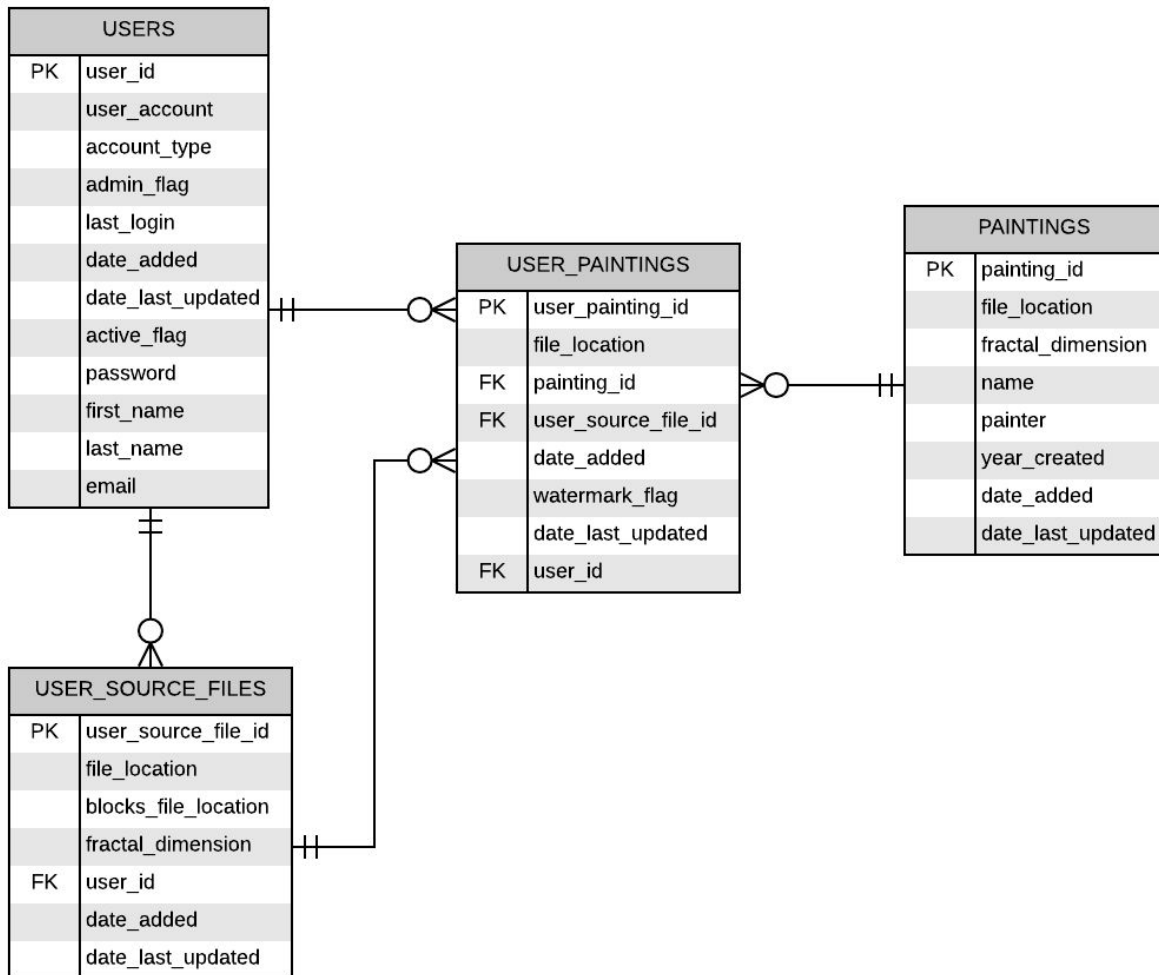


## Sequence Diagram Documentation

- User accesses site and their login is authenticated
  - User is redirected to profile page where they upload source code
- User uploads source code
  - Server accepts POST request and the BAM is created with the Python API and this is when the fractal dimension is calculated.
  - A GET request is accepted with the user source file ID; 3 Paintings are presented to the user that have the closest match to the fractal dimension of the users source code.
  - One painting is selected by user and a new painting is created that is a mix of the old painting and the source code.
  - User can buy painting to remove watermark; a POST request is accepted and the painting will be produced without watermark once payment goes through.
- Profile Page
  - User can see all the paintings they created on their profile page and the paintings are produced through a GET request that finds the ID of paintings in database.

# Software Project Design

## ER Diagram



## Database Documentation

1. **USERS** - This table holds user account information.
  - a. **user\_id** - The *primary key* for the table; it is a unique identifier for every user that signs up and receives an account on the site.
    - Data type: *uuid*
  - b. **user\_account** - The user's username, email, or other identifier that they used during the signup process. This should be unique in combination with the `account_type` field.
    - Data type: *text*
  - c. **account\_type** - Whether the user signed up with the local (default), Marist, Facebook, or Twitter account credentials.
    - Data type: *text*
  - d. **password** - Hash of the user's password for 'default' account type. This is required if `account_type` field value is 'default'.
    - Data type: *text*
  - e. **admin\_flag** - Flag that determines if the user has administrative privileges.
    - Data type: *boolean*
  - f. **last\_login** - Records the user's last login date and time.
    - Data type: *timestampz*
  - g. **date\_added** - The date and time the user is added to the system.
    - Data type: *timestampz*
  - h. **date\_last\_updated** - The date and time the user's account information is last updated.

- Data type: *timestampz*
- i. **active\_flag** - Flag that determines if the user account is deactivated or not.
  - Data type: *boolean*
- j. **first\_name** - First name of user if available.
  - Data type: *text*
- k. **last\_name** - Last name of user if available.
  - Data type: *text*
- l. **email** - Email associated with user if available.
  - Data type: *text*

2. **USER\_SOURCE\_FILES** - This table holds information relating to the source code files uploaded by the user.
  - a. **user\_source\_file\_id** - The *primary key* for the table; it is unique for every source file uploaded.
    - Data type: *uuid*
  - b. **user\_id** - A *foreign key*; it relates the user\_source\_files table to the users\_table (e.g. every piece of code uploaded will belong to one and only one user).
    - Data type: *uuid*
  - c. **file\_location** - Link to the source file that is stored in a separate location.
    - Data type: *text*
  - d. **Blocks\_file\_location** - Link to BAM or source file translated to blocks.

- Data type: *text*
  - e. **fractal\_dimension** - The calculated fractal dimension of the source file.
    - Data type: *numeric*
  - f. **date\_added** - The date and time the source file is uploaded.
    - Data type: *timestampz*
  - g. **date\_last\_updated** - The date and time the source file information is last updated.
    - Data type: *timestampz*
3. **PAINTINGS** - This table holds information about the paintings imported from the Metropolitan Museum of Art's database.
- a. **painting\_id** - The *primary key* for the table; it is unique for every painting.
    - Data type: *uuid*
  - b. **file\_location** - The link to the painting that is stored in a separate location.
    - Data type: *text*
  - c. **fractal\_dimension** - The calculated fractal dimension of the painting.
    - Data type: *numeric*
  - d. **name** - The name of the painting.
    - Data type: *text*

e. **painter** - The name of the painter. Due to the process of obtaining the name of the painter from the MET's API or from admin input with varying degrees of accuracy, the name will be considered atomic as it is given; Value may be 'unknown' for this reason.

➤ Data type: *text*

f. **year\_created** - The year that the painting was created. May be 'unknown'.

➤ Data type: *text*

g. **date\_added** - The date and time that the painting is added to the database.

➤ Data type: *timestampz*

h. **date\_last\_updated** - The date and time that the painting information is last updated.

➤ Data type: *timestampz*

4. **USER\_PAINTINGS** - This table holds information on the paintings that are created from the user's source file and selected painting.

a. **user\_painting\_id** - The *primary key* for the table; it is unique for every painting created by the user's source code.

➤ Data type: *uuid*

b. **painting\_id** - A *foreign key*; it relates the user\_paintings table to the paintings table (e.g. every user painting created will have come about from one and only one painting).

➤ Data type: *uuid*

- c. **user\_source\_file\_id** - A *foreign key*; it relates the user\_painting table to the user\_source\_file table (e.g. every user painting will be created from one and only one user source file).
  - Data type: *uuid*
- d. **user\_id** - A *foreign key*; it relates the user\_source\_files table to the users\_table (e.g. every piece of code uploaded will belong to one and only one user).
  - Data type: *uuid*
- e. **file\_location** - The link to the user painting that is stored in a separate location.
  - Data type: *text*
- f. **watermark\_flag** - Flag that determines whether the painting should be displayed with or without a watermark.
  - Data type: *boolean*
- g. **date\_added** - The date and time that the user painting is added to the database.
  - Data type: *timestampz*
- h. **date\_last\_updated** - The date and time that the user painting is last updated.
  - Data type: *timestampz*

## User Interface

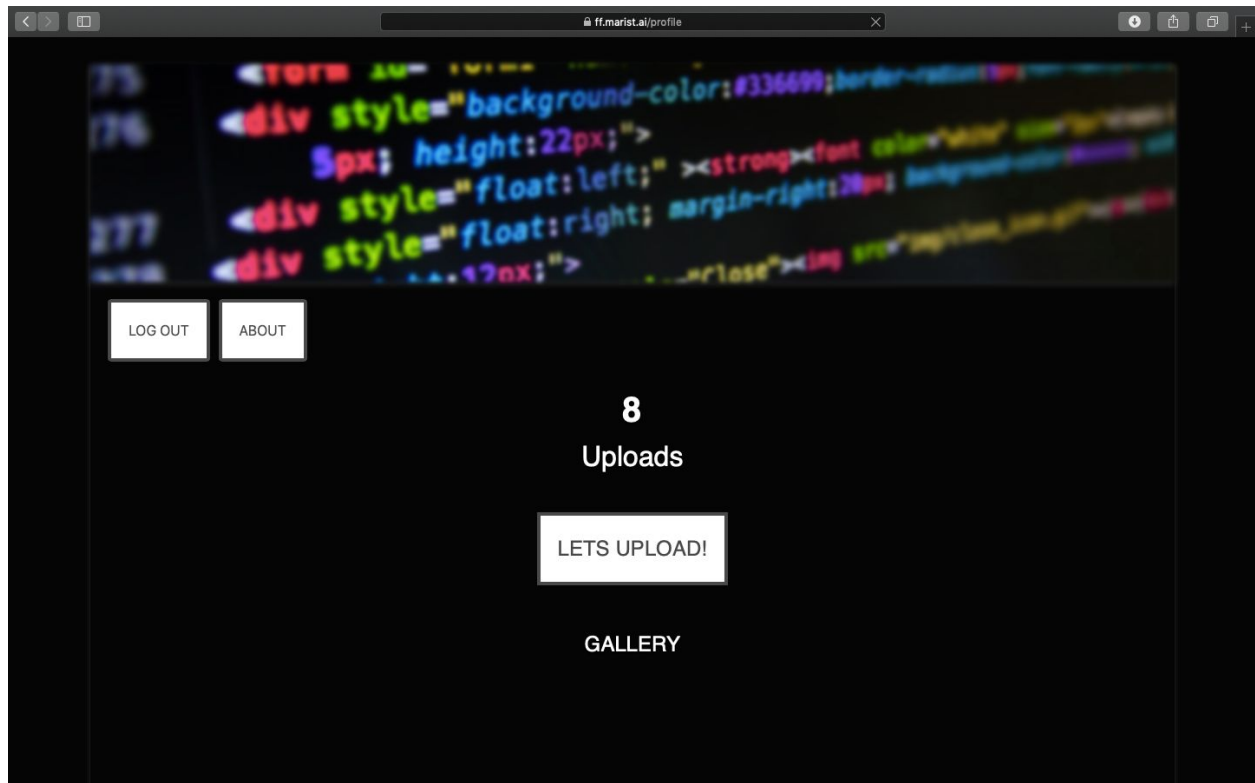
### Login Page:



The user is greeted by the Fractal Factory home page, where they have the option to sign in with a local account (a simple username and password combination), their Marist credentials, their Facebook credentials, or their Twitter credentials. There is also a button that leads the user to the about page which explains in further detail what the web application is meant to do.

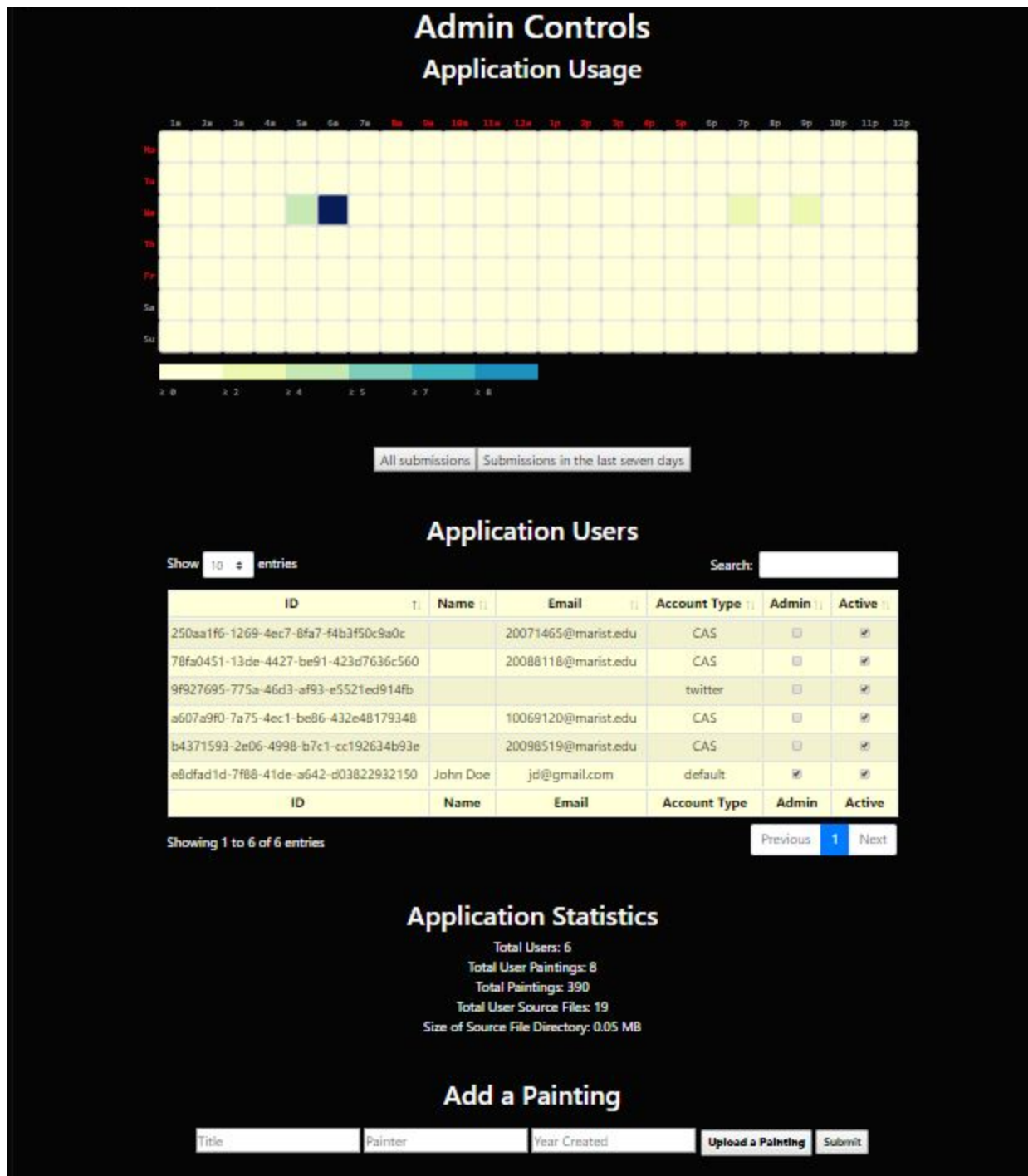


## User Profile Page:



The profile page displays how many uploads the user has as well as the actual paintings that were created from their source code. The log out button is present in the top left which takes the user back to the login screen. The about button again navigates the user to the page that explains exactly what the web application actually does. The 'Lets Upload' button takes the user to the upload page where the user can upload their source code using a text box or a file.

## Administrator Profile Page:



If the user is an administrator on the web application, they will be given access to the administrator tools. This is where they can see the application users, application statistics as well as upload new paintings into the database. Admins are also able to deactivate user accounts and grant other accounts admin privileges from here.

## Upload Page:

LOG OUT

BACK TO PROFILE

# Fractal Factory Upload

Choose a file Submit

Paste Code Here...


Submit Text Area

The upload page shows the user two ways they can upload code, either through the text box or through a file on the user's computer. Two separate buttons are being used to upload; one is specifically for the text box and the other is for the file selection. The log out button is present on the top left for easy access to the user and the 'Back to Profile' button is there so the user can go back and see what they have already created. Once source code has been uploaded, the user will be redirected to the results page where they will be given the option of 3 paintings to choose from that have fractal dimensions closest to the source code's BAM fractal dimension. Clicking on one of the paintings chooses one to use in generating a new painting, and the user will be redirected to the purchase page.

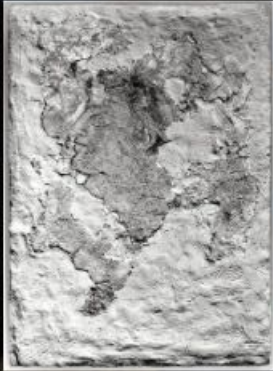
## Results Page:

[LOG OUT](#)[BACK TO PROFILE](#)


# Here are your Matches!!



Title: William Conklin  
Painter: Chinese Painter  
Year: 1800



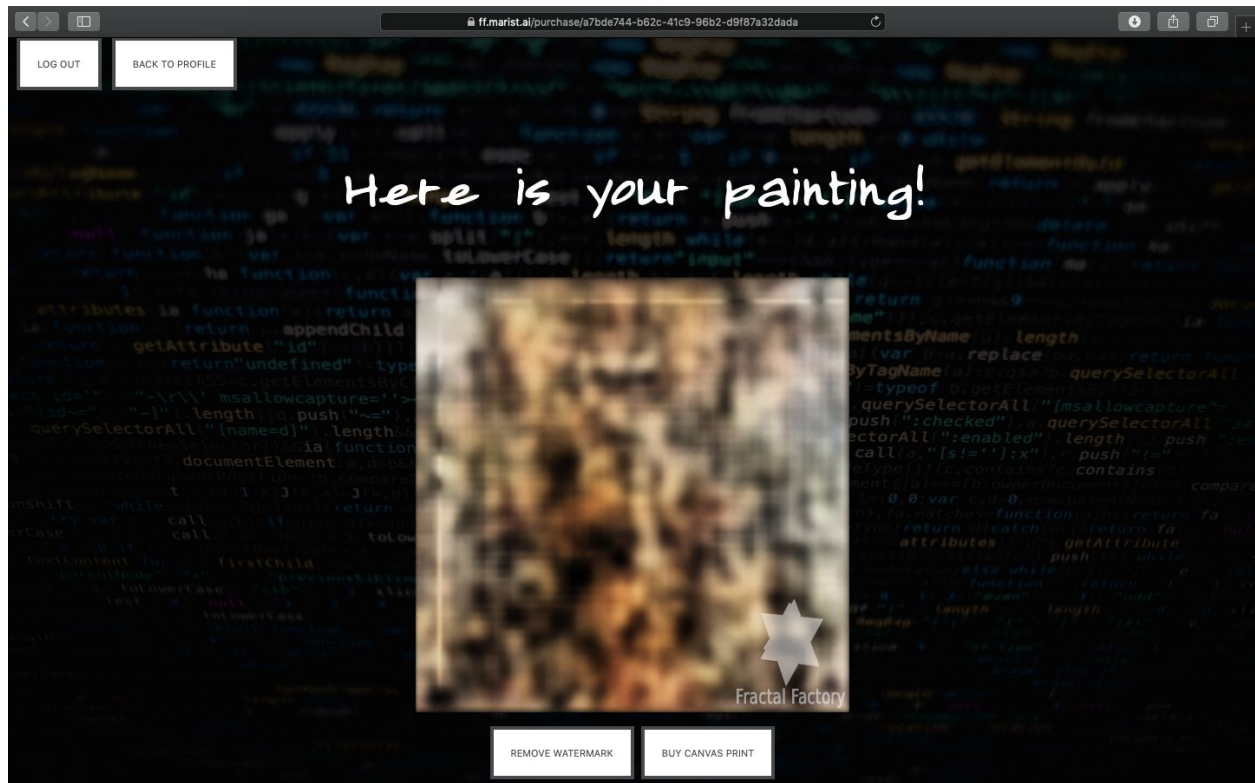
Title: Fragment of wall painting: male head  
Painter: Unidentified Artist  
Year: 700



Title: Pair of Hindu Manuscript Covers  
Painter: Unidentified Artist  
Year: 1199

The results page gives the user a choice of three paintings to choose from along with some metadata about the painting retrieved from the Metropolitan Museum of Art's API. The paintings have fractal dimensions that had the closest values to the source code's BAM fractal dimension. Clicking on one of the paintings will choose it as the last input needed to generate a new painting with the variational autoencoder neural network. The user will be redirected to the purchase page with the new painting once it is generated; a loading screen is placed on the page while they wait).

## Purchase Page



The purchase page shows the final painting that was created using the painting chosen by the user and the code that was submitted by the user. The log out button and the 'Back to Profile' button is present on the top left corner for easy access. On this page, the user has the option to remove the watermark using bitcoin through a third party, Coinbase, as well as the option to buy their created piece of art through a third party, CanvasPop.


## Coinbase Page:

The screenshot shows a web browser window with the URL `commerce.coinbase.com/checkout/edfc4b8e-886a-4199-862f-5a21e9af3d51`. The page features a dark header with a fractal image and the text "Fractal Factory: Pai..." and "Paintings produced by fractal-dimension from...". Below the header, a white box contains the text "Please enter the following information:". There are two input fields: "Full name" with a person icon and "Email". At the bottom of the white box is a blue "Continue" button. Below the white box, the Coinbase logo and "Commerce" text are visible.

The Coinbase page is where the user would enter their credentials in order to purchase the removal of the watermark. The user follows the prompts on the screen and will be redirected back to the purchase page with their final artwork where the watermark will then be removed.

## CanvasPop Page:

Fractal Factory  
powered by canvaspop



On the fence? Click here to see more about our poster prints.


Product

Poster

Size

10" x 10" \$12.00

Medium



Quantity

1

Total

\$12.00

Continue

or buy later

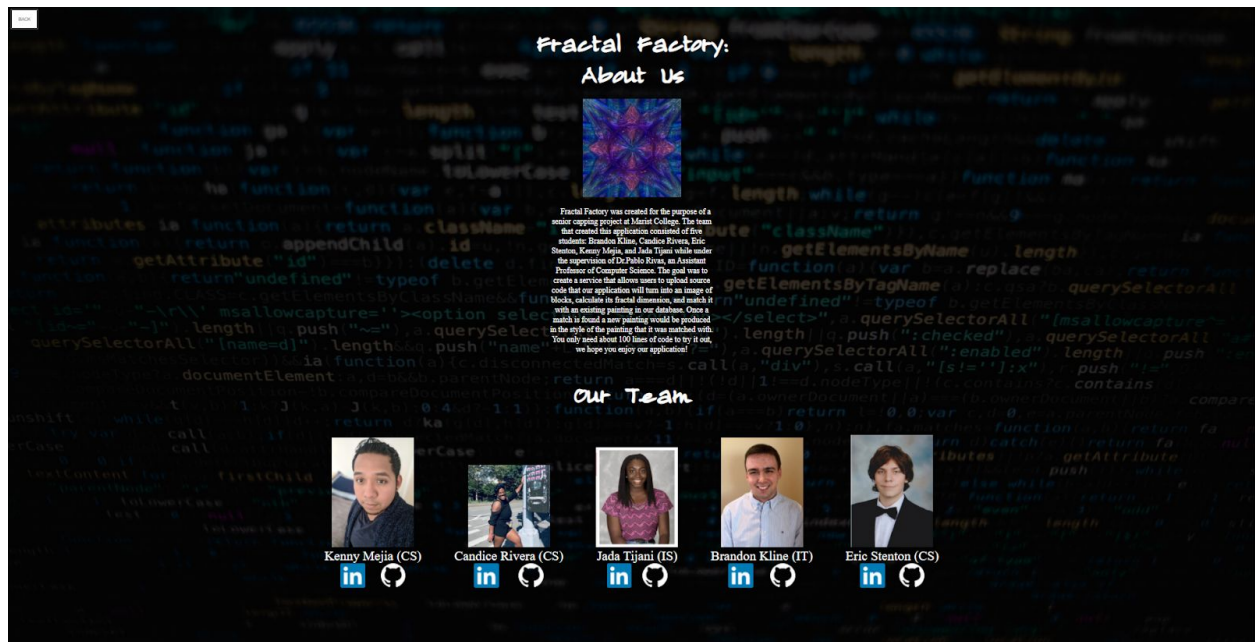
[Terms](#) [Privacy](#)

Copyright © 2019 CanvasPop

The CanvasPop page provides the user with access to the third party's printing services. The user can choose from a variety of printing options and submit an order. The user will be redirected to the purchase page of the main application.



## About Page:



The about page provides insight into the application and the team that worked on it. The 'About Us' section describes the inspiration for Fractal Factory and how it came to be. The 'Our Team' section shows the members of the development team, their roles, and links to their LinkedIn and GitHub pages.



# **Infrastructure Specifications**

## **IT Requirements**

- Server Platform (for each “server” required)
- Physical system requirements // Both servers are directly attached to the Marist network, with ff.marist.ai being publically accessible for front end communication and an additional server for backend processing.

Storage capacity:

- 28 GB (ff.marist.ai)
- 28 GB (Python API)

Speed requirements / response time parameters:

- < 3 sec for navigation, < 10 sec for fractal dimension calculation, < 30 sec for encoder painting generation

Scalability plans:

- Due to the difficult nature of interfacing a private backend VM with a public physical server, as well as the inability to use the previously allocated physical servers in the Hancock data center, there are currently no plans for scalability beyond the mainframe and server that are currently operating. However, the application is entirely portable to another machine should that need arise.

Physical system requirements :

- 1 CPU, 579 MB Memory (ff.marist.ai)
- 4 CPU, 16 GB Memory (Python API)

OS supported:

- Ubuntu 18 (Server + Desktop)

Number of servers:

- 2 (ff.marist.ai front-end production and private database server and mainframe hosting Python API)

## **Connectivity**

Network considerations :

- SSH via PuTTY

Interconnection to what other systems :

- MET database, variational autoencoder neural network

## Reliability

### **Service Level Agreements**

Uptime requirement :

- Continuous outside of 30 mins for package updates weekly and major software/hardware updates

Response time requirements :

- < 3 sec for navigation, < 10 sec for fractal dimension calculation, < 30 sec for encoder painting generation

## Recoverability

Where are things backed up? How often?

- Paintings and generated user paintings are all stored on the back-end mainframe.

Access to backups?

- Only admins

What data is transient and doesn't need to be stored longer term?

- Usage statistics can be derived from data stored for long term, so there is no need to store it.

## Security and Privacy

Database :

- PostgreSQL 11

Access controls by userID/roles userIDs and user/admin accounts :

- Users have access to personal account (can view past images, upload and save new images, pay for full size images)
- Admins have full access across site.

### **Update vs. Access**

Account information :

- Account info stored and accessed through database

User data:

- Email, first name, last name, username, password

Personal / registration :

- Registration through Facebook, Marist CAS, Twitter, or locally with username and password

### **Saved courses information**

FERPA/Privacy considerations :

- Private profile
- Storing user's code

Admin access controls :

- Deactivate users
- Manage administrators
- View usage statistics
- Add paintings

Adding new users, deactivating old:

- Add and deactivate from database

## **Maintenance**

### **Planned down time requirements**

Database maintenance:

- Daily backups to mainframe, weekly security/package updates

Times of year when IT does maintenance:

- With each new major update to software and new hardware

Times of year when the systems are not available:

- Marist maintenance periods and major hardware/software upgrades

# Prototype Deployment Instructions

## Github Repository Link

<https://github.com/StentonRR/Fractal-Factory>

## Accessing Project in Production

### Nodejs Application:

1. Connect to ff.marist.ai machine (35.231.105.129).
2. Change directory to /home/developer (This is where the application files are stored).
3. As sudo user, run the command *pm2 list* to show the application running and being managed by the node module PM2.
4. As sudo user, run the command *pm2 log* to view the log output of the application.

### Python API:

1. Connect to mainframe machine (148.100.33.25).
2. Navigate to directory the Flask API is stored.
3. PS command can be used to validate API is operational and running on the correct port (user may need to enter sudo mode if the API was started with sudo to see the process).

### Database:

1. Connect to ff.marist.ai machine (35.231.105.129).
2. Login to postgres user: *sudo su postgres*.
3. Run the command *psql -h localhost -d fractal\_factory -U ffadmin* and input the password 'ff66z7' to log into the production database as the 'ffadmin' user.
4. Access to production database through the command line should be granted at this point. Further commands can be found [here](#). Note that SQL commands can also be used in this command line interface.

## Testing Functionality

\*Note: Some endpoints require authentication or admin status. Also, consider modifying the '.env' file variables to reflect a testing environment such as pointing to another database and setting the 'NODE\_ENV' variable to 'development'.

### Registration/Login:

- The following are the Nodejs application's endpoints associated with both login and registration:
  - Local: /login & /register
  - Facebook: /auth/facebook & /auth/facebook/callback
  - Twitter: /auth/twitter & /auth/twitter/callback
  - CAS: /auth/cas & callback.<host>
- The local login consists of a POST containing the username and password which is handled by the passport library to authenticate and gain a session. The local registration consists of a POST containing the requested username, password, first name, last name, and email of the user (Note: The Facebook, Twitter, and CAS login/registration is handled through the passport library, thus it is abstracted from the main application. Both the login and registration is handled by the same endpoint with the callback being where the third party app redirects).
- To test local login, either use the front-end login page or an application capable of sending http requests (such as Postman) to send a POST to the /login endpoint of the application. The request should have in the body a 'username' and 'password' field filled out. A successful login will result in a redirect to the profile page. On the front-end of the login page, a message will have been inserted into the html describing the success or failure of login. In the database, the user's row within the 'user' table should will have the 'last\_login' field updated to reflect the current time of access.
- To test local registration, a POST should be sent to the /register endpoint. The body of the request should include the following fields filled out: 'username', 'password', 'firstname', 'lastname', 'email'. A successful registration will redirect to the login page again. On the front-end of the login page, a message will have been inserted into the html describing the success or failure of registration. In the database, a new row in the 'user' table will be created for the user with the 'account\_type' field marked as 'default'.

## Uploading Source Code:

- The following are the Nodejs application's endpoints associated with uploading source code:
  - /upload
  - /uploadText
- The '/upload' endpoint accepts a POST request containing a form in its body. The form should contain a single file of MIME type 'text'. If the file is successfully received, the application will save it to the specified directory for source files in the '.env' file, generate a BAM image of the source code, save the BAM image to the specified directory for BAM images, and calculate the fractal dimension of the BAM image. The user is then redirected to the results page. A database entry for the source file will be created in the 'user\_source\_files' table with each field filled out. If a field is null, then this denotes an error occurred in the process most likely dealing with the python API. A section on the python API is described later.
- The '/uploadText' endpoint accepts a POST request containing a single key called 'text' in the body. The value should be a string of source code. The following process is the same as the previous bullet point where it generates a BAM image and calculates its fractal dimension using the python API. The user is then redirected to the results page. Again, a database entry should be created for the source file.
- To test either of the aforementioned endpoints, Postman or another similar application that can send custom requests can be used to access the endpoints. The directories and database can then be monitored for the creation of files and new entries respectively.

## Creating a New Painting:

- The following is the Nodejs application's endpoint associated with creating user paintings:
  - /generate-user-painting/:userSourceFileId/:paintingId
- The above endpoint accepts a GET request with the user source file ID and painting ID in the URL. Using the BAM file associated with the source file ID and the painting with the painting ID, a new user painting will be generated through the python API. The painting file will be stored in the specified directory in the '.env' file. A database entry should also be created in the 'user\_paintings' table. The user will be redirected to the purchase page.
- To test, send a GET request to the above endpoint with the 'userSourceFileId' and 'paintingId' portions substituted with existing ID's

from the database. Examine the user paintings directory for a new file and a new entry in the database.

### Python API:

- The following are the python API's endpoints
  - /fractal-dimension
  - /generate-bam
  - /create-painting
- The 'fractal-dimension' endpoint accepts a POST containing a form in its body. The form should include a 'file' field containing the image that is to have its fractal dimension calculated as well as a 'type' field to denote whether it is a 'bam' or a 'painting'. To test the endpoint, send a POST request containing the aforementioned fields to the endpoint on the python API machine (the mainframe in our case). A JSON object will be returned containing the calculated fractal dimension of the provided image.
- The 'generate-bam' endpoint accepts a POST containing a form in its body. The form includes a single field called 'file' that contains the text file to be used to create the BAM image. The response will contain the generated BAM image in the body. To test, send a POST to the endpoint containing a file in the aforementioned manner and examine the response for the BAM image.
- The 'create-painting' endpoint accepts a POST containing a form in its body. The form contains a 'userSourceFile' and 'paintingFile' field that each hold a file; a text file of code and a painting image respectively. The response will contain the new painting image in the body. To test, send a POST with the file in the body as previously stated and ensure an image is received in the response.

### Other Endpoints:

- The following are other endpoints within the Nodejs application with a description of each and how to test their function:
  - /admin-status
    - Description: Endpoint accepts a POST request with an 'affectedUserId' and 'adminStatus' field in the body. Its function is to toggle the specified user's admin flag to either true or false as defined in the body.
    - Testing: Send a POST request to the endpoint with the aforementioned fields filled out. A user's ID can be obtained through the database or the administrative profile page.

- /active-status
  - Description: Endpoint accepts a POST request with an 'affectedUserId' and 'activeStatus' field in the body. Its function is to toggle the specified user's active flag to either true or false as defined in the body.
  - Testing: Send a POST request to the endpoint with the aforementioned fields filled out. A user's ID can be obtained through the database or the administrative profile page.
- /profile
  - Description: Not to be confused with the page route, this endpoint handles new paintings being added to the database from the administrative profile page. It accepts a POST request with a form in the body containing a file which is the new painting image, a 'name' field that is the title of the painting, a 'painter' field that is the name of the painter, and a 'yearCreated' field that is the year the painting was created.
  - Testing: Send a POST request to the endpoint with the previously mentioned form with its fields in the body. The response should be a redirect back to the profile page. To see if the painting was correctly added, check the database for a new entry with the provided details of the new painting.
- /watermark
  - Description: This endpoint's function is to remove a watermark from a painting. It accepts a POST request that contains a single field in the body labeled 'paintingId' that contains the ID of the painting to remove the watermark from.
  - Testing: Send a POST request to the endpoint with the painting ID in the body as stated above. The database can be examined to ensure the watermark flag is toggled from true to false.
- /logout
  - Description: This endpoint's purpose is to log out a user. It accepts a DELETE request.
  - Testing: Using the front-end application is the easiest way to test the logout endpoint through using one of the buttons due to the use of session cookies to tell if a user is logged in and who is logged in.
- /user-painting/:id



- Description: This endpoint's function is to access a generated user painting. It accepts a GET request where 'id' is replaced with the ID of the user painting.
- Testing: Send a GET request to the endpoint with a user painting ID obtained from the database. The response will contain a binary buffer of the user painting.
- /painting/:id
  - Description: This endpoint's purpose is to access a painting by its ID. It accepts a GET request where 'id' is replaced with the ID of the painting.
  - Testing: Send a GET request to the endpoint with a painting ID obtained from the database. The response will contain the painting in the form of an image file.
- /heatmap
  - Description: This endpoint returns statistics used to build the heatmap on the administrative profile page.
  - Testing: Send a GET request to the endpoint to get a JSON response with the heatmap data.

## Credentials

### CanvasPop:

Username/Email: fractalfactory5@gmail.com  
Password: fractal5!

### CoinBase:

Email: fractalfactory5@gmail.com  
Password: fractal5!

### Postgres Database:

Username/Email: ffadmin  
Password: ff66z7

### Default Admin Account:

Username/Email: ffadmin  
Password: }9TJu6C!Bu(s)Nfx

### Facebook:

Username/Email: fractalfactory5@gmail.com  
Password: fractal5!

### Twitter:

Username/Email: fractalfactory5@gmail.com  
Password: fractal5!

## Deployment Instructions

\*Note: The following instructions are for a Linux based machine, particularly Ubuntu. Modify instructions for other OS's, flavors of Linux, and machines as needed.

### Database:

1. Connect to ff.marist.ai machine (35.231.105.129)
2. Install Postgres 12 as sudo user following method 2 in the following link:  
<https://itsfoss.com/install-postgresql-ubuntu/>
  - a. Add GPG key: `wget --quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc | sudo apt-key add -`
  - b. Add repository: `sudo sh -c 'echo "deb http://apt.postgresql.org/pub/repos/apt/ `lsb_release -cs`-pgdg main" >> /etc/apt/sources.list.d/pgdg.list'`
  - c. Update: `sudo apt update`
  - d. Install: `sudo apt install postgresql postgresql-contrib`
3. Become postgres user: `sudo su postgres`
4. Activate postgres command interface: `psql`
5. Give postgres database user a password: `ALTER USER postgres WITH PASSWORD 'password';`
6. Create a new user (admin users and regular users as needed): `CREATE USER ffadmin WITH PASSWORD 'ff66z7';`
7. Make new admin user a superuser: `ALTER USER ffadmin WITH SUPERUSER;`
8. Create a database and run the SQL 'create' statements from the Github repository. Be sure to add the UUID plugin first; this only needs to be run once
9. Exit the psql interface (`\q`) and open the file `'/etc/postgresql/12/main/postgresql.conf'` with an editor (Note the version of postgres may be different)
10. Under the 'Connections and Authentication' heading, modify `'listen_addresses'` to include the IP's of the machines allowed to connect to the database. If any IP is allowed, put the `'*'`.
11. In the same directory, open the file called `'pg_hba.conf'` in an editor. Modify connection settings at the bottom of the file under 'IPv4 local connections' to define where users are allowed to log in from. Example (allows any host to connect to all databases as the ffadmin user with password): `host all ffadmin 0.0.0.0/0 md5`
12. Restart Postgres service: `/etc/init.d/postgresql restart`

## Python API:

1. Connect to mainframe machine (148.100.33.25)
2. Clone the application down to the machine from Github: *git clone <https://github.com/StentonRR/Fractal-Factory.git>*
3. Move the folder 'pythonApi' out of the main application folder. Main application folder can be deleted.
4. Install Python 2.7 if not already installed
  - a. `sudo apt install python-pip`
  - b. `sudo apt install python2.7`
5. Pip install the following libraries and their dependencies: numpy, scikit-image, keras, gdown, pillow, imageio, matplotlib, flask, python-dotenv, waitress
6. Start the Flask server by running *python app.py* in the pythonApi directory. Check output to ensure the server started correctly.

## Nodejs Application:

1. Connect to ff.marist.ai machine (35.231.105.129)
2. Clone the application down to the machine from Github: *git clone <https://github.com/StentonRR/Fractal-Factory.git>*
3. Create a copy of the '.env.example' file and call it '.env'. Put the following configuration into it (modify as needed):

```
# .env
NODE_ENV=production
PORT=80
SSLPORT=443
HOST=https://ff.marist.ai
# Program
PRIVKEY=/etc/letsencrypt/live/ff.marist.ai/privkey.pem
CERT=/etc/letsencrypt/live/ff.marist.ai/fullchain.pem
LOGFILE=server/logs/fractalFactory.log
LOGLEVEL=ALL
COOKIEKEY1=df1jhgljdjsgfljlj573498439875kdhngf
COOKIEKEY2=hk6TPBgghdpjCI1x9oG1RdIQHNzSE
TOPPAINTINGNUMBER=3
PYTHONAPI=http://148.100.33.25:8050
# Database
DBUSER=ffadmin
DBPASSWORD=ff66z7
DBHOST=localhost
DBPORT=5432
```

```

DBNAME=fractal_factory
# file locations
USERPAINTINGDIRECTORY=/home/developer/sourcePaintings/
PAINTINGDIRECTORY=/home/developer/paintings/
USERSOURCEFILEDIRECTORY=/home/developer/sourceFiles/
BLOCKFILEDIRECTORY=/home/developer/blocks/
WATERMARKFILE=./watermark.png
# CanvasPop Access Key
CANVASPOPKY=bf29cafbbc740f88ddd29232b7a6263e
# Facebook Developer Keys
FACEBOOKCLIENTID=2298137900477591
FACEBOOKCLIENTSECRET=a16031bbc57e1f3d8804a3ca0fe48943
FACEBOOKCALLBACKURL=https://ff.marist.ai/auth/facebook/callback
# Twitter Developer Keys
TWITTERCLIENTID=WMbPVa8uJ0zhAV4ejyDaCK1Wv
TWITTERCLIENTSECRET=3sDEjtM54FzqRHhn4tkxt00sUiOhAyuOfFSrv
zTYOUPOujSUln
TWITTERCLIENTCALLBACKURL=https://ff.marist.ai/auth/twitter/callback
# Cas URLs
CASMARISTCALLBACK=https://callback.ff.marist.ai

```

4. Make sure directories defined in '.env' file exist.
5. Create a certificate and key following the provided article:  
<https://itnext.io/node-express-letsencrypt-generate-a-free-ssl-certificate-and-run-an-https-server-in-5-minutes-a730fbe528ca>
6. In the Fractal-Factory directory, run the command (Ensure this is ran before installation. It makes sure a patched version of Cheerio is installed for the passport-cas2 library): *npm run preinstall*
7. In the same directory, run the following command to install the node modules: *npm install*
8. If the database needs to be filled with paintings, run the command in the same directory (where number is the amount of paintings you want added. Note: Make sure to check database and ensure all information for each column has been gathered in case of any errors or timeouts): *npm run addPaintings <number>*
9. Install PM2 globally on the machine (This will manage our application):  
*npm install pm2 -g*
10. Run the following command in the directory to start the application: *pm2 start ./server/fractalFactory.js*
11. Run the following command to check if the application started correctly:  
*pm2 list*

12. Run *pm2 log* to view the manager log. The application should have printed the two ports it is listening on (http and https). The front-end of the application should now be accessible.

## Cost Analysis / ROI

### Cost of Labor

- 3 Developers 30 hrs a week -15 weeks
- 1 Systems Administrator 15 hrs a week- 15 weeks
- 1 Project Manager 30 hrs a week - 15 weeks

The average salary for a Software Developer in New York, New York is \$88,690.  
The average salary for a Senior Systems Administrator in New York, New York is \$95,729.  
The average salary for a Project Manager is \$98,659

|                               |             |
|-------------------------------|-------------|
| <b>3 Developers</b>           | \$22,172.50 |
| <b>1 System Administrator</b> | \$23,932.25 |
| <b>1 Project Manager</b>      | \$24,664.75 |
| <b>Total:</b>                 | \$70,769.50 |

### Cost of Site

\* Note: Numbers are based off of Google VM pricing for N1 standard machines.

- Server:
  - Google VM 28 GB Storage, 1 vCPU and 579 MB Memory
- Mainframe:
  - `4 vCPU, 28 GB Storage and 16 GB Memory

| Server (per month) | Mainframe (per month) | Total (per month) |
|--------------------|-----------------------|-------------------|
| \$14.54*1 vCPU     | \$14.54*4 vCPU        | \$72.70           |
| \$0.40 * 28 GB     | \$0.40 * 28 GB        | \$22.40           |
| \$1.95*0.579       | \$1.95*16             | \$32.33           |
| \$26.87            | \$100.56              | \$127.43          |

## ROI

- CanvasPop offers the ability to add a markup percentage to any of the prints bought through their API. The default markup rate is 30%, however it can be lowered or raised. Monthly amount of money from this method will be symbolized with 'PP'. PP will be variable due to being based on how many prints are bought, what kind of print is bought, and what the markup rate is in a given month.
- The Coinbase payment interface for watermark removal can be set at any price through the provided Coinbase account in an earlier section. Monthly amount of money from this method will be symbolized with 'WP'. WP will be variable due to being based on how many watermark removals are bought in a given month.
- The above two methods are the primary ways the application can make money. The return on investment (per month) can be calculated by the following equation:
  - $ROI = PP + WP - 127.43 / 127.43 * 100$
- The team shows no interest in the continued management of the project, so the current CanvasPop markup rate is 0% and the Coinbase price for watermark removal is set to a negligible amount leading to an estimated ROI of -100%. The CanvasPop markup rate and Coinbase price can be increased to a desired amount through the accounts provided in an earlier section to begin increasing the ROI, however the amount of increase is variable based on the aforementioned considerations.
- If we are to estimate a potential ROI, the following assumptions can be made:
  - Suppose about 30 users interact with the site in a single month period.
  - The CanvasPop markup rate is set to the default, 30%.
  - The Coinbase price to remove a watermark is set to \$5 worth of bitcoin.
  - Each user pays to remove at least one watermark. ( $30 * \$5 = \$150$ )
  - Each user opts to purchase at least one print. Half of users purchase a 10" x 10" poster for \$12 and the other half purchase a 10" x 10" canvas with a .75" depth for \$21. ( $(15 * \$12 + 15 * \$21) * .3 = \$148.50$ )
  - Estimated ROI =  $\$150 + \$148.50 - 127.43 / 127.43 * 100 = \mathbf{134.25\% \text{ per month}}$



## Revised Project Plan

|           |  |  |                      |   |
|-----------|--|--|----------------------|---|
| 9/11/2019 | Assignment #1 Due                              | PDF including questions, user requirements, project plan, and UML use case diagram due                                     | All                  | Finalize and push PDF to Github                                     |
| 9/12/2019 | Wireframes                                     | Begin work on wireframes/mockups of front end pages  | Kenny, Candice       |   |
| 9/15/2019 | IT Requirements                                | Finalizing IT requirements and creating PDF to push  | All                  |   |
| 9/16/2019 | Meet with client: E-R Diagram, IT Requirements | Go over E-R diagram and IT requirements  | All                  | Eric - E-R diagram; Kenny, Brandon, Candice, Jada - IT Requirements |
| 9/18/2019 | Homework #2 Due                                | E-R diagrams complete with supporting documentation if explanation is required for anything you document in your diagrams. | All                  | Finalize and push E-R diagram and IT requirements                   |
| 9/19/2019 | Front-end work                                 | Begin work on front end, HTML and CSS to create the pages needed   | Kenny, Candice, Jada | Design sign-in page, profile page, about us page, results page      |
| 9/23/2019 | Database Design Meeting                        | Finalize ideas on best design for database and data generating   | All                  | Final details and gather sample data to populate database           |
| 9/25/2019 | Homework #3 Due                                | Complete mock-ups (ie. wireframes) of your user interface.   | All                  | Mockups completed and pushed to Github                              |
| 9/26/2019 | Database Testing                               | Ensure database is working and queries are outputting correctly  | Eric                 |   |
| 10/2/2019 | E-R diagram and Functional                     | Database system design completed and   | All                  | Functional database with sample data pushed to github               |

|            |                                       |   |             |   |
|------------|---------------------------------------|---|-------------|---|
| 10/3/2019  | Back-end work                         | Back-end coding started   | Eric, Kenny | Upload, neural networks, fractal calculator, code block converter, statistic tracking   |
| 10/7/2019  | Review back-end                       | Ensure that any problems with neural network and other pieces are worked out                                      | Eric, Kenny |   |
| 10/14/2019 | Back-end final review                 | Review all parts of back-end and test to make sure everything works as intended                                   | Eric, Kenny |   |
| 10/16/2019 | Early Demo                            | Early demo due and presented  | All         | User interfaces connected and navigable, a proper connection to the DBMS, all external APIs and accounts tested, implemented test cases in a functional stage |
| 10/16/2019 | Mid-semester peer reviews due         | Complete initial (ie. first pass feedback) peer reviews   | All         |   |
| 10/30/2019 | Homework #4 Due                       | Database prototype complete   | All         |   |
| 11/6/2019  | First Demo of Prototype               | Complete project prototype first-pass demo ready  | All         |   |
| 11/13/2019 | Project Portfolio Assigned            | Begin discussing and working on project portfolio   | All         |   |
| 11/20/2019 | User Validation & Test plan finalized | Test plan for all aspects of the prototype complete<br>User validation tests to be performed by client documented | All         |   |
| 11/25/2019 | Client Visit                          | Updates to user validation complete (as required)<br>One key question to ask the client documented                | All         |   |

# **Ethics Essay**

## **Introduction**

Fractal Factory (FF) is an application that takes the fractal dimension of a user's code, matches it to the fractal dimension of a painting in our database, and creates a new piece of art based on both the code and the art piece. This application requires users to either sign in with their Marist, Twitter, or Facebook accounts or simply sign up with their emails. Because of this, it requires that some parts of their data be collected and stored in our database which then brings up some possible ethical dilemmas. Issues such as user privacy, pornography, and final ownership are the main dilemmas that require a discussion.

## **User Right to Privacy**

When a user logs into Fractal Factory, they are agreeing to our Terms of Use as well as our Privacy Policy. The Terms of Use gives users a thorough explanation of what they are agreeing to when they log into the application as well as how their information will be used and what actions they are prohibited from engaging in. It also states that we reserve the right to terminate their account should any of these conditions be violated. Our Privacy Policy informs users that we, at Fractal Factory and Marist College, are dedicated to protecting their privacy and their personal information. Fractal Factory, specifically, collects user's emails for processing purposes. These processes include storing information only for login and identification purposes within our database. User emails will be stored and matched to a user ID in the database so the system will recognize login information whenever the user comes back to the site.

Because we authorize logins using third-party accounts like Facebook and Twitter, they also have different privacy policies that users are subject to should they choose to access Fractal Factory using their social media accounts. Therefore, users should be aware of third-party privacy policies, for there may be some points of protection that do not overlap.

When it comes to the gathering of data, there are some ethical issues that we, as developers, have to be mindful of. It is our job to think from the perspective of a Utilitarian in that we must build our applications with the betterment of the majority in mind. Fractal Factory is an app that seeks to work for everyone, work everywhere, respect user's privacy, and be considerate of its peers (Scott).

## **Pornography Problem**

With our use of hundreds of historic paintings, there runs a risk of their contents, that was once viewed as normal and artistic, being flagged as pornographic in our current day and age. It was very common for old art pieces to have images of nude people because they very much celebrated the form of the human body in those times. Because Fractal Factory is producing new art pieces based on those art pieces, it runs the risk of creating a piece that is potentially viewed as pornographic. If this happens, it is important that our users know this is purely an art form. Because the art is being described as an art form, it offers a layer of protection to us as developers.

When users register for Fractal Factory, they are acknowledging the fact that they are over the age of eighteen. Because users are consenting to being over a certain age, this means that we cannot be held liable if a pornographic type of image is viewed by a user under the age of eighteen. Ethically, this means also that we will never willingly put our users in an uncomfortable position. If we discover that one of our users is under the age of eighteen, we reserve the right to deactivate their account.

## **Ownership of Final Artwork**

Even though the other issues have been large topics of ethical discussion, this topic is by far the most important ethical dilemma: who owns the final artwork created by the AI? According to the US Copyright Office, an AI cannot be awarded copyright for something it has created (Hart). However, there is no requirement for human ownership; it is rather just implied that the creation of art is a human action. It has been argued that AIs also cannot be given ownership of their work because they are not human; when something goes wrong with the AI or it produces something it should not, the AI cannot be sued. Therefore, final ownership needs to go to a person or entity that owns the tool. Because AI is still not recognized as human persons, it has been determined that the current fix will be to award the owner the AI with the copyrights (Hart).

In the case of Fractal Factory, the user uploads the code, the Metropolitan Museum of Art owns the paintings in the database, the original artists own the art pieces, and we own the AI. Who owns the finished work of art? Fractal Factory. Even though we each own various parts of the finished product, FF owns the AI that creates the finished product despite the initial parts that go into it. There would be no finished product without the AI that we created that puts everything together. This being the verdict that has been set in place allows the ethical dilemma to be resolved if the idea of ownership ever comes into question.

## Conclusion

As with any other application, there are always going to be some dilemmas that will be faced. Fractal Factory is no exception. If we are going to put a product we deem as finished into the world, it is our responsibility to assess all the ethical dilemmas and make decisions that are best for the majority of our users. It is important to take the privacy of our users into consideration because currently, users care immensely about the privacy of their data. The warning of potential pornography is also an issue that needed addressing and has been addressed. Most importantly, the dilemma of ownership has been solved. Running an enjoyable and ethically-sound application is the goal of Fractal Factory, and we will conduct our business in a way that satisfies those goals.

## Works Cited

Hart, Robert David. "If an AI Creates a Work of Art, Who Owns the Rights to It?" *Quartz*, Quartz, 15 Aug. 2017, [qz.com/1054039/google-deepdream-art-if-an-ai-creates-a-work-of-art-who-owns-the-rights-to-it/](http://qz.com/1054039/google-deepdream-art-if-an-ai-creates-a-work-of-art-who-owns-the-rights-to-it/).

Scott, Adam D. "Building Web Apps That Respect a User's Privacy and Security." *O'Reilly | Safari*, O'Reilly Media, Inc., 24 Jan. 2017, [www.oreilly.com/library/view/building-web-apps/9781492042921/](http://www.oreilly.com/library/view/building-web-apps/9781492042921/).

# Appendix

## Database 3NF Proof

- Below are the four tables. Each contains five rows of example data:

### USERS

| user_id                              | user_account      | password   | first_name | last_name | email               | account_type | admin_flag | last_login          | date_added          | date_last_updated   |
|--------------------------------------|-------------------|--|------------|-----------|---------------------|--------------|------------|---------------------|---------------------|---------------------|
| 072ba068-3256-46e7-9138-5f454b92b81e | example@gmail.com | 23478207027842073230762374023                                    | Dave       | Lent      | example@gmail.com   | Google       | false      | 2014-08-17 19:15:53 | 2002-02-27 21:14:05 | 2009-12-22 10:34:32 |
| a67d93f5-d24c-458e-b090-ada03e10936e | admin             | \$2b\$10\$ilarA2jX6ZaiZwRne3l2ROG3fELZiC921QhVLzFL5WMjkkjOD.1nkq | Jessy      | Kent      | ex@gmail.com        | Default      | true       | 2017-11-11 07:57:24 | 2014-03-22 17:24:48 | 2017-11-11 07:57:24 |
| 1374801c-b520-4a55-aecc-b841930a05d4 | fc@yahoo.com      | 234476747607027842073230762374023                                | George     | Diez      | fc@yahoo.com        | Facebook     | false      | 2009-12-22 10:34:32 | 2006-05-20 17:03:43 | 2008-02-22 18:08:15 |
| 7ea674d9-eb9d-4ce4-9da9-0d0c50bfe55f | kjl8w             | 2347820702784207865742374023                                     | Jessica    | Brant     | example@hotmail.com | Marist       | true       | 2011-09-04 03:53:05 | 2003-03-12 02:55:17 | 2009-12-22 10:34:32 |
| 35eeb5a1-670a-46a6-ad9c-567d23fd339b | gg@gmail.com      | 2347820705685674073230762374023                                  | Felix      | Kant      | gg@gmail.com        | Google       | false      | 2017-11-11 07:57:24 | 2008-08-05 22:22:49 | 2017-11-11 07:57:24 |

## USER\_SOURCE\_FILES

| user_source_file_id                  | file_location         | blocks_file_location  | fractal_dimension | user_id                              | date_added          | date_last_updated   |
|--------------------------------------|-----------------------|-----------------------|-------------------|--------------------------------------|---------------------|---------------------|
| 5eac02f8-1aa5-4646-944d-6dd46827659c | /srv/source_files/uf0 | /srv/blocks_files/uf0 | 1.67              | 072ba068-3256-46e7-9138-5f454b92b81e | 2004-09-13 11:27:50 | 2014-08-17 19:15:53 |
| f02e5204-ecad-41c4-a344-37d8fadca447 | /srv/source_files/uf1 | /srv/blocks_files/uf1 | 1.91              | a67d93f5-d24c-458e-b090-ada03e10936e | 2017-11-11 07:57:24 | 2013-04-10 05:13:14 |
| dff03454-d205-4d57-8074-55b9019e7e02 | /srv/source_files/uf2 | /srv/blocks_files/uf2 | 1.34              | 1374801c-b520-4a55-aecc-b841930a05d4 | 2008-02-22 18:08:15 | 2017-11-11 07:57:24 |
| 86a2d9cd-9f5e-4ffe-b0cf-c64a0c9ab7ae | /srv/source_files/uf3 | /srv/blocks_files/uf4 | 1.56              | 7ea674d9-eb9d-4ce4-9da9-0d0c50bfe55f | 2009-12-22 10:34:32 | 2011-09-04 03:53:05 |
| 753cb0d6-240a-49e6-a07b-13430a902743 | /srv/source_files/uf4 |                       | 1.89              | 35eeb5a1-670a-46a6-ad9c-567d23fd339b | 2008-02-22 18:08:15 | 2005-01-29 04:45:45 |

## USER\_PAINTINGS

| user_painting_id                     | file_location           | painting_id                          | user_source_file_id                  | user_id                              | date_added          | watermark_flag | date_last_updated    |
|--------------------------------------|-------------------------|--------------------------------------|--------------------------------------|--------------------------------------|---------------------|----------------|----------------------|
| cd7870b2-2d4d-4ff9-9801-05c6fa7c44a8 | /srv/user_paintings/up0 | afbbadc3-d088-46b6-bbef-66494a659e7f | 5eac02f8-1aa5-4646-944d-6dd46827659c | 072ba068-3256-46e7-9138-5f454b92b81e | 2004-11-08 07:46:42 | true           | 2006-11-11 07:57:24  |
| 55ce7670-ccd5-4fc7-9b88-6cdc0743d9ee | /srv/user_paintings/up1 | 8eea83d0-5ff8-42bf-8b19-aa584b904a3d | f02e5204-ecad-41c4-a344-37d8fadca447 | a67d93f5-d24c-458e-b090-ada03e10936e | 2014-08-17 19:15:53 | true           | 2017-11-11 07:57:24  |
| b93fc5cc-32aa-459e-9008-2e9b513df827 | /srv/user_paintings/up2 | 2aec4b7-f48a-4692-b972-890f20038031  | dff03454-d205-4d57-8074-55b9019e7e02 | 1374801c-b520-4a55-aecc-b841930a05d4 | 2006-09-04 03:53:05 | false          | 2007-09-04 03:53:05  |
| 6019efa8-1657-42af-bdba-f7075a0d99b6 | /srv/user_paintings/up3 | f57633b0-969c-44b7-b859-6dad96ceeae0 | 86a2d9cd-9f5e-4ffe-b0cf-c64a0c9ab7ae | 7ea674d9-eb9d-4ce4-9da9-0d0c50bfe55f | 2007-04-10 05:13:14 | false          | 2008-04-10 05:13:14  |
| f3bdb8d5-7b8a-47ed-9ef3-ee375a928abe | /srv/user_paintings/up4 | 2ad8d839-9c7f-4682-8740-64b6413b3254 | 753cb0d6-240a-49e6-a07b-13430a902743 | 35eeb5a1-670a-46a6-ad9c-567d23fd339b | 2009-01-29 04:45:45 | true           | 20014-08-05 22:22:49 |

## PAINTINGS

| painting_id                          | file_location     | fractal_dimension | name                 | painter          | year_created | date_added          | date_last_updated   |
|--------------------------------------|-------------------|-------------------|----------------------|------------------|--------------|---------------------|---------------------|
| afbbadc3-d088-46b6-bbef-66494a659e7f | /srv/paintings/p0 | 1.26              | Mona Lisa            | Leonard Da Vinci | 1557         | 2000-11-21 05:37:50 | 2005-01-29 04:45:45 |
| 8eea83d0-5ff8-42bf-8b19-aa584b904a3d | /srv/paintings/p1 | 1.22              | The Starry Night     | Vincent Van Gogh | 1889         | 2001-12-13 20:06:52 | 2016-11-08 07:46:42 |
| 2aec4b7-f48a-4692-b972-890f20038031  | /srv/paintings/p2 | 1.67              | The Scream           | Edvard Munch     | 1893         | 2002-10-22 12:40:50 | 2008-12-13 20:06:52 |
| f57633b0-969c-44b7-b859-6dad96ceeae0 | /srv/paintings/p3 | 1.43              | The Night Watch      | Rembrandt        | 1642         | 2003-06-20 13:22:52 | 2011-09-04 03:53:05 |
| 2ad8d839-9c7f-4682-8740-64b6413b3254 | /srv/paintings/p4 | 1.87              | The Creation of Adam | Michelangelo     | 1512         | 2002-11-29 19:01:43 | 2017-11-11 07:57:24 |

- Each column contains only atomic values.
- There are no repeating groups.
- Each non-key column is dependent on the table's primary key.
- Each column is non-transitively dependent on the table's primary key.
- Left joining the user\_paintings and user\_source\_files tables to the users table, then left joining the paintings table to the user\_paintings table provides a complete table with referential integrity. Each user has one user painting and one user source file which makes use of each of the paintings, so no nulls are present. However, it is possible that a user may not have uploaded a source file or requested a painting which will result in nulls for those columns. Furthermore, not all paintings may be used to make user paintings, so nulls may result from that circumstance. Such nulls are expected and follow our business rules. Due to the length of the resulting table, it is shown below in five segments:





| paintings<br>Painting Id (Painti... | paintings<br>File Location (Pain... | #<br>paintings<br>Fractal Dimension ... | paintings<br>Name    | paintings<br>Painter | paintings<br>Year Created | paintings<br>Date Added (Paint... |
|-------------------------------------|-------------------------------------|---|----------------------|----------------------|---------------------------|-----------------------------------|
| AFBBADC3-D088-46B...                | /srv/paintings/p0                   | 1.260000                                | Mona Lisa            | Leonard Da Vinci     | 1557                      | 11/21/2000 5:37:50 AM             |
| 8EEA83D0-5FF8-42BF...               | /srv/paintings/p1                   | 1.220000                                | The Starry Night     | Vincent Van Gogh     | 1889                      | 12/13/2001 8:06:52 PM             |
| 2AEC487-F48A-469...                 | /srv/paintings/p2                   | 1.670000                                | The Scream           | Edvard Munch         | 1893                      | 10/22/2002 12:40:50 ...           |
| F57633B0-969C-44B7...               | /srv/paintings/p3                   | 1.430000                                | The Night Watch      | Rembrandt            | 1642                      | 6/20/2003 1:22:52 PM              |
| 2AD8D839-9C7F-468...                | /srv/paintings/p4                   | 1.870000                                | The Creation of Adam | Michelangelo         | 1512                      | 11/29/2002 7:01:43 PM             |

| paintings<br>Date Last Updated... | user_paintings<br>User Painting Id | user_paintings<br>Painting Id | user_paintings<br>User Source File Id | user_paintings<br>User Id (User Pain... | user_paintings<br>File Location | T/F<br>user_paintings<br>Watermark Flag |
|-----------------------------------|------------------------------------|-------------------------------|---------------------------------------|---|---------------------------------|---|
| 1/29/2005 4:45:45 AM              | CD7870B2-2D4D-4FF...               | AFBBADC3-D088-46B...          | 5EAC02F8-1AA5-4646...                 | 072BA068-3256-46E7...                   | /srv/user_paintings/u...        | True                                    |
| 11/8/2016 7:46:42 AM              | 55CE7670-CCD5-4FC7...              | 8EEA83D0-5FF8-42BF...         | F02E5204-ECAD-41C4...                 | A67D93F5-D24C-458...                    | /srv/user_paintings/u...        | True                                    |
| 12/13/2008 8:06:52 PM             | B93FC5CC-32AA-459E...              | 2AEC487-F48A-469...           | DFF03454-D205-4D5...                  | 1374801C-B520-4A55...                   | /srv/user_paintings/u...        | False                                   |
| 9/4/2011 3:53:05 AM               | 6019EFA8-1657-42AF...              | F57633B0-969C-44B7...         | 86A2D9CD-9F5E-4FFE...                 | 7EA674D9-EB9D-4CE...                    | /srv/user_paintings/u...        | False                                   |
| 11/11/2017 7:57:24 AM             | F3BD8BD5-7B8A-47E...               | 2AD8D839-9C7F-468...          | 753CB0D6-240A-49E...                  | 35EEB5A1-670A-46A...                    | /srv/user_paintings/u...        | True                                    |

| user_paintings<br>Date Added (User ... | user_paintings<br>Date Last Updated... | user_source_files<br>User Source File I... | user_source_files<br>User Id (User Sour... | user_source_files<br>File Location (Use... | user_source_files<br>Blocks File Location | #<br>user_source_files<br>Fractal Dimension |
|--|--|--|--|--|---|---|
| 11/8/2004 7:46:42 AM                   | 11/11/2006 7:57:24 AM                  | 5EAC02F8-1AA5-4646...                      | 072BA068-3256-46E7...                      | /srv/source_files/uf0                      | /srv/blocks_files/uf0                     | 1.670000                                    |
| 8/17/2014 7:15:53 PM                   | 11/11/2017 7:57:24 AM                  | F02E5204-ECAD-41C4...                      | A67D93F5-D24C-458...                       | /srv/source_files/uf1                      | /srv/blocks_files/uf1                     | 1.910000                                    |
| 9/4/2006 3:53:05 AM                    | 9/4/2007 3:53:05 AM                    | DFF03454-D205-4D5...                       | 1374801C-B520-4A55...                      | /srv/source_files/uf2                      | /srv/blocks_files/uf2                     | 1.340000                                    |
| 4/10/2007 5:13:14 AM                   | 4/10/2008 5:13:01 AM                   | 86A2D9CD-9F5E-4FFE...                      | 7EA674D9-EB9D-4CE...                       | /srv/source_files/uf3                      | /srv/blocks_files/uf3                     | 1.560000                                    |
| 1/29/2009 4:45:45 AM                   | 10/21/2019 12:00:00 ...                | 753CB0D6-240A-49E...                       | 35EEB5A1-670A-46A...                       | /srv/source_files/uf4                      | /srv/blocks_files/uf4                     | 1.890000                                    |

| user_source_files<br>Date Added (User ... | user_source_files<br>Date Last Updated... | users<br>User Id      | users<br>User Account | users<br>Password         | users<br>First Name | users<br>Last Name |
|---|---|-----------------------|-----------------------|---------------------------|---------------------|--------------------|
| 9/13/2004 11:27:50 AM                     | 8/17/2014 7:15:53 PM                      | 072BA068-3256-46E7... | example123            | 23478207027842073...      | Dave                | Lent               |
| 11/11/2017 7:57:24 AM                     | 4/10/2013 5:13:14 AM                      | A67D93F5-D24C-458...  | admin                 | \$2b\$10\$ilarA2jX6Zai... | Jessy               | Kent               |
| 2/22/2008 6:08:15 PM                      | 11/11/2017 7:57:24 AM                     | 1374801C-B520-4A55... | fc@yahoo.com          | 23447674760702784...      | George              | Diez               |
| 12/22/2009 10:34:32 ...                   | 9/4/2011 3:53:05 AM                       | 7EA674D9-EB9D-4CE...  | kjl8w                 | 23478207027842078...      | Jessica             | Brant              |
| 2/22/2008 6:08:15 PM                      | 1/29/2005 4:45:45 AM                      | 35EEB5A1-670A-46A...  | gg@gmail.com          | 23478207056856740...      | Felix               | Kant               |

| ABC<br>USERS        | ABC<br>USERS | T F<br>USERS | 🕒<br>USERS              | 🕒<br>USERS           | 🕒<br>USERS              | T F<br>USERS |
|---------------------|--------------|--------------|-------------------------|----------------------|-------------------------|--------------|
| Email               | Account Type | Admin Flag   | Last Login              | Date Added           | Date Last Updated       | Active Flag  |
| example@gmail.com   | Google       | False        | 8/17/2014 7:15:53 PM    | 2/27/2002 9:14:05 PM | 12/22/2009 10:34:32 ... | True         |
| ex@gmail.com        | Default      | True         | 11/11/2017 7:57:24 AM   | 3/22/2014 5:24:48 PM | 11/11/2017 7:57:24 AM   | True         |
| fc@yahoo.com        | Facebook     | False        | 12/22/2009 10:34:32 ... | 5/20/2006 5:03:43 PM | 2/22/2008 6:08:15 PM    | True         |
| example@hotmail.com | Marist       | True         | 9/4/2011 3:53:05 AM     | 3/12/2003 2:55:17 AM | 12/22/2009 10:34:32 ... | True         |
| gg@gmail.com        | Google       | False        | 11/11/2017 7:57:24 AM   | 8/5/2008 10:22:49 PM | 11/11/2017 7:57:24 AM   | True         |

## SQL Statements

### Create Statements:

-- Need to add uuid-oss extension to generate uuid's

--CREATE EXTENSION "uuid-oss";

CREATE TABLE users(

    user\_id UUID DEFAULT uuid\_generate\_v4() PRIMARY KEY,

    user\_account TEXT NOT NULL UNIQUE, -- account identifier

    password TEXT NOT NULL, -- Password hash or token

    first\_name TEXT NOT NULL,

    last\_name TEXT,

    email TEXT NOT NULL,

    account\_type TEXT NOT NULL,

    admin\_flag BOOLEAN DEFAULT 'f' NOT NULL,

    last\_login TIMESTAMPTZ DEFAULT NOW() NOT NULL,

    date\_added TIMESTAMPTZ DEFAULT NOW() NOT NULL,

    date\_last\_updated TIMESTAMPTZ DEFAULT NOW(),

    active\_flag BOOLEAN DEFAULT 't' NOT NULL

);

CREATE TABLE paintings(

    painting\_id UUID DEFAULT uuid\_generate\_v4() PRIMARY KEY,

    file\_location TEXT,

    fractal\_dimension NUMERIC, -- Null means it needs to be calculated

    -- Metadata for painting may be null ... be aware of this on front-end

    name TEXT,

    painter TEXT,

    year\_created TEXT,

```

        -- End of metadata

        date_added TIMESTAMPTZ DEFAULT NOW() NOT NULL,
        date_last_updated TIMESTAMPTZ DEFAULT NOW() NOT NULL
    );

CREATE TABLE user_source_files(
    user_source_file_id uuid DEFAULT uuid_generate_v4() PRIMARY KEY,
    user_id UUID NOT NULL REFERENCES users(user_id) ON DELETE RESTRICT, --
Referential integrity constraint
    file_location TEXT,
    blocks_file_location TEXT,
    fractal_dimension NUMERIC, -- Null means it needs to be calculated
    date_added TIMESTAMPTZ DEFAULT NOW() NOT NULL,
    date_last_updated TIMESTAMPTZ DEFAULT NOW() NOT NULL
);

CREATE TABLE user_paintings(
    user_painting_id UUID DEFAULT uuid_generate_v4() PRIMARY KEY,
    painting_id UUID NOT NULL REFERENCES paintings(painting_id) ON DELETE
RESTRICT, -- Referential integrity constraint
    user_source_file_id UUID NOT NULL REFERENCES
user_source_files(user_source_file_id) ON DELETE RESTRICT, -- Referential integrity
constraint
    user_id UUID NOT NULL REFERENCES users(user_id) ON DELETE RESTRICT, --
Referential integrity constraint
    file_location TEXT,
    watermark_flag BOOLEAN DEFAULT 't' NOT NULL,
    date_added TIMESTAMPTZ DEFAULT NOW() NOT NULL,
    date_last_updated TIMESTAMPTZ DEFAULT NOW() NOT NULL);

```

## Example Data Insert Statements:

```
INSERT INTO USERS(user_id, user_account, password, first_name, last_name, email,
account_type, admin_flag, last_login, date_added, date_last_updated) VALUES
('072ba068-3256-46e7-9138-5f454b92b81e', 'example123',
'23478207027842073230762374023', 'Dave', 'Lent', 'example@gmail.com', 'Google', false,
'2014-08-17 19:15:53', '2002-02-27 21:14:05', '2009-12-22 10:34:32'),
('a67d93f5-d24c-458e-b090-ada03e10936e', 'admin',
'$2b$10$ilarA2jX6ZaiZwRne3l2ROG3fELZiC921QhVLzFL5WMjkjOD.1nkq', 'Jessy', 'Kent',
'ex@gmail.com', 'Default', true, '2017-11-11 07:57:24', '2014-03-22 17:24:48', '2017-11-11
07:57:24'),
('1374801c-b520-4a55-aecc-b841930a05d4', 'fc@yahoo.com',
'234476747607027842073230762374023', 'George', 'Diez', 'fc@yahoo.com', 'Facebook', false,
'2009-12-22 10:34:32', '2006-05-20 17:03:43', '2008-02-22 18:08:15'),
('7ea674d9-eb9d-4ce4-9da9-0d0c50bfe55f', 'kjl8w', '2347820702784207865742374023',
'Jessica', 'Brant', 'example@hotmail.com', 'Marist', true, '2011-09-04 03:53:05', '2003-03-12
02:55:17', '2009-12-22 10:34:32'),
('35eeb5a1-670a-46a6-ad9c-567d23fd339b', 'gg@gmail.com',
'2347820705685674073230762374023', 'Felix', 'Kant', 'gg@gmail.com', 'Google', false,
'2017-11-11 07:57:24', '2008-08-05 22:22:49', '2017-11-11 07:57:24')
;

INSERT INTO PAINTINGS(painting_id, file_location, fractal_dimension, name, painter,
year_created, date_added, date_last_updated) VALUES
('afbbadc3-d088-46b6-bbef-66494a659e7f', '/srv/paintings/p0', 1.26, 'Mona Lisa', 'Leonard Da
Vinci', '1557', '2000-11-21 05:37:50', '2005-01-29 04:45:45'),
('8eea83d0-5ff8-42bf-8b19-aa584b904a3d', '/srv/paintings/p1', 1.22, 'The Starry Night', 'Vincent
Van Gogh', '1889', '2001-12-13 20:06:52', '2016-11-08 07:46:42'),
```

```

('2aec4b7-f48a-4692-b972-890f20038031', '/srv/paintings/p2', 1.67, 'The Scream', 'Edvard
Munch', '1893', '2002-10-22 12:40:50', '2008-12-13 20:06:52'),
('f57633b0-969c-44b7-b859-6dad96ceeae0', '/srv/paintings/p3', 1.43, 'The Night Watch',
'Rembrandt', '1642', '2003-06-20 13:22:52', '2011-09-04 03:53:05'),
('2ad8d839-9c7f-4682-8740-64b6413b3254', '/srv/paintings/p4', 1.87, 'The Creation of Adam',
'Michelangelo', '1512', '2002-11-29 19:01:43', '2017-11-11 07:57:24')
;

INSERT INTO USER_SOURCE_FILES(user_source_file_id, file_location, blocks_file_location,
fractal_dimension, user_id, date_added, date_last_updated) VALUES
('5eac02f8-1aa5-4646-944d-6dd46827659c', '/srv/source_files/uf0', '/srv/blocks_files/uf0',
1.67, '072ba068-3256-46e7-9138-5f454b92b81e', '2004-09-13 11:27:50', '2014-08-17
19:15:53'),
('f02e5204-ecad-41c4-a344-37d8fadca447', '/srv/source_files/uf1', '/srv/blocks_files/uf1', 1.91,
'a67d93f5-d24c-458e-b090-ada03e10936e', '2017-11-11 07:57:24', '2013-04-10 05:13:14'),
('dff03454-d205-4d57-8074-55b9019e7e02', '/srv/source_files/uf2', '/srv/blocks_files/uf2', 1.34,
'1374801c-b520-4a55-aecc-b841930a05d4', '2008-02-22 18:08:15', '2017-11-11 07:57:24'),
('86a2d9cd-9f5e-4ffe-b0cf-c64a0c9ab7ae', '/srv/source_files/uf3', '/srv/blocks_files/uf3', 1.56,
'7ea674d9-eb9d-4ce4-9da9-0d0c50bfe55f', '2009-12-22 10:34:32', '2011-09-04 03:53:05'),
('753cb0d6-240a-49e6-a07b-13430a902743', '/srv/source_files/uf4', '/srv/blocks_files/uf4',
1.89, '35eeb5a1-670a-46a6-ad9c-567d23fd339b', '2008-02-22 18:08:15', '2005-01-29 04:45:45')
;

INSERT INTO USER_PAINTINGS(user_painting_id, file_location, painting_id,
user_source_file_id, user_id, date_added, watermark_flag, date_last_updated) VALUES
('cd7870b2-2d4d-4ff9-9801-05c6fa7c44a8', '/srv/user_paintings/up0',
'affbbadc3-d088-46b6-bbef-66494a659e7f', '5eac02f8-1aa5-4646-944d-6dd46827659c',
'072ba068-3256-46e7-9138-5f454b92b81e', '2004-11-08 07:46:42', true, '2006-11-11 07:57:24'),

```

('55ce7670-ccd5-4fc7-9b88-6cdc0743d9ee', '/srv/user\_paintings/up1',  
'8eea83d0-5ff8-42bf-8b19-aa584b904a3d', 'f02e5204-ecad-41c4-a344-37d8fadca447',  
'a67d93f5-d24c-458e-b090-ada03e10936e', '2014-08-17 19:15:53', true, '2017-11-11 07:57:24'),  
( 'b93fc5cc-32aa-459e-9008-2e9b513df827', '/srv/user\_paintings/up2',  
'2aec4b7-f48a-4692-b972-890f20038031', 'dff03454-d205-4d57-8074-55b9019e7e02',  
'1374801c-b520-4a55-aecc-b841930a05d4', '2006-09-04 03:53:05', false, '2007-09-04  
03:53:05'),  
( '6019efa8-1657-42af-bdba-f7075a0d99b6', '/srv/user\_paintings/up3',  
'f57633b0-969c-44b7-b859-6dad96ceeae0', '86a2d9cd-9f5e-4ffe-b0cf-c64a0c9ab7ae',  
'7ea674d9-eb9d-4ce4-9da9-0d0c50bfe55f', '2007-04-10 05:13:14', false, '2008-04-10 05:13:14'),  
( 'f3bdb8d5-7b8a-47ed-9ef3-ee375a928abe', '/srv/user\_paintings/up4',  
'2ad8d839-9c7f-4682-8740-64b6413b3254', '753cb0d6-240a-49e6-a07b-13430a902743',  
'35eeb5a1-670a-46a6-ad9c-567d23fd339b', '2009-01-29 04:45:45', true, '20014-08-05  
22:22:49')  
;