**13/2/24: Roadmap**

[CSCI 2340 Art Generator](https://drive.google.com/drive/folders/144ychxfxFBiS3QB_ps2br-hSVueHUBEl?usp=drive_link)

* [CSCI 2340 Requirements](https://docs.google.com/document/d/1s-fqDJgYTXmqsbzJZpRWFa9KRA58WktoMIYTIrCHWus/edit?usp=drive_link)
* [CSCI 2340 Specifications](https://docs.google.com/document/d/1VgtjVZgx9Wl6B6iJw1Jla86mmwsoooUld5CC7cOwPAI/edit?usp=drive_link)
* [CSCI 2340 Architecture](https://docs.google.com/document/d/1V-cAnn9ahF3OpMUKfcruLaYOJADiX1i0FHMammK7tTM/edit?usp=drive_link)
* [CSCI 2340 Coding Standards](https://docs.google.com/document/d/1tBrvi83gBnP-oiooZCUB0BAnRE90bbJRQeYzcsD4MRg/edit?usp=drive_link)

**Project 1: Flames Webcam Filter**

EPIC and EOSDIS: earth/user cam, filtered through climate change data + fire generated

Reference: <https://www.instagram.com/p/C3HBTgPthr3/>

Data: EOSDIS data related to climate change

Input: user webcam, mouse location, selected data

Output: ASCII forms of webdata given climate data chosen

**Core Data Types → Algorithms:**

1. (Image1, Image2)
   1. Morph Cutting → .mp4
   2. Linear Interpolation → .mp4
2. Continuous/real-time webcam Image
   1. Filter/fire → Image
      1. "Semi-gross color palette. Animating the building blocks with little squiggles and then adding color / feedback effects." [Ref](https://www.instagram.com/p/C3HBTgPthr3/)
3. float[] (e.g. climate values)

**User Input Types:**

1. Geographic Location
2. Configurations
   1. Data Source: Choose among our pre-filled Database
   2. Speed / refresh rate
3. Static
   1. Query/Topic → "English String"
   2. Click →
      1. 1/0  toggle boolean buttons
      2. (int x, int y, int time)
   3. Geographic Location →
      1. Coordinates (float latitude, float longitude)
      2. String "description"
4. Real-time
   1. Mouse Location (int x, int y)
   2. Video .mp4
   3. Audio .mp3

**Roadmap:**

1. Algorithm: Motion/Interpolation between 2 little sets of data / images
   1. Input: data selection and settings
   2. Morph Cutting

**User Requirements:**

1. Visuals (depends on backend data and algorithm)
   1. The User can access art-gen through Desktop
   2. The User can access art-gen through Mobile
   3. The result should be entertaining and artistic
   4. The result should be understandable/meaningful (maybe with description) → have some intuitive connection to source
   5. The user should feel that they are expressing their intents and ideas through their controls and the generated visuals
2. User Interaction (input, control)
   1. Shareability
      1. 1.b
      2. I’d like to share my outputs
         1. download image / videos
         2. send data/seeds to be processed on another browser
         3. functionalities makes sharing outputs with friends easier
   2. Customization
      1. The user should be able to choose the NASA data source
      2. OPT: The user can input personal data to generate art from
      3. OPT:  The user would like to see other people’s outputs
      4. OPT:  Trending section maybe?
   3. Usability
      1. User should be able to learn how to use the application easily
      2. Should be intuitive
3. Security
   1. Secure data upload
   2. Secure connection
   3. Caching for faster performance
   4. Memory management
4. Textual Art Element

**Implementation Specifications**

1. Visuals
   1. Input:
2. User Interaction (input, control)
   1. The user interacts with the data parameters to change the intended output. Additionally, user motion through the camera will affect outputs
      1. Inputs:
         1. Data parameter selection
      2. Outputs:
         1. Display output from ASCII Flame Generator
   2. Customisation
      1. The user should be able to customize the data
         1. Input: suggestion for new databases to be added
         2. Output: None, potential addition in the future
   3. Optional Below:
   4. *Shareability*
      1. *Inputs: image (NASA API)/video (webcam), destination*
      2. *Outputs: success/failure*
3. Security

**Codebase Architectures**

1. Microservice Architectures
   1. Core architecture in form of a web-application
      1. User interaction
      2. Output display
      3. Data I/O (Database oriented)
      4. Service reused by all generators
   2. Connect to backend implementations of art generator
      1. Loosely coupled
      2. Implemented independently
   3. Uniform converter to convert generator output to displayable output
      1. For added generators and future addition
      2. Connect new features to the core  web-application
   4. User interface panel allow users to customer parameters of  art generator
      1. Clear instruction page for users/ documentations for developers

**Coding Standards**

Coding Conventions

**Git Branches:**

1. Use local branch (e.g backend, morgann, feature-X),
2. Pull from more core branch (e.g. main)
3. Resolve merge conflicts locally (use your discretion and just ask whoever wrote it if you have a conflicting design decision)
4. Push your branch to remote
5. Pull Request to core branch (main) to update it

Naming Conventions

**Next.tsx (Morgann):**

1. root = github-project-kebab-case
2. root/components/ComponentInPascalCase.tsx
3. root/app/<more-structure>/page-in-kebab-case.ts
4. root/styles/<more-structure>/styles-matching-page?
5. root/types/type-pascal.ts
   1. [quense/yup: Dead simple Object schema validation](https://github.com/jquense/yup)
   2. export const typeSchemaCamel = object({
   3. export type TypePascal = InferType<typeof typeSchemaCamel>;
6. root/constants.ts
   1. CONSTANT\_UPPER\_CASE = …

Ordering Conventions

Global Variables

…

Functions

…

Main

File System Organisation

rootDir ->

src->

FrontEnd ->

UI ->

Data I/O ->

GenerativeFrontEnd ->

BackEnd ->

Databases ->

GenerativeBackEnd ->

…

docs->

Requirements ->

Notes ->

Statuses ->

bins->