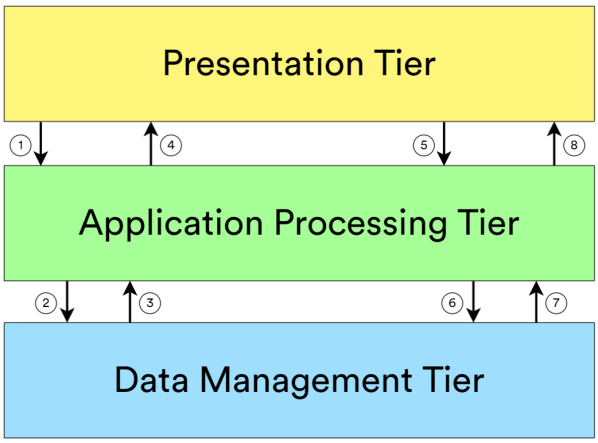
Solien Gallery Programmer's Guide

Aaron, Gabe G., Gabe L., Nick

Section 1: Top-Level Description

Before explaining the components of our application in detail, we describe our project's three tiers (presentation, application processing, and data management) at a high level. Namely, we discuss the technologies used in each of the three tiers, as well as the general communication flow between the tiers.



Presentation Tier I

This tier consists of several web pages that are written in HTML, Javascript, and CSS. Our team used Bootstrap and Google Fonts in the CSS code to improve the appearance of the front-end, and we used JQuery in our JavaScript code to facilitate AJAX requests (among other operations). The presentation tier largely serves as the interface between the user and the application processing tier; very little processing is done in the presentation tier itself. The only notable exception is the gallery creator page, which handles users' creation of their galleries (i.e., moving images between the image selector and the gallery, and rearranging images in the gallery).

Application Processing Tier I

This tier consists of several Python files, which are responsible for (1) retrieving the Solien NFTs associated with a wallet number, and (2) building the gallery image created by the user. Our team used many external libraries/frameworks in this tier. We list them below:

- Flask and Jinja are used to handle communication with the presentation tier.
- RQ (Redis Queue) is used to manage background task execution.
- The Blockchain API is used to facilitate queries to the Solana blockchain.
- PIL (Pillow) is used for image processing.
- The Cloudinary API is used to store images with Cloudinary.

Note that the application processing tier must replicate the gallery created by the user, because the gallery is represented entirely in HTML and CSS in the presentation tier and must be converted to a PNG image for download.

Data Management Tier I

This tier consists of external databases, such as the Solana blockchain[1] and Cloudinary, and internal databases, which are implemented as JSON files. We use two distinct kinds of databases in our application: (1) databases related to the NFT data, and (2) a database for image storage.

Communication Flow I

There are many links of communication between the application processing tier and the other tiers. We postpone a full description of these interactions until the next section. Instead, we provide here a high-level overview of the two processes that traverse all three tiers: (1) the retrieval of NFT metadata (steps 1-4), and (2) gallery image creation (steps 5-8). We describe each step in full below:

Metadata Retrieval:

1. The user has entered their wallet number in the homepage, and the presentation tier requests that the application processing tier retrieve the Solien NFT images associated with the wallet.
2. The application processing tier makes queries to the data management tier to retrieve the metadata of the Solien NFTs associated with the wallet.
3. The data management tier returns the appropriate metadata.

[1] Details on the blockchain are provided in the appendix.

4. The application processing tier passes the images of the Soliens (contained in the metadata) to the presentation tier.

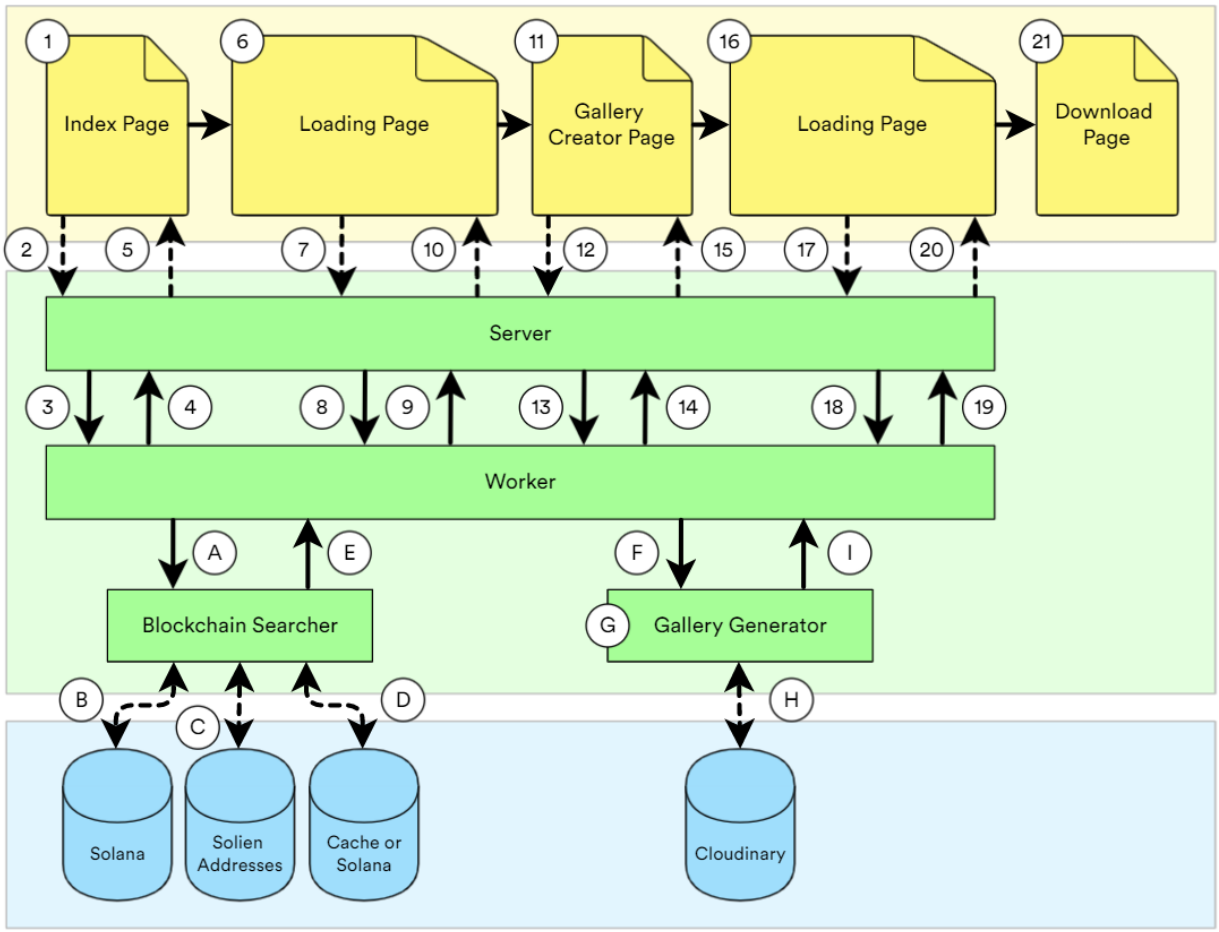
Image Building and Storage:

5. The user has created their gallery, and the presentation tier requests that the application processing tier provide a link to an image of the gallery.
6. The application tier creates the gallery as a PNG and requests that the data management tier store the gallery image.
7. The data management tier saves the gallery image and returns a link to the image.
8. The application tier passes the gallery image link to the presentation tier.

Once again, the flow above is not exactly how the three tiers communicate with one another. The exact details of communication between tiers will be provided in the next section, because these details rely on an understanding of the underlying implementation (e.g., how the Redis queue manages background tasks). The current section only serves as a high-level overview of the tasks performed by each tier.

Section 2: Lower-Level Description

We now provide a deeper explanation of the components of our application. The diagram below shows the constituent components of the three tiers from the previous section, color-coded for clarity.



Presentation Tier II

This section will provide more information on the specific pages in the Presentation Tier.

Index Page:

This is the first page of the website consisting of our site logo and name, a short description, an input field for the user's wallet address with a search button, and an About Us section at the bottom. The HTML on this page is straightforward; the only forms of user input are a text input field and a "Search" button. The "Search" button triggers the JavaScript method `enqueueSearch()`, which makes an AJAX request to the backend, with

the wallet address as an argument. This request asks the application processing tier to begin retrieving the Solien NFTs in the wallet as a background task. If this request is successful, the user is redirected to the Loading Page.

Loading Page:

We display this page to the user when we are running a background task in the application processing tier in between pages. It displays a pure CSS loading animation in the middle of the page, and loading text that changes via some JavaScript methods. This animation repeats while the loading page polls the application processing tier in the background, until the task is finished or failed. The loading page then redirects the user to the page corresponding to the response.

Gallery Creator Page:

This page consists of a gallery div for the NFTs in the gallery, a background image/color picker, a “Create” button, and a wallet div for the NFTs in the user’s wallet that are not in the gallery.

First I will explain the processes involved in adding/removing NFTs to the gallery. NFTs in the wallet div are HTMLImageObjects of custom CSS class `nft-not-in-gal` and NFTs in the gallery div are HTMLImageObjects of custom CSS class `nft-in-gal`. These custom classes have different key properties like sizing, the cursor involved when clicking on them, and the ability to be dragged (only `nft-in-gal` can be dragged). When a user clicks on an NFT in the wallet div, if there are less than four NFTs in the gallery div, a new `nft-in-gal` HTMLImageObject is populated into the gallery div with the same URL and ID. Then the same `nft-not-in-gal` HTMLImageObject clicked on is removed from the wallet div. The same process vice versa happens when you click on an NFT in the gallery div. NFTs in the gallery are also “draggable”, meaning that there is CSS, HTML, and JavaScript code which allows them to adjust their order in the gallery div by dragging them around.

The other feature of this page is the background image/color picker. When a user clicks a background image or the color picker, an `onclick` event is triggered which sets the background of the gallery div to either the image url displayed in the preview or the color chosen on the color picker.

Last, the create button triggers the `enqueueGallery()` method which iterates through the gallery div to store the URLs of the NFTs in the gallery into a list and sets the gallery

background info as cookies to then pass an AJAX request with the gallery URL to the backend. Upon a successful request, the page is redirected to a loading page in a new tab with an appropriate job id.

Download Page:

This page contains a preview of the gallery image as well as an href to direct the user to the Cloudinary image URL.

Error Page:

Although not listed in our Section 2 diagram, our application directs the user to the Error Page when an exception occurs in our code. The error is passed in through the response and displayed on the page. The page also contains an href linking the user to the initial page of our application.

Application Processing Tier II

This tier consists of four modules: Server, Worker, Blockchain Searcher, Gallery Generator.

Server:

The Server module consists of a file defining various Flask routes. We list the routes in the table below:

Route	Description
`/ or /index`	Index page, described above.
`/status/<page>/<job_id>`	Page that provides the status (e.g., “finished” or “failed”) and the result (e.g., gallery image URL) of a background task with the job ID `job_id` in JSON format. The `page` argument specifies the page that is being loaded. (This argument is necessary in constructing the JSON response.)
`/loading/<page>/<job_id>`	Loading page, described above. Both arguments serve the same purpose as in `/status/`.
`/enqueue_search/<wallet>`	Page that (1) enqueues a background task to search for the NFTs associated with the wallet address `wallet` and (2) returns the job ID associated with the background task.

`/gallery/<id_strs>`	Gallery creator page, described above. The `id_strs` argument specifies the links to the NFT images owned by the wallet provided.
`/enqueue_gallery/<id_strs>`	Page that (1) enqueues a background task to create a gallery image with the images specified by `id_strs`, and (2) returns the job ID associated with the background task.
`/download/<id_strs>`	Download page, described above. The `id_strs` argument specifies two unique identifiers for the gallery image link.
`/error/<status_code>`	Page that displays an error to the user, depending on the optional `status_code` argument.

Several important routes have been described in great detail in the previous subsection. We discuss the worker module in the subsequent subsection, which will likely elucidate the purpose of some of the remaining routes (e.g., `/status/`and `/enqueue_search/`). The only remaining route is `/error/`. We enumerate the possible error status codes below.

Status Code	Error Message (Explanation)
400	Wallet contains no Solien NFTs. (This error occurs when the user enters a wallet address with no Solien NFTs.)
404	The specified wallet number could not be found. (This error occurs when the user has entered an invalid wallet address.)
500	Cannot process call to blockchain. (This error occurs when we have lost access to The Blockchain API. It should not occur.)
505	A server error has occurred. (This is a general error message, for miscellaneous exceptions.)

Worker:

The Worker module is an integral component of the application processing tier, responsible for running background tasks asynchronously. Indeed, the Worker does most of the processing in our application; it is in charge of retrieving the metadata from NFTs, as well as building and storing the gallery image. Before the implementation of the Worker, long requests[2] (which took more than 30 seconds to complete) would cause a timeout error on

[2] As a side note, many requests would take longer than thirty seconds before the implementation of the cache, because each blockchain query required a linear traversal of the Solana blockchain.

Heroku. Running tasks in the background with the Worker allows processes to run for over 30 seconds if necessary.

The Worker module consists of `worker.py` and a task queue (hosted on a Redis server). When prompted by the presentation tier, the Server module will add tasks to the queue via the `/enqueue_search/` or `/enqueue_gallery/` pages, or check the status and outcome of tasks via the `/status/` page. At a high level, the Server hands off computationally intensive tasks to the Worker, and then polls the worker to check if tasks are completed. Tasks on the queue are given a unique job ID, which allows for subsequent retrieval from the queue; the job IDs returned by the `/enqueue_search/` and `/enqueue_gallery/` pages are propagated to the loading page (and `/status/` page) for this reason.

When exceptions are encountered by the Worker module, the Server module retrieves the traceback string and converts the relevant exception to a status code. The status code is propagated to the presentation tier by the `/status/` page, and it is ultimately used to direct to the appropriate `/error/` page.

Blockchain Searcher:

In the code base, the Blockchain Searcher module consists of two files in the `backend` directory: `solana_search.py` and `solana_util.py`. The `solana_search.py` file retrieves relevant wallet and NFT data from the Solana Blockchain. When the Worker calls the Blockchain Searcher module, it specifically invokes the `search(request)` method in `solana_search.py`, the main method that retrieves data from the blockchain. The `solana_util.py` file contains a utility class to create Python NFT objects, and it also contains the SearchRequest and SearchResponse class definitions, which standardize communication between the Worker and the Blockchain Searcher. It is important to note again that when an exception is raised in the Blockchain Searcher module, the module is exited, and the exception is propagated up to the Worker.

Gallery Generator:

The Gallery Generator replicates the gallery created by the user (i.e., converts the HTML gallery into a PNG image), and saves the resulting image in Cloudinary. In the codebase, the module consists of two files in the `backend` directory: `img_util.py` and `img_generate.py`. The `img_util.py` file defines an Image class to facilitate image processing, as well as GalleryRequest and GalleryResponse classes, which standardize communication between the Gallery Generator and the Worker. The `img_generate.py`

file defines the `generate_gallery(request)` method, which generates and saves the gallery image specified by the request. When the Worker calls the Gallery Generator module, it invokes this method.

Data Management Tier II

To provide a more in-depth understanding of the data management tier, this section describes our implementation of the Solana Blockchain, solien_addresses.json, cache, and Cloudinary in our project. Please look at the Appendix for a better understanding of the Solana Blockchain, NFTs, the cache, and Soliens.

Solana Blockchain, solien_addresses.json and cache:

Our project uses the Solana blockchain as a data source to read the NFTs associated with the user's inputted Solana wallet address. We read the NFT addresses associated with a Solana wallet address as well as the metadata associated with each NFT address from the Solana Blockchain.
The solien_addresses.json is a json file containing all 4019 Solien NFT addresses. We use the solien_addressess.json file to query the NFT addresses associated with a Solana wallet address, to filter out non-Solien NFTs.
Due to the nature of the Solana Blockchain, retrieving NFT metadata for an NFT address can take several seconds or longer. To ensure a swift user experience, there is a cache which is a local database of json files containing NFT metadata. Each json contains metadata for one NFT and is named `$nftaddress$.json` where `$nftaddress$` is the NFT address of the corresponding metadata. If an NFT is cached, instead of retrieving its metadata from the blockchain, we retrieve its metadata from the cache. The cache is explained in more detail in the appendix.

Cloudinary:

The gallery image built by the application processing tier is stored via Cloudinary, a cloud-based image and video management service. This allows for subsequent retrieval by the user (i.e., on the download page), which would not be possible without an external database, as Heroku does not provide a persistent file system. In brief, the application processing tier sends over the gallery image that should be stored, and Cloudinary returns a persistent link to the image.

Communication Flow II

We now describe in detail the full communication flow between the three tiers.

User Inputs Wallet Address:

This is the first page of our web application. The user is prompted a form to input their Solana Wallet address.

1. The user inputs their Solana Wallet address and then presses the “Search” button, prompting Step 2.

Enqueueing Metadata Retrieval:

After the user has entered their wallet address, a task is enqueued to search for the Solien NFTs belonging to the wallet.

2. The index page sends a request to the application processing tier to enqueue a search for the Solien NFTs corresponding to the wallet (via `/enqueue_search/`).
3. The Server module passes the request onto the Worker module.
4. The Worker module enqueues a background task and responds with the job ID of the task.
5. The Server module sends the job ID back to the index page. This ID will later be used to generate a loading page.

Retrieving Metadata (asynchronous):

At this stage, the application processing tier works with the data management tier to retrieve the Solien NFT data from the Solana blockchain and cache for the wallet address. This task is enqueued by the Worker and is performed asynchronously.

A. The Blockchain Searcher receives a request from the Worker to fetch the Solien NFT data associated with the user’s wallet address.
B. The Blockchain Searcher uses The Blockchain API to query the Solana blockchain and get a list of all of the NFT addresses associated with the wallet address. If the wallet address does not exist on the blockchain, an exception is raised to the Worker.
C. The Blockchain Searcher queries the list of NFT addresses with the Solien Addresses database to remove the non-Solien NFTs addresses from the list, creating a list of

just the Solien NFT addresses associated with the wallet. If this list is empty, the wallet contains no Solien NFTs and we raise an exception to the Worker.

D. The Blockchain Searcher iterates through the list of Solien NFT addresses to retrieve metadata associated with each NFT. For each Solien NFT address, it checks if the metadata is stored in our internal cache. If so, it creates an NFT object using the metadata from the cache and appends the NFT object to a list of NFT objects. If the metadata is not stored in the cache, the Blockchain Searcher uses The Blockchain API to retrieve the metadata for the Solien NFT address directly from the Solana blockchain. If an error occurs when retrieving metadata from the blockchain, it raises an error to the Worker. Then the Blockchain Searcher creates an NFT object using the NFT address and metadata from the blockchain and appends this to the list of NFT objects.
E. The Blockchain Searcher fulfills the request and returns the list of Solien NFT objects back to the Worker.

Polling the Worker for Metadata:

While the Worker retrieves the appropriate metadata from the Solana blockchain in the background, the presentation tier regularly polls the application processing tier to check if the task has been completed successfully.

Note: *Although the diagram shows steps 7-10 occurring after A-E, this is not always the case, since the worker program executes these steps asynchronously. Steps 7-10 are repeated until step E has been completed.*

6. The user is redirected to the loading page, which is specific to the background task described above.
7. The loading page requests a status update from the application processing tier, to check if the NFT metadata has been retrieved successfully (via `/status/`).
8. The Server module polls the Worker module.
9. The Worker module responds with the status of the job, along with the appropriate metadata if the task is complete.
10. The Worker module's response is sent back to the loading page by the Server module. Not shown: If the response specifies that an error occurred while executing the task, then the loading page redirects to the appropriate error page.

User Creates Gallery:

Once directed to the gallery creator page, the user creates and customizes a gallery of their NFTs.

11. The user designs their gallery by clicking and dragging the images of their NFTs. Once they are satisfied, they may press the “Create” button, which generates a list of the URLs from the NFTs in their gallery and prompts the request in Step 12 with a list of URLs (specified as ID strings) as an argument.

Enqueueing Gallery Build:

After the user has finished creating their gallery, a task is enqueued to create and store an image of the gallery.

12. The gallery creator page sends a request to the application processing tier to enqueue a task to create and store an image of the gallery (via `/enqueue_gallery/`).
13. The Server module passes the request onto the Worker module.
14. The Worker module enqueues a background task and responds with the job ID of the task.
15. The Server module sends the job ID back to the index page. This ID will later be used to generate a loading page.

Building and Storing Gallery Image (asynchronous):

At this stage, the application processing tier works with the data management tier to create and store an image of the gallery. This task is enqueued by the Worker and is performed asynchronously.

F. The Gallery Generator receives a request from the Worker to build and save the gallery image created by the user in the presentation tier.
G. The Gallery Generator replicates the HTML gallery (specified by the background image and a list of NFT images) as a PNG image.
H. The Gallery Generator stores the image in Cloudinary, which returns a link to the gallery image.
I. The Gallery Generator fulfills the request and returns the gallery image link back to the Worker.

Polling Worker for Gallery Image Link:

While the worker builds and stores the gallery image in the background, the presentation tier regularly polls the application processing tier to check if the task has been completed successfully.

Note: *Although the diagram shows steps 17-20 occurring after F-H, this is not always the case, since the worker program executes these steps asynchronously. Steps 17-20 are repeated until step H has been completed.*

16. The user is redirected to the loading page, which is specific to the background task described above.
17. The loading page requests a status update from the application processing tier, to check if the image has been created and stored successfully (via `/status/`).
18. The Server module polls the Worker module.
19. The Worker module responds with the status of the job, along with the appropriate image link if the task is complete.
20. The Worker module's response is sent back to the loading page by the Server module. Not shown: If the response specifies that an error occurred while executing the task, then the loading page redirects to the appropriate error page.

User Downloads Gallery:

After the gallery image has been created and stored on Cloudinary, the user can download the image.

21. The image URL from the response is displayed to the user and the "Click Here to Download Image" href is set to the image URL. The user may now click on the link to redirect to the image URL and save the image.

Section 3: Design Challenges

The Blockchain API

At the start of the semester, our team was brand new to Blockchain technology. Understanding the concepts of cryptocurrency blockchains on its own is a difficult task, and we decided to work with the Solana Blockchain, which only became widely used in July/August 2021. Because of this, there were very few resources available online in September, and we were left reading the Solana documentation from scratch. From there, we learned about Web3 and tried to implement our own library to read data from the Solana blockchain. This is where we encountered our first failure. Eventually, using a Web3 JavaScript library, we were able to read data off of the blockchain given an NFT or Wallet address. However, the data we received was encoded, and we could not decode it effectively. At this point, we found The Blockchain API, which provided us with several functions that we needed for this project and has allowed us to have a relatively-simple introduction to creating a web application involving blockchain technology.

Cache

Due to the nature of the Blockchain, reading data can often take upwards of 20 seconds. We wanted our application to be swift, and so we decided to implement a cache of the Solien NFTs featured in our project. This cache was acceptable because the data of NFTs does not change unless the original creators decide to change something. Since we know the creators of the Solien NFTs, we were guaranteed that this metadata would not change and if it did, we would be notified and able to update the cache. There is a program in the `cache` directory named `cache_solien_metadata.py` that has run through the `solien_addresses.json` (the JSON containing a list of all Solien NFT addresses) and creates a JSON file for every Solien NFT containing their metadata. The program can be repeatedly run until all Soliens are cached, and it also has checks to ensure that the cached metadata has not been corrupted.

Background Tasks

One issue that we encountered after our initial deployment was that several requests would timeout, because Heroku does not allow any request to take longer than 30 seconds. We resolved this issue by implementing background tasks with a Redis Queue (RQ) and a worker. There were a few difficulties in implementing processes as background tasks. First, there was surprisingly little documentation on working with RQ in Heroku, particularly with

respect to loading page design. Nevertheless, we were able to find an example of how to poll a job's status. Based on this example, we designed our presentation tier such that it would regularly poll the status of the background tasks, while displaying a loading screen to the user.

Second, error handling became more complex with a worker. Exceptions could no longer be caught by the Server module, because the tasks were run remotely. Indeed, it was the worker that was responsible for the background tasks, and so the Server module needed to parse the traceback of a failed job to find an exception (which it subsequently converted to an error code).

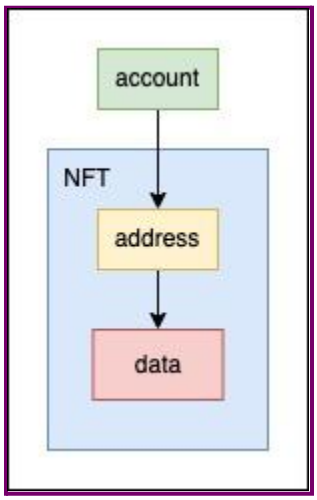
Section 4: Appendix

Solana Blockchain:

In our project, the Solana Blockchain is used as our main data source. The most simple way of describing the Solana Blockchain for our project's use-cases is it is a linked-list with nodes that contain account addresses which point to data. For our project, the most important data associated with these account addresses are NFTs (non-fungible tokens).

NFTs:

NFT stands for non-fungible token. Essentially, an NFT is a unique address which points to some data. As I said in the description of the Solana Blockchain, account addresses point to NFTs. There exists only one NFT with its unique address-data pair on the Solana blockchain and an NFT can only be in one account. This allows for NFTs to be "non-fungible", in the sense that user's can own unique digital items as NFTs which are not interchangeable with other NFTs on the blockchain. The most common NFT data relates to a piece of artwork, containing important data like the name, description, and a link to the art.



Section 5: Sources

Blockchain:

The Blockchain API: https://docs.theblockchainapi.com/
The Blockchain API (cont.): https://github.com/BL0CK-X/the-blockchain-api/blob/main/examples/solana-nft/get-nft-metadata/python_example.pyv
The Blockchain API (cont. pt. 2): https://github.com/BL0CK-X/the-blockchain-api/blob/main/examples/solana-wallet/get-wallet-nfts/python_example.py
Solana Blockchain: https://solana.com/
Benefits of Blockchain: https://ethereum.org/en/what-is-ethereum/
Solien NFTs Website: https://soliens.io/

Background Tasks:

Redis Queue with Heroku: https://devcenter.heroku.com/articles/python-rq
Redis Queue Example: https://beenje.github.io/blog/posts/running-background-tasks-with-flask-and-rq/

Image Creation:

Cloudinary API: https://cloudinary.com/documentation/image_upload_api_reference
Pillow API: https://pillow.readthedocs.io/en/stable/

Front End:

Video background: https://dev.to/terieyenike/create-a-video-landing-page-with-html-and-css-9b
Video background (cont.): https://www.youtube.com/watch?v=Gx_7GQtSdpc&t=618s
Loading animation: https://loading.io/css/
Using images: https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial/Using_images
Using images (cont.): https://softauthor.com/javascript-working-with-images/
Gallery drag and drop: https://www.youtube.com/watch?v=jfYWwQrtzzY&ab_channel=WebDevSimplified
Color picker: https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input/color
Background Selector: https://codepen.io/Noah57/pen/ZmWWqP