Pathogen: A Blockchain Program for Public Health Data
v1.0

Jason Wu
Advised by Dr. Timos Antonopoulos
CPSC 490
Senior Project

In partial fulfillment of the requirements for the
degree of Bachelor of Science in Computer Science

Yale School of Engineering and Applied Science
May 7th, 2022

Pathogen: A Blockchain Program for Public Health Data

Jason Wu

jason.c.wu@yale.edu
https://pathogen.jasonwu.io

CPSC 490
Senior Project
Dr. Timos Antonopouos

ABSTRACT

Coronavirus disease 2019 (COVID-19) has revealed major technical and social issues with modern methods of collecting public health data. National COVID-19 case surveillance currently uses an exceptionally expensive data supply chain involving multiple hospitals, healthcare providers, and laboratories. Data from these sources must then be channeled through several layers of legal approval, processed, and eventually sent to the Centers for Disease Control and Prevention (CDC) for epidemiological analysis.[1] This process is not only slow, but also poses concerns for data privacy, as the identifiability of data is reliant on there being no bad actors in the pipeline. Moreover, it results in an underreporting problem, as citizens have no strong incentives for providing their data to the supply chain.[2] This project proposes a program called *Pathogen* on the Solana blockchain, with the purpose of using blockchain protocols to solve issues with traditional public health data collection. Pathogen solves data privacy vulnerabilities by using unidentifiable Solana accounts to store diagnostic data, and also provides incentive for users to submit data via cryptocurrency rewards. Furthermore, the fast transaction speeds of the Solana blockchain makes collected profiles quickly accessible, meaning that both large corporations and independent researchers have the ability to view and analyze the data as soon as possible. The deliverables of this project include two primary components: a proof-of-concept program written using the Anchor framework, and a design for a more complex version of Pathogen for use in the real world (referred to as the *Pathogen Plan*). The Pathogen proof-of-concept uses native Solana (SOL) as a reward for submitting healthdata, whereas the Pathogen Plan proposes a much more sophisticated Solana Program Library token (SPL-token). The Pathogen proof-of-concept is currently deployed on the Solana DevNet and accessible at https://pathogen.jasonwu.io/. Users can interact with the program directly using a Phantom wallet connected to the Solana DevNet: https://phantom.app/.

1 INTRODUCTION

Section 1 of this report outlines existing methods of public health data collection, and also provides a brief overview for blockchain terminology and concepts relevant to the project. Section 2 discusses the scope and design of the proof-of-concept Pathogen project, and how it was created at the implementational level and what technical challenges were faced. Section 3 discusses the design of the Pathogen plan and provides a roadmap for implementing these changes on the proof-of-concept project. Finally, the conclusions of the project are discussed in Section 4 followed by references and acknowledgements.

1.1 Issues With Centralized Public Health Data Collection

The COVID-19 pandemic has revealed critical problems with public health data infrastructure in the United States. One major issue is that inconsistent bottlenecks in diagnostic test result processing have created very unpredictable lag-times between when an individual becomes tested and when their result is reflected in official statistics.[3] This results in time series infection data underreporting cases, as positive test results may exist but not be processed at any given point in time. Another issue that stems from regulation is the inconsistency of negative test result reporting. Labs are required by law to report positive cases to their respective state governments.[3] Negative tests however, are not covered by this regulation which results in poor data quality of negative test results as well as underreporting. This is an immense issue as negative test data is important for analyzing robust infection statistics and understanding cases with symptoms that may not have yet reached the diagnostic threshold. In addition, across states (and countries), there is also a lack of a uniform standard for data quality in diagnostic reports.[2,3] This is highly inefficient for two reasons. The first is that the inconsistent data format of reports requires lots of processing to be done in order to reasonably compare cases, which takes a lot of time and resources to complete. The second is that different data formats often have data fields that do not overlap, resulting in incomplete datasets. For example some profiles may contain information on factors like race/ethnicity, which is important for finding socio-economic related factors for infection, while profiles from another state/country may not include these details.

Outside of technical and logistical pain points described above, there are also large concerns around data accessibility that have recently come to light. While the COVID-19 infection dashboard created by Johns Hopkins University initially led states to create reporting dashboards of their own, recently many states have been shutting these down, making their infection data incredibly difficult to find and visualize.[4] This trend towards lowering accessibility of health data has also manifested through the decreasing public reporting frequency of COVID-19 data across states (Figure 1). Some of these metrics have been disappearing as a result of budget constraints, a concern for individual privacy, and possibly government censorship in some countries.[4] Regardless of the cause, these events bring to question whether governments have the moral

Updates per week

1 day / week   3 days   5 days   Daily   Varies for cases / deaths
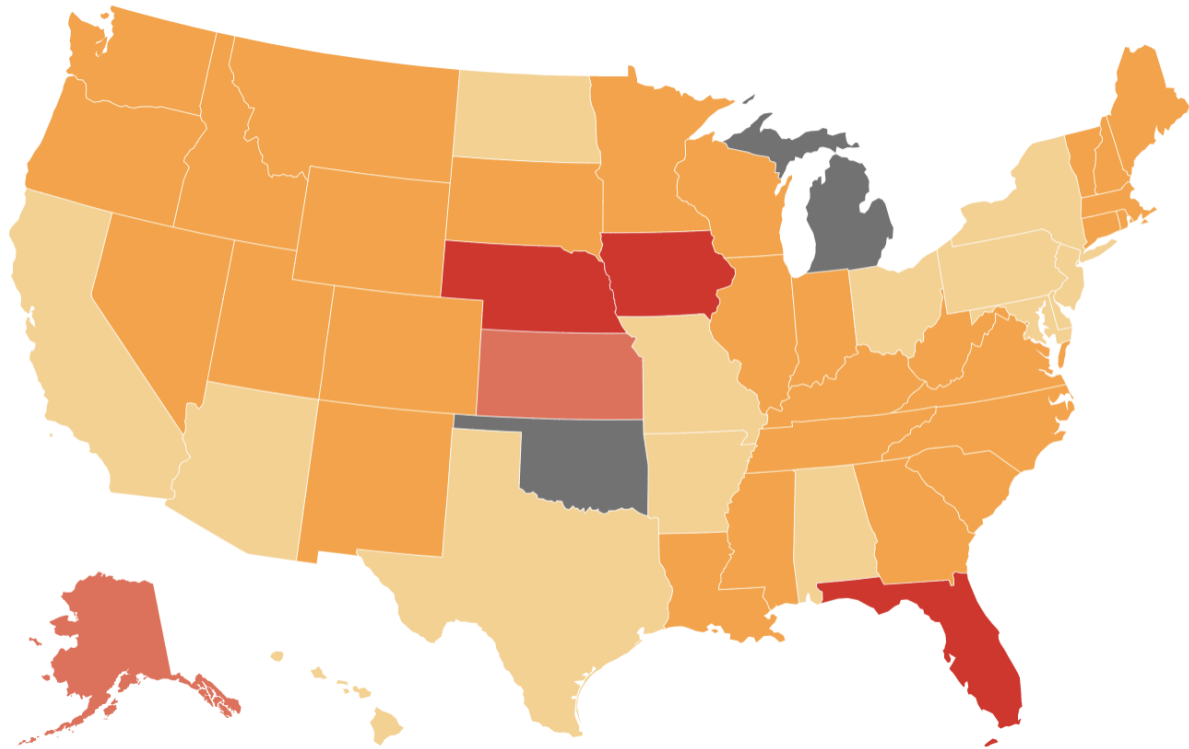
Map: Emily Barone • Source: JHU Coronavirus Resource Center

TIME

Figure 1. Reporting frequency of COVID-19 case and death counts. JHU Coronavirus
Resource Center.

obligation to make infection data publicly available at all times. While having this data is crucial
to finding ways to limit spread of infection, governments are ultimately centralized entities that
have the ability to publish and takedown specific types of information as they decide. While
there are requirements in place for data accessibility via the International Health Regulations
(IHR), they are incredibly difficult to enforce. For example, China reportedly sat on the genome
of the virus and patient data for some time before informing the World Health Organization
(WHO) despite the IHR requiring "timely" and "accurate" reporting of public health information
regarding the outbreak.[5] Thus, as long as public health data is owned and released by centralized
entities, accessibility of infection data will always be a concern.

1.2 The Current System

As discussed in Section 1.1, COVID-19 infection reporting varies heavily between states. At the national level, there is a need to aggregate all of these reports in order to produce infection statistics for the United States as a whole. The current process for collecting public health data reflects this need for aggregating data from several ununified data sources, and exists in the form of a massive data supply chain. At the high level, each state will aggregate data reports from hospitals, healthcare providers, and laboratories under their jurisdiction. The data then gets processed and sent to the CDC. Then, as required by international law, the CDC sends the data over to the World Health Organization (WHO) and publishes the case profiles publicly at https://data.cdc.gov/.
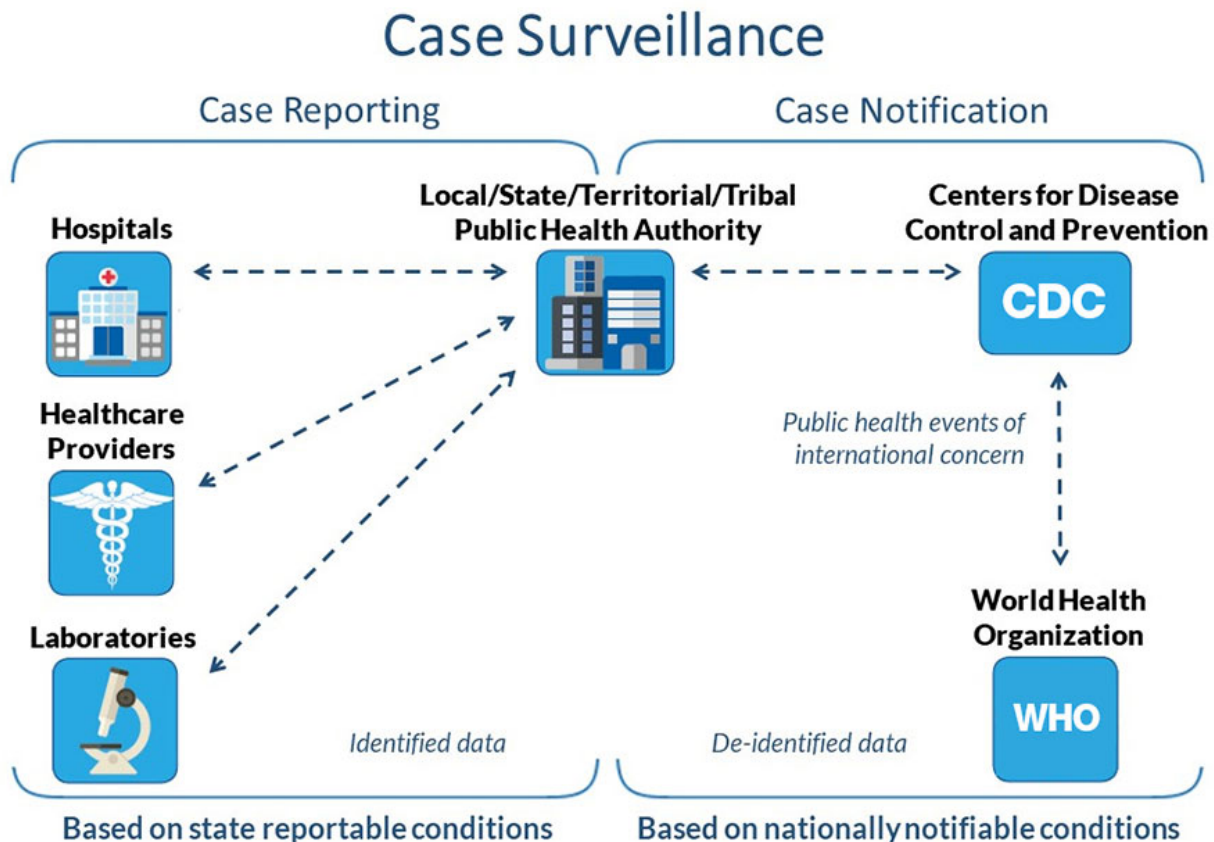

Figure 2. National COVID-19 case surveillance. Centers for Disease Control and Prevention.

Given the number of data sources and levels in the data supply chain depicted above, it is clear why the delays in confirmed case reports explained in Section 1.1 exist. Since patient profile data must be processed in batches, the time between a case occurring and eventually showing up in the public CDC report can be inconsistently high if there is even a single bottleneck in the pipeline. Another issue worth noting is that the data is not de-identified until after it reaches a

local public health authority. This means that if the public health authority either fails to sufficiently make the data unidentifiable, then data privacy will be at risk. In an ideal system, we would like for the profile data to be unidentifiable the moment it enters the pipeline. However, this is not currently the case as the original data sources are labs, hospitals, and healthcare providers, all of which need biographical patient details for their operations.

Additionally, it helps to understand what exactly the published data on the CDC website looks like. Figure 3 below is taken directly from the CDC website, and shows a table preview of the published public use data. Note that each row represents a de-identified patient, and that the data format is unified. There are many missing fields for most rows, which makes sense as the sources of data these profiles are coming from varies immensely as discussed earlier. In addition, this data is hosted by the CDC website, making the data's accessibility subject to the issues of a centralized entity owning data discussed in Section 1.1.

Table Preview

cdc_c...	cdc_r...	pos_s...	onset...	curre...	sex	age_g...	race_...	hosp_...	icu_yn	death...	medc...
2021/12/29	2021/12/29			Laborator...	Male	10 - 19 Ye...	Black, No...	No	Missing	Missing	Missing
2022/01/19				Laborator...	Male	10 - 19 Ye...	Black, No...	Missing	Missing	Missing	Missing
2021/12/28	2021/12/28			Probable ...	Male	10 - 19 Ye...	Black, No...	Missing	Missing	Missing	Missing
2021/12/22	2022/01/03		2021/12/22	Laborator...	Male	10 - 19 Ye...	Black, No...	No	Missing	No	Missing
2020/12/17	2022/01/21	2020/12/19	2020/12/17	Laborator...	Male	10 - 19 Ye...	Black, No...	No	Unknown	Missing	Yes
2022/01/23	2022/01/23			Laborator...	Male	10 - 19 Ye...	Black, No...	Missing	Missing	Missing	Missing
2021/08/09	2021/08/09			Laborator...	Male	10 - 19 Ye...	Black, No...	No	Missing	No	Missing
2020/12/27	2022/02/24	2020/12/28	2020/12/27	Laborator...	Male	10 - 19 Ye...	Black, No...	Unknown	Missing	Unknown	Missing
2021/01/05	2021/01/05			Laborator...	Male	10 - 19 Ye...	Black, No...	Missing	Missing	Missing	Missing
2021/08/01	2021/08/11		2021/08/01	Laborator...	Male	10 - 19 Ye...	Black, No...	Missing	Missing	Missing	Missing
2021/08/28	2022/02/24	2021/08/28		Laborator...	Male	10 - 19 Ye...	Black, No...	Unknown	Missing	Unknown	Missing
2021/09/14	2021/09/15	2021/09/14		Probable ...	Male	10 - 19 Ye...	Black, No...	No	Missing	No	Missing
2021/09/01	2021/09/14		2021/09/01	Laborator...	Male	10 - 19 Ye...	Black, No...	Missing	Missing	Missing	Missing

Figure 3. COVID-19 case surveillance public use data. Centers for Disease Control and Prevention. https://www.cdc.gov/coronavirus/2019-ncov/covid-data/faq-surveillance.html

1.3 Basic Blockchain Concepts

A blockchain is an immutable data structure that validates and records transactions in a completely decentralized fashion. Transactions on a blockchain typically involve the transfer of cryptocurrency (e.g. Bitcoin, Ethereum, Solana, etc.), but they can also include other data types as well like non-fungible tokens (NFTs). Transactions occur between wallets, which are

public-private key pairs that represent ownership on a blockchain. Most blockchains have several nodes responsible for validating transactions. When a transaction is broadcasted across the network, nodes must decide whether to accept the transaction into the ledger via a process called *consensus* (Figure 4). Once the nodes reach consensus, the transaction is added to the ledger as a new block, and the blockchain is updated for all nodes. This consensus process varies depending on the implementation of the blockchain. A lot of networks like Bitcoin and LiteCoin use a Proof-of-work (PoW) consensus algorithm, which requires that nodes in the network solve cryptographic puzzles in order to prove their honesty.[6] Other networks like Solana and soon Ethereum use a consensus algorithm called Proof-of-stake (PoS), which require validators to stake cryptocurrency to prove their honesty. PoW and PoS are the most popular consensus algorithms, though they each come with their own drawbacks. Since PoW is highly dependent on computing power, it has a very high energy cost from crypto mining that ultimately makes it less sustainable. While PoS is less wasteful of energy, it encourages hoarding of cryptocurrency resulting in less liquidity and overall spending.
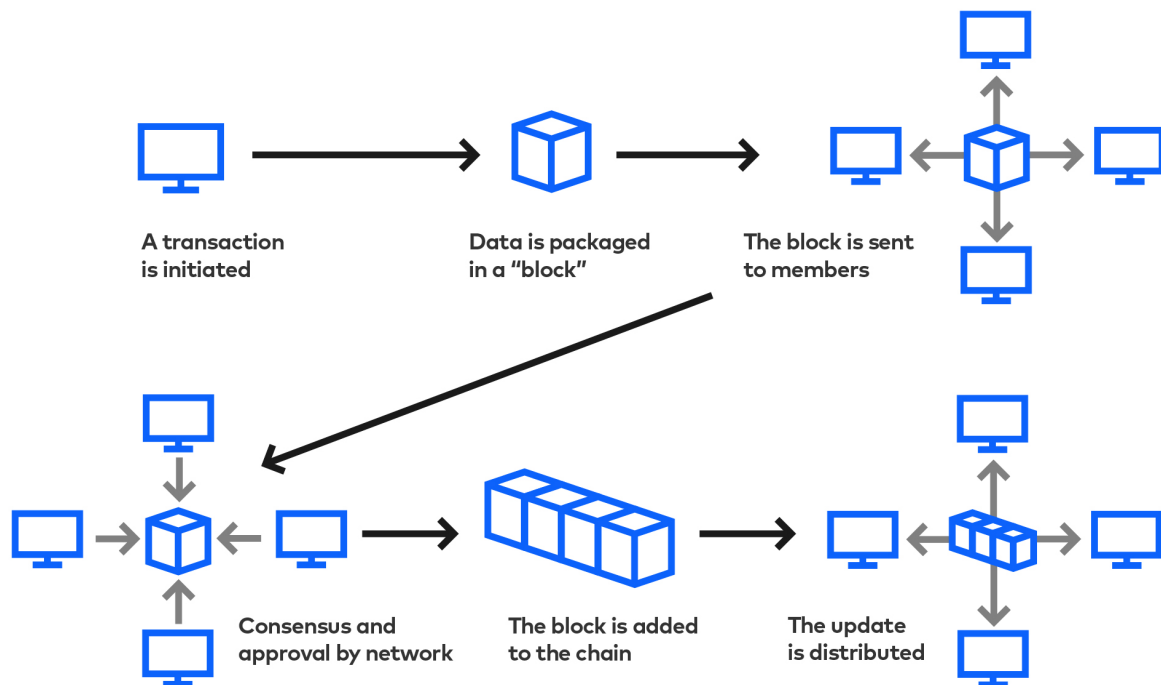

Figure 4. Validating transactions on a blockchain.
https://www.slalom.com/insights/how-blockchain-will-disrupt-your-industry

While the most widely known use-case for blockchains is maintaining a ledger of transactions for cryptocurrency, many networks offer capabilities beyond simple transactions. Many systems like Ethereum have the ability to host decentralized applications in the blockchain. This means

programmers can write programs that run in the blockchain using the computing power of the network. This is extremely exciting in that it enables applications to store and manipulate data in a completely decentralized way. This area of application development is known as *web3*, and represents the rising popularity of using decentralized blockchain systems to create production programs.[7] Many popular products like NFT marketplaces and DeFi (decentralized finance) protocols are coming out of the web3 movement, and the space is continuing to grow in popularity at the time of writing.

1.4 Solana

The most popular network for creating decentralized applications (dApps) is Ethereum, founded by Vitalik Buterin. While Etherum is a massive network that has formed the scaffold for many popular dApps, there are several issues with the network at the time of writing. While Ethereum is moving towards PoS, it currently uses a PoW consensus algorithm. This has resulted in extremely high transaction fees (known as "gas fees"). This means that even for small operations like transferring a small amount of ETH from one wallet to another, users may have to pay a hefty gas fee that may be more than the transfer amount itself. In addition, the Ethereum network can only handle around 30 transactions per second at the time of writing.[8] While this bottleneck may be increased in the future using techniques like sharding, it will likely take a significant amount of time before this is complete.

Recently a new contender for dApps has arisen called Solana, founded by Anatoly Yakovenko. Just like Ethereum, Solana allows users to create and deploy dApps, but it comes with many other improvements that make it highly advantageous over Ethereum. Solana uses a Proof-of-history (PoH) consensus algorithm, which is a modified form of PoS that uses verifiable delay functions (VDF) to validate blocks. This makes transactions incredibly fast, supporting up to 710,000 transactions per second.[9] Furthermore, the lack of PoW makes transaction fees incredibly low, and practically negligible compared to Ethereum gas fees. One final benefit is that Solana apps are written in a popular programming language Rust, whereas Etherum has its own language called Solidity. Outside of the network capabilities, at the software level Solana also supports the creation of fungible and non-fungible tokens much like Ethereum ERC20 and ERC721 tokens respectively. These can be created using the Solana Program Library (SPL).

For developers, working with Solana dApps is very different than Ethereum. On the Ethereum network, apps are contained in smart contracts which typically have functions and store data. On the other hand, smart contracts on Solana can only have functions and are unable to directly store data. In other words, Ethereum programs are stateful whereas Solana programs are not. In order to store program data on the Solana network, programs must create accounts, which are entities on the Solana blockchain with public-private key pairs, and store the data within these accounts. These accounts can then have their data modified by the program.

2 PATHOGEN PROOF-OF-CONCEPT

2.1 Project Scope & Goals

The Pathogen proof-of-concept is intended to be a well-scoped V1 of the greater Pathogen project. The main goal of Pathogen is to construct the anonymized patient table shown in Figure 3 in a way that adheres to three main requirements: speed, accessibility, and incentive alignment. While we have discussed speed and accessibility of data collection quite a bit in Section 1, incentive alignment is a topic that was not discussed as much since it has less to do with the COVID-19 data supply chain itself and more to do with diagnostic testing. In the past few years, many individuals have expressed a reluctance to complete regular diagnostic testing for COVID-19. These reasons vary from people fearing the consequences of a positive test result or even just finding the task too big of a hassle.[10] We should recognize that individuals do not have any forms of positive incentives to complete regular tests, only negative incentives (e.g., travel/work/legal requirements). Thus, the third issue the proof-of-concept should tackle is incentive alignment and providing citizens with good reasons to complete regular testing.

2.2 Project Design

The general design of the Pathogen proof-of-concept follows a simple bounty-reward system. Suppose that Organization X is in need of a dataset of 100,000 unidentified patient profiles for a research study. To provide individuals with an incentive for providing their anonymous diagnostic data, they can create a bounty with a value Y. For each individual who provides their data to the dataset request, a portion of the bounty Y is given to the person as a reward. This solves the incentive alignment problem quite well, as it gives a transparent reward to contributors and also requires that organizations in need of these large datasets provide a reasonable bounty to keep people interested.

While this design solves the incentive alignment issue, we still have to handle speed and accessibility. In addition, we also need some form of currency for storing the value of the bounty. All three of these complications are solved by developing Pathogen as a Solana dApp. The fast transaction speeds of the network make the speed of data publishing extremely fast. Furthermore, the public accessibility of the data is inherently robust because we store health data within Solana accounts, which are public. We also make sure the data is immutable by providing no means of updating submitted profiles, thereby guaranteeing that organizations that request datasets are unable to hide or remove this data from the public. As for storing the value of the bounty, we use Solana's native currency SOL to store and reward bounties.

2.3 Anchor Framework

The proof-of-concept was implemented using the Anchor Framework, which is one of the most popular frameworks for developing production Solana apps.[11] The framework itself provides some nice tooling for development and useful libraries for handling things like private key signing, but it does not fundamentally change the way Solana code is written. All functions and data definitions are still written in Rust. This framework was chosen as one of the personal learning goals for this project was learning to create fullstack Solana dApps under industry standards. Anchor includes boilerplate code for writing integration tests in TypeScript, and has many declarative language details in place that ensure that developers are following best practices. It was suggested by a friend who started a DeFi company recently funded by South Park Commons.[12]

2.4 Defining Solana Accounts

I started by first defining the Solana account structures for storing *pathogens* and *profiles*. Pathogens are entries that define a disease of interest (e.g. COVID-19), whereas profiles are deidentified patients that contain diagnostic and biographical information. Pathogens and profiles have a one to many relationship, as each pathogen can have several profiles contained in its dataset. These are the two main account structures used to store data. For the sake of the scope of the proof-of-concept, very minimal data was added to these accounts (Figure 5).

```rust
// 1. Account structs
#[account]
pub struct Pathogen {
    pub creator: Pubkey,
    pub code: String,
    pub name: String,
    pub reward_per_profile: u64,
    pub total_profiles: u64,
    pub created_at: i64,
}

#[account]
pub struct Profile {
    pub creator: Pubkey,
    pub pathogen: Pubkey,
    pub latest_test_result: String,
    pub latest_test_result_date: i64,
    pub age: u8,
}
```

Figure 5. Pathogen Accounts.

One of the challenges with defining accounts is that space usage must be computed manually. This includes components outside of the struct field definitions such as the discriminator. To make the space computations somewhat less cryptic in the code, I defined several constants that illustrate the space requirements more verbosely. Note that the space requirements of each account are added as a public constant on each of the structs named LEN. This constant can then be used by Anchor to determine how much space to allocate for each of these Solana accounts.

```rust
// 2. Sizing constants for determining space requirements
// Shared
const DISCRIMINATOR_LENGTH: usize = 8;
const PUBLIC_KEY_LENGTH: usize = 32;
const STRING_LENGTH_PREFIX: usize = 4; // Stores the size of the string
const TIMESTAMP_LENGTH: usize = 8;

// Pathogen account size properties
const MAX_CODE_LENGTH: usize = 25 * 4; // 25 chars max
const MAX_NAME_LENGTH: usize = 50 * 4; // 50 chars max
const REWARD_PER_PROFILE: usize = 8;
const TOTAL_PROFILES_LENGTH: usize = 8;

// Profile account size properties
const AGE_LENGTH: usize = 1;
const LATEST_TEST_RESULT_LENGTH: usize = 25 * 4; // 25 chars max

// 3. Total size implementations
impl Pathogen {
    pub const LEN: usize = DISCRIMINATOR_LENGTH
        + PUBLIC_KEY_LENGTH // Creator
        + (STRING_LENGTH_PREFIX + MAX_CODE_LENGTH) // Code
        + (STRING_LENGTH_PREFIX + MAX_NAME_LENGTH) // Name
        + REWARD_PER_PROFILE // Reward per profile
        + TOTAL_PROFILES_LENGTH // Total profiles
        + TIMESTAMP_LENGTH; // Created at
}

impl Profile {
    pub const LEN: usize = DISCRIMINATOR_LENGTH
        + PUBLIC_KEY_LENGTH // Creator
        + PUBLIC_KEY_LENGTH // Pathogen
        + (STRING_LENGTH_PREFIX + LATEST_TEST_RESULT_LENGTH)  // Latest test result
        + TIMESTAMP_LENGTH // Latest test result date
        + AGE_LENGTH; // Age
}
```

Figure 6. Pathogen Account Space Calculations.

2.4 Defining Solana Instructions

In Solana, instructions are the equivalent to an endpoint in a REST API. They are essentially callable functions that can manipulate, but not persist data. This is because, as we recall earlier, Solana programs are stateless. For the purpose of our proof-of-concept, two instructions are

required. One is creating a pathogen, and the other is creating a profile for a pathogen. We want any user to be able to create a pathogen so long as they are able to pay for the bounty pool. For a profile however, we require that the profile is created with a reference to a valid pathogen. These constraints are relatively straightforward to define using the Anchor framework. Note that in the code snippet below, the payer refers to the payer of transaction fees (Figure 7). Also note that we assign the space variable to the LEN constant we defined previously using space requirement calculations.

```rust
#[derive(Accounts)]
pub struct CreatePathogen<'info> {
    #[account(init, payer = creator, space = Pathogen::LEN)]
    pub pathogen: Account<'info, Pathogen>,
    #[account(mut)]
    pub creator: Signer<'info>,
    pub system_program: Program<'info, System>,
}

#[derive(Accounts)]
pub struct CreateProfile<'info> {
    #[account(init, payer = creator, space = Profile::LEN)]
    pub profile: Account<'info, Profile>,
    #[account(mut, rent_exempt = enforce)]
    pub pathogen: Account<'info, Pathogen>,
    #[account(mut)]
    pub creator: Signer<'info>,
    pub system_program: Program<'info, System>,
}
```

Figure 7. Pathogen Instructions Interfaces.

For brevity, the implementation of these instructions are not explained in detail in this report, as the code is relatively descriptive on its own. To view their implementations in Rust, view the file here: https://github.com/wu-json/pathogen/blob/main/programs/pathogen/src/lib.rs

2.4 Testing

One of the large benefits of the Anchor Framework is that it sets up a TypeScript testing framework for you (Mocha/Chai) as well as a Solana RPC client.[13,14] This allows us to interact with the instructions created in the previous steps quite easily using the Solana JSON RPC API. In addition to ensuring that the instructions work end-to-end, the benefit is that the Anchor Solana client used to make calls to the instructions works the exact same in tests and web3 clients. This meant that I was able to adapt a lot of the code written for tests to be used in the React web client I built later into the project. Another large benefit of the testing scaffold that Anchor creates is that it is possible to write complete integration tests. This means that constraints defined earlier (e.g. a profile can only be created with a reference to a valid pathogen

account) can be enforced by using different keypairs in the integration tests. Figure 8 contains a code snippet of an integration test I wrote for creating a pathogen.

```
it('can create a pathogen', async () => {
    const startingBalance = await getAccountBalance(
      provider.wallet.publicKey,
    );

    const pathogen = anchor.web3.Keypair.generate();
    await program.methods
      .createPathogen(
        'covid-19',
        'Coronavirus disease 2019',
        new BN(200 * LAMPORTS_PER_SOL),
        new BN(2),
      )
      .accounts({
        pathogen: pathogen.publicKey,
        creator: provider.wallet.publicKey,
        systemProgram: anchor.web3.SystemProgram.programId,
      })
      .signers([pathogen])
      .rpc();

    const pathogenAccount = await program.account.pathogen.fetch(
      pathogen.publicKey,
    );

    const endingBalance = await getAccountBalance(provider.wallet.publicKey);

    assert.ok(startingBalance - endingBalance > 200 * LAMPORTS_PER_SOL);
    assert.ok(
      (await getAccountBalance(pathogen.publicKey)) > 200 * LAMPORTS_PER_SOL,
    );
    assert.equal(
      pathogenAccount.creator.toBase58(),
      provider.wallet.publicKey.toBase58(),
    );
    assert.equal(pathogenAccount.code, 'covid-19');
    assert.equal(pathogenAccount.name, 'Coronavirus disease 2019');
    assert.equal(pathogenAccount.totalProfiles, 0);
    assert.equal(pathogenAccount.rewardPerProfile, 2);
    assert.ok(pathogenAccount.createdAt);
  });
```

Figure 8. Integration Test Example.

2.5 Web Client

The main personal focus of this project was to learn Solana development and Rust. Thus, for the frontend web client I chose to use a framework I was more familiar with so that I could develop quickly. I wrote the web client using the React framework in TypeScript.[15] This was a very straightforward decision to make since it was the framework with which I have had the most

experience, and it also uses TypeScript which is the same language I wrote the Solana instruction tests in, meaning the instruction call code could be re-used. In order to make accessing the data accounts (pathogens and profiles) extremely easy on the frontend, I wrote custom React hooks for each account type. For those unfamiliar with React hooks, they are simply a feature that allow frontend components to access state at any level in the component tree. Figure 9 gives an example of a hook for getting pathogens from the Solana accounts. Note how MyComponent is able to access the pathogen data in a single line.

```typescript
import { useEffect, useState } from 'react';

import useWorkspace from '../useWorkspace';

const usePathogens = () => {
  const { program } = useWorkspace();
  const [pathogens, setPathogens] = useState<any[]>([]);
  const [ready, setReady] = useState<boolean>(true);

  useEffect(() => {
    if (program) {
      program.account.pathogen.all().then(results => {
        setPathogens(results);
        setReady(true);
      });
    }
  }, [program]);

  return { pathogens, setPathogens, pathogensReady: ready };
};

const MyComponent = () => {
  const { pathogens } = usePathogens();
  return <></>;
}
```

Figure 9. Pathogen Hook Example.

Another new challenge of the web client was adding the ability to interface with crypto wallets in the browser. For the scope of this project, I wanted users to be able to use Phantom Solana wallets to interact with the application on the Solana DevNet. Phantom is a rather new company that specializes in Solana wallets specifically, and I wanted to use it since it is one of the few wallets I have not used before (i.e. not MetaMask or Coinbase).[17] Adding wallet connectivity was surprisingly easy using Solana's wallet adapter library, and simply required defining a provider around the application to inject the wallet context.[18] From there, I defined another custom hook called useWorkspace that allowed me to pull data and objects from the wallet context on an adhoc basis. This worked quite well and enabled me to write custom hooks requiring the Solana RPC API client (recall from Figure 9 that the usePathogens hook uses the useWorkspace hook) without much extra code.

For the UI of the web client, I used Figma to make some design prototypes before I started building. This made it very easy to implement the designs using Sass and antd react components, as I already knew what the client should look like as I was building it.
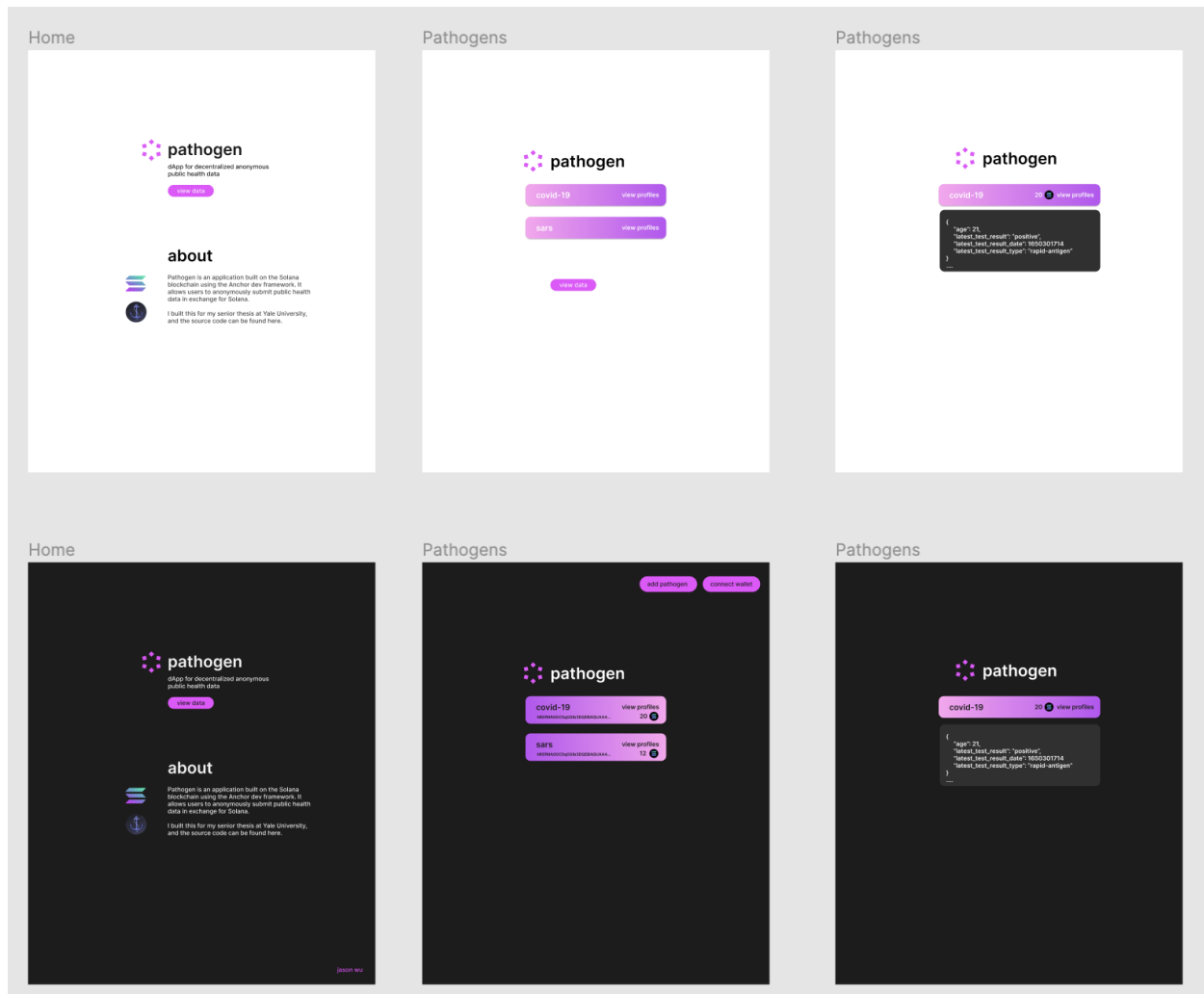

Figure 10. Pathogen Design Prototypes.

2.6 CI/CD

Continuous integration and deployment were a bit more difficult than anticipated as there were no straightforward ways of setting up the Anchor framework in a CI image at the time of writing. Thus, I had to opt for setting up and caching the Anchor CLI binary file using GitHub actions, and then installing Anchor each CI run. This makes the continuous integration workflow take around 15 minutes to set up Anchor and run the instruction integration tests, with most of that

time being used to set up the CLI itself. While this is certainly not ideal, the workflow is consistent enough for the purpose of this project.

```
build-and-test:
  name: Build and Test
  needs: setup-anchor-cli
  runs-on: ubuntu-20.04
  steps:
    - uses: actions/checkout@v2
    - uses: ./.github/actions/setup/
    - uses: actions/cache@v2
      name: Cache Solana Tool Suite
      id: cache-solana
      with:
        path: |
          ~/.cache/solana/
          ~/.local/share/solana/
        key: solana-${{ runner.os }}-v0000-1.8.14
    - run: sh -c "$(curl -sSfL https://release.solana.com/v1.8.14/install)"
      shell: bash
    - run: echo "/home/runner/.local/share/solana/install/active_release/bin" >> $GITHUB_PATH
      shell: bash

    - run: solana-keygen new --no-bip39-passphrase
      shell: bash
    - run: solana config set --url localhost
      shell: bash

    - uses: actions/download-artifact@v2
      with:
        name: anchor-binary
        path: ~/.cargo/bin/
    - run: chmod +x ~/.cargo/bin/anchor

    - run: yarn
    - run: anchor test
```
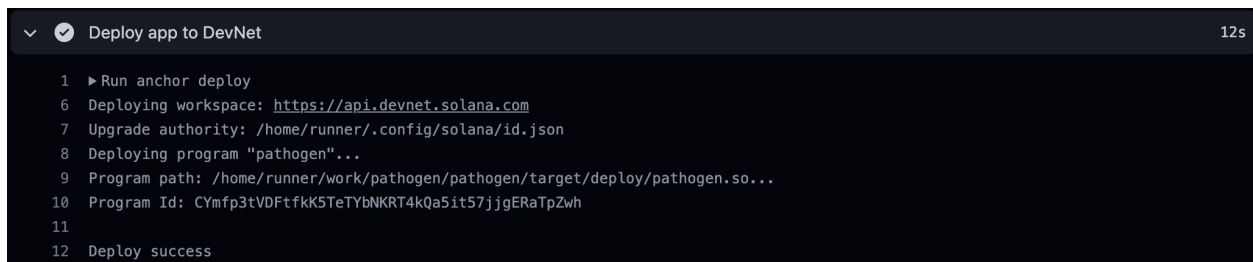
Figure 11. Pathogen CI Steps.

Continuous deployment was also convoluted for similar reasons. While installing Anchor CLI is handled by the workflow steps shown above, for deployment it is necessary to set up the Solana CLI, and also ensure that the deployment keypairs are consistent every deployment. The deployment keypairs must be the same every deployment otherwise the program will be deployed to a different address every time. Another complicated aspect of this is that the private key must be stored in a way where viewers of the GitHub repo cannot steal it, otherwise they will also be able to deploy to the program address. To handle this, I used the GPG library's AES256 encryption command to encrypt the required key pairs for deployment, and committed those to git. From there, I put the passphrase in the secret environment variables in the CI/CD pipeline, and decrypted them as a part of the workflow. This ended up working very well without many hiccups, and protects the app from being redeployed by someone else.

The continuous deployment step currently only runs on commits to main. When the trigger occurs, the continuous deployment workflow will run tests and ensure everything is working, and then automatically deploy the application to the DevNet. While the workflow only makes a DevNet deployment currently, it would be easy to change this to deploy to the Solana TestNet (staging) or MainNet (production). It would just require changing the solana configuration target url before each deployment to the desired network.


```
  ∨ ✓ Deploy app to DevNet                                                    12s

  1   ▶ Run anchor deploy
  6   Deploying workspace: https://api.devnet.solana.com
  7   Upgrade authority: /home/runner/.config/solana/id.json
  8   Deploying program "pathogen"...
  9   Program path: /home/runner/work/pathogen/pathogen/target/deploy/pathogen.so...
 10   Program Id: CYmfp3tVDFtfkK5TeTYbNKRT4kQa5it57jjgERaTpZwh
 11
 12   Deploy success
```
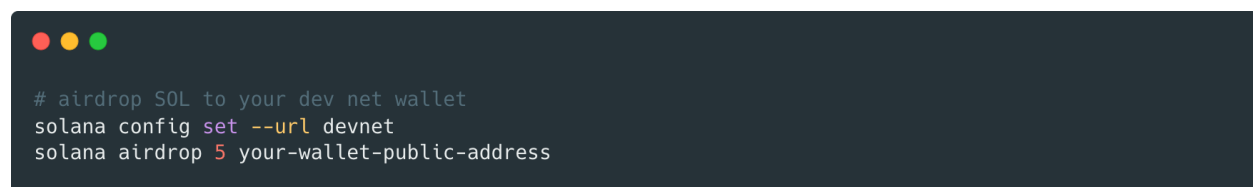
Figure 12. Pathogen Deploy CD Output.

Continuous deployment of the web client was a lot more straightforward, and was done through Netlify's GitHub integration. All commits to main are built and deployed directly by Netlify's service.[19] This meant that I did not need to manually re-deploy updates to the web client or the Solana dApp, meaning the entire repository has complete CI/CD.

2.7 Interacting With Pathogen on the DevNet

Pathogen is currently deployed on the Solana DevNet, and you can interact with it using a Phantom wallet with some SOL in it. To set up a Phantom wallet, go to this website and follow the instructions for your browser: https://phantom.app/. Once you have the Phantom browser extension set up, you need to go into the wallet configuration and choose DevNet instead of MainNet, which will get you onto the right network (Figure 14). Then, you will need to set up the Solana CLI by following the steps here: https://docs.solana.com/cli/install-solana-cli-tools. Once you have the CLI running, you can run the commands in Figure 13 to airdrop 5 SOL to your DevNet wallet. This will give you some funds to create pathogens.


```
# airdrop SOL to your dev net wallet
solana config set --url devnet
solana airdrop 5 your-wallet-public-address
```
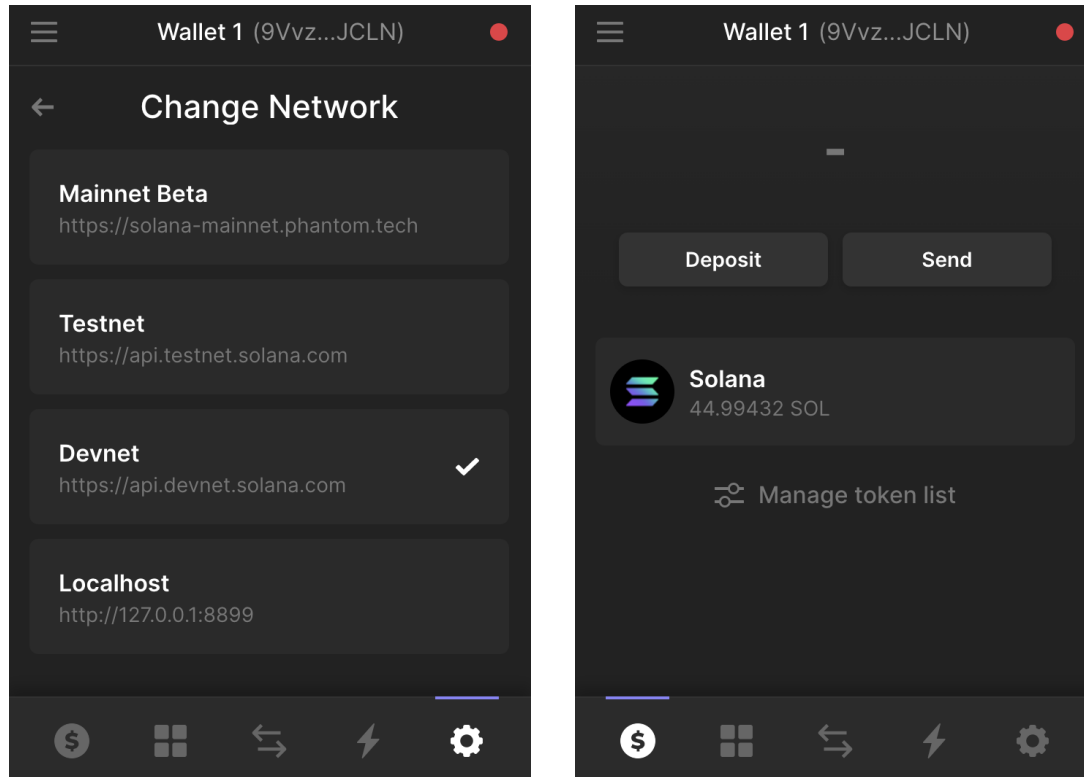
Figure 13. Airdrop SOL to Your DevNet Wallet.

Figure 14. Setting Up Your Phantom Wallet. *Make sure you are on the DevNet otherwise Pathogen will not work properly.*

Now that you have your wallet setup, you can go to https://pathogen.jasonwu.io/pathogens to see and interact with the proof-of-concept application. When you navigate to the site, you should be prompted to connect your Phantom wallet to the application. Once you do that, you should be able to see a list of Pathogens on the screen. I have created two example pathogens as of writing (covid-19 and senioritis-2022). For each of these diseases, you can view the submitted unidentifiable profiles by clicking the view details button of each disease. You will notice that the profiles look really similar to the type of data you would see in the CDC published patients table (Figure 3), which was the goal.

You can then experiment with creating your own pathogens or adding profiles to existing pathogens. There are buttons for each of these and you will notice that clicking one will bring up a form, followed by a Phantom confirmation screen indicating the change in SOL your wallet will experience if the transaction is confirmed.cYou will notice that creating a pathogen will subtract a bounty of SOL from your wallet, whereas adding a profile will pay you a portion of a SOL bounty to your wallet. This is the same behavior as intended for incentivizing patients to provide their health data. Also note that you can change the size of this bounty when you create a pathogen, which allows for the collection of various dataset sizes for organizations that may have less SOL on hand.
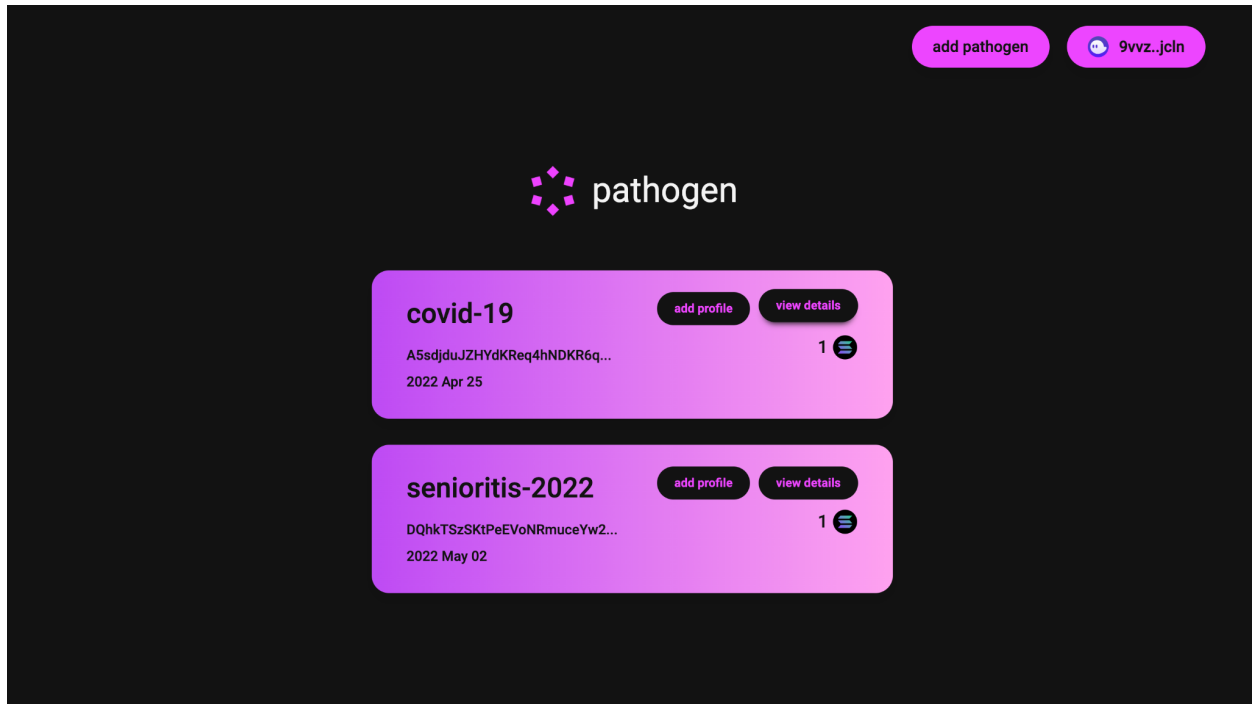
Figure 15. Pathogens Screen With Example Pathogens.

pathogen data
{
  "publicKey": "A5sdjduJZHYdKReq4hNDKR6qimApsGS4sFUxZcSoXmsV",
  "account": {
    "creator": "9VvzAW2rsCQzYZbT1D3bAhaoH34NHDDeSje7DxVJJCLN",
    "code": "covid-19",
    "name": "Coronavirus Disease 19",
    "totalProfiles": 2,
    "createdAt": "2022-04-25",
    "rewardPerProfileInLamports": 1000000000
  }
`

profile data
  {
    "publicKey": "6jJ9Y62Ri2ctPdJqVzLv4fKPC3CnAEJEA64i18uJigTt",
    "account": {
      "creator": "9VvzAW2rsCQzYZbT1D3bAhaoH34NHDDeSje7DxVJJCLN",
      "pathogen": "A5sdjduJZHYdKReq4hNDKR6qimApsGS4sFUxZcSoXmsV"
      "latestTestResult": "negative",
      "latestTestResultDate": "2022-05-02",
      "age": 42
    }
  },
  {
    "publicKey": "F5QFaeEuge2BnAG7159ktSPsZXW8paMGV3WEvebMehTM",
    "account": {
      "creator": "9VvzAW2rsCQzYZbT1D3bAhaoH34NHDDeSje7DxVJJCLN",
      "pathogen": "A5sdjduJZHYdKReq4hNDKR6qimApsGS4sFUxZcSoXmsV"
      "latestTestResult": "positive",
      "latestTestResultDate": "2022-04-29",
      "age": 21
    }
  }
}

Figure 16. Pathogen and Profile Data in JSON.

3 THE PATHOGEN PLAN

While the proof-of-concept project does a good job representing the actual idea of Pathogen and how it works at a high level, there are many loopholes and shortcomings that result from the oversimplified vision of the project it presents. This section of the report discusses what these shortcomings are and how the design of the proof-of-concept can be expanded in order to account for these issues. Most of these features were omitted from the proof-of-concept in order to keep the scope of the code deliverables reasonable for the length of a single semester. If time allows, I will add these after graduation as a side project. Star the GitHub repository to keep up with updates: https://github.com/wu-json/pathogen.

3.1 Replacing SOL With SPL Tokens

One of the issues with using SOL as the bounty/reward currency is that SOL is subject to fluctuations in price beyond the scope of the project. SOL is a rather popularly traded cryptocurrency meaning that it is subject to rapid fluctuations in price as investors gain and lose interest (like with any cryptocurrency). Since Pathogen dataset bounties/rewards are in SOL, their value will also fluctuate with these changes. To alleviate these effects, we should use SPL fungible tokens as bounties/rewards instead of SOL.

SPL tokens are tokens created using the Solana Program Library.[20] For Pathogen, we could have each pathogen create its own SPL fungible token supply representing the disease. For example, if an organization creates a pathogen called *covid-19*, a fresh supply of fungible tokens called *pathogen-covid-19* could be created and rewarded to contributors who provide their health data. This would be beneficial for two reasons. The first reason is that adding this layer of indirection would make the value less coupled (but not completely decoupled) to fluctuations in the price of SOL. The second reason is that the creation of a fungible token supply allows us to define new *tokenomics* for managing the supply of said token. Tokenomics broadly refer to the economics behind fungible tokens in cryptocurrency.[21] In the context of Pathogen, it is important in that it allows us to add a new layer of complexity in ensuring that the management of the supply of the pathogen fungible tokens aligns with our overall goal of incentivizing patients to willingly provide their health data.

How would pathogen tokenomics work if we were to implement them? First, tokens for a disease would be minted with a fixed supply as soon as a pathogen is created (i.e. creating a pathogen called *covid-19* results in X *pathogen-covid-19* tokens being created). For the rest of this paper, we will refer to fungible tokens created this way as PATH tokens. It is important that the supply is fixed at the time of initial minting so that the token value remains deflationary and therefore is not at risk for individuals losing incentive to provide data. From there, there are various measures we can take to adjust the supply to align with our goals as the disease progresses and data comes in over time:

1. *Burn an individual's PATH tokens when they die from a disease.*

   This would decrease the supply of tokens that represent diseases that are more deadly, which would thereby increase the price of the token and result in a more valuable incentive for people to provide their data. While there are some ethical questions about this (e.g. are there moral issues with using someone's life to determine the value of a currency?), the overall result of the process still aligns with the project's goal of making data collection easier and more accessible by incentivizing more people to provide data.

2. *Burn some PATH tokens when people experience serious symptoms from a disease, and mint some when people recover.*

   This uses similar logic to the previous mechanism. Diseases with more serious symptoms will result in a reduced mint supply, thereby increasing the value of the profile reward. On the contrary, we can also increase the supply by minting more tokens when people recover. This mechanism allows the blockchain to regulate the PATH token price for diseases in order to prevent diseases that spread quickly but have a high recovery rate from climbing in price even though people recover from it quite successfully.

3. *Mint more PATH tokens when the mint is dry and symptoms/death rate are getting worse.*

   One dangerous situation could be if a disease is so dangerous that the supply of its PATH tokens depletes completely, as people are dying/getting more sick from the disease and profiles have exhausted the fixed deflationary supply. In this case, there needs to be a mechanism for replenishing the mint when it is totally exhausted. The frequency at which this replenishing could occur could be adjusted based on how quickly the disease is progressing and how bad the symptoms/death rate becomes.

As illustrated by the points above, adding SPL tokens to the Pathogen program brings the complexity of the project up substantially. In fact, a separate thesis could be written on the potential options for the supply regulation of these tokens entirely. In addition to making the project less coupled to the price of SOL, they also enable more robust economic regulatory measures that can elevate the value behind bounties for the average person.

3.2 Incentivizing Time Series Data With DeFi

The DevNet Pathogen proof-of-concept also currently does not have a strong system in place for time series data. While individuals are incentivized to provide their latest test result for a reward, there is not much of an incentive for individuals to regularly submit their test results to form a time series dataset of infection when that bounty runs out. In order to ensure that people have a reason to continuously send their test results to a Pathogen dataset, we can leverage something called DeFi.

DeFi refers to decentralized finance, which is the idea of providing financial services free of any centralized authority like a bank or government.[22] In the context of Pathogen, the particular area of DeFi we are interested in is DeFi lending, which uses technical protocols to lend crypto out for interest. These include protocols like Compound and Anchor, which have been targets of many yield farming strategies in the crypto world.[23]

In order to keep patients interested in providing their diagnostic data on a temporally regular basis, we can do something like the following:

1. When users initially submit their data to a pathogen, lock up their reward for X period of time by investing it in a DeFi lending protocol to earn interest.
2. In intervals of Y weeks, if a user fails to upload a recent diagnostic test within interval Y, their rewards are slashed and sent directly back to the Pathogen mint.
3. If the user has been continuously uploading diagnostic tests to the pathogen dataset and X period is over, pull the rewards from the DeFi protocol and give them back to the user.

The system above is preferable because it keeps users submitting data at a regular temporal interval, and it also does not present an opportunity cost to the user as their funds are continuously earning interest on a DeFi protocol anyway. The one limitation of this is that whatever cryptocurrency is used as the bounty/reward must have a DeFi lending protocol that supports this. If we are using SPL tokens, there may need to be some extra plumbing to get this to work (likely converting the PATH tokens to some intermediate currency). Otherwise, this system works out nicely.

3.3 Security Loopholes

Cybersecurity is always the greatest concern with any dApp, as these applications run on cryptocurrency which can result in great financial losses if even a tiny bug is present. This section will touch upon some security concerns with Pathogen and potential improvements that can be made to prevent them.

One of the major concerns in the Pathogen proof-of-concept is that individuals will abuse profile submissions for the purpose of financial gain. For example, if I receive a negative test result for COVID-19 today from Yale's testing center, there is nothing stopping me from submitting that same test N number of times until I completely drain the bounty. A simple solution to this could be to implement some cooldown period for the frequency with which you can submit profiles. For example, if you try to submit more than one profile per day, you get an error. The issue with this solution however, is that we want to accept updates to one's diagnosis as soon as possible, thus cooldowns are not ideal since they introduce a potential lag in which test results get submitted. Consequently, a more ideal solution would be to provide some digital signature (e.g. an ID) that proves the submitted test result is a new one, validating it when each profile is

submitted, and only accepting the data and providing the reward if the signature is deemed valid. The question here becomes how we can properly validate test results, and whether companies that provide these signatures will give us access to the data needed to validate these in the first place.

Another very large concern is data quality. The Pathogen app currently does no validations (outside of formatting) to verify that information coming from the patient is correct. This is a much more difficult issue to solve than the previous problem, as it has to do less with implementing technical constraints and more to do with the limitations of decentralized systems themselves. In a traditionally decentralized system, no entity should have more authority than others in the network. In the case of verifying biographical and diagnostic data however, there usually needs to be some authoritative third party (e.g. a physician) that verifies the data is correct. In the future, it is entirely possible that an AI system could be capable of doing this, but that is not a solid assumption to make in the present. This issue could be solved by changing the way in which patients access Pathogen. Instead of having everyone enter their data on their own computers, it could be possible to only allow physicians to legally submit data on their patient's behalf. While this does introduce a bottleneck from the patient to the dataset, it would still be significantly faster than the current data supply chain. If authorized third parties are the only entities authorized to submit profiles to the blockchain, it is also important that they stake something valuable (e.g. SOL or some other crypto collateral), so that their assets can be slashed as a punishment if they do something bad on the network (e.g. submitting falsified data). Furthermore, third parties can also be offered some incentive for validating data like a small portion of the pathogen bounty. This is pretty similar to how blockchain nodes validate transactions.

4 CONCLUSION

While limited in scope, the Pathogen proof-of-concept laid a solid foundation for the future of the Pathogen project. The codebase is set up in a way where it would be easy to iterate and add the solutions mentioned in the Pathogen Plan in reasonably sized sprints. In the near future, I hope to build out the SPL-token and DeFi lending protocol components of the project, as they need to be implemented before deploying Pathogen to the Solana MainNet. I plan on keeping the project open-source so that other can fork and experiment with the project on a local Solana net.

Outside of the technical aspects of Pathogen, the potential use of this project in the real world still has many open possibilities. One thought is potentially embedding chips with Solana keypairs into testing kits that will verify a test result and forward the data to the Pathogen program automatically, thereby ensuring diagnostic validity and speed of data collection. I hope that I will be able to discuss these ideas with professionals in healthcare to further understand how decentralized systems can be used to improve health data quality across the world.

REFERENCE

1. Centers for Disease Control and Prevention. (n.d.). *FAQ: Covid-19 data and surveillance*. Centers for Disease Control and Prevention. Retrieved April 29, 2022, from https://www.cdc.gov/coronavirus/2019-ncov/covid-data/faq-surveillance.html

2. *Data around COVID-19 is a mess and here's why that ... - devex*. (n.d.). Retrieved April 29, 2022, from https://www.devex.com/news/data-around-covid-19-is-a-mess-and-here-s-why-that-matters-97077

3. Bloomberg. (n.d.). *Coronavirus Data in the U.S. Is Terrible, and Here's Why*. Bloomberg.com. Retrieved April 30, 2022, from https://www.bloomberg.com/news/articles/2020-05-01/why-coronavirus-reporting-data-is-so-bad

4. Barone, E. (2021, September 29). *Why a popular covid-19 dashboard is struggling to get data*. Time. Retrieved April 30, 2022, from https://time.com/6101967/covid-19-data-gaps/

5. Oona Hathaway and Alasdair Phillips-RobinsDecember 8, Phillips-Robins, O. H. and A., Hathaway, O., & Phillips-Robins, A. (2020, December 8). *Covid-19 and international law series: Who's pandemic response and the International Health Regulations*. Just Security. Retrieved April 30, 2022, from https://www.justsecurity.org/73753/covid-19-and-international-law-series-whos-pandemic-response-and-the-international-health-regulations/

6. Frankenfield, J. (2022, February 8). *Consensus mechanism (cryptocurrency)*. Investopedia. Retrieved April 30, 2022, from https://www.investopedia.com/terms/c/consensus-mechanism-cryptocurrency.asp

7. Roose, K. (2022, March 18). *What is web3?* The New York Times. Retrieved April 30, 2022, from https://www.nytimes.com/interactive/2022/03/18/technology/web3-definition-internet.html

8. Conway, L. (2021, July 15). *What is Ethereum 2.0?* The Street Crypto: Bitcoin and cryptocurrency news, advice, analysis and more. Retrieved April 30, 2022, from https://www.thestreet.com/crypto/ethereum/ethereum-2-upgrade-what-you-need-to-know#:~:text=Right%20now%2C%20Ethereum%20can%20only,using%20sharding%20and%20other%20tactics.

9. *Solana: A new architecture for a high performance blockchain*. (n.d.). Retrieved May 1, 2022, from https://solana.com/solana-whitepaper.pdf

10. Jane Williams Researcher at the Centre for Values, & Bridget Haire Postdoctoral Research Fellow. (2022, March 1). *Why some people don't want to take a COVID-19 test*. The Conversation. Retrieved April 30, 2022, from https://theconversation.com/why-some-people-dont-want-to-take-a-covid-19-test-141794

11. *Anchor Framework*. Introduction | ⚓ Anchor. (n.d.). Retrieved May 2, 2022, from https://project-serum.github.io/anchor/getting-started/introduction.html

12. *Accrue finance*. Accrue Finance. (n.d.). Retrieved May 2, 2022, from https://accrue.finance/

13. *The fun, simple, flexible JavaScript test framework*. Mocha. (n.d.). Retrieved May 2, 2022, from https://mochajs.org/

14. *Chai assertion library*. Chai. (n.d.). Retrieved May 2, 2022, from https://www.chaijs.com/

15. *React – a JavaScript library for building user interfaces*. – A JavaScript library for building user interfaces. (n.d.). Retrieved May 2, 2022, from https://reactjs.org/

16. *Hooks at a glance*. React. (n.d.). Retrieved May 2, 2022, from https://reactjs.org/docs/hooks-overview.html

17. *A friendly Solana Wallet built for Defi & NFTs*. Phantom. (n.d.). Retrieved May 2, 2022, from https://phantom.app/

18. *@solana/wallet-adapter*. Organization pages. (n.d.). Retrieved May 2, 2022, from https://solana-labs.github.io/wallet-adapter/

19. *Develop & deploy the best web experiences in record time*. Netlify. (n.d.). Retrieved May 2, 2022, from https://www.netlify.com/

20. *Token program*. Solana Program Library Docs. (n.d.). Retrieved May 2, 2022, from https://spl.solana.com/token

21. Stevens, R. (2022, April 11). *What is tokenomics and why is it important?* CoinDesk Latest Headlines RSS. Retrieved May 2, 2022, from https://www.coindesk.com/learn/what-is-tokenomics-and-why-is-it-important/#:~:text=A%20portmanteau%20of%20%E2%80%9Ctoken%E2%80%9D%20and,like%20what%20utility%20it%20has.

22. Sharma, R. (2022, March 24). *Decentralized finance (DEFI)*. Investopedia. Retrieved May 2, 2022, from https://www.investopedia.com/decentralized-finance-defi-5113835

23. *Anchor protocol vs compound Dai comparison - ANC/CDAI cryptocurrency comparison charts - 1 day*. Walletinvestor.com. (n.d.). Retrieved May 2, 2022, from https://walletinvestor.com/compare/anchor-protocol-vs-compound-dai