澳門理工大學
Universidade Politécnica de Macau
Macao Polytechnic University

Faculty of Applied Sciences
Bachelor of Science in Computing

COMP490 Final Year Project Progress Report

Academic Year 2022/23

Fully On-chain Information Systems on the Internet Computer Protocol

Project number:	29
Student ID:	P1908399
Student Name:	Kevyn Tang
Supervisor:	Dr. Jacky Tang
Assessor:	Dr. Patrick Pang

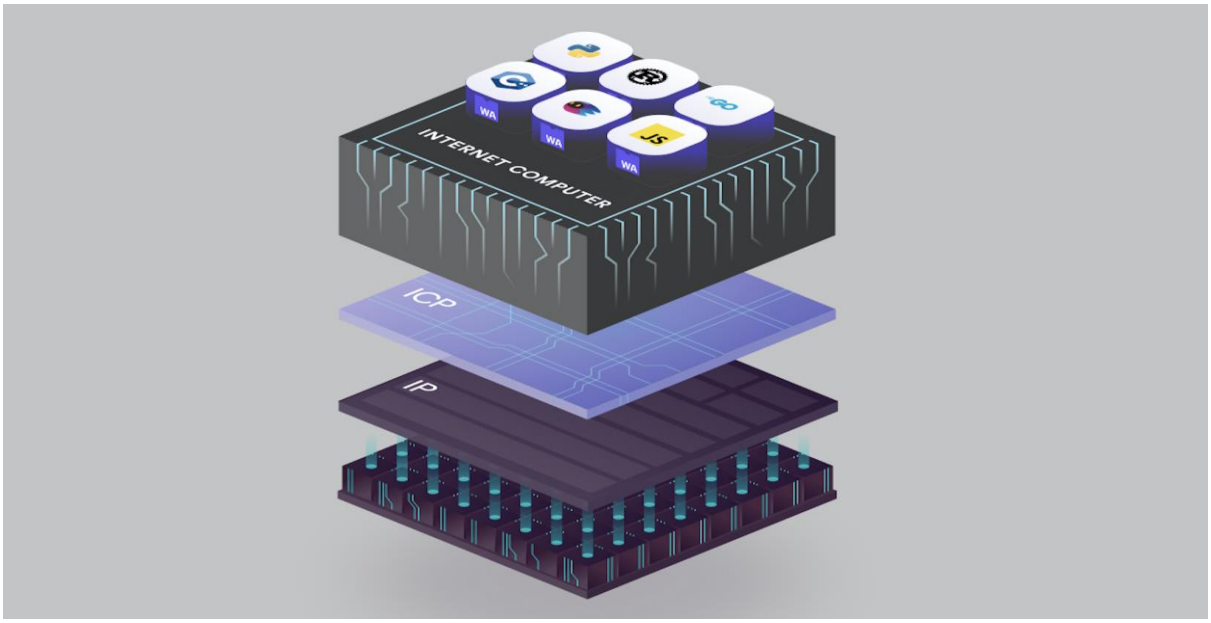
Submission Date:	24 Nov. 2022

Declaration of Originality

I, Kevyn, declare that this report and the work reported herein was composed by and originated entirely from me. This report has not been submitted in any form for another degree or diploma at any university or other institute of tertiary education. Information derived from the published and unpublished work of others has been acknowledged in the text and a list of references is given in the bibliography.

17 Nov., 2022

Abstract



The Internet Computer project from DFINITY foundation provides a limitless blockchain where allows full-stack information systems to be held completely on-chain and run at web speed. As conventional, centralized information systems are originally inflexible in many cases and vulnerable under multiple kinds of adversarial environments, this project explores the feasibility and principles of constructing decentralized information systems on the Internet Computer Protocol, and discusses the related security, efficiency, and performance problems from the engineering perspective.

Table of Contents

Table of Figures

1 Introduction

With the rapid development of modern information technology, various software products have been reshaping every facet of people's life. However, contrary to the original intention of the Internet, the applications on the web have seen more and more monopolized. Large dot-coms take charge of too many resources and centralized server clusters hold too much data. The weaknesses on traditional information systems have always plagued the industry. Many times, the information kept in a database were sold, stolen, or exploited by malicious individuals; the attack on the server caused long service downtimes; and the illegal tampering resulted in serious security issues and huge financial losses. How to reduce the vulnerabilities on the traditional information systems is currently one of the most significant challenges.

After being introduced at 2008 [1], the blockchain technology has been regarding as a big opportunity to bring about a revolution in software development. The potential of running completely on-chain applications was furtherly highlighted after the emergence of the smart contract [2]. The Internet Computer (IC) [3] is defined as the third generation of the blockchain system, where the first generation is Bitcoin [1], and the second generation is Ethereum [4]. Unlike previous blockchain systems, IC aims to be scalable and to run at web speed, one of its main technical components, the Canister, is a special type of smart contract, and it is capable of holding full-stack decentralized web applications.

The IC could be an important infrastructure for information systems in the Web3 context, such as privacy-preserved file storing and sharing, financial applications, social media, and decentralized autonomous organization services. This project explores the feasibility and principles of constructing new generation information systems on the Internet Computer Protocol, and analyses the related security, efficiency, and performance problems from the engineering perspective.

1.1 Objectives

Overall, the project's objective is to explore the decentralized information system design and development on the Internet Computer Protocol in a general track. On the one hand, we use the DFINITY toolchain to develop demonstrating applications, in the scope of performing common tasks in an information system, such as authentication, data input & output, and computations. On the other hand, we analyse and compare the system building on the IC with the ones building traditionally or with other blockchains (e.g., Ethereum), in the security, efficiency, and performance domains. The following list shows the breakdown goals:

- **Background research**:
  in this stage, literature review related to the topic is to be conducted, some analysis on the vulnerabilities of existing information systems is to be performed, and some extant solutions are to be discussed. At the final, better solutions based on the IC blockchain for some existing problems should be figured out on the conceptual level.
- **IC blockchain overview:**
  In this stage, a general overview of the IC blockchain system is to be documented, including reviewing the architecture, protocols and principles. Some overall estimations about the cost and efficiency are to be made, and the usage of the toolchain (e.g., DFX) and the basic components (e.g., the Internet Identity) for the on-chain software are to be figured out.
- **Backend canisters development:**
  This is to develop fully on-chain backend systems in practices and record the process. The Rust development kit (CDK-rs) is to be adopted. Several pre-set themes are to be explored, e.g., privacy-preserved data storage, social media app, and decentralized autonomous organization services.
- **Frontend canisters development**:
  This is to develop fully on-chain frontend system in practices and record the process. The work includes integrating modern web technologies (e.g., Typescript, React, Tailwind) into on-chain app development, and using the DFINITY agent.
- **Exploring Advanced features on the IC:**
  More advanced features provided by the IC (e.g., the HTTP outcalls) should be furtherly explored, and the utility of these features for on-chain information systems is to be discussed.

- **Analysis and comparisons on the final products**:
  Then the analysis of the final products is to be done, on the level of security, efficiency, and performance. The comparisons to other homogeneous applications (e.g., same service with Ethereum) are to be made. Specific conclusions for decision making are to be extracted out.
- **An exploration on future topics**:
  After accomplishing all the above goals, a brief statement of possible future directions and advanced topics are to be made, in order to promote some further results.

1.2 Risk Assessment

The following table summarizes the risks of this project by priority, and the figure below illustrates the impact and possibility of each risk.

Priority	Risks identifications and descriptions
1	**R1: Not Enough Time**. Time management must be conducted carefully. Because research processes are time consuming, an unreasonable schedule will result in the failure of the whole project.
2	**R2: Bias articles may affect project direction.** Solutions may be diversified in the topic's field, it's hard to judge whether a direction is correct or not at the initial stage. Systematic literature review should be done to avoid biases.
3	**R3: Lack of Computing resources or testing environment.** Experimental demo may require lots of calculating power and a specific testing environment.

4	**R4: Data governance policies violation.** Cautious research on Government police is necessary.
5	**R5: Business impracticality.** Some theories may work in the lab but are impractical in a real environment.

Table 1: Table of prioritized risk

High			R1
Medium	R4	R2, R5	
Low		R3	
Probability / Impact	**Low**	**Medium**	**High**

Figure 1: Probability impact matrix before proposed solution

1.3 Summary

This progress report is organized as follow: Chapter 2 introduces the background of the project, and summarizes our related work by conceptually figuring out the shape of fully on-chain information systems. Chapter 3 presents our currently completed work in the scope of real on-chain applications development. Chapter 4 explains the unfinished parts that are going to be done in the next semester.

2 Background and Related Work

This chapter introduces the background and related work of our project.

2.1 Traditional IT Vulnerabilities: The Background

Traditional information systems have developed over decades. From the stand-alone software at the early 21st century to the widely used client-server architecture and browser-server architecture software at the Internet era, we can find that the distribution and operation of contemporary software products is inseparable from a centralized service provider. Whether storing data, performing operations, or retrieving information, we all need a (physically or conceptually) centralized server system to execute the corresponding programs. Such a structure leads to the inherent vulnerability of traditional information systems. For example, a targeted attack on a certain server may lead to large-scale user data leakage and long-term downtime.

Targeted attacks from external malicious individuals can be addressed through a variety of traditional defend methods (e.g., firewall), but those that come from insiders, such as data theft from employees, or even malicious tampering from the system owners, are almost unsolvable. Because if we want to solve these problems, we need to separate the owner of the system from the system itself in some way [5], but with traditional technology, even if the two are separated to some extent (for example, through some kind of governance effort), the server must also have a controller to run stably.

In addition, traditional information systems also have many problems other than security. For example, a single server cluster cannot always provide stable performance and balanced delivery speed, however, building a complex server system that adapts to various situations often requires a large amount of capital and human resource. This also indirectly leads to the monopoly of too much computing power and resources by large companies.

In this context, the blockchain technology may provide us with new ideas to solve the above problems, especially when it is possible to build applications that are fully deployed on the chain.

2.2 The World Computer: A New Concept

In the 2010s, the concept of the world computer [6] emerged in the development of blockchain technology, and it focused on the question: is it possible for us to construct a blockchain that can carry all the computing needed in our human society without the centralized control?[7]

2.2.1 The Definition

A "World Computer" is what the IC is called. This entails offering a blockchain system that can serve the same functions as conventional information systems, host all of humanity's systems, and enable WEB3 applications that is entirely on-chain.

2.2.2 The History

Ethereum was the first blockchain capable of hosting Turing-complete smart contracts, which are secure code units that process and store data on the blockchain itself. In 2014, the startling realization that a blockchain may serve as a computer prompted someone to propose the concept of a "World Computer."

The original aim was that Ethereum would become a "World Computer," however over time, the Ethereum community lost interest in the notion. Clearly, a genuine World Computer blockchain would need to be able to house the majority of the world's computing and data in order to live up to its moniker, whereas Ethereum's design trajectory rendered this impossible.

Now, IC has officially taken up the banner of building a world computer. The unique architecture and rich infrastructure provide unprecedented possibilities for building complex on-chain information systems.

Figure 1: On-chain applications

2.2.3 The Internet Computer Blockchain

The Internet Computer (IC) [3] is a blockchain system of the next generation that offers a comprehensive IT stack on which any online system or WEB3 service may be constructed in a fully decentralized manner. IC is the only blockchain with smart contracts that deliver web experiences, other chain integrations, and interfaces with WEB2 APIs without the need for oracles [8]. The entire system operates thousands of times more efficient than other existing blockchain systems, and is powered only by innovative Chain-key Cryptography.

The Internet Computer Protocol (ICP) operates on nodes, which are servers with standardized hardware. To guarantee fault tolerance and decentralization, nodes are dispersed internationally and housed in separate data hubs. Nodes are divided into subnet replicas, with each subnet constituting its own blockchain that advances independently of the other subnets, resulting in unmatched blockchain performance and scalability. ICP connects and orchestrates all subnets to form the IC.

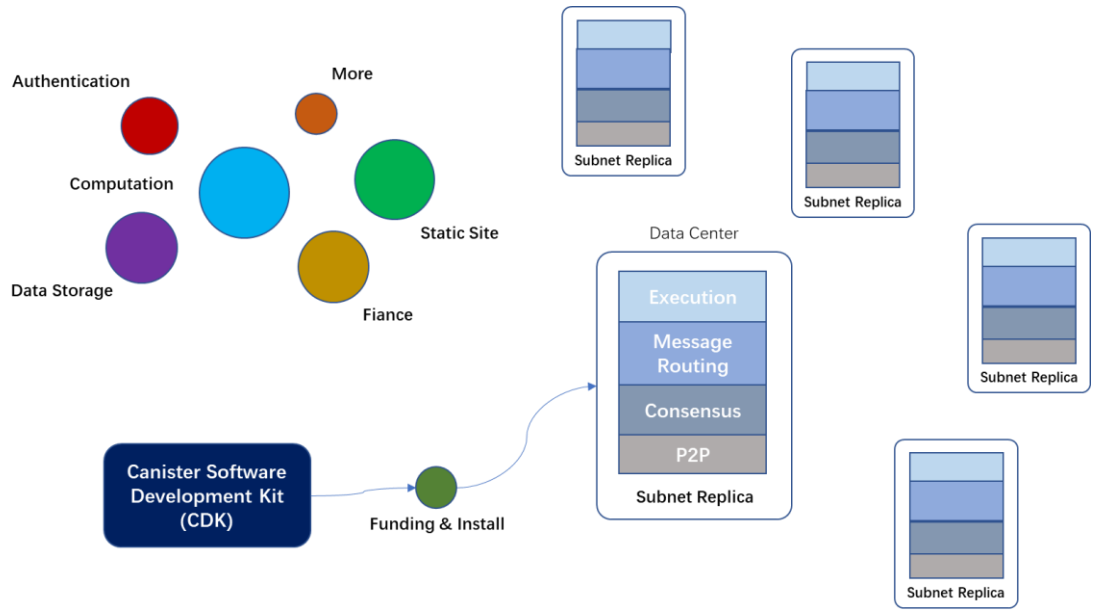


Figure 2: The IC Architecture

The IC improves and advances rapidly as a result of regular and seamless software updates that enhance performance, eliminate errors, and bring whole new features. Continuous hardware updates, the addition of nodes or subnets, make the IC almost limitless in terms of scalability: scaling the IC is always achievable by adding nodes.

2.2.4 The New Form of Information Systems

Canister, an idea proposed by the IC, may give rise to a new type of information system. Canisters are scalable smart contracts — interoperable computing units created for internet-scale services. On the IC, information systems may consist of one or many canisters. Some of the units can execute general computations or data storage duties (back-end), while others may provide browser-accessible web interfaces for end-users (front-end).

An on-chain information system, or a WEB3 service or dapp, can be composed from thousands or even millions of canisters. For instance, a social media service may generate one canister instance per user. Alternatively, a single canister can be a standalone service or self-holding application. Some key characteristics of canisters include:

- **General-purposed**: means that this category of smart contract programs is Turing complete (i.e., anything computable can be computed by a canister).
- **Tamper-resistant**: means that the program's instructions are faithfully executed and intermediate and final results are accurately saved and conveyed.
- **Autonomous**: means that a canister is implemented automatically by the network, without the need for any individual intervention.
- **Composable**: means that they can interact with one another
- **Tokenizable**: means they can utilize and exchange digital tokens.

Importantly, the IC permits a variety of canister mutability policies, ranging from completely immutable to unilaterally upgradable, with additional options in between. Some situations require immutability, such as a financial contract where it is crucial that the code in the canister never changes. In other situations where mutability is required, the container can be modified to better serve consumers or to correct a bug.

Before embarking on further exploration, we studied many features of the IC, such as the optimized proof-of-stake (PoS) model it adopts, and its many underlying protocols and components. Working on the IC, we can imagine such a future information system: it can have scalability from zero, strong anti-attack and tamper-proof capabilities, and no matter how mini the application, it can provide users around the world with balanced and stable performance and speed. We will discuss the details of the construction of such an information system in later chapters.

3 Completed Work

This semester's major accomplishments include the investigation of building on the IC's principles and best practices, the development of several different mini canisters, and the completion of a couple extra community contributions.

3.1 IC Infrastructure Investigation

The development of on-chain applications requires a comprehensive understanding of the entire toolchain and the IC system. In this semester, the infrastructure of the IC was thoroughly investigated.

For programmers, the significant part of the IC infrastructure consists primarily of the DFINITY Command-line Execution Environment (DFX), the Network Nervous System (NNS) [9], and the Internet Identity (II) [10]. All of them are underlying roads and bridges for on-chain software. The subsequent sections introduce them individually.

3.1.1 DFINITY Command-line Execution Environment

The DFINITY Command-line Execution Environment (DFX) is the principal tool for developing, deploying, and administering IC-developed dapps. The "dfx" parent command can be used with flags and subcommands to indicate the operations to run with or without optional parameters. The following diagram illustrates the major part of the DFX tool:

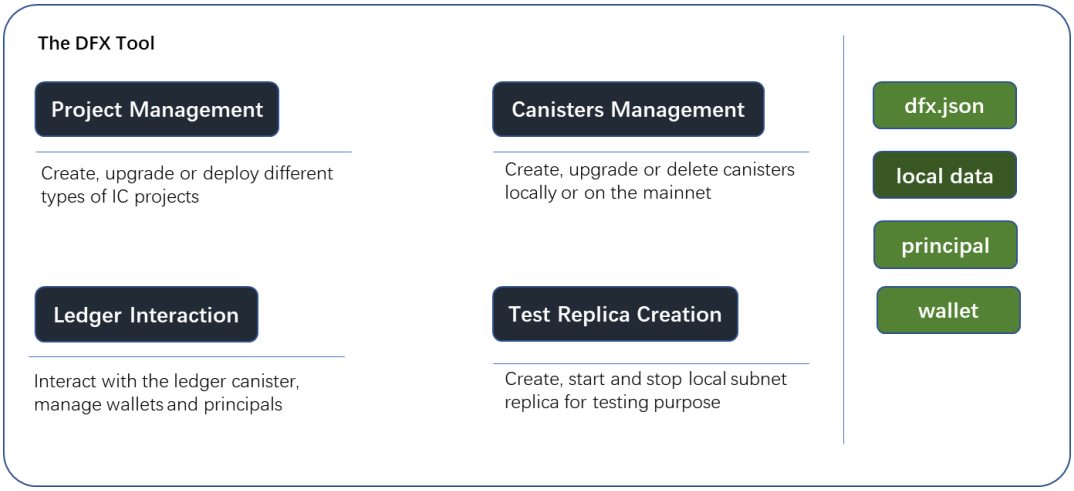


Figure 3: The DFX Tool

3.1.2 Network Nervous System

The Network Nervous System (NNS) is the decentralized truth source for the whole IC structure, holding data about all nodes and their subnet allocations. NNS is majorly consist of three canisters, each for governance, configuration, transactions recording. On NNS, token holders (stakers) can stake ICP tokens in order to vote on any aspect of the IC. It also enables charging computations on the IC by transforming ICP tokens into cycles, which are spent when decentralized applications are executed. According to the foundation, the NNS is the most advanced autonomous decentralized organization in the world.

The NNS regulates numerous facets of the IC blockchain, including:

- The quantity, location, and ownership of the nodes accepted from a data center provider
- The assignment of nodes to subnet blockchains.
- The permission status oof the upgrades to canisters.

The following diagram depicts the different components in the NNS system and the corresponding responsibilities.

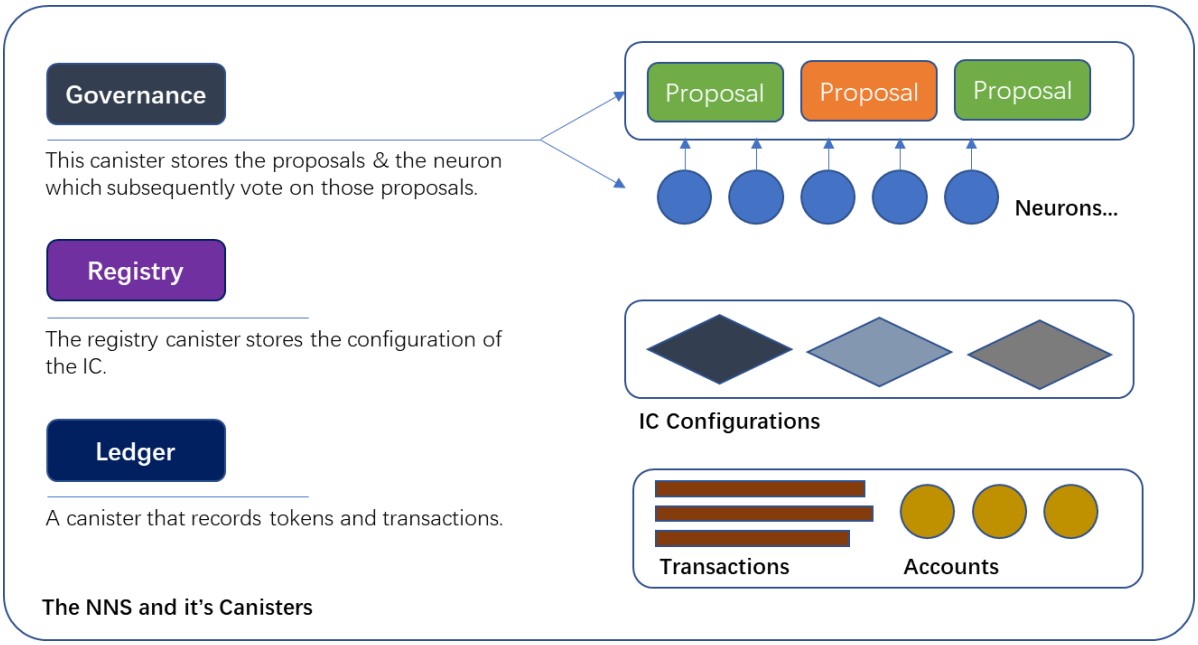


Figure 4: The NNS and it's Canisters

NNS is intended to provide the IC network with open, decentralized, and secure management. It has complete control over all network attributes. It can, for instance, upgrade the protocol and software used by the node machines that host the network; it can create new subnet

blockchains to increase network capacity; it can split subnets to divide their load; it can configure economic parameters that control how much users must pay for compute capacity; and, in extreme circumstances, it can freeze malicious canister smart contract software to protect the network. The NNS accepts and adopts or rejects suggestions based on the voting activity of "neurons" created by network users.

For the governance, participants use neurons to submit new proposals. After submission, proposals are either accepted or rejected, which can occur almost immediately or after a delay, based on the vote of all neurons. Each proposal is an instance of a particular proposal type, which dictates the information it contains. The NNS maintains a system function for each type of proposal, which is invoked whenever a proposal of that type is adopted. When a proposal is adopted by the NNS, it invokes the corresponding system function by filling the parameters with information extracted from the proposal's content. Each type of proposal falls under a particular proposal topic.

Observing how neurons emit votes, the NNS decides whether to accept or reject a proposal. By locking balances of the Internet Computer's native utility token (ICP), which is hosted on a ledger within the NNS, anyone can create a neuron.

3.1.3 Internet Identity

The Internet Identity (II) is an IC-supported framework for on-chain anonymous authentication. Users can build identity "anchors" to which they assign compatible cryptographically equipped devices, such as a laptop's fingerprint sensor, a smartphone's facial ID system, or a portable HSM such as a YubiKey or Ledger wallet. After that, they are able to register and authenticate to any dapp running on the IC using any of the devices they allocated to their anchor. This provides a high level of convenience, allowing users to authenticate to dapps of interest with minimal friction while benefiting from the highest level of cryptographic security, but without the need to directly manage or handle cryptographic key material, thereby preventing errors and the theft of the key material.

When an anchor is used to communicate with a dapp, the dapp sees a uniquely created pseudonym, this prevents users from being tracked across the different dapps they visit. An individual may build as many identity anchors as desired. In contrast to conventional authentication techniques, II does not require users to set and manage passwords or disclose identifying information to dapps or the II canister itself.

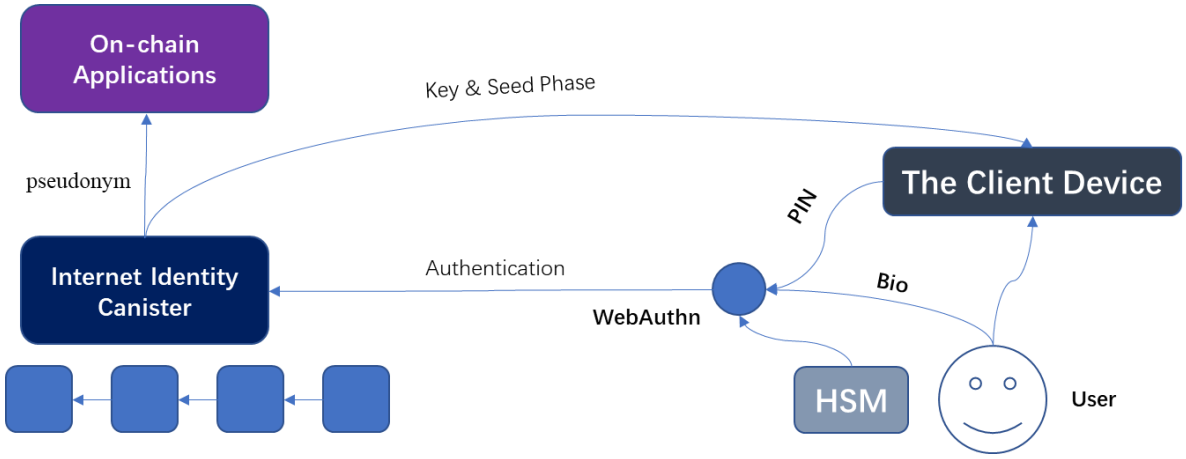


Figure 5: II Authentication Workflow 1

II is based on the Web Authentication (WebAuthn) API, which is supported by modern web browsers and operating systems, and the "chain key cryptography" framework. Client-side code is aware of the master chain key used by the IC to sign the list of public keys contained within each anchor's assigned devices.

Integrating Dapps with II request the user to authenticate using an identity anchor. If a person does not already have an identity anchor, it is straightforward to create one and add authentication mechanisms. A pair of cryptographic keys (private and public) is generated for each device that the user adds. The public key is saved on the IC blockchain, and the private key is secured within the authentication device together with any biometric data that controls access to it. The addition of several authentication devices to an identity anchor enables the user to access decentralized applications on multiple devices.

The user must first specify (or establish) an identity anchor in order to access a dapp that uses Internet Identity for authentication. After authenticating with the identity anchor via a designated device, the browser connects to the II canister and generates a session key for the decentralized application. Finally, the user must grant access to the decentralized application.

After downloading the authorization, the browser sends the user to the distributed application. The dapp checks the authorization from the II and provides the user access as an application-specific pseudonym identity. Internally, the user has a separate pseudonym for each dapp, but the same pseudonym across many devices for any given dapp. All of the gadgets are simply alternative authentication techniques the user can use to verify his or her II anchor.

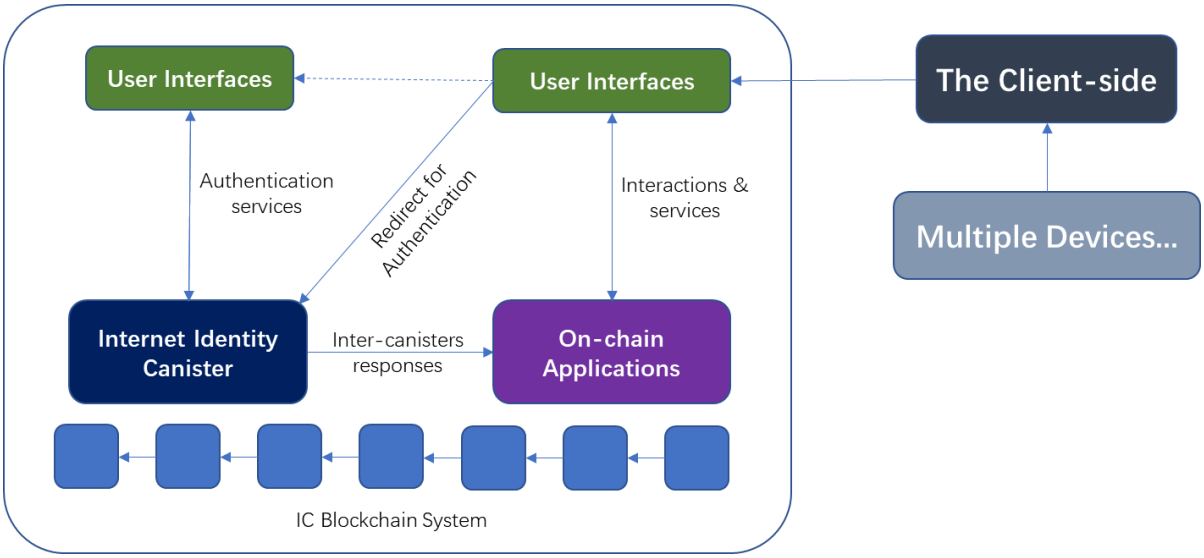


Figure 6: II Authentication Workflow 2

The user may register an unlimited number of identification anchors. A user may, for instance, build one anchor for usage with SocialFi or GameFi and another for pure DeFi [7]. They may only feel safe adding facial recognition to their SocialFi and GameFi anchors, and only employ portable HSM devices such as YubiKeys [11] and Ledger wallets with their pure DeFi anchor.

3.2 Project Structuring and Development Cycle

Application development always begins with the construction of the project. During this semester, we learned the project structure of numerous existing IC projects, such as single-canister projects that only provide back-end services, front-end projects that depend on the DFINITY JS agent, and multi-canister projects that integrate full-end functions.

A working IC project can have various different structures, but we believe that the most efficient project structure in the context of a fully on-chain application should be as follows: a minimal source contains two canisters, one providing back-end functionality and the other providing user interface (front-end) services, important configuration files should be centralized in the root directory of the project. Furthermore, front-end canister is capable of integrating cutting-edge web development technologies, such as webpack, typescript compiler, and popular frameworks such as React and PostCSS.

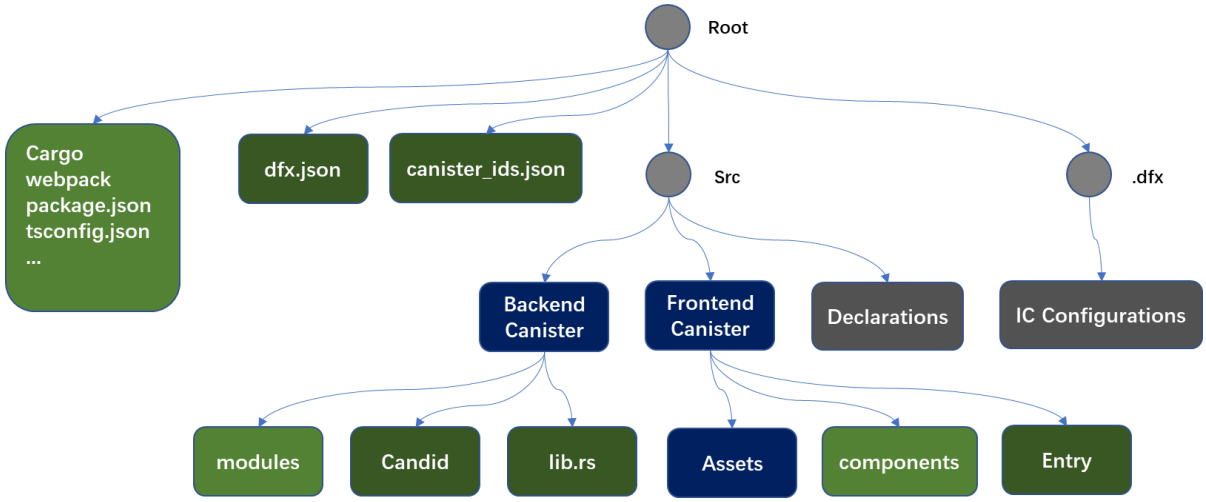


Figure 7: Optimal Project Structure

Under such a project structure, the on-chain information system development process will closely resemble the development of conventional applications. The majority of the steps involved in developing an on-chain application will be the declaration and definition of types and services (interfaces), the creation of back-end data structures and methods, and the implementation of client-side functions and graphic interfaces.

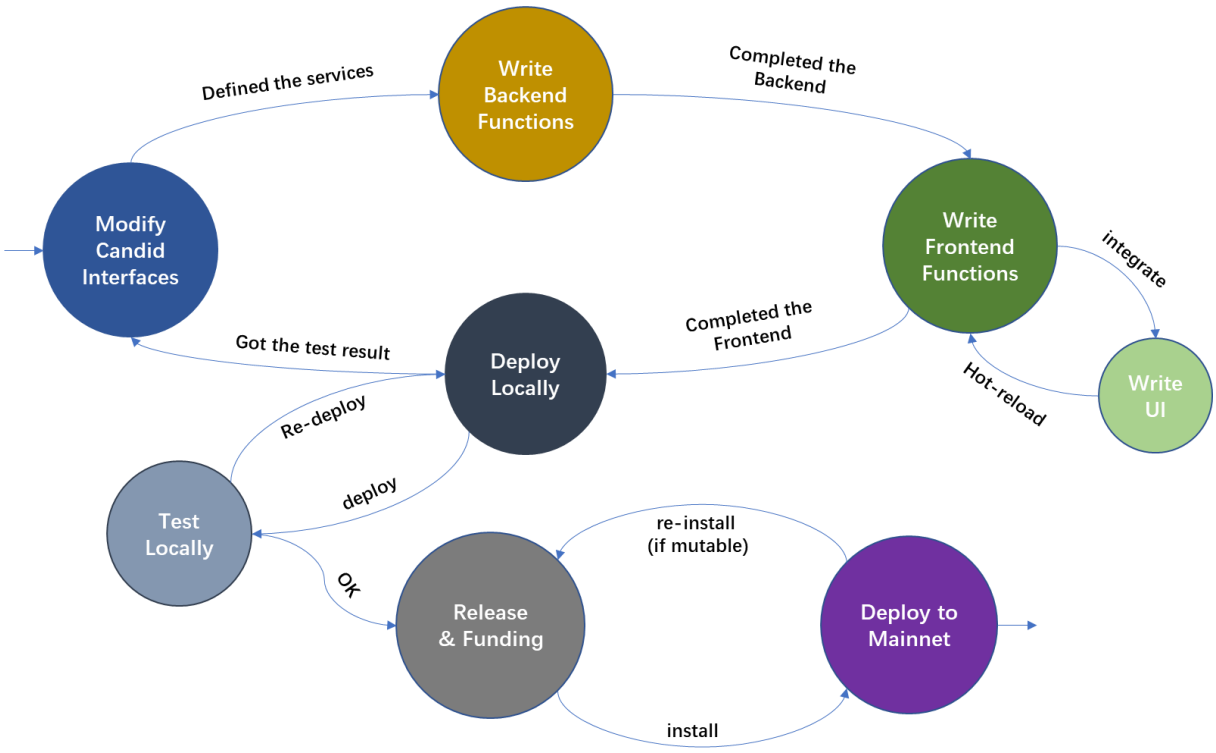


Figure 8: Development Cycle

In addition to the mode of overall backend implementation (which will be discussed in the following chapters), the most significant difference in on-chain project development is the testing and deployment procedure. The test of the on-chain project must be conducted on a local simulated blockchain (subnet replica), and sometimes external canisters must be deployed to the local subnet to complete the test. The canisters should be packaged into a single wasm file and deployed to the IC Mainnet following the functional test (this release and funding process will also be discussed in subsequent chapters).

In conclusion, the first step in this semester's attempt to develop applications on the IC blockchain system is to analyze the project structure and development cycle. In the dual canisters (front-end and back-end integration) structure that we believe to be optimal, the development cycle is comparable to that of conventional applications, with the main differences in the steps occurring during the testing and deployment phases. In later chapters, we will discuss in greater detail the implementation differences of the on-chain services t, as well as the release procedure and cost estimation of running a canister.

3.3 Canisters Construction

Information systems implementation on the IC blockchain can be divided into the design and development of various single canisters. As we mentioned above, canisters are modular, which means that they do not need to be a standalone application, but can instead serve as utility classes for multiple applications. In this semester, we primarily explored the implementation specifics of on-chain applications from the scope of identity authentication, data upload, query, and search, and services for decentralized autonomous organizations. Our work encompassed the most essential facets of the daily operations of a traditional (or new-type) software product.

3.3.1 On-chain Back-end Services

In this semester, we tried the backend canister development with the Rust programming language and the IC Development Kit for Rust (CDK-rs). This tech stack was selected because it provides optimal performance and can well demonstrate the underlying layers of the backend services.

Following the illustrated development cycle, the first step was to define interfaces in the candid file. Candid is an interface description language whose primary aim is to define the public interfaces of a service or software deployed as an IC canister. Candid's support for interoperability between services and frontends developed in multiple programming languages, such as Motoko, Rust, and JavaScript, is one of its primary advantages. The following shows a real candid file we wrote in this semester, in the context of a record I/O system:

```
type Pledge = record {
    "identity": principal;
    "time": text;
    "description": text;
    "display_name": text;
};

type PledgeContent = record {
    "description": text;
    "display_name": text;
};

service : {
    "get_pledge_list": (start: nat64, size: nat64) -> (vec Pledge) query;
    "update": (PledgeContent) -> (Pledge);
}
```

In this candid file, we defined the Pledge and PledgeContent types and two backend services, this file serves as a “glue” that can connect full-ends and shape the interface. After this definition, the next step was to wrote the code.

This concept of a backend canister consisting of both executable code and the canister state. On the IC blockchain, some of the traditional concepts such as database or disk file system are no longer exist. Instead, IC allows developers to use stable memory to store data in a canister, which meets the security and efficiency demands of on-chain applications as well.

Very similar to traditional applications, the most important part of on-chain backend programming is to implement the services (APIs), which means to make the system capable of handling the incoming requests and give out responses. In the context of the above candid interface, we firstly import the CDK-rs libs, define the types in Rust, and setup the memory space using RefCell.

```
use ic_cdk::{
    api::{time, caller},
    export::{
        candid::{CandidType, Deserialize},
        Principal,
    },
};
use ic_cdk_macros::*;
use std::cell::RefCell;

#[derive(Clone, Debug, CandidType, Deserialize)]
struct PledgeContent {
    pub display_name: String,
    pub description: String,
}

#[derive(Clone, Debug, CandidType, Deserialize)]
struct Pledge {
    pub identity: Principal,
    pub time: String,
    pub description: String,
    pub display_name: String,
}

// data structures
type PledgeList = Vec<Pledge>;

// initialize memory pieces using RefCell
thread_local! {
    static PLEDGE_LIST: RefCell<PledgeList> = RefCell::de-
fault();
}
```

This allowed the storage of data in the on-chain stable memory zones, the BTree data structure was adopted to hold a list of pledges (or post, record, anything else). In a production environment, the program is unstoppable and the memory zone can keep the data for any length of time.

In particular, the program at the backend relates to the behavior one should expect when calling an end-point of the canister. For the top-level services, there are only two types of calls: non-committing query calls and committing update calls. The simple difference is whether or not the call changes the canister state. As a developer, it is important to recognize this relationship between the calls that query the data and the calls that update the state. Different IC macros should be adopted to the two different types of the APIs. The following code illustrates a general IC query function that search for a profile and generate a response.

```
#[query(manual_reply = true)]
fn search(text: String) -> ManualReply<Option<Profile>> {
    let text = text.to_lowercase();
    PROFILE_STORE.with(|profile_store| {
        for (_, p) in profile_store.borrow().iter() {
            if  p.name.to_lowercase().contains(&text) ||
                p.description.to_lowercase().contains(&text)
            {
                return ManualReply::one(Some(p));
            }

            for x in p.keywords.iter() {
                if x.to_lowercase() == text {
                    return ManualReply::one(Some(p));
                }
            }
        }
        ManualReply::one(None::<Profile>)
    })
}
```

The major part of the function is about the operation on the memory cell, this is also a distinctive characteristic of on-chain information system development; the stable memory space is now responsible for the storage function that should be conventionally handled by the hard disk or database. The ManuallyReply wrapper can be used to prevent redundant data cloning and directly return the result.

The following code illustrates a general IC update function that receive new profiles and store them into the list.

```rust
#[update]
fn update(profile: Profile) {
    let principal_id = ic_cdk::api::caller();
    ID_STORE.with(|id_store| {
        id_store
            .borrow_mut()
            .insert(profile.name.clone(), principal_id);
    });
    PROFILE_STORE.with(|profile_store| {
        profile_store.borrow_mut().insert(principal_id,
profile);
    });
}
```

For this update call, the pass-in parameter could be created at front-end by JavaScript code, generated by other back-end canisters in Rust or Motoko, or be sent by the command=line tool; the defined candid interface can restrict the input and we wrote the code to insert the instance into the list. This example shows the nature of data updates applied on-chain mainly based on operations on memory cells. Although this method is very different from the traditional information system, we believe that it is feasible to implement storage services similar to conventional database or file system on the chain.

In this semester, we also explored the canister design of the most basic decentralized autonomous organizations, because the work done is also versatile and general in the sense of adding, deleting, modifying, and checking data on memory cells, so we only post the corresponding candid-based service interfaces here as a display.

```
service : (BasicDaoStableStorage) -> {
    // Get the current system params
    get_system_params: () -> (SystemParams);

    // Transfer tokens from the caller's account to another account
    transfer: (TransferArgs) -> (TransferResult);

    // Returns the amount of Tokens the caller owns
    account_balance: () -> (Tokens) query;

    // Lists all accounts
    list_accounts: () -> (vec Account) query;

    // Submit a proposal
    submit_proposal: (ProposalPayload) -> (SubmitProposalResult);

    // Return the proposal with the given ID, if one exists
    get_proposal: (nat64) -> (opt Proposal);

    // Return the list of all proposals
    list_proposals: () -> (vec Proposal);

    // Vote on an open proposal
    vote: (VoteArgs) -> (VoteResult);

    // Update system params. Only callable via proposal execution.
    update_system_params: (UpdateSystemParamsPayload) -> ();
}
```

3.3.2 Agent and Front-end Services

Compared with the back-end services, the development of the front-end canister on the IC blockchain system is not much special -- the work procedure of compiling webpack projects into wasm format products is also common in traditional applications. In this semester, we developed simple experimental UI canisters for the record I/O and DAO canisters we mentioned above, and integrated the React framework and CSS preprocessor in the project while using the jsx file as the core development pattern.

The following pictures show the front-end functionalities and illustrate the two-end data communications.

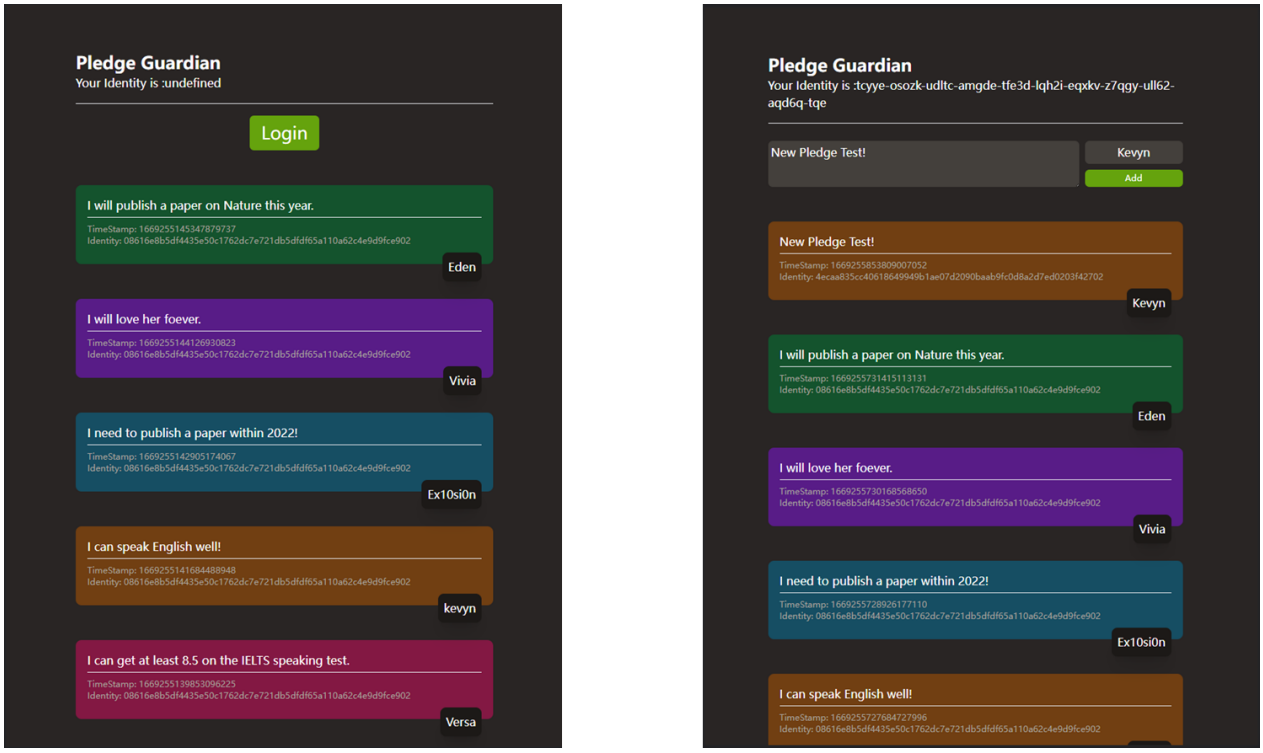


Figure 9: Record IO Front-end Canister

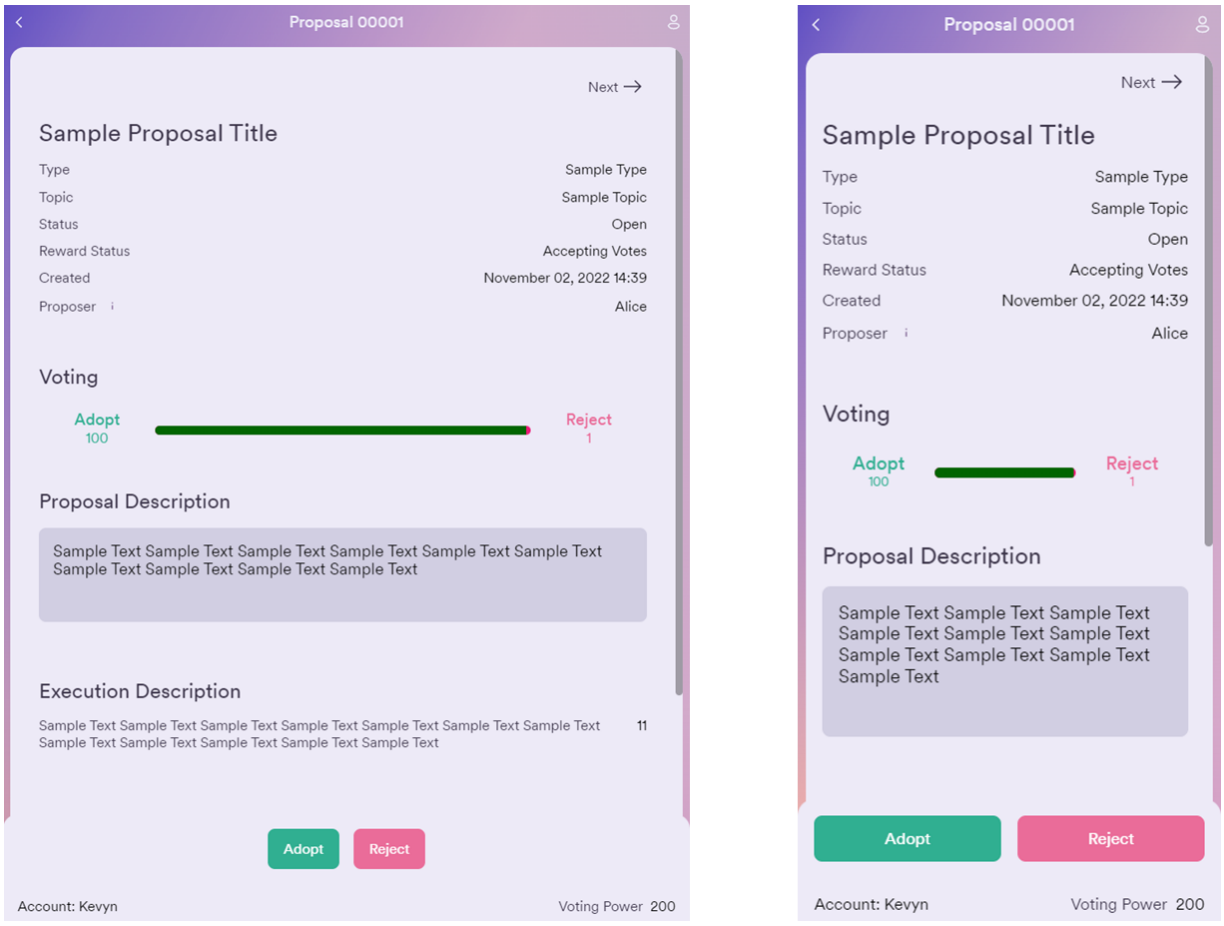


Figure 10: DAO Front-end Canister

One of the unique parts of the front-end canister development is the interaction between canisters using the agents and actors provided by IC. A typical example is to integrate the internet identity authentication service mentioned above in the front-end services. You can see a login button in the above record I/O front-end screenshot, to achieve this, we deployed the internet identity canister to the local subnet replica, and used IS agent and actor proxy for data communication between canisters. The following code and pictures show the process.

```
import React, { Component } from "react"
import { AuthClient } from "@dfinity/auth-client"
import { createRoot } from "react-dom/client"
import './index.css'

import Pledge from "./Pledge"
import InputArea from "./InputArea"
import Login from "./Login"

import { canisterId, createActor } from "../../declara-
tions/pledge_guardian_backend";
import { canisterId as IICanisterId } from "../../declara-
tions/internet_identity";
// ...
async login() {
    await this.authClient.login({
      identityProvider:
`http://127.0.0.1:16383/?canisterId=${ IICanisterId }`,
      onSuccess: async () => {
        this.actor = createActor(canisterId, {
          agentOptions: {
            identity: this.authClient.getIdentity(),
          },
        })
        this.setState({
          isAuthenticated: true,
          principal:
this.authClient.getIdentity().getPrincipal().toString()
        })
      },
    });
  }
```

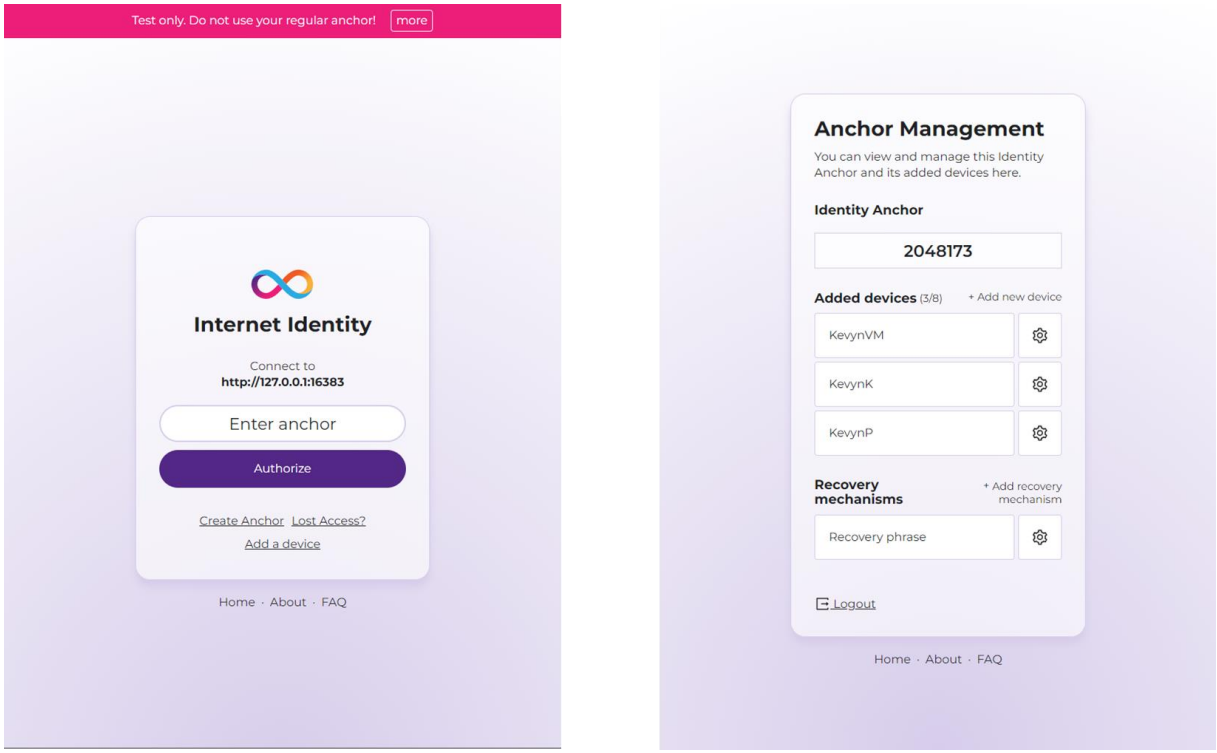


Figure 11: II Integration

3.3.3 Dapp Design

To consolidate our development practices in this semester, we advocated that several design decisions regarding the structure and category of an IC project should be taken prior to the development phase. The following lists the significant ones that developers should pay special attention to.

- **Single or multiple canister architecture:**

  One of the initial decisions to be taken is whether an application should be packaged in a single or multiple canisters. For instance, if we are writing a simple service that has no front-end, we can utilize a single canister to simplify project maintenance and management so that we can concentrate on adding features. If our program contains both front-end components and back-end business logic, it will likely consist of at least two canisters, one for managing user interface elements and the other for the application's back-end operations.

- **Canister reuse:**

In planning, we might also consider placing some common reusable services in their own canister so that they can be imported and called from other more-specialized canisters or made available for other developers to use.

- **Segregating services from types and utilities:**

  Another common practice is to place the code for the main actor in one file with two other separate files for defining the types and utility functions. For example, we might set up the back-end logic for our dapp to consist of the lib.rs with the functions that require an actor to send query and update calls, util.rs with helper functions that can be imported for the actor to use, and types.rs with all of the data type definitions for the dapp.

- **Selecting call types**

  For an IC canister, there are only two types of calls: non-committing query calls (any state change is discarded) and committing update calls (state changes are persisted). As a developer, it is important to recognize this relationship between the calls that query the canister and the calls that change the canister state. Query calls return results faster than update calls. Therefore, explicitly marking a function as a query is an effective strategy for improving application performance.

- **Selecting calls that should go through consensus**:

  Not all queries need to go through consensus. we should also consider the security and performance trade-off that queries don’t go through consensus and do not appear on the blockchain. For some dapps, that trade-off might be appropriate. For example, if we are developing a blog platform, retrieving articles matching a tag probably don’t warrant going through consensus to ensure that a majority of nodes agree on the results. However, retrieving sensitive information—like financial data—might needs more assurance about the results than a basic query provides.

- **Data storage and retrieval method**

  the Internet Computer enables the use of stable memory to handle long-term data storage—often referred to as orthogonal persistence—and to use query calls to retrieve your data. Efficiently retrieving data using one or more keys can typically be achieved by using data structures like hash tables. It is also possible to implement a more traditional database inside a canister.

3.3.4 Release & Funding

After completing the development of the record read and write canister, we try to deploy and fund it on the IC mainnet and calculate the corresponding cost.

Deploying applications on the IC blockchain needs to consume cycles. The cycle is very similar to the gas of Ethereum. Developers can purchase ICP digital currency, and the ICP they hold can be transform into cycles to support the installation and operations of their canisters on-chain, or can also be deposited in NNS to participate in the governance of IC. The following chart depicts the usage of the ICP cryptocurrency.

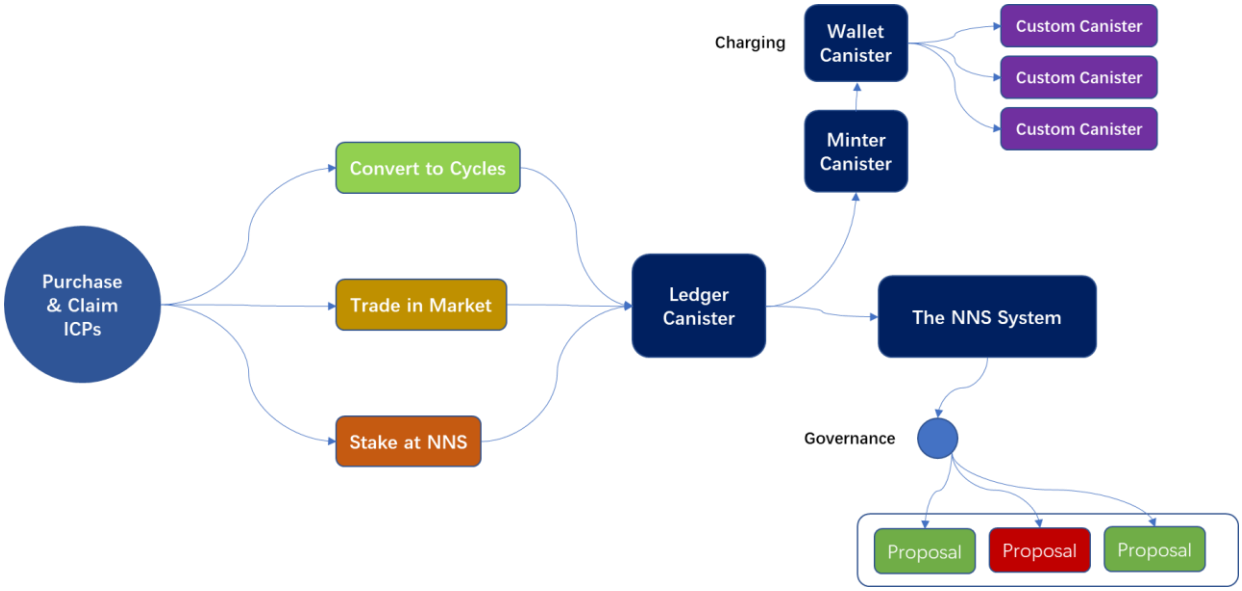


Figure 12: Usage of ICP

We consulted the operators in the IC forum and got the information that, at present, there are two ways to charge the canisters on the IC. One is to directly create new canisters with the developer's principal through the ledger wallet and store in cycles, and the other is to create or manage canisters and the corresponding cycles balance through the canister management system on the NNS app.

Cycles act as fuel for all of the canister's actions. In the deployment attempt, we conducted a rough cost analysis on the number of cycles consumed by different kinds of behaviors of the canisters. The cost data (in Trillion Cycles) for various types of operations were obtained from the DFINITY official website. For mini applications, the creation and installation of canister causes the largest cycle consumption (>=0.1TC), the consumption of update calls and data storage is relatively very small (<0.01 TC), and the consumption of cycles caused by

ordinary query calls is negligible. The following table and graphic record the cost in a sequence of operations, and the theoretical cost of different types of transactions.

Operation	Cost (in ICP)
Init	0.001
Create Canister	0.11
Back-end install	0.011
Queries	0
Front-end Install	0.21
Create Canister	0.1
Update	0
Create Canister	0.11
Queries	0
Update	0.001
Update	0.001
Queries	0
Queries	0
Back-end install	0.01
Queries	0.001

Operation	Cost (in CT)
Canister Creation	100,000,000,000
Per Compute Alloca	10,000,000
Query Execution	1,000
Update Execution	590,000
Inter Canister Call	260,000
Ingress Message Rec	1,200,000
GB Storage Per Secc	127,000

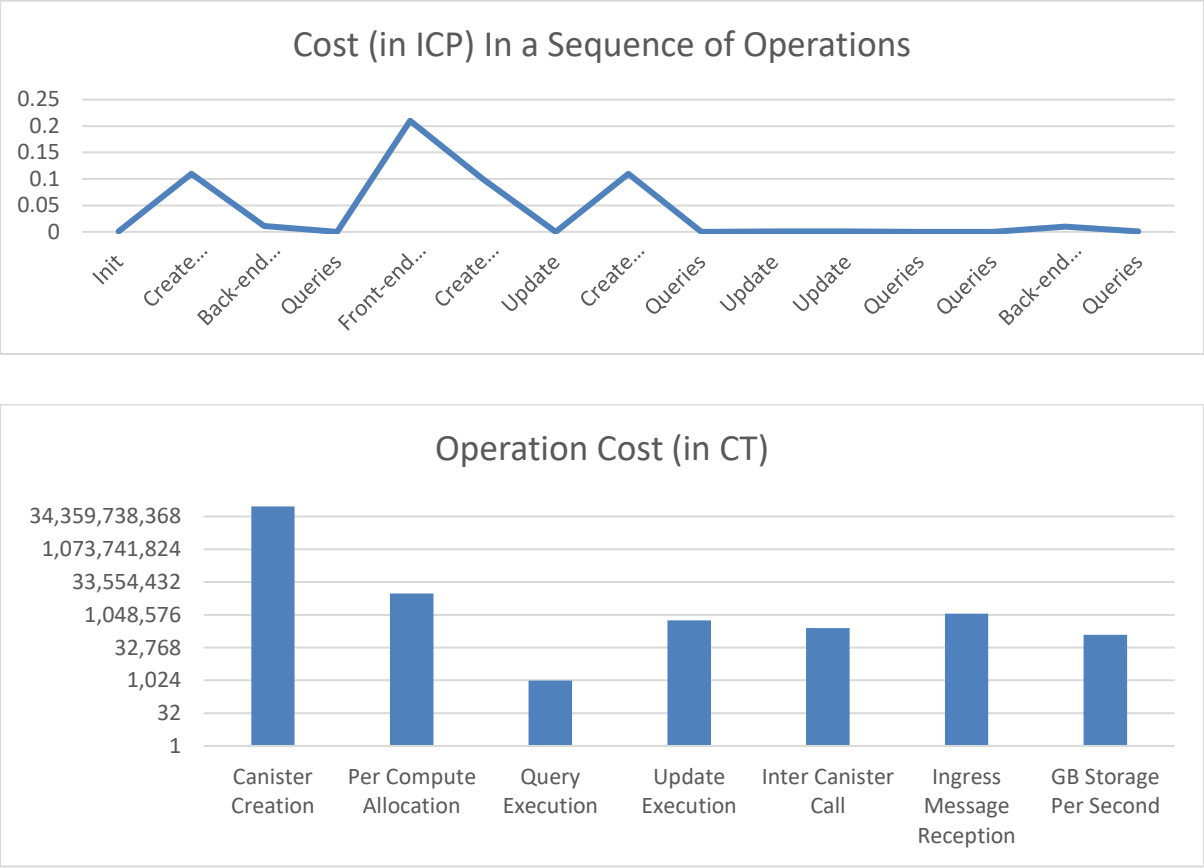


Figure 13: Cost Estimation

In terms of conversion, we currently have:

$$3.9\ USD = 1.0\ ICP = 2.9364\ Trillion\ Cycles$$

Considering the scalability provided by IC, deploying small applications on the chain will be cheaper than deploying to traditional cloud service providers (such as AWS), but the absolute cost of running applications on the IC chain is still higher than using traditional cloud server. Whether it is necessary to exchange higher costs for the advantages of on-chain applications, and how to further reduce costs are issues that we need to explore in the future.

3.4 Community Contribution

After all, during this semester of research and practice, we realized that we cannot understood IC without adequate communication with the DFINITY team and community. Therefore, we deeply participated in the discussions on various topics in IC's forum, discord, and official website community. We communicated with IC's development team under several topics (such as cost, integration, etc.). Also, we opened several issues and contributed dozens of lines of code and modifications to several tutorials to DFINITY's GitHub repository.

Figure 14: Contribution to DFINITY

4 On-going and Future Work

So far, we have completed an in-depth literature review and system analysis. We conducted the study of the entire IC blockchain system, and performed some real development practices. In this progress report, we described the architecture and the corresponding vulnerabilities of traditional information systems, and provided an insight into the architecture adopted by IC blockchain and the new possibility of on-chain information system development.

But there are still many unfinished parts of the current work. One of the major points is that we have not yet developed an innovative on-chain application that are truly cutting-edge and can create value. At the same time, we have not performed a detailed cost estimation and performance comparison of the deployed canisters, and many issues related to security have not been taken into account.

In the next semester, we will focus on the design and implementation of a complicated multi-canister on-chain information system, together with the analysis and optimization of various indicators of the final product. This means that we will first need to come up with an innovative application idea, to allow the corresponding product (in theory) can take full advantage of the on-chain features on IC. We listed some of the possible ideas in the next two sections, and the exact direction of this application will be fully determined by the end of this semester. The following picture shows the project road map for the next semester:

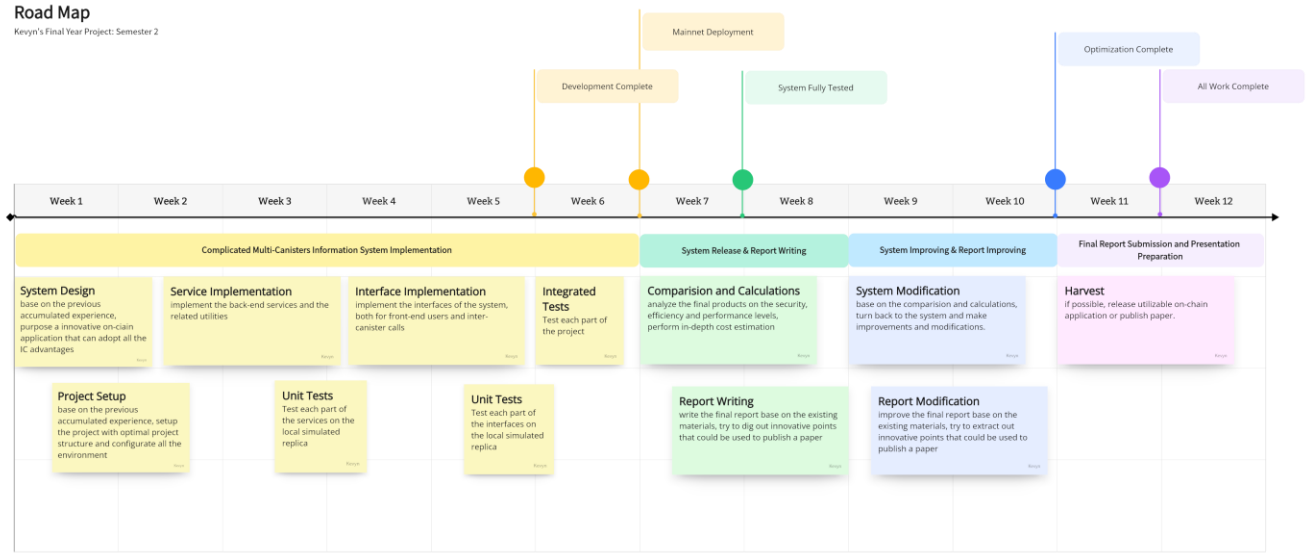


Figure 15: Gantt Chart

We hope that our projects will lead to publishable research results or a novel software product, the next two sections introduce some concrete open questions or topics worthy of advanced study that we discovered during this semester's work.

4.1 Road to Publication

During this semester's research and exploration, we have dug out some interesting questions that may lead to publication opportunities. For example: the research on the improvement of stability and security that may be brought about by applications running upon the IC architecture, the research on a more refined cost estimates and the related financial opportunities. Alternatively, system optimizations and performance improvements may be proposed by further studying the underlying protocols and algorithms of ICs (which are less mentioned in this report). Furthermore, many innovations at the application level can also bring publishing opportunities, such as the optimizations or variants of the DAO system, the improvement of the data storage and transaction systems on the old blockchain, and so on.

What is more certain is that, after gaining an extensive comprehension on the IC, we will undoubtedly focus our research in a more subdivided direction in the future.

4.2 Road to Innovative On-chain Application

Another direction corresponding to academic research is that we can develop on-chain applications with innovative or commercial value, such as: an on-chain mall that is completely paid for with virtual currency, privacy-preserved file storage and sharing systems [12] with high security level, or autonomous social media applications with self-evolving capabilities, and more.

After experimenting with regular and basic development, we will also focus our work on a more subdivided area. One worth-noting thing is that we will certainly pay more attention to products that can possibly take full advantage of the blockchain system and the decentralized feature.

5 Conclusion

The Internet Computer project of the DFINITY foundation offers an infinite blockchain that permits full-stack information systems to be held entirely and efficiently on-chain, which brings the new possibility of designing and building distributed information systems that avoid the weaknesses and drawbacks of traditional software. In this semester, we explored the viability of developing a fully on-chain application by focusing on the most fundamental and general aspects. We analyzed the architecture and current problems of existing, conventional software, delved deeply into the underlying architecture and developer-oriented infrastructure of the IC blockchain system, characterized the core structure and development cycle of the IC project, and consolidated a summary of a wealth of the canisters development practices.

But this report is not exhaustive, and there are still many places that our work has not covered, for example: we have not yet carried out a multi-faceted indicator analysis of the application on the IC as planned, and we have not been able to carry out complex, innovative application development, all of these will become the direction of our efforts in the next semester.

References

[1] S. Nakamoto, "Bitcoin whitepaper," *URL: https://bitcoin. org/bitcoin. pdf-(: 17.07. 2019)*, 2008.

[2] B. K. Mohanta, S. S. Panda, and D. Jena, "An Overview of Smart Contract and Use Cases in Blockchain Technology," in *2018 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, 10-12 July 2018 2018, pp. 1-4, doi: 10.1109/ICCCNT.2018.8494045.

[3] T. D. Team, "The Internet Computer for Geeks," 2022 2022. [Online]. Available: https://eprint.iacr.org/2022/087.

[4] W. Ethereum, "Ethereum Whitepaper," *Ethereum. URL: https://ethereum. org [accessed 2020-07-07]*, 2014.

[5] M. Zichichi, S. Ferretti, G. D'Angelo, and V. Rodríguez-Doncel, "Data governance through a multi-DLT architecture in view of the GDPR," *Cluster Computing*, 2022/08/10 2022, doi: 10.1007/s10586-022-03691-3.

[6] "the Internet Computer wiki." https://wiki.internetcomputer.org/wiki/Internet_Computer_wiki (accessed 2022).

[7] S. M. Werner, D. Perez, L. Gudgeon, A. Klages-Mundt, D. Harz, and W. J. Knottenbelt, "Sok: Decentralized finance (defi)," *arXiv preprint arXiv:2101.08778*, 2021.

[8] "the DFINITY Internet Computer officlal site." https://internetcomputer.org/what-is-the-ic (accessed 2022).

[9] "the DFINITY Internet Computer NNS app." https://nns.ic0.app/ (accessed 2022).

[10] "the Internet Computer officlal site." https://internetcomputer.org/what-is-the-ic (accessed.

[11] D. Merkel, "Yubikey," *Linux Journal*, vol. 2009, no. 177, p. 1, 2009.

[12] P. Sharma, R. Jindal, and M. D. Borah, "Blockchain Technology for Cloud Storage: A Systematic Literature Review," *ACM Comput. Surv.*, vol. 53, no. 4, p. Article 89, 2020, doi: 10.1145/3403954.