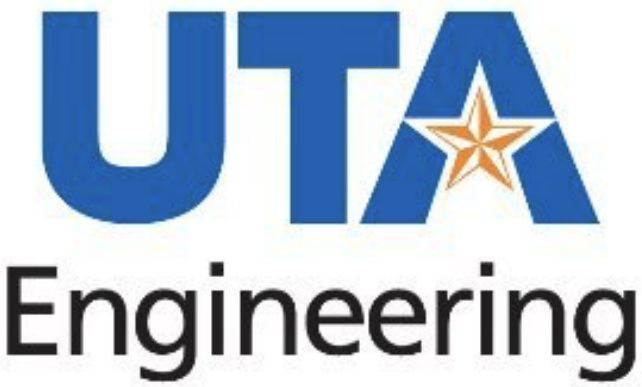
Department of Computer Science & Engineering
The University of Texas at Arlington

Detailed Design Specification
CSE 4317: Senior Design II
Summer 2024



Web3 Enjoyers
Web3id

Edward Alkire
Abhisek Kumar Jha

Revision History

Revision	Date	Author(s)	Description
0.1	7.19.2024	EA	document creation
1.0	7.19.2024	EA	version 1.0 completed

CONTENTS

1 Introduction **5**

2 System Overview **5**

3 Server Layer Subsystems **6**
3.1 Layer Hardware 6
3.2 Layer Software Dependencies 6
3.3 Subsystem 1: Database 6
3.4 Subsystem 2: Account Service 7
3.5 Subsystem 3: Web Controller 8
3.6 Subsystem 4: Authentication 8
3.7 Subsystem 5: Blockchain Indexing 9

4 Client Layer Subsystems **11**
4.1 Layer Operating System 11
4.2 Layer Software Dependencies 11
4.3 Subsystem 1: User Interface 11
4.4 Subsystem 2: Web3 Wallet 11

List of Figures

1 System Architecture 5
2 All DB requests pass through the application server 6
3 Account Service handles business logic, intermediary to user requests and data processing 7
4 Web controller handles http requests and response to user 8
5 Auth. service intermediary between user requests and app 8
6 Example subsystem description diagram 10

List of Tables

1 Introduction

The product concept is to enable sharing "identities". What makes this project worth doing and valuable is the goal of sharing blockchain verifiable data, to make attestations which can be verified from blockchain historical data, where the service enforces validity of such attestations. To make the app valuable to all kinds of users, it is designed to not require understanding of even usage of blockchain, but supports such features for applicable persons.

2 System Overview

The server layer is the bulk of the web application. The client layer is the application logic that runs in the user's browser. The blockchain layer is a relatively minor and optional layer, used where opted into, where it may be used to for verifiable and immutable data.

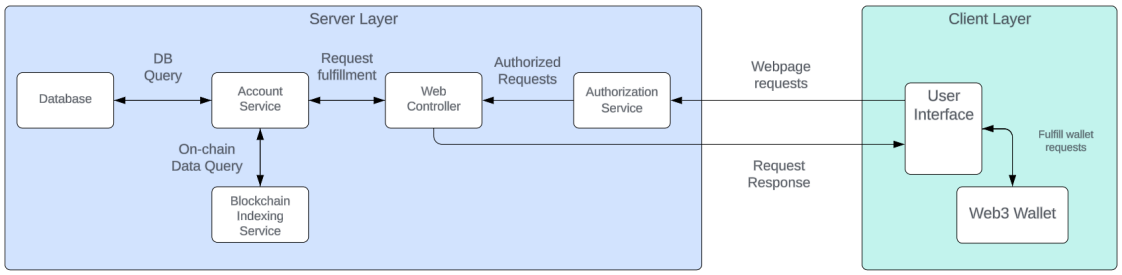


Figure 1: System Architecture

3 Server Layer Subsystems

In this section, the layer is described in terms of the hardware and software design. Specific implementation details, such as hardware components, programming languages, software dependencies, operating systems, etc. should be discussed. Any unnecessary items can be omitted (for example, a pure software module without any specific hardware should not include a hardware subsection). The organization, titles, and content of the sections below can be modified as necessary for the project.

The server is designed using NextJS 14 App Router, which enables writing server side code in JavaScript. NextJS is used for serving web pages to the client, handling requests to the database, business logic, and authentication constraints.

3.1 Layer Hardware

The NextJS server is hosted on a serverless cloud instance. The database is hosted on a separate serverless cloud instance of Postgres.

3.2 Layer Software Dependencies

- prisma: ^5.16.1,
- prisma/client: ^5.17.0,
- simplewebauthn: ^10.0.0,
- iron-session: ^8.0.2,
- next: 14.2.4,
- next-auth: ^4.24.7,
- typescript: <5.5.0

3.3 Subsystem 1: Database

The purpose of the database is to store and maintain relations of data to their user, including authentication credentials and data stored to be presented upon demand in application functionality.

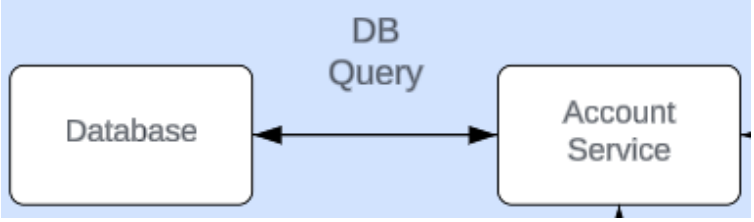


Figure 2: All DB requests pass through the application server

3.3.1 Subsystem Deployment

The database must be deployed to a cloud service provider, behind password protected authentication to preserve privacy of user data. Authentication is required to limit DB requests to only be made by the authenticated application server, where the specifics of authentication and authorization logic is implemented.

3.3.2 Subsystem Software Dependencies

The relevant dependencies to the database:
- prisma: ^5.16.1,
- Postgres 16

3.3.3 Subsystem Programming Languages

Typescript and Prisma schema language

3.3.4 Subsystem Data Structures

The database is structured using Prisma ORM (Object-relational mapping). Each data "entity" or "object" essentially has it's own DB table created by Prisma, as defined in the Prisma schema.

3.3.5 Subsystem Data Processing

Each database query is defined in it's own JavaScript function, and these functions are implemented seamlessly into the application code using "NextJS Server Actions". Under the surface, the NextJS framework automatically implements REST API endpoints for each function.

3.4 Subsystem 2: Account Service

The account service is essentially the business logic of the application. It consists of the application logic in how an authenticated user can interact with their data and the different ways unauthenticated users can interact if at all.

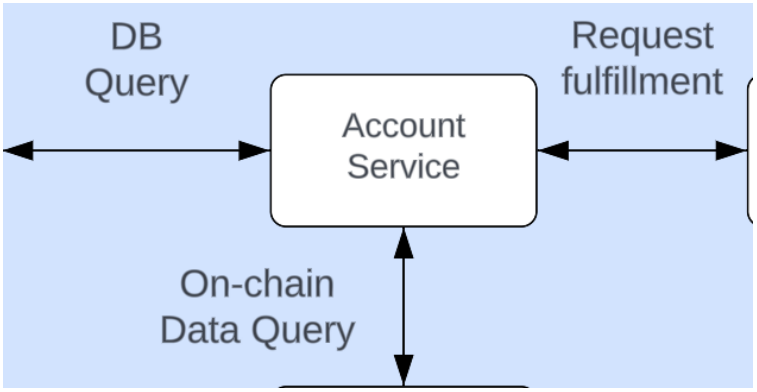


Figure 3: Account Service handles business logic, intermediary to user requests and data processing

3.4.1 Subsystem Software Dependencies

- NextJS 14
- prisma/client: ^5.17.0,

3.4.2 Subsystem Programming Languages

- Typescript
- Prisma Client API

3.4.3 Subsystem Data Structures

The data structures are simply JavaScript functions. NextJS automatically handles client-server communication when functions are implemented in client code. The function parameters are largely related to DB query parameters required for the operation.

3.4.4 Subsystem Data Processing

The account service business logic processes database requests, and makes DB queries using Prisma Client. Prisma Client enables automatically generated Postgres queries, made to work with the defined Prisma schema. Much of the business logic can be thought of as a wrapper around Prisma Client queries for DB data processing, where these queries are initiated by user requests.

3.5 Subsystem 3: Web Controller

The "web controller" subsystem is handled by the NextJS App Router framework. It handles http requests from users and returning responses as applicable. Typically these requests are to load a page of the app for the user to view and interact with.

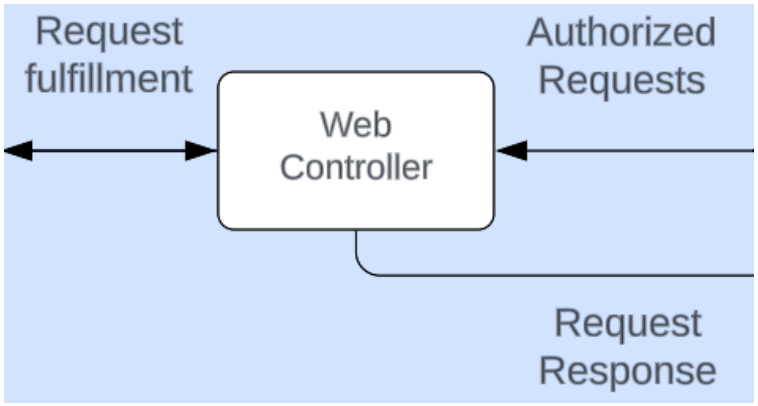


Figure 4: Web controller handles http requests and response to user

3.5.1 Subsystem Software Dependencies

- NextJS 14 App Router

3.5.2 Subsystem Programming Languages

- TypeScript

3.5.3 Subsystem Data Processing

The NextJS App Router framework automatically handles the "web controller" subsystem. NextJS library modules and TypeScript logic for functionality such as conditional rendering or redirecting might also be considered as "web controller" data processing. Web controller data processing is essentially handling http requests from users/clients and returning to them the applicable combination of: html and css code, JavaScript code for browser side user interface operation, or other data.

3.6 Subsystem 4: Authentication

This subsystem handles authentication using WebAuthn (implemented with passkeys), for passwordless authentication. Another part of the subsystem is cookie based session cookies. When a user is authenticated with their passkey, cookies are used so they can stay logged in for a set time, in other words for stateless authentication session management.

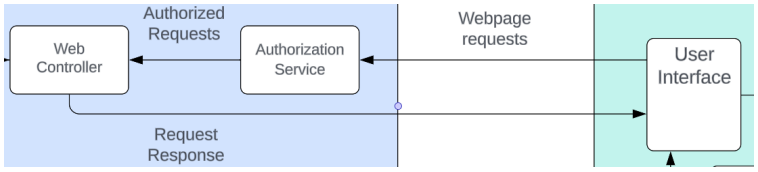


Figure 5: Auth. service intermediary between user requests and app

3.6.1 Subsystem Software Dependencies

- iron-session: ^8.0.2
- simplewebauthn: ^10.0.0,
- User device must support WebAuthn (passkey) authentication
- User browser must enable use of cookies

3.6.2 Subsystem Programming Languages

- TypeScript

3.6.3 Subsystem Data Structures

Cookies have a defined schema, such as the authenticated users id, time till expiry, or other desired metadata defining a user session. They are stored in the user's browser and encrypted for security purposes. The user ID is stored in the database, and is the primary key for retrieving user data. An additional data structure is the database stored schema for user's WebAuthn (passkey) credential, most notably including the public key for authentication purposes, among other less essential fields the WebAuthn standard includes.

3.6.4 Subsystem Data Processing

WebAuthn authentication uses asymmetric key cryptography. The authentication process saves a "challenge" (random string) in a cookie for the user. The user can then provide their biometric authentication, and sign the challenge with their private key; then send this signature to the server, where since the message has been signed by a private key the according public key can be recovered. If the recovered challenge message and public key are as expected, the user can be successfully authenticated. When the user is authenticated, they may stay authenticated with a "session cookie" by encrypting a cookie with session data, storing this in the user browser, where when the user makes future requests to the server, the server can retrieve the cookie from the http request header and decrypt it to identify the authenticated user if said cookie is present and non-expired (note that browsers automatically delete cookies upon expiry. Expiry isn't something that must be typically consciously checked for in implementation).

To give succinct description of cookie authentication: Cookies are transmitted in the header of the user's http requests, for user authentication, to maintain user's "session" in a stateless fashion. The server will have a password for encrypting cookies it generates, and decrypting cookies received in order to read the cookie's secure data contents.

3.7 Subsystem 5: Blockchain Indexing

Querying historical blockchain data is a non-trivial task due to how data is stored in blockchains, thus an indexer must be implemented for this purpose. Indexed historical blockchain data is used for making verifiable attestations of immutable blockchain historical event information.

3.7.1 Subsystem Hardware

Third-party cloud hosted indexing service

3.7.2 Subsystem Software Dependencies

Apollo client is used for client querying of a third party indexer hosted, which has been made queryable through a GraphQL API.
- apollo/client: ^3.10.8

3.7.3 Subsystem Programming Languages

- Typescript - GraphQL API schema and query specification standard

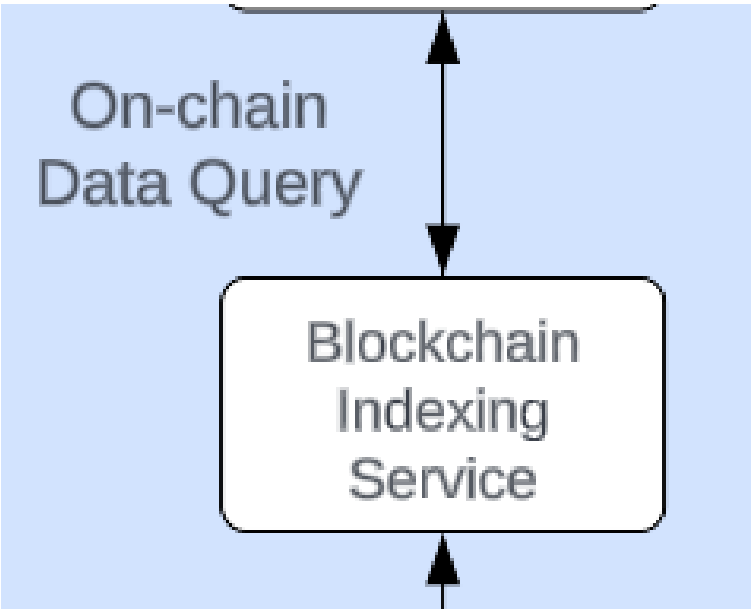


Figure 6: Example subsystem description diagram

3.7.4 Subsystem Data Structures

The major data structure is the GraphQL schema of said indexer. The third party indexer GraphQL schema is programatically inferencable using tools such as studio.apollographql.com. The inferenced schema defines how queries can be structured.

3.7.5 Subsystem Data Processing

This subsystem's data processing is querying third party GraphQL API for indexed blockchain data on-demand.

4 Client Layer Subsystems

The Client Layer is composed of the user interface, made in React, HTML, Tailwind (css), and JavaScript. These serve to enable an intuitive, interactive, and reactive user experience. Also apart of this layer is a Web3 Wallet subsystem for web3 functionality.

4.1 Layer Operating System

This layer requires a modern browser with JavaScript enabled, in-order to display the React front-end and UI. The user may want to install a self-custody Web3 Wallet such as MetaMask in order to use web3 features.

4.2 Layer Software Dependencies

- React 18,
- tailwindcss: ^3.4.4

4.3 Subsystem 1: User Interface

The User Interface is how the user interacts with the application.

4.3.1 Subsystem Software Dependencies

- React 18
- tailwindcss: ^3.4.4
- connectkit: ^1.8.2
- wagmi.sh
- viem.sh

4.3.2 Subsystem Programming Languages

- Typescript, HTML, Tailwind css

4.3.3 Subsystem Data Structures

- React framework enabled reactive state

4.3.4 Subsystem Data Processing

Process user input, return output of user initiated requests.

4.4 Subsystem 2: Web3 Wallet

Web3 Wallets may be used in the app in order to verify ownership of web3 accounts. This subsystem will utilize "connect kit" for easy connecting of web3 wallets to the app. Connectkit enables streamlined implementation, as opposed to having to write implementation for connecting all different web3 wallets manually.

4.4.1 Subsystem Hardware

The user may use "hardware" web3 wallets, as opposed to the more common browser extension based wallets.

4.4.2 Subsystem Software Dependencies

- connectkit: ^1.8.2
- wagmi.sh
- viem.sh

4.4.3 Subsystem Programming Languages

TypeScript

4.4.4 Subsystem Data Structures

Connected web3 accounts are represented by their "address", which is generally derived from the public key of the private key which controls the web3 account.

4.4.5 Subsystem Data Processing

Asymmetric key signature for verification works similarly to the process described in WebAuthn, for the Authentication subsystem of the Server Layer.