CODE OF DUTY

Exploring

Self-Sovereign Identity

Architectural Requirements

Index

Architectural Design Strategy

The SSI System is built upon the following design strategy

a. Structured Design: The main problem was broken down into several smaller problems, and each problem was solved, at specified sprints. The Demo 2 problems consisted of:

   i. User Sign-up: User signs up on SSI mobile app with Biometrics
   ii. User Interaction with Mobile App: User can access and edit Profile Page
   iii. User can view pending and past transactions
   iv. Transaction Contract: Organization initiates creation of credentials when User signs contract
   v. Data Control: User decides what data to show and hide from organization

   This strategy was chosen, due to the need for loose coupling and a level of cohesion that is variable.

   The Demo 3 problems consisted of:

   i. Replacing the database layer with blockchain
   ii. Maintaining all User and Organization operations
   iii. Integrating Smart Contracts to facilitate User and Organization interaction with each other and the blockchain

   The design strategy for Demo 3 remains the same, as the blockchain and interactions were mocked by the database system in Demo 2. Due to the loose coupling, the system was easily replaced by the blockchain, as required by the specifications.

   The Demo 4 problems consisted of:

   i. Creating the interfaces and interaction points for the Organization Client and the Credential Authority
   ii. Creating the MarketPlace service for monetizing user data
   iii. Integrating new Smart Contract for the MarketPlace service

   The design strategy for Demo 4 remains the same, as the blockchain architecture remains the foundation for the new functionality to be built upon.

b. Bottom-Up Design: Smaller components, at lower-level were built first, working upwards towards the final solution.

Architectural Styles

The SSI System makes use of the following architectural styles

a. Structure architectural styles:
   i. Layered Style:
      - Each layer provides services to the layer above it.
      - The system is decomposed into sub-tasks that have varying levels of abstraction.
      - Using the Layered Architectural Pattern

b. Distributed Systems:
   ii. Client-Server Style:
      - The client server pattern is a temporary pattern for our centralized version.
      - The Client-Server Architectural pattern describes the relationship between system users and the API.

c. Adaptive Systems Style:
   iii. Micro-Service Style:
      - Dividing our functionality into smaller sub-services will provide a modular and decoupled implementation, enabling us to easily update the system with minimal changes to other parts.
      - Using the Micro-Service Architectural Pattern

Architectural Quality Requirements

The SSI System has the following quality requirements

a. Security
   i. A core requirement is that of data and identity security. The system must only allow access to sensitive data through use of a decryption key, which can only be accessed via biometric data (thus unique to each user).
   ii. Data transfers over the internet should be secure, thus HTTPS protocol with TLS should be used on Layer 4.

b. Reliability
   i. Since data is linked to biometrics, the chance for data loss is limited to the user losing their biometric identity (nearly impossible).
   iii. The decryption needs to validate successful data decryption to ensure no invalid data is transmitted. Thus a value will be stored and tested each time data is decrypted.

c. Availability
   i. Since having access to your data anywhere and anytime is crucial, the system should have an ideal uptime of between 95%-99%.

d. Scalability
   i. The decentralized version of the app allows high scalability, where its previous centralized state, allowed for at least 20 requests per minute.

e. Portability
   i. The SSI app/system needs to be used interchangeably between different devices for the same user, retaining the same data/functionality. One user with multiple supported devices should have the same functionality on each separate device.
   ii. Since the app/system will be a cloud hosted service, it will be available/usable/installable on devices with internet connection. Thus the app will be 100% available to anyone with internet access and supported devices.

f. Usability
   i. The system gives a user full control of their data and identity. This control should not overwhelm the user, but be easily and efficiently managed.
   ii. The system should be accessible anywhere where the user has access to internet and a smart phone (for biometric authentication).

g. Compatibility
   i. The system/app should not be constricted by use of different devices. Smart phones should be able to interact with different versions/brands as long as both have biometric capabilities.
   ii. Ensure different biometric data from different devices are interchangeable.

Improvements on Quality Requirements for Demo 3:

a. Security: By replacing the database setup in the persistence layer with blockchain, user data is more secure. Blockchain has built-in encryption and does not allow modification of blocks, rather the additon of new blocks for each transaction, thereby preventing unauthorized modification and acces to user data stored on the blockchain. User data is only shared when a transaction is initiated by the user.

b. Reliability: With all previous measures of reliability in place, blockchain storage adds another level of reliability. It acts as a decentralized ledger, providing access to many users, from many points, and keeps track of all data and transactions on the blockchain.

c. Availability: The system should have an uptime between 95% - 99%. Metrics will be provided once all user functionality is full implemented.

d. Scalability: As stated in Demo 2, the decentralized version of the application allows extremely high scalability. The metrics to support this will be provided after all functionality is implemented.

The Portability, Usabaility and Compatability were all optimized

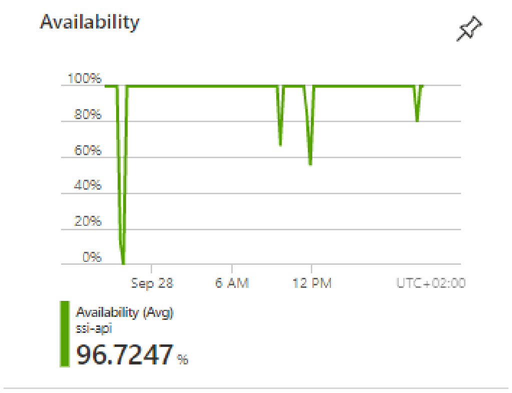
at Demo 2 and exceed all requirements specified.

Improvements on Quality Requirements for Demo 4:

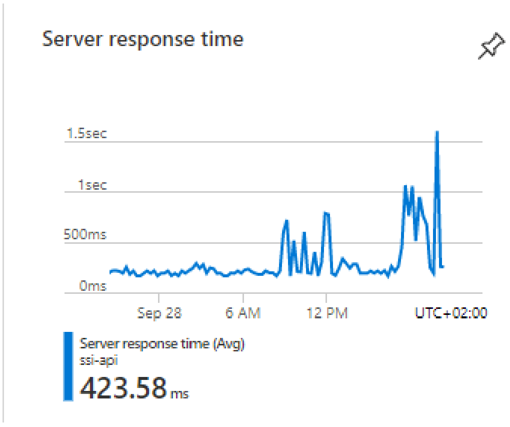
a. Security: By replacing the database setup in the persistence layer with blockchain, user data is more secure. Blockchain has built-in encryption and does not allow modification of blocks, rather the additon of new blocks for each transaction, thereby preventing unauthorized modification and acces to user data stored on the blockchain. User data is only shared when a transaction is initiated by the user. Encryption is used on user data and secrets, to ensure maximum security.

b. Reliability: With all previous measures of reliability in place, blockchain storage adds another level of reliability. It acts as a decentralized ledger, providing access to many users, from many points, and keeps track of all data and transactions on the blockchain.

c. Availability: The system has an uptime between 95% - 99%.



d. Scalability: As stated in Demo 2, the decentralized version of the application allows extremely high scalability, with an average server response time of 423.58ms.

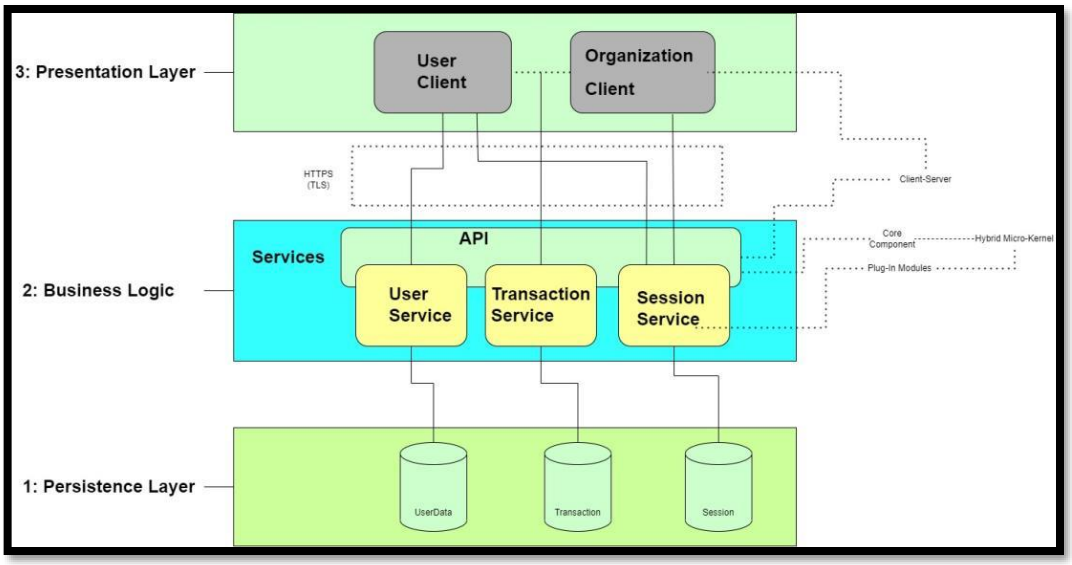


The Portability, Usabaility and Compatability were all optimizedat Demo 2 and exceed all requirements specified.

Architectural Design and Pattern

Detailed explanation of the SSI system Architectural Design and Patterns

Initial Architecture



a. Layered Architectural Pattern

- 3 Layers in our structure:

  3: Presentation Layer: Contains the UI for end users.

  2: Service Layer: Provides the UI layer services to access the persistence layer.

  1: Persistence Layer: Provides data access for the service layer.

b. Client-Server Architectural Pattern

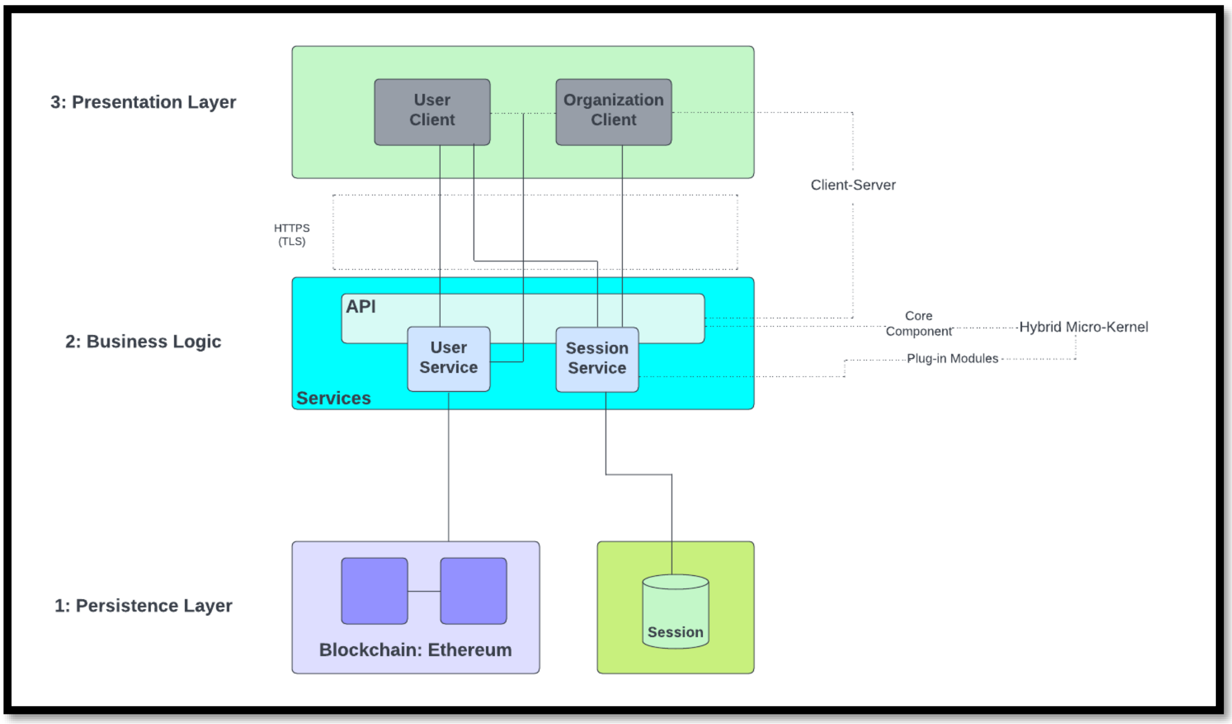
- Multiple clients are able to connect and use the provided services through our API.
- The layer 4 protocol that will be used is HTTPS, whereby the security protocol included will be TLS.

c. Hybrid Micro-Kernel Architectural Pattern

- Combination of monolithic API structure with Micro-Kernel pattern to ensure agility and easy translation to a Decentralized architecture for future implementation (Micro-services etc.)

  - 3 Plug-In Modules thusfar include
    - User
      - Provides services for all data and processes involving the User Database.
    - Transaction
      - Provides services for all data and processes involving the User Database.
    - Session
      - Provides the services to initialize and store sessions.

Intermediate Architecture



a. Layered Architectural Pattern

- 3 Layers in our structure:

  3: Presentation Layer: Contains the UI for end users.

  2: Service Layer: Provides the UI layer services to access the persistence layer.

  1: Persistence Layer: Provides data access for the service layer.

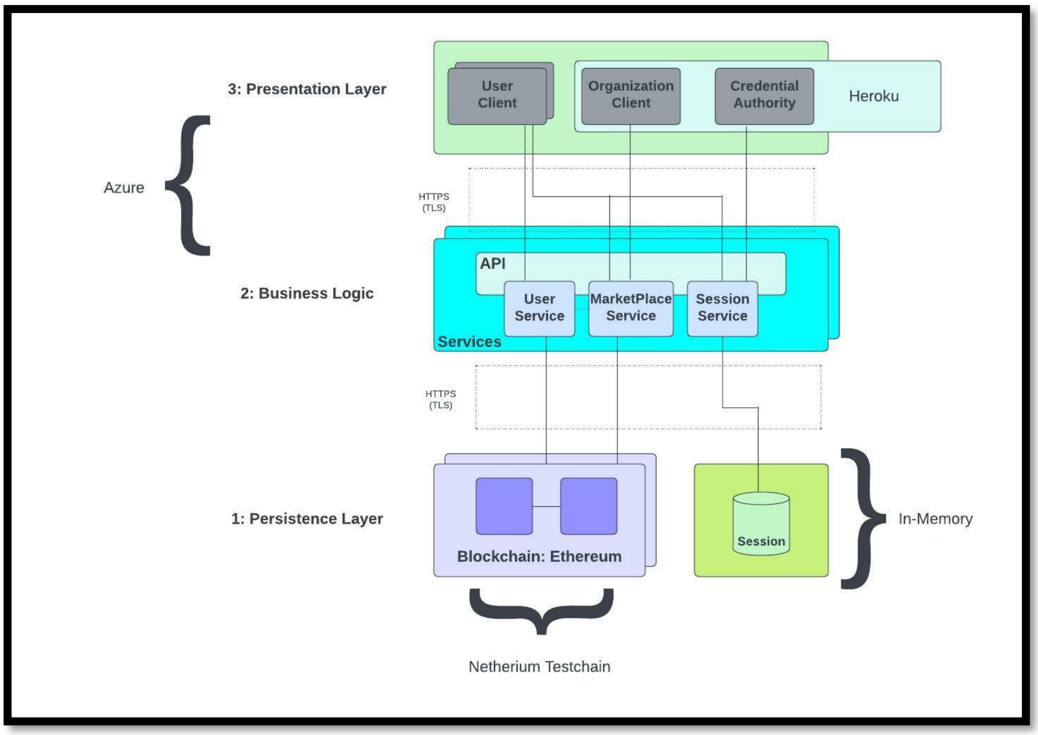
b. Client-Server Architectural Pattern

- The User Service translates to the Smart Contract, by which user transactions are made and interact with the blockchain.
- The layer 4 protocol that will be used is HTTPS, whereby the security protocol included will be TLS.
- The API is maintained as a security measure, to prevent direct access from the User to the Blockchain, via the Smart Contract.

c. Hybrid Micro-Kernel Architectural Pattern

- 3 Plug-In Modules thusfar include
  - User
    - Provides services for all data and processes involving the User access to the blockchain.
  - Session
    - Provides the services to initialize and store sessions. Still connected to a centralized database. To be moved to storage on the blockchain after Demo 3.

Final Architecture



d. Layered Architectural Pattern

- 3 Layers in our structure:

  3: Presentation Layer: Contains the UI for end users.

  2: Service Layer: Provides the UI layer services to access the persistence layer.

  1: Persistence Layer: Provides data access for the service layer.

e. Client-Server Architectural Pattern

- The User Service translates to the Smart Contract, by which user transactions are made and interact with the blockchain and MarketPlace Service translates to the Smart Contract, by which MarketPlace transactions are made and interact with the blockchain.
- The layer 4 protocol that will be used is HTTPS, whereby the security protocol included will be TLS.
- The API is maintained as a security measure, to prevent direct access from the User to the Blockchain, via the Smart Contracts.

f. Hybrid Micro-Kernel Architectural Pattern

- 3 Plug-In Modules thusfar include
  - User
    - Provides services for all data and processes involving the User access to the blockchain.
  - Session
    - Provides the services to initialize and store sessions. Still connected to a centralized database.
  - MarketPlace
    - Provides services for all data and processes involving the User transactions with the MarketPlace on the blockchain.

Architectural Constraints

The SSI System has the following architectural constraints

g. Microsoft Technology Stack recommended.
h. Hosting should be done with Azure.
i. .NET6 and C#.
j. Any frontend framework for Javascript/Typescript, Vue.js recommended.

Technology Choices

The SSI System technology stack was defined in order to solve the problems, conform to the constraints and maximise mentoring from Derivco

a. Vue.js:

Vue.js is a progressive, light-weight frontend framework for building versatile and reactive apps.

Vue has replaced our original framework, Angular, since we needed a smaller, faster frontend framework that our mentors are also familiar with. Angular has much more features, however this is considered too 'bulky' for what we are trying to build and the light weight Vue was thus decided on.

Pros:

- Light weight
- Versatile
- Reactive
- Proper mentor support

Cons:

- Most team members are unfamiliar with it
- Takes a bit of time to learn and develop features.

Vue.js fits our needs since it has everything we need to develop a mobile app and also create a desktop version in the future when needed.

It also effortlessly fits into our N-Layered and Client-Server architecture, since we have designed it to be loosely coupled, ensuring efficient integration amongst the layers. Vue is found on Layer 3: Presentation Layer of the architectural diagram and has the role of a client in the Client-Server pattern.

b. .NET6 and C#:

.NET is a developer platform supporting various technologies, languages, libraries etc. for developing apps ranging from mobile- to web-apps. C# is a general service, multi-paradigm programming language. C# has excellent support in .NET6 and both are currently being used. It also allows us to use the Nethereum Framework for decentralization, by using the Ethereum blockchain.

Pros:

- Vast resources and examples.
- Wide variety of libraries, support for multiple technologies and databases.
- Conforms with Microsoft Technology Stack constraint.
- Mentors are able to assist with technical issues.
- Nethereum Framework for blockchain
- Object-Oriented

Cons:

- Most of the team is unfamiliar with both .NET and C#.
- C# does not have the best performance.

.NET6 with C# covers Layer 2: Service Layer, and provides access to and centralization of the databases in the Persistence Layer (for Demo 2). Both are powerful and efficient technologies that enable us to access storage and act as a server, fulfilling the role of the Server in our Client-Server pattern.

c. Storage:

Demo 2:

Our main goal is to have a decentralized app, however at the moment to show our functionality before implementing blockchains, we have a centralized version.

MSSQL has been chosen for our databases. It is a relational database management system and conforms to the Microsoft Technology Stack constraint. As of now, it is used to mimic the blockchain that we will eventually implement.

Demo 3 & 4:

The Ethereum Blockchain is used in order to achieve decentralization. The Ethereum blockchain has built-in support for smart contracts, thus transaction handling is very sophisticated. Ethereum is well-known and secure, thus it serves all requirements we need for a decentralized ledger.

Pros:

- Decentralized
- Secure
- Support for smart contracts
- Easily integrated with .NET6

Cons:

- Difficult to learn and implement
- Vast amount of research required

Ethereum (and MSSQL) are located on the 1st layer; Persistence Layer in our N-tier pattern. It has contributed to the decentralization of our application.