Course: ECS 265, 4 Units, Fall 2021
Members: Rohan Sood, Utkarsh Drolia, Alejandro Armas, Micheal Yang, Kavyasree Kollipara
Title: Mid Term Progress Report for ECS 265 [Blockchain-as-a-service]

1. Task Breakdown

Our project can be broken down into 3 major parts. The frontend, backend and ResilientDB.

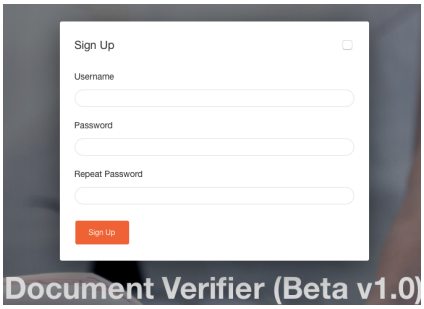
1.1. Front End

1.1.1. Components development

1.1.1.1. LandingPage component

Figure 1. Landing Page

This is the page that the users will land on when they navigate to our application. They will have the option to login or sign up. Following are login and sign up modals.



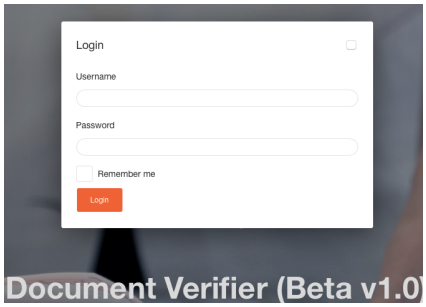


Figure 2. Sign Up (left) and Login (right) Modals

1.1.1.2. HomePage component

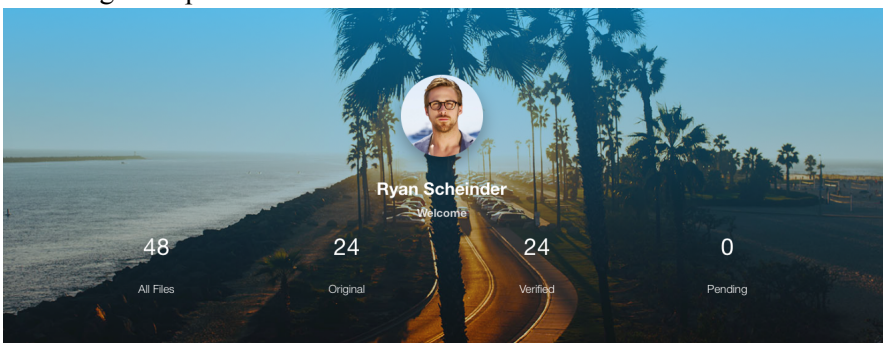


Figure 3. Home Page

This is the page the users will be navigated to once they've logged in. They will be able to see a count of their documents under various headings, and they can navigate to each of the tabs to see document lists under that subheading.

1.1.1.3. UploadPage component

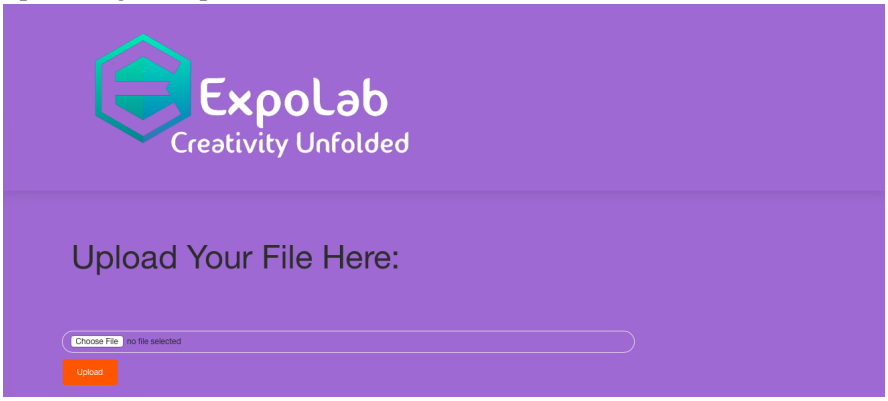


Figure 4. Upload Page

The users can navigate to this page via the homepage and can use this page to upload the documents as well as add other users needed to approve the document before it can be stored on the blockchain.

1.1.1.4. DocumentList component

The component to list the different documents under a particular subheading and give details such as date created, last edited, uploaded by, etc as well as action options to edit, approve, reject etc.

1.1.1.5. Main components

Different UI components used across the application, such as buttons, navigation, forms, progress bars, etc.

1.1.2. State management

1.1.2.1. Authentication state

To maintain the session for the user and authenticate each transaction using the token associated with that user.

1.1.2.2. Document state

To maintain the state of the document, whether it is pending approval, approved, rejected or pending transaction hash from blockchain.

1.1.3. Connect to backend server

Connect the frontend to the backend by calling the backend API's for different actions such as authenticating users, exchanging data and carrying out transactions.

1.1.4. Risks

1.1.4.1. This is a Prototype, which may have security concerns related to user registrations. At this stage, we will only use test data for safety reasons and will not open it to public access.

1.1.4.2. May not contain all demanding features. More features will be added on an ongoing basis.

1.1.4.3. May face copyrights issues for future production due to the use of unauthorized UI templates.

1.2. Back End

1.2.1. Components

1.2.1.1. Implemented REST APIs using Express JS Framework for user functionalities and some document functionalities.

1.2.1.2. User registration with details such as user email, first name, last name, and password.For Login, we used JSON Web tokens for user Authentication.

1.2.1.3. For document upload API, the user should include attributes such as the document, associated users, and the location of the file.

1.2.1.4. Edit the document to replace the whole document.

1.2.1.5. Edit data of the document.

1.2.1.6. Some REST APIs to list the users, search a user.

1.2.1.7. Integrated MongoDB for storing user data, document data, and transactional data.

1.2.1.8. Integrate Resilient DB to store and verify the document hash.

1.2.1.9. REST APIs for user transactions such as approve and reject.

1.2.1.10. REST API to list pending requests of a specific user.

1.2.2. Risks

1.2.2.1. Integration of Document Verification app and ResilientDB may lead to disruption of critical operational processes.

1.2.2.2. API security is not considered at this point of time. It can contribute to potential security problems.

1.2.2.3. Vulnerability of NoSQL Injection.

1.3. ResilientDB Service

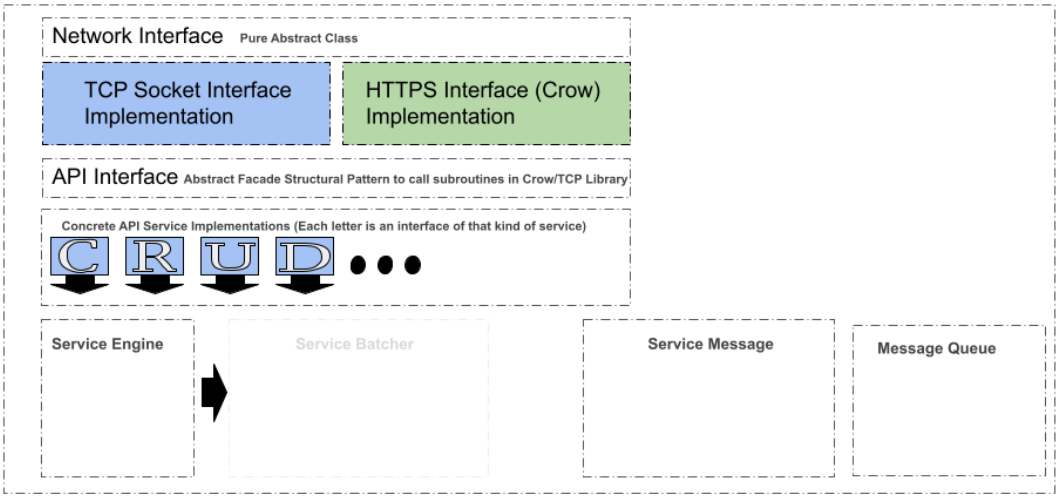


Figure 5. Client Aggregator Architecture

1.3.1. Client Aggregator

Node used to aggregate API service requests from client applications. This aggregator is functionally the 'Client' in the PBFT consensus protocol model. The motivation for re-architecting this entity is to provide a modular interface to support a wide array of decentralized applications as well as utility programs through networked connections (either TCP or HTTP Rest). For example, a target user would be the researcher concerned with transaction performance. A command line program would create a socket connection and send data. Additionally, this architecture enables us to offload the foreable compute associated with caching data from reads, updating this cache, batching transactions for performant consensus throughput, service prioritization, user authentication and more.

1.3.2. Service Engine

The complement to the Client Aggregator is the actual engine, layered on top of the replica nodes, which follows a message passing paradigm with dynamic dispatch for control flow. Replicas perform consensus on 'Service' messages and once they achieve a committed state, invoke routines within themselves; such as Reading and writing to blockchain before notifying the client aggregator of their result.

1.3.3. Risks

We've taken risks to unit test our additions with the catch2 framework because we've felt that it's something the resilientDB platform has been missing.

2. Timeline

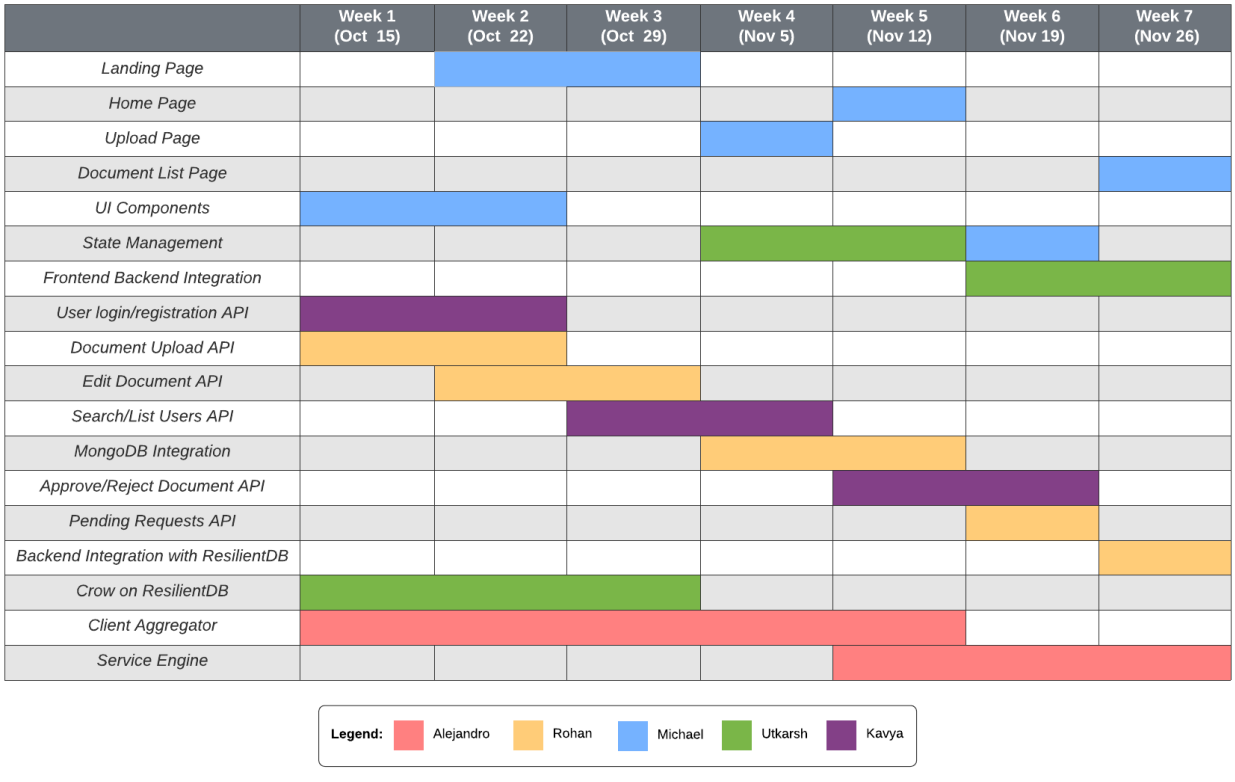


Figure 6. Gantt Chart

3. The Company and Team.

Rohan Sood: Responsible for organizing the whole team, along with architecting the solution in a modular way. Creating the backend APIs for the application on Node.js/Express and integrating them with the React frontend. Maintaining the database for the application on MongoDB is also one of the tasks taken up. Will also be helping on the front end development to expedite the whole process.

Alejandro Armas: Is the lead architect for the engine servicing API endpoints. Furthermore he is leading the design to extend the notion of transactions on ResilientDB, so that replicas may perform consensus on services.

Utkarsh Drolia: Responsible for developing the microservice on ResilientDB that will communicate with the backend for external applications and talk to ResilientDB to perform transactions, get transaction history, etc. The microservice will be built using the Crow framework. Also responsible for integrating the frontend and backend by using the API's created on the backend for authentication and data exchanges.

Michael Yang: Responsible for the front-end development of the web application. The application consists of three major parts: The user interface developed in React, the backend server developed in Express using RESTful API, and a communication channel between the backend and ResilientDB. Specifically, Michael is responsible for designing a user interface that allows original document owners to upload their documents that contain information that is sensitive to changes. It will also allow other users who intend to verify if their document is identical to the original copy to upload their documents for verification. The users will receive feedback and prompt from the app regarding their uploading and verification status.

Kavyasree Kollipara: Responsible for developing the backend using Express framework and integrating REST APIs of ResilientDB to the web application.The purpose is to connect the UI and REST APIs of ResilientDB. The backend is to implement the verification of the document by communicating the state with ResilientDB and reporting it to the user.