

Architecture & Design Assignment

System Overview

Tenants and landlords normally use a traditional lease agreement to protect their interests during the lease and use third-party payment instruments for the payment transaction process. The issues with this approach are that it relies on the tenant to remember to make timely and correct payments to the landlord and it is not able to completely store the transaction records generated during the lease period. Therefore, it is easy for both parties to generate inevitable disputes that must be solved through a third party. Blockchain technology can solve the problem of centralized access control and not being able to systematically manage the leasing process. Blockchain uses peer-to-peer technology, which makes it possible to implement decentralized applications. This is used to ensure that the stored block information cannot be tampered with.

Our solution is an Ethereum smart contract for rental contracts with a decentralized application (dApp) front-end that the user will interact with. By utilizing blockchain technology, specifically an Ethereum-based application, we are alleviating the tedious and complicated process of paying rent to a landlord. Traditionally, landlords must rely on the tenant to make the monthly payment, whereas a smart contract is binded by the code that is written. Additionally, there are numerous benefits of smart contracts including autonomy, security, not needing to trust third parties, cost-effectiveness by removing intermediaries, fast performance, accuracy, fairness, and visibility. It also offers more payment flexibility for those who want to use cryptocurrency. Our users will be tenants and landlords who are interested in a rental contract, especially those who are interested in alternative cryptocurrency payment methods as they become more widely popular. Their use case would be to utilize the smart contract for the rental contract payment. Through our dApp, they can navigate through starting a rental contract, checking existing contracts, or terminating a contract.

Software and Component Diagrams

(Following C4 Model Methodology)

Figure 1 - Context

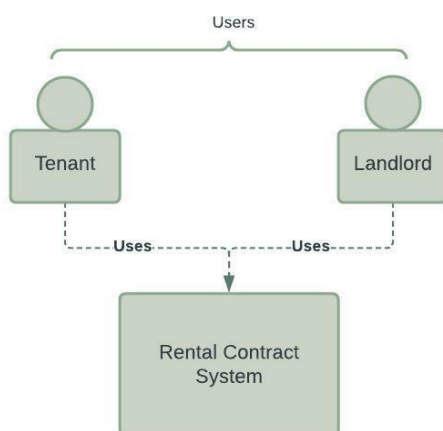


Figure 2 - Containers

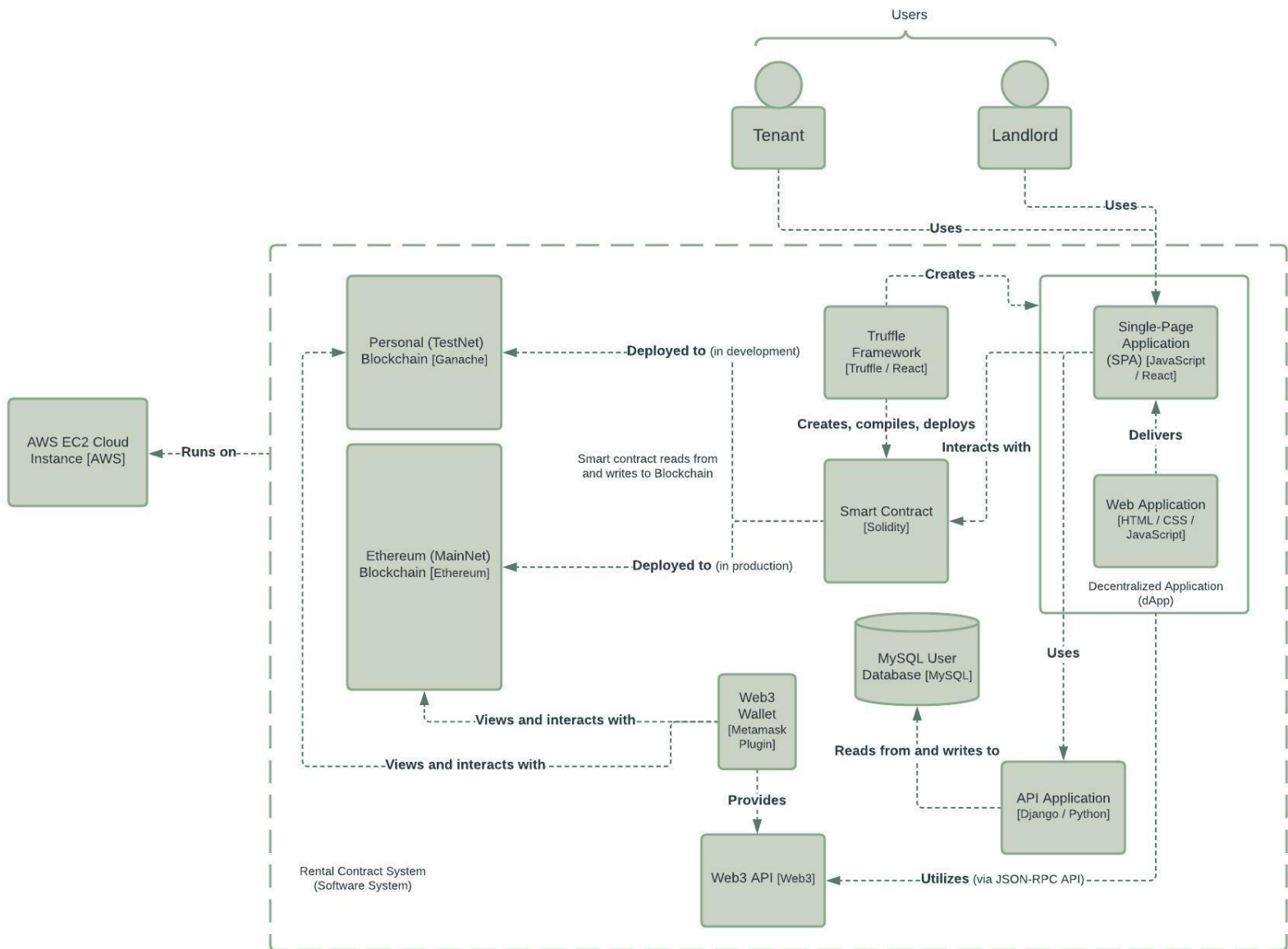
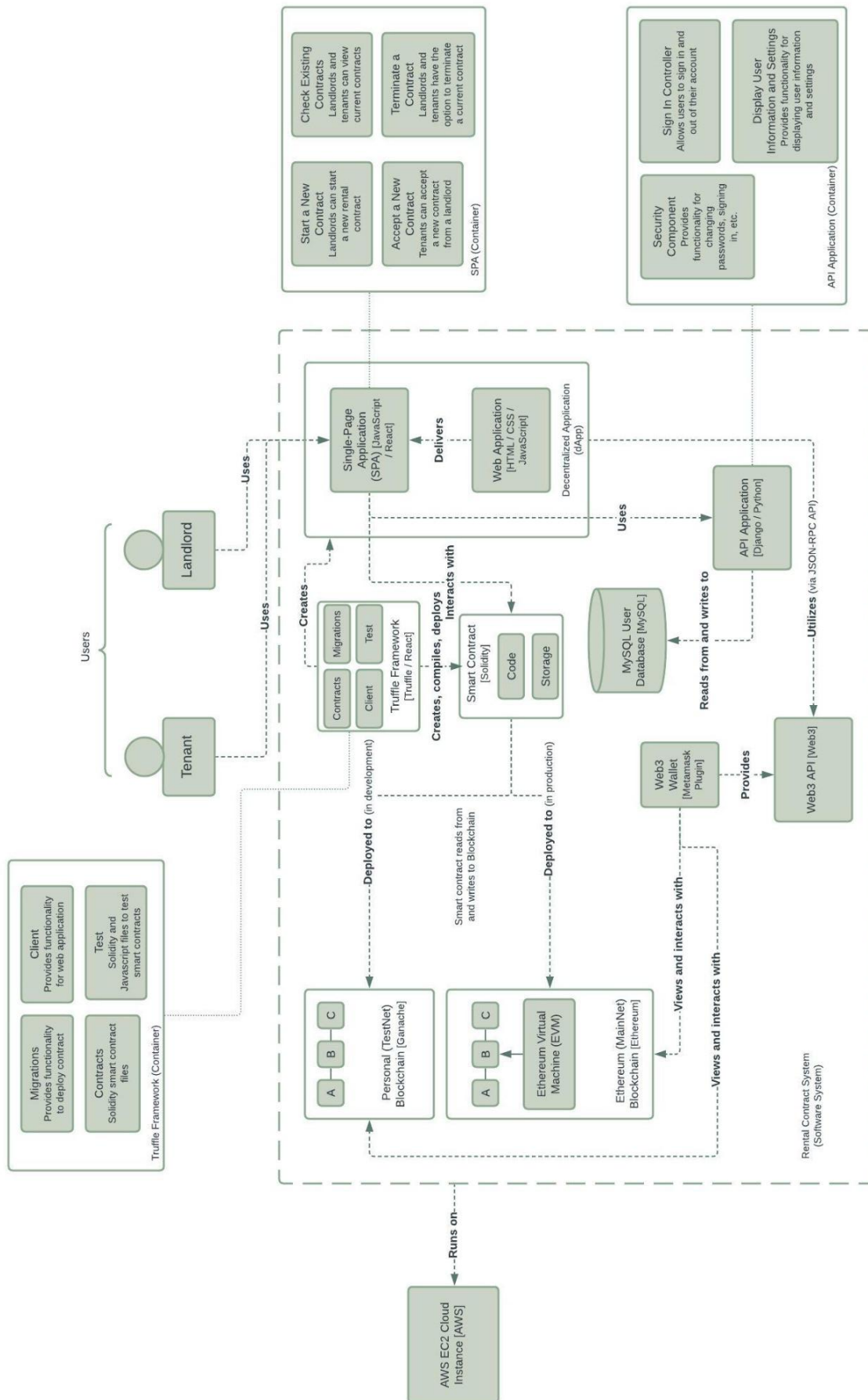


Figure 3 - Components



At the context level diagram (Figure 1), two types of users (landlord and tenant) will interact with the rental contract system.

Digging further under the hood, we have the container level diagram (Figure 2). Here, we have the pipeline from the user to the Ethereum blockchain, utilizing the dApp, smart contract, and necessary API and frameworks.

Finally, at the most detailed level, we have the component level diagram (Figure 3). First, the Truffle Framework is broken down into four main components, which are Migrations, Contracts, Client, and Test -- each of which has a specific role in generating the smart contract and dApp pages. Next, the Smart Contract consists of code and storage, and the Single-Page Application (SPA) consists of four main components for each use case -- starting a new contract, checking current contracts, accepting a new contract, and terminating existing contracts. Lastly, the API Application has three main components, which are the security component, the sign in controller, and the display of user information and settings.

Architecture Choices

Our dApp will have a layered software architecture. The following layers will be specified:

- Presentation Layer: pages for each user, the home page, tenant and landlord pages, our UI capabilities (CSS/HTML, etc)
- Business Layer: business logic for the contract will go here, the functionality and rules that are binding, customer objects
- Persistence Layer: the mapping of our database information to user objects
- Database Layer: datastore for our objects

On an individual level, the architecture of the smart contract is event-driven. The contract is created and deployed to the blockchain when a submission (and acceptance) button is clicked. After this, depending on the terms of the rental contract, the smart contract will execute a transaction (payment of x amount of money from the tenant to the landlord) a certain amount of times, where this transaction is triggered by a timer/the date (monthly). The contract will cease executing monthly transactions after either a) the amount of times specified in the contract has occurred, or b) the contract is prematurely terminated by one of the parties.

Like other browser extensions, the Metamask wallet has a microkernel (plug-in) architecture. It runs as a browser extension in Chrome, Firefox, or Edge. The dApp utilizes Metamask as an Ethereum wallet and web3 provider to interact with the Ethereum blockchain.

External Services

This application was created with a Truffle framework to integrate with the Ethereum blockchain. This gives us an outline for how to structure our development environment, testing framework, and assets, as well as how to interact with the blockchain. The front end uses React as a framework to break our design into reusable components, the framework was integrated with

truffle to make it easier to integrate with our smart contract. We will also be using Django as a framework to interact with our databases. In addition, we will use AWS to host our application on an EC2 instance.

Context and System Environment

The decentralized application (dApp) is designed to run in a web browser, as a larger desktop provides a better and more efficient experience for the user. Currently, Firefox, Chrome, and Edge are all supported web browsers, since they are supported for the Metamask plug-in. Metamask is an Ethereum wallet and web3 provider used to interact with the Ethereum blockchain. Users access their Ethereum wallet through a browser extension. It provides a fast connection to the Ethereum blockchain and enables secure and usable Ethereum-based web sites.

Datastore Choices

To store user information and settings, a MySQL database will be used. The database will store all user information, as well as account settings such as username and password. This will be queried upon login and logout. Essentially, the MySQL database is for storing anything that will not be stored on the blockchain. The decentralized application (dApp) will utilize the smart contract to read from and write to the blockchain. Additionally, we will be writing an API using the Django framework, which the dApp will utilize to read from and write to the user database. Also, the blockchain will store information about the contract transactions that can be read from.

Physical Layer

The physical layer we will use is an AWS EC2 cloud instance to store our source code and our MySQL database. All components will run in the AWS EC2 cloud instance, as seen in the software diagrams.

System Data Input and Output Mechanisms

The decentralized application (dApp) will contain a form for the creation of a smart contract. The user will input information into this form, and upon submission, a contract will be generated. The fields of the form will include the smart contract address, the addresses for both the landlord and the tenant, the gas fee, and the amount sent (in Eth and USD equivalent). The gas fee and amount sent fields will take text as input, while the smart contract address and user addresses will be a drop down menu. We will provide tips and feedback to the user through validation checks on the form. As output, upon submission of the form, the dApp will send the user a confirmation message for the transaction. Additionally, it will store a ledger of the transaction in the blockchain, and can be referenced from the current contracts page.

Team Member Responsibilities

We have split up responsibilities so that two team members will be focused on the front-end decentralized application (dApp) design and two team members will be focused on the back-end smart contract and user database development.

Claire and Genevieve will be working together on the front-end decentralized application (dApp) design. The front-end dApp will display different pages depending on the user (landlord or tenant). For example, only landlords will be able to start new contracts, and only tenants will be able to accept a new contract. Both users will be able to view existing contracts and terminate an existing contract. Claire and Genevieve will prototype the dApp desktop design using Figma. Claire will be responsible for the How-To, FAQ, and documentation pages, as well as the contract creation form page. Genevieve will be responsible for the contract viewing and terminating page, as well as helping Claire with documentation.

Ada and Katie will be working together on the back-end smart contract and user database development. Ada will be responsible for implementing the user database schema and design, as well as writing the API through Django for querying the user database. Katie will be responsible for writing the Solidity smart contract code that will create a new contract, allow the user to accept the contract, and query the blockchain to display current contracts.

Timeline

By November 19:

- Prototyping the database relational schema
- Prototyping the website pages on Figma
- Implement button execution of the simple smart contract
- create header and footer bars

By November 26th:

- Make a contract that is submitted once and executes x amounts of timely payments
- Create contract creation we page (ignore the tenant acceptance for now)

By December 3rd:

- Determine how to have users see different page options depending on user type
- Implement user login functionality
- Populate the database with user accounts for testing above functionality

By December 10th:

- Implement functionality for logged in users to attach wallets to their accounts (determine how the information entered about the wallets stored and then used when initiating payments between the users)

By December 17th:

- Implement tenant acceptance of a contract (front end page and back end functionality)
- Complete database API for querying

By December 31st:

- Create landlord and tenant “view existing contracts” pages
- Complete backend solidity functions that allow us to get informations about existing contracts from the blockchain