**Computer Science Senior Design Project Report**

**Crypto Casa**

**Rocco Salerno**

**&**

**Gibriel Spiteri**

**Advisor: Dr. Oren Segal**

**Fall 2018**

**12/19/18**

**Chapter 1: Project Scope**

**Section 1: Project Scope and Vision**

This application is designed to allow homeowners to rent out their property to individuals who are looking to stay in a home or apartment during their travels. The application utilizes the decentralization of the Ethereum Blockchain to avoid a main corporate entity from increasing prices, being an unsympathetic middleman in transaction disputes, and to avoid scams from taking place. The decentralized structure of the application means that the users are in full control of their funds and need not to utilize a third party service to facilitate transactions. Funds are stored in personal wallets for which the wallet owner has complete sovereignty over. Transactions remain secure using a Smart Contract, which holds the renters’ money and security deposit for a period. to ensure the homeowner can make a dispute claim against the renter if necessary. The renter reserves the same rights and can make a dispute against the homeowner when there are major inconveniences. The disputes are handled by a community jury that reviews cases and votes on who receives the funds in the case that the homeowner or renter breaks the community guidelines. The ultimate goal of the application is to be able to provide a safe service that builds a tight-knit community of home sharers where funds are transferable through cryptocurrency technology.

To approach this project we decided upon using iOS as our platform, Firebase as our storage, and NodeJS as our server. We decided upon using iOS as we were familiar with iOS development, Firebase was chosen over other database services like SQL as it has built in user authentication and image storage, and NodeJS was chosen over Java and .Net as there is an API that allows us to perform Ethereum transactions. At first, implementation of the Firebase was messy and daunting but we were able to successfully create a single database that handled user auth, image storage, and data storage. The NodeJS server also took time to set up as there was a steep learning curve involved in learning how to utilize the Web3 API. Our final product resulted in an a working iOS app that can list and rent out houses to users. Each house has a personal smart contract made for it that forwards funds to the homeowner after the renter pays for the nights they stay.

Chapter 2: Project Goals

● Launch a Smart Contract that stores funds between parties for a period of time

● Develop a community voting system to handle disputes between parties

● Integrate a direct messaging system between buyer and renter in the mobile application

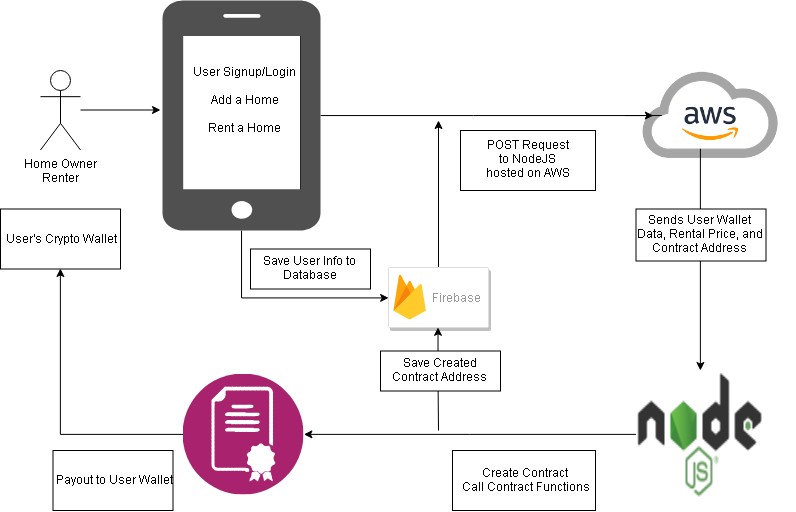
● Create a NodeJS Server with Web3JS API to connect to the blockchain

Chapter 3: Requirements

2.1: System Diagram

The mobile application will communicate directly with a database. The database will store values of the price of the home, address of the home, and the homeowner that is renting the house out. Another database will be used to store all user login information.

The mobile application will also communicate with the internal GPS of the device to populate nearby homes for rent.



2.2: Use Cases

|  |  |
| --- | --- |
| UC1 | User Sign up |
| Description | User sign up form |
| Rationale | User must have an account to use the application |
| Preconditions | None |
| Basic Course of Events | User will enter a name  User will enter a last name  User will enter an email address  User will enter a password  User will confirm password  User will submit information |
| Postconditions | The user will be forwarded to the home menu screen |

|  |  |
| --- | --- |
| UC2 | User Login |
| Description | User can login |
| Rationale | User must login to use application |
| Preconditions | UC1 |
| Basic Course of Events | User will enter email address  User will enter corresponding password |
| Postconditions | User will be forwarded to the home menu screen |

|  |  |
| --- | --- |
| UC3 | User search |
| Description | User can search for homes to rent |
| Rationale | To choose a house to rent |
| Preconditions | UC2 |
| Basic Course of Events | User enters state city, state, or zip code  User clicks submit |
| Postconditions | The user will be prompted with available homes |

|  |  |
| --- | --- |
| UC4 | User post |
| Description | User can post homes they want to rent out |
| Rationale | For users to rent from |
| Preconditions | UC2 |
| Basic Course of Events | User will enter Owner name  User will enter address  User will enter the state  User will enter the city  User will enter the zip code  User will upload a picture  User will enter the cost per night |
| Postconditions | User will be forwarded to a confirmation screen |

|  |  |
| --- | --- |
| UC5 | User message |
| Description | User can message the homeowner |
| Rationale | User gives detail about themselves |
| Preconditions | UC3 |
| Basic Course of Events | User will send a message to the owner about his/herself |
| Postconditions | The message will be forwarded to the homeowner. |

|  |  |
| --- | --- |
| UC6 | User rents |
| Description | User agrees to rent |
| Rationale | User chose a house to rent |
| Preconditions | UC3, UC4 |
| Basic Course of Events | User will choose dates to stay  User will click accept |
| Postconditions | User will be forwarded to an order confirmation page |

2.3: External Interface

|  |  |
| --- | --- |
| Name of Input/Output | Input-1 GPS location |
| Source of Input | Mobile devices location services |
| Purpose | To obtain user location |
| Processing | None |

2.4: Functional Requirements

|  |  |
| --- | --- |
| Name | FR-1 Verify user account |
| Summary | Checks if user account is valid |
| Rationale | User must have an account |
| System Behavior | System will search database for corresponding username and password. |
| References | None |

|  |  |
| --- | --- |
| Name | FR-2 User location |
| Summary | Locates where the user is |
| Rationale | Shows homes for rent nearby location |
| System Behavior | System will use mobile devices location service to grab position |
| References | FR-1 |

|  |  |
| --- | --- |
| Name | FR-3 User messaging |
| Summary | User can message homeowner |
| Rationale | User gives brief description of renting |
| System Behavior | User will send direct message via mobile application |
| References | FR-1 |

|  |  |
| --- | --- |
| Name | FR-4 Create Smart Contract |
| Summary | Smart contract created upon accepting house to rent |
| Rationale | User must pay for renting house |
| System Behavior | Smart contract will be created with user defined dates and funds will be taken from users’ wallet |
| References | FR-1 |

|  |  |
| --- | --- |
| Name | FR-5 User search |
| Summary | Lists all homes defined by user preferred location |
| Rationale | Gives user options to rent |
| System Behavior | System will search database based on users location preference |
| References | FR-1, FR-2 |

2.5: Non-Functional Requirements

2.5.1 Performance Requirements

The smart contract allows for users using the mobile application to make a transaction on the system. The response time for each transaction should be no longer then the average Ethereum transaction time which is no longer than 1 - 5 minutes.

2.5.2 Database Requirements

The mobile application will use a database to store user information and home renting information. User passwords will be hashed so they are better protected. Users’ wallet information will not be hashed because their wallet address is public.

2.5.3 Security Requirements

Since the mobile application will work off of the Ethereum network, each smart contract deployed will be hashed automatically. Transactions are verified by other systems on the network, so there is no human interaction with the security of the transaction.

**Chapter 4: Constraints**

3.1 Engineering Standards

· The Smart Contract follows the ERC20 token design standard.

· The ERC20 token standard describes the functions and events that an Ethereum token contract must implement.

· These functions are outlined in the Ethereum wiki

· https://theethereum.wiki/w/index.php/ERC20\_Token\_Standard

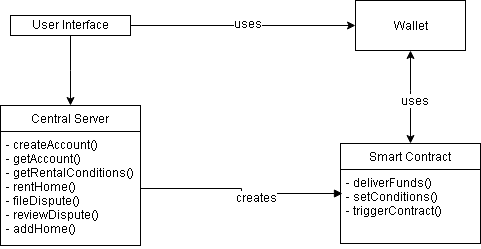
3.2 Realistic Constraints

· Legal: We realize there may be legal constraints when it comes to tax law and regulatory laws regarding individuals renting out their houses. Users renting their houses should first check their local laws to ensure our service is legal in their jurisdiction.

· Safety: With this service there exists an inherent risk to both the homeowner and renter. To protect both parties from scams we have implemented a dispute system that involves a community of “jurors” to review dispute cases. A user rating system has also been implemented for homeowners and renters to judge if they are willing to rent to a person or stay at a property. Both parties should exercise caution when living with strangers.

**Chapter 5: Project Design**

Section 4.1: Overview of System Components



4.1.1: The user interface is the bridge to utilizing the application’s functions. Here the user has access to several views that allow them to create accounts, login, add a rental property, search for a home to rent, and purchase a rental home.

4.1.2: The central server is the main model for performing the applications’ functions, it is the brain of the application.

4.1.3: The smart contract receives payment from the user’s wallet which triggers the contract’s set conditions. After, the specified contract length the funds are delivered to the appropriate user’s wallet.

Section 4.2: Structure and Relationship

The Central Server is the brains of the application, this is where most logic based functions will be implemented. The createAccount function is accessed by the user when they click on the register an account button, after they fill out the registration form the function is called and saves their data on our database. The getAccount function is called when a user tries to login into the app with their username and password, this function logs them into their account and satisfies FR-1. The getRentalConditions function is called when a user views a property on the rent a property page, it is also used for getting the conditions for the smart contract to trigger. The rentHome function triggers the smart contract to deploy and asks the renter to send funds to the contract. The fileDispute function stores the dispute complaint and evidence and posts it to the disputeCenter. The reviewDispute function triggers after every juror has given their decision and will cause the payment to be sent to the appropriate party. The addHome function triggers after the homeowner enters their information in the add a property page and it saves the information in our database.

The Smart Contract gives us the ability to store the users funds in a safe, untouchable, location for a period of time. The conditions of the smart contract are decided by the homeowner at the time they list their house on the market, the setConditions function utilizes the getRentalConditions function from the central server to set how long the contract will last and who should initially receive payment. The tiggerContract function is then called after the contract receives the funds from the renter, this confirms the rental period. Finally, after the contract has expired the deliverFunds function is called and sends the payment to the appropriate party’s address.

Section 4.3: User Interface

The user interface will consist of several different views including, the login screen, home page, add a property page, rent a property page, file a dispute page, and the handle a dispute page. All of the interfaces rely upon FR-1, which is satisfied after the user clears the login page. The homepage gives the user access to all other pages, having a button to access the add a property, rent a property, and dispute pages. The add a property page will ask a registered homeowner to input the homeowners name, the address of the property, the price, and pictures of the property. The rent a property page relies on FR-2,3,4,& 5 and will allow a user to search for a property using GPS. When the user clicks on a property they will be able to see the price of the property, the owners information, the user messaging system, and they will have the ability to rent the property from there. When a party wishes to file a dispute, they will click on the dispute button on the homepage where they will be taken to page that asks them to explain the issue and upload evidences to support their case. Finally, all users can click on the handle a dispute button on the homepage to participate as a juror in a dispute case. The dispute page lets a juror review the evidences provided by the two parties and make a decision on who gets the funds.

Section 4.4: Detailed Component Description

4.4.1: Component 1

|  |  |
| --- | --- |
| Identification | SC-1 Central Server |
| Type | A class |
| Purpose | Stores functions for sending and receiving data from the database and smart contract. |
| Inputs | User information, button presses |
| Outputs | Rental conditions, user information, dispute decisions |
| Data | Has access to the database which stores user and home information |
| Internal Structure | The class is programmed in swift, and communicates to the database with a web api in C#. The Central Server satisfies FR-1, 2, 3, 4 & 5 |
| Processing | Used for communicating with the database through a web api therefore we can send and receive information regarding user information and property information |
| Dependencies | Dependant upon a functional database and internet connection |
| Resources | Swift, SQL, C#, Web3 |
| Cross-References | FR-1, 2, 3, 4, 5 NFR 2.5.2 |

4.4.2: Component 2

|  |  |
| --- | --- |
| Identification | SC-2: Smart Contract |
| Type | A Class |
| Purpose | Receives funds from a user and sends them to another user after a set period of time |
| Inputs | Ethereum cryptocurrency and rental conditions |
| Outputs | Ethereum cryptocurrency |
| Data | Data is provided on contract creation, such as length of contract and price of contract |
| Internal Structure | Programmed in solidity. NFR- 2.5.1 Constraint 3.1 |
| Processing | Solidity code allows for the functionality to store Ethereum in an untouchable “safe” for a limited period of time. After the alloted time the Ethereum will then be sent to a receiving address |
| Dependencies | Internet connection |
| Resources | Solidity, Central Server class |
| Cross-References | NFR 2.5.1, 2.5.3, Constraint 3.1 |

4.4.3: Component 3

|  |  |
| --- | --- |
| Identification | SC-2: User Interface |
| Type | A View Class |
| Purpose | The junction between the user and our program |
| Inputs | User touch, button presses, and typing into text fields |
| Outputs | Visual Updates |
| Data | Users provide data into text fields and image uploaders, which are then posted to the database through the central server |
| Internal Structure | Programmed in Swift 4 with XCode. |
| Processing | Allows for the visual expression of our application so that people using the application can easily and effectively transact with our system |
| Dependencies | iPhone device and Internet Connection |
| Resources | Swift 4, Central Server class, iPhone 6 and better |
| Cross-References | N/A |

Section 4.5: Reuse

We will be using an open source voting api for smart contracts, which we will incorporate in our jury system to count the decisions jurors make.

Section 4.6: Design Decisions and Tradeoffs

We decided to make our application an iPhone mobile application rather than a web application. This was because we wanted to compete with the AirBnB app and also because we could apply our mobile device class to the senior project.

Also we had to make the decision of what database to use out of the options of SQL and Firebase. Ultimately, we went with SQL because of our familiarity with using it and learning about it in our Databases class.

Section 4.7: Resource List

Solidity Voting Contract

The Swift Programming Language - Book by Apple on the iTunes store

iPhoneX - Rocco’s iPhone used for testing the App

Gibriel Spiteri - Available Thursdays, Fridays, Saturdays, Sundays

Rocco Salerno - Available Monday, Tuesday, Friday, Saturday, Sunday

Pr. Oren Segal - Senior design advisor, available when requested

Section 4.8: Resource Course List

Software Engineering, Mobile Device Programming, Web Development, Databases

**Chapter 6: Project Plan**

**Section 5.1: Sprints**

5.1.1: Sprint 1 (30 hours) will be from June to July and consist of creating the user interface, web API, and the database. Once we have our Web API and Database up, we can then implement UC1 and UC2 which are the user login and user signup functions. Gibriel will work on the user login which is estimated at 5 hours and Rocco will work on the user signup which is also estimated at 5 hours. The database and Web API are estimated to take 20 hours.

5.1.2: Sprint 2 (25 hours) will be from August to September and we will continue to research the proper implementation of smart contracts within our application. Gibriel and Rocco will both be spending at least 10 hours each on research of how to integrate the smart contract. The remaining 5 hours will be for UC4, which is the ability for the user to post a listing.

5.1.3: Sprint 3 (20 hours) will be from October to November and we will implement a smart contract on the iOS mobile device, ensuring transactions over the local network are configured correctly. Once we have UC4 working properly, we then want to implement UC6, which is the ability for a user to rent. This would then invoke the smart contract, where we can test to see if it is working properly. Gibriel will be working on the smart contracts functions while Rocco will be working on the integration of the smart contract to Swift 4.

5.1.4: Sprint 4 (20 hours) will be the month of December and we will work to make the smart contract work over the Ethereum network outside of the local connection. Both Gibriel and Rocco will work on testing and debugging at this state. The estimate of debugging time is 15 hours and 5 hours to correctly implement the application over the Ethereum network.

**Section 5.2: Project Schedule**

5.2.1: Gibriel will be working on the user interface. The estimated time to work on the UI will be four weeks. Rocco will be working on the Web API and the Database. The Web API and database should be completed within two weeks. For the remaining two weeks, Rocco will help with the UI.

5.2.2: Both Gibriel and Rocco will research smart contracts and more specifically how to implement them on an iOS mobile device. This ensures that we both understand how our system works.

5.2.3: For the smart contract, Gibriel will work on the voting functions and Rocco will work on the Payout functions. The estimated time for both is two weeks of coding, with two weeks of proper debugging.

5.2.4: The implementation of the smart contract over the Ethereum network will be done by both Gibriel and Rocco. It is estimated that three weeks will be needed in order to debug and correct any issues that may arise.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Task Name | Duration | Start | Finish | Predecessors | Assigned To | % Complete | Status |
| Create User Login UI | 2h | 04/26/18 | 04/26/18 |  | Rocco Salerno | 100% | Complete |
| Create User Signup UI | 2h | 04/30/18 | 04/30/18 |  | Rocco Salerno | 100% | Complete |
| Upload Home UI | 2h | 05/01/18 | 05/01/18 |  | Gibriel Spiteri | 100% | Complete |
| Create Database | 3d | 06/01/18 | 06/05/18 |  | Rocco Salerno | 100% | Complete |
| Create Web API | 10d | 06/04/18 | 06/15/18 |  | Rocco Salerno | 50% | N/A |
| Connect Login to Database | 2d | 06/18/18 | 06/19/18 |  | Rocco  Salerno | 100% | Complete |
| Connect Signup to Database | 2d | 06/21/18 | 06/22/18 |  | Rocco  Salerno | 100% | Complete |
| Research of Smart Contract | 25d | 06/25/18 | 07/27/18 |  | Rocco Salerno | 70% | Complete |
| Research of Smart Contract | 25d | 07/04/18 | 08/07/18 |  | Gibriel Spiteri | 100% | Complete |
| Implement User Postings | 5d | 08/10/18 | 08/16/18 |  | Rocco Salerno | 100% | Complete |
| Implement User Posts | 5d | 08/10/18 | 08/16/18 |  | Rocco  Salerno | 100% | Complete |
| Have users rent home | 9d | 09/28/18 | 10/10/18 |  | Rocco Salerno | 100% | Complete |
| Have users rent home | 9d | 10/29/18 | 11/08/18 |  | Gibriel Spiteri | 100% | Complete |
| Debug Project | 13d | 11/26/18 | 12/12/18 |  | Rocco Salerno | 100% | Complete |
| Debug Project | 13d | 11/26/18 | 12/12/18 |  | Gibriel Spiteri | 100% | Complete |
| Implement App over Network | 5d | 12/12/18 | 12/18/18 |  | Gibriel Spiteri | 100% | Complete |

**Section 5.3: Risk Plan**

5.3.1. Network Problems:

It is possible that when deploying the smart contract to the network, we may encounter problems. We may need actual funds that we do not have. If this problem does arise, we will keep the smart contract on a test network using fake funds.

5.3.2. Mobile incompatibility:

Smart contracts are still very new and although there are applications out that include them, documentation for implementing them on a mobile device may be hard to find. If this problem comes up, we will revert over to a web application.

**Section 5.4: Estimated Financial Budget**

Due to the nature of this mobile application, it is estimated that we will have no financial budget because nothing will need to be purchased.

**Section 5.5: Teamwork Plan**

During the summer months we plan on meeting in person bi-weekly, while verbally meeting at least once a week. However, when September comes, we plan on meeting in person at least twice a week and verbally meeting twice a week as well. We will use applications such as Discord for communication purposes as well as screen sharing, GitHub for storing and monitoring all source code and version control, and Target Process as a management tool to see what tasks are still at hand and which ones we have completed.

**Chapter 7: Test Plan**

**Section 7.1 Test plan:** Unit testing along with backend testing was used heavily in our project. Unit testing was used on all major components added to the project. Backend testing was used primarily when dealing with Firebase database and when connecting to the server. After each main component was created, It was launched to a live iPhone for testing. If the component passed all the preconditions and gave us the results we expected, it was marked off as completed.

**Section 7.2 Test cases:**

|  |  |
| --- | --- |
| **Name** | **T.C*. 1 Verify that signup works*** |
| **Type** | **Unit Test** |
| **Description** | **Ensure that the signup page works as intended** |
| **Preconditions** | **N/A** |
| **Basic course of events** | ***1. Enter user name, password, and verify password***  ***2. Click on submit*** |
| **Expected results** | **User should be redirected to the login page** |
| **Acceptance criteria** | **User has entered an email and password, then verifies the passwords match** |
|  |  |

|  |  |
| --- | --- |
| **Name** | **T.C 2 Verify that logging in works** |
| **Type** | **Unit testing** |
| **Description** | **User should be able to successfully login** |
| **Preconditions** | **T.C 1** |
| **Basic course of events** | ***1. Enter email and password***  ***2. Click on Login*** |
| **Expected results** | **User should be redirected to the home page** |
| **Acceptance criteria** | **User has entered a valid email and password and will be redirected to the home page.** |
|  |  |

|  |  |
| --- | --- |
| **Name** | **T.C 3 Submit information to database** |
| **Type** | **Backend Testing** |
| **Description** | **User home details should be stored in the database** |
| **Preconditions** | **T.C 1, T.C 2** |
| **Basic course of events** | ***1. User fills his details on the form***  ***2. User picks an image to upload***  ***3. User fills out the home details***  ***4. User clicks submit*** |
| **Expected results** | **User and home details should be stored in the firebase database** |
| **Acceptance criteria** | **User has entered information in all fields, and has selected a photo to upload** |
|  |  |

|  |  |
| --- | --- |
| **Name** | **T.C 4 Checking for a WalletID** |
| **Type** | **Unit Testing** |
| **Description** | **User should have a wallet before they rent property** |
| **Preconditions** | **T.C 1, T.C 2** |
| **Basic course of events** | ***1. User clicks on the side menu***  ***2. User clicks on the “Add WalletID” option***  ***3. User enters their WalletID***  ***4. User clicks submit*** |
| **Expected results** | **User’s WalletID should be posted to the database and user redirected to home page** |
| **Acceptance criteria** | **User enters a valid WalletID** |
|  |  |

**Section 7.3 Test Schedule:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Test name and #** | **Person Responsible** | **Sprint #** | **Projected Completion Date** |
| **T.C 1** | **Rocco** | **2** | **October 2018** |
| **T.C 2** | **Rocco** | **2** | **October 2018** |
| **T.C 3** | **Rocco** | **3** | **November 2018** |
| **T.C 4** | **Rocco** | **4** | **December 2018** |

**Chapter 8: Project Methods**

**Section 8.1:** **Technical methods:**

**Section 8.2: Tools:**

* Platform: We made our app on the iOS platform using the Swift programming language as that is the mobile platform we were most familiar with.
* Database: We used Firebase for our database and storage server, Firebase allows us to store home listing information, photos, and user login information securely. All while being very manageable to handle on the Swift end. <https://firebase.google.com/>
* Blockchain API: We used Web3js on our NodeJS server to communicate with the Ethereum blockchain, allowing us to deploy our smart contracts and execute transactions swiftly. <https://web3js.readthedocs.io/en/1.0/>

**Section 8.3: Programming languages:**

* JavaScript: Used for server side code, receiving post requests, and communication with the blockchain through NodeJS and Web3js.
* Solidity: Used to create the Ethereum smart contract that holds and forwards funds to the homeowner. <https://solidity.readthedocs.io/en/v0.4.24/>
* Swift: Used to program the mobile apps functionality.

**Section 8.4. Use of open source code:** what open source code you used, describe it shortly, where you used it and cite references as needed.

**Section 8.5.** **New tools and knowledge**: list the new tools, new programming languages and new knowledge (technical methods) you learned and used for the first time in your project. For each include: the name, type (library, IDE, programming language, algorithm, etc.) and where you used it.

* Solidity, Programming Language: Solidity is a contract-oriented programming language influenced by C++. Python, and JavaScript. This language was used to develop the smart contract for renting out homes in our application.
* Remix, IDE: This IDE is used in conjunction with Solidity so that we can test the contracts that we create on a test Ethereum network as we program in new features. <https://remix.ethereum.org>
* Firebase, Database: Firebase is a Backend-as-a-Service that is ran by Google. We use Firebase realtime database, file storage, and user authenticator.
* Web3js, API: web3.js is a collection of libraries which allow you to interact with a local or remote ethereum node, using a HTTP or IPC connection. We use Web3 for the deployment of our smart contracts and for making user transactions.

**Chapter 9: Project Implementation and Results**

**Section 9.1: Implementation Strategy:**

Our team agreed to split the work based on our teams interests, the iOS application and the smart contract. Rocco was interested in mobile programming so he handled the development of the frontend and backend of the iOS app and Gibriel was interested in cryptocurrencies and blockchain development so he programmed the smart contract and the Nodejs server with Web3 API. We let eachother work at their own pace and develop features they felt would benefit the final product.

**Section 9.2. Sprints:**

The first sprint involved creating home rental page and home upload page for the iOS app and to create a basic smart contract that could send funds between two people on the blockchain side. In the second sprint Rocco added a database for storing user information and Gibriel added a constructor, payment functions, and transfer functions to the smart contract and set up the requirements for the Node server. In the third sprint Rocco ran into complications with the database and picture uploads, he attempted to fix these issues while working on the apps backend functions. Gibriel attempted to retrieve data from the database for use in deploying a contract through Web3 but this would not be successful until later. In sprint 4 Gibriel made data retrieval to the server functional through post requests, got the Web3 API working to create contracts and send transactions, and hosted the server on Amazon Web Services. Rocco worked to create the post requests on the swift side and he polished the app. Our team was very communicative with each other, we saw each other every Monday, Tuesday, and Wednesday to discuss our progress and help each other.

**Section 9.3. Results:**

A smart contract is created by the server every time a user adds a new home to the database, the contract then stores the home owners wallet ID for later function transactions. The smart contract is able to receive payment from a renter, payout to the homeowner, and refund the renter. The smart contract updates a map array to record who has rented the property and how much they have paid.

Contract Code:



Sample Contract Creation Code:



**Chapter 10: Risks**

When implementing our application we ran into risk 5.3.2. Mobile incompatibility. Our smart contract takes in several parameters that are stored in the Firebase, the trouble came when we tried to send these parameters from Firebase to NodeJS. Our solution was to use the “Express” library to host a server through NodeJS, we were then able to send post requests from Swift containing the parameters in a JSON object to the server . This way we were able to route the data from Firebase through Swift and into NodeJS where we could deploy the contract with Web3.

**Chapter 11: Financial Budget**

No capital was spent on the production of this application

**Chapter 12: Conclusions**

Our Project aimed to create a home rental application on a mobile device that facilitates monetary transactions through crypto currencies. The application was planned to implement a smart contract programmed in the Solidity language which would hold funds for a period of time and payout to the correct party after that time had elapsed. The respective party to payout to would be chosen by a panel of jurors in the event of a dispute. The mobile application successfully implemented the features to host and rent a home. We have a successful smart contract that is deployed whenever a user adds a home to the database, and transactions are sent to the contract when a user rents a home. The only feature we were not able to implement was the dispute system which is estimated to take another month to complete, however the current smart contract implementation is set up to be compatible with the dispute system. We decided to leave out the dispute system so that the main features of the application would be more polished and seamless for the user before we started to work on extra features.

Total Project Completion: 90%

**Appendix**



**The smart contract code**

****

**Contract creation in Nodejs/Web3**

**References**

Firebase: <https://firebase.google.com/>

Solidity: <https://solidity.readthedocs.io/en/v0.4.24/>

Remix Solidity IDE: <https://remix.ethereum.org>

Web3JS: <https://web3js.readthedocs.io/en/1.0/>

Smart contract voting app - <https://gist.github.com/maheshmurthy/3da385a42678c3e36a8328cbe47cae5b>

How to design a voting app - <https://medium.freecodecamp.org/developing-an-ethereum-decentralized-voting-application-a99de24992d9>

The swift programming language - <https://itunes.apple.com/us/book/the-swift-programming-language-swift-4-1/id881256329?mt=11>

Version control - <https://github.com/GibrielSpiteri/cryptoCasa>

Time management tool - <https://www.targetprocess.com/>