

Recommendation Letter System

Team members

Anurag Joshi
Kevin Sun
Muhammad Abdullah
Vinay Mishra
Yingtian Wang

Project Sponsors/Facilitators

Peter Henstock
Nicolas Tejera
Jay Bergerson

CSCI E-599 Software Engineering Capstone
Harvard Extension School

Submitted on: May 07, 2018

Table of Contents

Part 1 - Introduction	2
Part 2 - Design of Recommendation Letter System	8
2.1. Use case diagram	8
2.2. Student dashboard activity diagram	9
2.3. Recommender dashboard activity diagram	12
2.4. Blockchain smart contract programming	13
2.5. Blockchain smart contract compilation and deployment	14
2.6. Service Layer - Recommendation Letter Request Controller	16
2.7. Service Layer - Recommendation Letter Controller	16
2.8. Non-blockchain Component Class Diagram	17
2.9. User Authentication Service	18
Part 3 - Testing Results	19
3.1. End to end unit-testing	19
3.2. API testing by Postman	19
3.3. Blockchain contract unit testing	22
3.4. Integration unit test coverage	23
3.5. Contracts unit testing	24
Part 4 - Development Process & Lessons Learned Reflection	25
4.1. Meeting the requirements	25
4.2. Estimates	32
4.3. Risks	37
4.4. Team Dynamic	37
Part 5 - Appendix	38

Part 1 - Introduction

Abstract

The purpose of this project is to create a system for requesting and submitting recommendation letters. The current workflow for students submitting recommendation letters to universities is complex, laborious, and redundant. A user-friendly system with an intuitive interface, which strives to standardize the process, has the potential to solve a real-world problem in an efficient manner. This project attempts to take a fresh approach to submitting and requesting recommendation letters, one that solves the above issues. The project utilizes mature UI and service technologies like AngularJS and Express for front-end and innovative peer-to-peer technologies like blockchain (Ethereum) and distributed file storage (IPFS) in the back-end. Blockchain technology brings immutability, traceability and decentralization to the system.

Keywords

Blockchain, Recommendation Letter, Education, Decentralized data storage, Ethereum, Smart Contract, IPFS

Introduction

Typically, in a recommendation process, students ask their faculty or advisers for recommendations and then schools review these letters as part of their admission process. There are two main pain points with the current system. First, there is no standardization around what information schools request, so the students and recommenders are burdened with filling out different submissions for various school. If a student is submitting applications to multiple schools, letters and supporting information may need to be filled in multiple times. The other issue is that universities are burdened with determining the authenticity of these letters, confirming that both the student and the recommender are who they say they are. The lack of standardization of results needs to be addressed.

A recommendation letter system using blockchain is proposed to solve the problems above. The system has the potential to serve a vast base. There are expected to be about 3.6 million high school students graduating in 2018 in the United States, with many of them potentially needing recommendation letters.

Literature review

Recently there has been a growing use of blockchain in the field of education, for uses like storing and managing degree information and issuing online certificates. Massachusetts Institute of Technology media lab issues a certification stored on a blockchain network to students who pass the assessment [1]. Similarly, Sony Global Education uses a global assessment platform to provide services for storing and managing degree information using blockchain technology [2]. Blockchain technology is still under rapid development. The blockchain-based recommendation letter system presented here

continues this trend of using blockchain in education and it is also intended to serve as a stepping-stone to apply this technology in other fields like health care, finance and beyond.

Features of blockchain

The benefits that blockchain brings to the recommendation systems are immutability, traceability, and decentralization[3]. The decision to use blockchain technology in the recommendation letter system stems from the need of authentic and immutable data storage. It means that any recommendation written to a blockchain cannot be changed, not even by a system administrator. This provides transparency and convenience for audits. It also adds credibility to the system and prevents fraudulent behavior in the recommendation process. The blockchain feature of traceability means that all actions on blockchain are recorded chronologically. Therefore, the actions of school, recommender, and student is trackable by examining and validating the blocks. Decentralization comes from the fact that the validation process between nodes happens through cryptographic algorithms rather than through manually managed centralized organizations. Furthermore, because the recommendation letter data stays distributed, the system avoid a single point of failure of content attack. However, in order to connect the real world with blockchain data, usually a centralized organization which serves as an oracle [7] of the software system is necessary. Therefore pure decentralization can be difficult. The design of which portion of a system should be centralized or autonomously decentralized and how they should be incorporated with each other is one of the most difficult design decisions that needs to be addressed in blockchain-based applications. There is an intriguing ideology around decentralized autonomous technology that blockchain brings with it. Truly trustless systems without the cost of an intermediary have the potential to greatly reshape financial [4], notary [5], medical [6] applications among others.

Furthermore, the blockchain technology is new and exploratory. To achieve its immutability, blockchain is slower than most commonly used databases [8]. Scalability issues are one of the biggest challenges for blockchain technology to be more widely used in industry [9]. This is also one of the bottlenecks in the blockchain-based recommendation system.

Protocol vs Application

Blockchain should be considered as a protocol, the recommendation system is a specific purpose application that is to be built on top of this blockchain protocol [10]. To build a blockchain protocol from scratch is usually a years-long project and prone to failure. For example, one of the leading blockchain companies, R3, spent \$59 million and several years trying build a consortium blockchain before admitting failure. Thus, choosing the most well-supported and tested blockchain protocol is critical to the success of a blockchain-based application project. There are many blockchain protocols out there, but the two predominant ones are Ethereum and Hyperledger Fabric.

Ethereum vs Hyperledger Fabric

Ethereum [11] and Hyperledger Fabric provide developers with the Turing-complete programming languages [12] to develop smart contracts. A smart contract is software that executes automatically when conditions are met to fulfill certain obligations between parties.

Ethereum can be thought of as a globally decentralized, unownable computer for executing peer-to-peer contracts. Unlike a traditional cloud computing service like Amazon EC2, the Ethereum public blockchain is a single global computer that can be accessed by anyone with internet access. It provides a programming language called Solidity to develop smart contracts. Solidity-written smart contracts are executed by the Ethereum Virtual Machine (EVM). The students in recommendation systems are able to take advantage of deployed smart contracts on blockchain to interact with the recommenders. Hyperledger Fabric is also a distributed ledger platform that runs smart contracts with pluggable modules. Hyperledger Fabric is a permissioned blockchain using Byzantine fault-tolerant algorithms to reach consensus.

Public vs Private Blockchain

Blockchain can be classified as permissionless (public) or permissioned (private) [13]. In a public blockchain, anyone can participate in different ways such as by contributing to consensus, hosting nodes, making transactions, etc. On the other hand, only permissioned participants can interact with a private blockchain. A permissioned blockchain provides better access control management and flexibility, however, because of access restriction, it is difficult to design an economic model which incentivizes the miners to participate in the network. In public blockchain, the miners have been growing steadily over time, so miner incentivisation is less of a problem [14]. While Hyperledger Fabric [15] is permissioned only, the advantage of using Ethereum is that it can be used as both permissioned and permissionless blockchain. If we develop our recommendation letter system on Ethereum, the code can be deployed on the public Ethereum network or on a private network when we host our private nodes [16].

Blockchain Implementation

The recommendation letter system is built on the Ethereum blockchain for its powerful feature of smart contract. Smart contracts are not a replacement for real life contract, rather they are agreements of self-executing code at certain trigger event. In the case of recommendation letter system, these contracts are used to encode a set of pointers to the recommendation letter request and recommendation letter data that is stored in IPFS.

Ethereum can be slow and expensive to use in order to reach consensus among thousands of nodes. Storing all recommendation letters on Ethereum blockchain is expensive and cumbersome [17]. To solve these issues, an off-chain data storage solution named IPFS (Interplanetary File System) [18] is proposed. IPFS is a decentralized data storage system. Saving data on IPFS provides a unique hash. A common solution is to store the unique IPFS hash instead of storing all the data in the contract [19]. For the reason above, the recommendation letter file is stored in IPFS distributed database. In addition, to make every recommendation record trackable and verifiable, each letter request is also recorded on the blockchain. A letter request contract has the information about the requesting student, the recommender and the associated school program.

Cost

Running Ethereum contracts cost Gas[20], which is the execution fee for every execution made on ethereum and its price is expressed in ether. Ether is the native cryptocurrency on Ethereum. In the current implementation, to simplify the scope of this project, all the fees are paid to the miners by the service provider of the system, potentially by a consortium of schools who use this recommendation letter system. The cost can be significantly higher than a recommendation letter system that don't use blockchain.

For future improvement, a payment model can be introduced to built on top of the current implementation. In that case, each users such as student, recommender and school only need to pay a small amount of fee to execute the contracts, presumably under a few dollars worth of ether. This transaction of each payment is validated by Ethereum network, not validated by a central organization. This would reduce the necessity for a service provider, and form a self-maintained economic system. This implementation will reduce the cost to run the centralized organization, it should have a less cost overall compare to a system that do not use blockchain.

The challenge of a self-maintained payment model is less about settlement of payments, since the Ethereum network handles well itself, rather about user management. A transaction of a letter request can be made anonymously so that the system loses track of the real life identity of the user. A suggestion to solve the problem is to use the decentralized user identity management system uPort [21]. uPort is an open identity system which allows users to register their own identity on Ethereum, send/request credentials and sign transactions. Because the user identity and the payment both stays on Ethereum, it is easier to manage it together.

In addition, we have implemented a letter ownership contract which follows ERC721[22] token standard that can be used for the payment model.

Miners Incentivisation

Miners play an essential role in any blockchain application because they validate the data recorded on blockchain. Consideration of miner incentivisation is also a key factor for the design choice of public/private network. In the Ethereum public network, there are sufficient general purpose miner, so there are less concerns about whether a block is processed by a miner. In a private blockchain network, only the user of the system can mine the network, this provides the benefits of easier user management and privacy protection. However, due to the exclusion of general purpose miners, if there are not enough miners, the blockchain software suffers from longer and uncertain confirmation time that could lead to horrible even unusable user experience. Furthermore, a private network separates itself from the native cryptocurrency of Ether so that a new cryptocurrency or token may need to be designed to incentivize the miners.

Overall, miner incentivisation mechanism is a difficult and important consideration for the design decisions in the letter system. The recommendation letter system takes the approach to develop reusable code for both public and private Ethereum blockchain. This is also a reason to choose Ethereum than Hyperledger Fabric. Because Hyperledger fabric is for private blockchain only. Future developer can develop on top of our system. For a private network, a layer of miner permission management and

incentivisation mechanism is needed while for a public network, user data privacy protection needs to be enhanced.

Related Projects

Blockchain technology is in its early stage. Before the infrastructure or protocol level of blockchain become faster and mature, many application level of blockchain projects are still a proof of concept demo or under development.

Here are some projects that use blockchain for various purpose of record keeping:

MedRec: Medical Data Management on the Blockchain[23]

Library Management System: A community library based on Ethereum blockchain, implemented using Solidity and Truffle[24]

Conclusion

The blockchain-based recommendation letter system is a functional software that brings standardization to school's recommendation process. It uses blockchain for immutability, traceable and decentralized way of the recommendation letter record processing.

References

- [1] D. Skiba, "The potential of Blockchain in education and health care.," *Nursing education perspectives*, vol. 38, no. 4, pp. 220-221, 2017.
- [2] M. Hoy, "An introduction to the Blockchain and its implications for libraries and medicine.," *Med. Ref. Serv. Q.*, vol. 36, no. 3, pp. 273-279, 2017.
- [3] G. Chen, B. Xu, M. Lu and C. N.-S., "Exploring blockchain technology and its potential applications for education.," *Smart Learning Environments*, 2018.
- [4] A. Tapscott and D. Tapscott, "How blockchain is changing finance.," *Harvard Business Review*, vol. 1, 2017.
- [5] A. Arredondo, "Blockchain and certificate authority cryptography for an asynchronous on-line public notary system," December 2017. [Online]. Available: <https://repositories.lib.utexas.edu/handle/2152/63754?show=full>.
- [6] A. Azaria, A. Ekblaw, T. Vieira and A. Lippman, "MedRec: Medical Data Management on the Blockchain.," in *International Conference on Open and Big Data (OBD)*, 2016.
- [7] X. Xu, C. Pautasso, L. Zhu, V. Gramoli, A. Ponomarev, A. B. Tran and S. Chen, "The Blockchain as a Software Connector," in *13th Working IEEE/IFIP Conference on Software Architecture (WICSA)*, 2016.
- [8] C. K. e. al., "On Scaling Decentralized Blockchains.," *Financial Cryptography and Data Security. FC 2016. Lecture Notes in Computer Science*, Springer, Berlin, Heidelberg, 2016.
- [9] Z. Zheng, S. Xie, H.-N. Dai, X. Chen and H. Wang, "Blockchain Challenges and Opportunities: A Survey.," *International Journal of Web and Grid Services*, 2017.

- [10] "R3 Appears to Admit Defeat, Stops Blockchain Development," 25 Feb 2017. [Online]. Available: <https://themerikle.com/r3-admits-defeat-stops-blockchain-development/>. [Accessed 19 Apr 2018].
- [11] G. WOOD, "ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER," EIP-150 REVISION, 2017.
- [12] V. Buterin, "A next-generation smart contract and decentralized application platform.," Ethereum White Paper, 2014.
- [13] M. Vukolic, "Rethinking permissioned blockchains," April, 2017.
- [14] "Ethereum Network HashRate Growth Chart - Etherscan.," [Online]. Available: <https://etherscan.io/chart/hashrate>.
- [15] M. Valenta and P. Sandner, "Comparison of Ethereum, Hyperledger Fabric and Corda.," FSBC Working Paper, 2017.
- [16] "A public or private blockchain? New Ethereum project could mean," [Online]. Available: <https://www.americanbanker.com/opinion/a-public-or-private-blockchain-new-ethereum-project-could-mean-both>.
- [17] "blocks - Storing document/file in blockchain - Ethereum Stack Exchange.," 08 August 2016. [Online]. Available: <https://ethereum.stackexchange.com/questions/7842/storing-document-file-in-blockchain>. [Accessed 16 April 2018].
- [18] "IPFS," [Online]. Available: <https://ipfs.io/>. [Accessed 16 April 2018].
- [19] J. Benet, "IPFS - Content Addressed, Versioned, P2P File System," 14 Jul 2014. [Online]. Available: <https://arxiv.org/pdf/1407.3561v1.pdf>.
- [20] (2016, January 20). transactions - What is meant by the term "gas"? - Ethereum Stack Retrieved May 6, 2018, from <https://ethereum.stackexchange.com/questions/3/what-is-meant-by-the-term-gas>
- [21] "UPort," [Online]. Available: <https://www.uport.me/>. [Accessed 19 April 2018].
- [23] Azaria, A., Ekblaw, A., Vieira, T., & Lippman, A. (2016). MedRec: Using Blockchain for Medical Data Access and Permission Management. 2016 2nd International Conference on Open and Big Data (OBD). doi:10.1109/obd.2016.11
- [24] I. (n.d.). Imaginea/lms. Retrieved from <https://github.com/Imaginea/lms>

Part 2 - Design of Recommendation Letter System

The recommendation letter system architecture is similar to a standard 3-tier system. There is a user interface, a service layer containing business logic, and a backend for persistence. The difference is that the backend is divided into non-distributed (MongoDB [1]) and distributed (Ethereum [2], IPFS) systems for storage. The overall system is not a true DApp (decentralized application) [3], but a hybrid that contains both centralized and decentralized parts.

The system contains the following layers and technologies as shown in *Figure 1*:

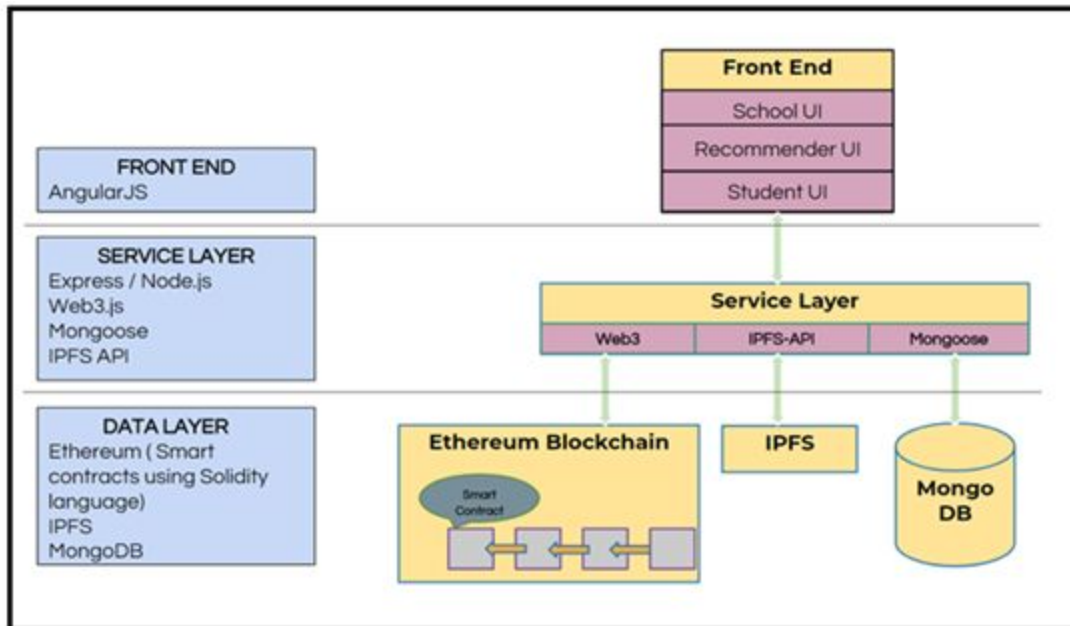


Figure 1: System technology stack

2.1. Use case diagram

Recommendation letter system use case diagram is depicted in *Figure 2*. The three actors in the system are student, recommender and school. The description of use cases is as follows:

- A. **Signup as student:** A student signs up into the system by entering name, email, address, city, state, zip and password.
- B. **Signup as recommender:** A recommender signs up into the system by entering name, email, address, city, state, zip and password.
- C. **Signup as school:** A school administrator signs up into the system by entering name, email, address, city, state, zip and password.
- D. **Login:** A student, recommender and school uses authorized email and password to login into the system. If the incorrect combination of login and password is provided, system displays the user authentication error. Upon successful authentication system displays the dashboard respective to the user role e.g. student dashboard, school dashboard or recommender dashboard.

- E. ***Request recommendation:*** A student initiates the activity to request recommendation from student dashboard.
- F. ***Search and select recommender:*** On the request recommendation page a student searches the name of recommender and selects a recommender.
- G. ***Search and select degree program:*** On the request recommendation page a student searches the name of school degree program and selects a program.
- H. ***Invite recommender:*** A student invites a new recommender to register into the system by entering the recommender email.
- I. ***Submit a pending recommendation:*** A recommender selects a pending recommendation request from the recommender dashboard.
- J. ***Upload a pdf document:*** On the submit recommendation page, a recommender presses the 'Upload' button to upload a pdf file from the system.
- K. ***Answer the question:*** On the submit recommendation page, a recommender answers the school program specific questions to submit the recommendation.
- L. ***View a submitted recommendation:*** On the recommender dashboard page, a recommender select the 'View' option for a submitted recommendation, which displays the pdf file as embedded document with program specific questions and respective answers.
- M. ***Setup degree program:*** On the school dashboard page, the school administrator presses the 'Add' button to add a degree program.
- N. ***Add questions specific to degree program:*** A school administrator enters the program name, selects the questions specific to the degree program and the option for the read access of recommendation letter to the student for this program.
- O. ***Look up recommendation letter for student:*** A school administrator enters the student name to view the recommendation letters submitted for the student application. The system displays the list of recommendation letters with corresponding recommenders name who submitted the letters. Using the 'View' button the pdf file is displayed as embedded document with program specific questions and respective answers.
- P. ***View degree program questions:*** On the school dashboard page the list of degree programs entered into the system is displayed. A school administrator presses the 'View' button to review the program specific questions and their respective answer choices.

2.2. Student dashboard activity diagram

Figure 3 describes the recommendation letter initiation process from the student dashboard. A student logs into the system and upon successful authentication, searches for the target program by entering the school name. The system provides the list of registered school programs. The student selects a program, searches for the recommender and selects one recommender. If the letter from this recommender already exists in the system, student submits this letter to school. Otherwise, student requests the letter from recommender who receives the notification to provide the letter on behalf of student. The recommender fills the survey form, attaches the letter pdf file and submits it into the system. The destination school and student both receive notifications in their dashboards for the recommender's letter submission.

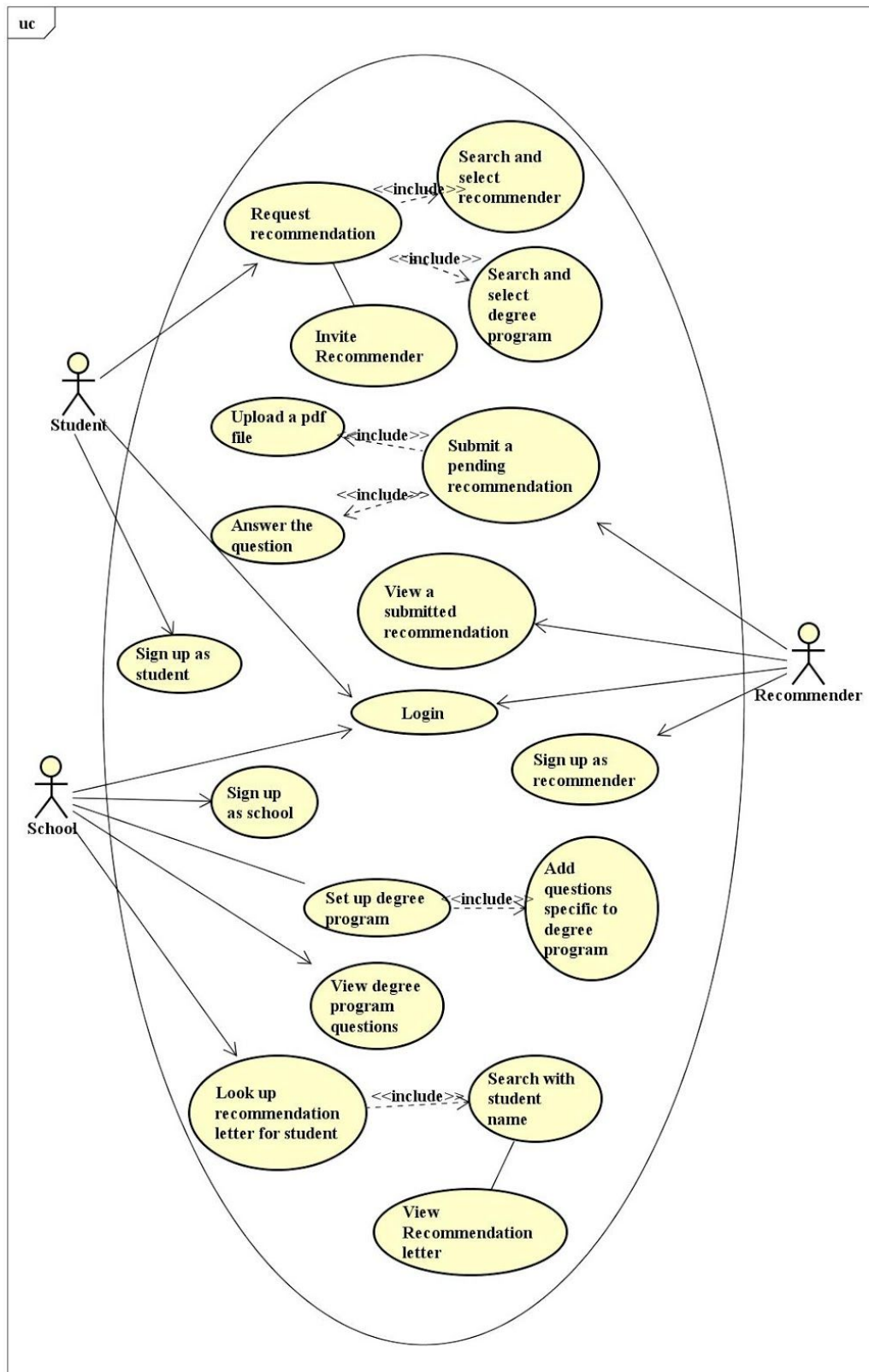


Figure 2: Recommendation letter system use case diagram

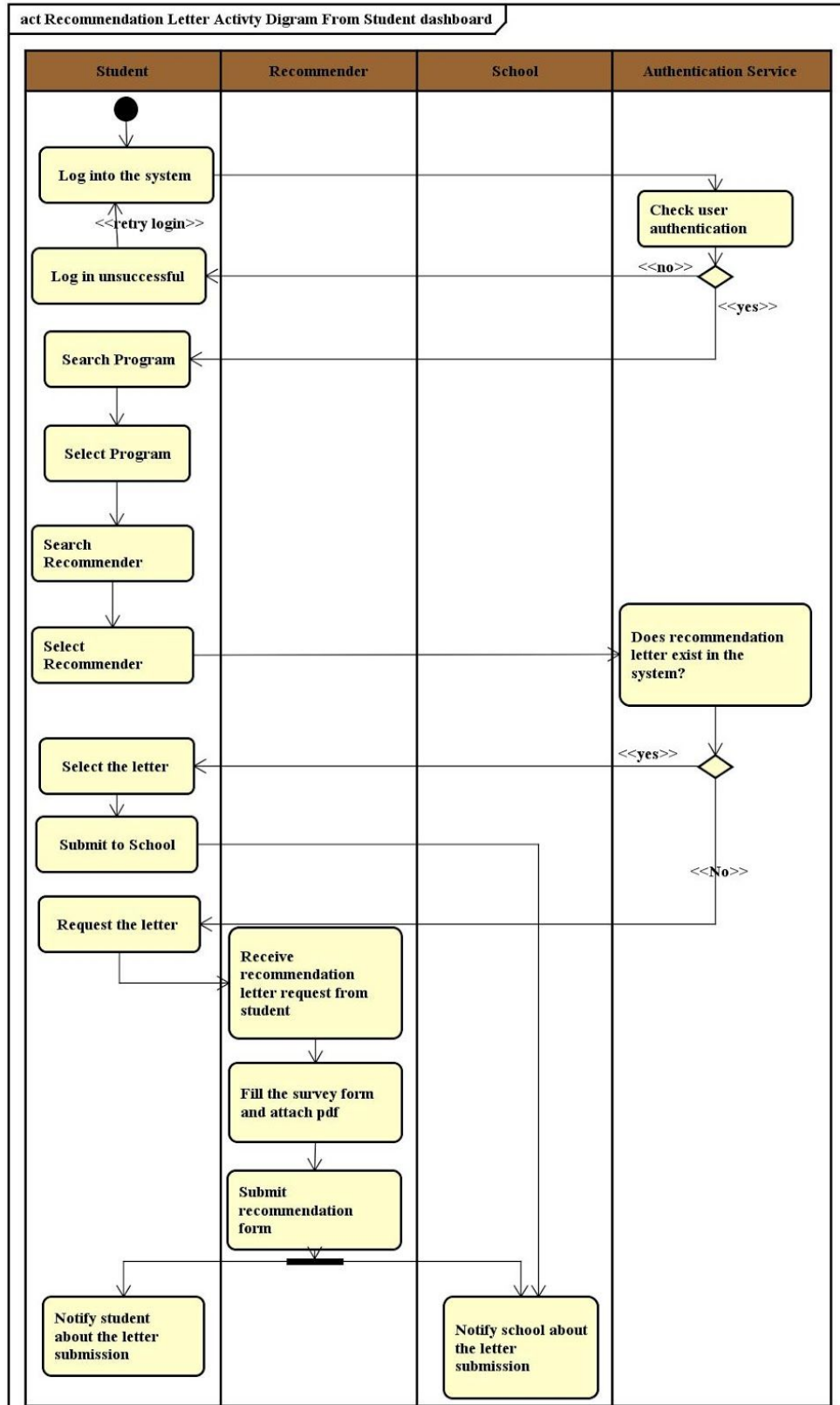


Figure 3: Student dashboard activity diagram

2.3. Recommender dashboard activity diagram

Figure 4 presents the activity diagram in the context of a recommender dashboard. The recommender logs into the system and if authenticated selects a pending letter request from the student. A questionnaire form is then presented to the recommender. The list of questions is determined by the degree program the student is seeking a recommendation for. The recommender answers the questions, uploads a pdf letter and submits the recommendation form. The school and student receive notification of letter submission and their respective dashboards are updated.

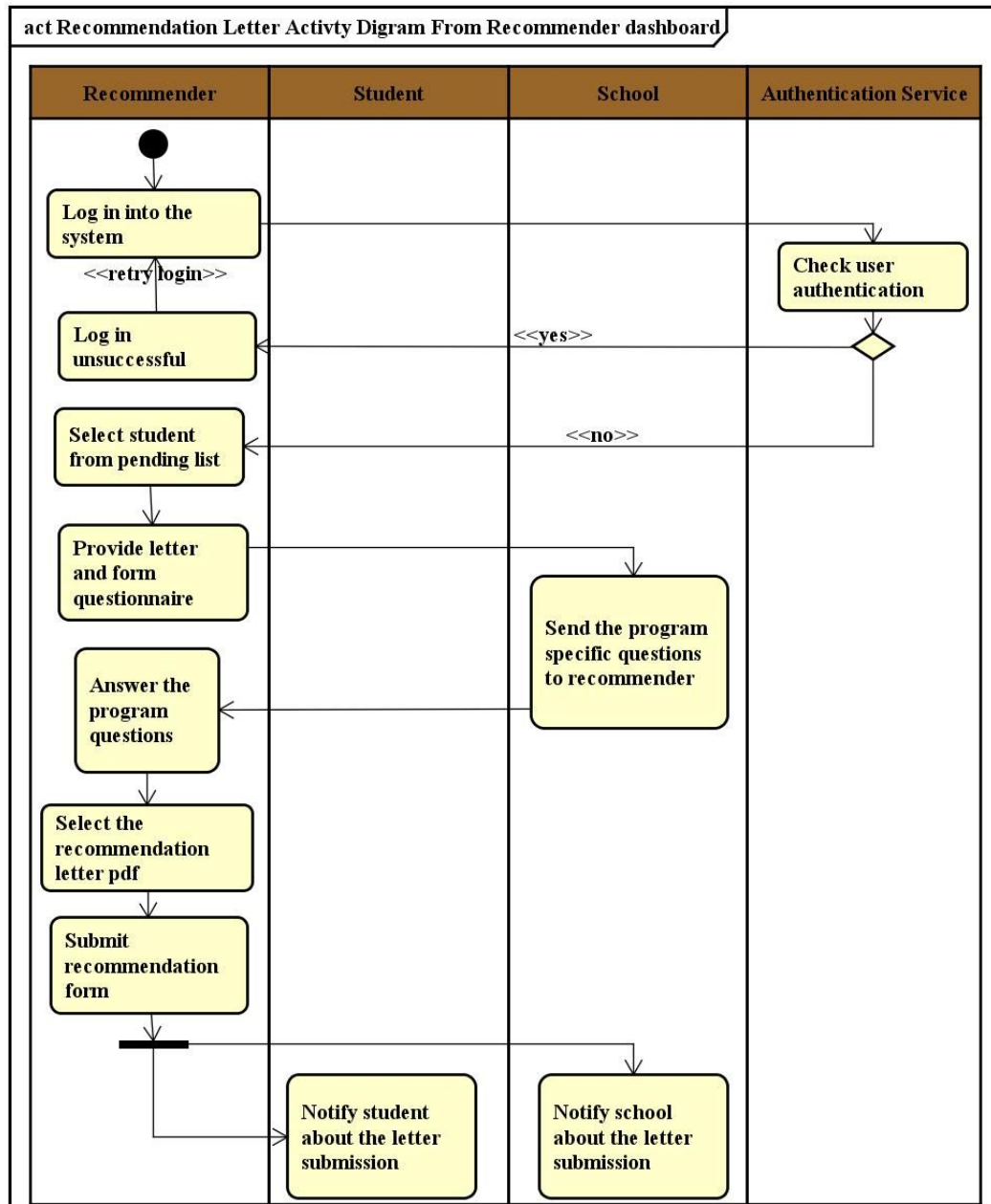


Figure 4: Recommender dashboard activity diagram

2.4. Blockchain Smart Contracts Programming

An important design decision is about the data that needs to be stored on the blockchain. As the core data object, the recommendation letter itself needs to be stored on the blockchain to achieve immutability. Since storing and accessing large data such as recommendation letter in blockchain is expensive, only the IPFS hash of the letter is stored with the contract. The physical recommendation letter file is stored in IPFS distributed database. In addition, to make every recommendation record trackable and verifiable, each letter request is also recorded on the blockchain. A letter request contract has the information about the requesting student, the recommender and the associated school program.

To record a recommendation letter object on the ethereum blockchain, a programming language called Solidity [5] is used to write contracts which are persisted on the blockchain. A contract is a program that runs on blockchain to set the rules that every participant has to follow.

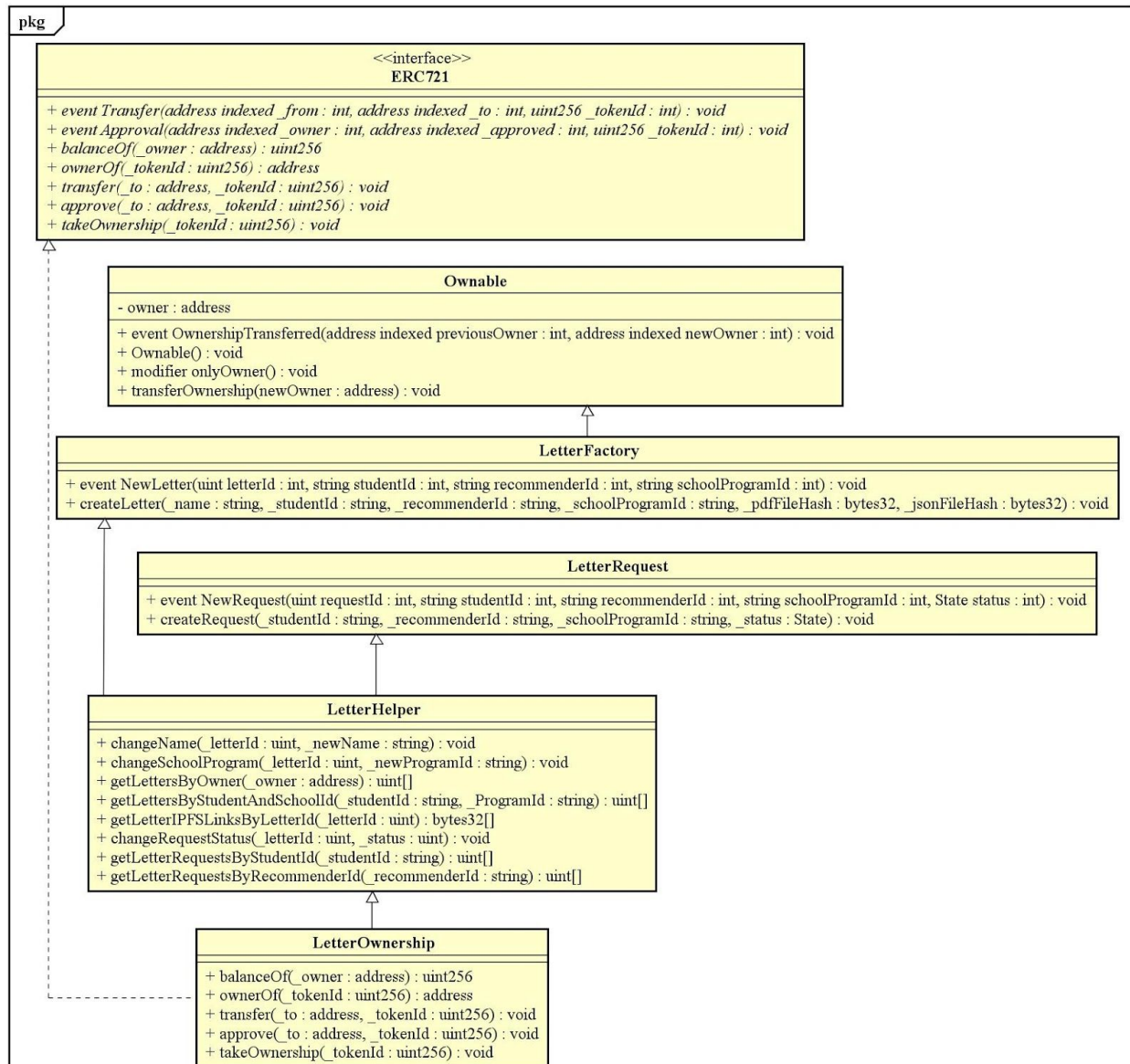


Figure 5: Blockchain contracts class diagram

A solidity contract acts similarly to classes in object oriented programming. The language provides the capability of defining interfaces and maintaining relationships between the contracts.

Figure 5 presents the class diagram of contracts deployed in the ethereum blockchain for this project. The description of contracts is as follows:

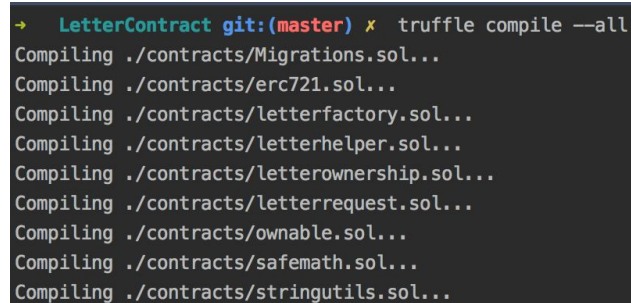
- ERC721 is non-fungible blockchain token interface, which provides functions for unique tokens to be managed, owned and traded.
- LetterOwnership contract implements the interface functions e.g. balanceOf(), ownerOf(), transfer() etc. This contract also inherits from LetterHelper contract.
- LetterFactory class has createLetter() function to generate a letter object. An event NewLetter is dispatched upon successful creation of the recommendation letter on blockchain. This event is listened on the client side javascript function which initiates the createLetter() on the deployed contract in the ethereum.
- LetterRequest class creates a Recommendation request object. An event NewRequest is dispatched upon successful creation of the recommendation request on blockchain. This event is listened on the client side javascript function which initiates the createRequest() on the deployed contract in the ethereum.
- LetterHelper class provides utility functions to interact with the other contracts. These utility functions are called from the javascript client using ethereum web3js library. Few important utilities in this contract are: getLetterIPFSLinksByLetterId() to retrieve the IPFS hash reference for a given letter Id, changeRequestStatus() to change the letter request status of a given letterId from pending to created when a recommender submits the recommendation.

Few other external contract libraries are also utilized:

- Safemath.sol: To provide a safety check on mathematical operations, e.g. avoiding overflow on multiplication of two numbers.
- Ownable.sol: The Ownable contract has an owner address, and provides basic authorization control functions, this simplifies the implementation of "user permissions".
- StringUtils.sol: To provide comparison of two strings since solidity language does not provide string manipulation functions as in the other major programming languages.

2.5. Blockchain Smart contracts Compilation and Deployment

In this project, Truffle Suite is used to compile the solidity code as shown in Figure 6.



```
→ LetterContract git:(master) ✕ truffle compile --all
Compiling ./contracts/Migrations.sol...
Compiling ./contracts/erc721.sol...
Compiling ./contracts/letterfactory.sol...
Compiling ./contracts/letterhelper.sol...
Compiling ./contracts/letterownership.sol...
Compiling ./contracts/letterrequest.sol...
Compiling ./contracts/ownable.sol...
Compiling ./contracts/safemath.sol...
Compiling ./contracts/stringutils.sol...
```

Figure 6: Compilation of Smart Contract

Artifacts (ABI) of the compilation will be placed in build/contracts/ directory. ABI stands for application binary interface, which is a JSON file that contains a list of the contract's functions and arguments. An Ethereum account will use this ABI to hash the function definition so it can create the EVM bytecode. Truffle suite uses the network configuration settings defined in truffle.js file for the deployment of contracts in the Ethereum. *Figure 7* shows the output of 'truffle migrate' command. After the deployment of solidity code, the execution of the smart contract program is running on Ethereum immutably.

```

→ LetterContract git:(master) ✕ truffle migrate
Using network 'development'.

Running migration: 1_initial_migration.js
  Deploying Migrations...
  ... 0x42d7a46265ffd98d89fe9e2bd8868aa2aa97dfd779cfe713340f993a1908d6cc
  Migrations: 0x8cdaf0cd259887258bc13a92c0a6da92698644c0
Saving artifacts...
Running migration: 2_deploy_contracts.js
  Deploying StringUtils...
  ... 0x5aa0291c588d11d3a3562419062b475ae369e5692af0280760360d911b92e2ef
  StringUtils: 0xf12b5dd4ead5f743c6baa640b0216200e89b60da
  Deploying Ownable...
  ... 0xcc6c4ae594f702b773559d7159288843b845948694a72f656e12c89d2e339112
  Ownable: 0x345ca3e014aaf5dca488057592ee47305d9b3e10
  Deploying SafeMath...
  ... 0x8d173d9559f4266b68c17c2e0f23022c0685ef03219576856a8086bd5b29e9f8
  SafeMath: 0xf25186b5081ff5ce73482ad761db0eb0d25abfbf
  Linking StringUtils to LetterHelper
  Linking StringUtils to LetterOwnership
  Deploying LetterFactory...
  ... 0x1847ec40ce64fa554ea6ab71028bd772547aec8bdfad82eae2ccac6dce53edf7
  LetterFactory: 0x8f0483125fcb9aaefa9209d8e9d7b9c8b9fb90f
  Deploying LetterRequest...
  ... 0x52457f953c2a4a4fb15eefbfe9466f4d77eeadb9a78cb5e0113603a12db0ecd4
  LetterRequest: 0x9fbda871d559710256a2502a2517b794b482db40
  Deploying LetterHelper...
  ... 0xb4dff2c1352e4cd2a0d1d43753b1a67012850cf7029d742fd500c76ddcb9b377
  LetterHelper: 0x2c2b9c9a4a25e24b174f26114e8926a9f2128fe4
  Deploying LetterOwnership...
  ... 0x49d9533c1a1e6688d33bde066427e120f56d88dbd2983ebf8ae3bd3d8983e93c
  LetterOwnership: 0x30753e4a8aad7f8597332e813735def5dd395028
Saving artifacts...

```

Figure 7: Deployment of Smart Contract

Each contract is deployed to the Ethereum network with a unique Ethereum address which is called the contract address. To make function calls you first look up the contract by its address and then make the call. Some of the function calls which do not change the state of the contract do not charge any gas cost(i.e. Ether in ethereum network) to the user. However, if a method changes the state of the contract (e.g. changing the recommendation letter request status from pending to created), the call to the contract method will need the gas amount being charged to the user with its account address. The code snippet is shown in *Figure 8* for the contract method changeRequestStatus(). If the provided gas parameter amount is not sufficient, Ethereum returns an error to the user as 'Out of gas limit'. The account address to invoke the method (e.g. accounts[0] in the code snippet) is registered account in the Ethereum network. In this project few default accounts are created at the start of Ethereum client in test network.


```

letterOwnershipContract.deployed()
  .then(function(instance) {
    instance.changeRequestStatus(newLetterId, 1, {
      from: accounts[0],
      gas: gasEstimateForTransaction
    }).then(function(reqStatusResult) {...})
  })

```

Figure 8: Contract method call with gas usage

2.6. Service Layer - Recommendation Letter Request Controller

Our service layer is designed to get the data from blockchain and interact with the rest of the app. It depends on the smart contracts already having been deployed to Ethereum. The letter request controller uses the web3 library to interact with the letter contracts that is on the blockchain. The Javascript version of this library called web3.js is used to interact with an Ethereum node using a HTTP or IPC connection.

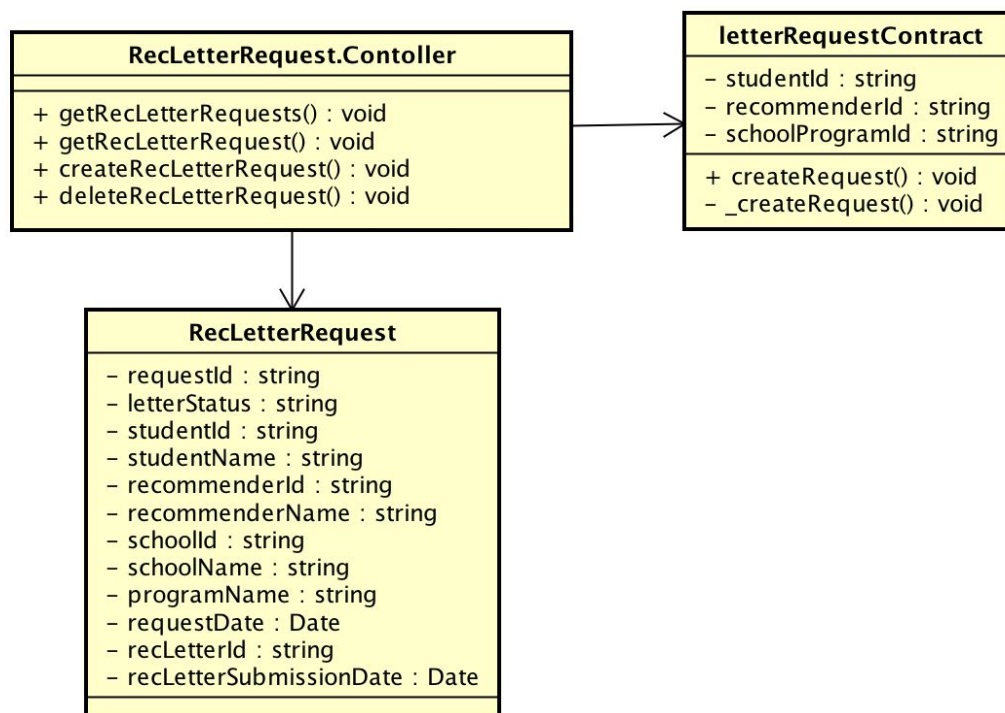


Figure 9: Letter request class diagram

2.7. Service Layer - Recommendation Letter Controller

To create a recommendation letter, the `createRecLetter` service handler uses web3 to invoke the `createLetter` function on the LetterFactory contract and then stores the letter content to IPFS to save gas cost, with the IPFS hash stored within the contract.

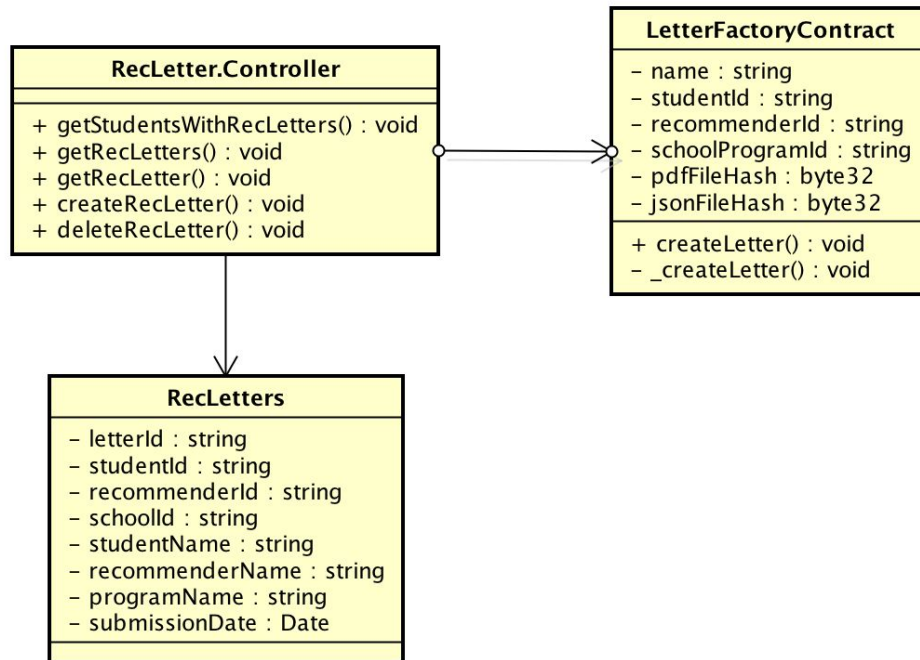


Figure 10: Recommendation Letter class diagram

2.8. Non-blockchain Component Class Diagram

We use MongoDB to store associated data such as Degree Programs and Candidate Questions.

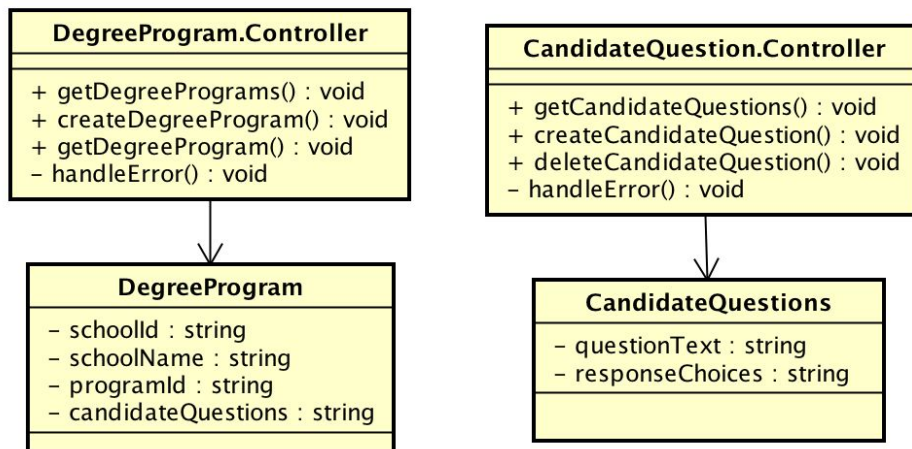


Figure 11: DegreeProgram and CandidateQuestion class diagram

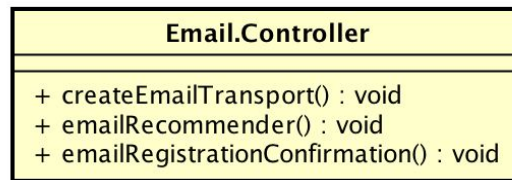


Figure 12: Email Service class Diagram

2.9. User Authentication Service

Blockchain has great potential to be used in identity management system, we have looked into a blockchain-based user management system called uPort. However, due to time and resource constraints, we chose to implement user authentication in Express.

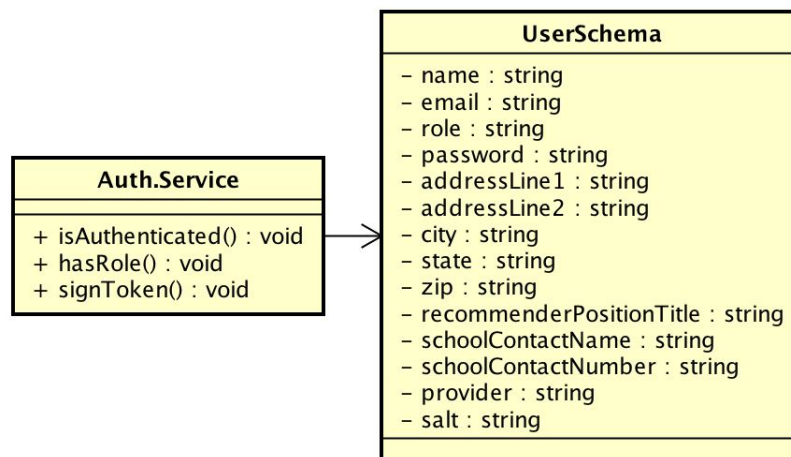


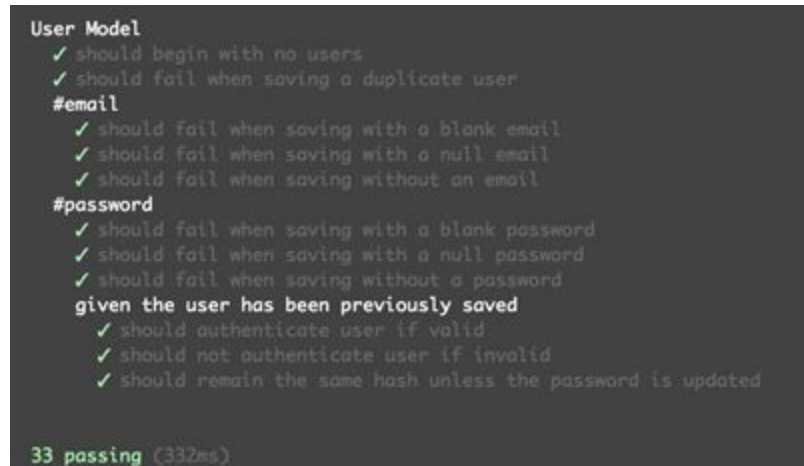
Figure 13: Authentication Service Class Diagram

The student, recommender, and school share the same user schema, the “role” attribute in the user schema identifies if the user is a student, recommender or school.

Part 3 Testing Results

3.1. Unit testing and end to end testing

The JavaScript framework Jasmine was used to create end to end tests as well as unit tests. The unit tests include testing whether pages have certain elements, including forms and inputs. The end to end tests involve executing JavaScript functions used in the angular controllers, and expecting the browser to respond accordingly, whether that includes page or page location changes.



```
User Model
✓ should begin with no users
✓ should fail when saving a duplicate user
#email
✓ should fail when saving with a blank email
✓ should fail when saving with a null email
✓ should fail when saving without an email
#password
✓ should fail when saving with a blank password
✓ should fail when saving with a null password
✓ should fail when saving without a password
given the user has been previously saved
  ✓ should authenticate user if valid
  ✓ should not authenticate user if invalid
  ✓ should remain the same hash unless the password is updated

33 passing (332ms)
```

Figure 14: User interface unit testing

3.2. API testing by postman

Postman is used to run a suite of tests on the server API. The tests are organized according how different API combinations are called to build UI screens. The postman testing scripts allow the values/parameters to be passed front one test to another, allowing us to test whether the APIs are property authenticating the users as well as to test the functionality within the context of a user request flow. The following screenshots provide some of our testing results.

11 PASSED	0 FAILED	UserLoginTests	No Environment	just now	Run Summary >
Iteration 1					
POST	User Login	http://localhost:3000/aut...	UserLoginTests / User Login	200 OK	259 ms 207 B
PASS	Valid HTTP Code				
PASS	Contains Auth Token				
GET	Get Logged-In User Details Without Auth Cookie	http://localhost:3000/api/...	... User Details Without Auth Cookie	401 Unauthorized	111 ms 1.256 KB
PASS	Valid HTTP Code				
GET	Get Logged-In User Details	http://localhost:3000/api/...	...Tests / Get Logged-In User Details	200 OK	29 ms 311 B
PASS	Valid HTTP Code				
PASS	Email Matches				
POST	User Login Missing User ID	http://localhost:3000/aut...	...Tests / User Login Missing User ID	401 Unauthorized	26 ms 33 B
PASS	Valid HTTP Code				
PASS	Valid Response Message				
POST	User Login Missing Password	http://localhost:3000/aut...	...sts / User Login Missing Password	401 Unauthorized	19 ms 33 B
PASS	Valid HTTP Code				
PASS	Valid Response Message				
POST	User Login Incorrect Credentials	http://localhost:3000/aut...	... / User Login Incorrect Credentials	401 Unauthorized	51 ms 66 B
PASS	Valid HTTP Code				
PASS	Valid Response Message				

Figure 15: Postman test script collection results for user login APIs

18 PASSED	0 FAILED	SchoolDashboardTests	No Environment	just now	
Iteration 1					
POST	School Login	http://localhost:3000/aut...	...oolDashboardTests / School Login	200 OK	257 ms 205 B
PASS	Valid HTTP Code				
PASS	Contains Auth Token				
GET	Get School Details Without Auth Cookie	http://localhost:3000/api/...	...chool Details Without Auth Cookie	401 Unauthorized	122 ms 1.256 KB
PASS	Valid HTTP Code				
GET	Get School Details	http://localhost:3000/api/...	...shboardTests / Get School Details	200 OK	36 ms 331 B
PASS	Valid HTTP Code				
PASS	Email Matches				
PASS	Role Matches				
GET	Get School Programs Without Auth Cookie	http://localhost:3000/api/...	...ol Programs Without Auth Cookie	401 Unauthorized	275 ms 1.256 KB
PASS	Valid HTTP Code				
GET	Get School Programs	http://localhost:3000/api/...	...boardTests / Get School Programs	200 OK	29 ms 1.726 KB
PASS	Valid HTTP Code				
PASS	School Id Matches				
PASS	School Name Matches				
GET	Search Students With Rec Letters Without Auth Cookie	http://localhost:3000/api/...	...h Rec Letters Without Auth Cookie	401 Unauthorized	106 ms 1.256 KB
PASS	Valid HTTP Code				
GET	Search Students With Rec Letters	http://localhost:3000/api/...	... / Search Students With Rec Letters	200 OK	54 ms 330 B
PASS	Valid HTTP Code				
PASS	Student Name				
PASS	School Id Matches				

Figure 16: Postman test script collection results for school dashboard related APIs

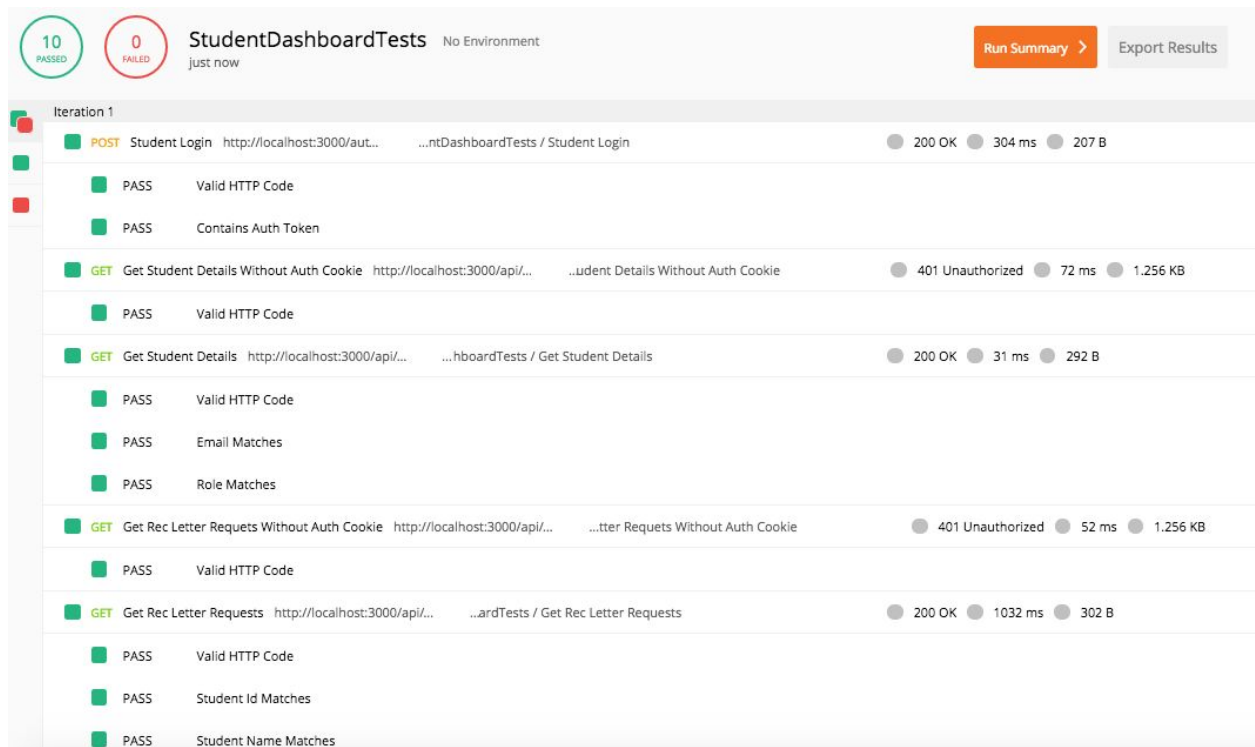


Figure 17: Postman test script collection results for student dashboard related APIs

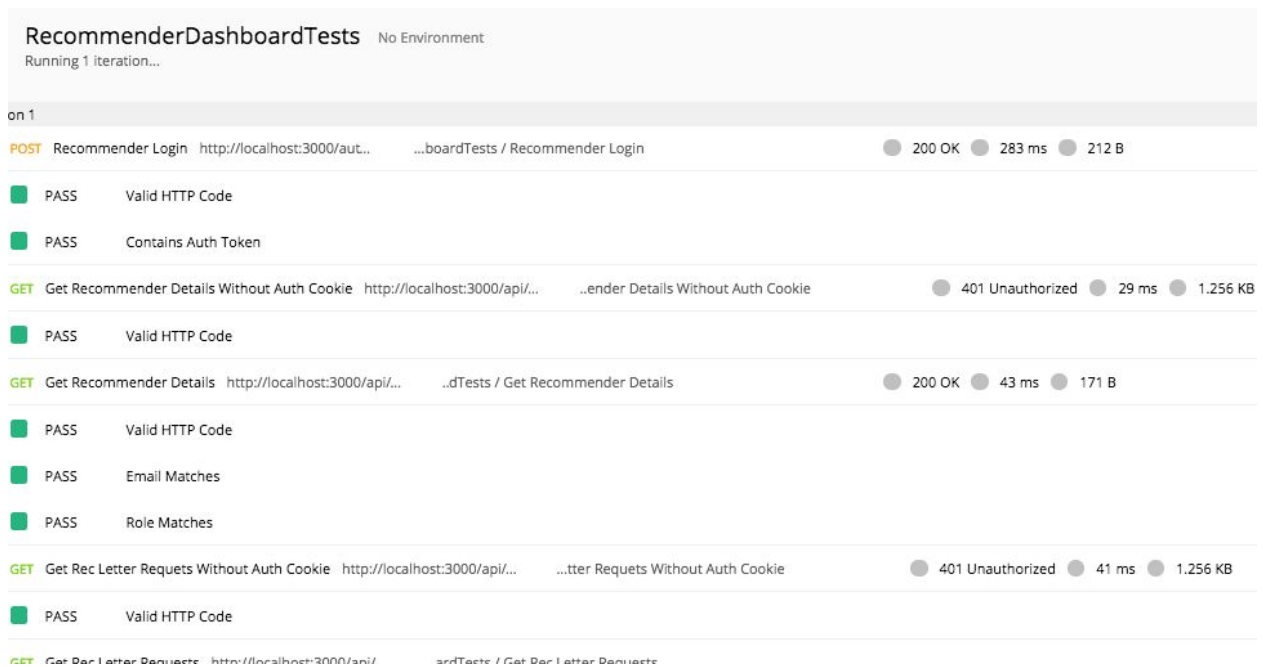


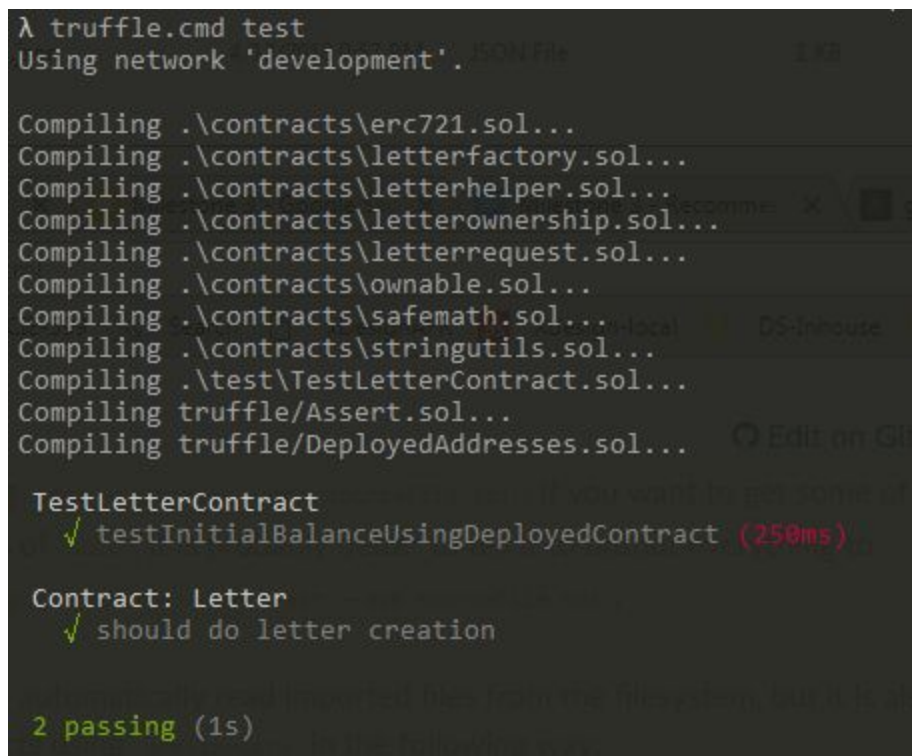
Figure 18: Postman test script collection results for recommender dashboard related APIs

3.3. Blockchain contract unit testing

Truffle suite [4] provides the framework to test the blockchain contracts using javascript and solidity [5] languages. In this project, the tests perform key functionality testing of blockchain contracts to ensure the integrity and robustness of the backend system.

Truffle framework uses Mocha [6] testing framework and Chai [7] for assertions in JavaScript tests. Each test uses contract() function which makes sure that all the contracts are redeployed in the running ethereum client so that the tests are run in clean contract state. The contract() function also has access to the accounts in the running ethereum client.

The current unit test in the system uploads two files in the IPFS, and using the hash of the saved documents LetterOwnership contract method createLetter() is executed using dummy studentId, recomemnderId and schoolId. Upon successful creation of the letter, the letterId is queried associated to same studentId and schoolid using getLettersByStudentAndSchoolId(). Another test for the validity of IPFS hashes in blockchain is tested using getLetterIPFSLinksByLetterId() which retrieves the document hashes for the given letterId. *Figure 19* shows the result of unit tests run.

A terminal window showing the execution of Truffle tests. The command 'truffle.cmd test' is run, and it uses the 'development' network. It lists the compilation of various Solidity contracts: .\contracts\erc721.sol, .\contracts\letterfactory.sol, .\contracts\letterhelper.sol, .\contracts\letterownership.sol, .\contracts\letterrequest.sol, .\contracts\ownable.sol, .\contracts\safemath.sol, .\contracts\stringutils.sol, .\test\TestLetterContract.sol, truffle\Assert.sol, and truffle\DeployedAddresses.sol. The test results for 'TestLetterContract' are shown: 'testInitialBalanceUsingDeployedContract' passes in 250ms, and 'should do letter creation' passes. The final result is '2 passing (1s)'.

```
λ truffle.cmd test
Using network 'development'.

Compiling .\contracts\erc721.sol...
Compiling .\contracts\letterfactory.sol...
Compiling .\contracts\letterhelper.sol...
Compiling .\contracts\letterownership.sol...
Compiling .\contracts\letterrequest.sol...
Compiling .\contracts\ownable.sol...
Compiling .\contracts\safemath.sol...
Compiling .\contracts\stringutils.sol...
Compiling .\test\TestLetterContract.sol...
Compiling truffle\Assert.sol...
Compiling truffle\DeployedAddresses.sol...

TestLetterContract
  ✓ testInitialBalanceUsingDeployedContract (250ms)

Contract: Letter
  ✓ should do letter creation

2 passing (1s)
```

Figure 19: Blockchain smart contract unit testing

The code coverage of solidity contract is computed using sol-coverage module which executes each unit test in clean environment. As shown in *Figure 20*, The test in our project covers 100% of the solidity contracts except the library functions.

all files contracts/

100% Statements 67/67 53.33% Branches 16/30 100% Functions 20/20 100% Lines 69/69

File	Statements	Branches	Functions	Lines
letterfactory.sol	100%	4/4	100%	0/0
letterhelper.sol	100%	48/48	53.57%	15/28
letterownership.sol	100%	12/12	50%	1/2
letterrequest.sol	100%	3/3	100%	0/0

Figure 20: Blockchain smart contract test coverage

3.4. Integration unit test coverage

Test coverage is calculated measured using the Istanbul 2.0 API, also known as nyc. The nyc javascript library works very well in conjunction with LCOV, a basic graphical front-end tool. Nyc measures which lines are uncovered by the test suite. Currently 668 of 1098 (60.84%) relevant lines are covered, but this number does not account for the coverage of our additional postman tests. Nyc is also run on coveralls, a web service that helps code coverage.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	60.54	30.13	65.22	60.84	
src	100	100	100	100	
mocha.conf.js	100	100	100	100	
mocha.global.js	100	100	100	100	
src/server	92.86	66.67	50	92.86	
app.js	92	100	66.67	92	18,19
index.js	100	66.67	100	100	4,6
routes.js	92.31	100	0	92.31	31
src/server/api	63.64	0	0	63.64	
web3helper.js	63.64	0	0	63.64	... 29,31,36,37,38
src/server/api/candidatequestion	93.06	55.56	96.61	95.04	
candidatequestion.controller.js	83.33	66.67	87.5	83.33	15,36,47
candidatequestion.integration.js	91.3	50	100	95.45	76,118,166
candidatequestion.spec.js	100	100	100	100	
index.js	100	100	100	100	
index.spec.js	95	100	90.91	95	18

Figure 21: Nyc run locally

LAST BUILD ON BRANCH MASTER	BRANCH: MASTER
COMMITTED 1 MAY 2018 - 0:02	COVERAGE INCREASED (+3.02%) TO 54.214%

Figure 22: Nyc run on coveralls

3.5. Continuous Integration

Travis CI is the free distributed continuous integration service used to run the Jasmine front-end and back-end tests continuous. It is integrated into both GitHub and Slack. When any branch of Git is pushed to GitHub, Travis CI runs the entire test suite, and returns whether the build passes or fails to the communication tool Slack.

Default Branch

✓ master	# 33 passed	3a85579	✓	✓	✓
28 builds	about 24 hours ago	Kevin Sun			

Figure 23: Travis CI showing master branch's passing build

Part 4. Development Process & Lessons Learned Reflection

4.1. Meeting the requirements

4.1.1. Requirements Overview

The functionality of the recommendation letter system is presented to the user as a web interface. The application must have screens that support the basic flow of students requesting recommendations, recommenders submitting recommendations, and schools viewing recommendations, along with other supplemental functionality. Below is a list of functional requirements broken down by screen. Please see Appendix 5.1 for screenshots.

Landing Page

This is the page users see when they first visit the web site.

Requirements	Status
Be able to navigate to login page	Completed
Be able to navigate to user registration for students, recommenders and schools	Completed

User Login

This page allows a valid user to login. All three types of users (students, recommenders, and schools) use this screen.

Requirements	Status
A valid username and password should login and redirect to the appropriate dashboard	Completed
Incorrect credentials result in an error	Completed
After logging in the user's name should be displayed in the header	Completed
Authentication and authorization work across the app, users can't access screens or data they don't have access to	Completed

Student Registration

New student users sign up using this page.

Requirements	Status
--------------	--------

Has forms that allow the student to enter their name, email, address and password	Completed
Duplicate email addresses are not allowed	Completed
The password confirmation box gives a validation error if the password is entered incorrectly	Completed
The “Sign Up” button registers the student user and then redirects to the student dashboard.	Completed
An email is sent to the new user to confirm their email address.	Partial

Student Dashboard

This page is a view of the student’s recommendation requests and their statuses.

Requirements	Status
List all the recommendation requests of the currently logged in student.	Completed
Give the status of these requests (pending, submitted).	Completed
Be able to navigate to the request recommendation screen.	Completed

Request Recommendation

Students can make new recommendation requests from here.

Requirements	Status
The recommenders are listed in a drop down.	Completed
The degree programs are listed in a drop down.	Completed
All dropdowns have autocomplete functionality.	Not Done
An invite email can be sent to recommenders not in the system.	Completed

Recommender Registration

New recommenders sign up using this page.

Requirements	Status
Has forms that let the recommender enter their name, email, position, address,	Completed

and password	
Duplicate email addresses are not allowed	Completed
The password confirmation box gives a validation error if the password is entered incorrectly	Completed
The “Sign Up” button registers the recommender user and then redirects to the recommender dashboard.	Completed
An email is sent to the new user to confirm their email address.	Partial

Recommender Dashboard

This page is a view of a recommender’s incoming requests and a history of their completed recommendations.

Requirements	Status
A message should indicate when then there’s no pending recommendations.	Not Done
Be able to navigate to submit pending recommendations	Completed
List completed recommendations	Completed
View completed recommendations	Completed

Submit Recommendations

The recommender uploads a pdf recommendation letter file and fills out an application questionnaire on this screen.

Requirements	Status
Browse and upload a pdf file	Completed
Not uploading the pdf results in a validation error	Completed
List multiple choice questions according to degree program	Completed
Questions are optional or mandatory.	Not Done
Students are sent a notification email upon submission of recommendation.	Not Done

School Registration

New school users sign up using this page.

Requirements	Status
Has forms that let the school contact enter the name of the school, a contact email, name, phone number, address and password.	Completed
Duplicate email addresses are not allowed	Completed
The password confirmation box gives a validation error if the password is entered incorrectly	Completed
The “Sign Up” button registers the school user and then redirects to the school dashboard.	Completed
An email is sent to the new user to confirm their email address.	Partial

School Dashboard

This is a view of degree programs and student recommendations.

Requirements	Status
The list of existing degree programs is shown	Completed
Navigate to create new degree program	Completed
Existing programs can be edited or deleted.	Not Done
Filter recommendation letters by student.	Completed

Setup Degree Program

Create a new degree program.

Requirements	Status
Can enter program name	Completed
Can select questions to use	Completed
Questions can be set as optional or mandatory.	Not Done
Can configure whether or not students can see recommendations associated with this degree program.	Completed

View Recommendation

The screen to actually view recommendations. Accessible in different contexts to the student, recommender and school.

Requirements	Status
Show the pdf file associated with the recommendation	Complete
Show the questions and answers associated with the recommendation	Complete

Besides the functional requirements, this project was also initiated with the goal of exploring blockchain related technologies. This project was successfully able to use blockchain to store and retrieve recommendation letters.

4.1.2. Completion of Requirements

Our customer signed off on our current build of the application as satisfactorily meeting the requirements. But as listed above there were several requirements that were either only partially completed or were not completed at all. As we got closer to the end of our available time the customer made it clear that the core flow of the application along with blockchain functionality were the most important things to focus on. To that end we ignored some user experience niceties - like having a label when the pending recommendation inbox is empty or having email alerts and a complete email confirmation step on user registration. We spent our time polishing the full flow of the recommendation letter system, especially focusing on getting the blockchain pieces done in a way that was both functionally correct and well-designed.

4.1.3. Development Process

Our project had some key differences from a traditional software capstone. We had a technology mandate built into our project proposal (use and learn about blockchain) and our actual use case of students seeking recommendation letters was not directly driven by the customer. Instead it was a use case meant to be challenging enough to have us apply this technology in an interesting way while still being simple enough to complete within a semester. Our development process reflected these unique constraints.

The first several weeks of the project were largely spent on analyzing our blockchain technology choices as well as determining if the project was feasible at all. Only one teammate had prior expertise in blockchain so much of this time was also spent on getting the rest of the team up to speed with concepts involved. There are different blockchain implementations available to develop on and after considering several alternatives we decided on Ethereum. Since storing complete files could lead to transactions that are costly in terms of gas on Ethereum, we also chose to use IPFS, a distributed storage system. The rest of our stack, Angular, Express, and Mongo were used because they were mature, developer-friendly technologies for building web applications. The main Ethereum client library, also seemed better supported in JavaScript than in other languages which helped sway our decisions about this stack as well.

As we moved from research into development we initially tried to have a strict, tightly regulated process where we would have detailed stories in Pivotal Tracker, with regular meetings and updates about what is being worked on and progress updates for each teammate. The overhead of this approach was not something we were able to sustain. The items in Pivotal regularly fell out of date, and frequent meetings were also not something the busy team could keep up.

Eventually the process morphed into a looser more hands-off approach. The team had a shared understanding of missing functionality, and then each teammate would commit to a piece of the missing work that they were interested in working on. We would make sure that no one was overlapping and give each other assistance and progress reports through our Slack chat. We still had regular meetings every week, along with ad hoc meetings when we needed longer discussions about things. Our tasks were also defined in a much more fine-grained way. Whereas user stories would have detailed descriptions which were not necessarily accurate representations of the work that needed to be done, “to-do list” like tasks allowed us to focus on what work we needed to do without extra ceremony.

We were inconsistent with our automated testing for the beginning of the project. We had a few sporadic unit tests but for the most part we relied on manual testing, sometimes through the UI but mostly through the API testing tool Postman. Later on in the project we started focusing more on automated testing. We researched our testing options for smart contracts and were able to successfully build a test suite for our Solidity code. In the service layer we converted our manual Postman calls into actual more organized reusable integration test scripts. Additionally there are some Express unit tests that provide coverage over some basic things in our service layer. We also investigated making UI tests in Angular but decided that it was not the best use of our resources since the UI is of secondary importance to the core functionality.

Though our use case was somewhat artificial we still had a customer who we relied on to sign off on the requirements and give us feedback as the project went forward. We had regular meetings with our customer (every 2 weeks or so) and adjusted our project priorities according to his feedback. For example, for incomplete requirements we told to prioritize blockchain related functionality over UX enhancements. We were fortunate to have a customer that was enthusiastic about the project and made himself available when our team needed his assistance.

Though we did not follow any well known development methodology like Scrum or XP verbatim, we occasionally borrowed some practices from these approaches. Our regular update meetings were like a mesh of the daily standups and sprint plannings that occur in Scrum. For exploring new technologies we assisted each other regularly and pair programmed a bit in the beginning as well. For the most part our process was pragmatic. We tried to do what worked for the project and for the team. There was responsibility and trust placed on the teammates to do their best for the project.

4.1.4. Retrospective

Overall our customer was satisfied with the final product and on the whole it felt like we were able to add functionality and make changes to the project pretty smoothly. Because of learning curve around the technology and the non-traditional nature of the project as a whole, there was certainly a lot of discovery along the way. We were happy with most of our decisions but certainly we could have done better in some areas as well.

There were several aspects of the project we were pleased with. Using the MEAN stack (Mongo, Express, Angular, Node.js) gave us technology that was both mature and also very flexible. JavaScript's dynamic typing and Mongo's document oriented NoSql approach let us make changes quickly without having our technology get in the way. This stack was also something that we did not have much experience with on the team so we were pleased to successfully learn and utilize a new stack. The other exciting technology that we were able to learn on this project of course, was blockchain. The research and implementation around blockchain and Ethereum were definitely highlights of the project for all of us. And finally the hands-off approach with minimal process and high trust between teammates was something that we generally were happy with. It led to a sense of responsibility and ownership that was good for the project overall.

There were some other parts of the project that we would do differently if we could. We neglected automated testing until past the halfway point. Keepings tests in line with functionality earlier would have helped us catch bugs and keep the code cleaner. The hands off approach also had its downsides. We sometimes ran into situations where no one took ownership of some of the less interesting aspects of the application. Some of the UX requirements that were dropped may have been possible if we had a stricter process. One other thing we faced, that is hard to do differently, is that since people specialized in different areas of the stack (UI, services, blockchain) not everyone got to try all the most interesting stuff. But overall there were lots of satisfying challenges to go around.

4.2. Estimates

The estimated hours for various tasks are based upon the expert opinion drawn from within the team. *Table 1* lists down tasks and their associated estimated and actual man hours with estimated and actual delivery in terms of project milestone.

Project estimation at the start of the project:

- Number of resources: 6
- Number of hours per resource per week: 12.5
- Number of weeks available for development and testing: 9
- Total man hours available: 675

However, the number of resources were reduced to 5 after 2 weeks in the project. The modified estimated hours are as follows:

- Number of resources: 5
- Number of hours per resource per week: 12.5
- Number of weeks available for development and testing: 9
- Total man hours available: 562

Tasks	Estimated Man Hours	Actual Man Hours	Deliverables	Estimated Delivery	Actual Delivery
Dev. Environment Setup(6 machines)	16	30	All team members have local development environments up and running	Milestone 2	Milestone 2
Landing Page	8	10	The landing page UI screen is complete and unit-tested	Milestone 2	Milestone 2
User Login DB Design	10	10		Milestone 2	Milestone 2
User Login DB Access Wrapper	10	10		Milestone 2	Milestone 2
User Login REST Service	10	10		Milestone 2	Milestone 2
User Login Page	10	8	The user login page UI screen is complete and unit-tested	Milestone 2	Milestone 2
Student Registration DB Design	12	12		Milestone 2	Milestone 2

Student Registration DB Access Wrapper	12	12		Milestone 2	Milestone 2
Student Registration REST Service	10	16		Milestone 2	Milestone 2
Student Registration Page	6	12	The student registration page UI screen is complete and unit-tested	Milestone 2	Milestone 2
Recommender Registration DB Design	6	10		Milestone 2	Milestone 2
Recommender Registration DB Access Wrapper	6	10		Milestone 2	Milestone 2
Recommender Registration REST Service	6	16		Milestone 2	Milestone 2
Recommender Registration Page	10	16	The recommender registration page UI screen is complete and unit-tested	Milestone 2	Milestone 2
School Registration DB Design	6	12		Milestone 2	Milestone 2
School Registration DB Access Wrapper	6	12		Milestone 2	Milestone 2
School Registration REST Service	10	18		Milestone 2	Milestone 2

School Registration Page	10	12	The school registration page UI screen is complete and unit-tested	Milestone 2	Milestone 2
Degree Program Setup Page	10	12		Milestone 2	Milestone 2
Recommendation Letter DB Design	6	12		Milestone 2	Milestone 2
Recommendation Letter DB Access Wrapper	14	20		Milestone 2	Milestone 2
Recommendation Letter List REST Service	14	16		Milestone 3	Milestone 2
Recommendation Letter Details REST Service	14	14		Milestone 3	Milestone 2
Recommenders List REST Service	14	16		Milestone 3	Milestone 2
School & Program List REST Service	14	14		Milestone 3	Milestone 2
Student Dashboard Page	10	10	The student dashboard UI screen is complete and unit-tested	Milestone 3	Milestone 3
Recommender Dashboard Page	10	10	The recommender dashboard UI screen is complete and unit-tested	Milestone 3	Milestone 3

School Dashboard Page	10	10	The school dashboard UI screen is complete and unit-tested	Milestone 3	Milestone 3
Request Recommendation Letter REST Service	16	16		Milestone 3	Milestone 3
Request Recommendation Letter Page	16	12	The UI screen to request a recommendation letter is complete and unit-tested	Milestone 3	Milestone 3
Submit Recommendation Letter REST Service	21	24		Milestone 3	Milestone 3
Submit Recommendation Letter Page	10	20	The UI screen to submit a recommendation letter in PDF format is complete and unit-tested	Milestone 3	Milestone 2
View Recommendation Letter Page	10	20	The UI screen to view a recommendation letter and the PDF file is complete and unit-tested	Milestone 3	Milestone 2
Smart Contract Design	20	30	The design of a recommendation letters smart	Milestone 2	Milestone 3

			contract is complete		
Blockchain Access Wrapper	50	70	Blockchain access related classes are tested and available to use within the project	Milestone 2 & 3	Milestone 3
Email Wrapper	5	12	Class to send notifications is ready and unit-tested	Milestone 3	Milestone 3
Testing	36	55	Final testing is complete	Milestone 3	Milestone 3
Documentation	100	130		Milestone 3	Milestone 3
Total Hours	562	639			

Table 1: Project estimation table

4.3. Risks

4.3.1. Blockchain Technology Risk

- Blockchain technology is difficult to understand because it has many obscure concepts and acronyms. To mitigate the risk, we negotiated with our customer and take the conservative approach to under promise. Fortunately, our customer understood the blockchain technology risk that we faced and cooperated to adjust the functionality requirement regarding the UI.
- The development environment for blockchain is changing fast, many open-source library we depend on is not well tested. To mitigate the risk, we picked the blockchain technology stack called Ethereum because it is relatively the most stable version and has most community support.

4.4. Team Dynamic

The project team adopted the combination of Extreme Programming(XP) and agile methodology for software development. Each of the team member contributed to the code to fulfill the project requirements. Each of the team member was remotely located and some in different time zones. Some

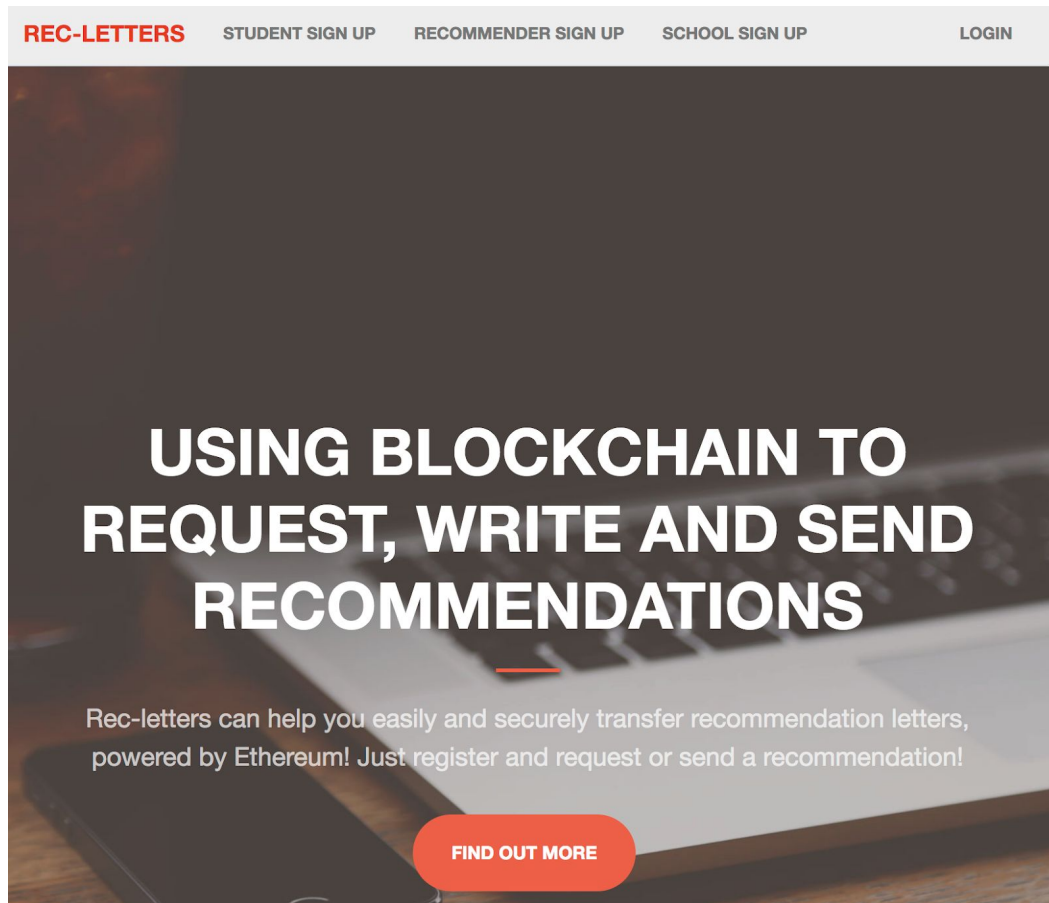
team members worked in pairs to brainstorm and clarify ideas. Pair programming helps in the situation when one team member is stuck either in setting up the environment, coding issue or understanding the requirement.

The team met every week to discuss the issues related to the design, requirements and setting up the short term goals. This helped in communication among team members to resolve any issues and concerns. Slack communication tool was utilized throughout the project duration with specific dedicated channels for discussions among team members and to receive notifications from github, travis CI and pivotal platforms. If a defect is found in the software, it is communicated to all the team members and a developer who is available and aware of the defective code corrects it with the addition of unit test for it. Travis CI tool is used to automate build and testing process. The code refactoring process was constantly adopted throughout project lifecycle to mitigate the code complexity and facilitate readability to find and fix the code defects.

Part 5. Appendix

A.1. Screenshots

A.1.1. *Landing Page*



A.1.2. *User Login*

Login

Email

Password

[LOGIN](#)

A.1.3. Student Registration

Student Sign Up

Name

Email

Address
Line 1

Address
Line 2
(Optional)

City

State

Zip

Password

Confirm
Password

SIGN UP

A.1.4. Student Dashboard

Student Dashboard					
Recommendations					
Date	School	Degree Program	Recommender	Status	View
2018-05-02T17:02:37.289Z	Harvard	Computer Science	Jay Bergeron	Pending	
2018-05-02T17:06:53.477Z	Stanford	Computer Science	Peter Henstock	Pending	
<div>REQUEST NEW RECOMMENDATION</div>					

A.1.5 Request Recommendation

REC-LETTERS
STUDENT DASHBOARD
REQUEST RECOMMENDATION

Request Recommendation

Select Recommender

Name

Peter Henstock

Don't see a recommender?

INVITE RECOMMENDER

Select Degree or School

Name

Harvard : Computer Science

REQUEST RECOMMENDATION

A.1.6. Recommender Registration

Recommender Sign Up

Name

Email

Position / Title
(Optional)

Address Line 1

Address Line 2
(Optional)

City

State

Zip

Password

Confirm Password

SIGN UP

A.1.7. Recommender Dashboard

Recommender Dashboard

Pending Recommendations

Request Date	Student	School	Degree Program	Submit
2018-05-02T17:23:47.868Z	Yingtian Wang	Stanford	Computer Science	<button>SUBMIT</button>

Submitted Recommendations

Student	Degree Program	Submitted On	View
Yingtian Wang	Computer Science	2018-05-02T17:23:15.106Z	

A.1.8. Submit Recommendations

Submit Recommendation Letter

Please upload the recommendation letter file in Adobe PDF format. Please also provide answers to the candidate questionnaire

Choose File recommendation.pdf

Please make appropriate selections for the applicant

In what capacity have you known the applicant?

- ☐ Large Class / Lecture
- ☐ Small Class / Seminar/ Lab
- ☐ Independent Study Student
- ☐ Advisee
- ☐ Other

How well do you know the applicant?

- ☐ See on a regular basis
- ☐ At one time
- ☐ Knew him/her only through in-class contact
- ☐ Only through records

SUBMIT

A.1.9. School Registration

School Sign Up

Name

Email

Contact Person

Contact Number

Address Line 1

Address Line 2 (Optional)

City

State

Zip

Password

Confirm Password

SIGN UP

A.1.10. School Dashboard

Degree Programs

ADD

Program Name	View
Computer Science	
Biology	
English	
Psychology	
History	

Recommendation Letter Lookup

Yingtian Wang

GO!

Student	Recommender	Degree Program	View
Yingtian Wang	Peter Henstock	Computer Science	

A.1.11. Setup Degree Program

Add a Degree Program for Harvard

Create Program

Program Name

Software Engineering

☒ Recommendations are visible to students.

Please select the candidate questions which need to be answered for by a recommender

Question Text

☒ Quantitative ability

Response Choices

Upper 50%

Lower 50%

Upper 25%

Upper 10%

No basis for judgment

☒ Flexibility

Upper 50%

Lower 50%

Upper 25%

Upper 10%

No basis for judgment

A.1.12. View Recommendation

View Recommendation Letter

In what capacity have you known the applicant?

Large Class / Lecture

How well do you know the applicant?

See on a regular basis

Microsoft Word - Sample letter of recommendation.docx

1 / 2



Sample letter of recommendation

[University letterhead]

[sender's name]
[sender's departmental address—if not printed on letterhead]
[sender's departmental phone number, if available]
[sender's departmental fax number—if not printed on letterhead]
[sender's institutional email address]

[today's date]

[recipient's name]
[recipient's institutional address]

Dear [recipient's name]; or To Whom it May Concern:

It is my pleasure to recommend Jane Doe for admission to [name of program] at [name of university]. I am a fifth-year Ph.D. student at the University of California, Berkeley. I came to know Jane when I was her Graduate Student Instructor for Philosophy 111: Ethical Relativism, taught by Professor John Smith. The course comprised [short description of course]. Jane distinguished herself by submitting an exceptionally well researched and interesting project on ethical practices in ancient Greece. I would rank her in the top 2% of students that I have taught in the past five years in respect of her writing ability and research skills.

A.2. User Stories

As a student, I want to register as a student so that I can request a letter

As a student, I want to select what school and degree program so that I can request a letter

As a student, I want to select what recommender to so that I can request a letter

As a student, I want to see a list of my request so that I know the my request status

As a recommender, I want to register as a recommender so that I can submit a letter

As a recommender, I want to view a list of request so that I can pick a request and recommend..

As a recommender, I want to upload a letter as a PDF so that I can submit a letter

As a recommender, I want to answer a list of questions about the candidate so that I can submit the recommendation with the letter.

As a school, I want to add a degree program so that students can pick the program

As a school, I want to select the candidate questions to be answered for a degree program so that recommender can answer those questions for the candidate.

As a school, I want to view the recommendation letter that is submitted so that I can process the admission process.

As a school, I want to search the recommendation letter so that I can process the admission process.

A.3. Developer Installation Manual

To run this code locally you will need to have the following prerequisites installed:

Git

Install git and clone the repo located here: <https://github.com/vmishra22/cscie599bc/tree/master/src>

MongoDB

Install MongoDB from the official website: <https://www.mongodb.com/>

Add the executable to your path, confirm that you can run “mongod” to start the server. You can leave all configuration as defaults.

Ethereum Client (Geth or Ganache)

These two clients will let you setup local, private blockchains for testing. We recommend using the Ganache UI for a simpler approach to hosting a local blockchain (<http://truffleframework.com/ganache>) .

Another approach is to use the go ethereum client (geth), this CLI will give you more flexibility than Ganache but may be more difficult to setup:

<https://github.com/ethereum/go-ethereum/wiki/Building-Ethereum>

IPFS

Install IPFS (<https://ipfs.io/docs/install>), confirm that you can run “ipfs daemon” successfully. You may need to update the IP and port of IPFS in the configurations (see below).

Node.js and npm

Install the latest stable release of node.js from <https://nodejs.org/en/>

Confirm that command line paths are setup correctly and that “npm” is accessible through the command line.

Once npm is installed, also install the following:

- In the *src* folder of the repo, run “npm i” to install all local dependencies.
- Install **gulp**, by running “npm install --global gulp”
- Install **truffle** by running “npm install -g truffle”
- The latest install of truffle (as of this writing) has an out of date solidity compiler. You will need to fix this by going into the directory of the truffle package and updating the solidity version. See here for an example: <https://ethereum.stackexchange.com/a/26485/34019>

A.4. Full Jasmine Test Suite

```
CandidateQuestion Model
Express server listening on 9000, in test mode
✓ should begin with no questions
✓ should fail when saving a duplicate question
#questionText
✓ should fail when saving with a blank question text
#responseChoices
✓ should not fail if there are no response choices
✓ should fail when saving with a blank response choice

Candidate Question API Router:
✓ should return an express router instance
GET /api/CandidateQuestions
✓ should verify authenticated and route to candidatequestion.controller.getCandidateQuestions
DELETE /api/CandidateQuestion/:id
✓ should verify authenticated and route to candidatequestion.controller.deleteCandidateQuestion
POST /api/CandidateQuestion
✓ should route to candidatequestion.controller.createCandidateQuestion

Thing API Router:
✓ should return an express router instance
GET /api/things
✓ should route to thing.controller.index
GET /api/things/:id
✓ should route to thing.controller.show
POST /api/things
✓ should route to thing.controller.create
PUT /api/things/:id
✓ should route to thing.controller.upsert
PATCH /api/things/:id
✓ should route to thing.controller.patch
DELETE /api/things/:id
✓ should route to thing.controller.destroy

User API Router:
✓ should return an express router instance
DELETE /api/users/:id
✓ should verify admin role and route to user.controller.destroy
GET /api/users/me
✓ should be authenticated and route to user.controller.me
PUT /api/users/:id/password
✓ should be authenticated and route to user.controller.changePassword
GET /api/users/:id
✓ should be authenticated and route to user.controller.show
POST /api/users
✓ should route to user.controller.create

User Model
✓ should begin with no users
✓ should fail when saving a duplicate user (38ms)
#email
✓ should fail when saving with a blank email
✓ should fail when saving with a null email
✓ should fail when saving without an email
#password
✓ should fail when saving with a blank password
✓ should fail when saving with a null password
✓ should fail when saving without a password
given the user has been previously saved
✓ should authenticate user if valid
✓ should not authenticate user if invalid
✓ should remain the same hash unless the password is updated

33 passing (345ms)
```

```

Candidate Question API:
Express server listening on 9000, in test mode
GET /api/CandidateQuestions
POST /auth/local 200 44.812 ms - 209
GET /api/CandidateQuestions 200 7.142 ms - 376
  ✓ should respond with the list of candidate questions when authenticated
GET /api/CandidateQuestions 401 10.626 ms - -
  ✓ should respond with a 401 when not authenticated
POST /api/CandidateQuestion
POST /auth/local 200 34.671 ms - 209
Entering createCandidateQuestion()..
{ questionText: 'dgqerhqeDescribe blah blah blah...',
  responseChoices: [ '1', '2' ] }
POST /api/CandidateQuestion 200 6.035 ms - 122
  ✓ should create the candidate question when authenticated
POST /api/CandidateQuestion 401 1.496 ms - -
  ✓ should respond with a 401 when not authenticated
DELETE /api/CandidateQuestion/:id
POST /auth/local 200 31.427 ms - 209
{ __v: 0,
  questionText: 'What is the...?',
  _id: 5aeb21dcf36d00dd88c11d48,
  responseChoices: [ 'yes', 'no', 'maybeso' ] }
Entering deleteCandidateQuestion()..id=5aeb21dcf36d00dd88c11d48
DELETE /api/CandidateQuestion/5aeb21dcf36d00dd88c11d48 200 2.851 ms - 14
  ✓ should delete the specified candidate question when authenticated
DELETE /api/CandidateQuestion/5aeb21dcf36d00dd88c11d48 401 0.935 ms - -
  ✓ should respond with a 401 when not authenticated

Thing API:
GET /api/things
GET /api/things 200 1.489 ms - 2
  ✓ should respond with JSON array
POST /api/things
POST /api/things 201 2.883 ms - 101
  ✓ should respond with the newly created thing
GET /api/things/:id
GET /api/things/5aeb21dcf36d00dd88c11d4c 200 1.695 ms - 101
  ✓ should respond with the requested thing
PUT /api/things/:id
PUT /api/things/5aeb21dcf36d00dd88c11d4c 200 5.954 ms - 103
  ✓ should respond with the updated thing
PUT /api/things/5aeb21dcf36d00dd88c11d4c 200 1.699 ms - 103
GET /api/things/5aeb21dcf36d00dd88c11d4c 200 2.012 ms - 103
  ✓ should respond with the updated thing on a subsequent GET
PATCH /api/things/:id
jsonpatch.apply is deprecated, please use `applyPatch` for applying patch sequences, or `applyOperation` to apply individual
operations.
PATCH /api/things/5aeb21dcf36d00dd88c11d4c 200 3.714 ms - 103
  ✓ should respond with the patched thing
DELETE /api/things/:id
DELETE /api/things/5aeb21dcf36d00dd88c11d4c 204 2.716 ms - -
  ✓ should respond with 204 on successful removal
DELETE /api/things/5aeb21dcf36d00dd88c11d4c 404 1.864 ms - -
  ✓ should respond with 404 when thing does not exist

User API:
GET /api/users/me
POST /auth/local 200 32.592 ms - 209
GET /api/users/me 200 2.799 ms - 107
  ✓ should respond with a user profile when authenticated
GET /api/users/me 401 0.915 ms - -
  ✓ should respond with a 401 when not authenticated

16 passing (360ms)

```


A.5. Full Test Coverage Results

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	60.54	30.13	65.22	60.84	
src	100	100	100	100	
mocha.conf.js	100	100	100	100	
mocha.global.js	100	100	100	100	
src/server	92.86	66.67	50	92.86	
app.js	92	100	66.67	92	18,19
index.js	100	66.67	100	100	4,6
routes.js	92.31	100	0	92.31	31
src/server/api	63.64	0	0	63.64	
web3helper.js	63.64	0	0	63.64	... 29,31,36,37,38
src/server/api/candidatequestion	93.06	55.56	96.61	95.04	
candidatequestion.controller.js	83.33	66.67	87.5	83.33	15,36,47
candidatequestion.integration.js	91.3	50	100	95.45	76,118,166
candidatequestion.spec.js	100	100	100	100	
index.js	100	100	100	100	
index.spec.js	95	100	90.91	95	18
src/server/api/degreeprogram	22.22	0	0	22.22	
degreeprogram.controller.js	2.78	0	0	2.78	... 74,75,76,77,79
index.js	100	100	100	100	
src/server/api/email	23.08	0	0	23.08	
email.controller.js	6.25	0	0	6.25	... 71,72,73,75,76
index.js	100	100	100	100	
src/server/api/recletter	10.16	0	0	10.16	
index.js	100	100	100	100	
recletter.controller.js	2.54	0	0	2.54	... 59,260,261,263
src/server/api/recletterrequest	9.63	0	0	9.85	
index.js	100	100	100	100	
recletterrequest.controller.js	3.17	0	0	3.25	... 46,247,248,250
src/server/api/thing	91.93	63.33	98.48	91.93	
index.js	100	100	100	100	
index.spec.js	100	100	100	100	
thing.controller.js	85.29	78.57	94.12	85.29	22,32,63,92,103
thing.events.js	100	100	100	100	
thing.integration.js	89.61	50	100	89.61	... 22,148,172,184
thing.model.js	100	100	100	100	
src/server/api/user	72.66	40.28	68.37	72.66	
index.js	100	100	100	100	
index.spec.js	100	100	100	100	
user.controller.js	12.96	5	8	12.96	... 13,125,129,136
user.events.js	100	100	100	100	
user.integration.js	92	50	100	92	45,60
user.model.js	73.91	54.17	65.22	73.91	... 52,253,255,272
user.model.spec.js	100	100	100	100	
src/server/auth	67.57	56.25	66.67	67.57	
auth.service.js	61.29	56.25	66.67	61.29	... 77,78,80,81,82
index.js	100	100	100	100	
src/server/auth/local	77.78	58.33	87.5	77.78	
index.js	84.62	66.67	100	84.62	13,16
passport.js	71.43	50	83.33	71.43	10,16,19,25
src/server/components/errors	11.11	0	0	11.11	
index.js	11.11	0	0	11.11	... 14,15,16,17,20
src/server/config/environment	100	87.5	100	100	
index.js	100	87.5	100	100	54
shared.js	100	100	100	100	
test.js	100	100	100	100	
src/server/model	96.3	100	80	96.3	
candidatequestions.js	94.12	100	80	94.12	54
degreeprograms.js	100	100	100	100	
recletterrequests.js	100	100	100	100	
reclletters.js	100	100	100	100	

A.6 Configuration

The express server has different configuration settings that may need to be adjusted if you change the ip or port of any external service.

MongoDB - The mongo uri can be adjusted here `/src/server/config/environment/development.js`

Ethereum - The uri for where your Ethereum client is running can also be adjusted here: `/src/server/config/environment/development.js`

IPFS - The uri for the IPFS daemon can be adjusted here: `src/server/api/web3helper.js`. There are two constants `ipfsHost` and `ipfsAPIPort` that can be adjusted.

Once all dependencies are installed, run the `start.sh` script in `src` to start up all the needed dependencies.

References

- [1] "MongoDB," [Online]. Available: <https://www.mongodb.com/>. [Accessed 23 Apr 2018].
- [2] "Ethereum," [Online]. Available: <https://www.ethereum.org/>. [Accessed 23 Apr 2018].
- [3] "What is a Decentralized Application? - CoinDesk," [Online]. Available: <https://www.coindesk.com/information/what-is-a-decentralized-application-dapp/>.. [Accessed 2018 1 April].
- [4] "Truffle Suite - Your Ethereum Swiss Army Knife.," [Online]. Available: <http://truffleframework.com/>. [Accessed 1 Apr 2018].
- [5] "Solidity," [Online]. Available: <https://solidity.readthedocs.io/en/v0.4.23/>. [Accessed 23 Apr 2018].
- [6] "Mocha - the fun, simple, flexible JavaScript test framework.," [Online]. Available: <https://mochajs.org/>. [Accessed 1 Apr 2018].
- [7] "Chai," [Online]. Available: <http://www.chaijs.com/>. [Accessed 1 Apr 2018].