Bilkent University

Department of Computer Engineering

Senior Design Project

Project Short-name: So FarM So Good

Low Level Design Report

Giray Baha Kezer, Fazilet Simge Er, Melih Ünsal, Kaan Atakan Öztürk

Supervisor: **Prof. Dr. Halil Altay Güvenir**

Jury Members: **Prof. Dr. Özcan Öztürk, Prof. Dr. Uğur Güdükbay**

Innovation Expert: **Kerem Erikçi**

February 17, 2020

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Senior Design Project course CS492.

Table of Contents

1.Introduction

Blockchain idea was introduced in the early twentieth century. By blockchain, both side of a trade doesn't need any mediator to prove that this trade has occurred. Nowadays, we use banks as a mediator in the trades. However, blockchain convert the trades as a decentralized form so that everyone is able to see all the trades real time, and this makes the trades more secure ever than before.

In Turkey, farmers, especially the small ones, have so many problems and this situation causes our production in agriculture to decrease day by day. However, we cannot put ourselves at risk for agriculture because agriculture constitutes around 13% of the country's exports. We also see that agricultural areas and the number of farmers has decreased for the last 5 years. So, the problems of agriculture should be resolved one by one and we plan to solve some problems of agriculture by So FarM So Good. In this project, we use blockchain in the trades between the virtual cooperatives and the farmers and between the farmers in cooperatives and the government. The aim of these project is to add value to the agricultural economy of the country by increasing the producers' contribution to total production.

This web-based platform is going to combine small farmers so that they will be able to sell their goods to the appropriate companies by the virtual cooperative we are going to establish. Normally, some companies have some limitations when they are aiming to buy goods from farmers and the farmers who are not able to produce under the threshold that the companies put, sell their crops to the local landowners for less that worth. By our virtual cooperative, they will be able to combine their crops and the cooperative is going to sell the crops and pay to the farmers proportional to the amount of products the farmers put into with a secure payment system which uses the new trend: Blockchain.

1.1 Object Design Trade-offs

1.1.1 Usability vs Functionality

The platform must have ease of use for our main customers, farmers. Farmers should be able to use this platform, which is a new technological move, easily. Thus, the UI must be kept as simple as possible to increase ease of use, but this makes adding more functionality to the system difficult.

1.1.2 Security vs Memory

In a blockchain project, security plays a big role. As the engineers coding the blockchain system, we must ensure that even if a small part of our system is hacked

or the data it contains are changed, the system must not be affected and the data must not be lost or get in the hands of irrelevant people. To be able to provide such a high security, the blockchain system must be saved in every computer involved. This makes the memory usage further increase as more data is added to the system.

1.1.3 Robustness vs Cost

Nearly in all software, bugs are a large part of the workload which comes with development. Due to that, the process of bug-fixing costs a lot of time. But in a blockchain system, bugs can be fatal. Therefore, we must make sure that the number of bugs is as close as possible to zero, although it will cost a lot of precious time. We will test the platform at various stages to hopefully circumvent this cost as much as possible.

1.1.4 Extensibility vs Cost

For the platform to be open to easy integrations, the code must be written neatly and the platform must have basic functionalities. For us to be able to do this, we must spend more time on the project than simply writing a working code. Since as engineering students writing a program the only cost of ours is time, it will cost us more than usual for us to write an extensible code.

1.2 Interface Documentation Guidelines

Class: Name of the Class
Description of the class
Properties
Attributes, variables
Methods
Names of the methods and their descriptions

1.3 Engineering Standards

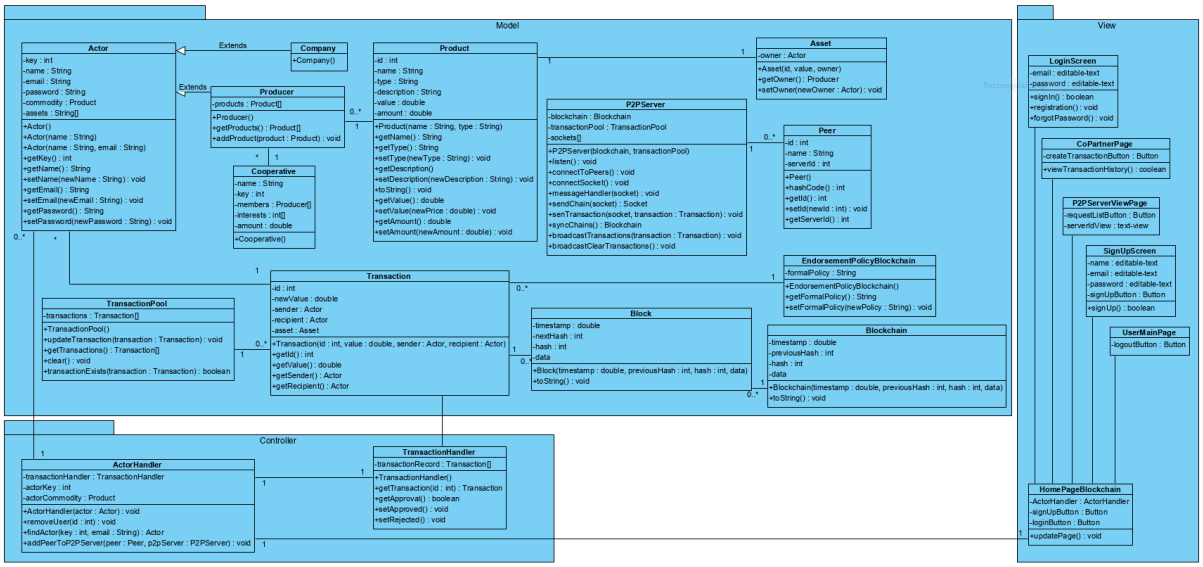
This report is written using IEEE format to write an organized report and meet engineering standards. Also, UML diagrams are used to identify, visualize and construct the project. They play an important role in the graphical representation of our project.

1.4 Definitions, acronyms, and abbreviations

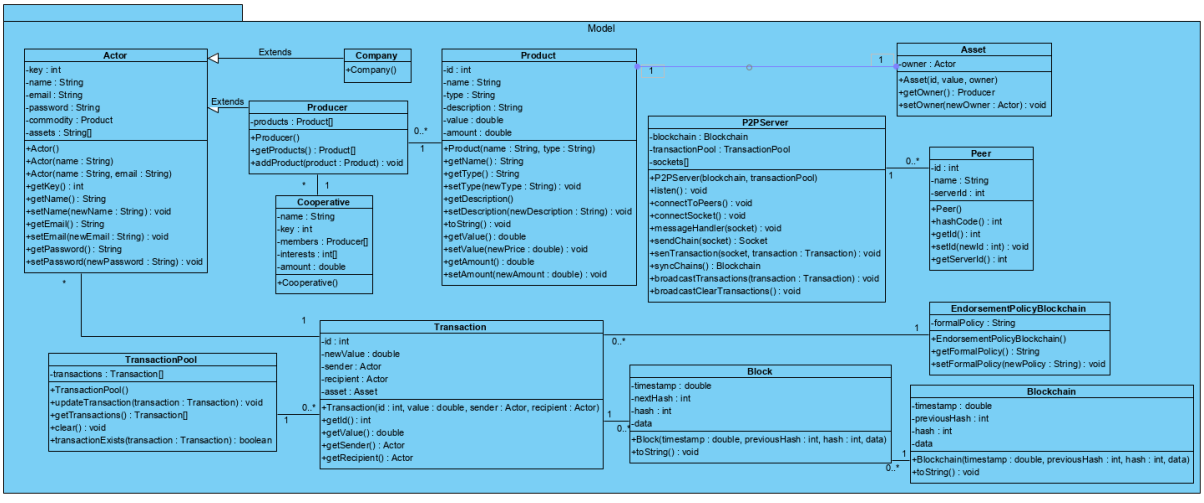
- **UI:** User Interface
- **IEEE:** The Institute of Electrical and Electronics Engineers
- **UML:** Unified Modeling Language
- **MVC:** Model-View-Controller
- **P2P:** Peer to Peer

2.Packages

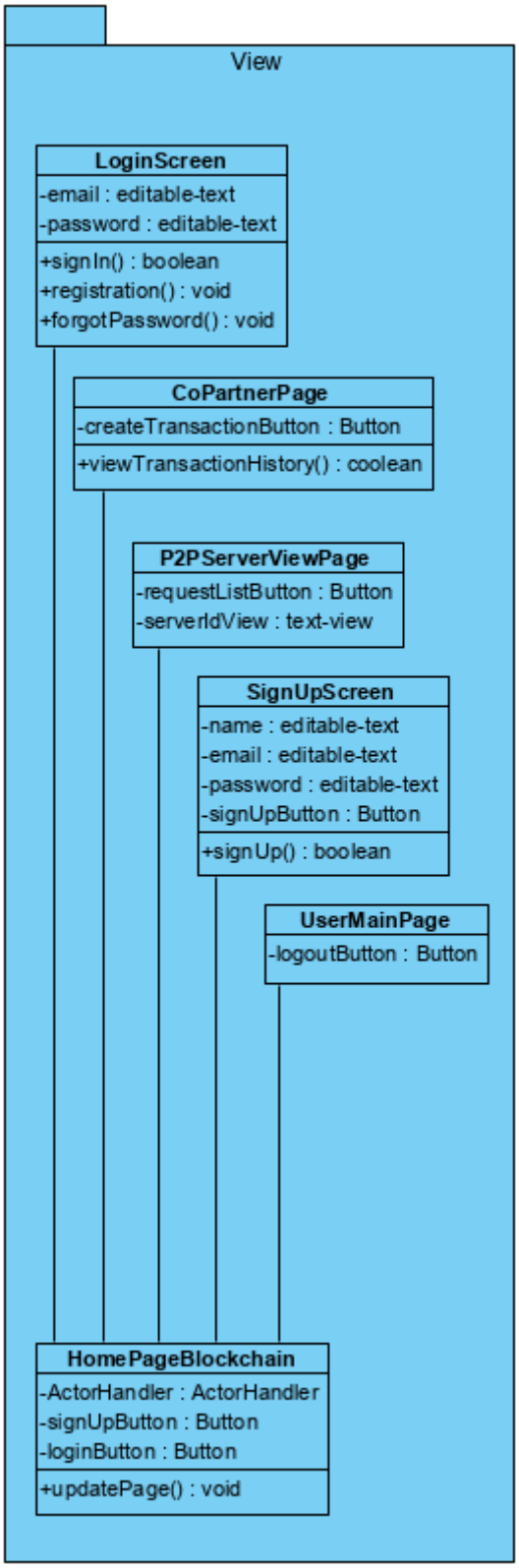
Full Class Diagram



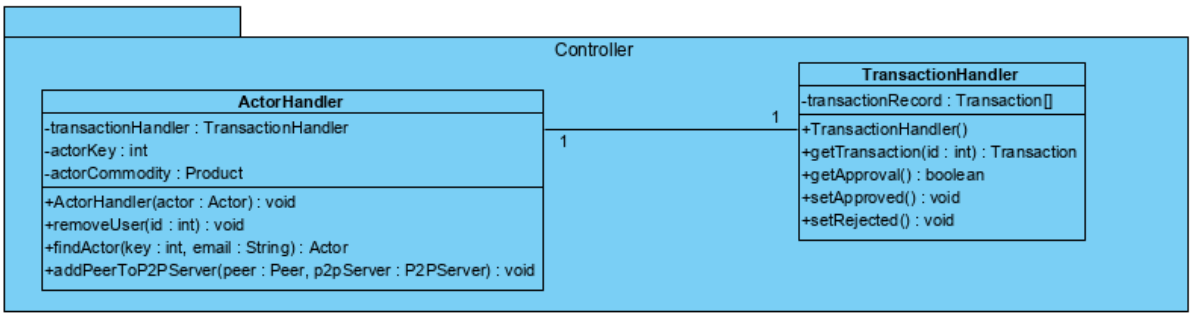
2.1 Model Class Diagram



2.2 View Class Diagram



2.3 Controller Class Diagram



3.Class Interfaces

SoFarMSoGood will be explained by documenting its most important methods and its properties. Since we will make our project in MVC design pattern, there will be three subsections as Model, View and Controller.

3.1 Model

Class: Actor
This class processes new blocks for commodities and assets.
Properties
int **key** - private key to traverse into blockchain blocks String **name**- a string which holds name of the actor. String **email**- A string which holds the email of the actor. String **password** - A string which holds the password of the actor. Product **commodity**- A product-based object to hold property inside value, harvest. String **assets** - A string based made cryptocurrency of actor's stakes.
Methods
getKey(): Returns the key of the actor. **getName():** Returns the name of the actor. **setName(String newName):** Changes the name of the actor with the given newName. **getEmail():** Returns the email of the actor. **setEmail(String newEmail):** It sets new assigned email to string. **getPassword():** Returns string based password. **setPassword(newPassword):** It sets password with a new string value.

Class:Producer
Producer class is responsible for product-based operations, it means adding products, selling products and taking Product object are operated by that class.
Properties
Product products[]: An array based product objects to hold products inside.
Methods
getProducts(): A method for taking products. **addProduct(newProduct):** A method takes new products as parameter used for adding new product amount.

Class: Cooperative
Cooperatives takes farmers products to pass the distinct quotas of companies.
Properties
String **name:** String based name property. String **key:** String based private key property. Producer **members[Producer]:** Producer based array holds members. int **interests[int]:** Interest based integer array holds interests inside. Amount **amount[double]:** Amount based double array holds amounts inside.
Methods
setCooperative(Copartner copartner): void

Class: Product
Objects of this class represents the real life products produced by producers.
Properties
int **id:** Integer based unique id property String **name:** String based name property String **type:** String based type property String **description:** String based description explaining the product double **value:** Double based value property double **amount:** Double based amount property
Methods
getName(): A method for taking the name of the product. **getType():** A method for taking the type of the product.

setType(String newType): A method for setting the type of the product.
getDescription(): A method for taking the description of the product.
setDescription(String newDescription): A method for setting the description of the product.
toString(): A method for printing the product.
getValue(): A method for taking the value of the product.
setValue(double newValue):A method for setting the value of the product.
getAmount(): A method for taking the amount of the product.
setAmount(double newAmount):A method for setting the amount of the product**.**

Class: Asset
The objects of this class represents a Product with an owner.
Properties
Actor **owner:** Actor based object showing the owner of the asset.
Methods
getOwner():A method for taking the owner of the asset **setOwner(Actor newOwner):**A method for taking the owner of the asset

Class: Transaction
Objects of this class represent results of exchanges between producers and the companies.
Properties
int **id:** Integer based unique id property double **newValue:** Double based updated value of the transaction Actor **sender:** Actor based sender property Actor **recipient:** Actor based recipient property Asset **asset:** Asset based property
Methods
getId(): A method for taking the id of the transaction **getValue():** A method for taking the value of the transaction **getSender():** A method for taking the sender of the transaction **getSender():** A method for taking the sender of the transaction

Class: TransactionPool
This class stores all transactions.
Properties
transactions[Transaction]**:** Transaction based array property holding the transactions
Methods
updateTransaction(Transaction transaction): A method for updating the transaction in the pool **addTransaction(Transaction transaction):** A method for adding a transaction to the pool **getTransactions():** A method for taking a transaction in the pool **clear():** A method for removing all the transactions in the pool **transactionExists(Transaction transaction):** A method for controlling if a transaction exists in the pool

Class:Block
Objects of this class stores the Transactions.
Properties
double **timestamp:** Double based time property showing the time of the creation of the block int **nextHash:** Integer based property to make a connection with the next block int **hash:** Integer based property to encrypt the block string **data:** A property that holding all the information about the block
Methods
toString(): A method to write the block

Class: Blockchain
This class stores the Blocks.
Properties
double **timestamp:** Double based time property showing the time of the creation of blockchain int **previousHash:** Integer based property to make a connection with the previous block int **hash:** Integer based property to encrypt the blockchain

string **data:** A property that holding all the information about the object
Methods
toString(): A method to write the block

Class:P2PServer
This class represents a dummy server living in the connection established between the pairs.
Properties
Blockchain **blockchain:** A blockchain based property TransactionPool **transactionPool:** A trancationPool based object holding all the transactions Socket **sockets[]:** A socket based array property
Methods
listen(): A method to listen a socket **connectToPeers():** A method to connect to server **connectSocket():** A method to connect to socket **messageHandler(Socket socket):** A method to handling the message coming from Socket **sendChain(Socket socket):** A method to send the chain to the appropriate socket **sendTransacion(Transaction transaction):** A method to send the transaction **brodcastClearTransactions():** A method to send the clear the transactions

Class: Company
This class extends one Actor. It is a type of an actor and it is a part of transactions with cooperatives or directly with farmers. Takes products and gives money commodities exchanged with money and product assignments. Its transactions hold on blockchain records.
Properties
Actor actor: Actor represent stakeholders could be processed transactions with commodities **Transaction transaction:** Transaction object holds transaction inside **Quality level:** Level represent for product's quality
Methods
validity(): returns boolean validity decision for company to be verified for transaction

Class: ActorHandler
This class implements user and actor duties
Properties
TransactionHandler: **TransactionHandler** actorKey: **int** actorCommodity: **Product**
Methods
addActorKey(key:int,name:String,email:String):void **removeUser**(id:int):void **findActor**(key:int,email:String):Actor **addPeerToP2PServer**(Peer:peer, P2PServer : P2PServer)

Class:TransactionHandler
This class regulates transactions.
Properties
transactionRecord:**Transaction**<>
Methods
getTransaction(id): Transaction **provokeTransaction**: void **getApproval()**:boolean **setApproval()**:void **setReject()**:void

Class:EndorsementPolicyBlockChain
This class regulates endorsement policy of the Blockchain.
Properties
formalPolicy:**String** holds policy variable for formal process
Methods
getformalPolicy:String represents formal policy getter method **setformalPolicy**:void represent formal policy setter method

Class: Quality
This class implements levels within the Blockchain.
Properties
level:**int**
Methods
getLevel():int

Class:Copartner
This class implements sign transactions.
Properties
list:**Arraylist**
Methods
getSign():boolean **setSign()**:void

3.2 View

Class: LoginScreen
This class enables users a login screen interface.
Properties
name:**String** password:**String** key:**int**
Methods
signIn():void **registration()**:void

Class: HomePageBlockchain
This class enables user home page of the system.
Properties
ActorHandler: **ActorHandler** page:**frame**
Methods
updatePage(frame:page):void

Class: CoPartnerPage
This class enables user CoPartnerPage interface.
Properties
coPartnerView:**page** **Methods**
viewTransactionHistory():Transaction list **createTransactionButton(Transaction transaction)**: void

Class: P2PServerViewPage
This class enables users P2PServerViewPage interface.
Properties
ServerIDView: **TextView**
Methods
requestList():void

Class: ApprovalView
This class enables users ApprovalView interface.
Properties
name:**String** keyID:**int**
Methods

signIn():void **signUp()**:void

Class: userMainPage
This class enables users userMainPage interface.
Properties
logoutButton:**Button** this button enables logout from page
Methods
-

Class: signUpScreen
This class enables users signUpScreen interface.
Properties
name:**text-view** represents name property of singed up user password:**int** represents password of singed up user email:**text-view** shows email of signed up user signUpButton:**button** enables signed up button for user
Methods
signUp():boolean holds boolean decision inside represent signed or not

3.3 Controller

Class: TransactionHandler
This class manages and handles transactions
Properties
transactionRecord:**Transaction<>**
Methods
getTransaction(id): Transaction **provokeTransaction**: void **getApproval()**:boolean **setApproval()**:void **setReject()**:void

Class: ActorHandler
This class actor of the
Properties
TransactionHandler: **TransactionHandler** actorKey: **int** actorCommodity: **Product**
Methods
addActorKey(key:int,name:String,email:String):void **removeUser(id:int)**:void **findActor(key:int,email:String)**:Actor **addPeerToP2PServer(Peer:peer, P2PServer : P2PServer)** **addPeerToP2PServer(Peer:per,P2PServer:P2PServer)**

4.Glossary

Hyperledger Composer: Hyperledger Composer is Blockchain Application Development framework which simplify the blockchain application development on Hyperledger Fabric[1]

IEEE: IEEE, an association dedicated to advancing innovation and technological excellence for the benefit of humanity, is the world's largest technical professional society[2]

P2P Server: Peer-to-peer, or P2P in its abbreviated form, refers to computer networks using a distributed architecture.[3]

5.References

[1] Blockchain Training Alliance. (2019). *Global Glossary of Blockchain Terms 2.0 in 5 Languages*. [online] Available at: https://blockchaintrainingalliance.com/pages/glossary-of-blockchain-terms

[2] "History of IEEE," *IEEE*. [Online]. Available: https://www.ieee.org/about/ieee-history.html. [Accessed: 16-Feb-2020].

[3] "What are P2P (peer-to-peer) networks and what are they used for?," *Digital Citizen*, 07-Dec-2019. [Online]. Available: https://www.digitalcitizen.life/what-is-p2p-peer-to-peer. [Accessed: 16-Feb-2020].