SDE Assignement 1
Suresh Bojjam
M20AIE313

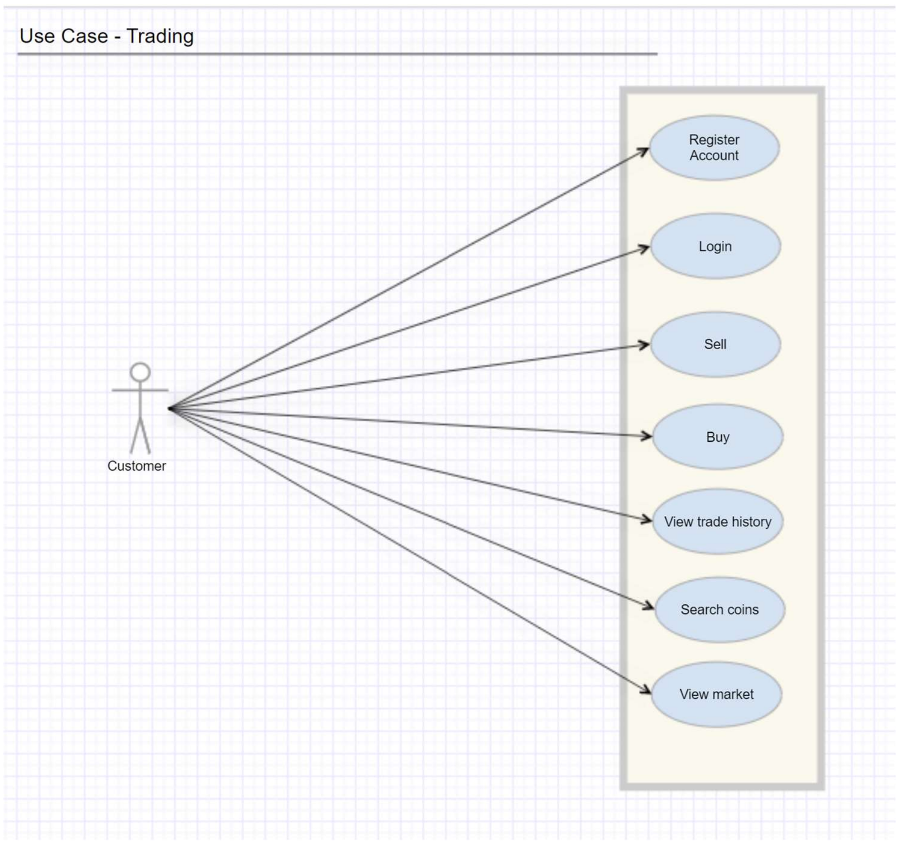
Q1) Pick an example of a blockchain-based application and draw 4+1 views for at least 2 scenarios. Explain each of these views in 200 words each.

Ans:
Example application 'Zebpay'

- Scenario 1: Trade
    - Buy/Sell available bitcoin
- Scenario 2: Wallet.
    - Send/Receive with wallet
- Scenario 3: Mining
    - Configure wallet to mining pool (your own code or system for mining) and Mine the bitcoin, then add to wallet.

Scenario1: Buy/Sell crypto coin
Major use cases

1. **Trade - buy and sell different crypto coins to earn the profit**
2. **Set predefined market limit to automate the buy/sell with stop price so without manual intervention trading can be happen**

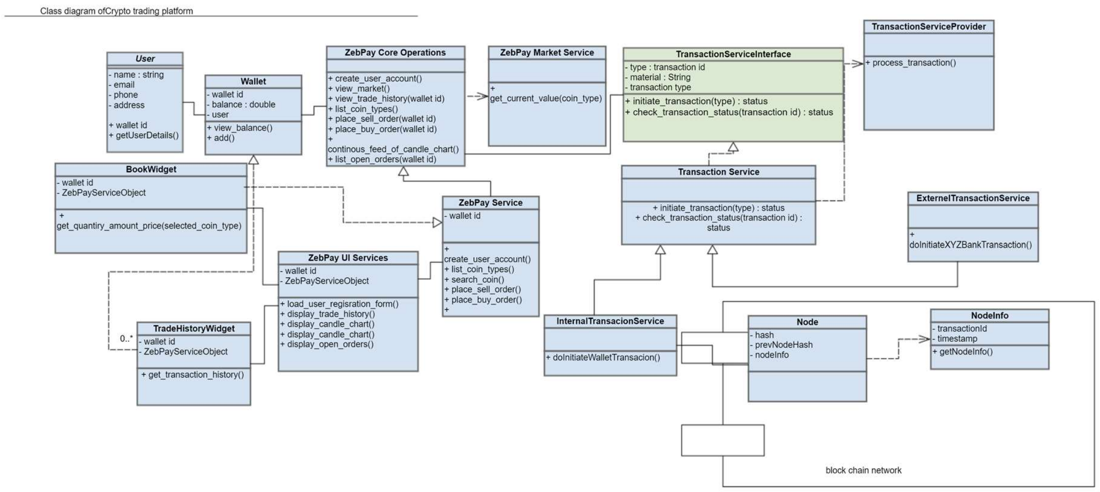


User actions:

SDE Assignement 1
Suresh Bojjam
M20AIE313

1. Create user account
2. Login into account
3. Sell crypt coin
4. Buy crypto coin
5. Search different coin
6. View current market data on different crypto coins
7. View trade history

Logical/Model view:



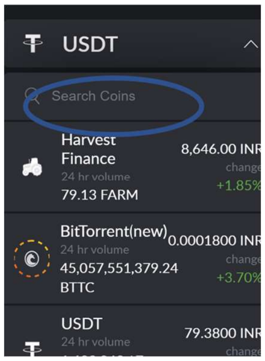
Different end user functionalities

1. Create a user account – creating the user account for zebpay platform.
2. Search coins – search different available crypto coins (bitcoin, ether, harvest finance, & etc) through the zebpay search option for trading.
3. Buy coin
    a. Place buy order – placing the buy crypto coin order
    b. Set market, limit and stop price – setting the predefined market limit for placing the auto buy order.
4. Sell coin
    a. Place sell order – placing the sell crypto coin order.
    b. Set market, limit and stop price – setting the predefined market limit for placing the auto sell order.
5. Overview
    a. UI widgets display
        i. Candle chart
        ii. Book – quantity, amount, & price
        iii. Open orders
        iv. Transaction history widget
6. Trade history

SDE Assignement 1
Suresh Bojjam
M20AIE313

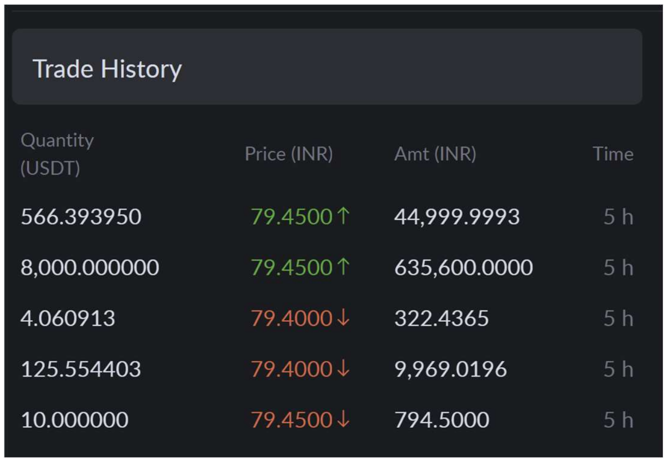
Models/Classes:

1. User – class User represents the entity of user with all user details like name, email, phone, & etc.
2. Wallet – class wallet represents digital wallet of object representation and responsible to maintain the wallet id, balance and respective user.
    a. *view_balance()* – to view the current balance of the wallet.
    b. add() – add a crypto coin to wallet
3. ZebPayCoreServices – class responsible for the core services of the platform like
    a. view_market_data() – responsible to get the different crypto currency value in given point in time.
    b. view_trade_history(wallet id) – responsible to the trade grab the trade history from the wallet and provide it to caller.
    c. list_coin_types() – responsible to provide the list of different crypto coins available to trade under the zebpay platform.
    d. place_sell_order(wallet id, quantity, coin_type) – responsible to place the sell order request.
    e. place_buy_order(wallet id, quantity, coin_type) – responsible to place the buy order request.
    f. list_open_orders(wallet id) – lists the current orders waiting for confirmation.
4. ZebPayMarketService – provides the market details
    a. get_current_value(wallet id) – get the current value of the given coin type.
5. ZebPayUIServices – responsible to render the UI
    a. load_user_registration_form() – responsible for rendering the user registration form and submit the form when user click on sign-up.
    b. display_trade_history() – responsible to render the TradeHistoryWidget and data.
    c. display_candle_chat() – responsible to display the candle chart based on current market value provided by ZebPayMarketService.
    d. display_open_orders() – responsible to render the current open orders. In other words orders waiting for approval or transaction to be completed.
6. ZebPayService


USDT
Search Coins
Harvest Finance 8,646.00 INR
24 hr volume change
79.13 FARM +1.85%
BitTorrent(new) 0.0001800 INR
24 hr volume change
45,057,551,379.24 BTTC +3.70%
USDT 79.3800 INR
24 hr volume change

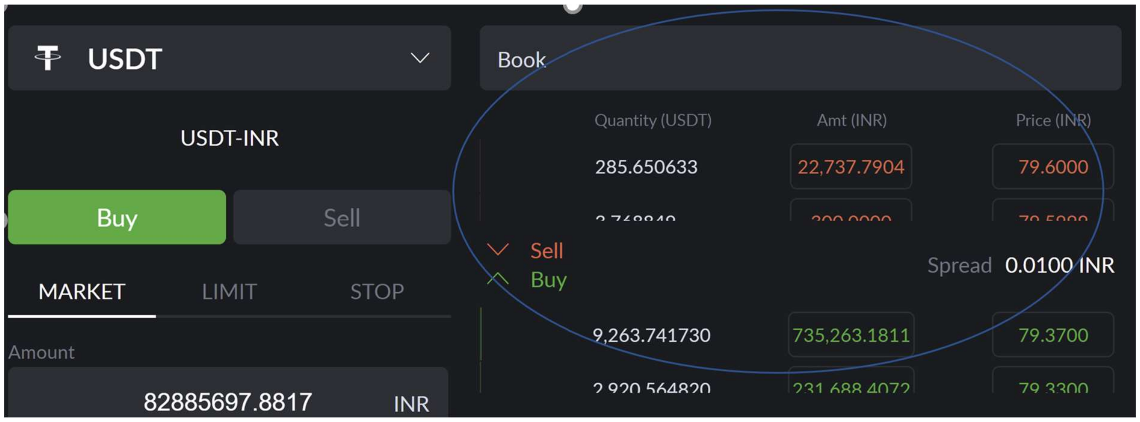
    a. search_coin() – search different coin
    b. place_sell_order() – place the sell order
    c. place_buy_order() – place the buy order
7. TradeHistoryWidget - widget is responsible to get the previous transactions from wallet and render the UI

SDE Assignement 1
Suresh Bojjam
M20AIE313


Trade History

Quantity (USDT)	Price (INR)	Amt (INR)	Time
566.393950	79.4500↑	44,999.9993	5 h
8,000.000000	79.4500↑	635,600.0000	5 h
4.060913	79.4000↓	322.4365	5 h
125.554403	79.4000↓	9,969.0196	5 h
10.000000	79.4500↓	794.5000	5 h

8. BookWidget – responsible for getting the recent market data of selected coin type and display quantity, amount & price to **Buy Or Sell**.

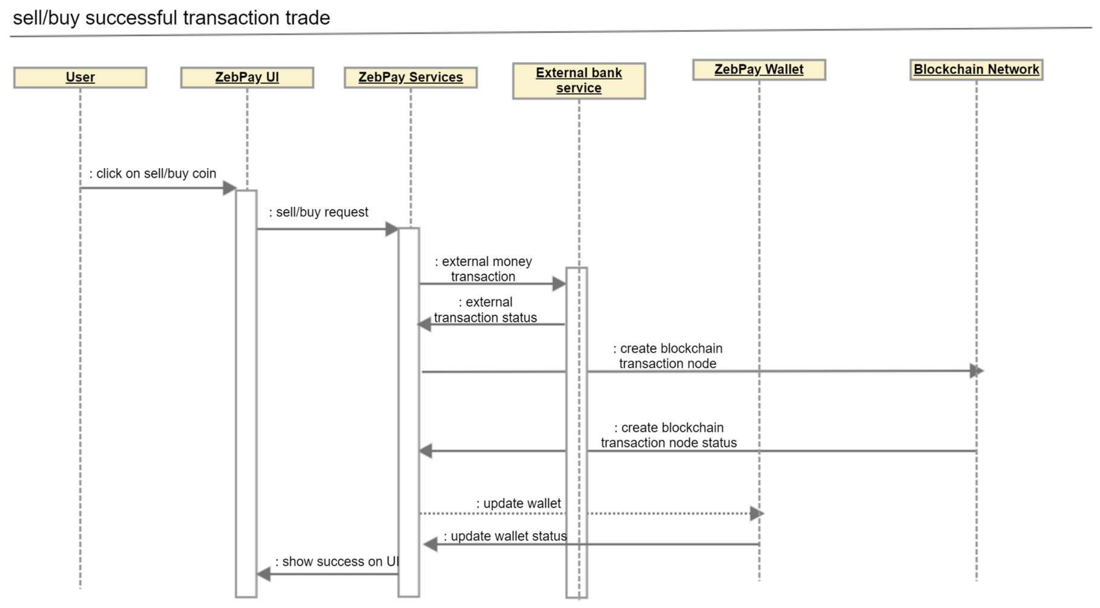


Quantity (USDT)	Amt (INR)	Price (INR)
285.650633	22,737.7904	79.6000
9,263.741730	735,263.1811	79.3700
2,920.564820	231,688.4072	79.3300

9. TransactionServiceInterface – abstraction of transaction service
   a. responsible for creating the transaction id
   b. get the transaction hash and previous has to form the block chain
   c. initiate_transaction(type): responsible for initiating the transaction of given type.
   d. check_transaction_status(transaction id): status – responsible to check the current status of the transaction from underlying transaction provider like bank, network transaction confirmation, & etc.
10. TransactionService – actual implementation of TransactionServiceInterface
11. TransactionServiceProvider – transaction provider for different banks. This class provides the different bank apis to load the money to buy the crypto coin.
12. ExternalTransactionService – actual class to initiate any external bank transactions
13. InternalTransactionService – responsible for wallet to wallet transactions based on given wallet id and create a block chain node for transaction. No relation with bank.

SDE Assignement 1
Suresh Bojjam
M20AIE313

Process view:



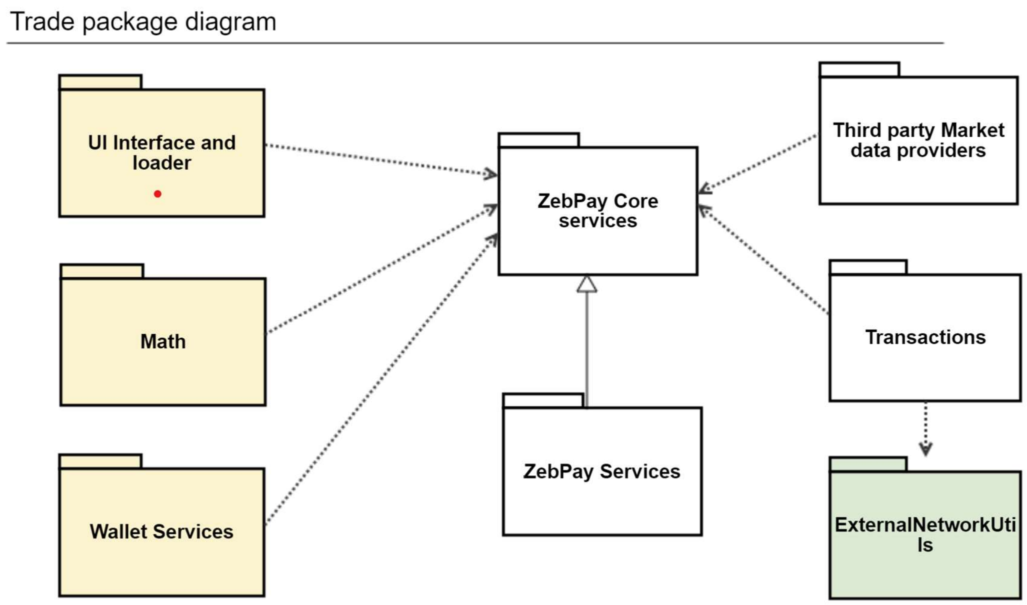
1. Actors
    a. User
    b. Trade customer interface
    c. Dex network and nodes
    d. External systems like bank
2. Process
    a. Place sell order
    b. Place buy order
    c. External transaction workflow
    d. Internal transaction workflow

Process Workflows:

1. Scenario: When user request for buy crypto
    a. User request the ZebPay service to buy a selected crypto coin through UI
    b. Service will contact the external bank service to load the equal amount of cash to buy the crypto coin.
    c. On successful bank transaction, ZebPay platform will reach block chain network with new node created (hash& transaction id generated) for transaction and validate with the network.
    d. On successful validation of network, ZebPay service will add the crypto coin to wallet.
2. Scenario: When user request for sell crypto
    a. User request the ZebPay service to sell the crypto and take the quantity, limit& stop price as inputs.
    b. On user request, crypto will be displayed under the **Book** for selling.

c. When others requested the buy scenario 1 will execute and then money will be added to the user bank account.

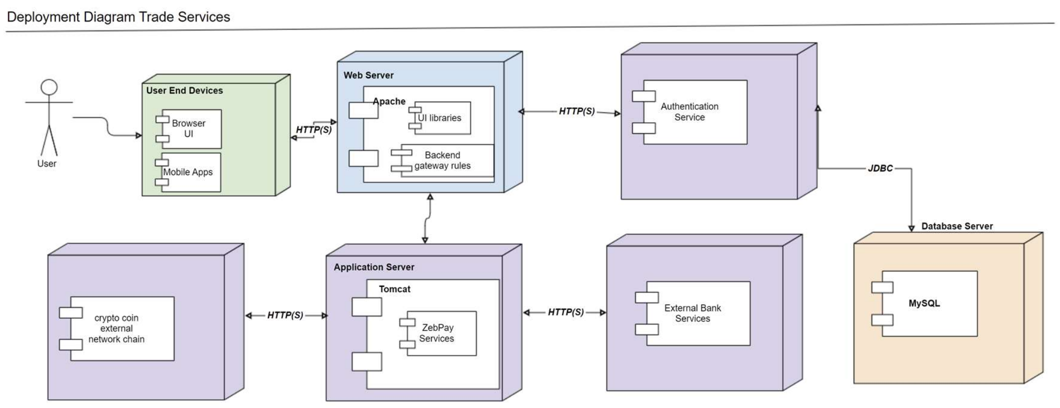
Development view:



1. UI Interface and loader – package responsible to have the all UI related libraries and code.
2. ZebPayCore – core services provided by the platform.
3. ZebPayServices – all business workflow logic will be there under this package.
4. WalletServices – wallet core operations like balance, add/subtract coin operations would be there under this package.
5. Math – math will be responsible to provide the utilities for safe math.
6. MarketDataProviders – Thridparty libraries to read the network and get the current market data or value of each crypto type.
7. Transactions – this package will have below business logic classes
    a. All external bank apis
    b. Internal transaction and validation process of block chain network
    c. Generate hash, transaction id
    d. Check transactions status
8. ExternalNetworkUtility – utility methods for calling over network like https, & etc.

SDE Assignement 1
Suresh Bojjam
M20AIE313

Physical View:



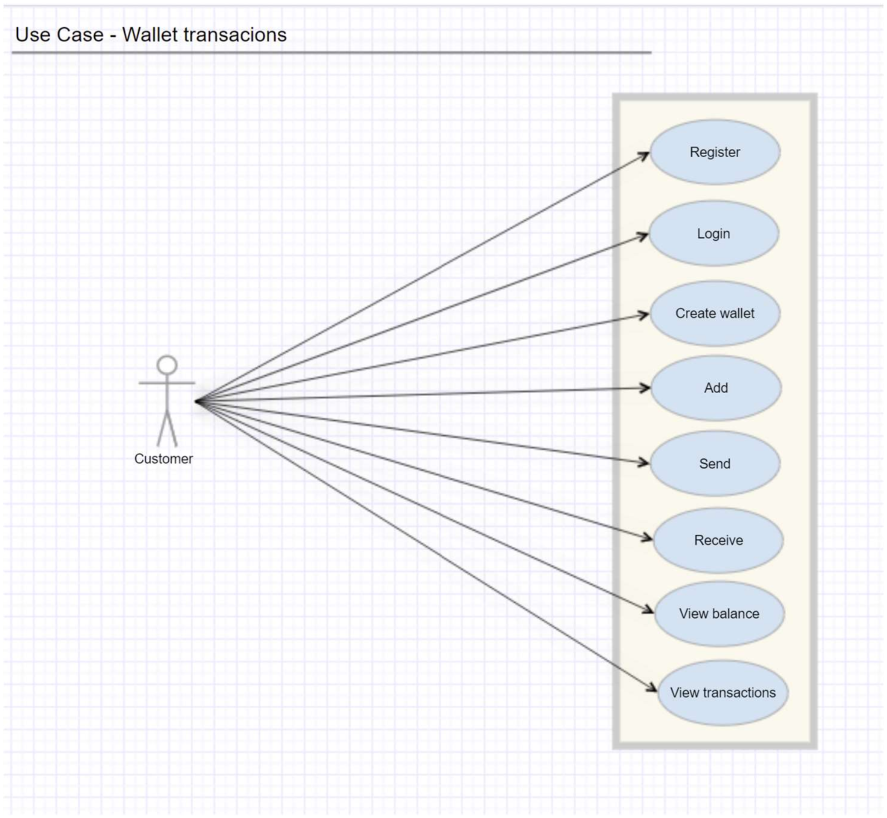
1. User End devices – browser, UI
2. WebServer
    a. responsible to get the UI code and render on Browser
    b. gateway - responsible to be act as a front facing server and redirect the apis to respect backend api server.
3. Application Server
    a. Server with ZebPay platform services deployed
4. Authentication service – service to authenticate the users
5. External Banks – all external bank libraries and apis.
6. MySql – database for authentication service

Scenario2: Send/Receive crypto coin with wallet transactions

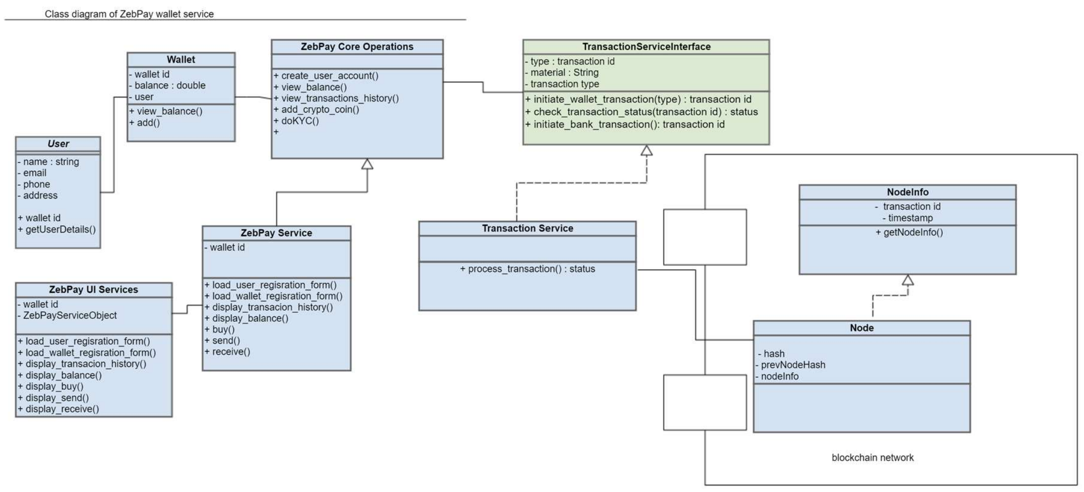
Some general use cases

3. **Send/receive the crypto money to different people**
4. **Exchange Crypto**
5. **Buy general goods on ecommerce platform with crypto wallet**

SDE Assignement 1
Suresh Bojjam
M20AIE313



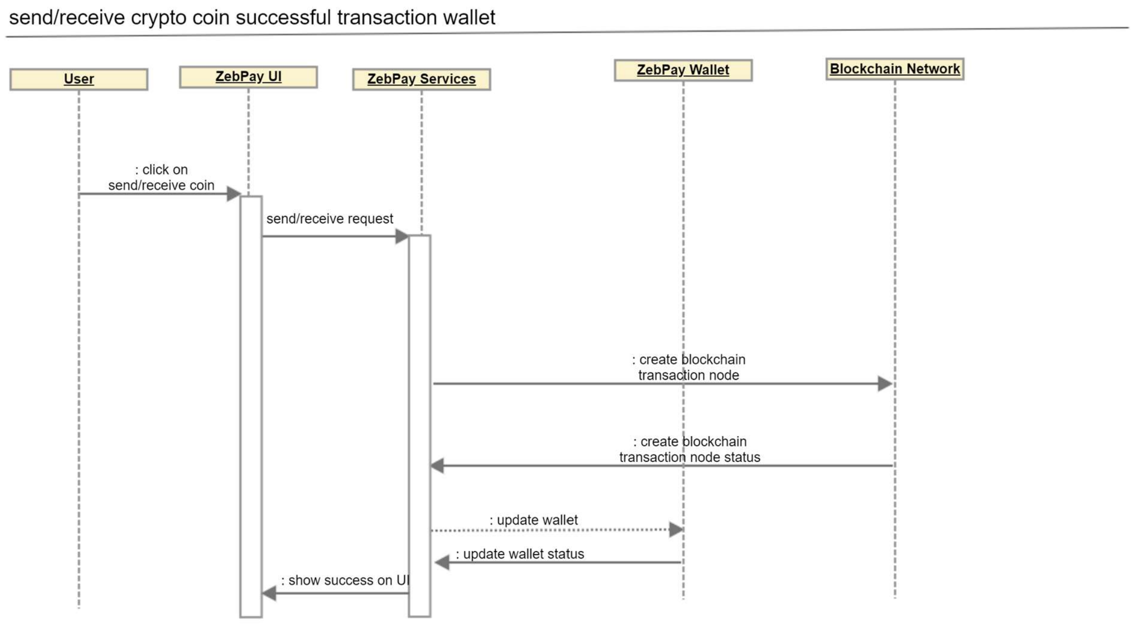
Logical/Model view:



1. Create wallet
2. Overview (View wallet balance)
3. Transfer to another wallet
   a. Send
   b. Receive
4. Recent transactions

5. Audit for internal purpose

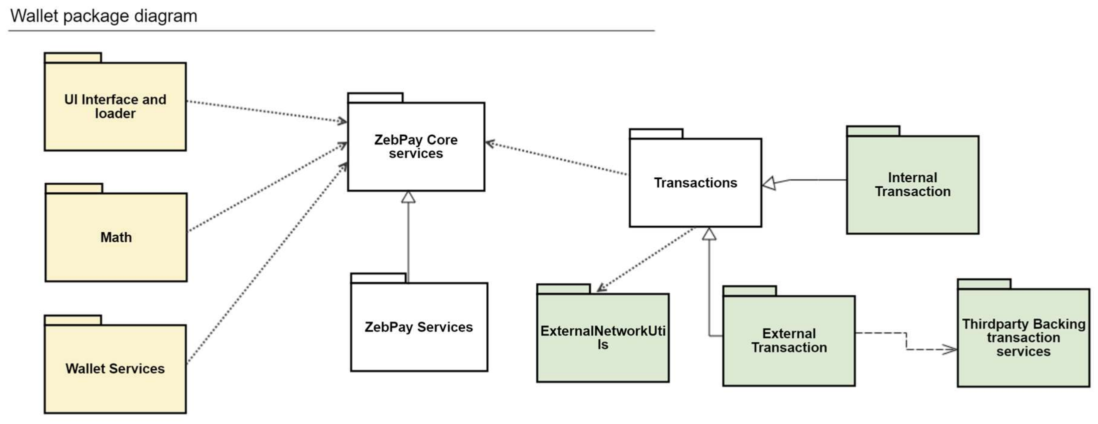
Process View:


send/receive crypto coin successful transaction wallet
User
ZebPay UI
ZebPay Services
ZebPay Wallet
Blockchain Network
: click on send/receive coin
send/receive request
: create blockchain transaction node
: create blockchain transaction node status
: update wallet
: update wallet status
: show success on UI

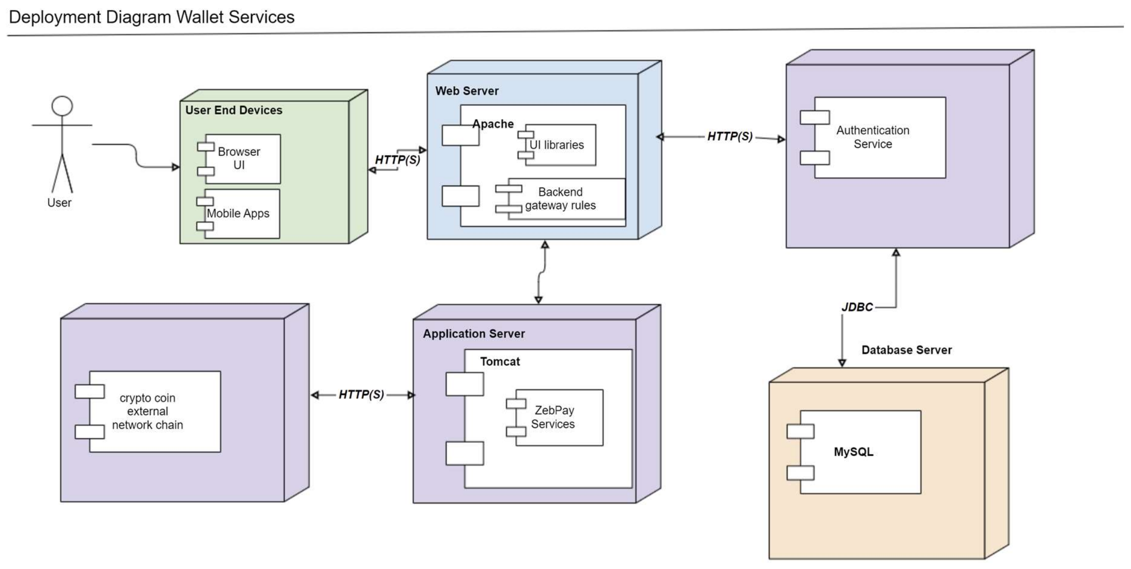
1. Senario1: When user request to send the crypto coin to a given wallet
    a. User see the available amount on wallet and request the send crypto to a given wallet id through ui.
        i. available crypto validate with requested amount.
        ii. user provide the quantity.
    b. ZebPay UI will contact the service with given input request details.
    c. ZebPay Service initiate the transaction through InternalTransactionService
    d. InternalTransaction creates a blockchain network node for the transaction and validate with Blockchain network
    e. On successful transaction status received, ZebPayService send the update wallet request.
    f. On successful wallet update, UI will update the wallet updated amount for both user requested for *send* and *received*.

SDE Assignement 1
Suresh Bojjam
M20AIE313

Development View:



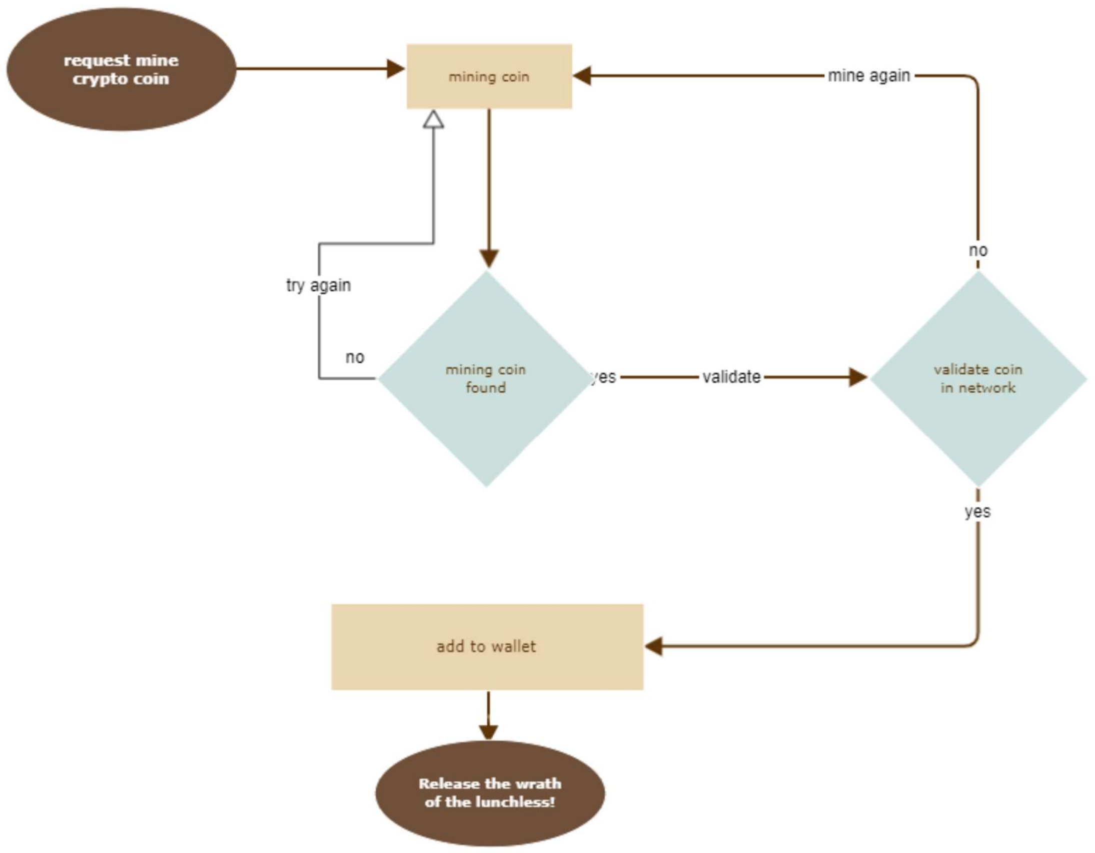
Physical View:



Scenario3: Mining
Some general use cases

1. **Mine the crypto coin**
2. **Configure wallet to mining pool (your own code or system for mining) and Mine the bitcoin, then add to wallet.**

SDE Assignement 1
Suresh Bojjam
M20AIE313



Q2) Identify two design patterns used in the app and explain each of the design patterns with the following subsections
a) Problem Statement
b) Context
c) Forces/Constraints and
d) Solutions - Static & Dynamic schematics
e) Consequences.

Factory pattern:
Problem statement:

1. feed of market data as a continuous stream for different type of crypto currencies require different libraries and apis to integrate under the same service.
2. Application doesn't know list of all service providers. Api would be responsible to collect the list of supported crypto coins and then collect the market data and provide it to UI.

Ex:
api1, collect the value of BitCoin
api2, collect the value of harvest finance
api3, collect the value of usdt

Context:

SDE Assignement 1
Suresh Bojjam
M20AIE313

1. one api should provide the current market data of all supported crypto currency value or market data with single client api without providing the any details.
2. And underlying factory method can be used to get the specific type of crypto currency market data by passing the type.
3. Adding of the new crypto coin to the application would be easier just by adding one more underlying factory method and caller wouldn't know any implementation details.

Forces/Constraints:
Solution – Static & Dynamics schematics:
Pseudo code

```
getMarketData{
List<Data> list[];
Int i=0;
For(CryptoType cryTyp: DifferentCryptoTypes){
        list [i++]= FactoryClass.getData(cryTyp);
        }
}
```

Consequences:

Abstract Factory Method pattern:

Problem statement: Multiple classes/objects for different transaction types, different service providers for each transaction type and complexity of code integrations.
Context: When we have multiple transaction types and respective underlying concrete objects. For a caller client/service, it will be difficult to maintain the all references and implement each method for each type of transaction.
Forces/Constraints: passing different set of parameters for different type of transaction may problematic with abstraction factory method.
Solution – Static & Dynamics schematics: With one factory abstract method and a context property object as a parameter will solve the problem

1. Context property object is a kind of hashmap with all required key, value pair of properties for given transaction. This will avoid the passing the of different parameters to the same factory method which is a declaration problem.
2. Below example consists two different set of parameters for two different transaction types.
    a. process_transactionAType(param1, param2)
    b. process_transactionBType(param1, param2, param3)
3. Solution:
    a. Process_transaction(HashMap<Object,Object> props)

SDE Assignement 1
Suresh Bojjam
M20AIE313

Pseudo code

```
AbstractFactory{
        process_transaction(transactionInfo){
                TransactionService txService;
                if(transactionInfo.getType is InternalTransaction){
                        txService = FactoryMethod.getInternalTransactionServiceType(transactionInfo.getServiceType());
                }else if(transactionInfo.getType is ExternalTransaction){
                        txService = FactoryMethod.getExternalTransactionServiceType(transactionInfo.getServiceType());
                }
        txService.processTransaction(transactionInfo);
        }
}
```

Consequences:

Singleton pattern:

Problem statement: Ensuring the unique TransactionId and hash generation.

Context: when more than one transaction executing concurrently, we need to ensure there are no duplicate transaction id and hash. Maintaining a singleton object for Hash/TransactionId generation will solve the problem.

Forces/Constraints: supporting of multiple transaction id generations for concurrent requests will be delayed.

Solution – Static & Dynamics schematics: Keeping a singleton object for transaction id generation and maintaining the queue of transaction ids will solve the problem.

Consequences: