DESCRIPTION

Credit card fraud system detection

Contents

Functional USER Requirements:

1- Register: Allows Account Registration

1.1- The system shall allow a non-registered user to create a new account.

1.2- The system shall require the following information from the user: first name, last name, username, password, E-mail, credit card number, credit card password, amount, location, CVV Number, expiry Date, average Withdraw Money, the security questions, the security answers.

1.3- The system shall confirm the username and credit card number are acceptable (not taken).

1.4- The system shall store the information in the database.

2- Login: Allows Account Login

2.1- The system shall allow a registered user to log-in to their account.

2.2- The system shall require a username and password from the user.

2.3- The system will verify the username and password, and the user will be considered logged-in.

3- Withdraw: Allows users to withdraw money

3.1-The system shall allow a registered and logged-in user to withdraw money.

3.2-The system shall require a location and amount from the user. 3.3-The system shall store the operation details in the database.

4- Deposit: Allows users to deposit money

4.1-The system shall allow a registered and logged-in user to deposit money.

4.2-The system shall require amount from the user.

4.3-The system shall store the operation details in the database.

5- Detection: allows to detect fraud

5.1- System shall detect unusual patterns such user's country, transactions and any other features.

5.2- User shall answer the security questions.

5.3- if transaction of re-verification is invalid, system shall take appropriate action.

6- Update Account Information: Allows users to update account information

6.1- The system shall allow a registered and logged-in user to update their account information.

6.2- The user shall be allowed to view and change their information after answering the security questions.

7- Account Transactions History: Allows users to view account Transactions history

7.1- The system shall allow a registered and logged-in user to view Transactions (withdraw and deposit) made with their account.

7.2- The system shall display the date of transaction, transaction type, credit card number, location, amount.

8- Help: Allows users to get help by showing FAQs

8.1- The system shall allow the user to view an overview of how to use the various operations defined above.

8.2- The system must allow the user to select one of the operations by name.

8.3- The system shall then display information on how to use that operation.

9- Logout: Allows users account to logout

9.1- The system shall allow the registered and logged-in user to exit his/her account, so that access to operations requiring a user to be logged in are now disabled.

Functional ADMIN Requirements:

1- Login:

1.1- The system shall allow admin to log-in to his account.

1.2- The system shall require a username, name, password and Admin Id number from the admin.

1.3- The system shall confirm the username not taken.

2- View transactions: System allows admin to view all users transactions details.

3- View users: All the registered user details will be displayed to the admin.

4- View, add, delete and update admins accounts:

admin shall be able to add, delete and update admins accounts.

Non-Functional Requirements

1) Look-and-Feel REQ

-The product should use at most four colors.

2) Usability and humanity REQ

-The product shall be easy to use from older people.

-The product shall be easy to use by who can’t read English.

-The product shall be easy to use on the first attempt by a member of the public without training.

-The product should explain to user how it works for new coming.

-The product should accept different kind of credit card.

3) Performance requirements

-The system shall verify e-mail of new registered user by sending verification message to the e-mail within 0.30 second.

-The system shall ask new registered three security questions.

-The system shall validate, verify and check the credit card by sending verification message to the bank within 0.5 second.

-The system shall view all transactions that user needs once asked from system within 0.2 second.

-The system shall use real-time payments fraud by monitoring every transaction happens and detect suspicious behavior of standard pattern of history.

-Based on user’s characteristic of spending, country and other patterns the system shall stop any transaction immediately if a deviation more than 30% withdraw is detected.

-The system shall re-verification by answering security questions to re-sign.

-If re-verification failed or more than 3 attempt invalid for answering security questions The system shall report the admin about fraud and block the user.

-The product shall, on average, operate without failure for1 days

4) Operational & Environmental REQs

-The product will be used in website so it must be protected against DDOs attacks.

5) Maintainability

-Any updates or defect fixes shall be able to be made on server-side computers only without any patches required by the user.

-The system must be easy to use.

-Users should be able to learn this system very quickly. It should be fairly intuitive.

-The reports should be clear and precise, and not overly technical.

-The information produced by the system should be understandable by anyone with general knowledge.

6) Cultural REQs

- The language used in the interface should be formal and polite.

-The product shall not display religious symbols or words.

7) Legal REQs

-The credit card belongs to 21 or older person.

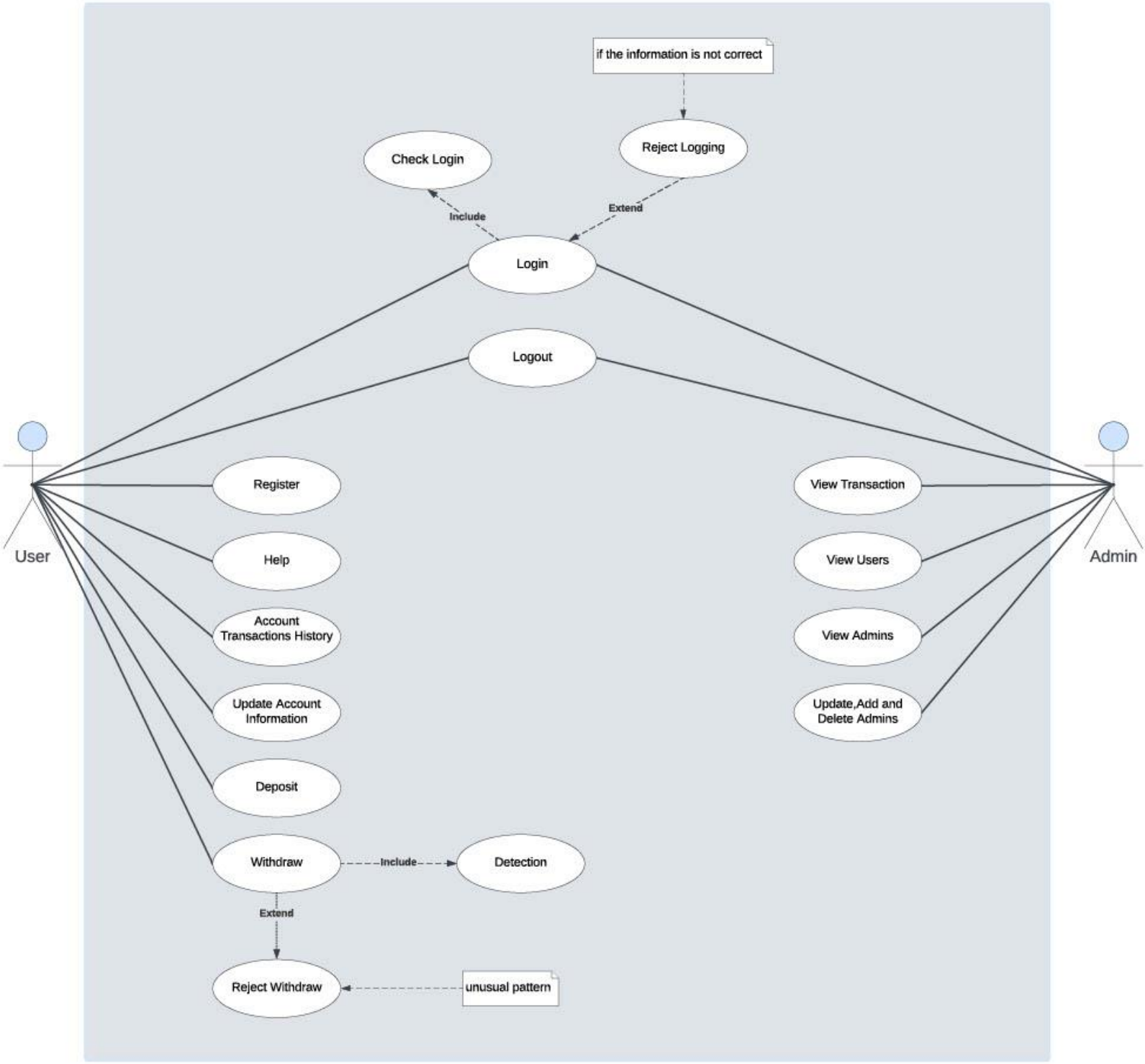
-The credit card is not fake or stolen.

8) Security REQs

-the product shall ensure that only authorized users are able to gain access.

-the product shall distinguish between authorized and non-authorized users.

USE-CASE


if the information is not correct
Reject Logging
Check Login
Include
Extend
Login
Logout
User
Admin
Register
Help
Account Transactions History
Update Account Information
Deposit
Withdraw
Include
Detection
Extend
Reject Withdraw
unusual pattern
View Transaction
View Users
View Admins
Update,Add and Delete Admins

The description of the use case.

The use case identification	register && 1.
The pre-condition.	None.
The main success scenario.	1) The new user will enter on website. 2) The user will full his information. 3) The system will check the information. 4) The system will accept this information. 5) The user will emphasize his information. 6) The system will store them in the database.
Post condition.	The user now have account on the system.
The goal of the use case.	Create an account on the system.

The use case identification	Log In && 2.
The pre-condition.	The User or Admin should have an account.
The main success scenario.	1) The user or the admin will enter his data (username and password). 2) The system will valid his account in the database. 3) The system will accept logging.
Post condition.	User/admin will be active on the system.
The goal of the use case.	Connect with the system.

The use case identification	Check logging &&3
The pre-condition	The user must enter the Log In information.
The main success scenario	1) System will search in the database about the logging information 2) The system will check the validation. 3) The system will accept the information or not.
The post condition	The accept logging will be active on the system.
The goal of the use case	Save the client.

The use case identification	Reject logging && 4.
The pre-condition.	The user or the admin must be entered his information account.
The main success scenario.	1) The user or the admin will enter his data logging. 2) The system will find invalid logging. 3) The system will not accept the logging.
Post condition.	The reject logging will be not active on the system.
The goal of the use case.	Save the protection.

The use case identification	Help && 5.
The pre-condition.	None.
The main success scenario.	1) The user finds any problem for doing anything. 2) The user will go to help page. 3) The system will show how to do the operation. 4) The user will follow the system.
Post condition.	The user will be able to do any operation on the system.
The goal of the use case.	Help the user (s).

The use case identification	Update account information && 6.
The pre-condition.	The user has an account on the system&& logging on the system.
The main success scenario.	1) The user wants to change or update information. 2) The system will ask for new updates. 3) The user will enter the new changing. 4) The user will conform the new changing. 5) The system will store the new changing in the database.
Post condition.	The account of the user must be changed with new data in the database.
The goal of the use case.	Make changing or update.

The use case identification	Update, add and delete admins && 7.
The pre-condition.	The admin has an account on the system && logging on the system.

The main success scenario.	1) The admin wants to make some update. 2) The system will ask for the new updates (add, delete, or update). 3) The admin will select the operation. 4) The system will ask for confirm. 5) The admin will confirm. 6) The system will perform the operation. 7) The system will store the changing in the database.
Post condition.	The account of the admin must be changed.
The goal of the use case.	Make change or update on the admin's account.

The use case identification	Withdraw &&89.
The pre-condition.	Logging on the system and has a valeted credit card.
The main success scenario	1) The user wants to make a withdraw form credit card. 2) The system will check if there is an unusual pattern. 3) The system will ask about the amount. 4) The system will accept the operation. 5) The system must store the details of the operation in the database.
Post condition.	The amount of money must change in the credit card after the operation
The goal of the use case.	Make withdraw from the credit card.

The use case identification	Reject the withdraw && 10.
The pre-condition.	The user is logging on the system.

The main success scenario.	1) The system will ask for credit card information. 2) The system will find unusual pattern. 3) The system will reject the operation.
Post condition.	The system does not accept any the operation after find unusual pattern.
The goal of the use case.	Save user from attacking.

The use case identification	deposit &&11.
The pre-condition.	The user logging on the system&.
The main success scenario.	1) The user wants to make a deposit. 2) The system will accept the operation. 3) The system will store the operation's details in the database.
Post condition.	The amount of money must change in the credit card after the operation
The goal of the use case.	Making deposit.

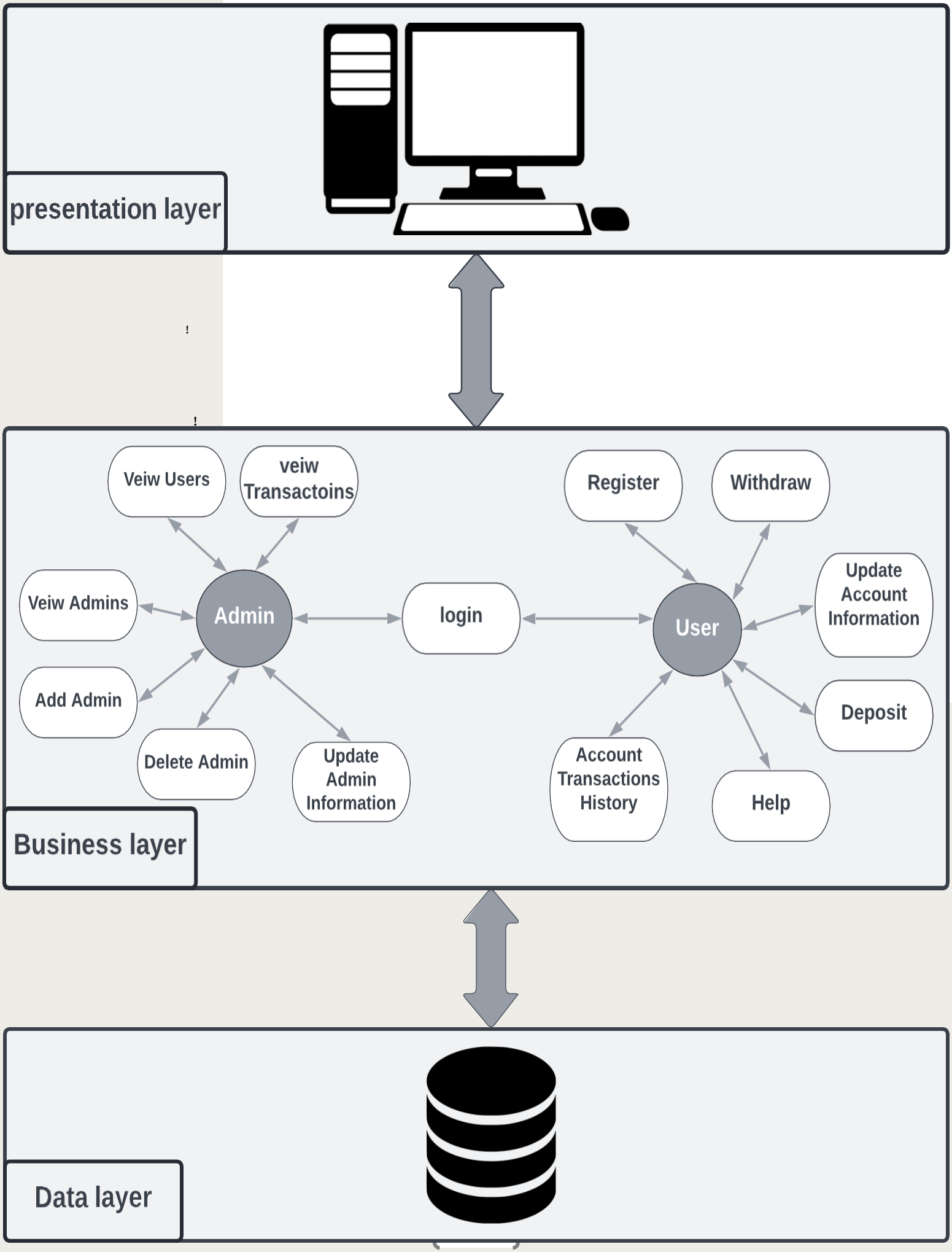
The use case identification	View transaction && 11.
The pre-condition.	The admin is logging on the system.
The main success scenario.	1) The admin wants to view the transaction. 2) The system will search in the database. 3) The system will show all transaction to the admin.
Post condition.	None.

The goal of the use case.	Show all transaction from user.

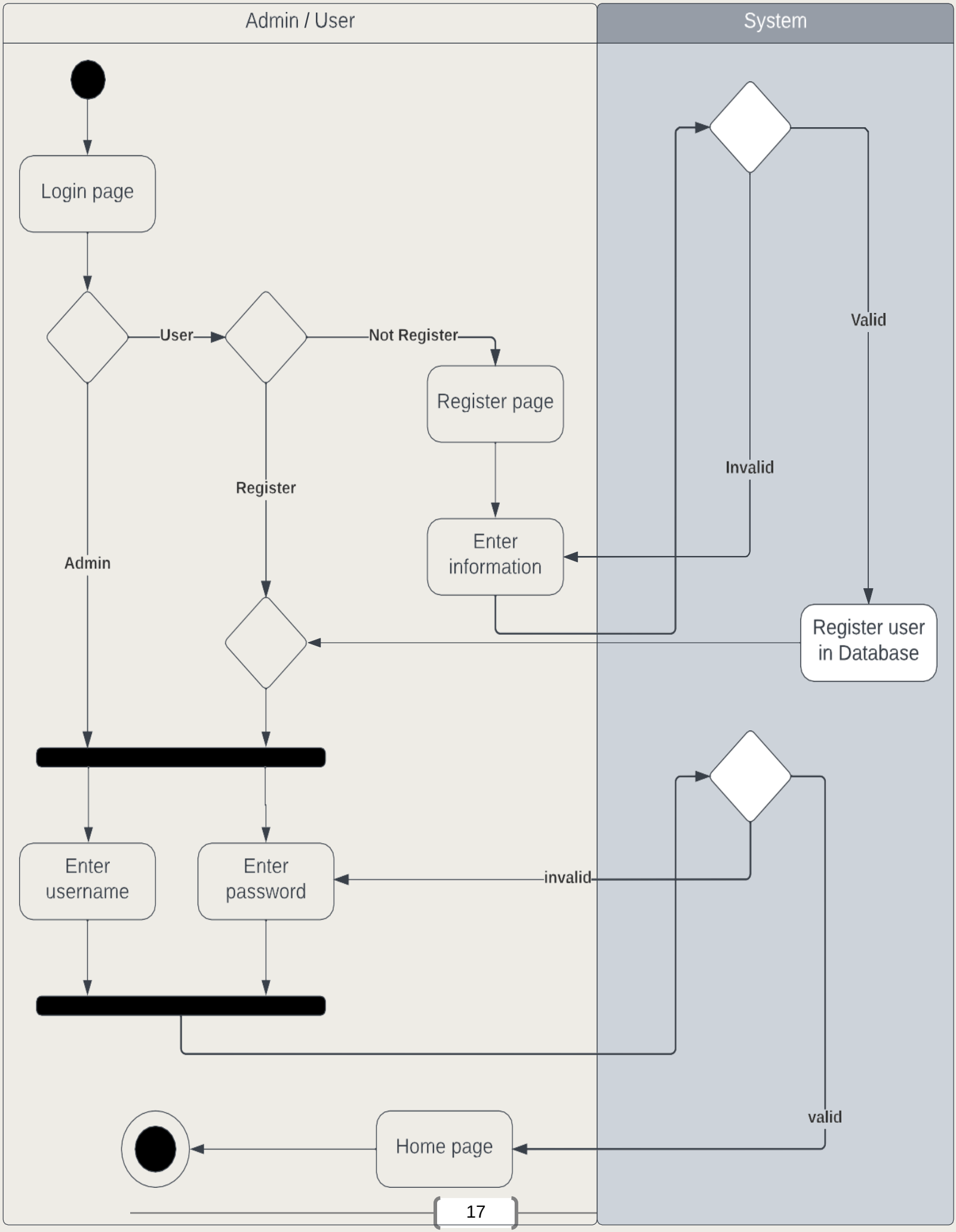
The use case identification	Account Transactions History && 13.
The pre-condition.	The user is logging on the system.
The main success scenario.	4) The user wants to see the transactions history of his account. 5) The system will ask for the specific operation. 6) The system will search in the database. 7) The system will display the transactions history to user.
Post condition.	**None.**
The goal of the use case.	display the transactions history to user.

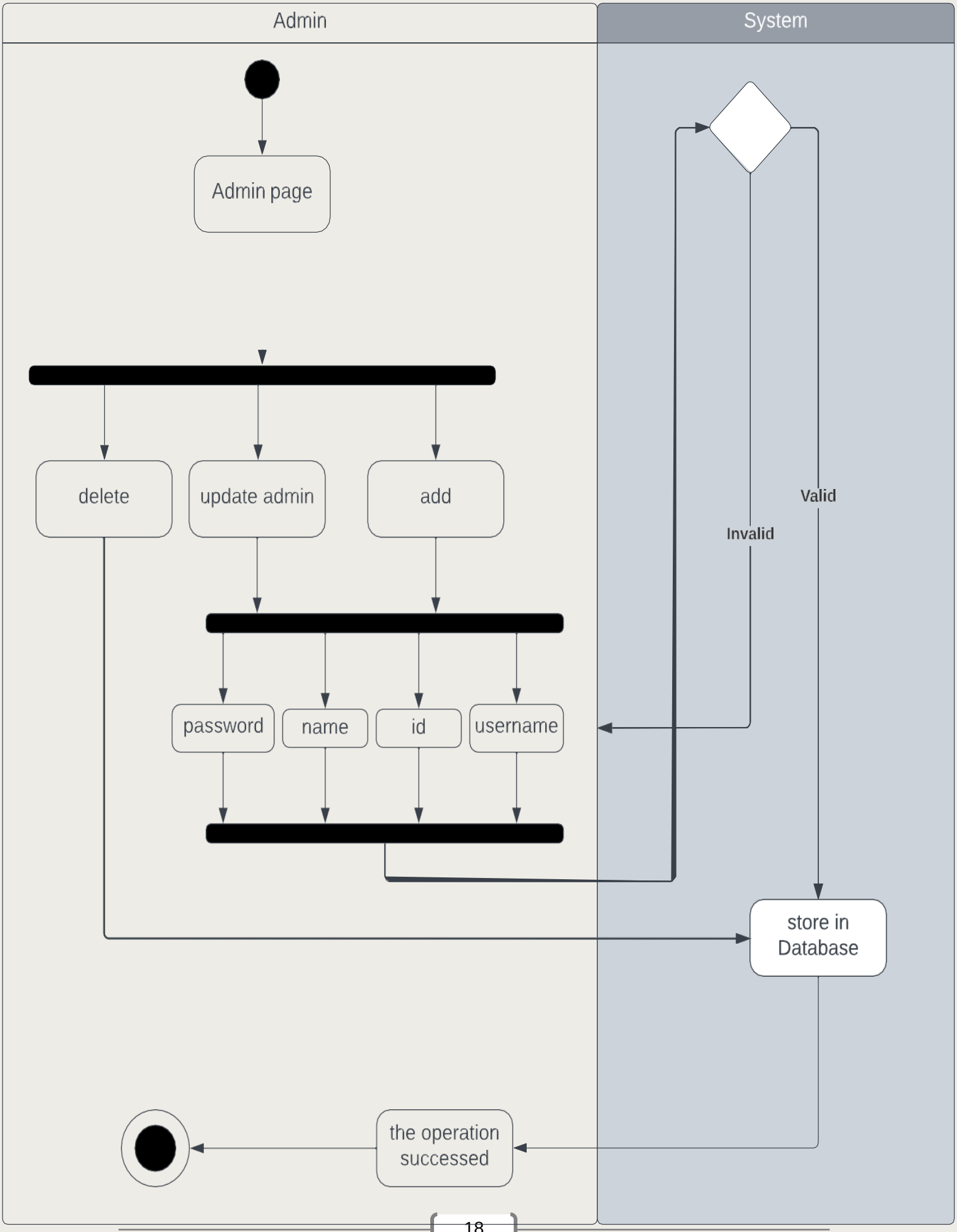
The use case identification	Log out &&14.
The pre-condition.	The user /admin is logging on the system.
The main success scenario.	1) The user or the admin wants to log out. 2) The system will ask them for confirm. 3) The system will perform the operation.
Post condition.	The user or the admin can't do anything until logging again
The goal of the use case.	Logging out from the system

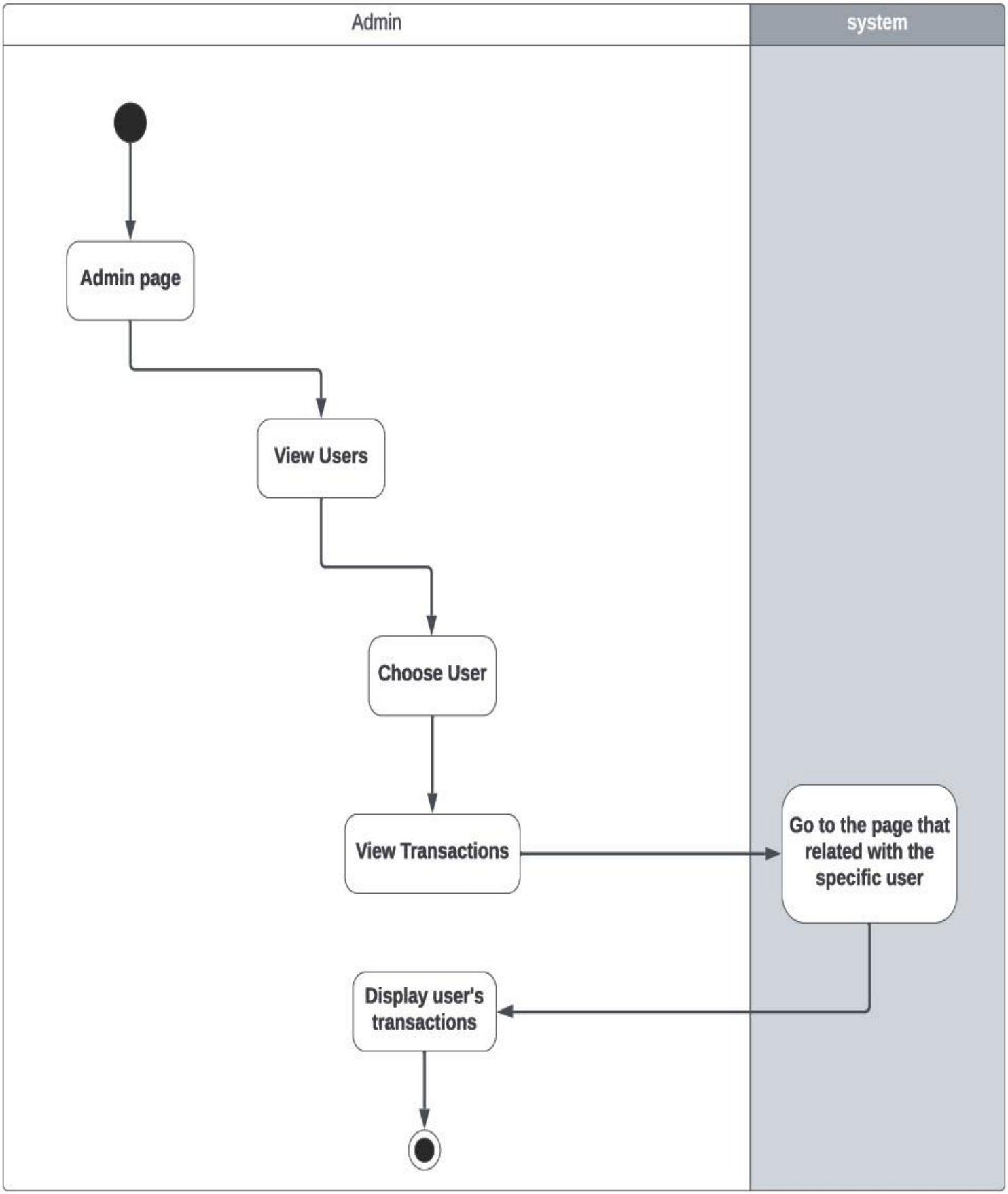
LAYERED SYSTEM ARCHITECTURES

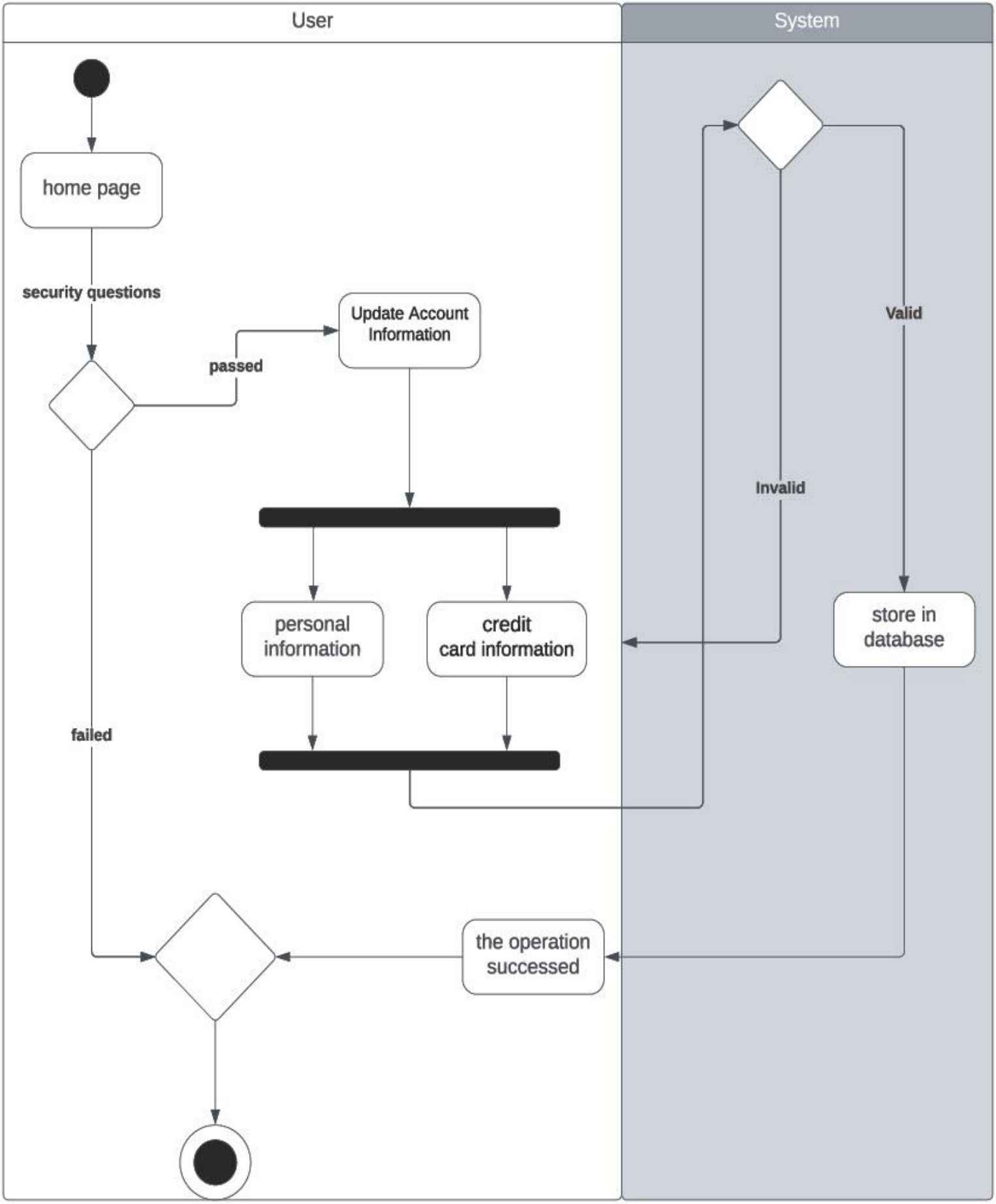

presentation layer
Veiw Users
veiw Transactoins
Veiw Admins
Admin
Add Admin
Delete Admin
Update Admin Information
login
Register
Withdraw
User
Update Account Information
Deposit
Account Transactions History
Help
Business layer
Data layer

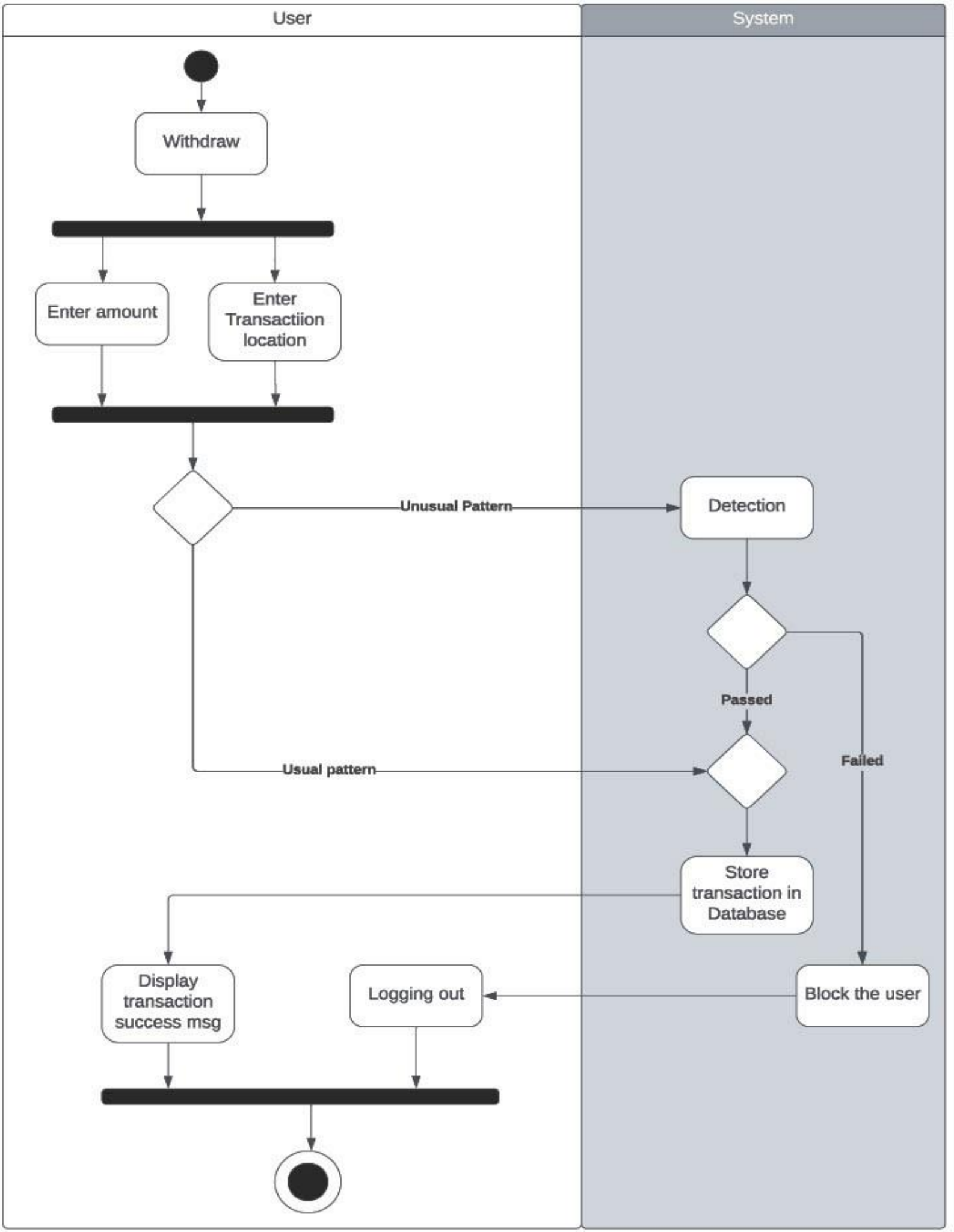
ACTIVITY DIAGRAM


Admin / User
System
Login page
User
Not Register
Register page
Valid
Invalid
Register
Enter information
Admin
Register user in Database
Enter username
Enter password
invalid
valid
Home page

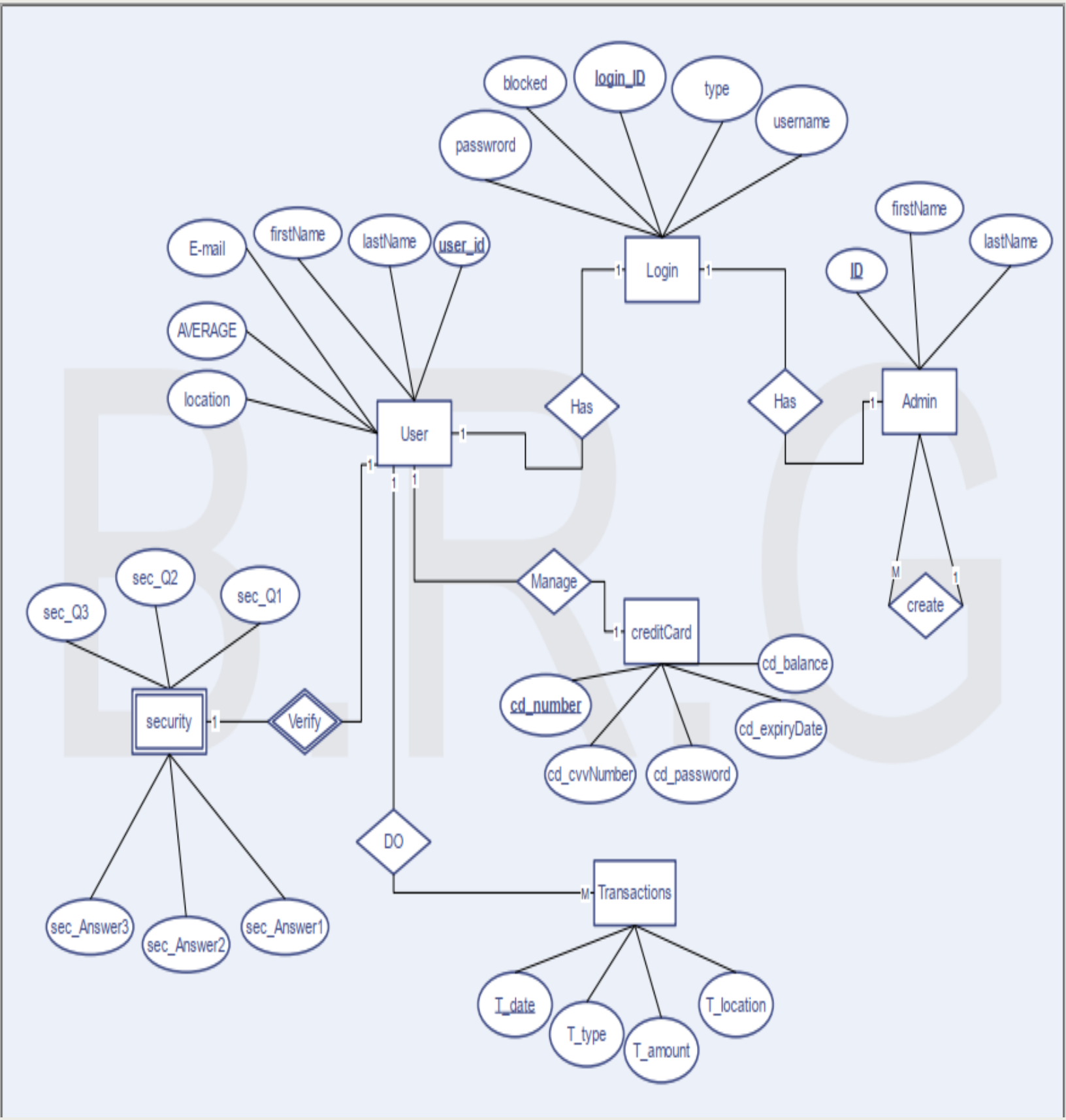

Admin
System
Admin page
delete
update admin
add
password
name
id
username
Valid
Invalid
store in Database
the operation successed

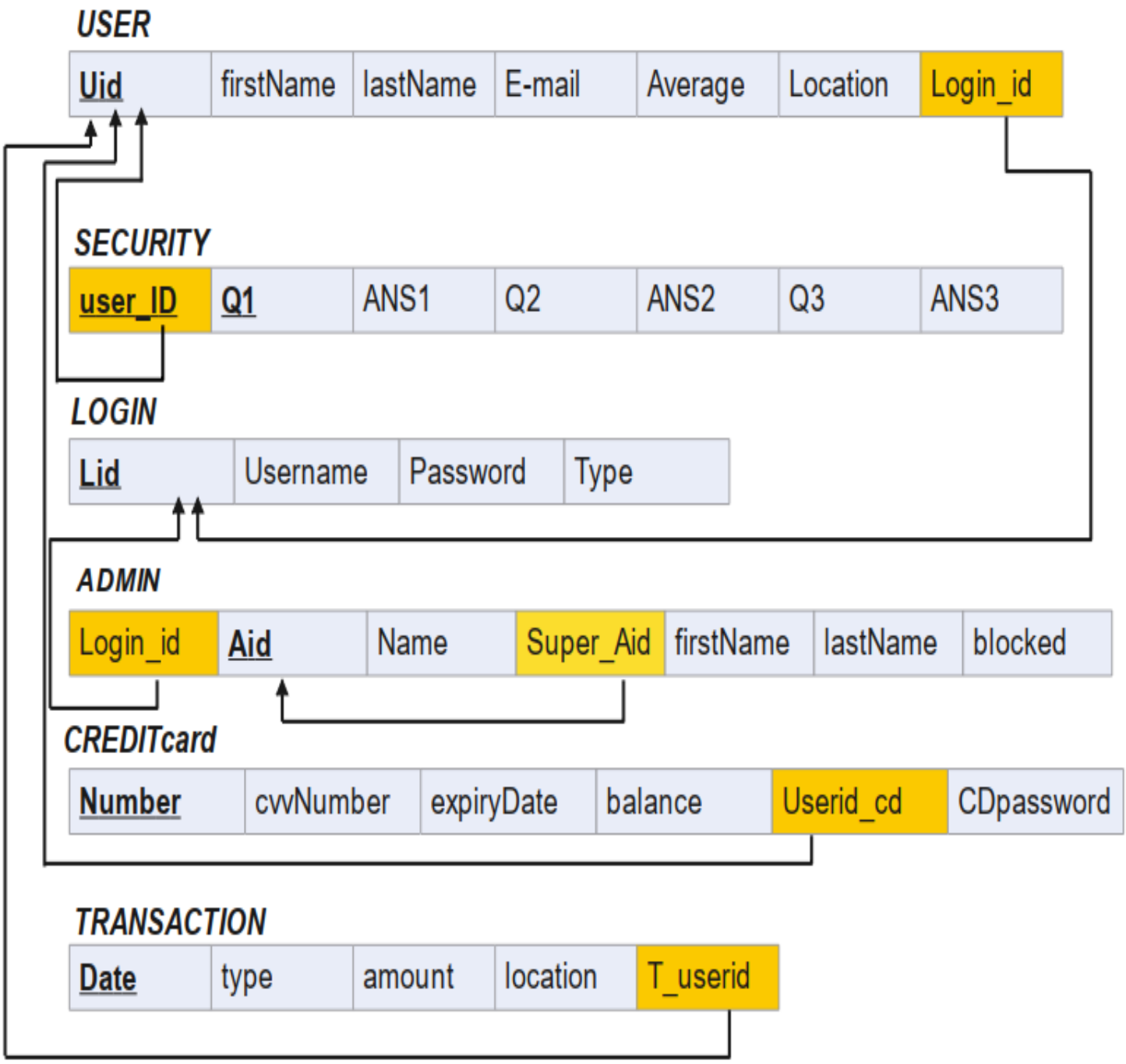

Admin
system
Admin page
View Users
Choose User
View Transactions
Go to the page that related with the specific user
Display user's transactions


User
System
home page
security questions
Update Account Information
passed
Valid
Invalid
personal information
credit card information
store in database
failed
the operation successed

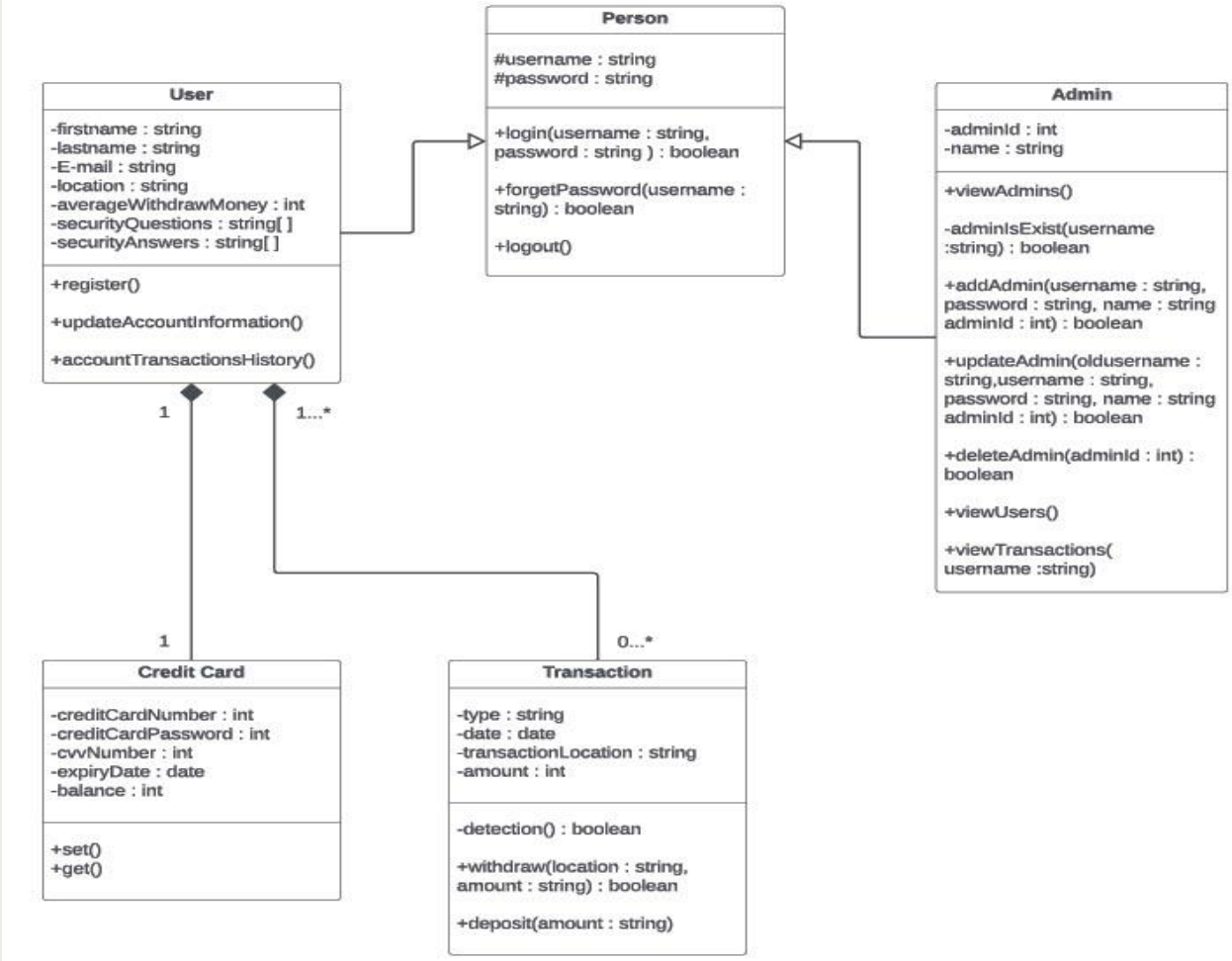

User
System
Withdraw
Enter amount
Enter Transactiion location
Unusual Pattern
Detection
Passed
Failed
Usual pattern
Store transaction in Database
Display transaction success msg
Logging out
Block the user

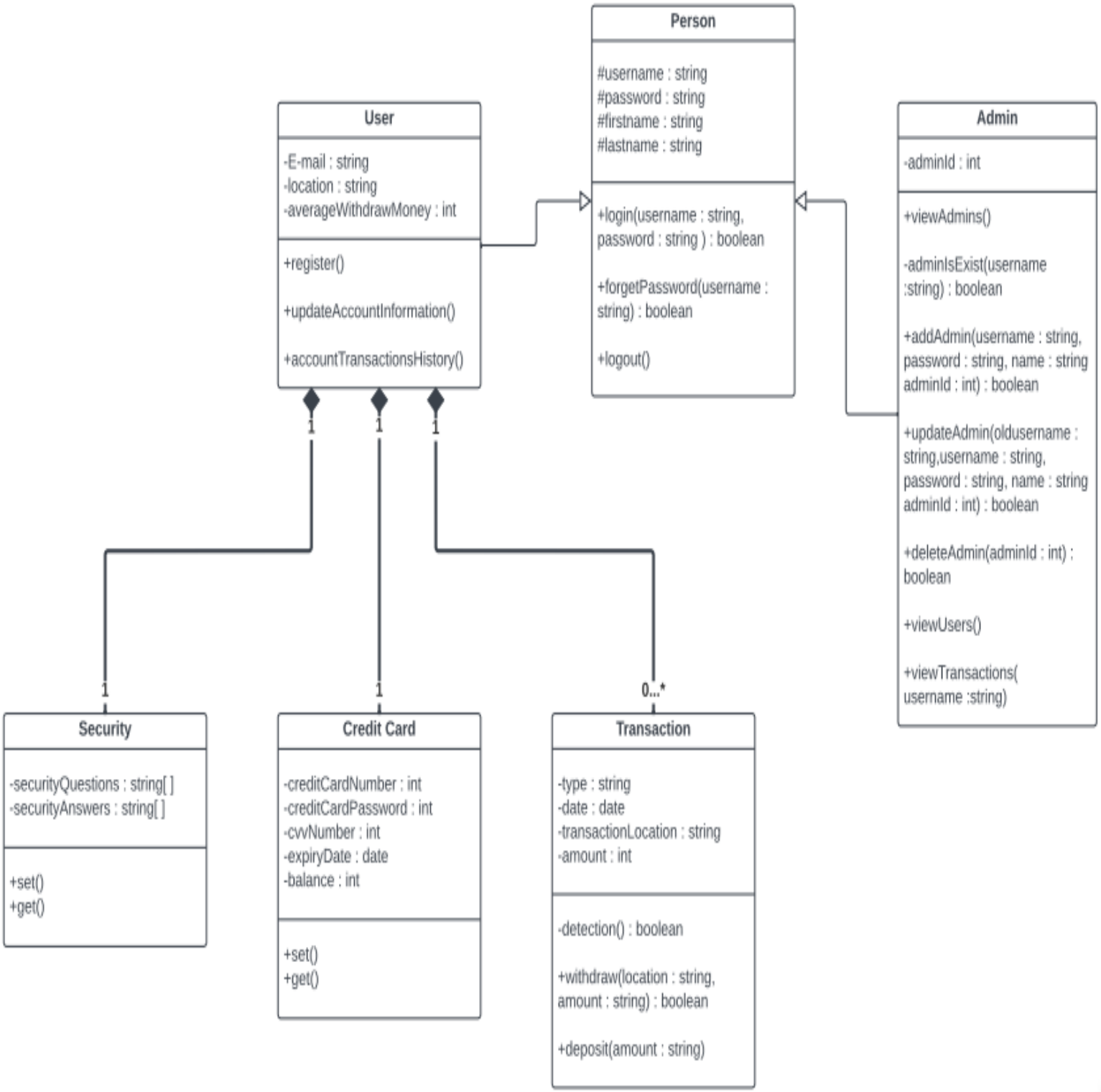
DATABASE SPECIFICATION

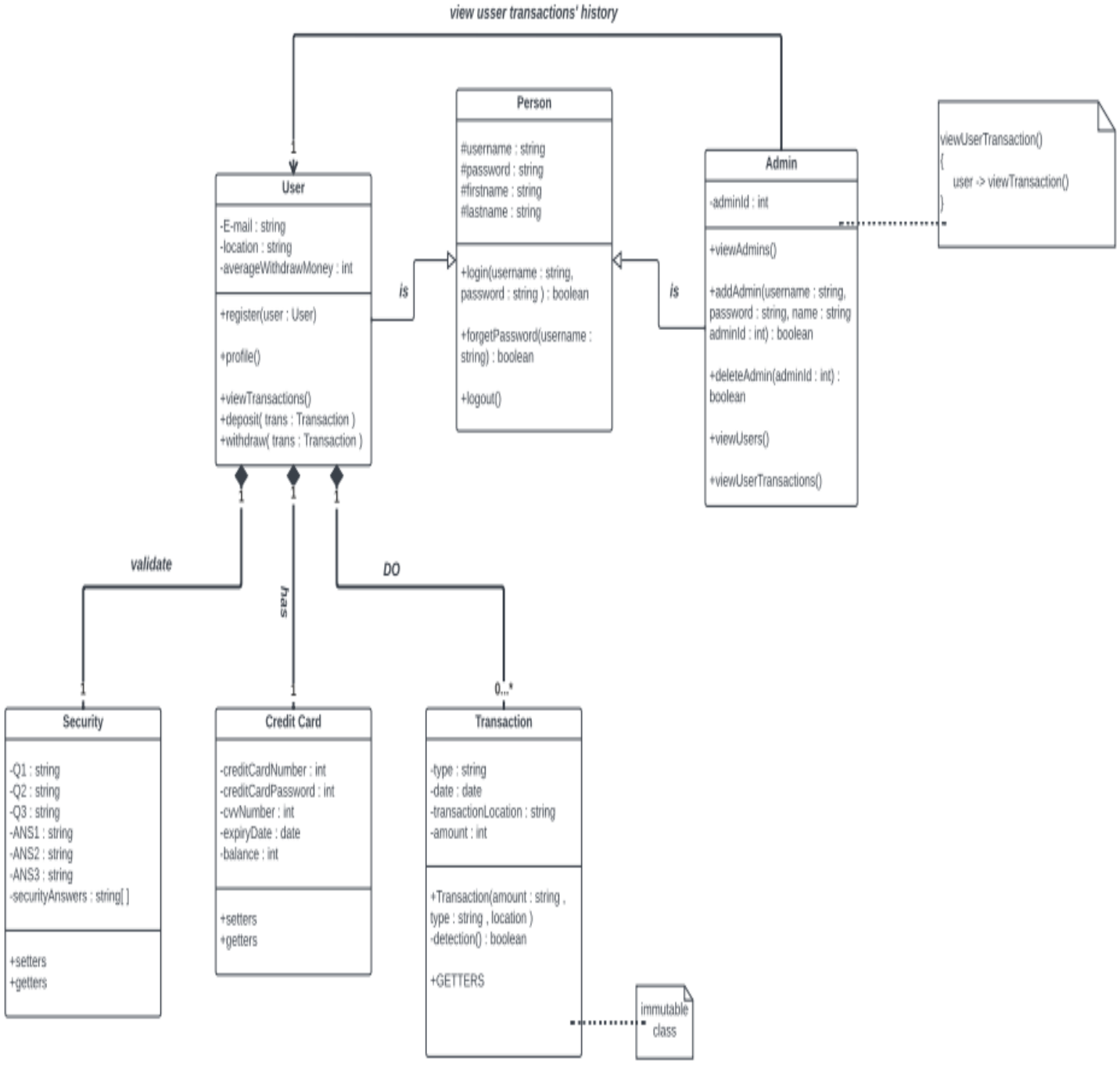

blocked
login_ID
type
username
passwrord
firstName
lastName
E-mail
user_id
Login
ID
firstName
lastName
AVERAGE
location
User
Has
Has
Admin
1
1
1
1
1
1
1
1
M
1
create
Manage
creditCard
cd_balance
cd_number
cd_expiryDate
cd_cvvNumber
cd_password
sec_Q2
sec_Q3
sec_Q1
security
Verify
DO
M
Transactions
sec_Answer3
sec_Answer2
sec_Answer1
T_date
T_type
T_amount
T_location


USER
Uid
firstName
lastName
E-mail
Average
Location
Login_id
SECURITY
user_ID
Q1
ANS1
Q2
ANS2
Q3
ANS3
LOGIN
Lid
Username
Password
Type
ADMIN
Login_id
Aid
Name
Super_Aid
firstName
lastName
blocked
CREDITcard
Number
cvvNumber
expiryDate
balance
Userid_cd
CDpassword
TRANSACTION
Date
type
amount
location
T_userid

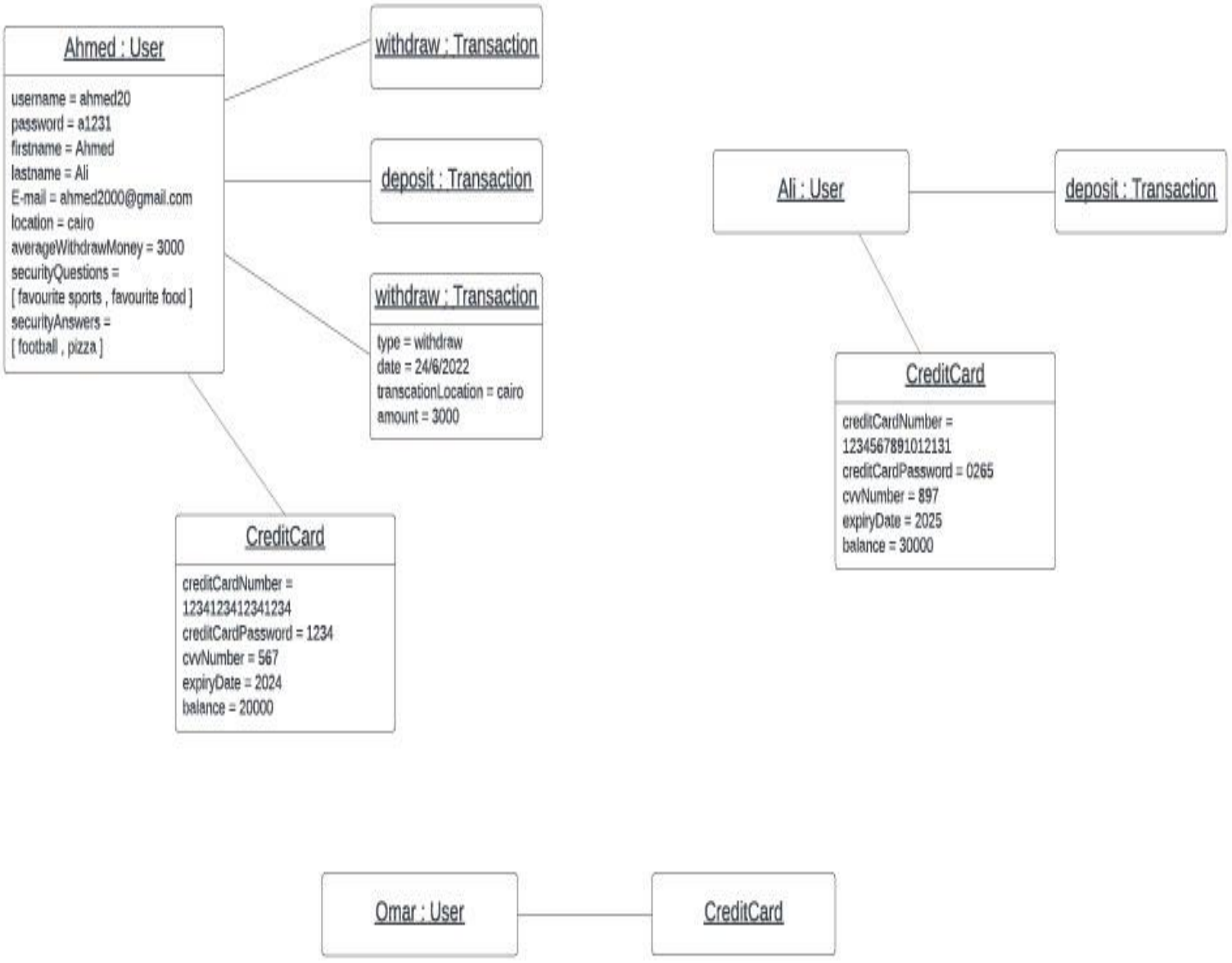
CLASS DIAGRAM


Person
#username : string
#password : string
+login(username : string, password : string) : boolean
+forgetPassword(username : string) : boolean
+logout()
User
-firstname : string
-lastname : string
-E-mail : string
-location : string
-averageWithdrawMoney : int
-securityQuestions : string[ ]
-securityAnswers : string[ ]
+register()
+updateAccountInformation()
+accountTransactionsHistory()
Admin
-adminId : int
-name : string
+viewAdmins()
-adminIsExist(username :string) : boolean
+addAdmin(username : string, password : string, name : string adminId : int) : boolean
+updateAdmin(oldusername : string,username : string, password : string, name : string adminId : int) : boolean
+deleteAdmin(adminId : int) : boolean
+viewUsers()
+viewTransactions(username :string)
1
1...*
1
0...*
Credit Card
-creditCardNumber : int
-creditCardPassword : int
-cvvNumber : int
-expiryDate : date
-balance : int
+set()
+get()
Transaction
-type : string
-date : date
-transactionLocation : string
-amount : int
-detection() : boolean
+withdraw(location : string, amount : string) : boolean
+deposit(amount : string)

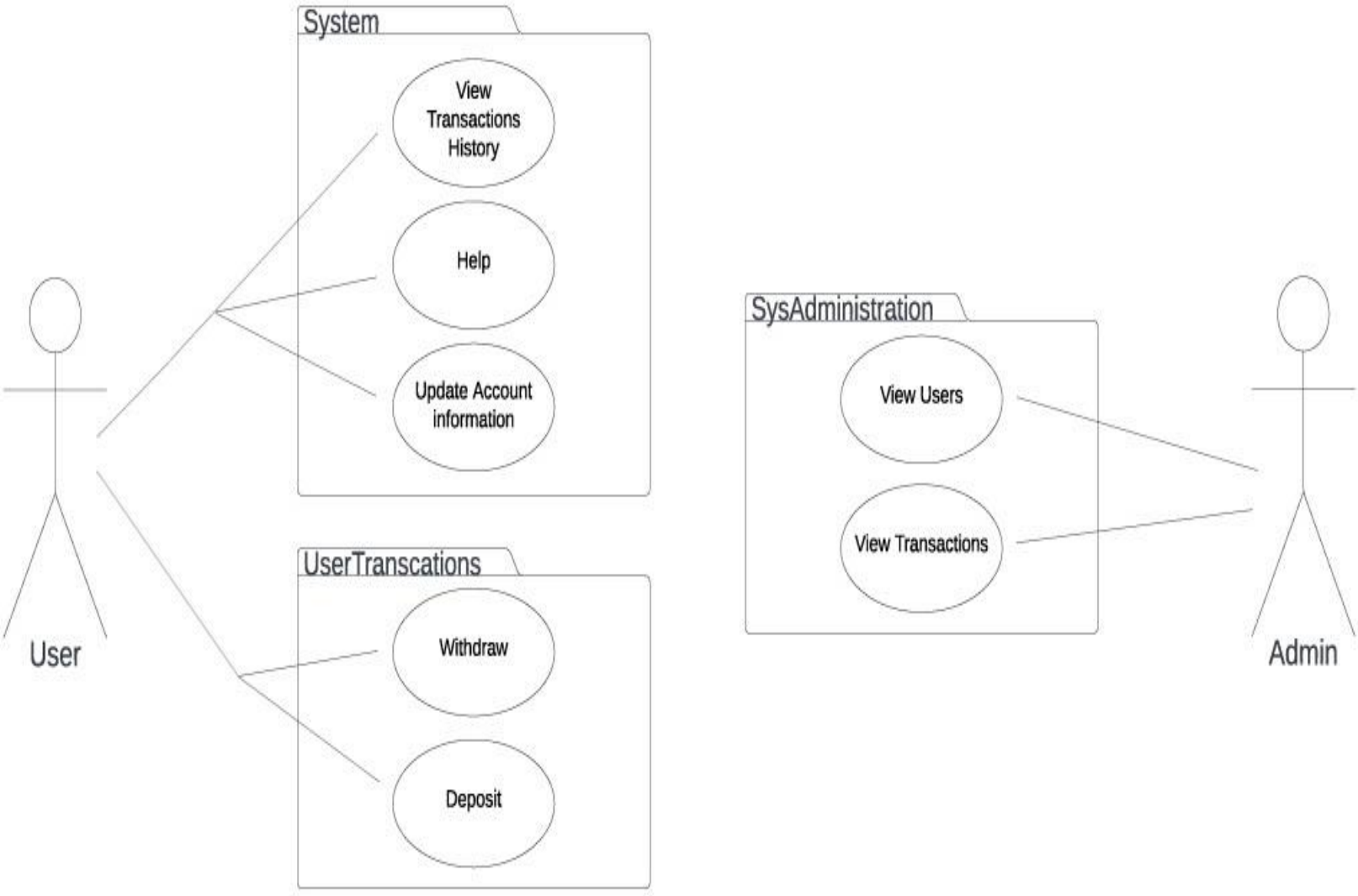

Person
#username : string
#password : string
#firstname : string
#lastname : string
+login(username : string, password : string) : boolean
+forgetPassword(username : string) : boolean
+logout()
User
-E-mail : string
-location : string
-averageWithdrawMoney : int
+register()
+updateAccountInformation()
+accountTransactionsHistory()
Admin
-adminId : int
+viewAdmins()
-adminIsExist(username :string) : boolean
+addAdmin(username : string, password : string, name : string adminId : int) : boolean
+updateAdmin(oldusername : string,username : string, password : string, name : string adminId : int) : boolean
+deleteAdmin(adminId : int) : boolean
+viewUsers()
+viewTransactions(username :string)
1
1
1
1
1
0...*
Security
-securityQuestions : string[ ]
-securityAnswers : string[ ]
+set()
+get()
Credit Card
-creditCardNumber : int
-creditCardPassword : int
-cvvNumber : int
-expiryDate : date
-balance : int
+set()
+get()
Transaction
-type : string
-date : date
-transactionLocation : string
-amount : int
-detection() : boolean
+withdraw(location : string, amount : string) : boolean
+deposit(amount : string)


view usser transactions' history
Person
#username : string
#password : string
#firstname : string
#lastname : string
+login(username : string, password : string) : boolean
+forgetPassword(username : string) : boolean
+logout()
User
-E-mail : string
-location : string
-averageWithdrawMoney : int
+register(user : User)
+profile()
+viewTransactions()
+deposit(trans : Transaction)
+withdraw(trans : Transaction)
Admin
-adminId : int
+viewAdmins()
+addAdmin(username : string, password : string, name : string adminId : int) : boolean
+deleteAdmin(adminId : int) : boolean
+viewUsers()
+viewUserTransactions()
viewUserTransaction()
{
user -> viewTransaction()
}
is
is
validate
has
DO
1
0...*
Security
-Q1 : string
-Q2 : string
-Q3 : string
-ANS1 : string
-ANS2 : string
-ANS3 : string
-securityAnswers : string[ ]
+setters
+getters
Credit Card
-creditCardNumber : int
-creditCardPassword : int
-cvvNumber : int
-expiryDate : date
-balance : int
+setters
+getters
Transaction
-type : string
-date : date
-transactionLocation : string
-amount : int
+Transaction(amount : string , type : string , location)
-detection() : boolean
+GETTERS
immutable class

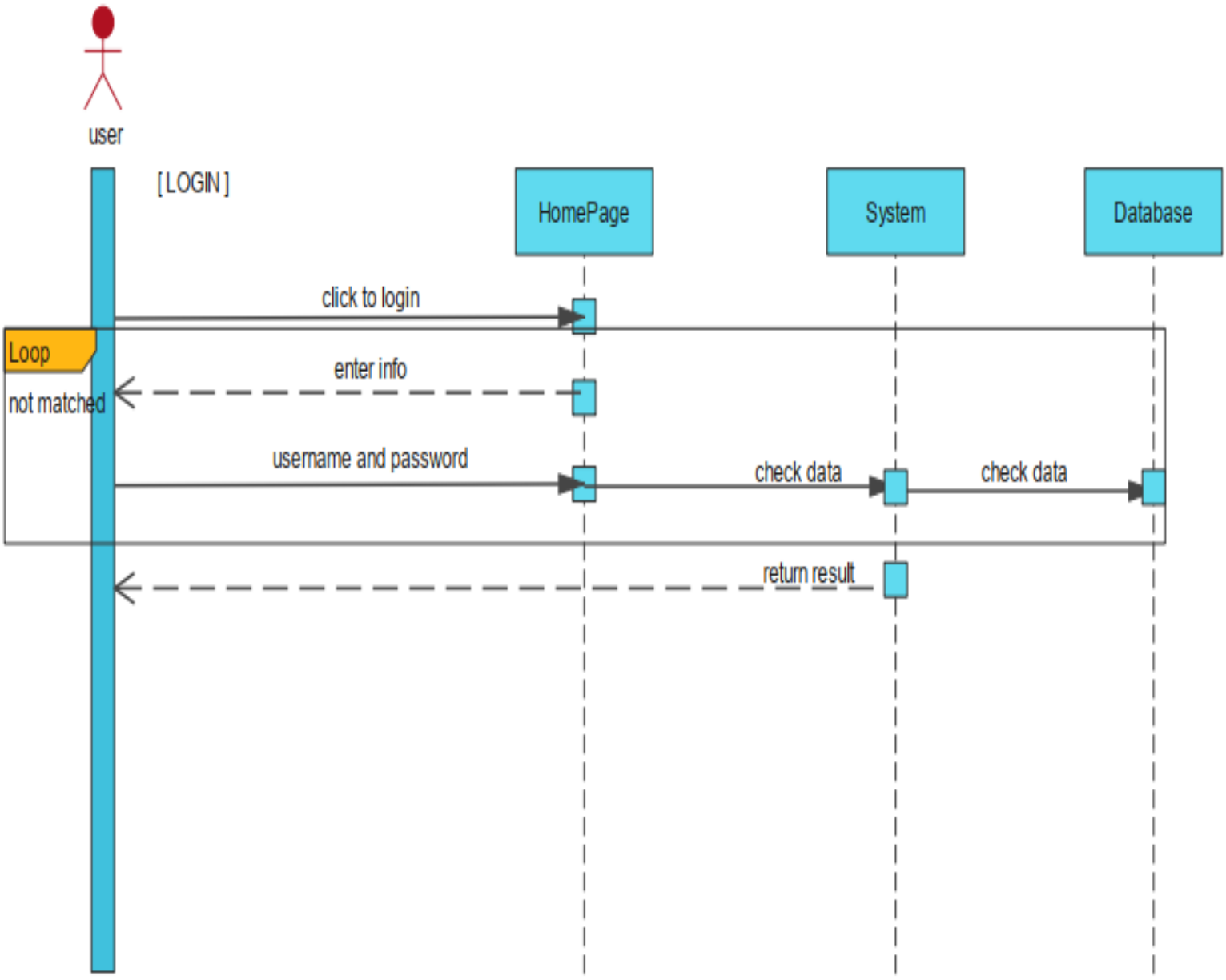
OBJECT DIAGRAM

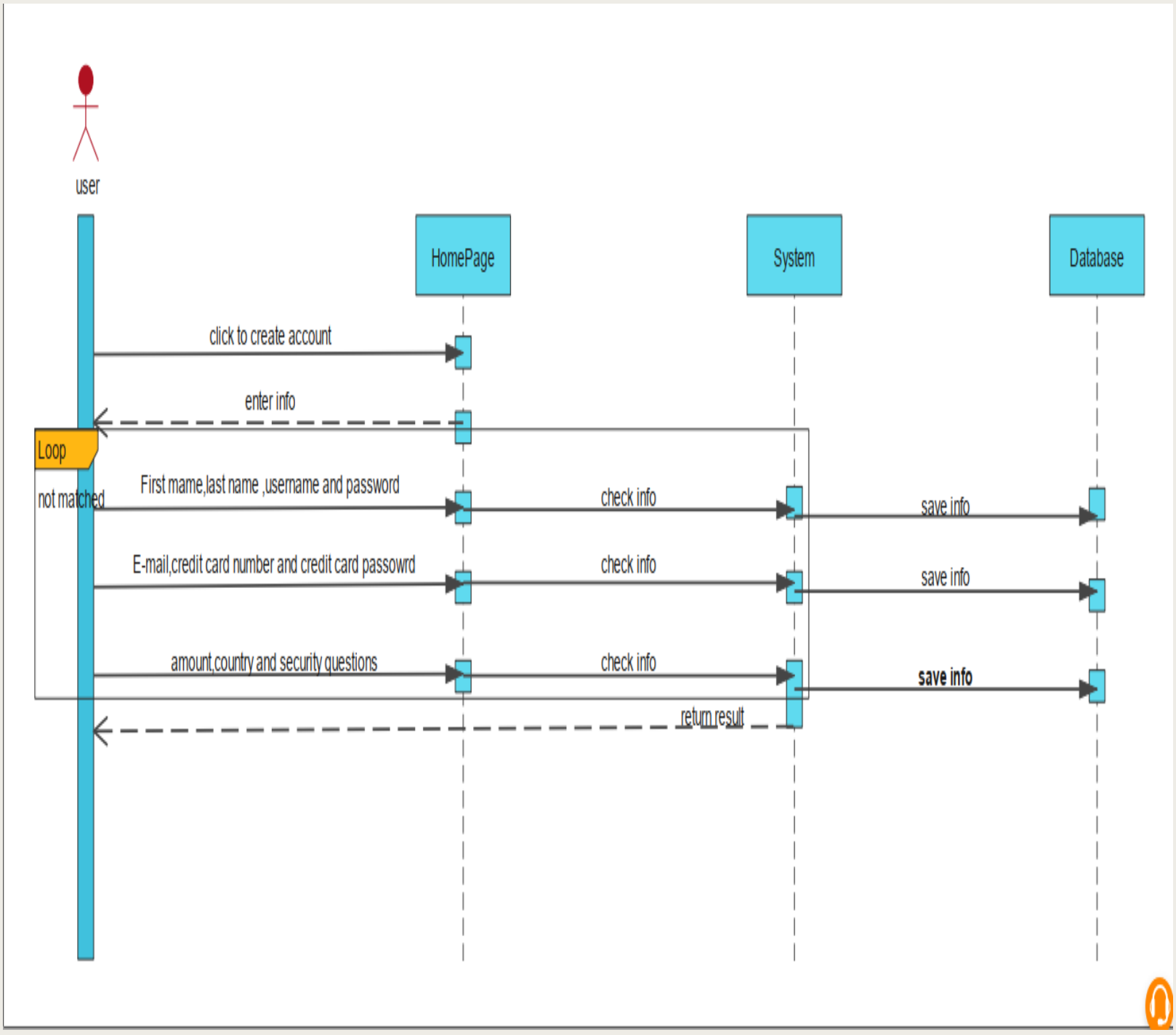

Ahmed : User
username = ahmed20
password = a1231
firstname = Ahmed
lastname = Ali
E-mail = ahmed2000@gmail.com
location = cairo
averageWithdrawMoney = 3000
securityQuestions =
[favourite sports , favourite food]
securityAnswers =
[football , pizza]
withdraw : Transaction
deposit : Transaction
withdraw : Transaction
type = withdraw
date = 24/6/2022
transcationLocation = cairo
amount = 3000
CreditCard
creditCardNumber =
1234123412341234
creditCardPassword = 1234
cvvNumber = 567
expiryDate = 2024
balance = 20000
Ali : User
deposit : Transaction
CreditCard
creditCardNumber =
1234567891012131
creditCardPassword = 0265
cvvNumber = 897
expiryDate = 2025
balance = 30000
Omar : User
CreditCard

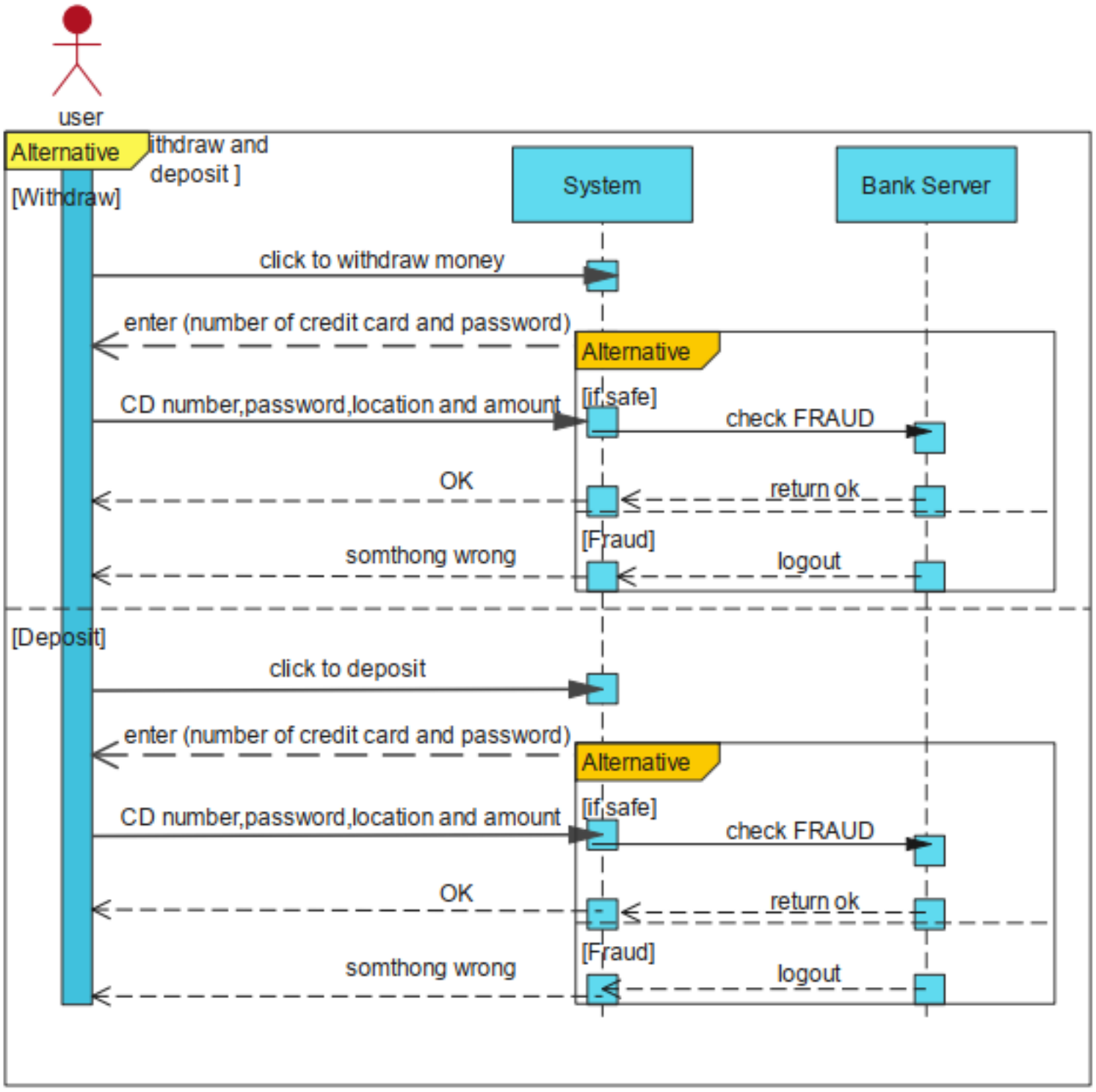
PACKAGE DIAGRAM

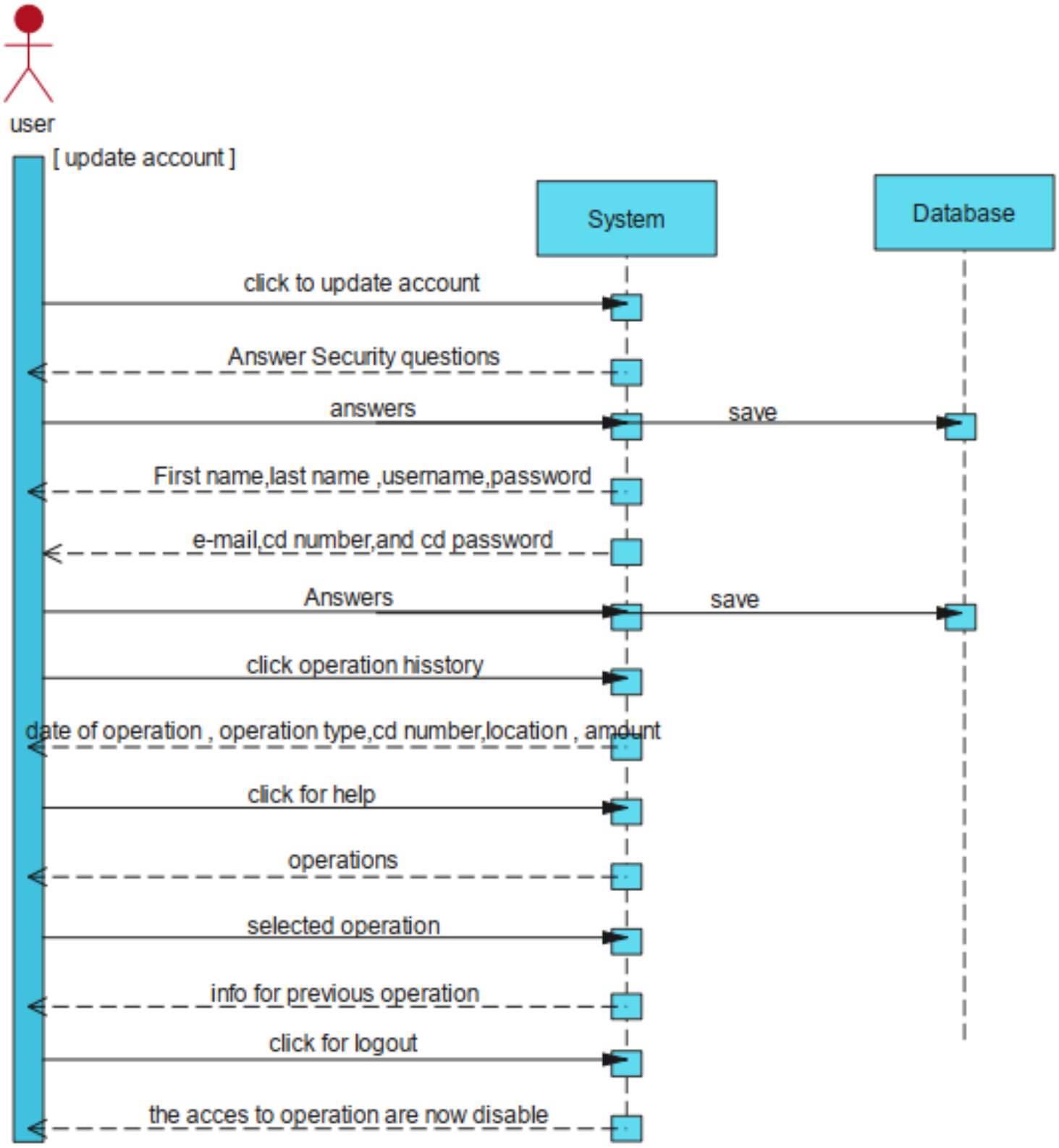

System
View Transactions History
Help
Update Account information
UserTranscations
Withdraw
Deposit
User
SysAdministration
View Users
View Transactions
Admin

SYQUENCE DIAGRAM

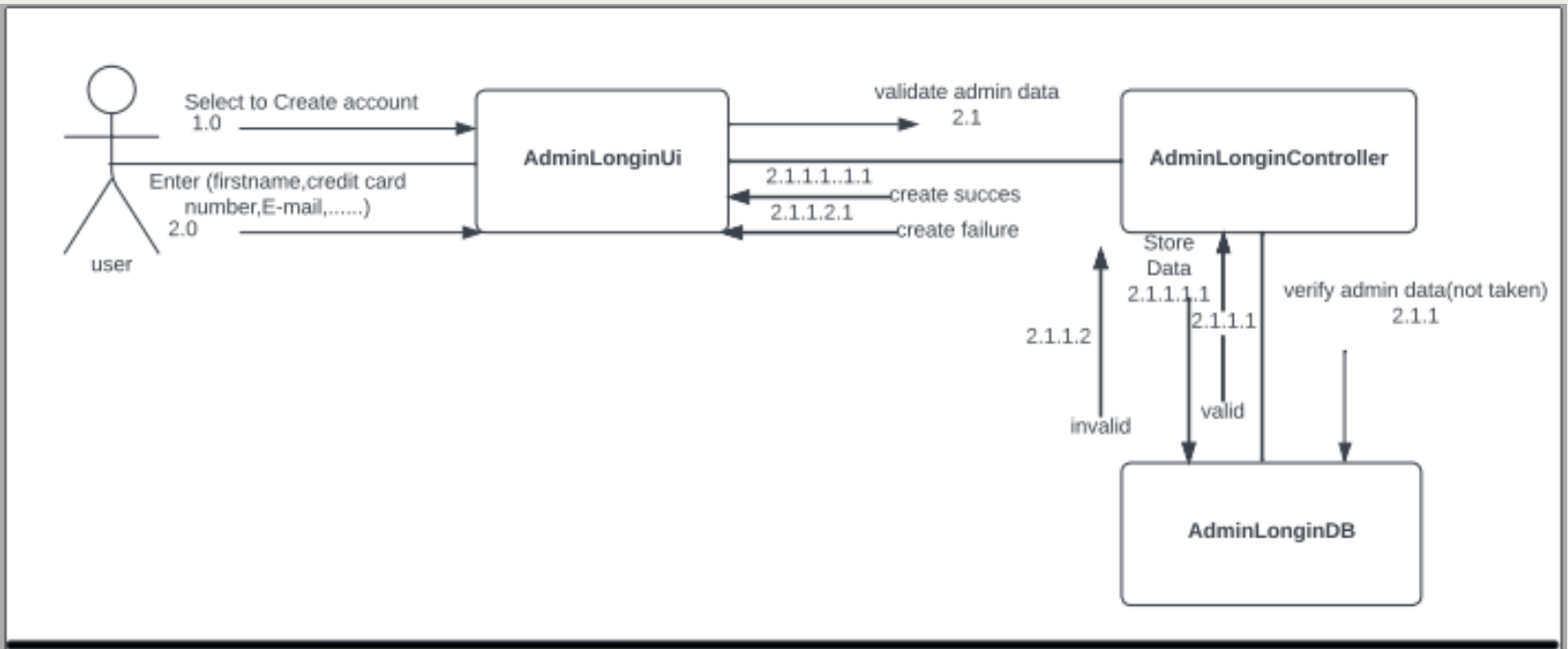

user
[LOGIN]
HomePage
System
Database
click to login
Loop
enter info
not matched
username and password
check data
check data
return result

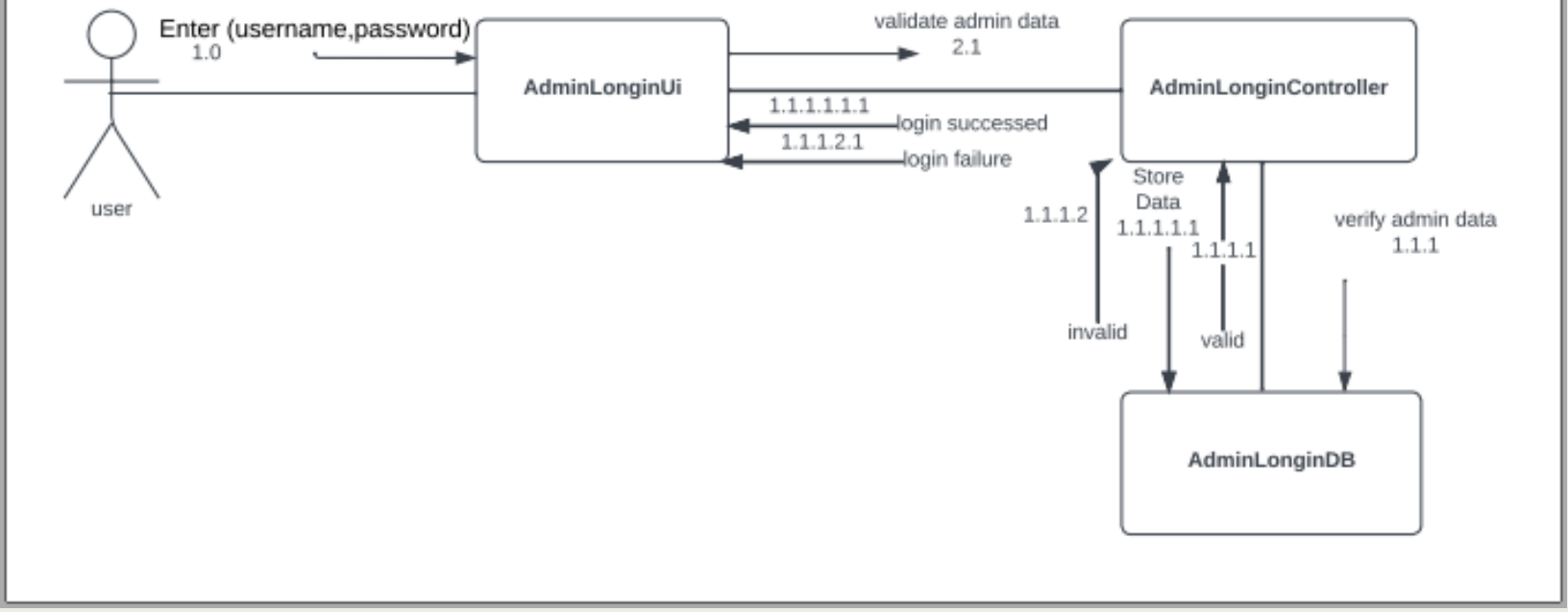

user
HomePage
System
Database
click to create account
enter info
Loop
not matched
First mame,last name ,username and password
check info
save info
E-mail,credit card number and credit card passowrd
check info
save info
amount,country and security questions
check info
save info
return result

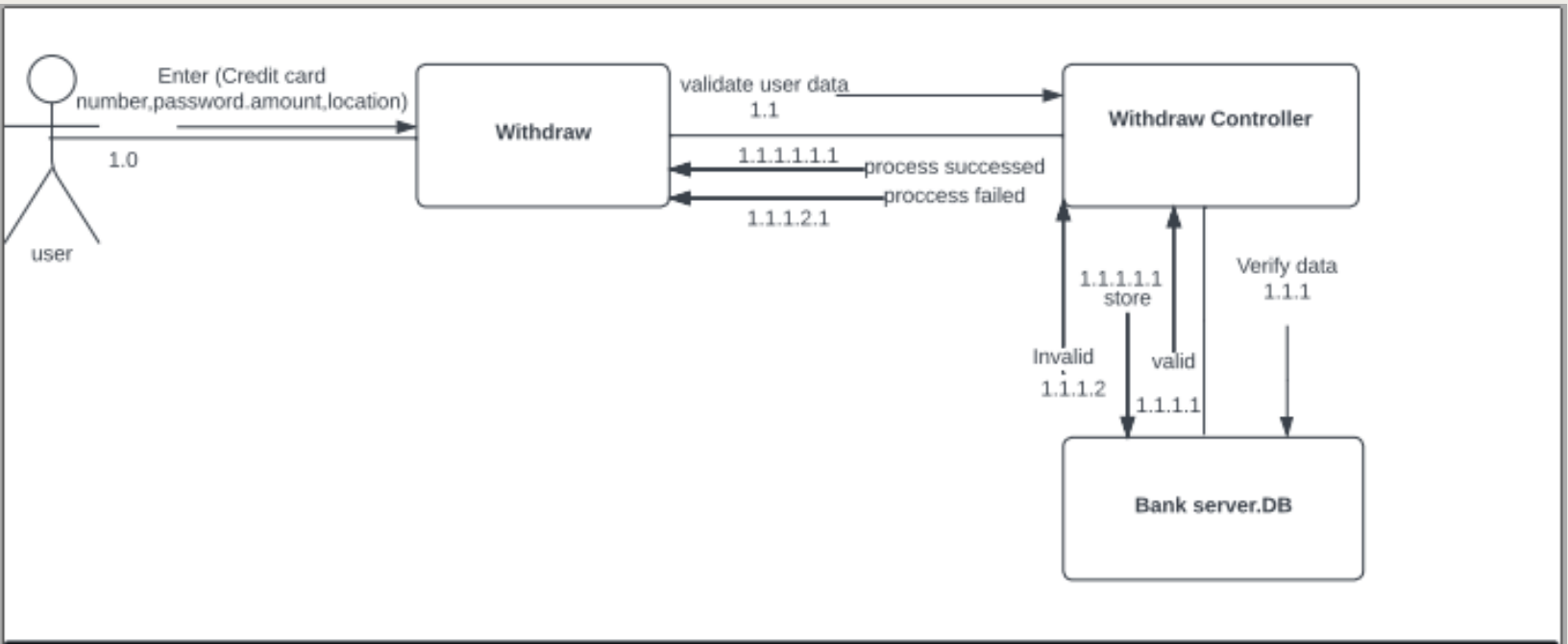

user
Alternative
ithdraw and
deposit]
[Withdraw]
System
Bank Server
click to withdraw money
enter (number of credit card and password)
Alternative
[if safe]
CD number,password,location and amount
check FRAUD
OK
return ok
[Fraud]
somthong wrong
logout
[Deposit]
click to deposit
enter (number of credit card and password)
Alternative
[if safe]
CD number,password,location and amount
check FRAUD
OK
return ok
[Fraud]
somthong wrong
logout

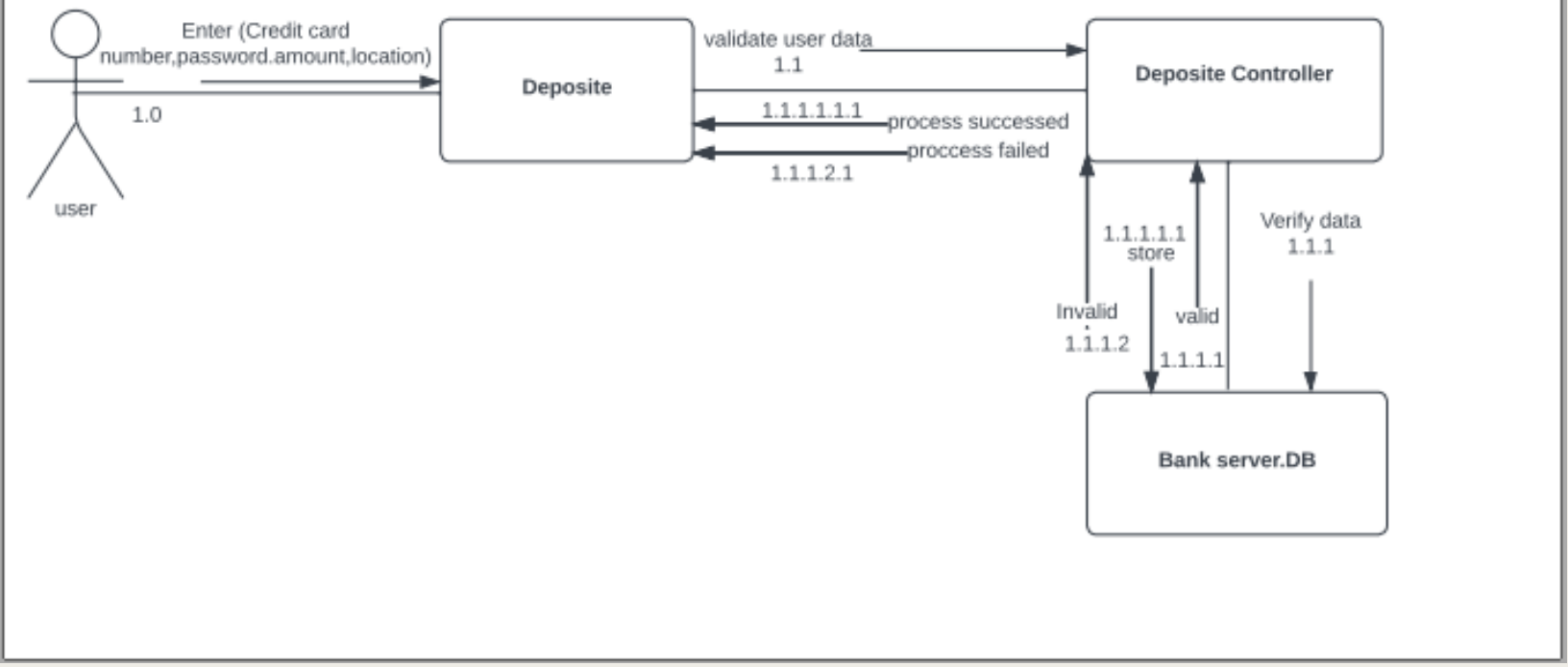

user
[update account]
System
Database
click to update account
Answer Security questions
answers
save
First name,last name ,username,password
e-mail,cd number,and cd password
Answers
save
click operation hisstory
date of operation , operation type,cd number,location , amount
click for help
operations
selected operation
info for previous operation
click for logout
the acces to operation are now disable

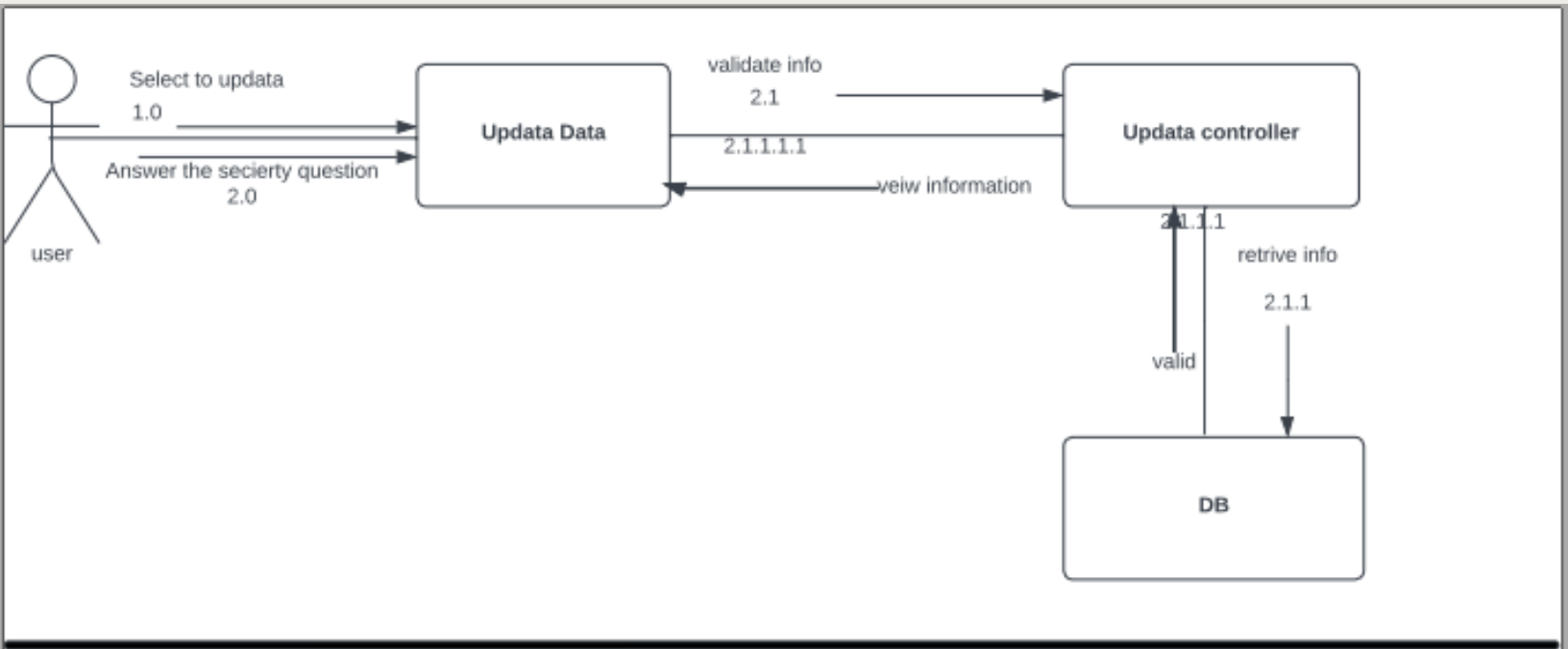
COLLABORATION DIAGRAM

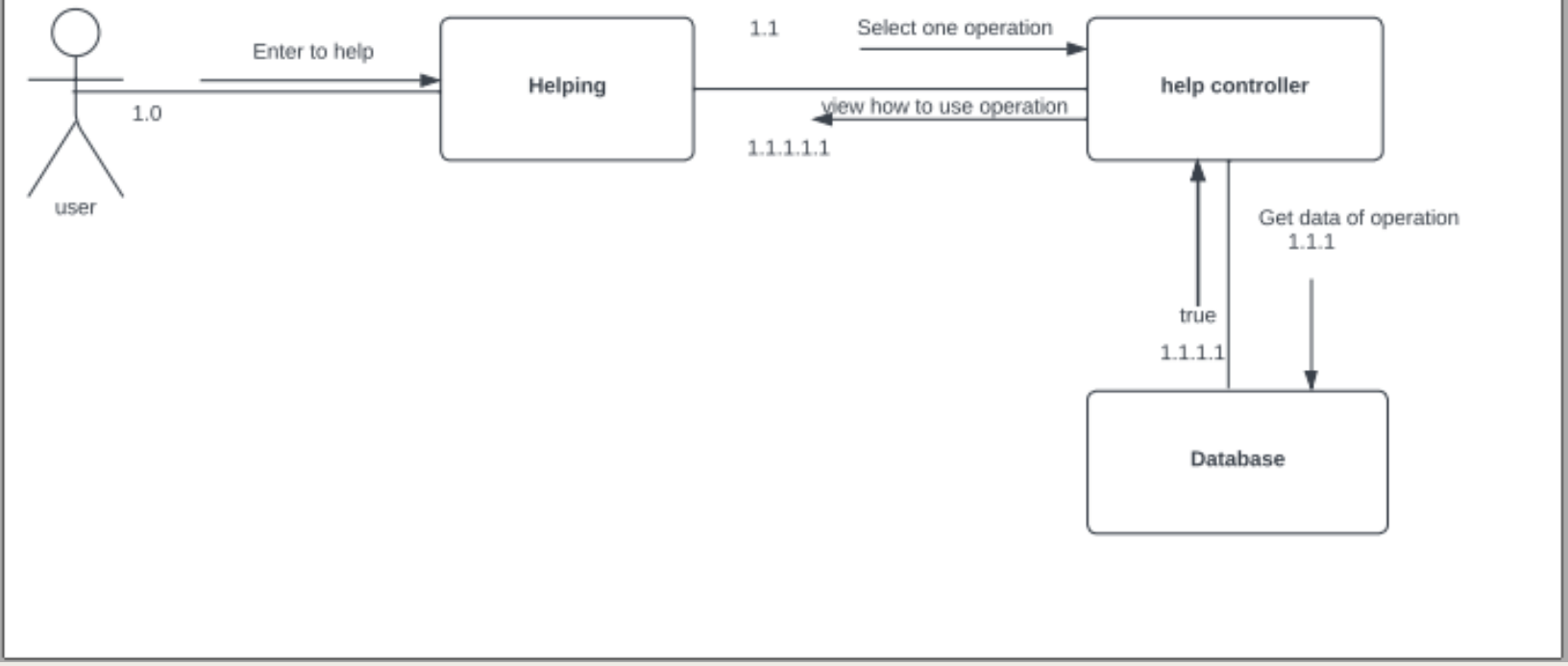

Select to Create account
1.0
Enter (firstname,credit card number,E-mail,.....)
2.0
user
AdminLonginUi
validate admin data
2.1
2.1.1.1..1.1
create succes
2.1.1.2.1
create failure
AdminLonginController
Store Data
2.1.1.1.1
2.1.1.1
verify admin data(not taken)
2.1.1
2.1.1.2
invalid
valid
AdminLonginDB

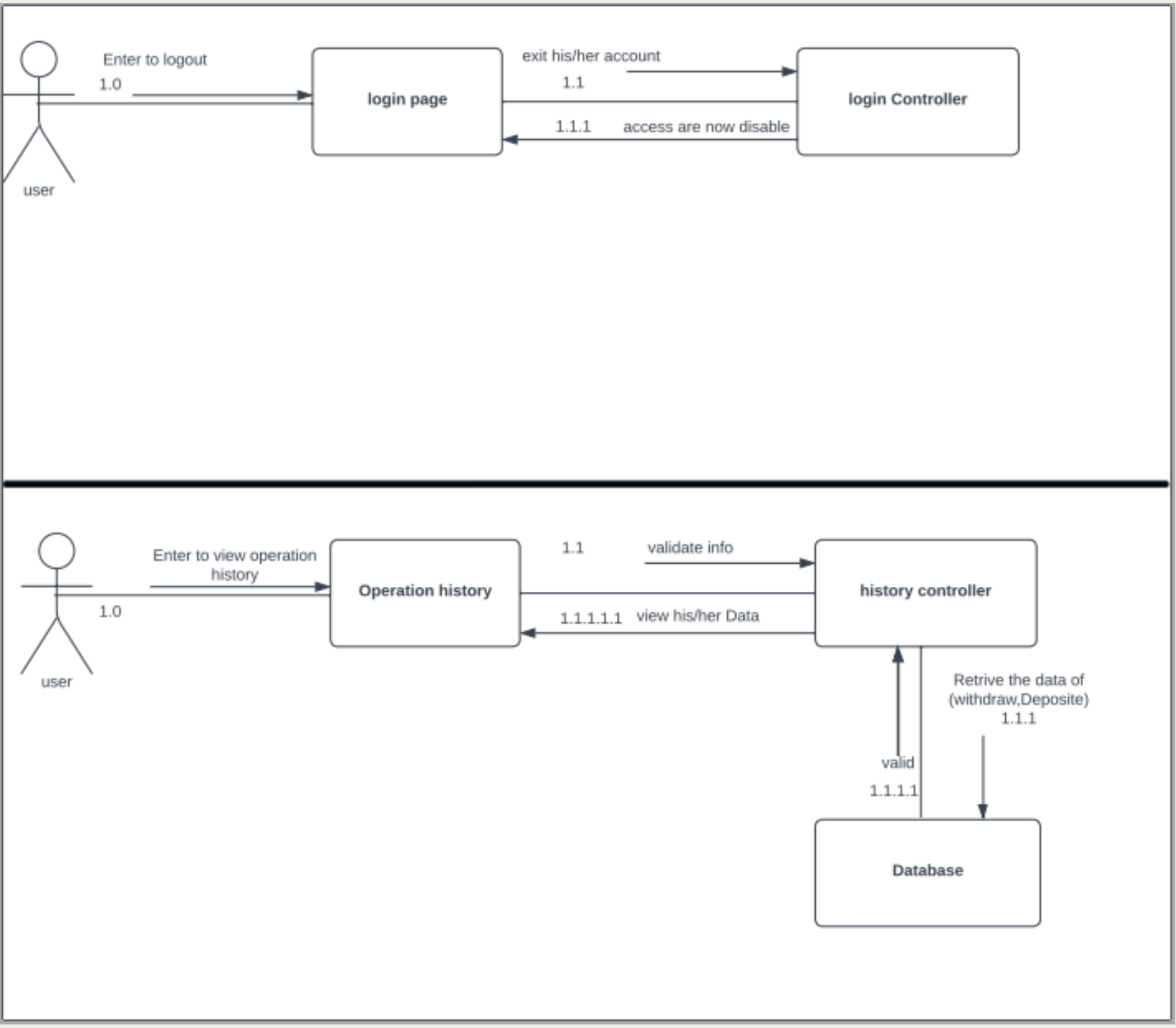

Enter (username,password)
1.0
user
AdminLonginUi
validate admin data
2.1
1.1.1.1.1.1
login successed
1.1.1.2.1
login failure
AdminLonginController
Store Data
1.1.1.1.1
1.1.1.1
verify admin data
1.1.1
1.1.1.2
invalid
valid
AdminLonginDB


Enter (Credit card
number,password.amount,location)
1.0
user
Withdraw
validate user data
1.1
Withdraw Controller
1.1.1.1.1.1
process successed
proccess failed
1.1.1.2.1
1.1.1.1.1
store
Verify data
1.1.1
Invalid
1.1.1.2
valid
1.1.1.1
Bank server.DB


Enter (Credit card
number,password.amount,location)
1.0
user
Deposite
validate user data
1.1
Deposite Controller
1.1.1.1.1.1
process successed
proccess failed
1.1.1.2.1
1.1.1.1.1
store
Verify data
1.1.1
Invalid
1.1.1.2
valid
1.1.1.1
Bank server.DB


Select to updata
1.0
Answer the secierty question
2.0
user
Updata Data
validate info
2.1
2.1.1.1.1
veiw information
Updata controller
2.1.1.1
retrive info
2.1.1
valid
DB


Enter to help
1.0
user
Helping
1.1
Select one operation
view how to use operation
1.1.1.1.1
help controller
Get data of operation
1.1.1
true
1.1.1.1
Database


Enter to logout
1.0
user
login page
exit his/her account
1.1
1.1.1
access are now disable
login Controller
Enter to view operation history
1.0
user
Operation history
1.1
validate info
1.1.1.1.1
view his/her Data
history controller
Retrive the data of (withdraw,Deposite)
1.1.1
valid
1.1.1.1
Database

DESIGN PATTERNS

1st Design Pattern

Name: **CREATIONAL** : (IMMUTABLE PATTERN)

Context: An immutable object is an object that has a state that never changes after creation.

Problem: How do you create a class whose instances are immutable?

Forces: There must be no loopholes that would allow 'illegal' modification of an immutable object.

Solution:

- Ensure that the constructor of the immutable class is the only place where the values of instance variables are set or modified.
- Instance methods which access properties must not change instance variables.

2nd Design Pattern

Name: **STRUCTURAL** : (DELEGATION PATTERN)

Context: You are designing a method in a class. You realize that another class has a method which provides the required service.
Inheritance is not appropriate (e.g. because the is-a rule does not apply).

Problem: How can you most effectively make use of a method that already exists in the
other class?

Forces: You want to minimize development cost by reusing methods.

Solution:

The delegating method in the delegator class calls a method in the delegate class to perform the required task. An association must exist between the delegator and delegate classes.

3rd Design Pattern

Name: **BEHAVIOURAL** (Observer (Publish-Subscribe) Pattern)

Context: When partitioning a system into individual classes you want the coupling between then to be loose so you have the flexibility to vary them independently.

Problem: A mechanism is needed to ensure that when the state of an object changes related objects are updated to keep them in step.

Forces: The different parts of a system have to kept in step with one another without being too tightly coupled.

Solution:

One object has the role of the subject/publisher and one or more other objects the role of observers/subscribers. The observers register themselves with the subject, & if the state of the subject changes the observers are notified & can the update themselves.

The are two variants:

the Push Model where the subject send the observers detailed information about the change that has occurred, and ..

- the Pull Model where the subject simple notifies the observers that there have been changes, and it's the responsibility of the observers to find out the details they need to update themselves.