System Analysis & Design CA1 Main Assignment
Student: Adam Sever 2018494

An accurate problem definition and description and an accurate list of system requirements:

accurate problem definition:

Ultra Vision is currently using a paper-based system which is slow and inefficient. This system will also hinder them when they choose to expand to another or multiple locations. UltraVision rents titles that they have in-house, so titles must be organized per store however the customer membership cards should be able to work across any store.

list of system requirements :

When will also need a billing system to charge customers automatically on purchases or late fees.

We must ensure the system is flexible and maintainable using SOLID principles in a way that will allow us to create:

1. more ways for a customer to interact with our system in the future (website, POS).
2. To easily Create Access plan Types
3. To create a transactional interaction with the system in a way that allows for scalability.

The system will need to create a GUI that will register input through a USB for a barcode scanner that can be interpreted as keyboard input.

Using this information the system can successfully determine the customer and allow him/her to carry out a rent.

The presenter must be threaded.

The presenter must be able to handle request from a socket(POS) or a HTTPS request(Website) however API calls will be encoded in JSON.

Security:
The membership card must have a photographic id printed on it as the billing system is automatic.

All communication through socket should be encrypted with RSA. HTTPS already handles this.

Physical System

Digital Tracking can be achieved with RFID strips on the products which will contain their corresponding ID in the database with the same strip for membership card but with a photo ID printed on them for security.

The Accu-rent company responsible for the returns will only have to give the System the ID of the product.

An existing card billing System will have to be established and implemented.

UML Modelling of the Ultra-Vision system using UML diagrams

This system uses MVP System Design

View

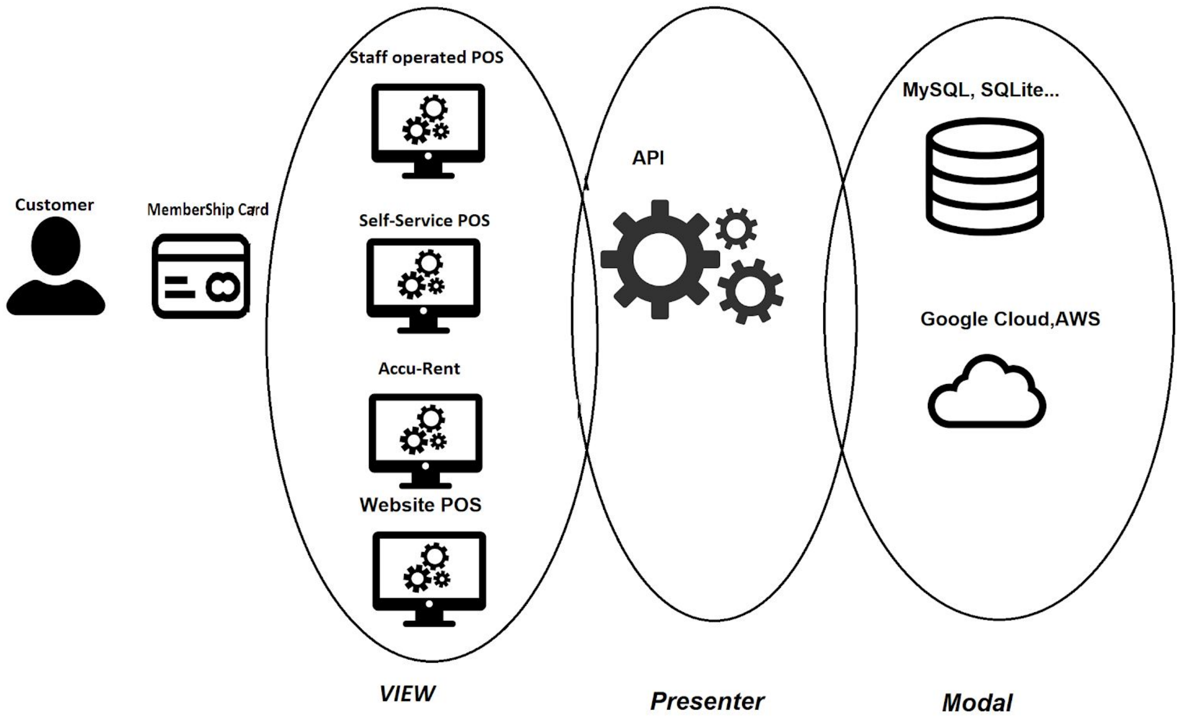
The purpose of the view is the display a graphic Interface to the Customer shop clerk assisting the Customer or a website for the customer and to allow them to access the rest of the system through a friendly and controlled manner.

Presenter

The purpose of the Presenter is to be the logic behind the system, ensure security and processing the requests sent by the view.

Modal

The modal is where the raw data is stored, security measures and data integrity measures will also be implemented here. This is a Relational Database.



What to achieve

Ensuring that the system is scalable, interchangeable and also Secure.

1.Scalable

The system allows from many and different point of sales systems to work side by side due to threading this can be greatly and smoothly scaled upon with the use of more in-house servers or through cloud computing.

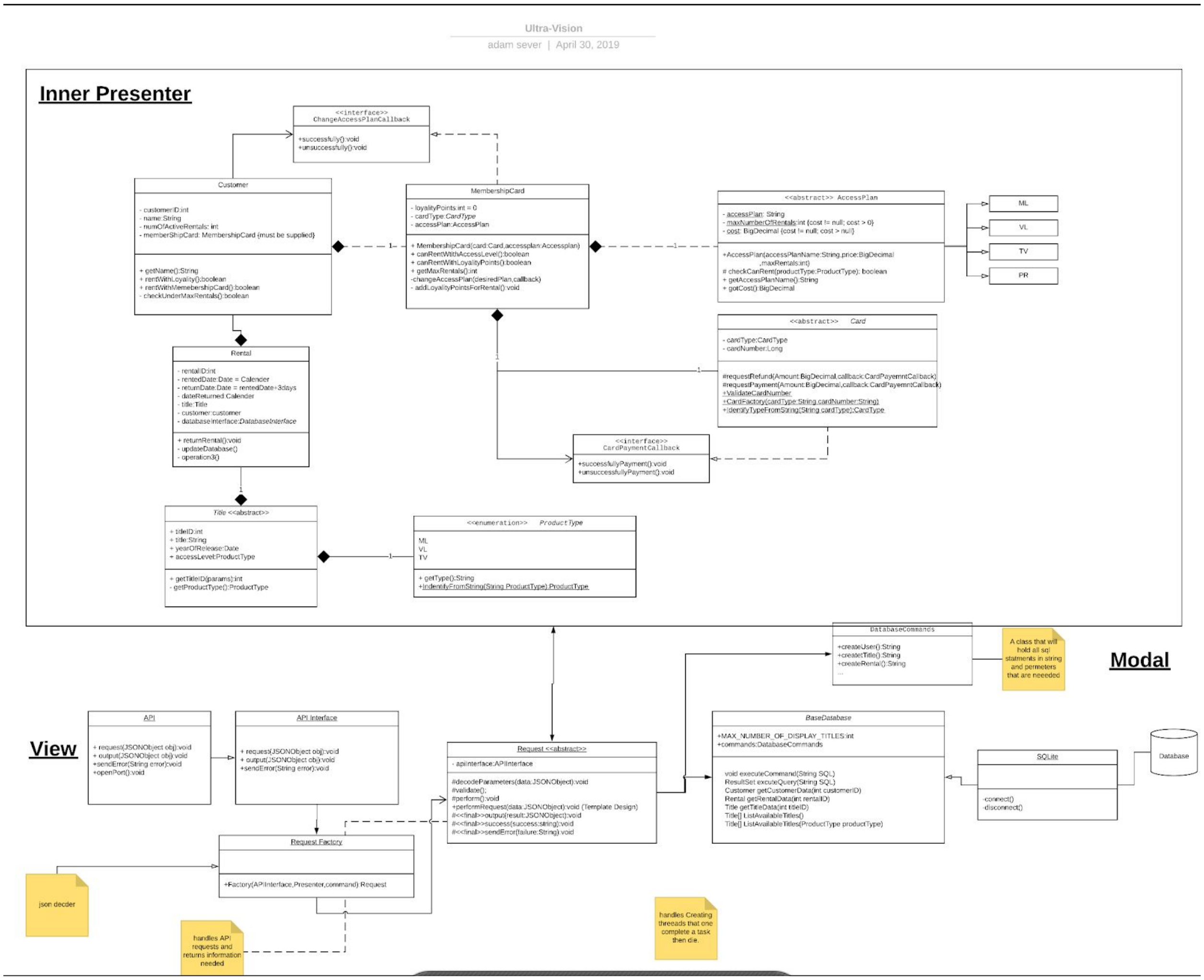
2.Interchangeable

The systems subsections View, Presenter, and Modal are completely independent of one another making them fault tolerant and also seamless updating.

3.Secure

Data integrity would have been a concern with paper trail paper due to physical damage however the Database can be continuously back up with the help of Google cloud.

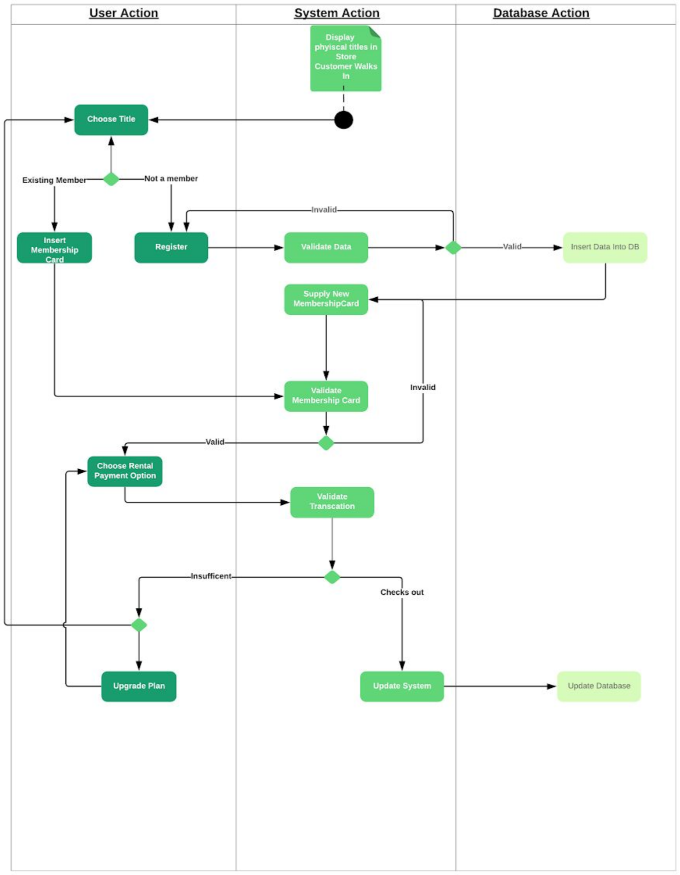
UML Class


Ultra-Vision
adam sever | April 30, 2019
Inner Presenter
<<interface>> ChangeAccessPlanCallback
+successfully():void
+unsuccessfully():void
Customer
- customerID:int
- name:String
- numOfActiveRentals: int
- memberShipCard: MembershipCard {must be supplied}
+ getName():String
+ rentWithLoyality():boolean
+ rentWithMemebershipCard():boolean
- checkUnderMaxRentals():boolean
MembershipCard
- loyalityPoints:int = 0
- cardType:CardType
- accessPlan:AccessPlan
+ MembershipCard(card:Card,accessplan:Accessplan)
+ canRentWtithAccessLevel():boolean
+ canRentWithLoyalityPoints():boolean
+ getMaxRentals():int
-changeAccessPlan(desiredPlan,callback)
- addLoyalityPointsForRental():void
1
<<abstract>> AccessPlan
- accessPlan: String
- maxNumberOfRentals:int {cost != null; cost > 0}
- cost: BigDecimal {cost != null; cost > null}
+AccessPlan(accessPlanName:String,price:BigDecimal ,maxRentals:int)
# checkCanRent(productType:ProductType): boolean
+ getAccessPlanName():String
+ gotCost():BigDecimal
ML
VL
TV
PR
<<abstract>> Card
- cardType:CardType
- cardNumber:Long
#requestRefund(Amount:BigDecimal,callback:CardPayemntCallback)
#requestPayment(Amount:BigDecimal,callback:CardPayemntCallback)
+ValidateCardNumber
+CardFactory(cardType:String,cardNumber:String)
+IdentifyTypeFromString(String cardType):CardType
<<interface>> CardPaymentCallback
+successfullyPayment():void
+unsuccessfullyPayment():void
Rental
- rentalID:int
- rentedDate:Date = Calender
- returnDate:Date = rentedDate+3days
- dateReturned:Calender
- title:Title
- customer:customer
- databaseInterface:DatabaseInterface
+ returnRental():void
- updateDatabase()
- operation3()
Title <<abstract>>
+ titleID:int
+ title:String
+ yearOfRelease:Date
+ accessLevel:ProductType
+ getTitleID(params):int
- getProductType():ProductType
<<enumeration>> ProductType
ML
VL
TV
+ getType():String
+IndentifyFromString(String ProductType):ProductType
DatabaseCommands
+createUser():String
+createtTitle():String
+createRental():String
...
A class that will hold all sql statments in string and permeters that are neeeded
Modal
View
API
+ request(JSONObject obj):void
+ output(JSONObject obj):void
+sendError(String error):void
+openPort():void
API Interface
+ request(JSONObject obj):void
+ output(JSONObject obj):void
+sendError(String error):void
Request <<abstract>>
- apiInterface:APIInterface
#decodeParameters(data:JSONObject):void
#validate();
#perform():void
+performRequest(data:JSONObject):void (Template Design)
#<<final>>output(result:JSONObject):void
#<<final>>success(success:string):void
#<<final>>sendError(failure:String):void
BaseDatabase
+MAX_NUMBER_OF_DISPLAY_TITLES:int
+commands:DatabaseCommands
void executeCommand(String SQL)
ResultSet excuteQuery(String SQL)
Customer getCustomerData(int customerID)
Rental getRentalData(int rentalID)
Title getTitleData(int titleID)
Title[] ListAvailableTitles()
Title[] ListAvailableTitles(ProductType productType)
SQLite
-connect()
-disconnect()
Database
Request Factory
+Factory(APIInterface,Presenter,command):Request
json decder
handles API requests and returns information needed
handles Creating threeads that one complete a task then die.

Link to High-Res

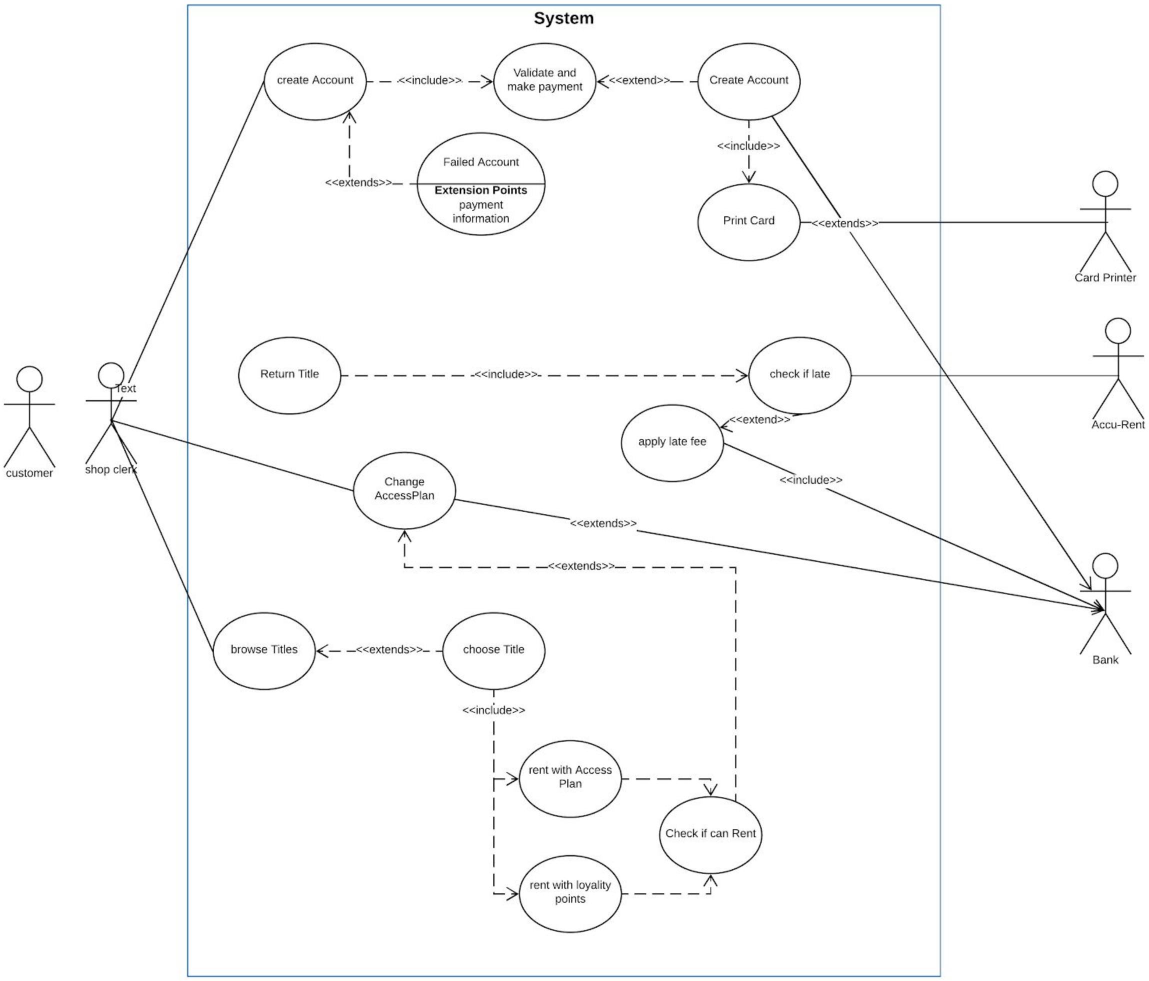
https://github.com/avacadoadam/UltraVisionAssigment/blob/master/Plan/Ultra%20Vision%20Object%20Diagram.png

UML Activity Diagram

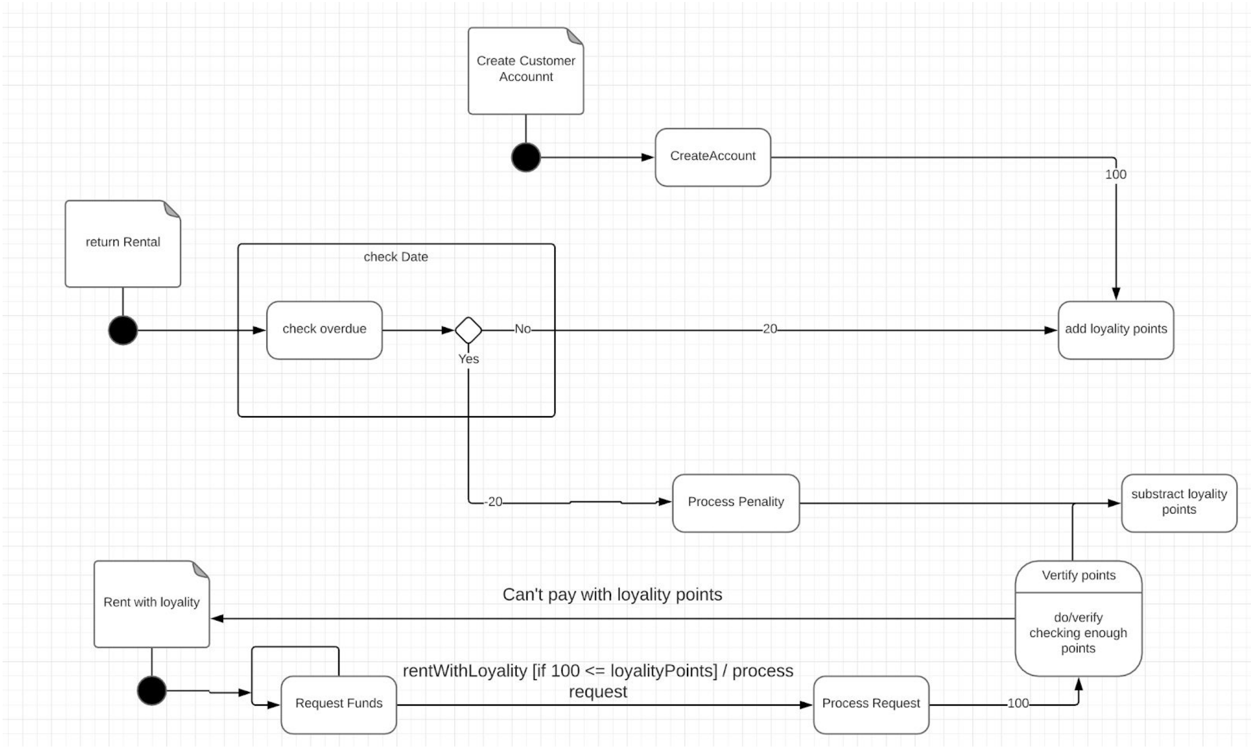


High Res = https://raw.githubusercontent.com/avacadoadam/UltraVisionAssigment/master/Plan/Activity%20Diagram%20for%20Ultra%20Vision.png

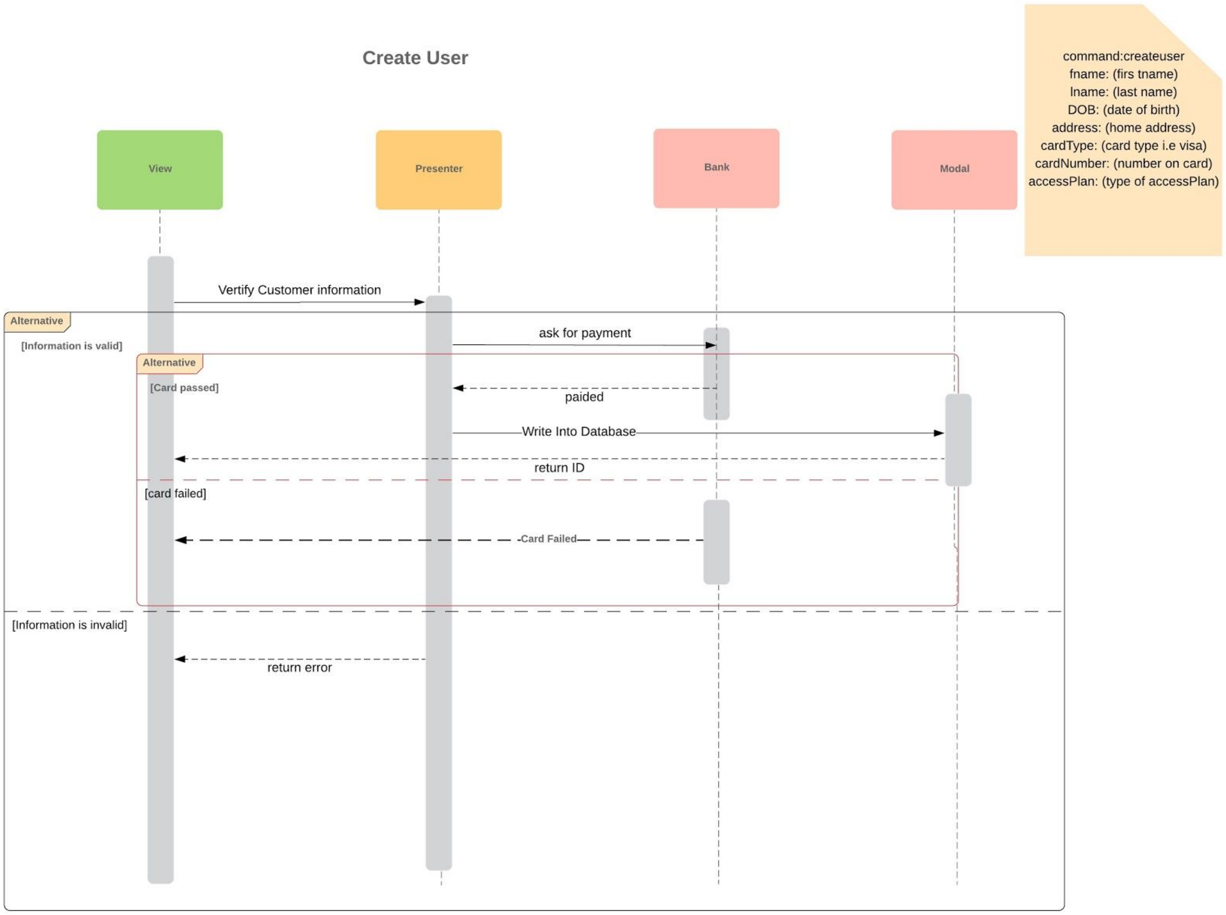
UML User Case



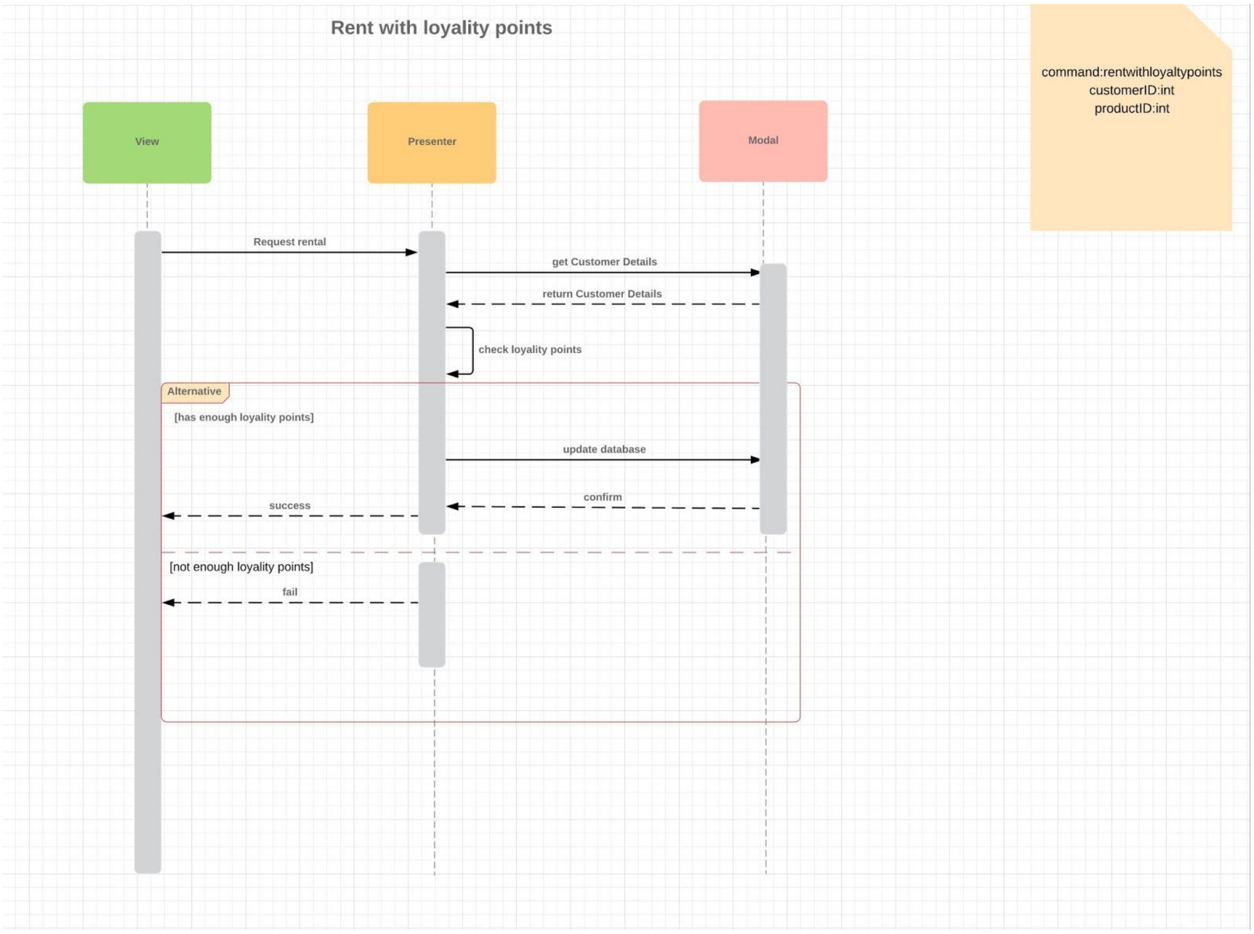
UML State Diagram Loyalty Points



UML Sequence for creating Customer API



UML Sequence for rent with loyalty points



Analysis of the section of Java code provided

1: In function addPoints(points:int) the function setRentAllowed is being called every time we add points which is unnecessary, sentRentAllowed should be refactored to canRent():boolean.

2: In function, availFreeRent has an unnecessary else block simply return false at the end to be in-line with Lint.

3: Again availFreeRent is calling setRentAllowed() which is simply checking if there is 100 points It would be more efficient to check when someone is trying to rent with loyalty points rather then every time the class functions are called.

One User Story regarding an additional desired software feature
User stories to be added

1: As a customer, I want to be able to check the rental policy before renting to ensure I am fully aware of my obligations.

2: As a system admin I want to be able to check overdue rentals so that I can follow up and take appropriate action

Research regarding software testing for this type of software project Research regarding dealing with complexity

To become agile we must integrate unit testing for the second principle in the agile manifesto which is "*Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.*"

"*Unit testing facilitates changes and simplifies integration*"
If we are expected to make changes late we must have an automated testing system can test the entire system fast to ensure our new update did not have an unexcepted result on our entire system.

We must also take into account principle 9
Continuous attention to technical excellence and good design enhances agility.

Unit testing is good design it improves code quality in that it finds every defect also writing tests before actual coding makes you think harder about the problem similar to good planning.

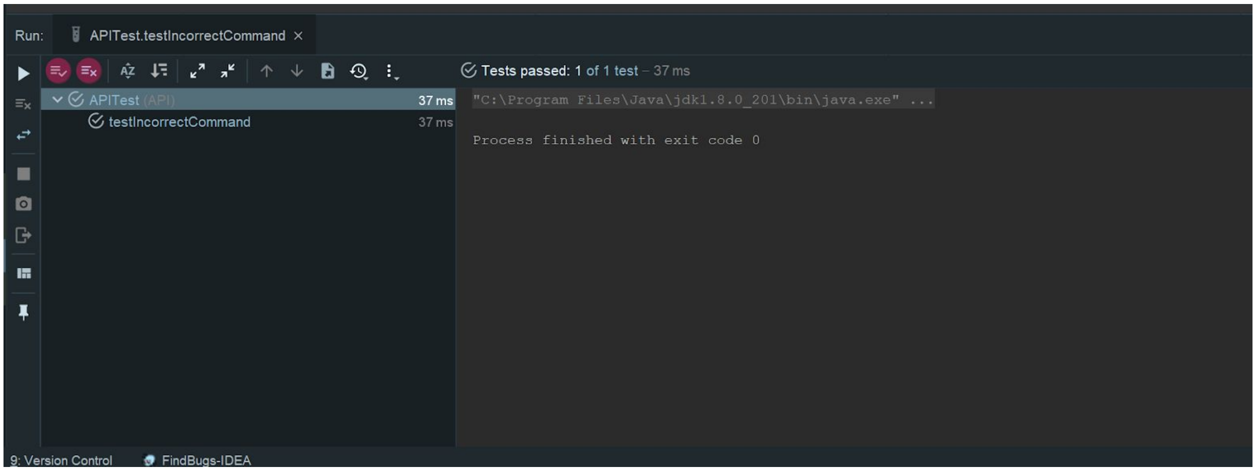
Provides Documentation
Just like a plan, it helps us in the further or new developers pick up where we left and follow our intended path and keep it bug-free.

Reduce Development Time
It can be argued that writing test is very time consuming however just like planning it will save the developer team time the road since bugs are found earlier and also found every update.

To test the Presenter mockito must be used to mock both the request for the View and response for the Modal. Using mockito to create a test for each API call. Then A larger stress test can be added to test how the system can deal under loads.

For the inner workings of the presenter, junit should be used to unit test the different class and then to can be combined with mockito to create larger more system-wide tests.

Example of automated testing:



Tests passed: 1 of 1 test – 37 ms

APITest (API) 37 ms

testIncorrectCommand 37 ms

```
"C:\Program Files\Java\jdk1.8.0_201\bin\java.exe" ...

Process finished with exit code 0
```

Report content and presentation:

One feature which has helped me tremendously was the use got github and version control I overlooked any aspects before such as branching merging and Intellji integration with it.

To allow me to consult the old code that I may have refactored. It also gives me great coordination with my Gantt chart is it shows me the date of commits (submission).

I have used automated testing before with android however now understand why planning is so important it allowed me to take a look at my system at an even higher and visual level that can help me determine time, complexity and issues that can arise.

It also helped me greatly as it made it clear what classes and parts of the system would be the best implement with a design pattern which without planning you would seem to have to write then refactor which is inefficient.

To view Code

https://github.com/avacadoadam/UltraVisionAssigment

References :

https://www.agilealliance.org/agile101/12-principles-behind-the-agile-manifesto/
https://www.lucidchart.com