**REPORT ASSIGNMENT 2 –GROUP 2**

**Member in group :**

**Võ Hữu Lộc**

**Lê Hoài Phúc**

**Phạm Lê Tấn Thịnh**

**Nguyễn Thành Tâm**

**Hồ Đình Trọng**

1. **use case specification**

UC1

Name: Save item for purchase.

Description:

+ Customers will enter the website.

+ Customers will choose the car they like to suit their event

+ While choosing a car, the customer may find a car that they like but they are not ready to rent because they want to see other cars, so they want to save that car in a list so they can later find the item that they were previously interested in.

UC2

Name: Payment Service

Description:

+ When customers have finished choosing the car they need, they will go through the step of paying the cost to rent that car

+ Here we create payments in cash, momo, vnpay, .. so that customers can choose the payment that suits them. And make sure all transactions will be done securely

UC3

Name: Review history

Description:

+ When the transaction is completed, customers can review their car rental history

+ Make sure that all car rental information, date and time, are confirmed on the history.

+ The car owner can review the rental history of each car, can make a list of information related to the rented car.

UC4

Name: Suggest car to rent

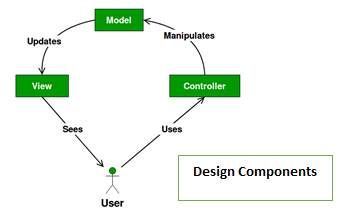
Description:

+ In the website we create a page to support customers when they have questions

+ Any questions we have about choosing or renting a car, if customers have questions, we will be supported by the staff

**II.Evaluation architecture style**

**1.MVC**



- Advantages:

* Development of the application becomes fast:

+ Using MVC can help simplify the architecture of the system so that developers can easily develop Model, View and Controller as the same time.

* Easy for multiple developers to collaborate and work together.

+ As mention above, developers can divide their task appropriately because Model, View, Controller can be developed separately.

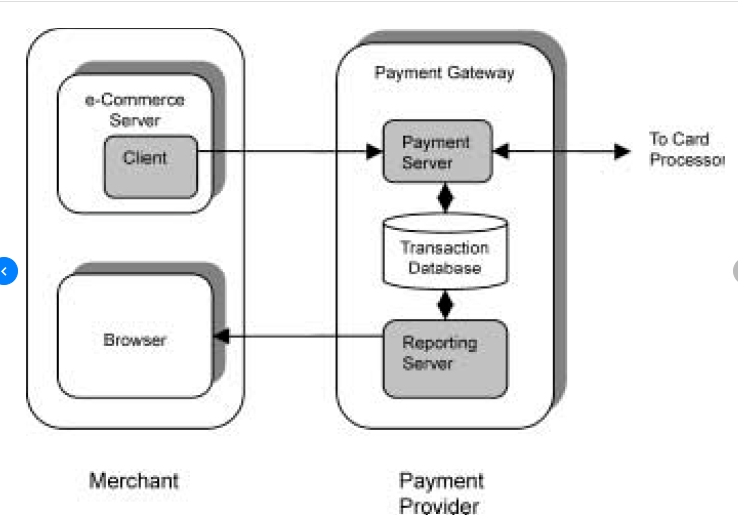
* Easier to Update the application.

+ When we want to update a View, we can easily to update UI without worry about change in the Model.

* Easier to Debug as we have multiple levels properly written in the application.

+ Debuging in MVC is easy when we can easily identify root cause of the exception.

**2.Client Server**



+ The payment gateway typically contains a payment server, a transaction database, and a reporting server. The payment server accepts encrypted transaction information from multiple clients as well as routes necessary information to the appropriate processing network depend- ing on the credit card. It also performs settlement of transactions. The transaction database contains all transaction-related information for initial orders, payment authorizations, and settlements. The reporting server is used to provide transaction reports such as settled transactions, unsettled transactions, and invalid transactions to the merchant. Thus, in this approach, all order details and some payment information reside in the merchant database, while the payment details reside in the gateway database

**3.Message Broker**:

**+ Provided communication between services that may not be running at the same time**.  The producer can send messages regardless of whether the consumer is active or not. All it needs is a running message broker. The same applies to the consumer.

**+ Improved system performance by introducing asynchronous processing.**High-consuming tasks can be distributed to separate processes. That will fasten up your application and increase user experience.

**+ Increased reliability by guaranteeing the transmission of messages.**Message brokers offer a redelivering mechanism. In case of consumer failure, it can redeliver the message immediately or after some specified time. It also supports routing of not delivered messages – it’s called a dead-letter mechanism

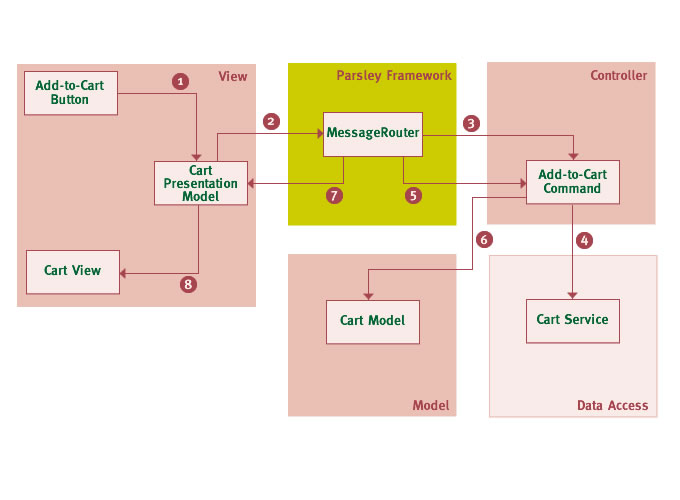
**III. UML design:**

* **UC1**
* Name: Save item for purchase.
* Description:

+ Customers will enter the website.

+ Customers will choose the car they like to suit their event

+ While choosing a car, the customer may find a car that they like but they are not ready to rent because they want to see other cars, so they want to save that car in a list so they can later find the item that they were previously interested in.



1: The user clicks the "Add to cart" button. With Flex for example this will generate a low level UI event. The event handler in the component invokes a method on the presentation model instance (usually injected into the view) which handles communication between View and Controller.

2: The model creates an application message possibly gathering any further information from the view and encapsulating this information in the message. This message will then be dispatched through Parsleys MessageRouter

3: The MessageRouter will then process any MessageHandlers or Commands which were registered for this message type (usually with Metadata tags on methods or properties, alternatively in MXML or XML). In this case the AddToCartCommand will be invoked.

4: The command will invoke a remote service method (the service usually being injected). The asynchronous operation will be managed by the framework which allows to avoid some of the usual plumbing code. Neither the command nor any other part of the application has to explicitly add a result handler to the AsyncToken returned by the remote call.

5: The framework receives the result. It is first returned to the command instance itself if it contains a result handler. But this is purely optional.

6: The result handler in the command might modify or cache the model before it gets passed to other result handlers.

7: Next the framework will invoke any other registered result handlers in the application. In this case the CartPM contains a result handler which will now be invoked causing the model instance to update its internal state.

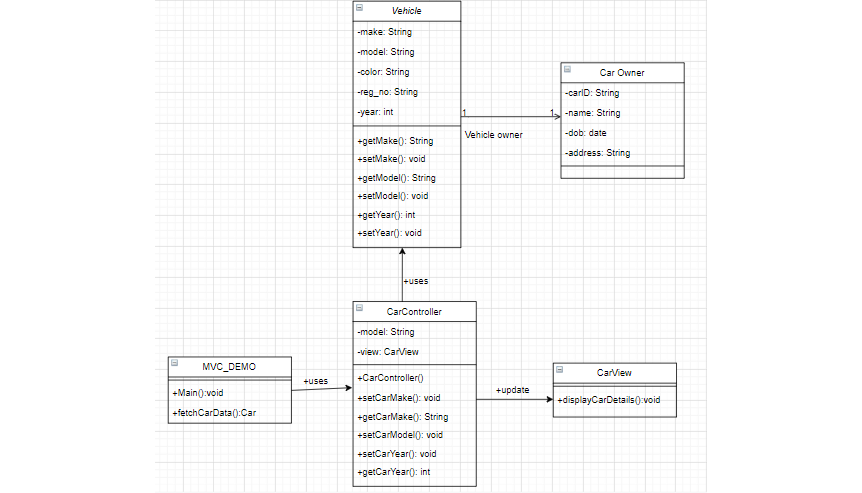
8: The view is refreshed, usually through binding to properties of the presentation model.

From the example above many core elements will have been setup and wired in a Parsley Context including the view as described in 9 Dynamic View Wiring. This usually includes all controller actions, their dependencies like remote services, and the registration for particular messages in the MessageRouter.

* **UC2**
* Name: Review history
* Description:

+ When customers have finished choosing the car they need, they will go through the step of paying the cost to rent that car

+ Here we create payments in cash, momo, vnpay, .. so that customers can choose the payment that suits them. And make sure all transactions will be done securely



**UC3**

Name: Payment serviceDescription:

+ When the transaction is completed, customers can review their car rental history

+ Make sure that all car rental information, date and time, are confirmed on the history.

+ The car owner can review the rental history of each car, can make a list of information related to the rented car.

Payment service: have 2 mode payment

+ Payment by card :

-Success

Client request to Mode payment : Card

Mode insert card to API bank

Api check system - > Success

Forward money to Bank

Return Success to Client

-Fail

Client request to Mode payment : Card

Mode insert card to API bank

Api check system - > Fail

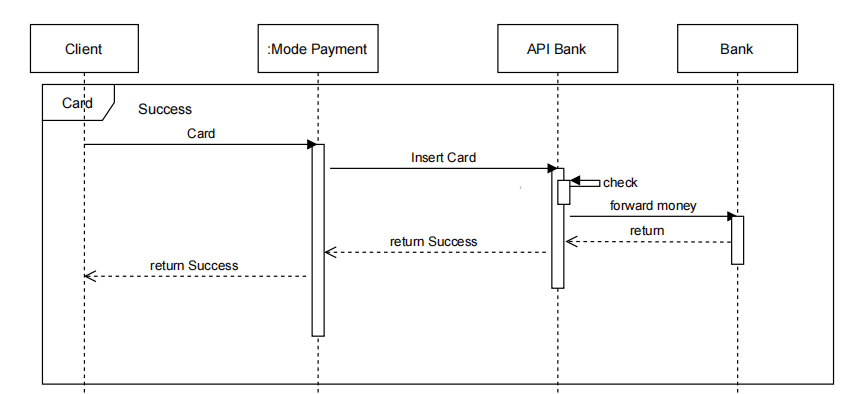
Return Fail to Client and return money to client

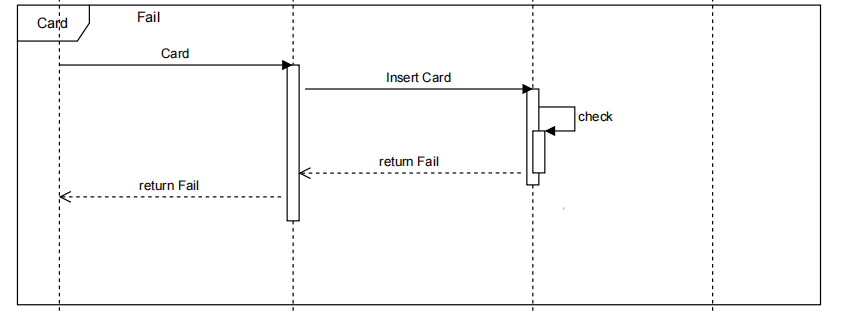
+ Payment by Cash:

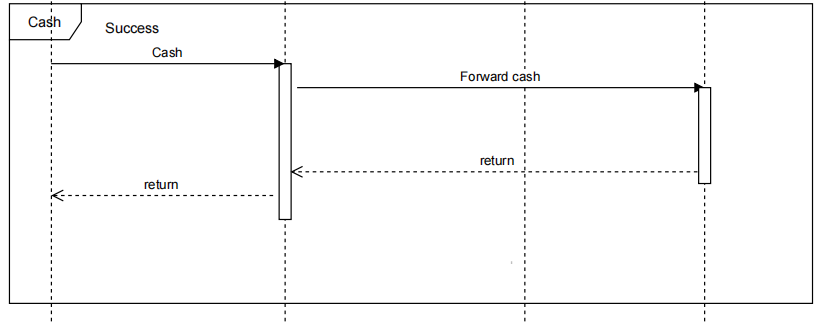
Client request to Mode payment : Cash

Mode payment request to Bank

Return Success to Client







**UC4**

Name: Support chatDescription:

+ In the website we create a page to support customers when they have questions

+ Any questions we have about choosing or renting a car, if customers have questions, we will be supported by the staff

(Sequence Diagram)

Message have 2 type :

SUCCESS

Staff send message to proxy1 : Sending message

Proxy1 forward message to message broker : sent message

to message broker message to Customer : Received message

return : return information for staff know customer received message

FAIL

Staff send message to proxy1 : Sending message

Proxy1 forward message to message broker : sent message

Message broker fail server

Return : return information cant sent message to customer

