SOEN 6461 (Software Design Methodologies)

Concordia University

Department of Computer Science and Software Engineering

Deliverable 2

Supervisor:
Prof. PANKAJ KAMTHAN

Students:

Team A
Varun Aggarwal (40225335)
Ujjawal Aggarwal (40183962)
Arjun Anghan (40193262)
Shivangi Arul (40217139)
Reva Balasundaram (40227644)
Imad Altaf (40170775)

April 10, 2023

Contents

PROBLEM 6

1.1 High-level solution domain model

A high-level solution domain model is a representation of the key classes and their responsibilities in a software system. It provides an overview of the system architecture and helps stakeholders to understand the functionality of the system at a high level.
A CRC card model is a specific type of solution domain model that uses index cards to represent classes and their responsibilities. Each card contains the name of the class, its responsibilities, and the other classes it collaborates with. This technique is often used in the early stages of software development to facilitate brainstorming and collaboration between team members. The purpose of a high-level solution domain model (including CRC card models) is to provide a shared understanding of the system architecture and ensure that all team members have a common vocabulary when discussing the system. This can help to identify potential issues and ensure that the system is well-designed and maintainable over time. Additionally, it can serve as a starting point for more detailed design and implementation work.
Here is the CRC card model for the igo TVM domain:

igo_TVM	
• Support both French and English languages. • Provide a way for travellers to recharge metro cards. • Provide a way for travellers to purchase tickets. • Provides two payment methods. • Generates the payment receipt. • Location of the TVM.	• PurchaseTicket • PaymentGateway • MetroCardRecharge • ReceiptPreference

Figure 1.1: CRC card for igo_TVM

MetroCardRecharge	igo_TVM
• Be used for transportation. • Have a magnetic chip that can be recharged at TVMs. • Provides various recharge type options.	• igo_TVM • PaymentGateway

Figure 1.2: CRC card for MetroCardRecharge

PurchaseTicket	igo_TVM
• Be used for transportation. • Be purchased from any TVM • Have limited usage based on selected options	• igo_TVM • PaymentGateway

Figure 1.3: CRC card for PurchaseTicket

PaymentGateway	igo_TVM Bank
• Handle daily payments made by travellers • Accept cash or debit/credit cards for transactions	• igo_TVM • Payment

Figure 1.4: CRC card for PaymentGateway

Bank	PaymentGateway
• Authenticate and verify payments made through credit/debit cards	• PaymentGateway • Payment

Figure 1.5: CRC card for Bank

Payment	igo_TVM PaymentGateway
• Contain details about a chosen transaction. • Displays the balance amount to dispense.	• PaymentGateway • Bank • ReceiptPreference

Figure 1.6: CRC card for Payment

ReceiptPreference	igo_TVM
• Provide evidence of the transaction with date, time, and price • Can be printed or emailed	• igo_TVM • Bank • Payment

Figure 1.7: CRC card for ReceiptPreference

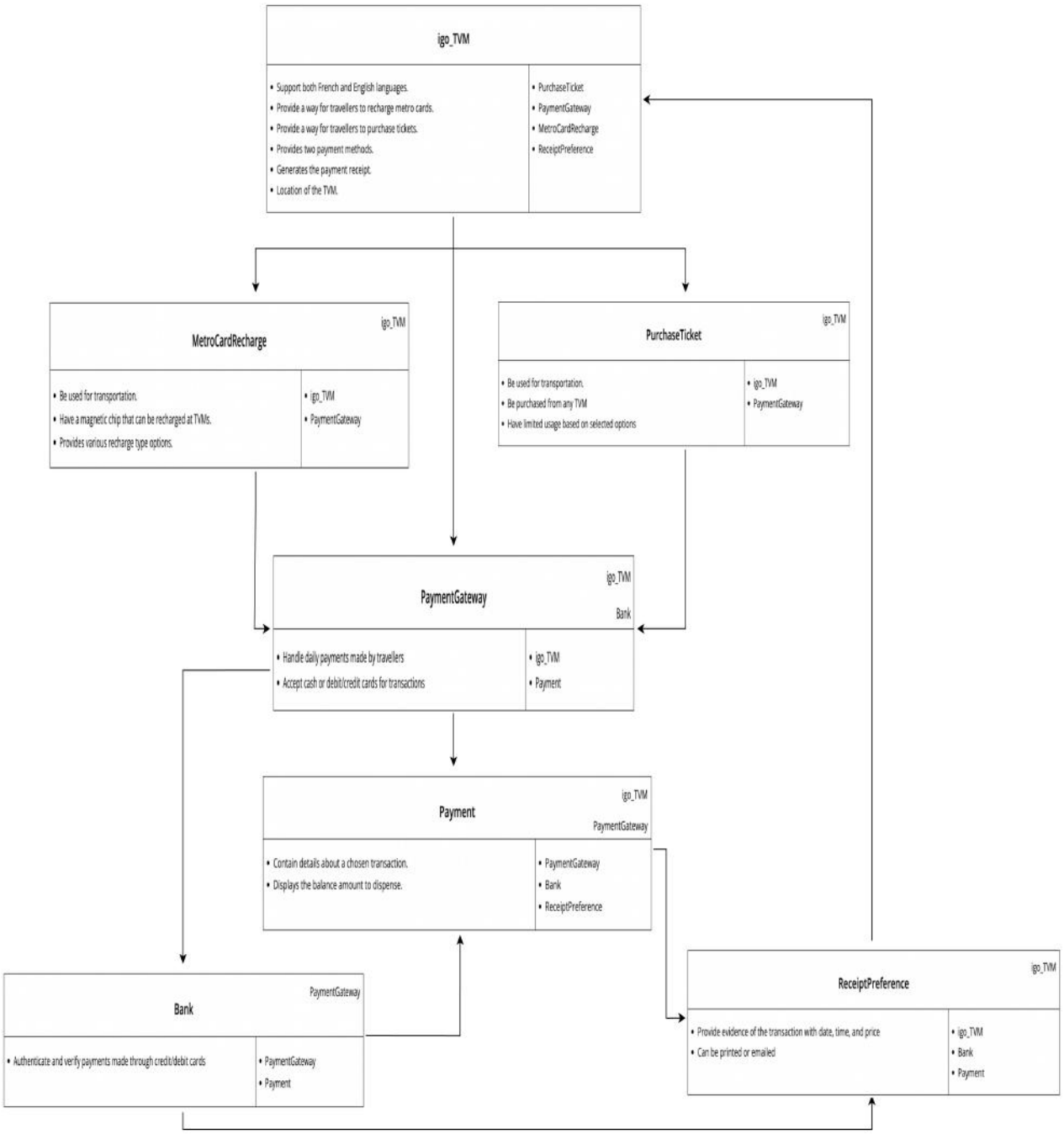


Figure 1.8: CRC Card Model

1.2 Brief description of igo TVM CRC model

- The igo_TVM class represents the ticket vending machine and has four collaborators, PurchaseTicket, MetroCardRecharge, PaymentGateway and ReceiptPreference. It serves as the primary element and allows travelers to select interaction language, purchase tickets and recharge metro cards. It has several relationships with other entities, such as PurchaseTicket, MetroCardRecharge, PaymentGateway and ReceiptPreference.
- The PurchaseTicket class represents paper-made tickets that can be purchased from any TVM. It has relationships with igo_TVM and PaymentGateway.
- The RechargeMetroCard class represents a smart card used for transportation and can be recharged at any TVM. It has relationships with igo_TVM and PaymentGateway.

- The PaymentGateway class represents the payment system used by the TVM that provides options to pay with cash or a card. It has a relationship with igo_TVM and Bank.
- The bank class represents the regulatory body that authenticates and verifies payments made by travelers. It has a relationship with PaymentGateway, Payment and ReceiptPreference.
- The Payment class represents details about a transaction. It has relationships with Banks and ReceiptPreference.
- The ReceiptPreference class represents the evidence of a transaction and includes the date, time, and price which can be email and paper-based. It has relationships with igo_TVM and Payment and Bank.

Overall, the CRC card model provides a clear and organized representation of the different classes, their responsibilities, collaborators and their relationships in the TVM domain, which can be helpful in understanding the system's functioning.

PROBLEM 7

2.1 Part-A

The Structural Design Model in UML is a static model that represents the structure of a system by defining the classes and their relationships. It provides a visual representation of the system's entities and how they are connected to each other.
The UML class diagram is a type of Structural Design Model that represents the classes, interfaces, and their relationships. It describes the static structure of a system by defining the attributes, methods, and associations between the classes. The class diagram is an essential tool for understanding the system's structure and its behavior. It helps to visualize the system's components, their relationships, and their properties. It also provides a foundation for generating the code of the system.
The UML class diagram for the Ticket Vending Machine (TVM) system shows the different classes and their relationships. The main class is igo_TVM, which represents the TVM itself and has four collaborators: PurchaseTicket, MetroCardRecharge, PaymentGateway, and ReceiptPreference. The PurchaseTicket class represents paper-made tickets that can be purchased from any TVM, while the RechargeMetroCard class represents a smart card used for transportation and can be recharged at any TVM.
Defining Class, attributes, properties and relationships of our TVM model as follows:

1. The igo_TVM class represents the ticket vending machine and allows travelers to select interaction language, purchase tickets, and recharge metro cards. It has attributes such as languageSelection and passSelection and functions such as setLanguage, selectedCard, and selectedTicket.

2. The RechargeMetroCard class represents a smart card used for transportation and can be recharged at any TVM. It has attributes such as cardNumber, rechargeAmount, and passType, and functions such as rechargeType, rechargeStatus, addBalance, and payment.

3. The PurchaseTicket class represents paper-made tickets that can be purchased from any TVM. It has attributes such as ticketNumber, ticketType, and ticketPrice, and functions such as getTicketPrice, getTicketNumber, issueTicket, and payment.

4. The PaymentGateway class represents the payment system used by the TVM that provides options to pay with cash or a card. It has attributes such as paymentType, paymentAmount, CashAmount, CreditCardNumber, CreditCardExpiry, and CVVNumber, and functions such as processPayment, validateCardDetails, verifyPayment, and checkBalance.

5. The Bank class represents the regulatory body that authenticates and verifies payments made by travelers. It has attributes such as CreditCardNumber, CVVNumber, transactionAmount, and transactionID, and functions such as searchAccountDetails, authenticatePayment, and verifyPayment.

6. The Payment class represents details about a transaction. It has attributes such as amount, transactionID, and paymentType, and functions such as setPayment, getBalanceAmount, isPaymentSuccessful, checkBalance, and getTransactionCode.

7. The ReceiptPreference class represents the evidence of a transaction and includes the date, time, and price which can be email and paper-based. It has attributes such as paper, email,

date, time, and price, and functions such as getEmail, setEmail, isPaperBased, setPaperBased, and generateReceipt.

8. There are several relationships between the classes, such as PurchaseTicket and RechargeMetroCard with igo_TVM and PaymentGateway, PaymentGateway with igo_TVM and Bank, Bank with PaymentGateway, Payment, and ReceiptPreference, Payment with Bank and ReceiptPreference, and ReceiptPreference with igo_TVM and Payment.

Overall, this UML Class diagram provides a clear representation of the various classes, their attributes, functions, and relationships within the Ticket Vending Machine system.

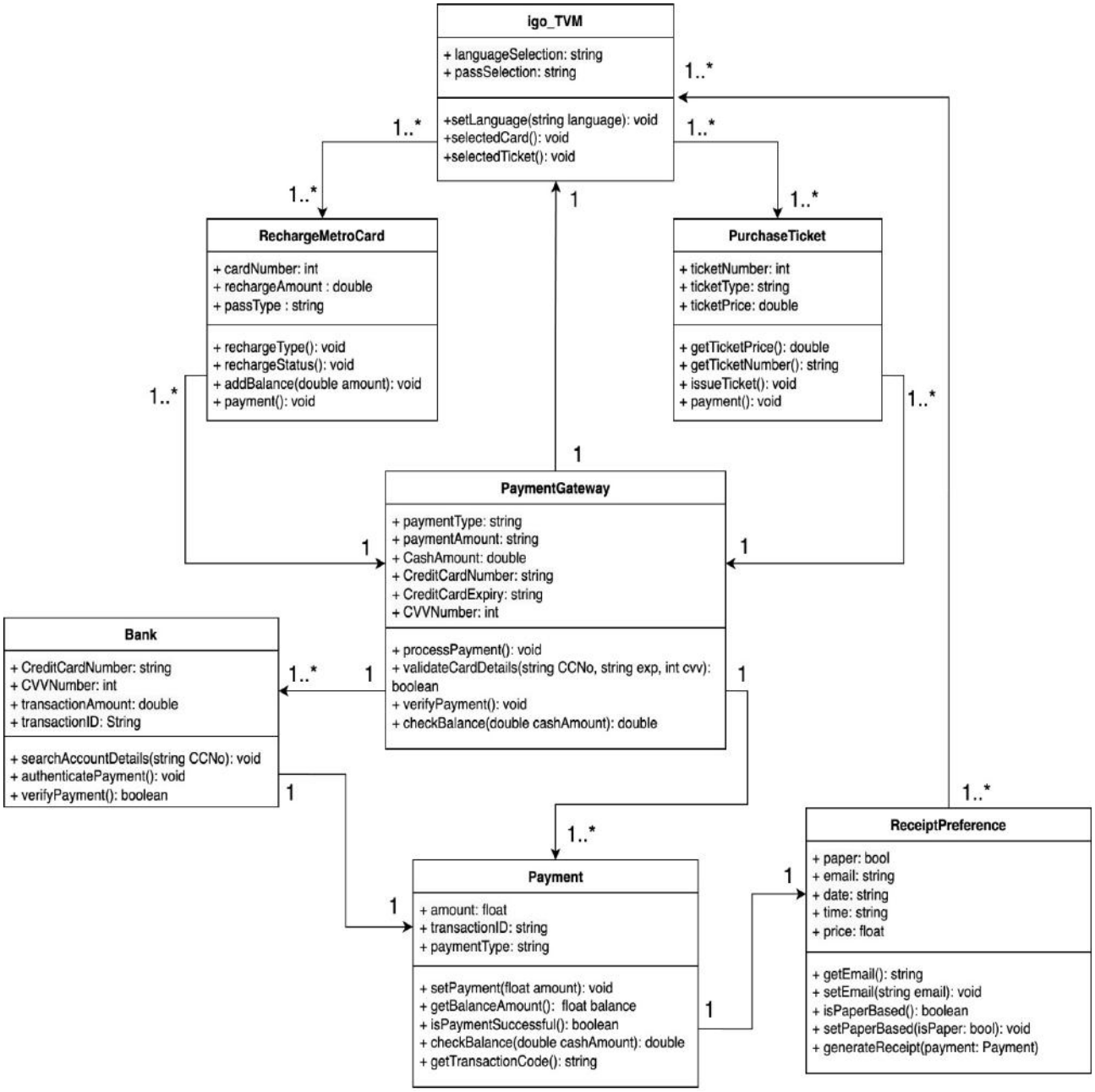


Figure 2.1: UML Class Diagram

2.2 Part-B

The behavioral design model is a type of software design model that focuses on the behavior of the system. It describes the functionality of the system by identifying and specifying the interactions between different components or objects in the system. The behavioral design model is concerned with the dynamic aspects of the system, such as the flow of control, communication, and data between different objects or components. The goal of the behavioral design model is to ensure that the system is functionally correct and meets the requirements of the users.
The behavioral design model can be expressed in terms of sequence diagrams. Sequence diagrams show the interactions between objects in a system in a time-based sequence. Here is a brief description of the Ticket Vending Machine system in terms of a sequence diagram:

1. The user selects the language they want to use on the TVM. This sends a message to the igo_TVM object to set the language.
2. The user selects the type of pass they want to purchase or recharge. This sends a message to the igo_TVM object to select the pass type.
3. The user selects the payment method they want to use. This sends a message to the igo_TVM object to select the payment method.
4. If the user selects the card payment method, the igo_TVM object sends a message to the PaymentGateway object to process the payment.
5. If the user selects the cash payment method, they insert the cash into the TVM. The TVM sends a message to the PaymentGateway object to check the balance.
6. The PaymentGateway object sends a message to the Bank object to authenticate the payment.
7. The Bank object verifies the payment and sends a message back to the PaymentGateway object.
8. If the payment is successful, the PaymentGateway object sends a message to the igo_TVM object to issue the pass or recharge the card.
9. The igo_TVM object sends a message to the ReceiptPreference object to generate a receipt for the transaction.
10. The ReceiptPreference object generates the receipt and sends it to the user via the selected method (paper or email).
11. The sequence ends when the transaction is complete and the user receives their pass or recharge.

It is possible to have multiple such diagrams. Therefore, there should be a collection of N UML Sequence Diagrams. Below are the N UML Sequence Diagrams for our igo TVM model:

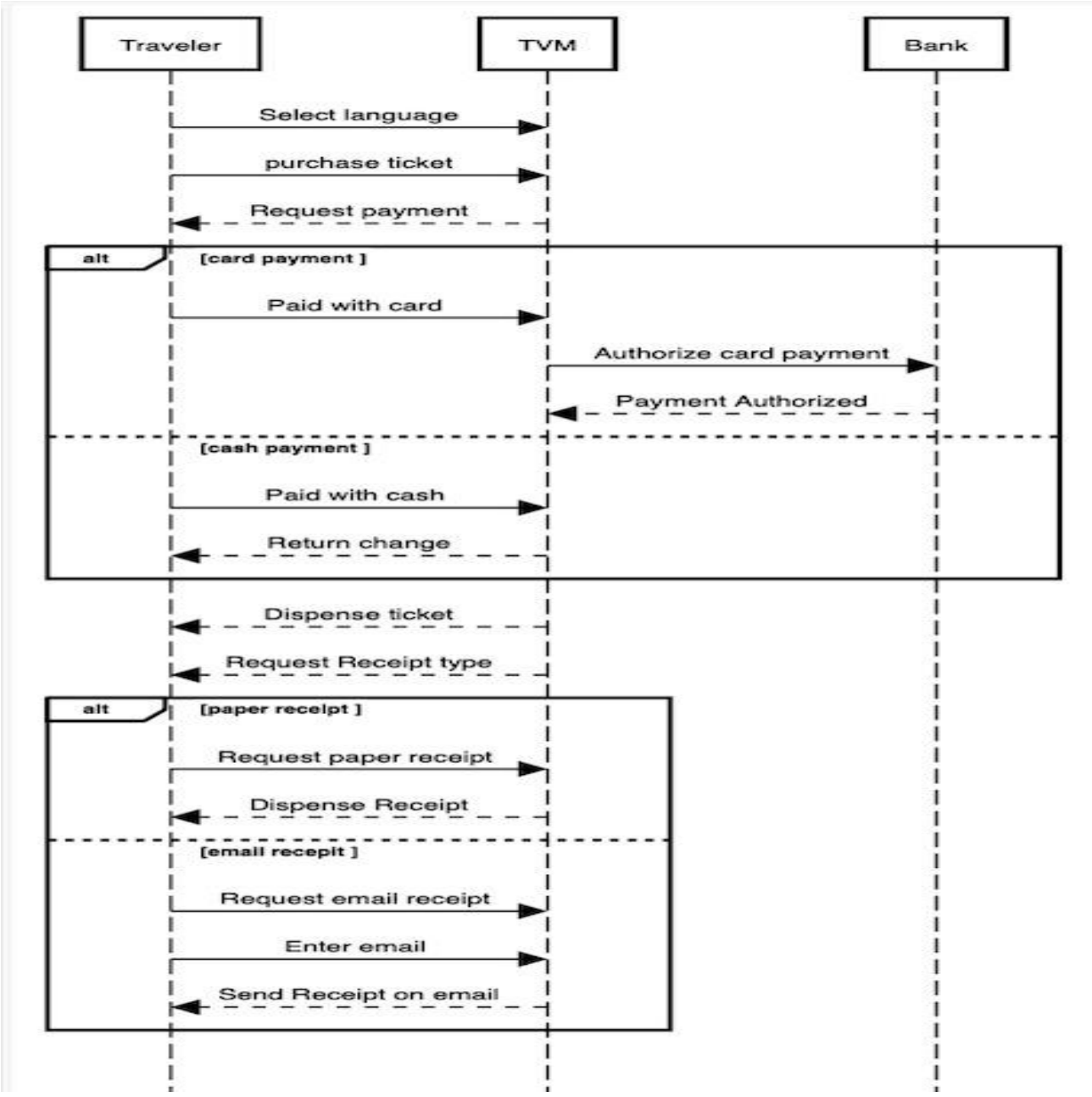


Figure 2.2: UML Sequence Diagram 1

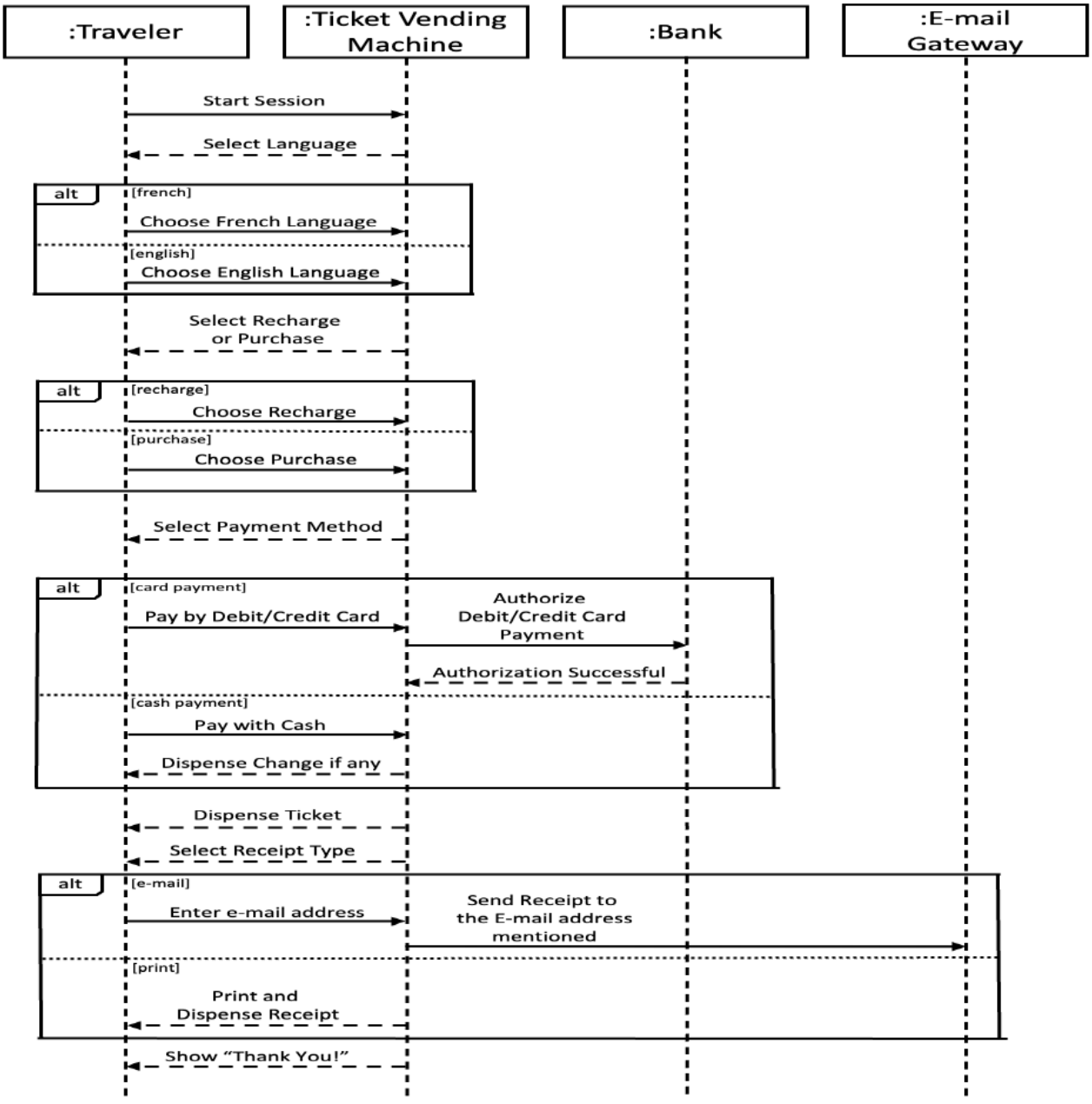


Figure 2.3: UML Sequence Diagram 2

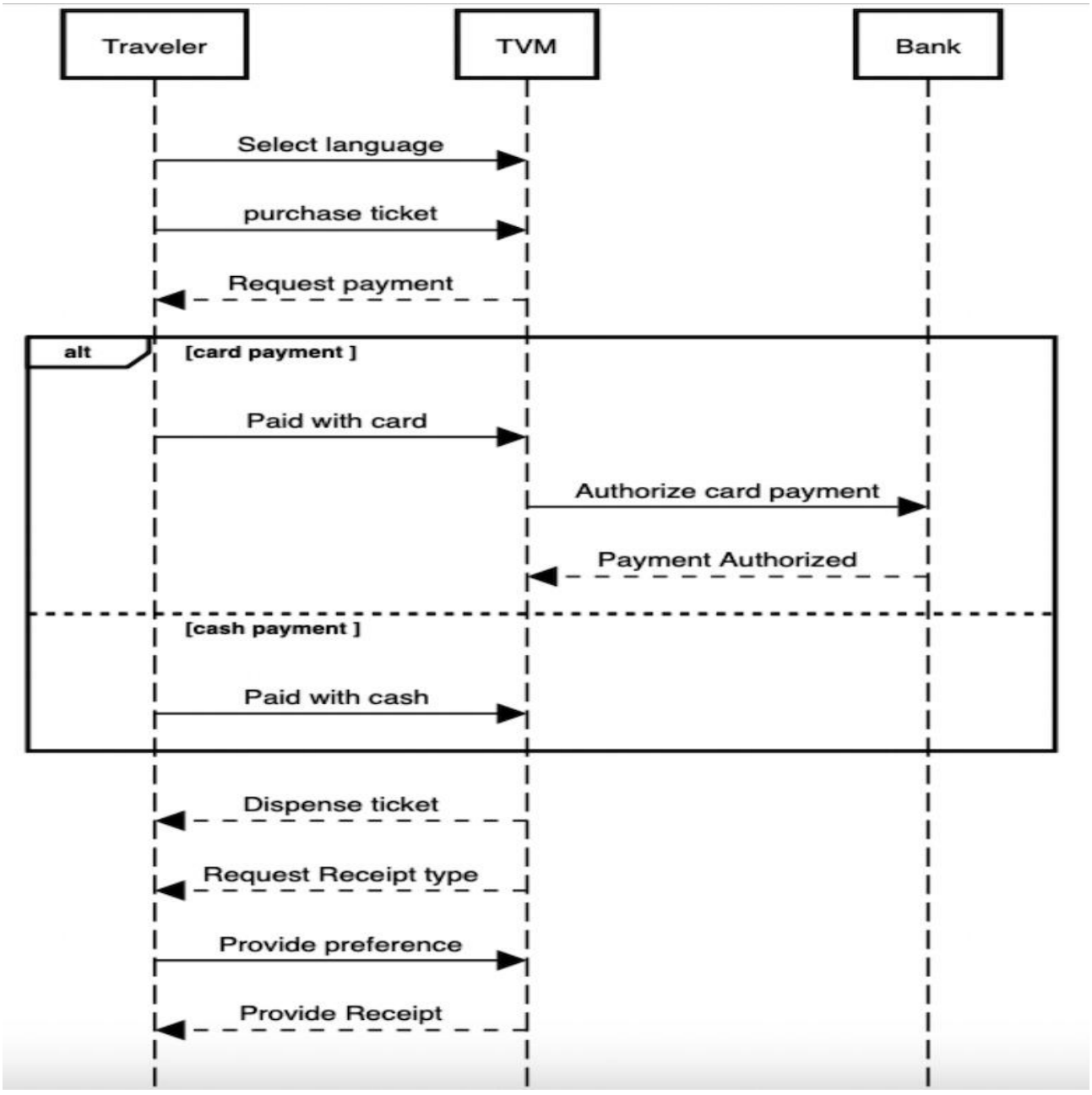


Figure 2.4: UML Sequence Diagram 3

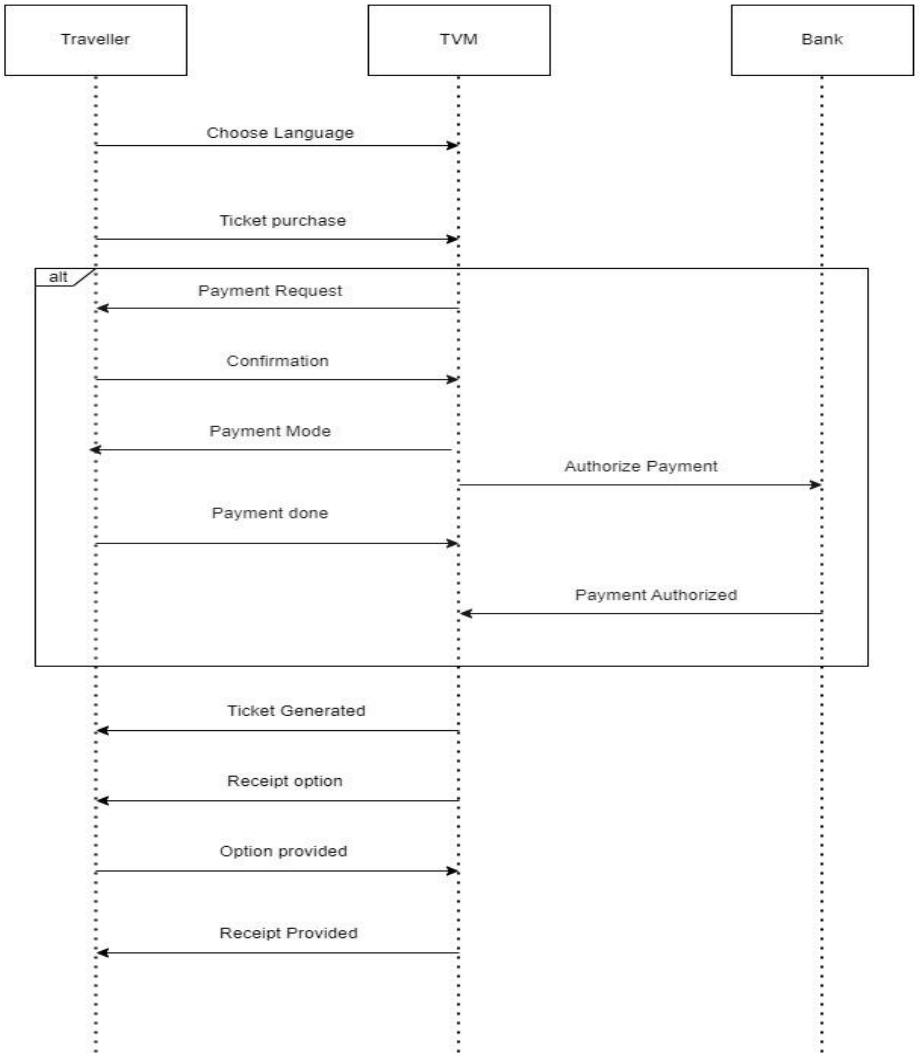


Figure 2.5: UML Sequence Diagram 4

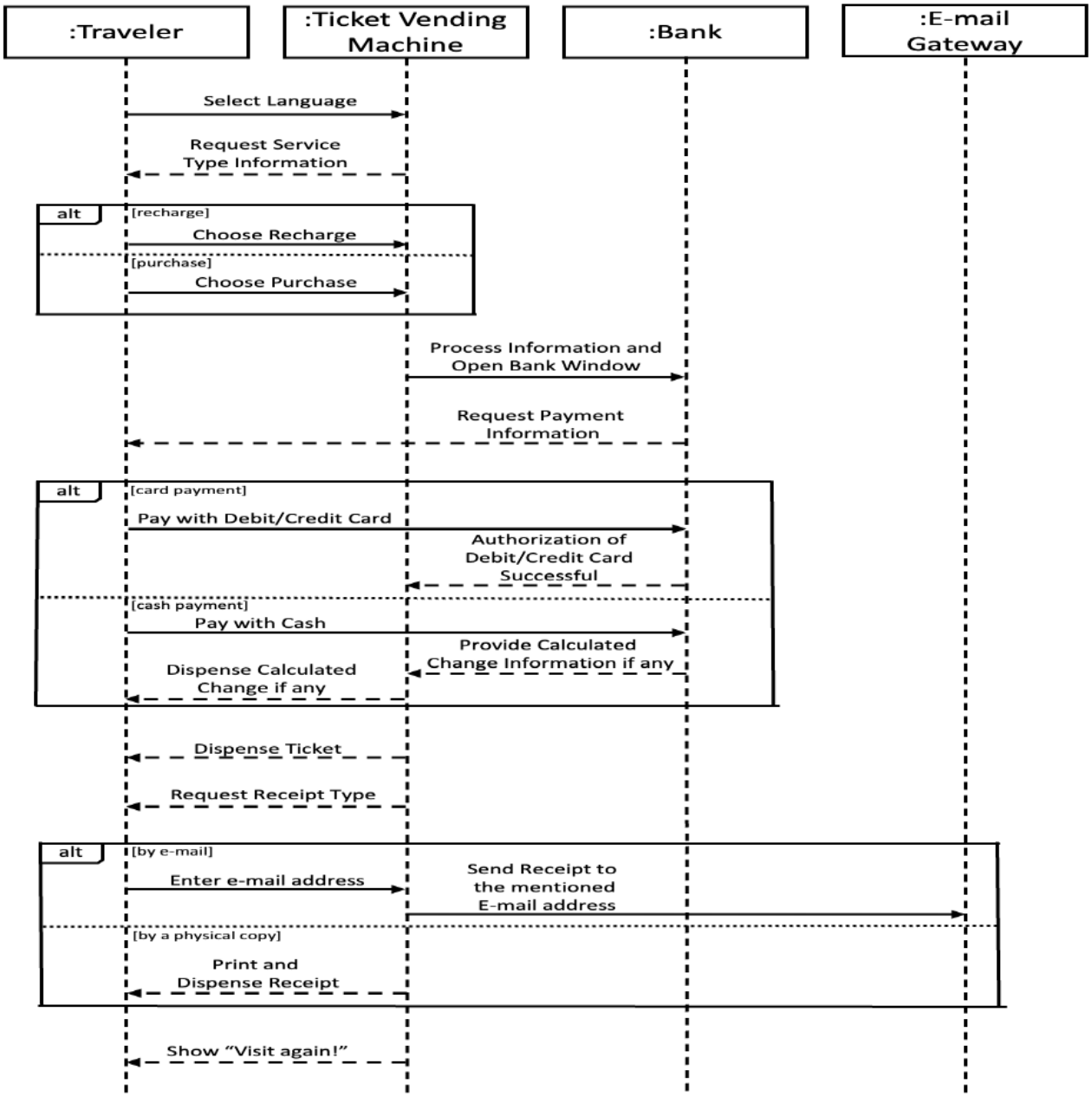


Figure 2.6: UML Sequence Diagram 5

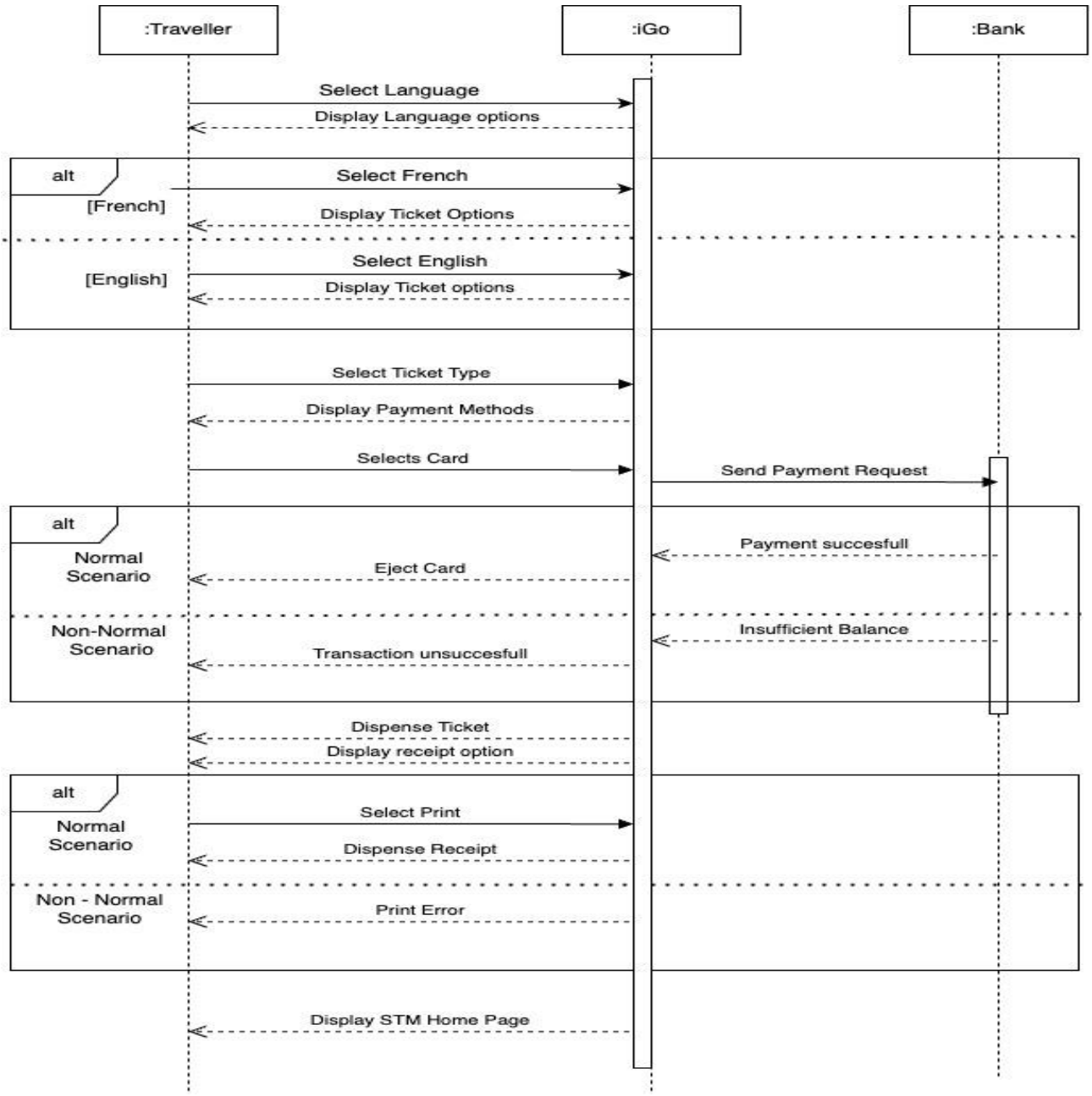


Figure 2.7: UML Sequence Diagram 6

PROBLEM 8

The igo TVM project is implemented using Java 8 for backend programming and JavaFX for graphical user interface (GUI). The program follows standard programming style along with object-oriented design principles, practices, and patterns.

3.1 Brief description of program implementing the low-level solution domain model of iGo

The main class starts by creating a main screen GUI with options to "Select Language" , "Recharge Metro Card" and "Purchase Ticket". The user can choose language among English and French based on his/her comfort. If the user clicks the "Recharge Metro Card " button, he/she is redirected to select the pass type option. This implementation allows the user to recharge metro card for a predefined set of durations using cash or credit/debit cards:

- 2 Weeks - 35 CAD
- 1 Month - 56 CAD
- 2 Month - 98 CAD
- 3 Month - 135 CAD

If the user clicks the "Purchase Ticket" button, he/she is redirected to select the ticket type option. This implementation allows the user to purchase a set of predefined tickets using cash or credit/debit cards:

- 1 trip - $ 3.5
- 2 trips - $ 6
- 1 Day - $ 11
- 1 Week - $ 25
- Unlimited Weekend - $ 17

For the payment, the user has two options: "Pay by Cash" or "Pay by Credit/Debit Card". If the user selects "Pay by Cash" , the user is taken to the next screen which has a dialog box with title "Enter cash". It has an input field and a 'Confirm' button. The input field is restricted to accept only numbers and a decimal point. On clicking the "Confirm" button this dialog box turns into a message dialog. This message dialog tells you about payment authorization status, whatever it may be according to the amount of cash entered. 'OK' button on this message dialog closes the dialog and control is taken back to the main screen. If "Pay by Credit/Debit Card" is selected, the user is taken to a new screen to input the card details. Once entered and the confirm button is hit, the GUI class will call the validateCard() method to check the validity of the card details. If payment is accepted, the user is taken to the receipt selection screen, where two options are present: paper or email receipt.
Note : Please refer this drive link or YouTube link for the demo video of our igo application:
Google Drive link: Link
YouTube Link: Youtube

PROBLEM 9

Testing our igo TVM using representative sample data means to perform a series of tests on the Ticket Vending Machine (TVM) using a set of data that represents the typical or expected use cases and scenarios. The goal of these tests is to verify that the TVM system behaves as expected, and to identify any issues or defects that need to be addressed before the system is deployed or released to the public.
Representative sample data should cover both positive and negative test scenarios, including:

- Successful purchases of different types of tickets using cash and credit/debit cards.
- Successful recharging of smart cards with different amounts using cash and credit/debit cards.
- Handling of incorrect or invalid inputs, such as wrong card information, insufficient funds, or incorrect ticket selections.
- Handling of system errors or malfunctions, such as network failures or hardware issues.
- Verification of the accuracy and correctness of transactions and receipts generated by the system.

The sample data should be designed to cover a range of possible scenarios and edge cases, to ensure that the TVM system is robust and reliable in a variety of real-world situations.

4.1 Negative Potential Uses

Negative potential uses refer to scenarios where the user intentionally or unintentionally tries to misuse the system, resulting in undesired outcomes. Testing for negative potential uses helps identify vulnerabilities and security risks in the system.

Below are the few negative potential test cases for igo TVM mode:

- If the user inserts invalid metro card then it cannot be verified in the test as it requires a hardware device and backend database, which is beyond the scope of this implementation.
- Test for insufficient balance with a valid debit/credit card is performed, however, verifying the balance amount is not possible in this implementation as it requires accessing the bank database, which is beyond the scope of this implementation.
- This test case involves the user using counterfeit currency to purchase a ticket. The system is expected to reject the payment, but as the verification is not feasible to implement, it is not included in this implementation.

These are just a few examples of test cases, and it is not an exhaustive list. The actual test cases may vary depending on the specific requirements and functionalities of the TVM system.

Below is the list of important User Acceptance Test Cases to test our igo TVM :

S.No	User Acceptance Test cases	Pass/Fail
1	Verify that the user can select their desired language.	
2	Verify that the user can recharge their metro card using cash.	
3	Verify that the user can recharge their metro card using credit/debit card	
4	Verify that the user can purchase a paper ticket using cash.	
5	Verify that the user can purchase a paper ticket using a credit/debit card.	
6	Verify that the user inserts cash equal or more than the required recharge/ticket price.	
7	Verify that the user inputs a valid Credit card number.	
8	Verify that the user can select their preferred receipt option.	

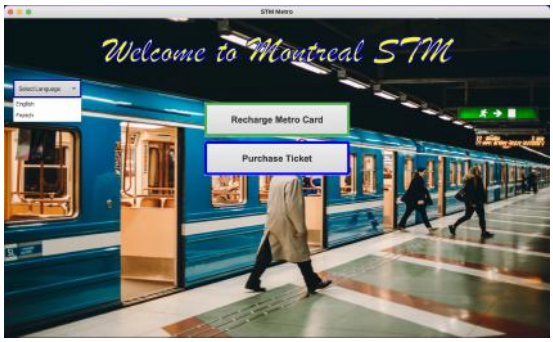
Table 4.1: User Acceptance Testcases

4.2 User Acceptance Test 1

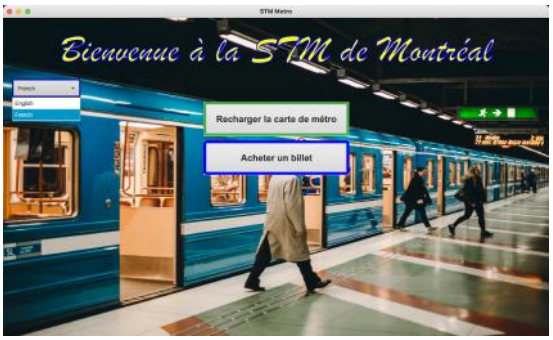
Test case : Verify that the user can select their desired language.
Test steps :

1. Press the "Select Language" button.
2. Verify that a list of available languages is displayed.
3. Select a language.
4. Verify that the TVM displays information in the selected language.

Screenshots



(a) Screenshot 1



(b) Screenshot 2

(a) Screenshot 3

User Acceptance Test case 1	Pass/Fail
Verify that the user can select their desired language.	Pass

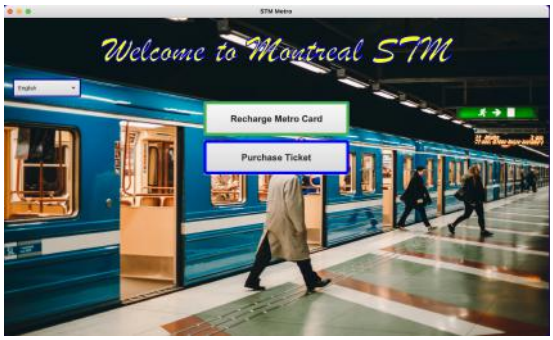
Table 4.2: User Acceptance Testcases

4.3 User Acceptance Test 2

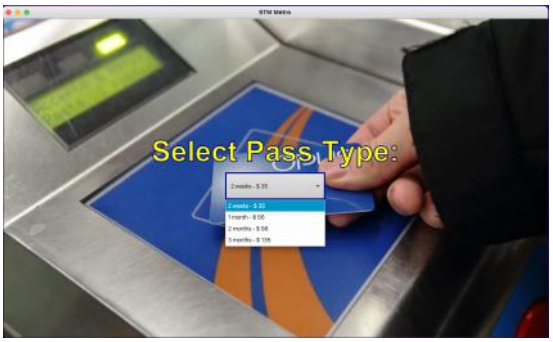
Test case : Verify that the user can recharge their metro card using cash.
Test steps :

1. Press the "Recharge Card" button.
2. Select the "Pay by Cash" option.
3. Insert payment.
4. input the cash amount inserted.
5. Verify that the recharge details are displayed on the screen.

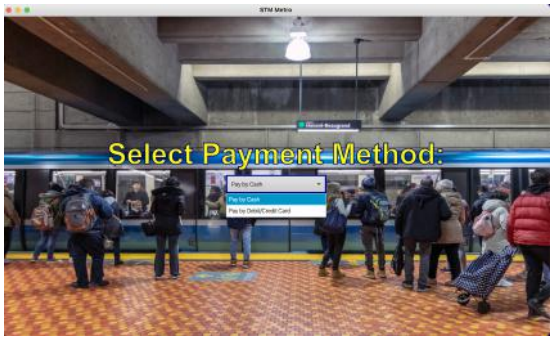
Screenshots



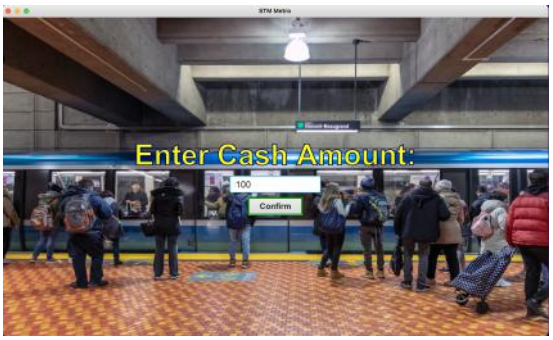
(a) Screenshot 1



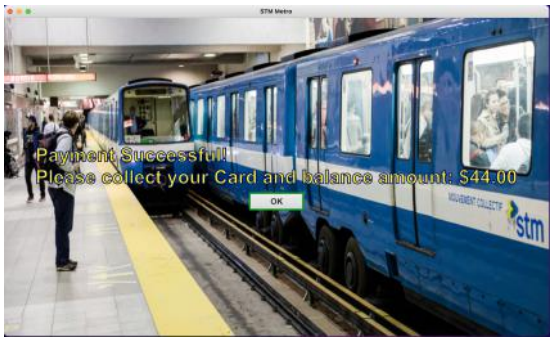
(b) Screenshot 2



(a) Screenshot 3



(b) Screenshot 4



(a) Screenshot 5

User Acceptance Test case 2	Pass/Fail
Verify that the user can recharge their metro card using cash.	Pass

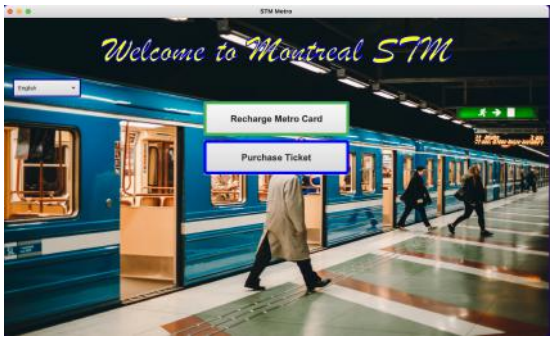
Table 4.3: User Acceptance Testcases

4.4 User Acceptance Test 3

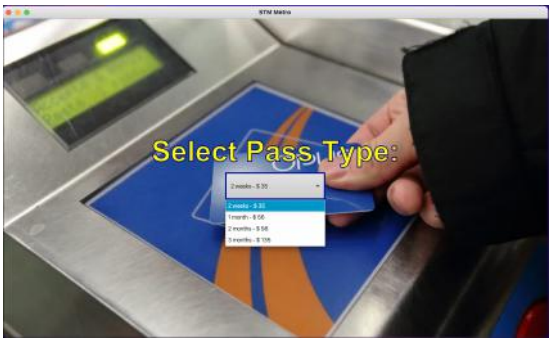
Test case : Verify that the user can recharge their metro card using credit/debit card.
Test steps :

1. Press the "Recharge Card" button.
2. Select the "Pay by Debit/Credit Card" option.
3. Enter card details.
4. Validate card Details.
5. Verify that the recharge details are displayed on the screen.

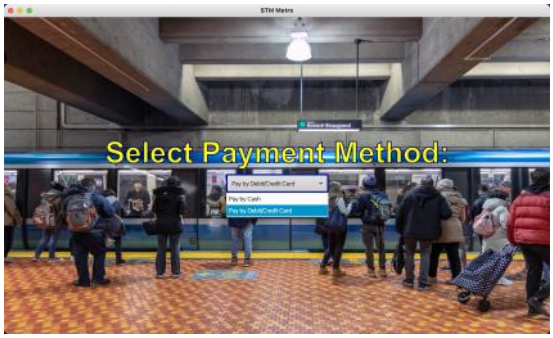
Screenshots



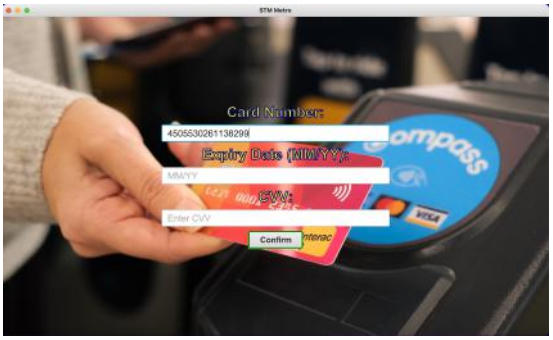
(a) Screenshot 1



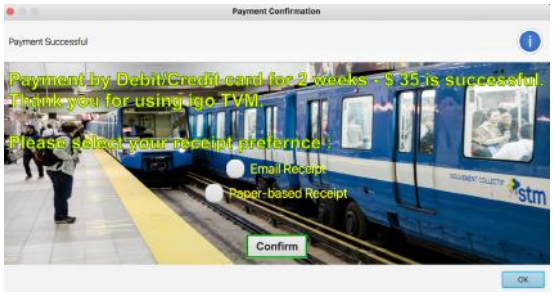
(b) Screenshot 2



(a) Screenshot 3



(b) Screenshot 4



(a) Screenshot 5

User Acceptance Test case 3	Pass/Fail
Verify that the user can recharge their metro card using credit/debit card.	Pass

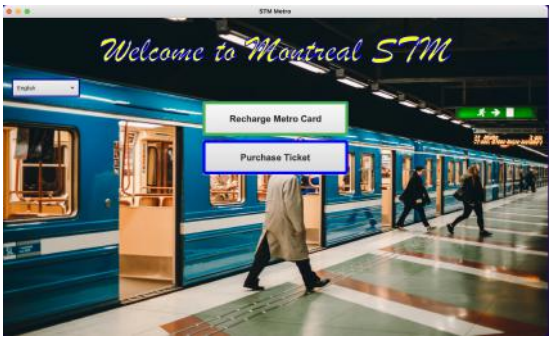
Table 4.4: User Acceptance Testcases

4.5 User Acceptance Test 4

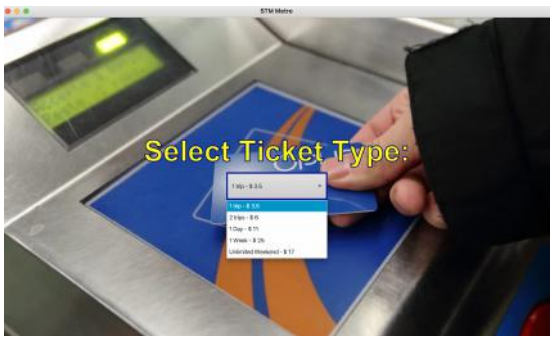
Test case : Verify that the user can purchase a paper ticket using cash.
Test steps :

1. Press the "Purchase Ticket" button.
2. Select the "Pay by Cash" option.
3. Insert payment.
4. input the cash amount inserted.
5. Verify that the ticket details are displayed on the screen.

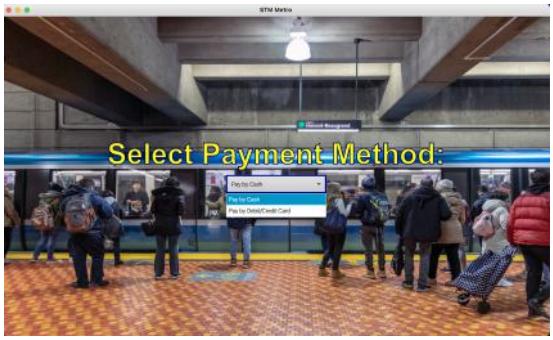
Screenshots



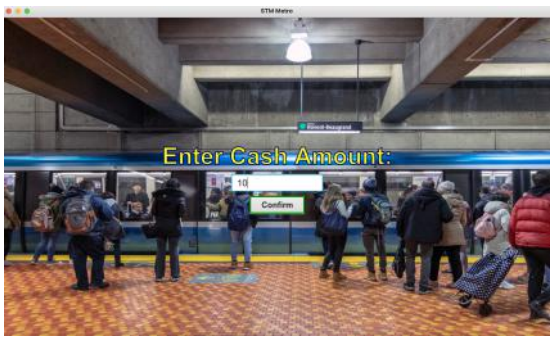
(a) Screenshot 1



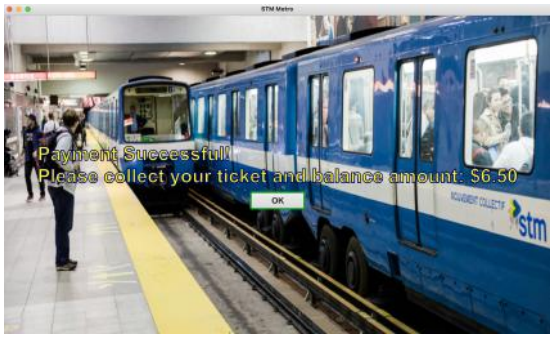
(b) Screenshot 2



(a) Screenshot 3



(b) Screenshot 4



(a) Screenshot 5

User Acceptance Test case 4	Pass/Fail
Verify that the user can purchase a ticket using cash.	Pass

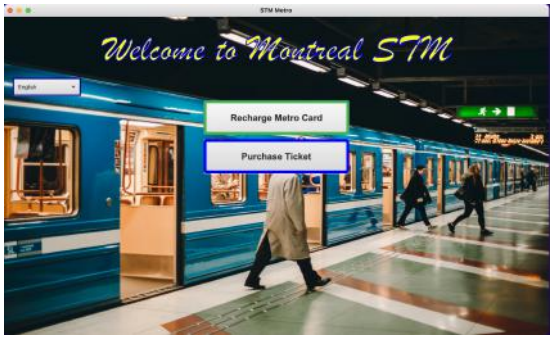
Table 4.5: User Acceptance Testcases

4.6 User Acceptance Test 5

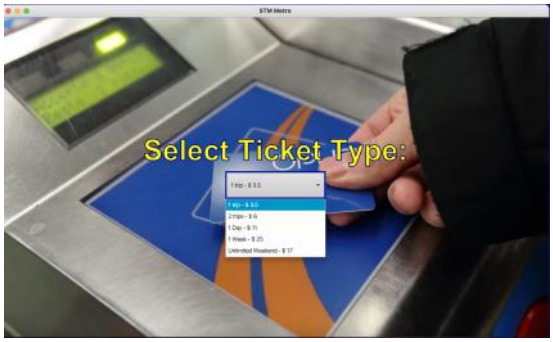
Test case : Verify that the user can purchase a ticket using credit/debit card.
Test steps :

1. Press the "Purchase Ticket" button.
2. Select the "Pay by Debit/Credit Card" option.
3. Enter card details.
4. Validate card Details.
5. Verify that the purchase details are displayed on the screen.

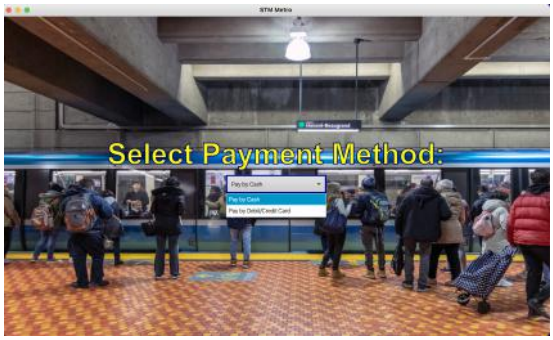
Screenshots



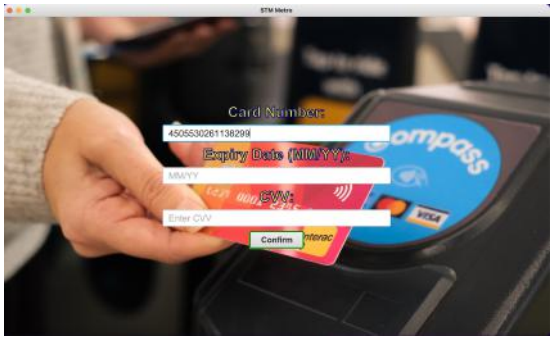
(a) Screenshot 1



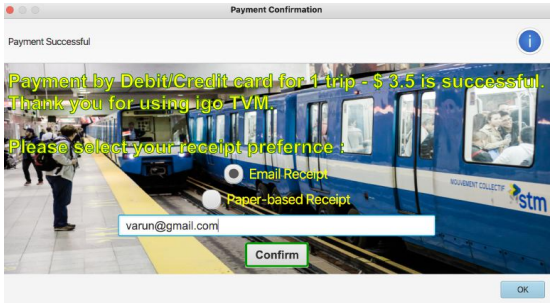
(b) Screenshot 2



(a) Screenshot 3



(b) Screenshot 4



(a) Screenshot 5

User Acceptance Test case 5	Pass/Fail
Verify that the user can purchase a ticket using a credit/debit card.	Pass

Table 4.6: User Acceptance Testcases

4.7 User Acceptance Test 6

Test case : Verify that the user inserts cash equal or more than the required recharge/ticket price.
Test steps :

1. Select the "Pay by Cash" option.
2. Insert payment.
3. input a cash amount inserted.

4. Verify that the inserted amount is equal or more than the ticket price.
5. If the inserted amount is less, then an alert box should pop up with the error message.

Screenshots

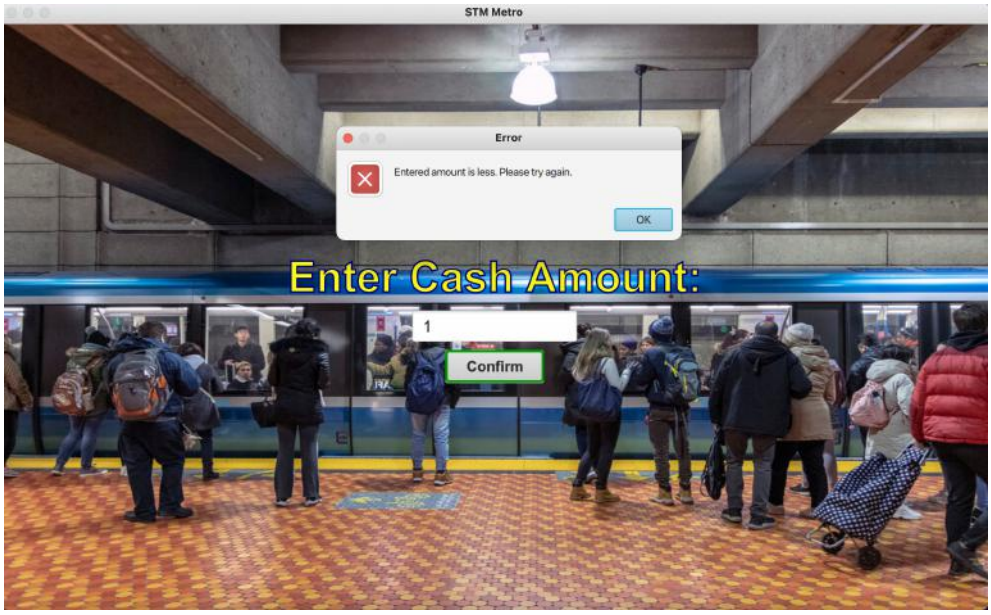


Figure 4.15: Screenshot 1

User Acceptance Test case 6	Pass/Fail
Verify that the user inserts cash equal or more than the required recharge/ticket price.	Pass

Table 4.7: User Acceptance Testcases

4.8 User Acceptance Test 7

Test case : Verify that the user inputs a valid Credit card number.
Test steps :

1. Press the "Pay with Credit Card" button.
2. Enter credit card details.
3. Verify that the user enters the correct 16 digits card number.
4. Verify that the user enters a valid expiry date.
5. Verify that the user enters 3 digits CVV number.
6. If the inserted card details are invalid, then an alert box should pop up with the error message.

Screenshots

(a) Screenshot 1

(b) Screenshot 2

(a) Screenshot 3

(b) Screenshot 4

User Acceptance Test case 7	Pass/Fail
Verify that the user inputs a valid Credit card number.	Pass

Table 4.8: User Acceptance Testcases

4.9 User Acceptance Test 8

Test case : Verify that the user can select their preferred receipt option.
Test steps :

1. Press the "Select Receipt Option" button.
2. Select "Email" or "Paper" receipt.
3. Verify that the receipt is generated in the selected format.

Screenshots

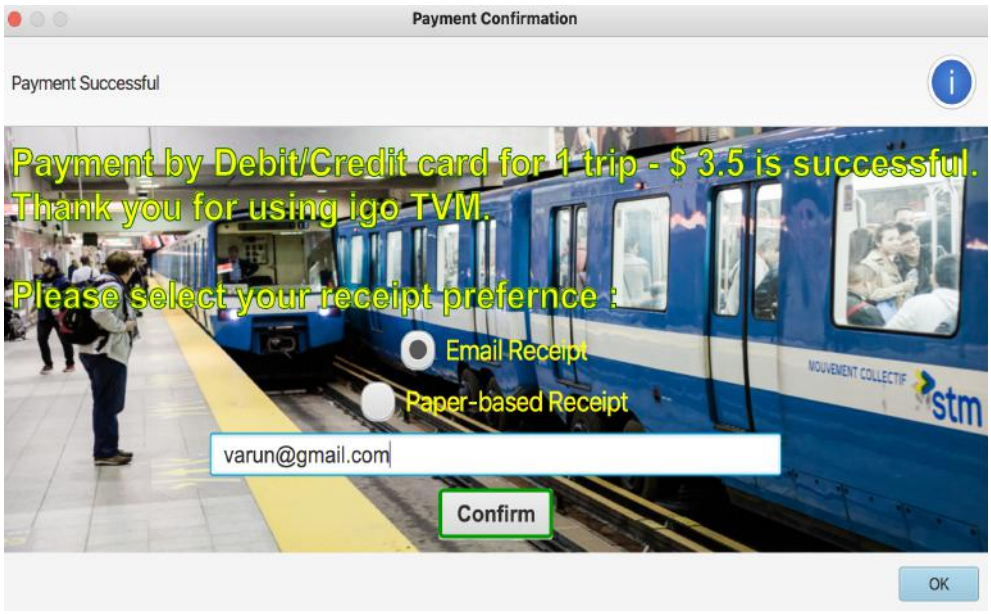


Figure 4.18: Screenshot 1

User Acceptance Test case 8	Pass/Fail
Verify that the user can select their preferred receipt option.	Pass

Table 4.9: User Acceptance Testcases

All the mentioned User Acceptance Test cases are passed Successfully.

S.No	User Acceptance Test cases	Pass/Fail
1	Verify that the user can select their desired language.	Pass
2	Verify that the user can recharge their metro card using cash.	Pass
3	Verify that the user can recharge their metro card using credit/debit card	Pass
4	Verify that the user can purchase a paper ticket using cash.	Pass
5	Verify that the user can purchase a paper ticket using a credit/debit card.	Pass
6	Verify that the user inserts cash equal or more than the required recharge/ticket price.	Pass
7	Verify that the user inputs a valid Credit card number.	Pass
8	Verify that the user can select their preferred receipt option.	Pass

Table 4.10: User Acceptance Testcases

References

Collaboration environments

1. **Github :** https://github.com/imadshawl/SOEN-6461
2. **Drive :** https://drive.google.com/drive/folders/123rlWX5hwoD6EMAZ576rDuOchzaqRcIQ?usp=sharing

REFERENCES

1. https://www.stm.info/en
2. **Lecture notes by Prof. Pankaj Kamthan**