**2. REQUIREMENTS:**

**2. The user applying for a Credit card**

2.1. The **Customer** logs in to their bank account.

2.2. The **Customer** applies for the Credit card and fills out the **Credit card application form.**

2.3. The **Customer database** verifies and validates the Customer’s **Credit card application form** and saves the information.

2.4. The **Online banking system** approves or denies the **Credit card application** and sends the confirmation to the **Customer**.

2.5. If approved, the **Online banking system** will create the Customer a Credit card account and the **Customer's Credit card details** are saved in the **Credit card database.**

2.6. Then the **Credit card activation** is successfully completed by the **Online banking system.**

**Use Case: UC-2 “User applying for Credit card”**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Use Case Name: User applying for Credit card** | | **ID:** **UC-2** | **Priority:** High | |
| **Actors:**  **Customer, Online banking system** | | | | |
| **Description:** **This use case walks through the steps a user goes through to apply for a Credit card.** | | | | |
| **Trigger:  The user wants to have a Credit card.** | | | | |
| **Type:**           **X** External Temporal | | | | |
| **Preconditions:**       **1. The user should have a bank account.** | | | | |
| **Normal Course:**  **2.0. Processes Login Page**  **2.0.a.** The **Customer** logs into their bank account.  **2.0.b.** The **Customer database verifies** whether the Customer **login credentials** are valid or invalid.  **2.0.c.** If the Customer's **login credentials** are **valid**, the **Customer database** sends a"**login access**” message to the **Customer.**  **2.0.d.** If the Customer's **login credentials** are **invalid**, the **Customer database** asks the Customer to **re-authenticate.** Go to step 2.0.a  **2.1. Processes CC Application Form**  **2.1.a.** The **Customer** applies for and submits a Credit card application form with the required information.  **2.1.b.** The **Online banking system** saves the details of Customers' Credit card applications.  **2.2.  Processes CC Approval**  **2.2.a.** The **Online banking system (OBS)** is used to verifyand validate **Customer details** as part of the **Credit card approval process.**  **2.2.b**. Then the **Credit card approval process** sends an “**approved or denied Ack**” message to the **Customer** about their Credit card application status.  **2.3. Processes CC Issuance**  **2.3.a.** Once Credit card application is approved**,** then **Credit card number** is generated by the **Online banking system.**  **2.3.b.** And then the **Credit card account** is created by the **Online banking system.**  **2.3.c.** The **credit card account and card details** are stored in the **Credit card database.**  **2.3.d.** Thenthe **Credit card database** sends confirmation of the CC “**Activation Ack**” message to the Customer. | | **Information for Steps:**  **2.0.a ←** login credentials  **2.0.b →**login credentials verification  **2.0.c ←** "login valid” message  **2.0.c →** "login access” message  **2.0.d ←**” login invalid” message  **2.0.d →”** re-authenticate" message.  **2.1.a ← CC** Application details  **2.1.b → CC** Application details  **2.2.a ←**Customer details verification validation      **2.2.b → “**Approve or Decline Ack” message      **2.3.a ←** CC number generation  **2.3.b ←** CC Account creation  **2.3.c →** CC number details  **2.3.c →** CC Account details  **2.3.d ←** CC “Activation Ack” message | | |
| **Alternative Course:**       Not implemented | | | | |
| **Post conditions: The user can access the Credit card and use it.** | | | | |
| **Summary Input** | **Source** | **Summary Output** | | **Destination** |
| **2.0.a** login credentials  **2.0.c** "login valid” message  **2.0.d”** login invalid” message  **2.1.a** CC Application details  **2.2.a** Customer details verification validation  **2.3.a** CC number generation  **2.3.b** CC Account creation | **2.0.a** Customer  **2.0.c** Customer database  **2.0.d** Customer database  **2.1.a** Customer  **2.2.a** Online banking system  **2.3.a** Online banking system  **2.3.b** Online banking system  **2.3.d** Customer | **2.0.b** login credentials verification  **2.0.c** "login access” message  **2.0.d”** re-authenticate" message.  **2.1.b CC** Application details  **2.2.b “**Approve or Decline  Ack” message  **2.3.c** CC number  details  **2.3.c** CC Account details  **2.3.d** CC “Activation Ack” message | | **2.0.b** Customer database  **2.0.c** Customer  **2.0.d** Customer  **2.1.b** Online banking system  **2.2.b** Customer  **2.3.b** Credit card database  **2.3.c** Credit card database  **2.3.d** Credit card database |

**UML of UC-2 “User applying for Credit card”:**

**Diagram

Description automatically generated**

**DFD-2 “User applying for Credit card” matches Use Case UC-2:**

**Diagram

Description automatically generated**

**Description of UC-2 “User applying for Credit card”**

* **2.0. Processes Login Page**

1. The “Customer” (External Entity “Customer”) sends IDF 2.0.a “login credentials” to P2.0. Processes Login Page. P2.0 sends ODF 2.0.b “login credentials verification” to DS D1 “Customer database”.
2. If the login credentials are valid, P2.0. “Processes Login Page” obtains the message “login valid” (IDF “2.0.c login valid”) from DS D1 “Customer database”. P2.0. “Processes Login Page” messages “login access” (ODF 2.0.c "login access”) message to” (External Entity “Customer”) enabling the User to access the bank account.
3. If the login credentials are invalid, P2.0. “Processes Login Page” obtains the message “login invalid” (IDF “2.0.d login invalid”) from DS D1 “Customer database”. P2.0. “Processes Login Page” messages “re-authenticate” (ODF 2.0.d "re-authenticate”) message to” (External Entity “Customer”) enabling the User to access the bank account.

Once login process is done, customer will start

* **2.1. Processes CC Application Form**

1. The Customer (External Entity “Customer”) sends CC Application details (IDF 2.1.a “CC Application details”) to P.2.1. Processes CC Application Form. P.2.1 Processes CC Application Form sends “CC Application details” (ODF 2.1.b “CC Application details”) to DS D2 Online banking system.

* **2.2.  Processes CC Approval**

1. The DS D2 Online banking system sends Customer details verification validation (IDF 2.2.a “Customer details verification validation”) to P.2.2.  Processes CC Approval. P.2.2.  Processes CC Approval sends “Approve or Decline Ack” (ODF 2.2.b “Approve or Decline Ack”) message to the Customer (External Entity “Customer”)

* **2.3. Processes CC Issuance**

1. The DS D2 Online banking system sends “CC number generation” (IDF 2.3.a “CC number generation”) and “CC Account creation” (IDF 2.3.b “CC Account creation”) to P.2.3 Processes CC Issuance.
2. The P.2.3 Processes CC Issuance sends “CC number details” (ODF 2.3.c “CC number details”) and “CC Account details” (ODF 2.3.c “CC Account details”) to the DS D3 Credit card database.
3. Then the P.2.3 Processes CC Issuance sends “CC Activation Ack” (ODF 2.3.d “CC Activation Ack”) message to the Customer (External Entity “Customer”)

## **CLASS DIAGRAM UC-2:**

Graphical user interface, application

Description automatically generated

**Description of Class Diagram CD-1:**

**Class Customer:**

**Description:**

Represents the Customer that interacts with the Customer database.

**Traceability:**

**Matches DFD-2 EE “Customer”**

**Multiplicity:**

An instance of class “Customer” holds as [1] – to - [1] instance of class “CustomerDatabase.”

**Main attributes:**

* **loginCredentials:** Holds the customer’s identification information (**matches DFD-2, IDF 2.0.a login credentials)**
* **customerApplicationDetails:** Holds the customer’s information (**matches DFD-2, IDF 2.1.b CC Application details)**

**Main Methods:**

* **getLoginCredentials() :** Returns private attribute “loginCredentials” (**matches DFD-2, IDF 2.0.a login credentials)**
* **authenticatesCustomer:** calls method “CustomerDatabase::aauthenticatesCredentials() **(matches DFD-2, P 2.0 Processes Login Page**)
* **getCreditCardApplicationDetails() :** calls method “Custonerdatabase::getCreditCardApplicationDetails() **(matches DFD-2, P 2.1 Processes CC Application Form)**
* **receivesApproveorDeclineAckMessage():** calls method “Custonerdatabase:: receivesApproveorDeclineAckMessage() **(matches DFD-2, P 2.2 Processes CC Approval**)
* **receivesCreditCardActivationAckMessage: (matches DFD-2, P 2.3 Processes CC Issuance**)

**Class CustomerDatabase:**

**Description:**

Represents the Customer that interacts with Cistomer database.

**Traceability:**

**Matches DFD-2 DS D1 “Customerdatabase”**

**Multiplicity:**

An instance of class “Customer CustomerDatabase” holds as [1] – to - [1] instance of class “Customer”

**Main attributes:**

* **loginCredentials:** Holds the customer’s identification information (**matches DFD-2, IDF 2.0.a login credentials)**

**Main Methods:**

* **authenticatesCustomer:** calls method “CustomerDatabase::aauthenticatesCredentials() **(matches DFD-2, P 2.0 Processes Login Page**)

**Class OnlineBankingSystem:**

**Description:**

Represents the Customer that interacts with the Online Banking System.

**Traceability:**

**Matches DFD-1 DS D2 “Online banking system”**

**Multiplicity:**

An instance of class “Customer” holds as [1] – to - [1] instance of class “OnlineBankingSystem”

**Main attributes:**

* **creditCardApplicationDetails():** Holds the customer’s credit Card Application Details information (**matches DFD-2, ODF 2.1.b CC Application details)**
* **creditCardNumberGeneration() :** generates credit card number **(matches DFD-2, IDF 2.3.a “CC number generation”)**
* **creditCardAccountCreation() :** Creates credit card account (**matches DFD-2, ODF 2.3.b “CC Account creation”)**

**Main Methods:**

* **customerDetailsVerificationValidation() :** calls method “OnlinebankingSystem::customerDetailsVerificationValidation” (**matches DFD-2, IDF 2.2.a “Customer details verification validation”)**

**Class CreditCarddatabase:**

**Description:**

Represents the Customer that interacts with the credit card database.

**Traceability:**

**Matches DFD-1 DS D3 “Credit card database”**

**Multiplicity:**

An instance of class “CreditCardDatabase.” holds as [1] – to - [1] instance of class “Customer”

**Main attributes:**

* **loginCredentials:** Holds the customer’s identification information (**matches DFD-2, IDF 2.3.c login credentials)**
* **creditCardNumberDetails():** Holds the customer’s credit card number details (**matches DFD-2, ODF 2.3.c CC number details)**
* **creditCardAccountDetails():** Holds the customer’s credit card account details (**matches DFD-2, ODF 2.3.c CC Account details)**

**Main Methods:**

* **authenticatesCustomer:** calls method “CustomerDatabase::aauthenticatesCredentials() **(matches DFD-2, P 2.0 Processes Login Page**)
* **storesCreditCardNumberDetails():** calls method “CreditCardDatabase::storesCreditCardNumberDetails() **(matches DFD-2, P 2.3 Processes CC Issuance**)
* **storesCreditCardAccountdetails():** calls method “CreditCardDatabase:: storesCreditCardAccountdetails() **(matches DFD-2, P 2.3 Processes CC Issuance**)

## **SEQUENCE DIAGRAM SD-2 FOR UC – 2:**

**Diagram

Description automatically generated**

**Description of Sequence Diagram SD 2:**

1. The instance of class “Customer” self-calls “(login credentials)” to return the private attribute “loginCredentails”(**matches DFD-2, IDF 2.0.a login credentials**), method Customer::loginCredentials().
2. Customer class invokes method “creditCardApplicationDetails()” from OnlineBankingSystem(**matches DFD-2, P 2.1 Processes CC Application Form),** method OnlineBankingSystem:: creditCardApplicationDetails ().
3. Customer will receive “ApproveorDeclineMessage**”**from OnlineBankingSystem (**matches DFD-2, P 2.2 processes CC Approval**), method OnlineBankingSystem::ApproveorDeclineMessage(). This sends an approval or declines Ack message to the customer.
4. Online banking system sends “creditCardNumberGeneration” **(matches IDF 2.3.a “CC number generation”)** and “credirCardAccountCreation” **(matches IDF 2.3.b “CC Account creation”) of P.2.3 Processes CC Issuance.**
5. Customer will receive “activationAckMessage**”** from creditcardDatabase **(matches DFD-2, P 2.3 Processes CC Issuance**) method creditcardDatabase:: activationAckMessage(). This sends an activation Ack message to the customer.

**Use case 2- The user applying for a Credit card.**

**Actors are** **Customer, Online banking system.**

**Preconditions of the use case the user should have a bank account.**

**Description of UC-2 “User applying for Credit card”**

* **2.0. Processes Login Page**

1. The “Customer” (External Entity “Customer”) sends IDF 2.0.a “login credentials” to P2.0. Processes Login Page. P2.0 sends ODF 2.0.b “login credentials verification” to DS D1 “Customer database”.
2. If the login credentials are valid, P2.0. “Processes Login Page” obtains the message “login valid” (IDF “2.0.c login valid”) from DS D1 “Customer database”. P2.0. “Processes Login Page” messages “login access” (ODF 2.0.c "login access”) message to” (External Entity “Customer”) enabling the User to access the bank account.
3. If the login credentials are invalid, P2.0. “Processes Login Page” obtains the message “login invalid” (IDF “2.0.d login invalid”) from DS D1 “Customer database”. P2.0. “Processes Login Page” messages “re-authenticate” (ODF 2.0.d "re-authenticate”) message to” (External Entity “Customer”) enabling the User to access the bank account.

Once login process is done, customer will start

* **2.1. Processes CC Application Form**

1. The Customer (External Entity “Customer”) sends CC Application details (IDF 2.1.a “CC Application details”) to P.2.1. Processes CC Application Form. P.2.1 Processes CC Application Form sends “CC Application details” (ODF 2.1.b “CC Application details”) to DS D2 Online banking system.

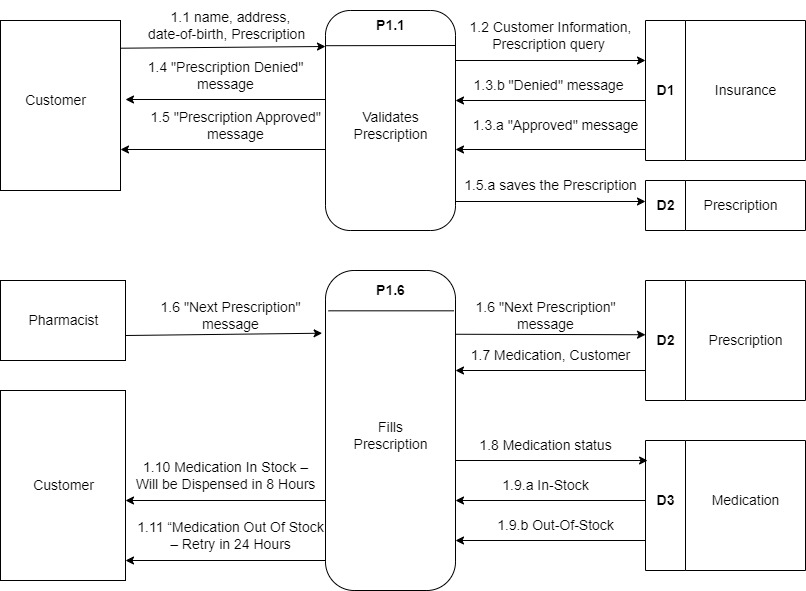
* **2.2.  Processes CC Approval**

1. The DS D2 Online banking system sends Customer details verification validation (IDF 2.2.a “Customer details verification validation”) to P.2.2.  Processes CC Approval. P.2.2.  Processes CC Approval sends “Approve or Decline Ack” (ODF 2.2.b “Approve or Decline Ack”) message to the Customer (External Entity “Customer”)

* **2.3. Processes CC Issuance**

1. The DS D2 Online banking system sends “CC number generation” (IDF 2.3.a “CC number generation”) and “CC Account creation” (IDF 2.3.b “CC Account creation”) to P.2.3 Processes CC Issuance.
2. The P.2.3 Processes CC Issuance sends “CC number details” (ODF 2.3.c “CC number details”) and “CC Account details” (ODF 2.3.c “CC Account details”) to the DS D3 Credit card database.
3. Then the P.2.3 Processes CC Issuance sends “CC Activation Ack” (ODF 2.3.d “CC Activation Ack”) message to the Customer (External Entity “Customer”)

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**



**A picture containing text, diagram, handwriting, plan

Description automatically generated**

A white board with writing on it

Description automatically generated with low confidence

## Description of Data Flow Diagram (DFD):

**Process 1.6 “Fills Prescription”**

a) The Pharmacist **(External Entity)** sends the Next Prescription message **(Input Data Flow “1.6 “Next Prescription” message)** to the process **(“P1.6 Fills Prescription”)** and the message **(Output Data Flow “1.6 “Next Prescription” message)** is checked against the data present in Prescription Data Repository **(Data Store “D2 Prescription”).**

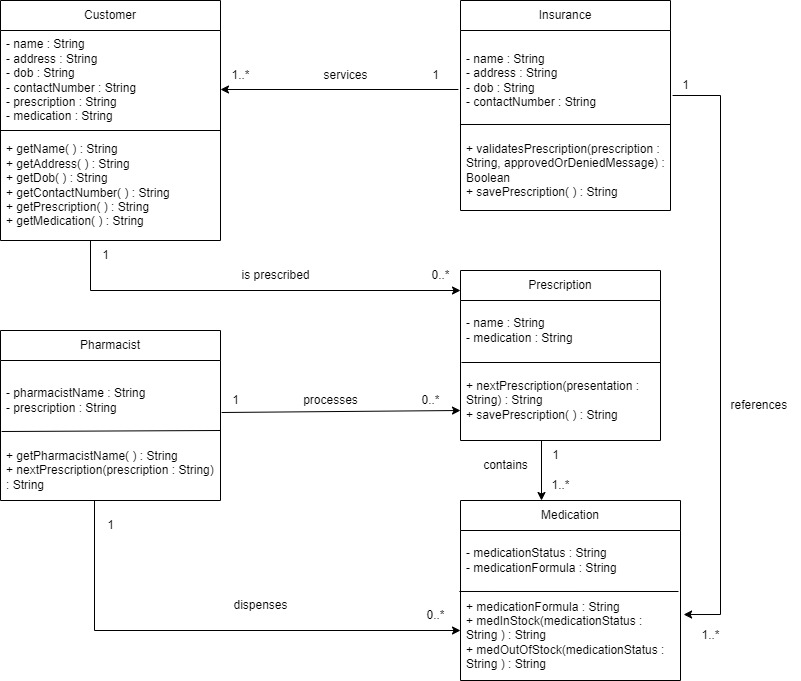
b) The process **(“P1.6 Fills Prescription”)** obtains the Medication and Customer insights **(Input Data Flow “1.7 Medication, Customer”)** from Prescription Data Repository **(Data Store “D2 Prescription”).**

c) The process **(“P1.6 Fills Prescription”)** checks the Medication status **(ODF “1.8 Medication status”)** if it is in stock or not, against the data in Medication Data Repository **(Data Store “D3 Medication”).**

d) If the process **(“P1.6 Fills Prescription”)** obtains the status as in-stock **(IDF “1.9.a In-Stock”)** from Medication **(Data Store “D3 Medication”)** then it displays **(ODF “1.10 Medication In Stock – Will be Dispensed in 8 Hours”)** message to the Customer **(EE).**

e) If the process **(“P1.6 Fills Prescription”)** obtains the status as out-of-stock **(IDF “1.9.b Out-Of-Stock”)** from Medication Data Repository **(Data Store “D3 Medication”)** then it displays **(ODF “1.11 Medication Out Of Stock – Retry in 24 Hours”)** message to the Customer **(EE).**

**(b) Create a Class Diagram for the entities mentioned in this case (Customer, Pharmacist, Prescription, Insurance, Medication)**

****

A white board with writing on it

Description automatically generated with low confidence

## Description of Class Diagram (CD):

**Class Customer:**

**Description:**

**Represents the Customer who interacts with the System to get medications.**

**Traceability:**

**Matches DFD EE “Customer”**

**Multiplicity:**

**An instance of class “Customer” holds as [1]-to-[0-to-many] instances of class “Prescription” since there exists only 1 customer with the given name, address, dob, and contact number. And for each Customer, there can be 0 or more prescriptions.**

**Class Pharmacist:**

**Description:**

**Represents a Pharmacist who interacts with the System to obtain the medications.**

**Traceability:**

**Matches DFD EE “Pharmacist”**

**Multiplicity:**

**An instance of class “Pharmacist” holds as [1]-to-[0-to-many] instances of class “Prescription” since there can be more 0 or more than 0 prescriptions that the Pharmacist needs.**

**An instance of class “Pharmacist” holds as [1]-to-[0-to-many] instances of class “Medications” since a particular Pharmacist can consist of either 0 or 1 or 2 or so on a number of prescriptions.**

**Class Insurance:**

**Description:**

**Represents the Data Store that contains Information about the Customer and Prescription details.**

**Traceability:**

**Matches DFD DS “D1 Insurance”**

**Multiplicity:**

**An instance of class “Insurance” holds as [1]-to-[1-to-many] instances of class “Customer” since it can provide service to 1 or more than 1 Customer.**

**An instance of class “Insurance” holds as [1]-to-[1-to-many] instances of class “Medication” since there can be 1 or more than 1 Medication.**

**Class Prescription:**

**Description:**

**Represents the Prescription obtained from the Customer or Pharmacist.**

**Traceability:**

**Matches DFD DS “D2 Prescription”**

**Multiplicity:**

**An instance of class “Prescription” holds as [1]-to-[1-to-many] instances of class “Medication” since there can be 1 or more than 1 medication in a Prescription.**

**There must exist at least 1 Medication.**

**Class Medication:**

**Description:**

**Represents the Medications that are present in the Prescription provided by either Customer or Pharmacist.**

**Traceability:**

**Matches DFD DS “D3 Medication”**

**Multiplicity:**

**An instance of class “Medication” holds as [1-to-many]-to- [1] instances of class “Insurance” since there can exist only 1 Insurance for the given Medication.**

**An instance of class “Medication” holds as [1-to-many]-to- [1] instances of class “Prescription” since medication can come from a particular Prescription only.**

**An instance of class “Medication” holds as [0-to-many]-to- [1] instances of class “Pharmacist” since there can be only 1 Pharmacist requesting that particular Medication. However, the number of Medications being requested can be more.**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**1. Process 1.1 “Validates Prescription”**

a) The “Customer” (EE “Customer”) provides his “name, address, date-of-birth (DOB) and prescription” (IDF “1.1 name, address, date-of-birth (DOB) and prescription”) which is structured by the process (P1.1 “Validates Prescription) as CustInfo. The “CustInfo and Prescription” (IDF “1.2 CustInfo and Prescription”) is checked against the DS “D1 Insurance Data Repository”.

b) After querying DS “D1 Insurance Data Repository” the process P1.1 obtains status message (ODF “1.3.a Approved”) or (ODF “1.3.b Denied) to process P1.1.

c) The process P1.1 checks against the status message. If the status message is (ODF ““1.3.b Denied”) then the process P1.1 sends (ODF “1.4 PrescDen”) to the EE “Customer”.

d) The process P1.1 checks against the status message. If the status message is (ODF ““1.3.a Approved”) then the process P1.1 sends (ODF “1.5 PrescApp”) to the EE “Customer” and saves (IDF 1.5.a “saves the Prescription”) via P1.1 into the DS “D2 Prescription Data Repository”.

**2. Process 1.3 “Files Prescription”**

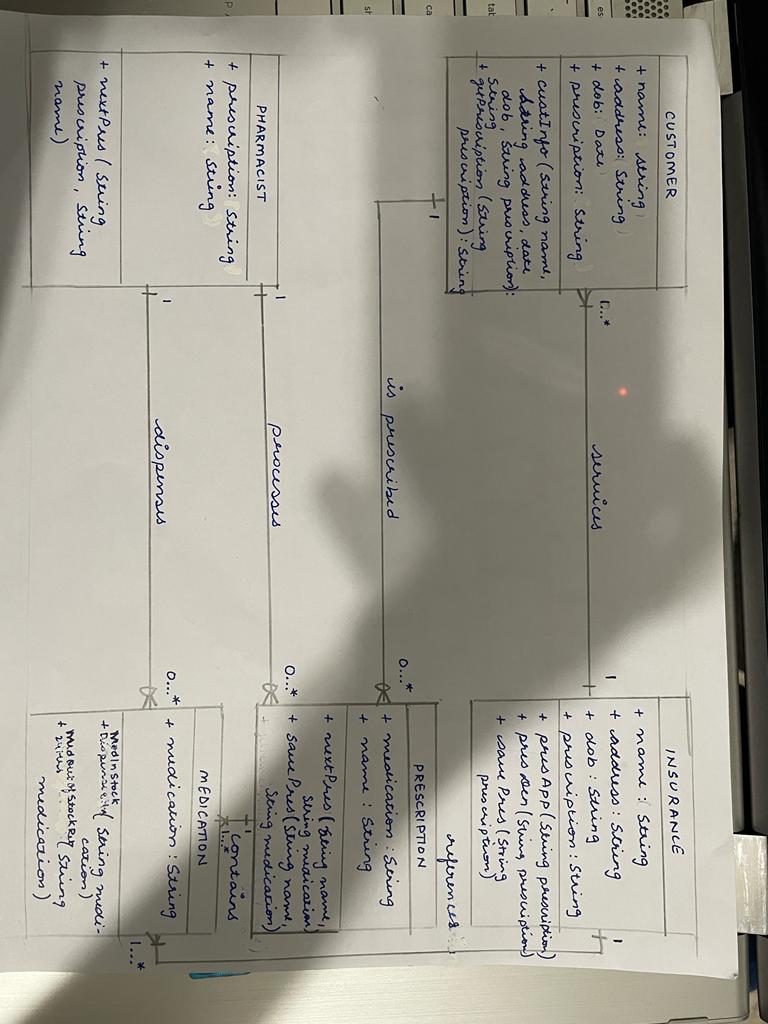
a) The “Pharmacist” (EE “Pharmacist”) messages the process (P1.3 “Files Prescription”) to process the next Prescription (IDF “2.1 NextPresc”) to the DS D2. The process P1.3 obtains Medication and Customer (ODF “2.2 Medication and Customer”) from the DS D2.

b) The process P1.3 checks if the Medication is in stock (INF “2.3 Medication”) from the DS “D3 Medication Data Repository”.

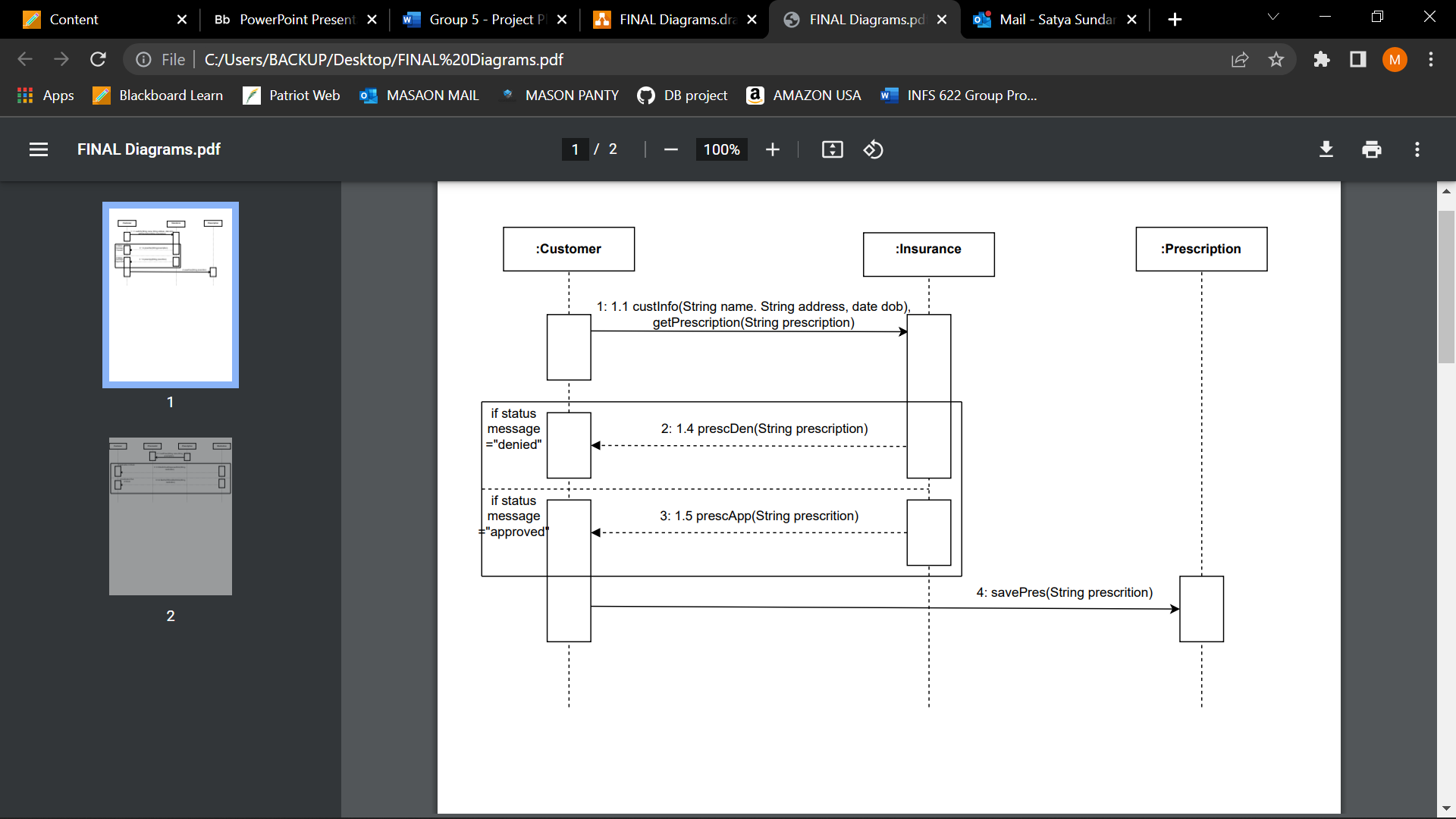
c) If the process P1.3 obtains status In-Stock (ODF “2.4.a InStock”) from the DS D3 then the process sends message to the EE “Customer” (ODF “2.5 MedInStockDispensed8Hrs”).

d) If the process P1.3 obtains status In-Stock (ODF “2.4.b OutOfStock”) from the DS D3 then the process sends message to the EE “Customer” (ODF “2.5 MedOutOfStockRet24Hrs”).

**(b)**



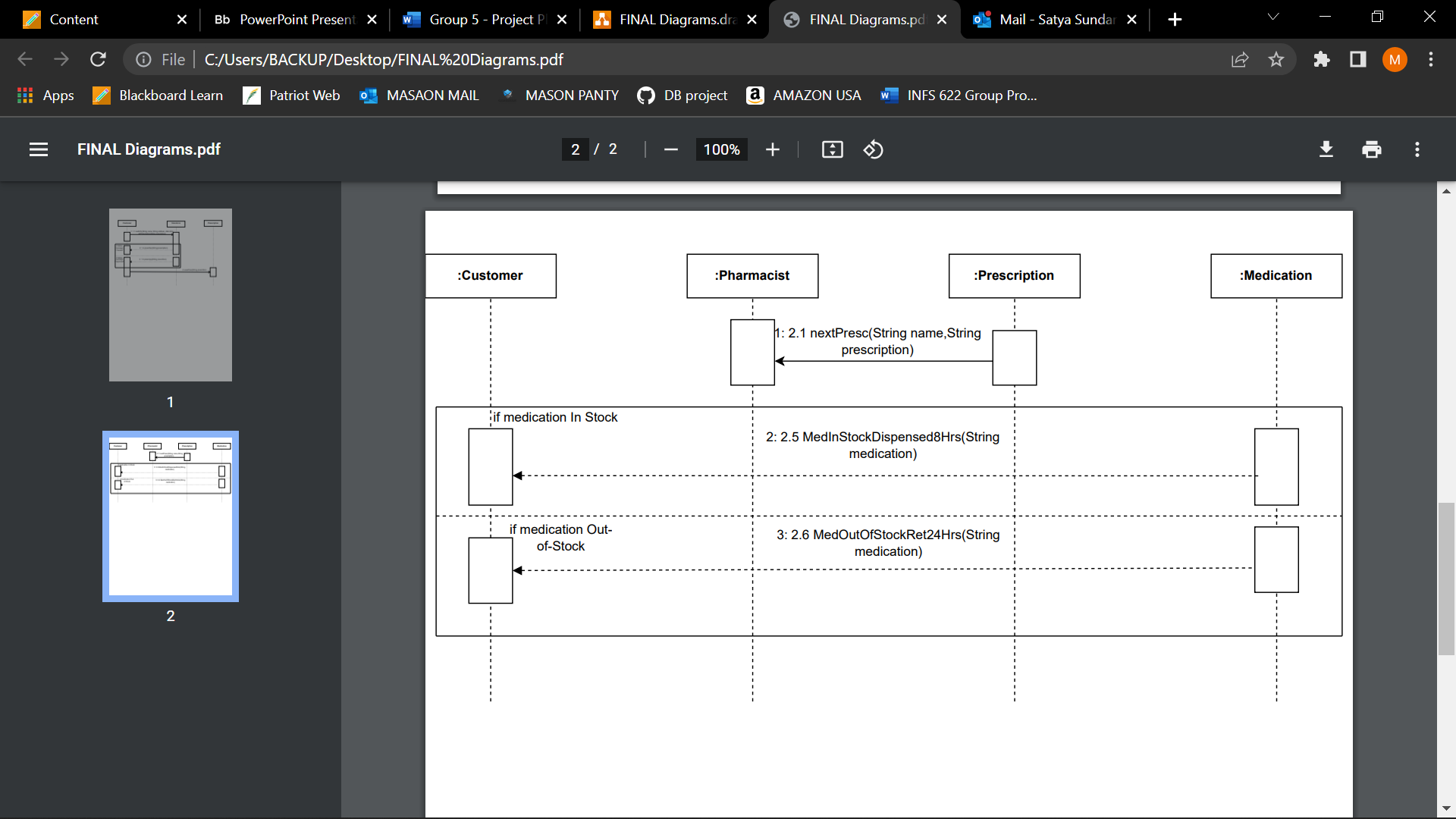
**(c)**



1. Object “:Customer” invokes method Customer :: custInfo(String name. String address, date dob), getPrescription(String prescription) and checks against the Object “:Insurance”.**(Matches DFD P1.1, Customer:: getPrescription)**

2. Customer:getPrescription(String prescription) is checked against the Object: “Insurance” and obtains a status message. If status message is Denied then the Object: “Insurance” returns Insurance:: presDen(String prescription) to Object: “Customer”. If status message is Approved then the Object: “Insurance” returns Insurance:: presApp(String prescription) Object: “Customer” and saves the prescription to Object: “Prescription”.

**(d)**



1. Object “:Prescription” invokes method Prescription :: nextPresc(String name, String prescription) and sends it to Object “:Pharmacist”.

2. Prescription :: nextPresc(String name, String prescription) is checked against the Object: “Medication” and obtains a status message. If status message is InStock then the Object: “Medication” returns Medication:: MedInStockDispensed8Hrs(String medication) to Object: “Customer”. If status message is Out-Of-Stock then the Object: “Medication” returns Medication:: MedOutOfStockRet24Hrs(String medication) to Object: “Customer”.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

The pharmacy deals with: **Customers**, **Pharmacists**, **Prescriptions**, **Insurances**, and **Medications**.

Both **Customers** and **Pharmacists** are a kind of **User**.  
Each **Customer** is prescribed [one]‐to‐[zero‐to‐many] **Prescriptions**.  
Each **Prescription** contains [one]‐to‐[one‐to‐many] **Medications**.

Each **Insurance** services [one]‐to‐[one‐to‐many] **Customers**.

Each **Insurance** references [one]‐to‐[one‐to‐ many] **Medications**.  
Each **Pharmacist** processes [one]‐to‐[zero‐to‐many] **Prescriptions** and dispenses [one]‐to‐[zero‐to‐many] **Medications**

**The first sequence of events** that the system performs is a Process that can be called **Validates Prescription**. Here are its steps:

* 1. The **Customer** provides their **name**, **address**, **date‐of‐birth (DOB)** and their **Prescription** to the Process.

**1.2** The Process groups the **Customer’s name, address, and DOB** into a structure named **Customer Information (CustInfo)**, and then the Process checks against the **Insurance Data Repository** that the **Customer Information** (**CustInfo**) has a valid insurance that covers all the medications in that **Prescription**. In order to do that the Process uses both the **Customer Information** (**CustInfo**) and the **Prescription** to query the **Insurance Data Repository.**

**1.3** After querying the **Insurance Data Repository** the Process obtains a (1.3.a) **Approved** or (1.3.b) **Denied** status message for that specific **Customer** and that specific **Prescription**, from the **Insurance Data Repository** (for simplicity let’s assume that if one medication in the prescription is not covered by the insurance then the return value from the Insurance Data Repository will be “**Denied**”, and only if all medications in that prescription are covered then the return value from the Insurance Data Repository will be “**Approved**”).

**1.4** If the status message obtained by the Process from the **Insurance Data Repository** is “**Denied**”, then the Process sends a “**Prescription Denied**” (**PrescDen**) message to the **Customer** (and this is the end of the entire case).

**1.5** Otherwise, if the status obtained by the Process from the **Insurance Data Repository** is “**Approved**”, then the Process sends a “**Prescription Approved**” (**PrescApp**) message to the **Customer**, and the process also **saves the Prescription** (1.5.a) into the **Prescriptions Data Repository**.

**The second sequence of events** that the system performs is a Process that can be called **Fills Prescription**. Here are its steps:

**2.1** The **Pharmacist** sends a message to the Process to get the **Next Prescription** (**NextPresc**) from the **Prescriptions Data Repository** (each **Prescription** is queued and waiting to be pulled by the **Pharmacist**

from the **Prescriptions Data Repository**). The Process forwards this message to the **Prescriptions Data Repository.**

**2.2** The Process obtains from the **Prescriptions Data Repository** a **Medication** and a **Customer** (the **Medication** and the **Customer** are part of the **Prescription** that is currently being handled by the **Pharmacist**).

**2.3** The Process checks if the **Medication** is in stock by querying the **Medication Data Repository**.

**2.4** The Process obtains from the **Medication Data Repository** whether that **Medication** is: (2.4.a) **In‐ Stock (InStock**) or (2.4.b) **Out‐Of‐Stock (OutOfStock**).

**2.5** If the status obtained by the Process from the **Medication Data Repository** is “**In‐Stock**”, then the Process sends to the **Customer** a message saying “**Medication In Stock – Will be Dispensed in 8 Hours**” (**MedInStockDispensed8Hrs**), and this is the end of the case.

**2.6** Otherwise, if the status obtained by the process from the **Medication Data Repository** is “**Out‐Of‐ Stock**”, then the Process sends to the **Customer** a message saying “**Medication Out Of Stock – Retry in 24 Hours**” (**MedOutOfStockRet24Hrs**), and this is the end of the case.