2/11/23

ASSIGNMENT 2

[PART-1 : System Modeling]

Q.1 Develop system modeling of Railway reservation system:

a) list out at least twenty functional requirements.

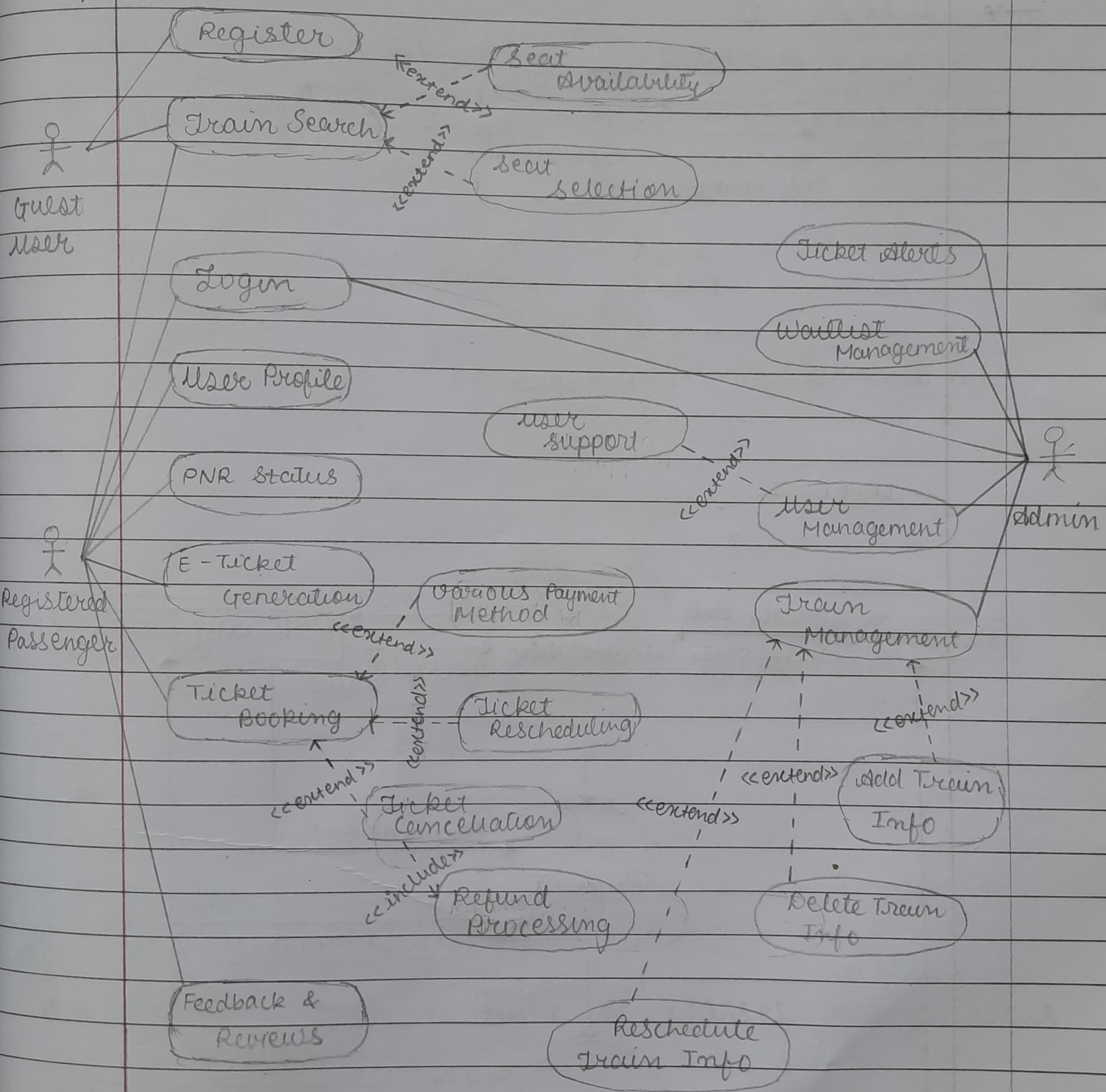
(1) Register
(2) login
(3) User Profile
(4) Train Search
(5) Seat Availability
(6) Seat Selection
(7) Ticket Booking
(8) Ticket Cancellation
(9) Ticket Rescheduling.
(10) PNR Status
(11) Ticket Alerts
(12) Feedback and Reviews
(13) User Support
(14) Refund Processing.
(15) Various Payment Methods.
(16) E-Ticket Generation
(17) Train Management
(18) User Management
(19) Waitlist Management
(20) Seat Allocation.

b) Static Modeling :

(i) Design neat and clear use case diagram

showing required dependencies and covering all the functional requirements.

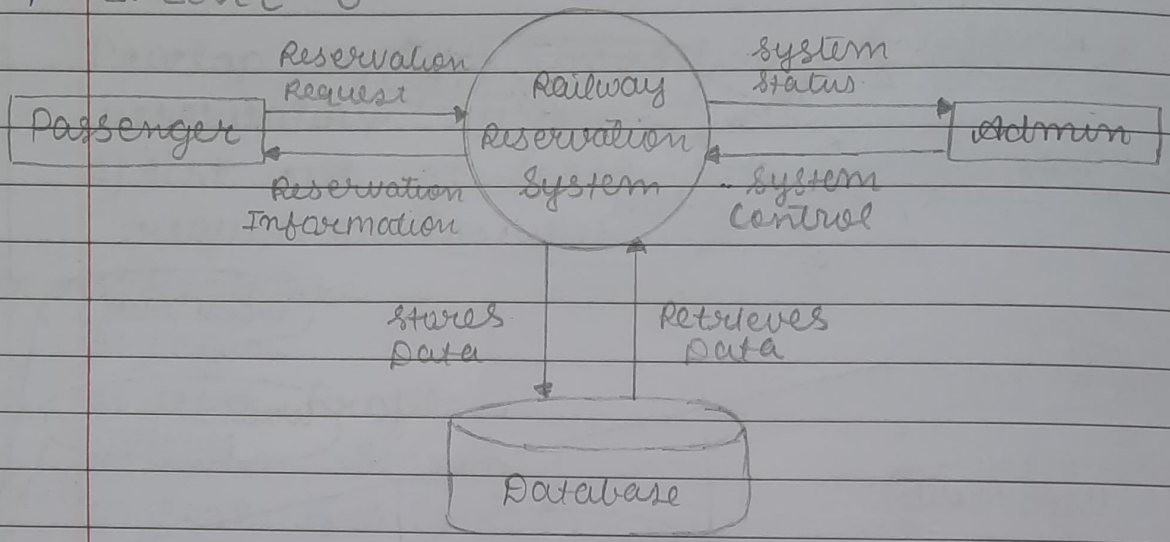
—4



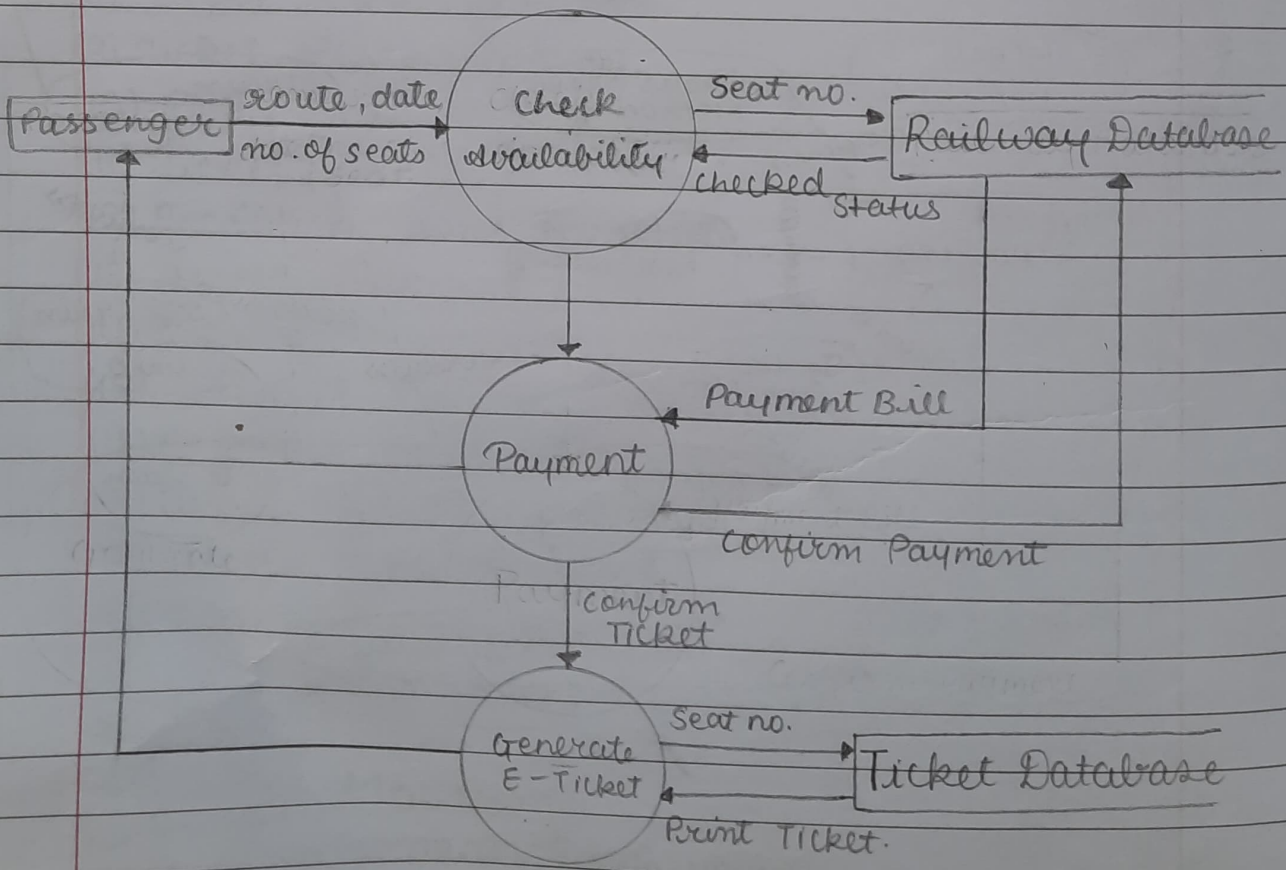
Railway Reservation System.

(ii) Design data flow diagram up to level-2. Ensure to cover all the use cases.

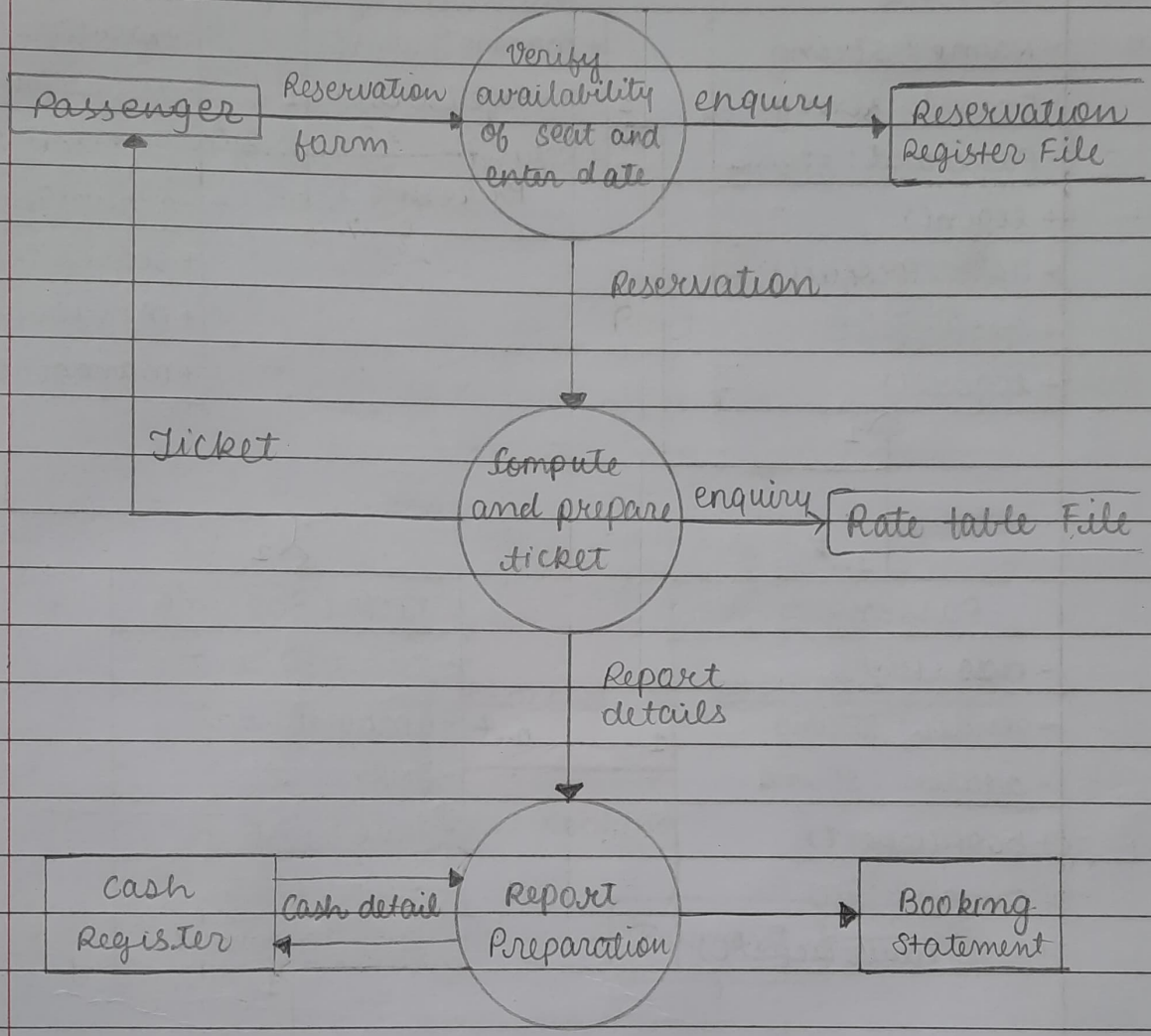
—4 1. LEVEL - 0



2. LEVEL - 1

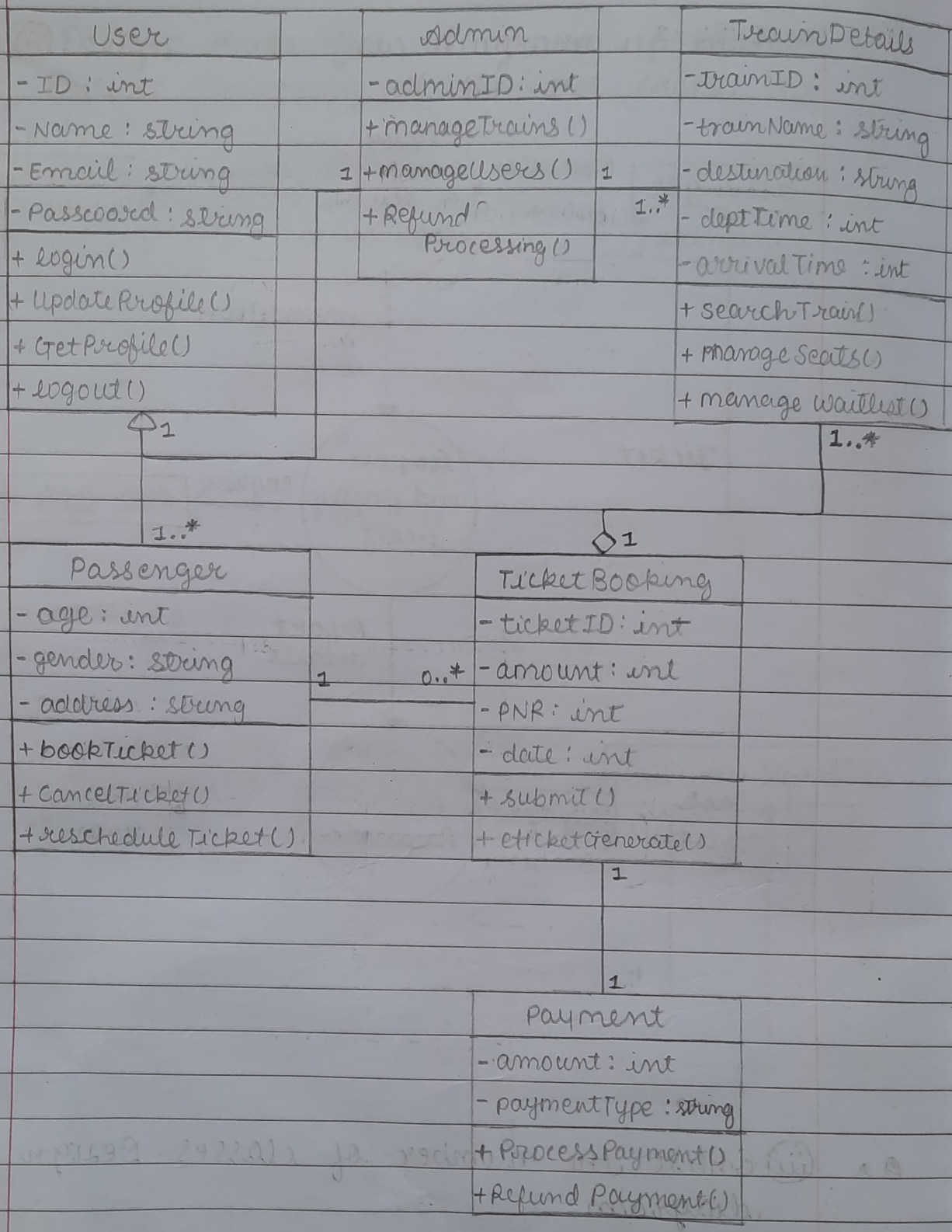


3. LEVEL - 2



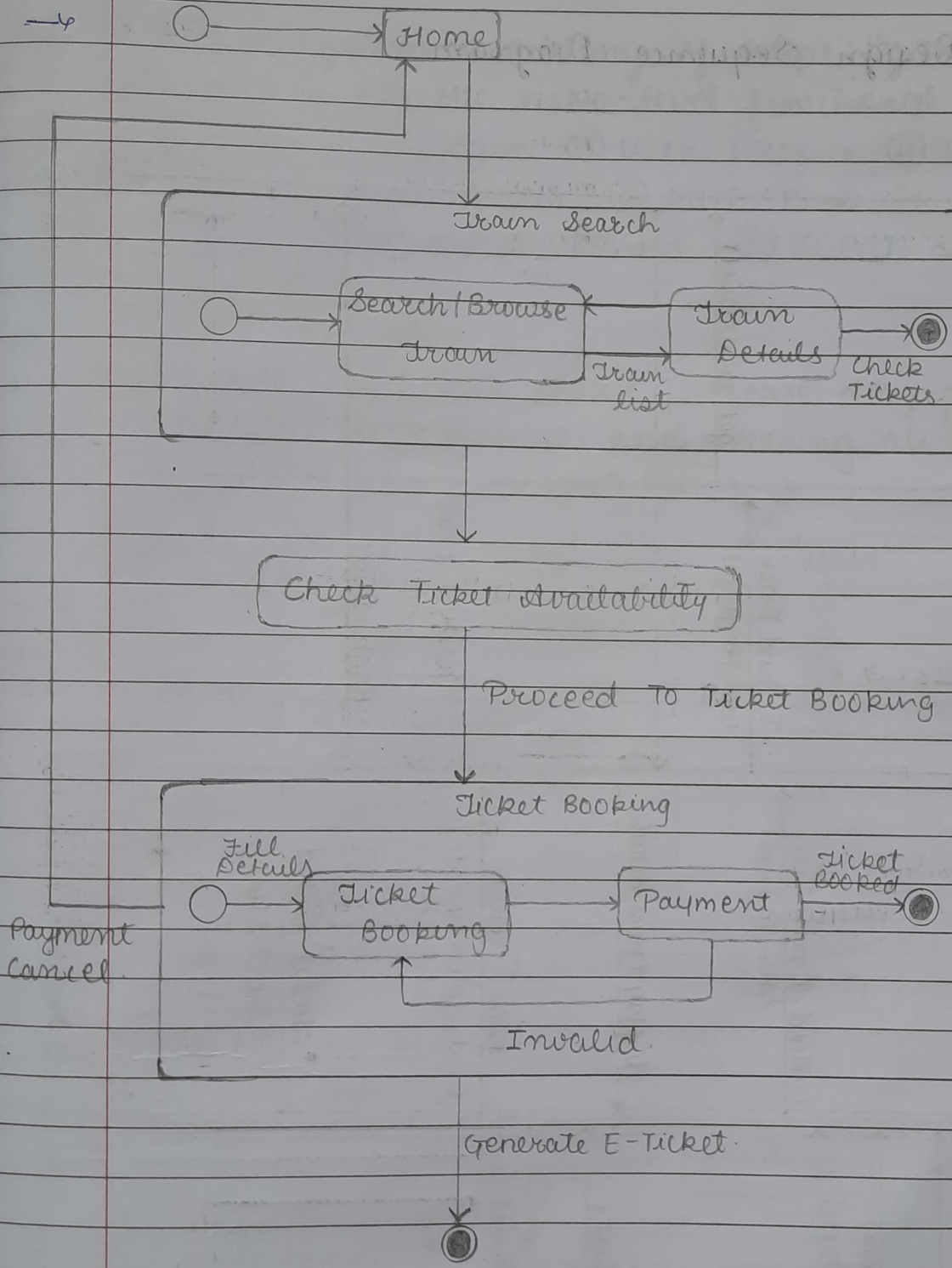
Q4 (iii) Enlist the number of classes. Design class diagram

— 6

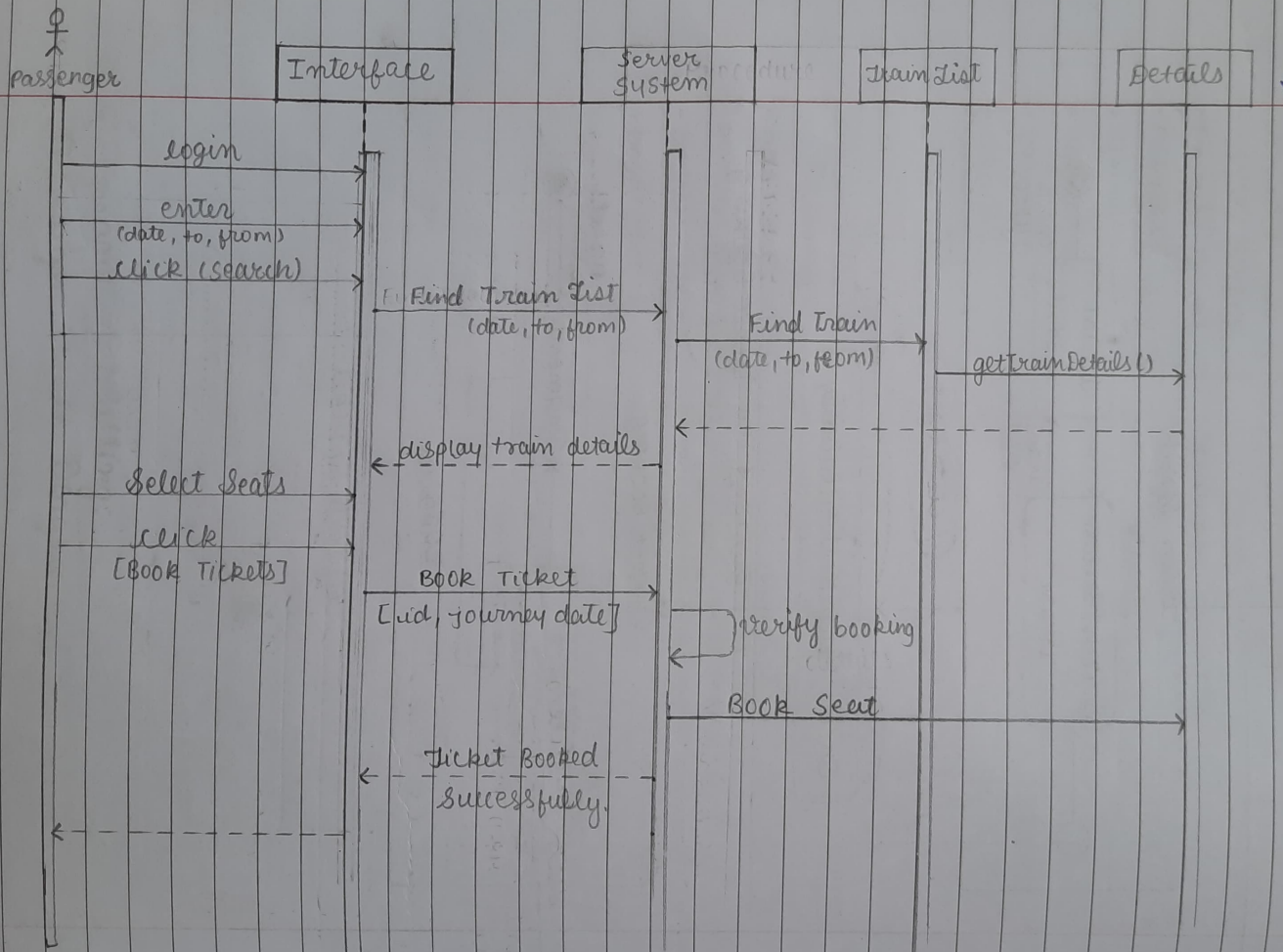


c) Dynamic Modelling:

① Design State Diagram


Home
Train Search
Search / Browse Train
Train Details
Train list
Check Tickets
Check Ticket Availability
Proceed To Ticket Booking
Ticket Booking
Fill Details
Ticket Booking
Payment
Ticket Booked
Payment Cancel
Invalid
Generate E-Ticket

(ii) Design Sequence Diagram



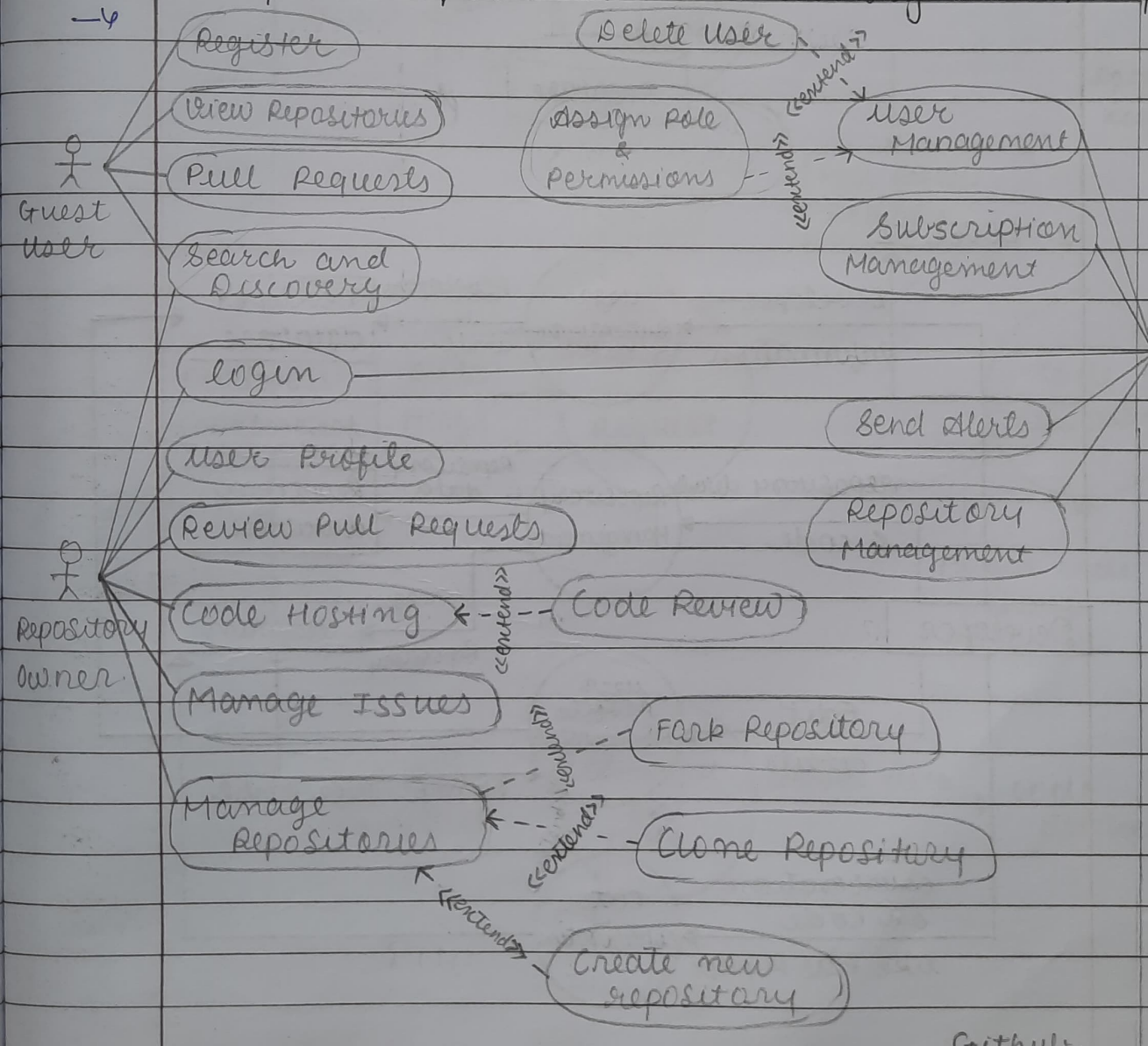
Q.2 Develop system modeling of https://github.com/:

a) List out all the user-level functional requirements

—↓ ① Register ② login ③ User Profile ④ Code HOSting ⑤ Manage Repositories ⑥ Pull Requests ⑦ Manage Projects ⑧ Manage Issues ⑨ Search & Discovery.

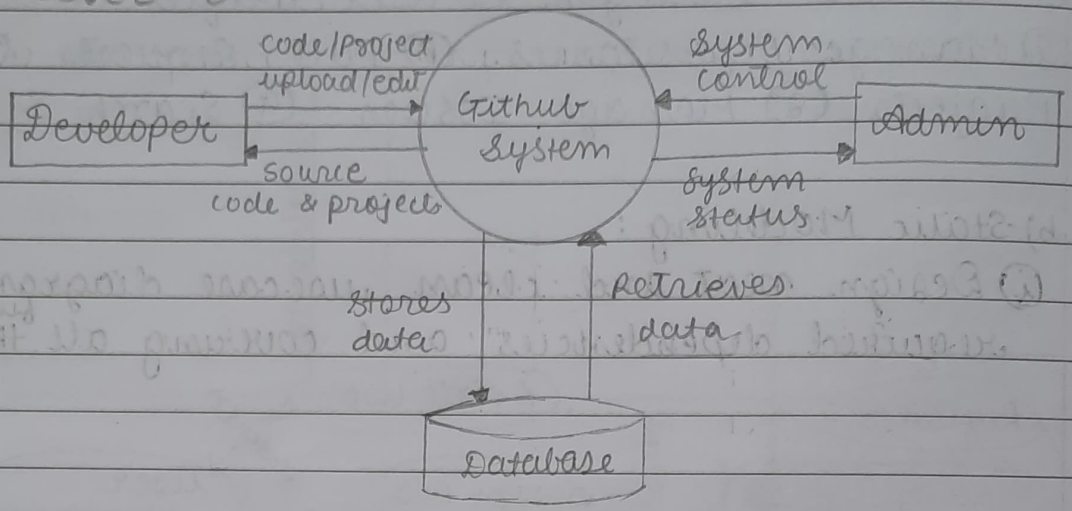
b) Static Modeling:

—↓ ⓘ Design neat and clean usecase diagram showing required dependencies and covering all the functional requirement

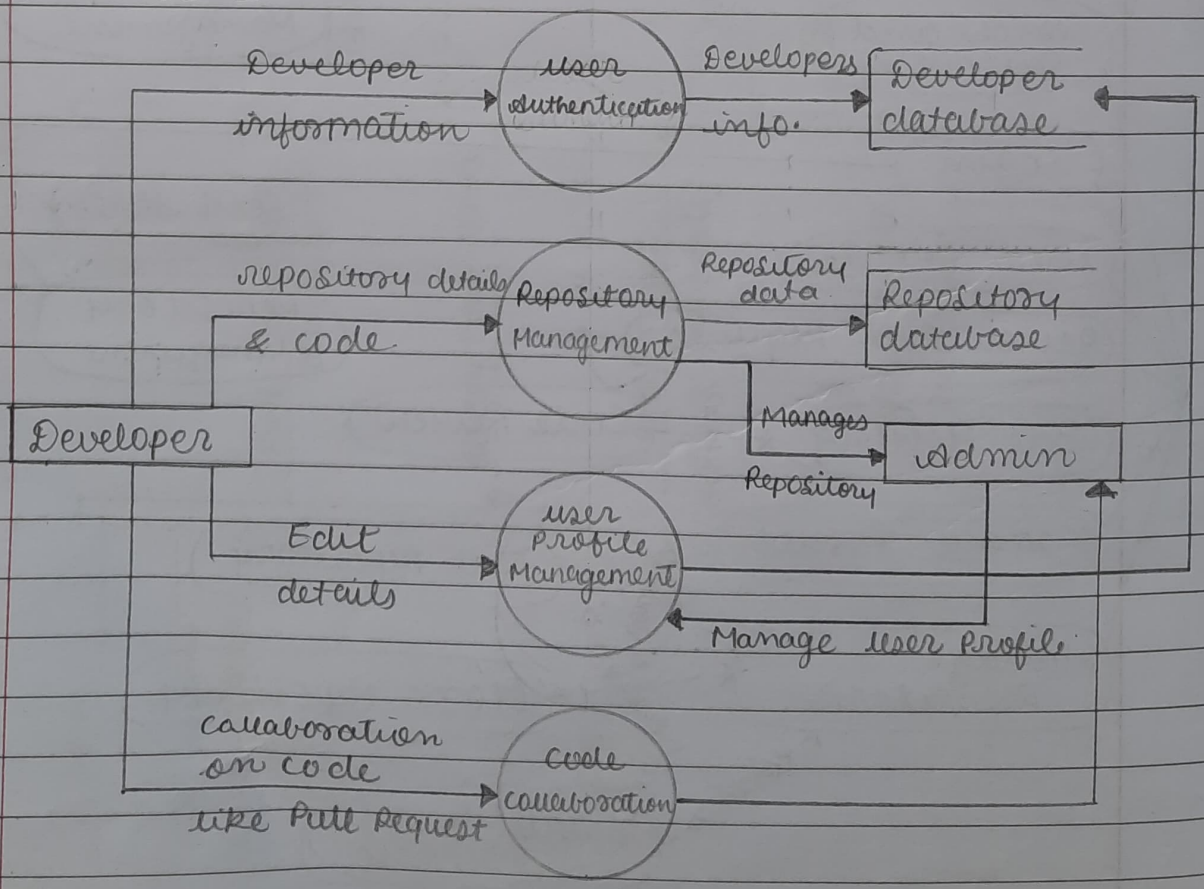


(ii) Design data flow diagram up to level - 3. Ensure to cover all the use cases.

1. LEVEL - 0

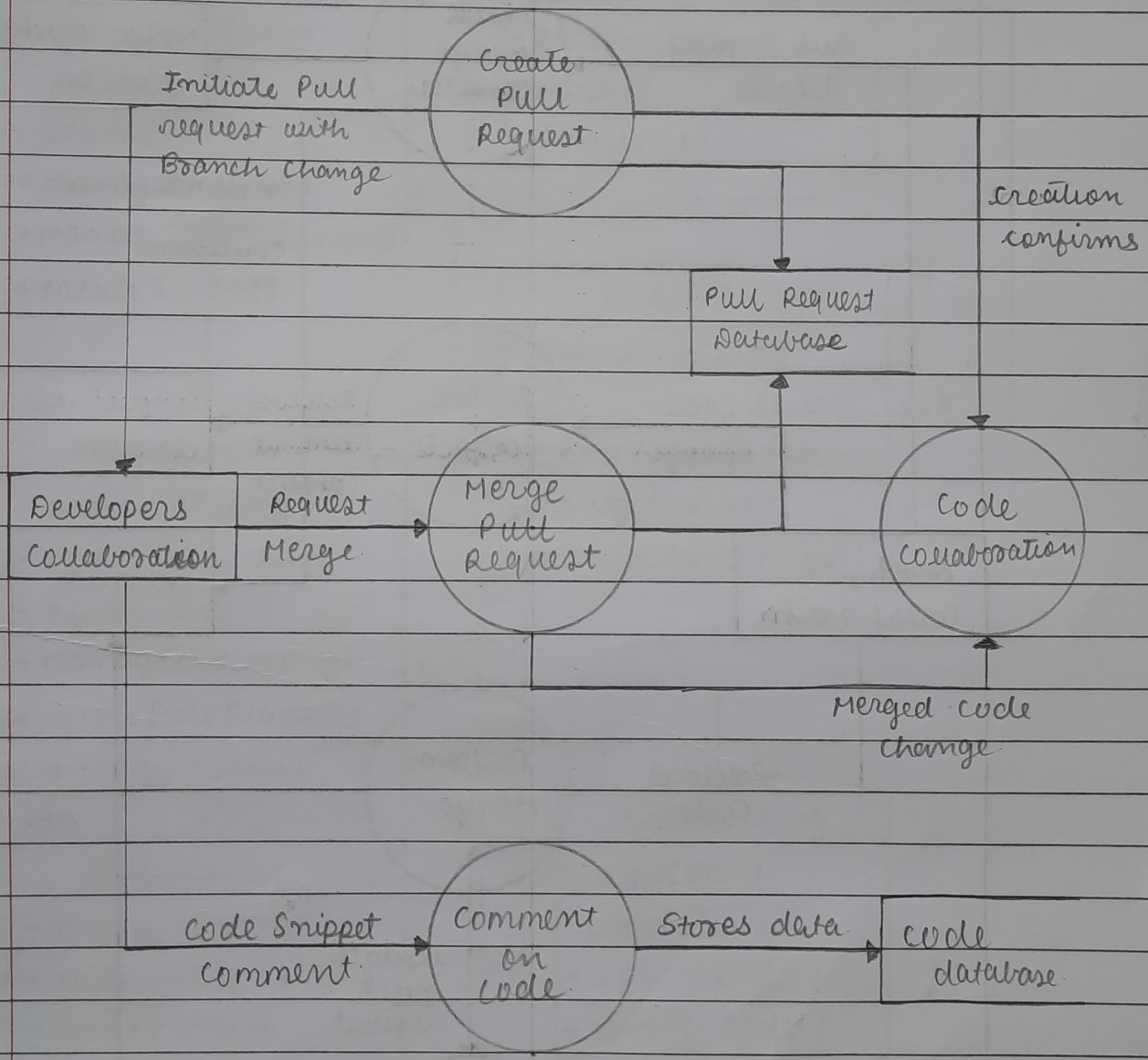


2. LEVEL - 1



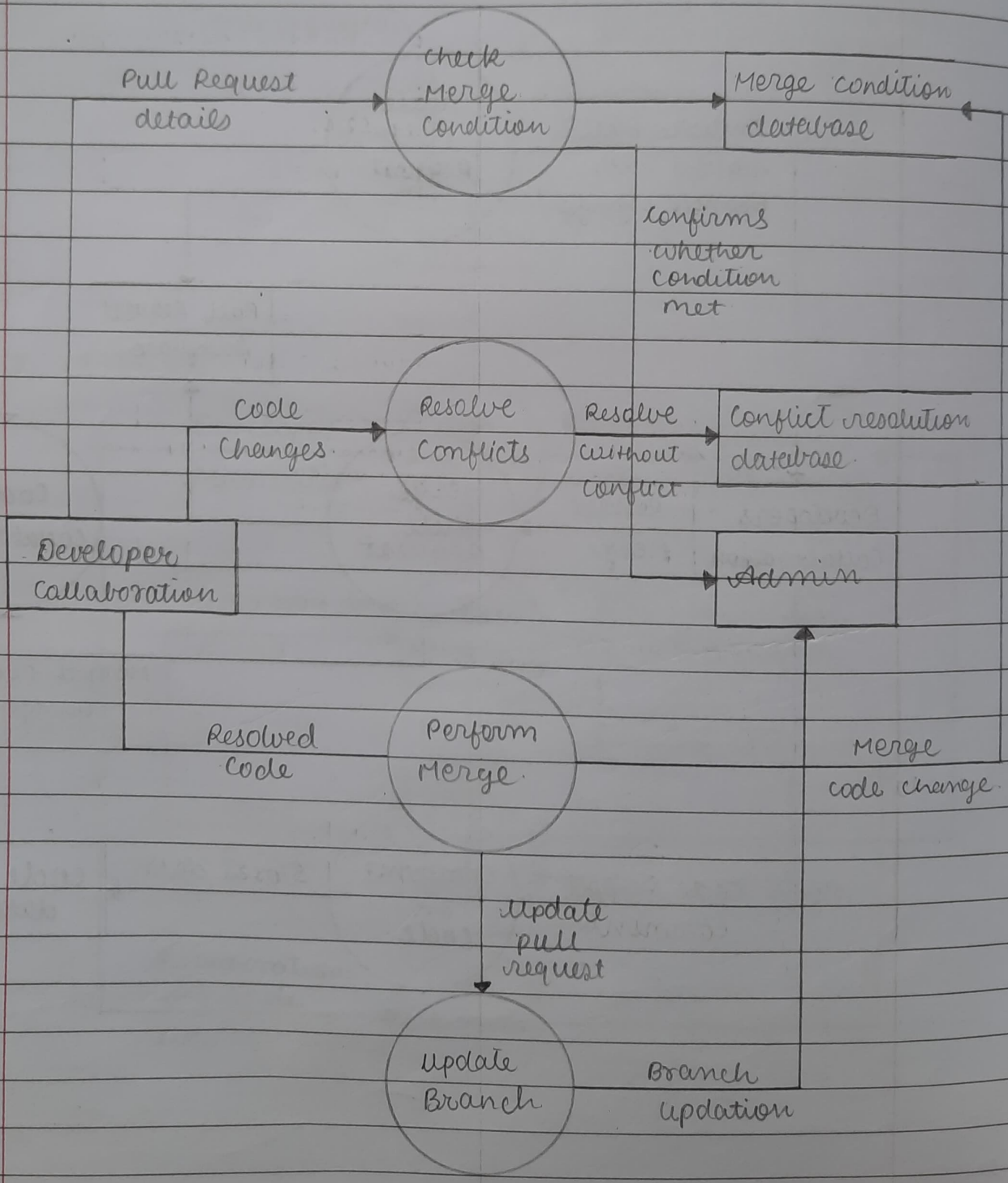
3. LEVEL - 2

For Level - 2 DFD involves breaking down one of the process from Level - 1 DFD, namely "Code Collaboration".

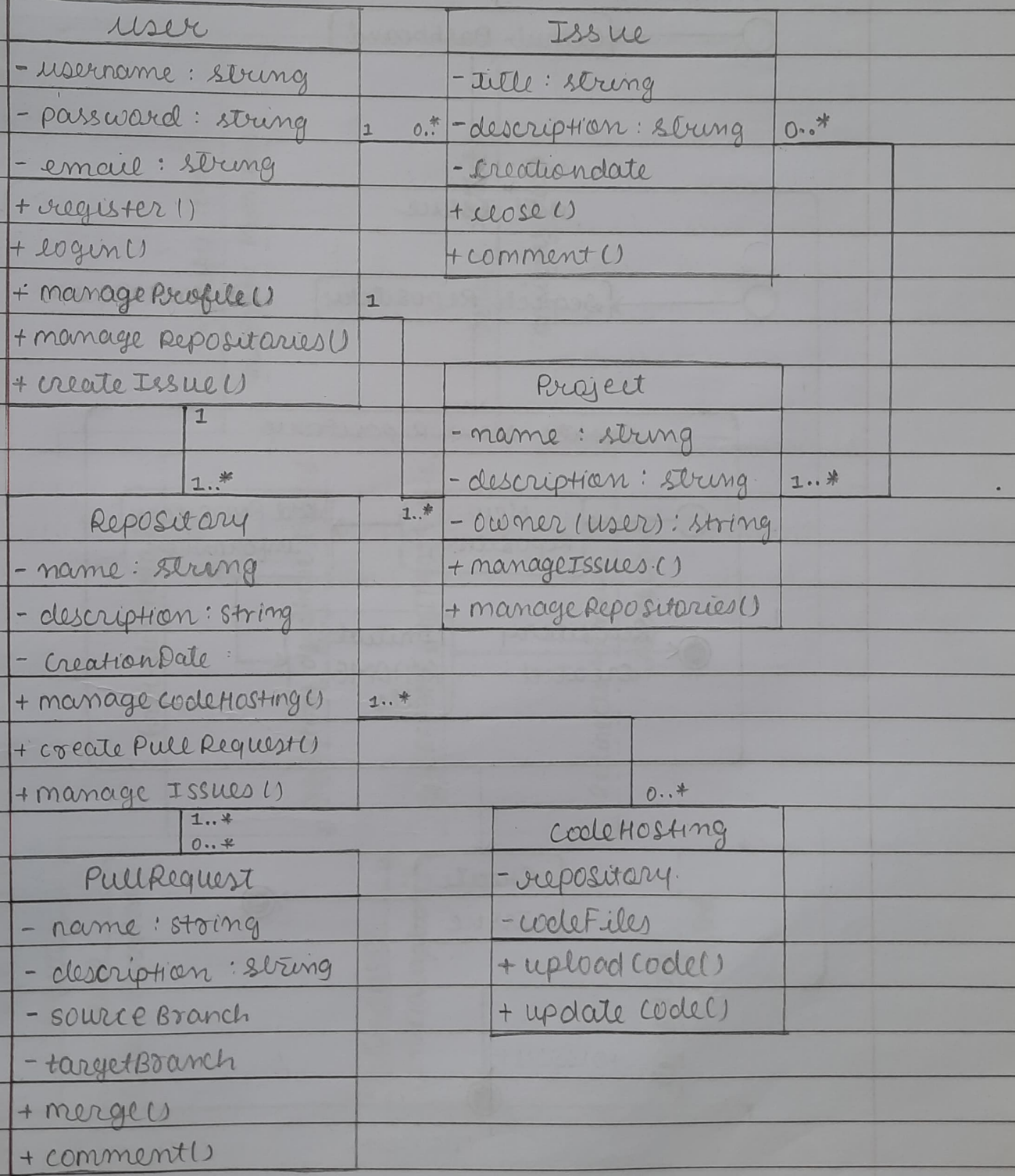


4. LEVEL-3.

For Level-3 DFD involves further detailing the subprocess "Merge Pull Request", from the level-2.

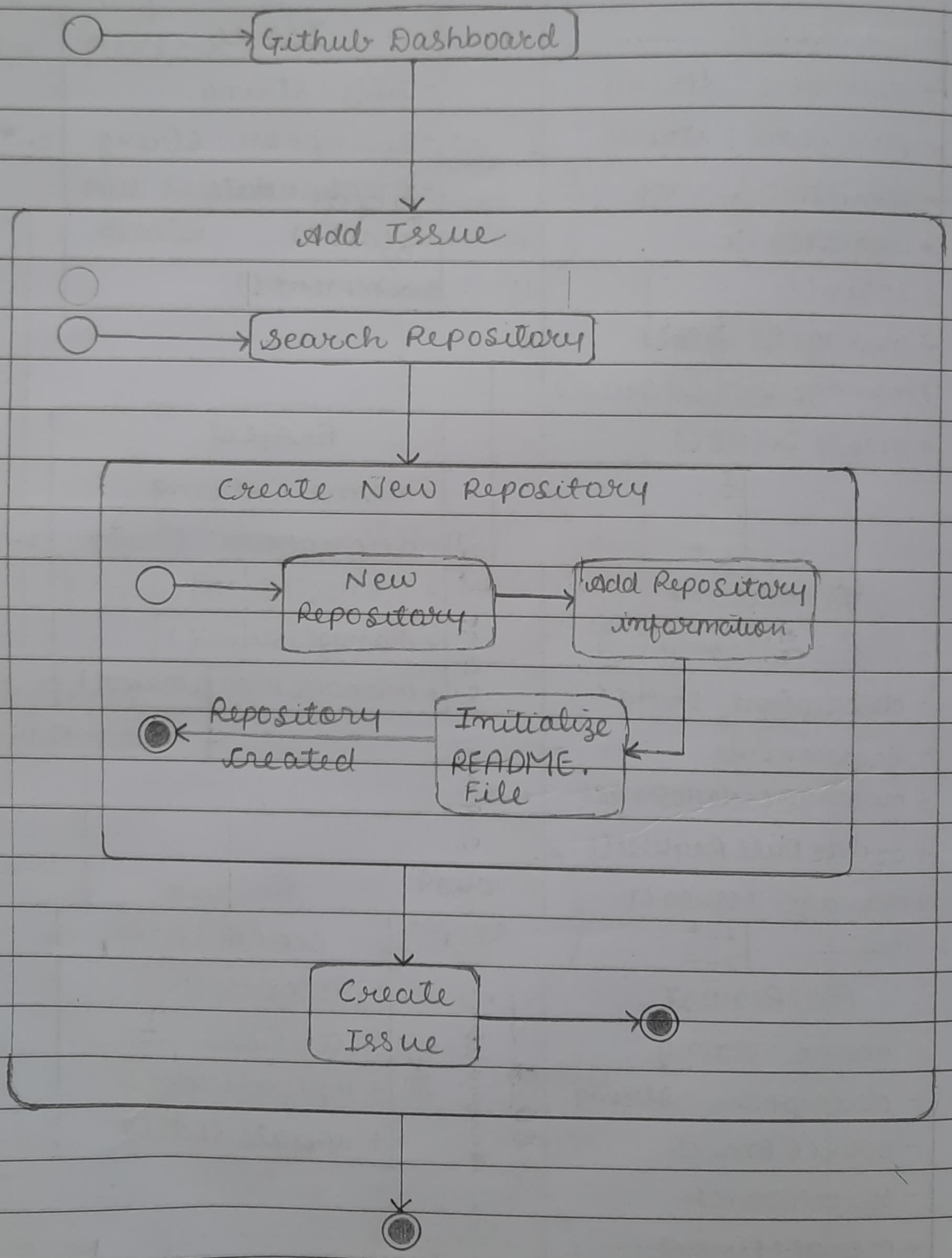


(iii) Enlist the number of classes. Design class diagram.

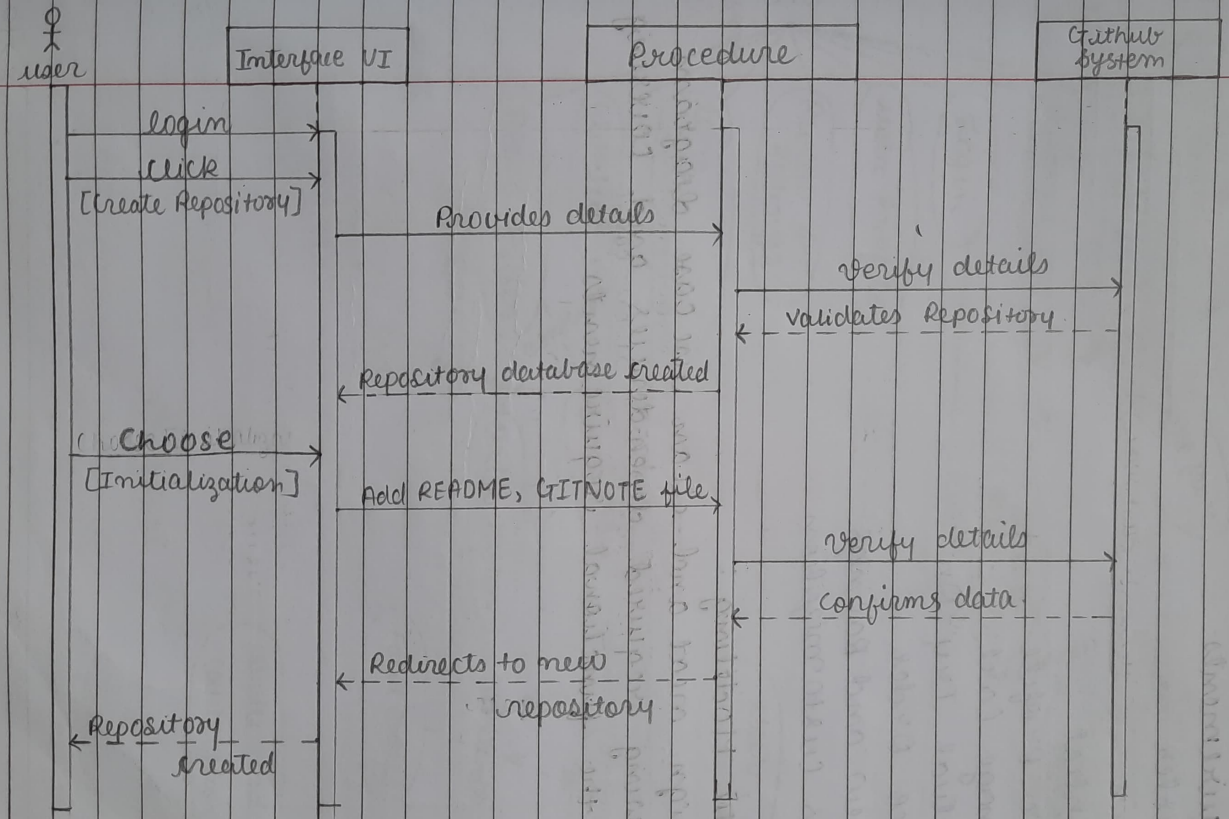


c) Dynamic Modelling

① Design State Diagram



(ii) Design Sequence Diagram



Q3 Development system modeling of https://www.lenskart.com/:

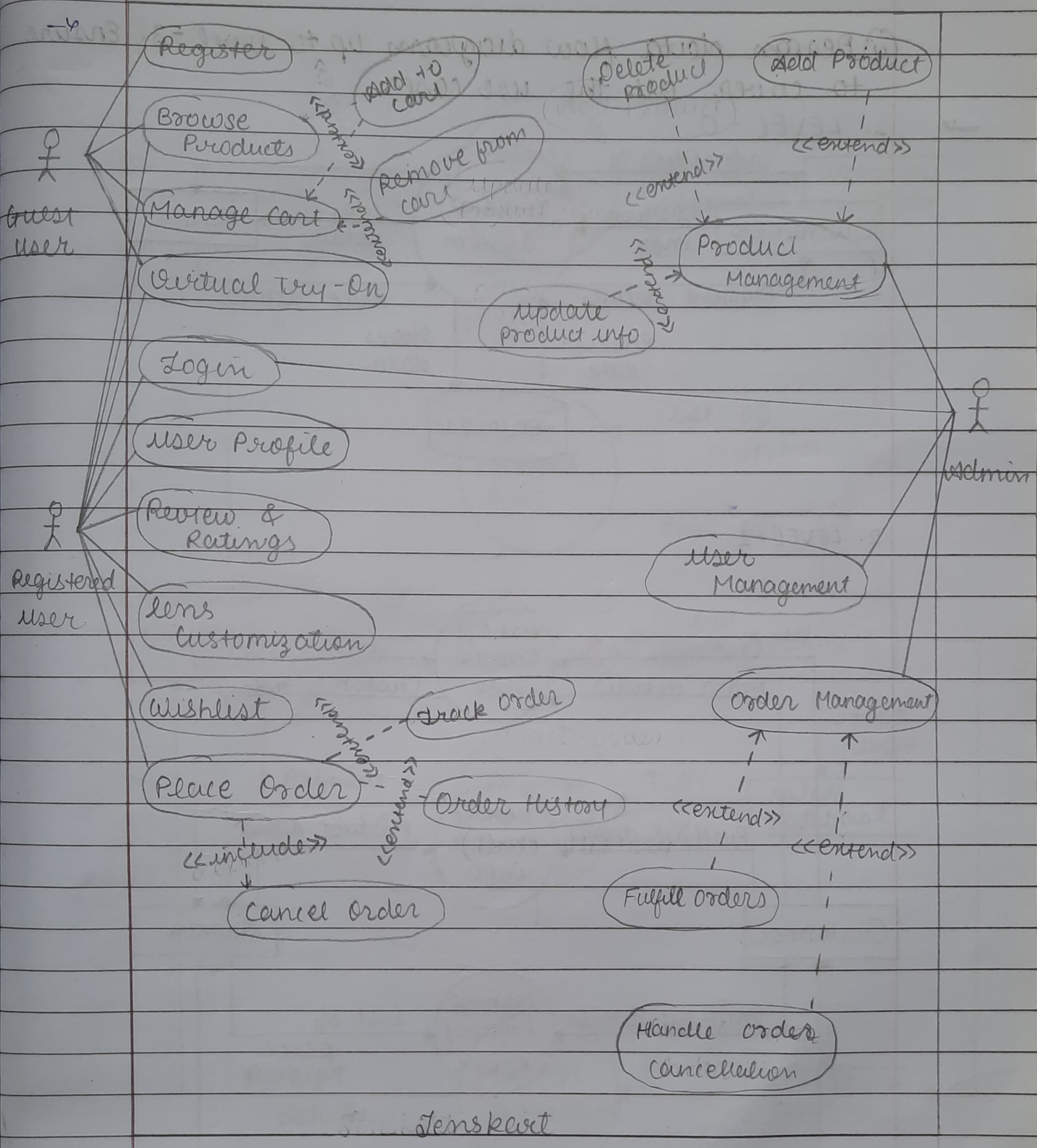
a) list out all the user-level functional requirements.

—> (1) Register
(2) Login
(3) Wishlist
(4) User Profile
(5) Manage Cart
(6) Virtual Try-On
(7) Place Order
(8) Review and Ratings
(9) lens customization

b) Static Modeling:

① Design neat and clean use case diagram showing required dependencies and covering all the functional requirements

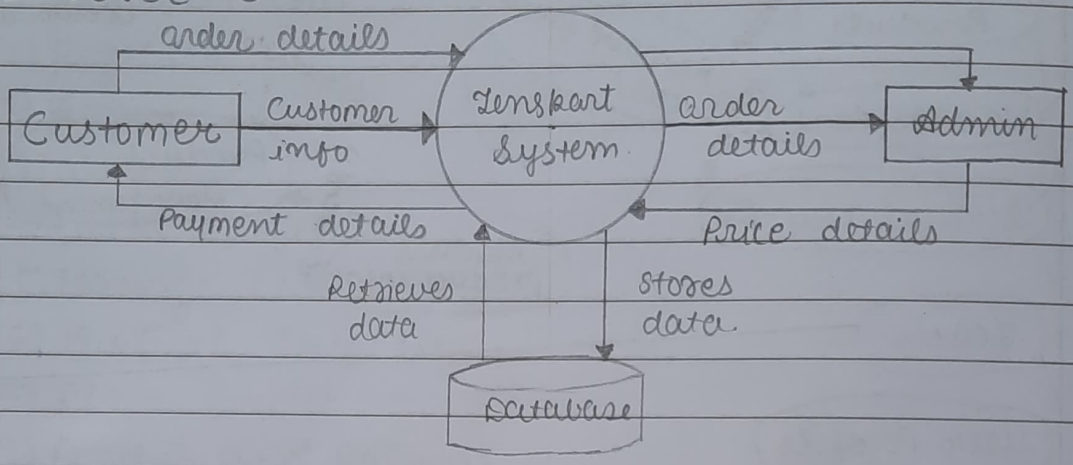
+++


Register
Add to cart
Delete product
Add Product
Browse Products
<<extend>>
Remove from cart
<<extend>>
<<extend>>
Guest User
Manage Cart
<<extend>>
Product Management
Virtual Try-On
<<extend>>
Update product info
Login
User Profile
Admin
Review & Ratings
User Management
Registered User
Lens Customization
Wishlist
<<extend>>
Track Order
Order Management
Place Order
<<extend>>
Order History
<<extend>>
<<extend>>
<<include>>
Cancel Order
Fulfill orders
Handle order cancellation

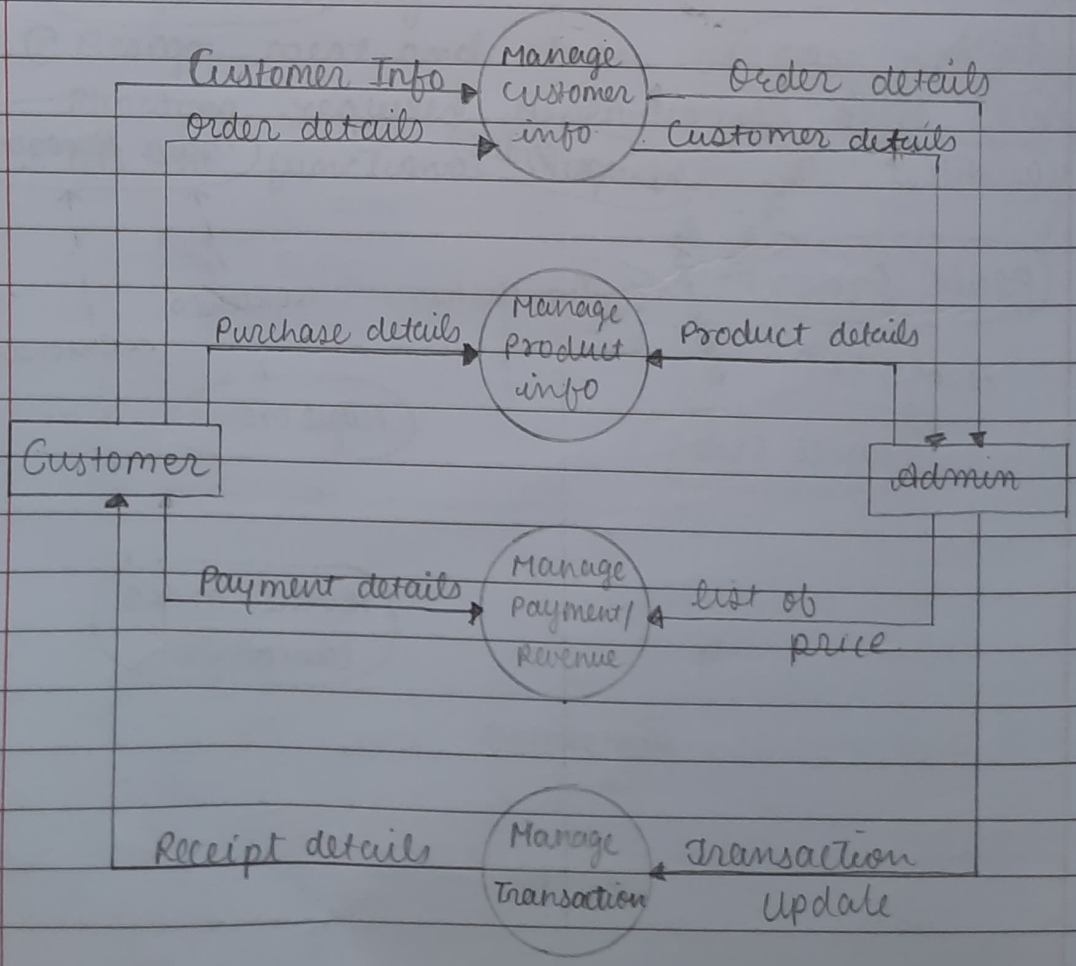
Tenskart

(ii) Design data flow diagram up to level - 3. Ensure to cover all the use cases.

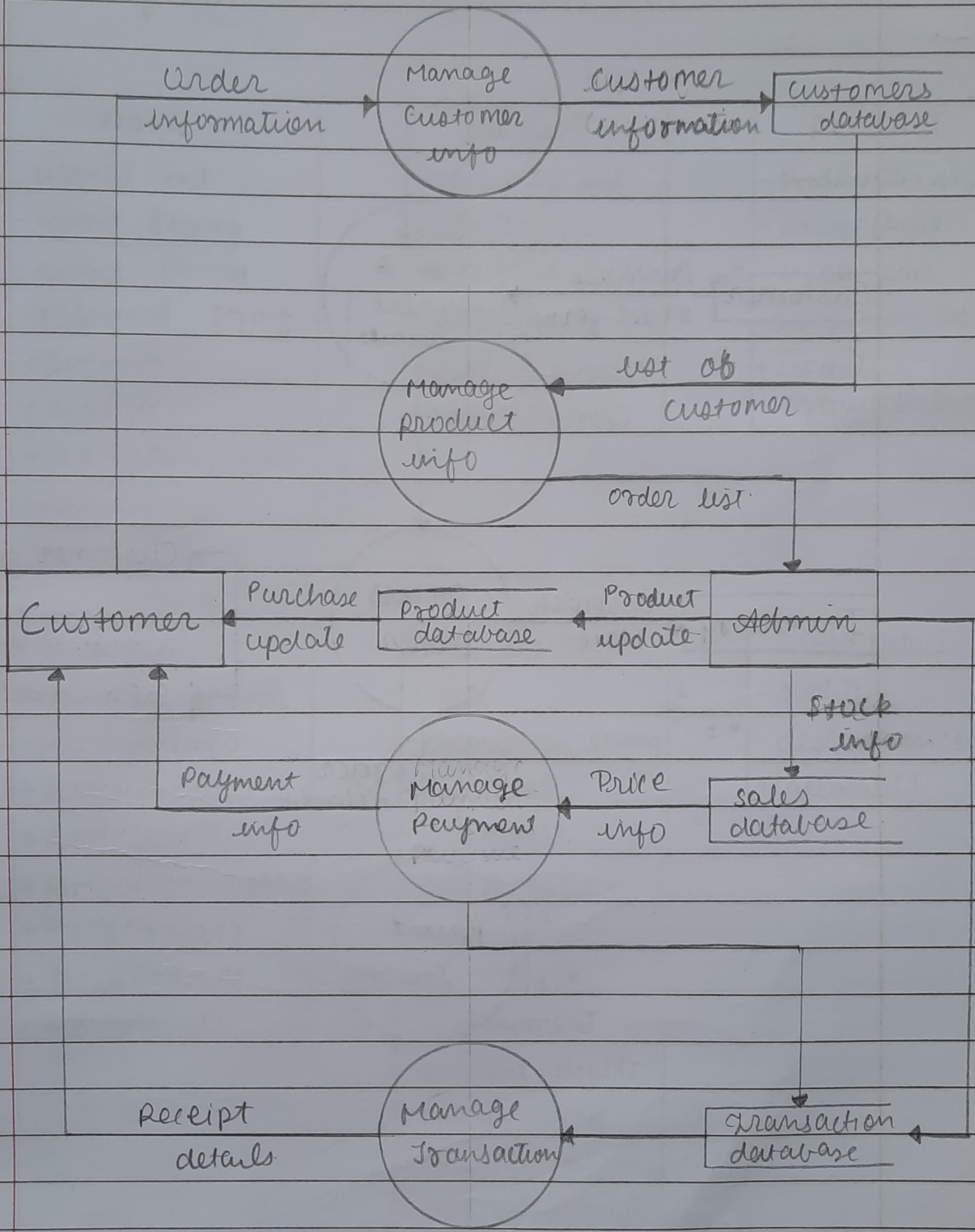
1. LEVEL - 0



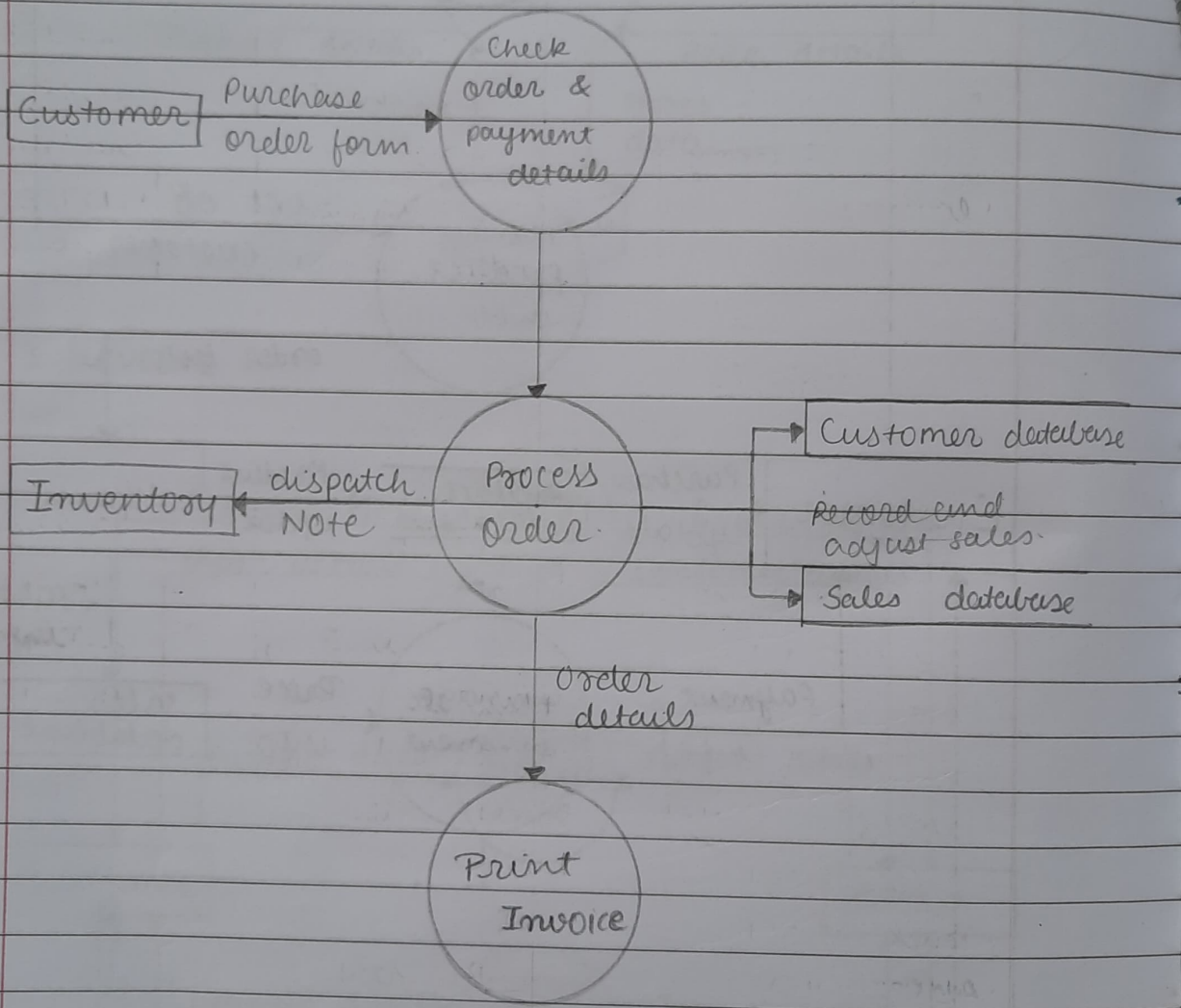
2. LEVEL - 1



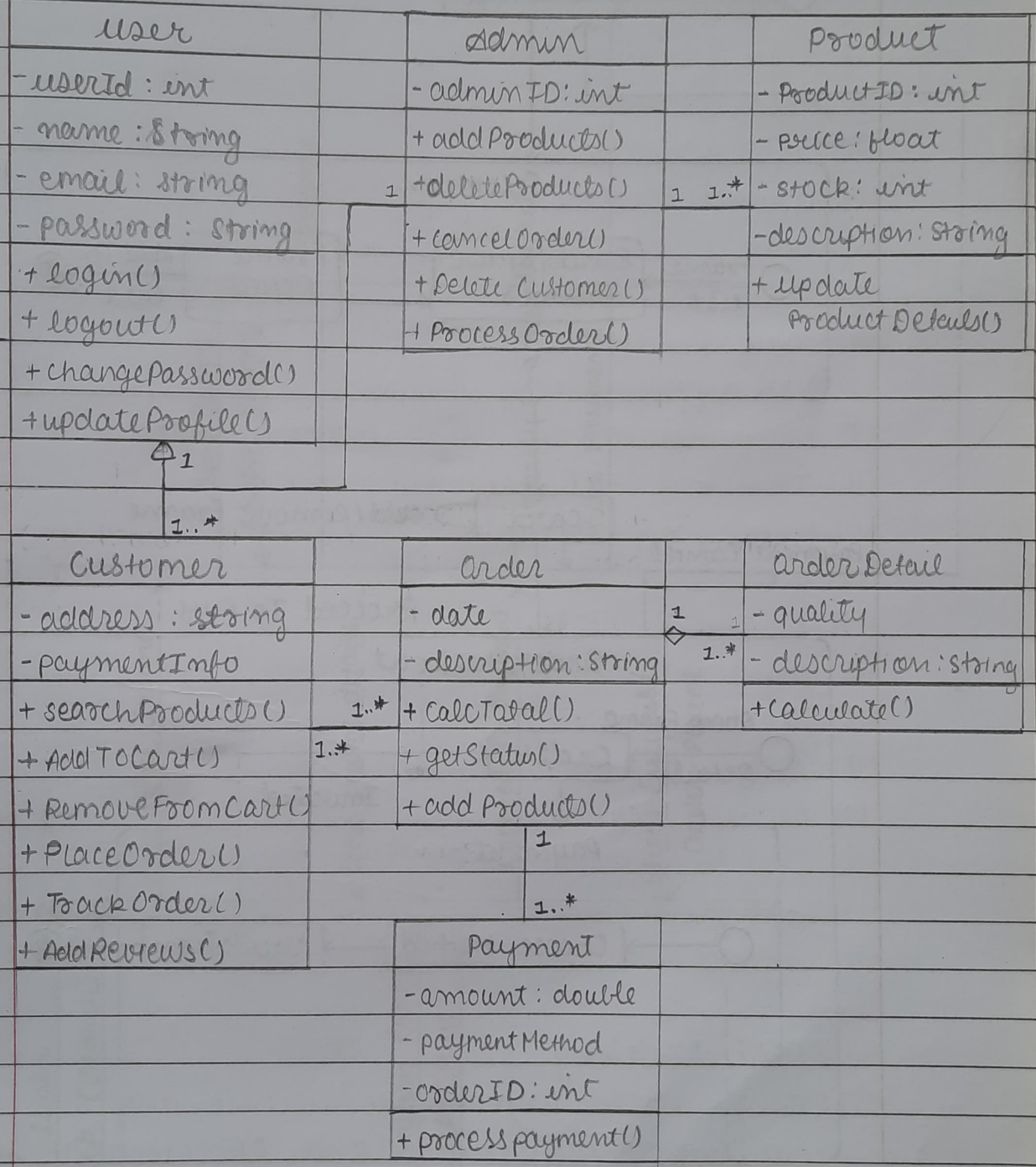
3 LEVEL-2


Order information
Manage Customer info
Customer information
customers database
Manage product info
list of customer
order list
Customer
Purchase update
Product database
Product update
Admin
Stock info
Payment info
Manage Payment
Price info
sales database
Receipt details
Manage Transaction
transaction database

4. LEVEL - 3

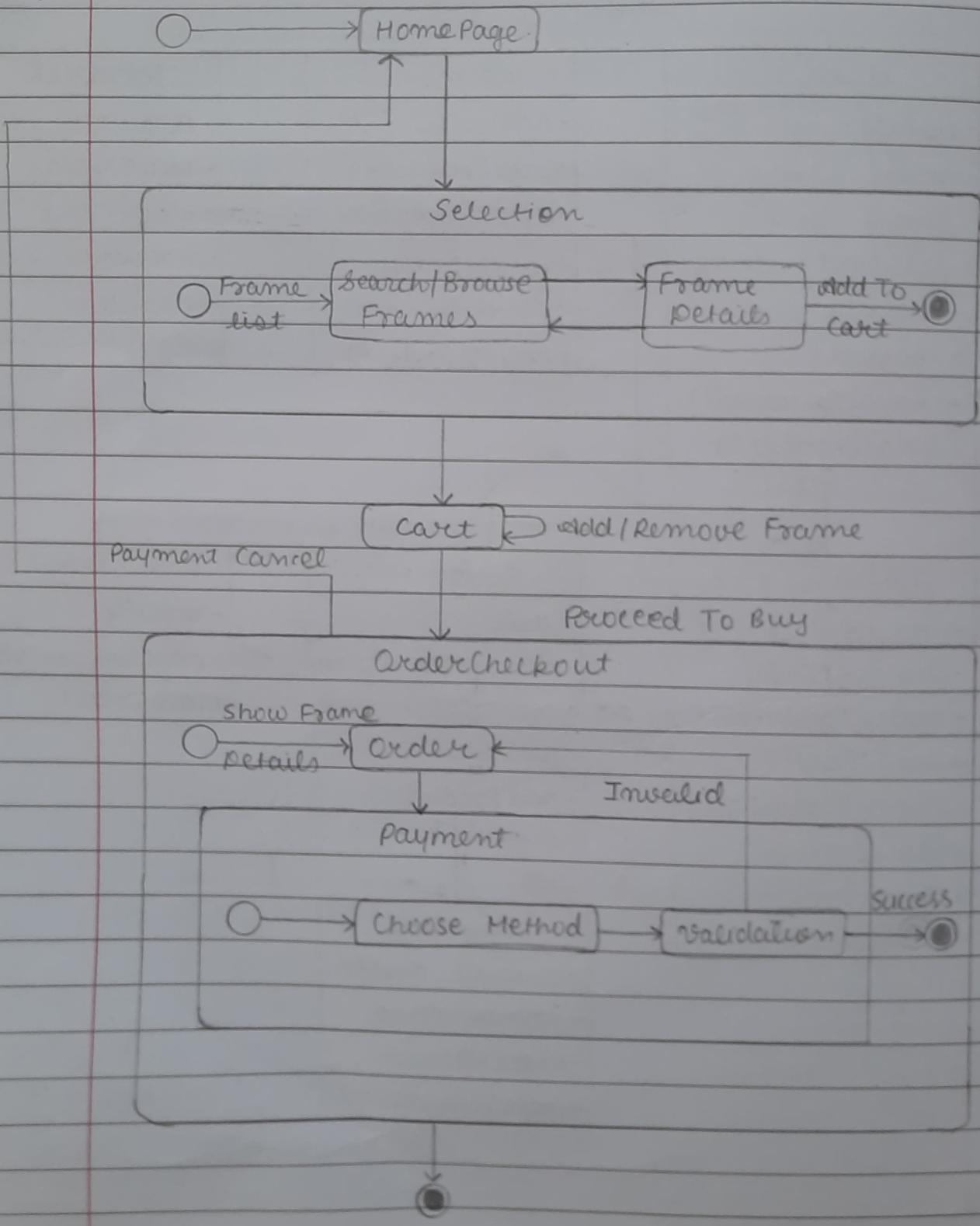

Customer
Purchase order form
Check order & payment details
Inventory
dispatch Note
Process Order
Customer database
Record and adjust sales
Sales database
Order details
Print Invoice

(iii) Enlist the number of classes. Design class diagram

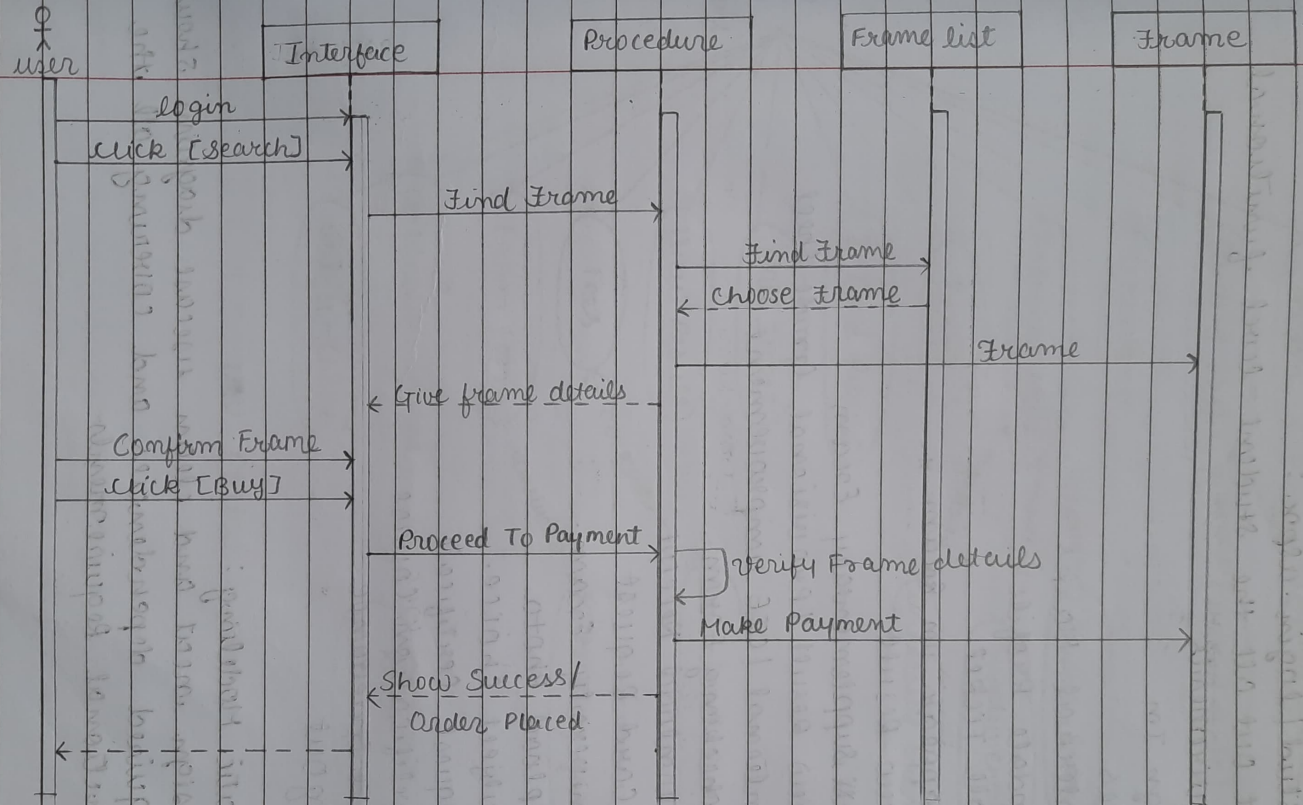


c) Dynamic Modeling:

① Design State Diagram.



(ii) Design sequence diagram



Q.4 Develop system modeling of https://app.situ.ac.in/stud/login.aspx:

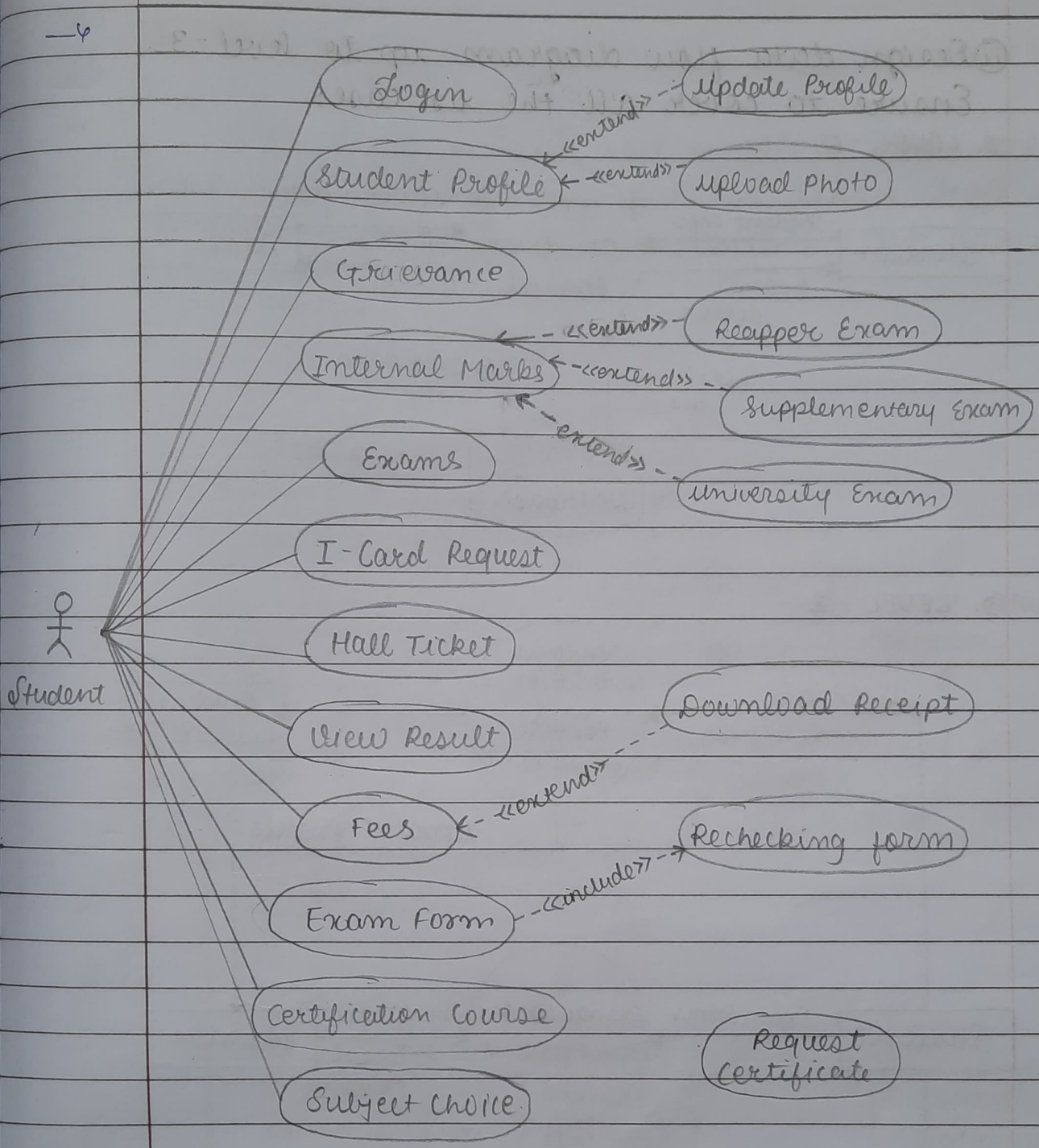
a) list out all the student-level funtional requirements

—ᵠ (1) Sign In
(2) Fees
(3) Internal Marks
(4) Update Profile
(5) Hall Ticket
(6) Reappear in Exam.
(7) View Result
(8) ~~NAME~~ Supplementary Exam
(9) View Result / Provisional Grade Sheet
(10) Internal / CIE improvement
(11) Rechecking Form.
(12) Download Receipt.
(13) I Card Request
(14) University Exam.
(15) Upload Photo
(16) Subject Choice.
(17) Request Certificate
(18) Certification Course.
(19) ~~Gre~~ Grievance.
(20) log out

b) Static Modeling:

(1) Design neat and clean usecase diagram showing required dependencies and covering all the functional Requirements

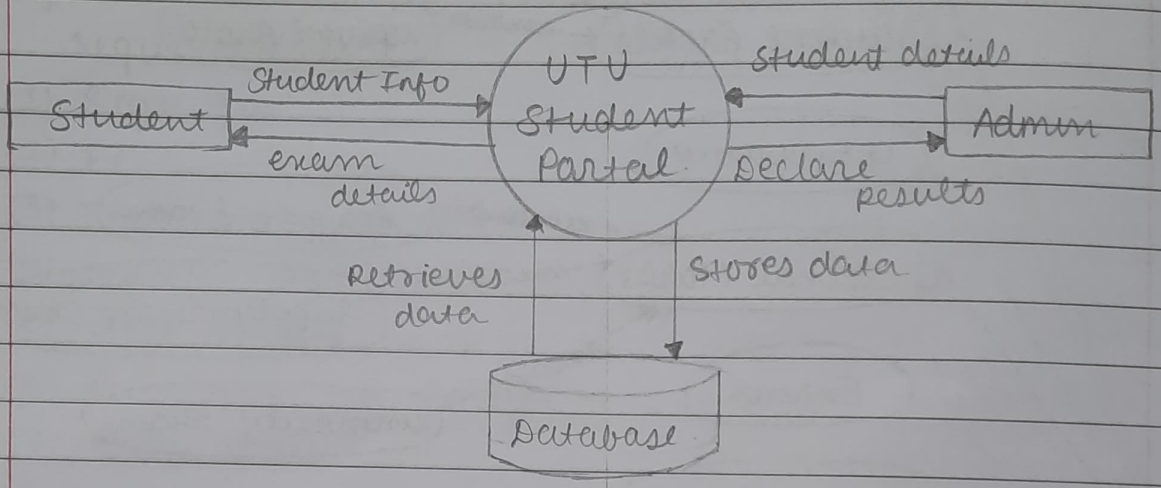
—6


Login
Update Profile
«extend»
Student Profile
«extend»
Upload Photo
Grievance
«extend»
Reapper Exam
Internal Marks
«extend»
Supplementary Exam
«extend»
Exams
University Exam
I-Card Request
Hall Ticket
Student
Download Receipt
View Result
«extend»
Fees
Rechecking form
«include»
Exam Form
Certification Course
Request Certificate
Subject Choice

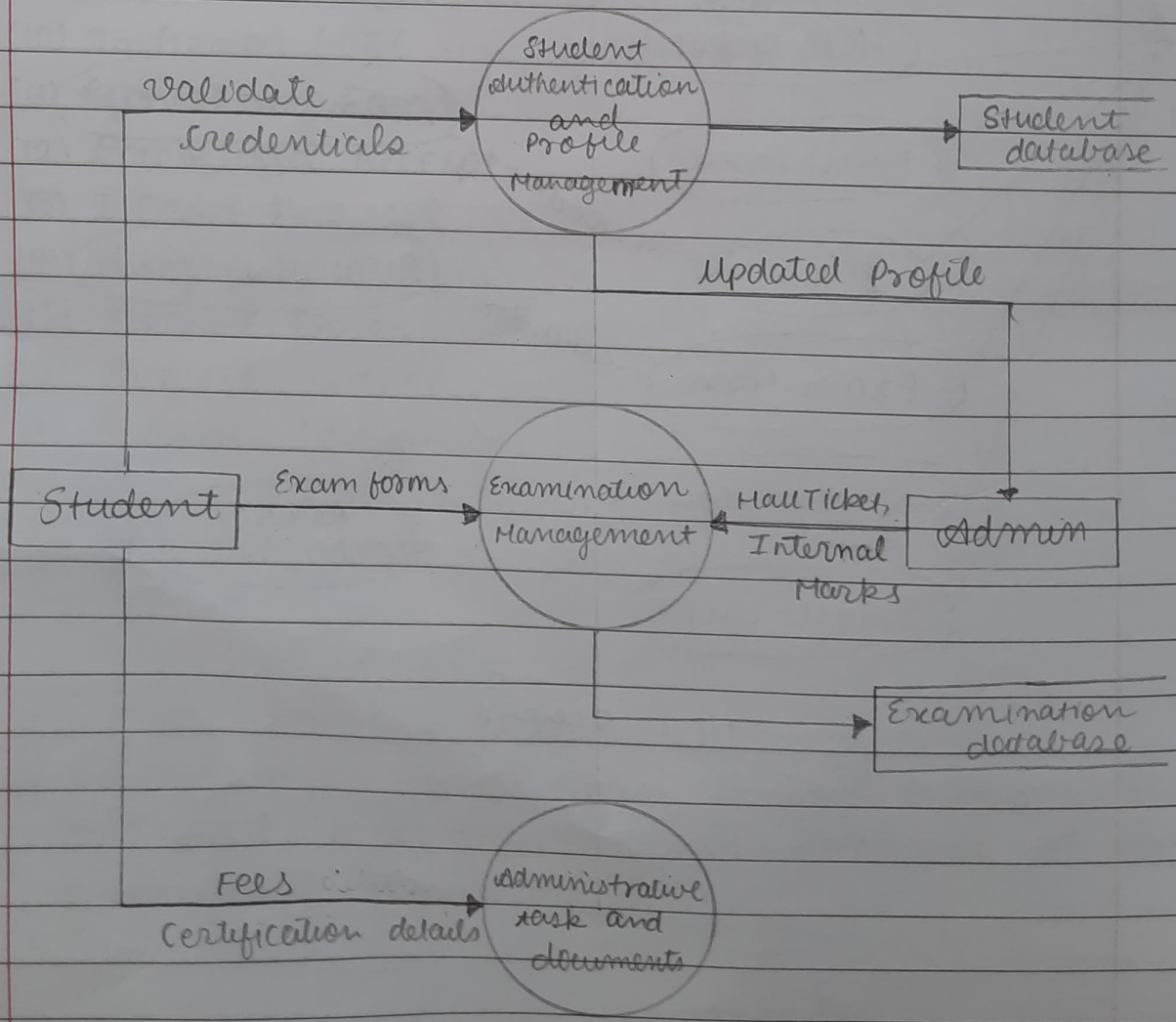
UTU Portal

(ii) Design data flow diagram up to level-3. Ensure to cover all the use cases.

1. LEVEL - 0

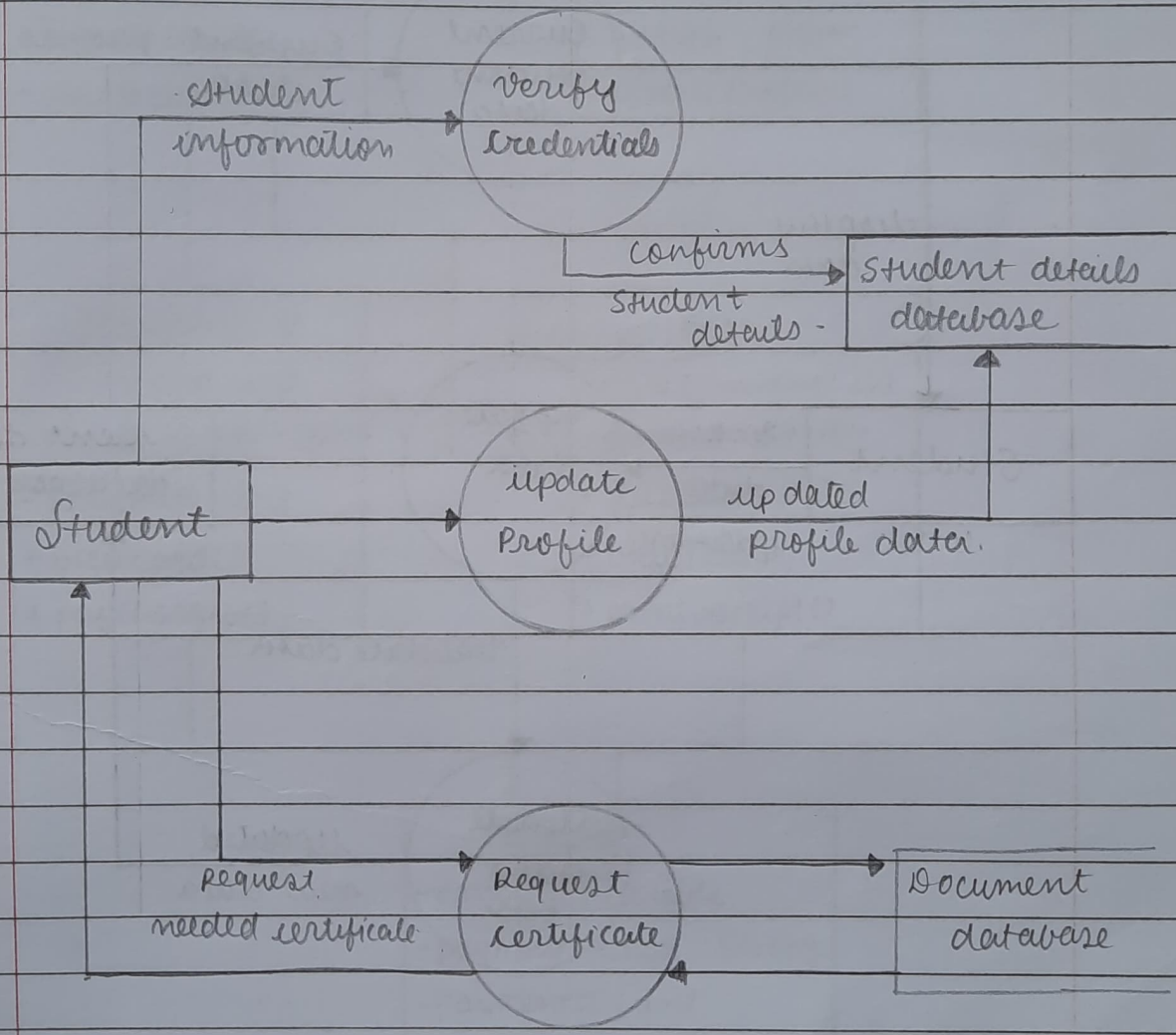


2. LEVEL - 1.



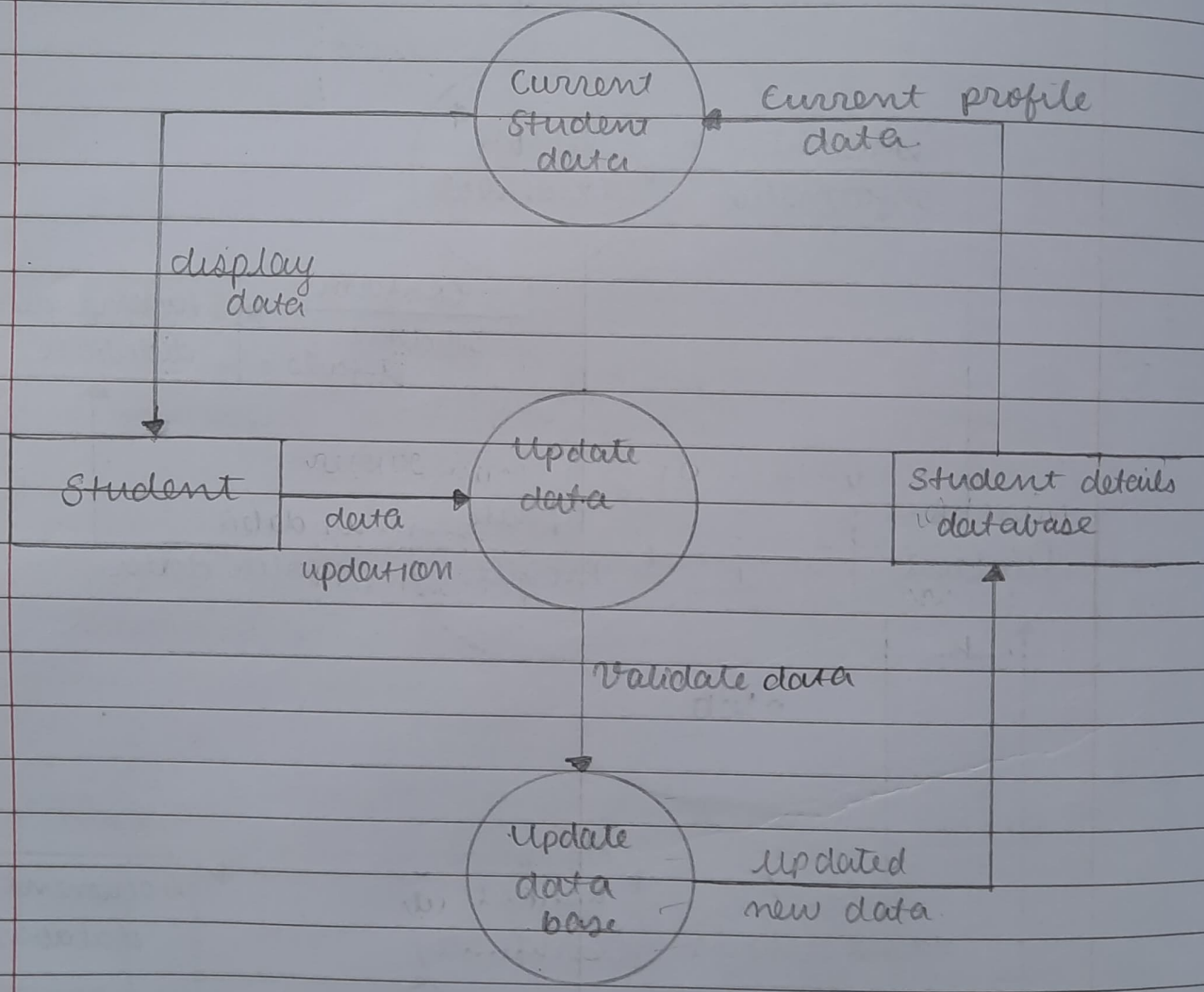
3. LEVEL - 2

For level-2 DFD, it involves the breaking down of one of the process from level-1, namely "Student Authentication and Profile Management".



4.1.1 LEVEL-3

For level-3 DFD, it involves further detailing of the subprocess "Update Profile" from level-2



(iii) Enlist the number of classes. Design class diagram.

→

Student
- enrollmentNo : int
- password : String
+ login()
+ UpdateProfile()
+ requestICard()
+ viewResult()

StudentProfile
- name : string
- email : String
- profilepicture
- phone : int
- address : string
+ updateProfile()

ICardRequest
- requestID : int
- enrollmentNo : int
- IssueData : int
+ viewCard()
+ paycharges()

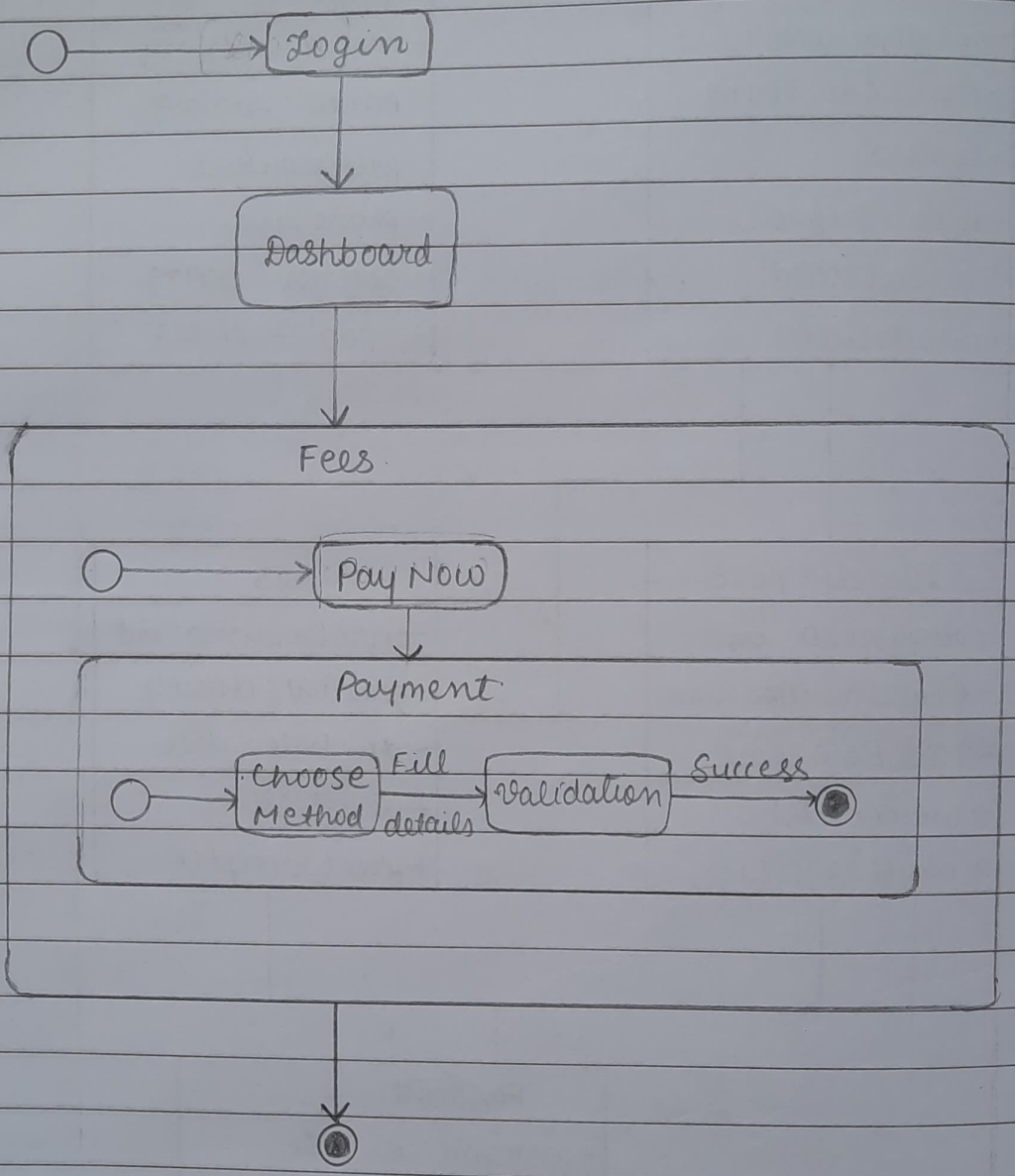
Fees
- enrollmentno : int
- amount : double
- due Date : date
+ viewFees()
+ printreceipt()

Payment
- amount : double
- paymentMethod : String
- paymentID : int
- payment Date : date
+ processPayment()

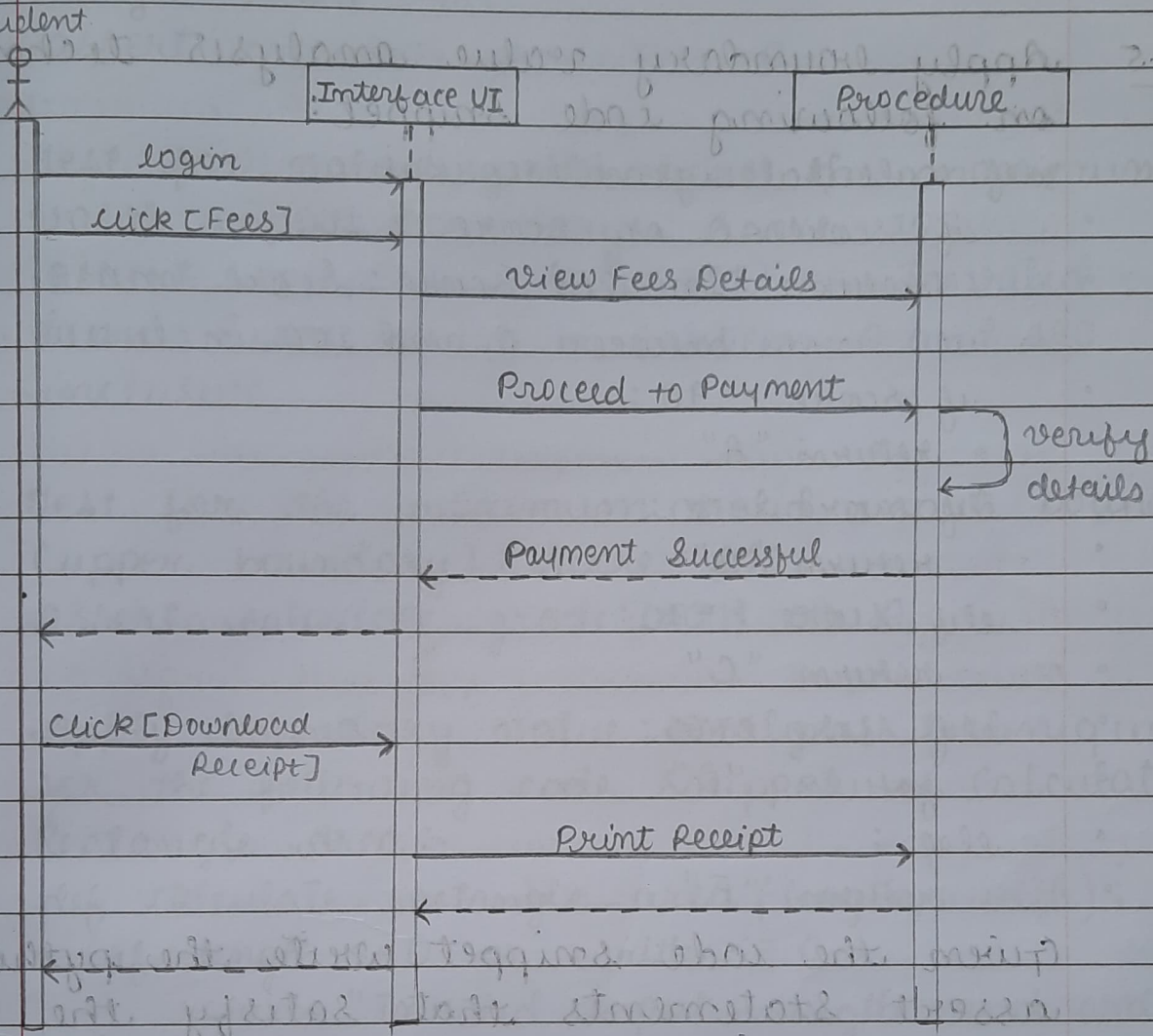
c) Dynamic Modeling :

(i) Design State Diagram

—>



(ii) Design Sequence diagram



[PART-2 : Testing]

Q.5 Apply boundary value analysis technique on following code snippet:

```
def calculate_grade(score):
    if score < 0 or score > 100:
        return "Invalid score: Score must be
                between 0 and 100 inclusive."
    if score >= 90:
        return "A"
    elif score >= 80:
        return "B"
    elif score >= 70:
        return "C"
    elif score >= 60:
        return "D"
    else:
        return "F"
```

Given the code snippet write the python assert statements that satisfy the following test cases:

i) Test for the minimum valid input boundary (lower boundary) : Score = 0

→ assert calculate_grade(0) == F

ii) Test for values just below the minimum valid input boundary : Score = -1.

→ assert calculate_grade(-1) == Invalid Score: Score must be between 0 and 100 inclusive.

iii) Test for typical input within the valid range: Score = 75.

→ assert calculate_grade(75) == C

& iv) Test for values just above the maximum
vi) valid input boundary: Score = 101

→ assert calculate_grade(101) == Invalid Score: Score must be between 0 and 100 inclusive.

v) Test for the maximum valid input boundary (upper boundary): Score = 100.

→ assert calculate_grade(100) == A

Q.6 Apply boundary value analysis technique for the following code snippet of Calculate Rectangle Area:

```
def calculate_rectangle_area(length, width):
    if length <= 0 or width <= 0:
        return "Invalid input: length and width must be positive values."
    area = length * width
    return area
```

Given the code snippet write the python assert statements that satisfy the following test cases: Boundary [0, 1000]

i) Test for the minimum valid input boundary for length and width (lower boundary).

→ assert calculate_rectangle_area(0^1, 0^2) == Invalid

input: length and width must be positive values.

ii) Test for values just below the minimum valid input boundary for both length and width.
→ assert calculate-rectangle-area(-1,-1) == Invalid input: length and width must be positive values

iii) Test for typical input within the valid range.
→ assert calculate-rectangle-area(50,50) == 2500

iv) Test for values just above the minimum valid input boundary for both length and width
→ assert calculate-rectangle-area(0.5,0.5) == 0.25

v) Test for the maximum valid input boundary for length and width (upper width)
→ assert calculate-rectangle-area(1000,1000) == 1000000

vi) Test for values just above the maximum valid input boundary for both length and width
→ assert calculate-rectangle-area(1001,1002) == 1003002

Q.7 Apply boundary value analysis technique for the following code snippet of is-prime:

```
def is-prime(number):
    if number < 2:
        return False
    for i in range(2, int(number ** 0.5) + 1):
```

```
        if number % i == 0:
            return False
    return True
```

Given the code snippet write the python assert statements that satisfy the following test cases: [2, 1000]

i) Test for the minimum valid input boundary for number (lower boundary).

→ assert is_prime(2) == True

ii) Test for values just below the minimum valid input boundary.

→ assert is_prime(1) == False

iii) Test for a small prime number within the valid range.

→ assert is_prime(11) == True

iv) Test for a non-prime number within the valid range.

→ assert is_prime(50) == False

v) Test for values just above the maximum valid input boundary

→ assert is_prime(1001) == False

vi) Test for the maximum valid input boundary for number (upper boundary).

→ assert is_prime(997) == True

Q.8 Explain Black-Box testing approach with its advantages and disadvantages.

—↓ Black-Box testing approach is a type of software testing that focuses on evaluating the functionality of a software application without requiring detailed knowledge of its internal code or design. The Tester assess the software by inputting various test cases and observing the corresponding outputs or behaviours.

- Advantages :~

i) Independence from Implementation

Testers do not need to have knowledge of the internal code, algorithms or data structures of the software being tested which makes it suitable for non-developer tester.

ii) User-Centric

Since this testing focuses on the software's external behaviour, it is an excellent way to assess whether the software meets the user expectations and requirements.

iii) Validation of Functional Requirements

It helps verify that the software ~~functions~~ functions correctly and meets its intended purpose.

iv) Early Detection of Usability Issues.
It can identify usability and user interface issues that may not be apparent in code-focused testing approaches.

- Disadvantages :-

i) Inefficiency in Identifying Bugs.
It is not ideal for finding certain types of bugs like low-level, implementation-specific bugs such as memory leaks, race conditions or performance issues.

ii) Test Case Design Challenges.
Designing effective test cases that cover a wide range of scenarios can be complex and time-consuming as it depends on the skill and creativity of the testers.

iii) Difficult in Reproducing Issues.
When a bug is found, it can be challenging to provide developers with enough information to reproduce and fix the problem because of lack insight into the code.

iv) Incomplete Testing
It may not cover all possible scenarios or cases, especially if the requirements are not well-documented or understood.

Q.9 Explain White-Box testing with its types and significance.

--> White-Box Testing or Glass-Box Testing, is a software testing approach that focuses on assessing the internal workings of a software application. Here, testers have access to the source code, architecture and design of the software. This allows testers to design test cases that target specific code paths, statements, branches and logic.

There are mainly two types of white box testing, which are:

i) Static Testing

Examines the code without executing it. It includes techniques like code reviews, inspections and walkthroughs to identify issues such as syntax errors, design flaws and adherence to coding standards.

Static code analysis involves using tools to analyze the source code for vulnerabilities, security issues without executing.

Code Review involves manual examination of the code by peers or a team to identify errors, improve code quality.

ii) Dynamic Testing

It involves assessing the internal workings of a software application during runtime by evaluating the behaviour of

the system. The primary goal is to ensure that the system software functions as intended, identifying any runtime errors or issues. Further Here are some key aspects:

i) Unit Testing: Focuses on testing individual units of the software in isolation to ensure that each component functions correctly.

ii) Integration Testing: Verifies the interactions between integrated components. It ensures that the different parts of the software work together seamlessly.

iii) System Testing: Validates the entire system software against its specified requirements

iv) Acceptance Testing: It confirms that the software satisfies the business requirements and is ready for deployment

- Significance :-

> Early Detection of Coding Errors
> Code Quality Assurance
> Improved Test Coverage
> Verification of Algorithm and Logic
> Debugging Support

Q.10 Discuss a six sigma strategy for quality assurance.

→ Six Sigma is a data-driven and structured methodology for improving the quality of processes and output. It aims to reduce defects and variations in processes to achieve a high level of quality and customer satisfaction. Here's the strategy for implementing Six Sigma in quality assurance:

i) Define

Identify the specific quality assurance area or process that requires improvement. Clearly define the problem, the process and its critical components.

ii) Measure

Collect data to assess the current state of the quality assurance process which involves gathering information about errors, defects and variations in the process. Identify key performance indicators (KPIs) and metrics that are relevant to quality assurance.

iii) Analyze

Identify the root cause of defects and variations in the quality assurance process. Use techniques like root cause analysis, process mapping and statistical analysis to pinpoint the source of problems. Prioritize most critical issues

iv) Improve

Develop and implement solutions to address the identified root causes. Use data-driven methods to design and test process improvements. Ensure that changes are well-documented.

v) Control

Establish control mechanisms and monitoring systems to sustain the improvements made in the quality assurance process. Implement Statistical Process Control (SPC) to track process performance and ensure it remains within acceptable limits.

vi) Verify

Const Continuously measure and validate the results of the improvements. Make adjustments and refinements as needed to maintain and improve quality over time and also ensure that it meets predefined goals and metrics.

Q.11 Explain ISO 9000 quality standards.

-> ISO 9000 is a family of international quality management standards that provide a framework for organisations to establish, implement, maintain and continually improve their quality system. These standards are developed by International Organization for Standardization (ISO).

The ISO 9000 is a series of three standards which various industries apply:

i) ISO 9001: Standards applies to the organization engaged in design, development, production and servicing of goods / product (Software)

ii) ISO 9002: Standards applies to those organization which do not design product but are only involved in the production.

iii) ISO 9003: Standards applied to organizations that are involved only in the installation and testing of the products.

- The benefits of implementing ISO 9000:

> Improved product and service quality
> Better risk management.
> Enhanced customer satisfaction
> Increased operational efficiency
> Improved relationships with suppliers.

Q.12 Define Debugging and discuss debugging ~~strategies~~ strategies.

→ Debugging is the process of identifying, analyzing and resolving issues in software or hardware systems. These issues can manifest as unexpected behaviours, errors, crashes or malfunctions.

. Debugging Strategies are :-

i) Understand the Code : Begin by thoroughly understanding the code. Read the code carefully to gain insights into its logic and structure.

ii) use logging : Insert log statements strategically in the code to track the flow of the program and the values of variables.

iii) Debugging Tools : Take advantages of debugging tools provided by the IDE.

iv) Isolate the Problem : Break the code into smaller sections and test each part individually.

v) Code Review : Have colleagues review your code. It can often help you catch issues.

vi) Explore Documentation : Consult documentation for the programming language, libraries and frameworks you are using.

Q.13 Discuss software testing strategy in context of spiral approach.

-4 The Spiral Model combines the idea of iterative development with the systematic aspects of the waterfall model. The model is divided into series of spirals, each representing a phase:

The Software testing strategy for spiral approach is:

i) Planning.

Clearly define the objectives of testing for the current spiral which includes determining the scope of testing, specific functions to be tested and testing techniques.

ii) Risk Analysis.

Identify and assess risk related to the software being developed. Risks can be technical, operational or related to project management. Based on identification, a comprehensive testing strategy is developed.

iii) Engineering

The actual development and prototyping of the software occurs. Testing activities during this phase should focus on verifying and validating the evolving product.

iv) Testing.

Here, the dynamic testing is done which consist of unit Testing, integration testing, system testing and acceptance testing. After that regression testing is done i.e. re-run previously executed tasks.

v) Evaluation

assess the results of testing activities and gather feedback from stakeholders. Based on that, make decisions about whether to proceed to the next spiral or to revisit and modify.

Q.14 Explain unit testing approach with an appropriate example.

→ Unit Testing is a software testing approach where individual units are tested in isolation to ensure they function as intended. This helps identify and fix bugs early in the development process and ensures that each unit works independently before integrating them into the larger systems.

Let's take a simple python code to perform unit testing:

code:

```
def add_numbers(a, b):
    return a+b
```

Now perform unit testing

```
import unittest
class test(unittest.TestCase):
    def add(self):
        result = add_numbers(3,5)   #TestCase-1
        self.assertEqual(result, 8)
```

Here, 'add_numbers' is a method which performs addition on the two parameters. To run tests against the program The 'test' class is a Test Case class that inherits from 'unittest.TestCase'.

The 'add' method is an individual test case. It uses the 'self.assertEqual' method to check if the actual result matches the expected result.

Q15 Explain integration testing approach with an appropriate example.

→ Integration Testing focuses on verifying the interactions and interfaces between different software components when they are integrated into a larger system. It is essential to detect and resolve issues that may arise when components interact with one another.

Example : E-commerce Website

Let's consider an e-commerce website that has different modules such as

- User authentication
- Product Catalog
- Shopping Cart
- Payment Processing

Now before integration Testing, each module will be tested in isolation to ensure that solo components work correctly.

Now, the modules are integrated and tested as a whole.

- Scenario-1: User Adds Products to Cart and Checks Out

① A user logs in (User Authentication Module)

② Browse the Products Catalog, adds items to shopping cart (product catalog, shopping cart)
③ Proceeds to checkout and completes the payment (shopping cart module, payment processing modules)

Q.16 Explain Garvin's Quality Dimensions.
→ David A. Garvin, introduced a framework of eight quality dimensions that organizations can use to assess and improve the quality of their products and services. These dimensions provide a comprehensive perspective on what constitutes quality in different contexts

i) Performance
This dimension focuses on the primary operating characteristics of a product or service. It is a fundamental quality dimension because it directly ~~meets~~ relates to meeting customer needs and expectations

ii) Features
Features refer to the additional characteristics and capabilities of a product or service beyond its basic functionality.

iii) Reliability
Reliability is the consistency and dependability of a product's performance over time which address the likelihood of the product failing down

iv) Comfarmance

Conformance measures how well a product adheres to established standards, specifications or requirements ensuring that the products meet the established criteria

v) Durability

Durability pertains to the expected lifespan and robustness of a product.

vi) Serviceability

It focuses on how easy it is to maintain and repair a product if it malfunctions or requires maintenance.

vii) Aesthetics

It relates to the visual appeal and sensory aspects of a product.

viii) Perceived Quality

It is how customers perceive and feel about a product. It's often influenced by factors like brand reputation, marketing, user experience and customer service.

~~~ x ~~~ x ~~~
~~~