****

**SDA ASSIGNMENT**

**DEPARTMENT: SOFTWARE ENGINEERING**

**NAME: KAMRAN FIAZ**

**REG NO: SP23-BSE-143**

**SUBJECT: SOFTWARE DESIGN & ARCHITECTURE**

**SEMESTER: 5TH**

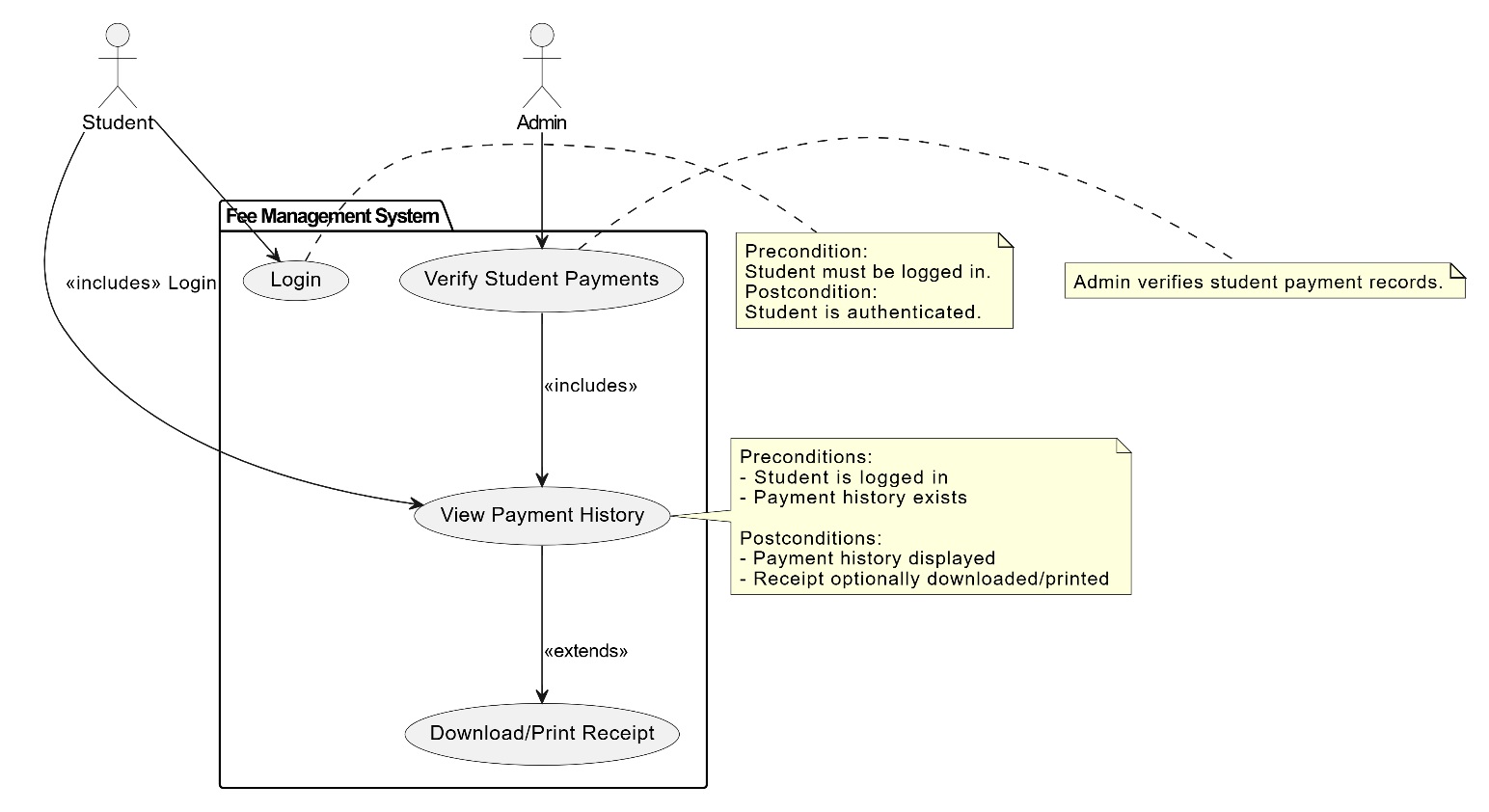
**SUBMITTED TO: SIR MUKHTIAR ZAMIN**

**SUBMISSION DATE : APRIL, 3, 2025**

**PROJECT NAME: FEE MANAGEMENT SYSTEM.**

**USECASE: VIEW PAYMENT HISTORY.**

1. **UseCase Diagram:**



1. **Fully Dressed UseCases:**

**Scope:** Fee Management System.

**Level:** User Goal.

**Primary Actor:** Student.

**Stakeholders and Interests:**

• **Student:** Wants to view and verify past payment details.

• **Admin:** May verify that the student has made payments.

• **System:** Maintains accurate and accessible records of all transactions.

**Preconditions:**

• Student must be logged into the system.

• Payment history must exist in the system database.

**Postconditions:**

• Payment history is displayed to the student.

• Optionally, a receipt may be downloaded or printed.

**Main Success Scenario (Basic Flow):**

1. Student logs into the system.

2. Student navigates to the “Payment History” section.

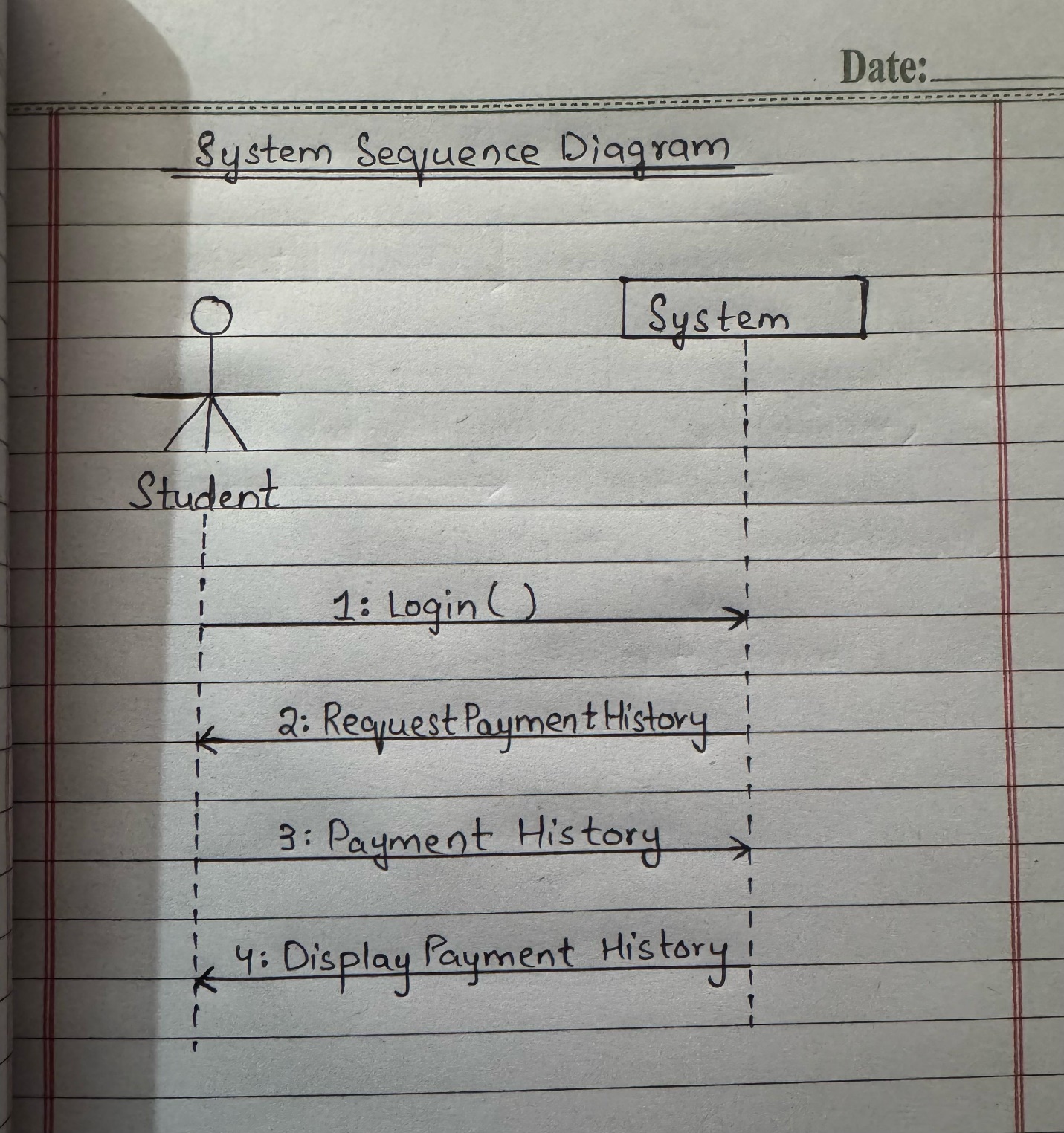
3. System retrieves payment records linked to the student’s ID.

4. System displays the list of payment transactions (date, amount, mode, status).

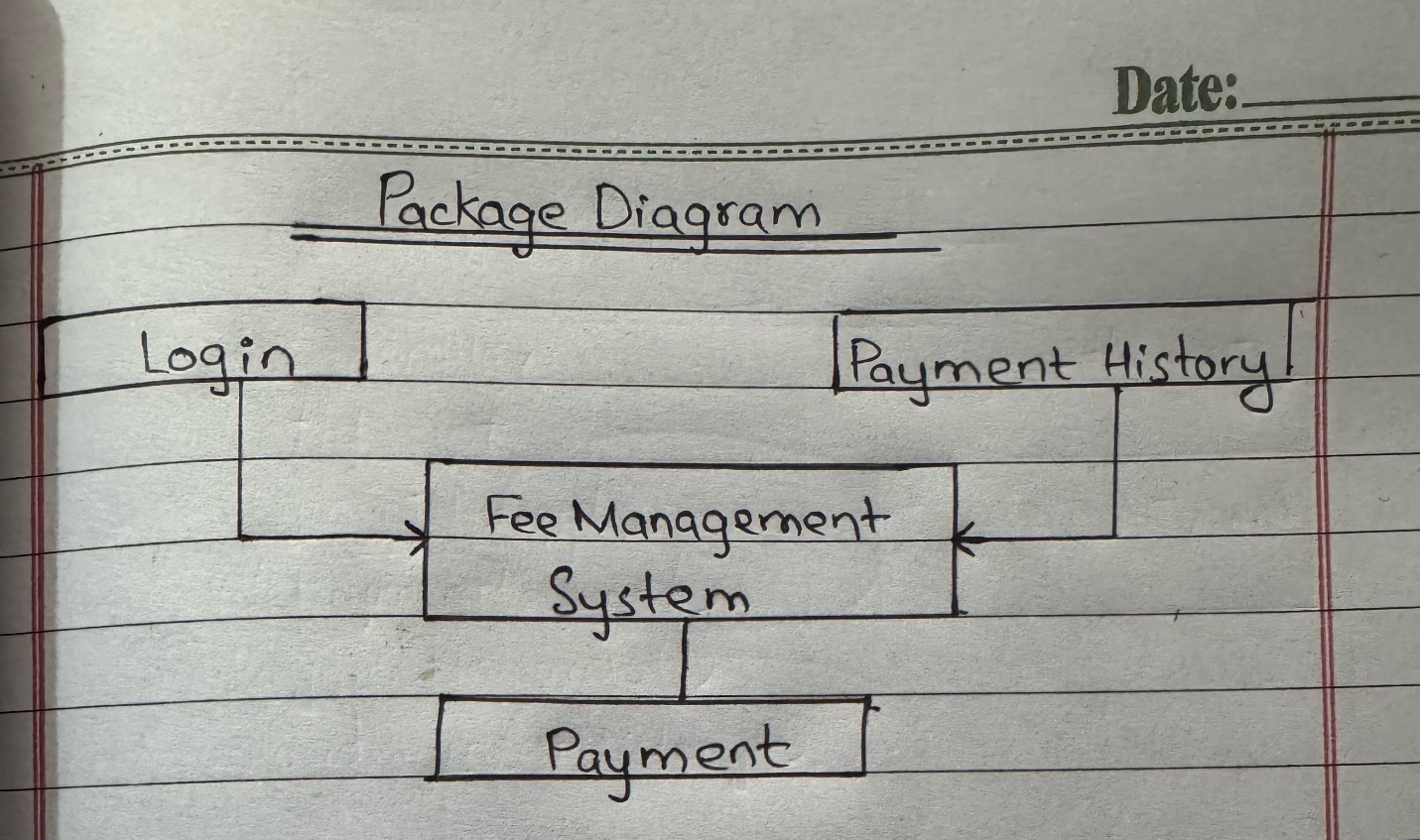
5. Student reviews the displayed records.

6. Student optionally chooses to download/print a receipt.

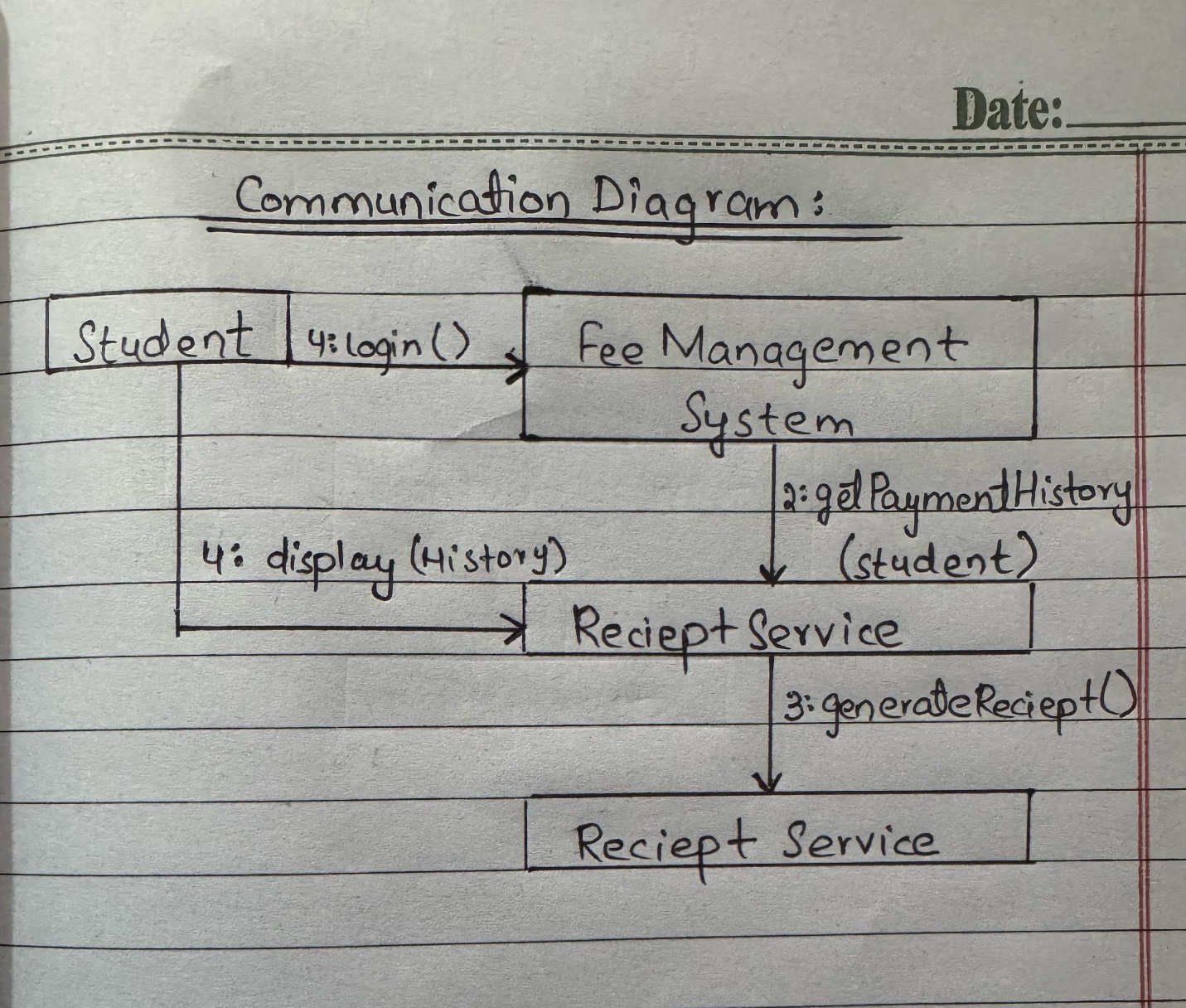
1. **System Sequence Diagram:**

****

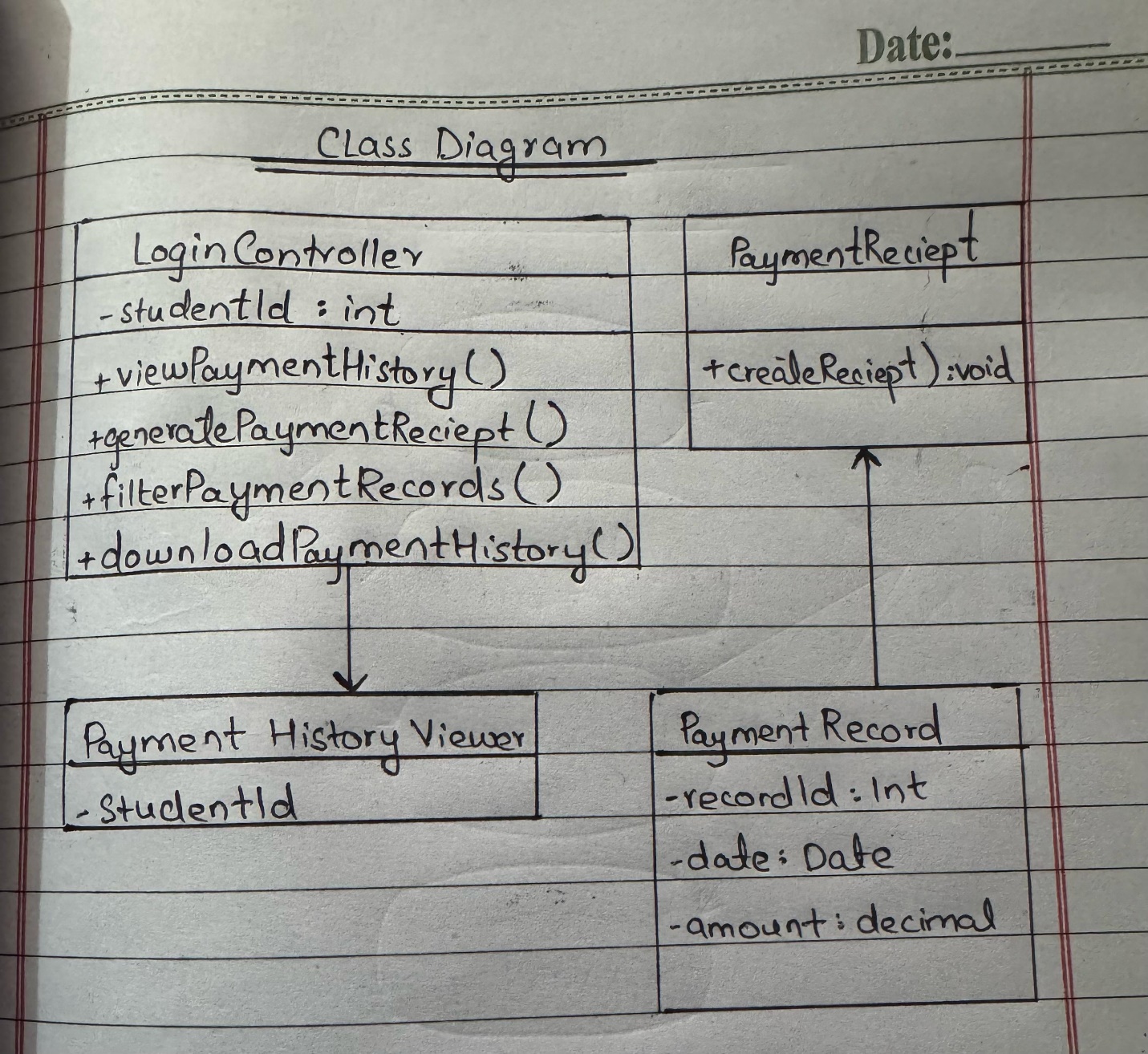
1. **Package Diagram:**

****

1. **Cummunication Diagram:**

****

1. **Class Diagram:**

****

1. **Coding Standards:**

**1. General Coding Standards:**

• Follow naming conventions:

• Variables/methods: camelCase (e.g., getPaymentHistory())

• Classes: PascalCase (e.g., PaymentHistoryController)

• Constants: UPPER\_SNAKE\_CASE

• Use meaningful names: Avoid temp, data, or xyz; instead use paymentList, studentId, etc.

• Keep functions short and single-responsibility

• Avoid hardcoding values; use config files or constants.

**2. Backend (e.g., Java, Python, Node.js):**

• Use layered architecture (Controller → Service → Repository)

• Example file structure (Java Spring Boot):

/controller/PaymentHistoryController.java

/service/PaymentHistoryService.java

/repository/PaymentRepository.java

/model/Payment.java

Handle exceptions with meaningful messages (e.g., PaymentHistoryNotFoundException)

• Use DTOs (Data Transfer Objects) to limit exposure of sensitive fields.

• REST API endpoint naming: /api/payments/history/{studentId}

**3. Frontend (e.g., React, Angular, Vue):**

• Component naming: PaymentHistoryComponent, PaymentRow

• Avoid inline styles, use CSS/SCSS files or CSS modules.

• Use state management (e.g., Redux, Vuex) where appropriate.

• Implement loading and error states

• Sanitize user input (especially filters and form data).

**4. Database Design:**

• Use normalized schema

• Table: payments

• payment\_id (PK)

• student\_id (FK)

• amount

• date

• payment\_method

• status

• Use consistent naming, avoid abbreviations

• Use indexes on student\_id, date for performance

**5. Security Best Practices:**

• Use parameterized queries to prevent SQL injection

• Mask sensitive info like card details in frontend/backend logs

• Authenticate and authorize access to payment history via tokens (e.g., JWT)

**6. Documentation & Comments:**

• Use doc comments (/\*\* \*/ in Java, /// in Dart) to explain function purpose

• Keep inline comments concise and relevant

• Write README or API documentation using Swagger/OpenAPI