Project Name: **Student Attendance Management System**
Responsible: **Pratyush Kakkar (20022933)**

Document Versions and Revisions			
Date	**Comment**	**Author(s)**	**Version**
19/04/2023	Document creation	Luciana Nascimento	1.0
19/12/2024		Pratyush Kakkar	**2.0**

CA2
Attendance Management System

1 Architectural Style(Pattern):

Model View Controller

The choice of **Model-View-Controller** (MVC) is suitable for the *Student Attendance Management System* (SAMS) because it is in harmony with the requirements necessary for the further development of the program, in terms of '**Modular Structure'**, '**Maintainability'** and '**Scalability'**. This architectural pattern is applicable to the Student Attendance Management System (SAMS) because it divides the responsibilities of the application into three categories: the *data* (**Model**), the presentation layer or *User Interface* (**View**), and the *logic* that connects the two (**Controller**). The benefits include:

Flexibility and Scalability

- MVC separates the system into **Model** (data), **View** (UI), and **Controller** (logic), making it easy to add new features or make changes.
- For example, adding new reports or user roles won't disrupt the rest of the system, helping it grow smoothly.

Faster Development

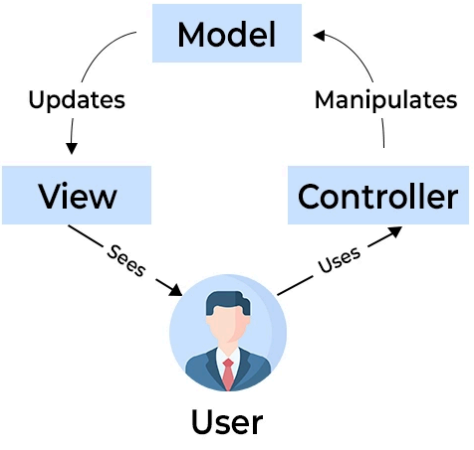
- Teams can work **in parallel**: front-end developers focus on the **View**, while back-end developers handle the **Model** and **Controller**.
- This speeds up development.

Easier Testing

- With the system divided into clear components, each part can be **tested individually.**

Supports Multiple Views

- MVC allows the same data and logic to serve **different user interfaces**.
- For example, students, teachers, and admins can each have their own view without duplicating code.



Improved Maintenance

- Updating the user interface won't affect the data or logic, and vice versa.
- This makes the system easier to **maintain** and ensures **updates** or bug fixes don't break unrelated parts.

Organisation of the Classes:

Model: which will be responsible for “Application Logic’ and ‘Data’ will contain:

1. **User** and the sub classes *Student*, *Teacher*, *Administrator*. This is because they contain important data such as Admin Credentials (e-mail, password), attendance percentages, etc. They are core entities of the system that manage attributes such as FirstName, LastName, etc.

2. **AttendanceRecord** as it stores Attendance Data (date, class, status) and update and retrieve records, as well as **AttendanceReport** as it summarizes data for classes or students by retrieving data from AttendanceRecord to generate Reports.

3. **Class & Schedule** because they manage class details and timings.

4. **ModifiedSchedule** since it tracks the changes in schedules, hence it helps admins manage and update the schedules.

View: is responsible for ‘presenting’ the data from the Model to the user and collecting inputs.

1. **StudentView** which will display attendance percentage and notifications to students.

2. **TeacherView** this interface will allow teachers to mark and view attendance.

3. **AdminView** will let admins modify schedules, manage users and view attendance.

Controller: handles user inputs and coordinates between the Model and the View.

1. **UserControlle**r would handle actions such as login, registration, etc.

2. **AttendanceController** processes requests from teachers to mark students as Present or Absent and retrieves attendance data to be viewed by users.

3. **ScheduleController** manages the function of modifying schedules.

Model	User, AttendanceRecord, Class & Schedule
View	StudentView, TeacherView, AdminView
Controller	UserController, AttendanceController, ScheduleController

Additional Classes:

the following classes need to be added to fit the **Model-View-Controller** architecture:

StudentView, TeacherView, AdminView

- Role-specific views tailored to **students**, **teachers**, and **admins**.
- Keeps the UI clean and focused:
    - **StudentView**: Shows attendance and schedules.
    - **TeacherView**: Allows marking attendance and generating reports.
    - **AdminView**: Manages users, schedules, and system reports.

UserController

- Manages user operations like **login, registration, an**d role-based actions, ensuring consistent handling for all user roles.

AttendanceController

- **Centralizes** all attendance tasks: marking attendance, viewing records, and generating reports.

ScheduleController

- Handles **class schedules**, including creation, updates, and viewing for teachers and students.

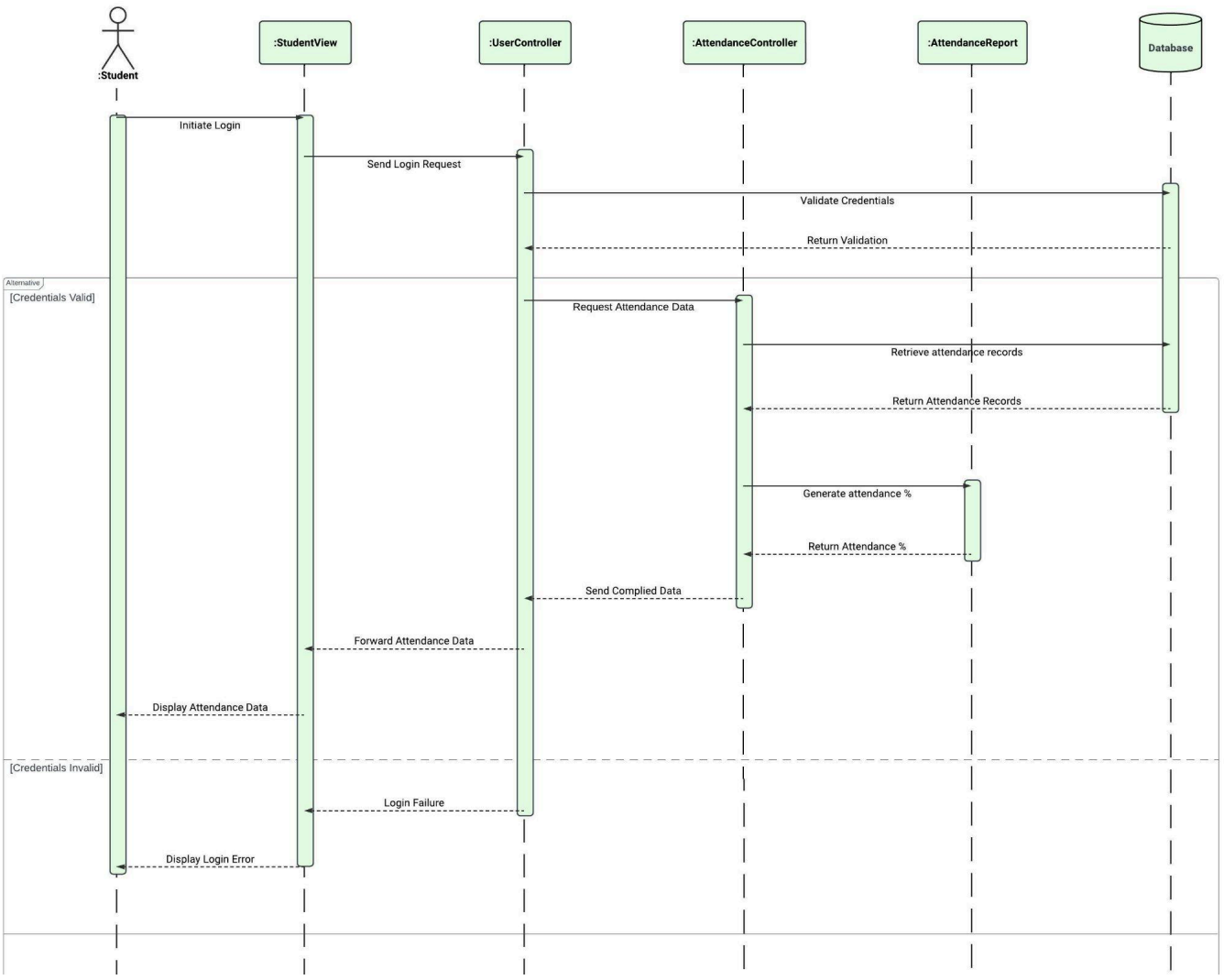
2 Sequence Diagrams

2.1 Sequence Diagram for Use Story: US FR02

As a Student, I want to view my attendance records so that I can keep track of my attendance history, including detailed information for each class. This will help me stay aware of my attendance rate and understand how it may impact my academic performance.

- The student must log into the system using their assigned username and password to ensure that only authorized users have access to their attendance data.
- The records should be detailed, showing the date, specific class or subject, and attendance status for each session (e.g., “Present,” or “Absent,”).
- The system should also provide an overall attendance percentage to give students a clear understanding of their attendance rate.

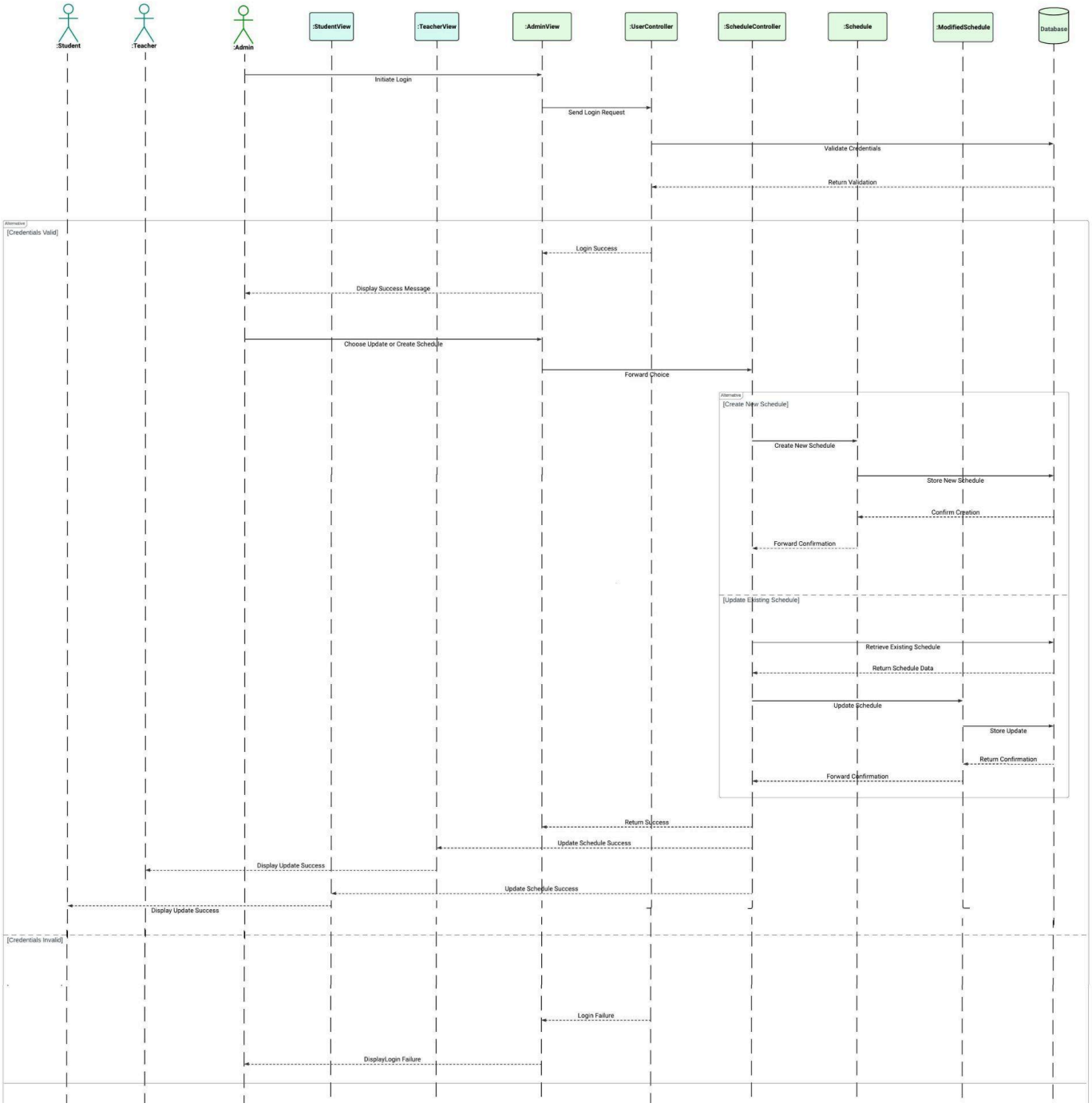
Sequence Diagram 1(Link)



2.2 Sequence Diagram for Use Story: US FOR FR03

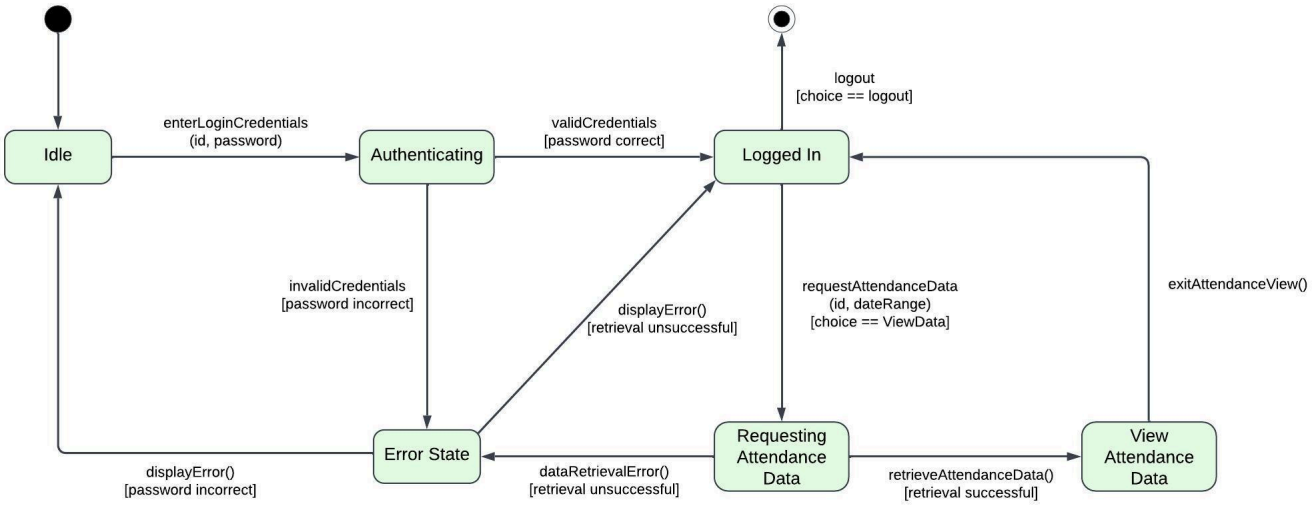
As an Administrator, I want to designate or update scheduled times for classes each day, so that a clear and organized timetable is available for all users in the system, enabling students, teachers, and other administrators to reference the daily class schedule as needed.

Sequence Diagram 2 (Link)



3 State Machine Diagram for Class ‘Student’

Student Login and Attendance View



State Chart Diagram (Link)

The diagram illustrates the flow of a Student logging into the system and viewing their attendance records. It begins with the student in a “Idle" or ‘Not Logged In’ state, progresses through authentication, and if successful, allows the student to access their attendance information. The diagram also accounts for potential errors during login or data retrieval, providing appropriate error handling and return paths.

4 Unit Test Design

Unit:

Class: AttendanceRecord

Attribute: Status

Validation rule 1: The status attribute cannot be empty

Validation rule 2: The status cannot not be updated to the same status it already was.

Unit:

Class: AttendanceRecord

Attribute: Justification

Validation rule 1: The justification should only contain alphabets, spaces and punctuation (no numbers).

Validation rule 2: The justification should not exceed 50 characters.

4.1 Unit Test 1:

Unit: class: AttendanceRecord

Testing Item: attribute 'Status'

Testing approach: Equivalence Partitioning

Steps: Identifying groups of inputs with common characteristics

Valid Partitions:

- **Valid Status:**
  - "Present"
  - "Absent"
  - "Late"

Invalid Partitions:

- **Invalid Status (Unrecognized Strings):**
  - "Unknown"
  - "Early"

- **Null or Empty String:**
  - null
  - "" (empty string)
- **Numeric Strings:**
  - "123"
- **Symbolic Strings:**
  - "$%^"

Testing cases:

Description: Test cases to evaluate if the status is not empty and of valid value	
Inputs	**Expected Results**
“Present”	Status is set to “Present”
“Absent”	Status is set to “Absent”
“Late”	Status is set to “Late”
“Unknown”	Status is set to “Absent”
“Early”	Status is set to “Absent”
“”	Status is set to “Absent”
null	Status is set to “Absent”
“123”	Status is set to “Absent”
“$%^”	Status is set to “Absent”

4.2 Unit Test 2:

Unit: class: AttendanceRecord

Testing Item: attribute ‘Justification’

Testing approach: Equivalence Partitioning

Steps:

Valid Partitions:

- **Valid Length (≤ 50 characters):**
  - "Sick due to flu."
  - "Traffic delay."
- **Valid Characters:**
  - Contains only alphabetic characters and punctuation (e.g., "Missed class due to illness.").

Invalid Partitions:

- **Exceeds Length (More than 50 characters):**
  - "This is a very long justification that exceeds fifty characters."
- **Contains Numbers:**
  - "Sick for 3 days."
  - "Missed class at 9:00 AM."
- **Contains Invalid Special Characters:**
  - "Illness due to @virus!"
  - "Absent because of #traffic."

Testing cases:

Description: Test Case to evaluate if justification is less than 50 characters and does not contain numbers or special characters.	
Inputs	**Expected Results**
“Sick due to flu”	Justification is accepted
"Traffic delay."	Justification is accepted
"This is a very long justification that exceeds fifty characters."	Justification is rejected (remains an empty string).
"Sick for 3 days."	Justification is rejected (remains an empty string).
"Missed class at 9:00 AM."	Justification is rejected (remains an empty string).
"Illness due to @virus!"	Justification is rejected (remains an empty string).
"Absent because of #traffic."	Justification is rejected (remains an empty string).

References

1. Tsui, F, Karam, O, & Bernal, B 2022, *Essentials of Software Engineering*, Jones & Bartlett Learning, LLC, Burlington.
2. Sommerville, I 2015, *Software Engineering*, Global Edition, Pearson Education, Limited, Harlow.
3. Richards, M, & Ford, N 2020, *Fundamentals of Software Architecture : An Engineering Approach*, O'Reilly Media, Incorporated, Sebastopol.
4. Kendall, K, & Kendall, J 2019, *Systems Analysis and Design*, Global Edition, Pearson Education, Limited, Harlow.
5. Pressman, R.S. (2015) *Software engineering: A practitioner's approach*. 8th edn. New York, NY: McGraw-Hill Education.