Vanier College
Continuing Education

06 Analysis Phase

System Development

420-940-VA Sect 87414

Teacher: Jay Patel

Content

- **Introduction to Analysis Phase**

Introduction to Analysis Phase

Purpose:

- The analysis phase is crucial in identifying and documenting the *requirements* of the system, ensuring a clear understanding of what the system is supposed to do.

Key Activities:

- **Gathering Requirements** from stakeholders through interviews, surveys, and observation.
- **Analyzing Requirements** to identify functional and non-functional needs.
- **Modeling the System** using diagrams like Data Flow Diagrams (DFD) , Use Case Diagrams, Sequence Diagrams

What is the Analysis Phase?

The Analysis Phase is a critical part of the *System Development Life Cycle (SDLC)*, where the *requirements* of the system are identified to a greater detail, documented, and analyzed to ensure clarity for future stages.

Importance of the Analysis Phase

- Ensures the system will meet the users' needs.
- Minimizes costly changes in later stages.
- Provides the foundation for *design* and *implementation* phases.

Key Deliverables of the Analysis Phase

- **Software Requirements Specification (SRS)**: Captures all functional and non-functional requirements.
- **User Stories:**A simple, concise description of a feature from the user's perspective.
- **Use Case Diagrams and Descriptions**: Defines user interactions with the system.
- **Data Flow Diagrams (DFD)**: Represents the flow of data within the system.
- **Sequence Diagrams**: Illustrates interactions between system components over time.

Key Takeaways:

- Focuses on defining *what* the system should do and understanding in terms of users/actors.
- Bridges the gap between business needs and technical implementation.
- Analysis acts as a strong foundation for the Implementation phase

Content

- Introduction to Analysis Phase
- **Software Requirement Specification Document**

Introduction to Software Requirements Specification (SRS)

- An **SRS** is a comprehensive document that outlines the **functional and nonfunctional requirements** of a system.
- It serves as a blueprint for development and guides the design, implementation, and testing phases.

Purpose:

- Ensures that all stakeholders have a **common understanding** of what the system will do.
- Acts as a **contract** between the client and development team.
- Provides a basis for **testing** and **validation** to ensure the system meets user needs.

Components of SSRS

- Introduction
- System Overview
- Functional Requirements
- Non-Functional Requirements
- Interfaces
- Assumptions and Dependencies
- Acceptance Criteria

Components of SSRS (1)

1. **Introduction**:
   a. Project overview, goals, and objectives.
   b. Scope of the system (what is included and excluded).
   c. Definitions, acronyms, and abbreviations.
2. **System Overview**:
   a. Describes the overall architecture and high-level functionalities.
   b. Key features and interactions with external systems.

Components of SSRS (2)

3. Functional Requirements:

a. Detailed description of system features and functions.
b. Use cases that define how the system will interact with users.

4. Non-Functional Requirements:

c. Performance, security, usability, reliability, and scalability.
d. Constraints and limitations (e.g., hardware, software, time constraints)

Components of SSRS (3)

5. Interfaces:

- Descriptions of user interfaces (UI), hardware interfaces, software interfaces, and communication interfaces.

6. Assumptions and Dependencies:

- List of assumptions made during the analysis phase.
- Dependencies on other systems or external factors.

7. Acceptance Criteria:

- Defines the conditions the system must meet to be accepted by the client.
- Helps in guiding testing and validation.

Why is SRS Crucial?

- **Common Understanding**:
  - Ensures all stakeholders (clients, developers, testers) are aligned on the system's goals and functions.
- **Reduces Ambiguity**:
  - Clarifies *what* the system should do, eliminating misunderstandings.
  - Avoids **scope creep** by providing clear requirements.
- **Acts as a Reference Point**:
  - Used throughout the SDLC as a **reference document**.
  - Helps developers understand *how* to implement features and testers understand *what* to test.
- **Foundation for Design and Development**:
  - Provides detailed functional and non-functional requirements to guide system design.
  - Basis for creating technical architecture and selecting appropriate technologies.
- **Facilitates Testing**:
  - Requirements in the SRS are turned into test cases for **validation**.
  - Ensures the system meets the client's expectations and performs as required.

Some Examples of SRS

E-Commerce Website SRS:

- **Functional Requirement**: Users should be able to search for products using keywords.
- **Non-Functional Requirement**: The system should handle up to 10,000 concurrent users.
- **Interface**: Integration with a payment gateway for processing online payments.

Some Examples of SRS

Healthcare Management System SRS:

- **Functional Requirement**: The system should allow doctors to view patient history and enter new diagnoses.
- **Non-Functional Requirement**: System availability must be 99.99%.
- **Interface**: Interface with external laboratory systems for test result imports.

Some Examples of SRS

Online Learning Platform SRS:

- **Functional Requirement**: Users should be able to register for courses and track their progress.
- **Non-Functional Requirement**: Course videos should load within 3 seconds under normal internet conditions.
- **Interface**: Integration with video hosting services (e.g., Vimeo, YouTube).

SRS for Student Grading System Software

Lets study the SRS for the grading system software the PDF uploaded in Reading Material Section.

Content

- Introduction to Analysis Phase
- Software Requirement Specification Document
- **User Stories**

Introduction to User Stories

What are User Stories?

- A **User Story** is a simple, concise description of a feature from the perspective of the user.
- It captures **what** the user wants to accomplish and **why** it is important.

Purpose:

- Helps developers understand user needs without technical details.
- Focuses on delivering value to users.

Format:

- *As a [type of user], I want [some goal] so that [some reason].*

Structure of User Stories

- **Role**:
  - Who is the user? (e.g., Faculty, Administrator)
  - *As a [faculty member]*
- **Goal**:
  - What does the user want to achieve? (e.g., Input grades, generate reports)
  - *I want [to input grades]*
- **Benefit**:
  - Why is this important to the user? (e.g., To finalize course evaluations)
  - *So that [I can finalize course evaluations].*

Example of User Stories

Example:

- *As a **faculty member**, I want to **input student grades** so that I can **finalize my course evaluations**.*

- **Key Elements**:
  - **Who**: Faculty member (role)
  - **What**: Input grades (goal)
  - **Why**: Finalize course evaluations (benefit)

Importance of User Stories

Why are User Stories Important?

1. **Focus on User Needs**:
   - Ensures that development prioritizes user experience and value.
2. **Simplifies Communication**:
   - Uses non-technical language to bridge the gap between users and developers.
3. **Encourages Collaboration**:
   - Drives discussions between stakeholders, developers, and testers.
4. **Prioritization**:
   - Allows teams to prioritize features based on user needs and impact.

Writing Effective User Stories

- **Be Specific**: Focus on one specific goal per user story.
- **Keep It Simple**:Use clear and concise language.
- **Make It Testable**: Define what conditions need to be met for the story to be considered "done."
- **Focus on Value**:Highlight how this feature will benefit the user.
- **Use the INVEST Criteria**:
  - **Independent**: Can be developed independently of other stories.
  - **Negotiable**: Can be discussed and changed.
  - **Valuable**: Provides value to the user.
  - **Estimable**: Can be estimated in terms of time/effort.
  - **Small**: Small enough to be completed within a sprint.
  - **Testable**: Can be tested against acceptance criteria.

Acceptance Criteria in User Stories

What are Acceptance Criteria?

- Acceptance criteria are the conditions that must be met for the story to be considered complete.

Purpose:

- Define the boundaries of a user story.
- Provide clear requirements for testing.

Example:

- *As a faculty member, I want to input student grades so that I can finalize my course evaluations.*
- **Acceptance Criteria**:
  - The system must validate grade entries.
  - Faculty must be able to upload grades in bulk via CSV.
  - Grades must be saved in the system for future reporting.

Example User Story with acceptance Criteria

User Story:

- *As a **student**, I want to **view my final grades** so that I can **track my academic performance**.*

Acceptance Criteria:

1. Students can view grades for each course they are enrolled in.
2. Grades must be displayed in a list format, sortable by course.
3. The system must be accessible via desktop and mobile devices.
4. The student can download the grade report in PDF format.

User Story Vs Use Case

User Stories:

- Focus on *what* the user wants to achieve.
- Written in simple, non-technical language.
- Easy to understand and prioritize.

Use Cases:

- Focus on *how* the system interacts with users.
- Provide detailed, step-by-step descriptions of system interactions.
- Used for more technical modeling.

Epics and User Story Mapping

Epics:

- Larger user stories that can be broken down into smaller stories.
- Used to represent broader features that need further refinement.

User Story Mapping:

- A visual technique used to organize user stories.
- Helps visualize the user's journey and prioritize features based on user needs.

Example: Epics and User Story Mapping

Epic:

- *As an **administrator**, I want to **manage the grading system** so that it runs efficiently.*

User Stories:

1. *As an **administrator**, I want to **add new faculty accounts** so that faculty can access the system.*
2. *As an **administrator**, I want to **monitor system performance** so that I can ensure smooth operations.*
3. *As an **administrator**, I want to **generate system usage reports** so that I can track usage metrics.*

How to Prioritize User Stories

Factors to Consider:

- **Value to the User**: Does this feature directly improve user experience?
- **Risk and Uncertainty**: Is this feature technically challenging or uncertain?
- **Dependencies**: Does this feature rely on other stories or system components?
- **Development Effort**: Can this feature be developed within the sprint or needs further breakdown?

Techniques:

- **MoSCoW**: Must have, Should have, Could have, Won't have.
- **Story Points**: Estimating effort based on complexity and time.
- **Backlog Grooming**: Regular review and reprioritization of user stories.

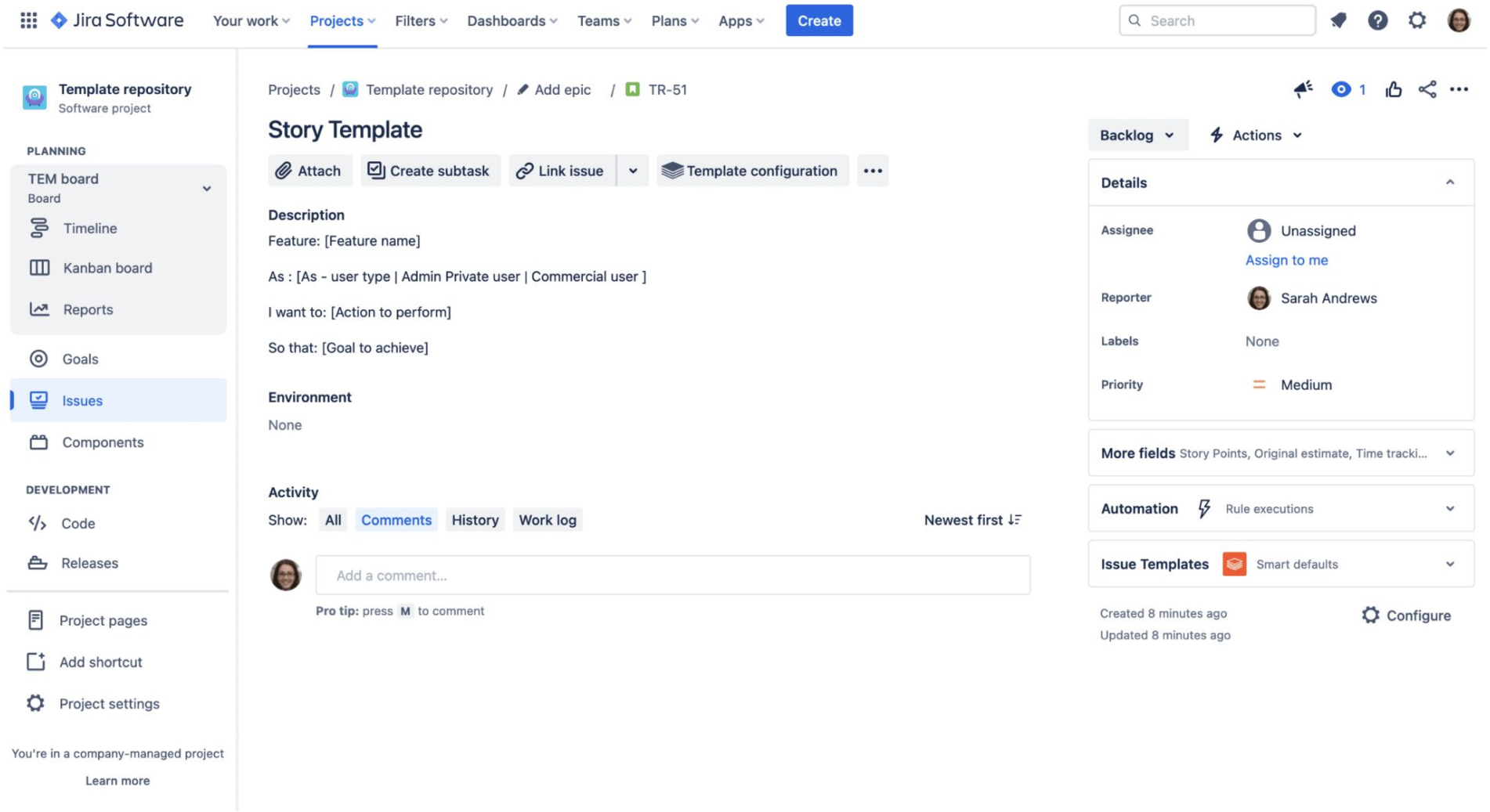
Activity: User Stories

Think about 3 user stories for A Student Grading System.

User Stories

ID	User Story	Priority	Acceptance Criteria	Story Points	Status	Dependencies
US-101	As a faculty member, I want to input grades so that I can finalize course evaluations.	High	System must validate grade entries. Grades must be available for reporting.	5	To Do	US-100: SIS Integration
US-102	As an administrator, I want to generate grade reports so that I can review student performance.	Med	Reports must be customizable by date range and course.	3	In Progress	None
US-103	As a student, I want to view my final grades so that I can track my performance.	Low	Grades must be displayed in a sortable list. Student must be able to download a PDF report.	2	To Do	None

User stories using JIRA



Content

- Introduction to Analysis Phase
- Software Requirement Specification Document
- User Stories
- **Use Case Diagrams and Descriptions**

User Case Diagrams

- **What are Use Case Diagrams?**
  - A **Use Case Diagram** is a visual representation that shows how *users (actors)* interact with a system to achieve specific goals.
  - Describes the *functional requirements* of a system.
  - Focuses on *what* the system should do rather than *how* it does it.
- **Purpose of Use Case Diagrams**:
  - Capture **interactions** between users and the system.
  - Helps stakeholders and developers understand **system functionality**.
  - Serves as a foundation for creating **detailed design** and **test cases**.

Components of a Use Case Diagram

- **Actors**
- **Use Cases**
- **System Boundary**
- **Relationships**

Components of a Use Case Diagram: Actor

Actors:

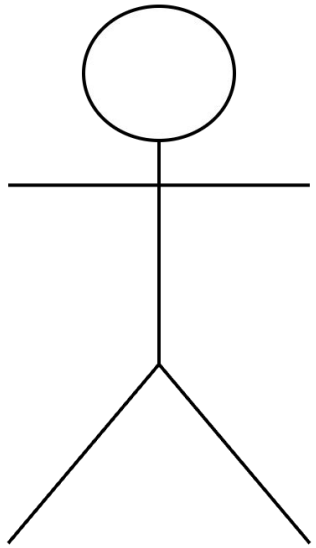
- Represents users or external systems that interact with the system.
- Can be a person, another system, or an organization.
- Example: Faculty, Student, Admin.

Primary Actors:

- Directly interact with the system.
- Initiate use cases.
- Example: Faculty, Student (in a Student Grading System).

Secondary Actors:

- Support the system and primary actors.
- Maintain system configurations or provide assistance.
- Example: Admin, IT Support (in a Student Grading System).

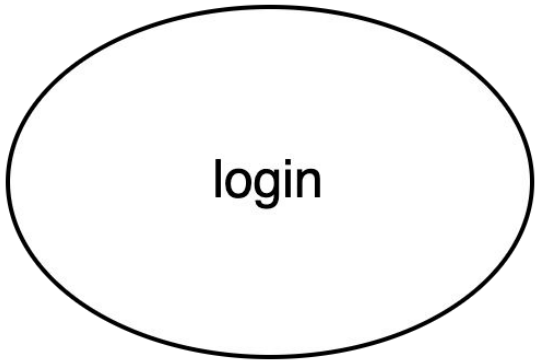


- Primary actors are placed on left of system and Secondary on Right of system

Components of a Use Case Diagram: Use Cases

Use Cases:

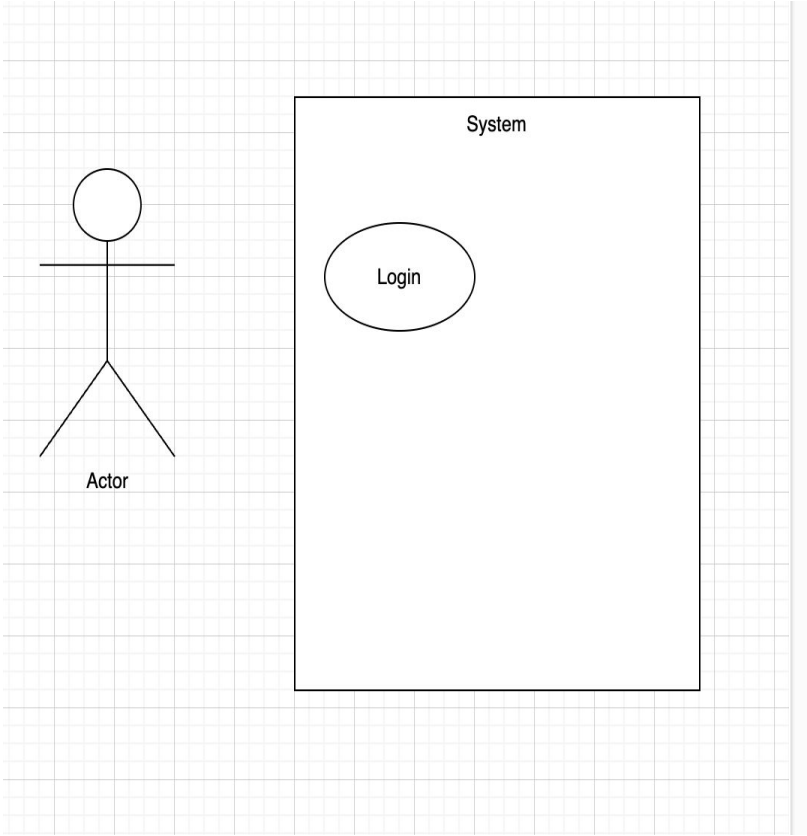
- Specific actions or goals that the system supports for the actors.
- Represented as Oval/Circles with text in middle which indicates the action
- Example: Login, Input Grades, Generate Report, View Grades.



Components of a Use Case Diagram: System

System Boundary:

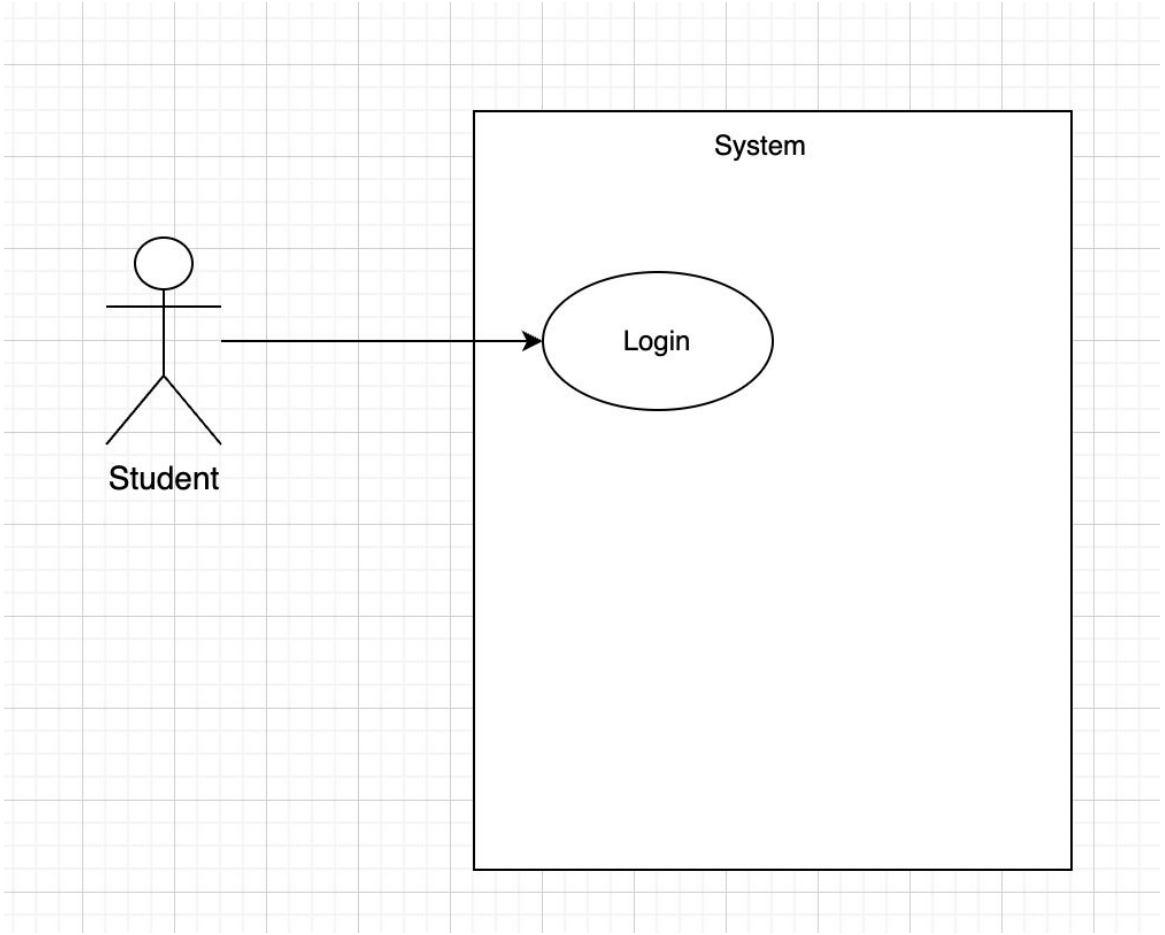
- Defines the *scope* of the system being developed.
- Separates what is included in the system from what is external to it.
- Represented in a Rectangle, Everything inside this is inside a system boundary



Components of a Use Case Diagram: Relationships

Association

- The most basic relationship between an actor and a use case.
- Example: Student **associates** with the “Login” use case.
- Represented by a solid arrow

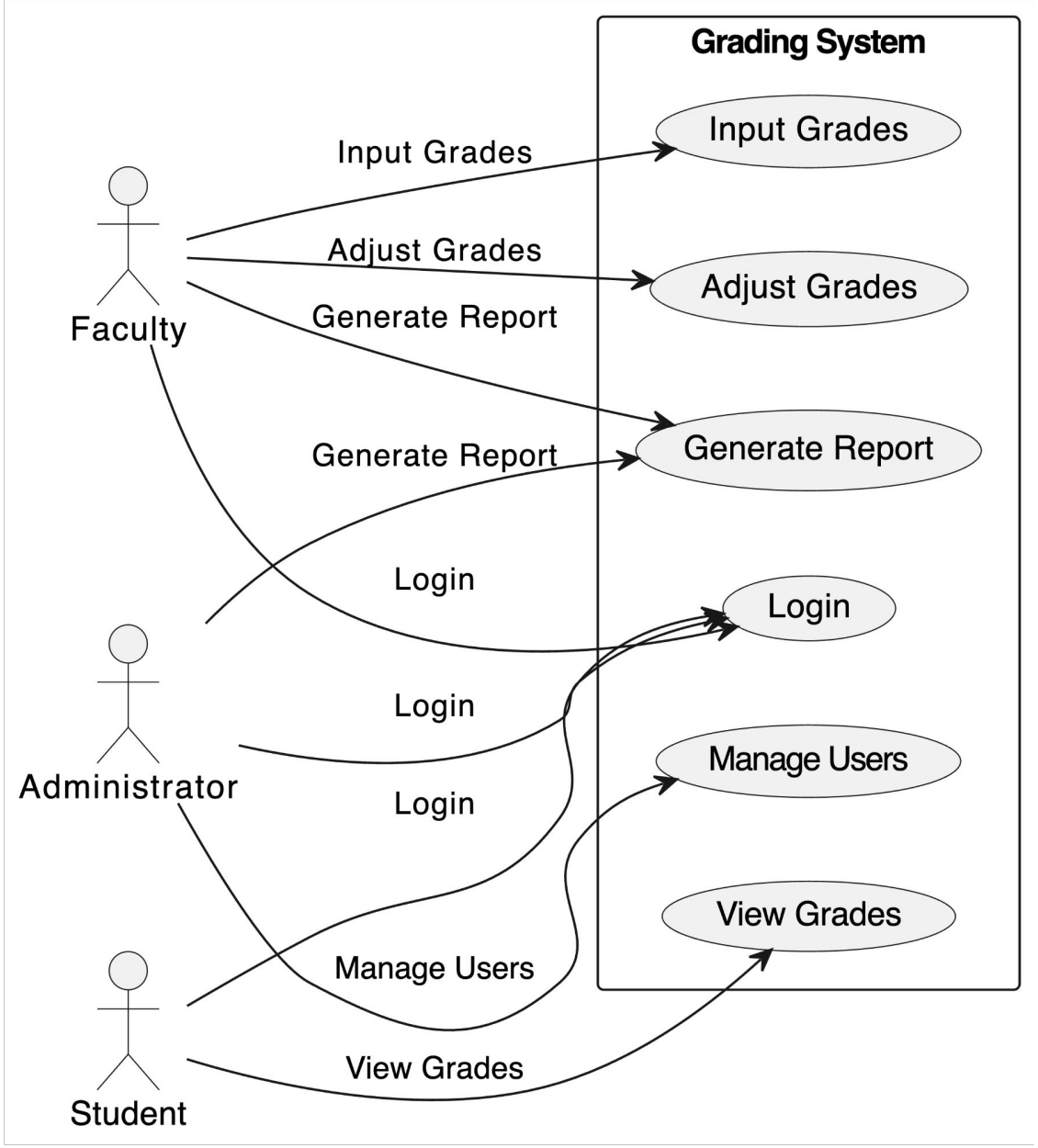


Grading System Use Case Diagram

- **Actors**:
  - Faculty, Student, Admin
- **Use Cases**:
  - Input Grades, Adjust Grades, Generate Report, View Grades
- **Relationships (Association)**:
  - **Faculty** interacts with:
    - Login, Input Grades, Adjust Grades, Generate Report
  - **Administrator** interacts with:
    - Login, Generate Report, Manage Users
  - **Student** interacts with:
    - Login, View Grades

Grading System Use Case Diagram

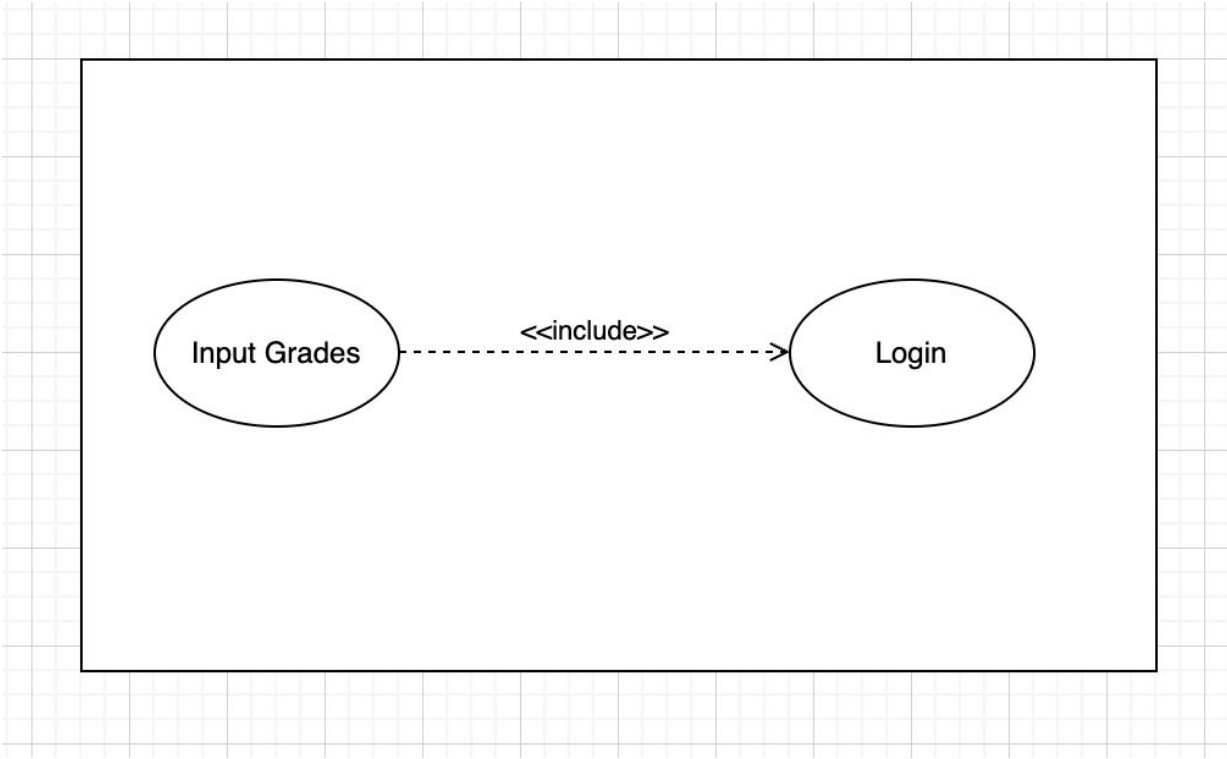
- **Actors**:
  - Faculty, Student, Admin
- **Use Cases**:
  - Input Grades, Adjust Grades, Generate Report, View Grades
- **Relationships (Association)**:
  - **Faculty** interacts with:
    - Login, Input Grades, Adjust Grades, Generate Report
  - **Administrator** interacts with:
    - Login, Generate Report, Manage Users
  - **Student** interacts with:
    - Login, View Grades



Components of a Use Case Diagram: Relationships

Include Relationship

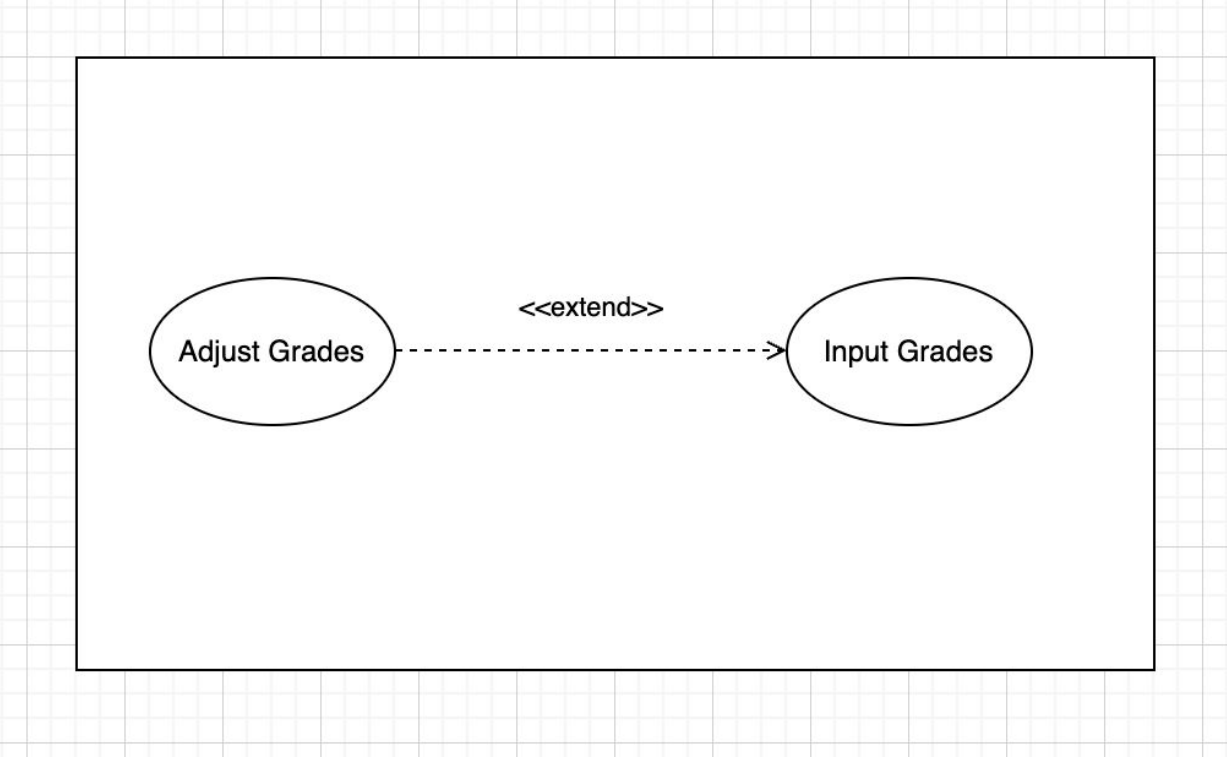
- Indicates that a use case **includes** the behavior of another use case. The included use case is mandatory and is always executed as part of the primary use case.
- To **reuse common functionality** that multiple use cases share.
- In the **Student Grading System**, The **Input Grades** use case includes the **Login** use case, as faculty must log in before inputting grades.
- When Use Case **A** includes Use Case **B**, it means that **B** is always executed as part of **A**.
- Use Case **B** must be performed either **before** or **during** Use Case **A**. The **Include** relationship is used to factor out common behavior shared by multiple use cases.
- **Example**: If **"Input Grades"** includes **"Login"**, it means that **Login** must be completed before the user can proceed with **Input Grades**. It is mandatory for **A** (Input Grades) to include **B** (Login).



Components of a Use Case Diagram: Relationships

Exclude Relationship

- Indicates that a use case can **extend** another use case, providing additional functionality that is not always executed. The extended use case happens under specific conditions.
- **Purpose**: To model **optional or conditional functionality**.
- **Example**: In the **Student Grading System**, the **Adjust Grades** use case might extend the **Input Grades** use case, as adjusting grades is an optional feature that faculty members use when necessary.
- **Meaning**: When Use Case **A** extends Use Case **B**, it means that **A** provides optional or additional functionality to **B** that happens only under certain conditions. **B** can be executed without **A**, but **A** adds more behavior if needed.
- **Order**: Use Case **A** (the extending case) only happens after or during **B** under specific conditions.
- **Example**: If **"Adjust Grades"** extends **"Input Grades"**, it means that after inputting grades, **Adjust Grades** may be invoked **if** adjustments are needed. It's conditional and optional.



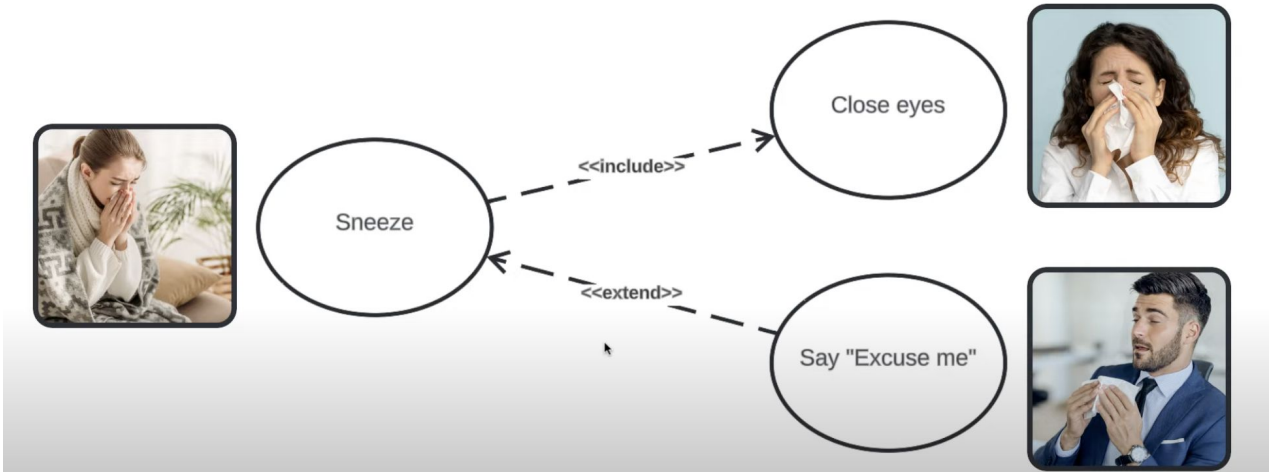
Include and Extend

<<include>> Relationship (Close Eyes):

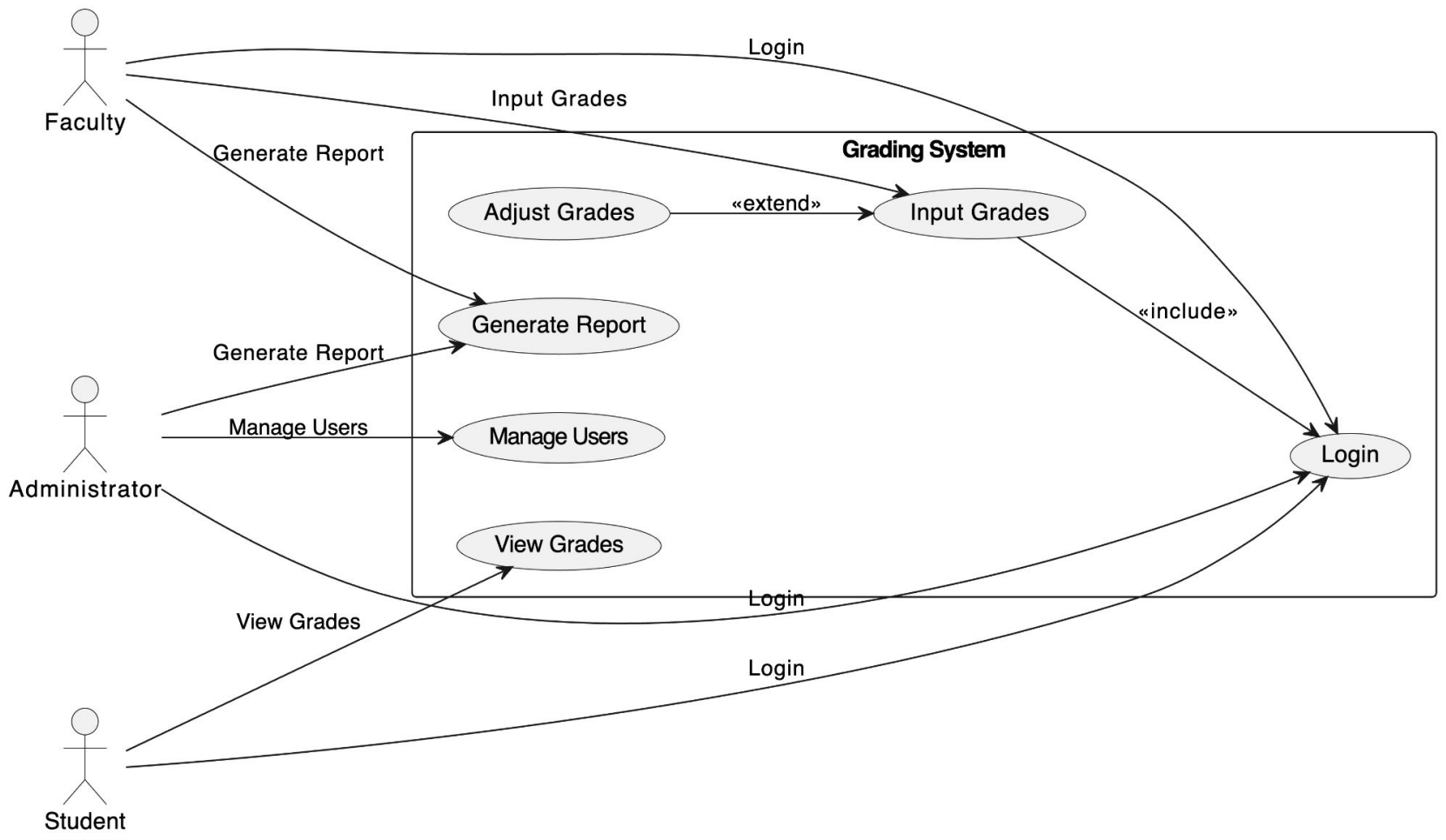
- This relationship indicates that "Close Eyes" is a part of the "Sneeze" use case and always occurs when someone sneezes. It is an integral part of the sneezing action, suggesting that every time the "Sneeze" use case is executed, the "Close Eyes" use case is also executed as a mandatory step.

<<extend>> Relationship (Say 'Excuse me'):

- This relationship shows that saying "Excuse me" is an optional extension of the "Sneeze" use case. It occurs only under certain conditions, perhaps based on social norms or the presence of others. It's not a mandatory part of sneezing, but it can extend the sneezing use case when the conditions are met.



Use Case Diagram with <<include>> and <<extend>>



Use Case Description

- **What is a Use Case Description?**
  - A detailed explanation of the steps involved in a particular use case.
  - Describes the sequence of actions, preconditions, and outcomes.
- **Key Elements**:
  1. **Use Case Name**: A short, descriptive title.
  2. **Actors**: Who is involved in this use case?
  3. **Preconditions**: What must be true for the use case to begin?
  4. **Main Flow**: Step-by-step interactions between the actor and the system.
  5. **Postconditions**: What happens after the use case completes?
  6. **Alternative Flow**: Any deviations from the main flow.

Use Case Description: Examples

Use Case: Login

- **Primary Actor**: Faculty, Administrator, Student
- **Preconditions**: The actor must have valid credentials.
- **Main Flow**:
  - The actor navigates to the login page.
  - The actor enters their username and password.
  - The system verifies the credentials.
  - If the credentials are valid, the system grants access and redirects the actor to their respective dashboard.
- **Postconditions**: The actor is logged into the system.
- **Alternate Flows**:
  - If credentials are invalid, the system displays an error message and prompts for re-entry.

Use Case Description: Examples

Use Case: Input Grades

- **Primary Actor**: Faculty
- **Preconditions**: Faculty must be logged in and assigned to at least one course.
- **Main Flow**:
  - Faculty selects a course from their course list.
  - Faculty enters or uploads grades for students.
  - The system validates the entered grades and saves them.
  - Confirmation of grade input is displayed to the faculty.
- **Postconditions**: Grades are updated in the system.
- **Alternate Flows**:
  - If grade validation fails (e.g., due to format errors), the system notifies the faculty and requests correction.

Benefits of Use Case Diagrams

Clarifies Functional Requirements:

- Ensures a shared understanding of what the system should do from a user's perspective.

Improves Communication:

- Provides a visual representation, making it easier for stakeholders and developers to communicate.

Supports Testing:

- Each use case can be turned into test cases to ensure that the system meets the required functionality.

Foundation for System Design:

- Helps transition from functional requirements to system design by outlining key system interactions.

Use Case Diagrams Vs User Story

Use Case Diagrams:

- Focus on the *how* and *who*.
- Shows specific interactions between users and the system.
- Visual, diagram-based.

User Stories:

- Focus on the *what* and *why*.
- Describes the user's goals in natural language.
- Narrative, text-based.

Content

- Introduction to Analysis Phase
- Software Requirement Specification Document
- User Stories
- Use Case Diagrams and Descriptions
- **Data Flow Diagrams**

Introduction to Data Flow Diagrams

What are Data Flow Diagrams?

- Data Flow Diagrams (DFDs) visually represent the flow of data within a system.
- They illustrate how data moves between processes, data stores, and external entities.

Purpose of DFDs:

- To provide a clear, visual representation of how data is processed and stored in a system.
- To help in the analysis, design, and optimization of data systems.

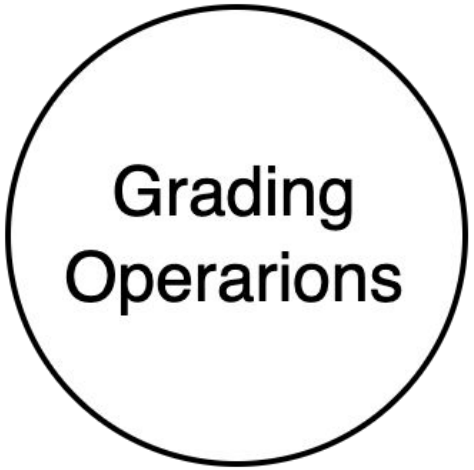
Components of Data Flow Diagrams

- **Processes**
- **Data Stores**
- **External Entities**
- **Data Flows**

Components of Data Flow Diagrams

Processes:

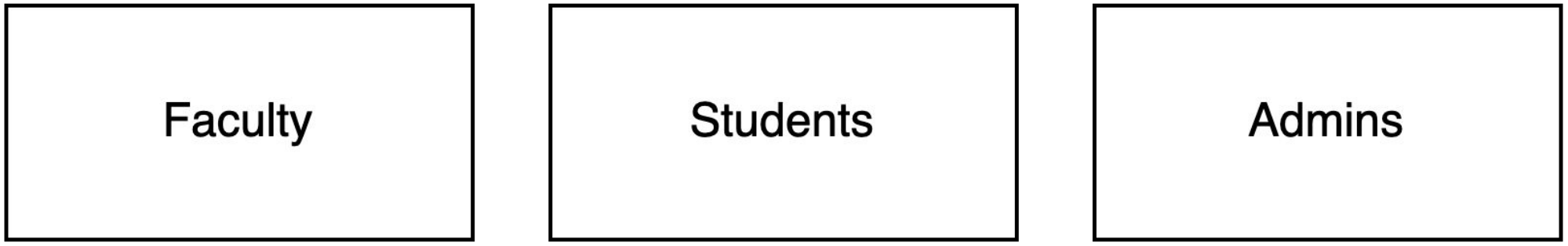
- Represent tasks or functions within the system that transform incoming data flows into outgoing data flows.
- Depicted as Circles
- **Example**: The **main process** in our project can be `Grading Operations` which is represented as a rectangle and acts as the central processing unit within the system.



Components of Data Flow Diagrams

External Entities:

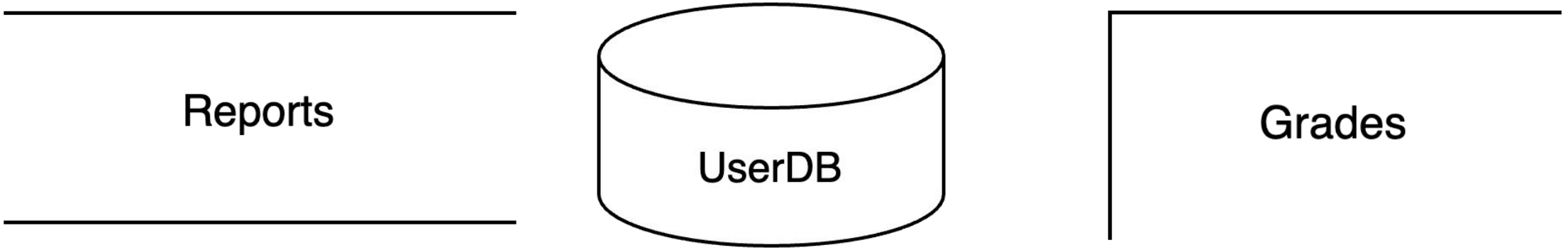
- Objects outside the system with which system communicates
- Sources or destinations of data outside the system boundaries.
- Shown as rectangles or rounded rectangles



Components of Data Flow Diagrams

Data Stores:

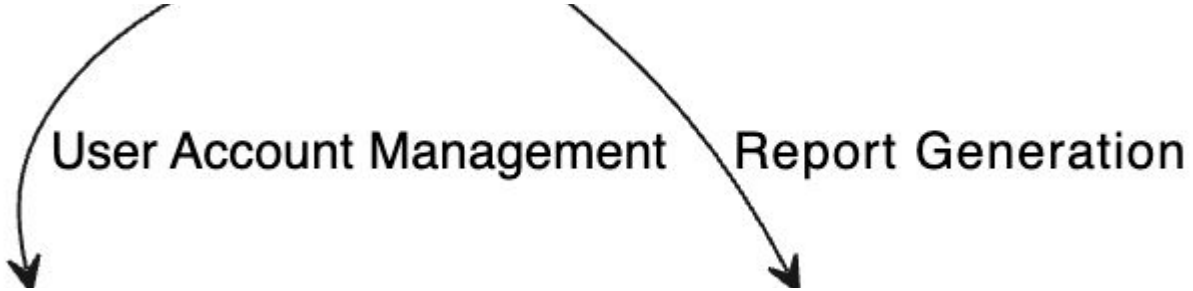
- Places where data is stored within the system.
- Represented by open-ended rectangles or parallel lines or cylinders
- **Example**: In our Project, Data Stores can be **Reports, UserDB, Grades**
- All of the following representations are correct



Components of Data Flow Diagrams

Data Flows

- The movement of data between processes, data stores, and external entities.
- Represented by arrows.



Components of Data Flow Diagrams

- **Processes**:
  a. Represent tasks or functions within the system that transform incoming data flows into outgoing data flows.
  b. Depicted as circles or rounded rectangles.
- **Data Stores**:
  a. Places where data is stored within the system.
  b. Represented by open-ended rectangles or parallel lines.
- **External Entities**:
  a. Sources or destinations of data outside the system boundaries.
  b. Shown as rectangles.
- **Data Flows**:
  a. The movement of data between processes, data stores, and external entities.
  b. Represented by arrows.

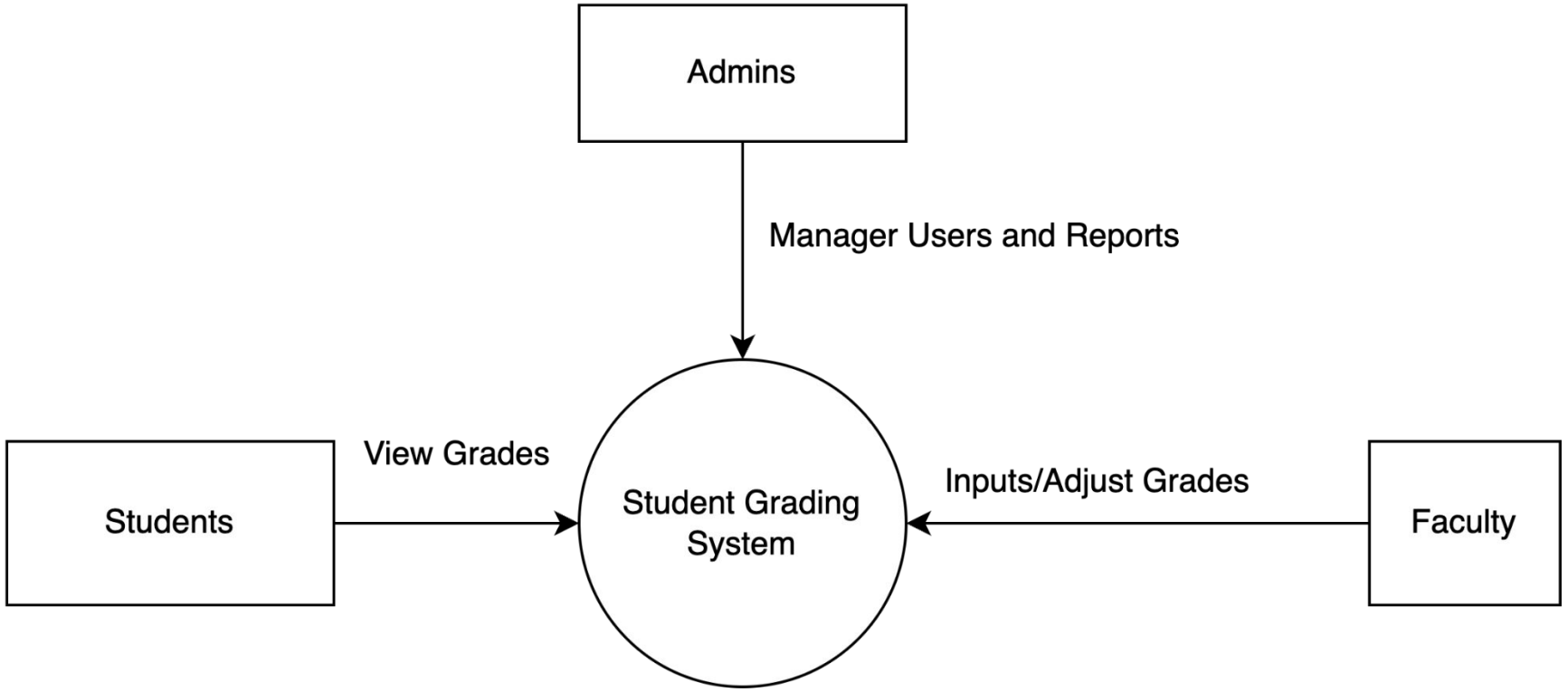
Levels of Data Flow Diagrams

- **Context Diagram**:
  a. The highest level of a DFD and contains only one process.
  b. Shows the system as a whole and its interactions with external entities.
- **Level 0 Diagram (The System Diagram)**:
  a. Expands the single process node from the context diagram into multiple processes.
  b. Provides an overview of the major functional areas of the system.
- **Level 1 Diagram**:
  a. Breaks down the processes from Level 0 into more detailed sub-processes.
  b. Shows more detailed flows of information within the system.
- **Level 2 Diagram and Beyond**:
  a. Further decomposes the processes into more specific, lower-level processes.
  b. Details are increasingly granular and specific to parts of the system.

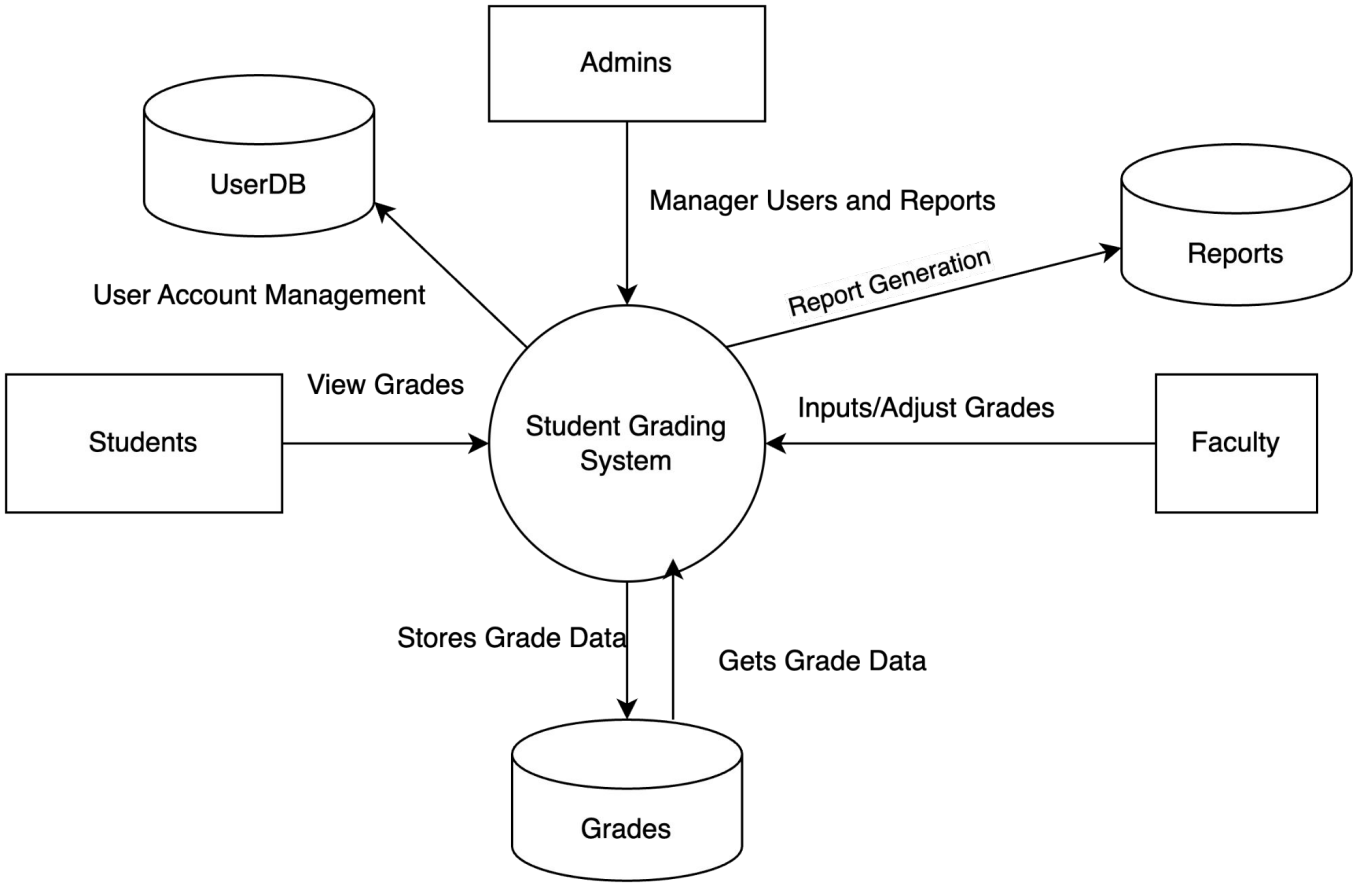
Levels of Data Flow Diagrams

- **Context Diagram**:
  a. The highest level of a DFD and contains only one process.
  b. Shows the system as a whole and its interactions with external entities.
- **Level 0 Diagram (The System Diagram)**:
  a. Expands the single process node from the context diagram into multiple processes.
  b. Provides an overview of the major functional areas of the system.
- **Level 1 Diagram**:
  a. Breaks down the processes from Level 0 into more detailed sub-processes.
  b. Shows more detailed flows of information within the system.
- **Level 2 Diagram and Beyond**:
  a. Further decomposes the processes into more specific, lower-level processes.
  b. Details are increasingly granular and specific to parts of the system.

Context Diagram



Level 0 Diagram



Level 1 DFD

A Level 1 Data Flow Diagram (DFD) expands upon the Level 0 diagram by breaking down each of the main processes into more detailed sub-processes. This allows for a deeper exploration of the system's functionality, showing how specific tasks are performed within each broader process. In the context of the Student Grading System, a Level 1 DFD would detail the internal workings of managing grades, reports, and user accounts.

Breaking Down the Processes into Level 1:

For the Student Grading System, let's detail the sub-processes for each main process identified in the Level 0 diagram:

1. **Process 1: Manage Grades**
    - **1.1 Enter Grades**: Faculty inputs grades into the system.
    - **1.2 Adjust Grades**: Faculty makes adjustments to existing grades.
    - **1.3 Calculate Final Grades**: System calculates final grades based on input and adjustments.
    - **1.4 Distribute Grades**: System makes grades available for students to view.

Level 1 DFD

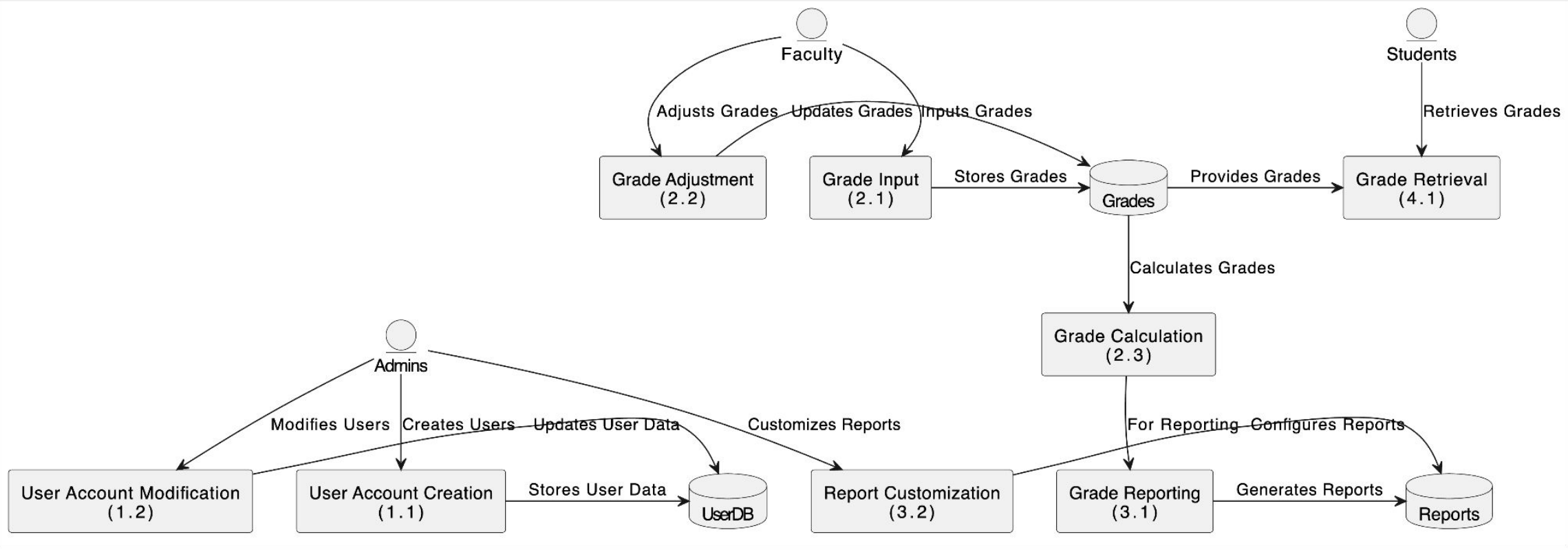
Process 2: Manage Reports

- **2.1 Generate Class Reports**: System compiles grade reports for different classes.
- **2.2 Generate Student Performance Reports**: System creates detailed reports on individual student performance.
- **2.3 Distribute Reports**: System provides reports to administrators and faculty.

Process 3: User Management

- **3.1 Create User Accounts**: Administrators add new user accounts into the system.
- **3.2 Update User Information**: Administrators update existing user account details.
- **3.3 Delete User Accounts**: Administrators remove user accounts from the system.
- **3.4 Handle Security**: System handles login security, password resets, and data protection.

Level 1 DFD



Content

- Introduction to Analysis Phase
- Software Requirement Specification Document
- User Stories
- Use Case Diagrams and Descriptions
- Data Flow Diagrams
- **Sequence Diagrams**

Introduction to Sequence Diagrams

Introduction to Sequence Diagrams

- **Definition**: Sequence diagrams show how objects interact in a given system over time. They are particularly useful for modeling the dynamic aspects of software systems.
- **Purpose**: To depict the order of message flows between participants (objects or processes) in a scenario.

Introduction to Sequence Diagrams

Introduction to Sequence Diagrams

- **Definition**: Sequence diagrams show how objects interact in a given system over time. They are particularly useful for modeling the dynamic aspects of software systems.
- **Purpose**: To depict the order of message flows between participants (objects or processes) in a scenario.

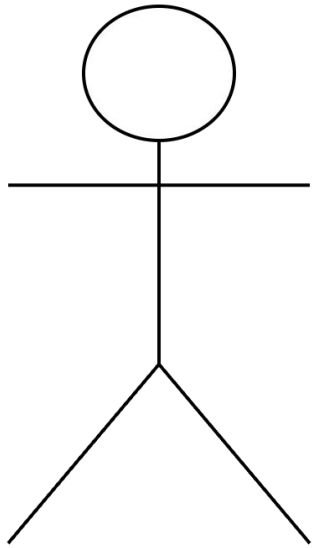
Components of Sequence Diagrams

- **Actors**
- **Objects**
- **Messages**
- **Alt Blocks**
- **Activation Bars**

Components of Sequence Diagrams: Actor

Actors:

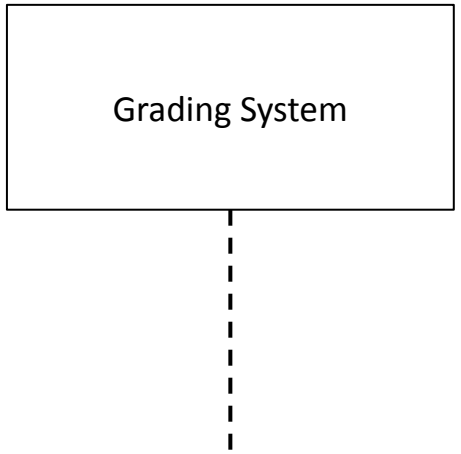
- Represents users or external systems that interact with the system.
- Can be a person, another system, or an organization.
- Example: Faculty, Student, Admin



Components of Sequence Diagrams: Objects

Objects and Lifelines

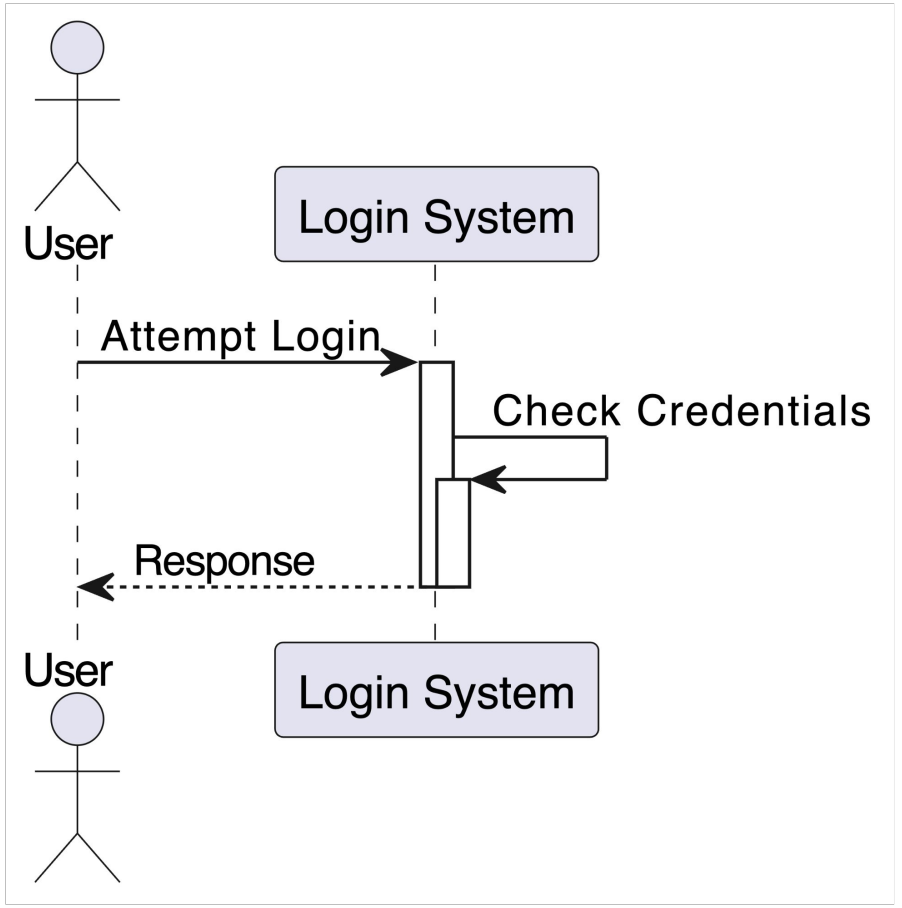
- Represented as rectangles
- A vertical dashed line is called **lifeline** which indicates the object's presence over time.
- Example, In Banking System, ATM, Bank Servers and Bank Can be objects
- And In our System, Grading System as of whole acts as an object



Components of Sequence Diagrams: Activation Bars

Activation Bars

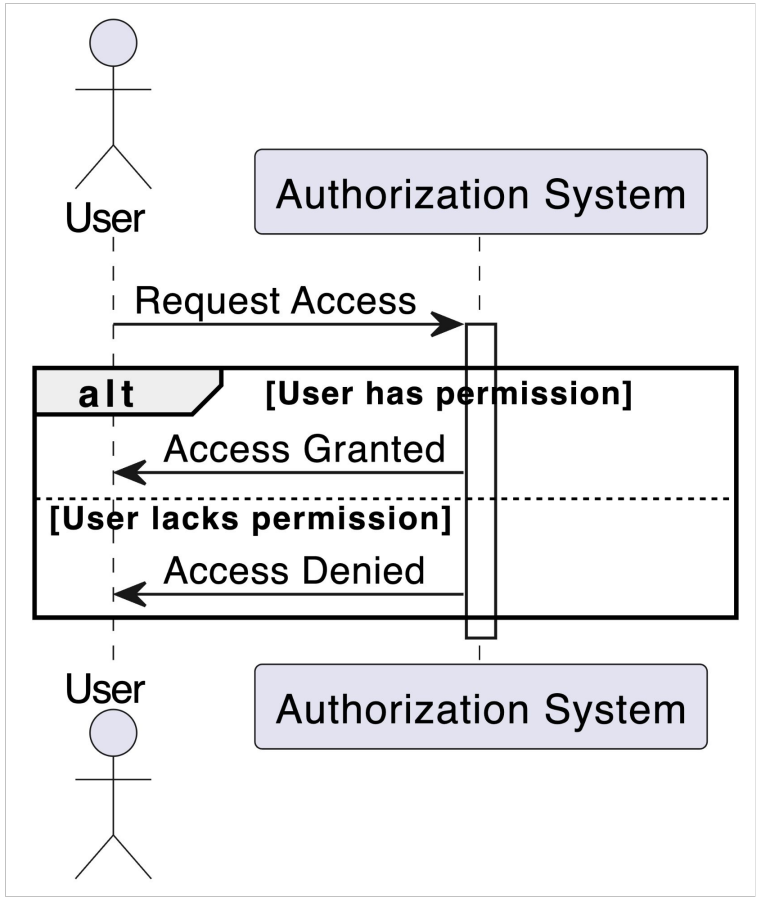
- The activation bar is the box placed on the lifeline.
- It is used to indicate that an object is active (or instantiated) during an interaction between two objects. The length of the rectangle indicates the duration of the objects staying active.



Components of Sequence Diagrams: Alt Blocks

Alt Blocks

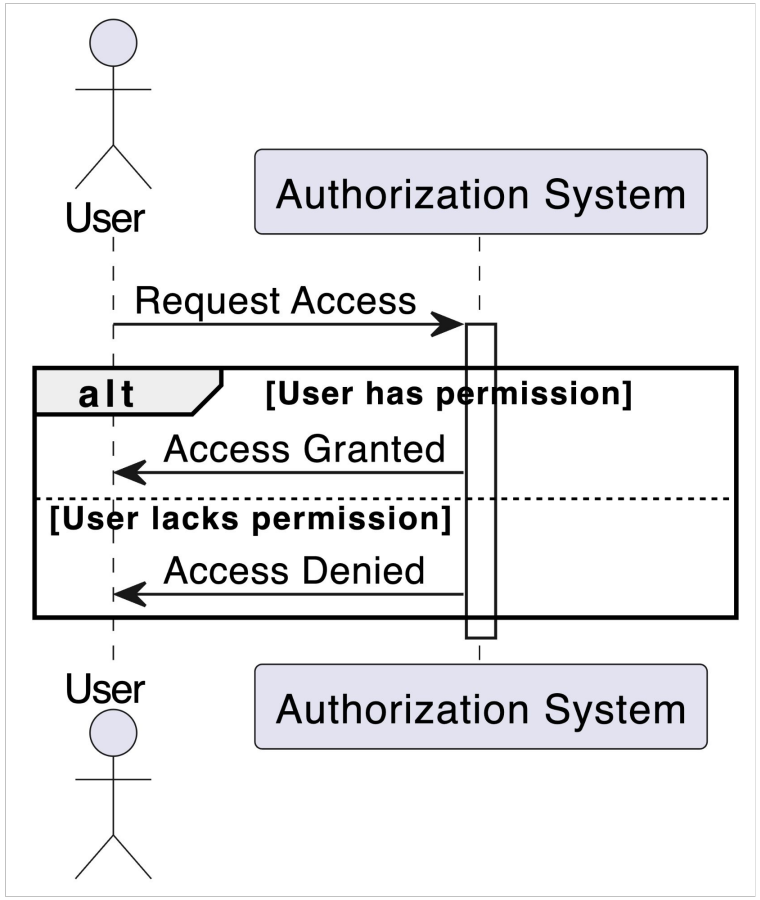
- **alt**: This marks the beginning of a conditional block.
- Inside the block, you can define multiple sections.
- Each section represents a different branch of execution based on different conditions.



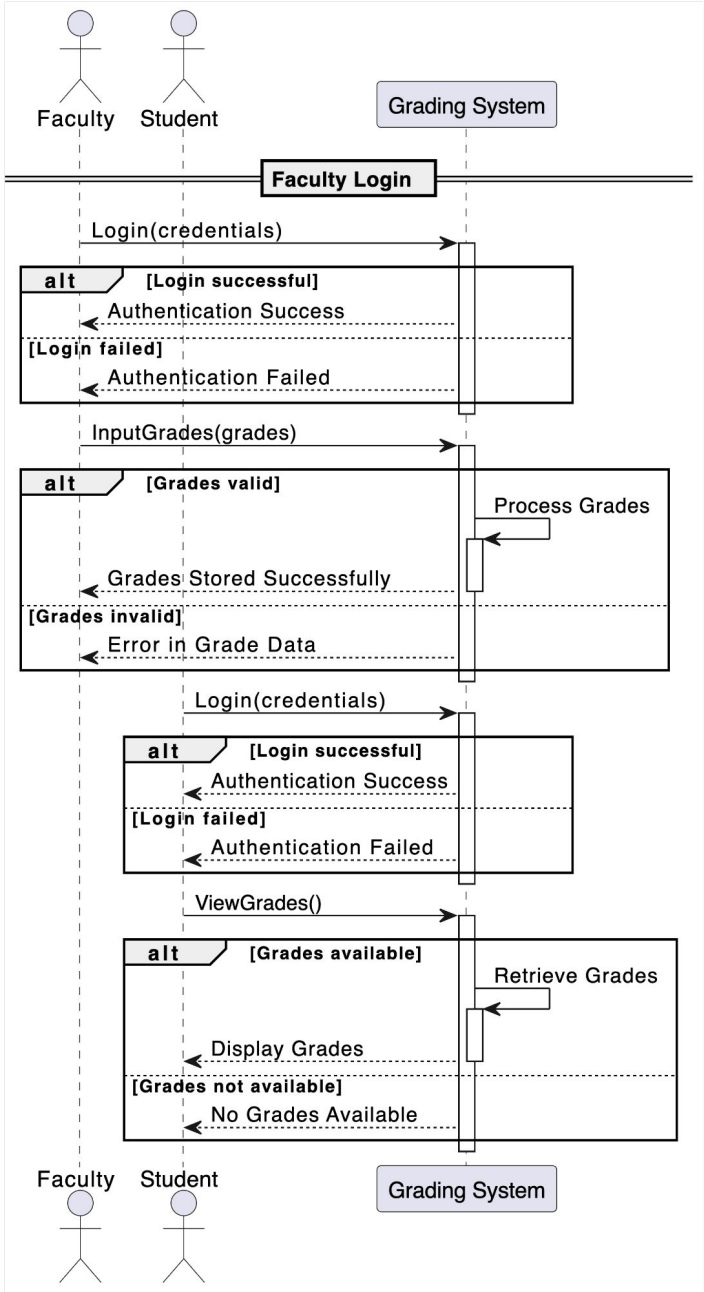
Components of Sequence Diagrams: Messages

Messages:

- The calls and responses between the actors and the system.
- Example: messages logins, grade input, and grade retrieval.
- In given example “Request Access”, “Access Granted” are messages



Example of a Sequence Diagram for our Student Grading System



Sequence Diagrams: Uses

- **Visualization**: This diagram can be used to visualize the interaction flow for typical use cases in the grading system. It helps in understanding the sequence of actions and responses.
- **Debugging and Development**: Developers can use this sequence diagram to ensure that all necessary interactions are handled correctly and to aid in debugging the flow of data.
- **Documentation**: This can serve as documentation to help new project members understand the system's functionality more quickly.

Revision

Revision Slide 1: Overview of the Analysis Phase

Purpose: To understand and document the system’s functional and non-functional requirements.

- **Main Deliverables**:
  - **Software Requirements Specification (SRS)**
  - **User Stories**
  - **Use Case Diagrams**
  - **Data Flow Diagrams (DFD)**
  - **Sequence Diagrams**
- **Goal**: Bridge the gap between stakeholders' needs and the system design.

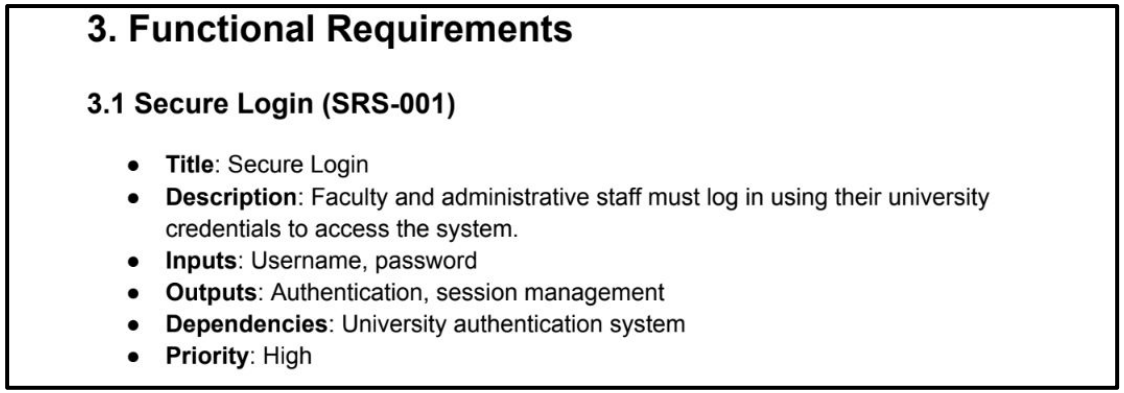
Revision Slide 2: SRS

What is SRS?: A detailed description of the system's functionalities, performance, and constraints.

Key Components:

- Functional Requirements
- Non-Functional Requirements
- Interfaces
- Assumptions and Dependencies
- Acceptance Criteria

SRS Document



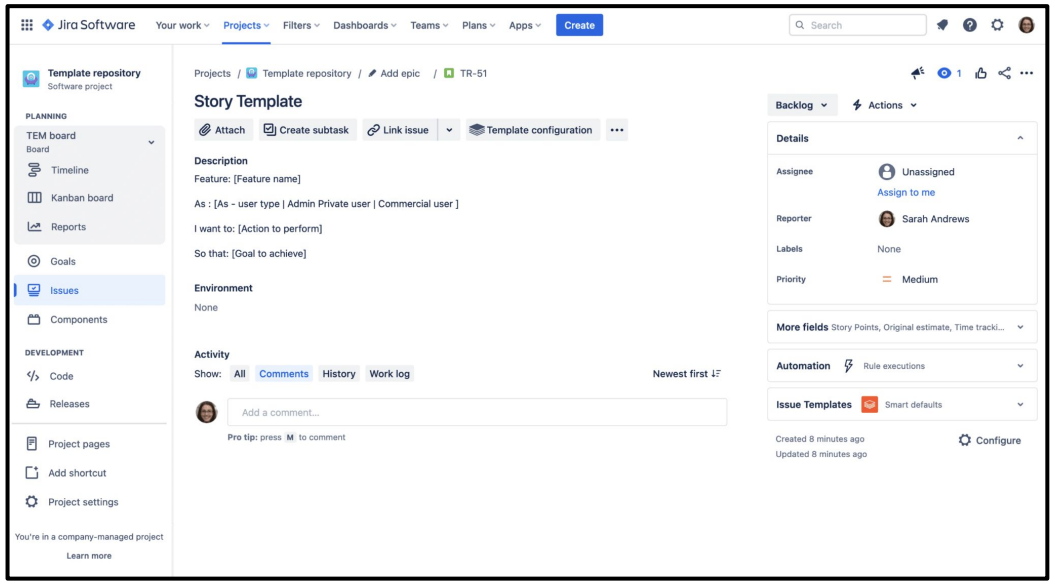
3. Functional Requirements

3.1 Secure Login (SRS-001)

- **Title**: Secure Login
- **Description**: Faculty and administrative staff must log in using their university credentials to access the system.
- **Inputs**: Username, password
- **Outputs**: Authentication, session management
- **Dependencies**: University authentication system
- **Priority**: High

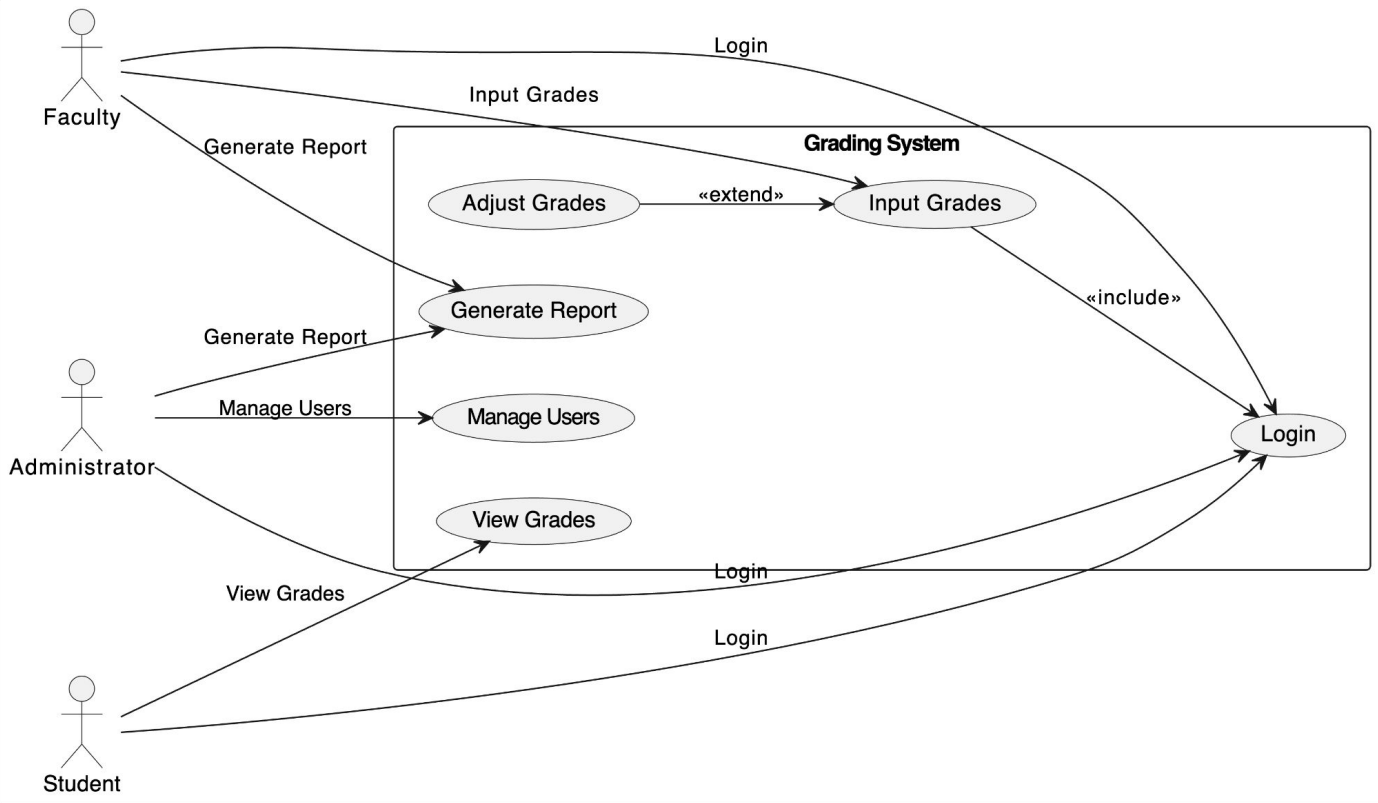
Revision Slide 3: User Stories

- **Definition**: A simple, concise description of a feature from the user's perspective.
- **Structure**:
  - **As a [type of user]**, I want [an action], so that [a benefit].
- **Purpose**: Focus on user needs and outcomes.
- **Tips for Writing**:
  - Be clear, concise, and customer-focused.
  - Include acceptance criteria for validation.



Revision Slide 4: Use Case Diagrams

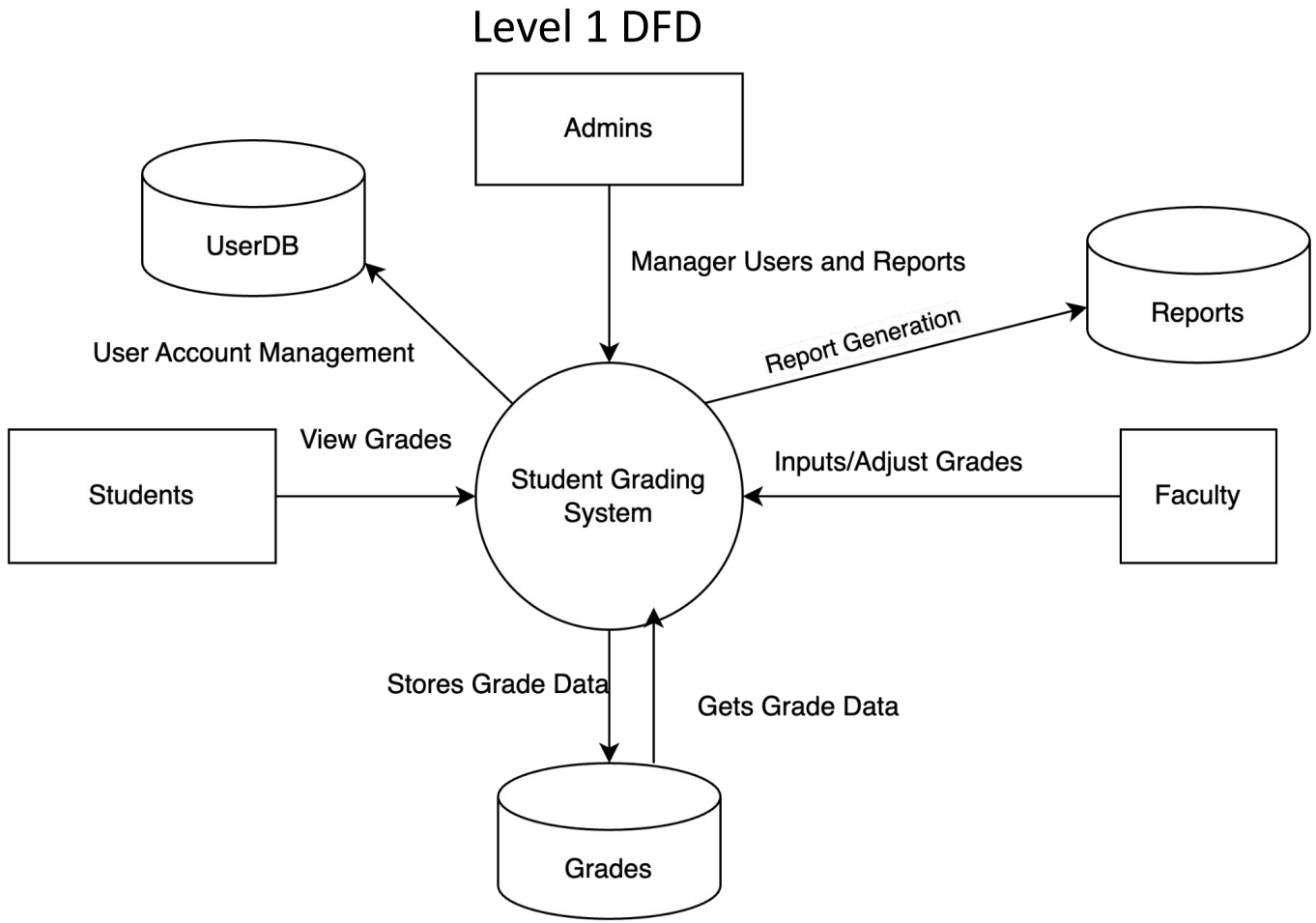
- **Definition**: Visual representation of user interactions with the system.
- **Components**:
  - **Actors**: External entities (users/systems) interacting with the system.
  - **Use Cases**: The system functions or services.
  - **Relationships**: Connections between actors and use cases.
- **Purpose**: Model functional requirements, helping identify all interactions between users and the system.



Revision Slide 5: Data Flow Diagrams

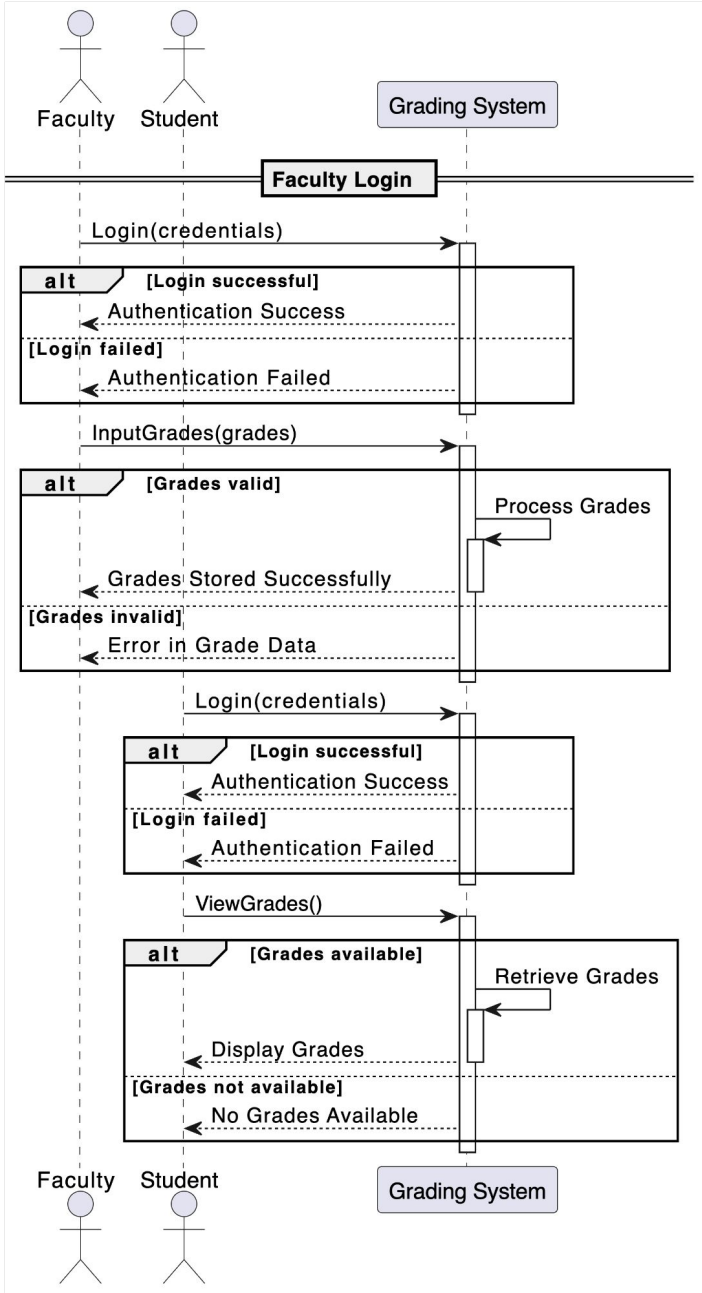
Definition: Diagrams that show how data flows through a system.

- **Levels**:
  - **Context Diagram**: High-level, one process, external entities.
  - **Level 0**: Breakdown into sub-processes.
  - **Level 1+**: Further decomposition of processes.
- **Components**:
  - **Processes**: Circles or ovals representing actions.
  - **Data Stores**: Repositories of data (cylinders).
  - **Data Flows**: Arrows showing data movement.
  - **External Entities**: Actors interacting with the system. Represented in Rectangle



Revision Slide 6: Sequence Diagrams

- **Definition**: Diagrams that represent interactions between objects in a time-ordered sequence.
- **Components**:
  - **Actors/Objects**: Entities participating in the interaction.
  - **Lifelines**: Vertical lines representing the life of an object.
  - **Messages**: Arrows showing communication between objects.
  - **Activation Bars**: Periods during which an object is active.
- **Purpose**: Show the order and timing of interactions.



Revision Slide 6: Key Takeaways from the Analysis Phase

- **Importance**:
  - Ensures a clear understanding of user requirements.
  - Establishes the foundation for system design and development.
- **Deliverables**:
  - Detailed SRS document.
  - Functional models: Use Case Diagrams, DFDs, ERDs.
  - Descriptions of user interactions: User Stories, Sequence Diagrams.
- **Impact**: Effective analysis reduces project risk by ensuring the system meets user needs and expectations.

References

General References:

- **Ian Sommerville, "Software Engineering" (10th Edition)** – Comprehensive guide on software engineering phases, including analysis.
- **Roger S. Pressman, "Software Engineering: A Practitioner's Approach" (8th Edition)** – Focus on requirements analysis, diagrams, and modeling techniques.
- **Object Management Group (OMG) UML Specification** – Official standard for creating UML diagrams.
  - Website: https://www.omg.org/spec/UML
- **Karl E. Wiegers & Joy Beatty, "Software Requirements" (3rd Edition)** – Detailed focus on capturing and managing software requirements.
- **IEEE Std 830-1998** – Industry standard for writing Software Requirements Specifications (SRS).

Specific Websites/Video:

- **Creately Sequence Diagram Tutorial** – Detailed guide on sequence diagrams and activation bars.
  - Website: https://creately.com/guides/sequence-diagram-tutorial
- **IBM Developer Sequence Diagram Tutorial** – Practical insights on sequence diagrams for developers.
  - Website: https://developer.ibm.com/articles/the-sequence-diagram/
- **YouTube Video: Sequence Diagram Tutorial** – Visual walkthrough of creating sequence diagrams.
  - YouTube: https://www.youtube.com/watch?v=4emxjxonNRI

Thank You!