
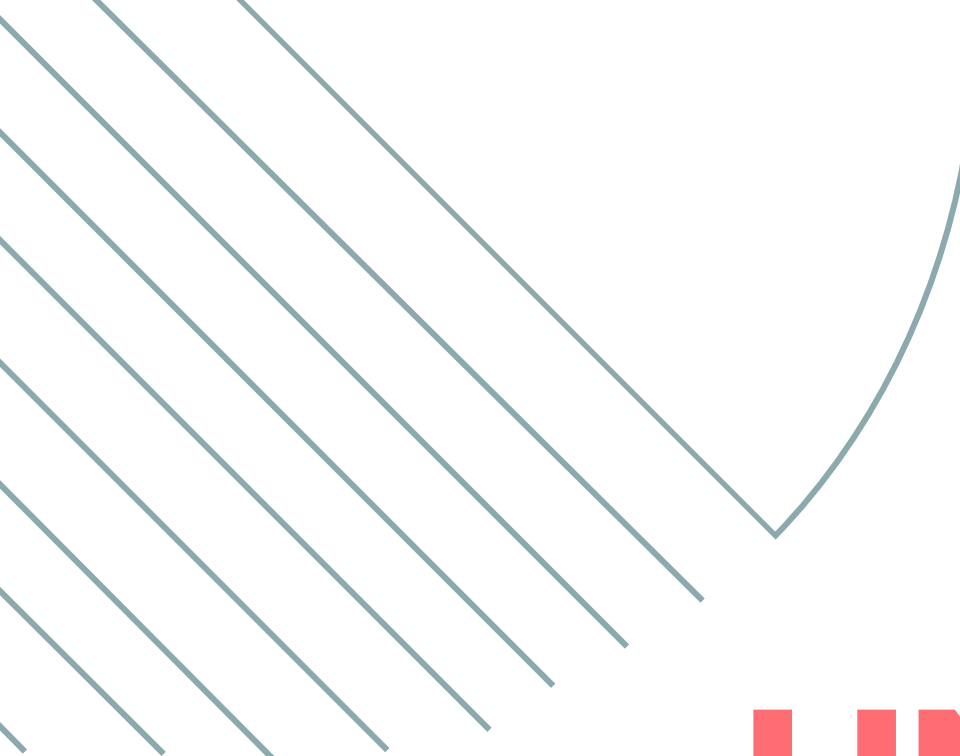




# UNDERSTANDING UML

User Diagrams, Class Diagrams,  
and Sequence Diagrams

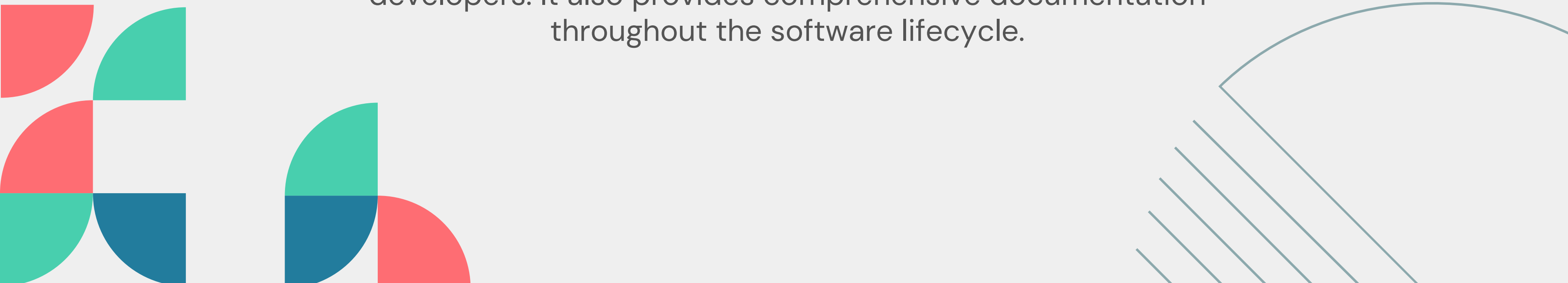


# UNIFIED MODELING LANGUAGE (UML)



# IMPORTANCE

Unified Modeling Language (UML) plays a crucial role in planning and communication among stakeholders, UI/UX designers, and developers. It also provides comprehensive documentation throughout the software lifecycle.



## CASE DIAGRAM

A use case diagram is a type of behavioral diagram defined by the (UML) that represents the functionality of a system from a user's perspective. It illustrates the system's interactions with external users (actors) and the key functions (use cases) the system provides.

## CLASS DIAGRAMS

Class diagrams are a type of static structure diagram in UML that describe the structure of a system by showing its classes, attributes, operations, and the relationships among the classes.

## SEQUENCE DIAGRAMS

Sequence diagrams are a type of interaction diagram in UML that show how objects interact in a given scenario of a use case. They capture the sequence of messages exchanged between objects and the order in which these interactions occur over time.



# CASE STUDY BACKGROUND

## **User Management API:**

User authentication, registration, profile management, and role-based access control.

## **Content Management API:**

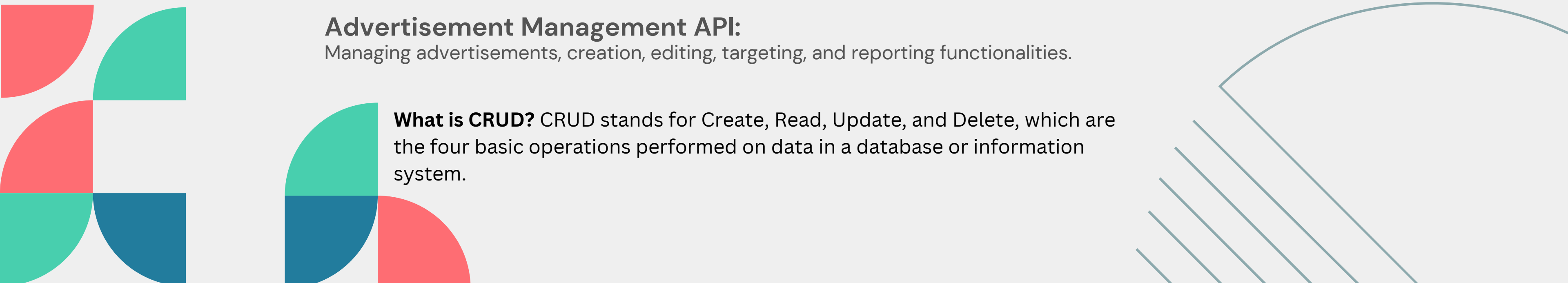
CRUD operations for content, support for rich text, media attachments, and tagging.

## **Media Management API:**

Uploading, managing, and serving media files, media storage, retrieval, and optimization.

## **Advertisement Management API:**

Managing advertisements, creation, editing, targeting, and reporting functionalities.



**What is CRUD?** CRUD stands for Create, Read, Update, and Delete, which are the four basic operations performed on data in a database or information system.



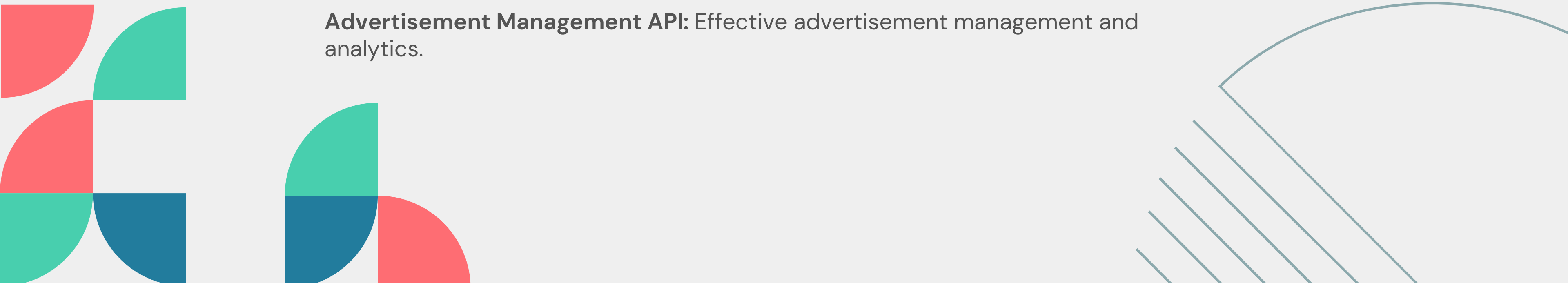
# ACCEPTANCE CRITERIA

**User Management API:** Secure user registration, login, profile updates, and token-based authentication.

**Content Management API:** Accurate content CRUD operations with proper formatting and retrieval.

**Media Management API:** Secure media upload, retrieval, and storage.

**Advertisement Management API:** Effective advertisement management and analytics.





# CLASS DIAGRAMS

**Classes:** Represent entities with attributes and operations.

**Relationships:**

- **Composition:** Strong ownership, part cannot exist without the whole.
- **Generalization:** Inheritance, "is-a" relationship.

## Example Class Diagram

- **User Management:**
  - **User:** Attributes (username, password, email), Operations (register, login, updateProfile).
  - **Role:** Attributes (roleName), Operations (assignRole).
  -
- **Content Management:**
  - **ContentItem:** Attributes (title, body, tags), Operations (create, update, delete).
  - **RichText:** Inherits from ContentItem, additional formatting attributes.





# CLASS DIAGRAMS

## Media Management:


- MediaFile: Attributes (filename, filepath), Operations (upload, retrieve).

## Advertisement Management:

- Advertisement: Attributes (adTitle, adContent, targetAudience), Operations (createAd, editAd).

## Key Components of a Class Diagram

**Classes:** Represent entities in the system. Each class is depicted as a rectangle divided into three compartments:

- **Class Name:** The name of the class.
  - **Attributes:** The properties or data fields of the class.
  - **Operations:** The methods or functions the class can perform.
- 





# CLASS DIAGRAMS

## Relationships:

- **Association:** A general connection between classes, represented by a line.
- **Multiplicity:** Specifies the number of instances of one class related to one instance of another class (e.g., 1..\*, 0..1).
- **Aggregation:** A special type of association representing a whole-part relationship, depicted by a hollow diamond.
- **Composition:** A stronger form of aggregation indicating ownership, depicted by a filled diamond.
- **Inheritance (Generalization):** Represents an "is-a" relationship, depicted by a solid line with a hollow arrowhead pointing to the parent class.
- **Dependency:** Represents a "uses-a" relationship, depicted by a dashed line with an arrow.





# SEQUENCE DIAGRAMS

## Sequence Diagram Concepts

- Lifeline: Represents an object's existence over time.
- Messages: Communication between objects.

## Example Sequence Diagrams

### User Registration:

- Actor (User) → UserController: register()
- UserController → UserService: validateUser()
- UserService → UserRepository: saveUser()
- UserRepository → Database: insertUser()
- Database → UserRepository: returnSuccess()
- UserRepository → UserService: returnSuccess()
- UserService → UserController: returnSuccess()
- UserController → Actor: registrationSuccess()



# SEQUENCE DIAGRAMS

## Content Creation:

- Actor (ContentCreator) → ContentController: createContent()
- ContentController → ContentService: validateContent()
- ContentService → ContentRepository: saveContent()
- ContentRepository → Database: insertContent()
- Database → ContentRepository: returnSuccess()
- ContentRepository → ContentService: returnSuccess()
- ContentService → ContentController: returnSuccess()
- ContentController → Actor: creationSuccess()

# SEQUENCE DIAGRAMS

## Key Components of a Sequence Diagram

- **Actors:** External entities that interact with the system (e.g., users, other systems).
- **Objects/Classes:** Represent the entities that participate in the interaction.
- **Lifelines:** Vertical dashed lines that represent the lifespan of an object during the interaction.
- **Activation Bars:** Thin rectangles on a lifeline that indicate when an object is active or executing a process.
- **Messages:** Horizontal arrows between lifelines that represent communication between objects. These can be:
  - **Synchronous Messages:** Represented by a solid arrowhead, indicating a call that waits for a response.
  - **Asynchronous Messages:** Represented by a stick arrowhead, indicating a call that does not wait for a response.
  - **Return Messages:** Dashed lines indicating the return of control or data.
  -
- **Frames:** Boxes that represent conditional or looping constructs.



# OPERATIONS AND SERVICE CONTROLLERS

## Operations in Classes

- Ensure each class from the Class Diagram has corresponding operations from Sequence Diagrams.
- Example: User class with register, login, and updateProfile methods.

## Service Controllers

- Role: Facilitate interaction between different subsystems.
- Example: UserService mediates between UserController and UserRepository





# PRACTICAL SESSION

## Creating Class Diagrams

- Objective: Create a Class Diagram using provided requirements.
- Activity: Draw classes for User Management, Content Management, Media Management, and Advertisement Management. Identify relationships like Composition and Generalization.

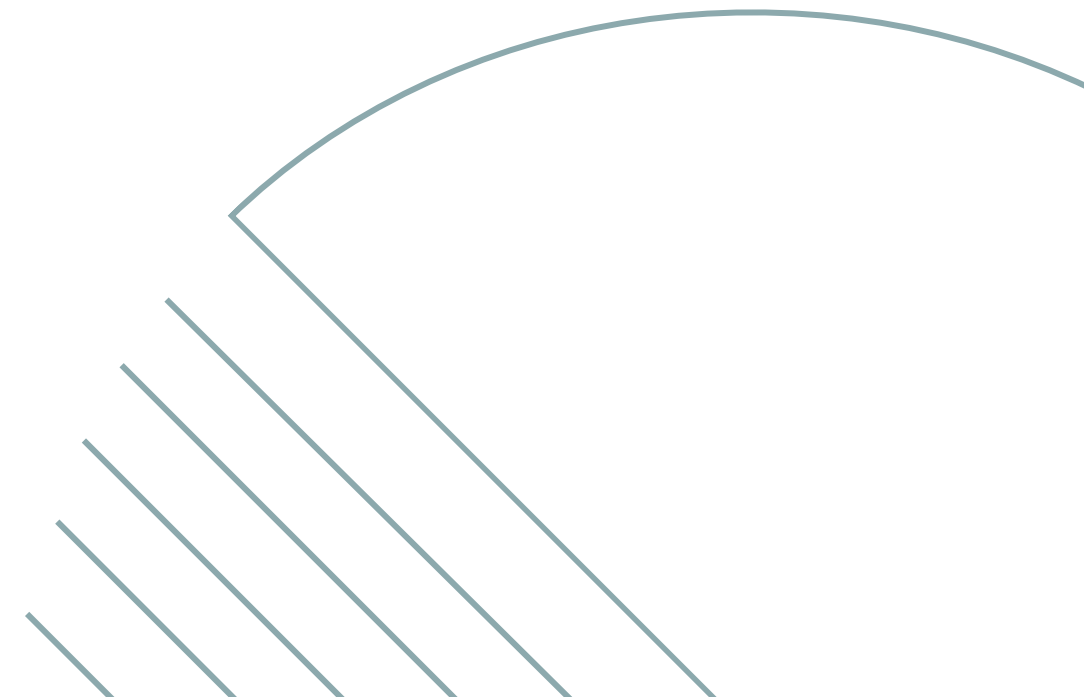
## Creating Sequence Diagrams

- Objective: Create Sequence Diagrams for user registration and content creation.
- Activity: Draw sequence diagrams showing object interactions for each operation.



# SUMMARY AND Q&A

- What is the importance of UML in system design?
- Explain Class and Sequence Diagrams?
- What is CRUD?
- A REST API is a set of web service endpoints that adhere to REST principles, enabling interaction with resources using standard HTTP methods like GET, POST, PUT, and DELETE.  
(Representational State Transfer Application Programming Interface)





# THANK YOU

[www.mouraleonardo.com](http://www.mouraleonardo.com)