Database Fundamentals

Season 2024-III

Project advance Databases
APP Mimo

Report From:

APP
Application and functionalities
Processes and required information
Steps to reach the entity relationship model
Relationship entity model

By: Kevin Estiven Lozano Duarte
Code: 20221020152

Eng. Carlos Andrés Sierra, M.Sc
Computer Engineering
Universidad Francisco José De Caldas

INDEX

1). APP

Mimo: application to learn how to program languages such as Python, JavaScript, SQL, HTML and Swift.

Functionalities:
Mimo works as a personalized teacher in various fields of programming, first collecting information from its users to generate a more direct approach.

- Mimo generates a library of various languages.
- User experiences to create courses and tests.
- Access to the system from any device.
- M-Learning
- An evaluation system and and be able to see your progress.

Its approach:

Mimo uses a modular approach in its programming:

1). Interactive lessons
2). Practical exercises
3). Immediate feedback
4). Gemification
5). Projects
6). Diversity of contents

We can also get to identify certain characteristics of this application is well true that there are many models similar to these applications one of the most famous examples or one of the pioneers of these applications is Duolingo because many of these DM-Learning applications store information about the courses in the following ways:

1). Course structure (databases)
2). API and Backend services
3). Cloud storage
4). Data security
5). Analysis and personalization

2). APLICATION AND FUNCTIONALATIES

This makes in applications similar to Duolingo or in this case in our application Mimo which are applications that have conquered the public because the services work as the backbone of the application to connect the Fronted in the Backend.

2.1). Microservices architecture

In this type of applications we can see a database structure with a relational model where there is a system of tables that organizes information including users, courses, lessons, progress and achievements:

2.1.1 Main services

2.1.2 APIs

2.1.3 Databases

We could also find a non-SQL model where there are documents or collections in a system like mongo must and data can be stored in document formats Jason a structure could be users courses.

2.2). Functionalities of the services

These applications probably use an-app that allows communication between the application and the server the API would handle requests like:
- get the light list of resources
- save user progress
- retrieve specific lessons

Endpoints: each functionality of the application may have a specific EndPoint. For example:
- Get/courses: retrieves all available courses.
- Post/progress: saves users' progress in the specific lesson
- Get/users/{id}: retrieves user profile information

Real-time services

These applications could employ technologies such as web sockets to manage real-time communication, which could be used to:
- Update progress in real time, without the need to reload the page or app.
- Send automatic notifications or reminders when the user reaches certain milestones or when new content is available

Authentication and session management

The system will probably use JWT tokens to manage user sessions those tokens are generated at the time of login and attached in the HTTP request headers to authenticate the user in case of request are also allowed to log in via Google Facebook or Apple, probably using OAuth 2.0 as authentication protocol.

2.3). Synchronization and scalability

These applications must be able to handle thousands of users simultaneously, especially during traffic peaks (e.g. during the promotion of a new course). This could be achieved by using autoscaling services such as those of AWS that allow the infrastructure to dynamically grow or shrink according to demand. To ensure the integrity and privacy of the data of the users of these apps, multiple layers of security such as encryption in transit, encryption at rest, and strict sex policies are likely to be employed.

To guarantee the continuous availability and performance of the services, these apps could use monitoring tools such as Prometheus Grafana or dataDog, these tools allow:
- Monitor the status of the microservices and the infrastructure
- Detect bottlenecks or service failures
- Adjust server capacity in real time according to user traffic.

2.4). Design patterns

2.4.1). Microservice patterns:

Each of the functionalities such as authentication user progress course management etc. are encapsulated in their own microservice what does this mean that each service is independent and takes care of a specific function and this architecture facilitates scalability, allows faster upgrades and can reduce the impact of possible failures. For example:

- authentication services: handle registration, login, and token verification.
- Course service: manages the creation, modification, and deletion of resources.
- Progress service: handles logging and querying of user progress
- Notification services: sends notifications about new courses or reminders.

This means that these microservices have some key characteristics, which are the decompensation of services, communication with pencil and an independent deployment between them.

2.4.2). MVC design pattern (model-View-controller)

It is likely that the Fronted of these applications are structured following the model-view-controller pattern. This creates an independence of operation which means that it facilitates a more efficient maintenance in case of errors and offers simpler ways to test the correct operation of the system allowing the scaling of the application in case of being required.

2.4.3). Repository pattern

In the management of data persistence, such as user courses or progress, the repository pattern can be seen reflected in that it separates the logic that retrieves the data and assigns them to an entity model from the business logic that acts on the model, this allows our business logic to be independent of the type of touch comprises the data source layer in short a repository mediates between the domain and the data mapping layers.

2.4.4). CQRS pattern (Command query Responsibility Segregation)

It is used to separate the responsibility of reading operations, it helps us to separate the queries (data retrieval) from the commands, that is to say, from the insertion, update and deletion of data.

2.4.5). Observer pattern

It is probably used to improve events in real time or identifications in the application that allows us that any object that is implemented in the subscriber interface can subscribe to the notifications of the events.

2.4.6). Checking pattern

To improve performance and reduce latency

2.4.7). Factory pattern

It could be in use for the creation of complex objects, such as the enrollment process in a course or the configurations of a new user for example: a user registers in the application and a Factory is in charge of creating the object.

2.4.8). Adapter pattern

Could be present to integrate external systems such as third party APIs for authentication, this allows a structural design that helps with collaboration between objects with incompatible interfaces.

2.4.9). Singleton Pattern

It could be used to manage global instances that are needed in multiple parts of the system this helps us to make sure that a class has a single instance and at the same time provides a global access point to that instance.

2.4.10). Builder Pattern

It would be used when we need to create complex objects for example: in the construction of multiple lessons it helps us to create these objects step by step and the pattern allows us to produce different types and representations of an object using the same building code.

1. Project Overview

- Project Name: Online Learning System

- Description: The database supports an online learning platform, managing users, courses, payments, progress, reviews, enrollments, and notifications. Design patterns such as Microservices, MVC, Repository, and others are employed.

2. Functional Requirements

- Users should be able to register, enroll in courses, track their progress, make payments, and leave reviews.

- Administrators should be able to manage courses, lessons, and users.

- Notifications should be generated and payments stored.

3. Entity-Relationship Model (ER)

The entity-relationship model defines the entities, relationships, and cardinalities within the system. Here are the key entities:

- User: Stores user information.

- Course: Stores course data, including the creator.

- Lesson: Each course has multiple lessons.

- Enrollment: Stores user enrollments in courses.

- Progress: Tracks user progress in courses and lessons.

- Payment: Stores information about payments made by users.

- Review: Stores user reviews of courses.

- Notification: Stores notifications sent to users.

4. Relationships

- User and Course: A user can create multiple courses, but a course is created by a single user (1:N).

- User and Enrollment: A user can be enrolled in multiple courses, and a course can have multiple users enrolled (N:M).

- User and Progress: A user can have progress in multiple courses and lessons, and a course or lesson can have progress from multiple users (N:M).

- User and Payment: A user can make multiple payments (1:N).

- User and Review: A user can write multiple reviews, and a course can have multiple reviews from different users (N:M).

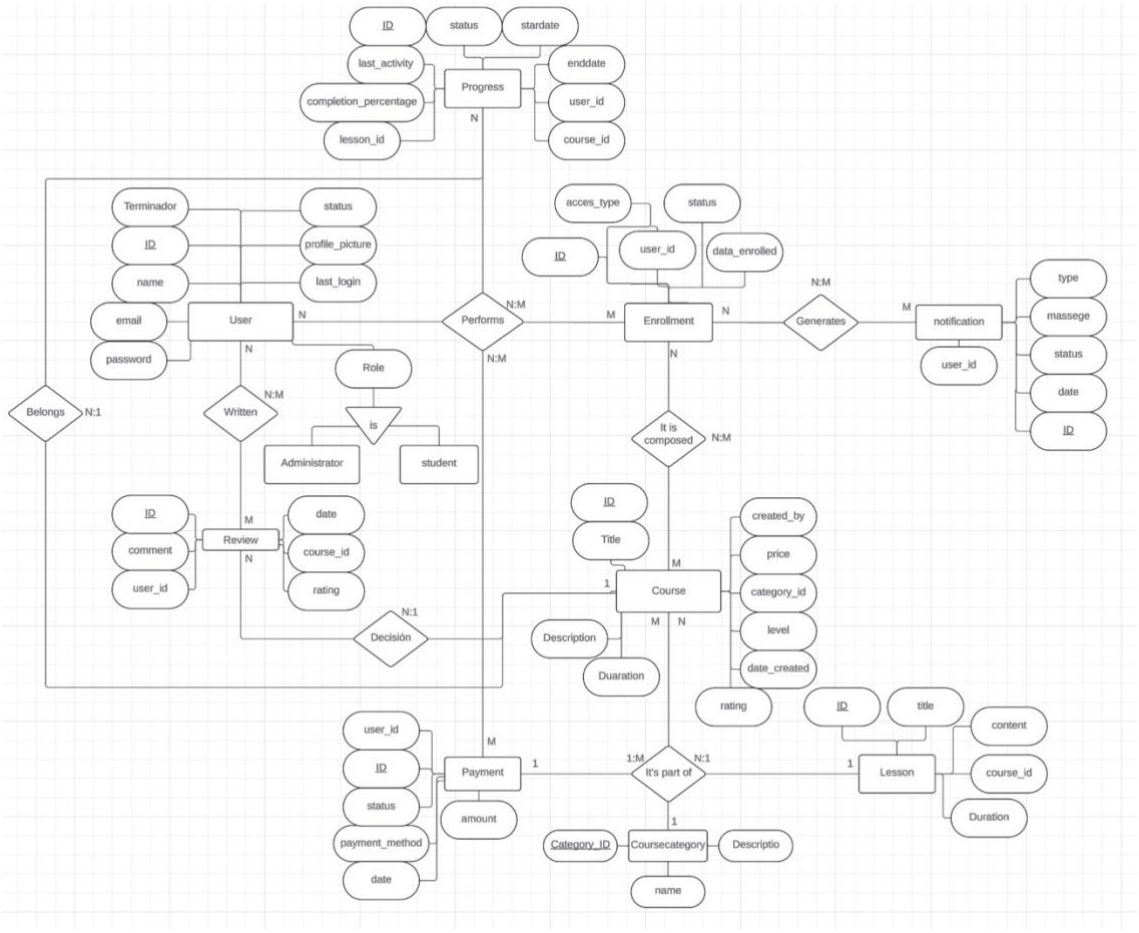
5. Relational Model

For each entity and relationship in the ER model, a corresponding table should be defined in the relational model:

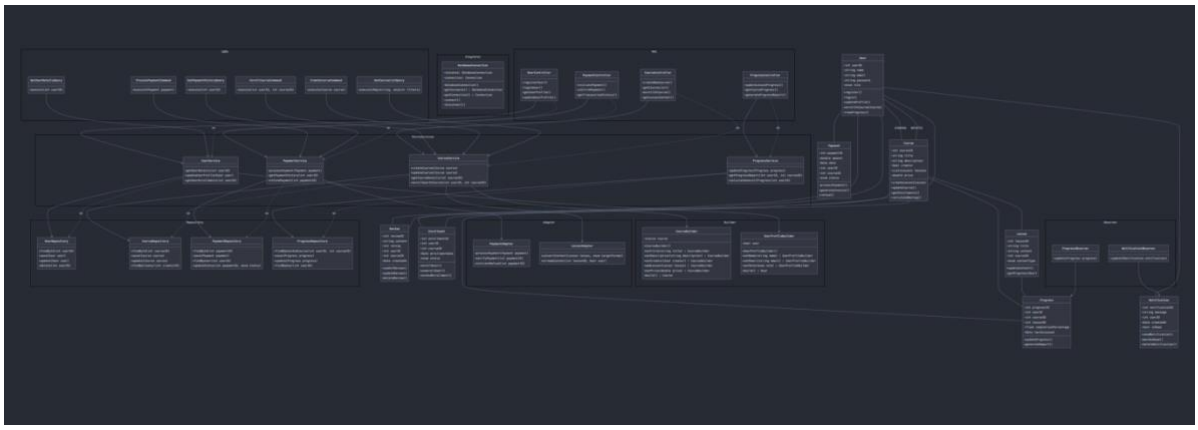
- User (User_ID, Username, Email, Password, Role, Created_At, status, last_ login)
- Course (Course_ID, Title, Description, Category_ID, Created_By, Created_At, level, created_by, rating, price, duration)
- Lesson (Lesson_ID, Title, Content, Course_ID, duaration,)
- Enrollment (Enrollment_ID, User_ID, Course_ID, Enrolled_At)
- Progress (Progress_ID, User_ID, Course_ID, Lesson_ID, Completion_Percentage, Updated_At)
- Payment (Payment_ID, User_ID, Amount, Payment_Date, payment_method, status)
- Review (Review_ID, User_ID, Course_ID, Rating, Comment, Created_At)
- Notification (Notification_ID, User_ID, Message, Sent_At, Read_At)
- CourseCategory (Category_ID, Name, Description, Course_ID)

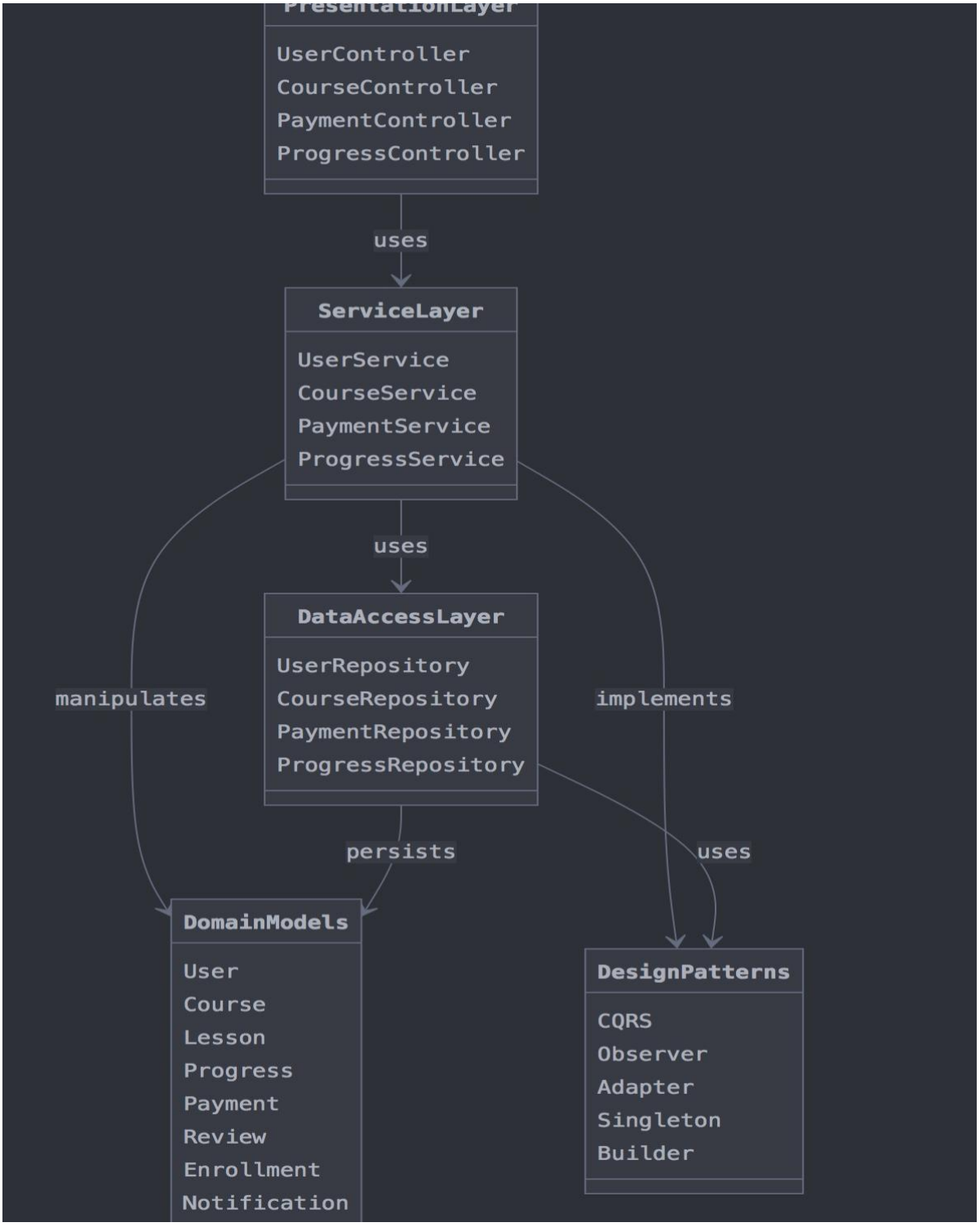
6. Diagrams

- Entity-Relationship (ER) Diagram: Visualizes the entities and their relationships with cardinalities.



- Class Diagram: Represents the entities as classes with their attributes and methods.





7. Design Patterns

The database structure is also designed considering several patterns:

- Microservices: Operations like payments, progress, and notifications are handled through independent microservices.

- Repository: For each entity, there is a repository that manages the interaction with the database.

- CQRS (Command Query Responsibility Segregation): Separates read and write operations in the database, for example, queries for progress and updates for progress.

8. Security Considerations

- Authentication: Implement role-based access control for resources (students, administrators).

- Data Protection: Sensitive data such as passwords should be stored securely (hashing).