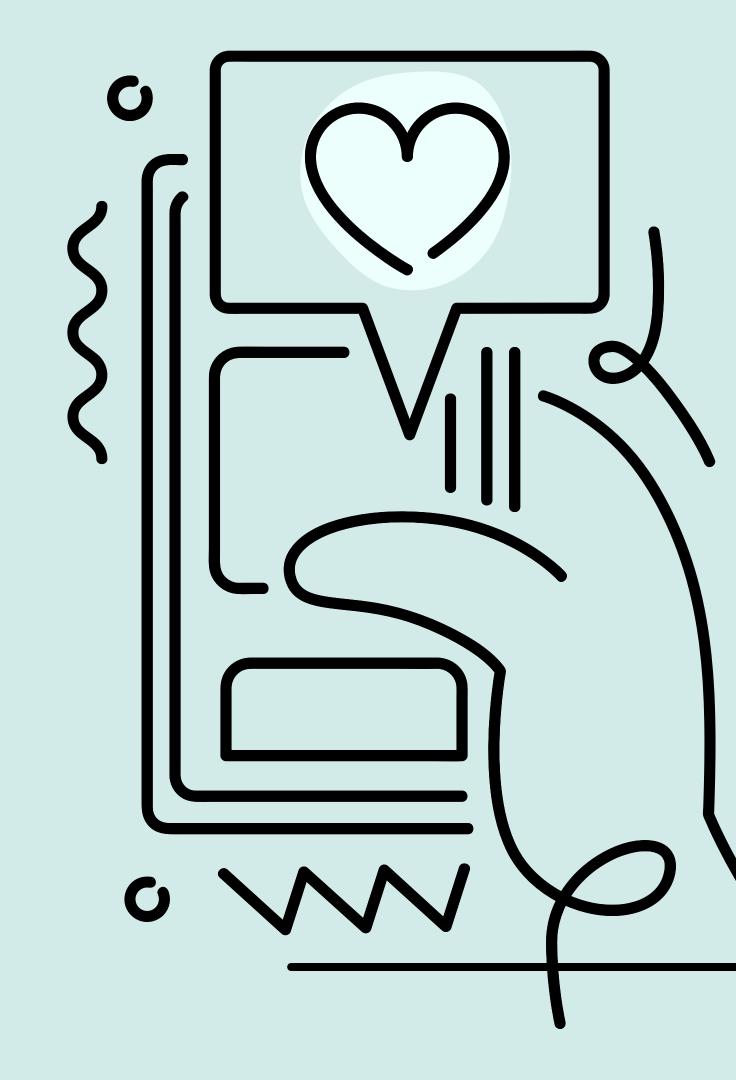
UD SHARE, A USER AND PUBLICATION MANAGEMENT PLATFORM WITH HYBRID DATABASE ARCHITECTURE

Date: July 09, 2025

Prepared by: Johan S. Ebratt, Paola B. Cuellar



PROBLEM STATEMENT

In a world where data is becoming more and more varied and there is a need to store everything, the opportunity arises to use different types of databases depending on the case. We want to understand and demonstrate how combining relational and non-relational databases can facilitate the resolution of information management and processing problems.



KEY FUNCTIONALITIES AND REQUIREMENTS

Non-Functional Requirements:

99.9% uptime with regional redundancy

Auto-scaling infrastructure

 Query response times under 100ms Key User Stories:

 Create, edit, and manage post visibility

 Follow/unfollow users and interact with posts

Report content and moderate users

Access analytics (for premium users)

TECHNOLOGIES USED

POSTGRESQL

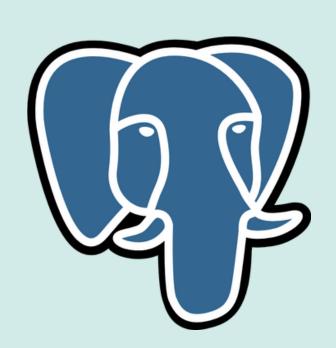
For structured data management.



To store flexible or variable information.

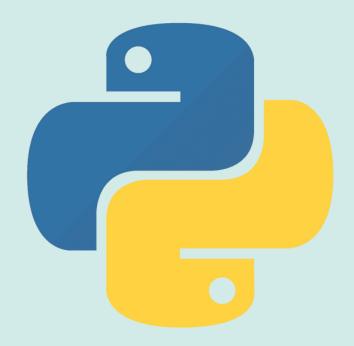
FASTAPI

As a framework to build the API efficiently.



PYTHON

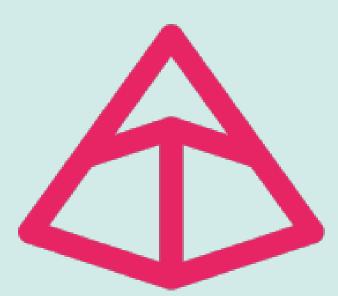
Main backend language.





PYDANTIC

To validate and structure data in the API.





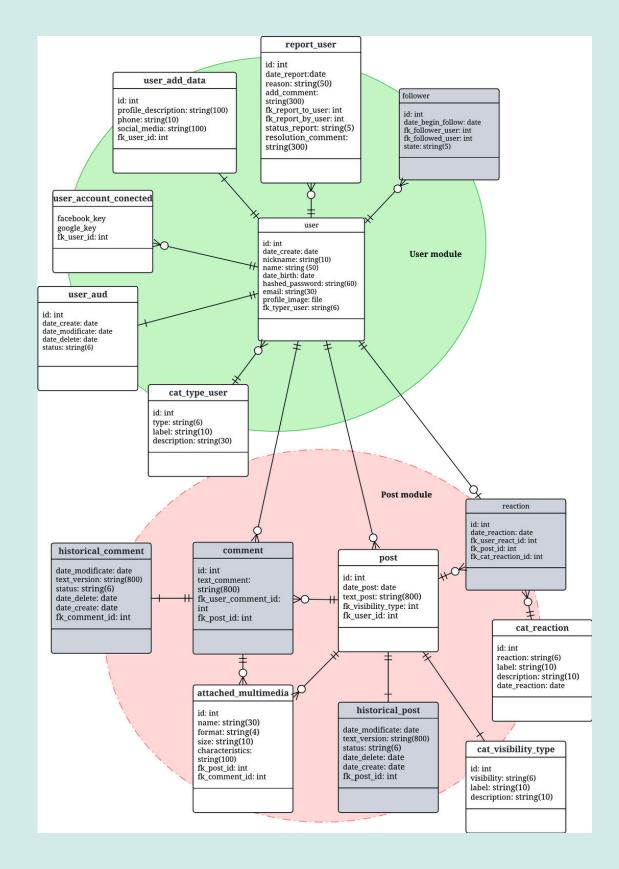
ENTITY-RELATIONSHIP MODEL

Modules:

- User (green): Account management, followers, reports and auditing
- Post (pink): Posts, comments, reactions and historics

Key points:

- Normalization: Use of cat_* for types and states.
- History: Versioning and auditing of changes.
- Attachments: Files associated with post or comment.



WHY A SQL PART?

- Clear structure: Allows explicit definition of relationships using primary and foreign keys.
- Data integrity: Ensures the database follows strict rules (ACID) for reliable and consistent operations.
- Complex queries: Supports operations like JOIN, aggregations, and distributed transactions for more complex scenarios.
- Vertical scalability: Ideal for applications where data is highly interrelated and consistency is required.

user_add_data

id: int profile_description: string(100) phone: string(10) social_media: string(100) fk_user_id: int

user aud

id: int date_create: date date_modificate: date date_delete: date status: string(6)

post

id: int date_post: date text_post: string(800) fk_visibility_type: int fk_user_id: int

cat_visibility_type

id: int
visibility: string(6)
label: string(10)
description: string(10)

report user

id: int
date_report:date
reason: string(50)
add_comment:
string(300)
fk_report_to_user: int
fk_report_by_user: int
status_report: string(5)
resolution_comment:
string(300)

user

id: int
date_create: date
nickname: string(10)
name: string (50)
date_birth: date
hashed_password: string(60)
email: string(30)
profile_image: file
fk_typer_user: string(6)

user_account_conected

facebook_key google_key fk_user_id: int

cat_type_user

id: int
type: string(6)
label: string(10)
description: string(30)

attached multimedia

id: int name: string(30) format: string(4) size: string(10) characteristics: string(100) fk_post_id: int fk_comment_id: int

cat reaction

id: int
reaction: string(6)
label: string(10)
description: string(10)
date_reaction: date

WHY A NOSQL PART?

- Horizontal scalability: Handles large data volumes distributed without losing performance.
- Schema flexibility: Ideal when data is semi-structured or evolves rapidly.
- Performance: Designed to support fast operations with high read/write loads and low latency.
- Diverse models: Offers different types of databases (key-value, document, column, graph) tailored to specific use cases (e.g., MongoDB for JSON, Cassandra for columns).

ud_sharem> show collections
comment
follower
historical_comment
historical_post
reaction

historical comment

date_modificate: date text_version: string(800) status: string(6) date_delete: date date_create: date fk_comment_id: int

comment

id: int text_comment: string(800) fk_user_comment_id: int fk_post_id: int

follower

id: int
date_begin_follow: date
fk_follower_user: int
fk_followed_user: int
state: string(5)

historical_post

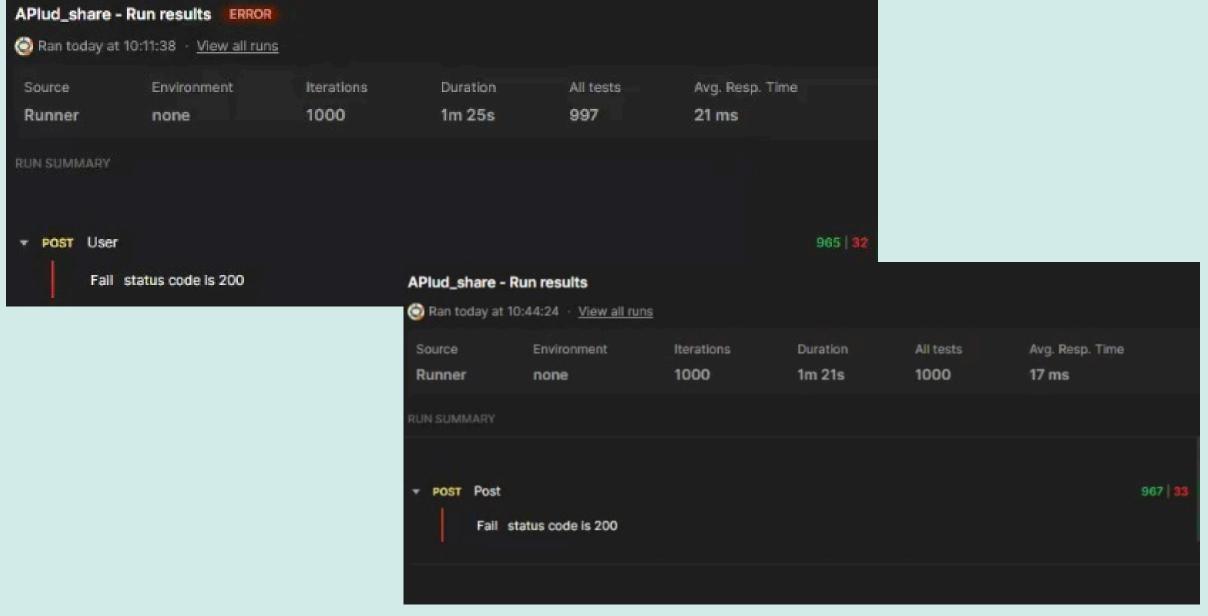
date_modificate: date text_version: string(800) status: string(6) date_delete: date date_create: date fk_post_id: int

reaction

id: int
date_reaction: date
fk_user_react_id: int
fk_post_id: int
fk_cat_reaction_id: int

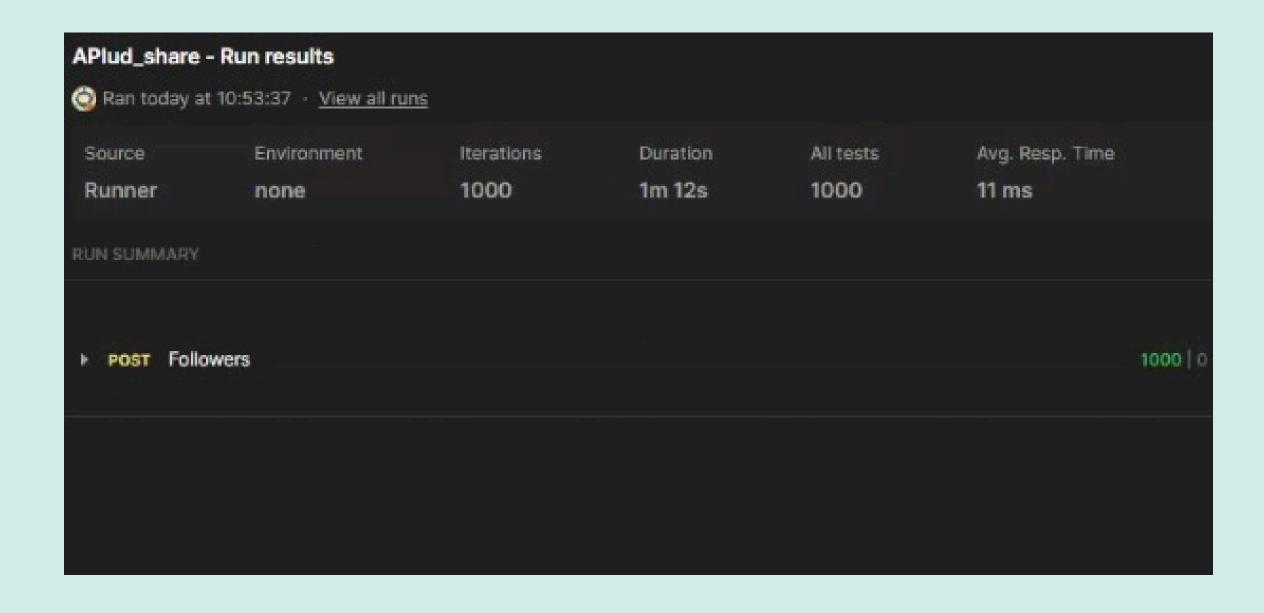
COMPARISON

For UD Share we load and request a good volume of data, below we can show what each database provides us in the processing of this data.





COMPARISON





MODULARITY IN THE PROJECT

- Scalability: Easily add new features without affecting the system.
- Simplified maintenance: Changes or fixes are made independently in specific modules.
- Code reuse: Modules are reusable in other parts of the system or projects.
- Integration of new technologies: Allows the integration of new tools without compromising the system.
- Efficient collaboration: Teams can work in parallel on independent modules.
- Improved security: Each module has its own permission management and protection.







IN THE FUTURE...

Messaging Module:

- Real-time direct messaging between users.
- Features: Notifications, message history.

Kafka Integration:

- Use Apache Kafka for real-time data streaming.
- Event-based reactions: Handle reactions as events in Kafka for real-time processing.

Extended Interactions:

- Custom reactions: Introduce new reaction types (e.g., emojis, GIFs).
- Video comments: Enable video-based comments on posts.

Al Integration:

- Implement recommendation algorithms based on user behavior.
- Automatic moderation: Al-driven content filtering in real-time.



CONCLUSIONS

- Modularity is essential for scalability, flexibility, and ease of maintenance, allowing future improvements without disrupting the core system.
- Relational databases provide strong data integrity and consistency, ideal for managing user data, relationships, and complex queries.
- NoSQL databases excel in handling high volumes of data with fast read/write operations, making them perfect for real-time interactions like comments and reactions.



