WORKSHOP II

SOFIA LOZANO MARTINEZ
20211020088

JOSÉ JESÚS CÉSPEDES RIVERA
20211020118

CARLOS ANDRÉS SIERRA VIRGUEZ

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

FRANCISCO JOSÉ DE CALDAS DISTRICT UNIVERSITY

FACULTY OF ENGINEERING
CURRICULAR PROJECT: SYSTEMS ENGINEERING
DATABASE II

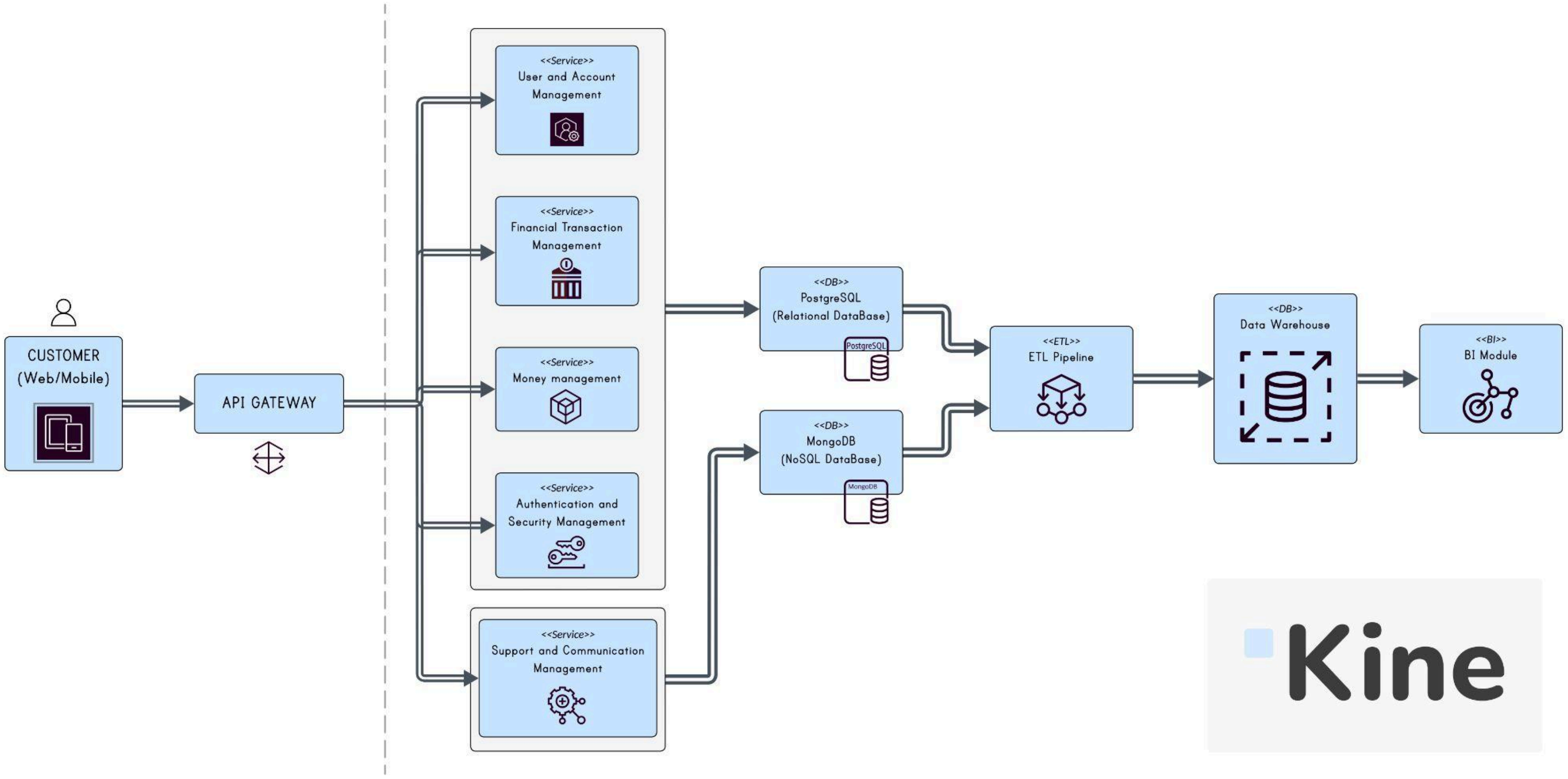
BOGOTÁ, D.C., JUNE 26
2025

INTRODUCTION

This second workshop for the Database course focuses on the design and analysis of a distributed data management system, based on the functional and structural architecture of a Nequi-like solution. This submission presents the progress of the data model, incorporating new entities and relationships that address key functionalities such as money management, authentication, customer support, and data analysis. The technologies used (PostgreSQL and MongoDB) are described, along with proposed SQL and NoSQL queries to retrieve the necessary information based on the user story requirements. Additionally, the complete data flow is presented—from user interaction to report visualization in the BI module—demonstrating a comprehensive approach to data management and analysis.

Although the system continues to replicate the reference database structure, in this phase of the project, a unique name is established to give the solution its own identity and to narrow the scope of the implementation. The system is now named Kine, and under this name, the specific components, modules, and functionalities of our customized version will be developed and consolidated.

DATA SYSTEM STRUCTURE



CUSTOMER (Web/Mobile)

Role: Represents the interface with the end user. Through mobile devices or browsers, customers interact with system features such as transactions, balance inquiries, and support.

API GATEWAY

Role: Single point of entry for customer requests. Responsible for authenticating, authorizing, and redirecting requests to the appropriate microservice, centralizing access logic.

Microservices

Each specialized in a functional domain:

- User and Account Management: Management of user and account information.
- Financial Transaction Management: Processes payments, transfers, and financial transactions.
- Money Management: Allows the user to plan and visualize their financial situation.
- Authentication and Security Management: Responsible for credential validation and overall system security.
- Support and Communication Management: Channels user support requests and communicates with other support services.
    - A NoSQL database (MongoDB) was integrated for this component, allowing flexibility in storing free text (such as support messages), open requests, and conversation tracking. This decision responds to the need to process and store semi-structured data with high variability.

Databases

For the development of the Kine system, inspired by Nequi's architecture, a technological combination based on PostgreSQL and MongoDB was chosen, responding to the scalability and consistency requirements of a digital banking environment. Nequi processes more than 38 million transactions daily—about 26,400 transactions per minute—and serves 21.3 million active customers, according to its CEO. To absorb a load of this magnitude, Kine requires a motor that guarantees strict integrity and proper concurrency.

PostgreSQL (Relational/Transactional core):

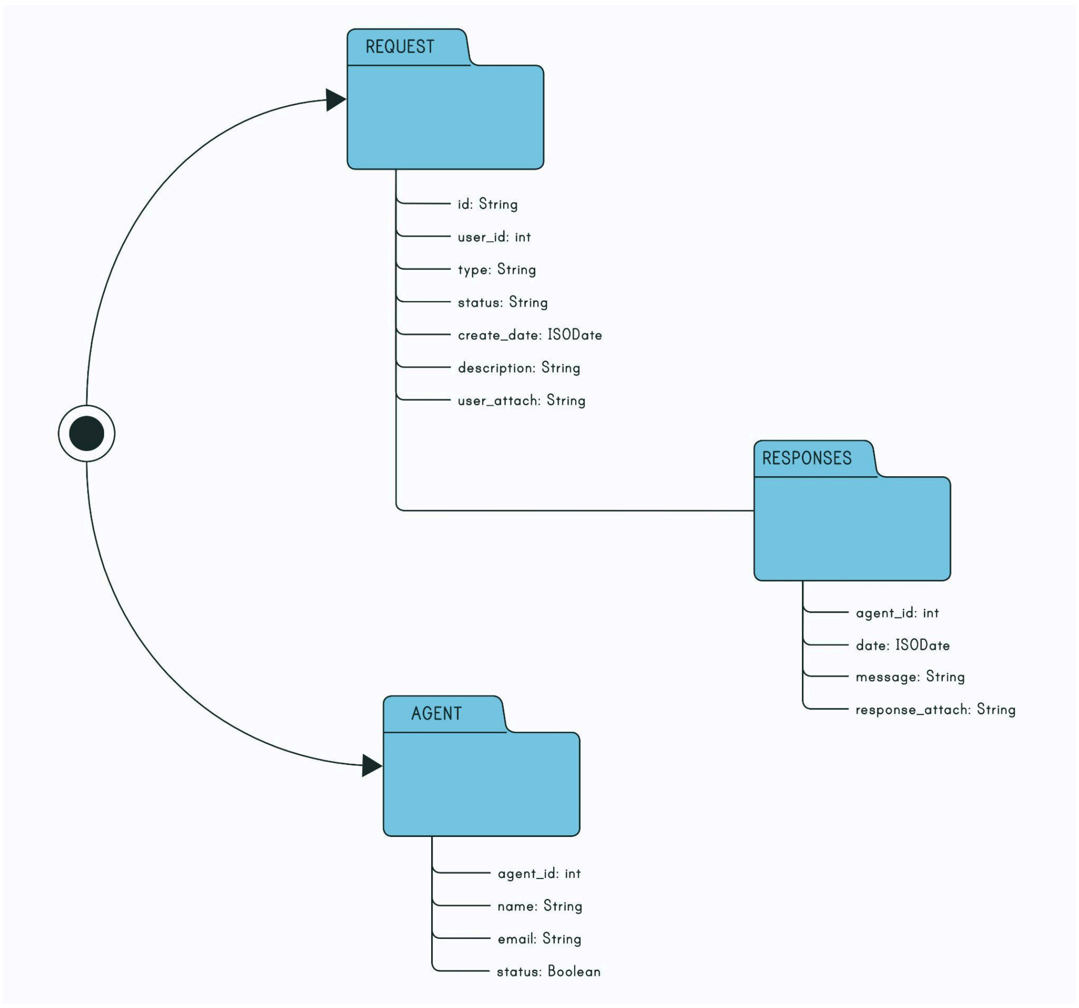
- **Role:** Stores critical structured and transactional data (accounts, transactions, profiles).
- **Justification:** PostgreSQL complies with ACID standards, which is essential for financial operations such as authentication, balance records, and transfers. Furthermore, its implementation of MVCC (Multi-Version Concurrency Control) allows for a high level of concurrency without blocking between reads and writes, optimizing latency in high-demand systems. This ability to execute complex queries, stored procedures, and robust indexes

ensures that PostgreSQL efficiently supports Kine's transactional flow, maintaining resilience and consistency.

MongoDB (NoSQL/Support core):

- **Role:** Stores semi-structured information, including logs, preferences, and dynamic data from the support module.
- **Justification:** At the same time, MongoDB is integrated to manage unstructured data (logs, audits, session metadata, and customer support) in an environment that requires schema flexibility and high scalability. Its document-oriented model allows it to adapt quickly to changes in data requirements, without the rigidity of a relational schema. In addition, through mechanisms such as horizontal sharding, MongoDB efficiently distributes large volumes of data and write-intensive loads, ensuring availability and performance in peak scenarios. This complements the strengths of PostgreSQL, enabling Kine to manage both the critical transactional core and auxiliary operations in an agile and scalable manner.

This model represents the data structure of the user support system, implemented with a document-oriented database (MongoDB). Its purpose is to manage Request (Petitions, Complaints and Claims) and their corresponding responses by customer service agents.



- ***Main collection (REQUEST):*** Each document in this collection represents a request made by a user, with the following key elements:
  - user_id: Reference to the user's identifier (from the relational system).
  - type: Type of request (complaint, request, etc.).
  - status: Status of the case (in_progress, resolved, etc.).
  - description: Detailed text of the request.
  - user_attachment: Embedded list of files sent by the user.
  - responses: Embedded list of responses from agents, each one with:
    - agent_id: Agent identifier (reference to agent collection).
    - message: Textual response.
    - attachment: Files sent by the agent in its response.

- ***Agente Collection:*** Contains the information of the support agents:
  - id
  - name
  - email
  - status

ETL Pipeline

Role: Extracts information from PostgreSQL and MongoDB, performs transformations (cleansing, aggregation), and loads it into the data warehouse.

Data Warehouse

Role: Central repository for analyzing large volumes of historical and aggregated data.

BI Module

Role: Visualization tool for generating dynamic dashboards and reports.

Data Flow

1. The client initiates an interaction from their mobile app or website.
2. The request is managed by the API Gateway, which channels it to the corresponding microservice.
3. The microservices access PostgreSQL for structured operations (users, accounts, transactions) and MongoDB for unstructured data (support, events).
4. The ETL Pipeline extracts data from both databases, transforms it, and loads it into the Data Warehouse
5. The analytics team accesses it through the BI Module to visualize key metrics and generate reports.

INFORMATION TO BE RETRIEVED BY THE SYSTEM

The following outlines the main types of information the system must retrieve in order to support the functionalities described in the user stories. This information aligns with the modeled entities and responds to business needs.

1. User Registration and Authentication (HU01, HU08)

To allow the registration of new users and ensure secure authentication, the system must be able to retrieve the following information:

- Verify whether a user is already registered by document or phone number (User.user_id, User.phone).
- Retrieve authentication data, such as the securely stored PIN and biometric method, if configured (Authentication.PIN, Authentication.biometrics).
- Retrieve the devices registered by the user and their current status (Device.status), to validate whether the device is trusted or if it is a new access attempt.

2. Balance Inquiry and Account Movements (HU02)

To allow users to check their available balance and add money to their account, the system must retrieve:

- The current balance of the account (Account.balance).
- The transaction history over a specific period.
- The status of the account (Account.status), indicating whether it is active or blocked.

3. Money Transfers (HU03)

To facilitate the sending and receiving of money between users, the system must retrieve:

- The user's available balance (Account.balance) to validate whether the transfer can be completed.
- The recipient’s information, including name and account status (User.full_name, Account.status), in case of a Nequi-to-Nequi transfer.
- The history of transfers made and received (Movement with type transfer), to be displayed in the user interface.

4. Payments to Merchants and Services (HU04, HU07)

This functionality allows users to make payments at merchants and for public services directly from their Nequi account. The system must retrieve:

- The available balance in the main account (Account.balance) to validate the availability of funds.

- The active card associated with the account, including its status and available limit (Card.status, Card.limit), if the payment is to be made using a card.
- The history of completed payments, including those made to merchants and services (Movement with type bill_payment).
- The reference or name of the merchant or service paid (Movement.reference), to display transaction details to the user.

5. Expense Management with Pockets and Cushion (HU05)

To provide users with tools that help them better organize their money through the use of pockets and the cushion, the system must retrieve:

- The list of pockets created by the user (Poken) and the available balance in each (Poken.balance).
- The current amount saved in the cushion (Cushion.balance) and whether it is locked or not (Cushion.locked).

6. Credit Management (HU06)

To support credit application and management functionality, the system must retrieve:

- The credit records associated with the user (Credit by account_id).
- The requested amount, disbursement status, and repayment terms (Credit.amount, Credit.status, Credit.term).
- The credit history and disbursement movements related to the loan (Movement with type credit).
- The account balance to verify if the funds from the credit have been received (Account.balance).

7. Support Management (HU08)

To provide user assistance and ensure proper tracking of their requests, the system must retrieve:

- The support tickets associated with the user (Support by user_id).
- The type of request submitted (query, report, response) (Support.type).
- The current status of the ticket (Support.status), to determine whether it is in progress, resolved, or closed.
- The full description of the ticket and its update history if applicable (Support.description).

QUERY PROPOSALS

POSTGRESQL - SQL

1. User Registration and Authentication

Allows the system to verify if a user already exists and manage secure authentication. Provides information about user identity, login credentials, and registered trusted devices.

- Query to verify if the user is already registered by document or phone number.
```
SELECT user_id, ID_document, phone
FROM User
WHERE ID_document = 'DOCUMENT' OR phone = 'PHONE_NUMBER';
```

- Query to retrieve the user's credentials (PIN and biometrics) from the Authentication entity.
```
SELECT PIN, biometrics
FROM Authentication
WHERE user_id = 'USER_ID';
```

- Query to retrieve all devices associated with the user, including their status and usage dates.
```
SELECT device_id, registration_date, last_used, status
FROM Device
WHERE user_id = 'USER_ID';
```

2. Balance Inquiry and Account Movements

Allows the user to check their balance and see how their account is moving—what their expenses have been and what their sources of income are.

- Query to retrieve the current balance and status (active or blocked) of the account.
```
SELECT balance, status
FROM Account
WHERE account_id = 'ACCOUNT_ID';
```

- Query to retrieve the account's transaction history over a specific period.
```
SELECT movement_id, amount, date_time, type_movement, status,
reference
FROM Movement
WHERE source_account_id = 'ACCOUNT_ID'
AND date_time BETWEEN '2025-05-01' AND '2025-05-31'
ORDER BY date_time DESC;
```

3. Money Transfers

Allows validation of available funds for transfer and provides recipient details. Also offers a transfer history useful for personal tracking or resolving disputes.

- Query to verify the available balance before making a transfer.
```
SELECT balance
FROM Account
WHERE account_id = 'ACCOUNT_ID';
```

- Query to retrieve the recipient's name and account status in a Nequi-to-Nequi transfer.
```
SELECT u.full_name, a.status
FROM User u
JOIN Account a ON u.user_id = a.user_id
WHERE a.account_id = 'DESTINATION_ACCOUNT_ID';
```

- Query to display transfers made or received, useful for the user interface.
```
SELECT movement_id, amount, date_time, reference
FROM Movement
WHERE source_account_id = 'ACCOUNT_ID'
AND type_movement = 'transfer'
ORDER BY date_time DESC;
```

4. Payments to Merchants and Services

Enables control over card usage, verifies limits, and retrieves payment history to merchants and public services. Provides transparency by showing the amount, provider, and date of each transaction.

- Query to retrieve the balance before making a payment.
```
SELECT balance
FROM Account
WHERE account_id = 'ACCOUNT_ID';
```

- Query to validate if the card is active and what its limit is.
```
SELECT card_id, status, limit
FROM Card
WHERE account_id = 'ACCOUNT_ID' AND status = 'active';
```

- Query to retrieve payments made to public services, along with their reference.
```
SELECT m.movement_id, m.amount, m.date_time, m.reference,
bp.service_company
FROM Movement m
JOIN BillPayment bp ON m.movement_id = bp.movement_id
WHERE m.source_account_id = 'ACCOUNT_ID'
ORDER BY m.date_time DESC;
```

5. Expense Management with Pockets and Cushion

Allows the user to organize their money into categories (pockets) or reserve it (cushion). Offers visibility into how the user manages and distributes their finances.

- Query to check the amount stored in the cushion and whether it is locked.
```
SELECT balance, locked
FROM Cushion
WHERE account_id = 'ACCOUNT_ID';
```

6. Credit Management

Provides access to active or requested loans, including terms and current status. Useful for tracking debt, due dates, interest rates, and the user's financial compliance.

- Query to retrieve the credits requested by the user, along with their status and conditions.
```
SELECT movement_id, credit_type, interest, term, due_date,
credit_status
FROM Credit
WHERE movement_id IN (
  SELECT movement_id
  FROM Movement
  WHERE source_account_id = 'ACCOUNT_ID'
);
```

- Query to display disbursements or transactions related to granted credits.
```
SELECT amount, date_time, status
FROM Movement
WHERE source_account_id = 'ACCOUNT_ID' AND type_movement =
'credit';
```

MONGODB - NOSQL

7. Support Management

- The support tickets associated with the user (Support by user_id).
  `db.request.find({ user_id: "user_id" })`

- The type of request submitted (query, report, response) (Support.type).
  `db.request.find({ type: "type" })`
  `db.request.distinct("type")` – This query allows you to view all types of existing requests.

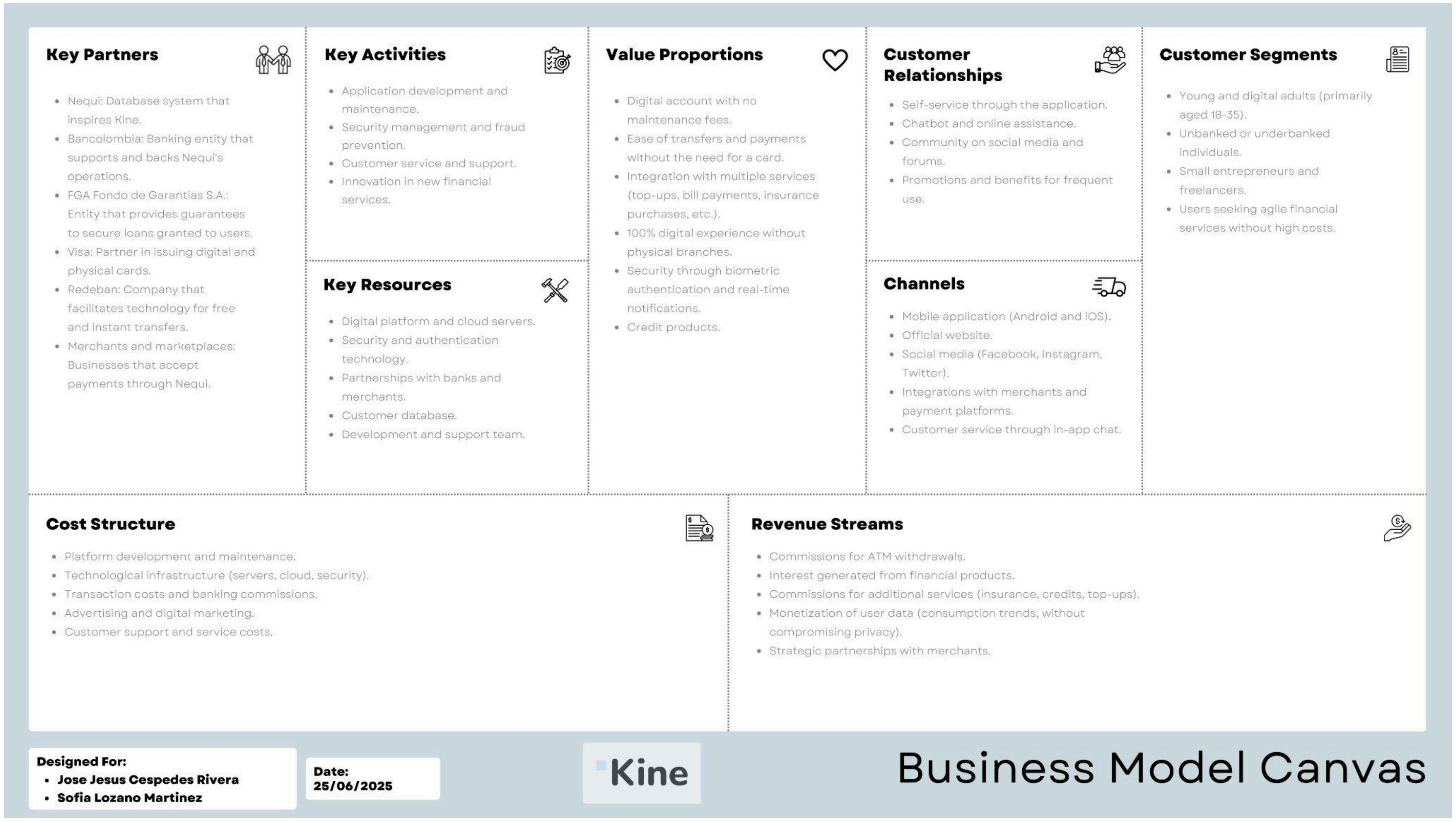
- The current status of the ticket (Support.status), to determine whether it is in progress, resolved, or closed.
```
db.pqr.aggregate([

  { $group: { _id: "$status", count: { $sum: 1 } }

])
```

WORKSHOP 1 CHANGES

Small adjustments were made to the model. The new model was focused on Kine, while maintaining the aspects of the previous Nequi's model. Some key partners and key activities were eliminated.



Key Partners

- Nequi: Database system that inspires Kine.
- Bancolombia: Banking entity that supports and backs Nequi's operations.
- FGA Fondo de Garantías S.A.: Entity that provides guarantees to secure loans granted to users.
- Visa: Partner in issuing digital and physical cards.
- Redeban: Company that facilitates technology for free and instant transfers.
- Merchants and marketplaces: Businesses that accept payments through Nequi.

Key Activities

- Application development and maintenance.
- Security management and fraud prevention.
- Customer service and support.
- Innovation in new financial services.

Key Resources

- Digital platform and cloud servers.
- Security and authentication technology.
- Partnerships with banks and merchants.
- Customer database.
- Development and support team.

Value Proportions

- Digital account with no maintenance fees.
- Ease of transfers and payments without the need for a card.
- Integration with multiple services (top-ups, bill payments, insurance purchases, etc.).
- 100% digital experience without physical branches.
- Security through biometric authentication and real-time notifications.
- Credit products.

Customer Relationships

- Self-service through the application.
- Chatbot and online assistance.
- Community on social media and forums.
- Promotions and benefits for frequent use.

Channels

- Mobile application (Android and iOS).
- Official website.
- Social media (Facebook, Instagram, Twitter).
- Integrations with merchants and payment platforms.
- Customer service through in-app chat.

Customer Segments

- Young and digital adults (primarily aged 18-35).
- Unbanked or underbanked individuals.
- Small entrepreneurs and freelancers.
- Users seeking agile financial services without high costs.

Cost Structure

- Platform development and maintenance.
- Technological infrastructure (servers, cloud, security).
- Transaction costs and banking commissions.
- Advertising and digital marketing.
- Customer support and service costs.

Revenue Streams

- Commissions for ATM withdrawals.
- Interest generated from financial products.
- Commissions for additional services (insurance, credits, top-ups).
- Monetization of user data (consumption trends, without compromising privacy).
- Strategic partnerships with merchants.

Designed For:
- **Jose Jesus Cespedes Rivera**
- **Sofia Lozano Martinez**

Date: **25/06/2025**

Kine

Business Model Canvas

USER STORY

Estimation and Prioritization Criteria

Estimation

In this project, user story estimation is carried out using the Story Points technique, on a discrete scale from 1 to 5:

Story Point	Meaning
1	Very simple story, low effort, no dependencies or risks
2	Simple story, minimal logic or validations required
3	Medium-complexity story, includes business logic, validations, and testing
4	Complex story, involves multiple steps or interaction with external services
5	Very complex story, high technical risk or strong dependencies on other systems

This scale allows us to evaluate the effort, technical complexity, associated risks, and work volume involved in each story. It does not represent exact time in hours, but rather the overall workload required from the development team.

Using this metric allows:

- Relative comparison between stories
- Better sprint planning
- Estimation of team velocity (how many points can be completed per iteration)

Priority

Priority is assigned based on the value delivered to the end user, the urgency of the requirement, and its impact on the core user flow of the application. For this project, we use a three-level scale:

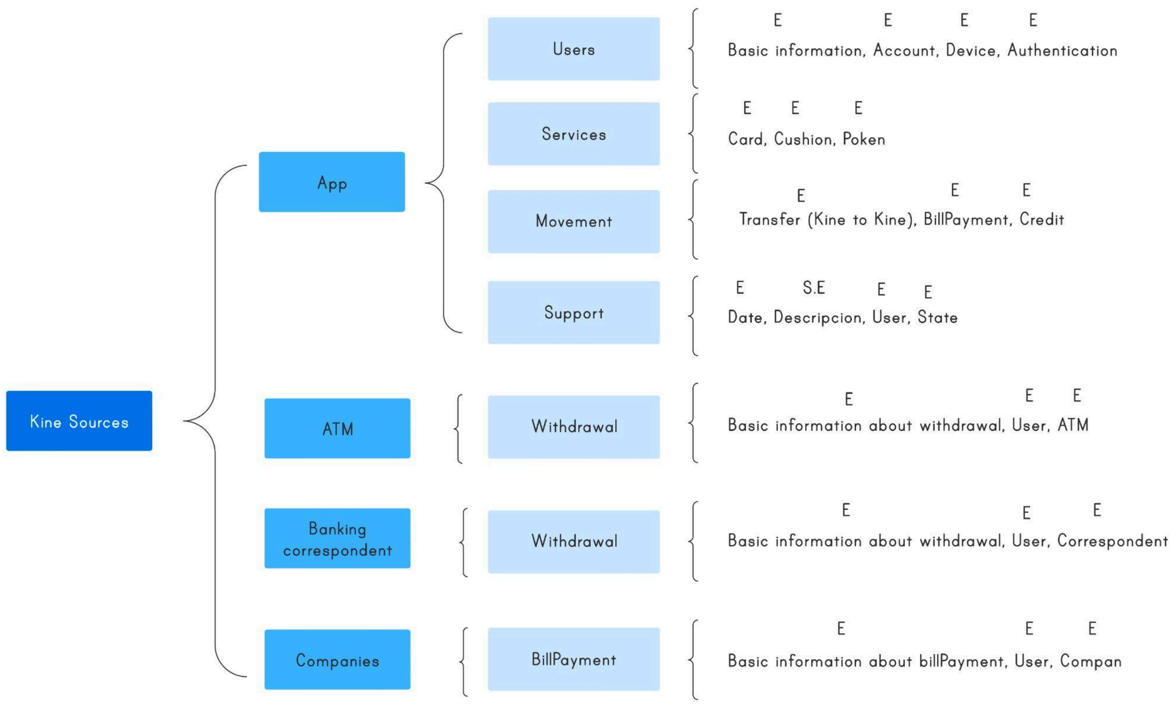
- High: Critical or core functionality for the basic use of Nequi (e.g., registration, balance top-up, money transfer).
- Medium: Important but non-blocking features (e.g., bill payments, savings goals).
- Low: Complementary or less urgent features (e.g., customization, non-essential transaction history).

This prioritization approach helps define which stories should be implemented first to deliver a functional minimum viable product (MVP), and which can be planned for later development phases.

The money management component was added, which is composed of the new entities added to the model: pocket, mattress, and card, which are related to the account entity.

Additionally, attributes were added to the device (login and status) and transaction (payment type) entities.

KINE SOURCES


Kine Sources
App
Users
E E E E
Basic information, Account, Device, Authentication
Services
E E E
Card, Cushion, Poken
Movement
E E E
Transfer (Kine to Kine), BillPayment, Credit
Support
E S.E E E
Date, Descripcion, User, State
ATM
Withdrawal
E E E
Basic information about withdrawal, User, ATM
Banking correspondent
Withdrawal
E E E
Basic information about withdrawal, User, Correspondent
Companies
BillPayment
E E E
Basic information about billPayment, User, Compan

INITIAL DATABASE ARCHITECTURE

1. Components:

    1. **User and Account Management**
    Responsible for storing and managing user personal information and account details, including balances, limits, types, and account status.
    2. **Financial Transaction Management**
    Records and monitors all transactions carried out by users, such as transfers, payments, credit ("Crédito Propulsor" and "Crédito Salvavidas"), and withdrawals, enabling detailed tracking of financial flows.
    3. **Money management**
    Manage the ways in which the user organizes, restricts, or uses their money within their Nequi account. This component doesn't represent a single entity, but rather a set of interrelated entities that operate within an Account, such as: wallet, mattress, and virtual card.
    4. **Authentication and Security Management**
    Controls access mechanisms to the system, including the use of PIN, biometrics, and device verification, ensuring data integrity and confidentiality.

5. **Support and Communication Management**
   Manages interactions between users and the support area, including inquiries, reports, responses, and the customer communication history.

2. Entities:

   1. **User and Account Management**
      a. User
      b. Account
      c. Device

   2. **Financial Transaction Management**
      a. Movement
      b. Transfer
      c. BillPayment
      d. TopUp
      e. Withdrawal
      f. Credit

   3. **Money Management**
      a. Pocket
      b. Cushion
      c. Card

   4. **Authentication and Security Management**
      a. Authentication

3. Attributes per Entity:

   a. User
   - ➔ user_id (PK)
   - ➔ full_name
   - ➔ ID_document
   - ➔ phone
   - ➔ email
   - ➔ country_of_residence
   - ➔ registration_date
   - ➔ status

   b. Account
   - ➔ account_id (PK)
   - ➔ user_id (FK)
   - ➔ type (low amount, savings)

- ➔ balance
- ➔ monthly_limit
- ➔ status
- ➔ creation_date
- ➔ exempt_4x1000

c. **Device**

- ➔ device_id (PK)
- ➔ user_id (FK)
- ➔ registration_date
- ➔ last_used
- ➔ status

d. **Movement**

- ➔ movement_id (PK)
- ➔ source_account_id (FK)
- ➔ amount
- ➔ date_time
- ➔ status
- ➔ type
- ➔ reference

e. **Transfer**

- ➔ movement_id (PK, FK)
- ➔ destination_account_id (FK)
- ➔ type

f. **Withdrawal**

- ➔ movement_id (PK, FK)
- ➔ channel

g. **BillPayment**

- ➔ movement_id (PK, FK)
- ➔ service_company
- ➔ reference_number

h. **Credit (Loan)**

- ➔ movement_id (PK, FK)
- ➔ credit_type (lifeline, booster)
- ➔ interest
- ➔ due_date
- ➔ credit_status

i. **Pocket**

- ➔ pocket_id (PK)
- ➔ account_id(FK)
- ➔ name
- ➔ balance
- ➔ creation_date

j. **Cushion**

- ➔ cushion_id(FK)
- ➔ account_id
- ➔ balance
- ➔ locked
- ➔ creation_date

k. **Card**

- ➔ card_id
- ➔ account_id(FK)
- ➔ card_number
- ➔ expiry_date
- ➔ cvv_hash
- ➔ status
- ➔ issued_date

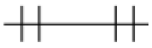
l. **Authentication**

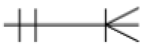
- ➔ auth_id (PK)
- ➔ user_id (FK)
- ➔ PIN

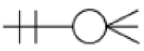
4. Relationships

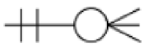
	a	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	*j*	*k*	*l*
a		✓	✓									✓
b	✓			✓					✓	✓	✓	
c	✓											
d		✓			✓	✓	✓	✓				
e				✓								
f				✓								
g				✓								
h				✓								
i		✓										
j		✓										
k		✓										
l	✓											

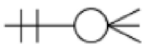
5. Relationships types

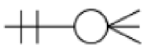
1. User (a)  Account (b): (1:1)
   A user has one account, and one account belongs to only one user.

2. User (a)  Device (c): (1:N)
   A user can register one or multiple devices, but each device is associated with only one user.

3. Account (b)  Movement (d): (1:N)
   An account can have zero or multiple movements, but each movement is associated with only one account.

4. Movement (d)  Transfer (e): (1:1)
   Each transfer is a specific type of movement.

5. Movement (d)  Withdrawal (f): (1:1)
   Each withdrawal is a specific type of movement.

6. Movement (d)  BillPayment (g): (1:1)
   Each bill payment is a specific type of movement.

7. Movement (d) Credit (h): (1:1)
   Each loan is a specific type of movement.

8. Account (b) Pocket(i): (1:N)

   A user can have zero or multiple pockets, but each pocket is associated with only one account.

9. Account (b) Cushion(j): (1:1)

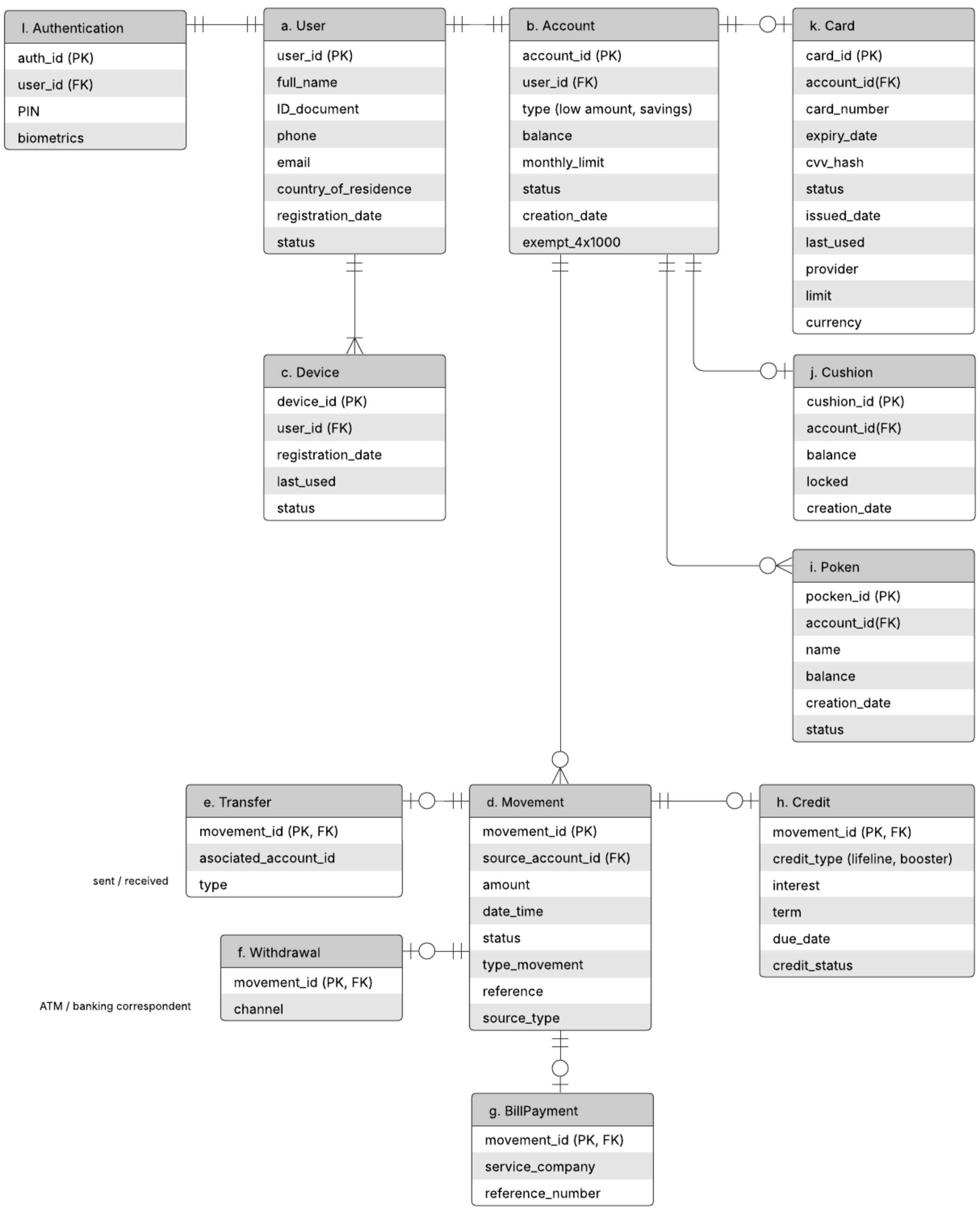
   A user can have zero or one cushion, but each cushion is associated with only one account.

10. Account (b) Card(k): (1:1)

    A user can have zero or one card, but each card belongs to a single account.

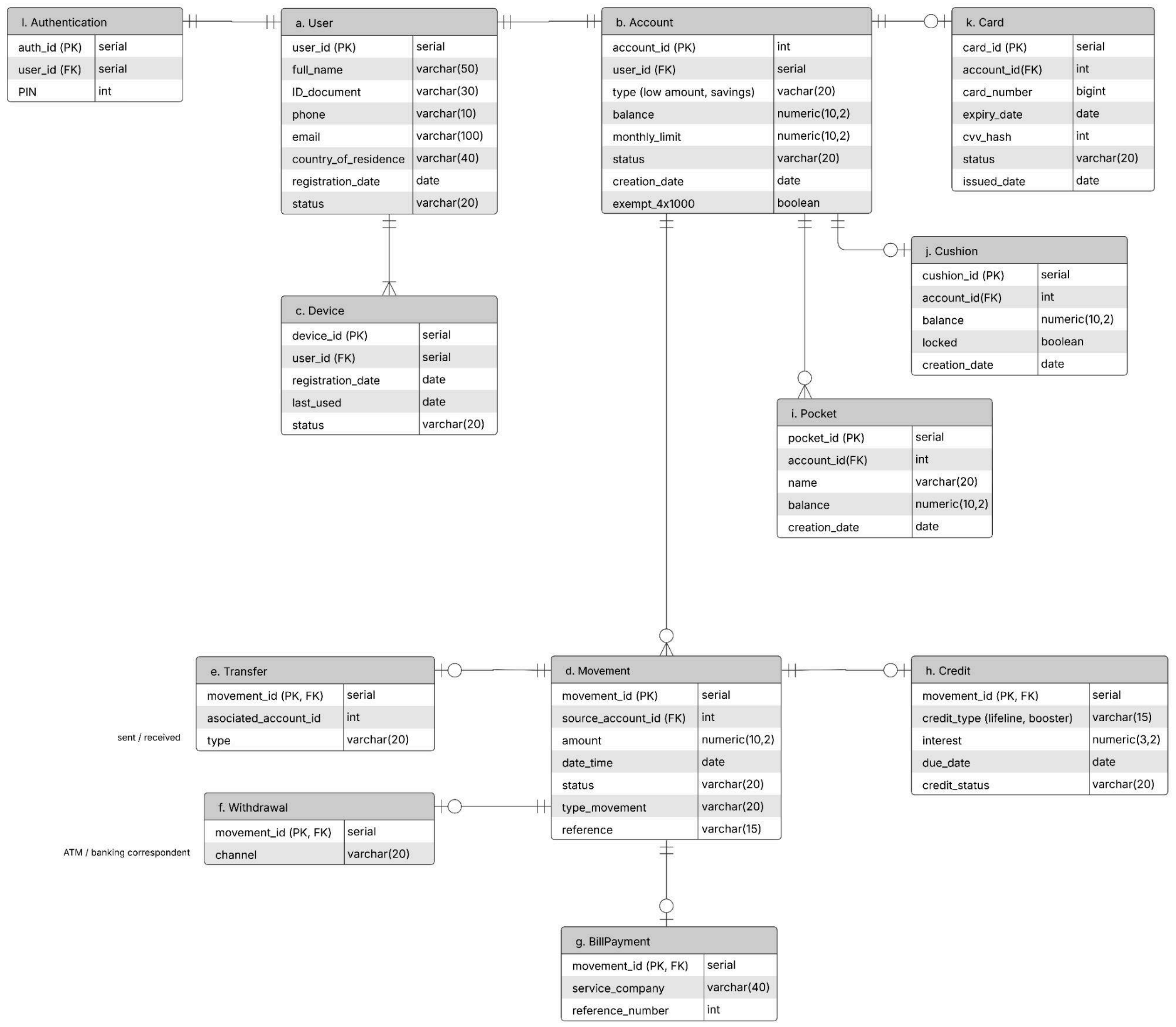
11. User (a) Authentication (l): (1:1)
    Each user has a unique authentication record, and each authentication belongs to only one user.

6. First Entity-Relationship Model Draw



Step (7-8): Our modeling has no many-to-many relationships.

9. Get Data-Structure Entity-Relationship Model



l. Authentication	
auth_id (PK)	serial
user_id (FK)	serial
PIN	int

a. User	
user_id (PK)	serial
full_name	varchar(50)
ID_document	varchar(30)
phone	varchar(10)
email	varchar(100)
country_of_residence	varchar(40)
registration_date	date
status	varchar(20)

b. Account	
account_id (PK)	int
user_id (FK)	serial
type (low amount, savings)	vachar(20)
balance	numeric(10,2)
monthly_limit	numeric(10,2)
status	varchar(20)
creation_date	date
exempt_4x1000	boolean

k. Card	
card_id (PK)	serial
account_id(FK)	int
card_number	bigint
expiry_date	date
cvv_hash	int
status	varchar(20)
issued_date	date

j. Cushion	
cushion_id (PK)	serial
account_id(FK)	int
balance	numeric(10,2)
locked	boolean
creation_date	date

c. Device	
device_id (PK)	serial
user_id (FK)	serial
registration_date	date
last_used	date
status	varchar(20)

i. Pocket	
pocket_id (PK)	serial
account_id(FK)	int
name	varchar(20)
balance	numeric(10,2)
creation_date	date

e. Transfer	
movement_id (PK, FK)	serial
asociated_account_id	int
type	varchar(20)

sent / received

d. Movement	
movement_id (PK)	serial
source_account_id (FK)	int
amount	numeric(10,2)
date_time	date
status	varchar(20)
type_movement	varchar(20)
reference	varchar(15)

h. Credit	
movement_id (PK, FK)	serial
credit_type (lifeline, booster)	varchar(15)
interest	numeric(3,2)
due_date	date
credit_status	varchar(20)

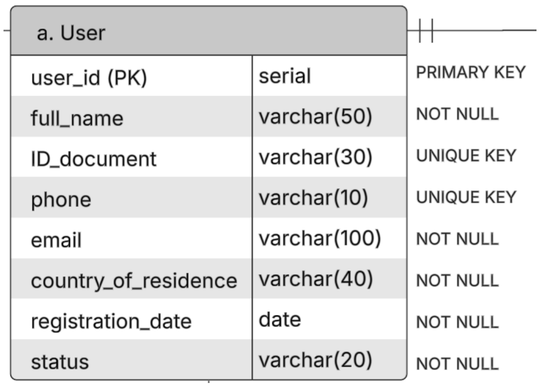
f. Withdrawal	
movement_id (PK, FK)	serial
channel	varchar(20)

ATM / banking correspondent

g. BillPayment	
movement_id (PK, FK)	serial
service_company	varchar(40)
reference_number	int

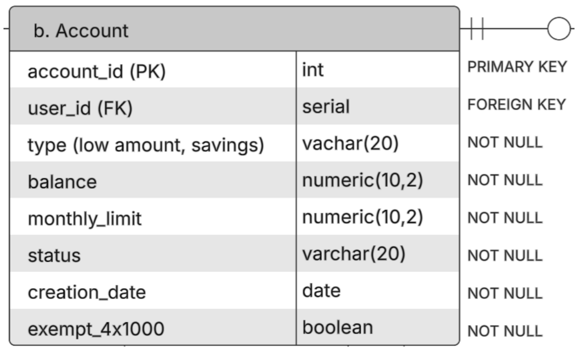
10. Define Constraints and Properties of Data

a. User



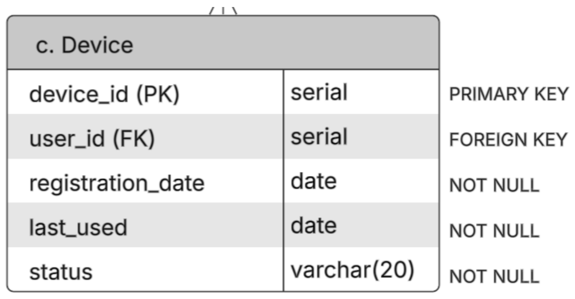
a. User		
user_id (PK)	serial	PRIMARY KEY
full_name	varchar(50)	NOT NULL
ID_document	varchar(30)	UNIQUE KEY
phone	varchar(10)	UNIQUE KEY
email	varchar(100)	NOT NULL
country_of_residence	varchar(40)	NOT NULL
registration_date	date	NOT NULL
status	varchar(20)	NOT NULL

b. Account



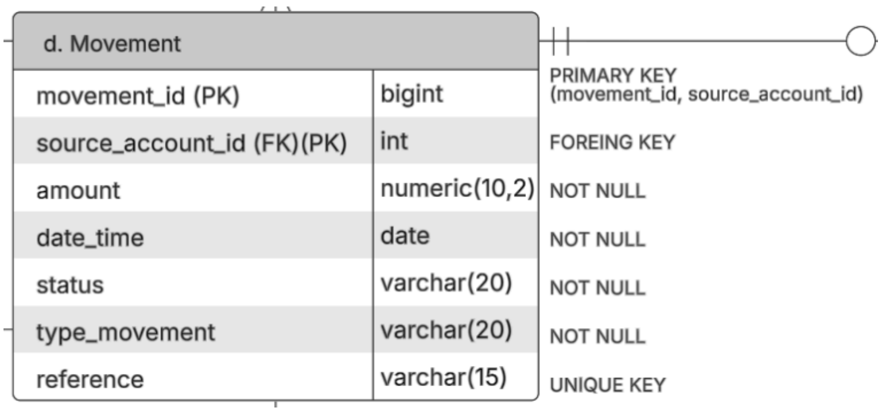
b. Account		
account_id (PK)	int	PRIMARY KEY
user_id (FK)	serial	FOREIGN KEY
type (low amount, savings)	vachar(20)	NOT NULL
balance	numeric(10,2)	NOT NULL
monthly_limit	numeric(10,2)	NOT NULL
status	varchar(20)	NOT NULL
creation_date	date	NOT NULL
exempt_4x1000	boolean	NOT NULL

c. Device



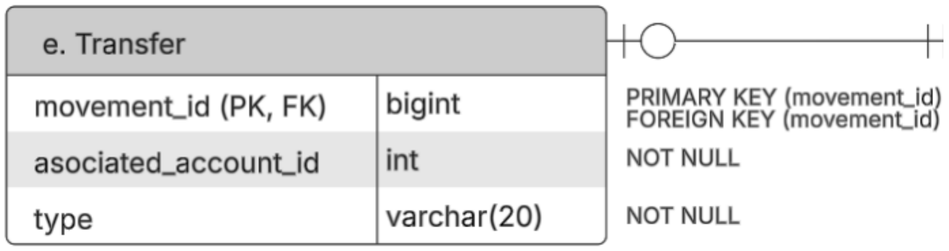
c. Device		
device_id (PK)	serial	PRIMARY KEY
user_id (FK)	serial	FOREIGN KEY
registration_date	date	NOT NULL
last_used	date	NOT NULL
status	varchar(20)	NOT NULL

d. Movement



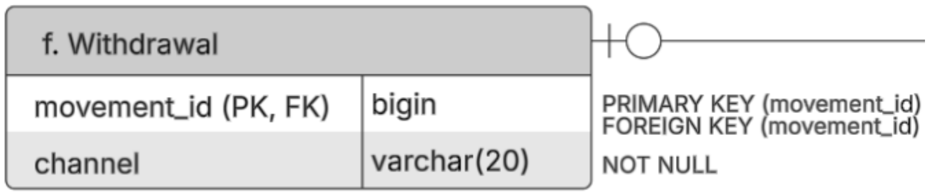
d. Movement		
movement_id (PK)	bigint	PRIMARY KEY (movement_id, source_account_id)
source_account_id (FK)(PK)	int	FOREING KEY
amount	numeric(10,2)	NOT NULL
date_time	date	NOT NULL
status	varchar(20)	NOT NULL
type_movement	varchar(20)	NOT NULL
reference	varchar(15)	UNIQUE KEY

e. Transfer



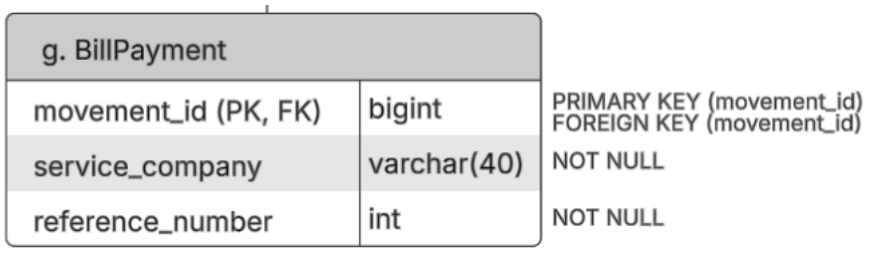
e. Transfer		
movement_id (PK, FK)	bigint	PRIMARY KEY (movement_id) FOREIGN KEY (movement_id)
asociated_account_id	int	NOT NULL
type	varchar(20)	NOT NULL

f. Withdrawal



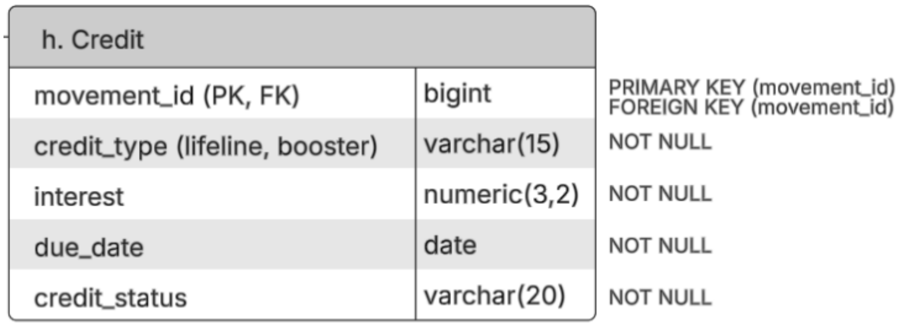
f. Withdrawal		
movement_id (PK, FK)	bigin	PRIMARY KEY (movement_id) FOREIGN KEY (movement_id)
channel	varchar(20)	NOT NULL

g. BillPayment



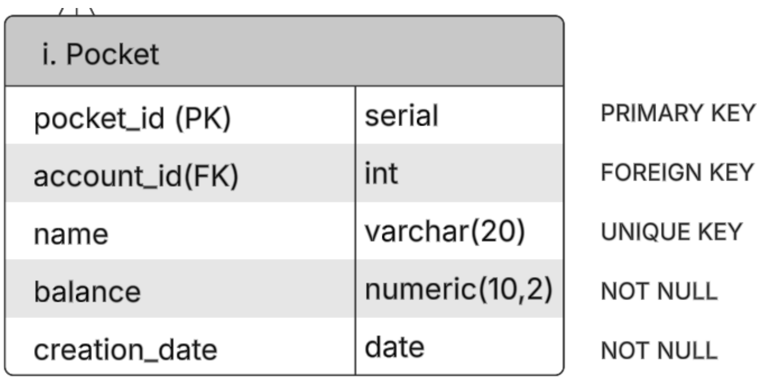
g. BillPayment		
movement_id (PK, FK)	bigint	PRIMARY KEY (movement_id) FOREIGN KEY (movement_id)
service_company	varchar(40)	NOT NULL
reference_number	int	NOT NULL

h. Credit



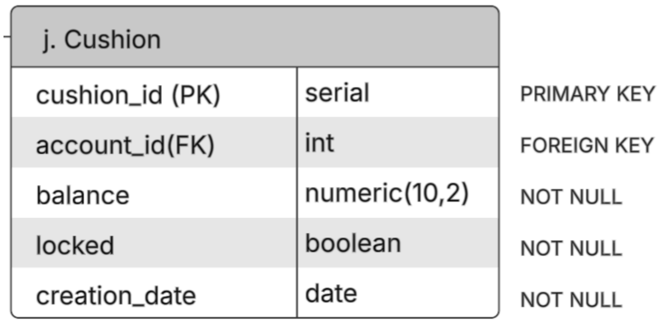
h. Credit		
movement_id (PK, FK)	bigint	PRIMARY KEY (movement_id) FOREIGN KEY (movement_id)
credit_type (lifeline, booster)	varchar(15)	NOT NULL
interest	numeric(3,2)	NOT NULL
due_date	date	NOT NULL
credit_status	varchar(20)	NOT NULL

i. Pocket



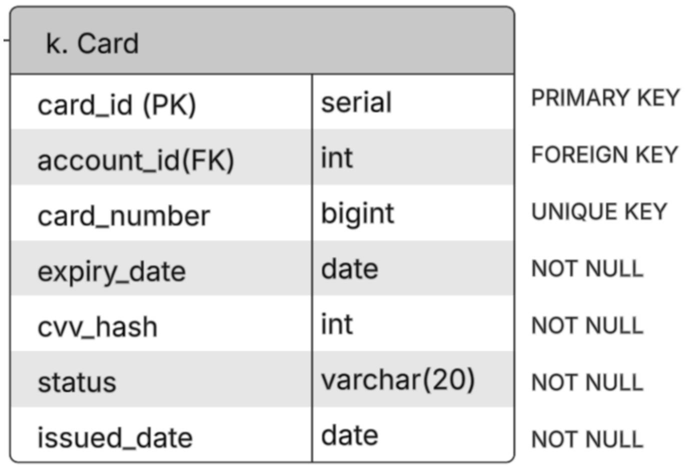
i. Pocket		
pocket_id (PK)	serial	PRIMARY KEY
account_id(FK)	int	FOREIGN KEY
name	varchar(20)	UNIQUE KEY
balance	numeric(10,2)	NOT NULL
creation_date	date	NOT NULL

j. Cushion



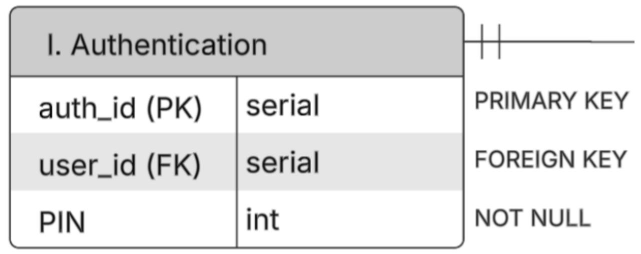
j. Cushion		
cushion_id (PK)	serial	PRIMARY KEY
account_id(FK)	int	FOREIGN KEY
balance	numeric(10,2)	NOT NULL
locked	boolean	NOT NULL
creation_date	date	NOT NULL

k. Card



k. Card		
card_id (PK)	serial	PRIMARY KEY
account_id(FK)	int	FOREIGN KEY
card_number	bigint	UNIQUE KEY
expiry_date	date	NOT NULL
cvv_hash	int	NOT NULL
status	varchar(20)	NOT NULL
issued_date	date	NOT NULL

l. Authentication



l. Authentication		
auth_id (PK)	serial	PRIMARY KEY
user_id (FK)	serial	FOREIGN KEY
PIN	int	NOT NULL

REFERENCES

- D. A. Ospina Henao, *“Mediante Nequi hacemos una marca de más de 38 millones de transacciones diarias”*, La República, 20 de enero de 2025. [Online]. Disponible: https://www.larepublica.co/finanzas/entrevista-con-andres-vasquez-echeverri-ceo-de-nequi-sobre-las-transacciones-en-colombia-4039379/
- PostgreSQL Global Development Group, *PostgreSQL Performance: Benchmarks and Use Cases*, 2023. [Online]. Disponible: https://www.postgresql.org/docs/current/performance-tips.html
- MongoDB Inc., *MongoDB Architecture Guide: Sharding and Scaling*, MongoDB Docs, 2024. [Online]. Disponible: https://www.mongodb.com/docs/manual/sharding/
- A. Jain, *Scaling PostgreSQL for High-Traffic Applications*, Percona, 2022. [En línea]. Disponible: https://www.percona.com/blog/scaling-postgresql-high-traffic-apps/
- MongoDB Inc., *MongoDB vs. Relational Databases*, MongoDB Documentation, 2023. [En línea]. Disponible: https://www.mongodb.com/compare/mongodb-vs-relational-databases