**Comparação e Análise de Modelos de Banco de Dados**

**Sumário do Documento**

1. **Visão Geral**
2. **Diferenças Arquiteturais Geral.**
   1. **Convenções de nomenclatura**
   2. **Abordagem de logging**
3. **Principais diferenças estruturais**
   1. **Gestão de Usuário**
   2. **Tratamento de pagamento**
   3. **Recursos de Jogo**
4. **Análise de Normalização**
   1. **Normalização do Modelo A**
   2. **Normalização do Modelo B**
5. **Recursos de Segurança**
   1. **Modelo A**
   2. **Modelo B**
6. **Considerações de Performance**
   1. **Modelo A**
   2. **Modelo B**
7. **Detalhamento técnico das tabelas**
   1. **app\_user**
   2. **streak**
   3. **theme**
   4. **plan**
   5. **powerup**
   6. **app\_user\_theme**
   7. **app\_user\_powerup**
   8. **plan\_payment**
   9. **user\_role**
8. **Detalhamento técnico das triggers**
   1. **Estrutura da Tabela de Log theme\_log**
   2. **Função de Log e Trigger para a Tabela theme**
   3. **Trigger para Registrar Alterações**
   4. **Validação de Dados – Implementação da Validação de Formato de Email**
   5. **Encriptação de Senha com Trigger e Procedure**
   6. **Automação de Datas com Procedures e Triggers**
9. **Extensibilidade**
   1. **Modelo A**
   2. **Modelo B**
10. **Recomendações**
    1. **Estratégia de Logging:**
    2. **Normalização:**
    3. **Integração de Recursos:**
    4. **Segurança:**
    5. **Padronização:**

**1. Visão Geral**

**Para atender à solicitação e assegurar uma documentação completa e detalhada do projeto, abordaremos cada instância de código com explicações minuciosas. O objetivo é oferecer um guia técnico compreensivo que será útil para consulta futura por pesquisadores e especialistas em modelagem de dados. A seção comparativa (Seção 6) destacará de forma específica as diferenças entre os modelos, incluindo aspectos de linguagem e normalização.**

* **Modelo A: O primeiro modelo (contendo app\_user, streak, theme, etc.)**
* **Modelo B: O segundo modelo (Delfis\_logico1G)**

**2. Diferenças Arquiteturais Gerais**

**2.1 Convenções de Nomenclatura**

* Modelo A utiliza snake\_case com termos em inglês (app\_user, plan\_payment)
* Modelo B utiliza PascalCase com termos em português (Usuario, Pagamento)
* Modelo B inclui mais elementos em português nos nomes das colunas (dt\_nasc, preco\_moedas)

**2.2 Abordagem de Logging**

* Modelo A implementa logging abrangente através de:
  + Tabelas de log separadas para cada tabela principal
  + Triggers para operações INSERT, UPDATE, DELETE
  + Captura do tipo de operação, timestamp e usuário
* Modelo B utiliza uma abordagem mais simples de rastreamento:
  + Flags booleanas (is\_updated, is\_deleted) nas tabelas principais
  + Sem tabelas de log separadas ou trilha de auditoria

**3. Principais Diferenças Estruturais**

**3.1 Gestão de Usuários**

**Modelo A:**

**CREATE TABLE app\_user (**

**username varchar(20) NOT NULL,**

**password text NOT NULL,**

**email varchar(100) NOT NULL,**

**...**

**)**

**Modelo B:**

**CREATE TABLE Usuario (**

**nome varchar(30),**

**senha text NOT NULL,**

**email varchar(100) NOT NULL,**

**...**

**)**

**Diferenças principais:**

* Modelo B separa usuários administrativos em uma tabela Adm separada
* Modelo A tem limites mais rigorosos no tamanho do nome de usuário (20 vs 30 caracteres)
* Modelo B inclui relacionamentos de ranking diretamente na tabela de usuário

**3.2 Tratamento de Pagamentos**

**Modelo A:**

**CREATE TABLE plan\_payment (**

**price numeric(10, 2) NOT NULL,**

**payment\_timestamp timestamp,**

**expiration\_timestamp timestamp,**

**...**

**)**

**Modelo B:**

**CREATE TABLE Pagamento (**

**preco numeric(10, 2) NOT NULL,**

**dt\_inicial timestamp NOT NULL,**

**dt\_expirado timestamp NOT NULL,**

**pix varchar(100) NOT NULL,**

**deb\_card varchar(100),**

**cred\_card varchar(100),**

**boleto varchar(100),**

**...**

**)**

**Diferenças principais:**

* Modelo B inclui métodos de pagamento específicos (PIX, cartão de débito, cartão de crédito, boleto)
* Modelo B tem uma tabela Moedas separada ligando pagamentos a usuários
* Modelo A integra o tratamento de pagamentos diretamente com planos

**3.3 Recursos de Jogo**

**Modelo A foca em:**

* Temas
* Power-ups
* Sequências (Streaks)
* Planos

**Modelo B adiciona:**

* Tabela explícita de Jogos
* Rastreamento de Partidas
* Sistema de Ranking
* Rastreamento mais detalhado da duração de power-ups

**4. Análise de Normalização**

**4.1 Normalização do Modelo A**

* Atinge 3FN (Terceira Forma Normal) através de:
  + Tabelas separadas para diferentes entidades (app\_user, theme, plan)
  + Tabelas de junção para relacionamentos muitos-para-muitos (app\_user\_theme, app\_user\_powerup)
  + Sem dependências transitivas
  + Relacionamentos claros de chave primária e estrangeira[

**4.2 Normalização do Modelo B**

* Parcialmente normalizado (2FN a 3FN) com alguma desnormalização:
  + Contém alguns dados redundantes (ex: informações de ranking na tabela Usuario)
  + Usa tabelas de junção para relacionamentos muitos-para-muitos (Moedas\_Temas, Moedas\_Powerup)
  + Inclui algumas colunas computadas (ex: tempo\_jogado na tabela Partida)

**5. Recursos de Segurança**

**5.1 Modelo A**

* Procedimento e trigger de criptografia de senha
* Função de validação de email
* Logging abrangente de auditoria
* Restrições rigorosas de tipo de dados

**5.2 Modelo B**

* Armazenamento básico de senha
* Gestão separada de usuários administradores
* Funcionalidade de exclusão lógica
* Rastreamento de status para entidades

**6. Considerações de Performance**

**6.1 Modelo A**

* Otimizado para rastreamento de auditoria
* Relacionamentos eficientes de chave estrangeira
* Automação baseada em triggers para certas operações

**6.2 Modelo B**

* Inclui colunas computadas para performance
* Relacionamentos mais complexos entre entidades
* Índices adicionais através de restrições UNIQUE

**7. Configuração Completa de Tabelas de Log e Triggers**

**7.1 Tabela app\_user**

Armazena os dados básicos de cada usuário no sistema, incluindo dados pessoais e associações a planos e papéis.

*CREATE TABLE app\_user (*

*username VARCHAR(20) NOT NULL, -- Nome de usuário único e obrigatório*

*password TEXT NOT NULL, -- Senha do usuário, que será encriptada*

*email VARCHAR(100) NOT NULL, -- Email único e obrigatório*

*birth\_date DATE NOT NULL, -- Data de nascimento obrigatória*

*level INTEGER DEFAULT 1 NOT NULL CHECK(level >= 1), -- Nível do usuário, valor mínimo de 1*

*points INTEGER DEFAULT 0 NOT NULL CHECK(points >= 0), -- Pontos acumulados, valor mínimo de 0*

*coins INTEGER DEFAULT 0 NOT NULL CHECK(coins >= 0), -- Moedas acumuladas, valor mínimo de 0*

*picture\_url TEXT, -- URL da imagem de perfil do usuário*

*id SERIAL PRIMARY KEY NOT NULL, -- Identificador único e chave primária*

*fk\_plan\_id INTEGER NOT NULL, -- Chave estrangeira para associação com o plano do usuário*

*created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP NOT NULL, -- Data de criação do registro*

*updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP, -- Última atualização do registro*

*fk\_user\_role\_id INTEGER NOT NULL, -- Chave estrangeira para função do usuário*

*name VARCHAR(150) NOT NULL, -- Nome completo do usuário*

*UNIQUE (username, email) -- Restrição para garantir unicidade de username e email*

*);*

**Lógica e Restrições**:

**Campos Obrigatórios**: username, password, email, birth\_date, level, points, coins, fk\_plan\_id, fk\_user\_role\_id, e name são campos essenciais para o cadastro do usuário.

UNIQUE (username, email): Impede a duplicidade de usuários com o mesmo nome de usuário ou email.

DEFAULT **e** CHECK: Asseguram que level, points e coins possuam valores mínimos ou padrão.

**Referências com Chaves Estrangeiras**: fk\_plan\_id e fk\_user\_role\_id ligam o usuário a um plano e papel específicos.

**7.2 Tabela streak**

Armazena os dados de atividades contínuas dos usuários, como um registro de sequência de atividades (streak).

*CREATE TABLE streak (*

*id SERIAL PRIMARY KEY NOT NULL, -- Identificador único e chave primária*

*initial\_date DATE NOT NULL, -- Data de início do streak*

*final\_date DATE, -- Data de término do streak, ajustada automaticamente*

*fk\_app\_user\_id INTEGER NOT NULL -- Chave estrangeira para associação com o usuário*

*);*

**Lógica e Restrições**:

**Chave Primária (**id**)**: Garante um identificador único para cada streak.

**Campo** initial\_date: Marca o início do streak.

**Campo** final\_date: Ajustado por uma procedure para calcular a duração da sequência de atividades.

**Chave Estrangeira (**fk\_app\_user\_id**)**: Estabelece a conexão entre o streak e o usuário correspondente.

**7.3 Tabela theme**

Armazena temas de personalização que os usuários podem adquirir e utilizar.

*CREATE TABLE theme (*

*name VARCHAR(30) NOT NULL, -- Nome do tema, campo obrigatório*

*price INTEGER NOT NULL CHECK(price >= 0), -- Preço do tema, valor mínimo de 0*

*store\_picture\_url TEXT NOT NULL, -- URL da imagem do tema, campo obrigatório*

*id SERIAL PRIMARY KEY NOT NULL, -- Identificador único e chave primária*

*UNIQUE (name, store\_picture\_url) -- Restrição de unicidade para nome e URL de imagem*

*);*

**Lógica e Restrições**:

UNIQUE (name, store\_picture\_url): Evita a criação de temas duplicados com o mesmo nome e imagem.

CHECK(price >= 0): Garante que o preço do tema seja um valor positivo.

**Identificador Único (**id**)**: Cada tema é identificado exclusivamente.

**7.4 Tabela plan**

Armazena os planos de assinatura disponíveis para os usuários, incluindo detalhes como preço e descrição.

*CREATE TABLE plan (*

*name VARCHAR(20) NOT NULL, -- Nome do plano, campo obrigatório*

*price NUMERIC(10, 2) NOT NULL CHECK(price >= 0), -- Preço do plano, valor não negativo*

*description VARCHAR(500) NOT NULL, -- Descrição do plano, campo obrigatório*

*id SERIAL PRIMARY KEY NOT NULL, -- Identificador único e chave primária*

*UNIQUE (name, description) -- Restrição de unicidade para nome e descrição*

*);*

**Lógica e Restrições**:

UNIQUE (name, description): Garante que cada plano seja único em nome e descrição.

CHECK(price >= 0): Restringe o preço a valores não negativos.

**Identificador Único (**id**)**: Cada plano é associado a um ID único.

**7.5 Tabela powerup**

Armazena informações sobre os poderes adicionais que os usuários podem adquirir e utilizar.

*CREATE TABLE powerup (*

*name VARCHAR(30) NOT NULL, -- Nome do power-up, campo obrigatório*

*price INTEGER NOT NULL CHECK(price >= 0), -- Preço do power-up, valor mínimo de 0*

*store\_picture\_url TEXT NOT NULL, -- URL da imagem do power-up, campo obrigatório*

*id SERIAL PRIMARY KEY NOT NULL, -- Identificador único e chave primária*

*UNIQUE (name, store\_picture\_url) -- Restringe duplicidade de nome e URL de imagem*

*);*

**Lógica e Restrições**:

UNIQUE (name, store\_picture\_url): Assegura a unicidade de nome e URL de imagem para cada power-up.

CHECK(price >= 0): Garante que o preço do power-up seja positivo.

**Identificador Único (**id**)**: Define um identificador exclusivo para cada power-up.

**7.6 Tabela app\_user\_theme**

Relaciona usuários com temas, incluindo um indicador se o tema está atualmente em uso pelo usuário.

*CREATE TABLE app\_user\_theme (*

*fk\_theme\_id INTEGER NOT NULL, -- Chave estrangeira para o tema*

*fk\_app\_user\_id INTEGER NOT NULL, -- Chave estrangeira para o usuário*

*is\_in\_use BOOLEAN NOT NULL, -- Indicador se o tema está em uso pelo usuário*

*id SERIAL PRIMARY KEY NOT NULL -- Identificador único para a associação*

*);*

**Lógica e Restrições**:

**Campos** fk\_theme\_id **e** fk\_app\_user\_id: Associam um tema específico a um usuário.

**Campo** is\_in\_use: Indica se o tema está atualmente em uso pelo usuário.

**Identificador Único (**id**)**: Cada associação entre usuário e tema é única.

**7.7 Tabela app\_user\_powerup**

Relaciona usuários com power-ups adquiridos.

*CREATE TABLE app\_user\_powerup (*

*fk\_app\_user\_id INTEGER NOT NULL, -- Chave estrangeira para o usuário*

*fk\_powerup\_id INTEGER NOT NULL, -- Chave estrangeira para o power-up*

*id SERIAL PRIMARY KEY NOT NULL -- Identificador único para a associação*

*);*

**Lógica e Restrições**:

**Campos** fk\_app\_user\_id **e** fk\_powerup\_id: Associam cada power-up a um usuário.

**Identificador Único (**id**)**: Garante a unicidade de cada associação entre usuário e power-up.

**7.8 Tabela plan\_payment**

Armazena os pagamentos feitos pelos usuários para adquirir planos, incluindo a data de pagamento e expiração.

*CREATE TABLE plan\_payment (*

*id SERIAL PRIMARY KEY NOT NULL, -- Identificador único para o pagamento*

*price NUMERIC(10, 2) NOT NULL CHECK(price > 0), -- Valor do pagamento, deve ser positivo*

*payment\_timestamp TIMESTAMP DEFAULT CURRENT\_TIMESTAMP NOT NULL, -- Data e hora do pagamento*

*expiration\_timestamp TIMESTAMP NOT NULL, -- Data de expiração do plano*

*fk\_plan\_id INTEGER NOT NULL, -- Chave estrangeira para o plano*

*fk\_app\_user\_id INTEGER NOT NULL -- Chave estrangeira para o usuário*

*);*

**Lógica e Restrições**:

**Campo** price: Valor pago, que deve ser maior que zero, garantido pela restrição CHECK.

**Campos** payment\_timestamp **e** expiration\_timestamp: Registram a data de pagamento e a data de expiração do plano.

**Chaves Estrangeiras** (fk\_plan\_id e fk\_app\_user\_id)\*\*: Associam o pagamento ao respectivo plano e usuário.

**7.9 Tabela user\_role**

Define os diferentes papéis dos usuários no sistema, como "administrador" ou "usuário comum".

*CREATE TABLE user\_role (*

*id SERIAL PRIMARY KEY, -- Identificador único e chave primária*

*name VARCHAR(20) NOT NULL, -- Nome do papel, campo obrigatório*

*UNIQUE(name) -- Restringe duplicidade de nome de papel*

*);*

**Lógica e Restrições**:

**Campo** name: Define o nome do papel, como "admin" ou "usuário", assegurando que cada papel tenha um nome único.

UNIQUE(name): Garante que cada nome de papel no sistema seja único.

**Identificador Único (**id**)**: Cada papel é identificado exclusivamente por um id.

Essas tabelas, restrições e lógica de funcionamento asseguram que o banco de dados opere com integridade, segurança e eficiência. Cada tabela foi detalhada com suas restrições de campo e lógica de relacionamento, conforme solicitado.

**8. Configuração Completa de Tabelas de Log e Triggers**

Para registrar mudanças nas tabelas principais (como app\_user, theme, plan, e powerup), criamos tabelas de log que armazenam cada operação executada, como INSERT, UPDATE, e DELETE. As tabelas de log, juntamente com funções de triggers, capturam dados como o tipo de operação, data e hora da operação, e o usuário que realizou a modificação.

**Exemplo: Log Completo da Tabela theme**

1. **Estrutura da Tabela de Log theme\_log**:

A tabela theme\_log mantém um registro de cada alteração feita na tabela theme, incluindo o preço, o nome e a URL da imagem do tema.

*CREATE TABLE theme\_log (*

*name VARCHAR(30) NOT NULL, -- Nome do tema, campo obrigatório*

*price INTEGER NOT NULL CHECK(price >= 0), -- Preço do tema, campo obrigatório com mínimo de 0*

*store\_picture\_url TEXT NOT NULL, -- URL da imagem do tema, campo obrigatório*

*id SERIAL PRIMARY KEY NOT NULL, -- Chave primária do log*

*Operation VARCHAR(10), -- Tipo de operação: INSERT, UPDATE, DELETE*

*UpdatedOn TIMESTAMP, -- Data e hora da operação*

*UpdatedBy VARCHAR(50) -- Usuário que realizou a operação*

*);*

**Lógica**:

* + A tabela registra dados principais da tabela theme, além dos campos Operation, UpdatedOn e UpdatedBy para capturar o tipo de operação, o momento em que ocorreu e o usuário responsável.
  + O campo Operation ajuda a identificar a natureza da operação (inserção, atualização ou exclusão) para futuras auditorias.

1. **Função de Log e Trigger para a Tabela theme**:

A função log\_theme\_changes usa a variável TG\_OP para identificar o tipo de operação realizada (inserção, atualização ou exclusão) e registra a ação com base nas novas informações.

CREATE OR REPLACE FUNCTION log\_theme\_changes()

RETURNS TRIGGER AS $$

BEGIN

IF (TG\_OP = 'DELETE') THEN

INSERT INTO theme\_log (name, price, store\_picture\_url, id, Operation, UpdatedOn, UpdatedBy)

VALUES (OLD.name, OLD.price, OLD.store\_picture\_url, OLD.id, 'DELETE', NOW(), current\_user);

ELSIF (TG\_OP = 'UPDATE') THEN

INSERT INTO theme\_log (name, price, store\_picture\_url, id, Operation, UpdatedOn, UpdatedBy)

VALUES (NEW.name, NEW.price, NEW.store\_picture\_url, NEW.id, 'UPDATE', NOW(), current\_user);

ELSE

INSERT INTO theme\_log (name, price, store\_picture\_url, id, Operation, UpdatedOn, UpdatedBy)

VALUES (NEW.name, NEW.price, NEW.store\_picture\_url, NEW.id, 'INSERT', NOW(), current\_user);

END IF;

RETURN NULL;

END;

$$ LANGUAGE plpgsql;

1. **Trigger para Registrar Alterações**:

A trigger theme\_trigger ativa a função de log após cada operação (inserção, atualização ou exclusão) na tabela theme.

CREATE TRIGGER theme\_trigger

AFTER INSERT OR UPDATE OR DELETE ON theme

FOR EACH ROW EXECUTE FUNCTION log\_theme\_changes();

**Lógica**:

* + A trigger é executada após cada operação (AFTER INSERT OR UPDATE OR DELETE) na tabela theme.
  + OLD e NEW são usados para capturar valores antes e depois da operação, respectivamente, para fins de auditoria completa.

**3. Validação de Dados – Implementação da Validação de Formato de Email**

Para garantir que o email inserido em app\_user esteja em um formato válido, uma função de validação validate\_email foi criada. Ela verifica se o email corresponde ao padrão estabelecido e é chamada por uma trigger antes de qualquer inserção ou atualização.

**Implementação da Validação de Email**

1. **Função de Validação**:

CREATE OR REPLACE FUNCTION validate\_email()

RETURNS TRIGGER AS $$

BEGIN

IF NEW.email !~ '^[A-Za-z0-9.\_%-]+@[A-Za-z0-9.-]+[.][A-Za-z]+$' THEN

RAISE EXCEPTION 'Formato de email inválido';

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

**Lógica**:

* + A função utiliza uma expressão regular para validar o email.
  + Caso o email não corresponda ao padrão (exemplo: usuario@dominio.com), a função gera uma exceção e a operação é cancelada.

1. **Trigger de Validação de Email**:

CREATE TRIGGER email\_validation\_trigger

BEFORE INSERT OR UPDATE ON app\_user

FOR EACH ROW EXECUTE FUNCTION validate\_email();

**Lógica da Trigger**:

* + Ativada antes de cada inserção ou atualização (BEFORE INSERT OR UPDATE), garantindo que o email seja validado antes de qualquer alteração definitiva.

**4. Encriptação de Senha com Trigger e Procedure**

Para proteger as senhas, uma procedure chamada encrypt\_password\_proc substitui cada letra da senha por uma letra subsequente, criando uma criptografia básica. Essa procedure é ativada por uma trigger antes de salvar a senha.

**Detalhamento da Encriptação de Senha**

1. **Procedure para Encriptação**:

CREATE OR REPLACE PROCEDURE encrypt\_password\_proc(INOUT p TEXT) LANGUAGE plpgsql AS $$

DECLARE

lower\_case\_alphabet TEXT[] := '{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z}';

upper\_case\_alphabet TEXT[] := '{A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z}';

char TEXT;

next\_char TEXT;

index INT;

result TEXT := '';

BEGIN

FOR i IN 1..LENGTH(p) LOOP

char := SUBSTRING(p FROM i FOR 1);

IF char = ANY(lower\_case\_alphabet) THEN

index := array\_position(lower\_case\_alphabet, char);

next\_char := lower\_case\_alphabet[(index % 26) + 1];

result := result || next\_char;

ELSIF char = ANY(upper\_case\_alphabet) THEN

index := array\_position(upper\_case\_alphabet, char);

next\_char := upper\_case\_alphabet[(index % 26) + 1];

result := result || next\_char;

ELSE

result := result || char;

END IF;

END LOOP;

p := result;

END;

$$;

**Lógica de Encriptação**:

* + A função percorre cada caractere da senha (FOR i IN 1..LENGTH(p)).
  + Letras minúsculas e maiúsculas são substituídas pela letra seguinte no alfabeto, criando uma simples encriptação.
  + Caracteres não alfabéticos são mantidos sem alterações.

1. **Trigger para Ativar a Encriptação**:

CREATE OR REPLACE FUNCTION trigger\_encrypt\_password() RETURNS TRIGGER AS $$

BEGIN

CALL encrypt\_password\_proc(NEW.password);

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER before\_insert\_encrypt

BEFORE INSERT ON app\_user

FOR EACH ROW

EXECUTE FUNCTION trigger\_encrypt\_password();

**Lógica da Trigger**:

* + A trigger chama encrypt\_password\_proc antes de cada inserção em app\_user, garantindo que a senha seja sempre encriptada antes de ser armazenada.

**5. Automação de Datas com Procedures e Triggers**

Procedures automáticas ajustam campos de data, como final\_date em streak e expiration\_timestamp em plan\_payment, garantindo o correto registro de períodos e datas de expiração.

**Exemplo: Procedure e Trigger para Atualização de Data Final em streak**

1. **Procedure para final\_date**:

CREATE OR REPLACE PROCEDURE update\_final\_date\_streak(p\_id INTEGER, p\_initial\_date DATE)

LANGUAGE plpgsql AS $$

BEGIN

UPDATE streak

SET final\_date = p\_initial\_date + INTERVAL '7 days'

WHERE id = p\_id;

END;

$$;

1. **Trigger de Atualização**:

CREATE TRIGGER after\_insert\_streak

AFTER INSERT ON streak

FOR EACH ROW

EXECUTE FUNCTION trigger\_update\_final\_date\_streak();

**Lógica**:

* + final\_date é definido como sete dias após initial\_date, automatizando a duração dos streaks.

**9. Extensibilidade**

**9.1 Modelo A**

* **Projetado para fácil adição de novos recursos**
* **Estrutura modular com limites claros**
* **Forte separação de responsabilidades**

**9.2 Modelo B**

* **Relacionamentos mais fortemente acoplados**
* **Recursos específicos de jogos integrados na estrutura central**
* **Mais complexo de modificar devido a interdependências**

**10. Recomendações**

1. **Estratégia de Logging:** 
   * **Adotar a abordagem abrangente de logging do Modelo A**
   * **Considerar adicionar campos de timestamp para created\_at/updated\_at**
2. **Normalização:** 
   * **Revisar a integração de ranking do Modelo B**
   * **Considerar separar métodos de pagamento em uma tabela separada**
3. **Integração de Recursos:** 
   * **Considerar incorporar o rastreamento de sessão de jogo do Modelo B**
   * **Avaliar a necessidade de gestão separada de usuários administradores**
4. **Segurança:** 
   * **Implementar a criptografia de senha do Modelo A no Modelo B**
   * **Adicionar validação de email em ambos os modelos**
5. **Padronização:** 
   * **Escolher uma convenção de nomenclatura consistente**
   * **Padronizar em uma língua (inglês ou português)**
   * **Alinhar definições de tipo de dados entre os modelos**