Programação Distribuída - 2025/2026

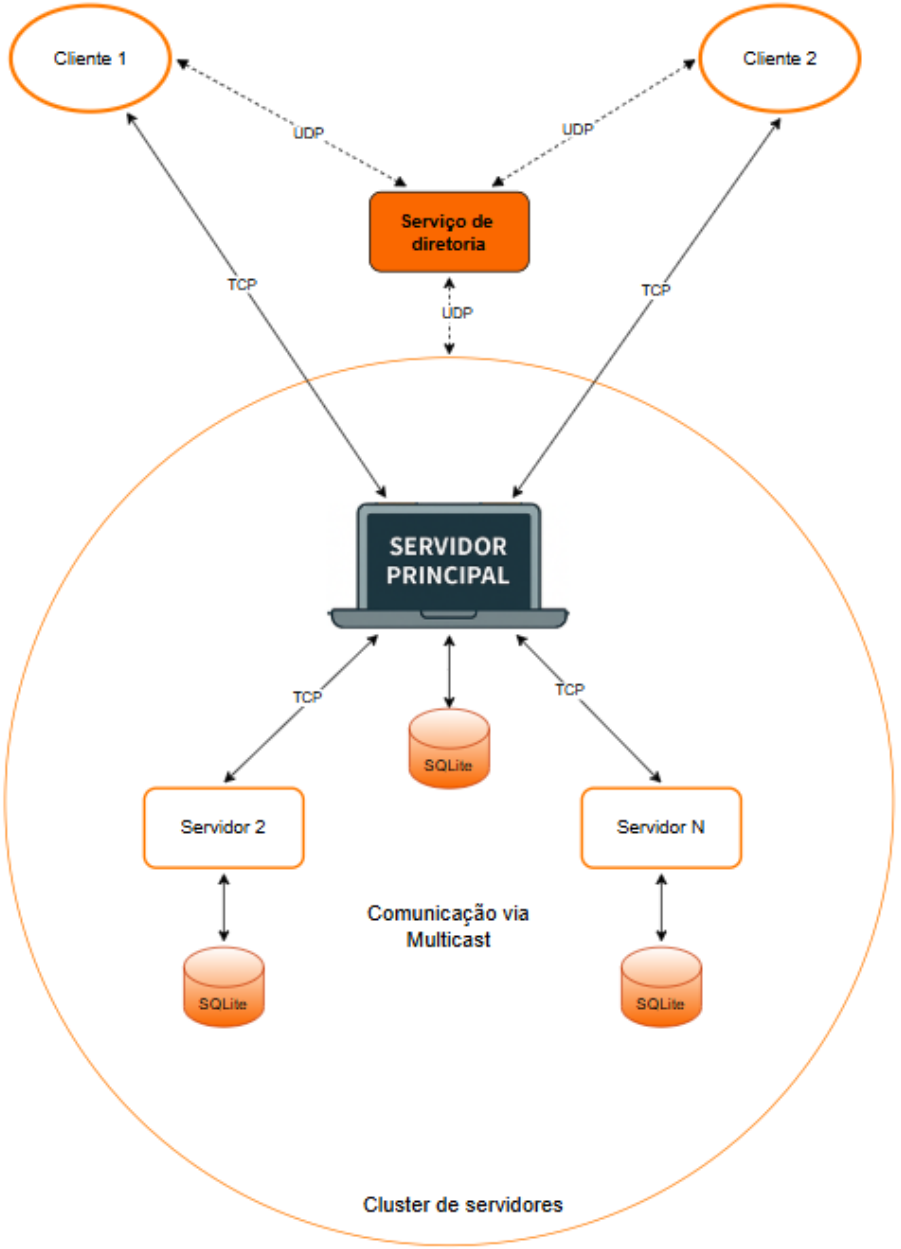
Dario Santos 2021110772
Evilania Lima 2021125566
José Bastos 2021127160

Introdução

- **Sistema distribuído tolerante a falhas**, com Servidor Primário e Servidores Secundários
- **Dois perfis de utilizador**: Docente e Estudante
- **Comunicação híbrida**: TCP, UDP e Multicast
- **Base de dados SQLite local**, com sincronização automática entre servidores
- **Promoção automática de um servidor secundário** quando ocorre uma falha

Arquitetura Geral

- Clientes (aplicação JavaFX)
- Serviço de Diretoria
- Servidor Primário
- Servidores Secundários
- Sincronização e tolerância a falhas



Serviço de diretoria

- Mantém lista ordenada de servidores ativos (o primeiro é o Primário)
- Responsável pela eleição do Primário
- Comandos: REGISTER (servidores) e REQUEST_SERVER (clientes)
- Fornece ao cliente o IP e porto TCP do servidor Primário
- Heartbeats e timeout de 17s (remove servidor inativo)
- Escuta UDP no porto indicado na linha de comando
- Ignora heartbeats de servidores não registados

Servidores e Sincronização

- Registo no Diretório
- Criação ou reutilização da BD (Primário)
- Download inicial da BD (Secundário)
- Heartbeats Multicast
- Promoção automática em caso de falha

Concorrência e Threads

Servidor

- ClientPool: atende múltiplos clientes TCP em paralelo.
- Threads de Multicast: envio e receção de heartbeats + queries SQL.

Cliente

- Thread dedicada para notificações assíncronas (mantém a sessão e atualiza a vista).

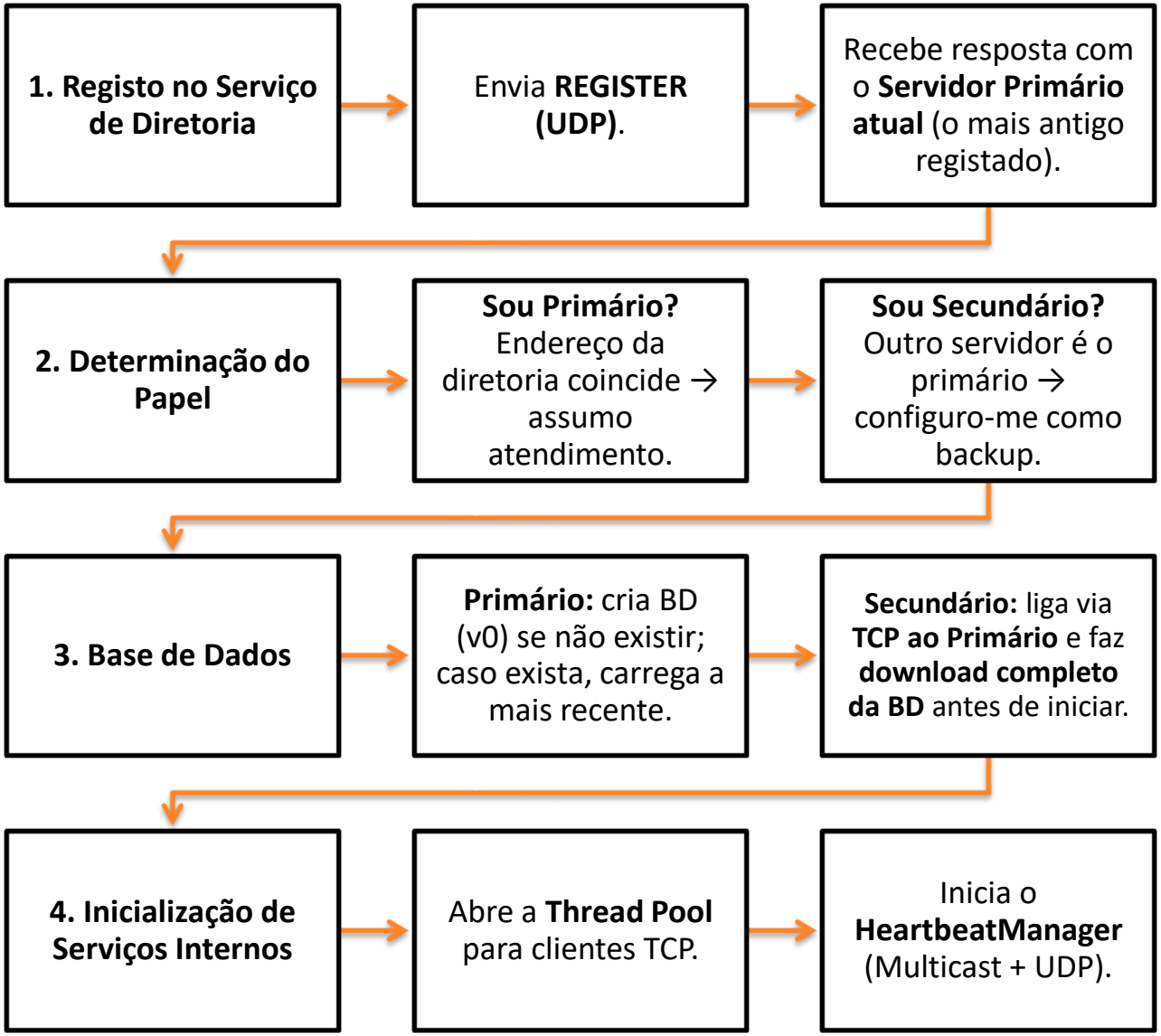
Sincronização

- ReentrantReadWriteLock: leituras concorrentes, escrita exclusiva.
- Diretoria: lista de servidores sincronizada para evitar condições de corrida.

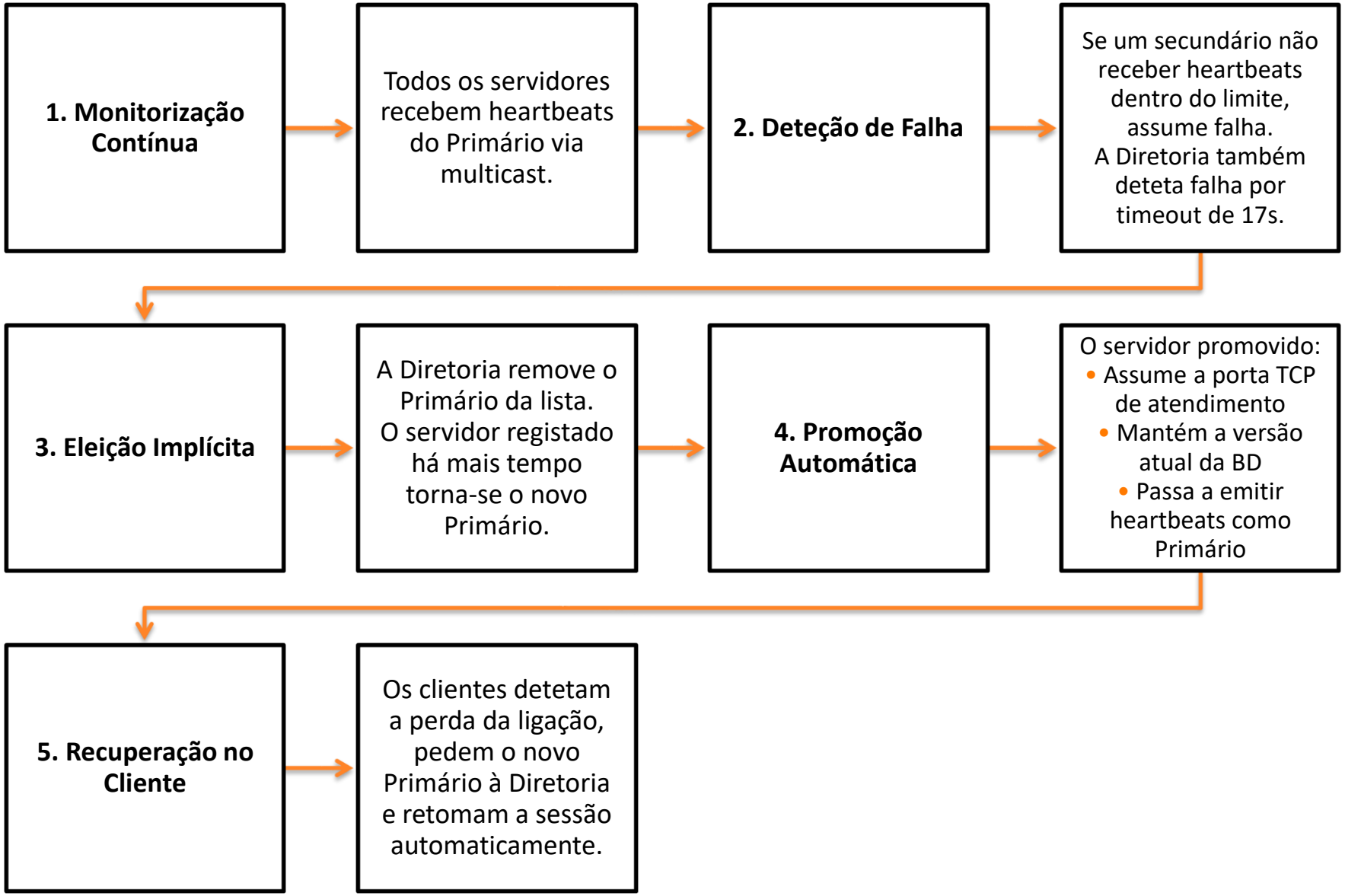
Interface (JavaFX)

- GUI atualizada com Platform.runLater() de forma segura.

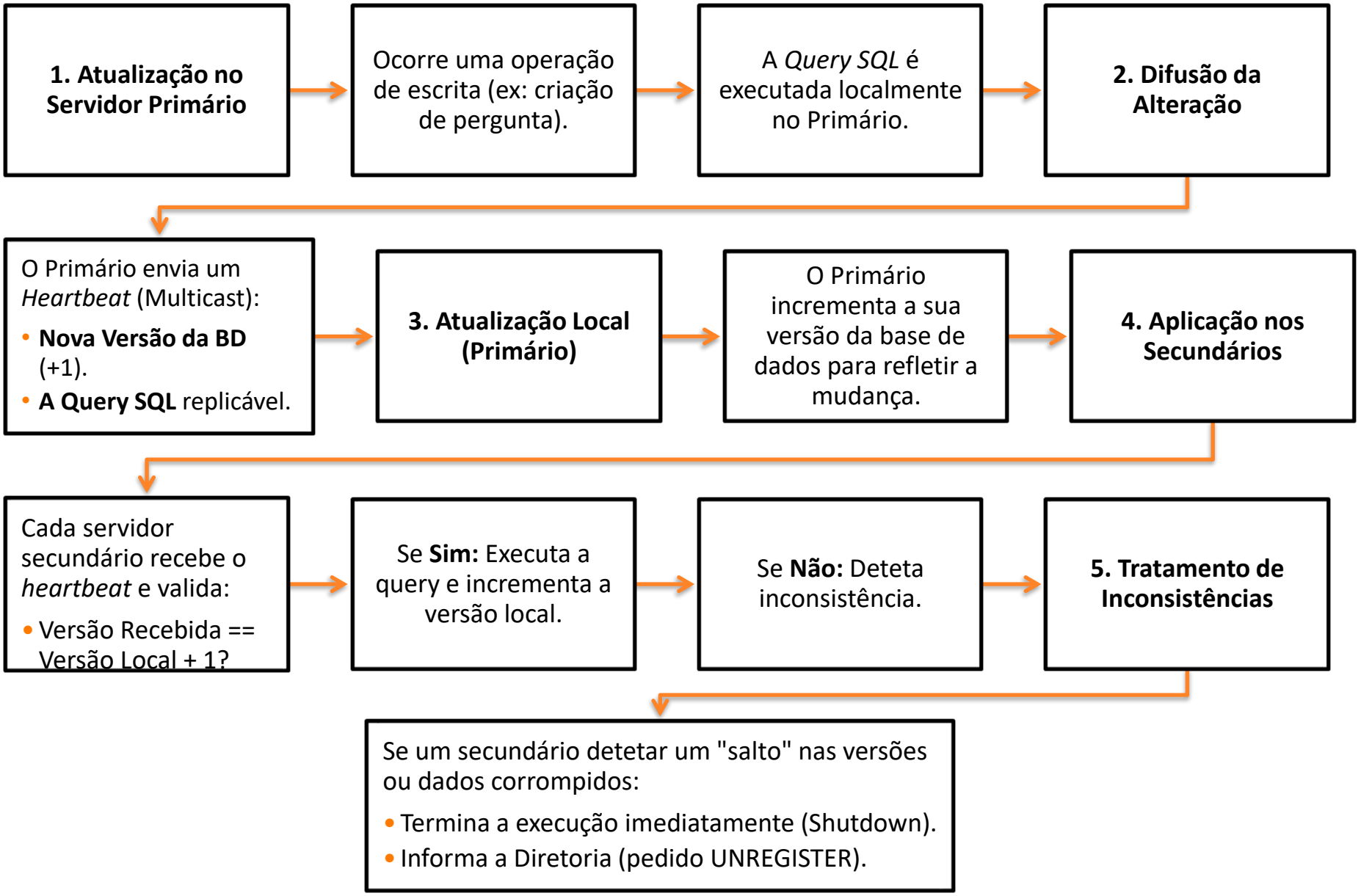
Fluxo de Arranque



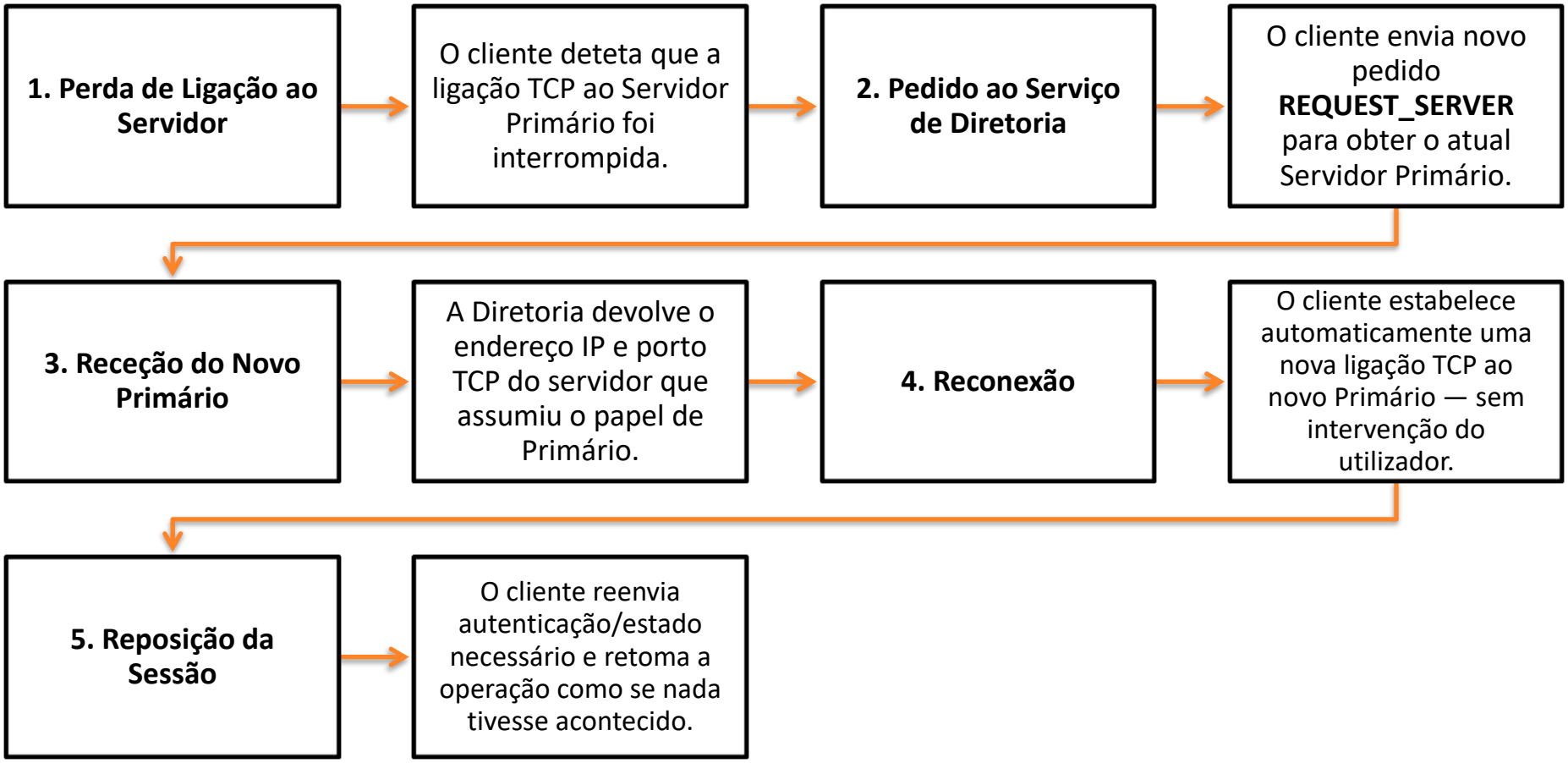
Fluxo de Promoção automática



Fluxo de Sincronização BD



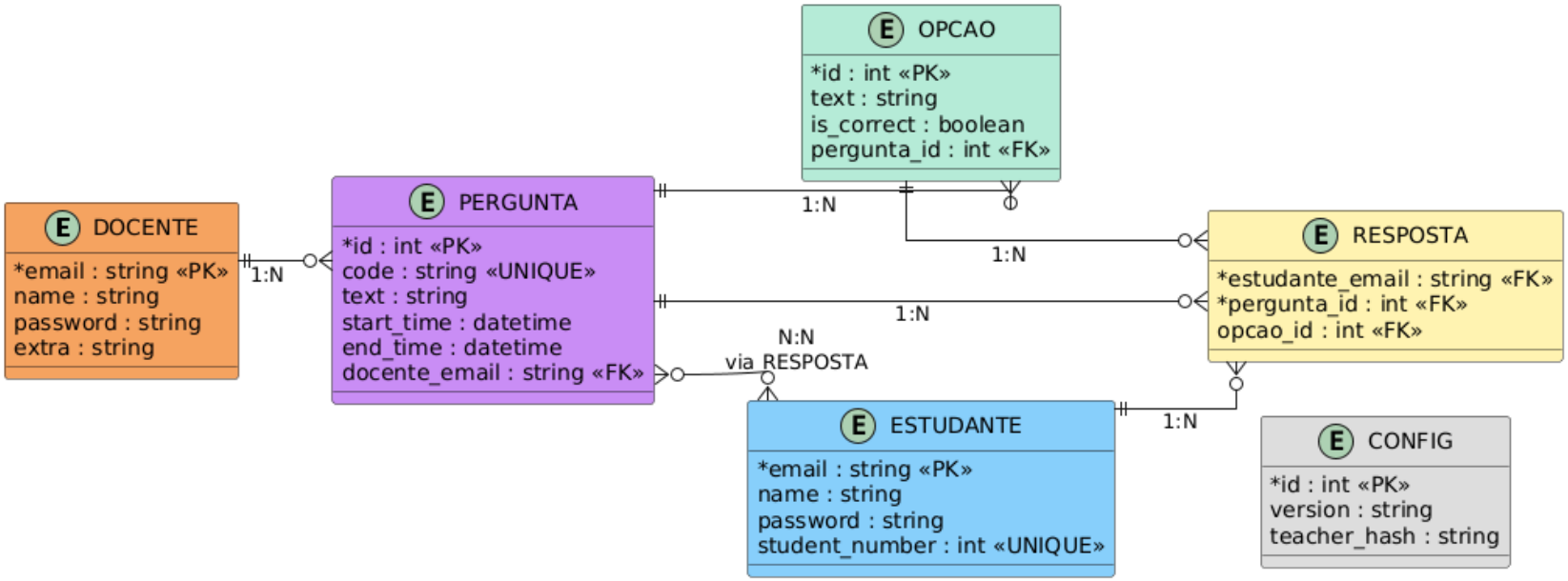
Recuperação Automática do Cliente



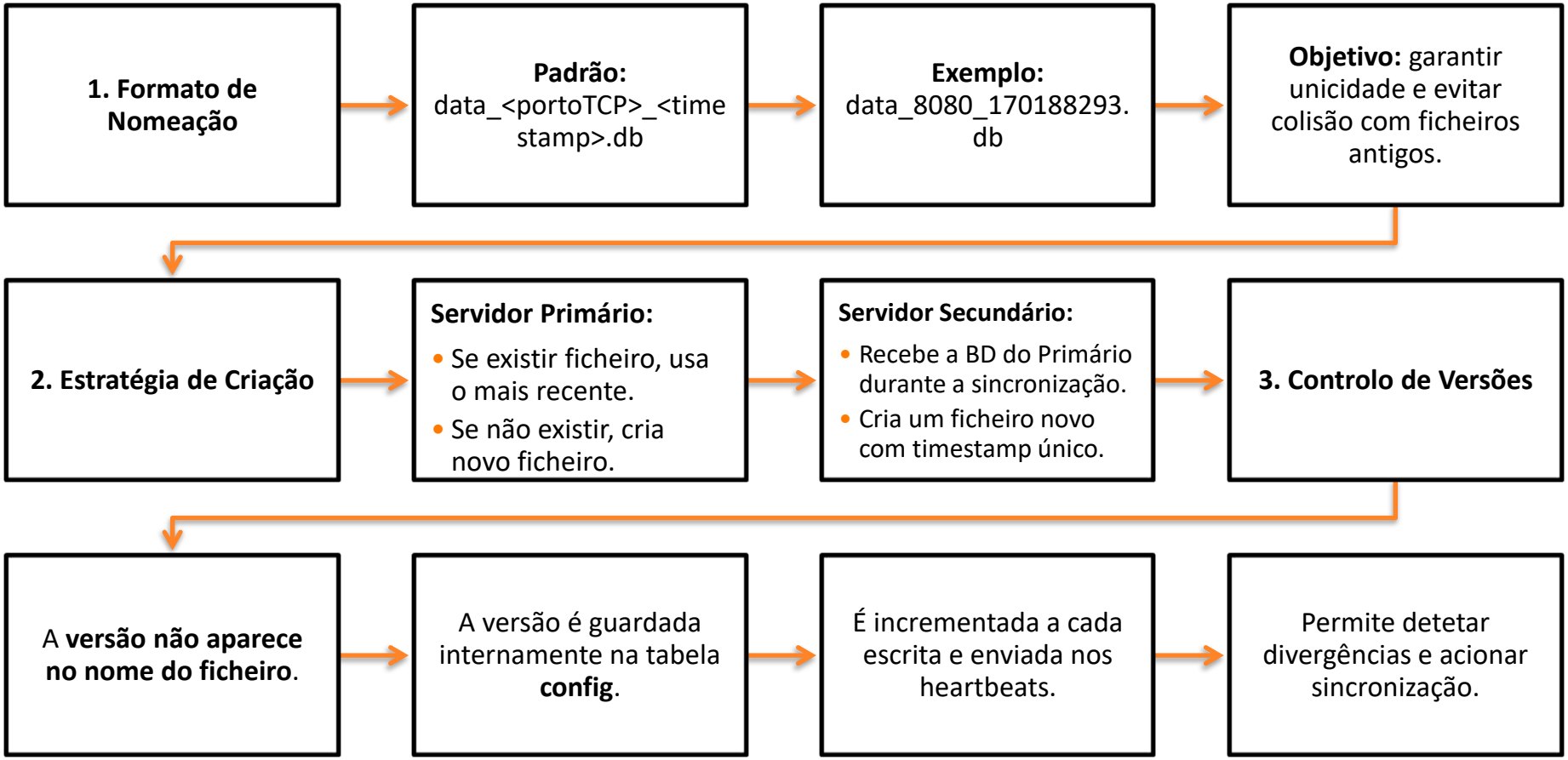
Persistência e Consistência

- **BD SQLite local por servidor** (cada servidor possui a sua cópia)
- **Apenas o Primário escreve** na BD
- **Número de versão da BD** incrementado a cada operação
- **Propagação da query SQL via Multicast** para manter consistência
- **WriteLock** garante exclusão mútua nas operações de escrita
- **Transferência inicial da BD é atómica** (sem concorrência durante o envio)

Modelo Físico da Base de Dados (SQLite)



Nomeação dos Ficheiros SQLite



Protocolos

TCP

Cliente-Servidor
Transferência BD

UDP

Diretoria

MULTICAST

Heartbeats
+
Queries SQL

SERIALIZAÇÃO DE OBJETOS

Aplicação Cliente (JavaFX)

- Arquitetura **MVC** (Views isoladas da lógica de rede).
- **Thread de escuta** a correr em background para receber eventos do servidor.
- **Platform.runLater** para atualizar a UI de forma segura e responsiva.
- Funcionalidades: **Login**, acesso a **perguntas ativas**, **submissão**, **histórico**, etc.

Funcionalidades Docente

Login

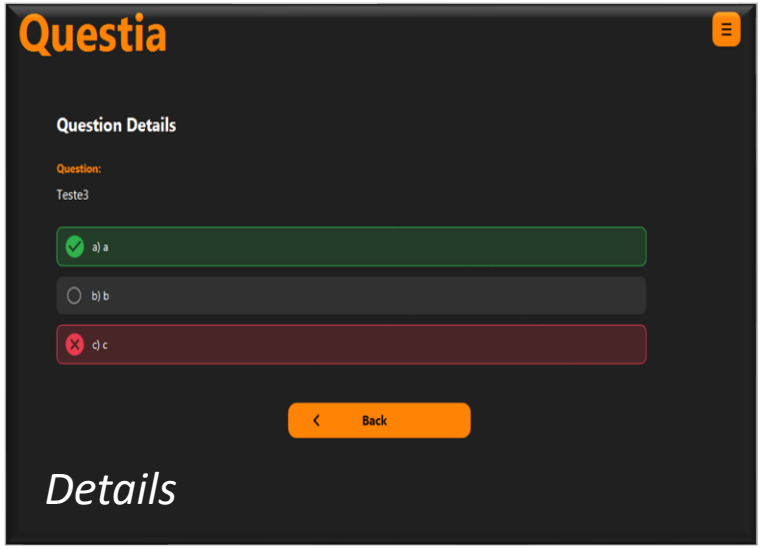
Profile

Home

Funcionalidades Docente

Funcionalidades Estudante

Funcionalidades Estudante



Objetivo da API REST

- Permitir que o sistema possa ser utilizado por aplicações Web, Mobile ou serviços externos, não apenas pelo cliente JavaFX.
- Fornecer uma interface pública e simples para operações essenciais: autenticação, obtenção de perguntas e submissão de respostas.
- Desacoplar a lógica de negócio da interface gráfica, facilitando evolução futura sem alterar o núcleo do servidor.
- Preparar o sistema para integração com novos clientes e para cenários de interoperabilidade (ex.: dashboards de docentes).
- Reutilizar a infraestrutura existente do servidor (QueryPerformer, validação, sincronização) sem duplicar lógica.

Arquitetura da API REST

- A API REST funciona como uma camada externa que expõe funcionalidades do servidor a clientes Web/Mobile.
- Cada pedido HTTP é tratado por um Controller, que valida dados e converte pedidos em chamadas internas.
- A lógica de negócio continua centralizada no servidor, através de:
  - QueryPerformer (acesso à BD)
  - Serviços de autenticação
  - Serviços de gestão de perguntas e respostas
- A API não interfere no mecanismo distribuído existente (TCP, multicast, diretoria).
- Suporta formatos JSON e autenticação via token (planeada para unificar com RMI).

Definição da API REST (Planeamento)

Header obrigatório nos endpoints autenticados:
Authorization: Bearer <token>

Método	Endpoint	Descrição	Parâmetros	Requer Token?
POST	`/api/login`	Autenticação e emissão de token JWT.	—	Não
POST	`/api/register`	Registo de estudante.	—	Não
GET	`/api/questions/{code}`	Obter pergunta ativa identificada por código.	`code` (path)	Sim
POST	`/api/questions/{id}/answer`	Submeter resposta do estudante.	`id` (path)	Sim
GET	`/api/history`	Obter histórico das respostas submetidas.	`?from`, `?to` (opcional)	Sim

Interface RMI (Planeamento)

Objetivo

- Facilitar futuras integrações com aplicações Java (ex.: aplicações desktop externas), sem necessidade de HTTP ou REST.

Funções principais planejadas

- Autenticação (login(email, password))
- Registo de novo estudante
  registerStudent(String email, String name, String password, int studentNumber)
- Obtenção de pergunta ativa
- Submissão de resposta
- Consulta de histórico

DTOs (Data Transfer Objects)

Objetivo

- Simplificar a troca de dados entre cliente, servidor e API, evitando expor a estrutura interna da base de dados.

Exemplos de DTOs (planeados)

- LoginRequest
- RegisterRequest
- QuestionDTO
- AnswerDTO
- HistoryDTO

Vantagens

- Reduz acoplamento entre camadas
- Estruturas claras e específicas para cada operação
- Facilita validação e serialização (JSON)
- Aumenta a segurança ao não expor entidades internas

Decisões, Justificações e Limitações

1. Sincronização da Base de Dados

Desafio: Garantir consistência entre réplicas num sistema distribuído.
Solução: Implementação de um controlo de versões robusto (SN incrementado a cada escrita).

- Se um secundário detetar um salto na versão, reinicia automaticamente e sincroniza novamente para garantir integridade.

2. Concorrência (Leitura vs. Escrita)

Desafio: Permitir que vários estudantes consultem perguntas enquanto o sistema processa escritas.
Solução: Uso de ReentrantReadWriteLock no DatabaseManager, garantindo:

- Paralelismo seguro em leituras
- Exclusividade em escritas

Decisões, Justificações e Limitações

3. Recuperação Transparente do Cliente
Desafio: Evitar falhas no cliente quando o servidor cai.
Solução: Cliente deteta EOF e reconecta automaticamente à Diretoria. Recebe o novo Primário sem interrupção da experiência.

4. Estratégia de Replicação
Decisão: Replicação baseada na propagação de instruções SQL via Multicast.
Justificação: Muito mais leve do que enviar ficheiros completos de BD. Permite que todos os servidores apliquem a mesma sequência de operações.

Manual de Utilizador: Configuração e Arranque

1. Serviço de diretoria

Comando: java pt.isec.pd.directory.Main <Porto_UDP>
Exemplo: java pt.isec.pd.directory.Main 9000

2. Servidores (Primário + Secundários)

Comando: java pt.isec.pd.server.Main <IP_Dir>:<Porto_Dir> <IP_Multicast> <Pasta_BD>
Exemplo: java pt.isec.pd.server.Main 127.0.0.1:9000 230.30.30.30 ./db_storage

O primeiro servidor iniciado torna-se Primário
Os restantes tornam-se Secundários e sincronizam automaticamente via TCP/Multicast

3. Cliente (Interface Gráfica JavaFX)

Comando: java pt.isec.pd.client.Main <IP_Dir> <Porto_Dir>
Exemplo: java pt.isec.pd.client.Main 127.0.0.1 9000

Manual de Utilizador: Perfil Docente

1. Registo
Selecionar Register: Teacher
Introduzir: nome, email, password
Código obrigatório: p4Ssw0!3d (controlo de acesso docente)

2. Criar Pergunta
Inserir enunciado e opções (≥ 2)
Marcar a opção correta
Definir Período de Disponibilidade (início e fim)
Guardar pergunta (fica ativa para os estudantes)

3. Consultar Resultados
Menu History
Ver estado das perguntas (Ativa / Futura / Expirada)
Ver respostas dos alunos
Exportar resultados para CSV/Excel

Manual de Utilizador: Perfil Estudante

1. Acesso
Registar como Student (número único)
Autenticar com email + password

2. Responder Pergunta
Inserir código da pergunta
Se estiver dentro do horário, a pergunta é apresentada
Selecionar a opção
Clicar Submit
Confirmação imediata de submissão

3. Histórico
Menu History
Ver todas as respostas submetidas

Conclusão

Objetivos Atingidos

1. Implementação de um sistema distribuído robusto e tolerante a falhas, capaz de continuar operacional mesmo que servidores secundários falhem.

2. Integração eficiente de protocolos híbridos:

- TCP para pedidos e respostas,
- UDP/Multicast para descoberta, versões e sincronização.

3. Desenvolvimento de uma interface gráfica (JavaFX) reativa, intuitiva e consistente para Docentes e Estudantes.

Conclusão

Principais Aprendizagens

- Gestão avançada de Threads e Concorrência em Java, incluindo coordenação entre múltiplos clientes e operações simultâneas.
- Mecanismos de sincronização e consistência de estado em clusters distribuídos.
- Conceção modular da aplicação, com clara separação entre Lógica de Negócio, Interface Gráfica e Acesso a Dados, facilitando manutenção e evolução.