isec Engenharia

**Programação Distribuida**

# RELATÓRIO DO TRABALHO PRÁTICO

Meta 1


Guilherme de Sousa Camacho - 2021138502

2024

# Índice

# Introdução

Neste trabalho pretende construir-se em Java 3 aplicativos: *Server*, *Backup Server* e *Client*. A comunicação entre Servidor e Cliente é feita através do protocolo TCP e entre *Server* e *Backup Server* é feita através do protocolo UDP

As aplicações *Server* e *Backup Server* são aplicativos de consola, ou seja, só mostra texto e não têm nenhum comando e a aplicação *Client* é algo semelhante a um aplicativo *mobile*, mas em *desktop*, ou seja, um aplicativo com parte gráfica.

No começo deste trabalho, foi decidido que iria ser implementado a parte gráfica no *Client, pois* não havia necessidade de criar uma interface de consola para a fim de teste de comunicação para no fim ser trocada por uma nova interface.

# Arquitetura do Sistema

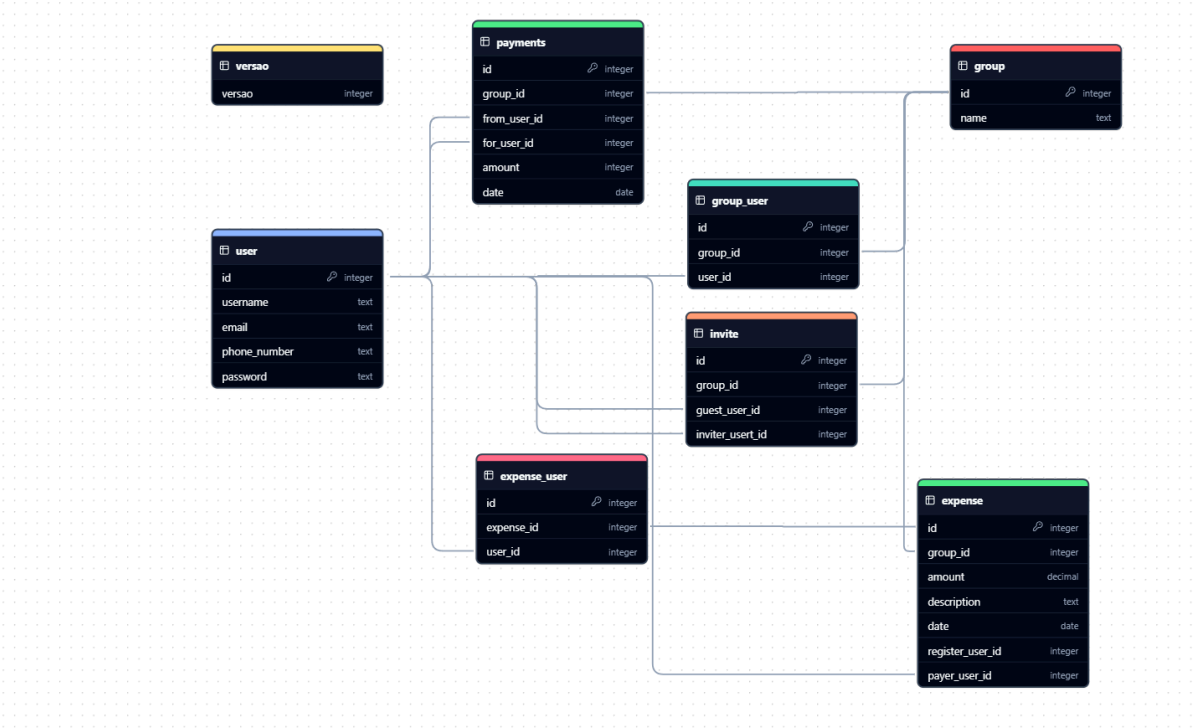
## Visão Geral

A arquitetura implementada segue um modelo distribuído onde o *Client* interage primariamente com o Server via conexões TCP para operações principais, enquanto o *Backup Server* mantém uma cópia redundante dos dados via comunicação UDP com o *Server* principal.
Ambas as comunicações usam serialização de objetos assim tendo uma melhor partilha e segura de dados, ou seja, sem estar a preocupar no *cast* de uma *string*, por exemplo, em outro valor, por exemplo, *double*.
Para um melhor uso do JavaFX e de outras dependências auxiliares, foi utilizador o *gradle* para gerir este projeto. Foi escolhido o *gradle* em vez do *maven, pois* o *gradle* tem a capacidade de compilar um subprojeto dependente de outro subprojeto sem ter a preocupar de compilar primeiro esse mesmo subprojeto.

## Componentes Principais

- SharedLib (biblioeteca onde está as classes comuns entre aplicativos)
- Client App
- Server App
- BackupServer App

## Base de dados




A base de dados foi feito com *SQLite* e o uso de 2 *desgin pattern*: *Data Access Object (DAO)* e *Data Transfer Object (DTO)* onde facilita o acesso ao registo da base de dados usando as operações *CRUD (Create Read Update Delete)* e minimizar o tamanho da informação passa entre *Server* e *Client*.
Para cada tabela existe uma camada DAO, assim sendo mais fácil buscar o registo pretendido. Cada DAO devolve um objeto *Entiy* que representa os dados da tabela num objeto, algo semelhante que uma *ORM (Object-Relational Mapping)* faria.
Todas as *DAOs* e o sistema de conexão com a base de dados está na classe *DataBaseManager* onde nela é estão 3 tipos de operações SQL: ler, escreve e escrever com retorno do índex. Dentro dessa mesma classe este a classe *SyncManager* onde nela é feita as operações de bloqueio no caso de uma *BackupServer* estiver a baixar uma cópia da base de dados atual.

# Comunicação TCP

A implementação desta comunicação foi desenvolvida como uma comunica semelhante a HTTP onde é feito um pedido e é devolvido um *feedback* ou valor pretendido e o uso do *Command Pattern* para ser mais fácil a escalabilidade do projeto pode assim criar e eliminar vários tipos de pedidos com maior facilidade e melhor controlo do que é feito nesse pedido.
Para tal, foi criado a interface *Request* e a classe abstrata *Response*.

# Funcionamento dos programas

## SharedLib

Projeto onde ficam todas as classes partilhadas entres os 3 aplicativos ou pelo menos entre 2 aplicativos.

### Database

Package onde ficam todas as classes relacionadas com a base de dados, sendo a gestão, sincronia, acesso e formatação da informação

#### DatabaseManager

Classe centrar para a gestão e acesso à base de dados, tal como: criação, leitura, escrita e sinalização de alteração da base de dados com o auxílio na sincronia com a classe *SyncManager*

#### Entity

Package onde ficam todas as classes *Entiy* que representam os dados registados da base de dados em objetos.
Estas classes suam o *Build pattern* para omitir informações que pode sem dessecarias ou que não são selecionadas numa *query*.

#### DAO

Package onde ficam todas as classes *DAO* sendo a camada de acesso entre o *Request* e a base de dados.
Como já foi explicada, estas classes têm as operações *CRUD* assim tendo maior flexibilidade de selecionar os dados pretendidos em vez do uso de uma *query* específica pretendida para o *Request*.

#### DTO

Package onde ficam todas as classes *DTO* que vai ser manda para o *Client*. Em geral, existe 2 tipos de *DTO*: *Preview*, onde mostra algumas informações basica para pré-visualização desse elemento e *Detail*, onde mostrar toda, ou quase toda no caso do *User*, informação desse objeto.

#### SyncManager

### Network

Package onde fica todos os *Requests* e *Respose* em que a comunicação entre *Server* e *Client* e o a class *Heartbeat* para a comunicação entre *Server* e *BackupServer*.

### Terminal

Package onde simplesmente tem uma classe com um método estático para imprimir o progresso do *download*/*upload* do ficheiro da base de dados entre *Server* e *BackupServer*.

## Cliente

### UI

Package onde fica todas as classes relacionadas com a parte gráfica da aplicação.

#### ViewManager

Classe gestora das vistas e da comunicação/acesso dos controladores com a classe gestora do modelo e da classe de comunicação via *socket TCP*.
Nesta classes existe um método próprio para fazer um *request* e fica à espera da *response* numa nova thread com essa tarefa de devolver o *response* com uma indicação de *loading* e um método para mostrar em caso da existência de um erro no meio do programa inicialmente pensado em ser visualizado numa janela de diálogo, mas ficando só no terminal por questão de prioridade.

#### NotificationManager

Classe centra para mostrar as notificações que o *socket TCP* recebe.

#### Controller

Package onde ficam todos os controlados das vistas.

#### Component

Package onde ficam componentes personalizados usados neste trabalho, tal como o *Card* e *Dialog*.

### ModelManager

Classes onde guarda as informações essenciais onde os *requests* vão buscar e entre outras.

### SocketManager

Classe onde gere a comunicação via *socket TCP*, e prepara para sinalizar quando uma notificação aparece.

# Servidor

## HeartbeatManager

Classes que cria o *multicast* para ser enviado os *heartbeats* e sinalização de envio de um *heartbeat* com uma *query* com a nova informação que os aplicativos *BackupServer* que deve efetuar para atualiza as suas base de dados.

## Heartbeat Sender

Thread onde fica a mandar os *heatbeats* a cada 10 segundos, buscando a versão da base de dados à classe *DataBaseManager*.

## BackupServer Receiver

Thread onde fica à espera da ligação *TCP* que o *BackupServer* faz como pedido para baixar o ficheiro da base de dados atual.

## BackupServer Handler

Thread onde manda todos os pedaços do ficheiro da base de dados que vai mandar e garantir que é finalizado esse mesmo socket *TCP* do *BackupServer*.

## SessionManager

Classes onde regista as threads que ficam a comunicar os *Clients* atualmente ligado para assim poder mandar as notificações necessárias.

## Client Receiver

Thread onde fica à espera da ligação *TCP* que os *Clients* fazem no arranque da aplicação.

## Client Handler

Thread dedicada para a comunicação entre *Client* e *Server*. Esta classe é a classe onde fica a parte do recetor de comando do *Command Pattern*, assim ele recebe o *request* e executa ela passando a referência da *DataBaseManager*.

# Servidor Backup

Programa com a sequência de ficar à espera do primeiro *heartbeat* para puder recber o ficheiro base de dados para fazer a sua cópia e com o seu *runtime* baseando no protocolo *UDP*.

# Conclusão

Na entrega deste trabalho foi fornecido um *wrapper* do *gradle* para poder executar os comandos *gradle* numa máquina em que não esteja instalada o *gradle*.
Foi tomada essa decisão, pois o tamanho total do projeto seria maior que a capacidade possível que o InforEstudante consegue aguentar (capacidade máxima: 100MB e o projeto com executáveis ou ficheiros *bytecodes* tem >200MB e o ficheiro comprimido não era o suficiente para diminuir mais de metade)
Antes da entrega, foi testa este mesmo *wrapper* numa máquina nova com Linux só com Java 21 JDK e foi executada como deve ser, sendo esperado que execute como esperado.
Todo o funcionamento do *gradle wrapper* está presente na documentação do gradle.
Há cerca da parte *GUI* do aplicativo *Client*, foi feito com a capacidade possível para este trabalho com os obstáculos de não poder fazer algo a mais para ficar semelhante ao aplicativo original onde foi a basear o layout todo.
Neste trabalho também existe a opção de *debug* em *runtime* de modo a teste ativada com uma flag presente nos ficheiros *batch*.
Em termos de trabalho todo, o *Client* é aquele que ficou mais prejudicial por ser mais difícil o sem desenvolvimento sozinho e a sua curvatura de complexidade existente, mas o *Server* está desenvolvido, em teoria, para poder efetuar todos os tipos de operações e o *BackupServer* está completo, mas podendo ter problemas em alguns casos especial não esperado na altura.
Tendo em conta que o trabalho é no máximo de 3 elementos, não me arrependo do estado atual dele, pois está pelo menos 80% completo baseado pelos requisitos e os extras do enunciado.