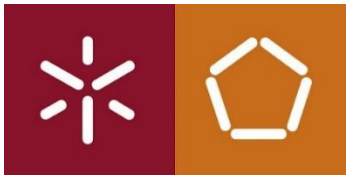

Universidade do Minho
Escola de Engenharia
Licenciatura em Engenharia Informática

SISTEMAS DISTRIBUÍDOS

2021/2022

Sistema de Reserva de Voos - SRV

GRUPO 25:

David Duarte A93253
Joana Alves A93290
Maria Cunha A93264
Vicente Moreira A93296

Data de Entrega: 14/01/2022

Índice

Índice de Figuras

Introdução

Este projeto tem como objetivo aprofundar os nossos conhecimentos obtidos através das aulas teóricas e práticas acerca do controlo de concorrência e conexões *TCP*. Para isso foi-nos proposto desenvolver uma plataforma de reserva de voos sob a forma de um par cliente-servidor. Para o funcionamento correto desta plataforma, a componente do cliente terá de ser capaz de conectar-se ao servidor e efetuar uma série de comandos pré-definidos. Já o servidor terá de ser capaz de não só receber estes comandos e respondê-los adequadamente assim como gerir uma base de dados de forma a evitar potenciais colisões de concorrência.

Arquitetura da Aplicação

Para esta plataforma, começamos por dividir os vários aspetos funcionais necessários em três categorias: estrutura e manipulação da Base de Dados, Protocolo de Comunicação e Transmissão e, por último, a Lógica das t*hreads* Servidor e Cliente.

Apresentamos aqui um esquema geral do funcionamento da plataforma SRV, neste exemplo apresentamos um possível cenário onde três clientes estão ligados ao Servidor e cada um a efetuar as suas operações independentemente.

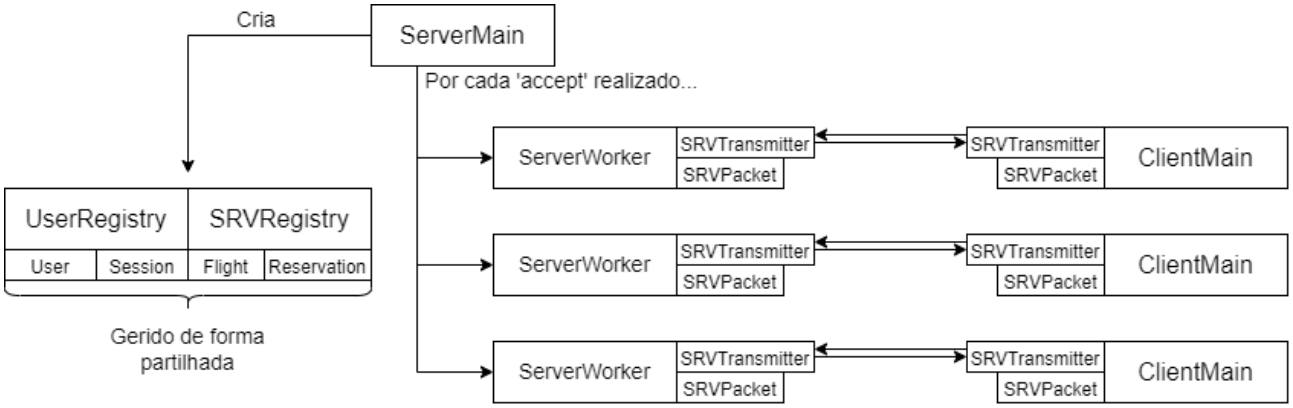

Figura 1 - Arquitetura da plataforma SRV

De seguida, apresentamos um diagrama temporal onde representamos a comunicação inicial entre Cliente e o Servidor (Autenticação):

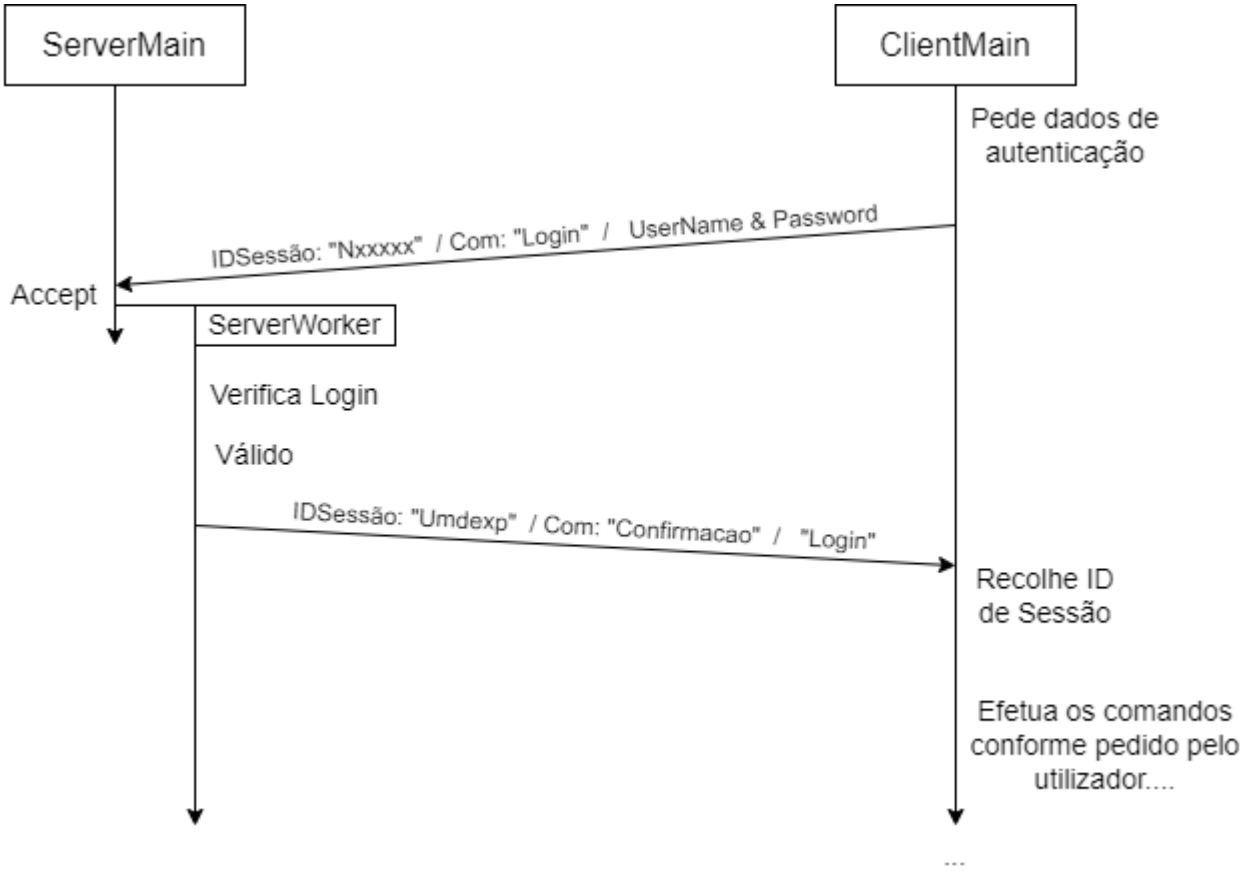

Figura 2 - Diagrama Temporal de Autenticação Cliente-Servidor

Especificação do Protocolo e Transmissão de Pacotes

Para que as componentes de servidor e cliente da plataforma pudessem comunicar entre si de forma rápida e eficaz, foi necessário desenvolver um protocolo de pacotes de forma a estabelecer a semântica e sintaxe das mensagens transmitidas. Também foi necessário criar uma um sistema fiável para enviar e receber estes pacotes através das *sockets*.

SRVPacket

Para os pacotes, começamos por definir que este seria composto por uma série de Strings, todas separadas pelo caracter "\n", pois facilitaria a serialização e desserialização dos pacotes. De seguida, definimos um cabeçalho inicial fixo. Este estaria presente em todos os pacotes transmitidos e contém duas informações, o ID da sessão do cliente que enviou o pacote (no caso do servidor, a que ID este está a responder), e o comando a ser executado/interpretado, quer pelo servidor ou pelo cliente. Por fim, definimos a continuação do pacote como uma lista de argumentos desse pacote, onde, dependendo do comando associado ao pacote, estes argumentos irão transmitir a informação necessária para a execução do comando. Apresentamos aqui todos os comandos definidos no protocolo final:

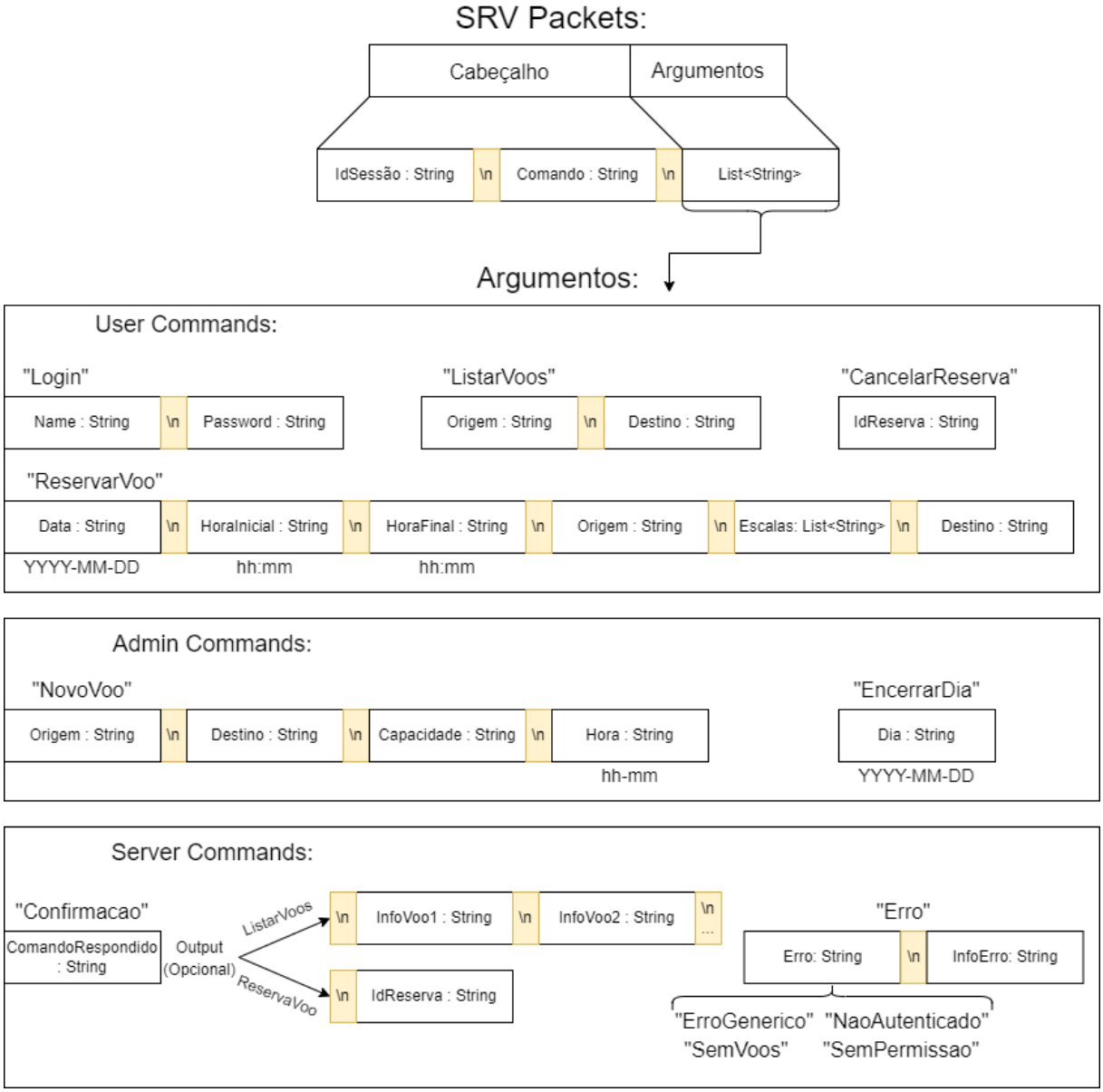

Figura 3 - Sintaxe dos pacotes SRVPackets

SRVTransmitter

Para o transmissor destes pacotes, desenvolvemos a classe "SRVTransmitter". Esta classe será responsável pela serialização do pacote e o seu envio de forma correta, assim como a receção de dados do exterior, convertendo-os para pacotes (receção de pacotes). Para a transmissão dos pacotes, esta recorre a um processo simples de transmissão.

Depois de obtido o pacote serializado, é avaliado o tamanho dos dados gerados e criado um cabeçalho contendo uma variável do tipo *short*, onde a informação do tamanho dos dados será enviada, seguido dos mesmos. Para a leitura de pacotes, este começa por ler uma variável do tipo *short*, recolhendo assim o tamanho do pacote a ser recebido, efetua a leitura total e desserializa a informação lida, obtendo um pacote.

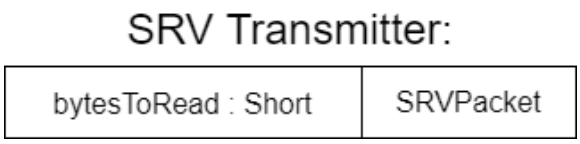

Figura 4 - Sintaxe de transmissão do SRVTransmitter

Estrutura da Base de Dados SRV

UserRegistry

Esta classe é responsável por armazenar a informação de todos os utilizadores no sistema, assim como efetuar os *logins*, *logouts* e gerir as sessões abertas e expiradas dos vários utilizadores. Para isso recorre a duas estruturas principais, uma para armazenar os utilizadores e outra para armazenar as sessões.

Quando esta classe é iniciada, esta lê de um ficheiro pré-definido "Users.csv" e grava todos os utilizadores do sistema. De seguida, quando um utilizador efetuar um *login*, esta autêntica os dados fornecidos e, caso sejam válidos, gera uma nova sessão para esse utilizador, definindo um ID aleatório e uma duração de sessão de 5 minutos. Sempre que este utilizador efetuar ações no servidor, este terá de apresentar o seu ID de sessão e a sua validade será avaliada. Caso a sessão seja válida, a duração desta é atualizada novamente, ou seja, adiciona uma duração de 5 minutos.

SRVRegistry

Esta classe é de elevada importância pois gere todos os aspetos relativos à informação da plataforma de SRV, como os voos, as reservas, as ocupações dos vários voos em cada dia e os dias de reservas encerrados pelos administradores.

No inicializar desta classe, esta lê de um ficheiro pré-definido "Flights.csv" e grava todos os voos do sistema. Depois de iniciada, são disponibilizadas todas as operações que podem ser realizadas através dos comandos dos clientes ou administradores, sendo a responsabilidade do Servidor (neste caso, a *thread* ServerWorker) recolher os argumentos do comando e efetuar a operação sobre esta. A classe disponibiliza as operações de listagem de voos, realização e cancelamento de reservas, adição de voos e encerramento de dias.

Controlo de Concorrência

Dado que a qualquer momento de execução do Servidor poderão existir múltiplas *threads* "ServerWorker", e, visto que estas partilham as mesmas estruturas de dados, podendo executar várias operações de modificação, adição ou remoção de dados simultaneamente, é essencial garantir que as leituras e escritas nestas estruturas ocorram de forma segura e controlada, de forma a evitar não só potenciais corrupções dos dados como problemas "lógicos" na execução do Servidor (por exemplo, efetuar uma reserva num certo dia apesar do pedido para o encerramento deste já ter sido efetuado).

Para evitar estas colisões, e aplicando o conhecimento aprendido nas aulas práticas, recorremos à utilização de *Locks* de concorrência, através da classe "ReentrantReadWriteLock". Optamos por utilizar os *ReadWrite locks* visto que algum dos serviços disponibilizados pelas estruturas de dados apenas recorriam à leitura destes, não justificando o "trancamento" do acesso de toda a estrutura para essa operação.

Aplicamos estes *Locks* nas duas principais estruturas de dados do servidor: "UserRegistry" e "SRVRegistry". Para a classe "UserRegistry" utilizamos apenas um *lock* global onde trancamos as escritas (writeLock) quando é efetuado um *login* ou *logout* e trancamos a leitura (*readLock*) na verificação de utilizadores já autenticados.

Para a classe "SRVRegistry", devido às várias estruturas de dados que esta contém, assim como as várias ações complexas que esta pode efetuar, decidimos utilizar dois *locks* independentes, sendo cada um responsável por "gerir" as suas estruturas. Definimos então o *lock* "flightsLock", responsável pelo controlo de concorrência na manipulação da estrutura que contém todos os voos do sistema (mapa 'flights'), e o *lock* "resOccCLoLock" que controla o acesso às estruturas de dados das reservas, ocupações e os dias fechados (mapas 'reservations', 'occupations' e 'closedDays'). Recorremos a esta estratégia, visto que tanto as reservas, como as ocupações dos voos e os dias encerrados estão ligados entre si, ou seja, para efetuar modificações numa das estruturas será necessário recorrer à leitura/escrita noutra. Por exemplo, efetuar uma reserva não só recorre à adição de uma reserva na lista, como requere a verificação se o dia desta não está encerrado e verificação das ocupações dos voos.

Funcionalidades Adicionais

Para além dos requisitos base requeridos, a plataforma SRV oferece funcionalidades extra como:

- Sistema de Sessões para a autenticação de Utilizadores.
- Funcionalidade 'ListarReservas': Listar o ID de todas as Reservas efetuadas pelo Utilizador.
- Interface multicor para fácil interpretação.
- Servidor mantém um log externo de todas as operações efetuadas.

Conclusão

Acreditamos que alcançamos de forma satisfatória todas as metas propostas pelos docentes, obtendo assim uma plataforma funcional e eficaz para a gestão de reservas de voos, respondendo às necessidades de controlo de concorrência para um sistema seguro.

Com este projeto, pudemos aplicar os conceitos de concorrência lecionados nas aulas e aprendemos a utilidade e importância que estes têm na gestão de sistemas.