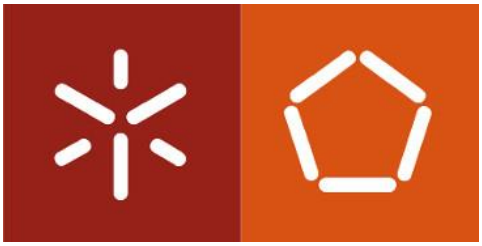
UNIVERSIDADE DO MINHO



Trabalho Prático I

Mestrado Integrado em Engenharia Biomédica

Aplicações Distribuídas

(1º Semestre/ Ano Letivo 2021/2022)

Realizado por:

Lara Alexandra Pereira Novo Martins Vaz (A88362)

Mariana Lindo Carvalho (A88360)

Tiago Miguel Parente Novais (A88397)

Braga

11 de dezembro de 2021

Resumo

O presente trabalho é baseado na implementação de uma aplicação baseada na Gestão do Processo Clínico e na Gestão Logística e de Medicamentos de forma integrada, suportado pela linguagem de programação JAVA. No âmbito da unidade curricular de Aplicações Distribuídas, o principal objetivo é transformar uma aplicação local numa aplicação distribuída, através da utilização de estruturas de dados e da arquitetura Cliente-Servidor utilizando RMI e controlo de concorrência.

No decorrer deste trabalho foi possível adquirir uma variada gama de conhecimentos na linguagem de programação utilizada, nomeadamente a definição de entidades e a criação de relações entre elas, a utilização de *HashMaps*, entre outras.

Índice

1. Introdução

O presente trabalho é baseado na implementação de uma aplicação baseada na Gestão do Processo Clínico e na Gestão Logística e de Medicamentos de forma integrada, suportado pela linguagem de programação JAVA. No âmbito da unidade curricular de Aplicações Distribuídas, o principal objetivo é transformar uma aplicação local numa aplicação distribuída, através da utilização de estruturas de dados e da arquitetura Cliente-Servidor utilizando RMI e controlo de concorrência.

Sendo este um tema de interesse na área da Engenharia Biomédica, foi solicitada a implementação de dois sistemas, um responsável pela gestão dos atos médicos, farmacêuticos e de enfermagem e outro que engloba toda a informação acerca do stock de medicamentos e outros artigos diversos.

Primeiramente, procedeu-se ao levantamento das condições necessárias para o desenvolvimento das aplicações envolvidas no processo de gestão. Assim, a etapa inicial centrou-se na definição das entidades envolvidas no projeto, bem como das interfaces dos diferentes serviços idealizados, considerando a lógica da gestão de negócio necessária para um correto funcionamento das aplicações.

Para a construção deste caso prático, é necessário cumprir os seguintes requisitos:

- Levantamento dos requisitos e definição das funcionalidades a desenvolver;
- Definição das entidades envolvidas;
- Definição das interfaces dos diferentes serviços que idealizar;
- Implementação da lógica de negócio necessárias ao correto funcionamento do sistema;
- Implementação da(s) aplicações cliente e servidor.

2. Preliminares

2.1. Estrutura de Dados

A linguagem JAVA utiliza uma estrutura de dados de forma a permitir o tratamento de grandes quantidades de informação. Para além disso, possibilita a organização dos dados de forma lógica e ainda a sua manipulação através da disponibilização de determinados métodos ao utilizador. No presente trabalho foram utilizadas estruturas de dados do tipo *ArrayList* e *HashMap*.

2.2. JAVA RMI

O Java RMI facilita o desenvolvimento de aplicações distribuídas baseadas em objetos. Assim, o seu principal objetivo é permitir a invocação de métodos de objetos remotos e, desta forma, permite simplificar a comunicação entre dois objetos em máquinas virtuais distintas. Assim, torna-se possível a troca de classes e objetos com diferentes complexidades entre clientes e servidores [1].

Deste modo, a arquitetura RMI possibilita que uma determinada aplicação cliente adquira uma interface (que define o comportamento) referente a uma classe (que contém o processo de implementação) que pode existir numa máquina virtual distinta. O JAVA RMI utiliza interfaces, que correspondem a um conjunto de especificações sintáticas de métodos que representam um comportamento particular, que qualquer classe, em qualquer ponto da hierarquia pode implementar [2].

Assim, para o desenvolvimento de uma aplicação através da utilização de RMI é necessário definir as interfaces remotas e as respetivas implementações. Posteriormente é necessário proceder à criação do servidor RMI que cria o objeto que implementa o serviço e o inclui nos objetos distribuídos [2].

3. Desenvolvimento do Sistema de Gestão

O sistema elaborado é constituído por onze entidades: **Funcionario**[1]**, Utente, Medico, Enfermeiro, Farmaceutico, Fornecedor, Medicamento, OutrosArtigos, AtoMedico, AtoFarmaceutico** e **AtoEnfermagem**.

O Sistema de Gestão do Processo Clínico (**GestorGCP**) tem como utilizadores os **Medicos**[2], os **Enfermeiros** e os **Utentes** e contém informações relativas aos **AtosMedicos** e **AtosEnfermagem** e aos respetivos participantes desses atos: **Utentes, Medicos, Enfermeiros**. Neste sistema estão registados todos os atos médicos, de enfermagem e de dispensa de medicamentos efetuados a um utente. Cada ato regista a identificação do utente, a identificação do agente (médico ou enfermeiro), uma data/hora e uma descrição. A descrição dos atos médicos consiste numa enumeração dos vários medicamentos (como analgésicos e antibióticos, entre outros) e artigos (como bisturis e estetoscópios, entre outros) utilizados no ato médico. Já aos atos de enfermagem, apenas estão associados artigos diversos, uma vez, que se considera que um enfermeiro não pode fazer a dispensa de medicamentos a um utente.

Relativamente às funcionalidades a que os utilizadores têm acesso no GPC, os funcionários **Medico** e **Enfermeiro** podem adicionar novos utentes, médicos e enfermeiros, e consequentemente criar os seus atos médicos e de enfermagem associados a estes. Além dessas operações, o grupo de **Funcionarios**, do qual fazem parte, podem ainda utilizar métodos que permitem procurar, alterar atributos ou remover utentes, médicos, enfermeiros e atos e métodos para obter estatísticas relevantes. Por sua vez, os **Utentes** apenas podem aceder aos seus próprios dados e visualizar os atos que lhes foram prestados.

Este sistema de gestão é implementado de forma que:

- a um **AtoMedico/AtoEnfermagem** corresponde um e um só utente;
- a um médico e a um utente pode corresponder mais do que um **AtoMedico/AtoEnfermagem**, isto é, o mesmo médico ou enfermeiro pode efetuar mais do que um ato.

[1] Os nomes das classes foram destacados a negrito, de modo a facilitar a leitura e compreensão deste documento.

[2] Apesar de não existir nenhuma classe Medicos, como a palavra no plural visa representar vários objetos do tipo Medico, então, nomes de classes no plural também estão assinalados a negrito.

Quanto ao Sistema de Gestão Logística e de Medicamentos (**GestorGLM**), os seus utilizadores são os **Farmaceuticos**, sendo que este sistema contém informações relativas aos participantes (**Farmaceuticos**) e também informações relativas aos **Medicamentos**, **OutrosArtigos** e respetivos **Fornecedores.** Neste sistema existem também informações sobre stocks de medicamentos e outros artigos diversos, sendo permitido registar aquisições, efetuar consultas sobre a quantidade armazenada e efetuar o registo de consumos. Todos estes atos são atos da responsabilidade de um farmacêutico, pelo que, como é óbvio, um farmacêutico tem acesso aos stocks quer dos medicamentos, quer de artigos diversos.

No GLM, os **Farmaceuticos** podem criar novos farmacêuticos e fornecedores, e consequentemente, criar atos farmacêuticos associados a estes. Além dessas operações, os **Farmaceuticos** podem ainda utilizar métodos que permitem procurar, alterar atributos ou remover farmacêuticos, fornecedores, medicamentos e outros artigos, métodos para obter estatísticas relevantes e métodos que permitem avaliar se determinados produtos estão com baixo stock.

De forma a garantir a confidencialidade dos dados geridos por estes sistemas de gestão, foi criada uma classe **Operacao**.

Para o presente trabalho, a arquitetura escolhida foi a Cliente/Servidor, em que o **GestorGPC,** o **GestorGLM** e o **Operacoes** são todos inicializados pelo servidor (**AplicacaoServidor**) e os clientes são os utilizadores das várias interfaces (**Utentes, Medicos, Enfermeiros e Farmaceuticos**).

3.1. Entidades

3.1.1. Classe Funcionario

A superclasse Funcionário tem como atributos nome, número de BI, NIF, morada e código-postal. Esta classe abstrata representa o conceito geral das entidades **Medico, Enfermeiro** e **Farmaceutico,** bem como as suas subclasses. É de realçar que se admitiu que o identificador único de um **Funcionario** é o seu NIF, pois, cada pessoa tem um NIF diferente.

A classe abstrata **Funcionario** apresenta também um construtor vazio e um construtor que recebe os dados como parâmetro e, assim preenche os atributos do objeto.

Possui também vários métodos *public* que permitem obter o valor de uma variável (*getters*) e outros que configuram o valor de uma variável (*setters*).

Todas as classes discutidas neste trabalho são *serializable*, uma vez que se pretende garantir uma correspondência biunívoca entre as referências de objeto e os seus identificadores. Assim, todas as classes apresentam também um *serialVersionUID*. Para além disso, as classes que representam entidades apresentam, também, um método *toString()*, que permite obter uma representação *String* de um objeto.

3.1.2. Classe Medico

A classe **Medico** possui como atributos a especialidade e a cédula do medico, assim como os atributos da superclasse **Funcionario**. Esta classe consiste basicamente num construtor que apenas invoca o construtor da superclasse utilizando a palavra *super* e um outro construtor faz o mesmo e recebe também os atributos especialidade e cédula. Possui também *getters* e *setters* dos seus atributos, assim como o *toString()*.

3.1.3. Classe Enfermeiro

A classe **Enfermeiro** possui como atributos a especialidade, assim como os atributos da superclasse **Funcionario**. Esta classe consiste basicamente num construtor que apenas invoca o construtor da superclasse utilizando a palavra *super* e um outro construtor faz o mesmo e recebe também o atributo especialidade. Possui também *getters* e *setters* dos seus atributos, assim como o *toString()*.

3.1.4. Classe Farmaceutico

A classe **Farmaceutico** possui como atributos os da superclasse **Funcionario**. Esta classe consiste basicamente num construtor que apenas invoca o construtor da superclasse utilizando a palavra *super* e possui também *getters* e *setters* dos seus atributos, assim como o *toString()*.

3.1.5. Classe Utente

A classe **Utente** é uma classe cujos atributos são nome, número de BI, NIF, morada e código-postal. Esta classe apresenta também um construtor vazio e um construtor que recebe os dados como parâmetro e, assim preenche os atributos do objeto. Possui também vários *getters* e *setters* dos seus atributos, assim como o *toString()*.

3.1.6. Classe Fornecedor

A classe **Fornecedor** é uma classe cujos atributos são o ID e o nome. Esta classe apresenta também um construtor que recebe os dados como parâmetro e, assim preenche os atributos do objeto. Possui também vários *getters* e *setters* dos seus atributos, assim como o *toString().*

3.1.7. Classe Medicamento

A classe **Medicamento** possui como atributos o ID, a denominação comum internacional (dci), o nome, a forma farmacêutica, a dosagem, o estado autorizado, o genérico, o **Fornecedor** e a quantidade em stock que existe do medicamento. Tem, também, um construtor, os *getters* e setters dos seus atributos, assim como o *toString().*

3.1.8. Classe OutrosArtigos

A classe **OutrosArtigos** possui como atributos o ID, o nome, o **Fornecedor** e a quantidade em stock que existe desse artigo. Tem, também, um construtor, os *getters* e *setters* dos seus atributos, assim como o *toString().*

3.1.9. Classe AtoMedico

A classe **AtoMedico** possui como atributos o ID, o **Medico** que prestou o ato e o **Utente** a quem o ato foi prestado, a hora, um *HashMap* com chaves que correspondem aos IDs dos artigos usados no ato e cujos valores são os respetivos atos e ainda um *HashMap* semelhante, mas para os medicamentos. Tem, também, um construtor, os *getters* e *setters* dos seus atributos, assim como o *toString().*

3.1.10. Classe AtoEnfermagem

A classe **AtoEnfermagem** possui como atributos o ID, o **Enfermagem** que prestou o ato e o **Utente** a quem o ato foi prestado, a hora, um *HashMap* com chaves que correspondem aos IDs dos artigos usados no ato e cujos valores são os respetivos atos. Tem, também, um construtor, os *getters* e *setters* dos seus atributos, assim como o *toString().*

3.1.11. Classe AtoFarmaceutico

A classe **AtoFarmaceutico** possui como atributos o ID, o **Farmaceutico** que prestou o ato de reposição de medicamentos/artigos, a hora, um *HashMap* com chaves que correspondem aos IDs dos artigos usados no ato e cujos valores são os respetivos atos. Tem, também, um construtor, os *getters* e *setters* dos seus atributos, assim como o *toString().*

3.2. Classe GestorGPC

A classe **GestorGPC** define todos os métodos que permitem a construção e manipulação das entidades envolvidas na Gestão do Processo Clínico (**Utente, Medico, Enfermeiro, AtoMedico, AtoEnfermagem**).

Esta classe tem oito atributos: utentes, medicos, enfermeiros, atosmedicos, atosenfermagem, atosfarmaceuticos, operacoes, loginGPC e loginUtentes. Os primeiros três atributos são *HashMaps*, cuja chave é o NIF da pessoa e o valor é o respetivo objeto que representa essa mesma pessoa. Por exemplo, no *HashMap* utentes, as chaves são o NIF do utente e os valores são os objetos da classe **Utente**. Desta forma, medicos tem como chave igualmente o NIF e como valor o médico correspondente. Da mesma forma se encontram organizados os enfermeiros. Quanto aos atosmedicos e aos atosenfermagem, também estes são *HashMaps*, cuja chave representa o ID do respetivo ato e os valores correspondem ao objeto da respetiva classe. Por exemplo, no *HashMap* atosmedicos, a chave é o ID do ato médico e o valor é o objeto **AtoMedico**. Relativamente aos atributos loginGPC e loginUtentes, são igualmente *HashMaps* onde a chave corresponde ao NIF do respetivo funcionário (medico ou enfermeiro) no caso do loginGPC, e ao NIF do utente, no caso do loginUtentes. Em ambos, o valor contém a password dos respetivos utilizadores. Por último, o atributo operacoes é o objeto da classe **Operacoes**, utilizado como intermediário entre a classe **GestorGPC** e **GestorGLM**, ou seja, contém toda a informação e métodos comuns a ambos os sistemas.

Nas secções seguintes são abordados alguns dos métodos desenvolvidos na classe **GestorGPC,** fundamentais para concretizar as funcionalidades pretendidas. De salientar que todos os métodos da classe **GestorGPC** que estejam relacionados com quaisquer alterações na base de dados são *synchronized*. Assim, quando vários clientes estiverem a utilizar estes métodos, a base de dados não é alterada de forma sobreposta e não há a perda de dados ou outros problemas associados.

3.2.1. Carregamento dos ficheiros para a base de dados

Os métodos load_meds, load_medicos, load_utentes, load_enfermeiros, load_passUtentes, load_passGPC e load_passGLM permitem carregar os dados dos ficheiros para a base de dados. Para além disso, foram desenvolvidos métodos para gravar, atualizar e carregar a *database1*.

3.2.2. Criar Utente, Médico, Enfermeiro e Farmacêutico

O método criaUtente permite adicionar um novo utente, e, por isso, tem como argumentos todos os atributos da classe **Utente**. Se o NIF do utente (chave do *Hashmap* utentes) já existir, então é lançada a exceção UtenteJaExiste. Se o mesmo NIF, na qualidade de chave do *HashMap* loginUtentes, não existir, então é lançada a exceção LoginUtenteNaoExiste. Caso contrário, é feita a autenticação do utilizador, confirmando a respetiva *pass*. Da mesma forma, se encontram estruturados os restantes métodos criaEnfermeiro, criaMedico e criaFarmaceutico.

É de realçar que todos os métodos existentes no GPC relativos aos utentes (exceto relativos a criar ou remover algo), existem duplicados. Isto porque, quer os utentes quer os médicos e enfermeiros podem consultar informações relativas aos utentes. Contudo, é necessário que existam métodos duplicados, pois, como os utentes, médicos e enfermeiros que têm acesso estão em dois *HashMaps* diferentes (loginUtentes e loginGPC), então, a validação do NIF do utilizador e respetiva *password* será diferente para utentes ou médicos e enfermeiros. Poderia se pensar que para evitar a duplicação, os utentes, médicos e enfermeiros que têm acesso aos dados, deveriam estar todos num mesmo login, contudo, isso permitiria que os utentes tivessem mais permissões do que aquelas que podem ter.

3.2.3. Obter informação do Utente, Médio, Enfermeiro e Farmacêutico

O método getUtenteInfo permite obter informação sobre um utente conhecendo o seu NIF. Se o NIF do utente não existir, então é lançada a exceção UtenteNaoExiste. Da mesma forma, se encontram estruturados os restantes métodos getEnfermeiroInfo, getMedicoInfo e getFarmaceuticoInfo.

3.2.4. Alterar informação do Utente, Médico, Enfermeiro e Farmacêutico

Com o objetivo de permitir a alteração da informação registada acerca de um utente, foram criados os métodos AlterarNomeUtente, AlterarBiUtente, AlterarNIFUtente, AlterarMoradaUtente, AlterarCodigoUtente que recebem como argumento o NIF do utente e o respetivo atributo que se pretende alterar. Da mesma forma estão estruturados os restantes métodos relacionados com a informação do médico, do enfermeiro e do farmacêutico, sendo que ao médico acrescenta o método AlterarEspecialidadeMedico e AlterarCedulaMedico e ao enfermeiro o método AlterarEspecialidadeEnfermeiro.

3.2.5. Procurar Médico, Enfermeiro, Utente e Farmacêutico

Com o objetivo de permitir o acesso a toda informação acerca de um utente conhecendo um dos atributos da classe **Utente**, foram criados os métodos procuraUtenteNIF, procuraBiUtente, procuraUtenteNome, procuraUtenteMorada, procuraUtenteCodigo que recebem como atributo o respetivo atributo pelo que qual se pretende identificar um utente. Da mesma forma, se encontram estruturados os restantes métodos relacionados com a informação do médico e do enfermeiro, sendo que ao médico acrescenta o método procurarMedicoEspecialidade e procurarMedicoCedula e ao enfermeiro o método procurarEnfermeiroEspecialidade.

3.2.6. Remover Médico, Enfermeiro, Utente e Farmacêutico

De forma a permitir a remoção de um utente, foi criado o método removeUtente que recebe como argumento o NIF do utente. Caso o utente não exista, é lançada a exceção UtenteNaoExiste. Da mesma forma, se encontram estruturados os restantes métodos removeEnfermeiro e removeMedico.

3.2.7. Criar Ato Médico e Ato de Enfermagem

O método criaAtoMedico permite efetuar o registo de um ato médico realizado. Através dos NIFs do utente e do médico do respetivo ato, passados como argumento, é estabelecida a relação com os objetos da classe **Medico** e **Utente**. A lista de medicamentos, bem como de artigos, utilizados no ato são igualmente passados como argumento com a respetiva quantidade utilizada. Caso o utente ou o médico não existam, são lançadas as exceções UtenteNaoExiste ou MedicoNaoExiste, respetivamente. De

forma automática, as quantidades de medicamentos e artigos utilizadas no ato são reduzidas ao stock dos mesmos. Esta última operação é da responsabilidade do gestorGLM, que procede a essa diminuição, através do intermediário **Operacoes**.

Da mesma forma, se encontra estruturado o método criaAtoEnfermagem à exceção de somente utilizar artigos diversos, visto que, como dito anteriormente, o enfermeiro não dispensa medicamentos.

3.2.8. Alterar informação dos Atos Médicos e dos Atos de Enfermagem

Com o objetivo de permitir a alteração da informação registada acerca de um ato, foram criados diversos métodos que recebem como argumento o ID do ato e o respetivo atributo que se pretende alterar.

3.2.9. Procurar Ato Médico e Ato de Enfermagem

Com o objetivo de permitir a alteração da informação registada acerca de um ato médico, foram criados os métodos alterarIDAtoMedico, alterarUtenteAtoMedico, alterarMedicoAtoMedico, alterarHoraAtoMedico, alterarIDListaArtigosAtoMedico. alteraQuantidadelistaArtigosAto que recebem como argumento o NIF do utente e o respetivo atributo que se pretende alterar.

3.2.10. Remover Ato Médico e Ato de Enfermagem

De forma a permitir a remoção de um ato, foram criados os métodos removeAtoMedico e removeAtoEnfermagem que recebem como argumento o ID do ato. Caso, o ato não exista, é lançada a exceção AtoEnfermagemNaoExiste ou AtoMedicoNaoExiste.

3.2.11. Login do Utente

Com o objetivo de permitir inserir na base de dados a informação que permite a autenticação do utente, foi criado o método criaLoginUtente.

3.2.12. Alterar NIF/Password do Utente

Com o objetivo de permitir a alteração na base de dados do NIF ou *password* que permite a autenticação do utente, foi criado o método criaLoginUtente.

3.2.13. Remove login Utente

Com o objetivo de permitir a remoção na base de dados da informação que permite a autenticação do utente, foi criado o método criaLoginUtente.

3.2.14. Login do GPC

Com explicado nas seções anteriores para o caso do login do Utente, a mesma lógica foi seguida para a realização dos métodos utilizados na autenticação do GPC.

3.3. Classe GestorGLM

A classe *GestorGLM* define todos os métodos que permitem a manipulação da informação sobre stocks de medicamentos e outros artigos diversos. Permite registar aquisições, efetuar consultas sobre a quantidade armazenada, e efetuar o registo de consumos, retirando ao stock tanto os medicamentos como os artigos utilizados nos vários atos.

3.3.1. Carregamento dos ficheiros para a base de dados

O método load_passGLM permite carregar os dados necessários ao login do farmacêutico do ficheiro para a base de dados. Para além disso, foram desenvolvidos métodos para gravar, atualizar e carregar a *database*.

3.3.2. Login do GLM

Com explicado nas seções anteriores (3.2.11, 3.2.12 e 3.2.13) para o caso do login do Utente, a mesma lógica foi seguida para a realização dos métodos utilizados na autenticação do GLM.

3.4. Classe Operacao

3.4.1. Carregamento dos ficheiros para a base de dados

Os métodos load_fornecedores, load_farmaceuticos load_meds, load_arts, permitem carregar os dados dos ficheiros para a base de dados. Para além disso, foram desenvolvidos métodos para gravar, atualizar e carregar a *database*.

3.4.2. Criar Fornecedor

O método criaFornecedor permite adicionar um novo fornecedor à base de dados, e, por isso, tem como argumentos todos os atributos da classe Fornecedor. Se o NIF do fornecedor (chave do *Hashmap* fornecedores) já existir na base de dados, então é lançada a exceção FornecedorJaExiste. É de salientar que os métodos utilizados para obter informação do fornecedor, procurar fornecedor por um atributo conhecimento e a sua remoção apresentam a mesma estrutura dos métodos referidos sequência de secções de 3.2.3 até 3.2.6.

3.4.3. Aumentar/Diminuir stock de medicamentos e artigos

De forma a permitir a atualização do stock de medicamentos e do stock dos artigos de cada vez que os mesmos são utilizados num ato médico ou farmacêutico, foram criados os métodos DiminuirStockArt, DiminuirStockMed, AumentaStockArt e AumentaStockMed.

3.4.4. Alertas de stocks baixos de medicamentos e de artigos

De forma a permitir o lançamento de alertas sempre que o stock de medicamentos e o stock dos artigos se encontre a baixo de um limite estipulado, foram criados os métodos AlertaArtigos e AlertaMedicamentos.

3.4.5. Criar Ato Farmacêutico

O método criaAtoFarmaceutico permite efetuar o registo de um ato farmacêutico realizada. Através do NIF do farmacêutico, passado como argumento, é feita a relação com a *HashMap* outrosartigos, que contém os objetos da classe **OutrosArtigos** e a *HashMap* medicamentos, que contém os objetos da classe **Medicamento** e as respetivas quantidades utilizadas num ato médico/ato de enfermagem. Caso o farmacêutico não exista na base de dados são lançadas as exceções FarmaceuticoNaoExiste.

3.4.6. Alterar informação do Ato Farmacêutico

A estrutura dos métodos utilizados para alterar a informação do ato Farmacêutico é semelhante à caracterizada na secção 3.2.8.

3.4.7. Procurar Ato Farmacêutico

A estrutura dos métodos utilizados para procurar um ato Farmacêutico conhecendo um dos seus atributos é semelhante à caracterizada na secção 3.2.9.

3.4.8. Criar login do GLM

A estrutura do método utilizado para o login do responsável pela gestão logística e de medicamentos é semelhante à caracterizada na secção 3.2.11.

3.5. Estatísticas

Os métodos aplicados para a realização de uma análise estatística da informação contida na base de dados encontram-se em classes diferentes. Desta forma, os referentes aos atos médicos e de enfermagem encontram-se na classe **GestorGPC** e aos atos farmacêuticos na classe **GestorGLM**.

3.5.1. Medicamentos/artigos mais utilizados

Para realizar uma análise do top 3 dos medicamentos mais utilizados nos vários atos, foram realizados os atos medicamentosMaisUtilizadosAtosMedicos, medicamentosMaisUtilizadosAtosFarmaceuticos e medicamentosMaisUtilizadosAtosEnfermagem.

3.5.2. Especialidades de médicos/enfermeiros mais comuns

Com os métodos especialidadesMedicosMaiscomuns e especialidadesEnfermeirosMaiscomuns foi possível analisar as especialidades mais comuns de entre os médicos e enfermeiros que constam na base de dados.

3.5.3. Listar Atos

Com o objetivo de listar os atos médicos e de enfermagem por especialidade, utente, médico/enfermeiro, ID do artigo/medicamento, foram criados os métodos listaAtoMedicoEspecialidade, listaAtoEnfermagemEspecialidade, listaAtoMedicoUtente, listaAtoEnfermagemUtente, listaAtoMedicoMedico,

listaAtoEnfermagemEnfermagem, listaAtoMedicoArtigo, listaAtoEnfermagemArtigo e listaAtoMedicoMedicamento.

3.5.4. Atos com mais medicamentos/artigos utilizados

Com os métodos medicamentosMaisUtilizadosAtosMedicos, medicamentosMaisUtilizadosAtosEnfermagem, artigosMaisUtilizadosAtosMedicos artigosMaisUtilizadosAtosEnfermagem, foi possível analisar o top 3 dos medicamentos/artigos que foram mais utilizados nos atos médicos e de enfermagem.

3.5.5. Fornecedores que mais forneceram artigos e medicamentos

Com os métodos fornecedoresMaisMedicamentosDif, fornecedoresMaisArtigosDif, fornecedoresMaisMedicamentos e fornecedoresMaisArtigos foi possível analisar o top 3 dos fornecedores que forneceram mais medicamentos/artigos e que medicamentos/artigos diferentes.

3.6. Classe HealthService

A classe **HealthService** é uma subclasse de UnicastRemoteObject e implementa as interfaces FuncionarioInterface, UtenteInterface e FarmaceuticoInterface. Esta classe contém todos os métodos do **GestorGPC**, do **GestorGLM** e do **Operacao** necessários para as interfaces.

3.7. Classe InitGestores e Configuracoes

A classe InitGestores é responsável pelo carregamento inicial dos ficheiros para a base de dados do respetivo gestor, ou seja, carrega os ficheiros e guarda-os em database.dat.

Quanto à classe **Configuracoes** foi criada com o objetivo de agregar as "configurações" necessárias e, desta forma, facilitar alterações.

3.8. Interfaces

O objetivo das interfaces é fazer a conexão entre as implementações localizadas dentro do servidor e o cliente através do RMI. Implementar uma interface, permite que uma classe se torne mais formal sobre o comportamento que fornece e corresponda ao conjunto de métodos que podem ser implementados. Assim, o RMI permite que um cliente obtenha uma interface para cada máquina virtual com os métodos que ele está autorizado a implementar. Foram definidas três interfaces: **FarmaceuticoInterface**, **FuncionarioInterface** e **UtenteInterface**

3.9. Servidor

A classe **AplicacaoServidor** possui uma *main* que cria uma instância da implementação do objeto remoto, exporta o objeto remoto e, em seguida, liga essa instância a um nome num JAVA RMI *registry*. Esta classe é o servidor da aplicação. Regista um serviço da **HealthService** na porta 5091, que pode ser alcançada através de rmi://localhost:5092/health.

3.10. Cliente

No package cliente, a chamada do método *Naming.lookup()* inspeciona o Registo RMI em execução no *host local* para uma ligação com *Configurations.rmiServiceLocation*. Em seguida, retorna um objeto que deve ser alcançado por qualquer interface remota que se queira, permitindo assim, usar esse objeto para chamar os métodos remotos definidos nessa interface.

Foram definidos três clientes: **FuncionarioAplicacao**, **UtenteAplicacao** e **FarmaceuticoAplicacao**. Na implementação destes diferentes clientes, foram criados vários programas cada um a chamar os métodos da interface que são necessários. Para exemplificar o seu funcionamento foram criados alguns casos práticos. De salientar que, na execução dos vários clientes podem ser enviadas exceções. Estas foram criadas como sendo subclasses da *RemoteException* para poderem ser enviadas pela rede, visto que, as exceções normais não podem, pois não são *Serializable*.

4. Conclusão

A realização deste trabalho permitiu a consolidação de diversos conhecimentos sobre a linguagem de programação JAVA e a implementação de objetos remotos. Com a utilização da arquitetura Cliente-Servidor (utilizando RMI) e de diversas estruturas de dados foi possível criar um sistema de gestão.

No desenvolvimento deste trabalho surgiram algumas dificuldades, principalmente no que diz respeito à sincronização e a partilha de informação entre os vários sistemas de gestão, garantindo simultaneamente a confidencialidade da mesma. Apesar disto, os objetivos propostos para este trabalho foram cumpridos mediante os testes realizados nos diferentes clientes.

No que diz respeito a melhorias futuras, poder-se-iam ter explorado mais os métodos criados, como por exemplo, para um estudo estatístico mais pormenorizado.

Referências

[1] Esmond Pitt and Kathy McNIFf. Java. rmi: The Remote Method Invocation GuIDe. AddisonWesley Longman Publishing Co., Inc., 2001.

[2] AGNALDO KIYOSHI Noda. Mecanismo de invocação remota de métodos em máquinas virtuais. PhD thesis, Master's thesis, Pontificia UniversIDade Católica do Paraná, 2005