

**AS 2021-2022****Projeto 1 - AS****Tema: Health Centre****Autores:****João Ferreira nºmec 80041****João Magalhães nºmec 79923****Data: 18/04/2022****Conteúdo**

1. INTRODUÇÃO	2
2. PRINCIPAIS ENTIDADES.....	2
3. COMUNICAÇÃO.....	3
4. <i>Control Centre</i>	3
5. <i>Health Centre</i>.....	5
5.1 MONITORES UTILIZADOS	5
5.1.1 <i>Entrance Hall</i>	5
5.1.2 FLUXO DO PACIENTE - <i>Entrance Hall</i>	6
5.1.3 <i>Evaluation Hall</i>	6
5.1.4 FLUXO DO PACIENTE - <i>Evaluation Hall</i>	6
5.1.5 <i>Waiting Hall</i>	6
5.1.6 FLUXO DO PACIENTE - <i>Waiting Hall</i>	6
5.1.7 <i>Medical Hall</i>	6
5.1.8 FLUXO DO PACIENTE - <i>Medical Hall</i>	7
5.1.9 <i>Payment Hall</i>	7
5.1.10 FLUXO DO PACIENTE - <i>Payment Hall</i>	7
5.2 METODOLOGIA PARA O DESENVOLVIMENTO DOS BOTÕES SUSPEND, RESUME, STOP, END	7
6. <i>LOGGER</i>.....	8
7. <i>COMO EXECUTAR O PROJETO</i>	8
8. <i>LIMITAÇÕES</i>.....	8
9. <i>CONTRIBUIÇÕES</i>.....	9
10. <i>CONCLUSÃO</i>	9



1 Introdução

Este trabalho foi realizado no âmbito da Unidade Curricular Arquiteturas de Software do Mestrado em Engenharia Informática da Universidade de Aveiro e visa a implementação de um sistema de simulação de um centro de saúde no qual várias entidades (*Control Centre*, *Health Centre*, múltiplos pacientes, *call centre*, quatro enfermeiras e quatro médicos) interagem entre si, concorrendo simultaneamente pelos recursos disponíveis. Durante cada simulação, existe um previamente determinado número de pacientes que irão percorrer o caminho desde a entrada até à sala de pagamento, passando entre estas nas salas de avaliação, espera e de atendimento. Cada simulação será controlada por um *Control Centre* que enviará comandos para o *Health Centre* como, por exemplo, o comando para parar a simulação.

Este projeto foi desenvolvido utilizando a linguagem de programação Java 11, utilizando sockets para estabelecer a comunicação entre o *Control Centre* e o *Health Centre*. Foram definidas duas interfaces gráficas para representar estas duas entidades, recorrendo à ferramenta Java Swing, permitindo assim observar mais facilmente o estado e localização dos pacientes.

2 Principais Entidades

Para a implementação do projeto proposto, foi necessário criar duas entidades principais: o *Control Centre* e o *Health Centre*. Estas entidades comunicam entre si através de sockets sendo o *Control Centre* o cliente e o *Health Centre* o servidor. O *Control Centre*, como o nome indica, permite controlar o estado da simulação, isto é, permite começar, parar ou até mesmo terminar a simulação. Por outro lado, o *Health Care* contém toda a lógica relacionada com o controlo da concorrência no acesso às regiões partilhadas. Para o controlo desse acesso às respetivas regiões criámos cinco monitores, o *MEntranceHall*, *MEvaluationHall*, *MWaitingHall*, *MMedicalHall* e o *MPaymentHall*. Criámos também 6 entidades ativas: o *TPatient*, *TNurse*, *TDoctor*, *TCashier*, *TCallCentre*, *TServerClient*.

- A classe *TCallCentre* representa a entidade que controla o movimento dos pacientes, verificando se há lugares vazios nos respetivos halls. Caso isso se verifique, chama os pacientes pela ordem de entrada no hall anterior ou, no caso de existir alguma restrição como, por exemplo, no waiting hall, no qual deve chamar pelo respetivo DoS (Cor) do paciente, chamando primeiro os que têm DoS superior (vermelho). Se na sala estiver já um paciente com DoS menor (azul), por exemplo, o *Call Centre* não vai ficar à espera dos que têm DoS superior (amarelo ou vermelho), ordenando assim a entrada do paciente com DoS inferior.
- A classe *TPatient* representa os pacientes que são criados em cada simulação e que vão executar diversas tarefas, como sair de um hall, entrar no seguinte e efetuar o pagamento.
- A classe *TNurse* representa as enfermeiras que realizam a avaliação do paciente e atribuem-lhe um grau de DoS (vermelho, amarelo ou azul).
- A classe *TDoctor* representa os médicos que realizam a consulta no *Medical Hall*.
- A classe *TCashier* representa a entidade que recebe os pagamentos dos pacientes e gere a ocupação do payment hall.
- A classe *TServerClient* representa a entidade que verifica as mensagens que são enviadas.



3 Comunicação

A comunicação entre o Control Centre e o Health Centre é realizada através de uma socket TCP. Como foi referido anteriormente, o CC funciona como cliente e o HC como servidor. O CC liga-se ao HC, assim que este estiver disponível, no IP 127.0.0.1 e na porta 8888, enviando-lhe mensagens de acordo com o botão que é premido na respetiva interface. Por sua vez, o HC está constantemente à espera de novos clientes e, assim que aceita um novo cliente através do método `accept()`, cria um novo objeto do tipo `TServerClient` (um objeto que estende a classe `Thread`). Posteriormente, é executado o método `start()` deste novo objeto. Este objeto será responsável por fazer a gestão das mensagens que são enviadas apenas por um cliente, ou seja, por exemplo, no caso de serem criados dez clientes teríamos dez instâncias desta classe. Esta implementação foi desenvolvida com base nas implementações das seguintes referências: [1] [3].

As mensagens enviadas pelo CC são diferentes de acordo com a ação que é pretendida. Assim, foram definidos diferentes tipos de mensagens para cada botão disponível na interface do CC. As mensagens que utilizamos para a execução deste projeto estão definidas na tabela acima.

Botão	Objetivo	Mensagem
Start	Iniciar a Simulação	"CC CREATE <número de adultos> <número de crianças textgreater <número de lugares disponíveis\> <time to move> <evaluation time> <medical time> <payment time>"
SUSPEND	Suspender a simulação	"CC SUSPEND"
RESUME	Retomar a simulação	"CC RESUME"
STOP	Parar a simulação atual. É possível executar mais do que uma simulação durante a mesma execução.	"CC STOP"
END	Terminar a execução.	"CC END"

As ações executadas são diferentes para cada tipo de mensagem. Assim, na tabela seguinte exibimos as ações que cada mensagem espoleta.

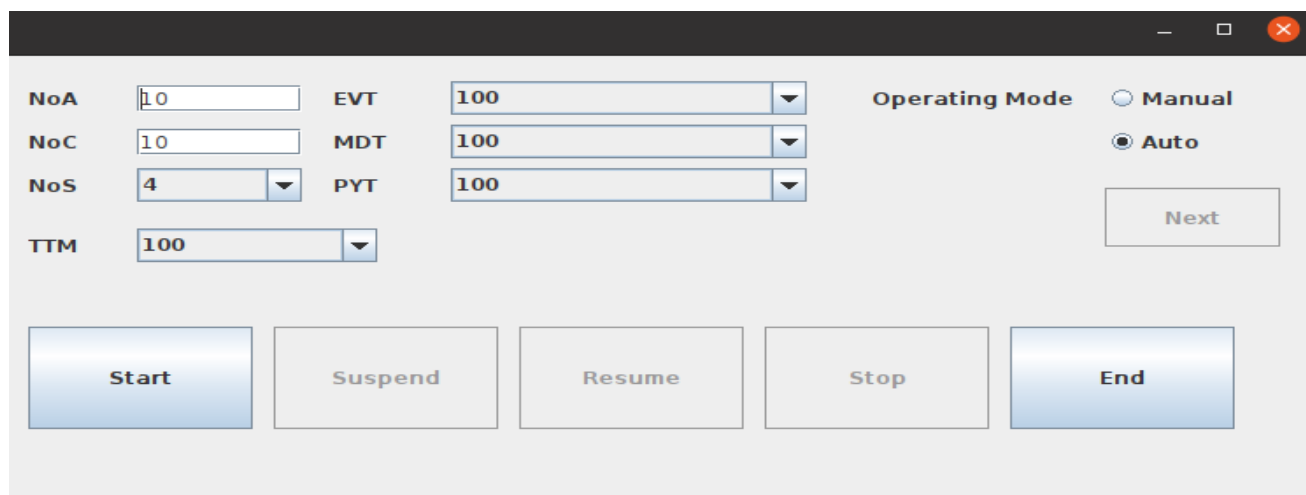
Botão	Ação
START	Início das threads do Call Centre, das quatro enfermeiras, dos quatro médicos e de todos os pacientes.
SUSPEND	Suspender as threads de todos os pacientes. Os pacientes que foram criados anteriormente são guardados numa <code>ArrayList</code> para que possam ser suspensos no momento desta ação.
RESUME	Retomar as threads de todos os pacientes.
STOP	Parar as threads de todos os pacientes.
END	Terminar todas as threads.

4 Control Centre

O *Control Centre*, tal como mencionado anteriormente, controla o estado da simulação, bem como quantos pacientes serão criados e os respetivos tempos de movimento, de avaliação por parte das enfermeiras, o tempo da consulta e o tempo de pagamento. O *Control Centre* apresenta uma interface visual, implementada

com Java Swing. A interface foi desenvolvida de modo a poder sofrer alterações, como, por exemplo, a possibilidade de alguns botões estarem desativados. Por exemplo, no início apenas os botões *Start*, *End* e os campos de controlo de execução como o *time to move*, número de adultos, entre outros estão disponíveis mas, pressionando o botão *start*, só ficam disponíveis os botões *suspend*, *resume* e *end*, ficando o resto da interface inativa. Na imagem abaixo podemos ver a interface que foi desenvolvida para representar o *Control Centre*, sendo que cada botão tem uma função diferente.

- NoA corresponde ao número de pacientes adultos que podemos criar no início de cada simulação. Este número está compreendido entre 0 e 50.
- NoC corresponde ao número de pacientes crianças que podemos criar no início de cada simulação. Este número está compreendido entre 0 e 50..
- NoS representa quantos lugares irão estar disponíveis em cada hall do *health centre*.
- TTM representa o *time to move* dos pacientes, sendo este igual para todos.
- EVT representa o tempo que demora a avaliação realizada pelas enfermeiras. No caso da opção para o EVT corresponder a um número aleatório, é atribuído um número aleatório diferente para cada paciente.
- MDT representa o tempo que demora a consulta realizada pelos médicos. No caso da opção para o MDT corresponder a um número aleatório, é atribuído um número aleatório diferente para cada paciente.
- PYT representa o tempo que demora o pagamento de cada paciente. No caso da opção para o PYT corresponder a um número aleatório, é atribuído um número aleatório diferente para cada paciente.
- *Operating Mode* corresponde ao modo da simulação, que pode ser automática ou manual.
- *Start* começa a simulação.
- *Suspend* suspende a simulação atual.
- *Resume* retoma a simulação atual.
- *Stop* termina a simulação atual.
- *End* termina a execução.



The screenshot shows a Java Swing window titled "Control Centre". It contains several input fields and buttons. On the left, there are four rows of controls: "NoA" with a text box containing "10", "NoC" with a text box containing "10", "NoS" with a dropdown menu showing "4", and "TTM" with a dropdown menu showing "100". To the right of these are three rows: "EVT" with a dropdown menu showing "100", "MDT" with a dropdown menu showing "100", and "PYT" with a dropdown menu showing "100". Further right, under the heading "Operating Mode", there are two radio buttons: "Manual" (unselected) and "Auto" (selected). Below the radio buttons is a "Next" button. At the bottom of the window, there are five large buttons: "Start", "Suspend", "Resume", "Stop", and "End". The "Start" and "End" buttons are highlighted with a blue gradient, while the others are plain grey.

Figura 1: Control Centre

5 Health Centre

De modo a desenvolvermos este projeto definimos vários monitores que representam as áreas por onde os pacientes irão passar até ao fim do respetivo ciclo de vida. Assim, os monitores definidos foram: *Entrance Hall* (MEntranceHall), *Evaluation Hall* (MEvaluationHall), *Waiting Hall* (MWaitingHall), *Medical Hall* (MMedicalHall) e *Payment Hall* (MPaymentHall). À medida que os pacientes vão passando por estas áreas são bloqueados, recorrendo a *Reentrant Locks*[2], ficando à espera de permissão para poderem entrar e/ou sair. Para a implementação destas áreas partilhadas poderíamos ter utilizado também métodos *synchronized* que nos levariam a um resultado equivalente, mas foi-nos pedido para utilizarmos *Reentrant Locks*.

Para além destes monitores, implementamos também um FIFO que foi utilizado para a gestão das respetivas salas de cada um dos corredores. Este FIFO foi desenvolvido com base no FIFO que foi mostrado nas aulas desta UC.

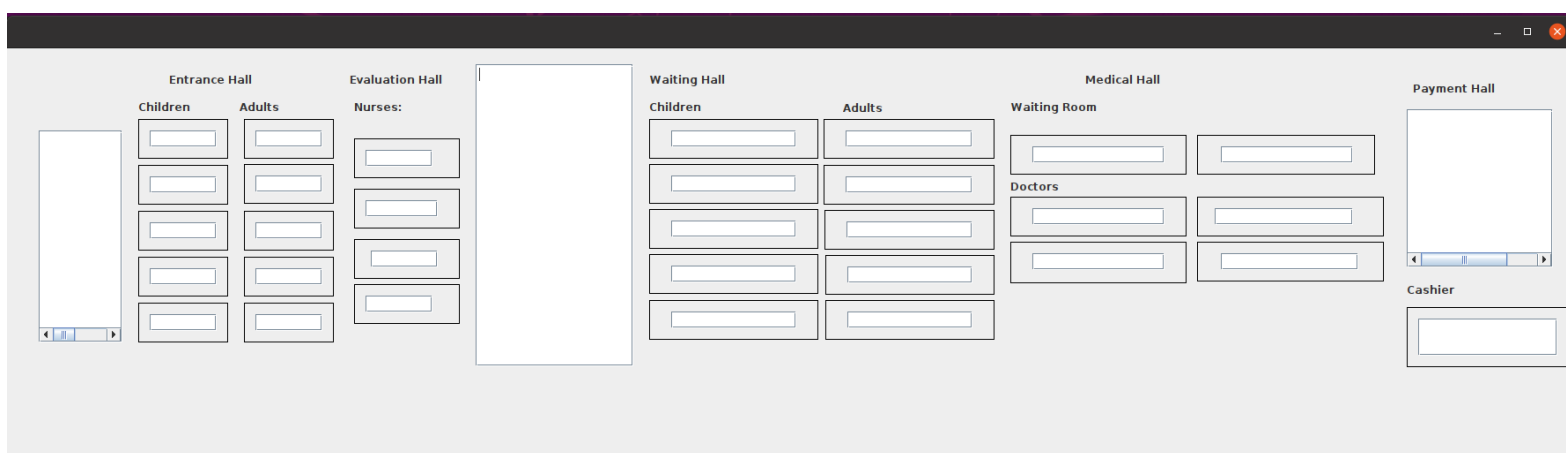


Figura 2: Health Centre

5.1 Monitores Utilizados

Como foi mencionado anteriormente, implementamos cinco monitores: *Entrance Hall* (MEntranceHall), *Evaluation Hall* (MEvaluationHall), *Waiting Hall* (MWaitingHall), *Medical Hall* (MMedicalHall) e *Payment Hall* (MPaymentHall). Para além destes monitores, utilizamos também o FIFO leccionado nas aulas da UC (devidamente corrigido).

5.1.1 Entrance Hall

O *Entrance Hall* é composto por uma sala inicial e também por duas salas (uma para as crianças e outra para os adultos). Antes de entrarem nestas segundas salas, os pacientes esperam na primeira sala até que haja lugares livres na respetiva segunda sala (das crianças ou dos adultos). Cada uma das segundas sala tem metade do total de lugares disponíveis (NoS). Sempre que um paciente entra no *Entrance Hall* é-lhe atribuído um número sequencial (ETN) e de seguida entra na segunda sala. Os pacientes saem da primeira sala pela ordem crescente do número ETN.



5.1.2 Fluxo do paciente - *Entrance Hall*

Os pacientes começam por entrar na primeira sala, não tendo quaisquer restrições em relação aos lugares disponíveis. Para entrarem na primeira sala, os pacientes invocam o método *joinFirstRoom()* do *Entrance Hall*, sendo adicionados a esta sala.

De seguida, executam um *sleep* de modo a simular o *time to move*. Posteriormente, entram na segunda sala se houver lugares livres, voltam a executar o *sleep* do *time to move* e ficam bloqueados à espera de uma chamada (*signal*) do *Call Centre*. Para implementar este bloqueio de cada paciente recorremos a um *lock* e uma *condition* desse mesmo *lock*, na qual o paciente irá invocar o método *await*.

5.1.3 *Evaluation Hall*

O *Evaluation Hall* é o local no qual é atribuída uma cor ao paciente. É composto por quatro salas, que são geridas por quatro enfermeiras (uma por cada sala). Os pacientes são chamados para esta sala pelo *Call Centre*, que os acorda invocando o método *signal*. Após entrarem nesta sala, os pacientes ficam bloqueados à espera que a enfermeira da respetiva sala lhes atribua uma cor. O tempo para a atribuição da cor de cada paciente (EVT) pode ser diferente entre pacientes. Após saírem do *Evaluation Hall* os pacientes dirigem-se para o *Waiting Hall*.

5.1.4 Fluxo do paciente - *Evaluation Hall*

Após passarem pelo *Entrance Hall* e serem acordados pelo *Call Centre*, os pacientes saem do *Entrance Hall* e entram no *Evaluation Hall*, se houver algum lugar disponível, e ficam à espera que a enfermeira da respetiva sala lhe atribua uma cor. O paciente fica à espera numa *condition* de um *lock*. Após ter uma cor atribuída, o paciente simula o *evaluation time* e o *time to move*.

Após esta atribuição, o paciente sai da segunda sala do *Evaluation Hall* e desloca-se para a primeira sala do *Waiting Hall*, que tem lugares ilimitados, ou seja, não há qualquer limitação de entrada.

5.1.5 *Waiting Hall*

O *Waiting Hall* é semelhante ao *Entrance Hall* na medida em que é composto por uma primeira sala sem limite de lugares e duas salas (uma para adultos e outra para crianças), sendo que cada uma tem metade dos lugares totais. Cada vez que um paciente entra no *Waiting Hall* é-lhe atribuído um número de espera. Caso não haja lugares livres nas salas, o paciente deve esperar, caso contrário deve avançar para as suas salas. Já dentro das salas, os pacientes são chamados para se moverem do *Waiting Hall* para o *Medical Hall* pelo *Call Centre* pela cor e pelo número.

5.1.6 Fluxo do paciente - *Waiting Hall*

Após saírem do *Evaluation Hall*, os pacientes entram na primeira sala do *Waiting Hall*. De seguida, entram na respetiva segunda sala, se houver lugares livres. Após a entrada na segunda sala, os pacientes ficam à espera da chamada do *Call Centre* numa *condition* específica da respetiva cor e tipo (adulto ou criança). Quando o *Call Centre* o chamar, o paciente move-se para o *Medical Hall*, de acordo com o respetivo *time to move*.

5.1.7 *Medical Hall*

O *Medical Hall* é composto por uma sala inicial com dois lugares (um para crianças e outro para adultos) e por quatro segundas salas com apenas um lugar cada. Cada sala é gerida por um médico.



Os pacientes saem da primeira sala e entram na segunda sempre que a correspondente segunda sala está livre. Quando entram na segunda sala, os pacientes despendem lá um determinado tempo em milissegundos (MDT).

5.1.8 Fluxo do paciente - *Medical Hall*

O paciente começa por entrar na primeira sala do *Medical Hall*. Cada paciente informa o *Call Centre* de que há um lugar livre no *Waiting Hall*, dado que acabou de sair desse corredor.

De seguida, cada paciente fica à espera de um *signal* do *call centre* para poder entrar na segunda sala. Posteriormente, o paciente fica à espera da consulta realizada pelo médico. Assim que o médico lhe faz *signal*, o paciente sai da segunda sala e movimenta-se para o *Payment Hall*, com o respetivo *time to move*.

5.1.9 *Payment Hall*

O *Payment Hall* é composto por duas salas:

- a primeira sala na qual os pacientes esperam pela chamada do *cashier* para a segunda sala. A primeira sala tem lugares ilimitados.
- a segunda sala na qual é realizado o pagamento de cada paciente, apenas com um lugar disponível.

5.1.10 Fluxo do paciente - *Payment Hall*

O paciente começa por entrar na primeira sala (sem limite de lugares disponíveis) e fica bloqueado à espera de um *signal* do *cashier*. Assim que for acordado pelo *cashier*, o paciente movimenta-se para a segunda sala, onde irá realizar o pagamento de acordo com o tempo de pagamento definido na interface do *control centre*. Após realizar o pagamento, o paciente sai da segunda sala e termina o respetivo ciclo de vida.

5.2 Metodologia para o desenvolvimento dos botões SUSPEND, RESUME, STOP, END

Como referido anteriormente, foram desenvolvidos botões com certas funcionalidades como suspender, retomar, interromper e terminar a simulação.

Na implementação realizada, no caso de querermos suspender a simulação, temos de pressionar o botão *suspend* e assim tornámos a *flag isPaused* em *true*. Assim, os pacientes ficam bloqueados num *lock*, à espera de uma nova ação nessa *flag*.

Se o objetivo for retomar a simulação que suspendemos anteriormente temos de clicar no botão *resume*, para tornarmos a *flag isPaused* em *false*. Posteriormente, os pacientes continuam o seu percurso.

De modo a interromper a simulação atual, devemos clicar no botão *stop*. Clicando neste botão a *flag isStopped* fica equivalente a *true*, interrompendo a ação dos pacientes. Posteriormente, é realizado um *interrupt* nas *threads* dos pacientes para os parar e assim ser possível criar uma nova simulação sem que nenhum destes pacientes esteja a executar.

No caso de querermos terminar todas as *threads* devemos clicar no botão *end*. Pressionando este botão realizamos um *System.exit(0)*, terminando a execução do programa.



6 Logger

Como pedido, o estado da simulação está a ser escrito num ficheiro de texto chamado *LOG.TXT* e no terminal. Para isso criámos um monitor chamado *MLogger*. Como este só pode ser monitorizado pelo HCP o *MLogger* é passado a todos os monitores de maneira a que seja possível perceber que passos estão a acontecer.

A informação inicial da simulação, isto é, o número de pacientes, o tempo que demoram a fazer um movimento, o tempo de uma consulta e os restantes campos iniciais, são conhecidos através do *TServerClient* que verifica as mensagens que são recebidas.

Qualquer alteração/passo na simulação que aconteça, isto é, sempre que uma thread paciente passar por um hall ou sempre que lhe é atribuída uma cor por exemplo, vai ser escrita no ficheiro *logger*. Se o utilizador alterar o estado da simulação ao clicar nos botões referidos no ponto anterior do relatório, o *logger* apresentará essa informação para assim sabermos sempre que existir uma alteração no estado da simulação.

7 Como Executar o projeto

Para executar este projeto é necessário executar:

- a classe *CCP_main.java* (package *CC*);
- a classe *HCP_main.java* (package *HC.Main*);

A execução destas duas classes irá disponibilizar as duas interfaces necessárias.

8 Limitações

Apesar deste projeto funcionar corretamente existem algumas limitações. Estas limitações não afetam o bom funcionamento do mesmo. No momento da implementação do logger tivemos algumas dificuldades, nomeadamente na ordem da escrita para o ficheiro em questão. Estas dificuldades refletiram-se em dois momentos:

- no momento *STOP* (parar a simulação) é perceptível pela interface que todos os pacientes param. Contudo, no logger é escrita a linha que indica que o botão *STOP* foi clicado e nas linhas abaixo aparecem poucos pacientes nas salas dos corredores existentes.
- conseguimos perceber, observando a interface, que os pacientes passam da primeira sala do entrance hall para a segunda pela ordem de chegada à primeira sala. Contudo, no logger as linhas são escritas com uma ordem errada (apesar dos pacientes saírem pela ordem correta). Este erro não se verifica se executarmos apenas com um tipo de pacientes (só adultos ou só crianças).

Pensamos que estes problemas estão relacionados com a ordem em que escrevemos para o ficheiro *LOG*. Finalmente, não implementamos o modo manual.



9 Contribuições

Consideramos que ambos os elementos do grupo contribuíram de forma equivalente para o desenvolvimento deste projeto. Assim, a contribuição de cada elemento foi igual a 50% do trabalho desenvolvido. As tarefas realizadas por cada elemento estão evidenciadas na tabela abaixo.

Contribuição	Autor
CC Interface	João Ferreira
HC Interface	João Magalhães
Comunicação via sockets	João Ferreira
Pacientes	João Magalhães
Enfermeiras	João Ferreira
Médicos	João Magalhães
Call Centre	João Ferreira
Cashier	João Magalhães
Entrance Hall	João Ferreira
Evaluation Hall	João Magalhães
Waiting Hall	João Ferreira
Medical Hall	João Magalhães
Payment Hall	João Ferreira
Logger	João Magalhães
Relatório	Ambos

10 Conclusão

Concluindo, consideramos que cumprimos todos os objetivos propostos, seguindo todas as especificações do enunciado e também definindo alguns parâmetros com base na nossa pesquisa à medida que fomos avançando no desenvolvimento deste projeto, tal como nos foi indicado.

Apesar das limitações anteriormente mencionadas neste documento, pensamos ter atingido o principal objetivo deste trabalho.

Referências

- [1] Multi-threaded client/server in java. <http://net-informations.com/java/net/multithreaded.htm>. Accessed: 2022-04-14.
- [2] Reentrantlock (java platform se 7) - oracle help center. <https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/locks/ReentrantLock.html>. Accessed: 2022-04-14.
- [3] David Dobervich. Java socket programming 4 - multi-client interactive sessions.