INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Departamento de Engenharia de Eletrónica e Telecomunicações e de Computadores

Desenvolvimento de Dashboard e ferramentas de análise para projeto PREMO

Rúben Dias

Licenciado em Engenharia de Informática e de Computadores

Trabalho de Projecto para obtenção do Grau de Mestre
em Engenharia Informática e de Computadores

Orientadores : Doutor Artur Jorge Ferreira
Doutora Iola Maria Silvério Pinto
Doutor Carlos José Brás Geraldes

Júri:
Presidente: Doutor Nuno Miguel Soares Datia
Vogais: Doutor Paulo Manuel Trigo Cândido da Silva
Doutor Artur Jorge Ferreira

Novembro, 2023

Agradecimentos

O desenvolvimento deste trabalho teve o contributo de diversas pessoas. Gostaria de agradecer aos professores Artur Ferreira, Iola Pinto e Carlos Geraldes por me atribuírem o trabalho e fazer parte do projeto PREMO. Ainda também pela orientação e suporte dado no desenvolvimento do mesmo, e pela disponibilidade que tiveram para questões que tive.

Também gostaria de agradecer aos colegas de trabalho no âmbito do projeto PREMO, nomeadamente a colega Cristiana Von Rekowski e o Doutor Luís Bento, por se mostrarem disponíveis a tirar questões que possam ter havido na realização deste trabalho, sobretudo na área clínica.

Este trabalho foi suportado pelo projeto FCT grant DSAIPA/DS/0117/2020 - PREMO - Predictive Models of COVID-19 Outcomes for Higher Risk Patients Towards a Precision Medicine.

Resumo

A doença provocada pelo Coronavírus 2019 (COVID-19) provocou um período de pandemia, devido à sua facilidade de transmissão e elevado número de casos de infeções. A evolução desta pandemia e as suas severas consequências para a mortalidade e morbilidade das populações, sobretudo das mais idosas, originou o desenvolvimento de inúmeros estudos científicos e projetos de investigação. O projeto de investigação *Predictive Models of COVID-19 Outcomes for Higher Risk Patients Towards a Precision Medicine* (PREMO) surge neste contexto.

Neste trabalho de projeto, trata-se o desenvolvimento do sistema de suporte informático ao projeto PREMO. Apresentam-se os aspetos principais da solução desenvolvida, ao nível de base de dados, *back-end* e *front-end*.

Sobre a base de dados, descreve-se o modelo de dados, e as suas funcionalidades. As implementações do *back-end* e *front-end* são descritas em detalhe.

Dá-se especial ênfase à componente de *front-end*, através da descrição da implementação de um painel (*dashboard*). Este painel possibilita a visualização interativa de dados de pacientes internados na Unidade de Cuidados Intensivos (UCI), ao longo das seis vagas de Doença por Coronavírus 2019 (COVID-19).

A utilização do *dashboard* possibilita aos clínicos a extração e visualização de dados de acordo com uma vasta seleção de características demográficas e clínicas, como por exemplo valores de normalidade versus não-normalidade de parâmetros laboratoriais de interesse, bem como a análise em detalhe ao longo das seis primeiras vagas da pandemia de COVID-19 em Portugal.

Palavras-chave: COVID-19; *dashboard*; PREMO; pandemia; visualização de dados; curvas de sobrevivência; vagas COVID-19

Abstract

The disease caused by the Coronavirus 2019 (COVID-19) caused a pandemic period, due to its ease of transmission and the elevated number of infections. The evolution of this pandemic and its severe consequences to the population mortality and morbidity, above all the most elderly, led to the development of different countless investigation projects and cientific studies. The investigation project *Predictive Models of COVID-19 Outcomes for Higher Risk Patients Towards a Precision Medicine* (PREMO) arises in this context.

This work deals with the development of informatic support system for the PREMO project. The main aspects of the developed solution are presented, at the database level, back-end and front-end.

Regarding the database, a data model is described, and its functionalities. The implementations of the back-end and front-end are described in detail.

A given special emphasis is given to the front-end component, through a dashboard implementation description. This dashboard allows the interactive visualization of data of hospitalized patients in the intensive care unit, in the span of the six waves of COVID-19.

The dashboard use allows the clinicians to extract and visualize data with a vast selection of clinical and demographic characteristics, such as normality versus non-normality values of laboratorial parameters of interest, as well as the detailed analysis throughout the first six COVID-19 waves in Portugal.

Keywords: COVID-19, dashboard, PREMO, pandemic, data visualization; survival curves; COVID-19 waves

Índice

Lista de Figuras

Lista de Tabelas

Lista de Listagens

Lista de Abreviaturas e Siglas

API *Application Programming Interface*. x, 39, 40, 41, 45, 46, 48, 49, 50, 56, 58, 59, 60, 70, 77, 81, 82

COVID-19 Doença por Coronavírus 2019. v, vii, xi, 1, 2, 3, 7, 8, 9, 10, 11, 16, 19, 21, 40, 52, 63, 67, 81

CSS *Cascading Style Sheets*. 46

CSV *Comma-separated values*. 50, 82

DCD Dados Clínicos e Demográficos. xi, xv, 15, 18, 19, 21, 34, 36, 37, 50, 81, 82

DL Dados Laboratoriais. xi, xv, 8, 15, 16, 17, 23, 34, 37, 50, 81, 82, 83

ECMO *Extracorporeal Membrane Oxygenation*. 21, 34, 35, 37

ER Entidade-Relação. xi, xii, 9, 34, 35, 82

ETL *Extract, Transform, and Load*. 3, 4, 15, 22, 23, 25, 28, 31, 39, 40, 41, 50, 58, 59, 60, 82, 83

GEE *Generalized Estimating Equation*. 8

HTML *HyperText Markup Language*. 45

HTTP *Hypertext Transfer Protocol*. 44, 48, 51, 58, 61

HTTPS *Hypertext Transfer Protocol Secure*. 58, 61

JSON *JavaScript Object Notation*. 26, 27, 33, 43, 50

JWT *JSON Web Token*. 51, 52

OWASP *Open Worldwide Application Security Project*. 51

PREMO *Predictive Models of COVID-19 Outcomes for Higher Risk Patients Towards a Precision Medicine*. v, vii, 2, 7, 15, 23

SPA *Single-Page Application*. 44
SQL *Structured Query Language*. 33, 36, 40, 49

TLS *Transport Layer Security*. 58, 59, 61

UCI Unidade de Cuidados Intensivos. v, xiii, 2, 7, 8, 16, 20, 33, 35, 37, 52, 53, 71, 72, 73, 74, 77, 79, 81
URL *Uniform Resource Locator*. 55, 58

valor-p Valor de probabilidade. xiii, 70, 71, 72
VMI Ventilação Mecânica Invasiva. 2, 8, 21, 34, 35, 37

1

Introdução

Este capítulo introdutório é composto por cinco secções. Na Secção 1.1 é apresentado o contexto e enquadramento deste trabalho. Seguem-se os objetivos do mesmo descritos na Secção 1.2. Na Secção 1.3 apresenta-se um resumo da solução desenvolvida. De seguida apresentam-se as contribuições do projeto na Secção 1.4. Por fim, na Secção 1.5 é descrita a organização global do documento.

1.1 Contexto e enquadramento

A COVID-19 é causada por um coronavírus o qual foi identificado, pela primeira vez, em humanos na cidade chinesa de Wuhan em Novembro de 2019. Esta originou um período de pandemia, devido ao seu elevado número de casos e facilidade de transmissão. Esta doença pode causar infeções respiratórias, sintomas semelhantes aos da gripe, e pode também evoluir para condições mais graves, como o caso de uma pneumonia e morte [57].

A evolução desta pandemia e as suas consequências refletiram-se na mortalidade e morbilidade das populações, originando o desenvolvimento de diversos estudos científicos e projetos de investigação, tais como [1] e [30]. O primeiro destes é um estudo, realizado em Lérida, Espanha, que compara a variante Alfa, detetada pela primeira vez no Reino Unido, e a variante Delta, detetada pela primeira vez na Índia. Neste concluiu-se que a variante Alfa é mais agressiva, com maior presença de sintomas do que a Delta, em grande parte devido à falta de vacinação dirigida à variante Delta [1].

O segundo estudo, compara as cinco primeiras vagas de COVID-19 na cidade de

Teerão no Irão. Neste, concluiu-se que a agressividade do vírus, e consequentemente a probabilidade de morte, aumentou da primeira vaga até à terceira, onde se verificam os casos mais graves. Na quarta e quinta vagas a gravidade decresceu gradualmente [30].

O presente trabalho foi realizado no âmbito do projeto de investigação *Predictive Models of COVID-19 Outcomes for Higher Risk Patients Towards a Precision Medicine* (PREMO) e aprovado pela Comissão de Ética Institucional do Centro Hospitalar Universitário de Lisboa Central (1043/2021, 20/05/2020). Este tem como objetivo o desenvolvimento de modelos preditivos da COVID-19, para promover decisões médicas mais rápidas e precisas, para a diminuição de eventos graves, recuperações mais rápidas e uma redução significativa de fatalidades.

Em relação ao projeto PREMO, foi realizado um estudo [53], com os dados das três primeiras vagas da COVID-19. Nesse estudo analisaram-se os dados demográficos, clínicos e laboratoriais de 337 pacientes com COVID-19 internados na UCI do Centro Hospitalar Universitário Lisboa Central, em Portugal, entre março de 2020 e março de 2021. A análise dos dados longitudinais obtidos ao longo da permanência dos pacientes na UCI, teve como principal objetivo a identificação de biomarcadores associados aos eventos mais severos, como a morte na UCI e a necessidade de Ventilação Mecânica Invasiva (VMI).

1.2 Objetivos

O objetivo do trabalho apresentado neste documento é o desenvolvimento de uma ferramenta de suporte informático, com painel de visualização (*dashboard*) de diversos dados clínicos de pacientes internados na UCI de hospitais de Lisboa, ao longo das seis primeiras vagas de COVID-19 em Portugal.

A primeira etapa deste projeto é a criação de diversos procedimentos para realizar a aquisição e o pré-processamento dos dados fornecidos. Segue-se a criação de uma base de dados relacional que agrega dados de diferentes formatos. Esta base de dados vai alimentar uma ferramenta de visualização, com um painel interativo (*dashboard*).

Este termo de *dashboard*, tem origem numa placa feita de madeira ou pele usado antigamente nas carroças, como forma de proteção do condutor e passageiros dos detritos ou sujidade vindo dos cavalos [10]. Esta placa, com a evolução das carroças para carros, tornou-se o sítio ideal para apresentar **informações importantes** ao condutor, como por exemplo a velocidade ou as rotações do motor. Hoje em dia, o termo *dashboard* refere a representação de informações que sejam úteis ao seu utilizador.

No caso destas aplicações informáticas, o termo refere-se à apresentação de gráficos ou diagramas para ter uma visualização mais facilitada dos dados. A definição indicada por Wexler [66] é, "*A dashboard is a visual display of data used to monitor and/or facilitate understanding*".

A aplicação a desenvolver tem os seguintes requisitos funcionais:

- Importação e exportação de dados;
- Visualização de dados a partir de vários tipos de gráficos estatísticos, tais como os de barras, circulares, dispersão e diagramas em caixa, curvas de sobrevivência e gráficos de linhas;
- Filtrar os dados pelas vagas de COVID-19;
- Preenchimento de formulários para seleção dos dados a visualizar;
- Gestão de utilizadores, como a adição, remoção e edição. Este apenas é disponibilizado a utilizadores com o papel de administrador;
- Descarregamento dos gráficos apresentados em vários formatos;
- Apresentação de indicadores estatísticos.

Os requisitos não funcionais são os seguintes:

- Tratamento dos dados recolhidos, de acordo com o seu tipo - transversal ou longitudinal;
- Sistema de autenticação, com devido armazenamento das senhas;
- Disponibilização da solução através de um *browser web*.

1.3 Solução desenvolvida

Nesta secção resume-se a arquitetura da solução desenvolvida. A solução proposta, apresentada na Figura 1.1 consiste num servidor web que fornece ao utilizador o *dashboard* que permite a interação do utilizador, um servidor aplicacional que realiza a lógica e comunicação entre o utilizador e a base de dados relacional, tendo-se seguido assim uma arquitetura de três camadas [27].

De forma a realizar a importação de novos dados, é necessário que estes passem pelo processo de *Extract, Transform, and Load* (ETL). Para tal, existe um serviço disponível que realiza esse procedimento, sendo que a comunicação entre o servidor aplicacional e o ETL é assíncrona através de uma fila de mensagens, uma vez que este processo poderá demorar horas dependendo da quantidade de dados. De forma a disponibilizar os dados recebidos no servidor aplicacional no ETL é utilizada uma base de dados não relacional (NOSQL).

A base de dados relacional (SQL) é utilizada para armazenar os dados permanentes da solução. A base de dados não relacional é utilizada principalmente para enviar

ficheiros do servidor aplicacional para o serviço ETL, sendo que também pode ser utilizada para saber o estado dos trabalhos em processamento.

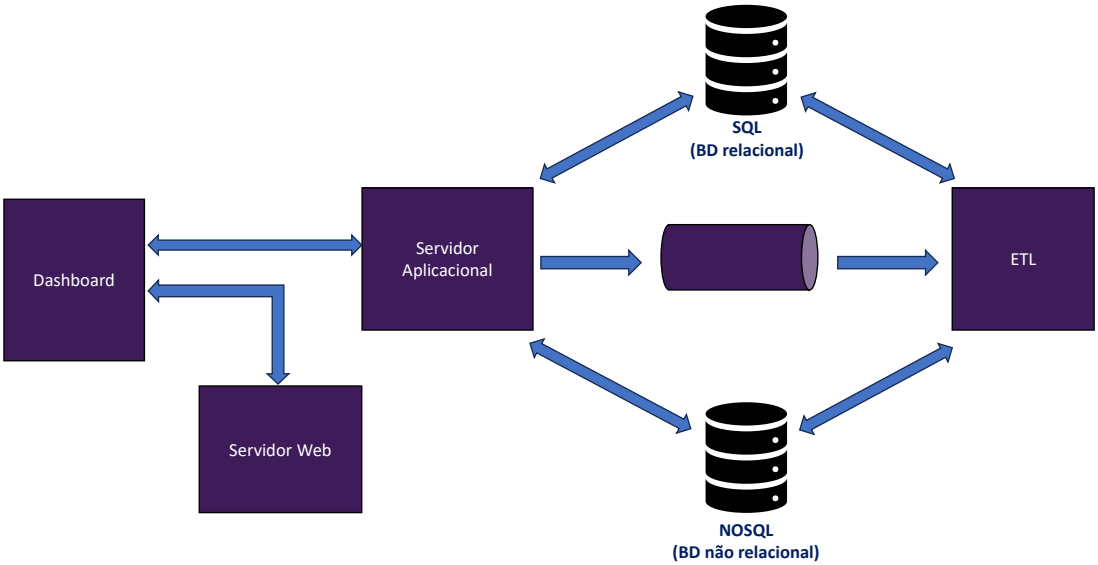


Figura 1.1: Diagrama de arquitetura da solução desenvolvida

1.4 Contribuições

Deste trabalho, resultaram os seguintes artigos científicos:

Rúben Dias, Artur Ferreira, Iola Pinto, Carlos Geraldes, Luís Bento, Cristiana Rekowski. "An interative dashboard for statistical analysis of COVID-19 data for PREMO project", Portuguese Conference on Pattern Recognition (RECPAD), Coimbra, Portugal, outubro de 2023.

Rúben Dias, Artur Ferreira, Iola Pinto, Carlos Geraldes, Cristiana Von Rekowski, Luís Bento. "An interactive dashboard for statistical analysis of Intensive Care Unit COVID-19 data", Ressubmetido após revisões, ao MDPI - BiomedInformatics, 28 novembro 2023.

O código encontra-se disponível em: `https://github.com/RubenDays/premo-dashboard-public`.

1.5 Organização do documento

O presente documento é constituído por seis capítulos, para além do Capítulo 1 que corresponde à introdução. É realizada uma descrição do trabalho relacionado e do estado de arte no Capítulo 2, tendo em conta o trabalho já realizado por [55]. No Capítulo 3 são descritos os passos para a realização do tratamento de dados,

que conduziram à criação de várias tabelas e vistas numa base de dados. No Capítulo 4 é apresentado o desenvolvimento do *dashboard*, descrevendo a arquitetura, as linguagens e ferramentas utilizadas, bem como os procedimentos inerentes à realização da solução. Com a solução criada, passou-se à fase da implantação, que se encontra descrita no Capítulo 5.

Os resultados obtidos, na forma de representações gráficas, diagramas ou estatísticas produzidas pela solução encontram-se no Capítulo 6, assim como as páginas relativas às restantes funcionalidades.

Por último, no Capítulo 7, resume-se o trabalho desenvolvido, apresentam-se as conclusões e as indicações para trabalho futuro.

O desenvolvimento deste trabalho de projeto contou também com a colaboração do Doutor Luís Bento, do centro Hospitalar de Lisboa Central.

2

Trabalho relacionado e estado de arte

Neste capítulo apresenta-se uma descrição de alguns trabalhos relacionados que se encontram na literatura. Na Secção 2.1 é realizada uma descrição dos trabalhos prévios no âmbito do projeto PREMO, os quais tiveram continuidade no presente trabalho. Na Secção 2.2 realiza-se uma pesquisa sobre *dashboards* dentro do contexto de dados biomédicos de COVID-19 e na Secção 2.3 refere-se uma pesquisa sobre *dashboards* fora do contexto de COVID-19.

2.1 Trabalho anterior

Nesta secção descrevem-se as diversas etapas do trabalho realizado relacionado com o tema desta tese. Concretamente define-se o enquadramento do presente estudo no âmbito do projeto PREMO.

O estudo realizado por [53], teve como objetivo a identificação de biomarcadores preditivos para desfechos severos em contexto da COVID-19, nomeadamente para a morte, em paciente críticos, internados na UCI. Para tal, foram analisados dados clínicos, demográficos e laboratoriais recolhidos ao longo do período de internamento dos pacientes na UCI.

Para inferir sobre a evolução do quadro clínico dos pacientes na primeira semana de internamento na UCI, foi realizada uma análise comparativa entre os dados do segundo e sétimo dia. Foram realizadas comparações da distribuição estatística dos parâmetros laboratoriais de interesse, entre o grupo de pacientes falecidos e o grupo de pacientes que obtiveram alta. Realizaram-se ainda estudos de análise de

sobrevivência, tendo-se estimado a curva de sobrevivência de Kaplan Meier para diferentes grupos de pacientes, caracterizados por evoluções distintas, ao longo do internamento na UCI.

A análise das associações entre os vários biomarcadores e a morte foram testadas com recurso a modelos univariados adequados a dados longitudinais - *Generalized Estimating Equation*s (GEEs).

No estudo realizado em [53], concluiu-se que os pacientes que faleceram eram consideravelmente mais velhos, possuíam maior número de comorbidades, necessitavam mais de VMI e passavam menos tempo no hospital em relação aos pacientes que receberam alta.

Por forma a dar suporte informático ao referido estudo, foi desenvolvido um trabalho [55] no qual são processados os dados das três primeiras vagas de COVID-19, em hospitais de Lisboa. O objetivo do trabalho, reportado em [55], consistiu na criação de uma base de dados relacional com os Dados Laboratoriais (DL) dos pacientes internados na UCI, utilizando para tal *scripts* em linguagem Python. Esta base de dados foi criada de forma dinâmica, permitindo uma adaptação rápida conforme a quantidade de biomarcadores diferentes que possam surgir.

O trabalho [55] apenas realiza a transformação dos dados, para serem colocados numa base de dados relacional não havendo *dashboard* associado ao mesmo. Os dados são longitudinais, encontram-se organizados em ficheiros Excel, um ficheiro por paciente, onde cada linha representa uma medição de biomarcador numa colheita desse paciente.

O trabalho anterior [55], realiza a passagem destes ficheiros Excel para a base de dados. Percorrem-se todos os ficheiros, extraindo os dados, excluindo algumas colunas que não são relevantes para o contexto, ou que tenham os valores todos a nulo. Também é realizado algum tratamento, como por exemplo a eliminação de tabulações ou espaços em branco em excesso nas *strings*. São removidos tuplos duplicados e no caso de existirem tuplos idênticos onde apenas o resultado difere, é realizada uma concatenação dos resultados separados por um '+', de forma a não se perder informação que poderá ser relevante.

Este trabalho resultou na criação das seguintes tabelas:

- **COLHEITA**, que contém o paciente, uma data de colheita e o serviço que solicitou a colheita;
- **PARAMETRO**, que contém as informações respeitantes às diferentes variáveis (parâmetros), como o nome da análise, nome do parâmetro e valores de referência;
- **RESULTADO**, que podem ser várias tabelas esparsas, ou seja, têm um número elevado de valores omissos, consoante o número de parâmetros existentes. Estas contêm os resultados de cada paciente para cada parâmetro, em forma de *string*. Para referenciar a tabela PARAMETRO, é usado como nome da

coluna o identificador do parâmetro da tabela PARAMETRO. Cada tabela de RESULTADO tem até 253 parâmetros.

O modelo Entidade-Relação (ER) desenvolvido em [55] encontra-se na Figura 2.1.

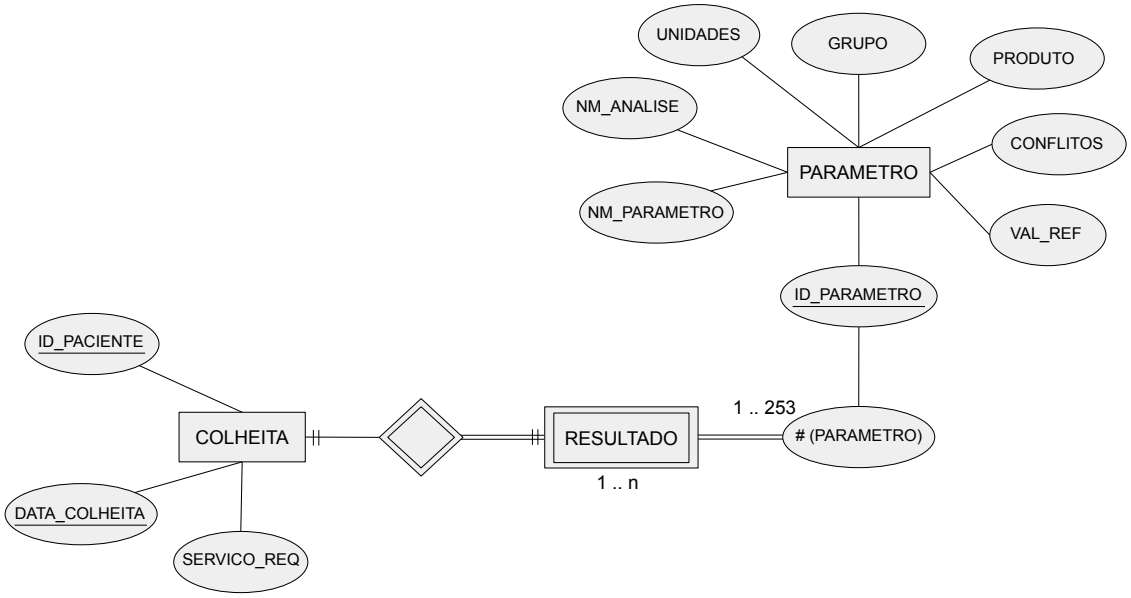


Figura 2.1: Modelo ER base [55]

2.2 *Dashboards* de dados de COVID-19

O surgimento da doença COVID-19, detetada pela primeira vez em humanos na cidade chinesa de Wuhan em 2019, provocou um período de pandemia. Este período incentivou o desenvolvimento de vários estudos científicos e projetos.

Dos projetos que consistiram no desenvolvimento de *dashboards*, identificam-se dois tipos diferentes. Um primeiro que ajuda a perceber onde é que houve maior número de infeções, número de recuperados, número de vacinações, número de óbitos, entre outros, de forma a acompanhar a propagação do vírus. Um exemplo deste tipo de *dashboard* foi criado pela *Johns Hopkins University Center for Systems Science and Engineering* [16] (Figura 2.2).

Foi também desenvolvido um pacote escrito na linguagem R que permite analisar e visualizar dados de COVID-19 [44], de forma a criar um *dashboard* deste tipo.

Figura 2.2: Exemplo de visualização do *dashboard* para acompanhar a propagação da COVID-19 criado pela *Johns Hopkins University Center for Systems Science and Engineering* [16]

Um outro exemplo, consiste num *dashboard* interativo, desenvolvido a partir da linguagem R, que permitiu acompanhar a COVID-19 nos Estados Unidos [67] (COVID-19 Watcher). Este aparenta estar descontinuado, dado que relativamente à data de escrita desta tese a última alteração no GitHub foi há três anos. Na Figura 2.3 é apresentada uma imagem deste *dashboard*. Também existem outros *dashboards*, como o exemplo de [4].

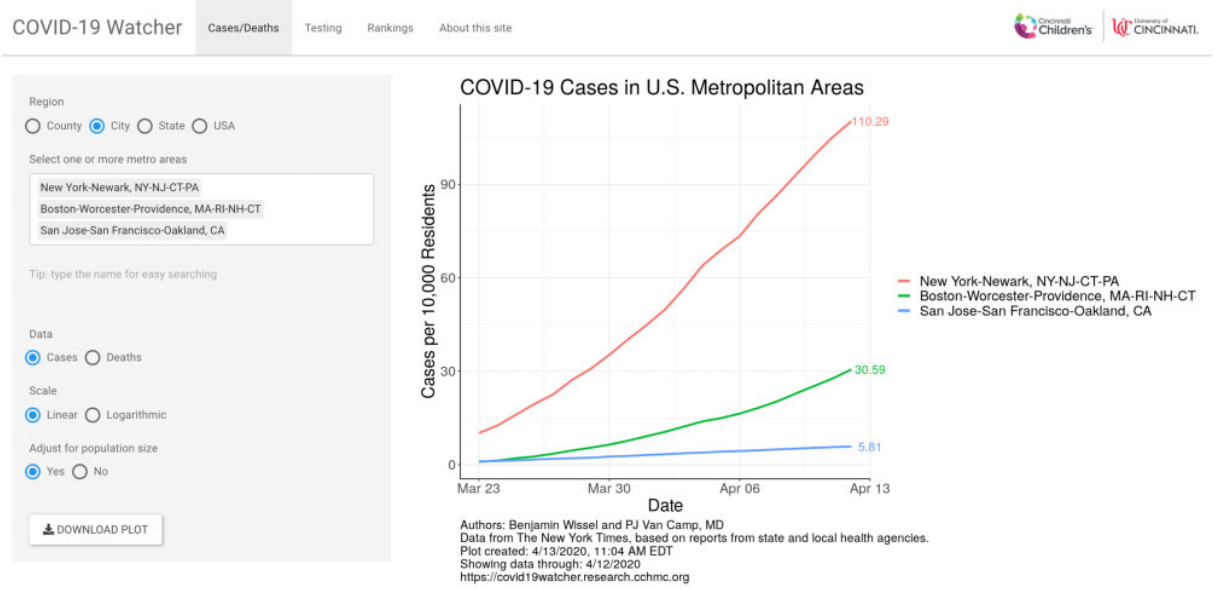


Figura 2.3: Imagem do *dashboard* do COVID-19 Watcher [67]

O segundo tipo de *dashboard* surge devido às situações onde os surtos súbitos e

prolongados afetaram os sistemas de saúde globalmente, o que gerou uma preocupação dos hospitais sobre como gerir da melhor forma o grande número de pacientes infetados e não infetados. Foi criado um *dashboard*, apresentado na Figura 2.4, suportado na aplicação *Microsoft Power BI* e *Cernel Computer Language*, para saber onde os pacientes se encontram internados e também o seu estado através de um esquema de cores. Esta visualização permite monitorizar os pacientes nas múltiplas enfermarias e intervir quando clinicamente necessário ou realocar os pacientes, resultando numa melhor gestão dos recursos [29].

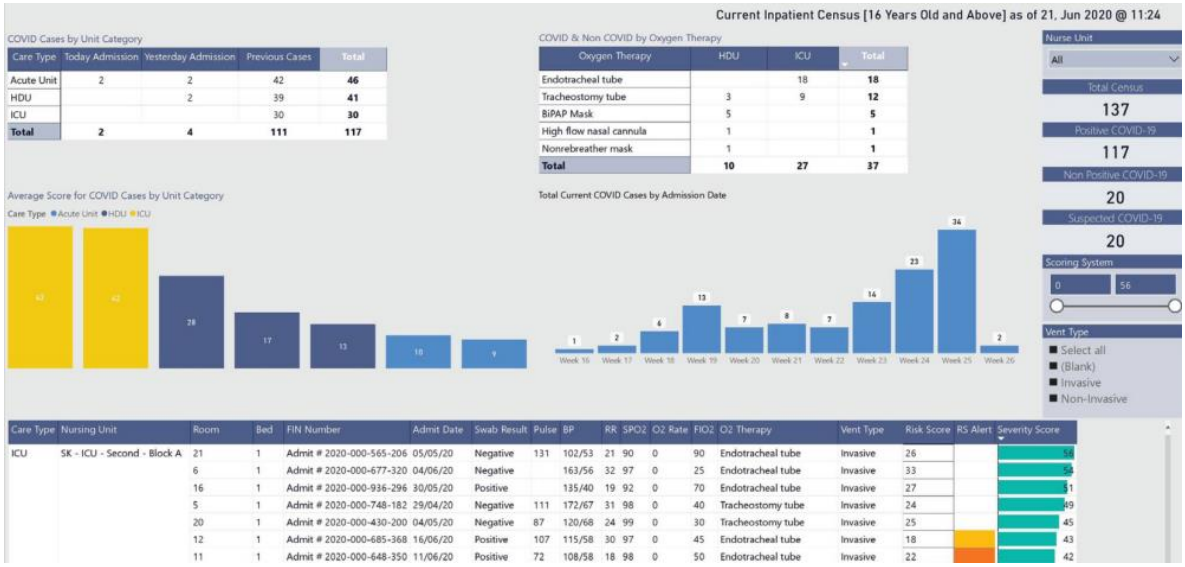


Figura 2.4: Exemplo de visualização do *dashboard* para gestão de recursos e pacientes [29]

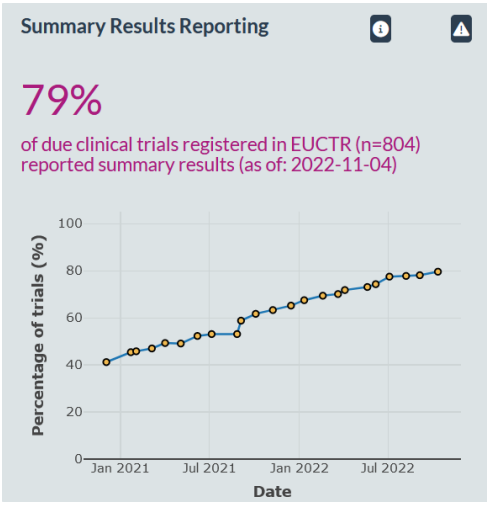
Com o mesmo objetivo, foi criado um outro *dashboard* [31], que permite monitorizar os equipamentos de ventilação (tipos de máquinas, modos de operação e pacientes nos ventiladores) e o estado dos pacientes (e.g. resultados de laboratório).

2.3 *Dashboards* para outro tipo de dados

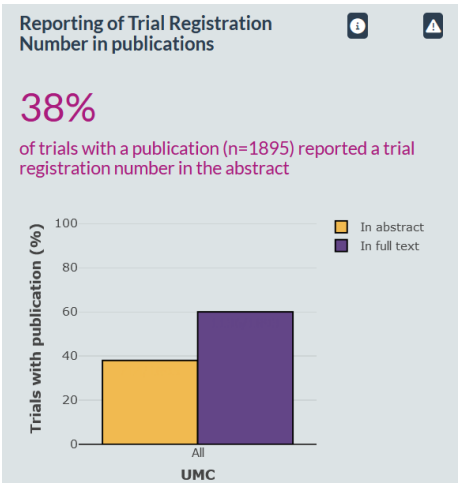
Visto existirem maioritariamente dois tipos diferentes de *dashboard* dedidados à doença COVID-19, e que nenhum desses tipos é exatamente o pretendido para o caso deste projeto, foi realizada uma pesquisa fora do âmbito do COVID-19.

Um exemplo de *dashboard* fora do âmbito do COVID-19, foi criado para analisar a transparência de ensaios clínicos para centros médicos universitários na Alemanha [19]. Os dados apresentados neste *dashboard* não são relevantes para o trabalho apresentado nesta tese, mas a disposição visual/organização do *dashboard* pareceu interessante. Neste *dashboard* são apresentados dados da evolução dos ensaios clínicos ao longo dos anos, como se pode observar (gráficos de linhas) na Figura 2.5a.

Também é possível analisar neste *dashboard* a quantidade de registo de ensaios com publicações em formato de *abstract versus* texto integral (gráficos de barras), conforme exemplificado na Figura 2.5b.



(a) Exemplo do gráfico de linhas



(b) Exemplo do gráfico de barras

Figura 2.5: Exemplo de gráficos no *dashboard* sobre transparência de ensaios clínicos [19]

Na Figura 2.6 é apresentada uma imagem de um outro exemplo de *dashboard*, criado em [39] usando a linguagem R, que permite a análise de dados relacionados com a higiene oral.

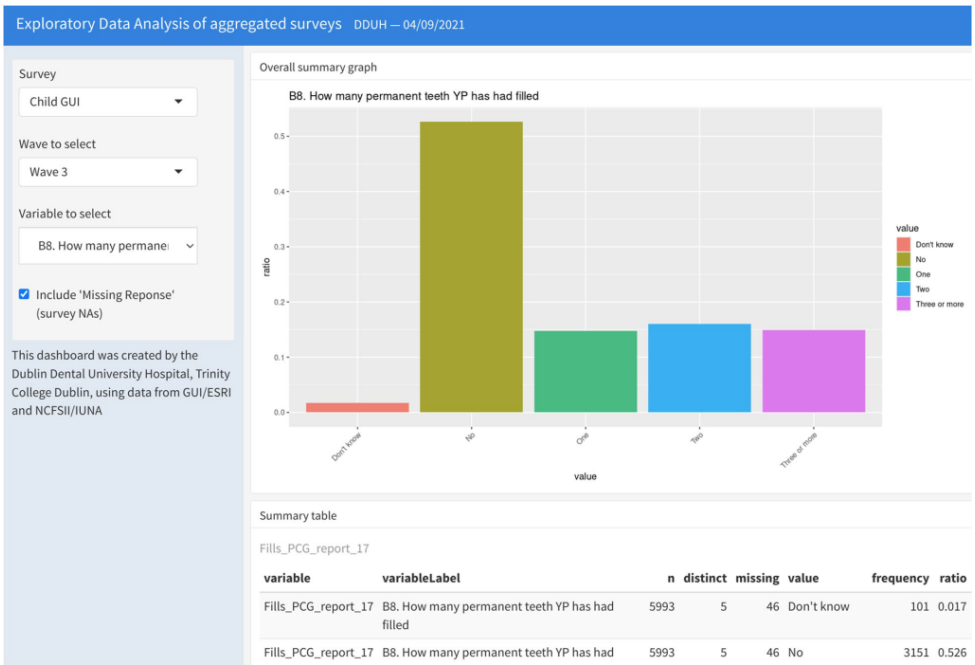


Figura 2.6: Exemplo de visualização do *dashboard* para análise de dados relacionados com a higiene oral [39]

Relacionado agora com a Malária, existe o *dashboard* criado por [34] usando a linguagem R que funciona em modo *offline*. Este *dashboard* permite analisar dados de parâmetros da Malária, em diferentes estados e distritos na Índia. Na Figura 2.7 é apresentada uma imagem do *dashboard*.

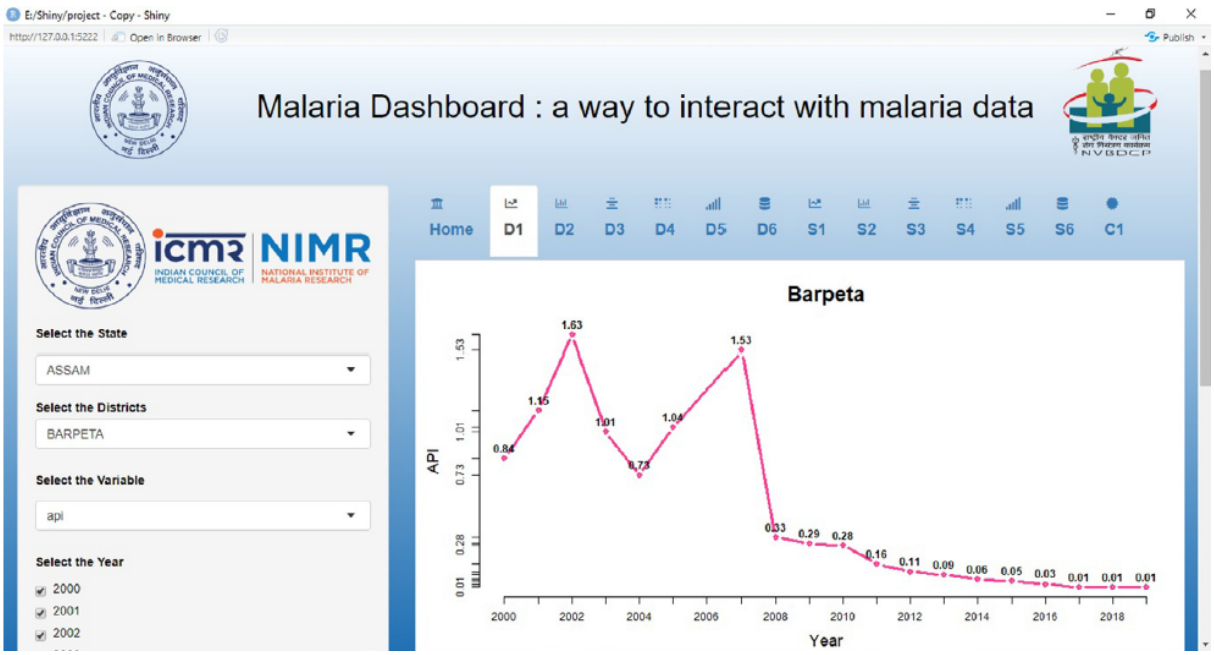


Figura 2.7: Exemplo de visualização do *dashboard* para análise de dados relacionados com a Malária [34]

3

Tratamento de dados

Neste capítulo descreve-se o procedimento para o pré-processamento e tratamento dos dados recebidos. Identificam-se dois tipos de dados: Dados Laboratoriais (DL) recolhidos em dois hospitais diferentes em Lisboa, e os Dados Clínicos e Demográficos (DCD) dos pacientes.

Para os DL, foi utilizada uma base já criada por [55], a qual se baseia em *scripts* em linguagem Python, para importar os dados em formato Excel para uma base dados em MariaDB.

Na Secção 3.1 é realizada a descrição dos dados originais (brutos) - DL e DCD. Na Secção 3.2 é apresentada a linguagem de programação utilizada. O processo ETL para os DL é apresentado na Secção 3.3 e também para os dados DCD na Secção 3.4.

Verificou-se a necessidade de proceder à agregação de diversos parâmetros, visto existirem vários que representam a mesma característica, no entanto, trazem associado valores de referência diferentes, por terem diferentes proveniências.

Quanto aos DCD, foi criado um modelo novo de forma a importar os mesmos para a base de dados, e com esses foi necessário ainda criar novas variáveis por forma a calcular outras informações que serão úteis ao projeto PREMO.

3.1 Descrição dos dados originais

Nesta secção é realizada a descrição dos dois tipos de dados brutos utilizados: os DL e os DCD. Para realizar este tipo de descrição foi utilizada a biblioteca 'pandas_profiling' que se encontra disponível no Python, a qual permite realizar esta descrição a partir de um 'DataFrame'.

Para o projeto atual foram considerados os dados de 1239 pacientes internados na UCI do Centro Hospitalar Lisboa Central, com datas de admissão entre 10 de março de 2020 e 23 de setembro de 2022. Relativamente às vagas de COVID-19, os dados a tratar correspondem às seis primeiras vagas de COVID-19 em Portugal.

Os DL, são longitudinais e representam os resultados das colheitas dos diversos pacientes ao longo do tempo de permanência na UCI. Existe um ficheiro Excel (.xls) para cada paciente, com formato idêntico ao exemplo apresentado na Figura 3.1. Cada linha neste ficheiro representa medição de um biomarcador de uma colheita para um paciente onde cada célula é um dado métrico mas considerada uma *string*.

DATA_COLHEITA	DATA_VALIDACAO	SERVICO_REQ	NM_ANALISE	GRUPO	NM_PARAMETRO	RESULTADO	UNIDADES	VAL_REF	PRODUTO
2020-10-30 23:01:00	2020-10-30 23:29:44	HSJ-UUM CI NIV.III - COVID	Hemograma	1	Gra IMAT #	1.21	x 10^9/L		SORO
2020-10-30 23:01:00	2020-10-30 23:29:44	HSJ-UUM CI NIV.III - COVID	Hemograma	1	Linf AT # / '	0.09	x 10^9/L		SORO
2020-10-30 23:01:00	2020-10-30 23:29:44	HSJ-UUM CI NIV.III - COVID	Hemograma	1	Gra IMAT %	3.50	%		SORO
2020-10-30 23:01:00	2020-10-30 23:29:44	HSJ-UUM CI NIV.III - COVID	Hemograma	1	Linf AT %	0.30	%		SORO
2020-10-30 23:01:00	2020-10-30 23:29:44	HSJ-UUM CI NIV.III - COVID	Hemograma	1	Alarmes do equipa	Neutrophilia Leukocytosis Monocytosis			SORO

Figura 3.1: Exemplo dos Dados Laboratoriais de um paciente, em bruto, contidos num ficheiro Excel

Para realizar uma descrição destes dados, sendo que são vários ficheiros, foi necessária a agregação destes numa só estrutura. Na Figura 3.2, é apresentada uma descrição sucinta dos DL, onde se pode observar que contém 18 colunas, 3 192 244 tuplos, 46.4% dos valores são omissos e que existem 2% de tuplos em duplicado.

Dataset statistics

Number of variables	18
Number of observations	3192244
Missing cells	26637990
Missing cells (%)	46.4%
Duplicate rows	63769
Duplicate rows (%)	2.0%

Variable types

DateTime	2
Text	9
Numeric	3
Unsupported	2
Categorical	2

Figura 3.2: Descrição dos DL

Na Tabela 3.1, descrevem-se as 18 colunas existentes, com o seu significado no contexto destes dados, e também a percentagem de valores omissos. Nesta tabela as colunas 'ISOLAMENTO', 'NM_MICRORGANISMO', 'NUM_BIOTIPO', 'ANTIBIOTICO', 'NM_ANTIBIOTICO', 'RESISTENCIA', 'NM_RESISTENCIA' E 'MIC' apresentam uma percentagem de valores omissos muito elevada (99.6% e superiores).

Tabela 3.1: Colunas presentes nos Dados Laboratoriais em bruto

Nome	**Significado**	Valores omissos (%)
DATA_COLHEITA	Refere-se à data de realização da colheita que originou o resultado laboratorial	0.0%
DATA_VALIDACAO	Data de validação da colheita	0.0%
SERVICO_REQ	Serviço que requisitou a análise	0.0%
NM_ANALISE	Refere-se ao nome da análise da referida colheita	0.0%
GRUPO	Código numérico para categorias que englobam as análises referidas em "NM_ANALISE"	0.0%
NM_PARAMETRO	Nome do parâmetro (biomarcador)	0.1%
RESULTADO	Resultado do biomarcador para a colheita	1.2%
UNIDADE	Unidades de medida do resultado	15.7%
VAL_REF	Valor de referência do parâmetro	19.7%
PRODUTO	Tipo de fluído biológico utilizado para realizar a análise	0.0%
ISOLAMENTO	Código utilizado para cada microrganismo, cujo nome é referido em "NM_MICRORGANISMO"	99.6%
NM_MICRORGANISMO	Nome do microrganismo isolado	99.6%
NUM_BIOTIPO*		> 99.9%
ANTIBIOTICO	Sigla para nome dos antibióticos a que cada microrganismo é resistente/sensível, referido em "NM_ANTIBIOTICO"	99.6%
NM_ANTIBIOTICO	Nome dos antibióticos a que cada microrganismo é sensível/resistente	99.6%
RESISTENCIA	Sigla para a resistência de um microrganismo de um determinado antibiótico (resistente/sensível - referido em "NM_RESISTENCIA")	99.6%
NM_RESISTENCIA	Resistência de microrganismo a um determinado antibiótico	99.6%
MIC*		> 99.9%

* Não foi possível identificar corretamente o significado das colunas marcadas.

De maneira a importar estes dados para a base de dados relacional, são utilizados os *scripts* realizados por [55], os quais consideram a eliminação das colunas com uma percentagem de valores omissos muito elevada.

Quanto aos DCD, que são dados transversais, encontram-se num só ficheiro Excel (.xls) onde cada linha representa os DCD de cada paciente. Na Tabela 3.2 é apresentado um resumo do significado de cada coluna, concretamente o nome da variável, o seu significado, a tipologia do dado estatístico apresentado no resultado, e o domínio de variação do resultado.

Tabela 3.2: Colunas presentes nos Dados Clínicos e Demográficos no ficheiro Excel

Nome	**Significado**	**Tipo**	**Domínio**
ID_PACIENTE	identificador do paciente	quantitativo discreto	>= 1
VAGA	vaga de COVID-19	qualitativo ordinal	
UUM	se pertence à unidade de urgência médica	qualitativo nominal - booleano	'0' (não pertence) e '1' (pertence)
IDADE	idade do paciente na data de colheita	quantitativo discreto	> 0
SEXO	género do paciente	qualitativo nominal - booleano	'0' (masculino) e '1' (feminino)
PAIS_ORIGEM	país de origem do paciente	qualitativo nominal	'1' (Portugal), '2' (África), '3' (Ásia), ´4' (Europa), '5' (América do Sul) e '6' (América do Norte)
D0_SINTOMAS	Data de início de sintomas	data	formato DD/MM/YYYY
D0_DIAGNOSTICO	Dia do primeiro diagnóstico	data	formato DD/MM/YYYY
DATA_SINT_DIAG	Menor data entre D0_SINTOMAS e D0_DIAGNOSTICO	data	formato DD/MM/YYYY
COVID	Indica se o paciente foi diagnosticado com COVID-19	qualitativo nominal - booleano	'0' (não) e '1' (sim)
MOTIVO_ADMISSAO	Motivo de admissão do paciente	qualitativo nominal	'1' (COVID), '2' (Cirúrgico urgente), '3' (EAM), '4' (AVC), '5' (Choque séptico), '6' (Alterações de ritmo cardíaco), '7' (S. Guillan-Barré), '8' (Insuf Renal), '9' (Alteração do estado de consciência), '10' (Outros)
VMI	Indica se utilizou VMI	qualitativo nominal - booleano	'0' (não) e '1' (sim)

DATA_INICIO_VMI	Data de início utilização de VMI	data	formato DD/MM/YYYY
DATA_FIM_VMI	Data de fim de utilização de VMI	data	formato DD/MM/YYYY
ECMO	Indica se o paciente utilizou ECMO	qualitativo nominal - booleano	'0' (não), '1' (sim)
DATA_INICIO_ECMO	Data de início utilização de ECMO	data	formato DD/MM/YYYY
DATA_FIM_ECMO	Data de fim de utilização de ECMO	data	formato DD/MM/YYYY
DATA_ADMISSAO_UCI	Data de início utilização de VMI	data	formato DD/MM/YYYY
DATA_ALTA_UCI	Data de fim de utilização de VMI	data	formato DD/MM/YYYY
DATA_ADMISSAO_ HOSPITAL	Data de início utilização de ECMO	data	formato DD/MM/YYYY
DATA_ALTA_HOSPITAL	Data de fim de utilização de ECMO	data	formato DD/MM/YYYY
OBITO_UCI	Indicação se o paciente faleceu na UCI	qualitativo nominal - booleano	'0' (não), '1' (sim)
OBITO_HOSPITAL	Indicação se o paciente faleceu no hospital	qualitativo nominal - booleano	'0' (não), '1' (sim)
VACINA	Indica foi administrada alguma vacina ao paciente	qualitativo nominal - booleano	'0' (não), '1' (sim)
VACINA_MARCA	Indicação da vacina administrada ao paciente	qualitativo nominal	'1' (Pfizer), '2' (Astra Zenca), '3' (Moderna), '4' (Janssen)
*	Indicação se possui ou não um conjunto de comorbidades	qualitativo nominal - booleano	'0' (não), '1' (sim)

* Existe uma coluna para cada uma das 28 comorbidades, sendo muito extenso para colocar numa tabela. O tipo e domínio são idênticos para todas.

Na Figura 3.3, é apresentada a descrição dos dados. Nesta observa-se que existem 52 colunas das quais 5 são numéricas, 37 categóricas e 10 datas. Também é indicada a quantidade de valores omissos (8% das células) e a quantidade de tuplos duplicados (0%).

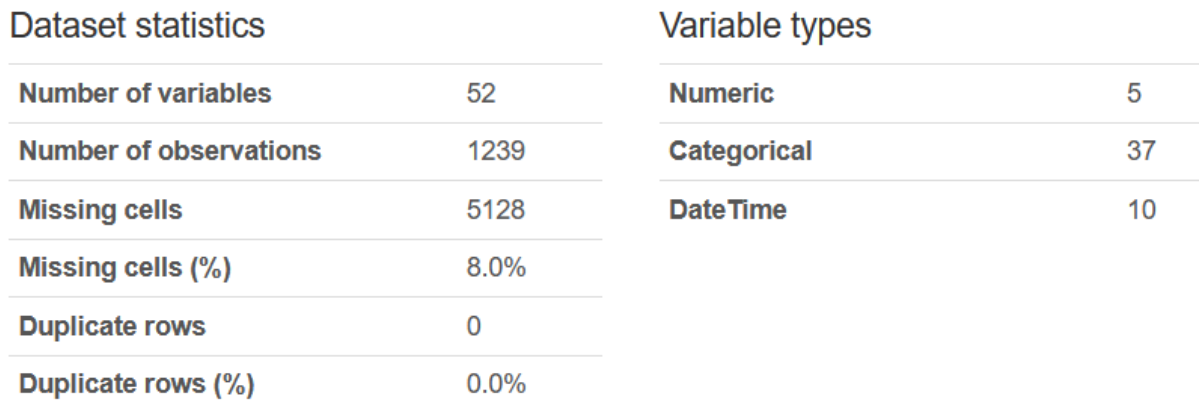


Figura 3.3: Descrição dos DCD

Na Tabela 3.3 é apresentado um resumo dos valores omissos para as várias colunas. Apenas existem datas para VMI e *Extracorporeal Membrane Oxygenation* (ECMO) se realmente o paciente teve essa necessidade. Pela tabela em questão, verifica-se que 33.1% dos pacientes não teve a necessidade de VMI, e 91.9% não teve a necessidade de ECMO. No caso da vacina, existem duas colunas, VACINA e VACINA_MARCA, onde se o valor em VACINA for '0' implica que na coluna VACINA_MARCA irá haver um valor omisso, dado que nas primeiras três vagas de COVID-19 não havia vacinas.

Tabela 3.3: Resumo de valores omissos dos Dados Clínicos e Demográficos

Nome	**Valores omissos**	**Valores omissos (%)**
D0_SINTOMAS	449	36.2%
D0_DIAGNOSTICO	199	16.1%
DATA_SINT_DIAG	199	16.1%
DATA_INICIO_VMI	410	33.1%
DATA_FIM_VMI	410	33.1%
DATA_INICIO_ECMO	1139	91.9%
DATA_FIM_ECMO	1139	91.9%
VACINA_MARCA	988	79.7%
DELIRIO	195	15.7%

3.2 Ferramentas e linguagens

Estas fases de ETL apresentam três características principais. A primeira é o facto de existirem várias tabelas RESULTADO que não se sabe à partida a quantidade, e é necessário utilizar todas. Outro aspeto é a necessidade de utilizar os identificadores da tabela PARAMETRO para indicar quais colunas de resultado se pretende.

Por último, é necessário ter alguma forma de conseguir manualmente correr estes fluxos, como por exemplo através de um pedido HTTP enquanto a aplicação se encontra em execução.

Foram experimentadas ferramentas ETL pagas como o 'Tableau', e grátis como o 'Pentaho'. Ambas as ferramentas apresentam diferentes limitações. Começando com o 'Tableau', é uma ferramenta paga, contudo há a possibilidade de possuir uma licença gratuita de estudante mas que apenas é valida por um ano, e este projeto encontra-se a ser trabalhado por período superior. Esta ferramenta tem forma de possibilitar trabalhar com o número dinâmico das várias tabelas de RESULTADO. Contudo não foi possível a tradução dos valores da tabela PARAMETRO para as colunas dos resultados. Por fim, relativamente à execução externa de fluxos, há a possibilidade mas é necessária uma versão paga.

Analisando agora a ferramenta grátis 'Pentaho', é uma ferramenta obviamente inferior ao 'Tableau', e que não satisfaz qualquer das características das fases de ETL.

Contudo, estas características são ultrapassadas utilizando uma linguagem de programação. Apesar do 'Tableau' permitir a utilização de *scripts*, continua a ter a limitação de ser paga. O 'Pentaho' também permite, mas iria resultar em diversos *scripts* para estas particularidades.

Assim, de maneira a não ter o ETL separado em duas vertentes (*scripts* e ferramenta), optou-se em apenas utilizar *scripts*, mais em concreto, através da linguagem Python.

3.3 Tratamento de dados laboratoriais

Nesta secção, descrevem-se as diferentes fases de processamento de ETL, conforme apresentado na Figura 3.4. A primeira fase, descrita na Subsecção 3.3.1 consiste na preparação dos dados, onde é realizada a importação dos mesmos para a base de dados, e também uma filtragem, de acordo com as indicações do Doutor Luís Bento do Centro Hospitalar de Lisboa Central. Após esta filtragem é realizada uma descrição dos dados resultantes.

Na segunda fase, é então realizada a transformação destes dados e também uma categorização, descrita na Subsecção 3.3.2. Realiza-se novamente uma descrição dos dados resultantes, de modo a ter uma ideia de mais alto nível do impacto da respetiva transformação.

Por último, na Subsecção 3.3.3 descreve-se a terceira fase onde é realizada a agregação dos valores e, novamente, realizada uma descrição dos dados de modo a clarificar o impacto da agregação nos dados resultantes.

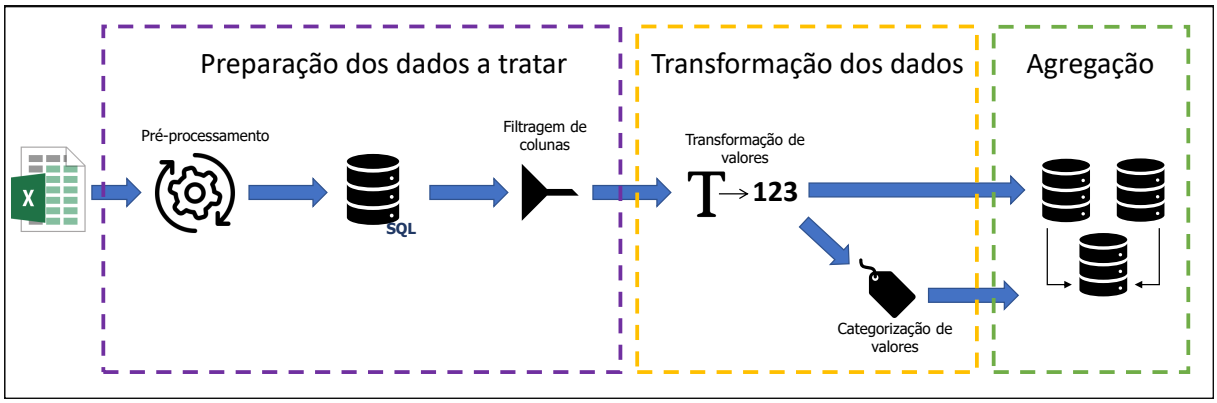


Figura 3.4: Diagrama das fases de ETL

3.3.1 Preparação

A Figura 3.5 descreve a primeira fase do processo ETL, que se inicia com o pré-processamento dos dados e consequente inserção na base de dados, segue-se um processo de filtragem dos dados de modo a utilizar apenas um subconjunto destes, de acordo com indicação da direção clínica do projeto PREMO.

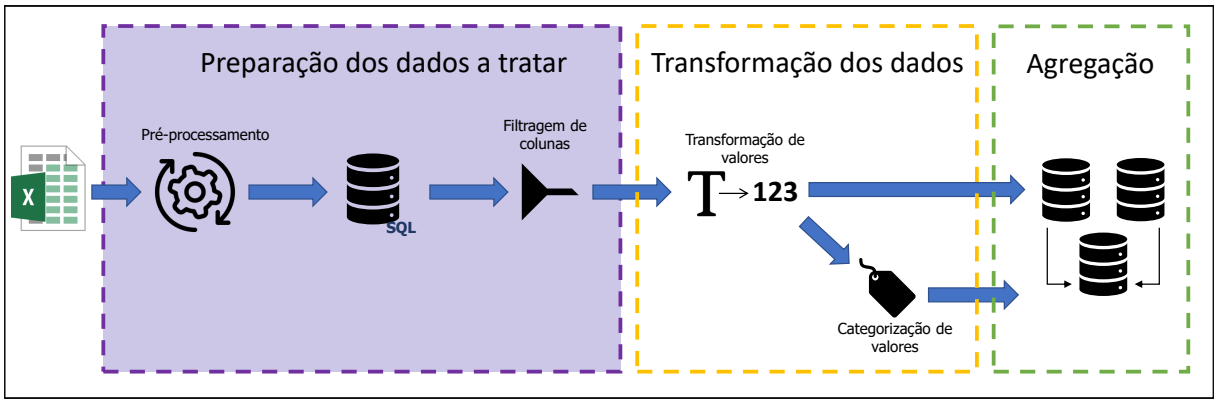


Figura 3.5: Primeira fase de ETL

A etapa de pré-processamento e consequente inserção na base de dados dos DL, é realizada com recurso aos *scripts* criados por [55]. Nesta etapa, já é realizado algum tratamento aos dados antes destes serem inseridos na base de dados, tais como a remoção de espaços em branco em excesso, tuplos duplicados e a remoção das colunas que não são relevantes para o contexto, indicadas na Secção 3.1.

Esta passagem para a base de dados resultou na criação de nove tabelas de RESULTADO, onde nas primeiras oito existem 253 parâmetros e na última existem 55 parâmetros, que totalizam em 2079 parâmetros. Note-se que o número de parâmetros pode aumentar com a inserção de mais dados que possam ocorrer em atualizações futuras.

De modo a viabilizar a manipulação isolada de um subconjunto destes parâmetros, decidiu-se criar uma vista, **'v_filtered_data'**, implementada através de um *script* em Python, **'f01_0_create_filtered_view'**, que cria a vista, dados os nomes dos parâmetros (NM_PARAMETRO) e os nomes das análises (NM_ANALISE) no referido subconjunto.

Este *script*, realiza uma interrogação na base de dados de modo a obter os diferentes identificadores dos respetivos parâmetros, com base nos seus nomes de parâmetro e de análise. Com estes identificadores, é possível calcular em que tabela 'RESULTADO' cada um se encontra, visto que, cada uma dessas tabelas possui 253 parâmetros ordenados. De modo a incluir todas as colheitas de todos os pacientes é ainda utilizada a tabela 'COLHEITA'.

Por indicação do clínico, existe uma lista de variáveis (211 parâmetros) classificados por nível de prioridade relativa à urgência do seu tratamento no âmbito do presente trabalho. Nesse documento os parâmetros encontram-se classificados em quatro diferentes níveis de prioridade, sendo '1' o nível de maior prioridade e 4 o nível de menor prioridade. Estipulou-se tratar os 43 parâmetros com o nível mais elevado de prioridade.

Com a análise deste documento, constata-se que os parâmetros contêm um formato 'NOME ANALISE_NOME PARAMETRO' (e.g. Hemograma_Plaquetas), conforme apresentado na Figura 3.6. Deste modo decidiu-se colocá-los num documento de texto, verificar os seus nomes na base de dados, e corrigir eventuais erros no ficheiro de texto, também eliminando quaisquer caracteres com acentos ou com eventuais caracteres, como por exemplo, 'ç'.

Prioridade	Variável	Valores de Referênc	Patológico
1	AlaninaaminotransferaseALT_AlaninaaminotransferaseALT	<41	Aumento
1	Antitrombina_Antitrombina	83,0 - 128,0	Aumento
1	APTTTTrombParcialActivada_Razao	0,8 - 1,2	Aumento
1	AspartatoaminotransferaseAST_AspartatoaminotransferaseAST	<40,0	Aumento
1	Bilirrubinadirecta_Bilirrubinadirecta	<0,5	Aumento
1	Bilirrubinatotal_Bilirrubinatotal	<1,4	Aumento
1	Calcio_Calcio	8,8 - 10,2	Aumento

Figura 3.6: Exemplo do ficheiro Excel com listagem de parâmetros prioritários

O *script* recebe o ficheiro de texto com os parâmetros pretendidos. Este encontra-se com o formato 'nome análise|nome parâmetro', onde os nomes das variáveis já foram corrigidos para ficarem de acordo com os nomes (NM_ANALISE e NM_PARAMETRO) que constam na base de dados, conforme apresentado no exemplo na Listagem 3.1. Desta maneira, é possível incluir mais parâmetros ou retirar, apenas modificando o ficheiro de texto sem a necessidade de alteração de código.

```
1 Hemograma|Plaquetas
2 Ionograma|Cloro
3 Ionograma|Potassio
4 Ionograma|Sodio
```

```
5 Mioglobina|Mioglobina
6 NT– proBNP|NT– proBNP
```

Listagem 3.1: Exemplo de conteúdo do ficheiro de texto que contém os parâmetros a serem tratados

De modo a obter mais informações gerais da vista criada, foi realizada a descrição dos dados da mesma.

Na Figura 3.7, pode-se observar o resultado e conforme indicado, existem 107 colunas, onde 20 delas são do tipo numérico, 1 delas tem o formato de data, 84 são textuais e 2 categóricas. Também é possível observar que existem 113 671 linhas, e que 91.1% das células existentes não têm valor. Por último, pode-se constatar que não existem tuplos duplicados.

De notar que nesta primeira fase do processo de ETL, há muitos valores omissos, uma vez que cada linha passa a ter vários resultados para uma colheita em vez de um resultado por linha. O que se sucede é que para uma dada colheita de um tipo só existem certos parâmetros que irão ter resultados, os restantes não terão porque dizem respeito a outro tipo de colheita.

Dataset statistics

Number of variables	107
Number of observations	113671
Missing cells	11084768
Missing cells (%)	91.1%
Duplicate rows	0
Duplicate rows (%)	0.0%

Variable types

Numeric	20
DateTime	1
Text	84
Categorical	2

Figura 3.7: Descrição dos dados na vista ‘v_filtered_data’

3.3.2 Transformação

Nesta subsecção, é descrita a segunda fase do processo de ETL (Figura 3.8). Esta é composta por duas etapas, onde na primeira é realizada a transformação dos dados, e na segunda fase é descrito o processo de categorização dos resultados das colheitas, consoante o seu valor de referência.

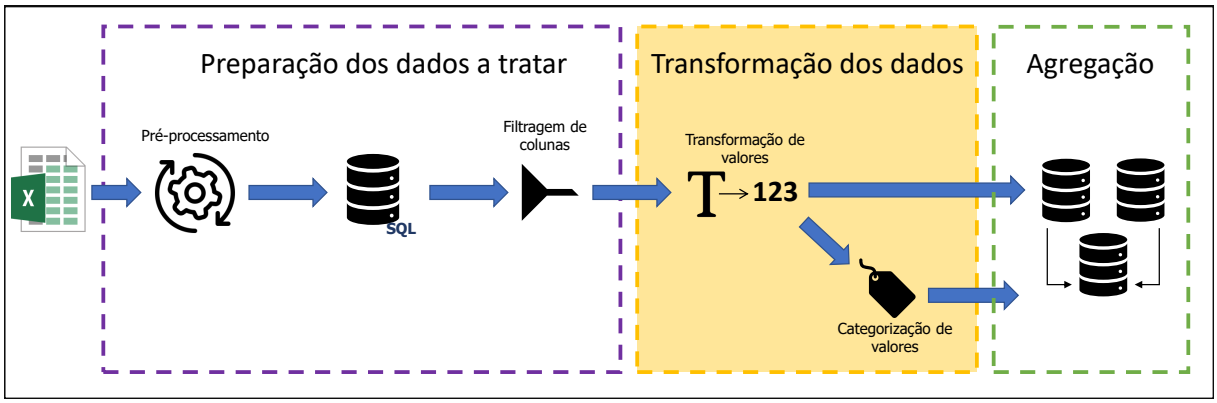


Figura 3.8: Segunda fase de ETL

Para analisar os tipos de incoerências (valores que não são possíveis de converter para um valor numérico, e.g. o valor '> 320.0') existentes na vista criada na Subsecção 3.3.1, realizou-se o *script*, **'filtered_measures'**, também usando a linguagem Python.

Este *script* obtém os dados que constam na vista 'v_filtered_data', e extrai os valores incoerentes para cada coluna correspondente a um parâmetro, bem como calcula a frequência absoluta de cada valor incoerente encontrado.

Com a informação obtida na execução deste *script*, constata-se a existência de vários valores incoerentes, como por exemplo os casos seguintes:

- com símbolos **'>'** e **'<'** que representam valores que as máquinas de colheita não conseguiram identificar o valor correto, visto apresentarem valores muito altos ou baixos, respetivamente;
- com o símbolo **'+'**, que foram valores criados na inserção dos ficheiros Excel para a base de dados relacional. Estes acontecem quando há presença de valores diferentes para a mesma colheita, do mesmo paciente na mesma data. Estes valores foram assim gerados de modo a não perder informação. Estes conflitos acontecem quando são feitas duas colheitas no mesmo espaço de tempo, como forma de validação/confirmação dos resultados;
- textuais.

Definiram-se as seguintes situações de processamento para estes valores incoerentes: i) nos valores que possuem o símbolo '**<**' ou '**>**', em geral, lê-se apenas o valor, ignorando o símbolo. No caso de se tratar de uma análise de 'hemograma', cujo nome de parâmetro é 'plaquetas', é considerado nulo; ii) nos que possuem o valor '**+**', é realizada uma média; iii) valores textuais passam a ser nulos.

Estando identificados os diferentes tipos de valores incoerentes, e de que modo se devem transformar, é então realizada a fase de transformação dos valores. Para tal, procedeu-se à criação do *script*, **'f01_transform_values'** em Python. Usou-se também um ficheiro, **'rules'**, em *JavaScript Object Notation* (JSON), onde se

encontram as regras de transformação para os diferentes parâmetros, indicados com o seu nome de análise e nome de parâmetro. A execução resulta na criação de uma nova tabela, **TRANSFORMED_FILTERED_VALUES**.

Este ficheiro JSON tem o formato conforme a Listagem 3.2. Este encontra-se dividido em dois objetos:

- **'hemograma|plaquetas'**, que representa as regras para o nome de parâmetro 'plaquetas' para a análise 'hemograma';
- **'all'**, representa as regras para os parâmetros não definidos neste documento.

Dentro de cada um destes objetos, existe:

- uma lista, **'transformations'**, que contém objetos com as transformações a aplicar. Assim temos uma ordem de execução das transformações a aplicar, onde a ordem interessa na lista, mas não interessa no objeto. Estes objetos têm uma chave representada por um símbolo e o valor representa o que fazer na presença desse símbolo, como:
  - **'avg'**, que indica que é para realizar uma média do valor, onde o separador da *string* é a chave;
  - **'ignore'**, indicando que o carácter, indicado na chave, é para ignorar, mantendo o resto do valor;
  - **'null'**, coloca o valor a nulo.
- e um objeto **'result'**, que contém dois campos, que indica qual o tipo de valor após as transformações e o que fazer caso não seja:
  - **'type'** que indica o suposto tipo de valor;
  - **'ifnot'** indica o que fazer, caso não seja possível converter para o valor indicado em 'type'.

Pensou-se numa configuração assim, de modo a poder aplicar transformações sobre diversos parâmetros sem a necessidade de alteração do código.

```
{
    "hemograma|plaquetas": {
        "transformations": [
            {
                "+": "avg"
            }
        ],
        "result": {
            "type": "number",
            "ifnot": "null"
```

```
        }
    },
    "all": {
        "transformations": [
            {
                "+": "avg",
                "<": "ignore",
                ">": "ignore"
            }
        ],
        "result": {
            "type": "number",
            "ifnot": "null"
        }
    }
}
```

Listagem 3.2: Conteúdo do ficheiro 'rules.json'

O último passo da segunda fase do processo ETL, consiste na categorização de valores. Esta categorização produz uma leitura quanto aos valores dos biomarcadores apresentados pelos pacientes. Permite identificar os pacientes, com valores dentro e fora dos respetivos valores de referência. O resultado da categorização encontra-se presente na tabela **'FILTERED_CAT_VALUES'**.

Os valores de referência podem ser binários (e.g. '< 5.0'), ou na forma de intervalo (e.g. '0.0 - 0.6'), conforme apresentado na Figura 3.9. Assim, as regras a aplicar para a categorização são as seguintes:

- **1**, indica que está dentro do valor de referência;
- **2**, indica que está fora do valor de referência, onde este é um caso binário;
- **3**, indica que está abaixo do valor de referência, no caso deste ser um intervalo;
- **4**, indica que está acima do valor de referência, no caso deste ser um intervalo.

ID_PARAMETRO	NM_ANALISE	NM_PARAMETRO	VAL_REF
47	Hemograma	Eosinofilos #	0.0 - 0.6
57	Proteina C reactiva (PCR)	Proteina C reactiva (PCR)	< 5.0

Figura 3.9: Valores de referência para os parâmetros '47' e '57'

No caso da Figura 3.9, o parâmetro '47' tem um valor de referência intervalar '0.0 - 0.6', que significa que os valores considerados normais estão contidos nesse

intervalo. Para o parâmetro ‘57’, tem-se um valor de referência ‘< 5.0’, que traduz que os valores normais são abaixo de ‘5.0’.

Para realizar este procedimento, foi criado o *script*, **‘f02_categorize_values’**, para extrair os valores contidos na tabela ‘TRANSFORMED_FILTERED_VALUES’, e de seguida proceder à categorização. Assume-se a Figura 3.10 como um exemplo do conteúdo da referida tabela.

ID_PACIENTE	DATA_COLHEITA	47	57
1	2020-10-30 23:01:00	0.14	265.7
205	2021-03-19 08:26:00	0.71	4.4

Figura 3.10: Exemplo de resultados de colheitas para os parâmetros ‘47’ e ‘57’ para os pacientes ‘1’ e ‘205’

Juntamente com estes parâmetros também são extraídos os seus valores de referência da tabela ‘PARAMETRO’, como se pode observar na Figura 3.9.

Assim, tem-se uma variável com os resultados das colheitas, e uma outra variável com os valores de referência para cada parâmetro. É realizada a categorização dos valores de cada parâmetro de acordo com os valores de referência.

Para identificar a regra a aplicar, verifica-se o primeiro carácter do valor de referência, após eliminar espaços. Se este símbolo for **‘<’**, então sabe-se que os valores adequados estão abaixo desse. O mesmo acontece com o símbolo **‘>’**, sendo superior a um certo valor. Se não existirem estes símbolos, então assume-se que os valores de normalidade correspondem a um intervalo, que está separado por um **‘-’**.

Em seguida procede-se à comparação dos resultados das colheitas com esses valores de referência, de modo a categorizá-los, criando assim a tabela com as novas variáveis categóricas. Esta tabela tem a estrutura apresentada na Figura 3.11.

ID_PACIENTE	DATA_COLHEITA	47	57
1	2020-10-30 23:01:00	1	2
205	2021-03-19 08:26:00	4	1

Figura 3.11: Exemplo da categorização dos resultados de colheitas para os parâmetros ‘47’ e ‘57’ para os pacientes ‘1’ e ‘205’

Com os exemplos apresentados na Figura 3.9 e na Figura 3.10 e com os resultados na Figura 3.11, pode-se observar que o paciente identificado com ‘1’ tem um valor dentro do intervalo para o parâmetro ‘47’, ou seja, será categorizado como ‘1’. Para o caso do parâmetro ‘57’, pode-se observar que está fora do valor de referência, e este sendo binário, será categorizado como ‘2’.

Quanto ao paciente identificado com '205', tem um valor acima do intervalo para o valor de referência do parâmetro '47', logo será categorizado como '4'. Relativamente ao parâmetro '57' deste mesmo paciente, pode-se observar que tem um valor dentro do valor de referência, ou seja, será categorizado como '1'.

Após a criação da tabela 'TRANSFORMED_FILTERED_VALUES', que contém os dados após passarem pelo processo de transformação, é realizada novamente uma descrição dos dados, para perceber como estes foram afetados, como se pode observar na Figura 3.12. Nesta figura, temos que o número de linhas diminuiu, como esperado visto estes serem os tuplos apenas com valores omissos. Quanto às variáveis, manteve-se o número mas os tipos foram alterados, desta vez existem 89 numéricos, 1 data e 17 categóricos. Tal como pretendido não existem mais dados do tipo texto.

Estes valores categóricos são definidos assim pela biblioteca utilizada porque existe apenas um pequeno número de valores distintos, conforme apresentado na Figura 3.13 que corresponde a uma destas variáveis.

Comparando com a Figura 3.7, repara-se que o número de observações diminuiu, isto devido à remoção de tuplos totalmente nulos. A proporção de valores omissos também diminuiu mesmo tendo em consideração que alguns valores passaram a ser omissos nesta fase de transformação.

Dataset statistics

Number of variables	107
Number of observations	95780
Missing cells	9212521
Missing cells (%)	89.9%
Duplicate rows	0
Duplicate rows (%)	0.0%

Variable types

Numeric	89
DateTime	1
Categorical	17

Figura 3.12: Descrição dos dados na tabela 'TRANSFORMED_FILTERED_VALUES'

1868
Categorical

HIGH CORRELATION MISSING UNIFORM

Distinct	4
Distinct (%)	100.0%
Missing	95776
Missing (%)	> 99.9%

Figura 3.13: Descrição dos dados na tabela 'TRANSFORMED_FILTERED_VALUES' para o parâmetro '1868'

3.3.3 Agregação

Esta subsecção descreve a terceira fase do processo ETL (Figura 3.14). Esta consiste em duas estapas, uma primeira que realiza a agregação dos parâmetros, e a segunda descreve como é realizada a agregação dos respetivos valores. Por fim, é realizado uma descrição dos dados sobre a tabela resultante da agregação, de modo a ter uma visão mais clara do impacto desta agregação nos dados.

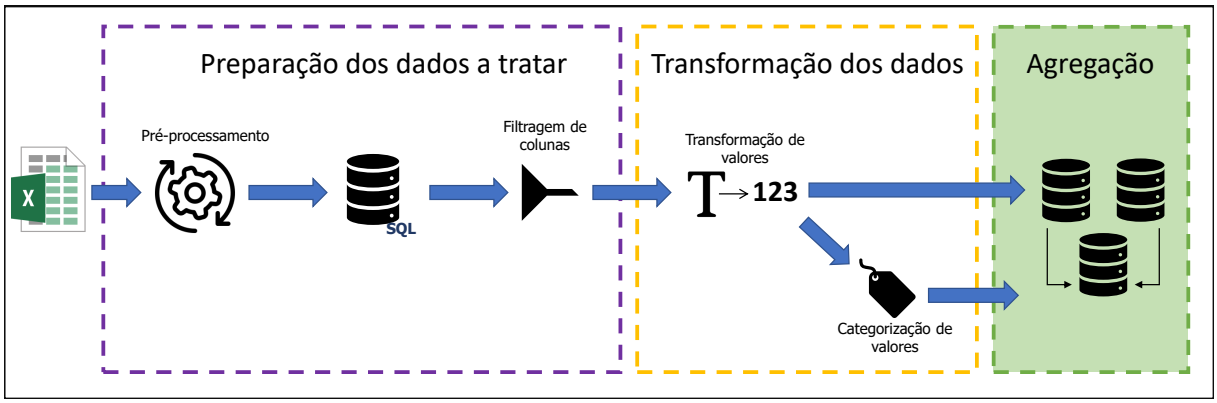


Figura 3.14: Terceira fase de ETL

Uma situação que se verifica na tabela 'PARAMETRO' é o facto de existirem vários parâmetros que contêm todos os valores iguais em todas as colunas exceto para o valor de referência. As colunas 'GRUPO' e 'CONFLITOS' podem ser diferentes, mas não são relevantes para o contexto, como se pode verificar no caso da 'Mioglobina' apresentado na Figura 3.15. Esta situação ocorre devido a este valor poder variar com o tempo, devido a calibrações, e às máquinas utilizadas. Assim, decidiu-se agrupar estes valores, por representarem o mesmo parâmetro.

+ Options

ID_PARAMETRO	NM_ANALISE	NM_PARAMETRO	PRODUTO	VAL_REF	UNIDADES	GRUPO	CONFLITOS
98	Mioglobina	Mioglobina	SORO	<154.9	ng/mL	5	0
383	Mioglobina	Mioglobina	SORO	<106.0	ng/mL	5	*NULL*

Figura 3.15: Parâmetros existentes de 'Mioglobina'

De modo a realizar a agregação dos parâmetros, foi necessário criar uma tabela que faça a ligação dos identificadores dos parâmetros para um novo que identifique apenas a análise e o nome do parâmetro, conforme apresentado num exemplo na Figura 3.16. Nesta figura, pode-se observar que os parâmetros com os identificadores (ID_PARAMETRO) '1251', '339' e '50' referem o mesmo parâmetro mas têm um valor de referência diferente, que resulta numa entrada diferente na tabela 'PARAMETRO'. Assim, estes terão o mesmo identificador quando agrupados (ID_MERGED). Os parâmetros com o identificador '47' e '57' são diferentes, portanto terão outros 'ID_MERGED'.

ID_PARAMETRO	NM_ANALISE	NM_PARAMETRO	VAL_REF	ID_MERGED
47	Hemograma	Eosinofilos #	0.0 - 0.6	10
50	Hemograma	Eritrocitos	4.4 - 5.9	11
57	Proteina C reactiva (PCR)	Proteina C reactiva (PCR)	< 5.0	12
339	Hemograma	Eritrocitos	3.8 - 5.0	11
1251	Hemograma	Eritrocitos	4.5 - 5.3	11

Figura 3.16: Exemplo de agregação de parâmetros

De maneira a realizar esta agregação por identificadores, foi desenvolvido um outro *script*, **'f03_create_merged_params'**. Este *script* utiliza a vista criada na Subsecção 3.3.1 ('v_filtered_data'), de modo a saber que parâmetros estão a ser utilizados, e a tabela 'PARAMETRO' para saber quais serão para agrupar pelo nome. Se os campos 'NOME_ANALISE' e 'NOME_PARAMETRO' forem idênticos então é atribuído o mesmo 'ID_MERGED'; se forem distintos é atribuído um diferente.

Com esta tabela e as tabelas criadas na Subsecção 3.3.2, é possível realizar a agregação dos valores dos vários parâmetros que referem o mesmo.

Para tal, pensou-se numa vista que junta os valores destes parâmetros, um vez que um paciente tendo um valor para um determinado parâmetro numa dada colheita, não vai ter outro resultado para um mesmo parâmetro cujo valor de referência seja diferente. As colheitas são referidas pelo identificador do paciente e pela data e hora em que foi efetuada, ou seja, teoricamente não haverá conflitos neste aspeto, pelo que é possível fazer a agregação.

Para realizar esta agregação foram criadas duas vistas, uma para os valores categóricos ('v_agreg_cat_values') e outra para os valores numéricos (v_agreg_num_values). Estas realizam a mesma execução, sendo que a única diferença é a tabela que utilizam. A de valores categóricos utiliza a tabela 'FILTERED_CAT_VALUES', e a de valores numéricos utiliza a tabela 'TRANSFORMED_FILTERED_VALUES', ambas criadas tal como descrito na Subsecção 3.3.2.

A interrogação para a realização da vista, consiste em duas seleções:

- a primeira, realiza as concatenações dos valores que representam o mesmo parâmetro, onde o valor nulo passa a ser uma *string* vazia;
- a segunda, realiza a transformação das *strings* vazias novamente para nulo, colocando o respetivo identificador agregado ('ID_MERGED').

Após esta agregação, foi realizada uma descrição dos dados sobre a vista criada, 'v_agreg_num_values', como apresentado na Figura 3.17. Nesta figura, pode-se observar que apenas contém 44 colunas, ao contrário do passo anterior (na transformação da Figura 3.12) que contém as 107 colunas. Nestas 44 colunas existem apenas variáveis numéricas e uma do tipo data, sendo que já não existem variáveis do tipo categórico visto terem sido agregadas, e assim contêm mais valores e são classificadas como numéricas. Também é possível observar que a percentagem de valores omissos baixou de 89.9%, da fase de transformação, para 75.4%.

Dataset statistics

Number of variables	44
Number of observations	95780
Missing cells	3178381
Missing cells (%)	75.4%
Duplicate rows	0
Duplicate rows (%)	0.0%

Variable types

Numeric	43
DateTime	1

Figura 3.17: Descrição dos dados para a vista 'v_agreg_num_values'

3.3.4 Obtenção de mínimos e máximos

A tomada de decisão do clínico, em contexto da UCI, baseia-se em meios de diagnóstico, nos quais se incluem o conjunto de resultados de análises laboratoriais e exames realizados ao longo do internamento. Para cada parâmetro laboratorial está identificado o intervalo de valores que pode assumir, assim como pontos de corte associados a padrões de risco relativos aos desfechos mais severos. Neste contexto, entendeu-se importante, em contacto com o Doutor Luís Bento, adicionar à base de dados, para cada parâmetro, a seguinte informação:

- valor máximo diário;
- valor mínimo diário;
- valor máximo diário registado no período da manhã;
- valor mínimo diário registado no período da manhã.

Para obter estas informações foi realizado um *script*, que gera uma interrogação na base de dados de forma a criar uma tabela com estas informações. Esta é criada usando as funções existentes no *Structured Query Language* (SQL) (MIN e MAX), e por serem operações relativamente pesadas, foi gerada uma tabela ('**PATIENT_MIN_MAX_RESULT**') ao invés de uma vista, de forma a que a pesquisa seja imediata e não seja necessário executar estas funções sempre que é realizada uma leitura.

Para realizar os valores para o período da manhã, é assumido apenas os resultados, cuja data de colheita esteja compreendida entre as 7h00 e as 11h00.

Foi também acordado realizar uma categorização dos valores mínimos e máximos. Sendo também fornecida informação de como categorizar os parâmetros "Plaquetas", "Bilirrubina total" e "Creatinina". Com esta informação foi criado um ficheiro JSON, como o apresentado na Listagem 3.3. Este contém listas referentes a cada parâmetro a categorizar, sendo que cada um destes contém um objeto com:

- um campo "**measure**", que indica que medida categorizar, sendo o "min" referente aos valores mínimos e "max" aos valores máximos;

- uma lista "**rules**", que contém objetos que indicam as regras de categorização para os valores dessa medida, sendo a chave a condição e o seu valor a categorização.

Decidiu-se realizar uma lista para cada parâmetro, visto assim dar a possibilidade de indicar mais medidas que poderão ser úteis de se categorizar, evitando alterações no código.

```
{
  "Hemograma|Plaquetas": [
    {
      "measure": "min",
      "rules": [
        { "< 20": "4" },
        { "< 50":  "3"},
        { "< 100":  "2"},
        { "< 150":  "1"},
        { ">= 150":  "0"}
      ]
    }
  ],
  ...
}
```

Listagem 3.3: Exemplo do ficheiro de configuração de categorização de valores mínimos e máximos

3.4 Tratamento dos dados clínicos e demográficos

A análise e modelação estatística dos dados com vista à obtenção de modelos preditivos dos eventos mais severos como a morte ou a necessidade de ventilação assistida, requer a utilização conjunta dos DL e dos DCD. Assim, foi necessário agregar as duas fontes de dados.

Para adicionar estas informações às existentes, foi fornecido um documento em Excel com as características clínicas e demográficas dos pacientes. Neste documento são apresentados os vários pacientes, um por linha, e as colunas dizem respeito às variáveis contidas na Tabela 3.2.

Para colocar estas informações na base de dados, foi necessário definir primeiro como seria o modelo ER para suportar estes dados.

Observando os dados, percebe-se que nem todos os pacientes usaram VMI ou ECMO (como apresentado na descrição destes dados na Secção 3.1), ou seja, poderão existir células vazias nestas colunas. Assim, pensou-se em criar uma tabela

para as datas de VMI e uma outra para a ECMO, de forma a não ter valores omissos na base de dados. O país de origem, a vacina e o motivo de admissão são enumerados e então também ficaram com a sua tabela. As comorbidades, que totalizam 28 colunas com valores binários, também foram colocadas numa tabela à parte. Decidiu-se criar a tabela 'PROCESSO' para os dados imutáveis do paciente, e a tabela 'EPISODIO' que se refere a uma admissão de um paciente na UCI. Assim, o modelo definido encontra-se na Figura 3.18.

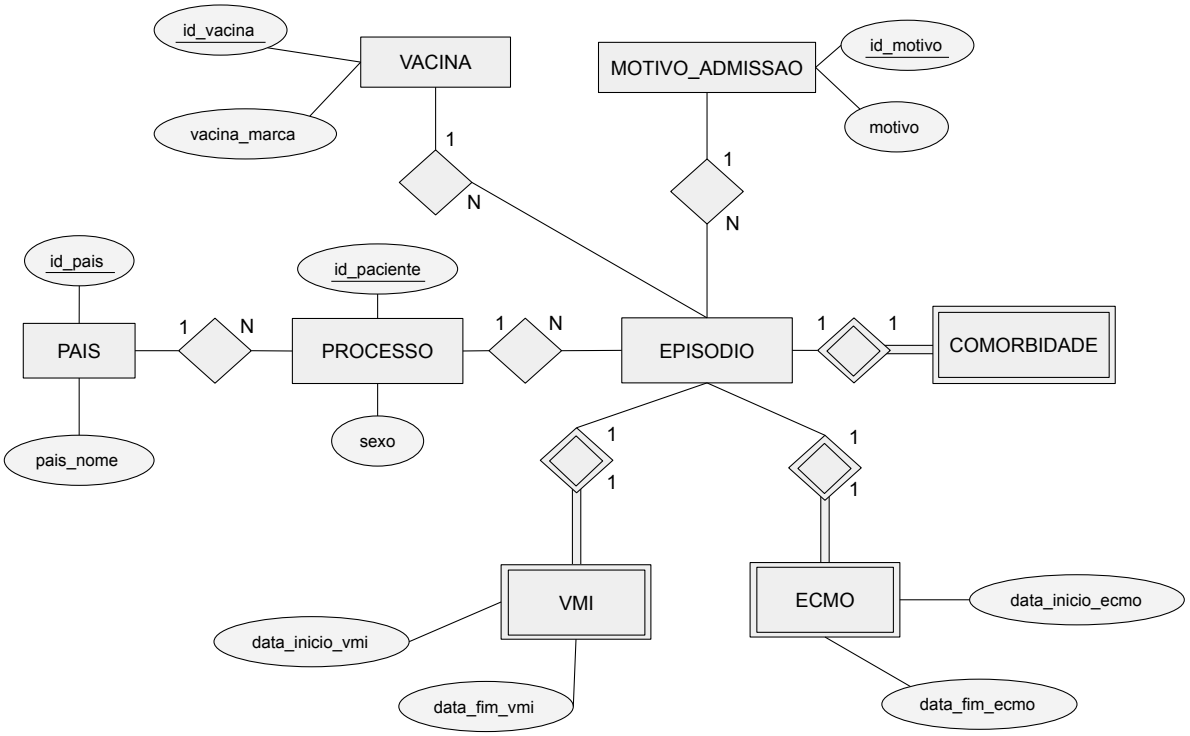


Figura 3.18: Modelo ER dos dados dos pacientes

Na Figura 3.18 não se encontram incluídos os atributos relativamente à tabela **EPISODIO** e à tabela **COMORBIDADE**, devido à sua extensão dificultar a visualização numa só imagem. Assim, a tabela EPISODIO contém:

- **id_episodio**, que é a chave primária da tabela,
- **id_paciente**, herdada de 'PROCESSO';
- **data_admissao_uci**;
- **idade**
- **vaga**;
- **covid**;
- **uum**;

- **data_sint_diag**;
- **data_alta_uci**;
- **data_alta_hospital**;
- **obito_uci**;
- **obito_hospital**;
- e por fim, as colunas que referem as tabelas **VACINA** e **MOTIVO_ADMISSAO**.

A tabela **COMORBIDADE**, por sua vez, contém: 1. delirio, 2. comorbilidades, 3. hta, 4. cardiopatia_isquemica, 5. icc, 6. diabetes, 7. dpco_asma_enfisema, 8. hipertensao_pulmonar, 9. insuficiencia_renal, 10. hbp, 11. dislipidemia, 12. hiperuricemia, 13. neoplasia_solida, 14. neoplasia_hematogica, 15. depressao, 16. amiloidose, 17. esclore_multipla, 18. obesidade, 19. hipotiroidismo, 20. sida, 21. disrritmia, 22. dhc, 23. avc, 24. transplantado 25. doenca_autoimune, 26. parkinson, 27. epilepsia, 28. esquizofrenia.

Para realizar a importação dos DCD contidos no ficheiro Excel para a base de dados, foi criado o *script* **f06_import_patient_data** em Python. De forma a ser mais poupado na utilização de espaço por parte da base de dados, foi decidido criar previamente as tabelas de forma a escolher quais os tipos das colunas, uma vez que as colunas não são dinâmicas. Quando uma tabela não se encontra criada, a biblioteca utilizada para realizar operações sobre a base de dados (pandas) cria-a com os maiores tipos de dados disponíveis, e.g. um número será do tipo BIGINT quando poderia ser do tipo INT, ou nos valores booleanos ser também do tipo BIGINT quando basta um TINYINT. Este último caso é mais relevante na tabela 'COMORBIDADE' onde todas as colunas, exceto a chave, são valores booleanos. Os valores INT ocupam 4 *bytes*, BIGINT ocupam 8 *bytes* e o TINYINT ocupa 1 *byte* [36].

Assim, foi criado um *script* SQL para criar as tabelas, e um outro para inserir os valores dos enumerados (e.g as tabelas 'VACINA', 'PAIS' e 'MOTIVO_ADMISSAO').

De forma a extrair os dados do ficheiro Excel e saber que colunas irão para que tabela, é realizada uma interrogação a cada uma destas tabelas. Com isto, são criados tuplos com a formatação 'nome_tabela, lista das colunas'. Esta abordagem é seguida caso seja necessário inserir novas colunas, eliminar ou mesmo alterar os nomes das colunas, basta alterar na base dados, sem alteração do código. Também permite extrair as colunas necessárias do Excel caso existam mais.

Tendo as colunas de cada tabela, são extraídos os valores do ficheiro Excel para cada tabela. Ao processar cada tabela, são eliminados os seus valores presentes na base de dados antes de serem inseridos os novos valores nas respetivas tabelas em modo concatenação. Foi assim pensado, de forma a que as tabelas não sejam eliminadas e perderem os tipos já criados.

O referido *script* também cria a vista '**v_patient_data**', que consiste na agregação destas tabelas, como se encontravam inicialmente no ficheiro Excel. Por fim, são geradas duas outras vistas, '**v_patient_data_result_num**' e '**v_patient_result_cat**', que consistem na agregação desta última com as tabelas com os resultados numéricos ('v_agreg_num_values') e categóricas ('v_agreg_cat_values'), respetivamente, criados na Subsecção 3.3.2.

Tendo os DL e os DCD presentes na base dados, é realizado agora processamento sobre estes dois tipos de dados. O objetivo passa por agregar os dois tipos de dados em apenas uma estrutura. Ao fazer esta agregação, são também geradas outras informações, dadas as datas de colheita, tais como:

- Indicação se o paciente se encontra com VMI ou ECMO, nessa data;
- O dia em que se encontrava na UCI, para essas datas;
- Considerando como referência a data, a contagem dos dias até à morte.

Estas informações podem ser adquiridas através de cálculos entre colunas que representam a data de colheita, e as datas de início e fim de VMI e ECMO. Relativamente aos dias na UCI, pode ser feita uma diferença de dias com a data de colheita e a data de admissão na UCI. Por fim, o número de dias não é tão direto, neste caso é necessário saber se o paciente morreu ou não e extrair a última data, que representa o dia da morte. Com isto, são geradas de duas novas vistas, **v_patient_data_result_cat** para os resultados categóricos e **v_patient_data_result_num** para os resultados numéricos.

4

Desenvolvimento da solução informática

Neste capítulo encontra-se descrito o desenvolvimento da solução, desde a arquitetura utilizada até à fase da implementação dos módulos que produzem e mostram ao utilizador os resultados das análises estatísticas e as representações gráficas.

Na Secção 4.1 é realizada uma breve introdução ao capítulo. A arquitetura da solução é descrita na Secção 4.2. Na Secção 4.3 é apresentada a escolha das linguagens, bibliotecas e *frameworks* utilizadas para a implementação. A implementação da solução, a interface do utilizador, a *Application Programming Interface* (API), e o sistema de autenticação dos utilizadores descrevem-se na Secção 4.4. Por fim, na Secção 4.5 fala-se da implementação dos diferentes módulos que permitem realizar as análises estatísticas dos dados, bem como os diferentes gráficos e diagramas disponíveis na solução.

4.1 Introdução

Após a realização de todos os procedimentos relativos ao pré-processamento dos dados originais e ao processo ETL, obteve-se uma base de dados a partir da qual é realizada a implementação da solução, desde a conceção da arquitetura, escolha de ferramentas e linguagens, até à própria implementação do código de forma a realizar as funcionalidades pretendidas.

4.2 Arquitetura da solução

Visto que este projeto teve como ponto inicial um trabalho realizado por um bolseiro [55], onde já se encontram *scripts* criados em Python e uma base de dados em MariaDB, o desenho da arquitetura foi concebido tendo em consideração esse trabalho. O objetivo do presente trabalho é a criação de um *dashboard* para visualizar dados estatísticos e representações gráficas de um conjunto de pacientes críticos com COVID-19. Adicionalmente, a aplicação permite também a importação e exportação de dados, bem como a gestão de utilizadores.

Na Figura 4.1, é apresentado o diagrama da solução proposta. Esta consiste num servidor *web*, num servidor aplicacional (API), numa base de dados SQL, numa base de dados NOSQL, uma fila de mensagens e um serviço ETL.

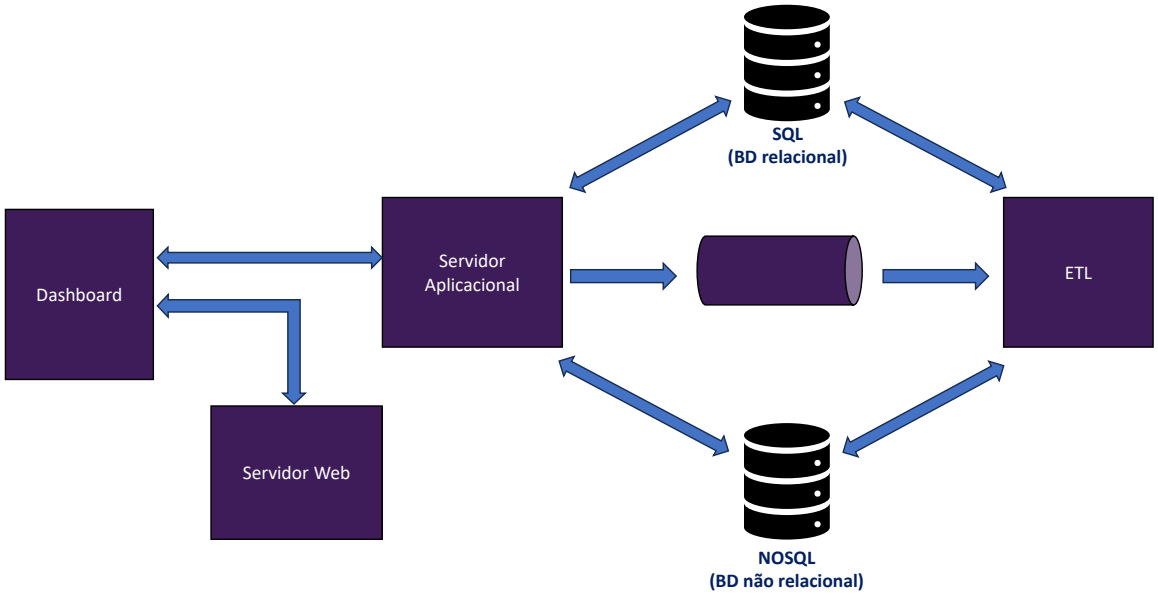


Figura 4.1: Arquitetura da solução

Para a visualização de dados, foi adotada uma arquitetura de três camadas, apresentada na Figura 4.2. Esta arquitetura consiste nas camadas de [3][27]:

- **Apresentação**, que diz respeito ao *dashboard*, fornecido pelo servidor *web*, que por sua vez fornece a interface do utilizador;
- **Lógica**, que corresponde ao servidor aplicacional. Este encontra-se na camada intermédia, que possui a lógica de negócio e o processamento de *inputs* de utilizador. Esta realiza também a ligação entre as duas outras camadas;
- **Base de dados**, com servidor de base de dados SQL, onde se encontram armazenados os dados persistentes da aplicação.

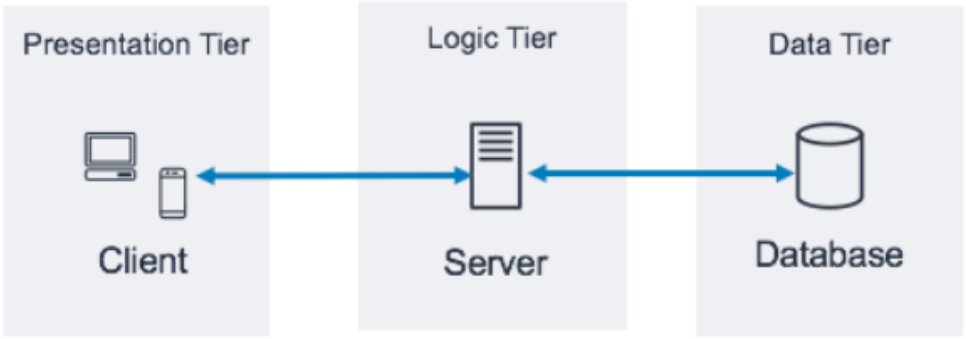


Figura 4.2: Arquitetura de três camadas [3]

Esta arquitetura permite o desacoplamento das várias camadas, ou seja, das várias funções. Assim, é possível a alteração de um camada sem alteração das restantes, incrementando a segurança visto que a camada do utilizador não consegue comunicar diretamente com a base de dados. A implementação em três camadas conduz a uma melhor escalabilidade, uma vez que cada camada pode ser escalada independentemente das outras. Outra vantagem desta separação é a redução da complexidade do projeto, permitindo que equipas de desenvolvimento distintas se foquem em diferentes camadas [27].

De forma a permitir a importação de novos dados enquanto a solução se encontra em produção, realiza-se comunicação assíncrona entre a API e o serviço de ETL através de uma fila de mensagens.

De forma a passar os ficheiros recebidos para importação da API para o ETL é utilizada uma base de dados NOSQL para armazenamento. Decidiu-se optar por um armazenamento deste tipo, pois, não há necessidade de ter uma base de dados relacional, visto não ser necessário realizar operações transacionais. Os dados são temporários, também não há necessidade de ter modelos mais complexos ou de interrogações para além da extração pelo identificador.

4.3 Linguagens e bibliotecas

Nesta secção são apresentadas as escolhas de linguagens e bibliotecas utilizadas para a realização do projeto para as três camadas.

Começando com a **camada de apresentação**, para a criação da interface do utilizador do *dashboard*, pensou-se numa aplicação *web*, visto não ser necessário instalação de *software*, uma vez que este se executa num *browser*. Sendo uma aplicação *web*, optou-se pela linguagem **JavaScript**, que é interpretada, *prototype-based*, *single-threaded*, dinâmica e multi-paradigma [37]. Adicionalmente é uma das linguagens mais utilizadas para o desenvolvimento *web* [60][61].

Sendo que seria utilizado JavaScript, optou-se também por incorporar **TypeScript** quando fosse relevante, que é uma linguagem *open-source* construída em cima de

JavaScript, o que significa que o que funciona em JavaScript também funciona em TypeScript. A linguagem TypeScript é tipificada, ou seja, tem noção de tipos, ao contrário do JavaScript, o que é vantajoso quando se trabalha em grandes aplicações de modo a gerir um pouco a complexidade das mesmas. De um modo muito geral, pode-se dizer que TypeScript é JavaScript mas com tipos, o que permite a transição de uma linguagem para a outra é a adição ou eliminação dos tipos [63].

Quanto às bibliotecas utilizadas para construir interfaces de utilizador, optou-se pela biblioteca em **React** que tem código aberto grátis e utiliza JavaScript. Nesta biblioteca há uma grande ênfase em reutilização de código, via componentes [50]. De forma a utilizar componentes com algum estilo foi utilizada a biblioteca **React Bootstrap**, que fornece isso mesmo.

Estas linguagens foram escolhidas pela sua popularidade, quantidade de documentação e também por já deter algum conhecimento sobre as mesmas.

Com base nos tipos de gráficos e diagramas que o *dashboard* incorpora, foi realizada uma pesquisa das bibliotecas mais adequadas, com acesso gratuito, e de código aberto. Desta pesquisa identificaram-se várias bibliotecas disponíveis onde as que se seguem são as que aparentam ser mais adequadas [21][22][23][24][25]:

- **D3** [9], que é uma biblioteca muito utilizada, no entanto, apresenta uma grande curva de aprendizagem;
- **Victory** [64], limitada a React, possuindo um pequeno sub-conjunto de gráficos e diagramas, mas destes, possui todos os mais importantes;
- **C3** [5], que é um *wrapper* do 'D3', que já contém alguns tipos de gráficos, mas não inclui o diagrama em caixa;
- **Chart.js** [6], a qual funciona com React, *Javascript*, Angular e Vue. Disponibiliza gráficos e diagramas visualmente apelativos, mas não contém diagramas em caixa;
- **Recharts** [52], a qual funciona apenas com React;
- **Plotly** [43], a qual disponibiliza um grande número de representações gráficas.

Destas bibliotecas, 'Plotly' foi a escolhida para usar neste trabalho uma vez que fornece um alto nível de flexibilidade na construção dos gráficos sem ter uma grande curva de aprendizagem. A biblioteca **react-plotly-js** contém os componentes React para a biblioteca 'Plotly'.

Quanto à **camada de lógica**, visto esta já ter uma componente escrita em **Python**, foi escolhida essa mesma linguagem. Assim sendo, é necessária a escolha de um biblioteca em Python para auxiliar na criação um servidor aplicacional. De modo similar à implementação da camada de apresentação, foi realizada uma pesquisa por essas bibliotecas [46][47][48][49]. Das bibliotecas encontradas, destacam-se:

- **Flask** [18], *framework* que aparenta ser adequada para prototipagem;
- **django** [12], *full-stack framework* para realizar as camadas de apresentação e lógica;
- **FastAPI** [17], *framework* simples e rápida. Também permite desenvolvimento *full-stack*;
- **Pyramid** [45], *framework* também conhecida mas aparenta não ter tanta documentação.

Destas foi escolhida a **FastAPI**, dado que possui uma boa documentação e é simples de usar.

Por último, a **camada de dados** encontra-se em MariaDB (base de dados relacional), visto já estar assim criada do trabalho realizado previamente por [55].

Relativamente à escolha da base de dados **NOSQL**, existem várias soluções, sendo as mais conhecidas:

- CouchDB;
- MongoDB;
- Redis;
- Cassandra.

Destas, a escolhida foi o **MongoDB** por ser uma base de dados documental (JSON) bem documentada, permitindo o armazenamento de ficheiros com qualquer dimensão (limitado ao computador).

Relativamente à **fila de mensagens**, também existem várias alternativas, mas a escolhida foi o **RabbitMQ** pela boa documentação. A importação não será utilizada com muita frequência, não é necessário um serviço de filas muito específico, desde que esteja garantido o seu funcionamento correto.

4.4 Implementação da solução

Nesta secção é descrita a implementação da solução informática referente à parte da camada de apresentação (servidor *web*) e à camada de lógica (servidor aplicacional).

4.4.1 Implementação do servidor *web*

O *dashboard* é fornecido pelo servidor *web*. Este consiste numa aplicação que corre num *browser web* seguindo uma implementação *Single-Page Application* (SPA). Uma implementação deste tipo implica que é carregado apenas um documento *web*, e para alterar o conteúdo do corpo da página a ser apresentado são efetuados pedidos *Hypertext Transfer Protocol* (HTTP).

De maneira a navegar pela página, existem duas barras de navegação: uma horizontal, e uma vertical que apenas se encontra disponível após autenticação.

A barra de navegação horizontal contém o título da aplicação e também um botão que permite iniciar/terminar a sessão.

A barra de navegação vertical, apenas acessível após a autenticação, contém direções para as páginas através de menus e sub menus. Na Figura 4.3 é apresentado um exemplo desta barra, onde existem dois menus (‘Dados Transversais’ e ‘Dados Longitudinais’), sendo que o botão de “Dados Longitudinais” dá acesso a dois sub menus (‘Curvas Sobrevivência’ e ‘Evolução Parâmetros’).

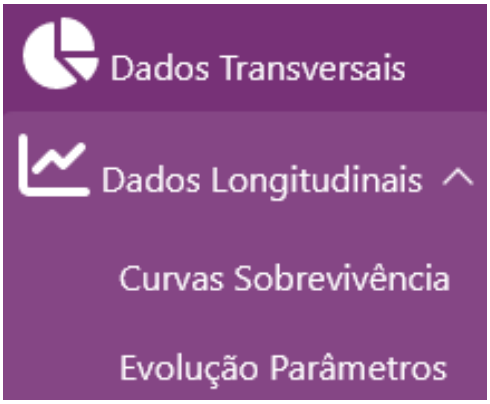


Figura 4.3: Exemplo da barra de navegação vertical

O componente principal desta barra de navegação vertical, divide a página em dois, um para a barra em si, e o resto para a restante página. Na página, encontram-se os formulários onde o utilizador preenche a informação que pretende visualizar, e onde é apresentada a representação gráfica correspondente.

Os formulários preenchidos pelo utilizador, podem possuir campos em comum. Assim, foram criados componentes para cada campo de formulário, de maneira a reutilizar o código, sempre que possível. No entanto, alguns destes componentes permitem alterações, como por exemplo, o número máximo de opções selecionadas em listas suspensas pode variar entre formulários.

Para ilustrar, é apresentado na Figura 4.4 um exemplo de formulário que permite visualizar a evolução temporal de uma variável quantitativa. Nesta, existe um componente de formulário (marcado a vermelho) que contém vários componentes de campo (marcados a verde).

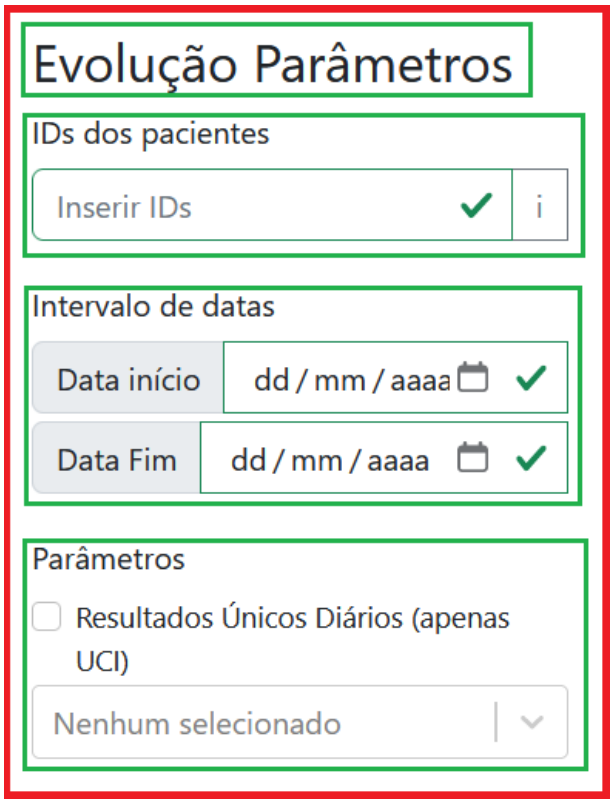


Figura 4.4: Formulário para geração de um gráfico de linhas para analisar a evolução de um parâmetro

Estes componentes de formulário internamente utilizam os componentes de campos, passando algumas informações, tais como:

- código de erros, utilizado por todos os componentes de campos, de forma a apresentar uma mensagem de erro;
- um *handler*, também utilizado por todos os componentes de campo, que será invocado quando existir alguma alteração no campo;
- lista de opções e de valores selecionados, para os casos de listas suspensas.

Para o caso das listas suspensas, foi utilizada a biblioteca 'React Select' [51], que permite criar estas listas, e sobre este foi criado um *wrapper* de forma a permitir a indicação do número máximo de valores selecionados.

Em relação à verificação do preenchimento dos campos, esta é realizada cada vez que o utilizador altera o conteúdo do formulário, com a exceção das datas, de maneira a apresentar uma mensagem de erro caso algo não se encontre corretamente preenchido. Antes de realizar o pedido para a API é feita novamente uma verificação de todos os campos. Desta forma é acautelada a possibilidade de o utilizador conseguir alterar o tipo de dados no documento *HyperText Markup Language*

(HTML). Se tudo estiver corretamente preenchido, é então realizado o respetivo pedido à API.

A API, por sua vez, também realiza verificações e em caso de erro envia uma resposta com indicação dos campos que estão em erro, gerando assim as mensagens de erros na interface do utilizador.

Cada gráfico corresponde a um componente que contém o desenho do gráfico e um conjunto de *inputs*, denominados controladores, de forma a controlar alguns aspetos da representação dos gráficos, como por exemplo, escolher alguma vaga em específico ou representar vários gráficos separados por vagas.

Por outras palavras, cada componente de gráfico é constituído por dois componentes:

- 'MyPlot', que também consiste em dois componentes:
  - 'ModalDownload' que permite descarregar a imagem num dos formatos disponíveis no Plotly (i.e. SVG, PNG, JPEG e WEBP) e também permite definir o nome do ficheiro;
  - 'Plot', da biblioteca 'react-plotly' que realiza o respetivo desenho do gráfico onde também é colocado por omissão uma classe *Cascading Style Sheets* (CSS).
- 'Controller', onde cada um dos gráficos tem um específico onde se define os campos para controlar os dados. Todos os gráficos existentes, exceto um, têm um que permite escolher a vaga em específico, e um outro para separar o gráfico em tantos outros onde cada um corresponde a uma vaga.

O primeiro componente, o '**ModalDownload**', permite o descarregamento do gráfico apresentando um formulário com uma caixa de texto para redefinir o nome, e uma lista que permite escolher qual o formato pretendido da imagem a descarregar. Na Figura 4.5, é apresentado a janela com o formulário para o descarregamento da imagem. É possível escolher os seguintes formatos, que estão disponíveis no 'Plotly':

- SVG;
- PNG;
- JPEG;
- WebP.

Figura 4.5: Janela para descarregamento de imagens

Para conseguir este descarregamento, o componente 'MyPlot' adiciona um novo botão ao objeto 'config', que será passado ao 'Plot'. Ao clicar neste botão, é ativado o modal referente ao 'ModalDownload', que por sua vez irá apresentar o respetivo formulário. Com este formulário preenchido é descarregada a imagem da mesma maneira que o 'Plotly' já o fazia, mas neste foi necessário fazê-lo de forma programática visto ter este formulário na fase intermédia do processo.

O segundo componente utilizado pelo 'MyPlot', é o 'Plot' (que pertence à biblioteca 'react-plotly'), que é usado para fazer o desenho dos gráficos com os dados que são passados.

Este componente **'Plot'** recebe, principalmente, três tipos de dados diferentes:

- '**data**', que se refere a uma lista de objetos, em que cada um destes refere aos gráficos a serem desenhados. Nestes objetos indica-se quais os valores no eixo das abcissas X, e nas ordenadas Y, tipo de gráficos, nome de legendas, cores, e entre outros;
- '**layout**', é um objeto onde se indicam informações sobre o *layout* da "página" onde os gráficos são desenhados, como por exemplo a dimensão, título, nome dos eixos, bordas, anotações (que são textos que se podem colocar juntamente com os gráficos), formas (são formas, como por exemplo retas, que podem ser colocadas juntamente com os gráficos), e entre outros;
- '**config**', é um objeto que permite definir propriedades como botões no menu e também interatividade com o gráficos, como por exemplo utilizar o *scroll* de modo a fazer *zoom*, adicionar ou remover botões no menu de botões, entre outros.

Existem vários componentes que utilizam este componente 'MyPlot', que estão encarregues de saber como processar os dados de modo ao 'MyPlot' conseguir desenhar o gráfico pretendido. Há um componente destes para cada tipo de gráfico utilizado na solução.

Em relação aos utilizadores, cada um possui um papel - 'admin' ou 'user'. Os utilizadores com o papel de 'admin' têm disponível uma funcionalidade adicional que consiste na **gestão de utilizadores**. Esta gestão permite, criar e remover utilizadores, desativar ou ativar utilizadores, trocar os seus papéis ou ainda reiniciar a senha.

Na interface do utilizador é apresentada uma tabela com os vários utilizadores com botões referentes à alteração do utilizador ou à eliminação do mesmo, com sistema de paginação. Também é apresentada uma barra de pesquisa para extrair apenas os utilizadores que correspondem ao texto inserido.

4.4.2 Implementação da API

O servidor aplicacional, escrito em Python, encontra-se dividido em vários módulos com organização, escalabilidade, reutilização de código e também a injeção de dependências. Estes módulos são:

- **endpoints**, que é onde se encontram definidas as rotas e os *handlers* da aplicação;
- **services**, que contém os contratos e as implementações específicas dos serviços que tratam da lógica da aplicação;
- **dal**, que contém contratos e as suas implementações específicas, que tratam da lógica de acesso à base de dados;
- **dto**, que são objetos utilizados para a passagem de informação entre camadas, sendo entre a camada de negócio e a de acesso a dados, ou mesmo entre servidores;
- **exceptions**, encontram-se as exceções criadas para uso da aplicação;
- **utils**, contém funções utilitárias para a aplicação.

Na Figura 4.6 encontra-se o diagrama da estrutura do servidor aplicacional. Nesta, um utilizador realiza pedidos à aplicação através do protocolo HTTP que se encontra disponível no módulo 'endpoints'. Este módulo reencaminha este pedido ao 'services' que irá realizar verificações de parâmetros recebidos e de regras de negócio e comunica com o módulo 'dal' de forma a obter informação da base de dados.

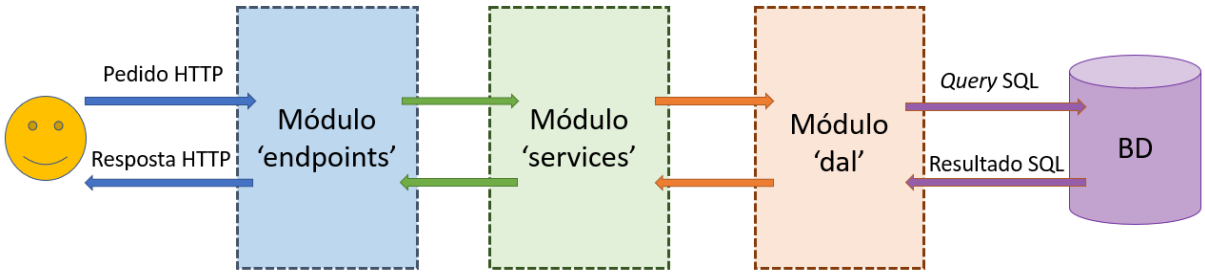


Figura 4.6: Diagrama da estrutura do servidor aplicacional

O *script* principal do servidor é o **server.py**. Este está encarregue de criar as instâncias necessárias, definir configurações do servidor e de colocar a aplicação a funcionar.

De modo a permitir a injeção de dependências nos módulos a utilizar, foi seguida uma abordagem de programação orientada a objetos, definido classes como contratos para comunicação entre os diferentes módulos.

No **módulo *'endpoints'***, são encontrados os *handlers* para as chamadas para as várias rotas existentes. Nestes é possível indicar se é necessário haver autenticação ou não, e se for necessário, se é necessário ter papel de administrador ou não, através da execução de funções previamente à execução do respetivo *handler*.

Estas funções simplesmente verificam a existência de *cookies* de acesso (*access token*), extraindo o utilizador presente e indicando a sua existência na base de dados e também qual o papel que o utilizador possui.

Com o sucesso dessa validação, é enviado o pedido para o respetivo serviço presente no **módulo *'services'***. Este módulo realiza as verificações necessárias quanto aos dados recebidos do pedido, como o tipo dos valores e restrições de domínio. Com isso, são realizados os pedidos necessários ao módulo 'dal' de forma a realizar alguma alteração de estado da base de dados, ou extrair informação de modo a construir a resposta para ser enviada ao utilizador.

Por fim, o **módulo 'dal'** realiza a lógica para o acesso aos dados na base de dados, com a construção e execução de interrogações SQL consoante as informações passadas pelo módulo anterior, retornando os resultados de volta ao módulo 'services'.

Uma preocupação a ter no desenvolvimento de aplicações *web*, neste caso na API, é a forma como as chamadas ao servidor são realizadas a nível de concorrência. Esta questão refere-se ao processamento de várias tarefas (neste caso as chamadas à API), que começam, executam e terminam em períodos diferentes que se podem sobrepor, sem qualquer ordem específica.

Não é desejável que estas chamadas sejam executadas por uma única *thread*, bloqueando outras chamadas que possam coexistir devido a um processamento mais longo de uma determinada chamada, bloqueando assim a única *thread* existente. Ou seja, como consequência pode ocorrer que um utilizador da aplicação fique à

espera que um outro utilizador acabe a sua tarefa. A biblioteca 'FastAPI' tem esta situação em consideração, na definição dos ***handlers*** dos ***endpoints***, de acordo com a documentação [17], sendo estes marcados com:

- ***async def***, para o caso de haver necessidade de processamento externo, como chamadas à base de dados que permitem processamento assíncrono, ou quando não é necessária qualquer comunicação externa;
- ***def***, para quando essas chamadas não permitem processamento assíncrono, ou em caso de existir dúvida.

A 'FastAPI' tem um comportamento diferente para essas diferentes marcações. Para o primeiro caso é executado por uma única *thread*, enquanto no segundo caso é utilizada uma *thread pool* externa para o processamento [17]. Assim, para processamentos assíncronos é utilizado o *async def*, e para os que não permitem assincronismo ou para processamento longo mas interno é utilizado o *def*.

A solução, além dos gráficos e diagramas, também permite a exportação dos DL e dos DCD num ficheiro Excel, e a importação de novos dados.

Esta funcionalidade de **exportação** permite extrair os respetivos dados, usando também alguns filtros como os identificadores dos pacientes e o intervalo de datas de colheitas. Na API são gerados os *bytes* do conteúdo de um ficheiro *Comma-separated values* (CSV) com os dados extraídos da base de dados, e finalmente comprimido num ficheiro ZIP. Decidiu-se optar por um ficheiro com a extensão CSV, por ter um tamanho inferior a um ficheiro com extensão xlsx/xls e o seu processamento ser bastante mais rápido (e.g. um redução de 7 minutos para 20 segundos, com os 1239 pacientes).

No caso da **importação de novos dados**, o utilizador envia um ficheiro ZIP, que pode conter duas pastas: 'lab_data', que contém os DL; e 'patient_data' que contém os DCD. Estes terão de passar pelo processo ETL antes de serem disponibilizados para visualização, processo que pode demorar bastante tempo, e não faz sentido o utilizador de um *browser web* ficar este tempo todo à espera de uma resposta HTTP. Assim a solução, consiste na criação de um serviço para correr o processo ETL onde a API irá despoletar esse processo de forma assíncrona utilizando filas de mensagens.

De forma a conseguir enviar os ficheiros (novos dados) da API para o serviço ETL, decidiu-se utilizar uma base de dados NOSQL. Assim, o ficheiro será guardado nesta nova base de dados pela API, gerando um identificador que será enviado na mensagem para o ETL, de forma a este último saber localizar o ficheiro a processar.

Na API é armazenado o binário do ficheiro comprimido na base de dados NOSQL e gerada uma nova entrada, em formato JSON, com o identificador do ficheiro. Este JSON contém um identificador da entrada, o identificador do ficheiro, um estado e o progresso do processamento.

4.4.3 Autenticação

Um aspeto importante numa aplicação *web*, é a questão da segurança, nomeadamente nos acessos à mesma. Assim, a aplicação desenvolvida considera um sistema de *login* como forma de **autenticação**, onde um utilizador insere o seu nome e a senha para entrar na aplicação.

Estas informações são armazenadas na base de dados relacional, sendo que, por motivos de segurança não é colocada realmente a senha, mas sim um ***hash*** da mesma [40]. Um *hash* é uma representação de tamanho fixo de uma *string* de qualquer tamanho, e tem como características ser determinístico e *one way*, ou seja, sabendo apenas o *hash* não é computacionalmente exequível (dependendo do algoritmo) descobrir a *string* original. Deste modo, mesmo que haja uma fuga desta base de dados, os perpetradores não conseguem aceder à aplicação, uma vez que não sabem a senha. Contudo, é ainda possível descobrir senhas comuns de vários utilizadores, visto que o *hash* tem como característica ser determinístico. Para evitar estas situações, é utilizado **'salt'**, uma sequência aleatória de caracteres que são adicionados à senha como forma de criar aleatoriedade ao *hash* resultante. Para utilização de algoritmos de *hash*, recorre-se à biblioteca **PassLib** [41], que fornece vários algoritmos para o efeito.

Segundo a documentação da 'FastAPI' é recomendado o algoritmo **BCrypt**, de 1999, que sobreviveu ao teste do tempo. No entanto, como documentado [40] na *Open Worldwide Application Security Project* (OWASP), que é uma organização sem fins lucrativos para melhorar a segurança de *software*, é recomendado o uso do algoritmo **Argon2id**, que é o vencedor de uma competição de *hashes* de senhas em 2015 [42], sendo que a segunda opção é o BCrypt.

A biblioteca 'FastAPI', disponibiliza módulos em Python que auxiliam no processo de autenticação, como descrito na documentação [17]. Para que o utilizador permaneça autenticado é gerado um *token*, mais concretamente um ***JSON Web Token* (JWT)**, que no fundo é uma *string* que verifica o utilizador [54].

Para obter este *token*, é necessário realizar um pedido HTTP para a rota /auth/login enviando no corpo do pedido um campo 'username' com o nome do utilizador, um campo 'password' com a respetiva senha, e um campo 'grant_type' com o valor "password". Também é necessário indicar o cabeçalho 'content-type' com o valor "application/x-www-form-urlencoded".

Este processo resultará numa resposta com o *status code* 422, caso não seja possível extrair qualquer um destes campos do corpo do pedido.

Uma vez que são extraídos com sucesso os campos obrigatórios, é realizada a geração do respetivo *token*. Para esta geração, é necessário autenticar o utilizador, verificando o seu nome e senha. Para tal, é extraído o utilizador e o *hash* da sua senha da base de dados, e é realizado o *hash* da senha que foi recebida e comparada com a armazenada na base de dados, resultando em respostas com *status code* 401, caso o utilizador não exista ou o *hash* da senha não seja igual à que está armazenada na base de dados.

Uma vez que o utilizador se encontra autenticado, é feita a geração do *token*. Para realizar esta geração é utilizado o algoritmo **HS256** para assinar, e como chave foram gerados 32 *bytes* aleatórios, a partir da ferramenta online 'openssl' encontrada em [8]. Este procedimento resulta numa sequência de 64 caracteres. O comando utilizado encontra-se na Listagem 4.1, assim como o resultado do comando.

```
$ openssl rand -hex 32

730e7324c666b0842dabc89055fb011af7a6989397f8a82cf6e2a24e9c96c9dd
```

Listagem 4.1: Comando 'openssl' para gerar 32 *bytes* aleatórios

O *token* é armazenado na forma de *cookie* contendo a *flag* **HTTPOnly**, que indica que não pode ser acedida via Javascript [7], sendo enviada ao servidor sempre que necessário. Esta é criada pelo servidor aplicacional com o sucesso da autenticação.

Esta solução apresentada não é invulnerável, mas oferece melhor proteção do que enviar o *token* ao utilizador e ser o *browser* a armazená-lo, seja em armazenamento local ou de sessão [2]. Também haveria a opção de armazená-lo localmente num estado de React, que seria possivelmente a opção mais segura mas não a mais prática, pois este seria perdido sempre que a página fosse reposta, o que implicaria iniciar a sessão novamente.

Qualquer caso de insucesso que possa acontecer, seja por falha de verificação do JWT, por o utilizador não existir, ou o nome não estar presente no JWT, ou estar inativo, é enviada uma resposta com o *status code* do tipo 4xx.

4.5 Métodos estatísticos para análise e representação gráfica dos dados

A solução informática processa e analisa os dados de acordo com o seu tipo - dados transversais ou dados longitudinais - permitindo que o utilizador possa aceder a dados estatísticos e representações gráficas para as variáveis de interesse. O utilizador pode solicitar uma consulta a ser realizada num dado momento no tempo (dados transversais), ou pode ser pedida a visualização da evolução temporal (dados longitudinais), dos vários parâmetros importantes, em contexto de doentes COVID-19, ao longo do tempo de internamento na UCI.

No que respeita aos métodos estatísticos para tratamento e análise dos dados, as variáveis nominais são analisadas pelas suas frequências absolutas e percentagens, sendo representadas por gráficos circulares ou gráficos de barras. Para as variáveis quantitativas apresenta-se o resumo descritivo caracterizado pelo mínimo, máximo, média, desvio padrão, percentil 25, 50 (mediana) e 75. Para representar graficamente os dados quantitativos utilizam-se diagramas em caixa, com a identificação de valores atípicos - *outliers* - caso existam. De forma a analisar a correlação entre

duas variáveis quantitativas são utilizados gráficos de dispersão, sendo disponibilizada a estimativa do coeficiente de correlação de Spearman e o respetivo valor-p do teste de hipótese à ausência de correlação entre as variáveis em estudo. Para tal, foi utilizada a biblioteca 'statistics.js' [58].

Para as variáveis quantitativas as comparações entre dois grupos independentes são realizadas pelo teste de Wilcoxon-Mann-Whitney utilizando a biblioteca [35], sendo utilizado o teste de Kruskal-Wallis para o caso de comparações entre três ou mais grupos, utilizando a biblioteca [32]. O estudo de sobrevivência é realizado pela estimação e representação das curvas de sobrevivência de Kaplan Meier. Para comparar estas curvas utilizam-se os testes de Logrank (também conhecido por Mantel-Haenszel, Mantel-Cox e Peto-Mantel-Haenzsel), Breslow (também conhecido por Wilcoxon) e Tarone-Ware [11].

A construção dos gráficos é realizada a nível da camada de apresentação, construída com a linguagem Javacript usando React, utilizando a biblioteca Plotly para a conceção dos mesmos. Esta solução possibilita a criação de vários tipos de gráficos, de acordo com a indicação do clínico.

Os tipos de gráficos disponibilizados são os seguintes:

- Gráficos barras e circulares - possibilitam descrever as contagens e as percentagens das diferentes classes das variáveis nominais;
- Gráficos de dispersão - permitem analisar a correlação entre duas variáveis quantitativas;
- Diagramas em caixa - possibilitam analisar a distribuição das variáveis quantitativas. Podem representar apenas uma distribuição para uma variável, ou serem diagramas em caixa em paralelo na mesma representação gráfica, e nesse caso permitem comparar as distribuições de uma variável quantitativa para as diferentes categorias de uma variável nominal de interesse;
- Curvas de sobrevivência - permitem analisar as probabilidades de sobrevivência de um conjunto de pacientes, ao longo do tempo de seguimento. Destaca-se neste estudo a comparação de curvas de sobrevivência para diferentes grupos de pacientes. O utilizador pode selecionar as categorias a comparar. Estas podem ser categorias de uma dada variável nominal ou, por exemplo, as categorias resultantes da categorização de variáveis quantitativas, através de pontos de corte associados a padrões de risco de certos parâmetros laboratoriais de interesse;
- Gráficos de linhas longitudinais - permitem analisar a evolução dos diferentes parâmetros ao longo do tempo de internamento na UCI. Existem duas variantes para este tipo de gráfico. A primeira permite comparar a evolução temporal do parâmetro de interesse para diferentes pacientes, e a segunda permite analisar de um modo geral os níveis de todos os pacientes.

Para construir cada um destes gráficos é necessário o preenchimento de um formulário que permite indicar vários filtros (e.g. pacientes, datas de colheita, vagas, e outros) e também indicar as variáveis nominais e quantitativas.

5

Implantação da solução informática

Neste capítulo é descrita a implantação da solução informática. Na Secção 5.1 é realizada uma introdução à implantação da solução.

Na Secção 5.2 encontra-se descrita a escolha da ferramenta. Na Secção 5.3 é feita uma breve descrição da ferramenta escolhida (Docker). A Secção 5.4 aborda a organização dos vários contentores, enquanto na Secção 5.5 descreve-se a criação das imagens destes contentores.

5.1 Introdução

Toda a solução descrita nos capítulos anteriores, por si, não traz tanto valor se não for realmente colocada em produção, ou seja disponibilizar de forma a poder ser utilizada pelos vários utilizadores que a requerem.

A fase de implantação de uma solução informática constitui um dos passos finais da aplicação, e é muitas vezes esquecida apesar de ser um dos passos mais importantes. Sem a implantação não há solução disponível para os utilizadores.

5.2 Escolha da ferramenta

Com a aplicação criada, é necessário fazer a implantação da mesma, de forma a colocá-la disponível para os utilizadores. O *deployment* aplicacional (*deployment* de *software*), é o processo de instalação, atualização, que permite a disponibilidade de uma ou mais aplicações para utilização, através de, por exemplo um *Uniform Resource Locator* (URL) num servidor [65].

No caso desta aplicação, pretende-se que seja disponibilizada de forma privada, apenas pode ser acedida por uma rede interna, uma vez que contém dados sensíveis de pacientes, apesar de estarem anonimizados. A aplicação em si contém as várias aplicações criadas no projeto, seja a API (em Python) ou a parte gráfica do *dashboard* (em Javascript).

Existem várias ferramentas e serviços que permitem disponibilizar as aplicações, e assim foi realizada uma pesquisa sobre estas para escolher uma. As ferramentas consideradas na pesquisa efetuada foram:

- **serve** [56], que consiste num módulo disponível em 'Javascript' que permite disponibilizar aplicações em 'React';
- **Nginx** [38], que para além de servir aplicações, também pode funcionar como balanceador de carga e *reverse proxy*;
- **Tomcat** [59], que é um projeto de Apache que funciona como servidor de *web* Java;
- **Docker** [13], que permite criar contentores isolados das várias aplicações, através de virtualização;
- **Heroku** [26], que permite construir, correr e operar aplicações na *cloud*;
- **Google App Engine** [20], um outro serviço de *cloud* que permite desenvolver aplicações.

Das várias ferramentas e serviços apresentados, a escolhida foi o **Docker**, pois esta permite desacopular as diferentes aplicações e colocá-las em execução separadamente. Também cria uma imagem de forma a facilitar a criação de um novo processo da aplicação.

5.3 Docker

O '**Docker**', é uma plataforma que permite criar, fazer a implantação, executar, atualizar e gerir contentores [28]. Estes contentores, consistem num empacotamento de tudo o que é necessário para colocar uma aplicação em execução, seja o código fonte, as bibliotecas, as dependências ou ficheiros de configuração.

Os contentores são executados em processos isolados entre si, criados e geridos pelo 'Docker', que é uma aplicação a ser executada sobre um sistema operativo, tal como representado na Figura 5.1.

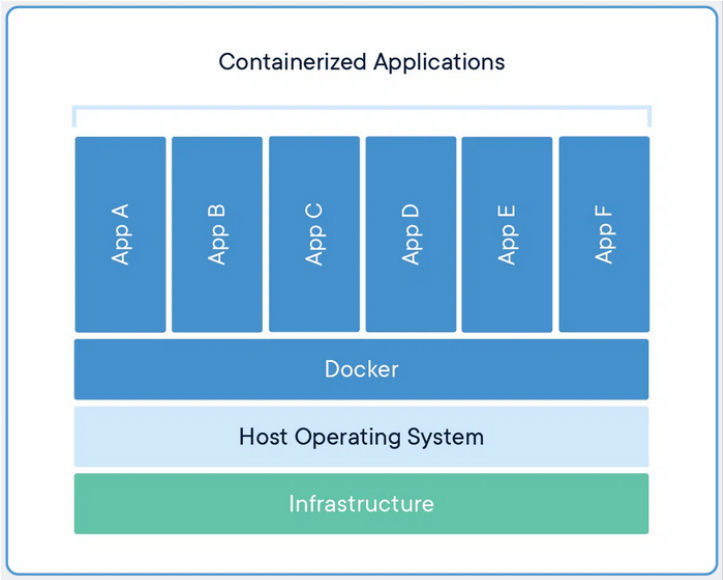


Figura 5.1: Diagrama de virtualização através de contentores [14]

Uma outra maneira de isolar aplicações seria através de máquinas virtuais, mas estas por sua vez, virtualizam o sistema operativo inteiro em vez de apenas a aplicação (ver Figura 5.2). Esta maneira teria maior uso de recursos (é necessário ter recursos disponíveis para reservá-los para a máquina virtual) e também o tempo de criação de uma máquina virtual, visto ser necessária a criação do sistema operativo inteiro com as várias bibliotecas e dependências, seja virtualização do tipo 1 (sobre *bare metal*) ou do tipo 2 (sobre o sistema operativo).

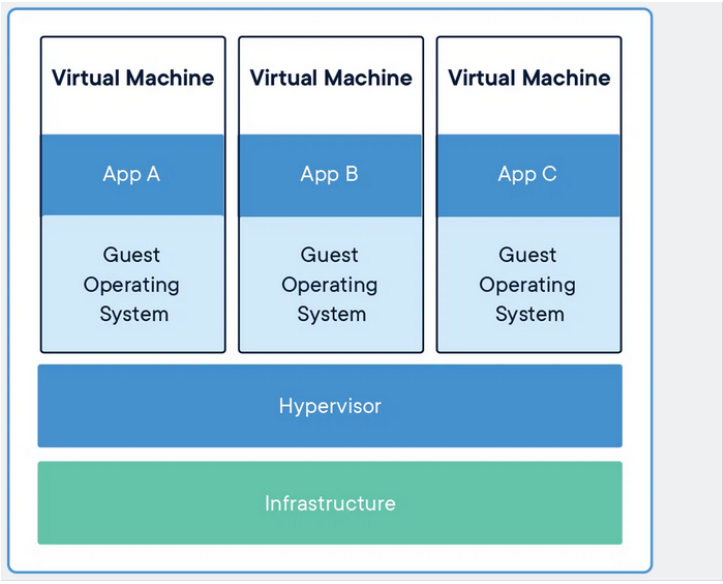


Figura 5.2: Diagrama de virtualização através de máquinas virtuais [14]

5.4 Organização dos contentores

Com a escolha do 'Docker', e com as aplicações criadas, a ideia é ter um contentor para cada processo (API, *dashboard*, fila de mensagens, ETL e base de dados NOSQL), de forma a serem processos isolados entre si, conforme apresentado na Figura 5.3. Em relação à base de dados NOSQL MongoDB, este possui dois serviços: a base de dados em si (MongoDB), e um *dashboard* para manipulação e visualização da mesma (Mongo-express).

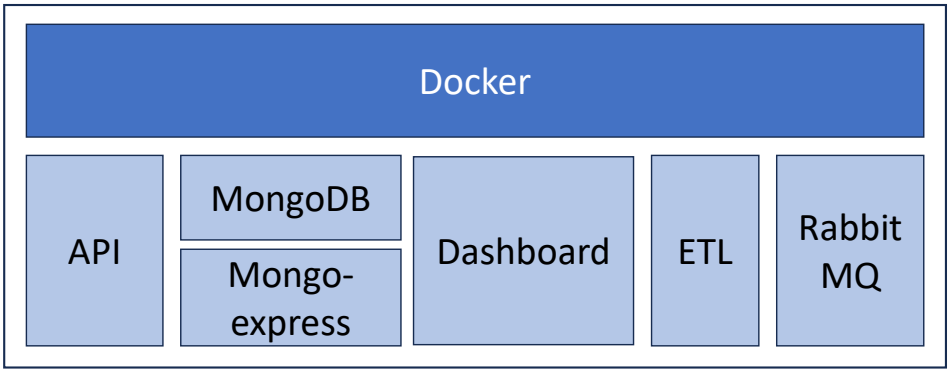


Figura 5.3: Diagrama inicial dos contentores

Também é necessário implementar a componente do *Hypertext Transfer Protocol Secure* (HTTPS). De acordo com a documentação apresentada na *framework* da 'FastAPI', existem várias ferramentas que tratam deste assunto e até são recomendadas algumas em [17]. Entre estas, há o '**Traefik**' [62], uma solução *open-source* de *reverse-proxy* HTTP e balanceador de carga, que também fornece geração e renovação automática de certificados *Transport Layer Security* (TLS) através de 'Let's Encrypt' (uma autoridade certificadora grátis e automática de certificados TLS) [33], e que funciona com o 'Docker'.

Assim, a ideia é ter este *reverse-proxy* com a geração e renovação automática de certificados TLS num outro contentor juntamente com os restantes. O diagrama evolui conforme o apresentado na Figura 5.4. Com isto, o 'Traefik' irá receber os pedidos HTTP dos utilizadores e redirecioná-los para a API ou *dashboard* consoante o URL do pedido.

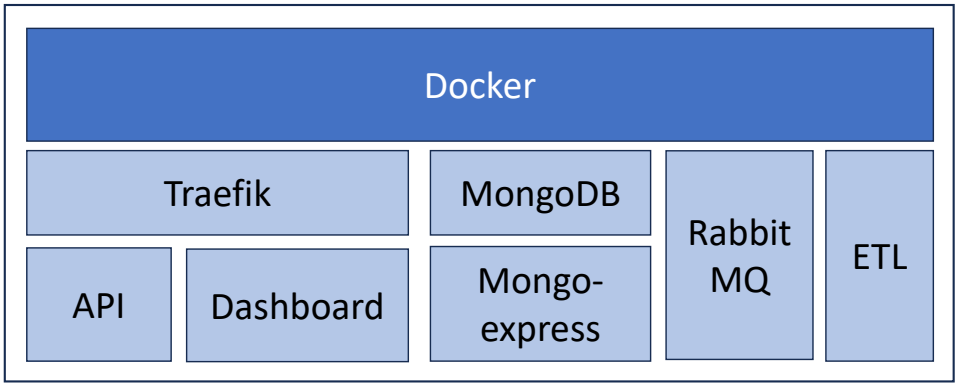


Figura 5.4: Diagrama dos contentores com *reverse-proxy*

Para executar a API e o ETL, basta executar o ficheiro principal que contém o 'main', visto este ser construído através da linguagem 'Python'. Estes simplesmente ficam em execução sem a necessidade de ter algo adicional.

Para o caso da aplicação 'React', é necessário construir os ficheiros estáticos de maneira a serem servidos por um servidor *web*. Para tal, é necessário algo que faça de servidor para aplicação, e das opções apresentadas na Secção 5.2, foi escolhida a ferramenta 'Nginx', uma vez que é muito utilizada e tem boa documentação e ainda tem ambientes para as imagens no 'Docker' [15].

De mencionar que o 'Nginx' também funciona como *reverse-proxy* sendo possível ter criação e renovação automática de certificados de TLS (através de 'crontab' no Linux), tal como o 'Traefik', e funciona como servidor *web*. Contudo, decidiu-se desta maneira, pois a ideia é ter as várias partes separadas e isoladas (API de um lado e *dashboard* do outro) e também porque é mais simples de realizar a automação dos certificados TLS em 'Traefik'. Se fosse implementado o 'Nginx' como *reverse-proxy* e servidor *web*, este iria ter duas funções em vez de uma.

Assim, fica-se com uma divisão conforme apresentada na Figura 5.5, onde há o contentor da API, o contentor da *dashboard* que é servida pela ferramenta 'Nginx', e por último o contentor do 'Traefik', conforme apresentado na Figura 5.5.

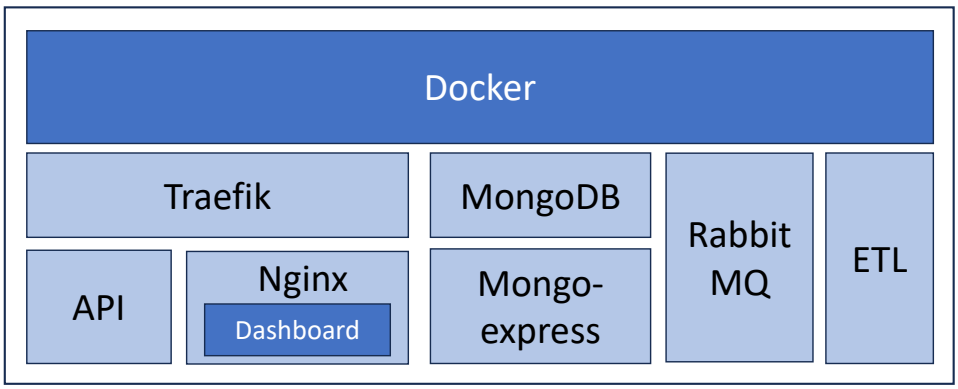


Figura 5.5: Diagrama final dos contentores

5.5 Criação das imagens Docker

Tendo a estrutura pensada e escolhidas as ferramentas que serão utilizadas, passa-se para a fase de implementação das imagens Docker, para a criação dos contentores. Estas imagens, são um conjunto de instruções para a criação de um contentor, como se fosse uma espécie de *template* para a criação de contentores.

Para a criação de imagens, é necessário criar um 'Dockerfile', um ficheiro que irá conter as tais instruções para as criações dos contentores, ou a criação de um ficheiro 'docker-compose.yml' que permite criar várias imagens de serviços diferentes. Nesta solução é utilizado este último ficheiro, sendo que para a criação da interface do utilizador, a API e o ETL são usados também 'Dockerfiles'.

No caso de aplicações em Python, API e ETL, é necessário gerar um 'Dockerfile'. Este ficheiro contém instruções para:

- Transferir e colocar em utilização um conector MariaDB mais recente, uma vez que o presente na imagem oficial do Docker contém uma versão mais antiga;
- Cópia do ficheiro 'requirements.txt' - contém todas as bibliotecas e as suas versões necessárias para a aplicação executar;
- Instalar essas dependências, é ainda definida uma variável de ambiente (PYTHONPATH), que indica ao interpretador do Python onde o código está localizado;
- Criar um utilizador não *root* com acesso à pasta onde se encontra a aplicação;
- Executar a aplicação.

A interface do utilizador, por sua vez, também necessita de um 'Dockerfile'. Neste caso, este 'Dockerfile' contém duas partes. A primeira, denominada de 'builder' que realiza a instalação das dependências e a construção dos ficheiros estáticos, num ambiente de 'node'. A segunda é o 'production' que trata de colocar a aplicação em funcionamento, num ambiente de 'Nginx'. Decidiu-se separar em duas fases, porque neste caso é necessário construir os ficheiros estáticos, e é preferível ter apenas o necessário no contentor em execução. Também desta maneira sabe-se que a execução será sempre igual, pois a fase de construção será sempre executada no mesmo ambiente.

A primeira fase do processo de construção deste ficheiro tem as seguintes instruções:

- Definir variáveis de ambiente no 'builder';
- Copiar os ficheiros 'package.json', 'package-lock.json' e 'npm-shrinkwrap.json' da máquina hóspede para o ambiente;

- Instalar as dependências indicadas nesses ficheiros em modo de produção;
- Copiar os restantes da máquina hóspede;
- Construir os ficheiros estáticos;

A segunda consiste em apenas copiar os ficheiros estáticos da fase anterior, trocar um ficheiro de configuração do Nginx, expor um porto e, finalmente, colocar o Nginx em execução de modo a servir os ficheiros gerados pelo React.

As restantes aplicações não necessitam de um 'Dockerfile'. No caso do Mongo apenas é necessário definir variáveis de ambiente, que estão no ficheiro '.env'.

Quanto ao 'Traefik', não tem um 'Dockerfile' mas é necessário ter algumas configurações no 'docker-compose', tais como:

- Indicar os portos abertos (80 para HTTP e 443 para HTTPS);
- Configurar a geração/renovação dos certificados TLS;
- Redirecionar de forma permanente o tráfego HTTP para HTTPS;
- Configurar alguns cabeçalhos de segurança disponíveis no Traefik.

Com o ficheiro 'docker-compose' e os 'Dockerfile' criados, basta executar o 'docker-compose' que irá criar as imagens e criar os diversos contentores como indicado, colocando assim a solução em modo de produção.

6

Resultados e visualização de dados

Neste capítulo são apresentadas as várias páginas disponíveis na solução na Secção 6.1. Adicionalmente são apresentados exemplos dos vários gráficos e diagramas, para representar os dados transversais na Secção 6.2, e por fim, na Secção 6.3 apresentam-se exemplos das representações gráficas para dados longitudinais.

6.1 Páginas do *dashboard*

O *dashboard* contém várias páginas para as diversas funcionalidades disponíveis. Na Figura 6.1 apresenta-se a página inicial da aplicação onde são apresentados dados gerais do que se encontra na base de dados, quanto ao número de pacientes, o número de colheitas e as vagas de COVID-19 associadas aos dados disponíveis. Para além dos dados presentes, também é apresentada a barra de navegação horizontal que contém o título da aplicação e um botão que permite terminar a sessão, bem como a barra de navegação lateral que permite navegar entre as diferentes páginas após a autenticação. Na imagem, apresenta-se a visualização obtida por um com um papel 'admin'. Este tipo de autenticação, para além das análises dos diferentes gráficos, permite também a gestão de utilizadores.

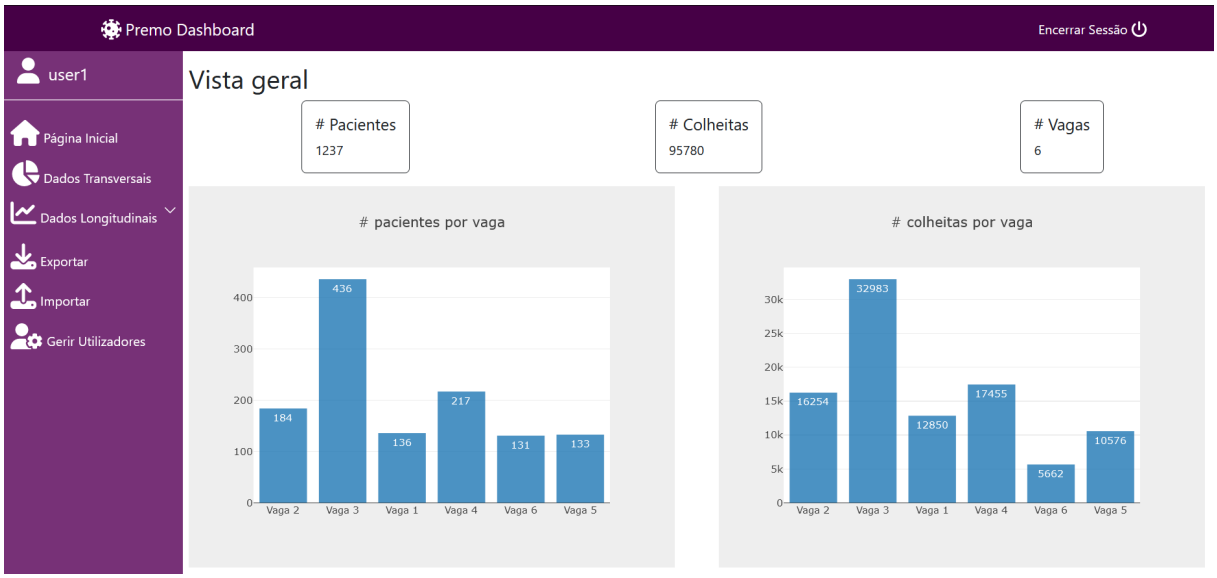


Figura 6.1: Vista geral da solução

Na Figura 6.2 é apresentada a página que contém o formulário para preenchimento de modo a apresentar as representações gráficas para dados transversais, como são exemplo os gráficos circulares/barras, diagramas em caixa e gráficos de dispersão.

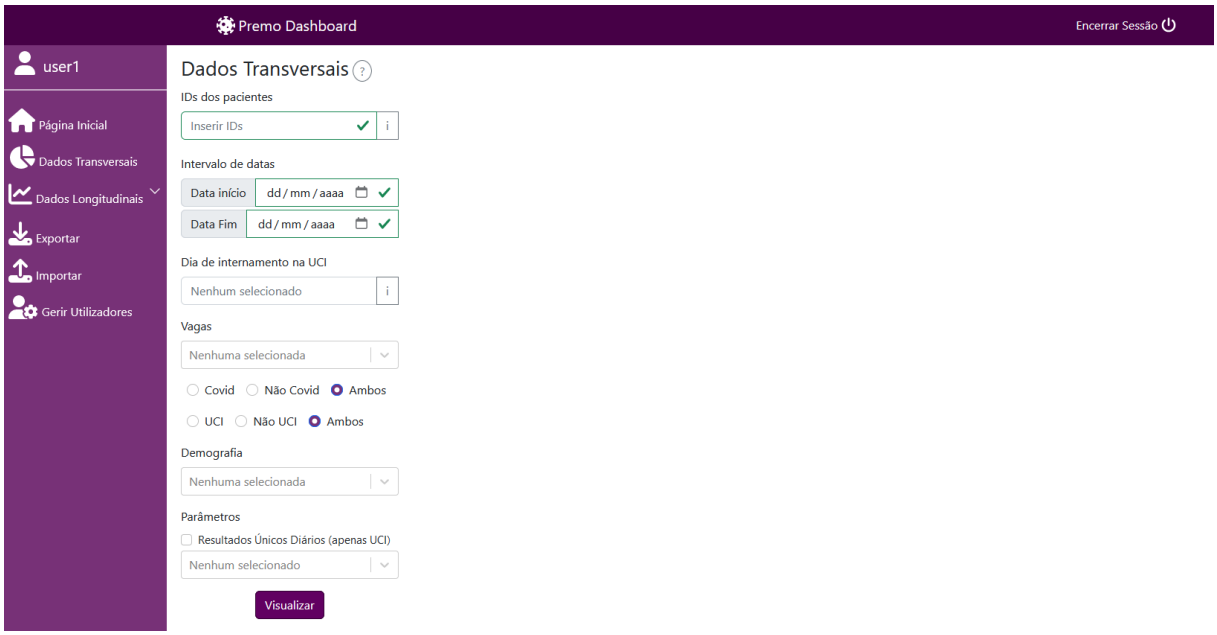


Figura 6.2: Página para visualização de gráficos obtidos a partir de dados transversais

Para o caso dos gráficos para os dados longitudinais, é apresentada na Figura 6.3 a página relativa às curvas de sobrevivência com o respetivo formulário. Enquanto na Figura 6.4 é apresentada a página para a análise da evolução de parâmetros, com o respetivo formulário.

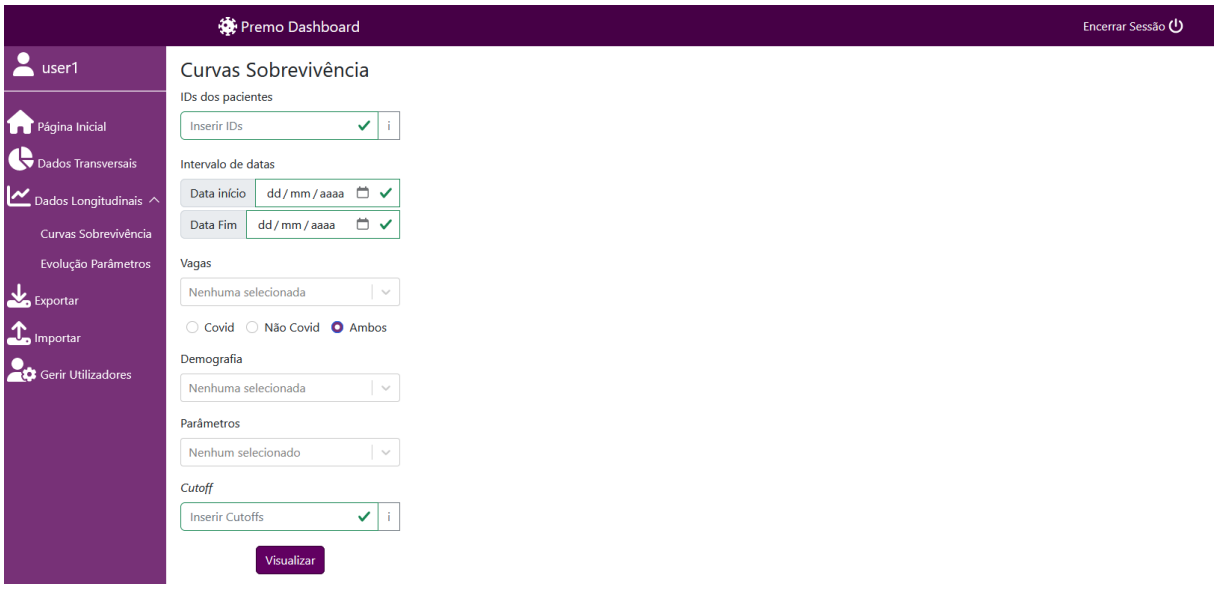


Figura 6.3: Página para visualização das curvas de sobrevivência

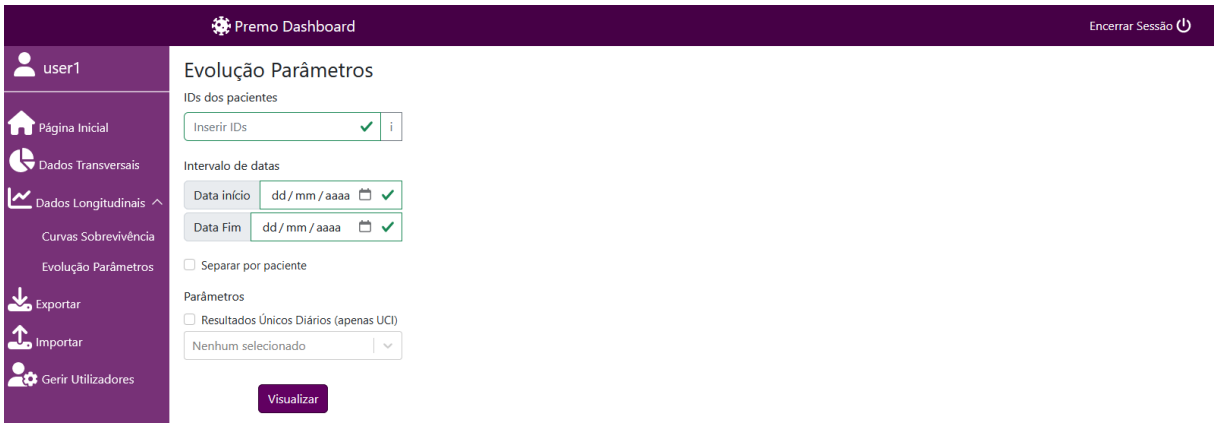


Figura 6.4: Página para visualização da evolução temporal de parâmetros

Para ambos os casos, dados transversais ou dados longitudinais, o utilizador deve realizar o preenchimento dos respetivos formulários, de maneira a extrair os dados para a construção dos gráficos.

Quanto à importação e exportação de dados, as páginas encontram-se apresentadas na Figura 6.5 e Figura 6.6, respetivamente.

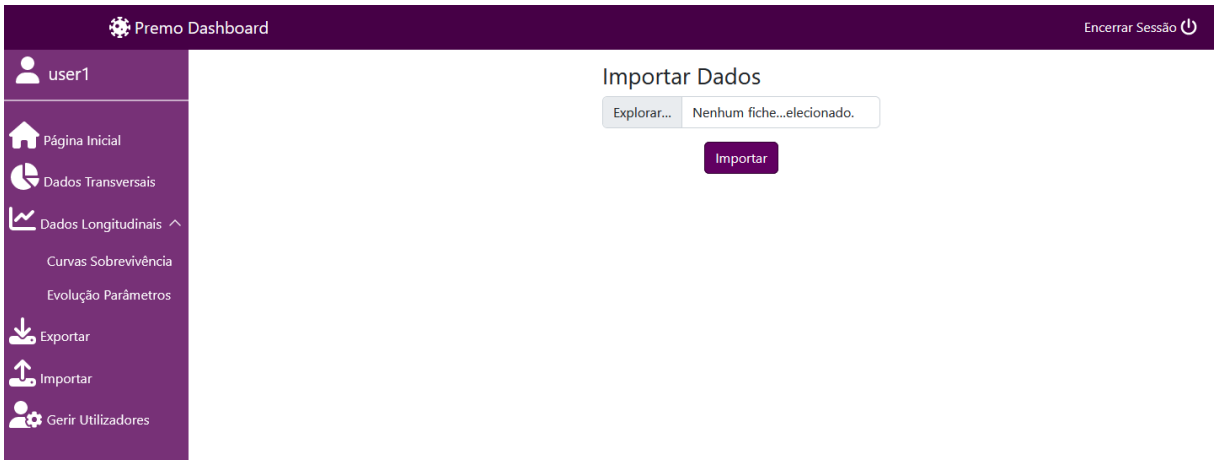


Figura 6.5: Página para importação de novos dados

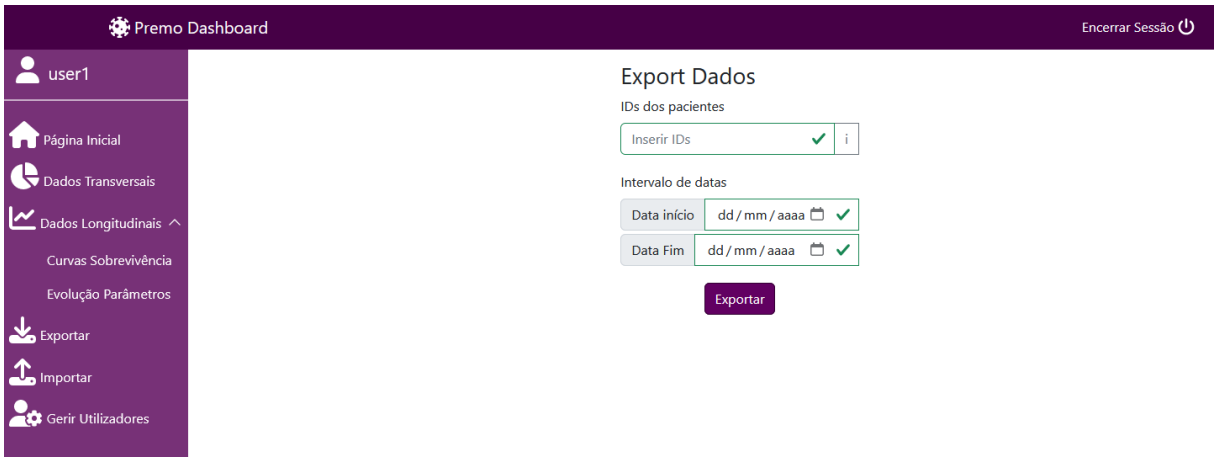


Figura 6.6: Página para exportação dos dados presentes

Por fim, existe a página para a gestão dos utilizadores, para os utilizadores que têm papel de 'admin'. Esta pode ser analisada na Figura 6.7.

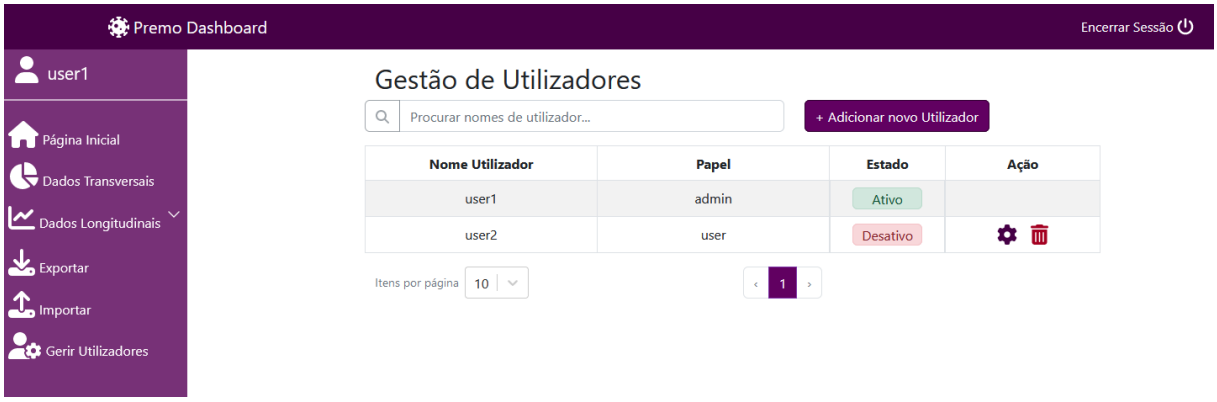


Figura 6.7: Página de gestão de utilizadores

6.2 Representações gráficas para os dados transversais

Os gráficos com dados transversais são aqueles que apresentam informação estatística relativa a dado momento no tempo, para variáveis do tipo quantitativo ou qualitativo.

O *dashboard* disponibiliza três tipos de gráficos com dados transversais: os circulares e barras sobre os quais se apresentam alguns detalhes na Subsecção 6.2.1, os diagramas em caixa que se encontram detalhados na Subsecção 6.2.2, e os diagramas de dispersão que são apresentados na Subsecção 6.2.3.

Para qualquer uma destas representações gráficas é possível visualizar os dados por vaga de COVID-19, escolhendo a opção para o respetivo seletor que se encontra no formulário.

6.2.1 Gráficos de barras e circulares

Os gráficos de barras e circulares permitem mostrar a frequência relativa de cada categoria de uma variável nominal. Na Figura 6.8 apresenta-se um exemplo um exemplo de um gráfico circular que indica a proporção de pacientes por género, e na Figura 6.9 é representado o mesmo mas num gráfico de barras. Na Figura 6.10 encontra-se um gráfico circular para cada vaga em separado, e na Figura 6.11 com gráficos de barras.

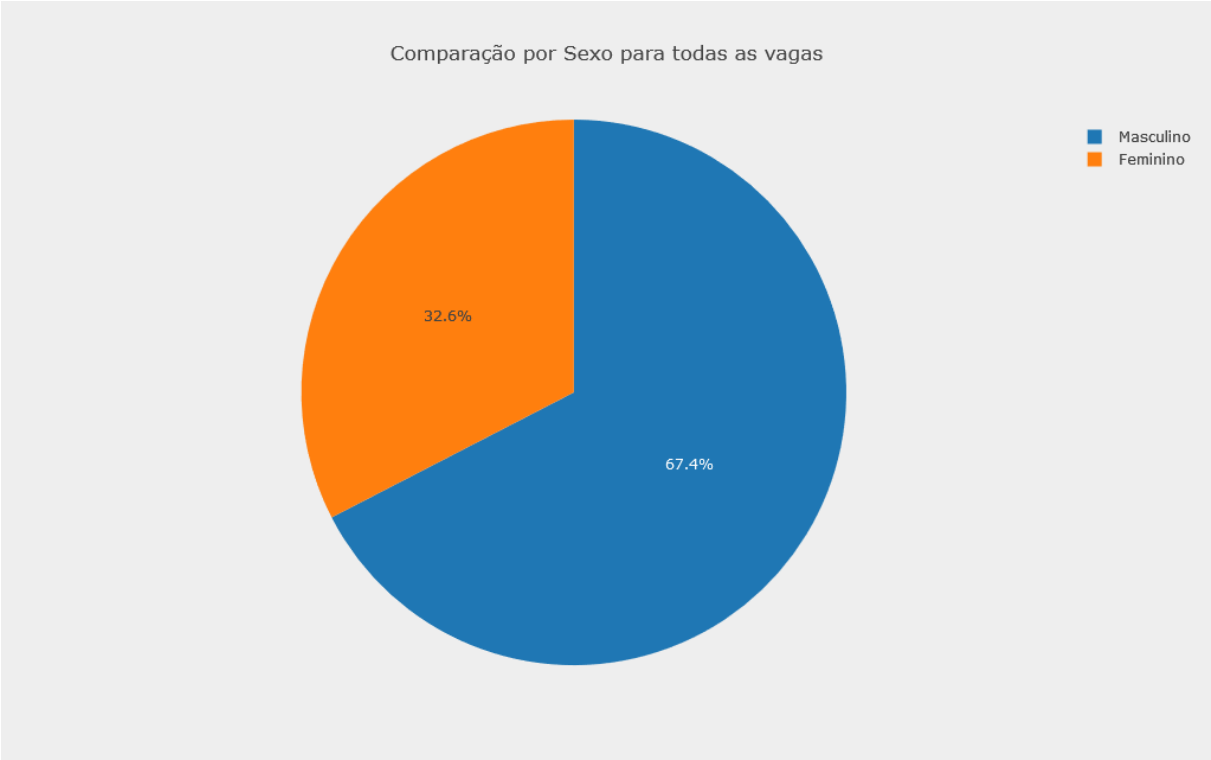


Figura 6.8: Gráfico circular com a percentagem de pacientes por género

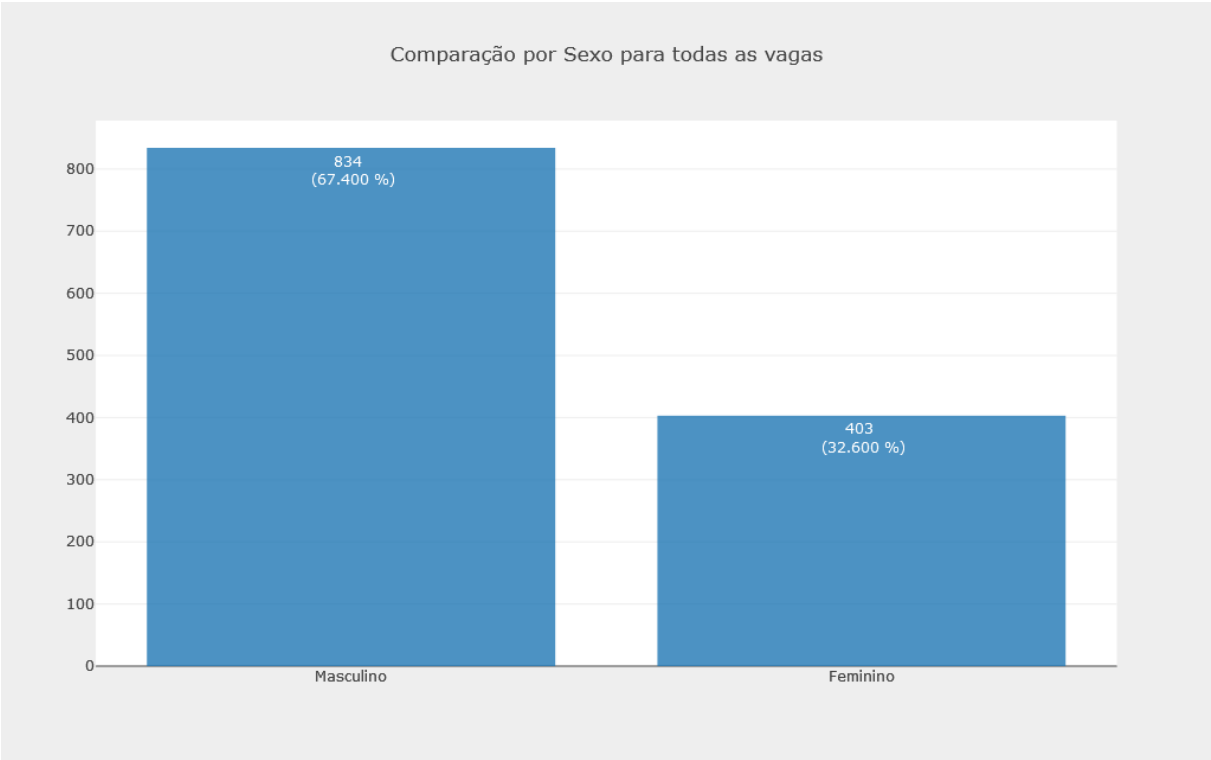


Figura 6.9: Gráfico de barras com a contagem e a percentagem de pacientes por género

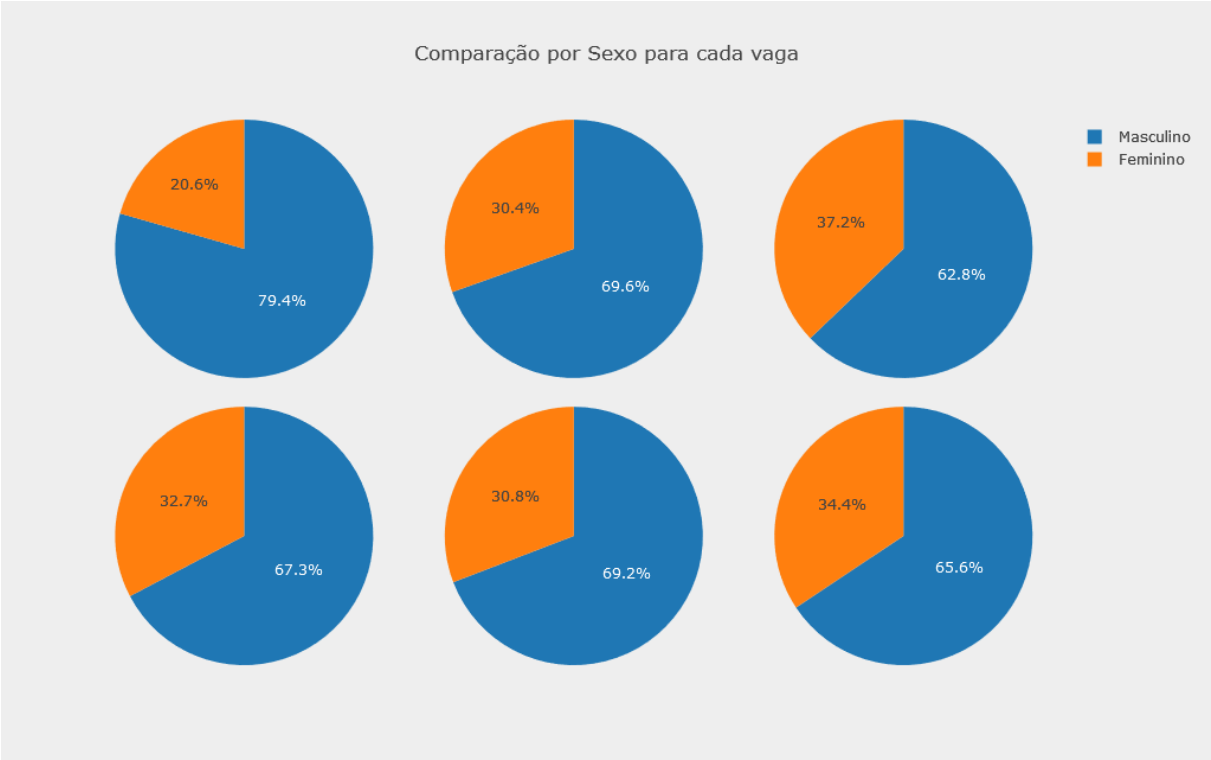


Figura 6.10: Gráficos circulares com a percentagem de pacientes por género, para cada vaga

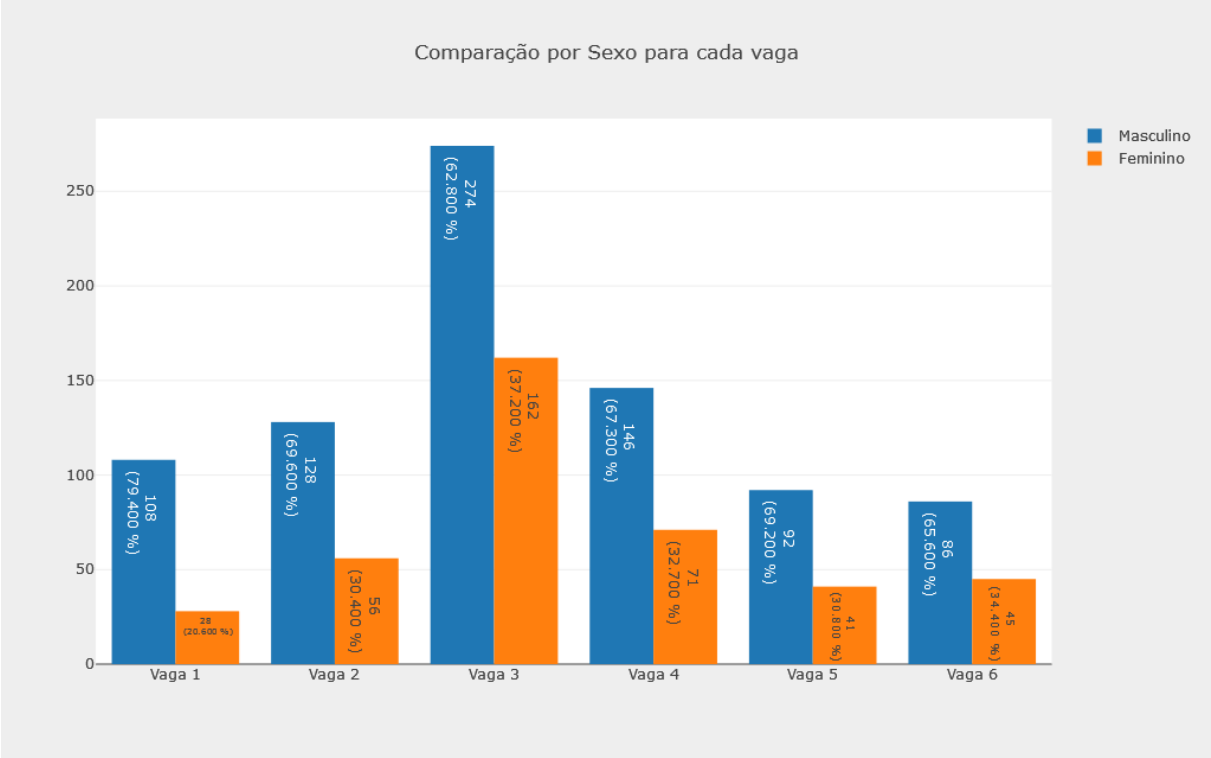


Figura 6.11: Gráficos de barras com contagem e a percentagem de pacientes por género, para cada vaga

Relativamente à exigência na conceção das diversas representações gráficas, estes gráficos são os mais simples uma vez que o único processamento necessário consiste na criação dos objetos para serem comunicados à biblioteca encarregue de fazer os desenhos. Apenas é necessária a soma dos valores quando são apresentadas todas as vagas em conjunto, uma vez que os dados utilizados (e recebidos da API) correspondem a frequências absolutas e relativas. Estes gráficos são produzidos quando no formulário se seleciona uma variável do tipo qualitativo nominal.

6.2.2 Diagramas em caixa

Os diagramas em caixa fornecem uma informação sobre a distribuição das observações para uma variável quantitativa, permitindo uma visualização resumida da variabilidade dos valores amostrais. Ao mover o cursor por cima do gráfico é possível visualizar também um resumo descritivo da variável em análise contendo a mediana, média, primeiro quartil, terceiro quartil, mínimo, máximo, desvio padrão e ainda valores atípicos, caso existam.

Outra funcionalidade disponível é a representação em paralelo de vários diagramas em caixa. A representação de vários diagramas em caixa permite realizar uma comparação entre diferentes grupos, tal como se pode observar no exemplo da Figura 6.12 no qual se observa a distribuição do nível de pH por género. Também é possível apresentar os Valor de probabilidades (valor-ps) relativos à aplicação de um teste de hipóteses para comparação de valores da distribuição do pH por género (Figura 6.13). E da mesma maneira que nos gráficos circulares, também neste caso é possível separar por vagas, tal como apresentado na Figura 6.14.

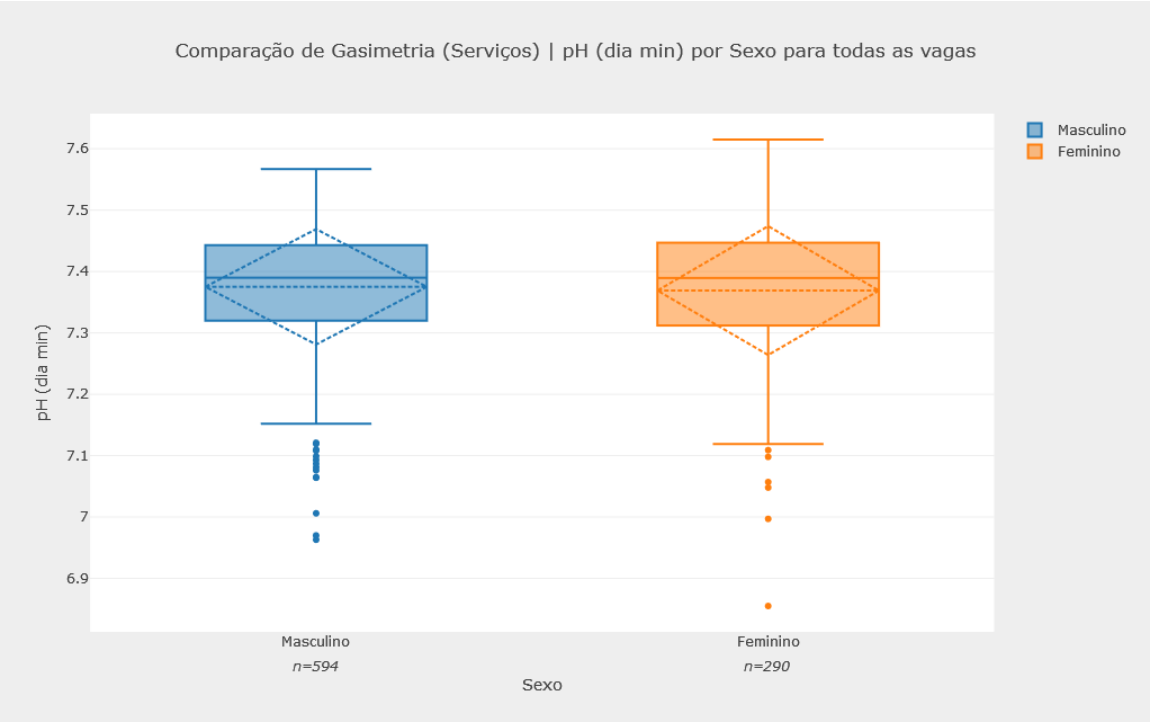


Figura 6.12: Diagramas em caixa para comparação da distribuição de pH por género, no primeiro dia na UCI

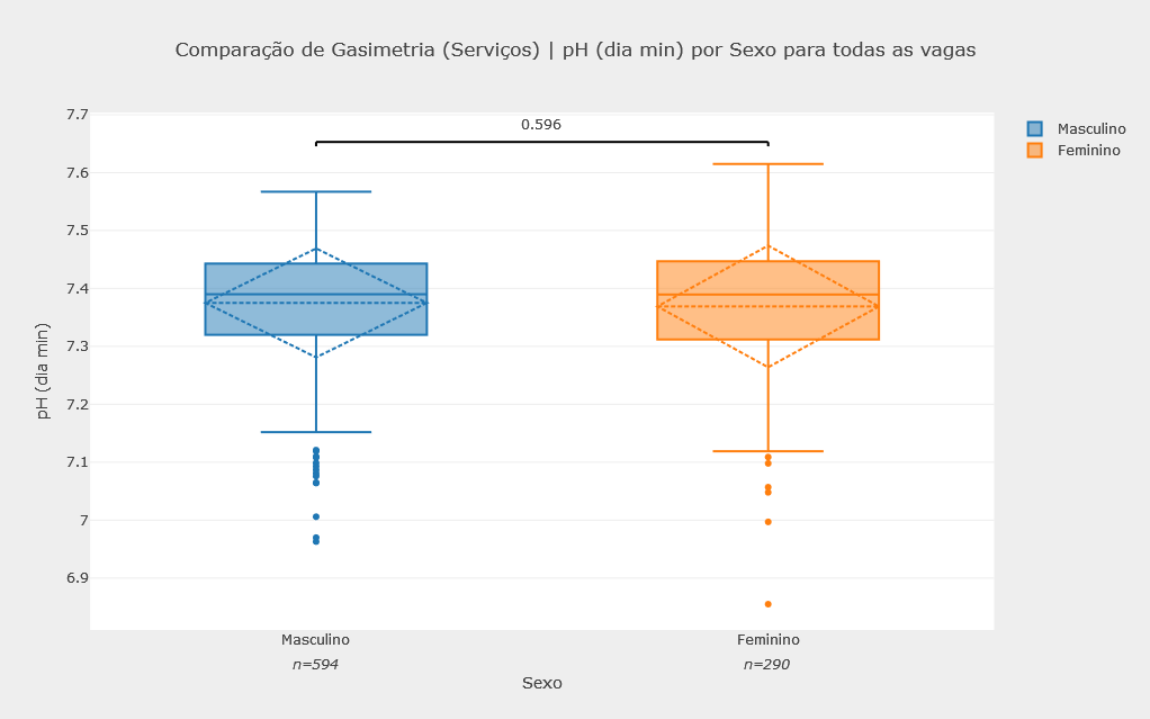


Figura 6.13: Diagramas em caixa para comparação da distribuição de pH por género com valor-p obtido através do teste de hipóteses de Mann-Whitney, no primeiro dia na UCI

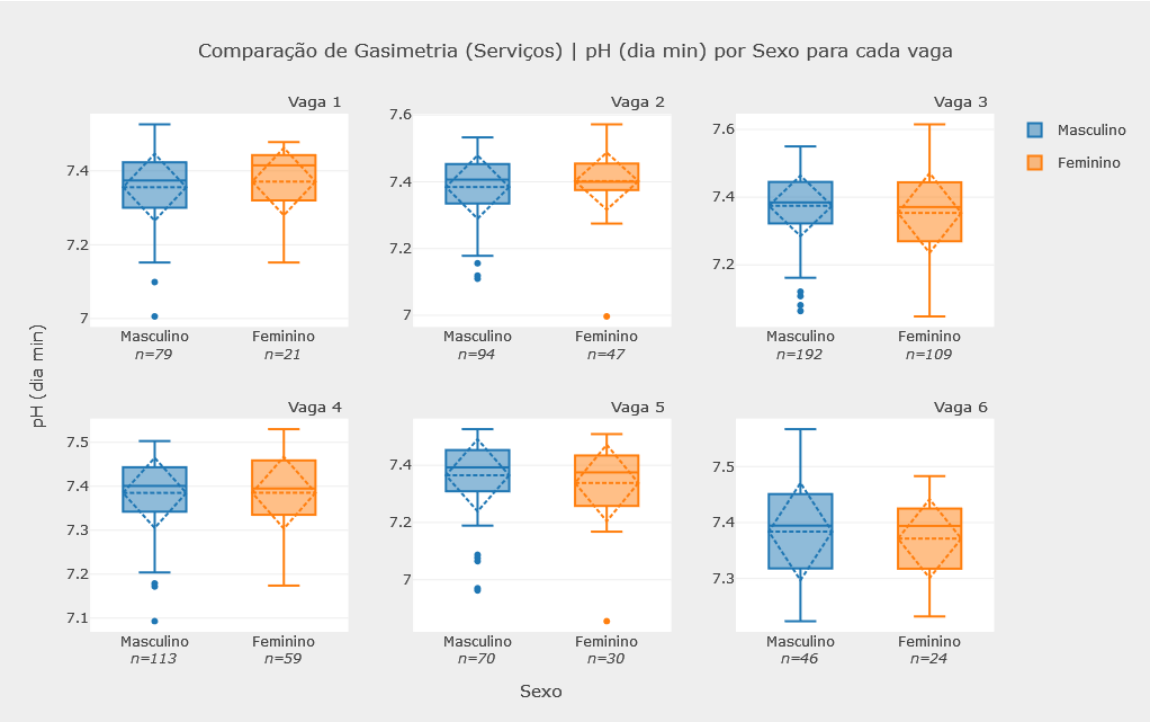


Figura 6.14: Diagramas em caixa para comparação da distribuição de pH por género, para cada vaga, no primeiro dia na UCI

A construção destes diagramas já apresentam uma maior complexidade no processamento necessário, quando comparado com o dos gráficos circulares, principalmente devido à necessidade de realizar os testes de hipóteses para comparação da distribuição dos dados para cada par de grupos. Adicionalmente são apresentados os valores-p de cada teste com linhas que permitem identificar quais os grupos que estão a ser comparados. Para o cálculo é utilizada uma biblioteca externa, e para a criação destas linhas é necessário criar retas e apresentar o respetivo valor-p. Este gráfico é gerado com a escolha de uma variável quantitativa e opcionalmente uma nominal. Quando o utilizador indica a variável nominal é gerado um diagrama de caixa relativo à distribuição da variável quantitativa em análise, para cada categoria da variável nominal.

6.2.3 Gráficos de dispersão

Os gráficos de dispersão, são construídos a partir de duas variáveis quantitativas e permitem visualizar a distribuição da nuvem de pontos correspondente às observações conjuntas da amostra bivariada, de acordo com a seleção do utilizador. Os gráficos ou diagramas de dispersão utilizam-se para analisar visualmente a existência de correlação entre as duas variáveis quantitativas. Um exemplo deste gráfico pode ser visualizado na Figura 6.15, onde é apresentada a relação entre os níveis de pH e de Lac, e na Figura 6.16 pode-se observar o mesmo separado por vagas.

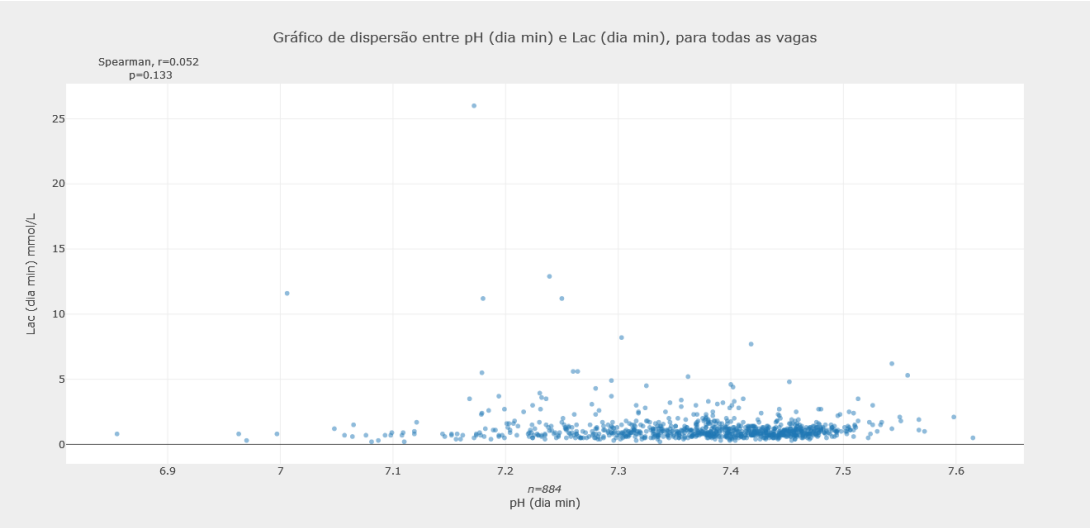


Figura 6.15: Gráfico de Dispersão para análise da correlação entre as variáveis pH e Lac (mmgo/L), no primeiro dia na UCI

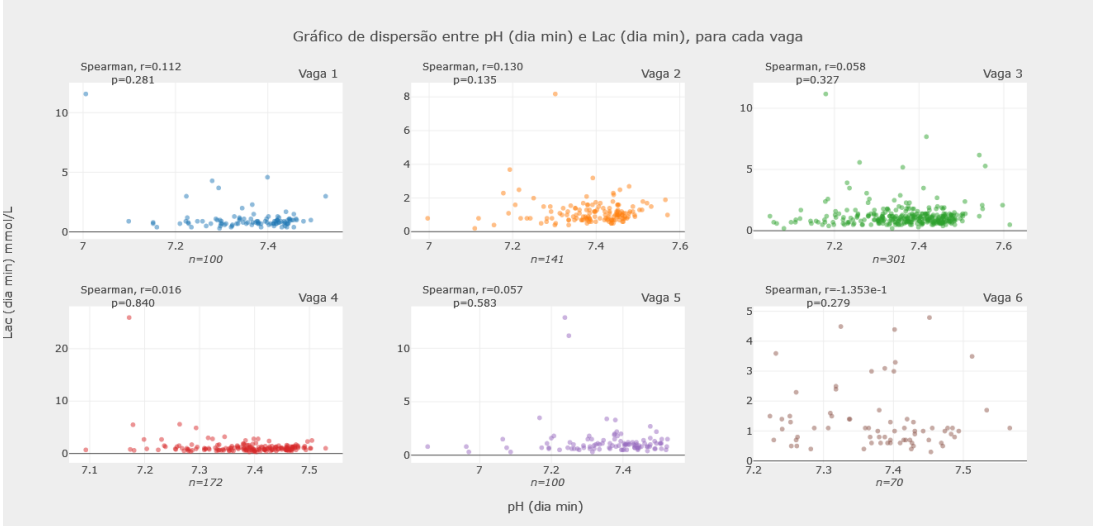


Figura 6.16: Gráficos de Dispersão para análise da correlação entre as variáveis pH e Lac (mmgo/L), no primeiro dia na UCI, por vaga

Estes gráficos têm um processamento idêntico ao dos gráficos circulares, sendo que nestes não são utilizadas contagens mas sim pontos. De modo a evitar que a geração do gráfico seja pesada demais devido ao eventual elevado número de pontos a serem gerados, estes são gerados a partir do motor WebGL do 'Plotly'.

6.3 Representações gráficas para os dados longitudinais

Os gráficos longitudinais permitem realizar uma visualização dos dados ao longo do tempo. Utilizaram-se dois tipos de gráficos longitudinais nesta solução: as curvas de sobrevivência de Kaplan Meier, que se encontram descritas na Subsecção 6.3.1; e os gráficos de linhas, descritos na Subsecção 6.3.2.

6.3.1 Curvas de sobrevivência

As curvas de sobrevivência são representações gráficas de funções de sobrevivência. Estas funções, por sua vez, estimam a probabilidade de um indivíduo sobreviver mais do que um determinado tempo.

No caso desta aplicação, os indivíduos em causa correspondem aos pacientes que se encontram internados na UCI, e o tempo é medido em dias passados na UCI. As curvas de sobrevivência utilizadas nesta aplicação baseiam-se no estimador de sobrevivência de Kaplan-Meier.

Na Figura 6.17, são apresentadas as curvas de sobrevivência onde os grupos de pacientes correspondem às duas classes de uma variável nominal, neste caso o género. Uma funcionalidade com grande interesse para a prática clínica e que também está disponível nesta aplicação é a visualização conjunta de curvas de sobrevivência para grupos de pacientes associados a padrões de risco. Esses padrões de risco são definidos por pontes de corte aplicados a variáveis quantitativas. Veja-se o exemplo apresentado na Figura 6.18.

Na Figura 6.19 é apresentado um gráfico com as curvas por grupo, onde cada grupo se caracteriza pela combinação de duas categorias. Essas categorias combinam duas variáveis nominais. Neste caso uma das variáveis é o género do paciente, e a outra variável foi obtida pela recodificação da variável quantitativa Mioglobina, usando um ponto de corte com interesse clínico. A aplicação permite a seleção de múltiplos pontos de corte de uma variável quantitativa e uma variável nominal.

Por fim, para cada situação existe sempre a possibilidade de realizar a separação das curvas por vagas, conforme apresentado na Figura 6.20, que representa as várias curvas para as diferentes vagas, onde os pacientes se encontram divididos por género.

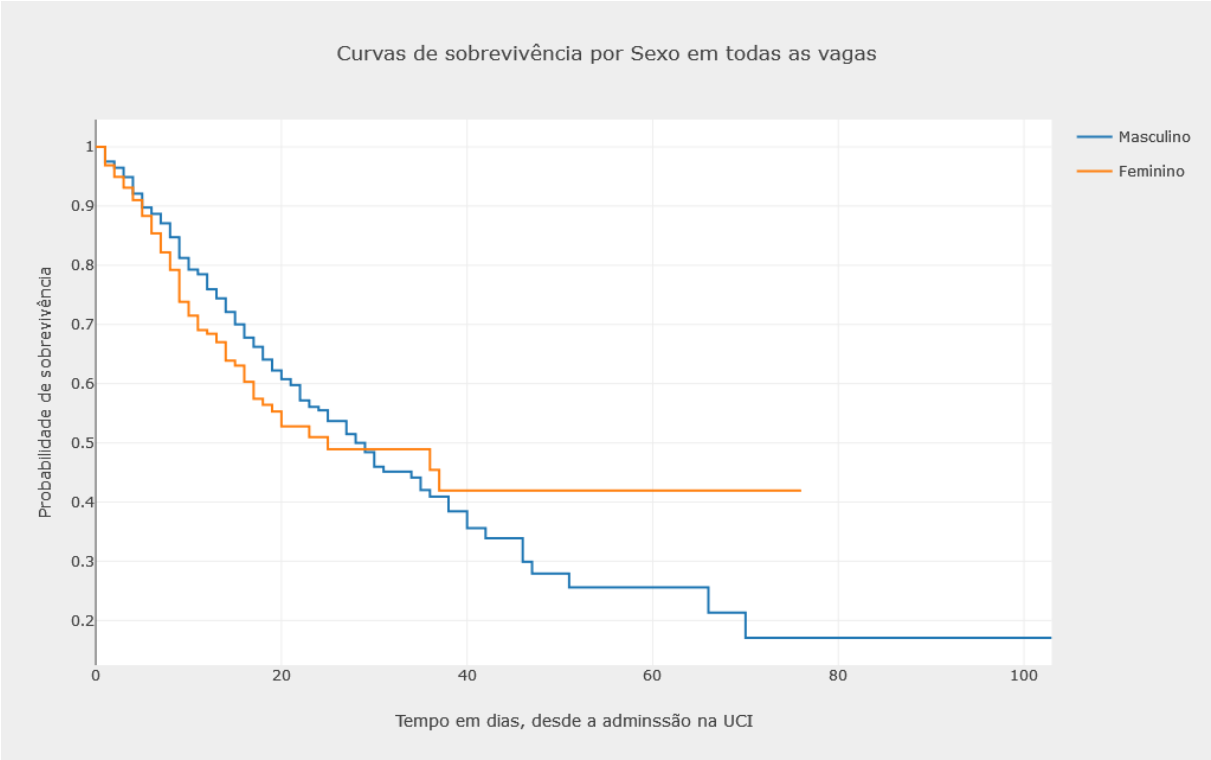


Figura 6.17: Curvas de sobrevivência de Kaplan Meier, por género

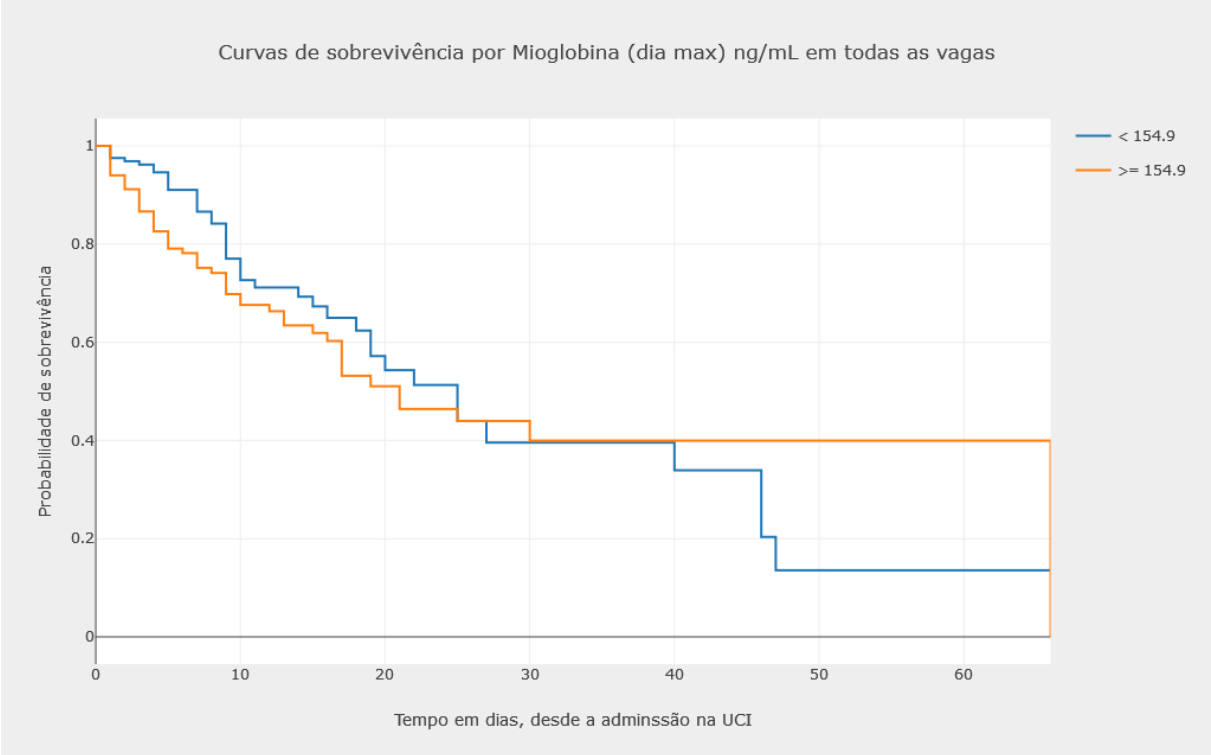


Figura 6.18: Curvas de sobrevivência de Kaplan Meier, com ponto de corte na Mioglobina

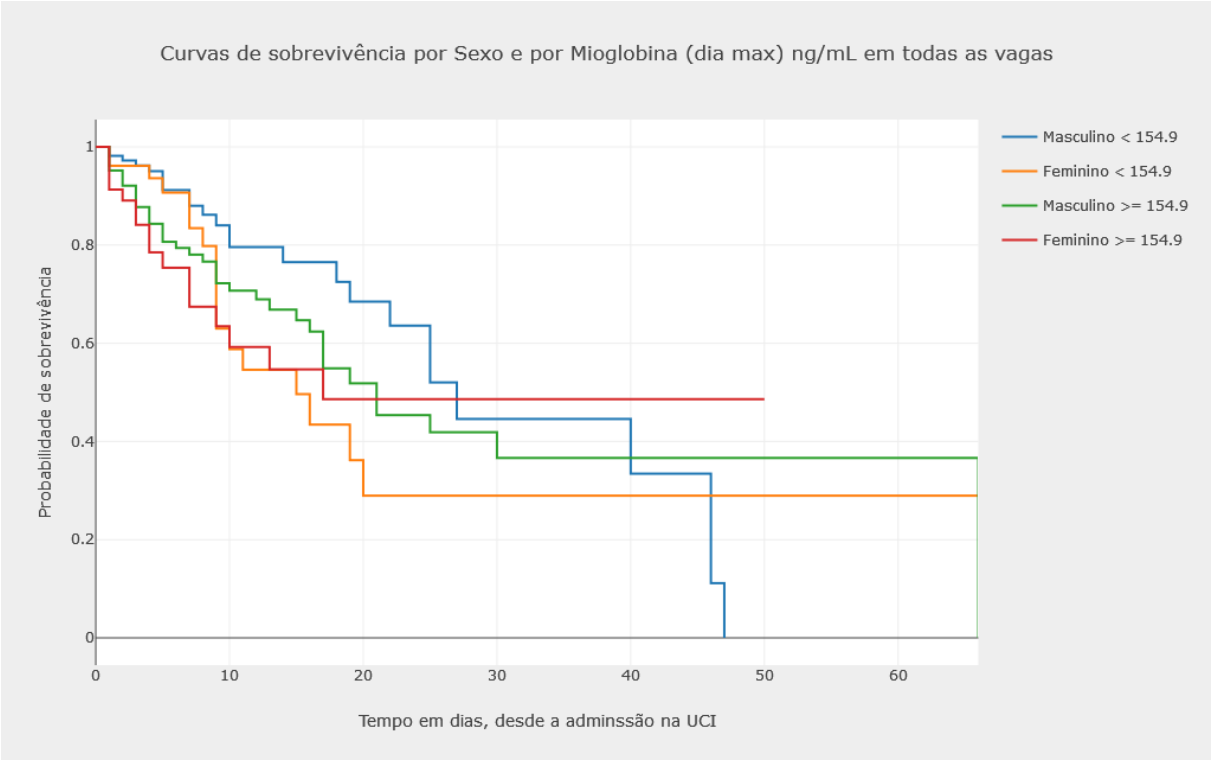


Figura 6.19: Curvas de sobrevivência de Kaplan Meier, por género com ponto de corte na Mioglobina

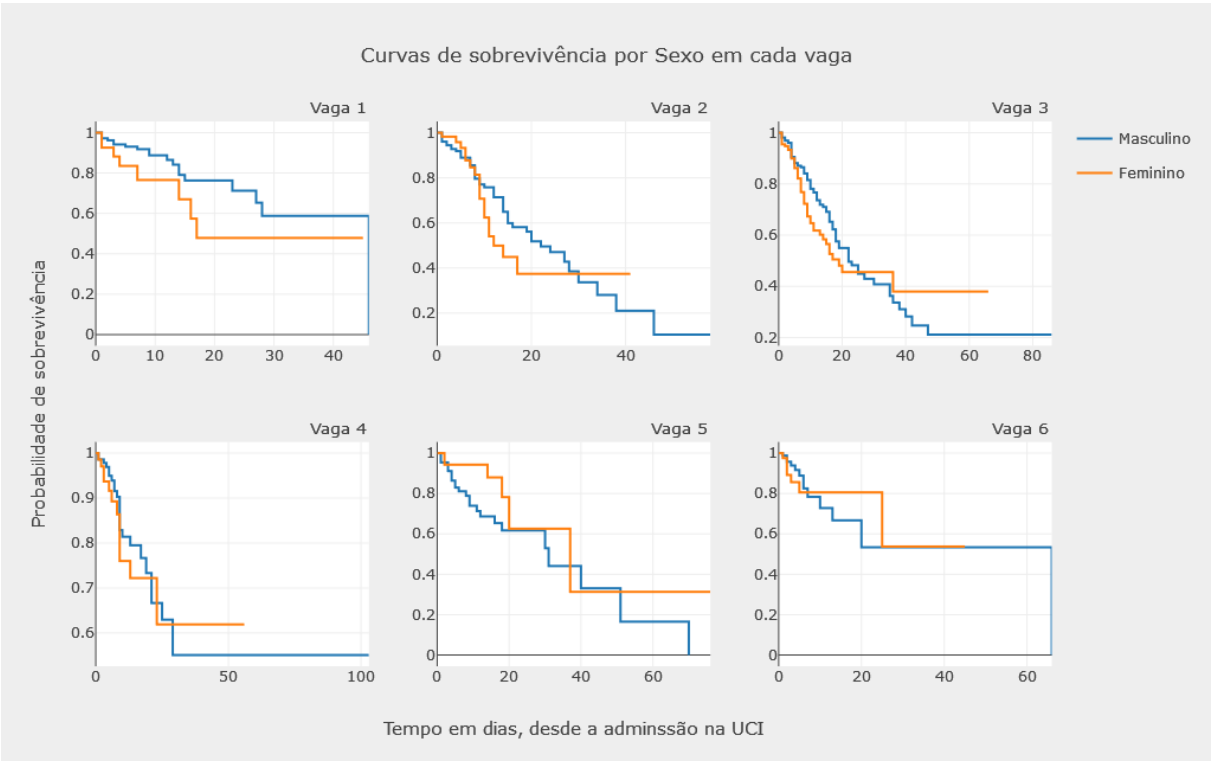


Figura 6.20: Curvas de sobrevivência de Kaplan Meier para cada vaga, por género

Visto não se encontrarem disponíveis bibliotecas para a geração destas curvas de sobrevivência em JavaScript como as existentes no Python, são geradas na API e não na interface do utilizador. Em relação à divisão dos vários grupos de pacientes, esta é realizada através da variável nominal (das categorias que possui) e/ou pelos pontos de corte referentes ao primeiro dia de internamento na UCI. Os respetivos testes para realizar a comparação das várias curvas são também geradas na API. Uma vez que os pontos das curvas se encontram na interface de utilizador, é gerado um gráfico de linhas, com a distinção que apenas seguem uma orientação horizontal e posteriormente vertical.

6.3.2 Gráficos de linhas

Os gráficos de linhas utilizam linhas para unir os diversos pontos. Estes gráficos são utilizados para apresentar a evolução de alguma variável (eixo da ordenada) consoante o valor no eixo das abcissas. No caso desta aplicação, estes gráficos são utilizados para representar a evolução de um dado parâmetro para os diferentes pacientes, ao longo do período de permanência na UCI.

Como já foi indicado na Secção 4.5, existem duas variantes para estes gráficos: uma primeira que permite analisar a evolução de um parâmetro para diferentes pacientes (Figura 6.21); e uma segunda que permite a análise de todos agregados, apresentada na Figura 6.21, e a respetiva separação por vaga, na Figura 6.23.

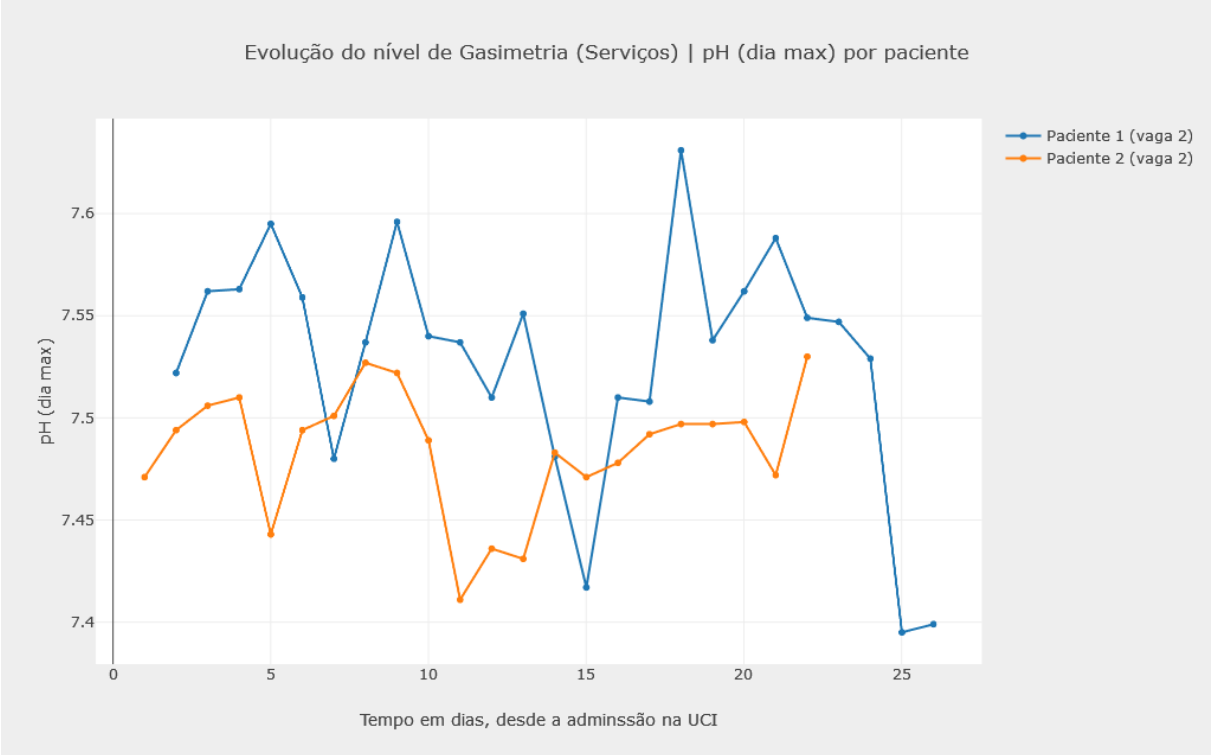


Figura 6.21: Gráfico de pontos com a evolução temporal de pH (máximo diário), para dois pacientes

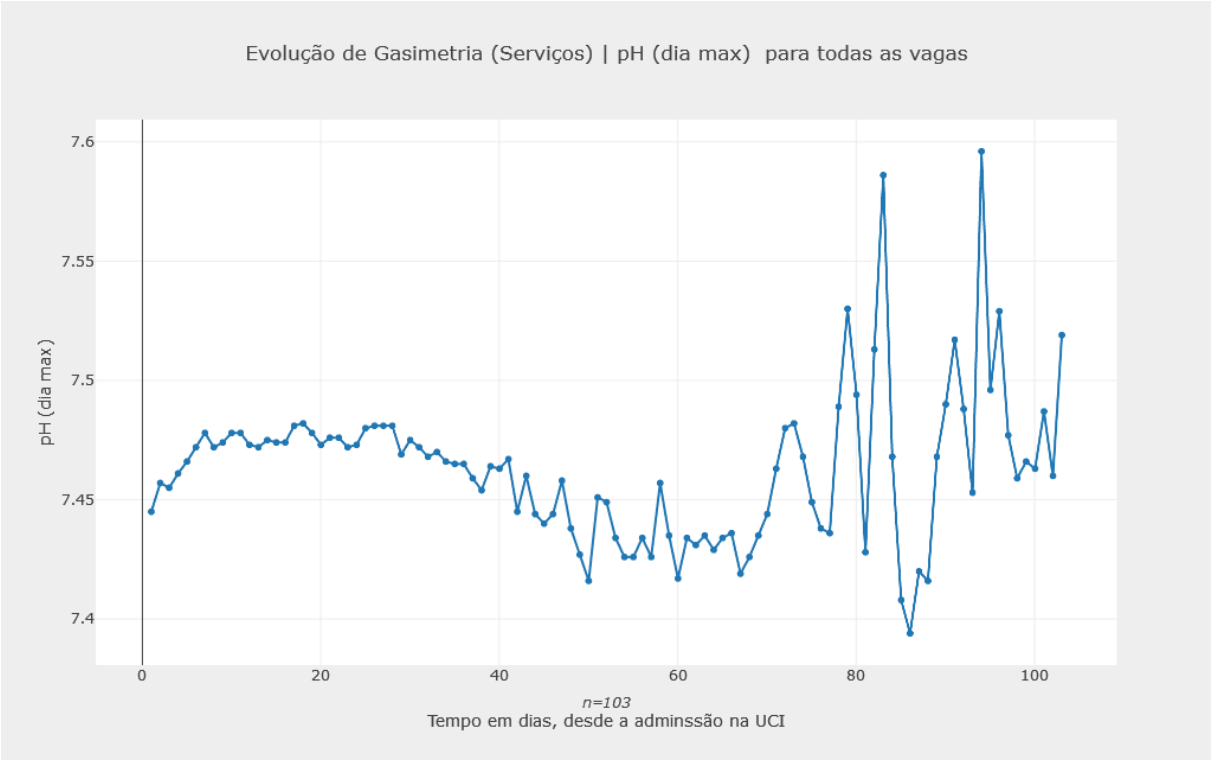


Figura 6.22: Gráfico de pontos com a evolução temporal de pH (máximo diário) com os pacientes agregados

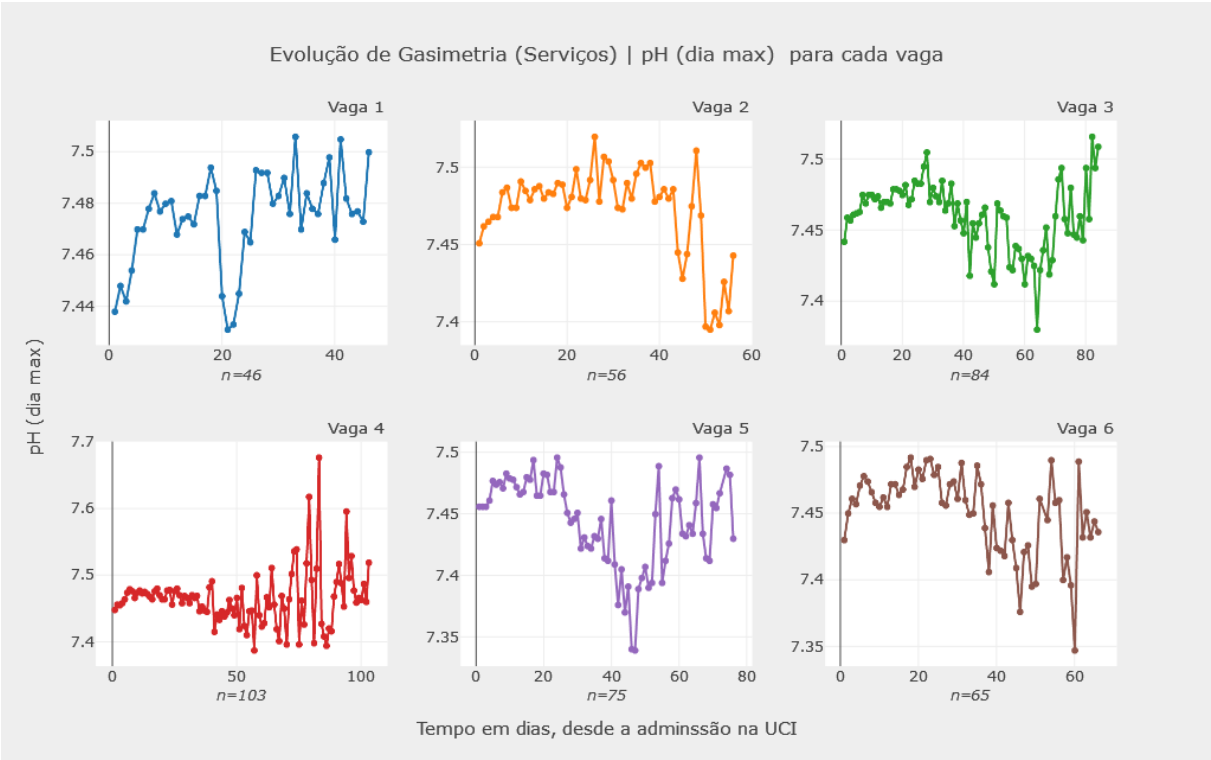


Figura 6.23: Gráfico de pontos com a evolução temporal de pH (máximo diário) com os pacientes agregados, por vaga

Estes gráficos exigem um maior nível de processamento, visto ser necessário que cada paciente tenha apenas um valor por dia de internamento na UCI. No caso de ser utilizada uma variável única diária então é usado o respetivo valor, senão é calculada a mediana dos valores que cada paciente teve em cada dia de internamento na UCI.

7

Conclusões e trabalho futuro

Neste capítulo são apresentadas as conclusões da solução desenvolvida na Secção 7.1, destacando o cumprimento dos vários objetivos. Na Secção 7.2, são apresentadas as principais ideias para desenvolver ou melhorar as existentes.

7.1 Conclusões

A COVID-19 provocou um período de pandemia, devido à sua facilidade de transmissão e elevado número de casos de infeção. Esta doença teve severas consequências na mortalidade e morbilidade das populações, sobretudo nas mais idosas.

A solução apresentada nesta tese, consiste na criação de um *dashboard* acessível por um *browser web*. Este permite aos clínicos visualizarem os dados a partir de diversos gráficos e diagramas, de forma a promover decisões médicas mais rápidas e precisas, para a diminuição de eventos graves, recuperações mais rápidas e uma redução de fatalidades.

Os dados processados são relativos a pacientes internados na UCI ao longo das primeiras seis vagas de COVID-19 em Portugal. Existem dois tipos de dados - os DL que são longitudinais, e os DCD que são transversais.

A solução apresentada consiste numa interface de utilizador (por *browser*), construída através da biblioteca React, onde são disponibilizados os diversos gráficos e diagramas criados com a biblioteca Plotly em JavaScript. Esta interface comunica com uma API realizada em Python construída sobre a *framework* FastAPI, de forma a extrair os dados necessários da base dados. Estas três partes seguem uma arquitetura de três camadas.

De forma a permitir a importação de novos dados enquanto a aplicação se encontra em execução, há uma comunicação assíncrona entre a API e o processo de ETL através de uma fila de mensagens RabbitMQ, usando também uma base de dados não relacional para o envio dos dados em bruto para processamento.

Existe um processo ETL, para cada tipo de dados, de forma a tratá-los através da remoção de colunas desnecessárias, observações em duplicado, ou valores que não fornecem qualquer informação.

Para os DL foi usado um trabalho já previamente realizado por [55] que permite fazer a importação destes para uma base de dados relacional, realizando já algum tratamento sobre os dados. Após esta importação é realizado o respetivo processo ETL que, em suma, consiste na seleção de subconjunto dos dados, a transformação destes para valores coerentes e remoção de tuplos sem informação, e por fim, na agregação de parâmetros e os seus respetivos valores.

No caso dos DCD foi criado um modelo ER para suportar estes dados na base de dados, e realizada a importação dos dados. Não houve necessidade de tratamento, visto os dados já se apresentarem no formato pretendido.

As diferentes partes que resultam na aplicação, encontram-se disponíveis através de contentores Docker. É também utilizado um *reverse-proxy* de forma a redirecionar os pedidos para o servidor *web* e para a API.

A solução permite a visualização de dados através de gráficos e diagramas com o preenchimento de formulários que permitem a filtragem dos dados. Os gráficos e diagramas disponíveis são:

- gráficos de barras e circulares - de forma a analisar as frequências de diferentes grupos de pacientes;
- diagramas em caixa - de forma a visualizar informação resumida da variabilidade de valores de um determinado grupo;
- gráficos de dispersão - de maneira a correlacionar duas variáveis quantitativas;
- curvas de sobrevivência - que estimam a probabilidade de sobrevivência de um indivíduo sobreviver mais do que um determinado tempo;
- gráficos de linhas - para analisar a evolução de um parâmetro, dos vários pacientes ou de todos em geral.

Todos estes gráficos e diagramas, com a exceção dos gráficos de linhas quando se trata dos vários pacientes, permitem a separação da visualização em vários gráficos, onde cada corresponde a uma vaga.

Para além da apresentação dos gráficos e diagramas, a solução também permite exportar os dados dos pacientes num ficheiro CSV comprimido, podendo utilizar

filtros como os identificadores dos pacientes a exportar ou as datas das colheitas. Também se encontra disponível a importação de novos dados enquanto a aplicação se encontra em execução. Por último, os utilizadores com o papel 'admin' têm acesso à gestão de utilizadores, como a alteração do papel, desativar/ativar o utilizador ou reiniciar a senha.

7.2 Trabalho futuro

A solução apresentada ainda pode sofrer melhorias das funcionalidades existentes, bem como adicionar novas funcionalidades.

Destas, uma funcionalidade a acrescentar seria a edição dos ficheiros de configuração a partir do *dashboard*, como:

- o ficheiro que contém os parâmetros a analisar, apresentado na Listagem 3.1;
- o que contém as regras de transformação dos parâmetros, apresentado na Listagem 3.2;
- e, por último, o ficheiro que estabelece como categorizar os parâmetros, como na Listagem 3.3.

Outro aspeto a desenvolver passa por adicionar na base de dados, a informação relativa à terapêutica. Estes, são também vários ficheiros Excel como os DL, onde cada um se refere a um paciente, cujo nome é o seu identificador. Estes ficheiros contêm os medicamentos administrados pelo paciente, a sua data de início e a data de fim.

A utilização de modelos preditivos sobre os dados existentes, já transformados pelo processo ETL, de forma a dar sugestões aos utilizados é outro aspeto a considerar.

Referências

[1] D. Florensa, J. Mateo, C. Miret, s. Godoy, F.Solsona & P. Godoy, "Severity of COVID-19 cases in the months of predominance of the Alpha and Delta variants", mar. de 2023.

[2] Eva Sarafinou. "Secure Browser Storage: The Facts". (), URL: https://auth0.com/blog/secure-browser-storage-the-facts/ (acedido em 05/03/2023).

[3] AWS. "Three-tier architecture overview". (), URL: https://docs.aws.amazon.com/whitepapers/latest/serverless-multi-tier-architectures-api-gateway-lambda/three-tier-architecture-overview.html (acedido em 15/02/2023).

[4] Stefano Barone, Alexander Chakhunashvili & Albert Comelli, "Building a statistical surveillance dashboard for COVID-19 infection worldwide", *Quality Engineering*, vol. 32, n.º 4, páginas 754–763, 2020.

[5] C3. "C3.js D3-based reusable chart library". (), URL: https://c3js.org/ (acedido em 12/02/2023).

[6] Chart.js. "Chart.js - Simple yet flexible JavaScript charting library for the modern web". (), URL: https://www.chartjs.org/ (acedido em 12/02/2023).

[7] MDN. "Using HTTP cookies". (), URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies#restrict_access_to_cookies (acedido em 05/03/2023).

[8] CrypTool. "CrypTool-Online - Cryptography for everybody". (), URL: https://www.cryptool.org/en/cto/openssl (acedido em 19/02/2023).

[9] D3. "Data-Driven Documents". (), URL: https://d3js.org/ (acedido em 11/02/2023).

[10] Wikipedia. "Dashboard". (), URL: https://en.wikipedia.org/wiki/Dashboard (acedido em 03/09/2023).

[11] David Collet, *Modelling Survival Data in Medical Research, Third Edition*. Chapman & Hall/CRC, 2014. doi: https://doi.org/10.1201/b18041.

[12] django. “django - The web framework for perfectionists with deadlines”. (), URL: https://www.djangoproject.com/ (acedido em 14/02/2023).

[13] Docker. “Docker”. (), URL: https://www.docker.com/ (acedido em 28/06/2023).

[14] Docker. “What is a Container? | Docker”. (), URL: https://www.docker.com/resources/what-container (acedido em 29/06/2023).

[15] Nginx. “nginx - Official Image | Docker Hub”. (), URL: https://hub.docker.com/_/nginx (acedido em 28/06/2023).

[16] Ensheng Dong, Jeremy Ratcliff, Tamara D Goyea, Aaron Katz, Ryan Lau, Timothy K Ng, Beatrice Garcia, Evan Bolt, Sarah Prata, David Zhang et al., “The Johns Hopkins University Center for Systems Science and Engineering COVID-19 Dashboard: data collection process, challenges faced, and lessons learned”, *The lancet infectious diseases*, vol. 22, n.º 12, e370–e376, 2022.

[17] FastAPI. “FastAPI - FastAPI framework, high performance, easy to learn, fast to code, ready for production”. (), URL: https://fastapi.tiangolo.com/ (acedido em 14/02/2023).

[18] Fask. “Flask - web development, one drop at a time”. (), URL: https://flask.palletsprojects.com/en/2.2.x/ (acedido em 14/02/2023).

[19] Delwen L Franzen, Benjamin Gregory Carlisle, Maia Salholz-Hillel, Nico Riedel & Daniel Strech, “Institutional dashboards on clinical trial transparency for University Medical Centers: A case study”, *Plos Medicine*, vol. 20, n.º 3, e1004175, 2023.

[20] Google. “App Engine Application Platform”. (), URL: https://cloud.google.com/appengine (acedido em 28/06/2023).

[21] Jakub Majorek. “19 Best JavaScript Data Visualization Libraries”. (), URL: https://www.monterail.com/blog/javascript-libraries-data-visualization (acedido em 11/02/2023).

[22] Software Testing Help. “Top 15 JavaScript Visualization Libraries”. (), URL: https://www.softwaretestinghelp.com/best-javascript-visualization-libraries/ (acedido em 11/02/2023).

[23] FusionCharts. “Top 20 JavaScript Libraries For Data Visualization For 2022”. (), URL: https://www.fusioncharts.com/blog/top-20-javascript-libraries-for-data-visualization-for-2022/ (acedido em 11/02/2023).

[24] Duomly. “25 the Best Javascript Data Visualization Libraries in 2022”. (), URL: https://www.blog.duomly.com/javascript-data-visualization-libraries/ (acedido em 11/02/2023).

[25] cube awesome tools. “Data visualization tools for application developers”. (), URL: https://awesome.cube.dev/?frameworks=react&licenses=open-source (acedido em 11/02/2023).

[26] Heroku. “Heroku”. (), URL: https://www.heroku.com/ (acedido em 28/06/2023).

[27] IBM. “What is three-tier architecture?” (), URL: https://www.ibm.com/topics/three-tier-architecture (acedido em 15/02/2023).

[28] IBM. “What is Docker”. (), URL: https://www.ibm.com/topics/docker (acedido em 28/06/2023).

[29] Halah Ibrahim, Sara Sorrell, Satish Chandrasekhar Nair, Ahmed Al Romaithi, Shamma Al Mazrouei & Ashraf Kamour, “Rapid development and utilization of a clinical intelligence dashboard for frontline clinicians to optimize critical resources during COVID-19”, *Acta Informatica Medica*, vol. 28, n.º 3, pág. 209, 2020.

[30] Rozhin Amin, Mohammad-Reza Sohrabi & Khatereh Hannani, “Five consecutive epidemiological waves of COVID-19: a population-based cross-sectional study on characteristics, policies, and health outcome”, mar. de 2023.

[31] Randeep S Jawa, Mathew A Tharakan, Chaowei Tsai, Victor L Garcia, James A Vosswinkel, Daniel N Rutigliano, Jerry A Rubano & Stony Brook Medicine Enterprise Analytics Team, “A reference guide to rapidly implementing an institutional dashboard for resource allocation and oversight during COVID-19 pandemic surge”, *JAMIA open*, vol. 3, n.º 4, páginas 518–522, 2020.

[32] stdlib. “kruskalTest”. (), URL: https://www.npmjs.com/package/@stdlib/stats-kruskal-test (acedido em 26/09/2023).

[33] Let’s Encrypt. “Let’s Encrypt”. (), URL: https://letsencrypt.org (acedido em 02/07/2023).

[34] Dr. Chander Yadav & Amit Sharma, “National Institute of Malaria Research-Malaria Dashboard (NIMR-MDB): A digital platform for analysis and visualization of epidemiological data”, *The Lancet Regional Health - Southeast Asia*, vol. 5, pág. 100 030, jul. de 2022. doi: 10.1016/j.lansea.2022.100030.

[35] gungorbudak. “Mann Whitney U test Javascript implementation”. (), URL: https://gist.github.com/gungorbudak/1c3989cc26b9567c6e50 (acedido em 26/09/2023).

[36] MariaDB. “Data Type Storage Requirements”. (), URL: https://mariadb.com/kb/en/data-type-storage-requirements/ (acedido em 05/02/2023).

[37] mozilla. “JavaScript | MDN”. (), URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript (acedido em 10/02/2023).

[38] Nginx. “Nginx”. (), URL: https://www.nginx.com/ (acedido em 28/06/2023).

[39] Vinay Sharma, Oscar Cassetti, Lewis Winning, Michael O’Sullivan & Michael Crowe, “Protocol for developing a dashboard for interactive cohort analysis of oral health-related data”, *BMC Oral Health*, vol. 23, abr. de 2023. doi: 10.1186/s12903-023-02895-2.

[40] OWASP. “Password Storage Cheat Sheet”. (), URL: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html (acedido em 19/02/2023).

[41] PassLib. “PassLib documentation”. (), URL: https://passlib.readthedocs.io/en/stable/index.html#passlib-release-documentation (acedido em 19/02/2023).

[42] Password Hashing Competition. “Password Hashing Competition”. (), URL: https://www.password-hashing.net/ (acedido em 19/02/2023).

[43] plotly. “Plotly - Javascript”. (), URL: https://plotly.com/javascript/ (acedido em 20/07/2023).

[44] Marcelo Ponce & Amit Sandhel, “covid19. analytics: An R package to obtain, analyze and visualize data from the Coronavirus disease pandemic”, *arXiv preprint arXiv:2009.01091*, 2020.

[45] Pyramid. “Pyramid - The Start Small, Finish Big Stay Finished Framework”. (), URL: https://trypyramid.com/ (acedido em 14/02/2023).

[46] Shyam Purkayastha. “Top 15 Python REST API Frameworks in 2022”. (), URL: https://rapidapi.com/blog/best-python-api-frameworks/ (acedido em 14/02/2023).

[47] Preet Kaur. “Top 5 Python REST API Frameworks”. (), URL: https://www.moesif.com/blog/api-product-management/api-analytics/Top-5-Python-REST-API-Frameworks/ (acedido em 14/02/2023).

[48] Samuel Nicholas. “Best Python REST API Framework Solutions for 2023”. (), URL: https://hevodata.com/learn/python-rest-api-framework/ (acedido em 14/02/2023).

[49] Mary Manzi. “How to build APIs in Python: 8 popular frameworks”. (), URL: https://www.techrepublic.com/article/build-apis-python/ (acedido em 14/02/2023).

[50] React. “React - A JavaScript library for building user interfaces”. (), URL: https://reactjs.org/ (acedido em 10/02/2023).

[51] React Select. “React Select”. (), URL: https://react-select.com/home (acedido em 05/07/2023).

[52] Recharts. “Recharts - A composable charting library built on React components”. (), URL: https://recharts.org/en-US/ (acedido em 12/02/2023).

[53] Cristiana da Palma Von Rekowski, *Development of Predictive Models for COVID-19 Prognosis based on Patients’ Demographic and Clinical Data*, out. de 2023.

[54] “The OAuth 2.0 Authorization Framework - Access Token”. (), URL: https://www.rfc-editor.org/rfc/rfc6749#section-1.4 (acedido em 19/02/2023).

[55] Diogo Filipe Ricardo Ribeiro, “Relatório do Bolseiro”, Instituto Superior de Engenharia de Lisboa, rel. téc., out. de 2022.

[56] serve. “Serve - npm”. (), URL: https://www.npmjs.com/package/serve (acedido em 28/06/2023).

[57] SNS. “SNS24 - COVID-19”. (), URL: https://www.sns24.gov.pt/tema/doencas-infecciosas/covid-19/#o-que-e-a-covid-19 (acedido em 20/02/2023).

[58] statistics.js. “statistiscs.js - Documentation”. (), URL: https://thisancog.github.io/statistics.js/ (acedido em 26/09/2023).

[59] Apache. “Apache Tomcat”. (), URL: https://tomcat.apache.org/ (acedido em 28/06/2023).

[60] Sruthi Veeraraghavan. “20 Most Popular Programming Languages to Learn in 2023”. (), URL: https://www.simplilearn.com/best-programming-languages-start-learning-today-article (acedido em 10/02/2023).

[61] David Yang. “The 9 Best Programming Languages to Learn in 2023”. (), URL: https://www.fullstackacademy.com/blog/nine-best-programming-languages-to-learn (acedido em 10/02/2023).

[62] Traefik. “Traefik Labs: Say Goodbye to Connectivity Chaos”. (), URL: https://traefik.io/ (acedido em 02/07/2023).

[63] Typescript. “TypeScript is JavaScript with syntax for types.” (), URL: https://www.typescriptlang.org/ (acedido em 10/02/2023).

[64] Victory. “Victory - React.js components for modular charting and data visualization.” (), URL: https://formidable.com/open-source/victory/ (acedido em 12/02/2023).

[65] VMware. “What is Application Deployment”. (), URL: https://www.vmware.com/topics/glossary/content/application-deployment.html (acedido em 28/06/2023).

[66] Steve Wexler, Jeffrey Shaffer & Andy Cotgreave, *The big book of dashboards: visualizing your data using real-world business scenarios*. John Wiley & Sons, 2017.

[67] Benjamin D Wissel, PJ Van Camp, Michal Kouril, Chad Weis, Tracy A Glauser, Peter S White, Isaac S Kohane & Judith W Dexheimer, “An interactive online dashboard for tracking COVID-19 in US counties, cities, and states in real time”, *Journal of the American Medical Informatics Association*, vol. 27, n.º 7, páginas 1121–1125, 2020.