



Plataforma de BI (Transplante de Córnea)

Mestrado em Engenharia Biomédica – Informática Médica

Aplicações Informáticas em Engenharia Biomédica
Processo Clínico Eletrónico

2º Semestre 2022/2023

Diogo Guedes Lameira, PG50332
Joana Maria Freitas Fernandes, PG50459
Maria da Conceição Vieira Mota, PG51210

Índice

1	Introdução.....	3
2	Implementação	4
2.1	Arquitetura.....	4
2.2	<i>Login</i>	5
2.3	Modelação do formulário.....	7
2.3.1	<i>Archetype Designer</i>	7
2.3.1	Criação do Modelo de Base de Dados.....	8
2.3.2	Criação de Rotas Relacionadas com <i>Compositions</i>	9
2.3.3	Página Web para Renderizar a Composição do Formulário.....	10
	Validação do formulário	10
2.4	Listagem de propostas.....	11
2.5	Carregamento de dados sintéticos.....	12
2.6	Modelação do <i>Data Warehouse</i>	13
2.6.1	Dimensões e Factos	13
2.6.2	População das tabelas.....	14
2.7	Análise dos dados.....	15
2.7.1	Criação de rotas.....	15
2.7.2	Desenho de gráficos.....	16
2.8	Menu.....	16
3	Conclusão	17

1 Introdução

O transplante de córnea é um procedimento cirúrgico que tem como objetivo substituir uma córnea doente ou danificada pela de um dador. Esta cirurgia envolve diferentes etapas desde a proposta de transplante, a chamada para transplante e a realização efetiva da cirurgia. Assim, para garantir a qualidade das informações recolhidas durante o processo, é fundamental armazenar os dados relacionados à proposta de forma estruturada e considerar um conjunto de características para auditar os processos de transplante, incluindo os dados do paciente (como identificação, data de nascimento e género), dados da proposta (como data da proposta, médico responsável, prioridade, diagnóstico, observações, complexidade e riscos), dados da consulta de anestesia (data e observações) e dados da inscrição (data de inscrição em lista ativa, data de realização do transplante, data de cancelamento, motivo de cancelamento e estado). Cada proposta de transplante de córnea pode ainda estar em diferentes estados, como a aguardar consulta de anestesia, em lista de espera ativa, realizado ou cancelado.

O objetivo deste projeto passa, portanto, por desenvolver uma aplicação capaz de apresentar uma lista de todas as propostas de transplante existentes, alteração de seus estados e também a criação de novas propostas. Assim, os principais requisitos para o desenvolvimento do mesmo passaram pela modulação do formulário, construção dos modelos de dados, criação de uma interface com autenticação, criação de interfaces para o registo de propostas e a apresentação da lista de propostas, assim como a apresentação de indicadores relevantes. Para tal foram utilizadas ferramentas como *Archetype Designer* para a modulação, *React* para a construção da interface do utilizador, *NodeJS* para o *backend* e as bases de dados *MongoDB* e *mySQL* para o armazenamento dos dados.

Para além disso, a aplicação também irá fornecer indicadores relevantes sobre os transplantes, como o tempo de espera para a cirurgia que, aliados a uma gestão eficiente das propostas, são essenciais para garantir a melhor prestação de cuidados possível. Nesse contexto, a área de *Business Intelligence* (BI) desempenha um papel fundamental. Esta envolve a recolha, análise e apresentação de informações para auxiliar na tomada de decisões e na monitorização do desempenho de uma organização. No contexto da aplicação desenvolvida, foram desenvolvidos uma variedade de *Key Performance Indicators* (KPI) com o objetivo de facilitar a análise dos dados. Através da aplicação, os utilizadores terão acesso a gráficos que têm o objetivo de fornecer uma visão clara e concisa do desempenho do processo de transplante de córnea, nomeadamente quanto ao número de transplantes, tempos de espera e cancelamentos. A análise desses indicadores poderá permitir que os profissionais

de saúde e gestores identifiquem áreas de melhoria nas instituições, tomem decisões mais informadas e adotem estratégias para otimizar o fluxo de transplantes e a qualidade dos cuidados prestados.

2 Implementação

2.1 Arquitetura

A aplicação foi desenvolvida utilizando a biblioteca *JavaScript React* para o *frontend*, combinada com alguns elementos da *framework Bootstrap* que permite agilizar o desenvolvimento de alguns componentes e garantir uma interface responsiva e visualmente apelativa. No *backend*, foi utilizado o *Node.js* para o desenvolvimento de um servidor eficiente. Os dados dos utilizadores e das propostas são armazenados no banco de dados *MongoDB*, enquanto o *MySQL* é utilizado para manipular os dados e executar *queries* que irão alimentar os gráficos apresentados no *frontend*.

Assim, podemos dividir a arquitetura em diferentes camadas. No *frontend*, o *React* é, então, utilizado como base para toda a interface da aplicação com a qual os utilizadores vão interagir, juntamente com a incorporação de alguns elementos *Bootstrap*. Para além disso, a biblioteca *Recharts* é utilizada para criar gráficos interativos com base nos dados fornecidos pelo *backend*. Por outro lado, no *backend*, o *Node.js* permite a construção de um servidor que lida com os pedidos dos clientes e gere a comunicação com as bases de dados. Ainda nesta camada, a *framework Express.js*, inerente ao *Node.js*, é utilizada para facilitar o roteamento dos pedidos HTTP no *backend* ao simplificar a criação de rotas e a definição de *endpoints* para lidar com as operações CRUD (*Create, Read, Update, Delete*). Quanto às bases de dados, o *MongoDB* é utilizado para armazenar os dados dos utilizadores e das propostas de transplante. Este é uma base de dados *NoSQL* importante para o armazenamento e recuperação eficiente de dados. Por outro lado, recorreu-se ainda ao *MySQL*, uma base de dados relacional que é utilizado, no contexto da aplicação, para armazenar e manipular os dados adicionais de modo a executar *queries* complexas que envolvem relacionamentos entre as tabelas.

No desenvolvimento da aplicação foi também utilizado o *Archetype Designer*, uma ferramenta de modulação, neste caso relevante para modular os formulários das propostas na aplicação, definindo os campos, tipos de dados e validações necessárias, garantindo a consistência nos dados que poderão ser recolhidos e armazenados de forma estruturada nas bases de dados.

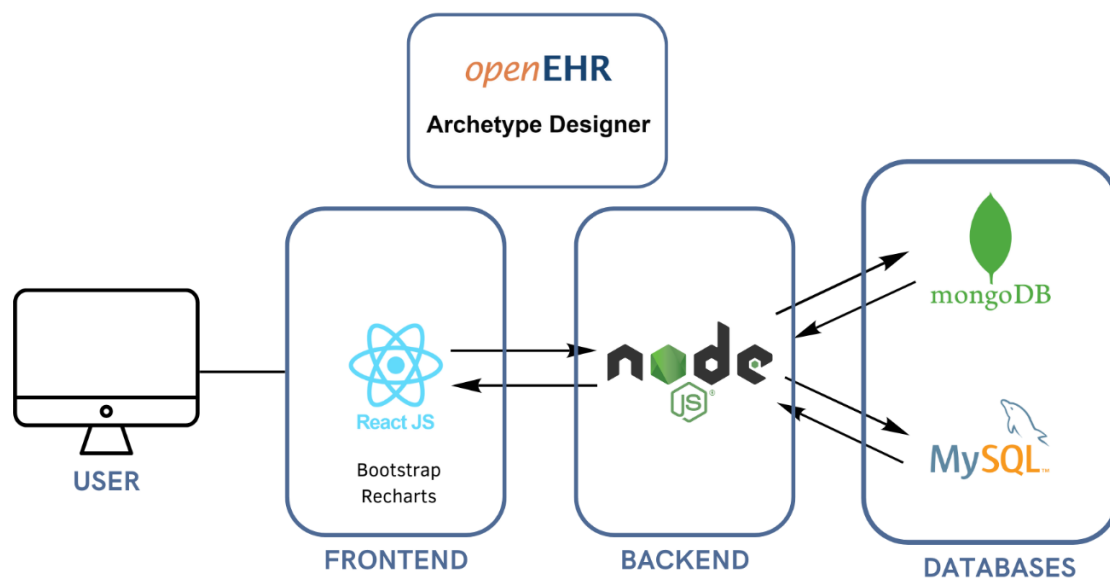


Figura 1 - Arquitetura da aplicação desenvolvida.

2.2 Login

No projeto desenvolvido, foi implementada a funcionalidade de se efetuar *login* através da inserção das credenciais do utilizador, ou seja, nome e palavra-passe. O presente capítulo serve para abordar o processo de criação e lógica deste mecanismo.

O ficheiro *users.js* inserido na pasta *controller* contém funções relacionadas com o controlo dos utilizadores da aplicação. A função denominada *newUser* é responsável por criar novos utilizadores na base de dados, recebendo como parâmetros o *user_id*, *username*, *password*, *first_name*, *last_name* e *email*, variáveis cujo nome é autoexplicativo. Neste arquivo, encontram-se também implementadas as funções *listUsers* e *getByUsername* que servem para listar todos os utilizadores presentes na base de dados e pesquisar um utilizador pelo seu nome, respetivamente.

O ficheiro *users.js* da pasta *routes* é um módulo Node.js que define rotas relacionadas com os utilizadores da aplicação desenvolvida. Deste modo, o principal objetivo do código aqui implementado é a definição da lógica para validar as credenciais de um utilizador, ao efetuar o login. Neste ficheiro o *UserController*, controlador que contém as funções para manipular utilizadores, é importado a partir do documento *users.js* localizado na pasta *controller*. Destaca-se a função *isValidUser* que é utilizada para averiguar se um utilizador está registado e se a senha inserida corresponde à sua senha real. Também a definição da rota */check-user* é um ponto relevante, visto que lida com as requisições de validação das credenciais durante o login, chamando a função *isValidUser*.

O ficheiro *users.js* inserido na pasta *model* define o esquema do modelo de dados para os utilizadores. O módulo *mongoose* é importado a fim de se obter uma interface para ser possível interagir com o *MongoDB* e a classe *Schema* do *mongoose* é também importada para se dar a criação de um novo esquema. Deste modo, criou-se o objeto *UserSchema* que define os campos para o modelo de cada utilizador. Os campos são os referidos no segundo parágrafo, incluindo também os parâmetros *blocked* e *active*. Por defeito, o valor da variável *blocked* é definido como *0* (não bloqueado) e o valor do campo *active* como *1* (ativo). Note-se que os campos *user_id* e *username* são definidos como '*unique: true*', o que significa que têm de ser únicos na base de dados.

Resumindo, o ficheiro *users.js* da pasta *router* lida com as requisições HTTP relacionadas com os utilizadores, chamando as funções apropriadas do *UserController* (ficheiro *users.js* inserido na pasta *controller*), que por sua vez utiliza o *UserSchema*, definido no documento *users.js* da pasta *model*, para interagir com a base de dados *MongoDB*.

Mais concretamente, quando um formulário de *login* é preenchido e submetido, a rota */check-user* é acionada, a função *isValidUser* é chamada para validar as credenciais fornecidas, sendo que o *UserController* verifica se o utilizador existe, se está ativo e se a palavra-passe fornecida é correta. Por fim, com base no resultado da validação, a resposta adequada é enviada.

Todos os ficheiros e pastas, relacionados com o processo de login, mencionados até aqui encontram-se localizados dentro da pasta *server*. Nos seguintes parágrafos, pode constatar-se que, para se dar esta autenticação, foram criados, na pasta *client*, ficheiros que implementam este mecanismo numa interface.

O arquivo *AuthContext.js* é responsável pela criação de um contexto de autenticação, possibilitando que os vários componentes da aplicação *React* tenham acesso aos dados relacionados com a autenticação, permitindo executar várias ações sem a necessidade de transferência dessas informações entre componentes. Deste ficheiro destaca-se a função *isValidAuth* que verifica se um utilizador se encontra autenticado. Esta função encontra-se dentro do *AuthProvider*, juntamente com as funções *authenticate* e *logout* que servem para autenticar e terminar a sessão, respetivamente.

O ficheiro *Login.js* lida com o formulário de login, envia as informações de autenticação para o servidor e atualiza o estado de autenticação, utilizando o contexto de autenticação do *AuthContext*, referido anteriormente. Sobre o código aqui implementado pode destacar-se a utilização do *useEffect* para verificar se um utilizador já está autenticado; se estiver, é redirecionado para a página inicial

(/homepage). Note-se ainda que o *handleSubmit* é chamado quando se clica no botão *Entrar* – recorre-se ao método POST com as informações inseridas no login (*username* e *password*), sendo que o URL de destino corresponde à rota */users/check-user*. Dependendo da resposta do servidor, é gerada uma mensagem de erro ou efetua-se a *autenticação*, sendo o utilizador redirecionado para a página */homepage*. É ainda neste ficheiro que se dá a criação da página do login a nível estético, configurando os campos para inserir o nome e senha, bem como outros elementos, como por exemplo, o botão para enviar o formulário.

Procedeu-se também à criação do ficheiro *Login.css* que auxilia na definição de estilos para a página de login mencionada anteriormente. Neste arquivo implementam-se apenas pormenores estéticos que são aplicados aos elementos HTML correspondentes.

2.3 Modelação do formulário

A modelação do formulário envolveu a identificação dos campos essenciais para a toda a gestão de proposta de transplante. Tendo em conta as especificações fornecidas no enunciado, foram identificadas as seguintes secções e campos respetivos:

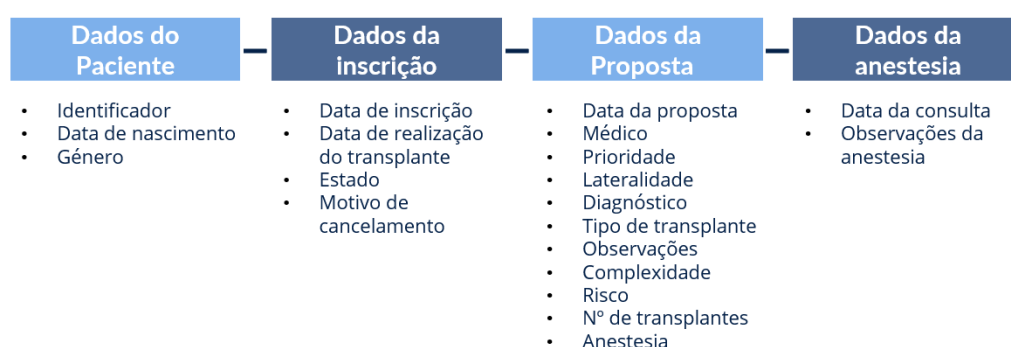


Figura 2 - Estrutura do Formulário

2.3.1 Archetype Designer

Para a formulação do formulário utilizamos o *Archetype Designer*, o qual é uma ferramenta fundamental na criação de arquétipos. Os arquétipos são modelos estruturados que definem as informações e o fluxo de trabalho num sistema de saúde.

Com o *Archetype Designer*, definimos os arquétipos de acordo com os requisitos especificados para o desenho do formulário, o qual está dividido em 4 secções:

- **Dados do paciente:** Esta secção contém informações dos dados demográficos do paciente, como o seu identificador, nome, género, data de nascimento, endereço, cidade, distrito, código postal e país.
- **Proposta de Transplante:** Aqui são registados o tipo de diagnóstico, prioridade, data de registo, comentário e requisitante.
- **Procedimento:** Esta secção está dividida em 3 partes. A primeira parte mostra o estado da proposta de transplante (cancelado, em espera, aguarda consulta, adiado ou interrompido) e o tipo de procedimento. A segunda parte inclui os detalhes específicos do transplante como o número de transplantes do paciente momento da proposta e se é necessário ou não de anestesia. A terceira parte contém dados sobre a localização anatómica, como a lateralidade (olho esquerdo ou direito), a complexidade e risco do procedimento, a data da proposta, data de conclusão, motivo de fecho da proposta e um campo para comentários adicionais.
- **Consulta de anestesia:** Nesta secção é registada a data de admissão, informações sobre o médico responsável (nome e identificador) e informações sobre a consulta de anestesia através de um campo para detalhar o resultado e outro para observações de anestesia.

Essas secções abrangem todas as informações relevantes necessárias para toda a gestão da proposta de transplante.

2.3.1 Criação do Modelo de Base de Dados

A criação do modelo de base de dados constitui uma etapa essencial no desenvolvimento da aplicação, uma vez que esta exige o armazenamento e manipulação de informações. Nesta fase, é projetada a estrutura lógica dos dados anteriormente mencionados.

Note-se que a conexão à base de dados é estabelecida utilizando-se o código `mongoose.connect(uri)` onde *uri* é o URL da conexão com o *MongoDB*. Este código encontra-se implementado sempre que é necessário realizar esta conexão.

Como mencionado anteriormente, na pasta *model* do servidor encontra-se o ficheiro *users.js* que define o esquema do modelo de dados para os utilizadores no *MongoDB*, utilizando o *Mongoose*, uma biblioteca de modelação de objetos do *MongoDB* para *Node.js*. Neste arquivo, é então criado o esquema *UserSchema* que define os campos e os tipos de dados esperados. No final, o modelo é exportado para ser utilizado noutras secções.

A mesma lógica é implementada em relação às *compositions* no ficheiro *composition.js* da pasta *model* do servidor. Primeiro, dá-se a importação do módulo *mongoose* para se aceder às funcionalidades do *Mongoose* e o *Schema* para criar um novo esquema. De seguida, é criado o *CompositionSchema* que define os campos e os tipos de dados esperados em cada parâmetro.

Dentro do *CompositionSchema* existem quatro campos: *proposal_id* que é uma *string* e é definido como único, garantindo que cada documento tenha um identificador exclusivo, *patient_id* e *transplants* que são números e *composition* que é do tipo misto (*mixed*), uma vez que pode conter diferentes tipos de dados. No final o esquema é exportado como um modelo que é registado no *Mongoose* com o nome *composition*.

2.3.2 Criação de Rotas Relacionadas com *Compositions*

No ficheiro *composition.js* que se encontra na pasta *routes* do servidor encontram-se definidas várias rotas. A primeira a ser criada é uma rota do tipo GET em */list/* que quando é acionada lista todas as *compositions*. A segunda rota definida é também do tipo GET em */:id/*, sendo que *id* se refere a um identificador específico; quando se acede a esta rota obtém-se uma proposta com o identificador fornecido. A rota do tipo POST em */create* é utilizada para criar uma nova *composition* com base nos dados fornecidos na solicitação. Por fim, define-se uma rota POST em */update* que, quando acionada, atualiza uma *composition* existente com os dados inseridos na solicitação. Os resultados destes procedimentos são retornados como uma resposta JSON. Note-se que, no final, o objeto aqui concebido é exportado, permitindo a utilização destas rotas em diferentes ficheiros e proporcionando uma melhor organização da aplicação.

Para se efetuarem as ações referidas no parágrafo anterior, foi utilizado um controlador, *CompositionController*. Este elemento encontra-se definido no ficheiro *composition.js* da pasta *controller* do servidor. Este arquivo contém então um conjunto de funções relacionadas à manipulação de *compositions*. De um modo geral, estas funções são utilizadas para realizar operações de criação, listagem e atualização de *compositions*, permitindo interagir com a base de dados.

A função *newComposition* é responsável por criar uma nova *composition* e guardar na base de dados. Esta função recebe como parâmetros o identificador do paciente (*patient_id*), o número de transplantes (*transplants*) e a própria *composition* (*composition*). Note-se que, inicialmente, verifica-se se o número de transplantes é válido e, em seguida, é criado um novo objeto *CompositionSchema* com os dados fornecidos.

A função *getNumberOfTransplants* retorna o número de transplantes de um utente, recebendo o *patient_id* como argumento e fazendo uma consulta à base de dados. A função *validateTransplants* valida o número de transplantes. A função *getNumberOfTransplants* é um método auxiliar que retorna o número de transplantes de um determinado paciente. A função *listAll* lista todas as *compositions* existentes. A função *getProposal* retorna uma proposta específica com base no *proposal_id* fornecido. Por fim, a função *updateComposition* é a responsável pela atualização da *composition* de uma determinada proposta.

2.3.3 Página *Web* para Renderizar a Composição do Formulário

Após a criação do arquétipo, foi desenvolvida uma página no front-end que permite a criação e edição do formulário com base no arquivo JDT gerado a partir do arquétipo.

Validação do formulário

A fim de garantir a consistência dos dados introduzidos pelo utilizado no formulário, foram realizadas algumas validações na criação e edição do formulário, identificado no modelo de dado como *composition*. As principais validações implementadas são:

- **Validação das datas**
 - Verifica-se se a data da proposta é posterior à data de registo.
 - A data de conclusão deve ser posterior à data de registo.
 - A data de conclusão deve ser posterior à data da proposta.
- **Número de transplantes**
 - Esta validação é realizada no servidor e tem como objetivo garantir que o número de transplantes do paciente na proposta atual é maior do que o número total de transplantes já registados para o mesmo paciente.
 - Caso o número de transplantes da proposta que está a ser criada seja maior do que o número total de transplantes registados para esse paciente, é enviada uma mensagem de erro para o utilizador.
 - Além disso, foi criado um índice único no *model* do *Composition* entre o *patient_id* e *transplants* para garantir que não existam duas propostas para o mesmo paciente com o mesmo número de transplantes. Assim garantimos que em cada proposta registada esse número é ser diferente.

- **Secção de anestesia**

Na secção de anestesia, são realizadas as seguintes validações:

- Caso a opção "Anestesia" esteja inativa, os campos de consulta de anestesia não podem ser preenchidos. Isso significa que não é possível inserir a data de admissão, identificação do médico, resultado e observações de anestesia.
- Caso a opção "Anestesia" esteja ativa:
 - Se a data de admissão estiver preenchida, as seguintes validações são aplicadas:
 - A data de admissão deve ser superior à data da proposta e à data de registo.
 - É obrigatório preencher os dados do médico.
 - Se a data de admissão não estiver preenchida, mas houver algum campo preenchido na secção de anestesia, indica que a data de admissão é obrigatória.

2.4 Listagem de propostas

A listagem de propostas é uma funcionalidade fundamental da aplicação, na medida que permite ao utilizador visualizar e modificar as propostas de transplante de córnea armazenadas na base de dados *MongoDB*. Esta lista é apresentada no formato de uma *DataTable* no *frontend* e fornece uma visão organizada e estruturada das informações. Assim, os utilizadores podem navegar pela tabela para visualizar as propostas existentes, obter informações detalhadas sobre cada uma e ainda organizar a tabela de acordo com uma coluna específica, por exemplo, data de registo mais recente ou mais antiga.

Ao carregar a página das *proposals*, o *frontend* envia um pedido ao servidor *Node.js*, que por sua vez comunica com o *MongoDB* para obter os dados das propostas, sendo retornados como resposta ao pedido. A *DataTable* exibe várias colunas relevantes das propostas, como a data de registo, data da proposta, prioridade, estado, entre outras informações importantes. Para além disso, cada linha da tabela representa uma proposta de transplante de córnea e os utilizadores têm a opção de visualizar mais detalhes sobre uma proposta específica ao clicar no botão com um *icon* de um olho, entrando assim no modo de visualização da *proposal*, onde podem ser consultadas informações completas sobre a proposta, incluindo todos os campos e detalhes associados. Adicionalmente, a tabela permite que o utilizador edite as propostas existentes ao clicar no botão de edição na linha desejada, com um *icon* de um lápis, o que abrirá o formulário preenchido com os dados da proposta, com a possibilidade de fazer alterações e guardar as alterações na base de dados, atualizando assim as informações da proposta.

Além da listagem e edição das propostas existentes, é também possível a criação de novas propostas. Ao clicar no botão de "Create" o utilizador é encaminhado para uma nova página com o formulário das propostas onde, após preencher os campos necessários, poderá ser guardada na base de dados.

Assim, esta abordagem proporciona aos utilizadores uma interface intuitiva e interativa para gerir as propostas de transplante de córnea permitindo facilmente visualizar, editar e criar novas propostas e garantindo uma gestão eficiente das informações relacionadas às mesmas.

2.5 Carregamento de dados sintéticos

A fim de obter dados para a realização da análise de dados (*Business Inteligente*), recebemos um ficheiro Excel com informações de algumas propostas de transplante contendo alguns dados de situações reais.

A	B	C	D	E	F	G	H	I	J	K	L
ID_DOENTE	DES_DIAGBASE	DES_PROCI	LOC_PROC	CON_ANEST	PRIORIDADE	DATA_PROPC	DATA_REGISTO	DATA_CONSULTA	DATA_FECHE	MOT_SAIDA	
906090	Queratopatia bolhosa OE + catarata	Transplante c	Esquerdo	1	2	2010-11-09	2010-10-14	2010-11-15	2013-05-31	Transplantado	
1210895	Queratopatia Bolhosa - OE	Transplante c	Esquerdo	1	2	2011-08-30	2011-08-30	2012-02-10	2013-01-30	Desistência	
1198	Queratopatia Bolhosa - OE	Transplante c	Esquerdo	1	2	2011-08-08	2011-08-08	2011-09-16	2013-07-30	Contra indicação definitiva	
377071	od-rejeição de enxerto	Transplante c	Direito	1	2	2011-10-14	2011-10-14	2012-02-10	2013-04-10	Transplantado	
154660	distrofia de fuchs	Transplante c	Esquerdo	1	2	2011-08-22	2011-08-22	2011-09-16	2013-07-23	Desistência	
100131	distrofia fuchs OE	Transplante c	Esquerdo	1	2	2011-07-25	2011-07-25	2011-09-16	2014-03-11	Desistência	
756295	distrofia endotelial congenita OD	Transplante c	Direito	0	2	2011-01-09	2011-01-09		2013-10-02	Transplantado	
1240954	falencia de enxerto	Transplante c	Esquerdo	0	2	2012-01-17	2012-01-17		2013-01-02	Transplantado	
735718	falencia enxerto	Transplante c	Esquerdo	0	2	2012-08-17	2012-08-17		2013-01-31	Transplantado	
208912	queratocone od	Transplante c	Direito	0	2	2012-07-30	2012-07-30		2013-02-08	Transplantado	
1417580	queratocone	Transplante c	Esquerdo	0	2	2012-07-30	2012-07-30		2013-01-31	Transplantado	

Figura 3 Excel fornecido com dados de propostas de transplante da córnea


No entanto, por questões de privacidade, não tivemos acesso aos nomes e datas de nascimento. Nesse sentido, foi sugerido que utilizássemos os dados de pacientes fictícios, mas com datas de nascimento e códigos postais válidos do ficheiro de doentes Covid utilizado nas aulas práticas. Assim sendo, foi necessário fazer um mapeamento entre estes os identificadores de pacientes. A atribuição não foi linear devido ao fato de que o ficheiro de transplantes continha múltiplos registos para o mesmo paciente. Para resolver essa situação, recorremos ao MySQL e realizamos as seguintes etapas:

- Ordenamos cada um dos identificadores distintos.
- Atribuímos um número sequencial a cada identificador.
- Realizamos a junção (*join*) com base no número sequencial igual.

Dessa forma, conseguimos mapear corretamente os identificadores dos pacientes dos dois conjuntos de dados, garantindo a consistência na atribuição.

Após essa etapa, foi necessário mapear os restantes campos do ficheiro de propostas de transplante com os campos do nosso formulário, bem como o mapeamento para os valores possíveis

de cada um deles. Na maioria dos casos, foi realizada uma atribuição por similaridade, no entanto, o campo de diagnóstico foi o mais complexo de processar. Isso se deve ao fato de que o conteúdo desse campo é de um campo de texto livre, onde foram utilizadas diferentes formas de escrever para indicar o mesmo diagnóstico. Alguns escritos com acentos, outros sem acentos, alguns utilizam diferentes combinações de abreviações e outros ainda podem estar escritos por extenso, sendo que todas estas variações indicam o mesmo tipo de diagnóstico. Mais uma vez foi necessário um processo manual de mapeamento dos diagnósticos existentes com os valores possíveis no nosso formulário. De seguida, geramos um novo ficheiro (*csv*) com a junção dos clientes feita anteriormente e com uma uniformização dos diagnósticos e criamos um script *server/synthetic_data/import.js* para fazer o mapeamento entre os dados do ficheiro e os campos do formulário, criar um registo com a estrutura de uma *Composition*, recorrendo à utilização de um *template server/synthetic_data/template_composition.json*, e inserir o registo no Mongo, através da API criada anteriormente no *server/controller/composition.js*.



```

1  template_composition.json
2  {
3    "items": [
4      {
5        "type": "block",
6        "key": "e9b7q",
7        "text": "${PATIENT_NAME}",
8        "type": "unstyled",
9        "depth": 0,
10       "inlineStyleRanges": [],
11       "entityRanges": []
12     },
13     {
14       "type": "block",
15       "key": "7vun2",
16       "text": "${PATIENT_ID}",
17       "type": "unstyled",
18       "depth": 0,
19       "inlineStyleRanges": [],
20       "entityRanges": []
21     },
22     {
23       "type": "block",
24       "key": "0p9u8",
25       "text": "${ADDRESS_LINE1}",
26       "type": "unstyled",
27       "depth": 0,
28       "inlineStyleRanges": [],
29       "entityRanges": []
30     },
31     {
32       "type": "block",
33       "key": "2zndp",
34       "text": "${PHONE}",
35       "type": "unstyled",
36       "depth": 0,
37       "inlineStyleRanges": [],
38       "entityRanges": []
39     },
40     {
41       "type": "block",
42       "key": "822eq",
43       "text": "${DISTRICT}",
44       "type": "unstyled",
45       "depth": 0,
46       "inlineStyleRanges": [],
47       "entityRanges": []
48     },
49     {
50       "type": "block",
51       "key": "8hhq",
52       "text": "${POSTAL_CODE}",
53       "type": "unstyled",
54       "depth": 0,
55       "inlineStyleRanges": [],
56       "entityRanges": []
57     },
58     {
59       "type": "block",
60       "key": "c0jao",
61       "text": "${COUNTRY}",
62       "type": "unstyled",
63       "depth": 0,
64       "inlineStyleRanges": [],
65       "entityRanges": []
66     },
67     {
68       "type": "text",
69       "code": "${GENDER_CODE}",
70       "text": "${GENDER_TEXT}"
71     },
72     {
73       "type": "text",
74       "code": "${BIRTH_DATE}",
75       "text": "${BIRTH_DATE}"
76     },
77     {
78       "type": "text",
79       "code": "${BIRTH_TIME}",
80       "text": "${BIRTH_TIME}"
81     },
82     {
83       "type": "text",
84       "code": "${DIAGNOSIS_CODE}",
85       "text": "${DIAGNOSIS_TEXT}"
86     },
87     {
88       "type": "text",
89       "code": "${PRIORITY_CODE}",
90       "text": "${PRIORITY_TEXT}"
91     },
92     {
93       "type": "text",
94       "code": "${REGISTRATION_DATE}",
95       "text": "${REGISTRATION_DATE}"
96     },
97     {
98       "type": "text",
99       "code": "${REGISTRATION_TIME}",
100      "text": "${REGISTRATION_TIME}"
101    },
102    {
103      "type": "text",
104      "code": "${SERVICE_DUE_DATE}",
105      "text": "${SERVICE_DUE_DATE}"
106    },
107    {
108      "type": "text",
109      "code": "${SERVICE_DUE_TIME}",
110      "text": "${SERVICE_DUE_TIME}"
111    }
112  ]
113 }

```

Figura 4- Parte do template para criação de uma composition

2.6 Modelação do *Data Warehouse*

A modelação do *Data Warehouse* é essencial para criar um ambiente eficiente e robusto de análise de dados. Esta abordagem envolve a criação de dimensões e fatos, onde as dimensões representam as características pelos quais os dados são analisados e os fatos as medidas quantitativas relacionadas com essas dimensões. O objetivo é explorar os dados de diferentes perspetivas a fim de obter informações que possam auxiliar na tomada de decisões.

2.6.1 Dimensões e Factos

Na etapa de modelação do *Data Warehouse*, realizamos a seleção das dimensões de análise sobre as quais pretendemos analisar os fatos e identificamos importante a criação das seguintes dimensões:

- **DIM_Tempo:** permite analisar os dados relacionados com o tempo das datas de propostas de transplante.
- **DIM_Prioridade:** permite analisar os dados com base nas diferentes prioridades atribuídas às propostas de transplante.
- **DIM_Diagnostico:** permite analisar os dados com base nos diferentes diagnósticos associados às propostas de transplante.
- **DIM_Tipo_Procedimento:** permite analisar os dados com base nos diferentes tipos de procedimentos de transplante realizados.
- **DIM_Lateralidade:** permite analisar os dados relacionados à lateralidade dos transplantes.
- **DIM_Anestesia:** permite analisar os casos que necessitaram ou não de anestesia nos procedimentos de transplante.
- **DIM_Motivo:** permite analisar os dados com base nos diferentes motivos que levaram ao cancelamento de um transplante.
- **DIM_paciente:** esta dimensão abrange as informações relacionadas com os pacientes, como a data de nascimento, o género e código postal.

Após a identificação das dimensões, definimos as medidas a serem integradas na estrutura de cada fato (tabela *FACT_Cornea*). As medidas incluem:

- **numero_transplantes:** número de transplantes de cada paciente.
- **idade:** idade do paciente no momento em que o transplante foi realizado.
- **tempo_espera_anestesia:** tempo de espera entre o registo da proposta de transplante e a realização da consulta de anestesia.
- **tempo_espera_anest_cir:** tempo de espera entre a consulta de anestesia e a realização da cirurgia de transplante.
- **tempo_espera_cirurgia:** tempo total de espera, desde o registo da proposta até a realização da cirurgia de transplante.

Após a definição das dimensões e factos criamos o script *data_warehouse/DW_Cornea.sql* para a criação do schema *DW_Cornea* com as respetivas tabelas das dimensões e tabela de factos.

2.6.2 População das tabelas

Para popular essas tabelas, fizemos modificações no script de carregamento *import.js* disponível em *server/synthetic_data/*, para criar uma tabela temporária no MySQL e, à medida que carregávamos as propostas no MongoDB, inserimos os mesmos registos nesta tabela no MySQL. Posteriormente, criamos um novo script *data_warehouse/propostas.sql* para popular cada uma das tabelas de dimensões e fatos a partir da tabela temporária.

2.7 Análise dos dados

Para realizar a análise dos dados armazenados no *Data Warehouse*, foi necessário desenvolver consultas (*queries*) que permitissem extrair as informações relevantes. Estas consultas desempenham um papel fundamental para disponibilizar, no *frontend*, os principais indicadores dos dados, também conhecidos como KPIs.

A visualização desses indicadores é feita através de gráficos intuitivos que facilitam a compreensão dos dados, fornecendo uma visão geral e concisa dos KPIs, permitindo assim uma análise eficaz dos dados.

2.7.1 Criação de rotas

Para aceder no *frontend* aos dados e gerar gráficos, procedemos à criação de rotas no servidor *Node.js* que executam as *queries* no *Data Warehouse* e devolvem a respetiva informação. Estas rotas foram essencialmente definidas para análise dos KPIs relacionados com o número transplantes, tempos de espera e motivos de cancelamento.

KPIs de Transplantes

- **/kpis/transplantes/diagnostico** - número de transplantes por diagnostico
- **/kpis/transplantes/procedimento** - número de transplantes por procedimento
- **/kpis/transplantes/lateralidade** - número de transplantes por lateralidade
- **/kpis/transplantes/prioridade** - transplantes por tipo de urgência
- **/kpis/transplantes/anestesia** - número de transplantes com e sem anestesia
- **/kpis/transplantes/feriados-fim-de-semana** - número de transplantes realizados em feriados e fins de semana
- **/kpis/transplantes/idade** - número de transplantes por idade do paciente na data de realização do transplante.

KPIs de Tempos de Espera

- **/kpis/tempo-espera/transplante/ano** - média de tempo de espera por ano (em dias)
- **/kpis/tempo-espera/transplante/mes** - média de tempo de espera por mês (em dias)
- **/kpis/tempo-espera/anestesia** - tempo médio de espera de transplantes com anestesia, por tipo de diagnostico, onde é calculada a diferença entre:

- a data de registo da proposta e a data da consulta de anestesia
- a data da consulta de anestesia e a data de realização do transplante
- a data de registo da proposta e data de realização do transplante (tempo total de espera)
- **/kpis/tempo-espera/** - média de tempo total de espera por tipo de diagnostico

KPIs de Motivos de Cancelamento

- **/kpis/motivo/desistencia/diagnostico** - número de desistências por diagnostico
- **/kpis/motivo/desistencia** - número de desistências por ano
- **/kpis/motivo/cid** - número de cancelamentos por *"Contra indicação definitiva"*
- **/kpis/motivo/cit** - número de cancelamentos por *"Contra indicação temporária"*
- **/kpis/motivo/morte** - número de cancelamentos por "Morte" do paciente

2.7.2 Desenho de gráficos

Todos os gráficos anteriormente mencionados, foram configurados esteticamente através do ficheiro *KpiChart.css*. Foi ainda criado um ficheiro *javascript* para cada um dos gráficos – estes documentos são componentes *React* que utilizam a biblioteca *Recharts* para exibir diferentes tipos de gráficos. Estes ficheiros têm uma estrutura semelhante, diferindo no que toca à configuração específica para cada gráfico.

É também nestas secções do código que ocorre a recolha dos dados que os gráficos utilizam. De modo resumido, existe uma API que é, basicamente, uma camada intermédia que fornece acesso aos dados para os componentes dos gráficos quando estes efetuam solicitações. Assim, é permitido que os gráficos sejam atualizados dinamicamente com as informações mais recentes.

2.8 Menu

Para tornar a aplicação mais intuitiva e interativa, foi implementado, na interface, um menu (barra lateral) que indica que páginas o utilizador pode aceder. Deste modo o acesso às diferentes secções do *website* é facilitado e o conteúdo encontra-se mais organizado.

Como se pode verificar na *Figura 5*, através deste elemento o utilizador consegue aceder à listagem de propostas existentes e aos gráficos originados, sendo que estes se encontram categorizados de

acordo com a lógica explicada anteriormente. Para além do acesso às várias páginas, o utilizador consegue efetuar o *logout*, terminando a sua sessão.

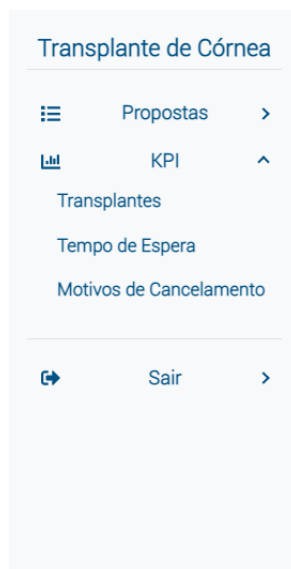


Figura 5- Barra lateral (menu) implementada

Para a criação deste elemento foi concebido o ficheiro *SideBar.css* que define tanto o estilo da barra lateral como algumas propriedades do restante conteúdo da página. Ou seja, é no código aqui implementado que se encontram detalhados os pormenores estéticos como a largura, cor de fundo, entre outros.

No ficheiro *SideBar.js* encontra-se o código responsável por exibir efetivamente o a barra lateral e por fazer com que os botões funcionem, indicando as páginas de redirecionamento. Note-se que a função *handleLogoutClick* é chamada quando o botão relativo ao *logout* é acionado.

3 Conclusão

De modo sucinto, o presente relatório descreveu o processo de análise de dados e criação de um ambiente de *Business Intelligence*. O projeto consistiu no desenvolvimento de uma aplicação para gerir propostas de transplantes da córnea, utilizando ferramentas como *Archetype Designer*, *React*, *NodeJS*, *MongoDB* e *MySQL*. Além disso foi também cumprido o objetivo de se fornecerem indicadores importantes em forma de gráficos. Em cenários reais, a análise destes dados pode auxiliar a identificar pontos a melhorar, otimizando a qualidade dos serviços clínicos.

Existem melhorias que podem ser consideradas para aprimorar o projeto como o aumento de dados para aumentar a precisão e representatividade das estatísticas concebidas, a elaboração de mais gráficos com novos indicadores de desempenho e a melhoria da interface, aprimorando o design e a usabilidade da aplicação, tornando-a mais intuitiva e legível.