



INSTITUTO POLITÉCNICO DE LISBOA
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

SALI - Sistema de Apoio a Lares de Idosos

Daniel Patrício n.º 43509, a43509@alunos.isel.pt

Tiago Pereira n.º 43592, a43592@alunos.isel.pt

Luís Guerra n.º 43755, a43755@alunos.isel.pt

Orientadores:

Prof. Nuno Leite - ISEL

Eng.º João Antunes - Xpand IT

Projeto e Seminário

Relatório Final

setembro de 2021

Resumo

Com o aumento da esperança de vida e o envelhecimento da população, existem em Portugal 2 526 lares para idosos nos quais estão institucionalizadas 99 234 pessoas e onde trabalham 60 mil profissionais SNS (2021).

Analisando e estruturando os processos transversais a todos os lares, foi possível chegar a um conjunto de funcionalidades que representam o núcleo das suas operações diárias, definindo assim as metas deste projeto.

O Sistema de Apoio a Lares de Idosos (SALI) tem um só objetivo: *Aumentar o bem-estar dos utentes melhorando o leque de ferramentas disponíveis para a gestão do dia-a-dia dos lares.*

Com este objetivo desenvolveu-se uma aplicação *web* destinada aos administradores para ajudar a solucionar questões administrativas e de gerência e uma aplicação móvel para apoiar os auxiliares de saúde na gestão das tarefas diárias pendentes, confirmando a identidade dos utentes sempre que necessário utilizando pulseiras *Near Field Communication* (NFC), e ainda oferecendo a possibilidade de realização de diagnósticos recorrendo ao módulo de Inteligência Artificial (IA).

Palavras-chave: *Android, API, Bem-Estar, Browser, Controlo, Doença, Gestão, Idoso, Lar, NFC, Sintoma.*

Lista de Acrónimos

API *Application Programming Interface*

DOM *Document Object Model*

FCM *Firebase Cloud Messaging*

HTML *HyperText Markup Language*

HTTP *HyperText Transfer Protocol*

IA *Inteligência Artificial*

IDP *Identity Provider*

MVC *Model View Controller*

NFC *Near Field Communication*

PKCE *Proof Key for Code Exchange*

QR *Quick Response*

REST *Representational State Transfer*

SALI *Sistema de Apoio a Lares de Idosos*

UI *User Interface*

Conteúdo

Resumo	iii
Lista de Acrónimos	v
Lista de Figuras	xi
Lista de Tabelas	xiii
1 Introdução	1
2 Formulação do Problema	3
2.1 Estado da Arte	4
2.2 Funcionalidades	5
2.2.1 Administrador	5
2.2.2 Funcionários	5
3 Abordagem	7
3.1 Arquitectura	8
3.2 Pulseira/Banda NFC	10
3.2.1 <i>Chips</i> NFC	10
3.3 Modelo de Dados	11
4 Servidor de Identidade	15
4.1 Implementação	16
4.2 Arquitectura	18
4.3 Peças de Software Auxiliares	18
4.4 Funcionalidades	19

5	Web API	23
5.1	Implementação	24
5.2	Arquitetura	24
5.2.1	Domínio	25
5.2.2	Aplicação	25
5.2.3	Infraestrutura	25
5.2.4	UI	26
5.3	Peças de <i>Software</i> Auxiliares	26
5.4	Funcionalidades	26
6	Aplicação Web	29
6.1	Implementação	30
6.2	Arquitetura	31
6.2.1	<i>App.js</i>	31
6.2.2	<i>Containers/Componentes</i>	32
6.2.3	<i>Services</i>	32
6.3	Peças de Software Auxiliares	32
6.4	Funcionalidades	33
7	Aplicação Móvel	35
7.1	Implementação	36
7.2	Arquitetura da Aplicação Móvel	37
7.2.1	<i>Componentes</i>	37
7.2.2	<i>App.js</i>	38
7.2.3	<i>Services</i>	38
7.2.4	<i>Navegação</i>	38
7.2.5	<i>Notificações</i>	39
7.3	Peças de Software Auxiliares	40
7.4	Funcionalidades	41
8	Componente Diagnóstico de Doenças - IA	43
8.1	Implementação	44
8.2	Arquitetura	45
8.3	Peças de Software Auxiliares	45
8.4	Funcionalidades	46
9	Conclusão	47
9.1	Trabalho Futuro	47

Referências	49
--------------------	-----------

Lista de Figuras

3.1	Arquitetura da solução	8
3.2	Chips NTAG213	10
3.3	Pulseiras NTAG213	11
3.4	Modelo relacional	13
4.1	Diagrama de utilização do servidor de identidade	16
4.2	Esquema Arquitetura MVC	18
4.3	Fluxo de informação entre IDP, clientes e recursos protegidos	19
4.4	Exemplo de <i>code verifier</i> e <i>code challenge</i>	20
4.5	Página de <i>login</i> do IDP	20
4.6	Exemplo de configuração de uma aplicação cliente	21
5.1	<i>Clean Architecture</i>	25
6.1	Arquitetura da aplicação <i>web</i>	31
7.1	Arquitetura da aplicação móvel	37
7.2	Árvore de componentes	38
8.1	Arquitetura do componente de diagnóstico de doenças	45

Lista de Tabelas

4.1	Características do <i>Identity Server</i>	17
4.2	Características do <i>Azure B2C</i>	17

Capítulo **1**

Introdução

O mundo encontra-se numa era de digitalização. A tecnologia, cada vez mais presente nos sistemas de saúde, tornou-se um forte aliado na gestão e eficiência tanto de sistemas como métodos de trabalho.

Os sistemas de lares de idosos podem também beneficiar com a introdução de tecnologia na gestão e métodos de trabalho, permitindo melhorar a sua eficiência, competitividade face a outros sistemas não tecnológicos e fortalecer a sinergia entre administração e auxiliares de saúde.

Em todos os ambientes de trabalho, especialmente aqueles que empregam mão-de-obra humana, ocorrem erros. Num lar, onde muitos pacientes se encontram fragilizados e necessitam de cuidados de saúde diários, um erro pode causar transtornos.

O SALI procura dar resposta e solucionar este problema. Assim, com o SALI será possível melhorar a comunicação entre a administração e auxiliares de saúde, fornecer informações que permitam facilitar a gestão do lar, e para os auxiliares de saúde, a disponibilização duma ferramenta que simplifica a realização das tarefas diárias e monitorização do estado de saúde dos pacientes.

Este relatório encontra-se estruturado em 9 capítulos. Nos três primeiros é dado a conhecer o problema que este projeto pretende resolver, as aplicações existentes no mercado que partilham o mesmo objetivo e por fim a abordagem, seguida pelo grupo, para resolver o problema apresentado no início. Do capítulo 4 ao 8 estão organizados cada um dos módulos da solução, todos com as mesmas subsecções: implementação, arquitetura, peças de software auxiliares e funcionalidades. No capítulo 9 elencam-se as conclusões do trabalho e são indicados pontos de futuras melhorias. Para simplificar a organização do relatório foram criadas adicionalmente as seguintes adendas:

- a. Suporte ao desenvolvimento - guia explicativo do processo de desenvolvimento e das ferramentas usadas.
- b. Suporte à manutenção - orientações gerais relativas à manutenção e monitorização das aplicações do projeto.
- c. Manual de utilização da aplicação *web* - explicação dos componentes visuais e instruções para a realização das operações suportadas.
- d. Manual de utilização da aplicação móvel - explicação dos componentes visuais e instruções para a realização das operações suportadas.

Capítulo 2

Formulação do Problema

Conteúdo

2.1	Estado da Arte	4
2.2	Funcionalidades	5
2.2.1	Administrador	5
2.2.2	Funcionários	5

2.1 Estado da Arte

Como a questão da informatização de sistemas de cuidados de saúde não é novidade, há sistemas já existentes que têm o mesmo propósito que o tratado no presente documento. Uma das aplicações que se destacam é a aplicação MySenior (2021). Esta aplicação permite registar tarefas que um auxiliar de saúde introduz no sistema e também gerir membros da administração e colaboradores.

Um dos pontos diferenciadores do nosso sistema em comparação com o MySenior é no registo de tarefas concluídas/não concluídas.

No SALI, esse registo poderá necessitar de confirmação do contacto entre o auxiliar e o utente através da leitura das pulseiras NFC enquanto que esse registo no MySenior é realizado num dispositivo centralizado, coletivo, sem componente móvel e sem a garantia da ocorrência de contacto entre os mesmos.

Durante a fase de pesquisa surgiu também outra aplicação, denominada de GericarePro (2021). É semelhante ao MySenior com um maior foco no historial médico do paciente, o que facilita as dosagens da medicação. Da mesma forma, esta não dispõe de componente móvel, sendo o portal da mesma, acedido por todos os membros da organização.

Neste contexto, o SALI disponibiliza também uma aplicação móvel a ser usada por cada auxiliar para que a interação com o sistema não seja centralizada. A utilização da aplicação móvel em conjunto com a pulseira NFC permite não só uma maior rapidez na reação e registo de algum imprevisto como também a garantia de contacto entre auxiliar e utente.

O SALI para além de garantir a maioria das funcionalidades das aplicações semelhantes existentes no mercado, pretende também disponibilizar novas funcionalidades. Uma delas é a possibilidade de, através da aplicação móvel, os auxiliares poderem fazer um diagnóstico a um utente indicando um conjunto de sintomas recolhidos por observação do mesmo.

Este diagnóstico, quando resulta numa previsão positiva de uma doença, alerta todo o sistema para essa possibilidade, dando assim a oportunidade aos colaboradores de reagir em consonância com o alerta produzido.

2.2 Funcionalidades

Por forma a cumprir os requerimentos acima descritos, o sistema terá de dar suporte a um conjunto de funcionalidades que se concentram, do ponto de vista do cliente, na aplicação *web* para o administrador e na aplicação móvel para os auxiliares de saúde.

2.2.1 Administrador

No que concerne à aplicação *web*, o sistema terá que permitir ao administrador:

- visualizar dados sobre o estado corrente do lar;
- procurar informações sobre qualquer utente do lar;
- criar e editar um plano personalizado de tarefas para um utente;
- procurar informações sobre qualquer funcionário do lar;
- relacionar utentes e funcionários;
- relacionar quartos e utentes;
- inserir utentes no sistema.

2.2.2 Funcionários

No contexto da aplicação móvel, os auxiliares de saúde poderão:

- visualizar e registar tarefas e incidências;
- receber notificações em caso de admissões de novos utentes;
- realizar diagnósticos a utentes;
- ler e escrever pulseiras NFC;
- visualizar dados de utentes;
- ver o registo de eventos diários.

Capítulo 3

Abordagem

Conteúdo

3.1	Arquitetura	8
3.2	Pulseira/Banda NFC	10
3.2.1	<i>Chips</i> NFC	10
3.3	Modelo de Dados	11

3.1 Arquitectura

A arquitetura do sistema foi construída para responder às funcionalidades necessárias que foram descritas no capítulo 2. No desenho do diagrama, foi analisada a forma como cada bloco comunicaria, e com que outros blocos, de modo a obter uma arquitetura estável e escalável. Deste modo, para cada componente, foi escolhida a tecnologia que satisfaria o funcionamento requerido, tendo sido proposto o diagrama ilustrado na figura 3.1.

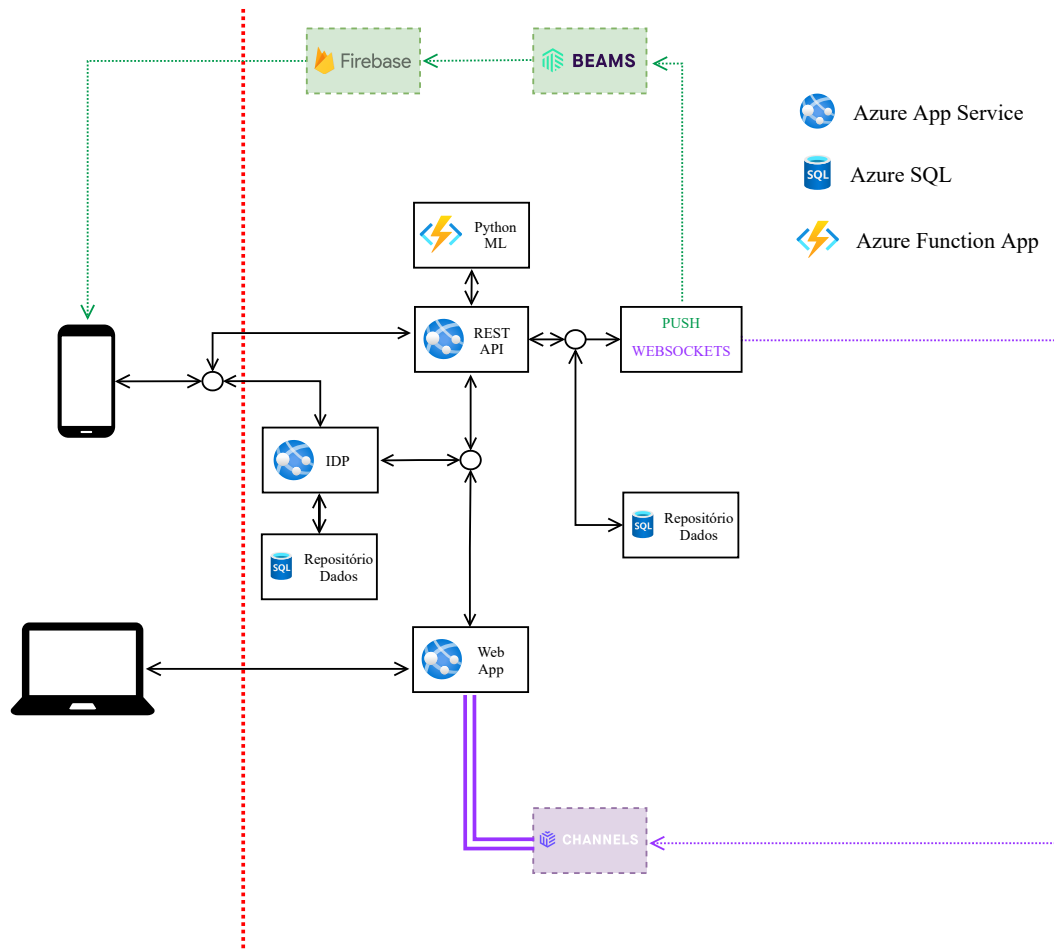


Figura 3.1: Arquitectura da solução

Do lado esquerdo da figura, separados pela linha vermelha, estão visíveis os dispositivos cliente. O *browser* do utilizador comunica com a aplicação *web* que por sua vez interage com o componente *Identity Provider* (IDP) para se autorizar e autenticar. Após a autenticação, esta fica habilitada a fazer pedidos destinados à *web Application Programming Interface* (API).

O cliente móvel, suportado por uma aplicação *Android*, partilha o mesmo modo de

operação que a cliente *web* sem a necessidade de intermediação de outro módulo como acontece na comunicação entre o *browser* e a aplicação *web*.

A API, componente central do sistema, comunica naturalmente com todos os outros elementos. Uma dessas interações é com o repositório de dados do sistema, que guarda o estado das entidades de domínio participantes. Este repositório de dados é suportado por duas bases de dados, sendo uma delas relativa ao sistema SALI e outra apenas ao IDP.

Para notificar os clientes conectados através de qualquer um dos canais disponíveis (*web* e móvel), a *web* API faz uso do módulo de notificações.

A aplicação móvel é notificada através de eventos *push*, e sendo esta direcionada para o sistema operativo *Android*, foi obrigatório configurar os parâmetros de *cloud messaging* do *Firebase Google* (2021) para habilitar a ligação com o serviço *Beams Pusher* (2021b).

O uso do *Beams* adiciona uma camada de abstração, libertando as dependências diretas de um *vendor*, o que possibilita estender o serviço também a dispositivos *iOS*.

A aplicação *web* é notificada através da receção de mensagens via *websocket*. Para gerir este protocolo, foi escolhido o serviço *Channels Pusher* (2021a), que adicionado ao *Beams* completam a totalidade de ferramentas de notificação disponíveis, geridas pela empresa Pusher. Deste modo existe uma maior facilidade de manutenção das operações, por ambas serem administráveis através do mesmo portal de gestão.

Por último, o fluxo de dados sobranete é destinado ao componente de diagnóstico de doenças. Este é usado sempre que é ordenado um pedido de diagnóstico, ativando um servidor que responde usando a especificação *HyperText Transfer Protocol* (HTTP), fazendo uso do modelo de dados, previamente treinado e testado.

Todos os componentes, para além dos fornecidos pela empresa *Pusher*, são instanciados pelo serviço *cloud Microsoft Azure* (*cloud* pública da *Microsoft*).

Para a implantação de cada um dos componentes foram usados os seguintes serviços de *cloud*:

- Web API - *Azure App Service* 
- Aplicação *web* - *Azure App Service* 
- Componente IA - *Azure Function App* 
- Repositórios de dados - *Azure SQL* 

Como critério de desenho, tentou-se separar ao máximo as responsabilidades, para que no futuro seja fácil a alteração dum determinado módulo, não implicando alterações nos restantes.

3.2 Pulseira/Banda NFC

Para confirmar a identidade do paciente e assim cumprir com o requisito mencionado na secção 2.1, foram analisadas várias alternativas técnicas e aquela que venceu foi o uso de NFC.

Tecnologias como *Bluetooth* ou códigos *Quick Response* (QR) foram descartadas porque requerem alguma forma de manutenção.

A primeira precisa de uma fonte de energia, além do módulo de comunicações o que aumenta o tamanho da pulseira, requer carregamentos periódicos e dificulta a impermeabilização.

A segunda necessita que o dispositivo do funcionário tenha capacidade fotográfica e requer que existam boas condições de luz ambiente para a correta leitura (dificuldade de leituras noturnas).

A alternativa escolhida foi o uso de *chips* NFC porque não requerem fontes de energia internas, permitem leituras em quaisquer condições e devido ao seu tamanho compacto, a impermeabilização é conseguida através de uma pulseira em silicone.

Assim, a manutenção da banda é nula durante a estadia do paciente.

3.2.1 *Chips* NFC

Com diversas especificações de *chips*, optou-se pela *NTAG213* (figura 3.2). Esta tem capacidade suficiente para guardar o identificador de cada utente, e é compatível com a grande maioria dos leitores presentes nos dispositivos móveis.

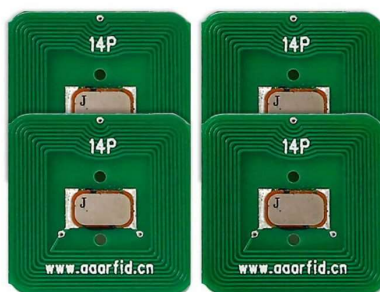


Figura 3.2: Chips NTAG213

O grupo comprou *chips* NFC integrados em pulseiras de silicone a dois fornecedores diferentes, um localizado no Reino Unido e o outro na República Popular da China.

O vendedor do Reino Unido é mais caro 85%, mas em contrapartida o tempo de entrega é menor.

O preço final por pulseira, ronda os 1,50€, optando-se pelo fornecedor chinês, podendo ainda ser mais reduzido caso sejam feitas encomendas de maiores quantidades.

Na figura 3.3 estão visíveis as pulseiras adquiridas.



Figura 3.3: Pulseiras NTAG213

3.3 Modelo de Dados

O modelo relacional representado na figura 3.4, contém as entidades necessárias para representar todas as funcionalidades propostas no capítulo 2. É de notar algumas restrições no nosso modelo:

- Um paciente terá que ter sempre, em todos os turnos, um funcionário designado e também um quarto designado. Existem tarefas comuns a todos os pacientes, mas também existe a possibilidade de adicionar tarefas personalizadas. O registo de um paciente no sistema será feito pelo administrador que, por sua vez, produz uma notificação no sistema direcionada aos funcionários.
- Um funcionário pode ser um auxiliar de saúde ou um enfermeiro e, no sistema, apenas podem estar registados num turno de trabalho. Os enfermeiros não têm nenhum paciente atribuído, mas podem excecionalmente registar ocorrências e realizar diagnósticos tal como os auxiliares de saúde. Os auxiliares de saúde podem ainda registar tarefas.

- Uma tarefa pode necessitar da leitura da pulseira NFC de um paciente para o seu registo ficar completo. Cada tarefa tem um tempo limite que terá de ser respeitado.
- O registo de uma ocorrência implica o envio de uma notificação para o administrador.
- O diagnóstico é uma funcionalidade de apoio aos funcionários, sendo o seu uso opcional. Se o diagnóstico for positivo, isso implica o envio de uma notificação para o administrador.
- Os eventos são acontecimentos extraordinários (p. ex. festas, visitas, etc.). Esta informação será direcionada tanto para o administrador, como para os funcionários.

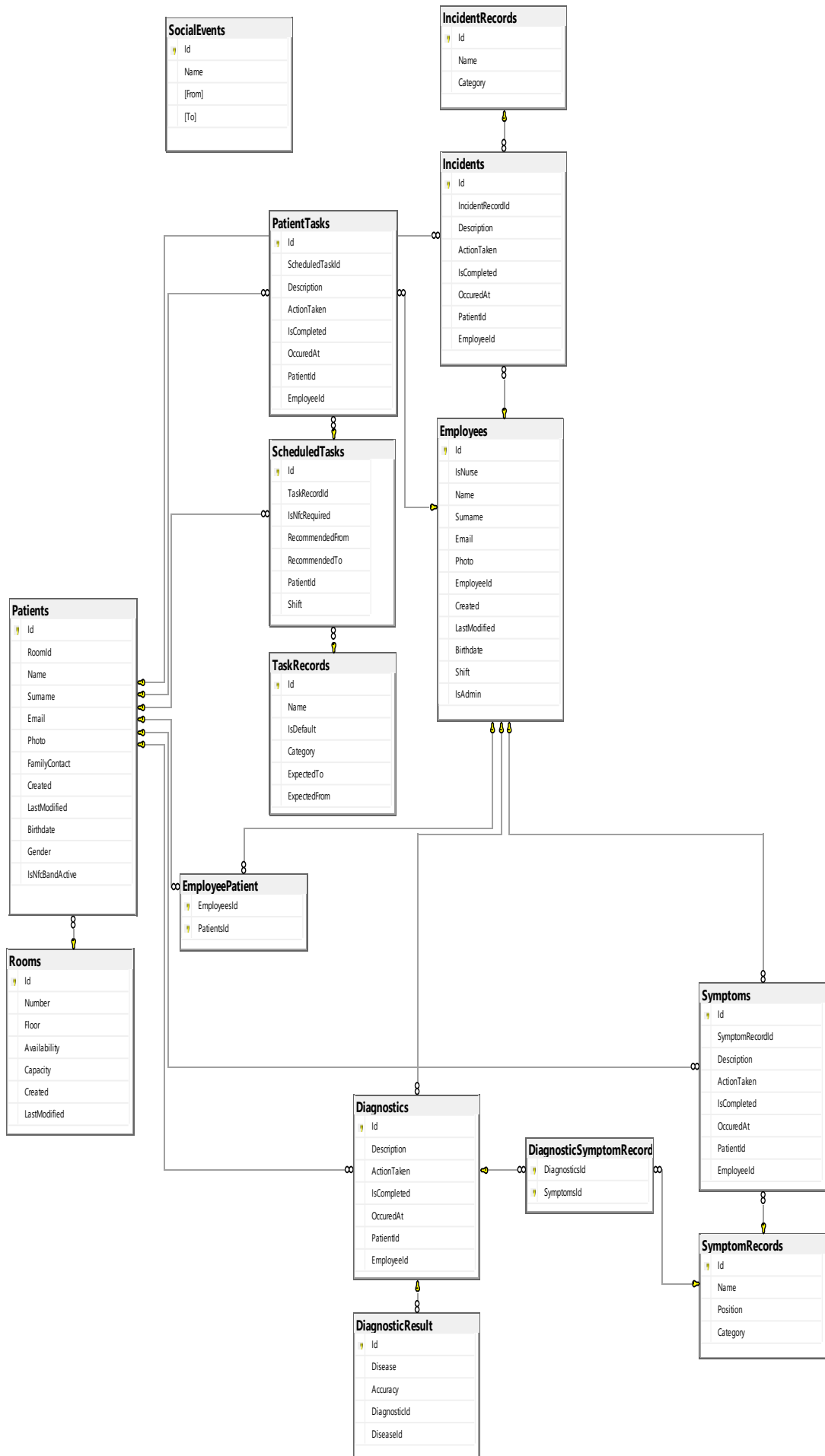


Figura 3.4: Modelo relacional

Capítulo 4

Servidor de Identidade

Conteúdo

4.1	Implementação	16
4.2	Arquitetura	18
4.3	Peças de Software Auxiliares	18
4.4	Funcionalidades	19

A generalidade das aplicações públicas tem recursos protegidos/privados. Para validar se o utilizador, pode ou não, aceder ao recurso, têm de existir mecanismos de autorização e autenticação que avaliem as suas permissões.

Num sistema com vários módulos, como é o proposto, caso não existisse um componente responsável pela verificação de identidade, haveria verificações em todos os componentes, duplicando assim código-fonte e aumentando a dificuldade de manutenção dos próprios módulos.

Com a solução de centralizar todos esses requisitos num único componente, o servidor de identidade, flexibilizou-se toda a solução para suportar, no futuro, mais clientes ou componentes.

A figura 4.1 ilustra as rotas de dados entre os módulos da solução e o utilizador final.

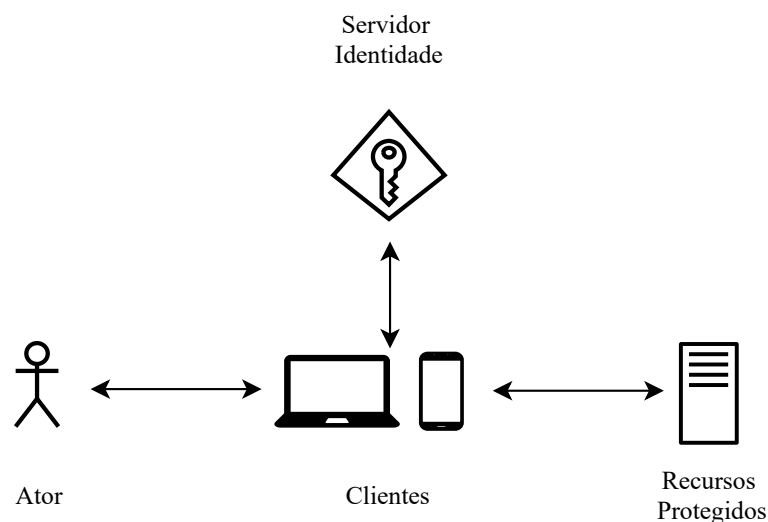


Figura 4.1: Diagrama de utilização do servidor de identidade

4.1 Implementação

Para o desenho deste elemento recorreu-se ao protocolo OpenID-Connect (2021), que consiste numa camada de identidade simples sobre o protocolo OAuth-2.0 (2021).

Este permite que os clientes verifiquem a identidade do utilizador com base na autenticação realizada por um servidor de autorização, bem como que obtenham informações básicas de perfil sobre o utilizador de maneira interoperável e semelhante a uma API REST.

O *OpenID Connect* permite que os clientes, solicitem e recebam informações sobre sessões autenticadas e utilizadores. O conjunto de especificações é extensível, permitindo

que os participantes usem recursos opcionais, como criptografia de dados de identidade, descoberta de provedores *OpenID* e gerenciamento de sessão, quando assim for necessário.

A solução escolhida que facilita a utilização e configuração deste protocolo denomina-se *Identity Server 4 IdentityServer4* (2021).

O *Identity Server 4* é um *middleware* que adiciona vários *endpoints* compatíveis com as especificações documentadas para os protocolos *OpenID Connect* e *OAuth 2.0*, a um projecto *.NET Core*. O *Identity Server 4* é *open-source* e grátis.

A tabela 4.1 resume as principais características do componente *Identity Server*.

Tabela 4.1: Características do *Identity Server*

	Identity Server 4
Suporte	Documentação
Infraestrutura	Responsabilidade do desenvolvedor
Preço	Grátis
Primeiros Passos	Difícil
Configurações	Código-fonte projecto <i>.NET</i>

Uma das alternativas possíveis Braybrook (2021), e a única que foi testada além da solução escolhida, chama-se *Azure B2C* Microsoft (2021g). Após criação de um projeto de teste e configuradas as bibliotecas de autenticação nos clientes móvel e web, retiraram-se várias conclusões que invalidaram o seu uso na solução proposta. A falta de flexibilidade na utilização e o preço ditaram o seu insucesso.

A tabela 4.2 resume as principais características do serviço *Azure B2C*.

Tabela 4.2: Características do *Azure B2C*

	Azure B2C
Suporte	Documentação + Linha apoio cliente <i>Azure</i>
Infraestrutura	Escolhida pelo <i>Azure</i>
Preço	Depende do número de utilizadores
Primeiros Passos	Fácil
Configurações	Portal <i>Azure</i>

4.2 Arquitetura

O padrão de arquitetura *Model View Controller* (MVC) separa uma aplicação em três grupos principais de componentes: Modelos, Vistas e Controladores.

Este padrão ajuda a conseguir a separação de responsabilidades. Usando este padrão, os pedidos do utilizador são encaminhados para um controlador que é responsável por interagir com o modelo para realizar as ações pretendidas.

O controlador escolhe a vista a ser exibida ao utilizador e fornece-lhe todos os dados do modelo necessários.

A figura 4.2 reflete as interações entre os grupos de componentes.

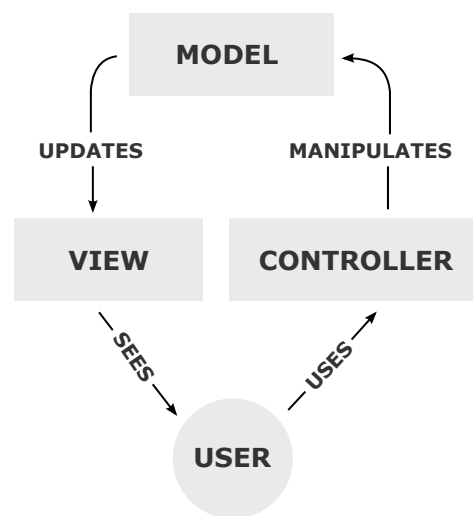


Figura 4.2: Esquema Arquitetura MVC

4.3 Peças de Software Auxiliares

Algumas das necessidades do servidor de identidade foram suprimidas com o uso de pacotes *NuGet Microsoft (2021c)* públicos, designadamente:

- *Entity Framework Core* Microsoft (2021d) – *mapper* de base de dados para objetos *.NET*. Suporta *LINQ*, controlo de alterações, atualizações e migrações de *schema*.
- *IdentityServer4.AspNetIdentity* Microsoft (2021f) – pacote para lidar com autorização e autenticação de uma aplicação *.NET Core*.
- Serilog (2021) – utilitário para *logs* na consola.

4.4 Funcionalidades

Através das configurações para cada cliente do sistema, foi possível escolher qual o fluxo de informação a ser usado.

Atualmente, o fluxo recomendado, quer para clientes móveis, quer *web*, é o Fluxo de Autorização com extensão *Proof Key for Code Exchange* (PKCE) OAuth (2021).

O PKCE é uma extensão de segurança ao protocolo *OAuth 2.0* para clientes públicos, com o objetivo de evitar que o código de autorização seja interceptado.

É útil principalmente para aplicações que usem o *client secret* (estático) atribuído aquando do registo sendo este substituído pelos parâmetros *code verifier* e *code challenge* (dinâmicos).

Na figura 4.3 é possível observar as diferentes rotas de informação trocadas entre todos os componentes da solução. A secção sombreada identifica o fluxo de pedidos no servidor de identidade.

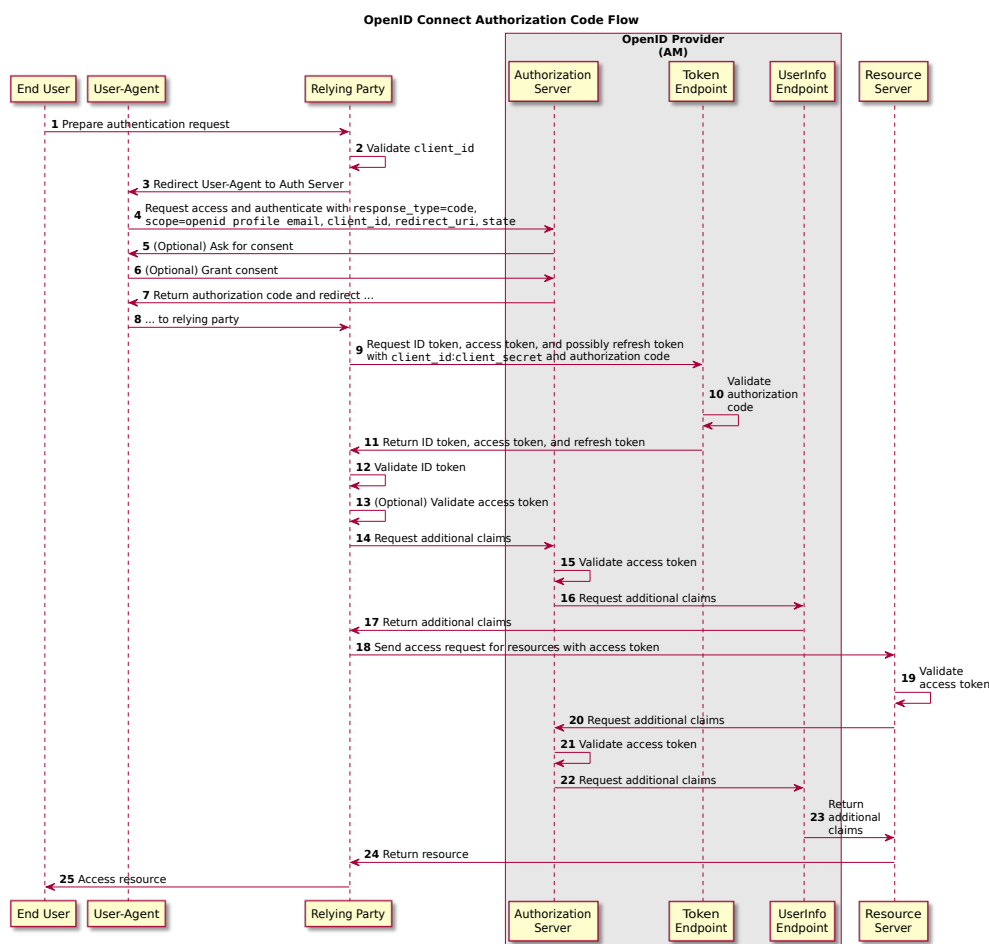


Figura 4.3: Fluxo de informação entre IDP, clientes e recursos protegidos

A extensão PKCE adiciona ao fluxo normal de autorização os seguintes passos:

1. As aplicações cliente geram um *code verifier* seguido de um *code challenge*.

A figura 4.4 reproduz um exemplo do formato dos dois requisitos.

```
{
  "code_verifier": "M25iVXpKU3puUjFaYWg3T1NDTDQtcW1R0UY5YXlwaLNoc@hhakxlfmZHag",
  "code_challenge": "qjrzSW9gMiUgpUvqgEPE4_-8swvyCtf0Vvg55o5S_es"
}
```

Figura 4.4: Exemplo de *code verifier* e *code challenge*

2. O utilizador é redirecionado para a página de *login* (visível na figura 4.5) do IDP juntamente com o *code challenge*, e autentica-se.

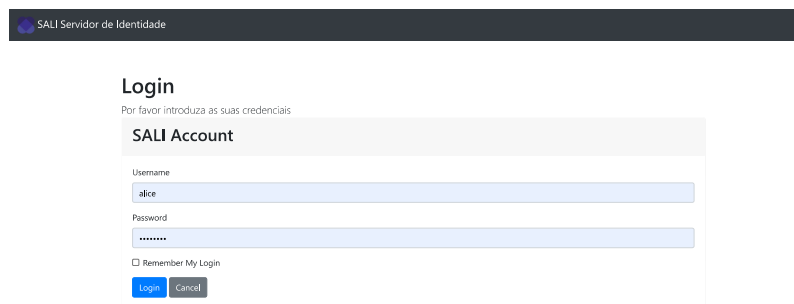


Figura 4.5: Página de *login* do IDP

3. O IDP redireciona o utilizador de novo para a aplicação cliente, e junta o *authorization code*.
4. Os clientes enviam o *authorization code* e o *code verifier* para o IDP. O servidor de identidade retorna os *access* e *ID tokens*, e opcionalmente um *refresh token*.

5. Por fim, os clientes estão agora habilitados a passar esses *tokens* para um recurso protegido (ex. uma API) em nome do utilizador autenticado.

A figura 4.6 espelha a configuração necessária no IDP para registo de uma nova aplicação cliente.

```
new Client {  
    ClientId = "React SPA",  
    ClientName = "SALI Frontend",  
    AllowedGrantTypes = GrantTypes.Code,  
    ClientSecrets = new List < Secret > {  
        new("SuperSecretPassword".Sha256())  
    },  
    AllowOfflineAccess = true,  
    AllowAccessTokensViaBrowser = true,  
    RequirePkce = true,  
    AllowedScopes = {  
        "saliapi.read",  
        "saliapi.write",  
        "roles",  
        StandardScopes.OpenId,  
        StandardScopes.Profile,  
        StandardScopes.OfflineAccess  
    },  
    RequireClientSecret = false,  
    RedirectUris = new [] {  
        "http://localhost:4200/signin-callback.html",  
        "https://dev-ft-isel-sali.azurewebsites.net/signin-callback.html"  
    },  
    PostLogoutRedirectUris = new List < string > () {  
        "http://localhost:4200/",  
        "https://dev-ft-isel-sali.azurewebsites.net/"  
    },  
    RequireConsent = false,  
}
```

Figura 4.6: Exemplo de configuração de uma aplicação cliente

Capítulo 5

Web API

Conteúdo

5.1	Implementação	24
5.2	Arquitetura	24
5.2.1	Domínio	25
5.2.2	Aplicação	25
5.2.3	Infraestrutura	25
5.2.4	UI	26
5.3	Peças de <i>Software</i> Auxiliares	26
5.4	Funcionalidades	26

O presente capítulo descreve a *web API*, que se constitui um componente central do sistema.

Todas as operações sobre as entidades de domínio da solução, são realizadas através de pedidos HTTP destinados à API.

5.1 Implementação

A linguagem escolhida foi *C#* complementada com a *framework ASP.NET Core Microsoft (2021h)* versão 5. As razões dessa escolha prenderam-se com o facto de durante o curso não ter havido oportunidade de explorar as capacidades do *.NET* para o desenvolvimento de aplicações *web*, a curiosidade de aprofundar os conhecimentos de Sistemas de Informação 2 (*SI2*) na camada de acesso a dados, utilizando desta vez uma abordagem *code-first*, e por fim, fazer uso da facilidade de integração com os serviços *Azure* diminuindo o tempo dispendido na construção das rotinas de *build* e *deploy*.

5.2 Arquitetura

A construção do plano da arquitetura foi pensada para se reger pelos mesmos pilares da *Clean Architecture* Martin (2021a).

O conceito tornado famoso por *Robert C. Martin* estabelece um conjunto de regras por forma a que por exemplo os princípios *SOLID* Martin (2021b) sejam respeitados, e que existam separações de responsabilidade entre os componentes.

A *Clean Architecture* leva à separação entre as regras de negócios estáveis (abstrações de nível superior) e os detalhes técnicos voláteis (detalhes de nível inferior), definindo limites claros.

As principais características são:

- testável;
- independente de *frameworks*;
- independente da *UI*;
- independente da base de dados.

Foi seguida a adaptação de Jason Taylor (2021b) da *Clean Architecture* aplicada a um projeto Taylor (2021a) .NET.

A figura 5.1 ilustra os diferentes anéis de responsabilidades onde os módulos estão organizados.

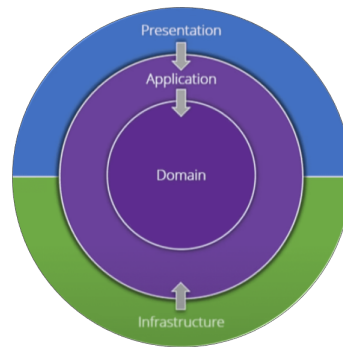


Figura 5.1: *Clean Architecture*

5.2.1 Domínio

No domínio, parte nuclear da arquitetura, residem todas as entidades do sistema. No contexto da solução descrita no presente documento, as definições de paciente ou de tarefa, podem ser encontradas aqui.

As entidades de domínio raramente sofrem alterações, ficando estáveis durante o ciclo de vida do projeto.

5.2.2 Aplicação

Na aplicação estão os comandos (envolvem escritas) e as *queries* (envolvem só leituras) que especificam as operações suportadas.

5.2.3 Infraestrutura

Esta camada contém classes para aceder a recursos externos, como bases de dados e serviços da *web*. As classes devem ser baseadas em interfaces definidas na camada de aplicação, permitindo assim desacoplar a ligação com as implementações concretas Sete (2021).

5.2.4 UI

Parte responsável pela interface HTTP e pela apresentação dos resultados das operações solicitadas. Consoante o resultado da operação, este módulo decide o *status code* do pedido, assim como, se necessário, a respetiva mensagem de erro.

5.3 Peças de Software Auxiliares

Algumas das necessidades da API foram suprimidas com o uso de pacotes *NuGet Microsoft (2021c)* públicos.

Foram utilizados os seguintes pacotes:

- *Entity Framework Core 5 Microsoft (2021e)* – pacote utilizado para aceder aos dados na BD - serve de *object-relational mapper (O/RM)*.
- *MediatR Bogard (2021b)* – pacote utilizado para mediar a execução de comandos.
- *AutoMapper Bogard (2021a)* – pacote utilizado para mapear objectos de tipos diferentes.
- *Fluent Validation GitHub (2021)* – pacote utilizado para validar os campos dos pedidos.
- *NSwag Suter (2021)* – pacote utilizado para implementar normas *OpenApi Initiative (2021)*.
- *Pusher Server Pusher (2021a)* – pacote utilizado para comunicar com o serviço *Channels*.
- *Identity Server 4 - Access token validator IdentityServer (2021)* – pacote utilizado para validar os *tokens* de acesso.
- *Nunit Prouse (2021)* – pacote utilizado para testes unitários.

5.4 Funcionalidades

A *web API*, desenvolvida como parte integradora e de convergência dos diferentes canais (*web* e *móvel*), disponibiliza as seguintes operações:

- criar, apagar e modificar os dados pessoais dos pacientes;

- ler os detalhes dos pacientes;
- personalizar a agenda/plano dos pacientes;
- ler as tarefas suportadas pelo lar;
- ler as tarefas reportadas no lar;
- reportar as tarefas concluídas;
- ler as ocorrências suportadas pelo lar;
- ler as ocorrências reportadas no lar;
- reportar as ocorrências sucedidas;
- ler os sintomas suportados pelo módulo IA;
- reportar sintomas observados;
- ler métricas sobre o lar.

Capítulo 6

Aplicação Web

Conteúdo

6.1	Implementação	30
6.2	Arquitetura	31
6.2.1	<i>App.js</i>	31
6.2.2	<i>Containers/Componentes</i>	32
6.2.3	<i>Services</i>	32
6.3	Peças de Software Auxiliares	32
6.4	Funcionalidades	33

Este capítulo pretende explicar como é implementado um dos lados do cliente do sistema, que neste caso será utilizada pelo administrador, a aplicação web. Esta aplicação dispõe de funcionalidades para realizar todas as operações de gestão de um lar e é responsável pela apresentação de informações que carecem de uma análise rigorosa e em alguns casos imediata, por parte dos administradores do lar, como por exemplo, o relatório de um incidente.

6.1 Implementação

Esta aplicação foi construída usando *React Facebook (2021c)*, uma biblioteca *JavaScript* que se foca na gestão de estado de componentes e na apresentação de componentes reutilizáveis sem estado. Esta biblioteca abstrai o programador do *Document Object Model* (DOM), oferecendo um desempenho muito bom e um modelo de programação simplificado.

Os elementos importantes do *React* são:

- Componentes - no paradigma de programação do *React*, todos os elementos gráficos, a apresentar na *User Interface* (UI) são componentes.
- *JSX (2021)* - extensão *JavaScript* que permite escrever *HyperText Markup Language* (HTML) em *React* de uma maneira mais simples e eficaz.
- Virtual DOM - objeto *JavaScript* guardado em memória que representa a UI e está sincronizado com o DOM real. Garante melhor desempenho visto que o seu uso é mais rápido que o DOM real.

Como dito anteriormente, esta biblioteca foca o seu funcionamento na gestão de estado de componentes para assim poder mostrar os elementos visuais pretendidos dependendo do estado dos componentes. Em *React* existem dois paradigmas principais que podem ser usados para gestão de estado num componente:

- O primeiro paradigma, que é o *legacy*, determina que os componentes que necessitam de gerir estado têm de ser declarados como classe e derivar de *React.Component* para poderem aceder ao objeto *state*. Existem também componentes de demonstração que não gerem estado podendo ser definidos apenas como funções.
- O segundo paradigma implementa *hooks*, que são um conjunto de métodos adicionados na versão 16.8 do *React*. Permitem que todos os componentes possam ser declarados como funções e mesmo assim poderem gerir o seu estado.

A aplicação *web* SALI foi desenvolvida sob o segundo paradigma (*hooks*), anteriormente descrito, pois este é considerado superior ao original da biblioteca Prasanjith (2021). Em primeiro lugar não determina que os componentes de demonstração tenham de ser funções, permitindo que ao longo do escalamento da aplicação não seja preciso mudar um componente de função para classe ou vice-versa. Outra vantagem é a de evitar o uso do “*this*”, elemento sempre presente na noção de classe. Ao optar pelo uso de *hooks* o programador abstrai-se do uso desta palavra visto que pode causar problemas devido ao contexto especial desta em *JavaScript*.

6.2 Arquitetura

A figura 6.1 ilustra, em forma de blocos, a estrutura da arquitetura da aplicação *web*.

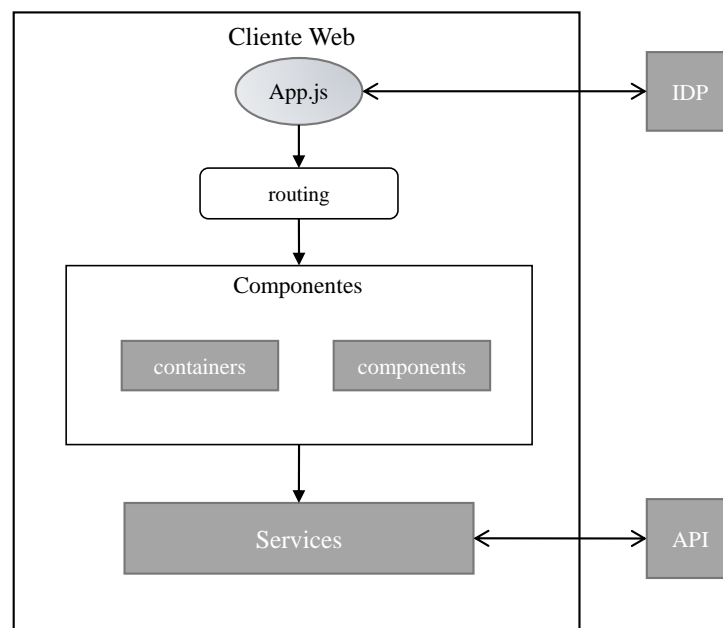


Figura 6.1: Arquitetura da aplicação *web*

6.2.1 *App.js*

Este ficheiro representa a entrada lógica da aplicação e é responsável pela comunicação com o módulo IDP, que resulta na autenticação de um utilizador para aceder à aplicação *web*.

Após a autenticação com sucesso, o utilizador é redirecionado para a página principal.

6.2.2 *Containers/Componentes*

Os *containers* são o conjunto de componentes que precisam de gerir estado. Estes componentes utilizam os métodos disponíveis pelos *React Hooks* para conseguirem ter o funcionamento desejado.

Estes componentes para além de gerirem o seu estado, usam os componentes de demonstração, por meio de atribuição de propriedades, para assim serem expostos na *User Interface* (UI) da aplicação com a informação necessária.

6.2.3 *Services*

Os *services* são o conjunto de ficheiros que contêm os métodos responsáveis pela obtenção de dados a partir da comunicação com a *Web API*. Estes métodos são chamados nos *containers* para preencherem os elementos visuais necessários para o funcionamento e propósito de cada ecrã.

6.3 Peças de Software Auxiliares

Foram utilizados os seguintes módulos de software auxiliares:

- *Axios* Axios-Project (2021) – pacote utilizado nos *services* para realizar pedidos HTTP ao servidor.
- *CoreUI* Creative-Labs (2021) – pacote que inclui componentes visuais que usam na sua base *Bootstrap* mas que estão adaptados às melhores práticas para aplicações do tipo “*Admin Dashboard*”.
- *Pusher-js* Pusher-Limited (2021) – pacote que realiza a comunicação com os canais *Pusher* por onde a aplicação gere notificações e atualizações de dados de uma forma instantânea.
- *React Router DOM* React-Training (2021) – pacote que inclui diversos componentes de navegação da aplicação.
- *OIDC-React* AS (2021) - pacote que inclui um componente *React* que fornece suporte ao uso dos protocolos *OpenID Connect* e *OAuth2*. Este componente é usado na comunicação da aplicação com o IDP.

- *Jest Junit* Facebook (2021a) - pacote que inclui ferramentas que permitem a realização de testes unitários.
- *Enzyme* Airbnb (2021) - pacote que integra testes relativos a componentes *React*.
- *DotEnv* Motte (2021) - pacote que permite o reconhecimento e possibilidade de declarar e usar variáveis no programa conforme o tipo de ambiente.
- *EnvCmd* Bluhm (2021) - pacote que permite correr a aplicação no modo de produção ou desenvolvimento.
- *React-Redux* Abramov (2021) - pacote que contém uma biblioteca para gerir estados.

6.4 Funcionalidades

Esta aplicação está direcionada para a utilização por parte do administrador do lar e tem o seguinte leque de funcionalidades disponíveis:

- visualização de informações sobre o lar de forma instantânea;
- criação e remoção de eventos sociais que acontecem no lar;
- visualização da lista de utentes do lar;
- criação de um utente;
- visualização dos detalhes de um utente;
- atribuição de funcionários a um utente;
- realização de um plano de tarefas personalizado por cada utente;
- visualização da lista de funcionários do lar;
- visualização dos detalhes de um funcionário;
- atribuição de utentes a um funcionário;
- visualização do histórico da realização de tarefas e de ocorrências de todos os utentes e funcionários;
- receção de notificações em caso de ocorrência ou diagnóstico positivo;

- visualização da disposição dos quartos do lar e os seus ocupantes;
- atribuição de quartos a utentes.

Capítulo 7

Aplicação Móvel

Conteúdo

7.1	Implementação	36
7.2	Arquitetura da Aplicação Móvel	37
7.2.1	<i>Componentes</i>	37
7.2.2	<i>App.js</i>	38
7.2.3	<i>Services</i>	38
7.2.4	<i>Navegação</i>	38
7.2.5	<i>Notificações</i>	39
7.3	Peças de Software Auxiliares	40
7.4	Funcionalidades	41

Este capítulo pretende explicar como foi implementado o cliente móvel, que é utilizado pelos funcionários do lar. Esta aplicação dispõe de funcionalidades que permitem o acesso a informações sobre os pacientes, realização de diagnósticos, gestão e registo de tarefas, bem como leitura de pulseiras NFC com o número identificador do paciente. De modo a confirmar a identidade dos pacientes, algumas operações são apenas acessíveis através da leitura das pulseiras NFC.

7.1 Implementação

A aplicação móvel tem apenas suporte para a plataforma *Android*, sendo por isso necessário um dispositivo com esse sistema operativo para a sua utilização. Escolheu-se realizar o projeto apenas para sistemas *Android* devido a restrições de tempo.

A escolha deste sistema em oposição ao *IOS* e *Windows* dispõe de várias vantagens. Uma das vantagens é a popularidade do sistema *Android* que segundo O’Dea (2021) é o mais usado na atualidade, permitindo abranger um maior número de dispositivos. Do ponto de vista de desenvolvimento, como todos os membros do grupo têm equipamentos com esse sistema, são facilitados os testes da aplicação num ambiente real.

A *framework* escolhida no âmbito da aplicação móvel deste projeto foi *React Native Facebook* (2021b). A decisão foi fundamentada pela intenção dos autores em estudar a infraestrutura *React*. Assim, usando o mesmo paradigma da aplicação *web*, tal como referido no capítulo 6, foi facilitada a prática do desenvolvimento dos clientes e futura manutenção do projeto.

7.2 Arquitetura da Aplicação Móvel

A figura 7.1 ilustra o diagrama de arquitetura da aplicação móvel.

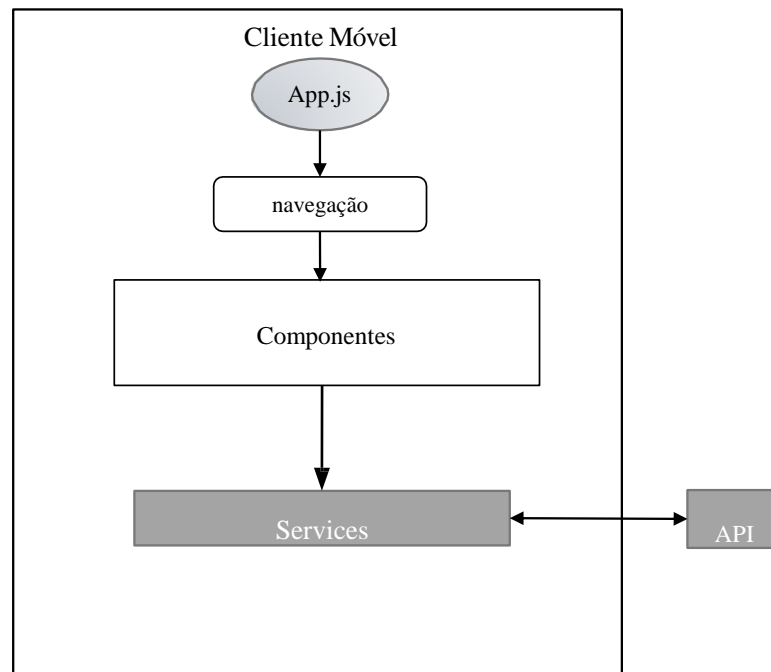


Figura 7.1: Arquitetura da aplicação móvel

7.2.1 Componentes

Ao contrário de outras *frameworks*, o *React Native* não tem nenhuma arquitetura padrão definida. A sua estruturação consiste numa árvore de componentes, sendo que os de maior ordem na árvore contêm os de menor ordem.

Cada componente pode ter estado ou não, oferecendo uma propriedade dinâmica que controla a sua vista.

A figura 7.2 representa a árvore de componentes mencionada.

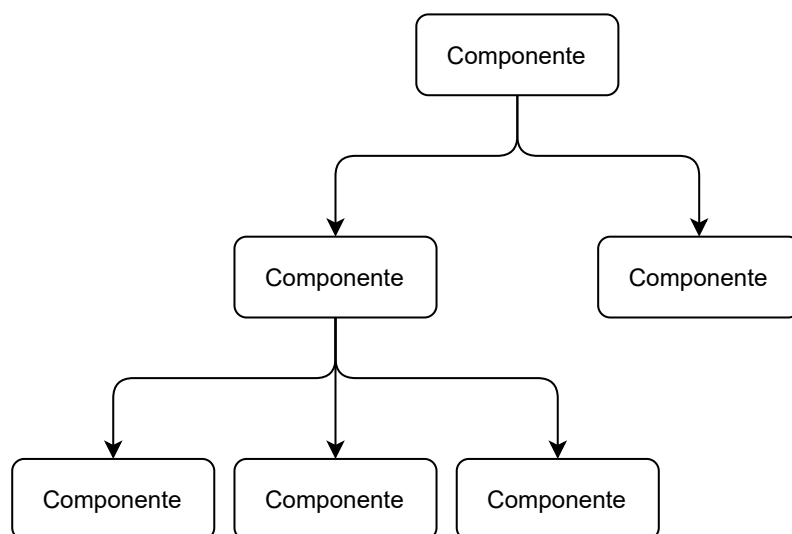


Figura 7.2: Árvore de componentes

7.2.2 *App.js*

Por ser o componente de maior ordem, é este que chama o módulo responsável pela navegação da aplicação. É também aqui onde é realizada a subscrição para a receção de notificações *push*.

7.2.3 *Services*

Representa o conjunto de métodos responsáveis pela realização de pedidos HTTP à *web* API. Estes métodos são usados em componentes que possuem estado.

7.2.4 *Navegação*

Para a navegação entre ecrãs, recorreu-se à biblioteca *React Navigation Osadnik (2021)*. Esta para além de permitir a transição entre ecrãs, faz a gestão do histórico de ecrãs e contém um conjunto de animações, típicas do sistema *Android*.

A navegação foi implementada com recurso às bibliotecas *React Navigation Drawer* e *React Navigation Stack* que pertencem ao pacote *React Navigation*.

A biblioteca *React Navigation Drawer* permite adicionar um menu lateral. Por sua vez, cada entrada neste contém um *stack* de ecrãs possíveis de serem navegados, disponibilizado pelo pacote *React Navigation Stack*.

7.2.5 Notificações

A aplicação móvel tira partido do mecanismo de notificações aquando da inserção de um novo utente no sistema e é necessário o registo na pulseira com as informações do mesmo. Os funcionários são assim alertados para a realização deste procedimento.

A implementação das notificações sustenta-se na biblioteca React Native Firebase *Invertase* (2021) para o uso da *Firebase Cloud Messaging* (FCM) e na biblioteca Pusher Beams *Pusher* (2021b). O *Pusher Beams* recorre à FCM para o envio de notificações *push* para aplicações do sistema *Android*. Quando a aplicação é iniciada, o dispositivo fica subscrito às notificações do serviço *Beams*.

As situações de funcionamento possíveis são:

- Aplicação em *background* ou terminada – Quando o dispositivo recebe uma notificação, após esta ser clicada pelo utilizador, a aplicação é iniciada no ecrã de autenticação. Após a introdução das credenciais com sucesso, surge um componente clicável que leva a aplicação a navegar para um ecrã com a lista de pacientes inseridos no sistema, mas com a pulseira por registar. Ao clicar num desses pacientes, é possível a transferência do número de identificação do paciente para a pulseira.
- Aplicação em *foreground* – Após o clique na notificação, através de *deep linking*, é mostrado automaticamente um ecrã com os dados do novo paciente. Após a confirmação dos detalhes pessoais do utente, a pulseira pode receber o respetivo identificador único.

7.3 Peças de Software Auxiliares

Foram usados os seguintes subsistemas auxiliares:

- *Axios Axios-Project (2021)* – pacote utilizado nos *services* para realizar pedidos HTTP ao servidor.
- *React Native Paper Callstack (2021)* – pacote que inclui componentes visuais.
- *React Native Push Notifications Trujillo (2021)* – pacote para a subscrição de notificações *push* com a aplicação em *foreground*.
- *React Native Pusher Push Notifications Sutter (2021)* – pacote para a subscrição de notificações *push* com a aplicação em *background*.
- *React Native Firebase Invertase (2021)* – pacote responsável pela receção de notificações através da FCM.
- *React Navigation Osadnik (2021)* – pacote que inclui diversos componentes de navegação da aplicação.
- *React Native NFC Manager RevtelTech (2021)* - pacote com a implementação das funções necessárias para a leitura e escrita de pulseiras NFC.
- *React Native App Auth Formidable-Labs (2021)* - pacote responsável pela autenticação.
- *React Native Inappbrowser Reborn Proyecto26 (2021)* - pacote para abrir o *browser* dentro da aplicação móvel.
- *React Native Swipe List View Sessler (2021)* - pacote utilizado para poder arrastar um elemento de uma lista para a esquerda ou direita no ecrã.
- *React Native Tabs Aksonov (2021)* - pacote que serve para utilizar abas.
- *React Redux Abramov (2021)* - pacote que contém uma biblioteca para gerir estados.
- *Jest Junit Facebook (2021a)* - pacote que inclui ferramentas que permitem a realização de testes unitários.
- *App Center Microsoft (2021a)* - pacote que inclui ferramentas para monitorização da aplicação com dados de análise e diagnóstico de erros.

7.4 Funcionalidades

A aplicação móvel, direcionada para o uso dos auxiliares de saúde, permite:

- visualizar e registar a conclusão de todas as tarefas diárias do auxiliar;
- receber notificações para o registo de um novo utente;
- ler e escrever as pulseiras NFC dos utentes;
- visualizar a lista de todos os utentes do lar;
- registar incidências relativas a um utente;
- realizar um diagnóstico a partir de uma lista de sintomas para previsão de doenças;
- visualizar histórico de ocorrências registadas dos utentes;
- visualizar detalhes de um utente;
- ver o registo de eventos sociais.

Capítulo 8

Componente Diagnóstico de Doenças - IA

Conteúdo

8.1	Implementação	44
8.2	Arquitetura	45
8.3	Peças de Software Auxiliares	45
8.4	Funcionalidades	46

Com o objetivo de expandir e adicionar funcionalidades ao sistema, chegou-se à conclusão que seria oportuno tentar ajudar os funcionários dos lares a detetar possíveis doenças de um utente. Para este efeito integrou-se a capacidade de, após o pedido de um diagnóstico, o sistema assinalar eficazmente uma ou mais possíveis doenças a partir dos sintomas introduzidos pelo auxiliar de saúde do lar. A predição de doenças permite antecipar a procura de ajuda médica profissional. Esta funcionalidade está disponível aos funcionários do lar através da aplicação móvel.

8.1 Implementação

A implementação deste componente está dividida em duas partes. A primeira parte diz respeito à construção do modelo de predição utilizando métodos de Inteligência Artificial (IA). A segunda parte assenta sobre a integração desse mesmo modelo com o resto do sistema.

Visto que o conhecimento sobre a área ao qual a construção do modelo diz respeito não foi abordada durante a licenciatura, decidiu-se recorrer a material público para satisfazer os requisitos do modelo. O resultado dessa pesquisa foi a assimilação do modelo *Pytorch* Zehra (2021) e o *dataset* Patil (2021) no sistema.

Durante a fase de testes, com o objetivo de adaptar e validar o modelo *Pytorch* e o *dataset*, foi utilizado um distribuidor de *Python*, a aplicação *Anaconda Analytics* (2021). Para criar e editar o documento que exhibe o resultado do modelo tirou-se partido da aplicação *Jupyter Notebook Project* (2021).

Antes de poder ser utilizado, o modelo é previamente treinado através de um *script* que emprega regressão logística que é uma técnica de análise de dados resultando numa predição, sendo posteriormente, exportado para um ficheiro com extensão *.pth*.

O uso de regressão logística no modelo não constava do planeamento inicial deste componente. Originalmente, o modelo estava previsto ser treinado aplicando a técnica computacional de redes neurais artificiais, mas devido à pequena dimensão do *dataset* encontrado, por ser insuficiente para treinar o modelo com essa técnica, foi descartada esta opção.

Para a integração deste modelo no sistema, utilizou-se o serviço *Azure Function App*, disponível na *cloud Microsoft Azure*. Esta decisão, baseou-se na capacidade de abstração que o serviço concede ao componente relativamente ao resto do sistema. O modelo e o *dataset* podem ser assim facilmente alterados no futuro sem implicar modificações noutros módulos.

As duas estratégias consideradas para ativar uma *function app* foram o uso de gati-

lhos temporais e gatilhos acionados por pedidos HTTP. No contexto do sistema SALI escolheu-se usar gatilhos HTTP.

8.2 Arquitetura

Este componente comunica com a *web* API de forma bidireccional enviando o diagnóstico dos sintomas pertinentes, retornado a resposta num novo pedido HTTP.

Na figura 8.1, está representada a arquitetura do componente de diagnóstico de doenças.

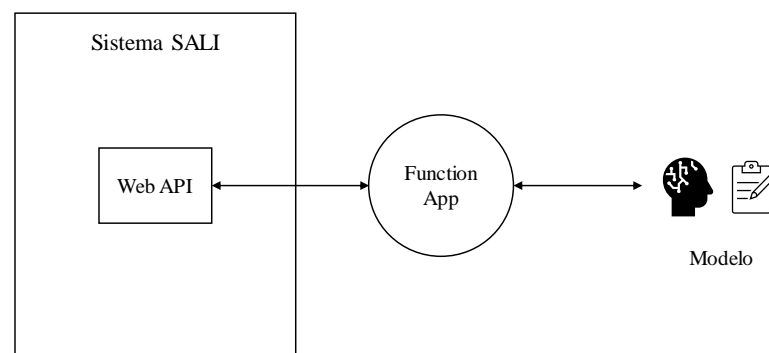


Figura 8.1: Arquitetura do componente de diagnóstico de doenças

8.3 Peças de Software Auxiliares

Foram usados os seguintes subsistemas auxiliares:

- *Azure Functions Microsoft (2021b)* – biblioteca que permite a utilização das *Azure Functions* em *Python*.
- *Pandas McKinney (2021)* – biblioteca que permite a manipulação de dados e análise em *Python*.
- *Numpy Oliphant (2021)* – adiciona suporte para arrays e matrizes multi-dimensionais e várias funções para operar sobre esses tipos.
- *Torch (2021) (FAIR)* – biblioteca com *framework* de *machine learning*.
- *Requests Python Software Foundation (2021)* – biblioteca para realizar pedidos HTTP.

8.4 Funcionalidades

O componente de predição de doenças permite:

- dado um conjunto de sintomas disponíveis no *dataset*, retorna uma lista de possíveis doenças com um grau de certeza associado a cada uma.

Conclusão

O Sistema de Apoio a Lares de Idosos (SALI) é um conjunto de aplicações que vem resolver uma parte dos problemas de gestão dos lares de idosos. Não só soluciona problemas de foro administrativo como também da gestão das necessidades diárias dos utentes.

Todos os requisitos propostos foram executados.

Com a transformação dos requisitos em tarefas, e fazendo uso de diversas ferramentas de gestão de projeto foi possível atingir um nível elevado de autonomia entre os membros do grupo, paralelizando várias linhas de trabalho.

Através da implementação inicial do sistema de *continuous integration/continuous delivery* um grande número de etapas existentes entre a realização programática da tarefa e a sua disponibilização ao utilizador final, foram automatizadas aproximando o ambiente de desenvolvimento deste projeto a qualquer outro da indústria profissional.

A totalidade das aplicações desenvolvidas estão acessíveis sem restrições, de forma pública e segura.

Para concluir, salientar que com a adição do *site* de apresentação do projeto, todo o trabalho desenvolvido está muito próximo de ser a base de uma futura empresa de produto, motivo que orgulha e motiva os constituintes do grupo.

9.1 Trabalho Futuro

Com todas as restrições temporais associadas a um projeto de 6 meses, no planeamento inicial foi adotada uma estratégia de priorização de requisitos, escolhendo aqueles que a curto prazo trouxessem mais valor à proposta.

Assim para implementação futura, foram deixados os seguintes pontos:

- Desenvolvimento de portal de gestão de funcionários.
- Automatização periódica do diagnóstico de doenças.
- Gestão de *stocks* de matérias primas com aviso em caso de rutura.
- Extensão da função de confirmação de identidade à roupa dos utentes.
- Controlo de acessos ao edifício do lar.
- Flexibilização da organização dos turnos dos funcionários.
- Notificações personalizadas dirigidas a um funcionário específico.
- Integração de planos de medicação.

Referências

- Abramov, D. (2021). React-redux. getting started. URL: <https://react-redux.js.org/> acedido pela última vez em 07/06/2021.
- Airbnb (2021). Enzyme. javascript testing utility for react. URL: <https://www.npmjs.com/package/enzyme> acedido pela última vez em 02/05/2021.
- Aksonov, P. (2021). React native tabs. URL: <https://github.com/aksonov/react-native-tabs?ref=morioh.com> acedido pela última vez em 13/09/2021.
- Analytics, C. (2021). Anaconda. URL: <https://www.anaconda.com/> acedido pela última vez em 11/09/2021.
- AS, B. (2021). Oidc-react. react component (authprovider) to provide openid connect and oauth2 protocol support. URL: <https://github.com/bjerkio/oidc-react> acedido pela última vez em 23/05/2021.
- Axios-Project (2021). Axios. URL: <https://axios-http.com/> acedido pela última vez em 07/06/2021.
- Bluhm, T. (2021). Envcmd. a simple node program for executing commands using an environment from an env file. URL: <https://www.npmjs.com/package/env-cmd> acedido pela última vez em 07/06/2021.
- Bogard, J. (2021a). A convention-based object-object mapper. URL: <https://automapper.org/> acedido pela última vez em 29/08/2021.
- Bogard, J. (2021b). Simple mediator implementation in .net. URL: <https://github.com/jbogard/MediatR> acedido pela última vez em 27/08/2021.
- Braybrook, R. (2021). Comparing the identity providers (idp's). URL: <https://medium.com/the-new-control-plane/comparing-the-identity-providers-idps-that-i-use-f57aac756c70> acedido pela última vez em 15/05/2021.
- Callstack (2021). React native paper. URL: <https://callstack.github.io/react-native-paper/> acedido pela última vez em 10/06/2021.

- Creative-Labs (2021). Coreui. introducing coreui. URL: <https://coreui.io/docs/getting-started/introduction/> acedido pela última vez em 31/03/2021.
- Facebook (2021a). Jest-junit. a jest reporter that creates compatible junit xml files. URL: <https://jestjs.io/> acedido pela última vez em 02/05/2021.
- Facebook (2021b). React-native. URL: <https://reactnative.dev/> acedido pela última vez em 10/06/2021.
- Facebook (2021c). React.js. about react. URL: <https://reactjs.org/> acedido pela última vez em 27/03/2021.
- Facebook's AI Research lab (FAIR), F. (2021). Pytorch. URL: <https://pytorch.org/> acedido pela última vez em 09/09/2021.
- Formidable-Labs (2021). React native app auth. URL: <https://github.com/FormidableLabs/react-native-app-auth> acedido pela última vez em 13/09/2021.
- GericarePro (2021). Software de gestão para residências de idosos. URL: <https://gericarepro.com/> acedido pela última vez em 09/03/2021.
- GitHub (2021). A popular .net library for building strongly-typed validation rules. URL: <https://fluentvalidation.net/> acedido pela última vez em 29/08/2021.
- Google (2021). Backed by google and loved by app development teams. URL: <https://firebase.google.com/> acedido pela última vez em 09/03/2021.
- IdentityServer (2021). Authentication handler for asp.net. URL: <https://github.com/IdentityServer/IdentityServer4.AccessTokenValidation> acedido pela última vez em 09/03/2021.
- IdentityServer4 (2021). Welcome to identityserver4. URL: <https://identityserver4.readthedocs.io/en/latest/> acedido pela última vez em 05/06/2021.
- Initiative, O. (2021). Openapi description document. URL: <https://www.openapis.org/> acedido pela última vez em 29/08/2021.
- Invertase (2021). React native firebase. URL: <https://rnfirebase.io/> acedido pela última vez em 23/05/2021.
- JSX (2021). Jsx. introducing jsx. URL: <https://reactjs.org/docs/introducing-jsx.html> acedido pela última vez em 07/06/2021.
- Martin, R. C. (2021a). The clean architecture. URL: <https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html> acedido pela última vez em 28/08/2021.
- Martin, R. C. (2021b). Solid relevance. URL: <https://blog.cleancoder.com/uncle-bob/2020/10/18/Solid-Relevance.html> acedido pela última vez em 29/08/2021.

- McKinney, W. (2021). Pandas. URL: <https://pandas.pydata.org/> acessado pela última vez em 09/09/2021.
- Microsoft (2021a). App center. URL: <https://docs.microsoft.com/pt-pt/appcenter/> acessado pela última vez em 12/06/2021.
- Microsoft (2021b). Azure functions. URL: <https://docs.microsoft.com/en-us/azure/azure-functions/functions-overview> acessado pela última vez em 09/09/2021.
- Microsoft (2021c). Create .net apps faster with nuget. URL: <https://www.nuget.org/> acessado pela última vez em 27/08/2021.
- Microsoft (2021d). Entity framework core. URL: <https://docs.microsoft.com/en-us/ef/core/> acessado pela última vez em 23/03/2021.
- Microsoft (2021e). Entity framework core. URL: <https://docs.microsoft.com/en-us/ef/core/> acessado pela última vez em 27/08/2021.
- Microsoft (2021f). Introduction to asp.net identity. URL: <https://docs.microsoft.com/en-us/aspnet/identity/overview/getting-started/introduction-to-aspnet-identity> acessado pela última vez em 23/05/2021.
- Microsoft (2021g). What is azure active directory b2c? URL: <https://docs.microsoft.com/en-US/azure/active-directory-b2c/overview> acessado pela última vez em 05/03/2021.
- Microsoft (2021h). What is .net? URL: <https://dotnet.microsoft.com/learn/dotnet/what-is-dotnet> acessado pela última vez em 27/08/2021.
- Motte, S. (2021). Dotenv. sync your .env files between machines, environments, and team members. URL: <https://www.npmjs.com/package/dotenv> acessado pela última vez em 07/06/2021.
- MySenior (2021). Mysenior gestão de lares. URL: <https://mysenior.com/> acessado pela última vez em 09/03/2021.
- OAuth (2021). OAuth 2.0 for browser-based apps. URL: <https://oauth.net/2/browser-based-apps/> acessado pela última vez em 23/03/2021.
- OAuth-2.0 (2021). OAuth 2 simplified. URL: <https://aaronparecki.com/oauth-2-simplified/> acessado pela última vez em 05/06/2021.
- O'Dea, S. (2021). Mobile operating systems' market share worldwide. URL: <https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/> acessado pela última vez em 16/09/2021.
- Oliphant, T. (2021). Numpy. URL: <https://numpy.org/> acessado pela última vez em 09/09/2021.

- OpenID-Connect (2021). Welcome to openid connect. URL: <https://openid.net/connect/> acedido pela última vez em 05/06/2021.
- Osadnik, S. S. M. (2021). React navigation. URL: <https://reactnavigation.org/docs/drawer-based-navigation/> acedido pela última vez em 11/06/2021.
- Patil, P. (2021). Disease symptom prediction. URL: <https://www.kaggle.com/itachi9604/disease-symptom-description-dataset/code> acedido pela última vez em 09/09/2021.
- Prasanjith, D. (2021). 6 reasons to use react hooks instead of classes. URL: <https://blog.bitsrc.io/6-reasons-to-use-react-hooks-instead-of-classes-7e3ee745fe04> acedido pela última vez em 18/06/2021.
- Project, J. (2021). Jupyter notebook. URL: <https://pytorch.org/> acedido pela última vez em 09/09/2021.
- Prouse, C. P. . R. (2021). Unit-testing framework for all .net languages. URL: <https://nunit.org/> acedido pela última vez em 09/03/2021.
- Proyecto26 (2021). React native inappbrowser reborn. URL: <https://github.com/proyecto26/react-native-inappbrowser> acedido pela última vez em 13/09/2021.
- Pusher (2021a). Bring next level realtime features to your apps. simple, scalable, secure. URL: <https://pusher.com/channels> acedido pela última vez em 09/03/2021.
- Pusher (2021b). Send push notifications with scalable delivery, security and insights. URL: <https://pusher.com/beams> acedido pela última vez em 09/03/2021.
- Pusher-Limited (2021). Pusher-js. pusher channels javascript client. URL: <https://www.npmjs.com/package/pusher-js> acedido pela última vez em 17/05/2021.
- Python Software Foundation, P. (2021). Requests. URL: <https://docs.python-requests.org/en/master/> acedido pela última vez em 09/09/2021.
- React-Training (2021). React router guides. URL: <https://reactrouter.com/web/guides/quick-start> acedido pela última vez em 15/05/2021.
- RevtelTech (2021). React native nfc manager. URL: <https://github.com/whitedogg13/react-native-nfc-manager> acedido pela última vez em 09/06/2021.
- Serilog (2021). Flexible, structured events — log file convenience. URL: <https://serilog.net/> acedido pela última vez em 23/03/2021.
- Sessler, J. (2021). React native swipe list view. URL: <https://github.com/jemise111/react-native-swipe-list-view> acedido pela última vez em 13/09/2021.
- Sete, M. (2021). Loosely coupled architecture. URL: <https://medium.com/>

- @marciosete/loosely-coupled-architecture-6a2b06082316 acessado pela última vez em 15/09/2021.
- SNS (2021). Estruturas residenciais para idosos. URL: <https://www.sns.gov.pt/noticias/2020/08/12/estruturas-residenciais-para-idosos/> acessado pela última vez em 20/08/2021.
- Suter, R. (2021). Nswag is a swagger/openapi 2.0 and 3.0 toolchain for .net. URL: <https://github.com/RicoSuter/NSwag> acessado pela última vez em 29/08/2021.
- Sutter, B. (2021). React native pusher push notifications. URL: <https://github.com/b8ne/react-native-pusher-push-notifications> acessado pela última vez em 13/09/2021.
- Taylor, J. (2021a). Clean architecture solution template. URL: <https://github.com/jasontaylordev/CleanArchitecture> acessado pela última vez em 09/06/2021.
- Taylor, J. (2021b). Solution architect. URL: <https://jasontaylor.dev/> acessado pela última vez em 09/06/2021.
- Trujillo, J. (2021). React native push notifications. URL: <https://github.com/zo0r/react-native-push-notification> acessado pela última vez em 9/09/2021.
- Zehra, F. (2021). Disease symptom prediction. URL: <https://medium.com/@farzeenzehra.fz/symptoms-to-disease-prediction-model-521c4715c15e> acessado pela última vez em 09/09/2021.

