

Interfaces Web - Forum IPT.

Relatório de Trabalho Prático Interfaces Web.

**António Gonçalves n.º 23787,
Paulo Sentinella n.º 24880**

Resumo

O nosso trabalho prático para a unidade curricular "Interfaces Web" envolveu a implementação de uma aplicação utilizando uma stack um tanto quanto personalizada composta por React.js para o frontend, Azure Functions para a criação de endpoints e PostgreSQL hospedado na AWS como banco de dados.

A arquitetura da aplicação foi estruturada de forma a aproveitar as vantagens de cada tecnologia escolhida bem como a disponibilidade delas serem gratuitas. O React.js foi utilizado para desenvolver a interface do utilizador, proporcionando uma experiência interativa e responsiva aos utilizadores finais.

Para gerenciar a lógica do backend, foram utilizadas Azure Functions, que são funções serverless (cloud). Estas funções foram responsáveis por criar os endpoints necessários para manipulação de dados e interação com a base de dados.

A base de dados escolhida foi o PostgreSQL, hospedado na AWS (Amazon Web Services).

A combinação destas tecnologias proporcionou uma solução mais robusta para o desenvolvimento da nossa aplicação web. O React.js através dos seus componentes oferece uma interface coesa e amigável, às Azure Functions trazem o conforto de garantir a execução eficiente das operações no backend, e o PostgreSQL na AWS oferece uma base de dados confiável e segura e relacional. A integração destas tecnologias possibilitou desenvolver uma aplicação completa, abrangendo desde a interface do utilizador até à gestão eficiente dos dados nos bastidores.

Índice

Resumo.....	5
Introdução.....	7
FrontEnd.....	8
React js.....	8
Estrutura do projeto.....	9
App.jsx e router-dom.....	10
Modelo de dados e entidades.....	15
Endpoints.....	17
Azure functions.....	17
Repositório (github).....	23
Principais ecrãs do site.....	25
Tela Login.....	25
Tela de Registo.....	26
Tela de Início após Login.....	27
Como Iniciar a aplicação (https://cyb3rwolf945.github.io/IWWEB/).....	28
1º Forma.....	28
2º Forma.....	28
Desafios.....	30
Conclusão.....	31
Bibliografia.....	33

Índice de figuras

Figura 1 - React.js logotipo.....	9
Figura 2 - Estrutura de pastas do projeto,.....	10
Figura 3 - Excerto de código retirado do ficheiro App.jsx.....	11
Figura 4 - Dependências do react-router-dom.....	12
Figura 5 - Estrutura da pasta Pages.....	12
Figura 6 - Estrutura da pasta Dashboard com as sub-páginas.....	13
Figura 7 - Estrutura da pasta Components.....	14
Figura 8 - Estrutura da pasta services.....	14
Figura 9 - Diagrama Entidade Relacionamento das nossas entidades.....	16
Figura 10 - Esquema de Azure Functions.....	18
Figura 11 - Eventos de funções serverless Azure.....	19
Figura 12 - Extensões do Azure para o visual studio.....	19
Figura 14 - Escolha do Evento.....	21
Figura 15 - Escolha do Nome.....	21
Figura 16 - Excerto de código.....	22
Figura 17 - Endpoints realizados para a implementação deste projeto.....	23
Figura 18 - Exemplo da sintaxe.....	25
Figura 19 - Página de login.....	26
Figura 20 - Tela de Registo.....	27
Figura 21 - Homepage.....	28
Figura 22 - Clique para redirecionar para a página de detalhes do canal.....	29
Figura 23 - Página de detalhes de engenharia multimédia.....	29
Figura 24 - Aceder à página de perfil do utilizador.....	30
Figura 25 - Página de perfil do utilizador.....	31
Figura 26 - Aceder à página de dashboard.....	31
Figura 27 - Página de dashboard.....	32
Figura 28 - Gestão de utilizadores.....	33
Figura 29 - Gestão de publicações.....	34
Figura 30 - Gestão de Canais.....	34
Figura 31 - Web server local.....	36
Figura 32 - HomePage no github pages.....	37

Introdução

O projeto final da unidade curricular "Interfaces Web" exigiu a criação de uma aplicação react utilizando a plataforma proposta pelo professor “sheety”, para a criação das funcionalidades CRUD, bem como a nossa base de dados.

Apesar destas recomendações, o nosso grupo preferiu adotar um caminho diferente no desenvolvimento desta aplicação, utilizando ferramentas em associação ao react como MUI (Material UI) e Tailwind.

No backend, optou-se por utilizar Azure Functions, aproveitando suas capacidades serverless. Sendo que estas funções ficaram responsáveis por criar os endpoints necessários para a manipulação de dados e interação com a base de dados.

A escolha do PostgreSQL como base de dados, hospedada também em cloud na AWS, trouxe confiabilidade e segurança para o armazenamento e gerenciamento eficiente dos dados.

Em resumo, a combinação estratégica de React.js, Azure Functions e PostgreSQL na AWS (Amazon Web Services) permitiu-nos o desenvolvimento de uma aplicação web mais completa, destacando-se pela sua versatilidade em termos de funções e componentes funcionais no Frontend.

FrontEnd

React.js

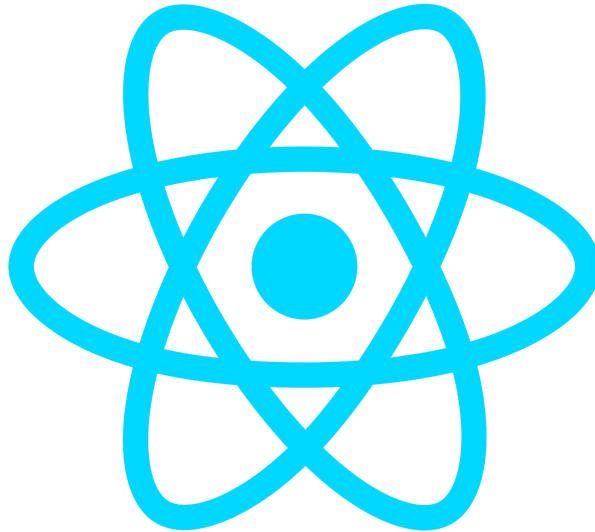


Figura 1 - React.js logotipo

O React.js é uma biblioteca JavaScript amplamente utilizada para desenvolvimento de interfaces web. A sua popularidade deve-se, em parte, à sua abordagem por componentes, onde é possível realizar a construção de uma página a partir da conjunção de vários componentes reutilizáveis, tornando a abordagem DRY (Don't Repeat Yourself) mais eficiente. Com o React, é possível construir interfaces interativas e responsivas de forma mais facilitada. Além disso, o conceito de virtual DOM contribui para um desempenho otimizado, atualizando apenas as partes necessárias da página.

Estrutura do projeto

No que toca a frontend, tomamos a iniciativa de realizar a divisão do nosso projeto em 3 partes: Pages, components, services, desta forma, permitindo a melhor organização da nossa aplicação.

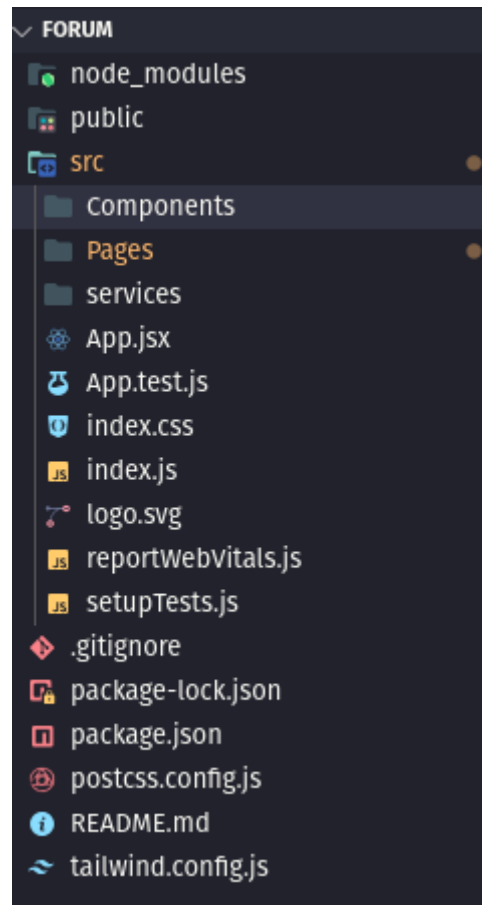


Figura 2 - Estrutura de pastas do projeto,

Assim, ao analisarmos a imagem apresentada, é possível notar a presença das pastas mencionadas anteriormente - components, pages e services - localizadas dentro da nossa pasta src no projeto. Acompanhando essas pastas, encontramos um arquivo vital designado por App.jsx. A partir deste arquivo, procedemos à renderização dos nossos componentes ou páginas, conforme a rota correspondente.

App.jsx e router-dom

O router-dom do React, presente no ficheiro App.jsx, desempenha um papel crucial na gestão da navegação na nossa aplicação com arquitetura de Single Page Application (SPA). Com o auxílio deste módulo conseguimos obter um conjunto de ferramentas para criar rotas, permitindo que a aplicação responda dinamicamente às interações do utilizador sem a necessidade de recarregar a página.

As rotas são configuradas usando os componentes Routes, Route e Navigate do router-dom.

```
<Routes>
  { /* Public Routes */}
  <Route path="/" element={user ? <HomePage /> : <Login />} />
  <Route path="/Login" element={user === null ? <Navigate to="/" /> : <Login />} />
  <Route path="/Registo" element={user === null ? <Navigate to="/" /> : <Registo />} />

  { /* Protected Routes */}
  {user && (
    <>
      <Route path="/profile/:id" element={<UserProfilePage selected="info"/>} />
      <Route path="/canal/:id" element={<ChannelPage />} />
      <Route path="/Chat" element={<Chat />} />
      <Route path="/pdf" element={<PDF />} />
      { /* Dashboard com nested Routes */}
      {
        user.admin_privileges === true ? (
          <Route path="/DashBoard" element={<DashBoard />} />
          { /* Redirect para o geral sempre que o utilizador meter uma rota dentro do sub domínio dashboard invalida */}
          <Route index element={<Navigate to="/Dashboard/Geral" />} />
          <Route path="Geral" element={<Geral />} />
          <Route path="users" element={<Users />} />
          <Route path="Pubs" element={<Pubs />} />
          <Route path="Canais" element={<Canais />} />
        ) : <Route path="/" element={<Navigate to="/" />} />
      }
    )
  )}
</>
```

Figura 3 - Excerto de código retirado do ficheiro App.jsx

Por questões de segurança e de acessibilidade, adotamos uma filosofia de rotas públicas e privadas, permitindo serem acedidas as rotas privadas caso no localStorage seja definido o nosso “user” que é atribuído o valor após o mesmo fazer o seu login. Caso este não tenha valor entramos nas rotas privadas que apenas são válidas para utilizadores que não tenham feito login.

No nosso projeto ainda contamos com um Backoffice onde é possível realizar a gestão de utilizadores, canais e publicações, sendo só possível aceder ao mesmo caso o utilizador tenha permissões de utilizador, mais uma vez sendo este campo guardado no localStorage uma vez feito Login na aplicação.

```
    },
    "devDependencies": {
      "dotenv": "^16.3.1",
      "react-router-dom": "^6.20.0",
      "tailwindcss": "^3.3.5"
    }
  }
}
```

Figura 4 - Dependências do react-router-dom

É importante notar que mais uma vez caso não tivéssemos a biblioteca "react-router-dom" não seria possível esta estrutura e organização eficiente de rotas.

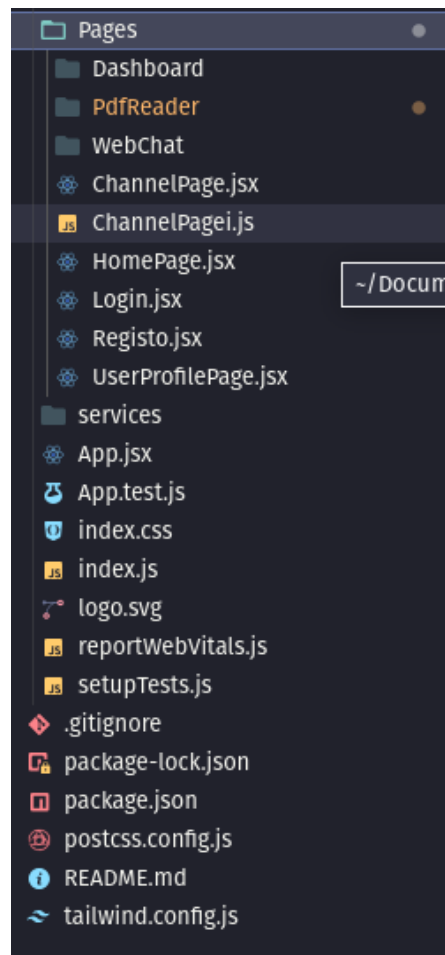


Figura 5 - Estrutura da pasta Pages

A partir da figura 5 é possível notar que a pasta Pages irá ter como conteúdo as páginas da nossa aplicação. Sendo que por motivos de organização, quando temos páginas que tem por sua vez sub páginas como é o caso do Dashboard fez se uso da criação de uma pasta para os inserir.

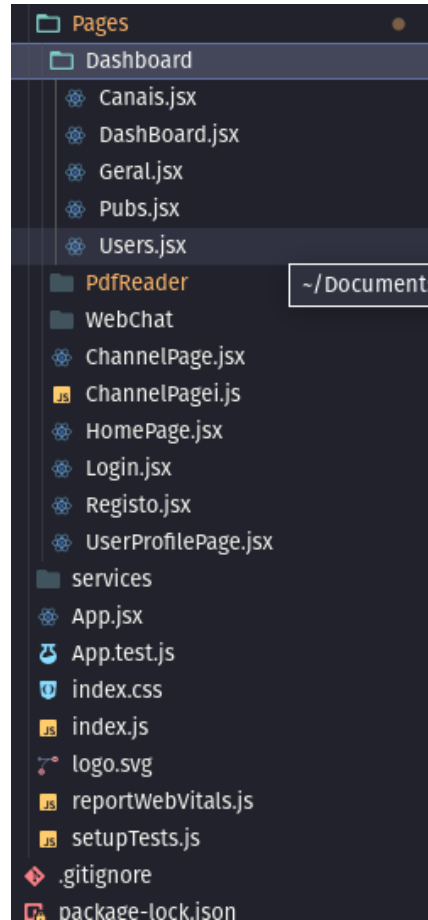


Figura 6 - Estrutura da pasta Dashboard com as sub-páginas.

Na parte dos componentes seguimos a mesma filosofia, pois nesta página estão todos os componentes que utilizamos na nossa aplicação para formulação das nossas páginas.

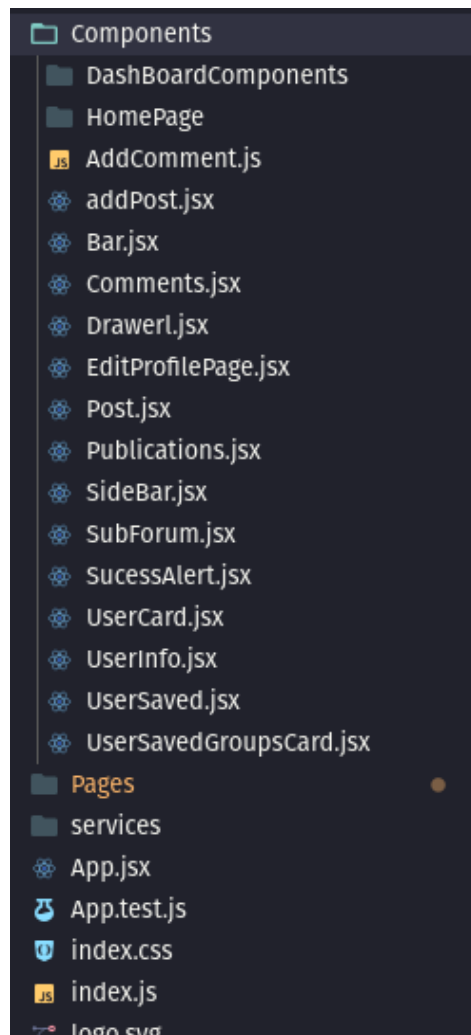


Figura 7 - Estrutura da pasta Components

Após termos observado a estrutura das pastas pages e componentes resta-nos a estrutura da pasta services, onde na mesma são nos apresentados 4 ficheiros responsáveis pelo fetch as nossas funções serverless da azure para cada entidade.

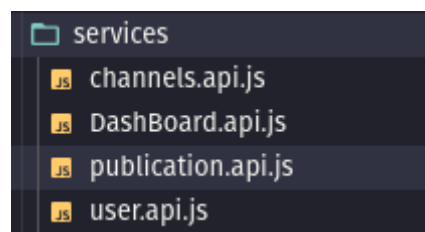


Figura 8 - Estrutura da pasta services

Modelo de dados e entidades

O projeto é composto por diversas entidades, entre elas estão os utilizadores, comentários, publicações e canais.

Estas entidades estão todas refletidas na nossa base dados SQL (postgree) alojada na AWS(Amazon Web Services).

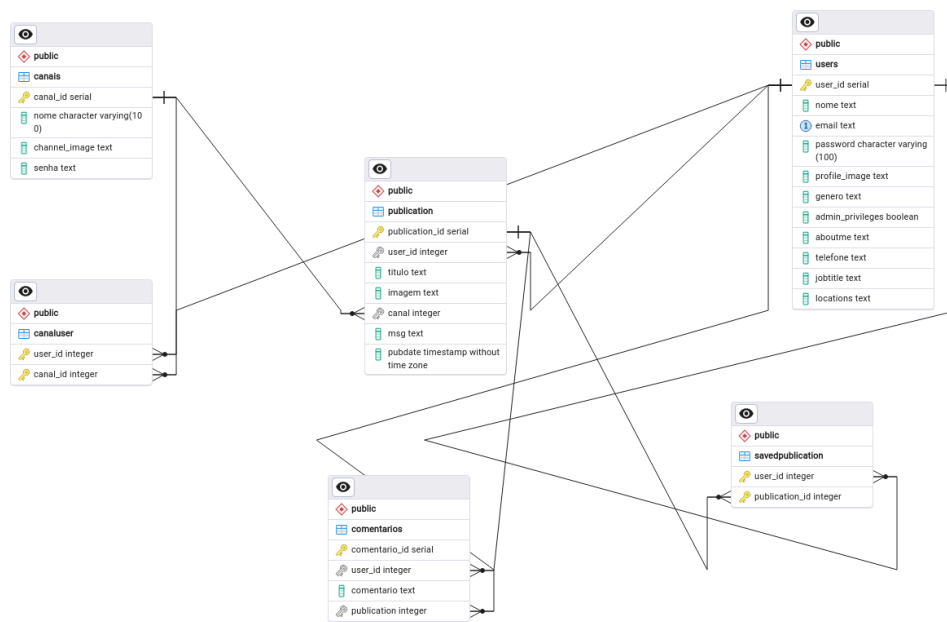


Figura 9 - Diagrama Entidade Relacionamento das nossas entidades.

Portanto temos ao todo 4 entidades, sendo o tema deste projeto um fórum temos com base a primeira delas que diz respeito aos utilizadores, sendo peças centrais que participam ativamente na dinâmica da plataforma. Os mesmos têm a capacidade de criar e partilhar conteúdo (publicações) na forma de posts, estabelecendo uma conexão direta com os diferentes canais disponíveis.

Os canais, por sua vez, representam os espaços virtuais destinados a agrupar e organizar os posts, oferecendo uma estrutura de paginação e filtragem. Cada canal está associado a um tema específico, proporcionando uma experiência de navegação mais intuitiva e direcionada aos interesses dos utilizadores.

No que diz respeito às publicações, estas assumem um papel fundamental pois em forma de posts criados pelos utilizadores e categorizados nos canais correspondentes permitem a comunicação veículos de informação, opinião e interação, alimentando o conteúdo dentro do fórum. É importante sublinhar que as publicações não são estáticas; podem evoluir ao longo do tempo mediante a adição de comentários.

Os comentários, última peça deste ecossistema, representam as reações, discussões e interações que se desenrolam em torno das publicações. Podem ser adicionados pelos utilizadores após a criação de um post, enriquecendo assim a troca de ideias e perspetivas. Os comentários desempenham um papel vital trazendo um espaço dinâmico para a expressão de opiniões e partilhas.

Assim, estas quatro entidades - utilizadores, canais, publicações e comentários - entrelaçam-se harmoniosamente para criar um ambiente rico e com diversidade, cumprindo o propósito central da nossa aplicação.

Endpoints

Azure functions

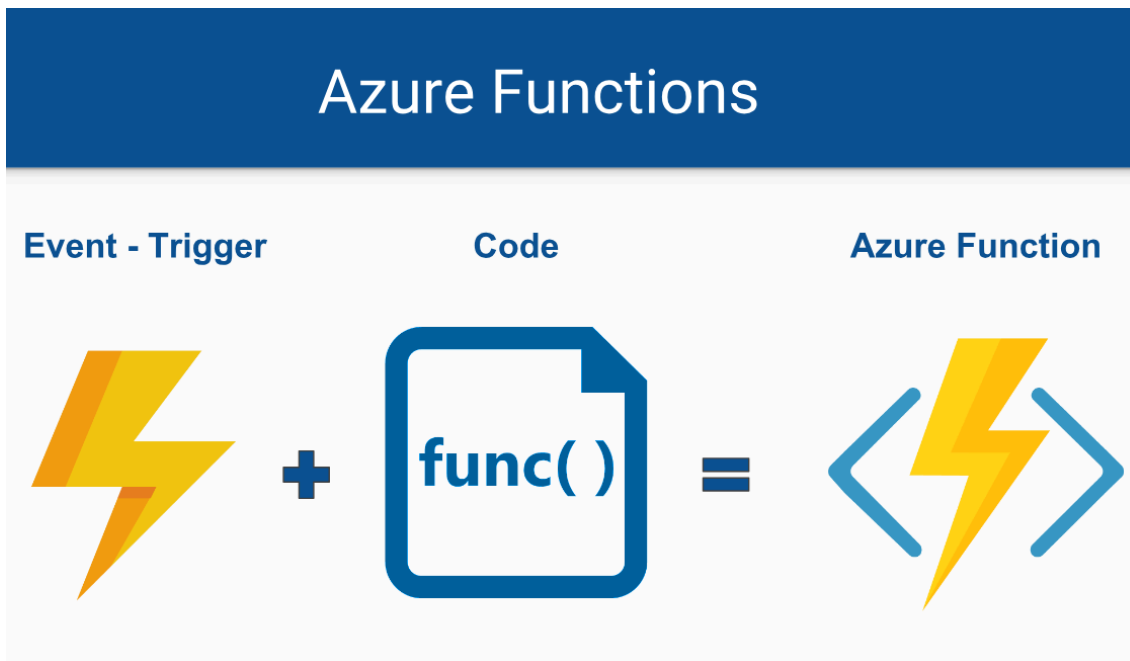


Figura 10 - Esquema de Azure Functions

Azure Functions, é um serviço serverless, a sua essência consiste em código executado na nuvem, acionado por eventos específicos.

A partir das mesmas é possível implementar pequenos pedaços de código e definir quando se deseja executá-los. Relativamente ao problema de “se não temos recursos suficientes para correr o nosso código”, não nos precisamos de preocupar pois a Azure garante que o código tenha sempre os recursos de necessários para funcionar sem problemas, mesmo quando a demanda é relativamente alta.

Algo bastante bom das funções serverless da Azure é o facto de podermos escrever as nossas funções em várias linguagens, sendo que, não será necessário cingir-nos a uma específica. Pelo facto de estarmos a falar de funções na nuvem, o valor dos planos de consumo também é diferente e podemos pagar apenas pelo tempo de uso, se as nossas funções nunca forem ativas nunca seremos cobrados. Sendo que para este projeto foi decidido usar as funções azure exatamente pelo facto de até um milhão de pedidos por mês, estas são gratuitas.

Cada função funciona através de eventos. Por exemplo, caso tenhamos escolhido uma função `httpTrigger` (como foi feito para o uso neste projeto), significa que a função é executada quando recebe um HTTP request.

Aqui estão alguns dos eventos que a azure disponibiliza:

Trigger	Code executes on
HTTPTrigger	New HTTP request
TimerTrigger	Timer schedule
BlobTrigger	New blob added to an azure storage container
QueueTrigger	New message arrived on an azure storage queue
ServiceBusTrigger	New message arrived on a Service Bus queue or topic
EventHubTrigger	New message delivered to an event hub

Figura 11 - Eventos de funções serverless Azure.

Para a criação de uma função basta, conectarmos a nossa conta do azure ao visual studio(instalando as extensões azure account e azure functions):

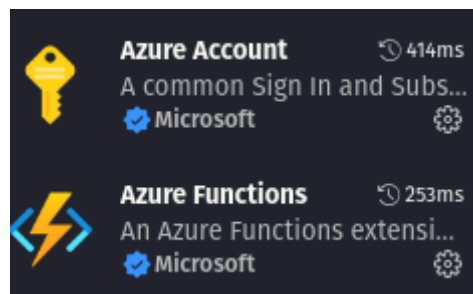


Figura 12 - Extensões do Azure para o visual studio.



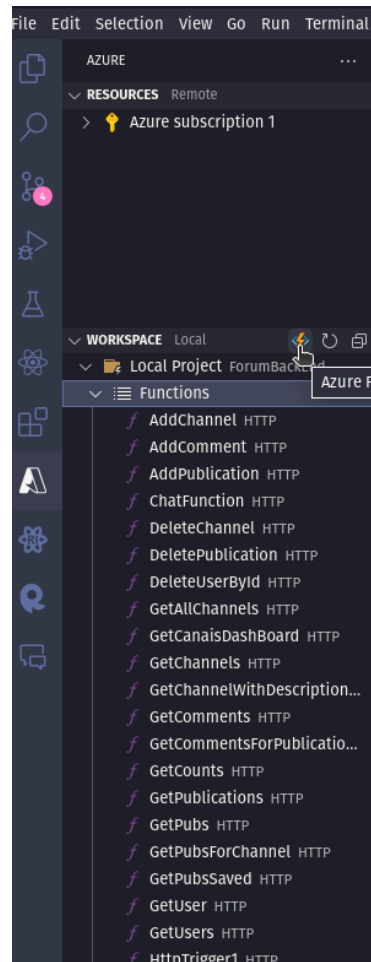


Figura 13 - Iniciar criação de uma função.



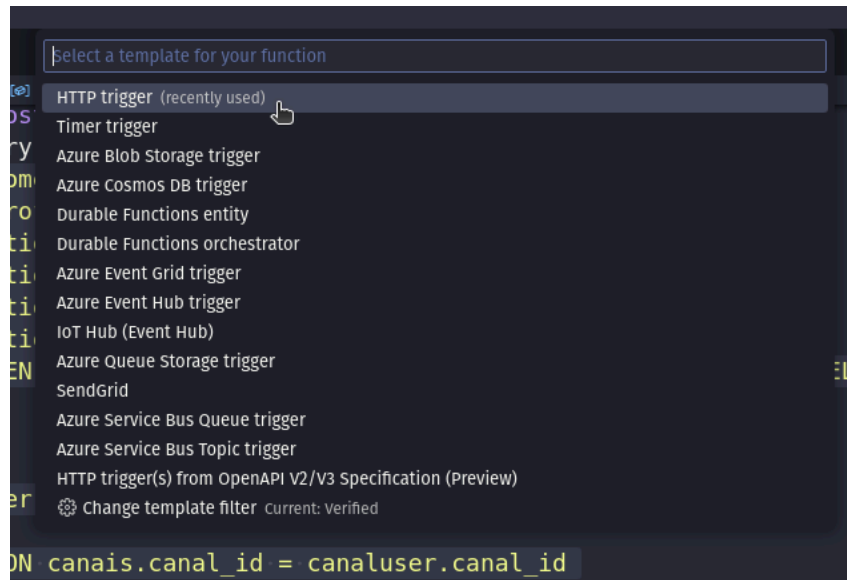


Figura 14 - Escolha do Evento.

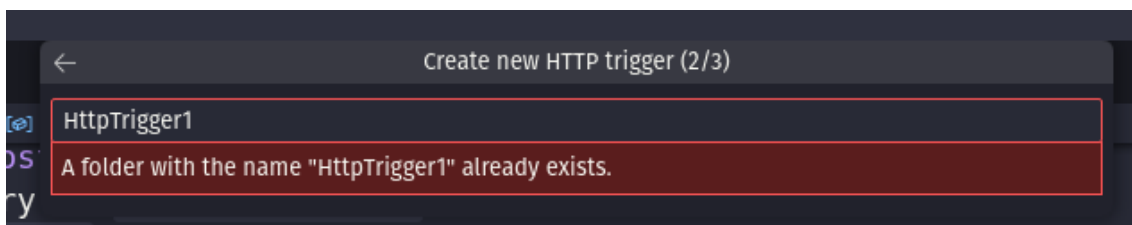


Figura 15 - Escolha do Nome.

Após os passos seguintes teremos apenas de abrir a nossa função e começar a programar.

```
const httpTrigger: AzureFunction = async function (context: Context, req: HttpRequest): Promise<void> {
    try {
        const contentType = req.headers['content-type'];

        if (contentType && contentType.toLowerCase() === 'application/json') {
            const requestBody = req.body as userUpdate;

            if (
                !requestBody
            ) {
                context.res = {
                    status: 400,
                    body: "Bad Request. user_id and admin_privileges are required in the JSON body.",
                };
                return;
            }
        }
    }
}
```

Figura 16 - Excerto de código.

Com esta ferramenta fizemos funções desde simplesmente ir buscar utilizadores como é o caso do GetUsers até ao ponto de funções mais complicadas como no caso do Login.

Abaixo está representada uma figura com todas as funções feitas para a realização deste projeto.

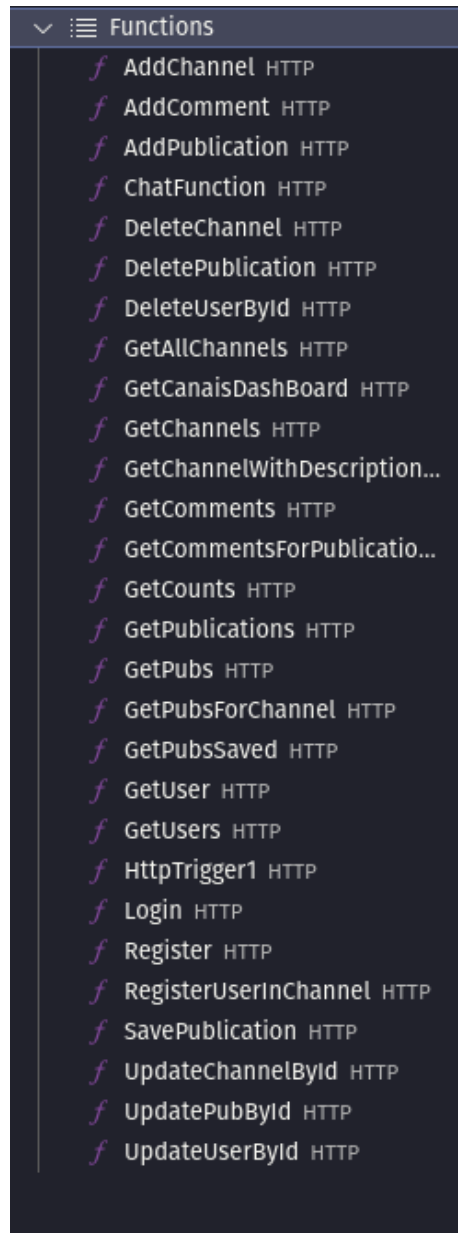


Figura 17 - Endpoints realizados para a implementação deste projeto.

Repositório (github)

A escolha do Git como sistema de controlo de versões para este projeto foi fundamental, trazendo diversas vantagens. Como é conhecido o Git é amplamente reconhecido pela sua eficiência na gestão de alterações de código, documentos e outros tipos de ficheiros.

O nosso grupo pode dizer que sentiu mais segurança por ter trabalhado com esta ferramenta, pois se tivéssemos feito algo que não seria o pretendido poderíamos voltar sempre um commit atrás, o que nos revelou uma rede de segurança. Algo que também tivemos oportunidade de fazer foi a divisão de Branchs de forma a no fundo ficarmos com 3 branches (Paulo, antonio, main).

Os Branchs com os nomes dos elementos são responsáveis pelas alterações efetuadas por cada um, já o main é onde poderemos ver a versão mais atualizada e funcional, desta forma só vai para o main quando temos certeza que já temos algo funcional, garantindo uma aplicação mais eficiente e funcional.

No que toca à utilização propriamente dita para nos auxiliar, decidimos enquanto grupo utilizar o GitHub Desktop, em associação com o Git, proporcionando uma interface gráfica intuitiva para quem o utiliza. O GitHub Desktop simplifica operações comuns do Git, como commits, criação de ramos e sincronização com repositórios remotos.

Em adição para “merge conflicts” foi usado o github desktop para nos informar que havia um, e posteriormente tratados no visual studio.

Para finalizar devo dizer que a interface do github desktop deixa os merge mais fácil pela sua sintaxe pois quando pretendemos realizar um merge de outro branch para o nosso a sua sintaxe é intuitiva dizendo-nos exatamente isso, já noutras plataformas como o caso do SoucetTree isso não acontece portanto achamos que isso possa ser pontos a favor do Github Desktop.

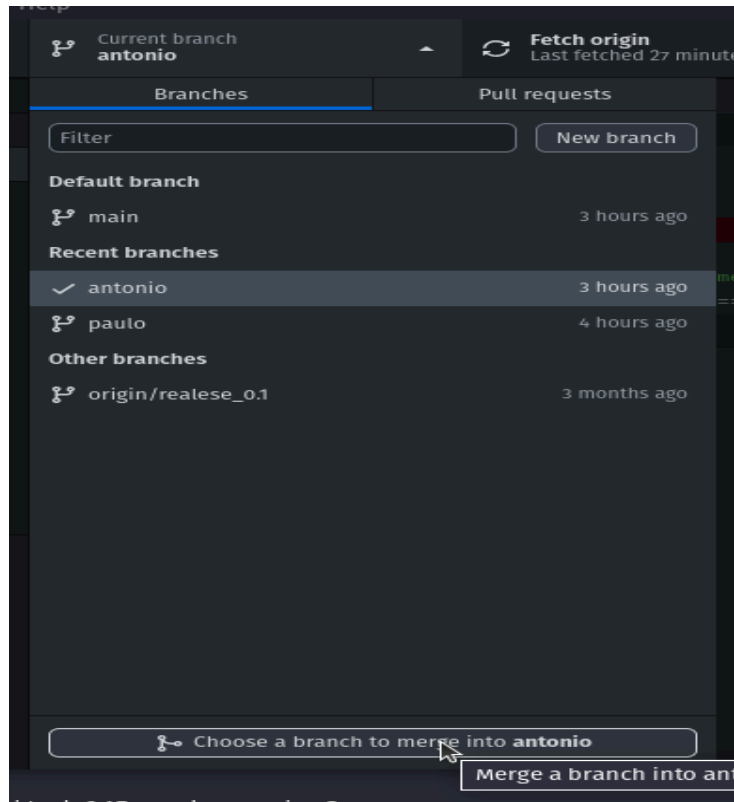


Figura 18 - Exemplo da sintaxe.

Os links para os nossos repositórios são :

Repositório Backend →

<https://github.com/Cyb3rWolf945/IwBackend---AWS---AZURE>

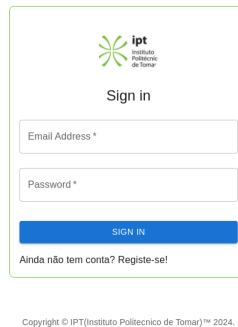
Repositório FrontEnd →

<https://github.com/PDSentinella/ForumIPT>

Em resumo, o Git é uma escolha robusta para o controlo de versões, destacando-se pela sua eficácia e flexibilidade. O GitHub Desktop complementa esta escolha ao fornecer uma interface gráfica que torna as funcionalidades do Git mais acessíveis, promovendo uma gestão de versões mais eficiente e simplificada no contexto deste projeto.

Principais ecrãs do site

Tela Login



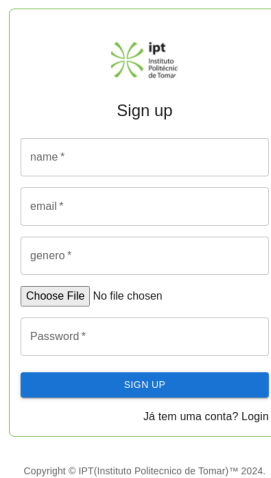
The screenshot shows a login form for the Instituto Politécnico de Tomar. At the top is the IPT logo, which consists of a green stylized flower-like icon and the text 'ipt Instituto Politécnico de Tomar'. Below the logo is the text 'Sign in'. There are two input fields: 'Email Address *' and 'Password *'. Below these fields is a blue button with the text 'SIGN IN'. At the bottom of the form, there is a link that says 'Ainda não tem conta? Registe-se!'. Below the entire form, there is a small copyright notice: 'Copyright © IPT(Instituto Politécnico de Tomar)™ 2024.'

Figura 19 - Página de login.

Como início da nossa aplicação, nós escolhemos que fosse sempre direta para o login caso o utilizador não tivesse a sua sessão iniciada.

Como podemos observar na figura 19, é constituída por um simples formulário onde são requisitados os campos email e password em ordem a fazer o login. Caso o utilizador não esteja registado na aplicação, basta clicar na frase abaixo e será redirecionado para a próxima tela de registo.

Tela de Registo



The screenshot shows a web form for signing up at IPT (Instituto Politécnico de Tomar). The form is titled "Sign up" and includes the following fields and elements:

- Logo:** IPT Instituto Politécnico de Tomar
- name ***: Text input field
- email ***: Text input field
- genero ***: Text input field
- Image Upload:** A button labeled "Choose File" and a text label "No file chosen".
- Password ***: Text input field
- SIGN UP:** A blue button to submit the form.
- Link:** "Já tem uma conta? Login" (Already have an account? Login)
- Footer:** "Copyright © IPT(Instituto Politécnico de Tomar)™ 2024."

Figura 20 - Tela de Registo.

Como podemos observar na figura 20 a tela de registo será efectivamente um formulário semelhante ao login mas com mais campos, sendo estes género e imagem de perfil. Os restantes campos podem ser adicionados posteriormente na área de perfil do utilizador.

Tela de Início após Login

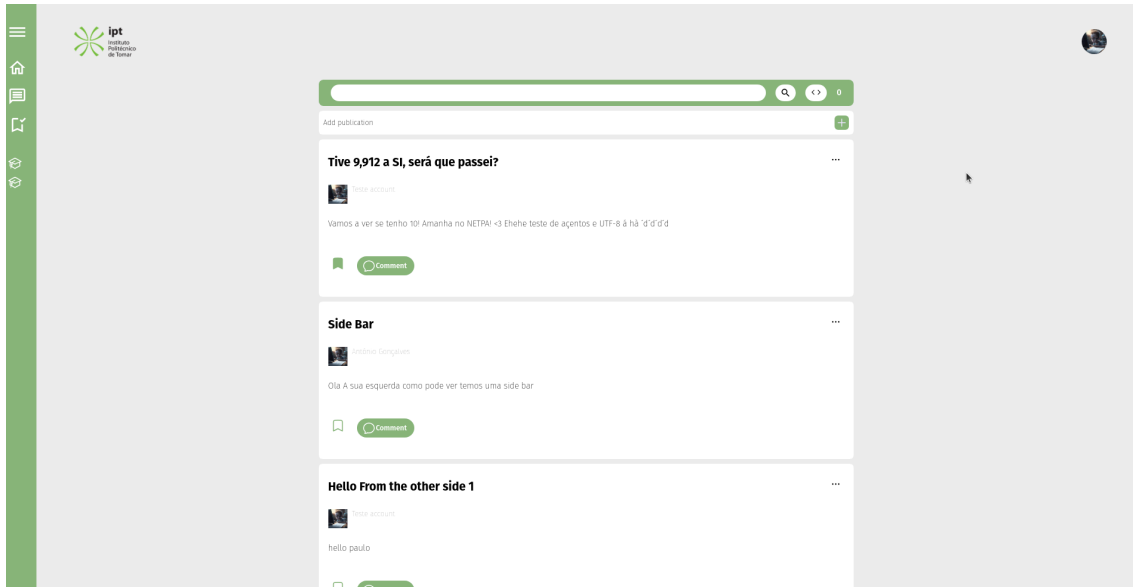


Figura 21 - Homepage.

Esta página é exatamente a primeira página com que o utilizador é abordado ao realizar com sucesso o seu login.

Esta página faz uso de 1 endpoint este sendo o “GetPublications”, onde vamos buscar as publicações com uma paginação que começa na página 0 e vai aumentando sucessivamente.

Já na mesma página temos uma sidebar do lado esquerdo onde temos uma lista com os canais que o utilizador se encontra registado.

Para isso, fazemos uso de um endpoint designado por “GetCHannels” que vai buscar os canais para um dado utilizador.

Tela de detalhes de uma página.

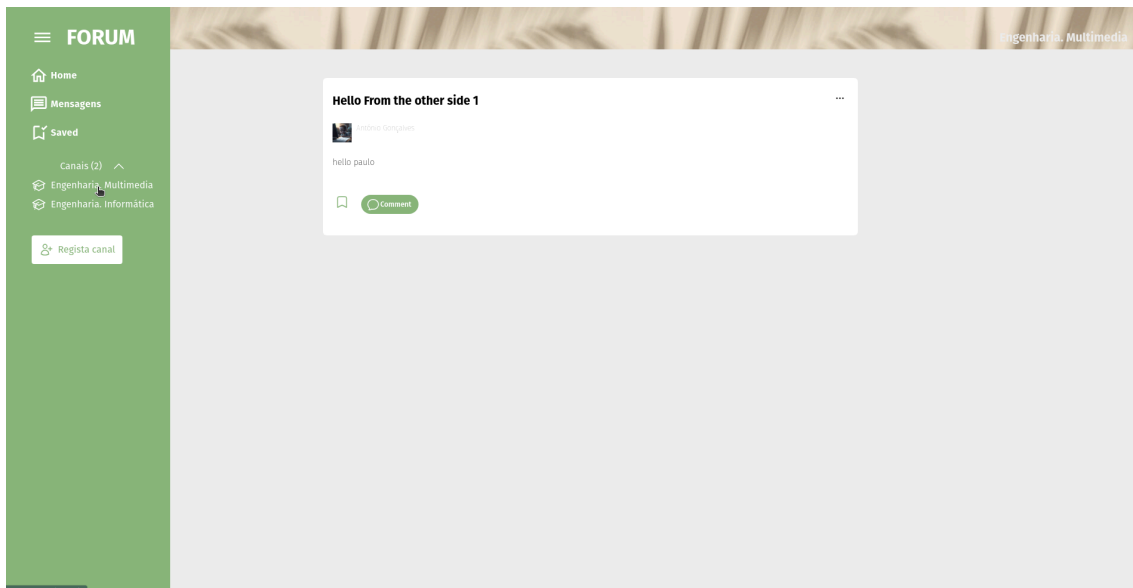


Figura 22 - Clique para redirecionar para a página de detalhes do canal.

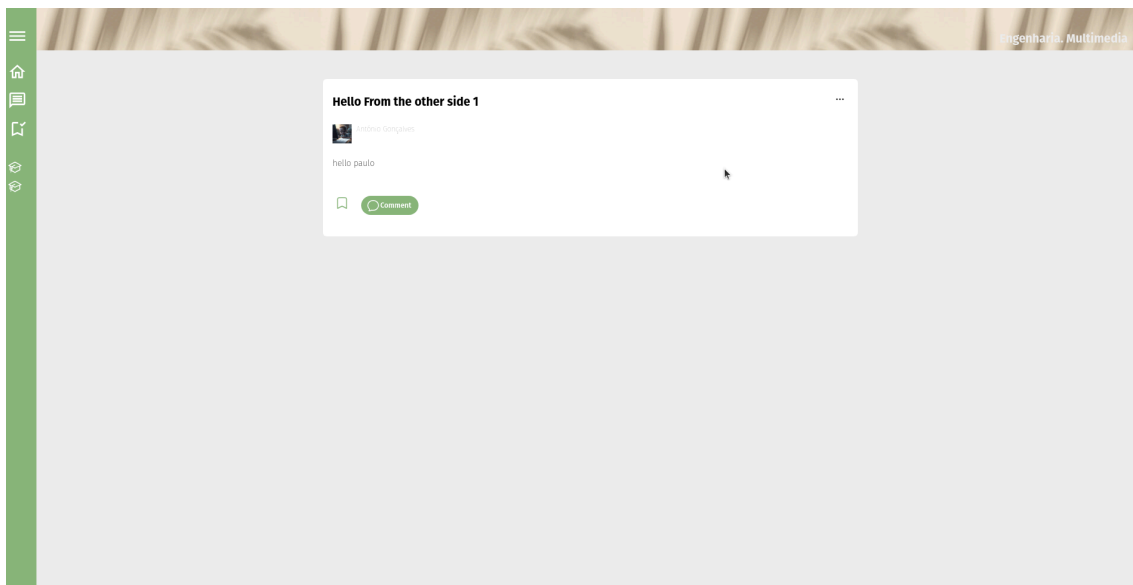


Figura 23 - Página de detalhes de engenharia multimédia.

Tela de perfil

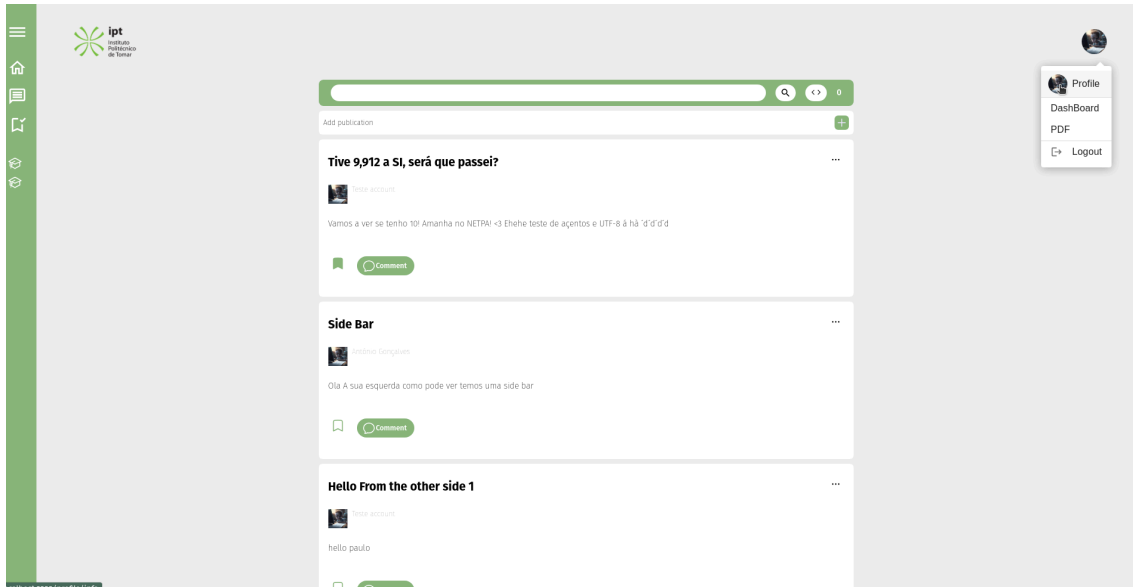


Figura 24 - Aceder à página de perfil do utilizador.

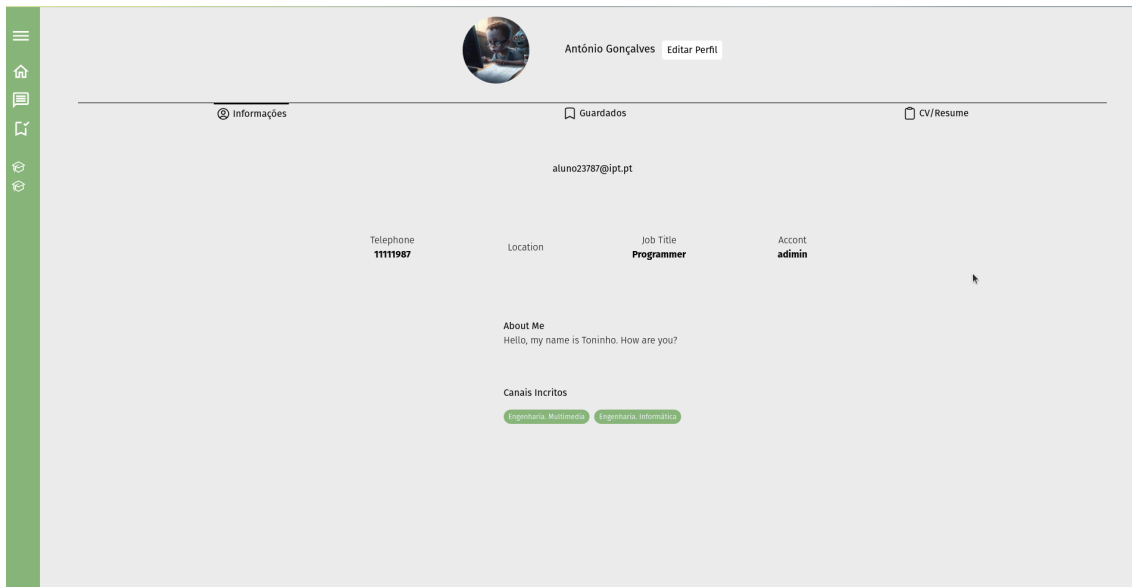


Figura 25 - Página de perfil do utilizador.

Como podemos ver na figura 25, estamos perante a página de detalhes do utilizador, aqui fazemos uso dos endpoints: “GetUser” responsável por obter os atributos do utilizador e “GetChannels” responsável por obter os canais fixados na parte de baixo.

Tela de dashboard

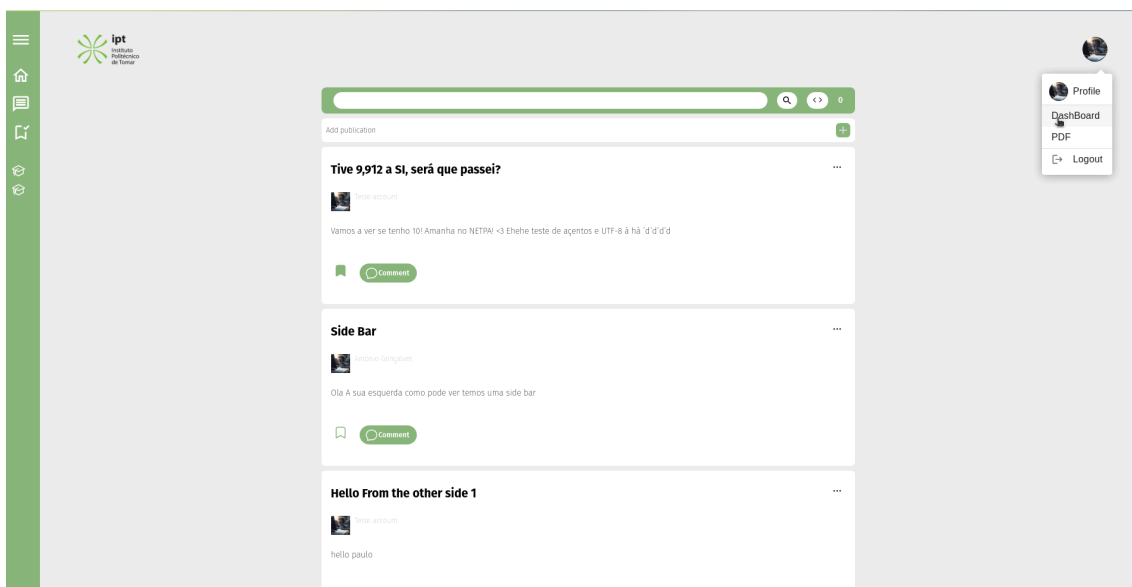


Figura 26 - Aceder à página de dashboard.

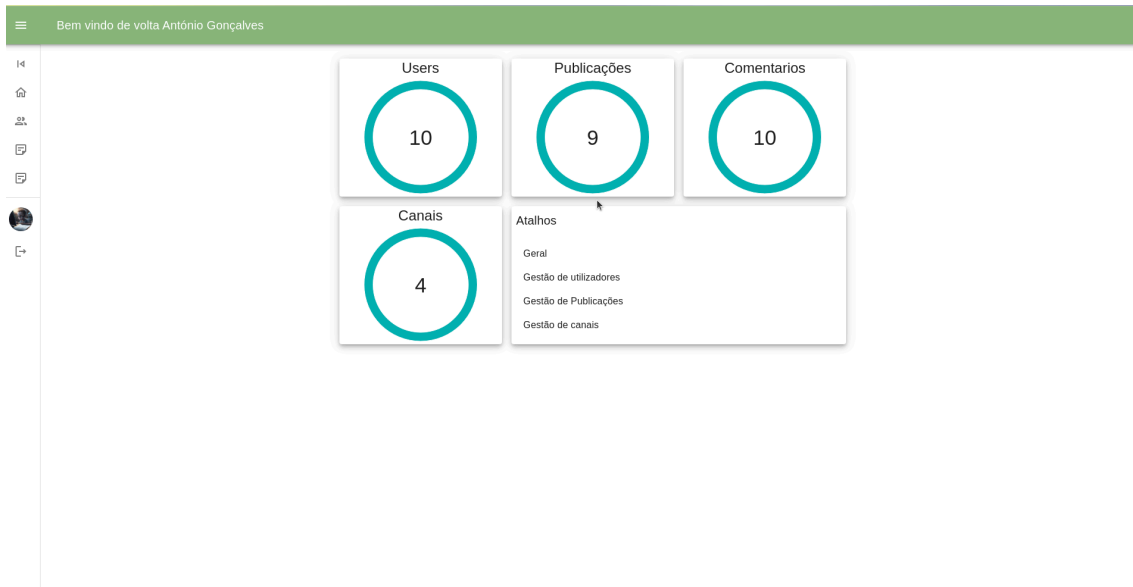


Figura 27 - Página de dashboard.

Esta é a página com que somos recebidos, ao entrar no nosso dashboard (Backoffice), podemos ver a contagem das diversas entidades, sendo refletida uma vista geral sobre o número de utilizadores, publicações feitas pelos mesmos, comentários e quantos canais já existem. Ao lado ainda temos os atalhos para as restantes páginas do Dashboard.

Esta página é responsável pelo principal CRUD das entidades desta aplicação, sendo só possível aceder se o utilizador for administrador.

Tela de utilizadores do Dashboard

Avatar	ID	Nome	Email	Password	Genero	Admin	About	Telefone	Profissão	Localização	Edit	Remove
	35	Tiago Cardoso	aluno21607@ipt.pt	*****	Masculino	×						
	33	nome	email	*****	genero	✓	aboutme	telephone	jobtitle	locations		
	34	João	joao@gmail.com	*****	carlo	×	Ola meu nome					
	8	Teste account	testeaccount123@gmail.c...	*****	admin	×						
	2	Paulo daniel	aluno24880@ipt.pt	*****	masculino	✓	Hy my name is Pa...	345632112	Programmer	Tomar		

Rows per page: 5 1-5 of 9

Figura 28 - Gestão de utilizadores.

Nesta página temos a oportunidade de editar campos como email, password, gênero, privilégios de administrador, bem como remover o utilizador de todo da tabela.

Para isso usamos endpoints como “UpdateUserById” e “DeleteUserById”.

Tela de publicações do Dashboard

Imagem	ID	Título	Canal Associado	Mensagem	Data de publicação	Edit	Remove
	1	Date of the final exams	Engenharia, Informática	Dear Students I want to inform you that after 6 months of our cooperation I...	22/01/2024		
	15	Tive 9,912 a SI, será que passei?	Engenharia, Informática	Vamos a ver se se tenho 10! Amanha no NETPA! <3 Ehehe teste de acento...	30/01/2024		
	16	Hello From the other side	Engenharia, Informática	Helloo	30/01/2024		
	17	Side Bar	Engenharia, Informática	Ola A sua esquerda como pode ver temos uma side bar	30/01/2024		
	18	Testando Adicionar Publicações	Engenharia, Informática	Bom como podem ver, estamos adicionando publicações	30/01/2024		

Figura 29 - Gestão de publicações.

Tal como na “[gestão de utilizadores](#)”, nesta página temos a oportunidade de editar campos como título e mensagem (msg), bem como remover a publicação de todo da tabela.

Para isso usamos endpoints como “UpdatePubById” e “DeletePublication”.

Tela de Canais Dashboard

Imagem	ID	Canal	Senha	Edit	Remove
	2	Engenharia, Multimedia	\$2b\$10\$36eF19wnbVoLed6rS1BOQ8TA...		
	4	Conservação e restauro	\$2b\$10\$ZDhaZlpQs967LSyeCd#8Xuböp...		
	5	Tecnologia, Química	\$2b\$10\$S0whpJD7bcnm2DpRkHeOr.D/x...		
	1	Engenharia, Informática	\$2b\$10\$gprE39CcC1TbFo7dEc.yech15j...		

Figura 30 - Gestão de Canais.

Mais uma vez, tal como vimos na “[gestão de publicações](#)”, nesta página temos a oportunidade de editar campos como nome do canal e senha de acesso do mesmo, bem como remover o canal de todo da tabela.

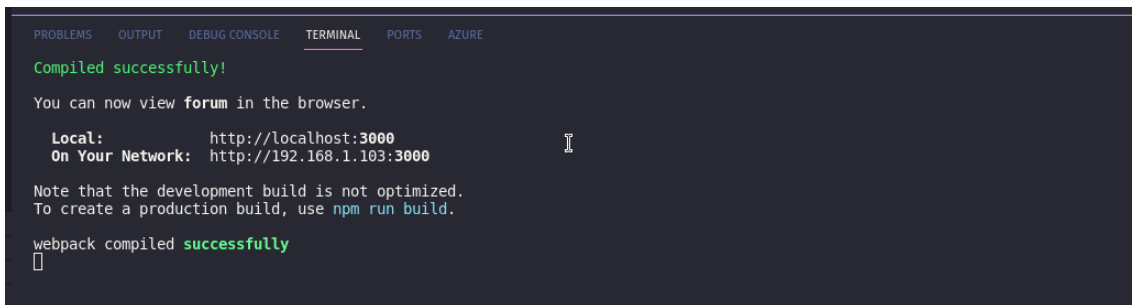
Para isso usamos endpoints como “UpdateChannelById” e “DeleteChannel”.

Como Iniciar a aplicação (<https://cyb3rwolf945.github.io/IWWEB/>)

Existem 2 formas de iniciar o projeto, uma diretamente na web e outra localmente a partir do comando “npm start”.

1º Forma

A primeira forma para ser possível iniciar o projeto localmente usando o comando “npm start”, de forma a criar um web server local na porta 3000.



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  AZURE
Compiled successfully!
You can now view forum in the browser.
  Local:    http://localhost:3000
  On Your Network:  http://192.168.1.103:3000
Note that the development build is not optimized.
To create a production build, use npm run build.
webpack compiled successfully
```

Figura 31 - Web server local.

Este comando iniciará um servidor web local. Em seguida, basta abrir o navegador no url “http://localhost:3000”.

2º Forma

A segunda forma é a partir da plataforma github pages, onde não é preciso iniciar nada, basta aceder através do link <https://cyb3rwolf945.github.io/IWWEB/>.

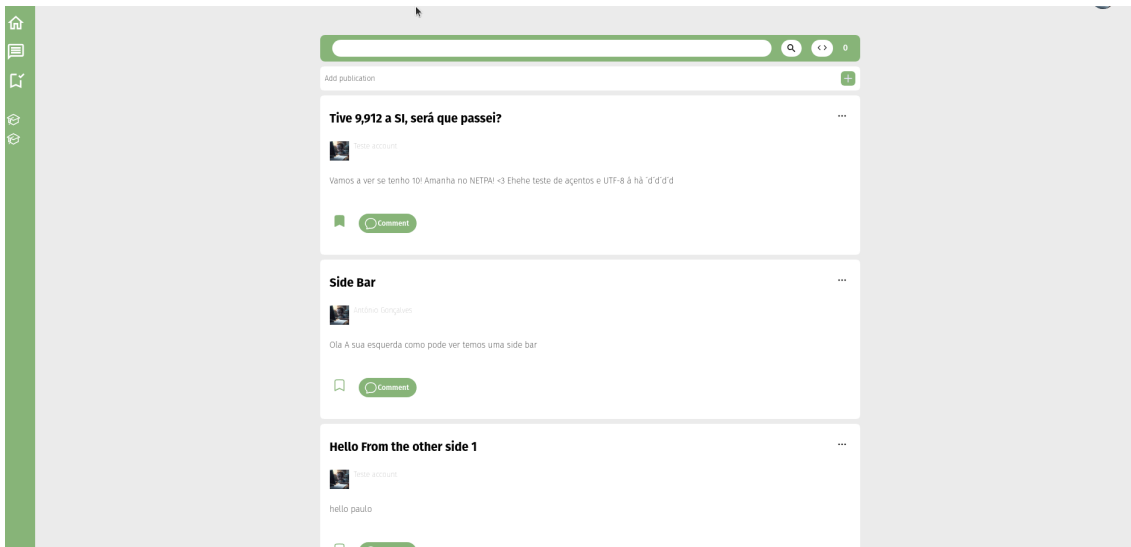


Figura 32 - HomePage no github pages.

Após clicar na respectiva opção uma janela no navegador será aberta automaticamente.

Para concluir podemos referir que a partir do github pages, é que a versão de build não sei “Out of the box”, é preciso alguns ajustes nomeadamente no que toca a rotas, pois o módulo “BrowserRouter”, tivemos de usar o HashedRouter.

Desafios

Durante a realização deste trabalho, deparamo-nos com vários desafios que exigiram a nossa resiliência. Desde o início, deparamo-nos com erros de Cors origin nas funções do Azure, o que inicialmente complicou o desenvolvimento. Foi um processo de aprendizagem constante, especialmente quando lidamos com fusões que não foram bem-sucedidas. A integração da base de dados também demandou um período considerável para ser implementada com sucesso no início.

Contudo, talvez o maior desafio que enfrentamos ao longo deste projeto tenha sido a utilização do próprio fetch. Em alguns momentos, parecia que não estava a funcionar conforme o esperado, o que nos obrigou a analisar a forma como estávamos a enviar os dados no formato JSON. Este aspecto em particular revelou-se mais complexo do que esperávamos e exigiu uma abordagem cuidadosa para garantir a eficácia das funcionalidades, sendo que, talvez esta complexidade pudesse ser resolvida através de bibliotecas como o “React Query”.

Em suma, cada obstáculo encontrado foi uma oportunidade de aprendizagem e aprimoramento das nossas habilidades.

Conclusão

Para concluir o nosso relatório sobre o projeto, devemos sumarizar, dizendo que achamos que ao envolver a utilização ReactJS, Azure Functions e uma base de dados PostgreSQL, contribuiu para a nossa percepção de diferentes tecnologias atuais e ofereceu-nos uma percepção abrangente de como montar uma solução flexível e escalável. Temos de ter noção que com este projeto conseguimos aprender sobre como constituir páginas a partir de componentes em react , usar bibliotecas como o react-router-dom, para implementar rotas na nossa aplicação. Permitiu-nos ter noção de como conectar um frontend a um backend de forma a fazer esta passagem de dados para API que foi algo que não aprendemos, nem tivemos oportunidade de realizar em mais nenhuma unidade curricular, o que foi uma ótima experiência para nós. Com o desenvolver deste projecto nós também aprendemos a importância de

Bibliografia

<https://chsakell.com/2018/11/18/building-serverless-apps-with-azure-functions/>

<https://learn.microsoft.com/en-us/azure/azure-functions/functions-bindings-http-webhook-trigger?tabs=python-v2%2Cisolated-process%2Cnodejs-v4%2Cfunctionsv2&pivots=programming-language-typescript>

<https://gmfuster.medium.com/deploying-a-react-app-to-github-pages-24c3e5485589>