Relatório de Projeto Final: Innovation Lab

Projeto Final

Realizado por:

Anna Helena Pereira Drumond

Joana Pais Valente

Cliente:

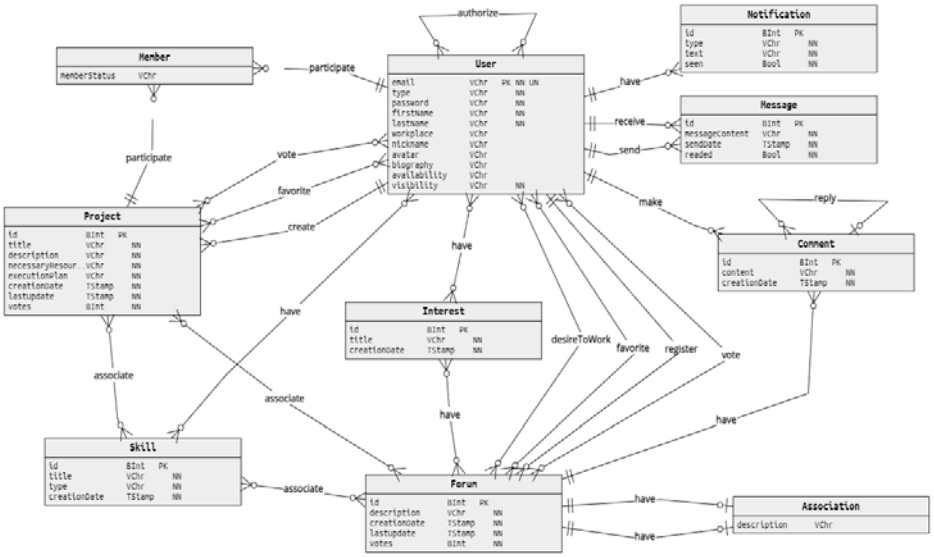
Critical Software

Orientação:

Evgheni Polisciuc

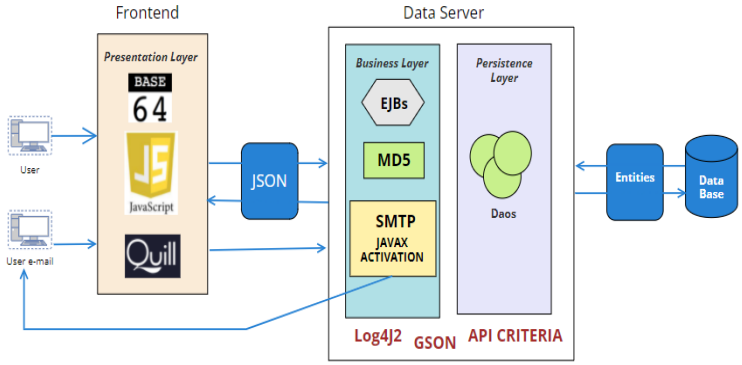
Índice

Do Diagrama Entidade Relacionamento



1 - Diagrama Entidade-Relacionamento

Arquitetura da Solução



2 - Arquitetura da Solução

Das decisões necessárias à implementação do projeto.

1. **Requisitos de Segurança**

1.1. Armazenamento das Palavras-passe:

Para garantir um método seguro quanto ao armazenamento das palavras-passe, foi utilizado o algoritmo de hash MD5 (Message-Digest algorithm 5). Cabe destacar que o MD5 é um algoritmo unidirecional, e, por isso, não pode ser transformado novamente na palavra-passe original. Com isto, o método de verificação é garantido, pela comparação das duas hash (uma da base de dados, e a outra da tentativa de login). Foram ainda adicionados à palavra-passe um "*salt*" e um "*pepper*", garantindo assim um algoritmo hash mais seguro e individual a cada utilizador mesmo em caso de palavras-passe semelhantes. Para maior entendimento ver: snippet método de login. A geração do algoritmo fica à cargo da classe **Encryption** que possui três métodos: um para geração do algoritmo, onde é obtida uma instância do algoritmo MD5, um segundo que recebe o valor origem, no caso, a palavra-passe já com o *salt* e *pepper*, trata de verificações e invoca o terceiro, enviando o valor origem como parâmetro e gerando a chave/hash em si. Para maiores detalhes: https://git.dei.uc.pt/snippets/43.

1.2. Controlo de Acesso:

Para que a aplicação não dependesse de informações armazenadas ou recebidas pelo *browser*/cliente, foi utilizado um objeto do tipo **HttpServletRequest.** Com isto, foi possível no início da sessão (login) armazenar o email do utilizador (chave primária desta entidade), permitindo a validação do mesmo nas demais requisições, onde o utilizador sempre será autenticado através deste email armazenado. Este objeto do tipo **HttpServletRequest** também permitiu recuperar dados da sessão que foram necessários na implementação de outros requisitos adiante analisados. Para garantir a persistência da sessão entre as requisições, foi implementado uma configuração no ficheiro de configurações do servidor Wildfly. Desta forma, em momento nenhum, a aplicação armazena ou troca informações pelo browser/cliente. Para maiores detalhes: https://git.dei.uc.pt/snippets/45.

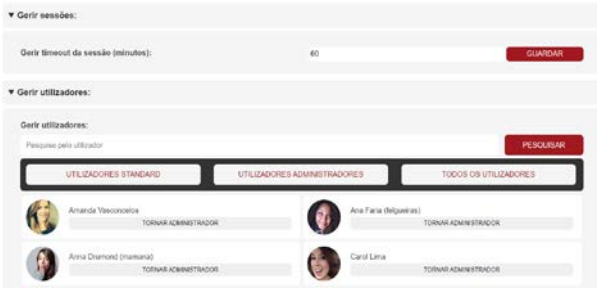
1.3. Session Timeout:

O sistema possui uma entidade denominada Configurations, responsável por oferecer alguns suportes na configuração da aplicação. Esta classe ajuda a garantir o término da sessão após um período de inatividade, pois permitiu armazenar na base de dados um atributo keyword denominado “timeout” e o valor deste timeout, permitindo inclusive a modificação/configuração deste valor pelo administrador do sistema. Pelo lado do frontend, foi implementado um método que irá receber o valor deste Timeout, e irá controlar a inatividade do utilizador e, caso o prazo estabelecido seja ultrapassado, este método providenciará o logout e reencaminhará a pessoa para a página de login.

1.4. Permissões de um administrador do sistema:

Para este requisito, além de verificações implementadas em endpoints do backend, pelo frontend, sempre que um administrador do sistema esteja logado, serão exibidos os botões e/ou páginas de edições/alterações de conteúdo. Por exemplo, ao tentar aceder à página de uma Ideia/Necessidade ou Projeto, caso o utilizador logado seja um administrador do sistema, este terá sempre o botão de edição que lhe permite aceder à página para editar o respetivo conteúdo. Além disso, este tipo de utilizador conta com uma secção própria no sub-menu da área pessoal (imagem acima), e ao aceder a opção “Administração” terá acesso a um ecrã com opções colapsáveis (imagem à esquerda) que lhe permite alterar o valor do Session Timeout, gerir todos os utilizadores do sistema, bem como alterar o status dos mesmos. A fim de facilitar esta gestão e procurar por determinada pessoa, o ecrã conta com um filtro com uma busca ativa por nome/apelido ou alcunha.

4 - Menu



3- Página de administração do sistema

1.5. Logs:

Para garantir o registo em logs de todas as atividades do sistema, foi utilizada a ferramenta Log4j2 configurada de forma programática. Foi decidido que os logs seriam registados num ficheiro armazenado dentro do servidor Wildfly. A implementação e configuração da ferramenta, ficou a cargo da classe **LogGenerator** que possui três métodos: o primeiro recebe o objeto **HttpServletRequest** e um objeto ActionLog (enum que armazena todas as ações do programa) e é responsável pela configuração do log a ser gerado; este invoca o segundo método, responsável por garantir que seja “capturado” o IP do cliente, mesmo que este esteja utilizando um proxy aquando do acesso a aplicação. O primeiro método irá ainda invocar o terceiro método denominado returnEmailActionAuthor(), responsável por garantir que seja obtido o email (chave primária) do cliente, ou seja, o identificador do autor da ação. E por fim, a cada requisição (atividade) executada por qualquer utilizador do sistema será invocado o método generateAndWriteLog(), que irá gerar e inserir um novo log no ficheiro armazenado no servidor.

2. Requisitos Funcionais

Primeiramente, cabe informar que no que se refere às consultas realizadas à base de dados, todas foram construídas utilizando a **API CRITERIA**, tal decisão fundamentou-se no facto de que a API permitiu consultas mais específicas e estruturadas, cruciais para este projeto. Cabe também esclarecer que, a entidade Idea e a entidade Necessity por serem idênticas, foram condensadas numa única entidade denominada Forum.

2.1. Dos requisitos referentes exclusivamente a utilizador:

Antes de mais, cabe esclarecer que a entidade User conta com um enum com quatro tipos disponíveis: (1) VISITOR, (2) ADMINISTRATOR, (3) UNNAUTHENTICATED, (4) STANDARD.É importante esclarecer que ficou definido que utilizadores do tipo 1 não podem editar/criar nada na aplicação, assim como não podem ser convidados ou pedir participação em projetos, enviar/receber mensagens e comentários, somente lhes sendo permitido visualizar a lista de ideias e necessidades registadas na aplicação.

2.1.1. Envio de emails para Validação de Registo e Reset de Password:

Para levar a cabo esta função, foi criado o e-mail “innovationlab.aor@gmail.com”, utilizado o servidor SMTP do Gmail/Google, bem como foram efetuadas algumas configurações no servidor Wildfly. A lógica necessária foi implementada na classe **SendHTMLEmail,** que configura, cria e envia o e-mail ao utilizador. Este e-mail, por sua vez, é formado pelo html que estrutura o corpo do mesmo, o link que irá redirecionar o utilizador para a página respetiva (reset password ou validar registo) e um token provisório, que será utilizado para autenticar o utilizador, garantindo que a respetiva ação seja realizada pelo utilizador correto. No que se refere ao requisito sobre a autenticação e criação de sessão, tal facto já foi explicado no item 1.2 deste relatório.

2.1.2. Visibilidade da área pessoal:

No que se refere a este requisito, ficou definida uma relação recursiva ManyToMany na entidade User, que conta ainda com um segundo enum com os valores PUBLIC, PRIVATE, ESPECIFIC. Para garantir a funcionalidade da visibilidade específica,

a aplicação conta com sete endpoints: (1) um para busca ativa de utilizadores, e que a cada letra escrita vai à base de dados buscar todos os utilizadores que tenham nome, apelido ou alcunha com a(s) letra(s) fornecida(s); (2) outro para a busca ativa por local de trabalho; (3) um responsável por adicionar utilizadores um a um na lista de visualizadores; (4) um endpoint que permite adicionar utilizadores por skill(s), interesse(s) e/ou workplace; (5) outro que permite remover todos os utilizadores da lista; (6) um que permite remover todos os utilizadores com um determinado local de trabalho; e, por fim, (7) um que permite excluir um a um. Os métodos de adicionar/remover utilizadores obedecem a uma lógica similar: primeiro serão trazidos da base de dados, o utilizador a ser adicionado, o utilizador que está a alterar a sua visibilidade, e a atual lista de visualizadores da área pessoal do último. De seguida, será feita uma breve verificação para que não sejam adicionados utilizadores repetidos, a seguir este novo utilizador é então adicionado à lista garantindo a atualização da mesma, que por sua vez será vinculada de novo ao utilizador, e, por fim, será persistido na base de dados. A lógica de desassociação, segue os mesmos princípios, porém adaptados para a lógica de remover os elementos da lista. Quanto à óptica do utilizador, este tem acesso a um ecrã onde pode gerir a privacidade da sua área pessoal, podendo adicionar pessoas um a um ou por grupos, cabendo destacar que conta sempre com buscas ativas para facilitar e otimizar a sua interação com a aplicação. Além disso, conta ainda com a hipótese de ver toda a sua lista de visualizadores (onde terá acesso à possibilidade de filtrar utilizadores por nome/alcunha ou apelido). Cabe esclarecer que, neste momento não se trata de uma busca ativa, mas de um filtro, de forma a que, caso o utilizador queira remover um determinado utilizador, possa filtrar a lista e encontrar mais facilmente esta pessoa. Por fim, tem ainda ao seu dispor a possibilidade de remover todos os visualizadores da sua lista.

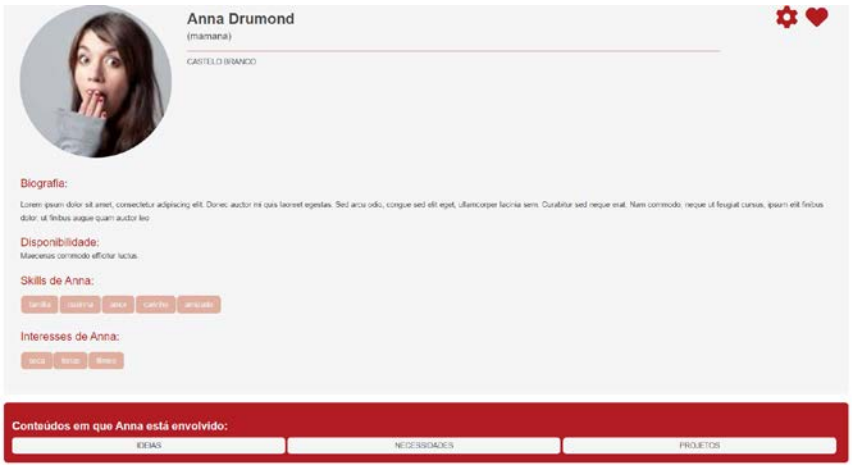
2.1.3. Registo de novos utilizadores:

Para se registar um novo utilizador deve adentrar na aplicação e aceder a parte definida como “Registe-se”, sendo direcionado a um ecrã onde deve preencher seu nome, apelido, email, password e confirmação da password, bem como o seu local de trabalho. Preenchidos estes dados e submetido o pedido, este utilizador recebe o status VISITOR, e lhe é direcionado um email para que possa validar o seu registo. Para esta funcionalidade é utilizada a lógica do item 2.1.1 já descrito acima.

2.2. Dos requisitos referentes à Página Pessoal

Para pessoas com acesso, exibe os dados do utilizador, seus interesses e skills, as ideias/necessidades registadas pelo utilizador (podendo ser filtrados por skill(s) e/ou interesse(s) e ordenáveis por data de criação e votos) e os projetos em que o mesmo consta como envolvido (terminados ou ativo) com a opção de filtrar por skills e ordenar por data e votos.

Para o próprio exibe ainda seus conteúdos favoritos (com opção de filtrar por categoria e ordenar por data de criação, última atualização e quantidade de votos). As lógicas de implementação das listas, filtros e ordenações, seguem os mesmos princípios que serão detalhados mais adiante, há ainda para o próprio utilizador as seguintes funcionalidades:



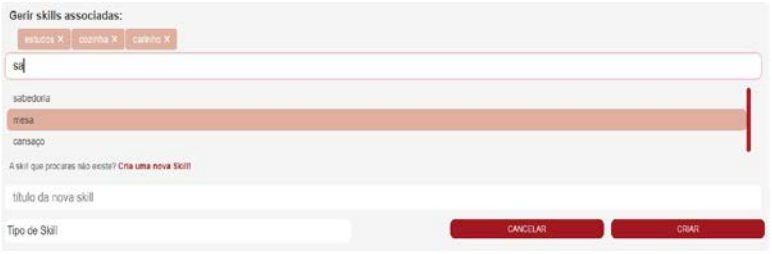
5 - Página pessoal

2.2.1. Busca ativa Skills/Interesses:

A entidade Skill e a entidade Interest são bastante similares, ambas possuem relações ManyToMany com a entidade User e com a entidade Forum (entidade Idea/Necessity), a exceção é que a entidade Skill possui o enum SkillType com os valores KNOWLEDGE, SOFTWARE, HARDWARE, WORKINGTOOLS e, ainda, uma relação ManyToMany com a entidade Project. Para evitar ao máximo a repetição de skills/interesses já registados no sistema, a aplicação possui um endpoint que permite a busca ativa destas entidades, e que, a cada letra digitada, vai à base de dados pesquisar por todas as skills/interesses (conforme o caso), que contenham a(s) letra(s) fornecida(s), para então, devolver ao frontend todas as skills/interesses encontradas, favorecendo assim uma busca ativa antes da criação de novas entidades. No frontend, esta lista é exibida ao utilizador, permitindo que o mesmo escolha uma das opções apresentadas. É importante destacar que, para evitar que o utilizador escolha opções repetidas, somente lhe são exibidas skills/interesses ainda não escolhidos. Abaixo segue uma breve amostra da funcionalidade na óptica do utilizador, sendo similar no caso de interesses.

2.2.2.Criar/registar Skill/Interesse:

Para esta parte, o programa conta com um endpoint relativamente simples, que irá receber do frontend os respetivos dados da skill/interesse digitados pelo utilizador, depois irá converter estes dados para sua respetiva entidade, para, por fim, persistir a nova entidade na base de dados. Cabe destacar que a busca ativa por skills/interesses foi também implementada em outros pontos da aplicação, tais como nas páginas de editar e criar Ideias e Necessidades, adicionar utilizadores a lista de visualizadores por Skill/Interesse, editar e criar Projetos, bem como nos momentos em que o utilizador pode filtrar conteúdos por Skill(s)/Interesse(s), sempre com o objetivo de facilitar a usabilidade da aplicação por parte do utilizador. No frontend, o utilizador terá acesso a esta funcionalidade, assim como à hipótese de associar o respetivo conteúdo a si, na parte da aplicação destinada à edição do seu perfil.



6 - Busca ativa por skills

2.2.3.Associação da entidade Skill/Interest com a entidade User:

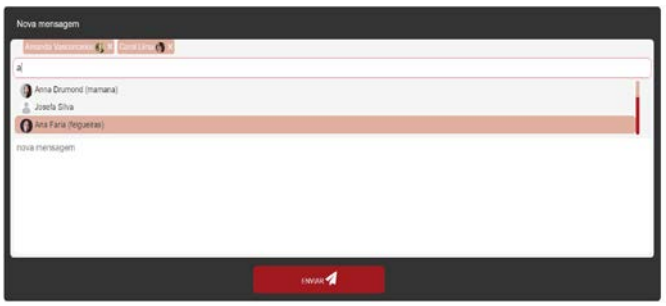
A aplicação contempla um endpoint que recebe o email do utilizador e uma String Json formada pelos ids das skill(s)/interesse(s) que serão associadas ao mesmo, e que, após a sua conversão, facilitada através da biblioteca Gson, permitirá a sua associação ao user. Esta associação segue a seguinte lógica: primeiro serão trazidos da base de dados, a lista de Skills ou de Interests daquele utilizador e a lista de utilizadores que possuem aquela skill/interesse. De seguida, será feita uma breve verificação a fim de que não sejam adicionados elementos repetidos, e, então, cada novo skill/interesse é adicionado à lista do utilizador, assim como, o utilizador será adicionado à lista de utilizadores que possuem aquele skill/interesse, garantindo a atualização das mesmas. Ambas as listas, por sua vez, serão vinculadas de novo às suas respetivas entidades, que por fim serão atualizadas (merge) na base de dados. A aplicação conta ainda com endpoints que comportam a lógica de desassociação, relativa às associações acima descritas, que seguem os mesmos princípios, porém adaptados para a lógica de remover os elementos das listas.

2.2.4.Favoritos:

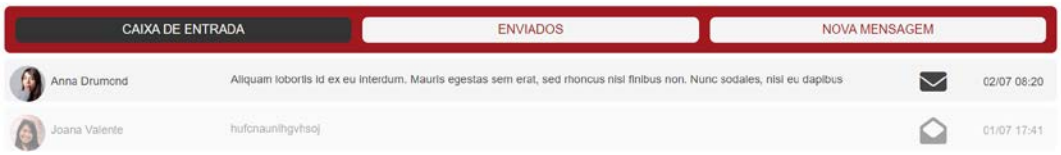
O próprio utilizador, na sua área pessoal, conta com a exibição das ideias/necessidades e projetos que marcou como favorito. Esta lista pode ser filtrada por categorias e ordenada por data de criação, data da última postagem ou votação. As lógicas de implementação das listas, filtros e ordenações, seguem os mesmos princípios que serão detalhados mais adiante.

2.2.5.Mensagens Pessoais:

Para permitir a troca de mensagens entre utilizadores, a entidade User possui duas relações OneToMany com a entidade Message, a fim de armazenarmos o user sender e o user receiver de cada mensagem trocada. Para cumprir este requisito em sua totalidade, a aplicação conta com um endpoint para busca ativa de destinatários, que, a cada letra(s) escrita, vai à base de dados buscar todos os utilizadores que tenham qualquer nome, apelido ou alcunha igual ao valor digitado e retorna esta lista, permitindo que o remetente da mensagem escolha uma das opções apresentadas. Foi definido que um utilizador pode enviar a mesma mensagem a um ou a vários remetentes, conforme deseje (imagem abaixo à direita). Há ainda um segundo endpoint responsável pela lógica que irá, de facto, criar a mensagem. Para isso, este método recebe como parâmetro os emails do remetente e do(s) destinatário(s), traz estes utilizadores na base de dados, recebe também a mensagem trazida do frontend e a converte a mesma numa entidade, persistindo-a na base de dados. Para criar as relações necessárias, é ainda trazida da base de dados a lista de mensagens enviadas do utilizador sender (remetente) e a lista de mensagens recebidas do utilizador receiver (destinatário), para que as mesmas sejam atualizadas com a nova mensagem, vinculadas novamente aos seus respetivos utilizadores, que por fim, serão atualizados (merge) na base de dados já com suas listas atualizadas. Por último, relativamente às consultas das mensagens pessoais de um utilizador. Para este requisito o programa conta com três endpoints distintos: um que irá buscar a lista de mensagens recebidas, e outro para as mensagens enviadas e um terceiro que traz na base de dados todas as mensagens recebidas/enviadas entre dois determinados utilizadores. Há ainda um último endpoint que garante que o utilizador receiver possa mudar o status "*readed*" de suas mensagens, permitindo, desta forma, marcar as mensagens como lidas. Estas surgem



7 - Envio de mensagens



8 - Página de mensagens

com um aspeto diferenciador e com um ícone demonstrativo do seu estado, o que permite a sua diferenciação das restantes. A listagem de mensagens, pode ser visualizada de três maneiras diversas (Histórico de mensagens), porém sempre com a listagem ordenada da mensagem mais recente para a mais antiga: somente as mensagens recebidas, somente as mensagens enviadas, e caso o utilizador clique em qualquer uma das mensagens para lê-la, será redirecionado para um outro ecrã com todas as mensagens trocadas entre ele e aquele outro utilizador.

É fulcral referir que, no momento deste redirecionamento, a nova página abre no local exato da exibição da mensagem clicada pelo utilizador destacando-a das demais, permitindo aceder diretamente ao conteúdo escolhido, porém permitindo também que possa ser visualizada todas as demais mensagens trocadas com aquela pessoa.

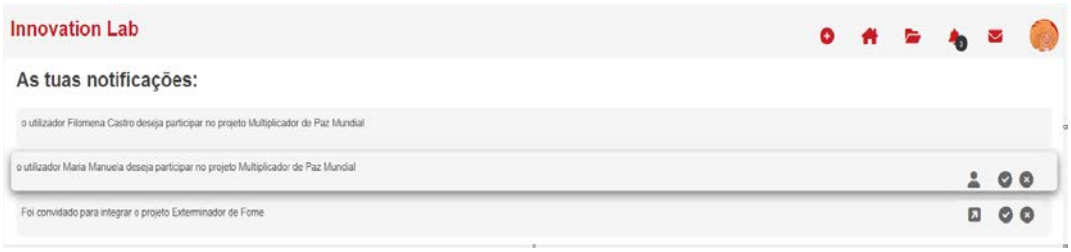
10 - Histórico de mensagens

Foi implementado ainda, que o utilizador tenha sempre no seu menu (presente em todas as páginas da app) a informação da existência de mensagens não lidas, bem como de novas notificações, devendo ser citado também que é no menu que fica a opção de **logout**, a fim de garantir que utilizador, em todos os momentos, tenha acesso a um link que faz a finalização da sessão (Menu).

9 - Menu

2.2.6.Notificações:

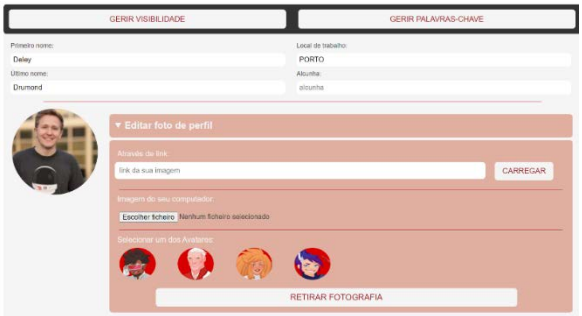
Possuem uma relação ManyToOne com a entidade User. Existem três tipos de notificações, o primeiro é gerado quando um utilizador solicita sua integração/participação num projeto, neste caso os administradores do projeto serão notificados, o segundo tipo é gerado quando este pedido é aceite, neste caso o utilizador que fez o pedido será notificado que seu pedido foi aceite ou rejeitado, e há ainda um terceiro tipo gerado quando um utilizador é convidado a participar de um projeto, neste caso este convidado será notificado para que possa aceitar ou não. Para a criação das notificações, sempre que decorreu com sucesso uma das ações citadas, é então invocado o método responsável por criar a notificação. O método então, irá criar uma nova entidade Notification, que terá seu respetivo "NotificationType", e seus demais atributos para que possa ser então persistida na base de dados. Além disso, é trazida da base de dados a lista de notificações do utilizador a quem se destina a notificação, para que esta lista seja atualizada, vinculada novamente ao utilizador, e este seja atualizado (merge) na base de dados. O programa conta ainda com um endpoint que contém a lógica para que um utilizador possa marcar uma determinada notificação como "lida". Pelo lado do frontend o utilizador terá acesso à uma página que exibe todas as notificações recebidas, de acordo com a respetiva situação e, de acordo com o tipo de notificação será exibido um ícone que permite ao utilizador ir visitar a página pessoal da pessoa a quem se refere o pedido ou a página do projeto a que se refere o convite, por fim são são exibidos ainda os ícones que permitem ao utilizador aceitar ou rejeitar àquele convite ou pedido:



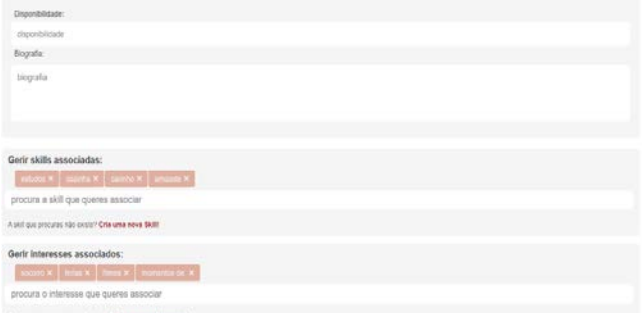
11 - Página de notificações

2.2.7.Editar Perfil:

A aplicação conta com uma área para o utilizador poder gerir seus dados pessoais, sendo que é que aqui que o mesmo pode associar a si/criar novas skills e interesses e alterar seus dados pessoais:

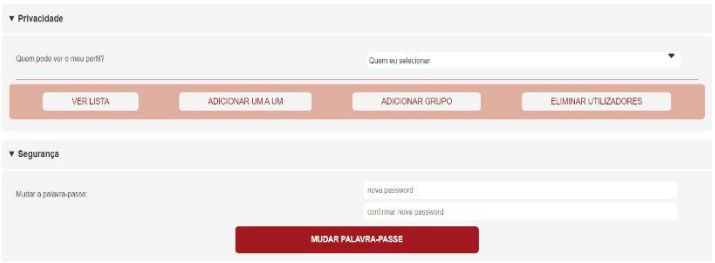


12 - Editar perfil (início)



13 - Editar perfil (fim)

Esta página lhe permite ainda aceder a uma segunda página que dá acesso às funcionalidades de alterar palavra-passe e gerir a visibilidade de sua área pessoal (imagem abaixo à esquerda), que por sua vez permite ao utilizador aceder a página para ver/excluir/pesquisar as pessoas autorizadas a ver o seu perfil (Lista de visualizadores):



14 - Gestão de visualizadores

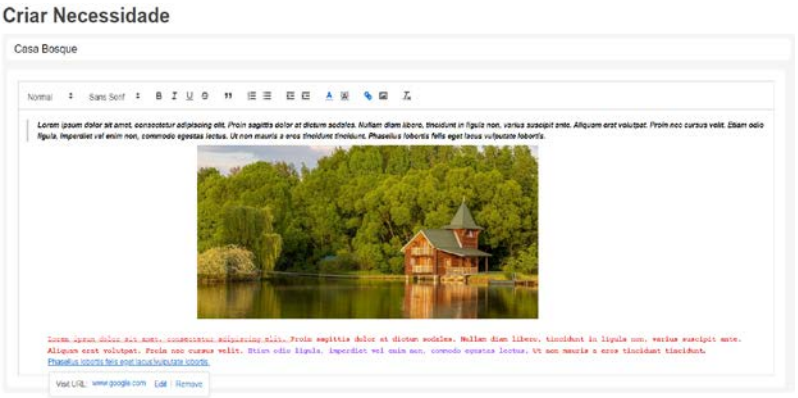
15 - Lista de visualizadores

2.3. Fórum de Ideias e Necessidades:

Antes de mais, vale reforçar que Idea e Necessity por serem idênticas, foram condensadas em uma única entidade denominada Forum, que conforme diagrama constante no início deste relatório possui: três relações ManyToMany (com Skill, Interest e Project), uma relação ManyToOne com a entidade User, duas relações OneToMany com a entidade Associação (criada devido a possibilidade de existência de de um texto descritivo, quando da associação entre Forums), e por fim um relação OneToMany com a entidade Comment. Esta página implementa as funcionalidades de ordenação e filtros descritos no item 2.3.8, e exibe informações do título, criador, quantidade de votos, data de criação e data da última interação (neste caso a última vez que alguém interagiu com o conteúdo, seja ao comentar ou editar o mesmo) do respetivo conteúdo.

2.3.1. Criar uma Ideia/Necessidade (Forum):

Para esta parte, o programa conta com um endpoint relativamente simples, que irá receber do frontend os respetivos dados da ideia/necessidade escritos pelo utilizador, convertendo-os na sua entidade com tipo respetivo, para, por fim, persistir a nova entidade na base de dados. Quanto à implementação deste requisito pelo lado do frontend, e tendo em conta o requisito de que este conteúdo deveria suportar a sua descrição em texto livre e rico, foi utilizado a ferramenta Quill Rich Text Editor, sendo configurado programaticamente os módulos a serem utilizados pelo editor. Com isto, o utilizador ao criar o seu conteúdo, dispõe de diversas opções de edição. A fim de garantir que, aquando do acesso ao conteúdo, sejam preservadas e exibidas todas as edições realizadas no momento da sua criação, a lógica implementada no lado do frontend irá "capturar" todo o conteúdo HTML gerado pelo Quill, inclusive as tags HTML de estilos, e enviará tudo para a base de dados. Foi implementada ainda a lógica que permite a edição de uma ideia/necessidade.

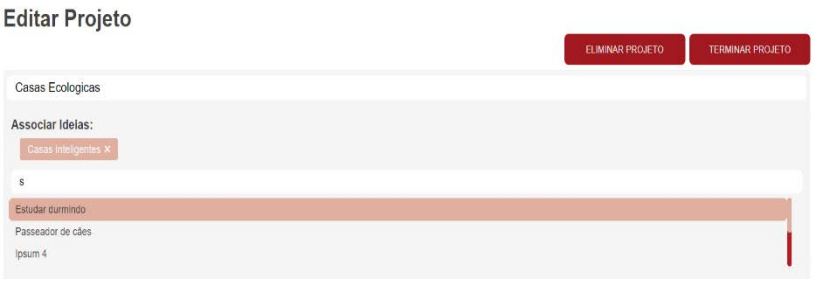


16 - Criar necessidade

2.3.2. Associação de uma Ideia/Necessidade (Forum) com outra Ideia ou com uma Necessidade:

Quanto às associações entre entidades Forum, para a lógica envolvida foi implementado um endpoint que irá receber do frontend os ids das entidades que se vão associar, o texto descritivo desta associação e o email do utilizador que está a praticar a ação. Este endpoint, por sua vez, invoca o método dotado da seguinte lógica: primeiro serão trazidos da base de dados a entidade original e a entidade que será associada à mesma, bem como são trazidas as suas respetivas listas de associações, lógica esta, similar ao já explicitado no item 2.2.3 deste relatório. Após uma breve verificação para que não sejam adicionados elementos repetidos, o Forum a ser associado é então adicionado às listas de associações do Forum original e vice-versa, seguido da persistência da nova associação e por último da atualização (merge) da entidade relativa ao Forum original e da entidade do Forum associada naquele momento, ambos já com suas respetivas listas de associações atualizadas. A aplicação conta ainda com endpoints que comportam a lógica de desassociação, relativa às associações descritas, que seguem os mesmos princípios, porém adaptados para a lógica de remover os elementos das listas.

2.3.3. Associação de uma Ideia/Necessidade (Forum) com Skills/Interesses:



17 - Editar projeto

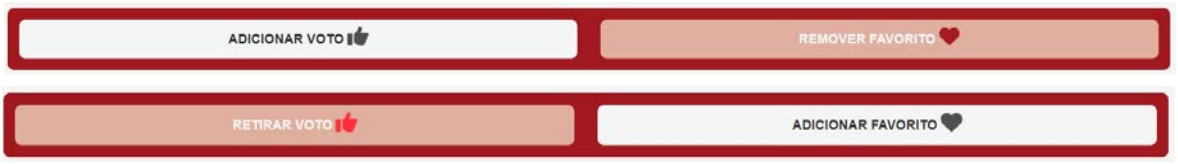
A lógica envolvida para estas associações segue o mesmo princípio descrito no item 2.2.3 deste breve relatório, porém com as respetivas entidades a associação atual. Pelo lado do frontend, esta funcionalidade conta com a busca ativa descrita no item 2.2.1.

2.3.4. Associação de uma Ideia/Necessidade (Forum) com Projetos:

A lógica envolvida para esta associação segue o mesmo princípio descrito no item 2.3.3 deste breve relatório. Pelo lado do frontend, esta funcionalidade conta com uma busca ativa similar àquela descrita no item 2.2.1, porém adaptada à respetiva situação:

2.3.5. Marcar uma Ideia/Necessidade (Forum) como favorita:

Para que um utilizador possa adicionar um determinado Forum à sua lista de Forums favoritos, a lógica também é semelhante ao descrito no item 2.2.3 supramencionado, contudo, cabe destacar que, neste caso, será trazida da base de dados o Forum a ser favoritado, o utilizador que quer favoritar aquele Forum, bem como as respetivas listas de associações entre estas entidades. A seguir, este Forum será adicionado a lista de Forums favoritos daquele utilizador, e este por sua vez, será adicionado à lista de utilizadores que favoritaram aquele determinado Forum. O programa contém ainda a hipótese de retirar esta marcação como favorito. Já pelo lado do frontend, estas funcionalidades apresentarão o layout abaixo, conforme o estado em que se encontrem a funcionalidade se encontra na página da ideia/necessidade, onde o utilizador contará com um mesmo botão para adicionar/remover favorito, que apresentará um layout de acordo com a funcionalidade atual do mesmo:



18 - Adicionar favorito / voto

Além desta opção, a página de ideia/necessidade traz seus dados básicos, e ainda a funcionalidade de votar/retirar voto, exibe as ideias/necessidades, skills e interesses associados ao conteúdo, a data de criação do mesmo e a quantidade de votos.

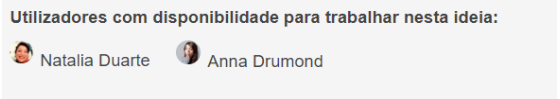
2.3.6. Votar em uma ideia/Necessidade (Forum):

Para cumprimento deste requisito, a lógica utilizada foi a mesma destacada no item 2.3.5 acima, inclusive no que se refere ao layout disponível para o utilizador, sendo que as listas de associações neste caso, são respetivas à associação a ser criada neste momento. Reitera-se que o programa contém ainda o endpoint com a lógica para o utilizador retirar este voto.

2.3.7. Utilizador indicar que tem disponibilidade para trabalhar em uma Ideia/Necessidade:

Para cumprimento deste requisito, a lógica utilizada foi a mesma destacada no item 2.3.4. Esta funcionalidade está implementada na página da Ideia/Necessidade, sendo exibido ao utilizador um botão para esta ação. Caso o utilizador opte por mostrar disponibilidade, esta indicação é exibida na página do respetivo conteúdo (exemplo à esquerda).

19 - Página de necessidade



20 - utilizadores disponíveis

2.3.8. Pesquisar/Busca ativa por Ideias e Necessidades (Forum):

Para que um utilizador possa aceder a lista de todos os Forums registados no sistema e filtrá-la por categorias (Ideias/Necessidades), Interesses e/ou Skills associadas às Ideias e Necessidades, foram implementados endpoints que receberão os critérios de pesquisa escolhidos pelo utilizador, e utilizarão da API CRITERIA para fazer a busca na base de dados a fim de retornar a lista respetiva ao padrão determinado na pesquisa. Quanto à ordenação, a lista de Forums pode ser ordenada pela sua de data de criação (registo inicial da Ideia/Necessidade), data da última interação (de qualquer utilizador) e votação, ficou a cargo do frontend implementar diretamente esta lógica através da comparação entre os elementos da lista, verificando os maiores e menores valores e ordenando a lista de acordo com o pedido do utilizador, qual seja, em ordem crescente ou decrescente. Para mais detalhes quanto a lógica de ordenação ver: https://git.dei.uc.pt/snippets/53. Quanto ao layout adotado para esta funcionalidade, a página respetiva ao feed de Ideias e Necessidades, conta com conteúdos colapsáveis, permitindo um layout mais limpo e funcional. Ao optar por filtrar conteúdos, o utilizador terá acesso a espaços para fazer busca ativa por interesse(s) e/ou skill(s), a serem usados em seu critério de pesquisa, e também a hipótese de filtrar por categorias. Logo abaixo terá ainda a hipótese de ordenar o conteúdo visualizado em ordem ascendente ou decrescente, conforme queira:

21 - Busca ativa de ideias / necessidades

22 - Botões de ordenação

2.3.9. Comentar uma Ideia/Necessidade (Forum):

Para a funcionalidade dos comentários, a aplicação conta com a entidade Comment que possui uma relação ManyToOne com a entidade User e outra ManyToOne com a entidade Forum, para além destas possui ainda uma relação OneToMany recursiva consigo mesma. Para organização dos comentários e de suas respetivas respostas, a entidade Comment possui um atributo boolean "*reply*". A lógica de criação de um comentário é relativamente simples: o endpoint recebe o email do utilizador criador do comentário, o Forum onde realizou o comentário e o conteúdo do comentário em si. A seguir, é criada uma nova entidade Comment, com o atributo *reply* definido como *false* (pois não se trata de uma resposta, mas sim de comentário original), podendo, com isto, ser persistida na base de dados. Para responder a um comentário, o programa conta com um segundo endpoint que recebe o id do comentário original, o utilizador criador da resposta e o texto em si. Este comentário (resposta) é então convertido numa entidade, depois são definidas as relações de forma similar às já demonstradas anteriormente neste relatório, todavia, ressalva-se que o atributo *reply* recebe o valor *true*. Esta resposta é inserida na lista de respostas do comentário original, que por fim será atualizado (merge) na base de dados. No que se refere a visualização dos comentários, esta funcionalidade é encontrada na página da respetiva ideia/necessidade, e trata-se de um conteúdo colapsável exibido ao final da página (imagem ao lado). Uma vez que o utilizador, "abre" o colapsável tem acesso a toda lista de comentários deste Forum, que exibem o comentador, o conteúdo do mesmo bem como sua data de criação e quantidade de respostas associadas, e com as opções de editar, responder e/ou excluir de acordo com seu status (criador do comentário/resposta ou administrador do sistema).

23 - Comentários

24 - Comentários

2.3.10. Editar Ideia/Necessidade:

Foi implementada ainda uma página que permite ao criador do conteúdo ou ao administrador do sistema editar qualquer uma das associações ou informações de uma ideia/necessidade. Este ecrã permite também a edição do texto descritivo opcional que

explica o motivo ou contexto da associação entre uma ideia/necessidade com outra ideia/necessidade. Está ainda ao dispor do utilizador a possibilidade de apagar uma ideia/necessidade.

2.4. Gestão de Projetos:

O sistema contém uma entidade denominada Project que possui uma relação ManyToOne com a entidade User, relações ManyToMany com as entidades Forum e Skills, e ainda mais duas relações ManyToMany com a entidade User. Cabe destacar ainda que para os requisitos inerentes a Projetos, o programa traz ainda a entidade Member a fim de ser possível a gestão dos membros envolvidos num projeto. Com isso, temos a última relação definida na entidade Projeto, que é uma relação OneToMany com a entidade Member. É essencial referir que a entidade Projeto possui um atributo boolean "*active*" de forma a verificar se aquele determinado projeto ainda está ativo ou se já foi concluído, existindo um endpoint que permite a um administrador daquele projeto alterar este status. Como nas demais entidades, foi implementada ainda a lógica que permite a edição de um projeto.

2.4.1. Criação de um Projeto:

Para esta parte, o programa conta com um endpoint relativamente simples, que irá verificar se o utilizador que está a tentar criar um projeto já se encontra envolvido nalgum projeto com status ativo. Caso não esteja, vai receber do frontend os respetivos dados do Projeto digitados pelo utilizador, depois irá converter estes dados para uma nova entidade, para, por fim, persistir a mesma na base de dados. Do lado do frontend, consta um editor de texto idêntico ao descrito no item 2.3.1 deste relatório. Deve-se referir que o projeto somente pode ser criado caso tenha sido associado a si, no mínimo, uma ideia ou necessidade

2.4.2. Favoritar um Projeto / Votar em um Projeto:

A lógica envolvida para este requisito segue o mesmo princípio descrito no item 2.3.5 e 2.3.6, respetivamente, deste breve relatório.

2.4.3. Associar Projeto a uma ideia/necessidade (Forum) / Associar uma Skill a um Projeto:

A lógica envolvida para esta associação segue o mesmo princípio descrito no item 2.2.3 deste breve relatório, bem como segue a mesma lógica, porém inversa para a desassociação destas entidades.

2.4.4. Gestão de membros de um Projeto:

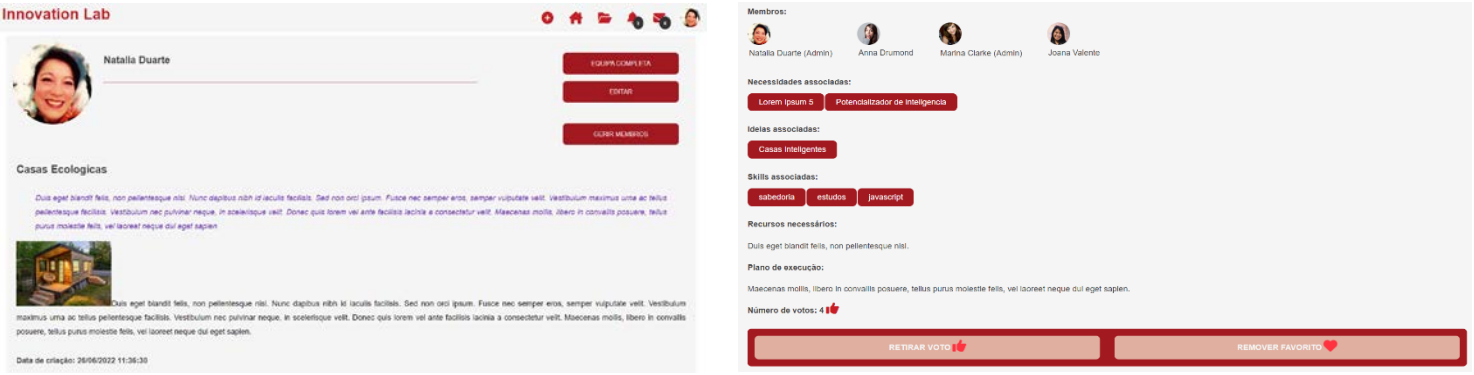
Para esta parte ficou definido que: um membro pode ter quatro status (1) SOLICITOR, (2) INVITED, (3) ADMINISTRATOR, (4) PARTICIPATOR, definidos num Enum na entidade Member. No momento que um utilizador é convidado a integrar um projeto, é verificada a quantidade de membros dos tipos 3 e 4 já associados (tendo em conta que, o limite é de 4 membros), bem como, são feitas verificações a fim de evitar a possibilidade de convidar um utilizador que já se encontre envolvido em um projeto com status ativo. Para selecionar os utilizadores a serem convidados (seguindo a mesma lógica programática do item 2.2.3), estes utilizadores ficam com o status 2, sendo necessário que o utilizador confirme o seu desejo em integrar o projeto para se firmar a associação entre as entidades (esta que atende ao formato programático já destacado no item 2.2.3 deste relatório).Após o convite com sucesso, o utilizador convidado recebe então uma notificação sobre o mesmo e somente irá integrar de facto o projeto, caso aceite este convite. Quanto à hipótese de um utilizador pedir para integrar um projeto, também são feitas as mesmas verificações já citadas. E, caso as verificações sejam todas ultrapassadas, e a requisição decorra com sucesso, este utilizador **s**erá adicionado à lista de membros daquele projeto (mesma lógica do item 2.2.3), porém, com o status 1. Para que o utilizador passe ao status 4, a aplicação possui um endpoint que permite a qualquer um dos administradores do projeto aceitar o pedido (firmando as associações entre as entidades, da mesma forma como foi supramencionado). É de referir que todos os administradores do projeto serão notificados deste pedido, e, se for aceite, a notificação é eliminada da lista dos restantes administradores. O programa também contém um endpoint com a lógica necessária para os administradores do projeto rejeitarem o pedido, que segue uma lógica similar, porém inversa. Pelo lado do frontend, a lógica quanto à receção de notificações e aceitação ou não de um convite/pedido está relacionada com a página de notificações (ver item 2.2.6). Ademais, existe ainda a lógica para promover/despromover um membro status 4 para status 3 e vice-versa, caso um administrador seja despromovido para participante perde a hipótese de editar/alterar o projeto, bem como tem retirada de sua lista de notificações, aquelas que são pertinentes a pedidos de utilizadores para integrar o projeto. A aplicação contém ainda o endpoint com a lógica que permite remover membros de um projeto.

Pelo lado do frontend, para esta segunda parte, existe a página de gerir membros de um projeto (imagem acima e ao lado), acessível apenas a administradores do sistema/projeto que conta com uma busca ativa que irá retornar todos os utilizadores que não sejam participantes ou administradores em um projeto ativo, ou seja, aqueles utilizadores que poderiam de fato, ser convidados a integrar um projeto, além disso esta página, permite visualizar a lista de administradores, participantes, convites e pedidos pendentes:

25 - Gerir membros de um projeto

2.4.5.Página do Projeto:

Esta página traz todas as informações do projeto, além de exibir sua data de criação, quantidade de votos, opção de votar/retirar voto, incluir/remover de favoritos, necessidades/ideias associadas, skills associadas, vai ainda possuir botões a serem exibidos de acordo com a situação do projeto e/ou status da pessoa logada no momento, podendo exibir informações de que a equipa do projeto está completa ou um botão para solicitar a participação no projeto, que somente será exibido caso o projeto tenha menos de 4 membros, podendo também exibir um botão com a opção de gerir membros, e um botão de editar, caso quem estiver logado seja um administrador do projeto ou administrador do sistema.



26 - Página do projeto

No que se refere a convidar utilizadores para integrar um projeto, esta funcionalidade está localizada na página de gestão de membros do projeto citada no item 2.4.4 acima.

2.4.6.Pesquisar/Filtrar Projetos:

Para que um utilizador possa aceder à lista de todos os Projetos registados no sistema e filtrar esta lista por Ideias, Necessidades ou Skills associadas a um projeto foram implementados endpoints que irão receber os critérios de pesquisa escolhidos pelo utilizador para fazer a busca na base de dados a fim de retornar a lista respetiva ao padrão determinado na pesquisa. Quanto à hipótese de ordenar esta lista em termos de data de criação e número de membros em falta, esta implementação ficou a cargo do frontend ficou a cargo do frontend (a lógica desta funcionalidade, bem como o layout na óptica do utilizador, são similares ao descrito no item 2.3.8 deste relatório).

2.4.7.Editar Projeto:

Foi implementada ainda uma página que permite aos administradores do projeto ou ao administrador do sistema editar qualquer uma das associações ou informações de um projeto. Este ecrã permite também que o projeto tenha seu status alterado de projeto ativo para projeto terminado, bem como permite que o utilizador possa apagar este projeto.

3. Segurança frontend:

Por último, cabe informar que foram garantidos alguns critérios de segurança a nível de acesso pelo utilizador. O primeiro refere-se ao momento em que o mesmo “entra” em cada página da aplicação. Neste momento de “entrada” é sempre verificado se este utilizador tem permissão para ali estar, e, somente após ser feita esta validação, é que então a página e os seus respetivos conteúdos são carregados, permitindo uma maior segurança, mesmo que um utilizador venha a alterar diretamente a URL da página para aceder conteúdos que não está autorizado. O segundo critério foi a utilização da ferramenta de codificação BASE64, em momentos onde se fazia necessário transportar informações entre páginas. Desta maneira, os dados necessários são sempre codificados antes de serem incluídos no “data URL”, e após na página de destino os mesmos são decodificados para o respetivo uso, garantindo a segurança destes dados. Ademais, as páginas contam com lógicas de tratamentos de erros e o programa conta ainda com uma página com um layout informativo para erros mais gerais, onde neste caso, o utilizador será redirecionado a esta página a fim de ser comunicado de que algo correu mal.