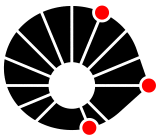
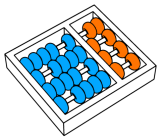




Plataforma de Match de Projetos

Caio L. Silveira de Sousa *Matheus V. Mazon*
Thomas G. Ferreira *Gustavo H. Libraiz Teixeira*
Lucas H. Machado Domingues *Juliana Freitag Borin*

Relatório Técnico - IC-PFG-22-11
Projeto Final de Graduação
2022 - Julho

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

Plataforma de Match de Projetos

Caio Lucas Silveira de Sousa* Matheus Vicente Mazon* Thomas Gomes Ferreira*
Gustavo Henrique Libraiz Teixeira* Lucas Henrique Machado Domingues*
Juliana Freitag Borin*

Resumo

Alunos da universidade estão sempre engajados em projetos, sejam estes provenientes de disciplinas acadêmicas ou não. Em sua maioria, são projetos complexos que envolvem grandes quantidades de pessoas e procuram por habilidades e interesses específicos. Desse modo, surge uma questão: como encontrar projetos e pessoas?

Este trabalho consiste na implementação de uma plataforma de *Match de Projetos*, em que perfis interessados (alunos, professores, entidades externas à universidade) em projetos podem encontrá-los por meio de uma rede social dedicada a isso. Assim como idealizadores e responsáveis por projetos podem encontrar pessoas para agregar na força de trabalho.

Ao fim do desenvolvimento, foi possível entregar uma plataforma que implementa a maioria dos requisitos definidos inicialmente, e dessa maneira, apresenta aplicabilidade em contextos reais. Além disso, a plataforma, disponível no formato *Web* em *React*, consome uma API *REST* implementada em uma arquitetura de microsserviços, apresentando potenciais ganhos em escalabilidade e disponibilidade.

1 Introdução

O Espaço Plasma é um *coworking*, localizado na Universidade Estadual de Campinas, que possui o objetivo de incentivar o desenvolvimento acadêmico, científico e tecnológico através da criação de um espaço interdisciplinar e colaborativo para a criação de projetos. Sabe-se, atualmente, alguns dos fatores mais importantes do desenvolvimento técno-científico na nossa sociedade são divulgação de informação e conhecimento, comunicação e colaboração. Pensando nisso, esse trabalho teve o intuito de realizar uma plataforma que ajudará o Plasma na divulgação de projetos.

A plataforma criada e nomeada "Plataforma de *Match* de Projetos" tem funcionalidades como criação, busca de projetos e perfis, demonstração de interesse e notificações para proporcionar a divulgação de projetos, engajamento e demonstração de interesse entre usuários e projetos simulando uma rede social entre eles. Todas as funcionalidades da plataforma serão descritas ao longo desse relatório, juntamente com todo o processo de planejamento e desenvolvimento do projeto. O *match*, funcionalidade que recebe o nome da plataforma, é uma dupla relação entre um projeto e um usuário. Na tela principal, é possível buscar por projetos ou usuários, e marcar interesse nestes. Ao marcar interesse em um projeto, temos uma relação do usuário para esse projeto. Já para marcar interesse em um usuário, é preciso selecionar um projeto em que seja dono, e ao marcar interesse, temos uma relação do projeto para esse usuário. O match ocorre quando existe interesse mútuo entre um projeto e um usuário.

Na seção de metodologia é discutido todo o processo de planejamento e pré desenvolvimento. Ela é dividida em duas partes:

*Instituto de Computação, Universidade Estadual de Campinas, 13083-852 Campinas, SP

- Levantamento de Requisitos: Discussão sobre as funcionalidades requeridas no projeto;
- Decisão de Arquitetura: Discussão sobre as tecnologias e estrutura que foram utilizadas no projeto. Com o pensamento que a plataforma é uma rede social de projetos, visamos a construção de uma estrutura que foca na escalabilidade e disponibilidade optando por uma estrutura principal de microsserviços alinhada com computação em nuvem.

Na seção de desenvolvimento é discutido todo o processo da implementação do projeto, detalhes de arquitetura e tecnologias utilizadas. Ela é dividida em duas partes:

- Backend: Apresenta o processo de desenvolvimento estrutura de backend do projeto, a apresentação das linguagens utilizadas e os microsserviços criados.
- Frontend: Apresenta as tecnologias utilizadas na camada de frontend do projeto.

Na seção de Resultados e discussões são apresentadas todas as telas desenvolvidas no projeto junto com a explicação de seu funcionamento.

Na seção de Considerações Finais são apresentadas considerações sobre o processo de desenvolvimento, conclusões sobre e possíveis melhorias futuras para o projeto.

2 Metodologia

Nesta seção serão discutidas as etapas anteriores ao desenvolvimento das funcionalidades da plataforma, como o levantamento de requisitos e decisões técnicas em geral.

2.1 Levantamento de Requisitos

O levantamento dos requisitos e das regras de negócio foi feito de maneira pragmática. A descrição inicial do projeto foi alinhada com os interessados pelo projeto, e após algumas reuniões e sessões de *brainstorm*, os requisitos foram definidos.

Em suma, foi definido que seria necessário desenvolver uma plataforma em que usuários compartilham seus projetos com a comunidade estudantil, além de seus interesses. O diferencial da plataforma seria a capacidade dos integrantes de um projeto encontrarem usuários da plataforma interessados pelo projeto. Isso seria feito a partir de um *match* em que usuários podem manifestar interesse por um projeto, assim como a recíproca é verdadeira: proprietários de um projeto podem manisfestar interesse em um usuário.

De maneira mais específica, segue a lista de requisitos levantada:

- Sistema de autenticação, que permite identificação e verificação de membros ligados à comunidade acadêmica.
- Cargos de usuários em projetos.
- Perfis de usuários, com informações sobre a pessoa, formas de contato, imagem de perfil, etc.
- Cadastro de projetos por usuários, com informações sobre o projeto e seus integrantes.
- Vitrine de projetos com filtros a partir de interesses em comum.
- Vitrine de perfis com filtros a partir de interesses em comum.
- Sistema de curtidas em projetos.

- Match entre usuário e projeto.
- Sistema de notificações, em que eventos importantes são notificados para as partes interessadas. Por exemplo, se ocorre um match entre um usuário e um projeto, o usuário em questão deve ser notificado, assim como os integrantes do projeto.

2.2 Decisão de Arquitetura

Diante do desafio de construir uma rede social, fica claro que escalabilidade e disponibilidade são exigências de um sistema desse tipo. Como uma forma de mitigar esse problema, desacoplar os serviços do sistema se mostra cada vez mais uma solução efetiva.

Como uma solução concreta, a arquitetura de microsserviços alinhada com computação em nuvem ganha notoriedade na indústria. Ao contrário da tradicional arquitetura monolítica, os microsserviços são unidades, geralmente pequenas, escaláveis e independentes dos demais serviços.

Mesmo que os microsserviços sejam projetados para serem os mais independentes possíveis, é natural que eventualmente seja necessário que eles se comuniquem. A estratégia de comunicação síncrona não é muito recomendada, devido ao potencial acoplamento entre os serviços. Nesse contexto, o padrão *publish/subscribe* se mostra uma opção cada vez mais adotada pela indústria. Nesse padrão, um evento é publicado por um serviço e consumido por outro, sem haver uma comunicação síncrona e direta entre dois serviços. *RabbitMQ* e *AWS SNS/SQS* são ferramentas conhecidas como *Message brokers* e implementam esse padrão de projeto.

Com base nas informações acima, segue um diagrama da arquitetura da solução final utilizada:

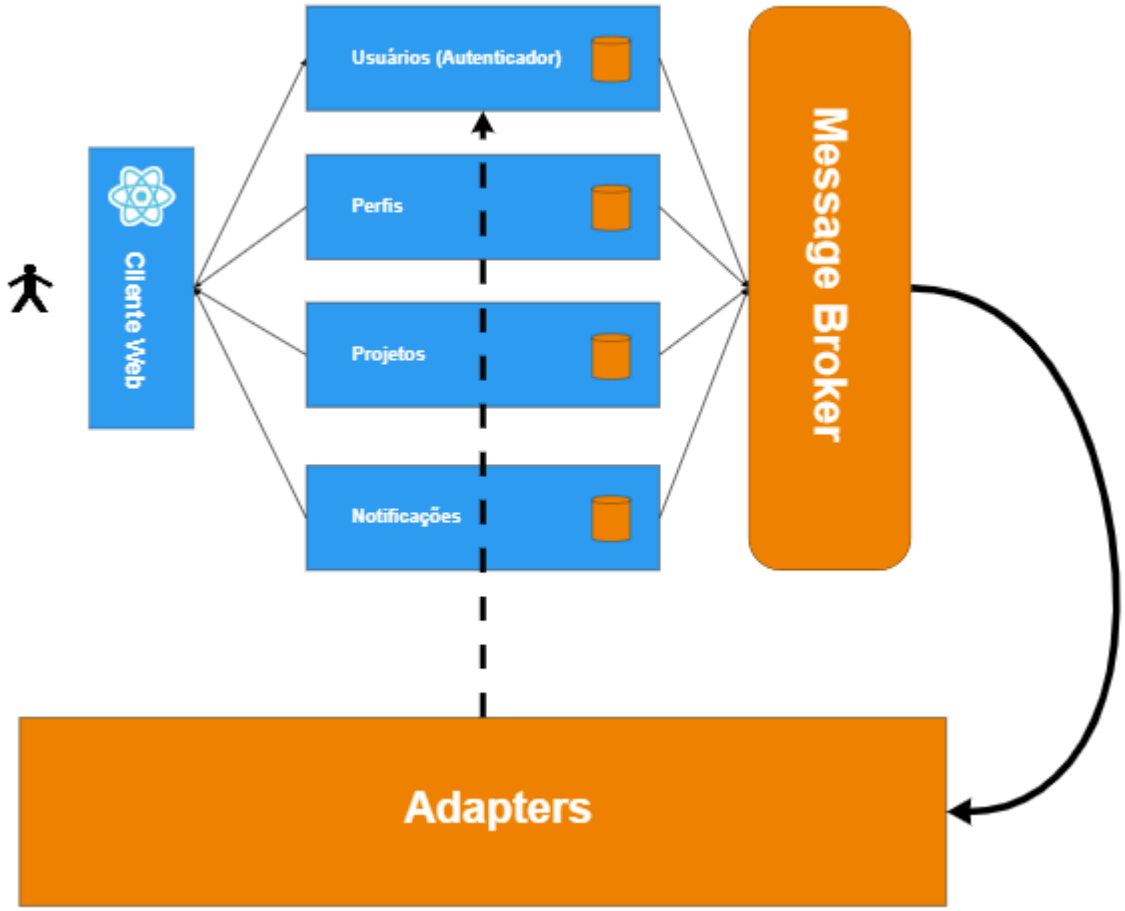


Figura 1: Arquitetura de microsserviços utilizada com comunicação assíncrona via *message broker*

A *Figura 1* mostra os microsserviços desenvolvidos no projeto, que recebem e respondem requisições de um *client web* desenvolvido em *React*. A comunicação entre os serviços é realizada a partir de um *Message Broker* cujas mensagens são processadas por *Adapters*, que são *workers* que fazem o *pooling* das mensagens e executam as ações necessárias de acordo com a mensagem.

Note que cada microsserviço tem seu próprio banco de dados, o que é uma vantagem do ponto de vista da independência entre serviços, mas podem ocorrer eventuais inconsistências entre informações correlatas nos bancos de dados. É possível citar mais algumas desvantagens na arquitetura de microsserviços:

- Dificuldade em manter consistência entre dados.
- Complexidade na comunicação entre serviços e execução de transações.
- Testes de ações que envolvem mais de um microserviço ou o escopo do projeto como um todo.

Por fim, decidimos pela arquitetura de microsserviços, utilizando as ferramentas *SNS (Simple Notification Service)* e *SQS (Simple Queue Service)* da *AWS* para implementar a comunicação assíncrona entre os serviços. Em síntese, um serviço publica uma mensagem para um tópico SNS. Filas SQS fazem inscrições nesse tópico, enfileirando as mensagens recebidas do tópico. Por sua

vez, outro microsserviço consome as mensagens da fila SQS e executa as operações referentes às mensagens recebidas. Este processo é exemplificado na Figura 2.

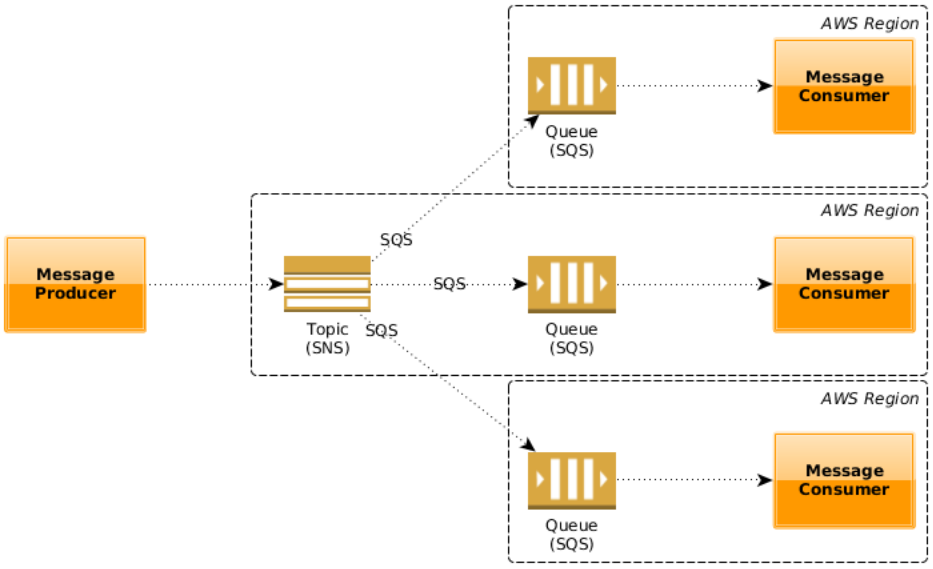


Figura 2: Exemplo de funcionamento de uma arquitetura com um tópico SNS e filas SQS

Apesar dos desafios técnicos de um sistema distribuído, julgamos que as vantagens de escalabilidade e disponibilidade de um sistema como esse são essenciais para um bom desempenho de uma rede social.

3 Desenvolvimento

O sistema de controle de versão de código utilizado foi o *GIT*. Foi criado um repositório para o front-end e um repositório para cada um dos microsserviços. O *GitHub* foi a plataforma utilizada para o armazenamento dos repositórios, além da implementação da integração e entrega contínua, a partir de *builds*, testes e *deploys* automatizados.

Quanto à hospedagem, tanto o *front-end* quanto o *back-end* foram armazenados em nuvem, a partir dos planos gratuitos disponíveis pela plataforma *Heroku*.

3.1 Back-End

Cada um dos microsserviços foi desenvolvido seguindo a estrutura de *API REST*, atendendo as requisições a partir de mensagens *HTTP* com o conteúdo principal no formato *JSON*.

O padrão de projeto *Controller-Service-Repository* foi utilizado em todo o código referente à implementação da *API REST*:

- *Controller*: Camada responsável por receber e transmitir a entrada do usuário para o(s) serviço(s).
- *Service*: Camada responsável por executar as regras de negócio a partir da entrada do usuário.
- *Repository*: Camada responsável pela interação com o banco de dados.

Ressalta-se que orientação a objetos e o conceito de injeção de dependências foram utilizados em todo o escopo do código. Em especial, o último facilitou a implementação dos testes unitários e de integração, haja vista que a injeção de dependências facilita a criação de objetos simulados, permitindo testar unidades de código de forma independente.

3.1.1 Tecnologias utilizadas

De modo geral, foi utilizada a linguagem *Python* acompanhada do framework *FastAPI* para o desenvolvimento da *API REST*. Em mais detalhes, seguem as tecnologias utilizadas:

- FastAPI: Framework em *Python* auto-documentável, que implementa o framework *ASGI (Asynchronous Server Gateway Interface)*, uma interface assíncrona para a linguagem. É um framework que promete desempenho e rápido desenvolvimento, com validação e documentação feita de maneira automática, a partir de anotações de tipo definidas pelo desenvolvedor.
- PostgreSQL: Banco de dados relacional.
- SQLAlchemy: *Object Relational Mapper (ORM)* que converte entidades e métodos do banco de dados relacional para uma estrutura orientada a objetos definida pela linguagem (*Python*). O ORM muitas vezes facilita as consultas ao banco de dados e possui maiores garantias quanto à segurança da aplicação, prevenindo ataques comuns causados por falhas no desenvolvimento de consultas.
- Alembic: Ferramenta de migração de banco de dados. Facilita realizar as alterações necessárias no banco de dados durante o desenvolvimento.
- AWS SNS e AWS SQS: Como citado no tópico anterior, o SNS é responsável por rotear as mensagens publicadas pelos microsserviços para os *subscribers*. Na nossa aplicação, os *subscribers* são as filas SQS, responsáveis por enfileirar as mensagens, esperando pelo consumo delas. Utilizando estes dois serviços, não precisamos nos preocupar com a entrega e o recebimento das mensagens.
- AWS S3: Ferramenta de armazenamento de objetos por meio de uma interface de serviço WEB. Na nossa aplicação, foi necessário armazenar imagens de perfis e de projetos, por exemplo.

3.1.2 Microsserviços

O microsserviço de autenticação é responsável pela criação, definição de *cargos* de usuário e geração de *tokens* de acesso. Como uma regra geral, cada um dos microsserviços foi desenvolvido com seu próprio sistema de autorização. A maioria das requisições foi desenvolvida com base nos valores decodificados do *token* do usuário, que contém informações, como *username* e *cargos*, por exemplo. De acordo com os cargos do usuário, cada um dos microsserviços define suas próprias regras de autorização.

3.1.2.1 Usuários (Autenticação)

O microsserviço de autenticação foi desenvolvido seguindo o protocolo de autenticação *OAuth 2*, implementando o modelo mais simples de geração de *token*, o *Password Grant Type*. Basicamente o usuário concede o nome de usuário e senha a partir dos dados de formulário, e recebe em troca

um *token* com um certo tempo de expiração. Não é a forma de autenticação mais segura, porém é tipicamente usada para conceder acesso à aplicativos internos.

Dentre as funcionalidades implementadas nesse serviço, destacam-se:

- Criação de usuário: São feitas as validações dos dados do usuário, permitindo apenas usuários com *username* único e com e-mail da UNICAMP. Além disso, as senhas são armazenadas de maneira criptografada a partir do algoritmo *Bcrypt*, ou seja, não é possível recuperar a senha, somente o *hash* dela. Ao fim, é enviado um e-mail de confirmação para o usuário.
- Email de confirmação: Ao clicar no link de confirmação, o usuário é marcado como "Email Verificado"no banco de dados, e a partir desse momento, ele pode realizar login.
- Login: Ao realizar o login, a partir de seu usuário e senha, é concedido um *token* ao usuário, válido para todos os microsserviços, que compartilham da mesma *secret key*. Esse *token* tem um tempo de expiração de uma hora. O *token* também armazena informações úteis, tais como e-mail, nome, nome de usuário e GUIDs (Identificadores de 128 bits compartilhados entre os serviços).

3.1.2.2 Perfis

O microsserviço de perfis é responsável pelo gerenciamento e customização de perfis de usuários. Nele é possível vincular imagens de perfil, informações de contato, interesses e cursos que os usuários frequentam no momento. Note que usuário e perfil são entidades presentes em bancos distintos. Dessa maneira, toda vez que um usuário é criado, o microsserviço de autenticação envia uma mensagem, que posteriormente é consumida, e um perfil é criado para o usuário em questão. A troca de mensagens entre serviços será melhor abordada em um próximo tópico.

Dentre as *features* implementadas nesse serviço, destacam-se:

- Criação de perfil: É uma rota especial, requisitada pelo *adapter*, que é um *runner* que faz o *pooling* das mensagens da fila. Só é possível consumir essa rota a partir de um *token* com cargo administrativo.
- CRUDs do **próprio perfil**: Foram implementadas rotas para atualizar o próprio perfil, além da adição de vínculos de interesses, cursos, informações de contato, imagens de perfil, etc. Essas rotas apenas dizem respeito ao usuário referente ao *token* enviado na requisição. Não é possível acessar essas rotas sem o envio de um *token* válido.
- Imagem de perfil: Como supracitado, é possível adicionar uma imagem na rota de atualizar o perfil. A imagem, enviada no corpo da requisição no formato *base64* é armazenada em um *bucket* S3 da AWS.
- Listagem de perfis: Foi implementada uma rota de listagem de perfis, com paginação via *cursor* e filtros. Em especial, essa é uma rota importante para visualizar perfis de interesse, compatíveis com o projeto de algum usuário, por exemplo.

3.1.2.3 Projetos

O microsserviço de projetos é um componente fundamental da plataforma. Nele, são definidas as regras de negócio referentes aos projetos, e seus vínculos com os usuários. Em especial, o *match* bilateral entre usuários e projetos, é implementado nesse microsserviço.

Dentre as *features* implementadas nesse serviço, destacam-se:

- Criação de projeto: Rota de criação de projeto por um usuário. Ele se torna *owner* do projeto, o que o garante permissões especiais, além de ser notificado sobre quaisquer atualizações.
- CRUDs de projeto em geral.
- Vínculos de cursos e interesses relacionados em um projeto.
- Imagem de projeto: É possível adicionar uma imagem na rota de atualizar o projeto. A imagem, enviada no corpo da requisição no formato *base64* é armazenada em um *bucket* S3 da AWS.
- Match bilateral: Rota que implementa uma das *features* principais da plataforma. Nessa rota, é criada ou atualizada uma entidade no banco de dados, a qual armazena a relação entre um usuário e um projeto, no que diz respeito ao interesse bilateral. Isso é, é definido se há interesse de usuário pelo projeto e/ou interesse do projeto pelo usuário. Essa rota também tem um papel fundamental na comunicação assíncrona entre os serviços, porém isso será abordado em um próximo tópico.
- Listagem de projetos: Foi implementada uma rota de listagem de projetos, com paginação via *cursor* e filtros. Em especial, essa é uma rota importante para visualizar projetos de interesse, compatíveis com usuários da plataforma.

3.1.2.4 Notificações

O microsserviço de notificações, por sua vez, é responsável pela exibição das notificações do usuário na plataforma.

Dentre as *features* implementadas nesse serviço, destacam-se:

- Exibição de notificações: Rota que retorna as notificações do usuário a partir de seu *token*. É possível selecionar apenas notificações lidas ou não-lidas.
- Marcar noticações como lidas: Rota que marca notificações como lidas, em lote.

3.1.2.5 Troca de Mensagens

Como supracitado, a comunicação entre os microsserviços se deu de maneira assíncrona, a partir das ferramentas *SNS* e *SQS* da *AWS*.

Retomando, os microsserviços enviam mensagens para um tópico *SNS*, que são roteadas para os *subscribers* do tópico. Na plataforma desenvolvida, os *subscribers* são filas *SQS*, responsáveis por enfileirar as mensagens, e torná-las disponíveis para consumo.

Na plataforma, as filas são consumidas a partir de um *worker*, que faz o *pooling* das mensagens das filas, e executa as regras de negócio com base nelas. Esse *worker*, nomeado como *adapter*, realiza as operações necessárias em cada um dos microsserviços relacionados a partir de comunicação síncrona via *HTTP*.

Dentre os tipos de mensagem definidas na plataforma, destacam-se:

- Criação de perfil para novo usuário: Como já citado, como perfil e usuário são entidades distintas, foi necessário abordar esse problema. Portanto, quando um usuário é criado, um perfil é criado para o usuário de forma automática.

- Notificação de interesse de usuário em um projeto: Quando um usuário manifesta interesse em um projeto, é gerada uma notificação para todos os *owners* do projeto em questão.
- Notificação de interesse de projeto em usuário: Quando um projeto manifesta interesse em um usuário, é gerada uma notificação para o usuário em questão.
- Notificação de ***Match***: Quando ocorre interesse bilateral entre usuário e projeto, são geradas notificações para os *owners* do projeto, assim como para o usuário.

Outros tipos de mensagem também foram implementados, porém dizem respeito à aspectos mais técnicos, como o compartilhamento de informações que podem ser úteis. Por exemplo, parte da tabela de usuário é compartilhada com o microsserviço de perfis.

3.2 Front-End

O front-end da plataforma foi implementado como uma aplicação web, que consome a API implementada pelo back-end, de acordo com as ações do usuário. A aplicação foi criada como um React App.

3.2.1 Tecnologias utilizadas

- HTML/CSS/JS: Base de toda aplicação web. Foram utilizados em conjunto com as bibliotecas.
- React: Biblioteca JavaScript para criação de interfaces de usuário, criada e mantida pelo Facebook.
- JSX (JavaScript XML): Extensão de sintaxe para o JavaScript, facilitando a criação dos componentes React.
- Material-UI: Biblioteca de componentes React, seguindo o padrão de design criado pelo Google (material design).
- Axios: Biblioteca JavaScript que foi utilizada para a comunicação assíncrona com a API do back-end.

4 Resultados e Discussão

Nessa seção serão abordados os resultados do desenvolvimento da aplicação, isto é, o resultado final das telas e seus recursos principais.

4.1 Telas de Login/Cadastro

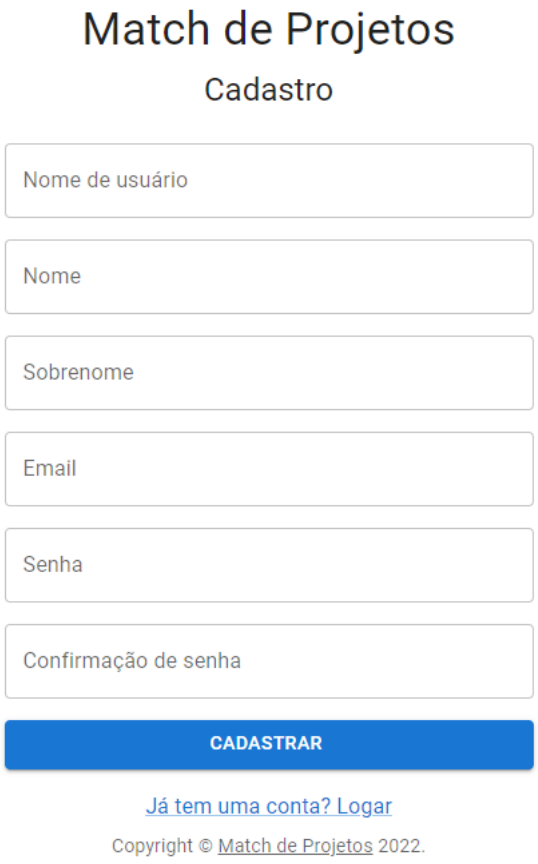


Figura 3: Tela de cadastro

Figura 4: Confirmação de e-mail

Figura 5: Tela de login

A Figura 3 exibe a tela de cadastro de usuários. Além das regras de negócio impostas pelo back-end, o front-end induz o usuário a criar senhas mais complexas, com um mínimo de número de caracteres, dígitos, minúsculo/maísculo e caracteres especiais.

Com o sucesso da criação da conta do usuário, um e-mail é enviado ao usuário, como é demonstrado na Figura 4.

Confirmado o e-mail do usuário, é possível realizar o login. Como supracitado, a rota de login retorna um token ao usuário, com um tempo de expiração de cerca de uma hora. Esse token é armazenado pelo front-end e é utilizado até que o back-end retorne um erro de autorização. Quando isso ocorre, o front-end redireciona o usuário à tela de login novamente, para que ele solicite um novo token.

4.2 Tela de perfil do usuário atual

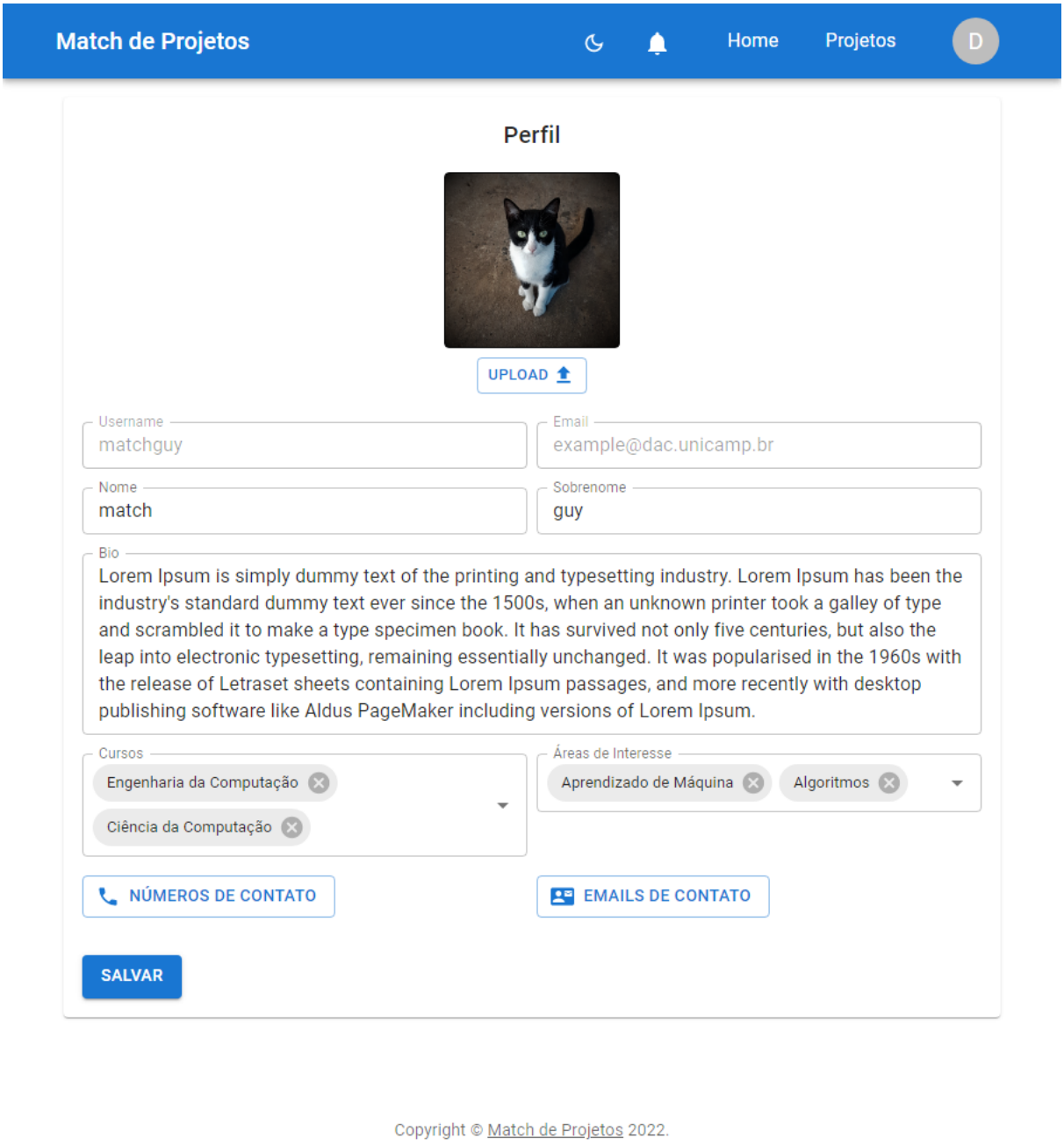
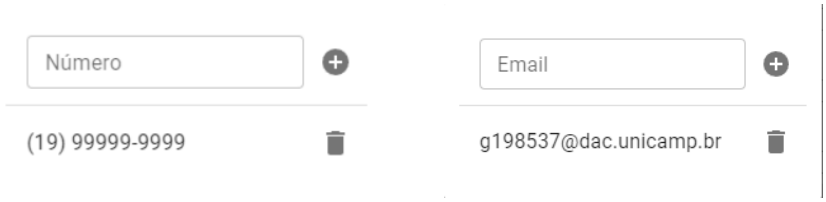


Figura 6: Tela de perfil do usuário



(a) Inserção de telefones de contato (b) Inserção de e-mails de contato

Figura 7: Inserção de informações de contato

Como exibido pela Figura 6 e Figura 7, o usuário é capaz de vincular uma série de informações ao seu perfil:

- Nome e "Sobre Mim".
- Cursos e áreas de interesse: importante para encontrar usuários com interesses parecidos.
- Informações de contato.
- Imagem de perfil.

4.3 Telas de projetos

A seção de projetos é composta de 3 partes principais:

- Meus projetos: projetos em que o usuário atual participa.
- Tenho interesse: projetos em que o usuário atual manifestou interesse.
- Criar projeto: tela de criação de projeto.

4.3.1 Meus Projetos

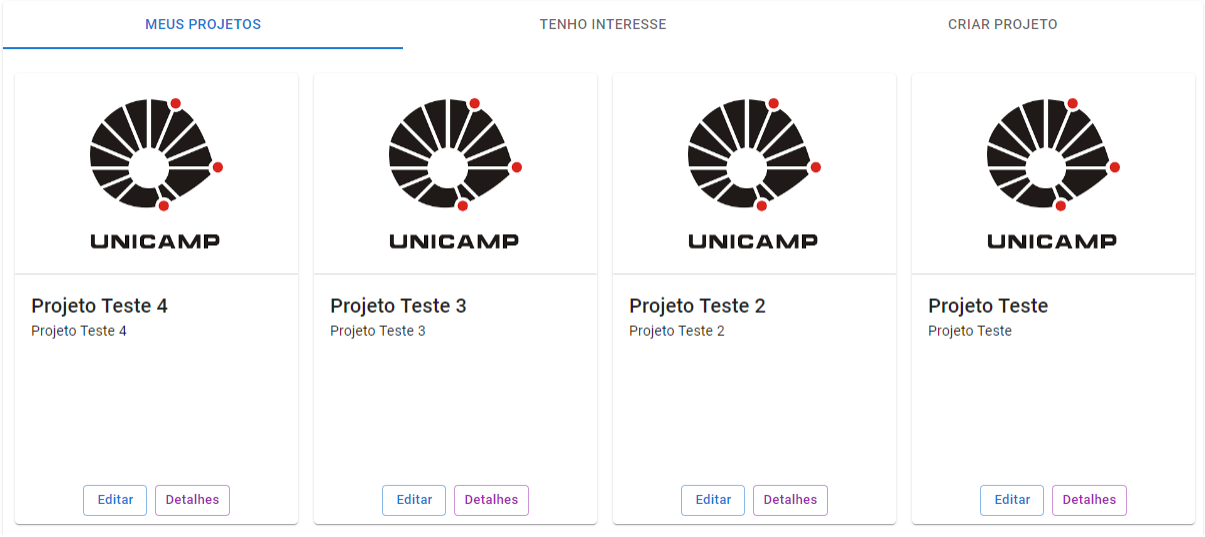


Figura 8: Projetos em que o usuário atual participa

Nessa tela, o usuário é capaz de visualizar os projetos em que participa atualmente. É possível visualizar os detalhes dos projetos, assim como editá-los.

4.3.2 Projetos de interesse

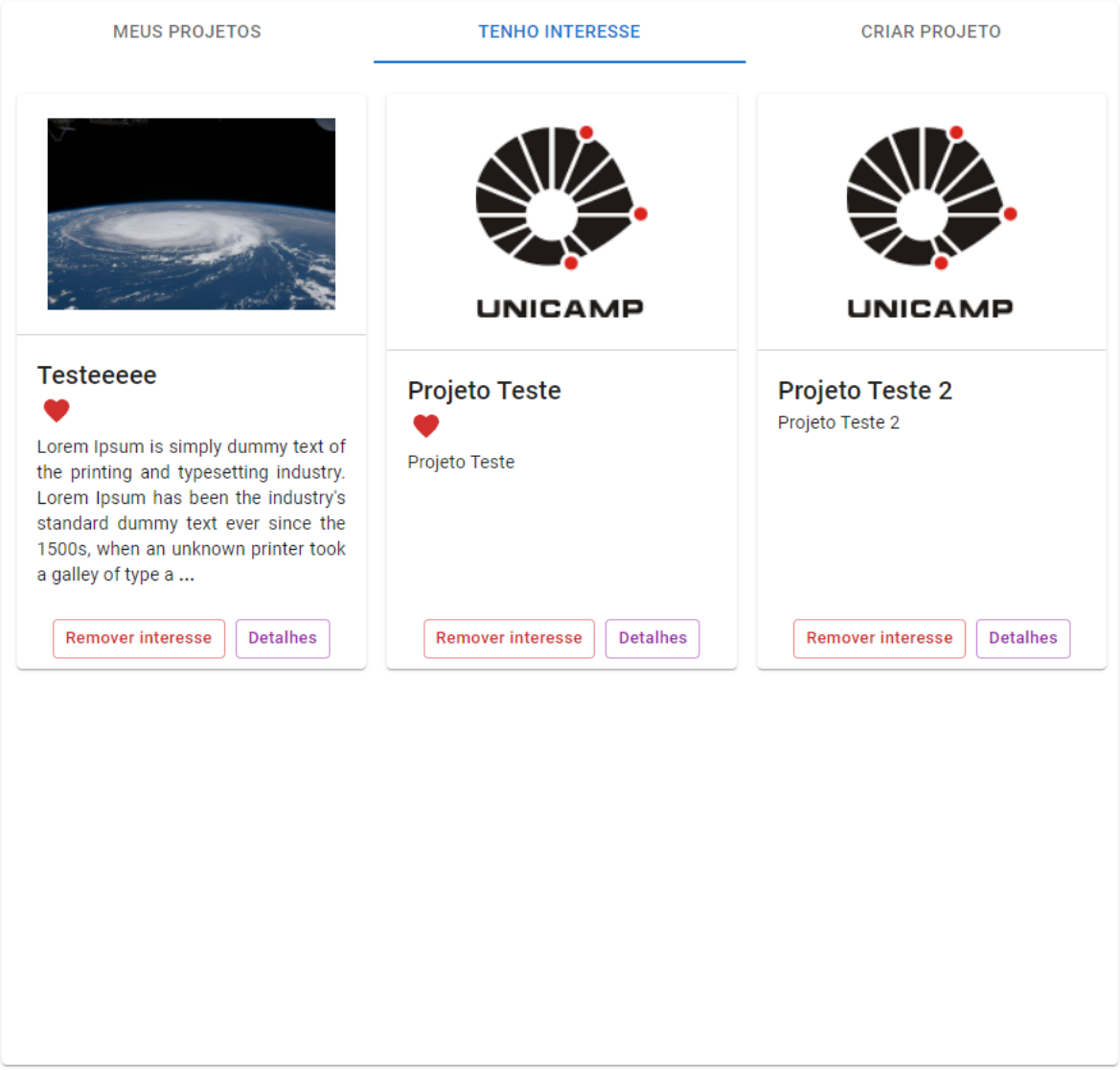


Figura 9: Projetos em que o usuário atual manifestou interesse

Nessa tela, o usuário é capaz de visualizar os projetos em que manifestou interesse. É possível visualizar os detalhes dos projetos, assim como remover o interesse do projeto. Note também, como exibe a Figura 9, que é possível verificar se o usuário tem um match com o projeto, a partir do ícone em formato de coração. A implementação das telas do match será discutida adiante.

4.3.3 Criação de projeto

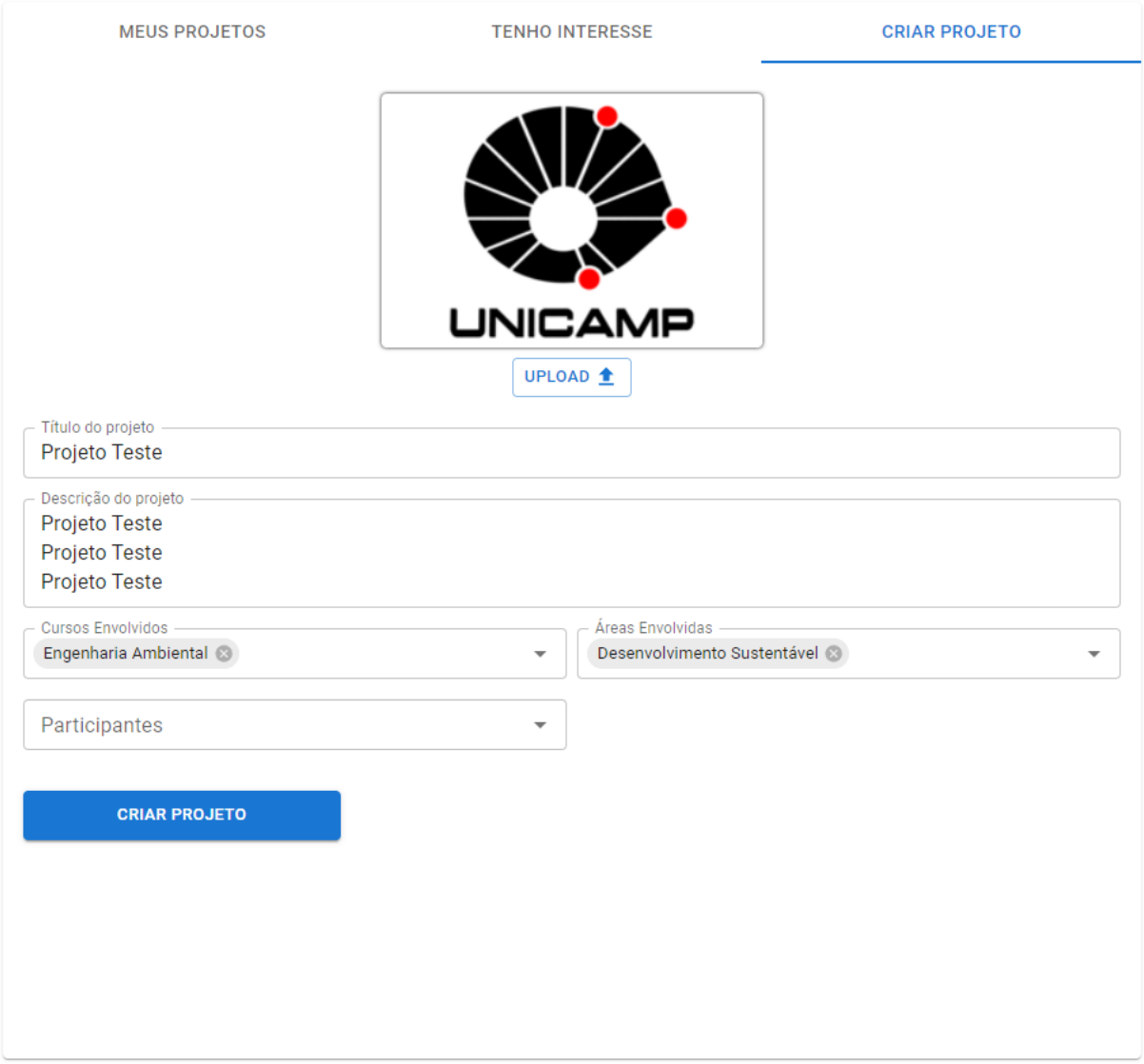


Figura 10: Projetos em que o usuário atual manifestou interesse

Como exibido pela Figura 10, o usuário é capaz de vincular uma série de informações ao seu projeto:

- Título e descrição.
- Cursos e áreas de interesse: importante para encontrar usuários com interesses parecidos.
- Usuários participantes.

4.4 Tela Home

É a tela inicial da plataforma, assim como uma das mais importantes. É nela que o usuário é exposto à algumas das principais funcionalidades da plataforma:

- Vitrine de projetos.
- Vitrine de perfis.
- Match de projetos.

4.4.1 Vitrine de projetos

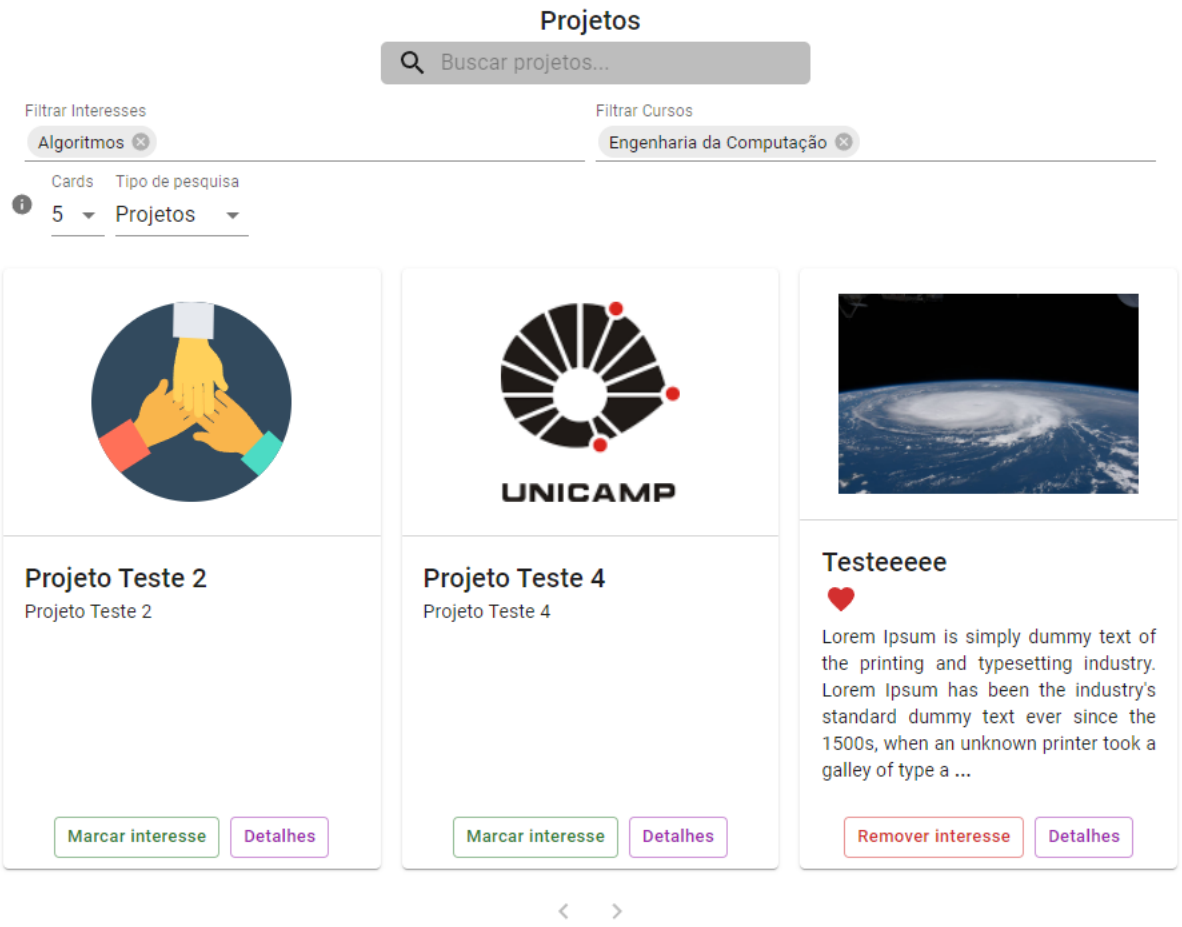


Figura 11: Vitrine de projetos

Como é possível se visualizar a Figura 11, a vitrine de projetos foi implementada, com funcionalidades como:

- Filtros por interesses em comum: é possível filtrar projetos por interesses relacionados.
- Filtros por cursos relacionados: é possível filtrar projetos por cursos relacionados.
- Paginação: é possível escolher a quantidade de cards por página, e alterar a página a partir das setas no canto inferior.

Além disso, cada um dos cards é munido da imagem principal do projeto, seu título e descrição e uma das principais funcionalidades da plataforma, a implementação do match bilateral. Através

do card, é possível adicionar ou remover intereresse de um projeto, o que como já explicado, gera uma notificação para os donos do projeto, que podem também demonstrar interesse nesse usuário, como será exibido adiante. Também é possível visualizar maiores detalhes do projeto.

Note também, que se há match entre um usuário e projeto, um ícone de coração é exibido no card. Dessa forma, o usuário é capaz de identificar que o projeto também manifestou interesse nele.

4.4.2 Vitrine de perfis

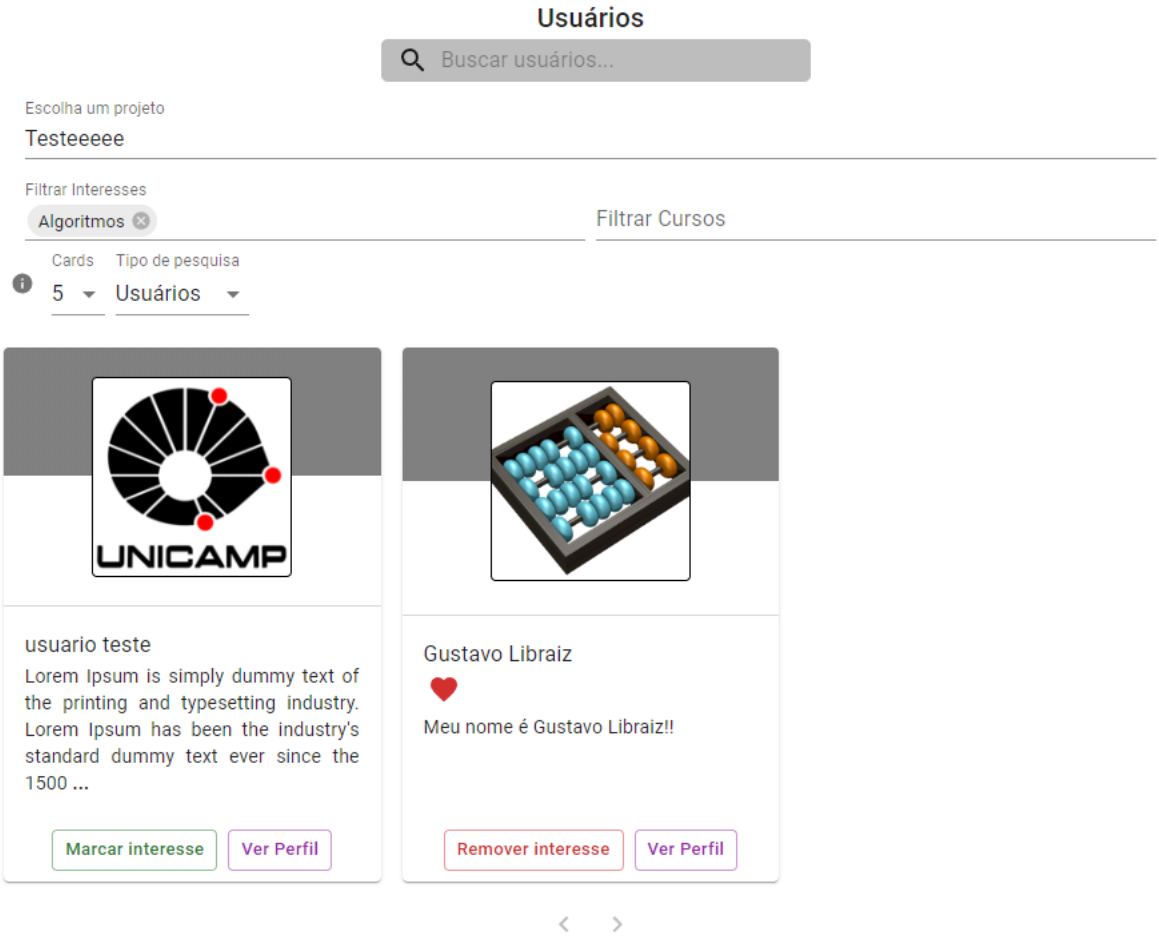


Figura 12: Vitrine de perfis

Como é possível se visualizar a Figura 12, a vitrine de perfis foi implementada, com funcionalidades como:

- Filtros por interesses em comum: é possível filtrar perfis por interesses relacionados.
- Filtros por cursos relacionados: é possível filtrar perfis por cursos relacionados.
- Paginação: é possível escolher a quantidade de cards por página, e alterar a página a partir das setas no canto inferior.

Além disso, cada um dos cards é munido da imagem de perfil, o nome do usuário, além de uma descrição básico do seu perfil. Assim como na vitrine de projetos, o match bilateral é implementado, porém de uma maneira um pouco diferente. Através da caixa de seleção "Escolha um projeto", o usuário atual é capaz de selecionar um projeto em que é owner. Selecionado o projeto, o usuário atual, que também é owner do projeto, pode adicionar ou remover interesse do projeto selecionado em relação ao usuário do card. Como já explicado, isso gera uma notificação ao usuário em questão, que também pode demonstrar interesse ao usuário. Também é possível visualizar maiores detalhes dos perfis do usuário.

Note também, que se há match entre um usuário e projeto, um ícone de coração é exibido no card. Dessa forma, o dono do projeto é capaz de identificar se o usuário também manifestou interesse no projeto.

4.5 Notificações

O front-end realiza um pooling, requisitando novas notificações dada uma certa quantia de tempo.

O usuário é capaz de visualizar suas notificações através do ícone de sino, como é exibido no header da página. O número de notificações não-lidas é exibido, como mostra a figura:

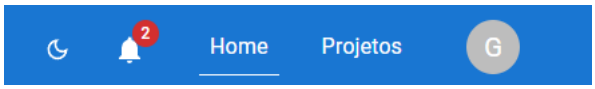
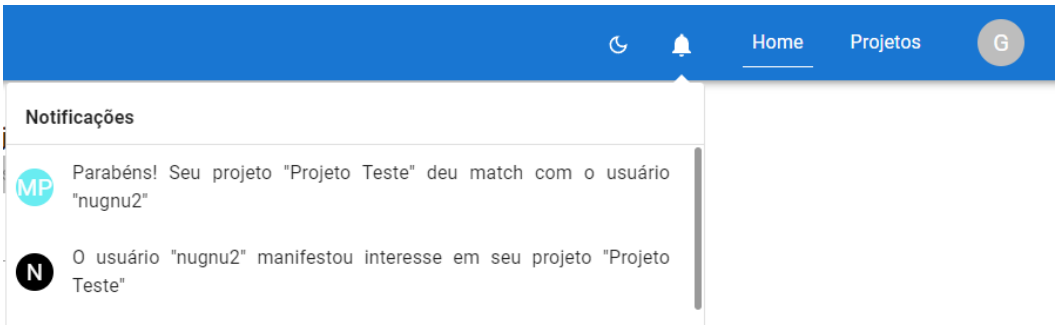
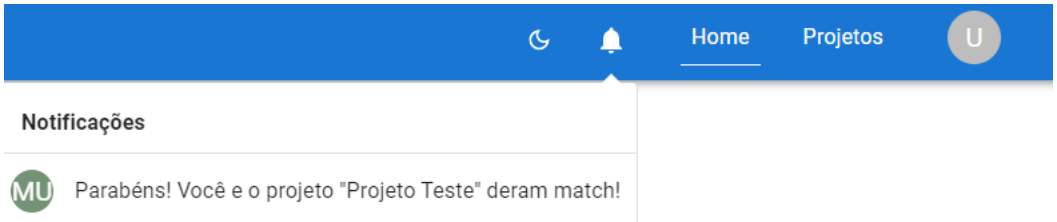


Figura 13: Header da página com notificações não-lidas

Ao clicar no sino, as notificações do usuário são exibidas em uma lista, como a seguir:



(a) Notificações do owner do projeto



(b) Notificações do usuário que manifestou interesse

Figura 14: Fluxo de notificação de usuário que manifestou interesse em um projeto

Note que as notificações mais recentes são exibidas primeiro. Além disso, as notificações da figura 14 demonstram um fluxo completo de match bilateral entre usuário e projeto iniciada pelo usuário. É possível também clicar em cada uma das notificações, o que expande um modal com interações úteis ao fluxo de match.

4.5.1 Notificação de interesse de usuário em um projeto

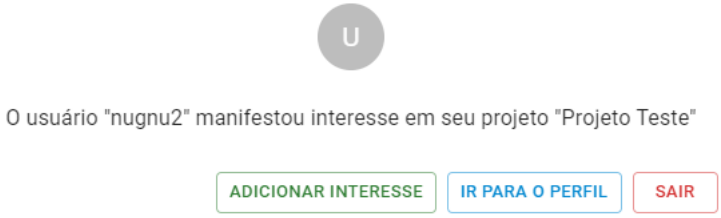


Figura 15: Exemplo de modal de notificação de interesse de usuário em um projeto

Ao clicar em uma notificação desse tipo, um modal é expandido, contendo funcionalidades úteis para a plataforma, como exibido na Figura 15. É possível visitar o perfil do usuário, assim como adicionar ou remover interesse do projeto por esse usuário, facilitando que seja feito ou desfeito o match com o usuário em questão.

4.5.2 Notificação de interesse de projeto em um usuário

Figura 16: Exemplo de modal de notificação de interesse de projeto em um usuário

Ao clicar em uma notificação desse tipo, um modal é expandido, contendo funcionalidades úteis para a plataforma, como exibido na Figura 16. É possível visitar o projeto que manifestou interesse pelo usuário, assim como adicionar ou remover interesse do usário pelo projeto, facilitando que seja feito ou desfeito o match com o projeto em questão.

4.5.3 Notificação de match

Figura 17: Exemplo de modal de notificação de match

São notificações mais simples. Dado o match, os donos do projetos recebem uma notificação, que se expandida, exibe um modal com a opção de visualizar o perfil do usuário em que se foi manifestado o interesse. Já o usuário em questão, recebe uma notificação, que se expandida, exibe um modal com opção de visualizar o projeto.

5 Considerações Finais

Durante o desenvolvimento da plataforma, diversas frentes da construção de um *software* foram exploradas. Foram levantados requisitos e regras de negócio, foi definida uma arquitetura e diversas ferramentas técnicas foram utilizadas. A construção da arquitetura de microsserviços, implementada com comunicação assíncrona através de mensagens, foi um desafio, porém recompensadora, através dos resultados e da escalabilidade obtida. Ao fim, foi possível construir uma plataforma consistente e com funcionalidades que executam a maioria dos requisitos levantados inicialmente. Além disso, o código foi armazenado com suporte a *CI/CD*, com testes unitários e de integração e implantação contínua, de forma que será possível mantê-lo e expandí-lo para desenvolvimento futuro.

No que diz respeito ao trabalho futuro, é possível notar alguns pontos de melhoria em relação à plataforma final entregue:

- Melhoria no conteúdo dos projetos: No estado atual, as informações contempladas em um projeto na plataforma são mínimas, e podem não ser suficientes para que o projeto de uma equipe seja mantido na plataforma. Em suma, a plataforma é útil para buscar novos candidatos para projetos, porém é muito provável que donos de projetos utilizem outras ferramentas para explorar conteúdos não existentes atualmente na plataforma.
- Algoritmos de recomendação: Atualmente na plataforma, é possível filtrar projetos e perfis por cursos relacionados e interesses, porém não foi implementado um algoritmo de relevância, capaz de sugerir projetos a partir do perfil do usuário.

Em síntese, através das técnicas de engenharia de *software* exploradas durante o desenvolvimento da plataforma, foi construída uma ferramenta consistente que cumpre a maioria dos requisitos levantados e que apresenta ganhos em escalabilidade e disponibilidade através da arquitetura em microsserviços, cujo foco é o desacoplamento entre unidades.

Referências

[1] A. V. Aho, J. E. Hopcroft and J. D. Ullman, *The Design and Analysis of Computer Algorithms,* Addison-Wesley (1901).

[2] D. E. Knuth and L. Lamport, *A structural analysis of the role of gnus and gnats in the post-modernistic, crypto-existential Weltanschauung of neo-liberal Tibeto-Vietnamese leaf blower operators as manifest in the sexual symbology of the Los Angeles Phone Directory.* Journal of Gnu Technology, **23** (6), 12–87 (March 1996).

[3] Hasselbring and G. Steinacker, "Microservice architectures for scalability, agility and reliability in e-commerce," in IEEE ICSA Workshops, 2017, pp. 243–246.