Projeto Final – Biorama

Aplicação Web

Formandos:

João Correia

Lucas Silvestre

Vladimiro Bonaparte

Formadores:

Gonçalo Feliciano

Vitor Custódio

Curso Técnico Especializado em Programação e Sistemas de Informação

14 de fevereiro de 2025

Índice

1 Introdução

O projeto Biorama foi desenvolvido como parte do projeto final do curso Técnico Especializado em Sistemas de Informação, com o objetivo de criar uma plataforma web inovadora para a comercialização e divulgação de produtos sustentáveis e de origem local.

A plataforma visa apoiar pequenos agricultores e comerciantes regionais, permitindo-lhes vender diretamente aos consumidores e promover a economia local sem necessidade de investimentos elevados.

Desde o início, optámos por uma abordagem personalizada, construindo a aplicação de raiz, sem templates pré-concebidos. Utilizámos Laravel para o backend e React com Inertia.js para o frontend, sendo que esta última tecnologia não foi abordada no curso, exigindo aprendizagem autónoma e uma adaptação contínua.

A plataforma oferece uma experiência de utilizador completa e segura, com funcionalidades como autenticação, gestão de perfis, histórico de encomendas, moradas de entrega e carrinho de compras otimizado. Para reforçar a credibilidade do mercado, foi implementado um sistema de avaliações e comentários diretamente à loja onde foi realizada a encomenda. Além disso, a geolocalização e pesquisa avançada facilitam a descoberta de lojas e produtos próximos.

No âmbito da gestão comercial, a plataforma permite um controlo eficiente de vendedores, lojas e produtos, suportando múltiplos métodos de pagamento, emissão de faturas com NIF.

O desenvolvimento do projeto exigiu gestão de tarefas eficiente, seguindo a metodologia Scrum para definir prioridades e adaptar estratégias. O uso de Git e a realização de testes contínuos garantiram um código estável e funcional.

Este relatório documenta todas as fases do projeto, desde a modelação da base de dados e implementação das funcionalidades até aos desafios enfrentados e melhorias futuras.

O Biorama consolidou os conhecimentos adquiridos no curso, reforçando a nossa autonomia na aprendizagem de novas tecnologias e a capacidade de desenvolver soluções inovadoras. Este projeto foi um marco no nosso percurso formativo, preparando-nos para os desafios do mercado de trabalho.

2 Planeamento

O planeamento é uma das etapas mais importantes no desenvolvimento de um projeto de software, pois define a estrutura base e orienta todas as fases subsequentes. Um bom planeamento permite antecipar riscos, otimizar recursos e garantir que os prazos sejam cumpridos, assegurando um desenvolvimento eficiente e alinhado com os objetivos do projeto.

Neste projeto, o planeamento envolveu uma abordagem abrangente, estruturada com base na definição da base de dados, na arquitetura do *backend* e *frontend*, na validação de dados e segurança e na integração com sistemas externos.

Para garantir uma implementação robusta e eficiente das funcionalidades, realizámos um planeamento detalhado que contemplou:

- Definição e estruturação da base de dados para suportar todas as operações essenciais da aplicação, assegurando integridade e eficiência no armazenamento e recuperação de dados.
- Planeamento do *backend*, estabelecendo a forma como os dados seriam processados, armazenados e disponibilizados através de *APIs*.
- Desenvolvimento do *frontend*, onde definimos a melhor abordagem para estruturar a interface e proporcionar uma experiência de utilização fluida e intuitiva.
- Validação de dados em dois níveis:
    - *Frontend* – Implementação de mecanismos de validação de formulários para garantir que os dados inseridos pelos utilizadores cumpriam os requisitos definidos.
    - *Backend* – Reforço da segurança com validações adicionais no servidor, garantindo que os dados recebidos estavam corretos e protegidos contra tentativas de inserção indevida.
- Definição de prioridades e fluxo lógico de desenvolvimento, estruturando o trabalho por fases e garantindo que cada funcionalidade fosse desenvolvida de forma interdependente e otimizada.
- Planeamento da interligação com *APIs* externas e tecnologias complementares, permitindo enriquecer a aplicação com funcionalidades adicionais e assegurar uma integração fluida com outros serviços.

Este planeamento rigoroso permitiu-nos antecipar desafios, distribuir tarefas de forma eficiente e garantir que cada componente do sistema funcionasse de forma coesa, resultando numa aplicação bem estruturada, segura e escalável.

2.1 Metodologia de Desenvolvimento

Para o desenvolvimento da aplicação, foi adotada a metodologia *Scrum*, uma *framework* ágil que permite a entrega contínua de funcionalidades através de ciclos iterativos chamados sprints.

Esta abordagem foi escolhida devido à sua flexibilidade e capacidade de adaptação a eventuais mudanças nos requisitos do projeto, garantindo um desenvolvimento estruturado e eficiente.

Enquanto grupo, seguimos uma abordagem colaborativa, onde cada membro assumiu responsabilidades específicas dentro de cada sprint, assegurando um fluxo de trabalho contínuo e bem organizado.

O desenvolvimento da aplicação foi dividido nas seguintes fases, garantindo a integração harmoniosa entre *frontend* e *backend*.

- **Planeamento e Estruturação da Base de Dados**

Para suportar toda a lógica da aplicação, iniciámos o desenvolvimento com a definição do modelo relacional, assegurando que todas as tabelas e relações estivessem corretamente estruturadas.

Utilizámos *migrations* para a criação da base de dados, permitindo um controlo eficaz da sua evolução ao longo do desenvolvimento.

- **Desenvolvimento do Backend**

O *backend* foi desenvolvido com *Laravel*, garantindo uma arquitetura robusta e modular. Durante esta fase, implementámos:

  - Controladores e modelos, permitindo a interação entre a base de dados e o *frontend* através de *endpoints* bem definidos.
  - *Migrations*, *seeders* e *factories*, assegurando a criação estruturada de tabelas e a inserção de dados essenciais para os testes e validações.
  - Camada de segurança e validação, garantindo que os dados recebidos fossem verificados corretamente antes de serem processados.

- **Desenvolvimento do Frontend**

A interface da aplicação foi construída com React + Inertia.js, utilizando Vite para otimizar a compilação e garantir uma experiência fluida e responsiva. O desenvolvimento do *frontend* seguiu as seguintes diretrizes:

- Criação de um layout em componentes, garantindo a reutilização eficiente de elementos da interface.
- Gestão de estados e interação com o *backend*, assegurando que os dados fossem apresentados e atualizados de forma dinâmica e eficiente.
- Validação de formulários no *frontend*, reduzindo a possibilidade de erros antes de submeter os dados ao *backend*.

- **Implementação das Funcionalidades Principais**

Seguindo a metodologia *Scrum*, planeámos e implementámos cada funcionalidade de forma progressiva, garantindo que todas as partes do sistema estivessem interligadas corretamente. Durante este processo, realizámos revisões regulares para validar a usabilidade e eficiência da aplicação.

- **Testes e Otimizações**

Para garantir um sistema estável e funcional, adotámos uma abordagem iterativa de testes contínuos durante cada sprint. Foram realizadas:

- Testes no *backend*, validando a integridade dos dados e a segurança das APIs.
- Testes no *frontend*, assegurando que a experiência do utilizador fosse intuitiva e fluida.
- Correção de erros e otimizações, refinando a aplicação antes da entrega final.

Através desta metodologia, conseguimos desenvolver um produto robusto e bem estruturado, garantindo que cada fase do desenvolvimento estivesse alinhada com os objetivos do projeto e as necessidades dos utilizadores.

2.2 Cronograma

O cronograma do projeto foi estruturado em fases com prazos bem delineados, garantindo um fluxo contínuo de desenvolvimento e permitindo a entrega de um produto funcional e otimizado no prazo estipulado.

O grupo decidiu iniciar o planeamento e organização do desenvolvimento com antecedência, devido à elevada complexidade e ambição do projeto.

Essa abordagem permitiu uma melhor distribuição das tarefas, minimizando riscos e garantindo um desenvolvimento mais controlado e eficiente. Assim, o trabalho foi distribuído ao longo de 12 semanas, organizando os sprints da seguinte forma:

2.2.1 Sprint 1 - Autenticação - Semana 1-2

A autenticação foi implementada utilizando a autenticação intrínseca do Laravel, garantindo um sistema seguro e eficiente.

O registo de utilizadores exige uma verificação por email, utilizando Laravel Mail para o envio de um *link* de confirmação.

Para garantir a persistência da sessão, utilizámos os mecanismos nativos do Laravel, assegurando que o utilizador permanece autenticado mesmo após um *refresh* da página.

Além disso, foi incorporada a funcionalidade de recuperação de password, permitindo aos utilizadores redefinirem as suas credenciais através de um email de redefinição.

A implementação do *logout* assegura que todas as sessões ativas do utilizador são encerradas de forma segura.

2.2.2 Sprint 2 – Gestão de perfil do utilizador - Semana 3

Para garantir uma experiência de utilização intuitiva e eficiente, planeámos a navegação desta secção através de uma *sidebar*, permitindo ao utilizador alternar facilmente entre as diferentes áreas do seu perfil.

Dividimos as funcionalidades em categorias distintas para facilitar a navegação, proporcionando uma estrutura clara e organizada.

A abordagem ao desenvolvimento seguiu um fluxo lógico, iniciando-se com o design da interface e experiência do utilizador, assegurando que cada ação, como editar dados, visualizar encomendas ou gerir moradas, fosse acessível e intuitiva. Posteriormente, avançámos para a implementação técnica de cada funcionalidade, assegurando que o utilizador pudesse editar, visualizar e eliminar informação de forma eficiente.

Além disso, integrámos a aba de notificações, permitindo que os utilizadores recebessem alertas sobre alterações no estado das suas encomendas ou novas encomendas realizadas.

2.2.3 Sprint 3 – Registo de Perfil de vendedores - Semana 4 e 5

O registo de vendedor foi estruturado em três fases distintas para facilitar o processo e garantir uma experiência fluida ao utilizador:

- **1ª fase:** Recolha dos dados pessoais do utilizador ou, caso aplicável, os dados da sua empresa, garantindo a validação necessária para prosseguir com a criação da conta de vendedor.
- **2ª fase:** Criação da primeira loja do vendedor, onde são introduzidas informações essenciais como nome, descrição, localização e detalhes de contacto.
- **3ª fase:** Possibilidade de adicionar os primeiros produtos à loja recém-criada, permitindo que o vendedor comece a operar de imediato na plataforma.

Esta abordagem estruturada garantiu que o fluxo de *onboarding* dos vendedores fosse intuitivo, evitando sobrecarga de informações e permitindo uma transição suave para a gestão da loja dentro da aplicação.

2.2.4 Sprint 4 – Área de gestão do vendedor - Semana 6 até Semana 11

A Área de Gestão do Vendedor foi planeada para fornecer uma visão completa dos negócios do utilizador na plataforma. Devido à sua complexidade e ao número de funcionalidades envolvidas, esta *feature* foi desenvolvida ao longo de várias semanas, sendo dividida em **sprints** para garantir um desenvolvimento organizado e eficiente. A abordagem seguiu uma lógica progressiva, assegurando que cada funcionalidade fosse implementada de forma sólida antes de avançar para a seguinte.

Planeamento dos Sprints da Área de Gestão do Vendedor

- **Sprint 1 (Semana 6-7): Perfil do Vendedor**

O primeiro passo no desenvolvimento desta funcionalidade foi garantir que o vendedor pudesse visualizar e editar as suas informações. Assim, começámos por implementar:

Exibição dos dados do vendedor, incluindo informações pessoais e, se aplicável, dados da empresa.

Edição das informações do vendedor, permitindo atualizar os seus dados facilmente.

- **Sprint 2 (Semana 7-8): Gestão de Lojas**

Após definir a estrutura do perfil do vendedor, avançámos para a funcionalidade de gestão de lojas, onde implementámos:

  - Listagem de lojas criadas pelo vendedor.
  - Possibilidade de criar uma loja (respeitando o limite de 3 lojas por vendedor).
  - Edição e remoção de lojas.

- **Sprint 3 (Semana 8-9): Gestão de Produtos**

Com a gestão das lojas estabelecida, passámos ao desenvolvimento das funcionalidades relacionadas com os produtos dentro de cada loja:

  - Visualização detalhada de cada loja específica.
  - Listagem dos produtos disponíveis em cada loja.
  - Adição de novos produtos com imagens, descrição, preço da unidade de produto.
  - Edição e remoção de produtos existentes.
  - Implementação de um sistema de pesquisa para facilitar a busca por produtos dentro da loja.
  - Exibição de *reviews* associadas aos produtos vendidos na loja.

- **Sprint 4 (Semana 9-10): Gestão de Encomendas**

Após a implementação da gestão de produtos, o próximo foco foi a funcionalidade de encomendas. Para isso, desenvolvemos:

  - Listagem de encomendas organizadas por loja ou exibição global de todas as encomendas do vendedor.
  - Visualização dos detalhes das encomendas, incluindo cliente, produtos adquiridos e estado da encomenda.
  - Possibilidade de atualizar o estado de uma encomenda (exemplo: "Em Processamento", "Enviado", "Concluído").
  - Opção de cancelamento de encomendas diretamente pela interface do vendedor.

- **Sprint 5 (Semana 10-11): *Dashboard* Dinâmico**

Na última fase do desenvolvimento da Área de Gestão do Vendedor, implementámos um *dashboard* interativo para oferecer uma visão geral dos negócios do vendedor. As funcionalidades incluíram:

  - Exibição de métricas importantes, como volume de vendas, número de encomendas e avaliações de clientes.
  - Representação gráfica dos dados para facilitar a interpretação de tendências e desempenho da loja.

Esta funcionalidade foi desenvolvida **paralelamente** com as restantes áreas da aplicação, garantindo que a experiência do vendedor estivesse alinhada com o fluxo de navegação do cliente. Devido à complexidade desta *feature*, foi necessário dedicar um período de desenvolvimento mais extenso para garantir uma implementação robusta e escalável.

Este planeamento permitiu que a **Área de Gestão do Vendedor** fosse construída de forma progressiva, assegurando que cada componente estivesse completamente funcional antes de avançar para a seguinte, resultando numa interface intuitiva e eficiente para os utilizadores da plataforma.

2.2.5 Sprint 5 - Implementação da Listagem de Produtos e Páginas Essenciais – Semana 7

Nesta fase, focámo-nos na construção das principais páginas da aplicação, assegurando que a navegação fosse intuitiva e funcional. As implementações incluíram:

- **Homepage:** Desenvolvida para apresentar produtos em destaque, lojas recomendadas e produtos em destaques das lojas mais próximas, proporcionando uma primeira experiência envolvente ao utilizador.

Figura 1 HomePage

- **Cards de Produtos:** Cada produto foi representado num card com imagem, nome, preço e vendedor, garantindo um design responsivo e apelativo.

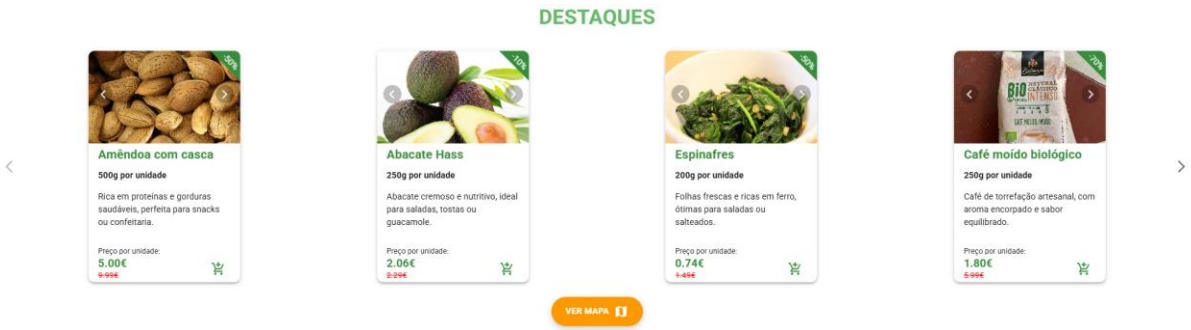

Figura 2 Cards de Produtos

- **Página de Lojas:** Criámos um layout dedicado onde cada loja apresenta a sua descrição, localização e lista de produtos disponíveis.

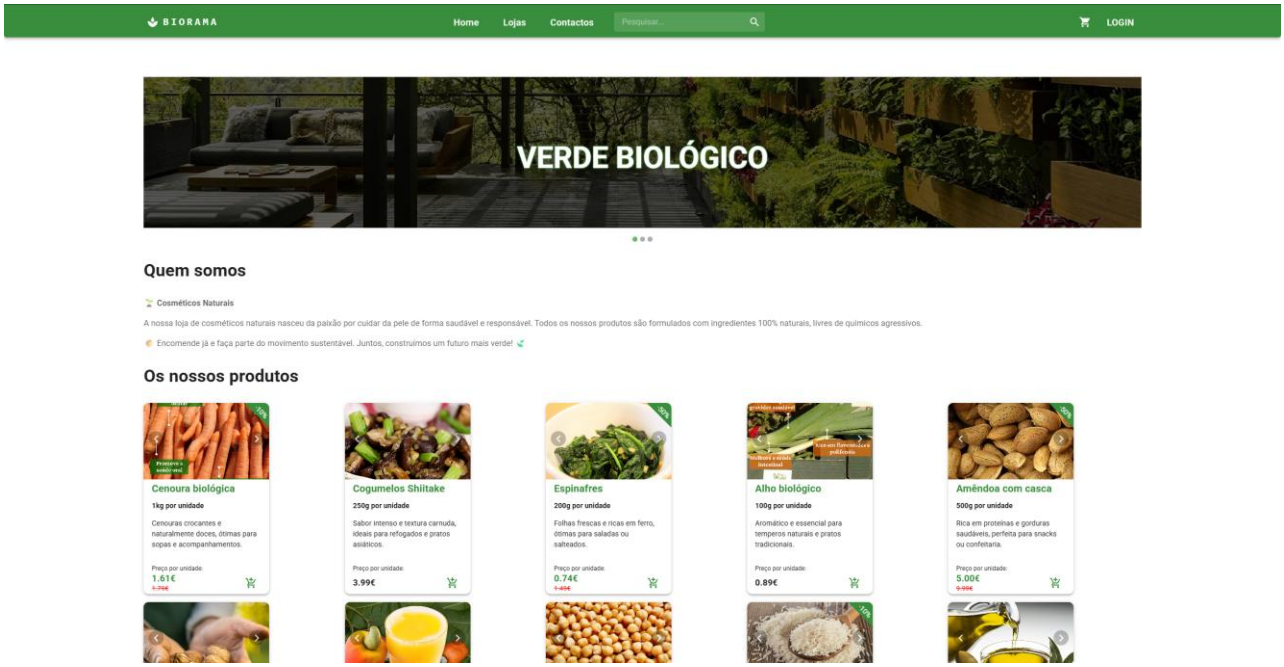

Figura 3 Página de Loja

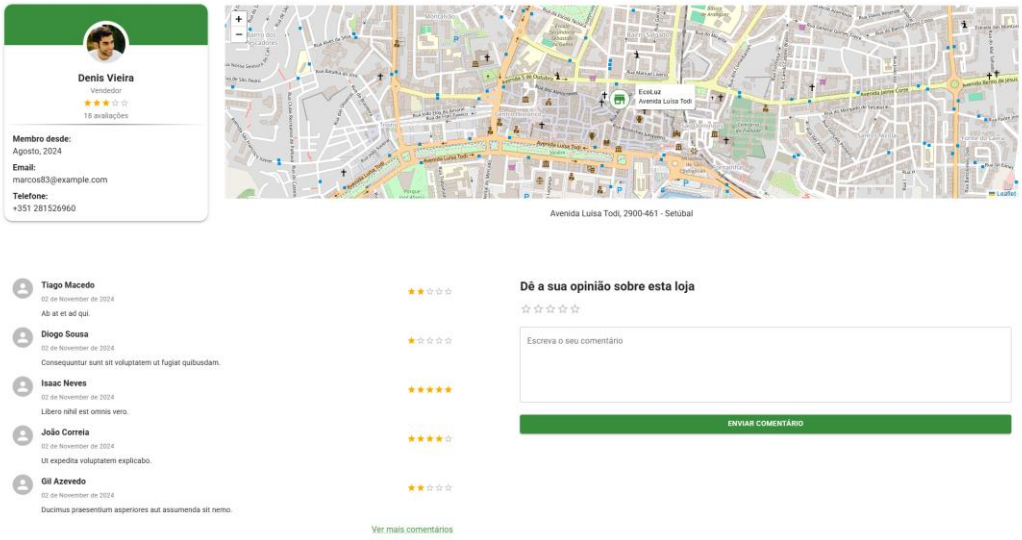

Figura 4 Página de Loja 2

- **Página do Carrinho de Compras:** Desenvolvida para proporcionar um fluxo de compra claro e simplificado, permitindo que o utilizador visualize os produtos selecionados, modifique quantidades e avance para o checkout.

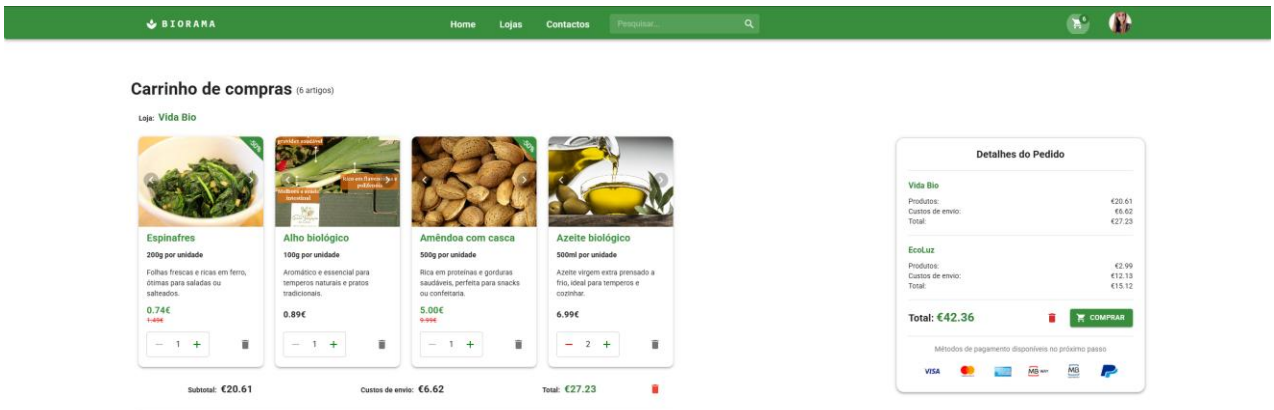

Figura 5 Carrinho de compras

- **Navegação e Usabilidade:** Implementámos menus intuitivos e filtros de pesquisa para melhorar a experiência do utilizador e facilitar a descoberta de lojas.

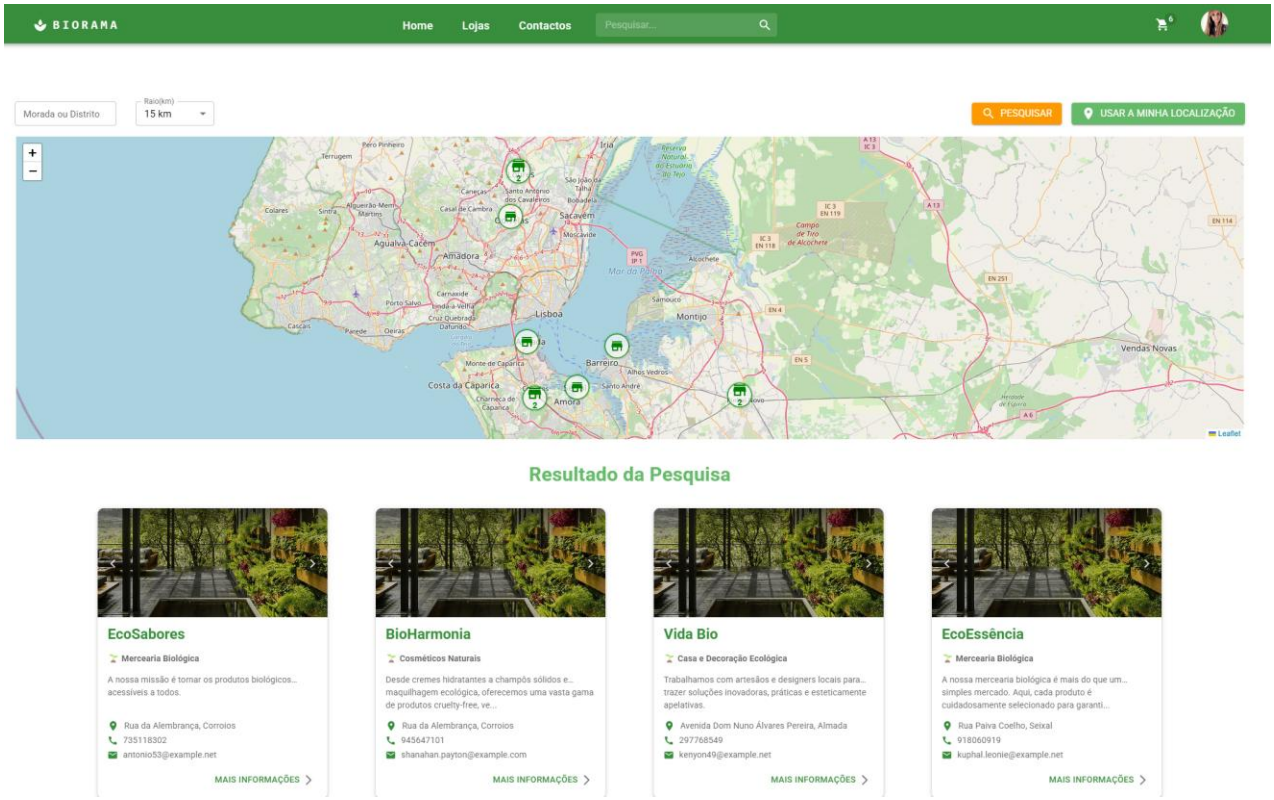

Figura 6 Página para procurar lojas

2.2.6 Sprint 6 - Desenvolvimento do Mapa Interativo – Semana 8

Para melhorar a experiência do utilizador na descoberta de lojas próximas, desenvolvemos um mapa interativo, utilizando geolocalização para fornecer informações em tempo real. As funcionalidades implementadas incluíram:

- **Geolocalização do Utilizador:** Permitiu identificar a localização do utilizador e exibir as lojas mais próximas.
- **Exibição de Lojas no Mapa:** Cada loja foi representada por um ícone interativo, permitindo que o utilizador aceda rapidamente à página da loja.
- **Integração na Página da Loja:** O mapa foi incorporado na página de cada loja, mostrando a sua localização exata e facilitando a navegação para o cliente.
- **Geocodificação de Moradas:** Garantimos que todas as moradas inseridas pelos vendedores fossem corretamente convertidas em coordenadas para serem exibidas no mapa.

2.2.7 Sprint 7 - Implementação da Pesquisa – Semana 9

A pesquisa foi identificada como um elemento essencial para a navegabilidade da aplicação, permitindo que os utilizadores encontrassem rapidamente as lojas disponíveis na plataforma.

No entanto, devido a limitações de tempo e à priorização de outras funcionalidades críticas, neste sprint conseguimos apenas implementar a pesquisa por lojas.

A implementação focou-se nos seguintes aspetos:

- **Barra de Pesquisa Global**: Disponível em todas as páginas da aplicação, permitindo que o utilizador procurasse lojas pelo nome.
- **Apresentação dos Resultados**: Cada loja encontrada era apresentada num card de loja, contendo informações relevantes, como nome, localização e o card é um link direto para a página dessa loja.
- **Redireccionamento para a Página da Loja**: O utilizador podia clicar diretamente no card para ser levado à página da loja correspondente.

Inicialmente, a nossa intenção era expandir a pesquisa para incluir produtos e vendedores, oferecendo resultados diferenciados com opções interativas, como adicionar produtos ao carrinho diretamente da pesquisa e visualizar perfis de vendedores. No entanto, devido a constrangimentos

de tempo e à complexidade adicional envolvida, não foi possível avançar com essa implementação nesta sprint.

A pesquisa foi, portanto, desenvolvida com um escopo mais restrito, ficando o aprimoramento desta funcionalidade planeado para futuras iterações do projeto

2.2.8 Sprint 8 - Implementação da Gestão de Encomendas, Métodos de Pagamento e Envio de Faturas – Semana 10

Nesta fase, garantimos que o fluxo de compra e pagamento estivesse completo e integrado com a lógica da aplicação. As implementações incluíram:

- **Gestão de Encomendas**: Criámos um sistema onde os utilizadores podiam consultar o histórico de encomendas, visualizar detalhes e acompanhar o estado da entrega.
- **Cálculo de Portes de Envio**: Desenvolvemos um algoritmo que calcula o custo do envio com base no peso da encomenda e na distância do destinatário.
- **Escolha de Método de Pagamento**: Integramos diferentes opções de pagamento, incluindo cartão de crédito/débito e pagamentos digitais.
- **Criação de Faturas**: Implementámos um sistema que permite gerar faturas automaticamente para os clientes após a conclusão da compra.
- **Escolha de Método de Pagamento:** Integramos diferentes opções de pagamento, incluindo cartão de crédito/débito ou pagamento via PayPal.
- **Envio de Faturas:** Implementámos um sistema que permite enviar faturas automaticamente para os clientes após a conclusão da compra.

2.2.9 Sprint 9 - Ligação das Encomendas e Notificações na Gestão de Perfil do Utilizador – Semana 11

Nesta última fase, focámo-nos na integração das **notificações em tempo real** e na ligação com as encomendas. As funcionalidades desenvolvidas foram:

- **Notificações para Alterações no Estado das Encomendas**: Os utilizadores passaram a receber notificações sempre que o estado da sua encomenda fosse atualizado (exemplo: "Encomenda enviada", "Encomenda entregue").
- **Histórico de Notificações**: Criámos uma aba de notificações dentro do perfil do utilizador para que pudesse rever atualizações anteriores.
- **Integração Total com o Perfil do Utilizador**: As notificações e encomendas foram organizadas dentro da gestão do perfil, permitindo ao utilizador aceder rapidamente a todas as informações relevantes.

Testes contínuos (Durante todo o projeto): Sempre que uma nova funcionalidade foi implementada, foram realizados testes para garantir a consistência da aplicação e a integração correta entre funcionalidades.

3 Requisitos

O levantamento de requisitos para a nossa aplicação foi realizado com o objetivo de garantir que a solução desenvolvida atende às necessidades dos utilizadores e dos vendedores de produtos sustentáveis do projeto.

A estruturação dos requisitos baseou-se em critérios de viabilidade técnica, usabilidade e escalabilidade do sistema, garantindo que a plataforma possa crescer de forma eficiente sem comprometer a experiência do utilizador.

A identificação das funcionalidades a implementar levou em consideração a experiência de navegação dos utilizadores, a segurança na gestão de dados e a otimização do fluxo de compras e vendas.

A organização e priorização dos requisitos permitiu um planeamento estruturado do desenvolvimento, dividindo as funcionalidades em diferentes níveis de importância.

Dessa forma, assegurou-se que os elementos essenciais fossem desenvolvidos prioritariamente, enquanto funcionalidades complementares poderiam ser adicionadas conforme a evolução do projeto.

3.1 Requisitos Funcionais

Autenticação e Registo

- O utilizador pode registar-se e iniciar sessão com e-mail e palavra-passe.
- O utilizador deve verificar o seu e-mail através de um link enviado antes de concluir o registo.
- O utilizador pode recuperar a palavra-passe através de um e-mail de redefinição.
- O utilizador pode manter a sessão ativa, optando por se lembrar da sessão ao iniciar sessão.
- O sistema deve garantir segurança no armazenamento das credenciais, utilizando *hashing* de palavras-passe. Pesquisa e Listagem de Produtos

Pesquisa e Listagem de Produtos

- O utilizador pode pesquisar lojas pelo nome através da barra de pesquisa global.
- A página de resultados deve exibir todas as lojas correspondentes à pesquisa, apresentando cartões com informações básicas e um link para a página da loja.
- No futuro, a pesquisa será expandida para permitir pesquisa por produtos e vendedores, com diferentes funcionalidades.

Gestão do Carrinho de Compras

- O utilizador pode adicionar produtos ao carrinho e definir a quantidade desejada.
- O utilizador pode remover produtos do carrinho a qualquer momento.
- O carrinho deve atualizar automaticamente o total da compra, considerando os produtos adicionados.
- O utilizador pode avançar para o checkout, onde escolhe o método de pagamento e a morada de entrega.

Gestão de Encomendas

- O utilizador pode criar uma encomenda, selecionando produtos e definindo a morada de entrega.
- O vendedor pode visualizar e gerir as encomendas recebidas, alterando o estado da encomenda (exemplo: "Em processamento", "Enviado", "Concluído").
- O administrador pode visualizar todas as encomendas, gerir disputas e cancelar pedidos se necessário.
- O utilizador recebe notificações sobre o estado da encomenda, garantindo acompanhamento em tempo real.

Gestão de Lojas e Produtos (Vendedores)

- O vendedor pode criar, editar e apagar lojas virtuais, com um limite máximo de três lojas.
- O vendedor pode adicionar, editar e remover produtos da sua loja.
- O vendedor pode gerir o stock de produtos, atualizando as quantidades disponíveis.
- O vendedor pode visualizar todas as encomendas associadas às suas lojas e modificar o estado das mesmas.
- O vendedor pode analisar métricas de desempenho no painel de controlo, como volume de vendas e avaliações.

Mapa Interativo

- O utilizador pode visualizar lojas próximas com base na sua localização, utilizando geolocalização.
- O utilizador pode clicar no ícone da loja no mapa para aceder diretamente à página da loja.
- O sistema armazena coordenadas das lojas através da geocodificação das moradas inseridas pelos vendedores.

Avaliações e Comentários

- O comprador pode avaliar uma loja do vendedor após uma compra, atribuindo uma pontuação de 1 a 5 estrelas e adicionar um comentário.
- Cada vendedor possui um rating global, que corresponde à média das avaliações de todas as suas lojas.

3.2 Requisitos não funcionais

Desempenho

- A aplicação deve responder instantaneamente às interações do utilizador, sem necessidade de recarregar páginas, aproveitando a renderização dinâmica do *React*.
- As atualizações da interface devem ser refletidas em tempo real através de sincronização eficiente com a base de dados.

Usabilidade

- A interface deve ser intuitiva e responsiva, garantindo uma experiência fluida em qualquer dispositivo.
- Todos os componentes *Material UI* devem ser utilizados de forma consistente para garantir uma experiência homogénea e moderna.
- A experiência do utilizador deve ser otimizada para evitar passos desnecessários no fluxo de compra e gestão de lojas.
- O design deve permitir ações rápidas, evitando cliques redundantes e mantendo uma navegação fluida.

Compatibilidade

- A aplicação deve ser compatível com todos os navegadores modernos (Google Chrome, Mozilla Firefox, Microsoft Edge e Safari) na última versão estável.
- Deve suportar diferentes resoluções de ecrã, garantindo que a experiência seja otimizada para:
    - Desktop (>1200px)
    - Tablets (768px - 1200px)
    - Dispositivos móveis (<768px)
- A aplicação deve ser otimizada para toques e gestos em dispositivos móveis, garantindo uma navegação eficiente sem depender de cliques de rato.
- O sistema deve manter suporte a futuras atualizações de bibliotecas como React e Material UI sem comprometer a estabilidade da aplicação.

3.3 Priorização de Requisitos

A priorização dos requisitos é fundamental para garantir um desenvolvimento eficiente, focando primeiro nas funcionalidades essenciais para o funcionamento da nossa aplicação.

Para isso, utilizamos a **Matriz MoSCoW**, que classifica os requisitos em quatro categorias:

***Must Have* (Obrigatórios)**

São requisitos críticos para o funcionamento da aplicação. Sem eles, o sistema não pode ser considerado funcional.

- Autenticação de utilizadores (Login/Registo via email).
- Pesquisa e listagem de produtos nome de loja.
- Gestão de carrinho de compras (adicionar/remover produtos, visualizar total).
- Processamento de encomendas (seleção de morada, checkout e confirmação).
- Gestão de lojas e produtos para vendedores.
- Mapa interativo para localização de lojas próximas.
- Segurança na autenticação.
- Compatibilidade com dispositivos móveis e desktop.

***Should Have* (Importantes)**

Funcionalidades desejáveis, que melhoram a experiência do utilizador e a gestão da plataforma, mas que podem ser implementadas numa segunda fase.

- Histórico de encomendas para utilizadores e vendedores.
- Sistema de avaliações e comentários para vendedores e produtos.
- Gestão avançada de permissões (diferenciar administradores, vendedores e clientes).
- Sistema de *caching* para otimização do tempo de resposta.
- Melhoria na acessibilidade (suporte para leitores de ecrã e navegação por teclado).

***Could Have* (Opcionais)**

Funcionalidades que agregam valor à aplicação, mas não são essenciais no lançamento inicial.

- Chat ao vivo entre compradores e vendedores.
- Estatísticas de vendas para vendedores.

- Recomendações personalizadas de produtos baseadas no histórico do utilizador.
- Implementação de múltiplos idiomas para expansão internacional.

***Won't Have* (Para Evolução Futura)**

Funcionalidades que não serão desenvolvidas na fase inicial, mas podem ser adicionadas em futuras atualizações.

- Integração com múltiplos métodos de pagamento.
- Sistema de fidelização e promoções para clientes frequentes.
- Marketplace com anúncios patrocinados para vendedores destacados.
- Suporte para integração com redes sociais para partilha de produtos.
- Aplicação mobile.

4 Desenho do Sistema e Arquitetura

O desenvolvimento da aplicação exigiu um planeamento detalhado, estruturado em torno de uma arquitetura modular e escalável, garantindo separação de responsabilidades e eficiência na comunicação entre os componentes do sistema.

A solução foi desenvolvida utilizando *Laravel* (PHP) no *backend* e *React* com Inertia.js no *frontend*, interligados por uma *API RESTful* e uma base de dados *MySQL*.

4.1 Desenho do Sistema

O sistema foi desenhado para suportar três camadas principais:

- **Camada de Apresentação (Frontend)**

Desenvolvido em *React* com *Vite*, garantindo carregamento rápido e melhor performance.

Utilização de Inertia.js para eliminar a necessidade de uma API REST dedicada para servir o *frontend*, reduzindo a complexidade da gestão do estado.

Material UI para a estilização dos componentes, proporcionando uma experiência de utilizador moderna e responsiva.

Formik e YUP para o controlo e validação dos formulários, otimizando o desempenho através da gestão eficiente do estado dos inputs.

- **Camada de Lógica de Negócio (Backend)**

Desenvolvido com Laravel 11, garantindo um *backend* robusto e modular.

Estruturado segundo o padrão MVC (*Model-View-Controller*), facilitando a organização do código.

Autenticação e gestão de permissões utilizando *Laravel Breeze* para fornecer autenticação segura com *hashing* de senhas via *bcrypt*.

Validação de dados com *Form Requests*, permitindo uma verificação centralizada e reutilizável de dados enviados pelo *frontend*.

Sistema de Notificações integrado com *Laravel Notifications*, permitindo notificações via base de dados.

- **Camada de Persistência de Dados (Base de Dados)**

MySQL como sistema de gestão de base de dados, garantindo estabilidade e escalabilidade.

Migrações e *Seeders* utilizados para estruturar e popular a base de dados, permitindo recriação eficiente do esquema de dados.

Relacionamentos entre tabelas modelados utilizando *Eloquent ORM*, garantindo consultas otimizadas.

- **Comunicação entre Componentes**

A comunicação entre as camadas foi desenvolvida de forma eficiente, com:

APIs RESTful para troca de informações entre o *frontend e backend*.

Laravel Sanctum para autenticação baseada em *tokens* e gestão de sessões.

Axios para chamadas API no *frontend*, permitindo integração fluida entre *frontend e backend*.

Cache Redis para melhorar a performance e reduzir carga sobre a base de dados.

As principais rotas da aplicação incluem:

Autenticação

POST /entrar - Inicia sessão.

POST /registar - Regista um novo utilizador.

POST /sair - Termina a sessão.

POST /recuperar-palavra-passe - Recuperação de palavra-passe.

Gestão de Perfil

GET /perfil - Exibe o perfil do utilizador.

POST /editar-perfil/{user} - Atualiza as informações do utilizador.

GET /get-user - Obtém as informações do utilizador autenticado.

Gestão de Moradas

GET /get-moradas - Lista todas as moradas do utilizador.

POST /adicionar-morada - Adiciona uma nova morada.

POST /editar-morada/{morada} - Edita uma morada existente.

PATCH /morada/{id}/set-morada-fav - Define uma morada como favorita.

DELETE /apagar-morada/{morada_id} - Remove uma morada do utilizador.

Gestão de Lojas e Produtos

GET /lojas - Lista todas as lojas.

GET /loja/{id} - Obtém detalhes de uma loja.

POST /create/store - Cria uma loja.

POST /stores/{store}/update - Atualiza as informações de uma loja.

POST /registar-vendedor-produto/{store_id} - Adiciona um produto a uma loja.

DELETE /products/{product} - Remove um produto.

GET /search-products/{storeId} - Pesquisa produtos numa loja específica.

Gestão de Encomendas

POST /encomendar - Cria uma encomenda.

GET /encomendas-user - Lista todas as encomendas do utilizador.

GET /encomenda-user/{id} - Obtém detalhes de uma encomenda específica.

POST /paypal/create-order - Cria uma encomenda via PayPal.

POST /paypal/capture-order - Captura uma encomenda via PayPal.

Painel do Vendedor (Dashboard)

GET /dashboard - Mostra as informações do vendedor.

GET /dashboard/lojas - Lista as lojas do vendedor.

GET /dashboard/encomendas - Lista as encomendas recebidas pelo vendedor.

GET /dashboard/stores/{storeId}/orders - Obtém encomendas específicas de uma loja.

DELETE /dashboard/lojas/{id} - Remove uma loja do vendedor.

4.2 Arquitetura e Tecnologia

A arquitetura do sistema foi definida para garantir modularidade, segurança e escalabilidade, baseando-se em conceitos modernos de desenvolvimento.

A aplicação segue um modelo *client-server*, onde:

- O *frontend* (React + Inertia.js) interage diretamente com o *backend*, sem necessidade de uma API intermediária.
- O *backend* (*Laravel*) é responsável por gerir a lógica de negócio e fornecer dados ao *frontend*.
- A base de dados (*MySQL*) armazena todas as informações essenciais da aplicação.

O sistema foi projetado com foco na escalabilidade, permitindo que novos módulos e funcionalidades possam ser adicionados sem comprometer a performance ou segurança.

4.2.1 Tecnologias Utilizadas

A escolha das tecnologias foi feita com base em critérios de segurança, desempenho e manutenibilidade, garantindo uma aplicação robusta e escalável.

O *backend* foi desenvolvido em *Laravel* 11, um *framework* PHP que proporciona uma estrutura organizada baseada no padrão MVC (*Model-View-Controller*). A autenticação foi implementada com *Laravel Sanctum*, permitindo a gestão segura de sessões e *tokens*.

Para o controlo de acessos e permissões, utilizámos o pacote *Spatie Laravel Permission*, garantindo um controlo granular entre utilizadores comuns e vendedores.

No *frontend*, foi utilizada a combinação de *React + Inertia.js*, eliminando a necessidade de uma API REST separada para a comunicação entre *frontend e backend*, melhorando a performance e simplificando a estrutura do projeto.

Vite foi escolhido para otimizar o processo de compilação e melhorar os tempos de carregamento da aplicação. A estilização da interface foi feita com Material UI, proporcionando um design moderno e responsivo.

Para validação e envio de formulários, adotámos as bibliotecas *Formik e Yup*, reduzindo a necessidade de verificações repetitivas no *backend*.

O *Axios* foi utilizado para realizar chamadas assíncronas à API do *Laravel*, garantindo uma comunicação eficiente entre as diferentes camadas da aplicação.

A persistência de dados foi gerida através do MySQL, com a estrutura da base de dados definida por meio de *migrations* e os dados inseridos na base dados emm contexto de desenvolvimento foram realizados por *factories* e enviados por *seeders*.

A comunicação eficiente entre tabelas foi garantida pelo uso do *Eloquent ORM*, permitindo manipulação otimizada de dados e relações entre modelos.

Para funcionalidades adicionais, foram integradas *APIs* externas para melhorar a experiência do utilizador e otimizar processos internos:

- **CTT Código Postal API**: Utilizada para obter automaticamente o nome da rua, localidade e coordenadas da morada inserida pelo utilizador, garantindo maior precisão no preenchimento de endereços e reduzindo erros manuais.

- **OSRM API (*Open Source Routing Machine*)**: Implementada para calcular automaticamente os portes de envio, com base na distância real em estrada entre a morada do utilizador e a morada da loja onde é realizada a compra, proporcionando um cálculo de custos mais realista e transparente.
- ***Leaflet.js* + *OpenStreetMap***: Integrado para exibição das lojas num mapa interativo, permitindo aos utilizadores localizar rapidamente os estabelecimentos e obter direções precisas.

A utilização destas tecnologias e integrações permitiu criar uma aplicação eficiente, segura e otimizada, garantindo uma experiência fluida tanto para os utilizadores finais como para os vendedores.

Segurança e Otimização

Para garantir a proteção dos dados e otimizar o desempenho, foram implementadas as seguintes práticas:

- ***Middleware* de Segurança:**
  - *auth* para proteger rotas que exigem autenticação.
  - *role:vendor* para restringir o acesso a determinadas funcionalidades a vendedores.

- **Proteção contra CSRF e CORS:**
  - Todas as requisições POST, PUT, PATCH e DELETE são protegidas contra *Cross-Site Request Forgery* (CSRF).
  - Políticas CORS configuradas para evitar acessos não autorizados a partir de domínios desconhecidos.
- ***Eager Loading* em Consultas SQL:**
  - Uso de *with() e load()* para reduzir *queries* excessivas, melhorando a eficiência.

Com esta abordagem arquitetural e tecnológica, garantimos que a aplicação seja modular, segura, eficiente e escalável, proporcionando uma experiência de utilizador fluida e confiável.

4.3 Diagramas e Modelos

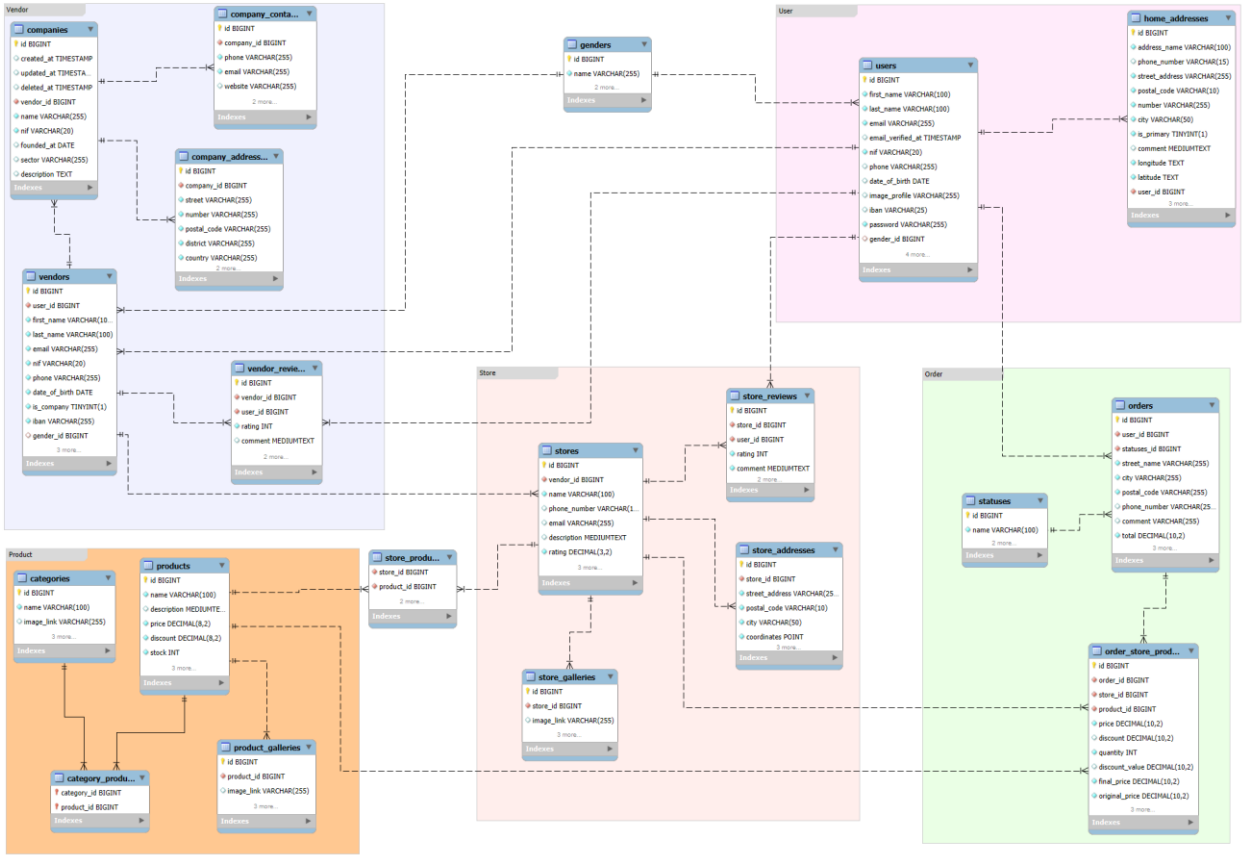

Figura 7 Diagrama Base Dados

A base de dados foi planeada de forma a abranger as necessidades dos utilizadores comuns, vendedores, lojas e produtos, garantindo uma navegação fluida e um funcionamento robusto da aplicação.

4.3.1 Gestão de Utilizadores

As informações dos utilizadores estão centralizadas na tabela users, que contém dados essenciais como nome, email, senha encriptada, género e IBAN (caso o utilizador seja um vendedor). Esta tabela está relacionada com:

- **home_addresses**: Armazena as moradas dos utilizadores, permitindo múltiplos endereços associados a um único utilizador.
    - Utiliza coordenadas geográficas (latitude e longitude), permitindo integração com mapas interativos.
    - O código postal pode ser validado via API CTT Código Postal para preenchimento automático da rua e localidade.
- **orders**: Relação direta entre um utilizador e as suas encomendas. Cada encomenda não é associada a uma morada através da chave estrangeira, pois decidimos guardar de modo que o utilizador se edita ou apaga-se a sua morada não afeta-se a tabela relativamente à encomenda.
- **genders**: Permite categorizar utilizadores por género, sendo uma tabela auxiliar.

4.3.2 Gestão de Vendedores

O sistema permite que um utilizador comum se torne vendedor, criando um perfil de vendedor na tabela *vendors.*

Esta tabela contém:

- **company_id**: Se o vendedor representar uma empresa, é referenciado na tabela **companies**, que armazena informações da empresa, como nome, setor, data de fundação e descrição.
- **vendor_reviews**: Sistema de avaliações do vendedor, concebido para permitir que os utilizadores atribuam classificações e deixem comentários sobre a sua experiência de compra. Embora esta funcionalidade não tenha sido implementada nesta fase, devido à priorização de outros requisitos essenciais, o modelo de dados foi estruturado de forma a suportar a sua futura integração sem necessidade de grandes alterações.

A estrutura modular do vendedor permite a integração fácil com funcionalidades como gestão de encomendas e relatórios de vendas

4.3.3 Gestão de Lojas e Produtos

Cada vendedor pode ter até três lojas, que são geridas na tabela **stores**. Esta tabela contém:

- **vendor_id**: Relaciona cada loja a um vendedor específico.
- **store_reviews**: relação com a tabela store_review que permite aos utilizadores avaliarem e comentem sobre a experiência com a loja, podendo deixar a sua avaliação em rating (varia entre 1 e 5 estrelas) e deixar um comentário
- **store_addresses**: Mantém as moradas das lojas, associadas a coordenadas geográficas para funcionalidade de mapa interativo.
- **store_galleries:** Permite o upload de múltiplas imagens por loja.

Os produtos são geridos na tabela **products**, que possui relações diretas com:

- **categories**: A categorização de produtos foi uma funcionalidade não implementada devido à priorização de funcionalidades mais essenciais para o sistema, o modelo de dados foi estruturado de forma a suportar a sua integração.
- **product_galleries**: Permite o upload de múltiplas imagens por produto.
- **store_products**: Liga produtos a lojas, permitindo que um mesmo produto seja vendido em múltiplas lojas.

Essa estrutura facilita a pesquisa de produtos e garante que cada loja tenha um catálogo único e personalizável.

4.3.4 Carrinho de Compras e Gestão de Encomendas

A compra de produtos segue um fluxo organizado através das tabelas:

- **orders**: Representa uma encomenda feita por um utilizador, armazenando o estado e os detalhes do pedido.
- **order_store_products**: Relaciona encomendas, lojas e produtos, garantindo um sistema de checkout que suporta múltiplas lojas numa única compra, mas que sejam tratadas de forma individual no backend. Assim, o sistema foi desenvolvido para dar uma experiencia de utilização de apenas compra única ao cliente mas cada vendedor recebe uma encomenda separada de cada sua loja.
- **statuses**: Permite a atualização do estado de uma encomenda, como pendente, processando, enviado ou concluído.

A inclusão de campos na tabela order_store_products como: discount, price (referente ao preço unitário), quantity, discount_value, final_price e original_price permitem o rastreamento de promoções aplicadas no momento da compra.

5 Implementação

A fase de implementação corresponde à transformação do planeamento detalhado em código funcional, garantindo que a aplicação cumpra os requisitos definidos e seja eficiente, segura e escalável.

O desenvolvimento da nossa aplicação foi conduzido com base em boas práticas de programação, utilizando tecnologias modernas como *Laravel* no *backend* e React.js no *frontend*, garantindo uma experiência dinâmica e fluida para o utilizador.

A estratégia de desenvolvimento baseou-se na separação de responsabilidades entre *frontend* e *backend*, assegurando que ambos os lados da aplicação comunicassem de forma eficiente e otimizada.

Além disso, foi utilizada uma abordagem modula, permitindo uma maior facilidade na manutenção e escalabilidade do projeto.

5.1 Ambiente de Desenvolvimento

O ambiente de desenvolvimento foi configurado para garantir eficiência, produtividade e escalabilidade, utilizando ferramentas que facilitam a escrita, teste e manutenção do código.

Ferramentas Utilizadas

- *PhpStorm e Visual Studio Code*: Escolhidos como IDE's para o desenvolvimento da nossa aplicação.
- MySQL Workbench: Ferramenta gráfica para modelação e gestão da base de dados, facilitando a execução de *queries* e a visualização das relações entre tabelas.
- Docker: Usado para virtualização do ambiente de desenvolvimento, garantindo consistência entre as máquinas dos programadores e eliminando problemas de dependências locais.

Tecnologias Adotadas

Backend:

- Laravel (PHP 8.3): Escolhido por permitir rápida criação de APIs RESTful e integração simplificada com a base de dados através do Eloquent ORM.
- MySQL: Base de dados relacional utilizada devido à sua robustez e escalabilidade, suportando todas as operações essenciais da aplicação.
- Laravel Sanctum: Implementado para gestão da autenticação e segurança, garantindo que cada utilizador tenha sessões seguras e protegidas.

Frontend:

- React.js: Escolhido para proporcionar uma SPA (Single Page Application), eliminando recarregamentos desnecessários e tornando a navegação fluida.
- Material UI (MUI): Utilizado para a criação de componentes responsivos e modernos, garantindo uma interface coesa e intuitiva.
- React Router DOM: Biblioteca utilizada para gerir navegação dinâmica, permitindo que as páginas sejam carregadas sem afetar a experiência do utilizador.
- API CTT Código Postal: API utilizada para, a partir do código postal, obter e armazenar automaticamente a rua correspondente e as suas coordenadas geográficas.
- OSRM API: API utilizada para o cálculo dos portes de envio com base na distância em estrada entre a morada de entrega selecionada pelo utilizador e a morada da loja onde é feita a encomenda.

Comunicação *Backend-Frontend*:

- Inertia.js: Utilizado para comunicação direta entre Laravel e React, eliminando a necessidade de uma API REST convencional e melhorando a performance da aplicação.
- Axios: Implementado para requisições assíncronas entre *frontend* e *backend*, garantindo eficiência e menor tempo de resposta.

Gestão de Estado:

- MobX: Escolhido para gerir o estado global da aplicação, evitando *prop-drilling* excessivo e garantindo persistência de dados na sessão.

Mapas e Geolocalização:

- *Leaflet e OpenStreetMap*: Utilizados para exibição de lojas num mapa interativo, permitindo que os utilizadores encontrem lojas próximas.

A combinação destas ferramentas garantiu um desenvolvimento ágil, organizado e altamente escalável, proporcionando um ambiente sólido para a construção da plataforma.

5.2 Desenvolvimento

O desenvolvimento da nossa aplicação seguiu uma abordagem modular e escalável, permitindo uma integração fluída entre *backend* e *frontend*.

Durante a construção da aplicação, garantimos que cada funcionalidade fosse implementada com boas práticas de código, assegurando segurança, eficiência e facilidade de manutenção.

No *backend*, utilizámos Laravel, organizando a arquitetura da aplicação com base no padrão MVC (*Model-View-Controller*), o que facilita a organização do código e a sua futura manutenção.

Para garantir uma comunicação eficiente entre as diferentes camadas da aplicação, foram implementadas *APIs RESTful*, permitindo que o *frontend* acedesse e manipulasse dados de forma estruturada.

A gestão da base de dados foi realizada através do *Eloquent ORM*, permitindo consultas eficientes e relações bem definidas entre as tabelas.

Além disso, a segurança da autenticação foi assegurada através do *Laravel Sanctum*, proporcionando uma gestão segura de sessões e autenticação baseada em tokens.

Para estruturar e popular a base de dados, utilizámos *migrations e seeders*, permitindo um desenvolvimento controlado e escalável.

No *frontend*, recorremos a React juntamente com Material UI (MUI) para desenvolver uma interface moderna, responsiva e intuitiva.

A comunicação com o *backend* foi assegurada através de Inertia.js e Axios, garantindo uma integração direta e eficiente entre as camadas da aplicação.

Para garantir uma experiência de utilização segura e funcional, implementámos validação de formulários com *Formik e Yup*, evitando a inserção de dados incorretos e reduzindo a carga no *backend.*

Para otimizar a gestão do estado da aplicação, adotámos MobX, permitindo um controlo eficiente dos dados e evitando chamadas desnecessárias à API. Foram criadas várias Stores específicas para diferentes funcionalidades, garantindo uma organização modular e facilitando a reutilização de estados e ações em toda a aplicação. Algumas das principais Stores implementadas incluem:

- **AuthStore**, responsável pela gestão da autenticação, garantindo que o login do utilizador seja persistente e permitindo a atualização dos seus dados sem necessidade de múltiplas chamadas ao servidor.
- **ShopStore**, dedicada à gestão de lojas, permitindo a criação, edição e remoção de lojas de forma dinâmica.
- **OrderStore**, encarregue do armazenamento e manipulação de dados de encomendas, facilitando a filtragem, ordenação e gestão dos pedidos por parte dos utilizadores e vendedores.

Esta abordagem modular e bem estruturada permitiu-nos desenvolver uma aplicação coesa, onde cada componente interage de forma eficiente com os demais, assegurando uma experiência fluida para o utilizador e facilitando futuras melhorias e expansões da plataforma.

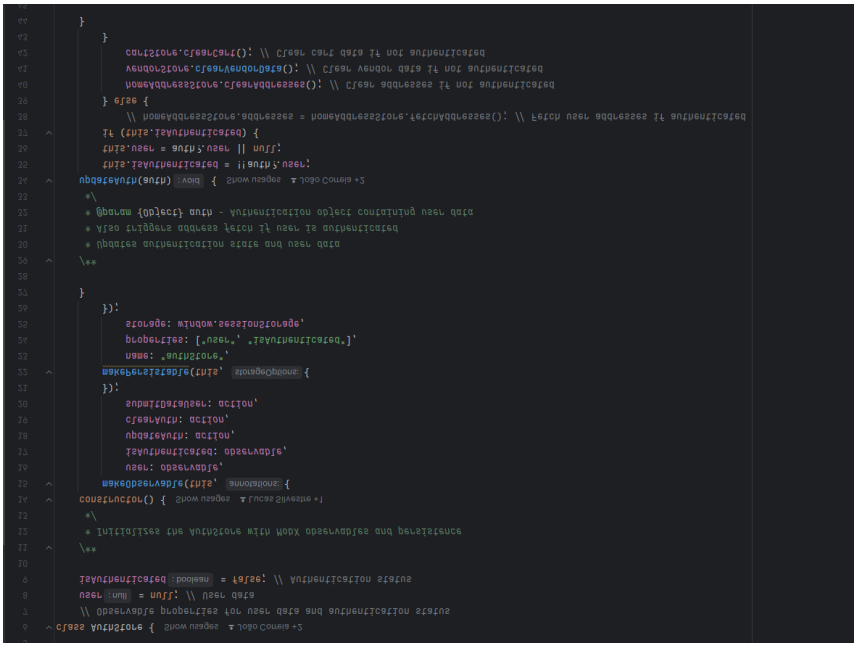

Figura 8 Exemplo Store (AuthStore) Mobx

5.3 Controle de Versão

Para garantir um fluxo de desenvolvimento seguro e organizado, utilizámos Git como sistema de controlo de versão, com o repositório alojado no GitHub.

O grupo trabalhou maioritariamente na *branch main* onde as funcionalidades foram desenvolvidas e integradas diretamente, garantindo uma implementação contínua e sem comprometer a estabilidade da aplicação.

Embora não tenhamos seguido a metodologia Git Flow, mantivemos um *branch develop* reservado para testar funcionalidades mais críticas, novas tecnologias ou abordagens de desenvolvimento antes de as integrar na versão estável. Isto permitiu-nos experimentar e validar alterações mais complexas sem comprometer a versão principal da aplicação.

Para assegurar a qualidade do código, realizámos sempre que era feito um *merge* da versão estável (*main*) com uma nova funcionalidade. Estes testes garantiram que todas as alterações fossem devidamente validadas antes de serem incorporadas no projeto.

Além disso, adotámos a prática de commits descritivos, assegurando que cada alteração fosse devidamente documentada e fácil de rastrear. Esta abordagem permitiu-nos manter um histórico claro das modificações, facilitando a colaboração e a manutenção do projeto ao longo do tempo.

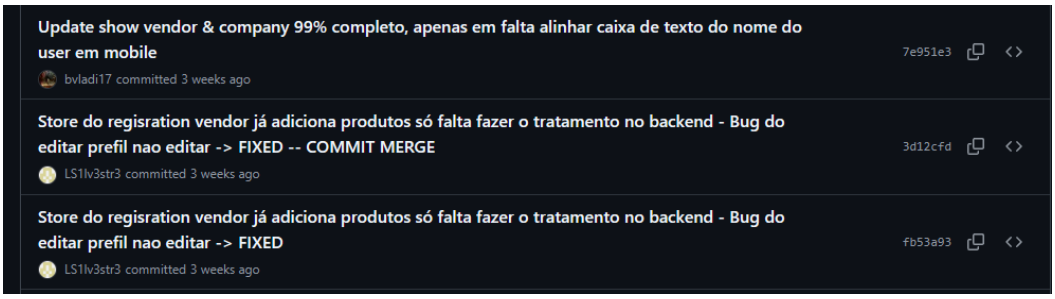

Figura 9 Exemplos de commits descritivos utilizados durante o desenvolvimento

6 Testes

Durante o desenvolvimento da nossa aplicação, os testes foram realizados de forma contínua, focando-se principalmente na verificação das funcionalidades implementadas.

O grupo adotou uma abordagem prática, testando cada funcionalidade manualmente, assumindo o papel de utilizadores finais para garantir que tudo funcionava corretamente.

Os testes foram efetuados diretamente na interface, validando a navegação, interação com os formulários, submissão de dados e respostas do sistema.

Além disso, testámos as stores do MobX, garantindo que a gestão de estado funcionava como esperado.

Sempre que um erro era identificado, corrigíamos de imediato e voltávamos a testar, garantindo um funcionamento fluído e sem falhas evidentes. Embora não tenham sido implementados testes automatizados, o processo iterativo permitiu assegurar a estabilidade da aplicação antes da entrega final.

7 Evolução

A nossa aplicação foi desenvolvida com uma arquitetura escalável, permitindo futuras expansões e otimizações para melhorar a experiência do utilizador e a gestão da plataforma.

Apesar de a versão atual oferecer um conjunto sólido de funcionalidades, identificámos várias melhorias e novas implementações que poderão ser desenvolvidas em versões futuras para tornar a plataforma ainda mais completa e eficiente.

Uma das melhorias fundamentais será o aumento da capacidade de utilizadores, através da otimização do backend e da base de dados. Para garantir um desempenho fluido mesmo com um grande volume de acessos simultâneos, poderemos adotar técnicas de *caching* e balanceamento de carga**,** reduzindo a latência e melhorando o tempo de resposta da aplicação.

A pesquisa avançada será outro ponto essencial a evoluir. Atualmente, a pesquisa permite encontrar apenas lojas, mas pretendemos expandir esta funcionalidade para incluir a busca por produtos e vendedores, facilitando a descoberta de itens específicos e melhorando a experiência do utilizador na plataforma.

A interface gráfica da aplicação poderá ser refinada, tornando a navegação ainda mais intuitiva e agradável. A introdução de novos componentes visuais e animações dinâmicas ajudará a criar uma experiência de utilização mais envolvente, garantindo que a plataforma se mantenha moderna e atrativa.

Para facilitar a comunicação entre compradores e vendedores, será implementado um sistema de mensagens, permitindo que os utilizadores possam esclarecer dúvidas diretamente com os vendedores antes de realizar uma compra.

Paralelamente, os vendedores passarão a receber notificações em tempo real sempre que forem feitas novas encomendas, receberem avaliações ou mensagens de clientes, agilizando a sua resposta e melhorando a gestão do negócio.

A autenticação com redes sociais será uma funcionalidade relevante para melhorar a acessibilidade da plataforma. A possibilidade de os utilizadores realizarem login com Google, Facebook ou outras redes sociais tornará o processo de registo mais rápido e intuitivo, incentivando novos utilizadores a aderirem à aplicação sem a necessidade de criar uma conta manualmente.

Além disso, consideramos implementar um sistema de tickets para produtos, onde os clientes poderão abrir pedidos de suporte diretamente em determinados produtos. Esta funcionalidade permitirá esclarecer dúvidas antes da compra ou reportar problemas de forma mais organizada e eficiente.

Apesar de o site ter sido desenvolvido com design totalmente responsivo, acreditamos que a presença no mercado mobile deve ser fortalecida com o desenvolvimento de uma aplicação móvel dedicada. Uma app permitirá uma experiência ainda mais otimizada para dispositivos móveis, com notificações push, melhor desempenho e uma interface pensada especificamente para smartphones e tablets, garantindo que os utilizadores tenham acesso à plataforma de forma mais prática e acessível.

Estas melhorias e novas funcionalidades têm como objetivo tornar o Biorama uma plataforma ainda mais completa, eficiente e adaptada às necessidades do mercado, proporcionando uma experiência superior tanto para compradores como para vendedores e garantindo a sua evolução contínua.

8 Conclusões

O projeto Biorama representou a conclusão do nosso percurso no Curso Técnico Especializado em Programação de Sistemas de Informação, permitindo-nos aplicar e aprofundar os conhecimentos adquiridos ao longo da formação.

Sob a orientação dos formadores Vítor Custódio e Gonçalo Feliciano, percorremos todas as fases do desenvolvimento de um software real, desde o planeamento até à implementação. Durante este processo, enfrentámos desafios que fortaleceram as nossas competências técnicas e organizacionais, preparando-nos para o ambiente profissional.

Uma das decisões mais relevantes do projeto foi a adoção de tecnologias não abordadas no curso, nomeadamente React com Inertia.js para o frontend. Esta escolha exigiu uma aprendizagem autónoma contínua e uma adaptação constante, proporcionando-nos uma compreensão mais aprofundada da integração entre frontend e backend. Em paralelo, utilizámos Laravel para a lógica do servidor, consolidando os conceitos adquiridos ao longo do curso.

Além do desenvolvimento técnico, este projeto também nos permitiu aprimorar competências organizacionais essenciais. Desde a estruturação da base de dados até à implementação das funcionalidades principais, seguimos uma abordagem bem definida, respeitando a organização e priorização dos requisitos estabelecidos. A gestão eficiente do trabalho em equipa e a capacidade de adaptação a desafios inesperados foram fundamentais para garantir uma implementação estruturada e eficaz.

A metodologia Scrum desempenhou um papel central na organização do projeto, permitindo-nos gerir tarefas, definir prioridades e adaptar estratégias conforme necessário. O uso do Git como sistema de controlo de versão e a realização de testes contínuos garantiram um código limpo, funcional e sustentável.

Em retrospetiva, o Biorama não só consolidou os conhecimentos adquiridos ao longo do curso, como também nos proporcionou experiência prática na resolução de problemas do mundo real, reforçou a nossa autonomia na aprendizagem de novas tecnologias e aprimorou a nossa capacidade de tomada de decisão.

Este projeto revelou-se um marco importante no nosso percurso formativo, preparando-nos para enfrentar os desafios do mercado de trabalho com maior confiança, organização e inovação.