ESCOLA
SUPERIOR
DE TECNOLOGIA
E GESTÃO
POLITÉCNICO
DO PORTO

LICENCIATURA
ENGENHARIA INFORMÁTICA

BoLine

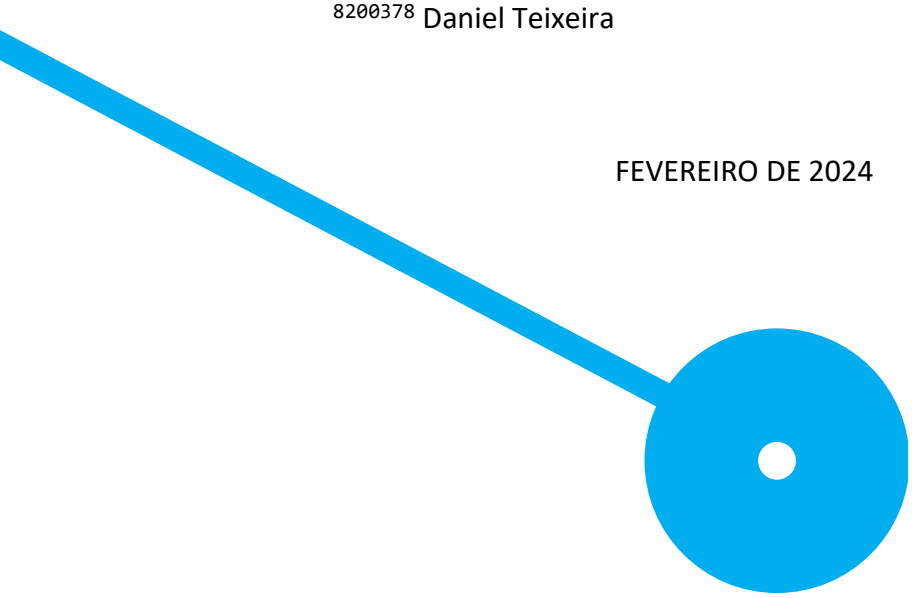
Grupo 07

8210333 Bruno Novais

8190302 Hélder Branco

8200378 Daniel Teixeira

FEVEREIRO DE 2024



ESCOLA
SUPERIOR
DE TECNOLOGIA
E GESTÃO
POLITÉCNICO
DO PORTO

LICENCIATURA
ENGENHARIA INFORMÁTICA

BoLine

Doutor Fábio Silva

Doutor João Ramos

Doutor Nelson Rodrigues

RESUMO

Este trabalho é elaborado no ano letivo 2022/2023 no seguimento do enunciado do Trabalho Prático, respetivo à Unidade Curricular de Programação em Ambiente Web, pertencente ao plano de estudos do 2º ano do 2º semestre, das Licenciaturas em Engenharia Informática e Segurança Informática em Redes de Computadores, ministradas na Escola Superior de Tecnologia e Gestão, Unidade Orgânica do Instituto Politécnico do Porto.

O seguinte trabalho tem em vista a implementação de uma aplicação web para a promoção do património português. Esta aplicação deve gerir um conjunto de imóveis de interesse histórico como por exemplo, a venda de bilhetes para estes locais ou eventos que aconteçam nos mesmos, a promoção dos locais e gerar estatísticas sobre os mesmos, com objetivo de fomentar a visita aos mesmos.

Portanto, tendo em consideração o projeto que nos foi apresentado, foi desenvolvida uma aplicação web que promove e gere todo o património de interesse público, sendo que o desenvolvimento da aplicação foi dividido em 2 *Milestones*.

Índice

DOCUMENTO

FIGURAS

TABELAS

CAP. I Contextualização e Motivação

01. Introdução

1.1. Objetivos

O projeto tem como principal objetivo o desenvolvimento de uma aplicação web que passa pela promoção e gestão de visitantes em cada local, a plataforma deve oferecer suporte a vários imóveis e permitir construir estatísticas. O objetivo principal passa pelos seguintes objetivos de menor dimensão:

- # Gerir património oferecido na plataforma, com descrição, ilustrações/vídeo e com uma lista de eventos associados;
- # Gerir a emissão de bilhetes e oferecer um programa de lealdade de forma a fomentar a visita a estes mesmos locais/eventos;
- # Gerir estatísticas em um *dashboard* de administração indicando métricas relevantes.
- # Sistema de pontos, de forma a fomentar a visita a estes locais.

A 1ª *Milestone* trata-se do desenvolvimento do *backoffice* da aplicação sendo os pontos mais importantes os seguintes:

- # Registo de funcionários com o papel de administradores;
- # Gestão de locais/monumentos e respetivos eventos;
- # Registo de utilizadores e emissão de bilhetes;
- # Sistema de pontos.

A 2ª *Milestone* assenta no desenvolvimento da aplicação web toda ela funcional. É também desenvolvido um *frontoffice* recorrendo à plataforma Angular utilizando os serviços web para a gestão da mesma. Relativamente ao servidor, são utilizados *endpointsREST e uma REST_API* com base em *nodeJS* e *ExpressJS* para criar serviços que dão apoio ao suporte da aplicação.

O *frontoffice* é também onde o utilizador comum interage com a plataforma podendo ver a oferta e comprar os seus bilhetes através do sistema de pontos ou métodos de pagamento disponibilizado e ver todos os detalhes da sua conta(perfil, bilhetes comprados,…).

1.2. Resultados

Da implementação do projeto surge como resultado um sistema que corresponde aos objetivos inicialmente propostos.

A aplicação permite a um grupo reservado de utilizadores com permissão de administradores a adição de novo património/edição e criação de bilhetes para visita ou eventos. Este grupo de utilizadores deve continuamente atualizar a informação dos locais.

Aos visitantes/clientes é lhes permitido registar-se na plataforma, procurar locais/eventos do seu agrado que pretendem adquirir e comprá-los. Os bilhetes comprados devem ficar registados no seu perfil.

O programa de fidelização permite a contabilização dos pontos dos clientes adquiridos na compra de bilhetes e fidelização do mesmo, gestão dos pontos do cliente, este programa através dos pontos fornece ao cliente descontos ou até mesmo entradas gratuitas.

1.3. Organização do documento

O presente documento encontra-se estruturado pelos seguintes tópicos:

- Conceptualização do problema/projeto

  Neste capítulo abordar-se-á a definição das necessidades do projeto bem como a arquitetura conceptual. O desenho da arquitetura contempla de forma sucinta a implementação do projeto.
- Metodologia de operacionalização do trabalho

  O capítulo abordará o método de desenvolvimento assim como todas as especificações técnicas utilizadas.
- Conclusão

  Este último capítulo destacará além da reflexão crítica, o trabalho futuro. Trabalho este que futuramente poderia incrementar positivamente o atual resultado.

1.4. Conceitos

HTML (HyperText Markup Language) é uma linguagem de marcação usada para estruturar o conteúdo de uma página web. Ela define a estrutura lógica e os elementos presentes em um documento, permitindo que os navegadores interpretem e exibam o conteúdo de forma adequada aos usuários.

CSS (Cascading Style Sheets) é uma linguagem de estilo utilizada para descrever a apresentação e o layout de documentos HTML ou XML. Ele permite definir a aparência dos elementos de uma página web, como cores, fontes, tamanhos, posicionamento, espaçamento e outros atributos visuais.

JavaScript é uma linguagem de programação amplamente utilizada para desenvolvimento web. Ela permite adicionar interatividade e comportamento dinâmico às páginas da web. O JavaScript é executado no lado do cliente, ou seja, no navegador do usuário, permitindo a criação de funcionalidades como validação de formulários, manipulação de elementos HTML, interação com APIs, animações, entre outros.

Node.js é um ambiente de tempo de execução de código aberto, baseado na engine V8 do Google Chrome, que permite executar JavaScript no lado do servidor. Ele fornece uma plataforma para criar aplicativos de rede escaláveis e de alto desempenho.

MongoDB é um banco de dados NoSQL orientado a documentos, projetado para armazenar, recuperar e gerenciar dados de forma flexível e escalável. Diferente dos bancos de dados relacionais tradicionais, o MongoDB armazena os dados em documentos no formato BSON (Binary JSON), que é uma representação binária do formato JSON.

Swagger é uma ferramenta de código aberto para projetar, construir, documentar e consumir APIs RESTful. Ele fornece uma maneira fácil e padronizada de descrever, documentar e testar APIs, facilitando a comunicação entre desenvolvedores e permitindo a integração de diferentes sistemas.

Uma **REST API (Application Programming Interface)** é um conjunto de padrões e convenções que permite a comunicação entre sistemas através do protocolo HTTP. Ela define um conjunto de endpoints (URLs) que podem ser acessados para realizar operações sobre os recursos de um sistema.

Uma **API**, por sua vez, é um conjunto de regras e definições que permite que diferentes softwares se comuniquem e interajam uns com os outros. Ela define os métodos e formatos de comunicação que devem ser utilizados para acessar e manipular os recursos disponibilizados pela API.

A principal diferença entre uma REST API e uma API em geral é que a REST API segue os princípios e padrões da arquitetura REST (Representational State Transfer), que define uma abordagem leve e escalável para a criação de serviços web. A REST API utiliza os métodos HTTP (GET, POST, PUT, DELETE, etc.) para manipular os recursos, e os dados são geralmente retornados em formato JSON ou XML.

CORS (Cross-Origin Resource Sharing) é um mecanismo de segurança utilizado pelos navegadores webs para controlar as solicitações de recursos que são feitas entre diferentes origens (domínios, protocolos ou portas).

Angular é uma framework de desenvolvimento de aplicações web de código aberto, desenvolvida e mantida pelo Google. Ela permite a criação de aplicativos web dinâmicos e escaláveis, utilizando a linguagem TypeScript.

TypeScript é uma linguagem de programação desenvolvida pela Microsoft que é uma extensão do JavaScript. Ela adiciona recursos de tipagem estática, classes, interfaces, herança e outros recursos de programação orientada a objetos ao JavaScript tradicional.

O **padrão MVC (Model-View-Controller)** é um padrão de arquitetura de software que separa a aplicação em três componentes principais: o Model (modelo), o View (visualização) e o Controller (controlador). Ele tem como objetivo separar as preocupações e melhorar a organização e a manutenção do código.

Todos os conceitos abordados anteriormente serão utilizados por nós ao longo do desenvolvimento deste projeto.

CAP. II Conceptualização do problema/projeto

04. Requisitos

Para o sucesso dos objetivos devem ser executados um conjunto de requisitos para o correto funcionamento do sistema. Requisitos estes estão divididos pelas entidades que interagem com o mesmo. Recordando existem como entidades:

1. Cliente
2. Administrador (funcionário com role de administrador)

Deste modo cada um deve responder aos requisitos:

1. Cliente
   1.1. Registar
   1.2. Login
   1.3. Compra de bilhetes(comprar/cancelar)
   1.4. Gestão de dados pessoais
2. Administrador
   2.1. Registar
   2.2. Login
   2.3. Gestão de locais/eventos (criar, editar, remover)
   2.4. Gestão de bilhetes
   2.5. Gestão de Sistema de pontos
   2.6. Gestão de utilizadores

05. Arquitetura Conceptual

1ºMilestone

A arquitetura permite ter uma visão prévia de determinado objeto. Para o ser humano idealizar as fisicamente os seus pensamentos recorre ao desenho. E

nada melhor que idealizar o sistema de notificações através de uma arquitetura conceptual. Abaixo apresenta-se a arquitetura do sistema.

Com base nos seguintes models/schemas desenvolvemos todo o nosso projeto:

```
const UserSchema = new Schema({
    name: {
        type: String,
        required: true
    },
    email: {
        type: String,
        required: true,
        unique: true
    },
    password: {
        type: String
    },
    status: {
        type: Boolean,
        required: true,
        default: true
    },
    role: {
        type: String,
        required: true,
        default: 'USER'
    },
    image: {
        type: String,
        default: 'user.png'
    }
});
```

```
const TicketSchema = new Schema({
    user_id: {
        type: String,
        required: true
    },
    event_id: {
        type: String,
        required: true
    },
    status: {
        //ativo- antes de usar
        //cancelado- evento cancelado ou cancelamento do utilizador
        //expirado - não usado
        //usado- depois de valido
        type: String,
        required: true
    },
    price: {
        type: Number,
        required: true
    }
});
```

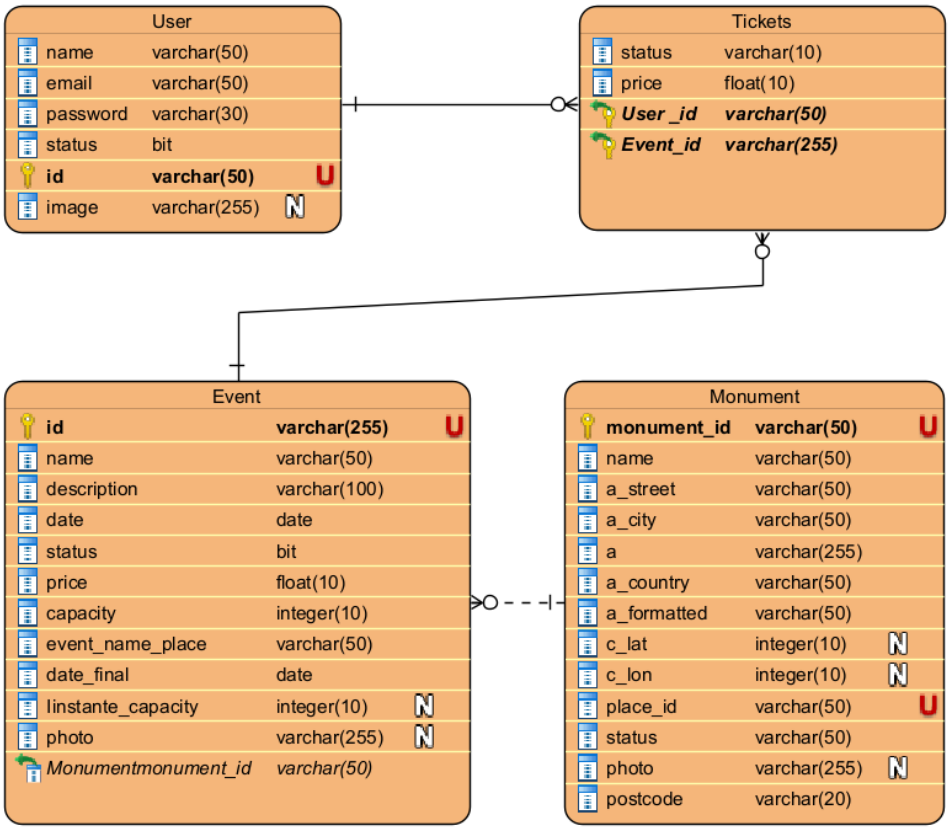
```
const EventSchema = new Schema({
    name: {
        type: String,
        required: true
    },
    description: {
        type: String,
        required: true
    },
    date: {
        type: Date,
        required: false
    },
    date_final: {
        type: Date,
        required: false
    },
    status: {
        type: Boolean,
        default: true
    },
    monument_id: {
        type: String,
        required: true
        //type: Schema.Types.ObjectId,
        //ref: 'Monument'
    },
    price: {
        type: Number,
        required: true
    },
```

```
    price: {
        type: Number,
        required: true
    },
    capacity: {
        type: Number,
        required: true
    },
    id: {
        type: Number,
        required: false
    },
    event_place_name: {
        type: String,
        required: true
    },
    instante_capacity: {
        type: Number,
        required: false
    },
    photo: {
        type: String,
        required: false
    }
});
```

```
const MonumentSchema = new Schema({
    name: {
        type: String,
        required: true,
        unique: true
    },
    address: {
        street: {
            type: String,
            required: true
        },
        city: {
            type: String,
            required: true
        },
        county: {
            type: String,
            required: true
        },
        postcode: {
            type: String,
            required: true
        },
        country: {
            type: String,
            required: true
        },
        formatted: {
            type: String,
            required: true
        }
    },
```

```
    coordinates: {
        lat: {
            type: Number,
            required: true
        },
        lon: {
            type: Number,
            required: true
        }
    },
    place_id: {
        type: String,
        required: true,
        unique: true
    },
    status: {
        type: String,
        required: true
    },
    photo: {
        type: String,
        required: false
    }
});
```

Como referido anteriormente, com base nos models/schemas apresentados construímos uma base de dados em Mongo Atlas para suportar no nosso projeto.



Desenhamos o modelo conceptual da base de dados a perceber de que forma temos a informação organizada e como se relacionaria toda a informações.

Poderíamos ter simplificado a forma como a nossa base de dados está criada para a 1ª *Milestone*, mas rapidamente nos apercebemos que futuramente na 2ª *Milestone*, teríamos bastante trabalho a alterar a forma como esta se encontra organizada.

Monument é os locais que temos falado ao longo do trabalho, a tabela *Event* são os eventos ou visitas associadas que através do *id_monumet* ficam associados a um local. Para um *monument* pode existir 0 ou mais *eventos*. A grande maioria da informação sobre um monument é conseguida através da API *Geoapify.*

A tabela Tickets são os bilhetes que o *user* pode comprar que estão associados a um evento, pode ser uma visita ou evento em determinado local. Para existir ticket tem de existir um evento associada.

O *user* tem de existir para comprar um ticket e pode comprar 0 ou mais.

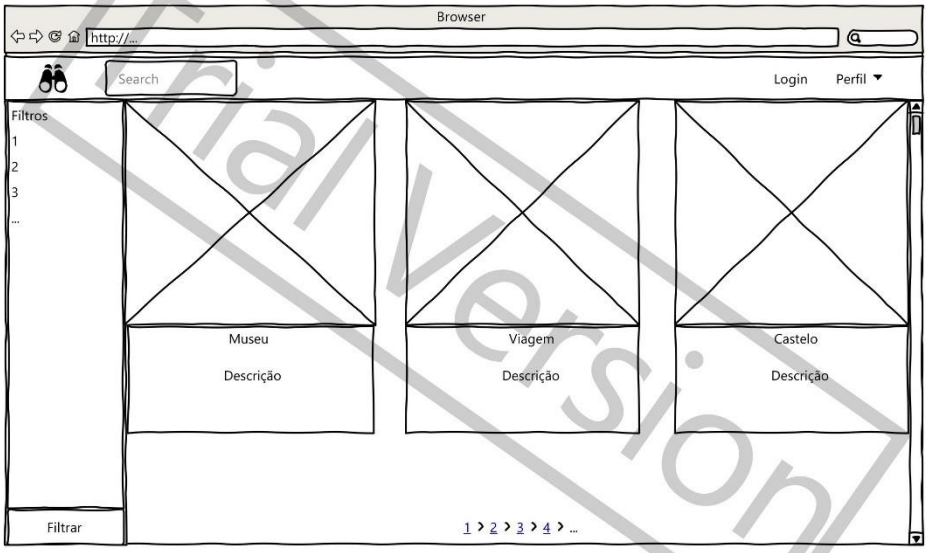


Figura 8 – Home Page BackEnd

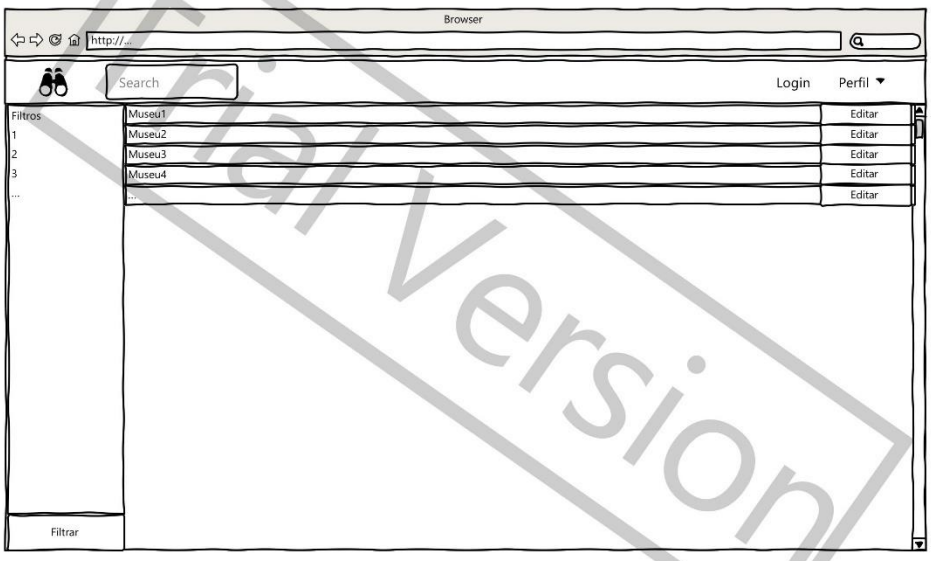


Figura 9 - Gestão dos locais BackEnd

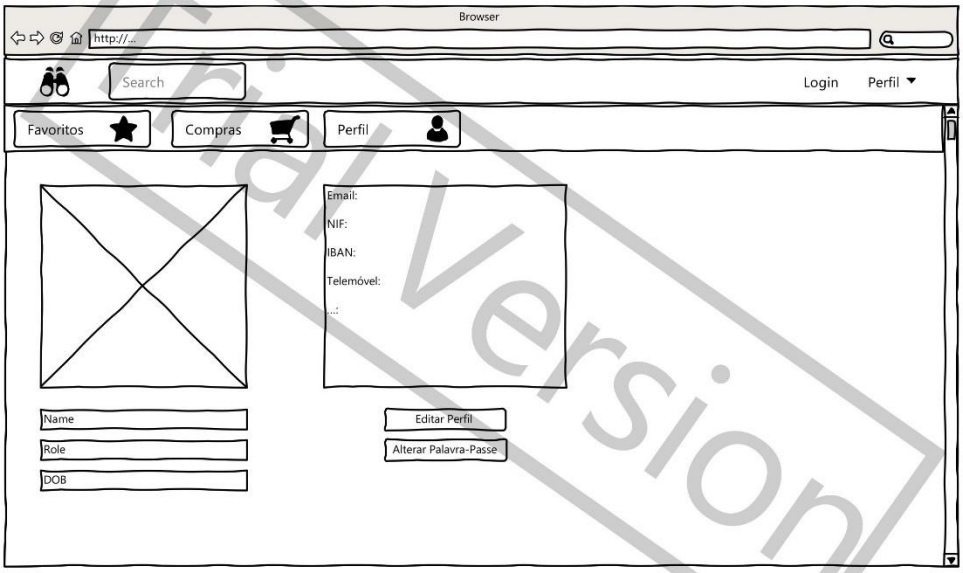


Figura 10–Gestão de Users Backend

Como se pode perceber, na primeira entrega a gestão dos locais, venda de bilhetes e a gestão dos utilizadores são os pontos fulcrais do desenvolvimento da aplicação. Estes mockups ajudaram-nos a perceber o que era necessário desenvolver e, de uma forma muito simples como poderia ficar visualmente a nossa aplicação. Assim, todo o grupo tem um modelo a seguir para todas as páginas web ficarem parecidas.

A gestão de locais, bilhetes e utilizadores está apenas disponível a pessoas com o role de administradores, sendo que esse role irá ser verificado através de um *token/cookie* criado aquando do login.

2ºMilestone

Com base na estrutura de componentes da framework Angular desenvolvemos o seguinte diagrama de forma a ser nos simples a perceção do trabalho a ser realizado nesta segunda entrega.

Este diagrama facilitou-nos a perceber os componentes e serviços que teríamos de criar e como os componentes e serviços se iriam relacionar.

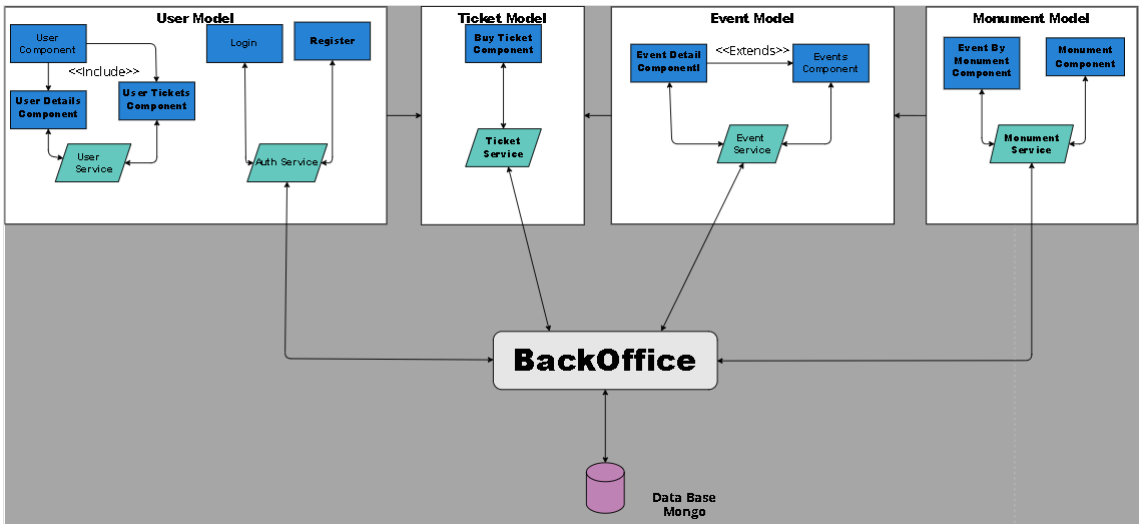


Figura 11 - Diagram Basic Flow

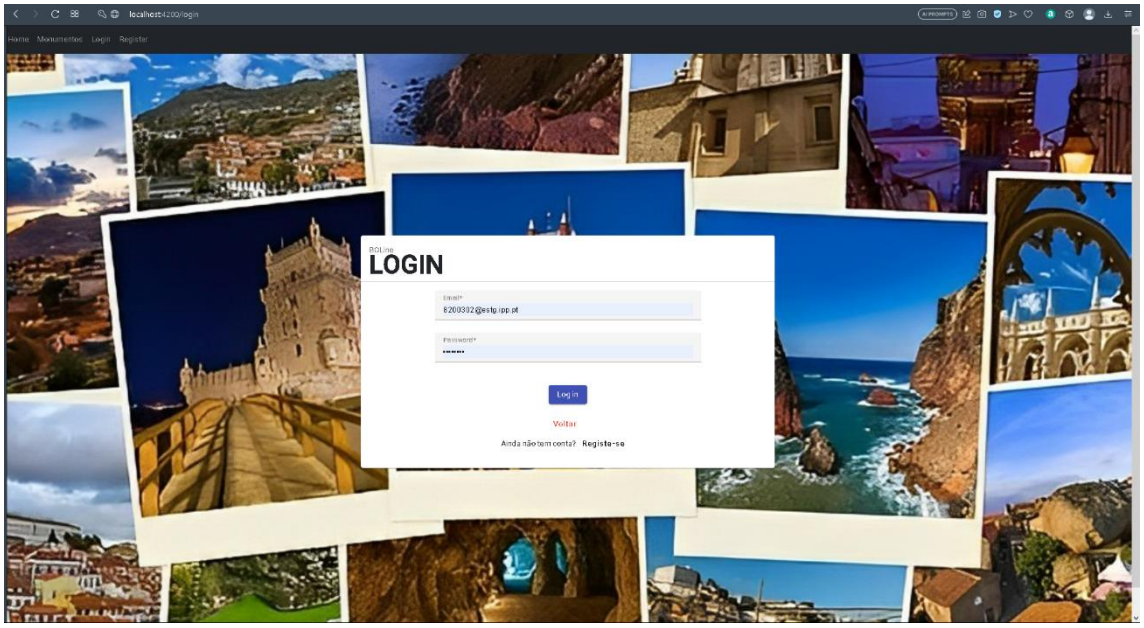


Figura 12- Login FrontOffice

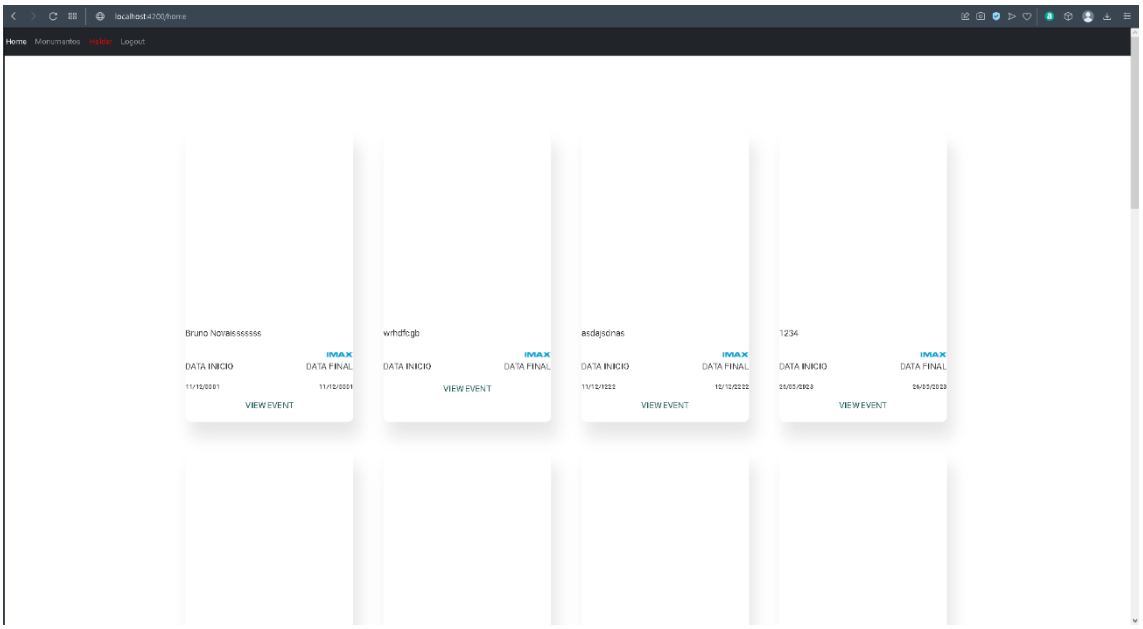


Figura 13- Home Page FrontOffice

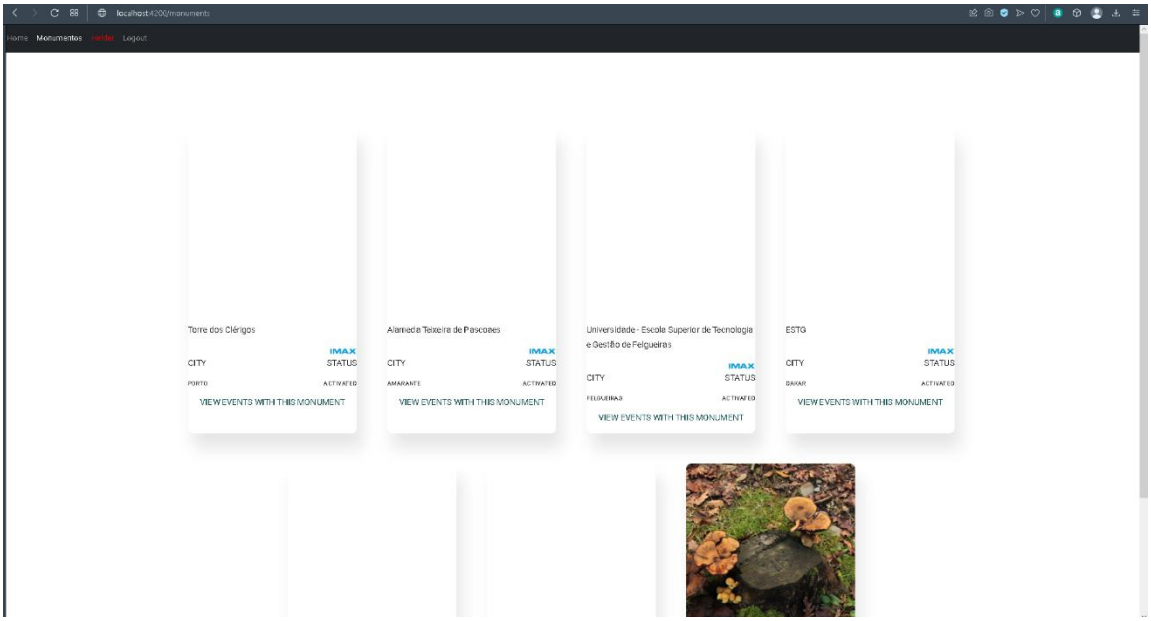


Figura 14- Monumets Page FrontEnd

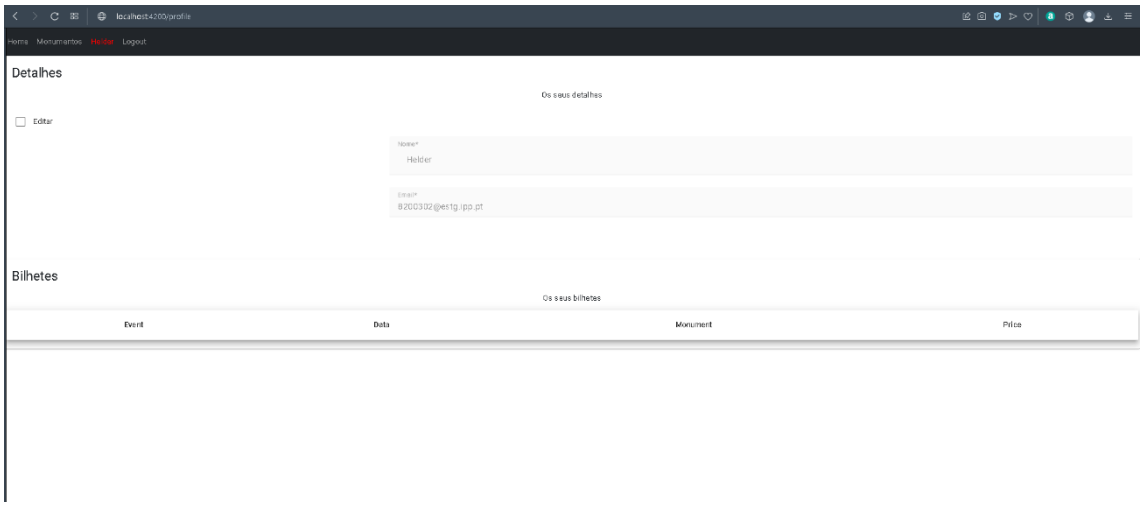


Figura 15- User Profile FrontOffice

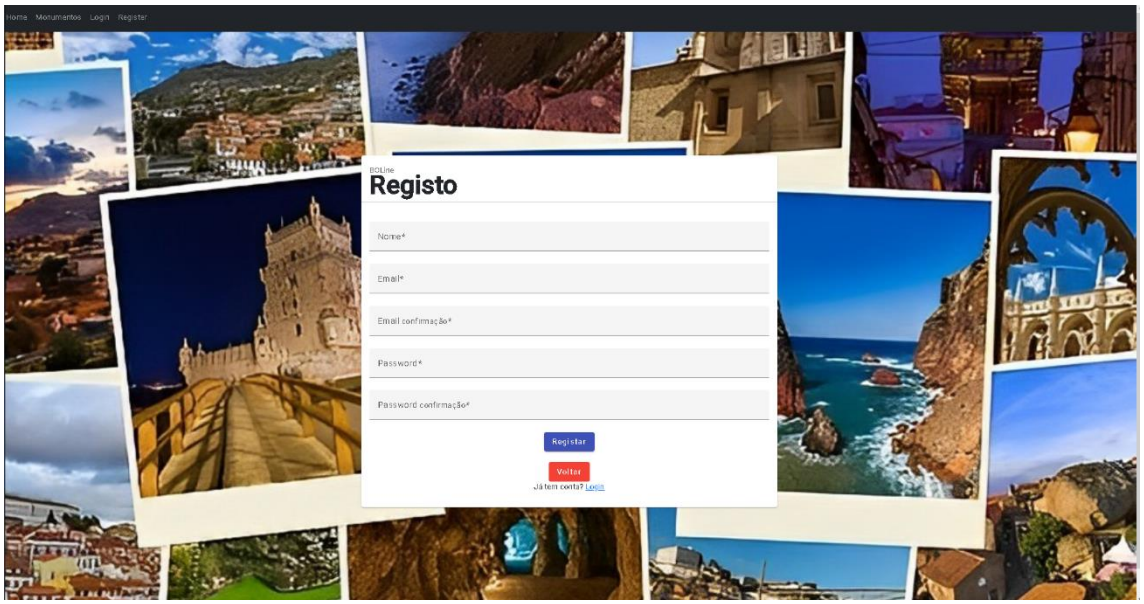


Figura 16 - Register FrontOffice

Após a análise das imagens e mockups apresentados do FrontOffice, fica evidente que essa parte da aplicação é muito mais atrativa em comparação ao backoffice. Isto acontece porque o FrontOffice tem como principal objetivo atrair novos clientes e proporcionar uma experiência agradável para o utilizador comum.

É importante ressaltar que ainda há áreas que neste momento de desenvolvimento do projeto, como as imagens de monumentos e eventos ainda não foram devidamente colocadas, mas até ao final do projeto iremos o fazer.

O FrontOffice desempenha um papel fundamental como a interface principal para os utilizadores comuns. É nessa parte da aplicação que eles podem comprar bilhetes,

visualizar e editar seus perfis, verificar os pontos ganhos e os bilhetes adquiridos, além de explorar a oferta de monumentos e eventos disponíveis para compra no BoLine.

Para os novos utilizadores, há a opção de se registar na plataforma, e os bilhetes previamente adquiridos (em formato físico) podem ser associados à sua conta. A página inicial apresenta uma visão geral de todos os eventos disponíveis no momento, nos quais o utilizador pode comprar bilhetes.

Por meio da barra de navegação, é possível aceder a página de monumentos e escolher eventos específicos em cada monumento. Além disso, o utilizador pode visualizar seu perfil, realizar alterações, verificar seus bilhetes adquiridos e acompanhar seus pontos.

Em resumo, o FrontOffice foi projetado com o objetivo de proporcionar uma experiência agradável e funcional aos clientes, apresentando uma interface intuitiva e atraente. O trabalho em andamento visa aprimorar ainda mais essa parte da aplicação, buscando oferecer aos utilizadores uma experiência completa e satisfatória no uso do BoLine.

CAP. III Metodologia de operacionalização do trabalho

06. Processo e Metodologia de trabalho

Milestone 1:

Durante a primeira milestone, recebemos o problema e iniciamos o nosso projeto ao analisar cuidadosamente o enunciado, compreendendo plenamente as tarefas e objetivos que deveríamos alcançar. Uma das primeiras etapas que abordamos foi a estruturação das informações, onde criamos os modelos/schemas necessários para representar os dados de forma adequada. Esse processo permitiu nos ter uma visão clara da estrutura do projeto antes mesmo de começarmos a desenvolver.

Em seguida, dedicamos esforços ao desenvolvimento do modelo conceitual da base de dados, usando MongoDB como solução de armazenamento. Esta etapa foi crucial para organizarmos e definirmos a forma como as informações seriam armazenadas, permitindo uma recuperação eficiente dos dados no futuro.

Para garantir que todos os membros do grupo estivessem alinhados e seguindo a mesma direção, criamos mockups que serviram como guias visuais do projeto.

Esses mockups nos permitiram visualizar antecipadamente como a aplicação seria apresentada e funcionaria, proporcionando uma base sólida para o desenvolvimento.

Como parte de uma abordagem colaborativa e organizada, escolhemos o GitHub como repositório central para o projeto, permitindo que cada membro contribuísse com seu trabalho e mantivéssemos um histórico de todas as alterações realizadas. Para facilitar ainda mais a gestão do projeto, utilizamos o SourceTree como ferramenta para gerenciar as atualizações e controlar as versões do código.

No processo de desenvolvimento, focamos na implementação de controllers que permitissem interagir com a base de dados e realizar operações CRUD (Create, Read, Update, Delete) de forma eficiente. Criamos métodos para atender às diferentes necessidades, como recuperar informações de um usuário, criar um monumento ou remover um evento específico. Além disso, implementamos mecanismos de validação para garantir a integridade e segurança dos dados, como a verificação de senhas e a existência de usuários.

Um aspecto importante do projeto foi a integração com a API Geoapify. Utilizamos a API para obter informações geográficas relevantes, enriquecendo os dados dos monumentos. Investimos tempo a estudar a documentação da API para compreender como fazer os pedidos corretamente e processar as informações retornadas, de modo a fornecer uma experiência rica aos usuários da aplicação.

Com as bases sólidas estabelecidas, iniciamos o desenvolvimento das páginas web usando HTML e incorporando as informações dinamicamente por meio dos controllers. Para dar um visual atraente e responsivo às páginas, aproveitamos as bibliotecas do Bootstrap, que forneceram recursos de HTML, JavaScript e CSS prontos para uso. Dessa forma, conseguimos criar uma interface amigável e intuitiva para os usuários.

Durante todo o processo de desenvolvimento, encontramos desafios e obstáculos que exigiram correções de erros e ajustes no código. Essas correções

foram importantes para garantir a qualidade e o bom funcionamento da aplicação. Além disso, adotamos o padrão de arquitetura MVC (Model-View-Controller) para separar as responsabilidades das diferentes partes do projeto, tornando-o mais organizado e fácil de manter.

Para facilitar a colaboração e o compartilhamento de conhecimento, utilizamos o Visual Studio Code como ambiente de desenvolvimento principal, oferecendo recursos avançados de edição e depuração de código. Além disso, o Visual Paradigm Code nos auxiliou no desenvolvimento de mockups e na definição do modeloda base de dados, garantindo uma representação visual clara e coerente do projeto.

Milestone 2:

Após recebermos a nota da primeira entrega e recebermos o feedback dos professores, aproveitamos essa oportunidade para aprimorar ainda mais o projeto e corrigir aspectos que precisavam de melhorias. Essa etapa foi fundamental para fortalecer a base já estabelecida e preparar o terreno para a segunda entrega.

Dedicamos esforços para adicionar verificações de token, a fim de restringir o acesso a determinadas ferramentas e funcionalidades apenas a usuários com função de admin. Isto trouxe uma camada adicional de segurança ao sistema, protegendo os dados sensíveis e garantindo que apenas usuários autorizados pudessem realizar determinadas ações.

Uma das melhorias implementadas foi a funcionalidade de edição, onde os campos agora são preenchidos automaticamente com os dados anteriores, facilitando a atualização de informações e proporcionando uma experiência mais fluida aos usuários.

Outra melhoria importante foi permitir que os usuários pudessem comprar mais de um bilhete, ampliando as possibilidades de participação nos eventos

oferecidos pela plataforma. Essa funcionalidade trouxe mais flexibilidade e comodidade aos usuários, atendendo às suas necessidades individuais.

Além disso, dedicamos tempo a melhorar o tratamento de erros em toda a aplicação, garantindo que mensagens claras e informativas fossem exibidas aos usuários em caso de falhas ou problemas inesperados. Esta abordagem contribuiu para uma experiência de usuário mais amigável e transparente.

Durante a revisão do código, eliminamos código comentado, garantindo uma base de código mais limpa e legível. Essa prática é importante para manter a qualidade do código e facilitar a compreensão e manutenção futura.

Como parte do relatório, refizemos o modelo conceitual do projeto para que correspondesse fielmente à estrutura da base de dados real. Essa correção garantiu que as informações apresentadas no relatório estivessem alinhadas com a implementação real do sistema.

Para melhorar a gestão do projeto, adicionamos tempos estimados para o desenvolvimento de cada tarefa e funcionalidade. Essa prática nos ajudou a planejar melhor as atividades, distribuir as tarefas de forma equilibrada e cumprir os prazos estabelecidos.

Com todos esses aspectos aprimorados e concluídos, concentramo-nos em criar o projeto FrontEnd, desenvolvendo uma Rest API e suas rotas no backend para fornecer os dados necessários ao frontend. Além disso, configuramos o CORS (Cross-origin resource sharing) para garantir que as requisições do frontend fossem permitidas e processadas corretamente.

Após a configuração da comunicação entre o frontend e o backend, criamos os serviços e componentes necessários para a interface do usuário. Estes elementos foram desenvolvidos com atenção aos detalhes, levando em consideração aspectos como o sistema de pontos, métodos de pagamento e a experiência do usuário como um todo.

Ao nos aproximarmos da conclusão do projeto, dedicamos tempo para testar e resolver qualquer problema identificado. O nosso objetivo era entregar uma aplicação estável e livre de erros, proporcionando aos usuários a melhor experiência possível.

Assim como na primeira milestone, enfrentamos desafios ao longo do desenvolvimento que exigiram correções e adaptações.

Para a segunda milestone, utilizamos a framework Angular como base para o desenvolvimento, complementada pela biblioteca Bootstrap, que proporcionou um design responsivo e moderno à aplicação. O padrão MVC continuou sendo seguido, garantindo uma estrutura organizada e modular para o projeto.

Novamente, o GitHub e o SourceTree foram essenciais como repositório e ferramenta de organização do código-fonte, permitindo um controle eficiente das versões e facilitando a colaboração em equipe.

O Visual Studio Code foi mantido como ambiente de desenvolvimento principal, proporcionando recursos avançados e uma experiência de codificação eficiente. O Visual Paradigm Code continuou a auxiliar no desenvolvimento de mockups e na definição do modelo de banco de dados, mantendo uma abordagem visual para o projeto.

CAP. IV Desenvolvimento da solução

De forma a responder aos requisitos enumerados em Requisitos:

1. Cliente

Registar

Começamos por preencher os campos existentes no formulário que aparece na página, caso algum deles não esteja devidamente preenchido é enviado um aviso a dizer que falta preencher certo campo ou que está preenchido incorretamente.

Após todos os campos estarem devidamente preenchidos é enviado um pedido POST à base de dados, caso os dados enviados já existiam é enviado um alerta a dizer que este utilizador já existe.
Se ainda não existir, este utilizador é adicionado com sucesso e já pode fazer login na nossa aplicação web.
Aos clientes é atribuído por definição o role de cliente.
Caso o cliente por algum motivo não se queira registar na nossa aplicação, definimos um utilizador *default* para resolver estes casos. Todas as compras feitas através deste ficarão associadas ao mesmo.

Login

No login verificamos se os dados inseridos correspondem a algum utilizador que já temos na nossa base de dados, se isto for verdade é criado um token nas cookies com uso do JWT.
Se o utilizador não estiver registado surge um alerta que o utilizador não existe e deve ser registado.

Compra de bilhetes

Compra/Seleção

A gestão de bilhetes da parte do cliente passa por poder comprar bilhetes e cancelar os mesmos caso já não queira usufruir.
O cliente deve procurar na página o local/evento que quer visitar e se ainda houver bilhetes disponíveis compra os que desejar e estiverem disponíveis, pode também filtrar estes pela cidade onde deseja visitar certo monumento ou participar em algum evento.
Estes irão aparecer numa aba do seu perfil onde estarão todos os bilhetes que comprará até ao dia de hoje.

Pagamento

O cliente terá à sua disposição vários métodos de pagamento como por exemplo PayPal e cartão bancário, deve selecionar um método e

preencher os dados necessários para efetuar o pagamento. Caso este não se realize a compra fica sem efeito.

Através do sistema de pontos o cliente poderá receber alguns pontos em cada compra ou até mesmo comprar bilhetes isto se o bilhete não for superior a 50€ e tiver no mínimo 50 pontos (cada bilhete com custo <50€, pode ser comprado com 50 pontos).

Cancelar

Ao cancelar os bilhetes, poderá ou não ser reembolsado ficando a critério do administrador do local que queria visitar e são disponibilizados novamente este número de bilhetes cancelados para compra. Estes continuaram a aparecer no seu perfil só que com o status de cancelado.

Gestão de dados pessoais

O cliente pode visitar o seu perfil e alterar os seus dados pessoais tais como, nome, email, password, etc. Consegue também ver os bilhetes que já comprou através da nossa plataforma e os pontos que tem disponíveis no momento.

2. Administrador

Registar

Começamos por preencher os campos existentes no formulário que aparece na página, caso algum deles não esteja devidamente preenchido é enviado um aviso a dizer que falta preencher certo campo ou que está preenchido incorretamente.

Após todos os campos estarem devidamente preenchidos, é enviado um pedido POST à base de dados. Caso os dados enviados já existam, é enviado um alerta a dizer que este utilizador já existe.

Se ainda não existir, este utilizador é adicionado com sucesso e já pode fazer login na nossa aplicação web.

Login

No login verificamos se os dados inseridos correspondem a algum utilizador que já temos na nossa base de dados, se isto for verdade é criado um *token* nas *cookies* com o JWT.
Se o utilizador não estiver registado surge um alerta que o utilizador não existe e deve ser registado.

Gestão de Locais/eventos

Adicionar

Através do nome de determinado local que o administrador deseja inserir na aplicação web faz um pedido GET à API *Geoapify* para receber metadados do local.
Caso queira adicionar um evento deve primeiro criar o local e depois definir dados relativos ao evento como por exemplo data e horas, tema, etc. Ao criar o evento, associa este evento ao local que pretende.

Editar

O administrador pode também editar locais ou eventos. Ao editar irão aparecer os dados precedentes e poderá atualizar os dados que desejar.

Eliminar

O mesmo eliminar locais ou eventos. Ao eliminar todos os tickets associados a um destes devem também deixar de estar disponíveis para compra.

Gestão de Bilhetes

Adicionar

O administrador de determinado local tem a funcionalidade de criar bilhetes sobre determinado local(visita) ou evento.

Deve definir um custo para o bilhete, um número máximo de bilhetes (capacidade do local) e caso seja um evento ou visitas com horário marcado definir os horários.

Editar

O administrador de determinado local tem a funcionalidade de editar bilhetes sobre determinado local(visita) ou evento. Tal como na edição de eventos/monumentos este funciona da mesma forma e é o mesmo processo.

Gestão de Sistema de pontos

O sistema de pontos funciona da seguinte forma, por cada compra que o cliente faz recebe um ponto por cada euro gasto. Com a quantia de 50 pontos pode comprar um bilhete usando os mesmos.
O cliente pode visualizar os seus pontos no perfil.

Gestão de utilizadores

O administrador pode criar ou eliminar utilizadores da plataforma.
Pode criar um utilizador por exemplo, numa compra física, com o intuito de posteriormente quando o utilizador se registar na plataforma irão surgir os bilhetes que este comprou ou então um novo funcionário.
Não há grande interesse em o administrador eliminar um utilizador, ao não ser que aconteça algum erro ou algum funcionário seja despedido.

Tempo estimado para o desenvolvimento

Funcionalidade	Tempo Estimado (dias)

Tabela 1 – Tempo Esperado de Implementação das funcionalidades

CAP. V Conclusão

07. Reflexão crítica dos resultados

Em síntese, não só do projeto, mas também de tudo o que anteriormente foi abordado, apresenta-se abaixo a listagem das funcionalidades do projeto e a respetiva indicação se foram ou não implementadas. A informação está disposta na tabela que contém a Funcionalidade e o Estado da funcionalidade. O Estado assume os valores:

- ☑, caso a funcionalidade esteja implementada
- ☒, caso a funcionalidade não esteja implementada

Funcionalidade	Estado

Login	☑
Registo de funcionários (Administradores)	☑
Gestão de locais e eventos	☑
Gestão de bilhetes	☑
Registo de clientes	☑
Sistemas de Pontos	☑
Venda de bilhetes	☑
Pagamento	☑
Filtrar(eventos/monumentos) por cidade	☑
Gestão de dados pessoais	☑

Tabela 2 - Estado de implementação das funcionalidades

08. Conclusão e trabalho futuro

Conseguimos alcançar os objetivos propostos para o projeto, implementando todos os requisitos solicitados de acordo com as especificações.

Como trabalho futuro para melhorar o projeto desenvolvido, identificamos algumas áreas que poderiam ser aprimoradas. Em particular, destacamos a melhorias da interface tanto no backoffice quanto no frontoffice, com o objetivo de proporcionar uma experiência de usuário mais agradável. Isso incluiria a atenção aos detalhes, aprimoramento da navegação e a adição de recursos visuais que tornem a aplicação mais atrativa e intuitiva.

Além disso, sugerimos a implementação de mais filtros para facilitar a procura e a organização dos dados, tornando a experiência do usuário mais eficiente e personalizada. Também propomos melhorias no sistema de pontos, explorando formas de incentivar e recompensar os usuários, assim como a inclusão de métodos de pagamento adicionais para oferecer mais opções aos usuários.

Em resumo, o trabalho desenvolvido atende satisfatoriamente aos objetivos e requisitos propostos. No entanto, reconhecemos que ainda há espaço para aprimoramentos e evolução, a fim de tornar a aplicação web ainda mais desejada e relevante no mundo dinâmico, personalizado e social em que vivemos.

09. WebGrafia

- W3School
- CORS
- CORS(1)
- Swagger
- Angular
- Node js
- BootStrap
- Moodle da UC

Grupo 7, composto por Bruno Novais, estudante n.º 8210333, Hélder Branco, estudante n.º 8200302 e Daniel Teixeira, estudante n.º 8200378, do curso Licenciatura em Engenharia Informática da Escola Superior de Tecnologia e Gestão do Instituto Politécnico do Porto, declara que não fez plágio nem auto plágio, pelo que o trabalho intitulado “BoLine” é original e da sua autoria, não tendo sido usado previamente para qualquer outro fim.

Felgueiras, 02 de maio de 2023

CÓDIGO DE BOAS PRÁTICAS E DE CONDUTA

REGULAMENTO P.PORTO/P-005/2020

ARTIGO 8.º - DECLRAÇÃO DE COMPROMISSO

DESPACHO P.PORTO/P-040/2020 (ADAPTADO)