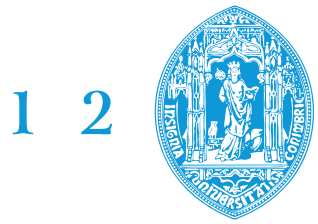
UNIVERSIDADE D
COIMBRA

Francisco Pires Moreira Ribeiro

SALA-Z: SISTEMA DE BILHÉTICA COM NFTS E GESTÃO DE PAGAMENTO

Dissertação no âmbito do Mestrado em Engenharia Informática, especialização em Engenharia de Software, orientada pelo Professor Doutor Paulo Alexandre Ferreira Simões e pelo Eng. Manuel António Cunha Nunes, e submetida à Faculdade de Ciências e Tecnologia / Departamento de Engenharia Informática.

julho de 2022



FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE D
COIMBRA

DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

Francisco Pires Moreira Ribeiro

Sala-Z: Sistema de Bilhética com NFTs e Gestão de Pagamento

Relatório de estágio no âmbito do Mestrado em Engenharia Informática, especialização em Engenharia de Software orientada pelo Professor Doutor Paulo Alexandre Ferreira Simões e pelo Eng. Manuel António Cunha Nunes, e submetida à Faculdade de Ciências e Tecnologia / Departamento de Engenharia Informática.

julho de 2022

Agradecimentos

Em primeiro lugar, quero agradecer aos meus pais que me suportaram em tudo o que fosse preciso e me disponibilizaram os meios para realizar este percurso académico da maneira mais confortável possível.

Aos meus avós pelo enorme apoio que forneceram e por me manterem vivo e saudável através dos inúmeros almoços e jantares.

À minha irmã por ter sido uma excelente colega de casa e por dar o exemplo do que é ser um estudante exemplar.

De seguida, quero agradecer aos meus orientadores de estágio Manuel Nunes e Paulo Simões por todo o apoio que forneceram durante o estágio. Obrigado também a todos os elementos da Grama por me acolherem de uma maneira impecável dentro da empresa e por todo o apoio que deram, o que fez com que este relatório tenha sido bem sucedido.

Por fim, quero agradecer a todos os meus amigos que me acompanharam durante todo este percurso.

Resumo

A empresa Grama apostou na Sala-Z, uma plataforma que liga os artistas e gestores de salas de espetáculos de modo a que estes tenham um espaço onde facilmente haja comunicação e se facilite todo o processo complexo de realizar um evento. O objetivo deste estágio é integrar nesta dita plataforma um elemento essencial do processo: o espectador do evento. Para este efeito, o aluno ficou encarregue de desenvolver um sistema de bilhética para a plataforma suportar venda de bilhetes para os eventos. Para que a plataforma aproveite o avanço da tecnologia desta área, este sistema suportará bilhetes em *NFTs* de modo a garantir autenticidade e transparência, destacando-se assim dos seus competidores. *Blockchain* é uma tecnologia recente e promissora no ambiente tecnológico atual e os objetivos para esta nova plataforma passam por tirar partido das vantagens que esta oferece. Ao utilizar *NFTs* como bilhetes num sistema de bilhética, o utilizador consegue obter a transparência e segurança que as plataformas de bilhética tradicionais simplesmente não conseguem oferecer, tornando assim todo o processo de compra de bilhetes mais robusto. Para o lado do vendedor, problemas como controlo de mercado secundário também é resolvido de uma maneira simples e eficaz.

Neste relatório é exposto todo o processo de estudo prévio das tecnologias envolvidas, análise de requisitos, conceção da arquitetura, desenvolvimento do sistema de bilhética da plataforma Sala-Z e validação do mesmo de acordo com as boas práticas de Engenharia de Software.

Palavras-Chave

Sistema de bilhética, *Web application*, AWS, *Blockchain*, *Non Fungible Tokens*, *Smart Contracts*

Conteúdo

Acronyms

A2A *Application-to-Application*.

A2P *Application-to-Person*.

AWS *Amazon Web Services*.

AZ *Availability Zones*.

CLI *Command Line Interface*.

CRUD *Create Read Update and Delete*.

DRY *Don't Repeat Yourself*.

DTO *Data Transfer Objects*.

EC2 *Elastic Compute Cloud*.

EJB *Enterprise Java Beans*.

ELB *Elastic Load Balancing*.

ER Entidade-Relacionamento.

ERC *Ethereum Request for Comments*.

ES *Epic Stories*.

EVM *Ethereum Virtual Machine*.

GDPR *General Data Protection Regulation*.

ICO *Initial Coin Offering*.

JWT *JSON Web Tokens*.

LLVM *Low Level Virtual Machine*.

NFT *Non-Fungible Token*.

PII *Personally Identifiable Information*.

RDS *Relational Database Service*.

S3 *Simple Storage Service*.

SCP *Stellar Consensus Protocol*.

SES *Simple Email Service*.

SNS *Simple Notification Service*.

SQS *Simple Queue Service*.

TI Tecnologia de Informação.

TPS *Transactions per second*.

UML *Unified Modeling Language*.

XML *Extensible Markup Language*.

XSS *Cross Site Scripting*.

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

O presente relatório foi desenvolvido no âmbito do estágio curricular do Mestrado de Engenharia Informática com especialização em Engenharia de *Software* da Faculdade de Ciências e Tecnologia da Universidade de Coimbra no ano letivo 2021/2022.

Este estágio foi acompanhado pelo professor Paulo Alexandre Ferreira Simões do departamento de Engenharia Informática da Universidade de Coimbra e pelo engenheiro Manuel António Cunha Nunes, da parte da empresa.

1.1 Enquadramento e Motivação

O estágio proporcionado ao aluno foi atribuído pela empresa Grama. A Grama foi fundada em 2017, sendo uma empresa de desenvolvimento de *software* sediada em Coimbra. Procura atuar em todo o ciclo de desenvolvimento de *software*, utilizando as mais recentes tendências tecnológicas da indústria, assegurando qualidade do *software* através da combinação de testes automáticos e manuais. A equipa de fundadores apresenta uma vasta experiência no desenvolvimento e implementação de projetos complexos de Tecnologia de Informação (TI) para alguns dos maiores operadores de telecomunicações da Europa e da América do Sul.

Foi proposto pela empresa que o aluno implementasse uma interface de venda de bilhetes e toda a gestão de pagamento na aplicação interna Sala-Z. A Sala-Z é uma plataforma que permite ligar gestores de salas de espetáculo a artistas e promotores de eventos, sendo uma ferramenta de apoio à gestão das salas de espetáculos e dos seus eventos. Além das duas tarefas mencionadas, o aluno também vai oferecer garantia de autenticidade dos bilhetes através de *NFTs* numa rede *blockchain*.

Este sistema de bilhética vai permitir monetização na plataforma, algo que dantes só poderia ser feito através de publicidade. Além disso, com a nova tecnologia *blockchain* introduzida na plataforma, o utilizador poderá usufruir de um novo grau de segurança e transparência previamente inexistente. O facto de tecnolo-

gia *blockchain* oferecer armazenamento de dados imutáveis e acessíveis a todos, problemas como confiança no vendedor e revenda de bilhetes em mercados secundários a preços exorbitantes poderão ser controlados facilmente.

1.2 Objetivo

Os objetivos para este estágio passam assim por todo o planeamento, arquitetura e implementação da solução de sistema de bilhética da aplicação Sala-Z. É necessário fazer um estudo prévio do estado da tecnologia, escolher as melhores tecnologias de *back end* e *front end* tendo em conta se é adequado ou não no contexto da aplicação existente e perceber como vai ser integrada a componente *blockchain*. Por fim, vão ser aplicados os testes necessários para garantir o bom funcionamento das funcionalidades implementadas.

1.3 Estrutura do Documento

O presente documento está estruturado em 8 capítulos, organizados como se descreve de seguida.

No **Capítulo 1** é feita uma introdução geral do documento, apresentando o enquadramento, a motivação, e os objetivos do estágio.

No **Capítulo 2** é descrita a metodologia de trabalho aplicada durante o estágio, incluindo planeamento através de diagramas de Gantt. Também são discutidos os potenciais riscos associados a este estágio.

No **Capítulo 3** é apresentado o estado da arte, sendo estudadas as possíveis tecnologias a utilizar no desenvolvimento do projeto. Além das tecnologias, é também feita uma introdução ao estado atual da plataforma Sala-Z, uma análise da concorrência à plataforma e apresentado o processo de seleção relativo às tecnologias a adotar para desenvolvimento da plataforma.

No **Capítulo 4** é apresentada a análise de requisitos da plataforma. São apresentadas as *user stories* feitas a partir de *epic stories* fornecidas ao aluno, que são depois transformadas em requisitos funcionais e, por fim, em diagramas de *use cases*.

No **Capítulo 5** é discutida a arquitetura planeada para o projeto. São definidos e apresentados os vários diagramas de arquitetura de acordo com o modelo C4, bem como uma planificação da estrutura de dados da plataforma.

No **Capítulo 6** é descrito o desenvolvimento do projeto durante o segundo semestre. Aborda as diferentes alterações ocorridas, um resumo do trabalho realizado e os riscos que se materializaram.

No **Capítulo 7** é apresentada a metodologia de testes aplicada pelo aluno de modo a testar e assegurar que os requisitos levantados foram concluídos.

Por fim, no **Capítulo 8** é feita a conclusão do documento, salientando de novo a proposta de estágio, o planeamento, os principais obstáculos durante o desenvolvimento e uma secção de trabalho futuro na plataforma.

Capítulo 2

Metodologia e Planeamento

Antes de se desenvolver um produto, é necessário a equipa formular um plano de desenvolvimento para que seja possível acompanhar e saber o seu estado atual. Este planeamento é essencial para determinar o sucesso do produto, bem como identificar mais facilmente os custos e recursos necessários ao seu desenvolvimento.

Para tal, esta secção tem como objetivo definir este plano e mencionar todas as ferramentas necessárias para uma boa execução do mesmo, bem como uma análise dos riscos do projeto e do desenvolvimento do mesmo.

2.1 Metodologia

Para o desenvolvimento deste trabalho vai ser utilizada uma metodologia *agile*. A metodologia *agile* caracteriza-se por dividir cada projeto em pequenas partes que devem ser completas num curto espaço de tempo definido previamente. Assim, torna-se mais simples compreender prioridades e introduzir mudanças no desenvolvimento do projeto, caso seja necessário. Existem quatro valores que definem bem a filosofia da metodologia *agile*, que são [30]:

- Valorizar mais os indivíduos e interações do que processos e ferramentas.
- Valorizar mais software funcional do que documentação abrangente.
- Valorizar mais a colaboração com o cliente do que negociação de contratos.
- Valorizar mais a resposta a mudanças do que seguir estritamente os planos.

2.1.1 Scrum

A equipa optou por uma metodologia *agile* baseada em *scrum*. Esta metodologia adiciona mais três valores aos valores definidos anteriormente [30]:

- **Transparência**: Todos os elementos da equipa têm acesso a toda a informação do produto.
- **Inspeção**: Validações regulares são essenciais no decorrer do projeto.
- **Desvios**: Implementação de novas medidas são necessárias quando são detetados desvios no projeto através das inspeções.

Scrum é estruturado em vários períodos temporais, normalmente de 2 a 4 semanas, nos quais a equipa desenvolve um número específico de tarefas do produto. Estes períodos chamam-se ***Sprints***. A lista de tarefas para cada *sprint* denomina-se de ***Backlog*** e é gerida pelo *product owner*, que é um dos papeis nesta metodologia. No final de cada *Sprint*, é produzida uma nova versão do produto com as novas funcionalidades implementadas, designada por ***release***.

Para isto ser possível, três **papeis** têm que ser distribuídos e cada papel fica responsável por certas tarefas na gestão do desenvolvimento:

- ***Scrum Master***: é o líder da equipa e ajuda a equipa com dificuldades com o desenvolvimento das tarefas do *backlog*.
- ***Product Owner***: responsável pela junção da área de negócio com a área de técnica. É o criador das *user stories* e gestor do *Backlog*.
- ***Developer***: elemento da equipa responsável por desenvolver as tarefas do *Backlog*.

No contexto do estágio, o aluno recebeu uma lista com *Epic Stories* que teve de transformar em *User Stories* e identificar os requisitos associados (*User Stories* e *Epic Stories* são definidas melhor no Capítulo 4). Visto não ter feito as *Epics*, não se classifica como *Product Owner*. O aluno enquadra-se como *developer* tendo também na equipa um *designer* para a realização dos *mockups* referentes às funcionalidades a implementar.

Para além dos papeis, numa metodologia *scrum* devem ser consideradas três cerimónias:

- ***Sprint Planning***: reunião no início de cada *Sprint* onde se discutem as tarefas a implementar nesse *Sprint*. Devem estar presentes a equipa de desenvolvimento, o *Scrum Master* e o *Product Owner*.
- ***Daily Scrum***: reunião diária para perceber o progresso do *Sprint*. É aqui que se discutem potenciais problemas encontrados e se comunica o ponto de situação de cada elemento da equipa de desenvolvimento.
- ***Sprint Review***: reunião no final de cada *Sprint* para avaliar o resultado e apresentá-lo aos *stakeholders*. Tal como no *Sprint Planning*, todos os elementos devem estar presentes.

Durante o primeiro semestre, o aluno não adotou um estado formal desta metodologia, visto ser um trabalho mais orientado à pesquisa e escrita do presente documento. Contudo, para ser acompanhado pela empresa, o aluno teve reuniões semanais de acompanhamento que serviram para apresentar o trabalho da semana e comunicar eventuais dificuldades.

2.1.2 Ferramentas e Comunicação

De modo a haver organização na equipa são necessários meios de comunicação eficazes bem como ferramentas úteis para cada tarefa. Deste modo, várias ferramentas foram utilizadas durante o primeiro semestre:

- **Discord**: Ferramenta de comunicação que disponibiliza vários serviços como salas digitais de texto, chamadas ou vídeo chamadas com possibilidade de partilha de ecrã. Esta ferramenta foi utilizada para as reuniões semanais do aluno com a empresa.
- **Slack**: Também uma ferramenta de comunicação mais orientada para planeamento de eventos dentro da empresa.
- **Linear**: Ferramenta de gestão de projeto que representa o *Backlog* de *Scrum*. Foi utilizada pelo aluno no início do segundo semestre para ajudar na gestão das funcionalidades pedidas.
- **Webmail**: Cliente de email para troca de documentos e marcação de reuniões.

2.2 Planeamento

Esta secção foi dividida entre o planeamento do primeiro semestre e o planeamento do segundo semestre. Vão ser expostos cada planeamento e o decorrer real de cada um, completando com uma análise.

2.2.1 Primeiro Semestre

Para o primeiro semestre foi criado um diagrama de Gantt (ver Figura 2.1) para estruturar o trabalho e definir datas para cada tarefa prevista no plano de trabalhos. Em adição, para haver uma comparação entre o planeamento e o decorrer real do semestre, foi elaborado outro diagrama de Gantt (ver Figura 2.2) com os tempos reais de cada etapa - de modo a detetar desvios significativos e implementar, se necessário, medidas corretivas. As barras a azul indicam o tempo planeado enquanto que as barras verdes indicam o tempo real decorrido para a tarefa.

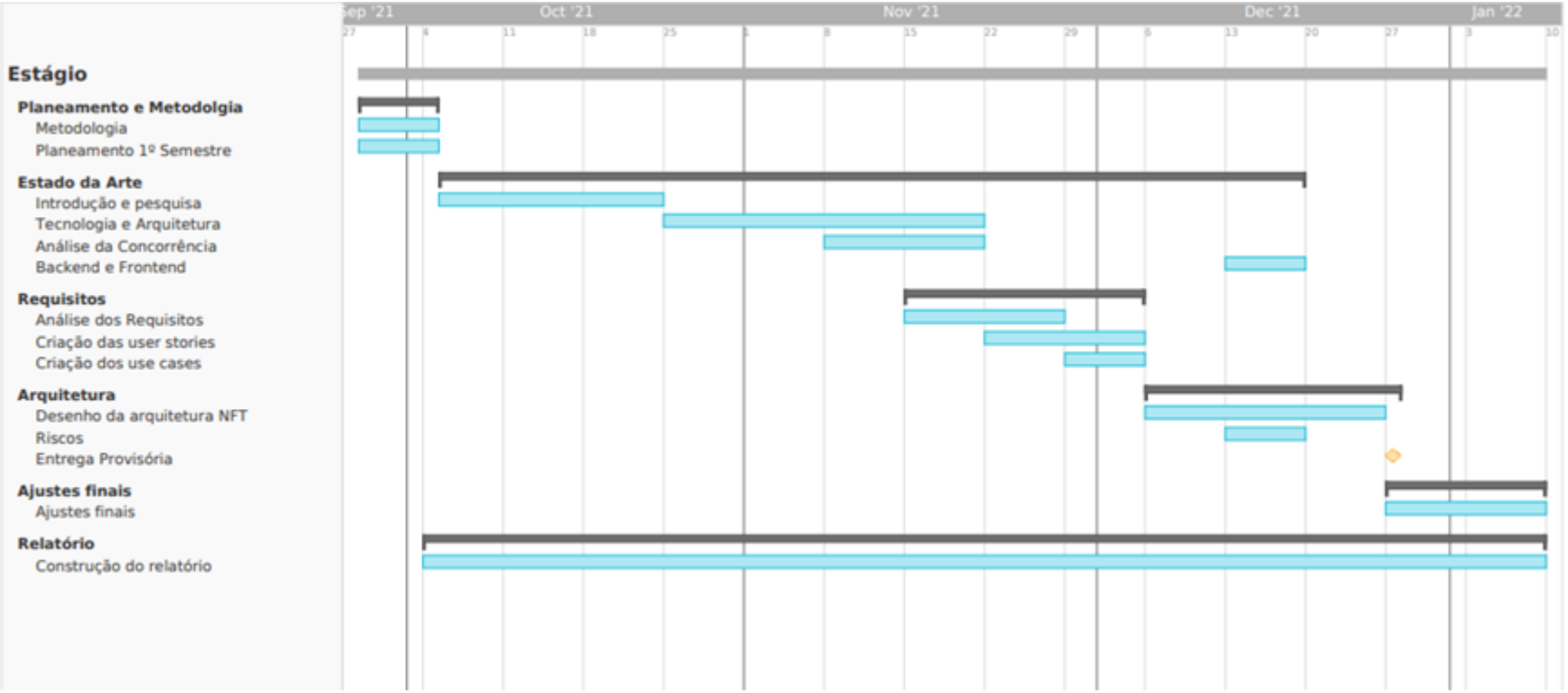


Figura 2.1: Diagrama de Gantt para o planeamento inicial do primeiro semestre

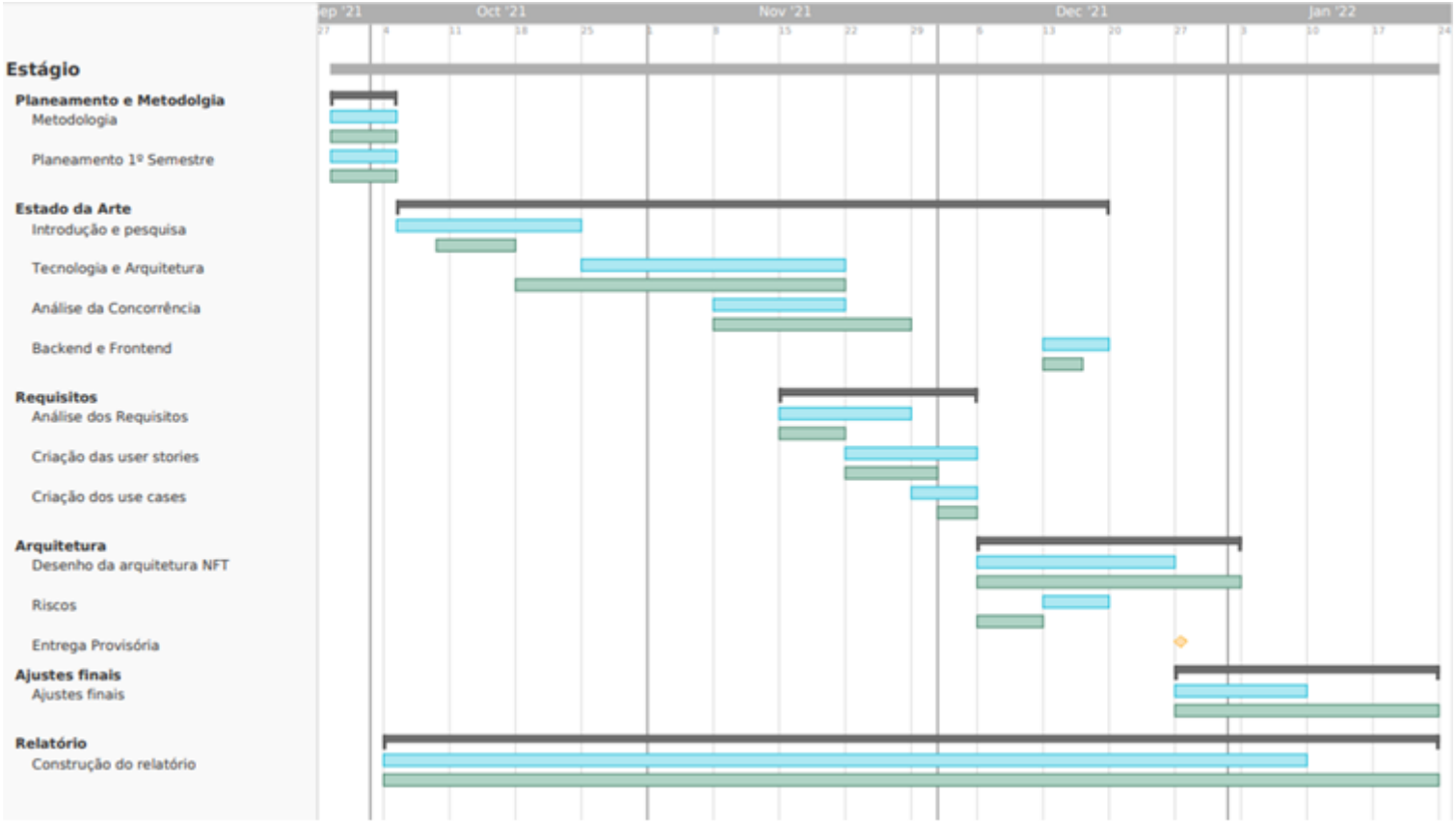


Figura 2.2: Diagrama de Gantt para o planeamento inicial do primeiro semestre (Real)

É possível identificar alguns desvios ao planeamento inicial, nomeadamente nas tarefas de tecnologia e arquitetura, que demoraram mais tempo do que o suposto. Isto deveu-se ao facto de haver uma grande quantidade de informação que teve de ser analisada pelo aluno e por este não ter muitos conhecimentos prévios na área, atrasou o processo de seleção. Contudo, o prazo final do planeamento foi cumprido apenas com ligeiros atrasos.

2.2.2 Segundo Semestre

O aluno elaborou um diagrama de Gantt de modo a planificar o semestre que é apresentado na Figura 2.3. Neste constam uma fase inicial do desenvolvimento, onde o aluno explorou a plataforma resolvendo *bugs* de modo a acostumar-se ao código. De seguida foram planeados 9 *sprints* para o desenvolvimento, tendo cada um uma duração de 2 semanas. Devido ao estado inicial do projeto, a equipa, juntamente com o aluno, decidiram não planear logo os detalhes de cada *sprint* e deixar o planeamento mais aprofundado para o início de cada um. Cada *sprint* é acompanhado então por este *sprint planning* e um *sprint review* no final. Por último, é reservado um tempo semelhante a um *sprint* para o aluno completar o documento da dissertação de estágio final.

Mesmo não tendo um plano detalhado para cada *sprint*, o aluno resolveu planear por alto os conteúdos dos *sprints* no início do segundo semestre:

- **Sprint #1**: Criação dos microsserviços e integração do método de comunicação entre os mesmos.
- **Sprint #2**: Gestão de eventos e Portal público.
- **Sprint #3**: Gestão de eventos e Portal público (2).
- **Sprint #4**: Reserva de bilhetes.
- **Sprint #5**: Gestão de pagamento.
- **Sprint #6**: Integração de bilhetes *NFT*.
- **Sprint #7**: Integração de bilhetes *NFT* (2).
- **Sprint #8**: Estatísticas e tarefas específicas de *front end*.
- **Sprint #9**: Realização de testes na plataforma.

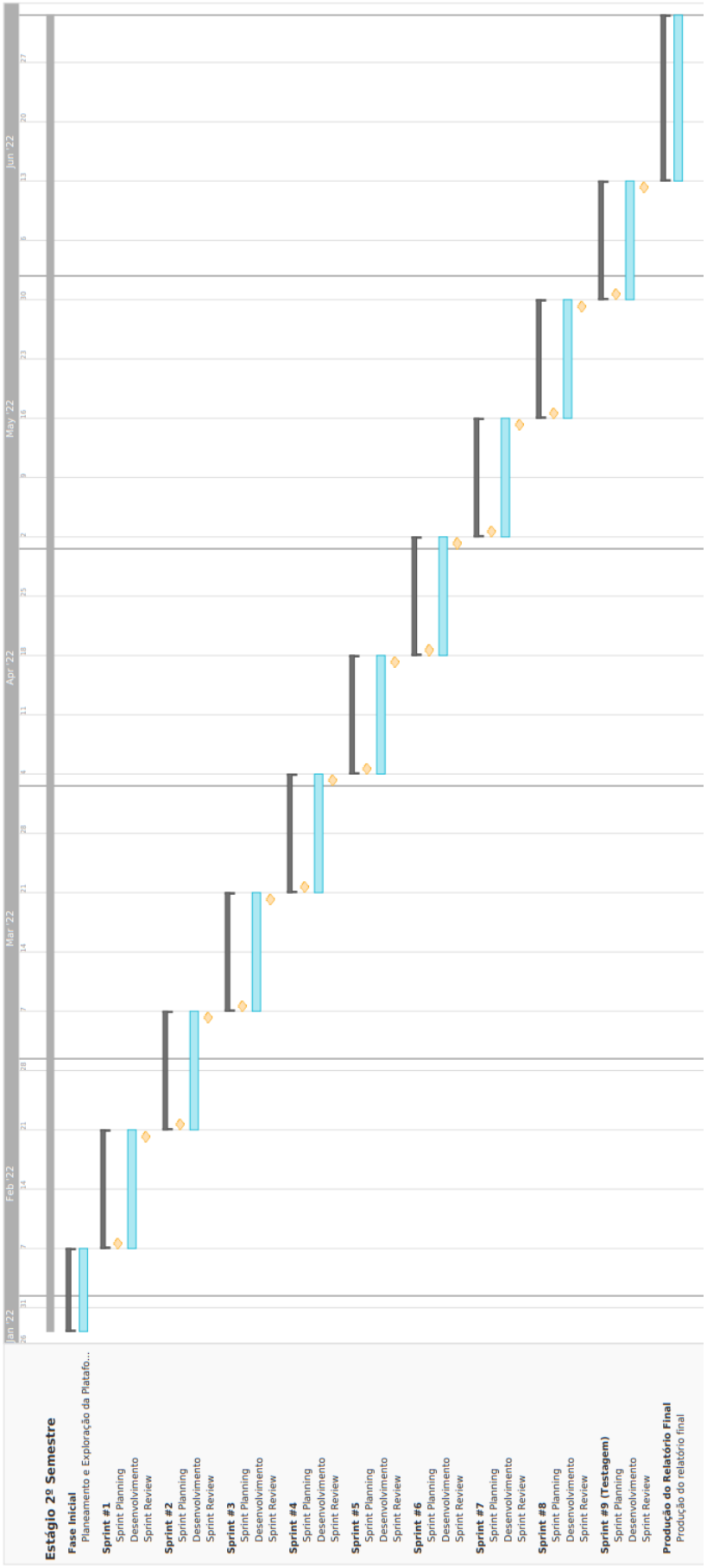


Figura 2.3: Diagrama de Gantt relativo ao segundo semestre

Na Figura 2.4 podemos ver que as datas dos *sprints* corresponderam às datas reais do percurso do segundo semestre. O aluno teve as reuniões diárias tal como é dito na metodologia escolhida, que serviram para acompanhar o aluno em quaisquer problemas que ocorressem durante o *sprint*.

No início de cada *sprint* era feita a estimação das tarefas através de uma discussão sobre a complexidade das mesmas e eram atribuídas a cada uma, um número na **escala de Fibonacci**. À medida que se avança na escala, a complexidade da tarefa aumenta da mesma maneira, logo, complexidades vão aumentando de uma maneira quase exponencial. No estágio foram atribuídos apenas os números 1,2,3,5 e 8 desta escala. Tarefas estimadas acima de 8 devem ser repartidas em sub-tarefas de modo a planear melhor o *sprint*. Para cada *sprint*, foi atribuído um esforço de **16** pontos na escala de Fibonacci.

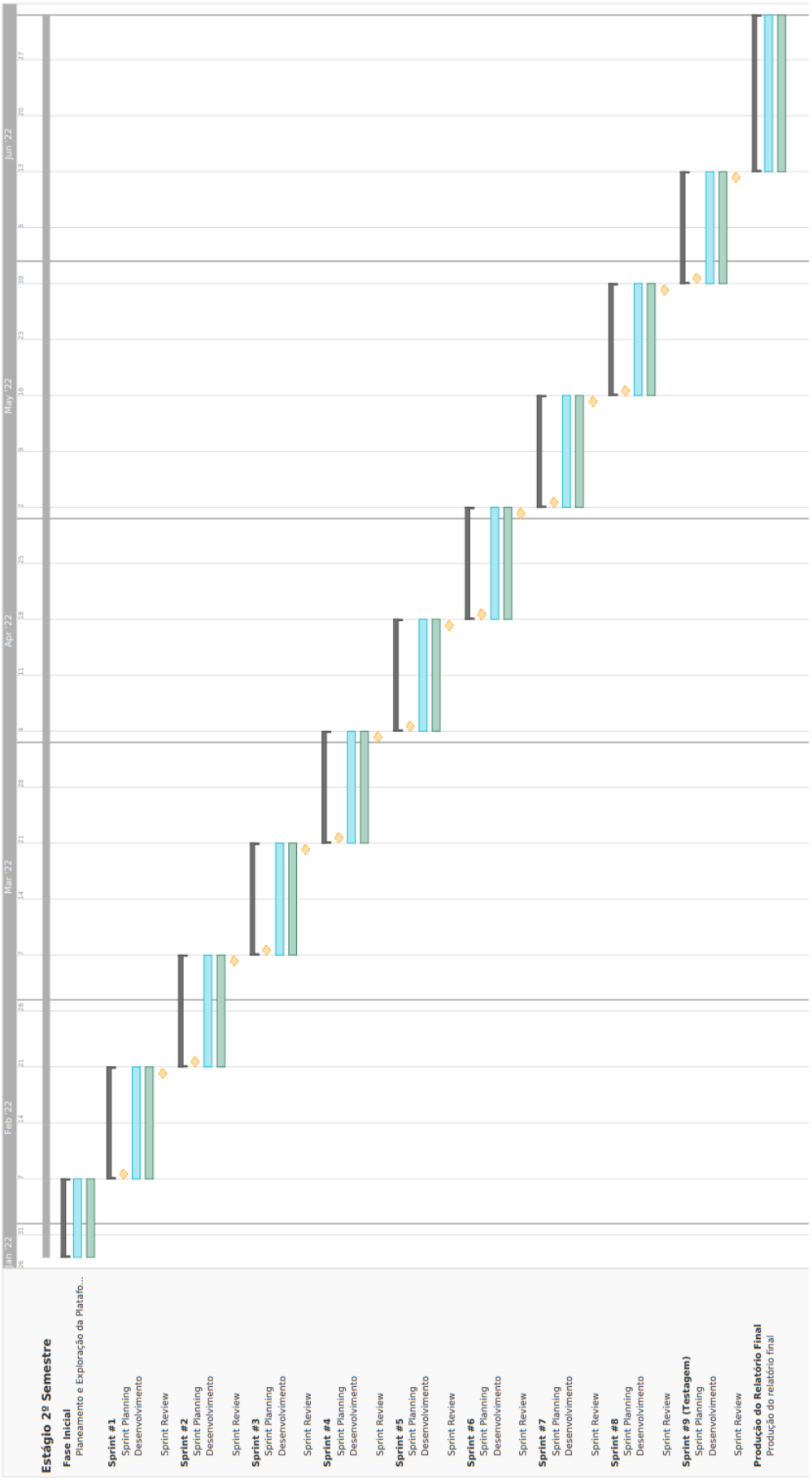


Figura 2.4: Diagrama de Gantt relativo ao segundo semestre (Real)

No final de cada *sprint* foram feitos *sprint retrospectives* na plataforma **Parabol**. Esta ferramenta serve para este tipo de reuniões onde é disponibilizado um espaço para que a equipa possa expressar e discutir os problemas, sugestões ou comentários que possam ter no final de cada *sprint*. Na Figura 2.5 é mostrado um resumo, enviado por email pela plataforma, no final de uma *sprint retrospective* com todos os comentários que foram discutidos e melhorias que ficaram por aplicar nos *sprints* seguintes.

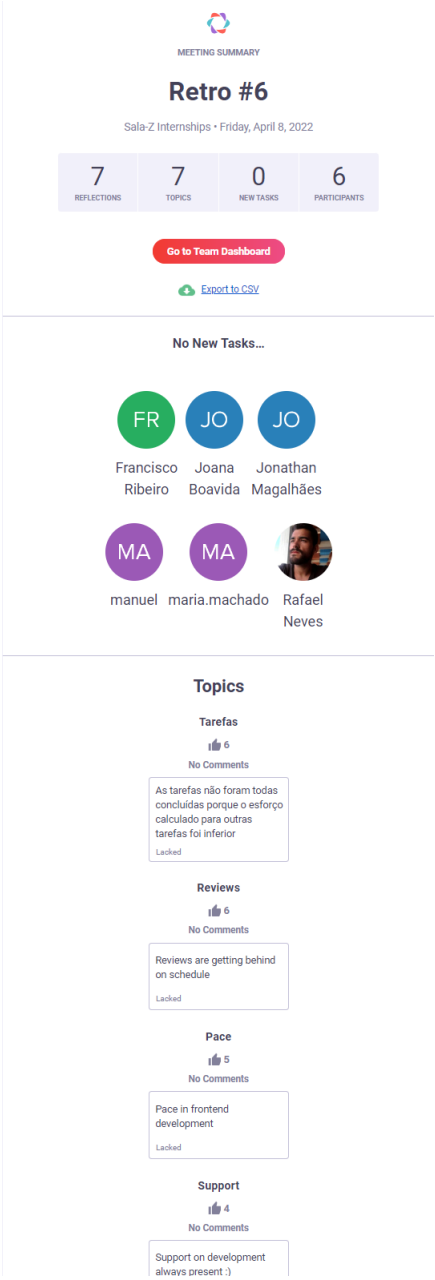


Figura 2.5: Resumo de uma *sprint restrospective* na plataforma *Parabol*

2.3 Riscos

Antes do desenvolvimento é prudente realizar uma análise de riscos de modo a estabelecer atempadamente um plano de mitigação. Cada risco será composto pela sua **descrição**, pela **estratégia de mitigação**, pelo potencial **impacto** e pela **probabilidade** estimada. Enunciam-se de seguida os seis riscos mais relevantes que foram identificados neste processo:

1. **Tecnologias** – Devido ao facto de o aluno não ter experiência prévia nas tecnologias a serem utilizadas durante o desenvolvimento do projeto, este risco torna-se bastante impactante no resultado do mesmo no tempo previsto.

   Estratégia de mitigação: Tempo tem que ser investido na aprendizagem e adaptação das tecnologias pelo que a estratégia de mitigação adequada para este risco será um bom planeamento inicial do segundo semestre que tenha em conta este período. Inicialmente vai ser preciso mais tempo, porém para o final o aluno já estará mais familiarizado com as tecnologias e deverá ter um desempenho superior.

   Impacto: Alto

   Probabilidade: 60%-80%

2. **Tecnologias recentes** – Ainda em relação às tecnologias, mais propriamente *Blockchain* e *NFTs*, estas são tecnologias relativamente novas e com um grande grau de instabilidade visto que o ecossistema ainda está numa fase de adaptação às ferramentas disponíveis. Isto pode criar imprevistos durante o desenvolvimento uma vez que certas ferramentas possam já não ser indicadas para o objetivo do aluno, ou até mesmo descontinuadas.

   Estratégia de mitigação: O aluno vai ter que estar continuamente em pesquisa sobre estas tecnologias para tentar conseguir prever quais as que mais se adequam ao projeto em qualquer instante do mesmo. A metodologia utilizada também tem em conta este risco.

   Impacto: Baixo

   Probabilidade: 30%-40%

3. **Experiência** – O aluno poderá sentir dificuldades na adaptação inicial das ferramentas e ambiente de desenvolvimento da empresa, o que pode afetar o tempo de desenvolvimento. No entanto, tem conhecimentos teóricos e alguma experiência através da componente prática do curso da Universidade.

   Estratégia de mitigação: Tal como na estratégia de mitigação das Tecnologias, um bom planeamento será necessário para a adaptação às ferramentas. Em relação aos processos do ambiente de desenvolvimento da empresa, o aluno terá apoio do orientador que fornecerá uma ajuda inicial para uma aprendizagem mais rápida.

   Impacto: Médio

   Probabilidade: 10%-30%

4. **Pouca adesão** – As funcionalidades em questão utilizam uma tecnologia bastante recente (*Blockchain*) e ainda com pouca adesão mundialmente devido a vários problemas como volatilidade e segurança. Já existem plataformas (algumas irão ser estudadas no Capítulo 3) que fazem com que esta tecnologia se desenvolva, porém ainda se encontra num estado inicial, o que faz com que as funcionalidades implementadas na plataforma sejam alvo de pouco uso.

   Estratégia de mitigação: Apostar na divulgação destas funcionalidades num sítio onde a maioria dos utilizadores ou potenciais utilizadores consigam ver, mas, no entanto, permitir realizar as operações implementadas nesta tecnologia da maneira tradicional que neste caso é a compra de bilhetes sem integração na *Blockchain* por NFTs.

   Impacto: Baixo

   Probabilidade: 70%-80%

5. **Pandemia** – Durante este ano a situação global ainda se encontra com medidas restritivas face ao COVID-19. Estas medidas podem prevenir que o aluno se encontre presencialmente no ambiente de trabalho o que pode vir a dificultar o processo de aprendizagem e desenvolvimento.

   Estratégia de mitigação: Para mitigar este risco, o aluno tem que utilizar as várias ferramentas de comunicação utilizadas pela empresa, como utilizou durante o primeiro semestre. Estas ferramentas têm que ser utilizadas se as medidas não permitirem que o aluno esteja presencialmente. Também é de salientar que dependendo da gravidade da situação, o aluno pode evitar ficar a trabalhar remotamente se forem cumpridas medidas que permitam estar fisicamente no espaço da empresa.

   Impacto: Baixo

   Probabilidade: 50%-70%

6. **Trabalho em conjunto** – Visto que no contexto do estágio, o aluno irá trabalhar com outro estagiário no mesmo projeto, algumas componentes vão muito provavelmente ser partilhadas e algum do trabalho do aluno irá até depender de componentes implementadas pelo outro estagiário. Esta condição pode afetar o tempo de desenvolvimento uma vez que se os trabalhos estiverem dessincronizados, será necessário mais tempo para resolução de conflitos e integração de mudanças.

   Estratégia de mitigação: Para mitigar este risco, meios de comunicação e reuniões sistemáticas têm que ser garantidos. Tanto o aluno como o outro estagiário têm que estar sempre a par do trabalho do outro sempre que um esteja a implementar novo código em componentes partilhadas. A utilização de um sistema de controlo de versões no projeto também é obviamente obrigatória.

   Impacto: Baixo

   Probabilidade: 80%-100%

Face a estes riscos descritos, foi elaborada uma matriz de riscos apresentada na figura 2.6.

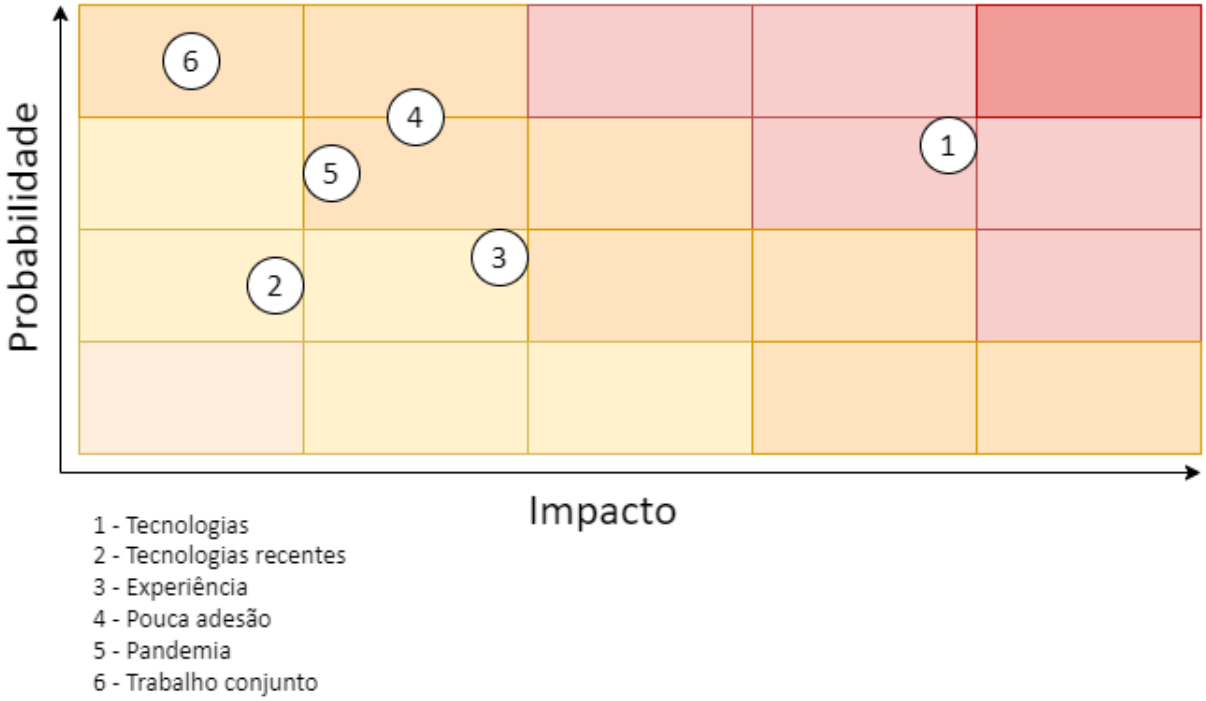


Figura 2.6: Matriz de riscos de desenvolvimento

Capítulo 3

Estado da Arte

Ao longo deste capítulo é feito um ponto de situação sobre a plataforma Sala-Z que existe e análise da concorrência da mesma, um estudo sobre a tecnologia *Blockchain*, *NFTs* e as suas aplicações, quais as plataformas que implementam este tipo de *token* e uma comparação informada orientada ao objetivo do estágio. Também é referido um breve estudo e comparação entre tecnologias *front end* e *back end*. Por último, é tomada a decisão de quais as tecnologias a utilizar com base no estudo prévio.

Blockchain é uma tecnologia revolucionária relativamente recente que tem como objetivo descentralizar os processos existentes no ambiente digital para fins de segurança, transparência e confiabilidade. Uma das aplicações de *blockchain* é o conceito de *Non-Fungible Token* (NFT). Um NFT é um tipo de cripto moeda derivada dos *smart contracts* de *Ethereum* caracterizada por ser infungível, isto é, que não pode ser trocada por outra do mesmo tipo e manter o mesmo valor uma vez que é única [49]. Com esta propriedade pode facilmente utilizar-se este tipo de *tokens* para representar ativos como imóveis, arte, bilhetes e outras propriedades.

3.1 *Background*: Sala-Z

O Sala-Z é uma aplicação interna da Grama que tem como objetivo oferecer uma plataforma para artistas e gestores de salas de espetáculos de modo que estes consigam organizar melhor todo o processo da criação de um evento, bem como oferecer possibilidade de venda de bilhetes para esses mesmos eventos.

A versão anterior da plataforma Sala-Z oferece funcionalidades como autenticação na plataforma, registo de salas de espetáculos, estatísticas dos eventos para os gestores das salas de espetáculos, gestão dos utilizadores na plataforma, criação de eventos nas salas de espetáculos, entre outros (ver Figura 3.1).

A visão para o futuro desta versão da plataforma passava por implementar várias funcionalidades de forma a agilizar alguns pontos fracos e outras completamente novas como um espaço para utilizadores não autenticados poderem ver os eventos e comprar bilhetes. Outras destas funcionalidades consistiam na criação de

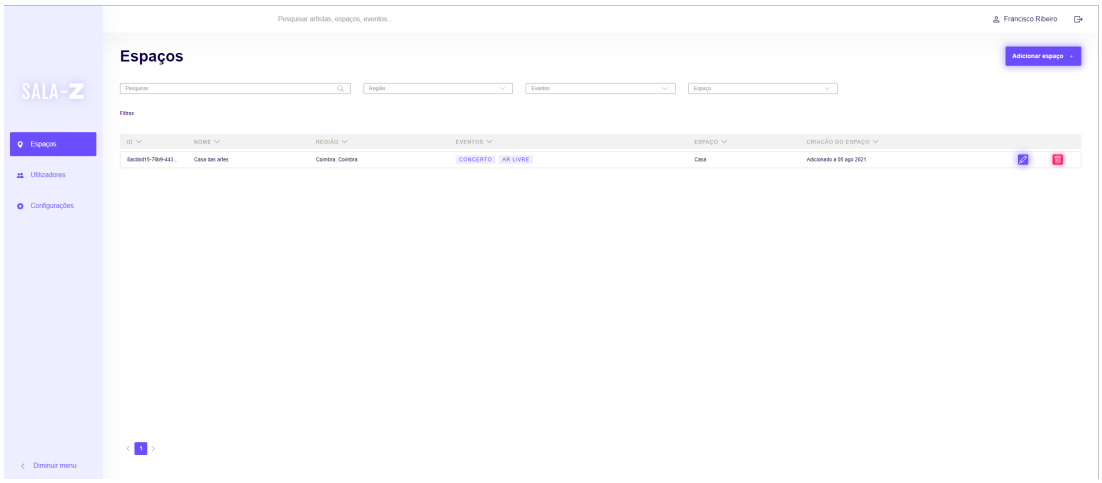


Figura 3.1: *Screenshot* da antiga versão da plataforma Sala-Z

uma área de inventário para eventos, um espaço para comunicação entre o *staff* de um evento e gestão de bilhética através de tecnologia *Blockchain*, como foi o trabalho desenvolvido neste documento.

Também é importante referir as tecnologias utilizadas na versão atual do Sala-Z. A aplicação segue uma arquitetura de microsserviços implementados em *Java*, mais especificamente com a *framework Quarkus*. O *front end*, uma *Single Page Web Application* escrito em *Vue.js*, comunica com os serviços no *back end* através de uma API. Toda a infraestrutura da aplicação é suportada pela AWS. Cada microsserviço está alojado num *container* EC2. Existe uma única base de dados com vários *schemas* em *PostgreSQL* alojada num *container RDS*. Para a distribuição de carga existe um sistema ELB que trata disso automaticamente. Por fim, o conteúdo estático do *front end* está alojado num *bucket* S3 e distribuído aos utilizadores através da *CloudFront* da AWS. Conteúdo como imagens, videos e o próprio código *Vue* da aplicação estão guardados neste *bucket*. Existe outro *bucket* designado para armazenar conteúdo dinâmico produzido pela aplicação.

3.2 Gateways de pagamento

Para realizar pagamentos numa aplicação é necessário utilizar *gateways* de pagamento visto que implementar um sistema de pagamento de raiz seria bastante mais complexo e, potencialmente, exposto a ataques. Uma *gateway* de pagamento é uma tecnologia que captura e transfere dados de pagamento do cliente para o vendedor [34]. Por outras palavras, a *gateway* de pagamento serve como intermediário entre as duas entidades para garantir ao comprador que a transação é feita com segurança e de uma só vez. Existem várias opções para a escolha de uma *gateway* de pagamento. De seguida, vão ser apresentadas algumas destas opções, oferecendo também uma comparação entre as mesmas.

3.2.1 SIBS Payment Gateway

A *SIBS Payment Gateway* é uma solução destinada a vendedores com uma loja online que necessitam de um método de pagamento prático e seguro. Várias opções são fornecidas como *MB WAY*, referência Multibanco ou cartão bancário. A solução também oferece uma vista agregada das transações e fornece um conjunto de operações sobre a mesma, que melhora o dia-a-dia do negócio [46]. Este serviço oferece um conjunto de APIs que permitem realizar todo o processo da compra como o *checkout*, geração de referências multibanco ou *MB Way*, cancelamento de compra ou devolução, embora a documentação para estas APIs seja um pouco complexa.

O custo deste serviço não foi fornecido visto que para saber esta informação, um projeto tem que ser criado e desenvolvido primeiramente.

3.2.2 EuPago

A *euPago* é uma empresa portuguesa sediada no Porto fundada em 2012. O serviço que oferece é também um *payment gateway* que oferece acesso a um *backoffice* que permite consultar as referências geradas bem como gerar manualmente uma referência. Possui também duas *APIs*, *REST* e *SOAP* que permitem ao *developer* gerar as referências de pagamento automaticamente, implementando-as na sua plataforma. Funcionalidades como reembolsos, débitos diretos, *split-payments*, *openbank*, consulta de *payouts* e referências são todas suportadas [35]. As instruções para a integração destas *APIs* vêm num ficheiro *pdf* e estão bastante claras e concisas.

Para esta opção não existe custo de adesão, mas os custos são variados dependendo da escolha do método de pagamento. A Tabela 3.1 contém a informação dos custos.

Método de Pagamento	Tarifário	Observações
Referências Multibanco	Tarifa Fixa: 0,70€ Tarifa Mista: 0,25€ + 1,8% da transação	Na tarifa mista, o valor máximo é de 1,85€ por transação. Acresce IVA.
Payshop	0,60€	Acresce IVA.
MB Way	0,07€ + 0,7%	Máximo 1,65€ por transação. Acresce IVA.
Paysafecard	12% da transação	Acresce IVA.
Paysafecash	0,25€ + 3% da transação	Acresce IVA.
Débitos diretos	0,45€	Acresce IVA.
Cartão de Crédito 3DS	0,25€ + 2,25% da transação	Acresce IVA.

Tabela 3.1: Custos e tarifas do serviço EuPago

3.2.3 Stripe

Stripe é uma plataforma de pagamentos mais orientada aos *developers* e à facilidade de integração. Este *gateway* de pagamento oferece APIs para pagamento, opções de *no-code* onde o *developer* não tem que escrever código, *webhooks* para suportar pagamentos realizados posteriormente e um *backoffice* de alto desempenho. Embora um novo utilizador talvez tenha dificuldades numa fase inicial devido ao grande número de funcionalidades e escolhas, o apoio da comunidade é bastante extensivo e existem inúmeros recursos para a integração deste serviço. Infelizmente, uma grande desvantagem deste serviço é a ausência de suporte para pagamentos *MBWay* visto ser uma plataforma internacional sem integração de métodos de pagamento locais. *Stripe* aceita *Multibanco*, mas ainda é uma integração em fase *beta*, havendo, por isso, a possibilidade de surgirem problemas.

Não existe custo de adesão ou subscrição anual pelo que este serviço opera num modelo *Pay-as-you-go*. Contudo, existem taxas por transação. Para cartões de crédito ou débito **europeus** as taxas são de **0,25€ + 1.4%**. Para cartões internacionais, as taxas são de **0,25€ + 2,9%**. Para outras funcionalidades como geração de referências e sistema de informação atualizado no *backoffice*, existem diferentes taxas que podem ser consultadas no site da *Stripe* [48].

Na Tabela 3.2 encontra-se uma comparação destes três serviços estudados.

	SIBS Payment Gateway	EuPago	Stripe
API	REST	REST, SOAP	REST
Pagamentos com cartão débito/crédito	Sim	Sim	Sim
Pagamentos com MB Way	Sim	Sim	Não
Cancelamentos e Devoluções	Sim	Sim	Sim
Dificuldade de Integração	Difícil/Média	Média/Fácil	Fácil
Taxas de adesão	Não	Não	Não

Tabela 3.2: Comparação entre os diferentes gateways de pagamento

3.3 Blockchain

Uma *blockchain* é um grande registo digital implementado de uma maneira distribuída capaz de oferecer proteção contra adulteração de dados, devido à ausência de uma autoridade central que controle o sistema [50]. Os participantes, ou *nodes*, têm a sua cópia da *blockchain* que é governada através de um mecanismo de consenso onde são especificadas as regras de criação de blocos e transações, de modo a que cada um destes *nodes* saiba como gerir e sincronizar a informação.

A tecnologia *blockchain* tem bases fortes em criptografia. Para perceber como esta tecnologia funciona, é necessário perceber primeiro o conceito de *hashing*. Uma

hash é o resultado de uma *hashing function* com um certo *input*. Estas funções convertem esse *input* num *output* completamente diferente. Até agora o que foi descrito foi meramente uma função normal, contudo, este tipo de funções têm certas propriedades que as tornam bastante interessantes, definidas por Merkle [47]:

- A função de *hash* (H) pode ser aplicada a um bloco de dados de qualquer tamanho.
- H produz um *output* de tamanho fixo.
- É facilmente produzido o resultado da função H com um qualquer *input* x.
- Dado H e H(x), é computacionalmente inviável descobrir x.
- Dado H e H(x), é computacionalmente inviável descobrir um y tal que a *hash* de y seja igual à *hash* de x.

Com isto em mente, uma *blockchain* consiste numa série de blocos de informação ligados entre si contendo a *hash* da sua própria informação e a *hash* do bloco anterior. Desta maneira temos então a ligação entre cada bloco que vai sucessivamente aumentando à medida que mais blocos se juntam. Devido às propriedades definidas em cima, pode-se ver facilmente porque é que esta tecnologia oferece a tal segurança e imutabilidade dos dados. Caso um interveniente tente mudar informação de um bloco, a *hash* desse bloco inevitavelmente muda, quebrando assim a corrente. Para alterar informação de um bloco com sucesso, o interveniente teria de mudar a *hash* desse bloco mais a *hash* de todos os blocos que vêm a seguir a esse na *blockchain*. Quanto mais antigo o bloco for, logicamente mais trabalho computacional terá de realizar e menos probabilidade de suceder com a alteração.

Para este grande sistema distribuído funcionar, tem que haver um conjunto de normas e regras de inserção de blocos de modo a que todas as transações sejam agrupadas em blocos e que a mesma informação seja inserida corretamente e sincronizadamente em todas as cópias em cada *node*. A este conjunto de regras dá-se o nome de mecanismos de consenso, alguns destes detalhados de seguida.

3.3.1 Mecanismos de Consenso

O ***Proof of work*** é o mecanismo de consenso mais popular e é o mecanismo adotado pela *Bitcoin*. Um *node* publica um bloco se for o primeiro a resolver um *puzzle* de computação intensiva [52]. A solução para este puzzle é a prova de que o *node* "trabalhou"e a este processo dá-se o nome de ***mining***. Ao publicar um bloco, o *node* que o publicou recebe recompensas na rede em forma de cripto moeadas ou taxas de transações dos blocos de modo a compensar o trabalho que teve ao calcular a solução do *puzzle*. Um exemplo de um *puzzle* comum é descobrir uma *hash* para o bloco de modo a que esta seja inferior a um certo número previamente definido. O *node* tenta várias *hashes* para o bloco através de mudanças ligeiras num número incluído dentro do bloco chamado *nonce* que foi feito mesmo para este

propósito. Pode-se ajustar a dificuldade do *puzzle* através da alteração do valor previamente definido que representa o limite superior da *hash*. Para blocos na *Bitcoin* este desafio leva cerca de 10 minutos a ser completado. Este mecanismo é bastante seguro, no entanto é prejudicial ao meio ambiente visto que gasta muita energia durante a computação de *hashes*.

O ***Proof of stake*** é um mecanismo de consenso que veio para resolver os problemas de escalabilidade e ambientais do *Proof of Work*. Este mecanismo parte do princípio de que quanto mais valor um utilizador tem investido na rede, mais este quer que a rede seja bem sucedida [50]. Cada utilizador, ou *node*, pode bloquear um valor de moedas dessa rede numa conta especial ou através do próprio protocolo, ficando assim impossibilitado de as usar. A este processo dá-se o nome de *staking*. Ao fazer *staking*, um utilizador pode publicar blocos e ganhar as recompensas da rede sem ter que realizar os trabalhos computacionais do mecanismo anterior.

O mecanismo ***Round Robin*** dita que a criação dos blocos é feita de uma maneira sequencial por todos os nós do sistema. Previsivelmente, é um mecanismo bastante rápido e eficiente, porém o nível de segurança é muito baixo sendo apenas exequível numa *blockchain* com permissões em que cada nó é autenticado e reconhecido por todos os outros nós [50].

O ***Proof of Authority*** ou ***Proof of Identity*** funciona através da definição de nós publicadores de blocos confiáveis através de provas concretas da ligação a entidades reais. Estas entidades têm que estar bem definidas dentro da *blockchain* e vão perdendo ou ganhando reputação consoante os blocos que publicarem. Porém, este mecanismo também só se aplica a *blockchains* com permissões e com grandes níveis de segurança [50].

3.4 Smart Contracts

Smart Contracts são códigos executáveis que correm em cima de uma *blockchain* e facilitam, executam e impõem um acordo entre duas entidades que podem ou não confiar-se mutuamente sem a presença de uma terceira entidade [39]. Os participantes sabem sempre qual o resultado da transação antes de a realizar. Existem muitas vantagens para se utilizar esta tecnologia nomeadamente, **rapidez** uma vez que assim que as condições estão corretas o contrato é executado imediatamente sem quaisquer burocracias; **Transparência**, visto que não existem terceiros e os registos das transações são partilhadas com todos os elementos da *blockchain* impossibilitando a sua alteração após estarem concluídos e inseridos no bloco; **Segurança**, pois todas as transações referidas estão encriptadas e subsequentemente são praticamente impossíveis de sofrer ataques, e por último, tem a vantagem de serem mais económicos que os processos de acordos sem auxílio desta plataforma, visto que eliminam os terceiros que levam taxas e tempo.

Porém, nem todas as *blockchains* oferecem esta funcionalidade. Algumas das plataformas preparadas para executar estes programas vão ser discutidas e comparadas de seguida.

3.4.1 Ethereum

Ethereum foi a primeira *blockchain* a introduzir este conceito de *smart contracts*. Suporta *smart contracts* com funcionalidades avançadas e altamente customizáveis com a ajuda de um ambiente de *runtime Turing-complete* chamado *Ethereum Virtual Machine* (EVM) [39]. Este ambiente é executado em cada *node* na rede para executar estes *smart contracts*. Para a programação destes pedaços de código é utilizada a linguagem de programação *Solidity*. Este código *high-level* por sua vez é compilado e transformado em *bytecode* legível pela EVM. Porém, executar este código na *blockchain* torna-se bastante dispendioso visto que *Ethereum* é a plataforma mais popular para este tipo de programas e consequentemente o preço torna-se maior. Parte do problema deve-se ao facto de ser uma rede *Proof of Work* o que torna a inclusão de blocos mais lenta levando assim ao aumento do preço por transação. *Ethereum 2.0* é um projeto para melhorar significativamente estes aspetos negativos, sendo a principal melhoria a transição de um mecanismo *Proof of Work* para *Proof of Stake*.

3.4.2 Flow

Flow é uma *blockchain* orientada aos desenvolvedores de modo a facilitar o processo de criação dos *smart contracts*. Tem uma arquitetura única baseando-se em *multi-role* dos nós da rede e está desenhada para não ter problemas de escalabilidade mesmo sem *sharding* (divisão da *blockchain* em várias cadeias mais pequenas). É uma plataforma que se foca em oferecer a possibilidade de criar negócios baseados em *crypto*, sobretudo na representação de ativos.

Distinguem-se pela sua arquitetura *multi-role*, programação orientada aos recursos através de uma linguagem chamada *Cadence*, múltiplas vantagens para os desenvolvedores durante o processo de desenvolvimento nomeadamente *logs* automáticos dos contratos no *Flow Emulator*, e facilidade para os consumidores no tempo de trocar *fiat* (dinheiro institucionalizado) para *crypto*.

3.4.3 Solana

Solana é uma *blockchain* que se distingue pela sua grande capacidade de transações por segundo. Foi criada por um grupo de desenvolvedores provenientes de grandes empresas como *Intel*, *Dropbox* e *Qualcomm*. A maneira como conseguem este feito provém da sua arquitetura *Proof of History*. Em vez de agrupar várias transações num bloco como é feito na arquitetura *Proof of Work*, cada transação é essencialmente o seu próprio bloco e usa a transação anterior como prova de que está correta, tendo assim verificadores para verificar a integridade desta corrente.

O ambiente de execução dos seus *smart contracts* é *Low Level Virtual Machine* (LLVM) e usa C e Rust como linguagem de programação dos mesmos, o que torna um pouco difícil o seu desenvolvimento, porém as vantagens facilmente superam as desvantagens nesta plataforma. Solana também é bastante apelativo devido ao seu preço de transações que é cerca de 60 vezes mais barato que *Ethereum*. Isto

é devido aos seus baixos tempos de criação de novos blocos para a *blockchain* e grande capacidade máxima de transações num único bloco, o que torna as transações menos dispendiosas.

3.4.4 Polkadot

Polkadot foi criada pelo cofundador de *Ethereum* Gavin Wood. Não se trata totalmente de uma única *blockchain*, mas sim de um ecossistema onde várias plataformas *blockchain* se interligam. É uma plataforma que oferece um grande grau de liberdade aos desenvolvedores, dando capacidades de criarem as suas próprias *blockchains* e *tokens* que conseguem interagir entre si. O seu grande grau de interoperabilidade vem da tecnologia *Relay Chain*, que é basicamente uma *blockchain* de *blockchains* chamadas *parachains*.

3.4.5 Stellar

Stellar é uma *blockchain* que não é *Turing complete*, o que significa que não consegue implementar algoritmos com uma complexidade maior, estando assim apenas capaz de suportar *smart contracts* muito básicos nomeadamente representações de trocas simples de dinheiro, *Initial Coin Offering* (ICO) e acordos com garantias. Porém, visto ser rápido, eficiente e barato, a IBM apostou nesta plataforma e criou a *World Wire* que é uma plataforma simples de troca de dinheiro. Não existe um ambiente ou linguagem específica de programação, logo, os programadores podem utilizar qualquer linguagem o que torna vantajoso a escolha da plataforma. Infelizmente, a criação de *NFTs* nesta plataforma é possível, mas não adere à norma e é mais uma solução específica à plataforma.

É de notar que o mecanismo de consenso personalizado chamado *Stellar Consensus Protocol* (SCP) tem problemas de centralização visto que apenas existem dois nós de rede fundamentais para o mecanismo funcionar e estes são controlados pela Fundação *Stellar* [40].

3.4.6 Polygon

Polygon é uma plataforma semelhante a *Polkadot* no sentido de ser uma *blockchain* de *blockchains* por funcionar na *Layer 2*. Porém, esta foca-se na integração em ambiente *Ethereum*. É um protocolo e uma *framework* para construir e ligar *blockchains* compatíveis com *Ethereum* [41]. A maior vantagem que oferece é conseguir extrair todas as vantagens da rede *Ethereum* sem comprometer a segurança. *Polygon* também é uma boa solução devido à sua escalabilidade atingindo 65 000 transações por segundo na *side chain* criada.

3.5 Comparação das plataformas Smart Contracts

Na Tabela 3.3 podemos ver uma comparação destas plataformas, contendo as métricas mais importantes para realizar a escolha da melhor plataforma. A verde estão indicados os valores mais vantajosos em relação ao resto das opções.

A velocidade está descrita em *Transactions per second* (TPS) e está diretamente correlacionada com a habilidade da plataforma de ser escalável. O custo médio por transação é bastante volátil e depende muito da carga na rede, do preço da moeda associada visto que estas taxas são pagas nessa mesma moeda e do tipo de transação, uma vez que se forem chamadas funções complexas de *smart contracts*, o preço da transação será mais alto do que simples transações de moedas de uma conta para outra. Estes valores foram retirados no mês de junho de 2022 [5] [6] [8] [9] [51] [14].

	Velocidade	Custo médio por transação	Desenvolvimento
Ethereum	20 tps	3€-4€	Documentação vasta e clara
Flow	100 tps	<0.001€	Documentação clara
Solana	75000 tps	<0.001€	Documentação clara
Polkadot	166666 tps	1€-2€	Documentação complexa
Stellar	3000 tps	<0.001€	Documentação clara
Polygon	65000 tps	<0.01€	Documentação vasta e clara (Ethereum)

Tabela 3.3: Comparação das plataformas de Smart Contracts

3.6 Norma ERC 721 (*Non Fungible Token*)

A norma *Ethereum Request for Comments* (ERC) 721, como anteriormente mencionada, descreve a maneira como implementar um *Non Fungible Token* nas plataformas de *Smart Contracts*. Segundo esta norma, para um *token* se considerar um *NFT* é necessário implementar no contrato os eventos e funções especificados na Figura 3.2 e na Figura 3.3.

Este código foi retirado da página da *Ethereum* e está escrito em *Solidity* [4]. De seguida, estas funções e estes eventos serão descritos em mais detalhe.

3.6.1 Funções

O **balanceOf** recebe uma variável do tipo *address* que representa uma carteira onde são guardados os *tokens*. Retorna o número de *NFTs* que essa carteira contém. Caso o *address* seja o *zero address* (*address* composto apenas por zeros) esta função retorna um erro.

```
    function balanceOf(address _owner) external view returns (uint256);
    function ownerOf(uint256 _tokenId) external view returns (address);
    function safeTransferFrom(address _from, address _to, uint256
_tokenId, bytes data) external payable;
    function safeTransferFrom(address _from, address _to, uint256
_tokenId) external payable;
    function transferFrom(address _from, address _to, uint256 _tokenId)
external payable;
    function approve(address _approved, uint256 _tokenId) external
payable;
    function setApprovalForAll(address _operator, bool _approved)
external;
    function getApproved(uint256 _tokenId) external view returns
(address);
    function isApprovedForAll(address _owner, address _operator)
external view returns (bool);
```

Figura 3.2: Funções do *standard* ERC 721

```
    event Transfer(address indexed _from, address indexed _to, uint256
indexed _tokenId);
    event Approval(address indexed _owner, address indexed _approved,
uint256 indexed _tokenId);
    event ApprovalForAll(address indexed _owner, address indexed
_operator, bool _approved);
```

Figura 3.3: Eventos do *standard* ERC 721

O **ownerOf** recebe uma variável inteira que representa o id de um *NFT* que é gerado no momento da sua criação e é globalmente único juntamente com o *address* do dono desse *NFT*. Esta função retorna o *address* do dono desse *NFT*. *Tokens* pertencentes ao *zero address* também retornam um erro sendo considerados inválidos.

O **safeTransferFrom** recebe dois *addresses*, um que representa a carteira de onde se quer transferir um *NFT* e outro que representa a carteira para onde se quer enviar. Recebe também um *token id* que representa o *NFT* em questão e uma variável do tipo *byte* chamado *data* que serve para enviar dados adicionais que são enviados para o *address* destinatário com a transferência. Uma transferência só pode ocorrer quando o *address* remetente coincide com o *address* do contrato em si, estando este disponível numa variável global chamada *msg.sender* (apenas em *Ethereum*), ou o *address* é um *address* aprovado para aquele *NFT* ou ainda se for um *address* operador que já vão ser discutidos mais à frente. Quando a transferência é finalizada, esta função verifica se o *address* destinatário é um *smart contract* (através da variável *codesize* > 1) e, se for, executa uma chamada da função *onERC721Received* neste e espera receber uma resposta com todos os dados cor-

retos relativos à transferência que acabou de ser realizada. Existe também uma função igual a esta com uma diferente assinatura que apenas omite o parâmetro data.

O **transferFrom** é idêntico ao anterior sendo que difere apenas na confirmação de que o *address* destinatário é de facto capaz de receber *NFTs* que tem que ser feita manualmente. Caso este não seja, o *NFT* pode ser irremediavelmente perdido.

O **approve** recebe como parâmetros um *address* e um *token id*. Tem como objetivo permitir um *address* de poder transferir um certo *NFT*, este claro se for o *address* dono do *NFT* ou um operador do dono. Apenas um *address* pode ser aprovado para um *NFT* e para afirmar que um *NFT* não tem qualquer *address* aprovado basta chamar esta função com o *zero address* como parâmetro.

O **setApprovalForAll** leva como parâmetros um *address* e um *boolean*. O objetivo desta função é aprovar ou revogar direitos de gestão da carteira do *address* que chamou esta função. Estes *addresses* são chamados de operadores e têm acesso completo a todos os *tokens* da carteira a que têm acesso. Esta função emite também um evento chamado *ApprovalForAll* no final. Este contrato tem, no entanto, que permitir múltiplos operadores nas suas funções.

O **getApproved** permite descobrir qual o *address* que foi aprovado para um determinado *NFT*, levando então um *token id* como parâmetro. Retorna o *address* aprovado ou *zero address* se não houver nenhum.

O **isApprovedForAll** leva dois *addresses* como parâmetros, um representando o dono e o outro representando o *address* que se quer verificar se é operador da carteira do dono ou não. Retorna *true* ou *false* dependendo do resultado.

3.6.2 Eventos

Estes eventos servem para notificar clientes que estão à escuta de alterações no *smart contract*. Estes clientes podem ser aplicações *front end* que são apresentados ao utilizador ou podem ser aplicações estatísticas para acompanhar as transações do contrato.

O evento **Transfer** é emitido quando uma transferência é executada, isto é, cada vez que um dono de um *NFT* muda, através das funções *safeTransferFrom* ou *transferFrom*. A informação que oferece são dois *addresses* representando a carteira remetente e destinatário e o *token id*. Também é emitido quando o *NFT* é criado ou destruído estando assim o *address* remetente a *zero address* se tiver sido criado, ou o *address* destinatário a *zero address* se tiver sido destruído. Este evento também é responsável por redefinir o *approved address* do *NFT* em questão para o *zero address* visto que este tem um novo dono.

O evento **Approval** serve para informar que o *address* aprovado de um *NFT* mudou. A informação dada consiste em dois *addresses* a representar o dono do *NFT* e o novo *address* aprovado, e o *token id* do *NFT*. Se o novo *address* aprovado for o *zero address* isto significa que este *NFT* não tem qualquer *address* aprovado.

O evento **ApprovalForAll** serve para informar quando um operador ganha ou perde os privilégios de operador para um certo *address*. Informa através de dois *addresses* também tal como o evento anterior, mas difere no último campo que contém um *boolean* a informar se perdeu ou ganhou.

Se um *smart contract* implementar todos os métodos e eventos descritos, é considerado um *NFT* na plataforma em questão. Implementações divergem um pouco de plataforma para plataforma até para adicionar funcionalidades extra, contudo, têm equivalências a estes métodos num certo grau. Uma das implementações mais conhecidas e antigas na plataforma *Ethereum* é *CryptoKitties* que usa os *NFTs* (em formato ERC 721) para compras e vendas de gatos digitais únicos.

3.6.3 ERC 1155

A norma ERC 721 foi a primeira definição de um NFT. Contudo, à medida que as necessidades tecnológicas vão evoluindo, foi necessário melhorar esta definição para suportar funcionalidades melhores mantendo o conceito chave da infungibilidade. Pode-se afirmar então que o ERC 1155 é a segunda versão melhorada do ERC 721.

Imaginemos que se desenvolveu um jogo *online* em que cada utilizador cria uma personagem e navega por um mundo onde vai apanhando itens. Este mundo é partilhado por todos os jogadores e existem vários itens espalhados por esse mundo. Com a norma ERC 721, cada um destes itens teria de ser completamente único e mais nenhum dos jogadores poderia apanhar o mesmo item. A norma ERC 1155 permite o conceito de *semi-fungible tokens* que, neste cenário, representa vários itens do mesmo tipo, nomeadamente, uma espada. Temos um certo número de itens iguais mas finitos, que podem ser distribuídos pelos jogadores.

Com este exemplo podemos claramente situar a norma ERC 1155 entre os *non fungible tokens* e os *fungible tokens* pelo facto de se conseguir criar uma classe de *tokens* única, isto é, mantendo o mesmo grupo de propriedades em cada token, mas também mantendo a infungibilidade perante outro tipo de *tokens* que cumpram esta norma.

Existem outras vantagens [3] que vieram a partir desta nova norma nomeadamente:

- Possibilidade de criar vários tipos de *tokens* no mesmo contrato.
- Segurança nas transações melhorada.
- Otimização de código que resulta num decréscimo em taxas de transação.
- Metadata dinâmica para os *tokens* conseguindo assim ter vários URIs no mesmo contrato.

Com esta nova norma, o ERC 721 torna-se praticamente obsoleto uma vez que o ERC 1155 consegue reproduzir todas as funcionalidades definidas na norma antiga e oferecer mais caso seja necessário.

3.7 Arquitetura de projetos NFT

Existem dois padrões de design para o paradigma *NFT*: o primeiro é estabelecido por *top to bottom* em que o criador gera os *NFTs* e os vende ao comprador. O outro padrão consiste em criar um *template NFT* em que cada utilizador pode criar os seus próprios *NFTs* e vendê-los. [49]

Para a sua implementação, é necessário criar um *smart contract* que implemente o standard *NFT* e fazer *deploy* do mesmo na rede escolhida. Este *smart contract* vai servir apenas para manter registo dos donos dos *NFTs* criados por ele e essa informação vai estar sempre disponível na *blockchain*. Normalmente, uma grande parte destes projetos também inclui outro *smart contract* que é responsável por disponibilizar uma plataforma onde se vendem estes *NFTs* que representa basicamente um mercado. Neste *smart contract* é onde se implementa funcionalidades como leilões e verificação das condições de venda e compra.

Na parte do utilizador, alguns requisitos têm que ser cumpridos. O utilizador tem que conectar uma *crypto wallet* para ser autenticado se quiser poder interagir com o *smart contract* da plataforma.

Uma *wallet* é apenas mais um *smart contract* que gere o acesso às moedas e tokens de um determinado endereço [33]. Este endereço é gerado através da geração de *public* e *private keys* utilizadas para assinar todas as transações na *blockchain*. Existem vários tipos de wallets [37]:

- **Desktop wallets** que têm a sua aplicação e podem ser descarregadas e utilizadas através de um computador. Estas são as mais comuns tendo como exemplo *Armory* ou *Bitcoin Knots*.
- **Virtual wallets** são *wallets* que operam na *cloud* logo estão disponíveis para acesso através de qualquer dispositivo na *internet*.
- **Hardware wallets** são *wallets* que não guardam as chaves privadas *online*, mas sim em *hardware*, normalmente em dispositivos USB. São a opção mais segura, porém não são tão convenientes quando o utilizador quer fazer várias transações por dia.

A maneira como isto tudo se liga é através das *frameworks* que comunicam com os nós das redes *blockchain*. Estas *frameworks* intermediárias servem para fazer a ligação entre o *back end* da aplicação (ou *front end*, caso se tenha uma plataforma completamente descentralizada) e a rede *blockchain*. A *wallet* que se mencionou anteriormente serve para gerir os fundos do utilizador e assinar as transações executadas por estas *frameworks*. No caso de *Ethereum*, esta *framework* é a **Web3.js**. No caso de *Solana*, **Anchor** e no caso de *Flow*, **FCL-JS**. Cada plataforma tem a sua *Framework*, de modo que clientes consigam interagir com os nós.

No entanto, utilizar apenas estas *frameworks* torna o processo de desenvolver uma aplicação descentralizada bastante difícil porque existem vários problemas que têm que ser resolvidos. Problemas como em *Ethereum* por exemplo, a recolha de eventos da *blockchain* apenas pela *framework* intermediária não ser suficiente para

ser integrada em ambiente de produção da aplicação. Publicações de transações têm vários aspetos a ter em conta como gestão de *nonce counters*, estimações de *gas* e *transaction republishing*. E, acima de tudo, a segurança da aplicação pode estar comprometida [45]. Para resolver estes problemas, existem serviços que fornecem *back ends* e *nodes* dedicados à aplicação. Estes serviços oferecem também uma maneira eficaz de acompanhar as transações da aplicação e fornecer um espaço para guardar a *metadata* de *NFTs*. Exemplos destes serviços são **Infura**, **Alchemy** e **Moralis**. Uma boa analogia para descrever estes serviços é: plataformas de desenvolvimento de *blockchain* são para uma aplicação descentralizada o que *AWS* é para uma aplicação *web*.

Na Figura 3.4 representa-se a arquitetura geral de uma *Dapp* ou *Descentralized App*.

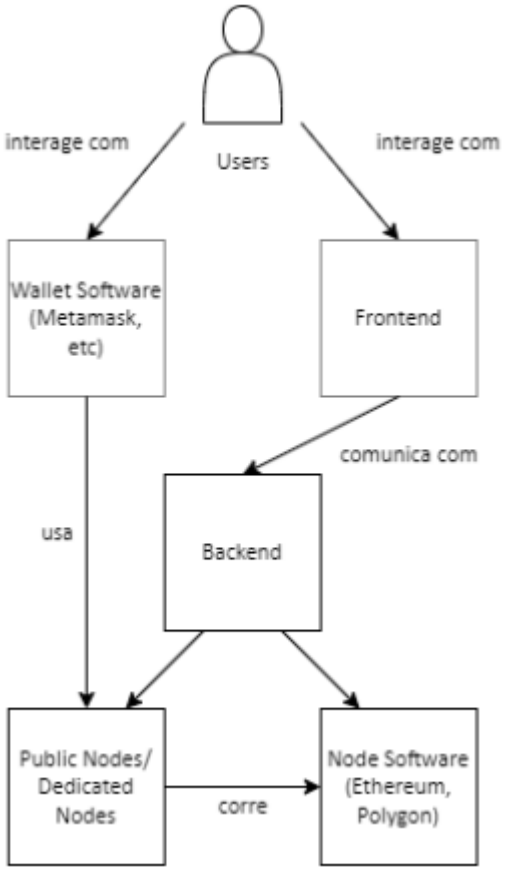


Figura 3.4: Arquitetura de uma aplicação descentralizada ou semi-descentralizada

3.7.1 Metadata

Por último, quando um *NFT* é criado passa a incluir metadados, tais como a ligação para o recurso que o *NFT* representa. Este recurso encontra-se *off-chain* (fora da *blockchain*), principalmente por ser bastante dispendioso guardar muita informação na *blockchain* visto que tem que ser processada, verificada e replicada

pelos *nodes*. Então coloca-se um problema: Como é que se consegue garantir que a ligação que se encontra no *NFT* aponta sempre para o mesmo recurso sem que este sofra quaisquer alterações desde a sua criação? Este problema é resolvido por uma tecnologia chamada **IPFS**. O *IPFS* é uma rede *peer-to-peer* cujo objetivo é armazenar dados. Para localizar os recursos, em vez de utilizar localização como o *http*, utiliza *Content Addressing* para gerar um *content identifier* (CID) que é derivado diretamente do conteúdo em si (*hash*) e faz a ligação para os dados na rede [10]. O seguinte *link* demonstra um exemplo de um URI gerado pelo protocolo. O seguinte campo de texto demonstra um típico endereço *URI* gerado pelo protocolo:

ipfs://bafybedlfn3294f2nknd290ejknd01kj1jri39vma1d/nft-image.png

Deste modo, este *link* apenas está válido para a *nft-image.png* com o conteúdo que foi utilizado para gerar o mesmo.

3.8 Aplicações e Análise da Concorrência

Nesta secção vão ser discutidas as oportunidades que *NFTs* geram em vários contextos e, de seguida, vão ser apresentadas as soluções existentes especificamente para sistemas de bilhética, mostrando soluções tradicionais e algumas soluções que já utilizam *NFTs* para representar os bilhetes.

Esta é uma tecnologia com bastantes oportunidades no mundo atual. Inicialmente, os *NFTs* tiveram início como colecionáveis em jogos digitais e oferecem a capacidade de manter registo de quem tem e teve determinados objetos dentro do jogo [49]. Do lado dos desenvolvedores destes jogos, estes também conseguem lucrar bastante pois podem aplicar uma taxa de revenda destes objetos cada vez que são revendidos. Estes *tokens* também servem para identificar qualquer tipo de bens físicos tais como imóveis, joias, arte e vestuário. E com esta representação também é possível manter registos sobre a sua origem, transações que ocorreram e as suas respetivas datas sem possibilidade de alterar estes dados.

Na área da venda de bilhetes para eventos os *NFTs* vêm resolver problemas antiquíssimos. Antes dos *NFTs*, os compradores tinham obrigatoriamente que confiar na entidade intermediária que vendia os bilhetes, esperando assim que na altura do evento o seu bilhete era de facto válido para a entrada. Outro problema resolvido é o dos revendedores de bilhetes como maneira de lucrar. Quando o evento tem uma grande procura, o número de bilhetes esgota rapidamente e de seguida são revendidos por preços absurdos com margens de lucro de 300% ou mais (conforme a procura). Com os *NFTs*, os vendedores originais conseguem rastrear cada bilhete emitido e proibir ou limitar a margem de lucro de revenda [44].

De seguida são apresentadas as principais soluções encontradas pelo estagiário.

3.8.1 Soluções tradicionais

A **Bol – Bilheteira Online** é uma plataforma que tem como objetivo promover a cultura através da venda de bilhetes para eventos em Portugal continental e ilhas. Atuam nesta área há 25 anos e vendem bilhetes para todo o tipo de cultura, nomeadamente teatro, museus, dança, música, festivais e também eventos de desporto. Têm uma interface através do seu *website* onde se pode escolher o lugar no evento e pagar com cartão débito ou crédito, *Paypal* ou *MBWay*. Têm também vários pontos de venda físicos espalhados pelo país maioritariamente através de acordos com grandes lojas como *FNAC*, *Worten* e *CTT* [1].

A **Ticketline** é uma plataforma de venda de bilhetes para todo o tipo de eventos através da sua tecnologia de gestão de bilhética *TicketNet*. Dispõe de mais de 540 pontos de venda, 460 salas de espetáculos, acesso a compras online 24 horas/dia através do seu *website* e tem uma linha telefónica para reservas e informações. Uma das vantagens desta plataforma é que oferece também gestão de acesso às salas através do sistema *TicketNetAcessos* que gere terminais, portas de entrada, tipos de bilhete e atualiza toda essa informação em tempo real [17].

A **EventGenius** é sem dúvida a mais robusta e eficaz das três mencionadas. Permite que um utilizador que organiza um evento tenha as ferramentas necessárias para o bom desempenho do mesmo. Oferece soluções tecnológicas para bilhética, pagamentos dentro do evento, *marketing* do evento, transporte para o evento e controlo de acesso dentro do estabelecimento. Oferece tecnologias como pontos de venda *cashless* e *contactless*. Ganhou vários prémios e destaca-se pela sua simplicidade e oferta [7].

3.8.2 Soluções à base de *NFTs*

Centaurify é uma aplicação que tem como objetivo revolucionar o mercado de bilhética transformando os bilhetes em *NFTs* e consequentemente recompensar os artistas, organizadores e os compradores [2]. É uma solução baseada na blockchain Cardano. Inicialmente, o percurso vai começar através do mercado da música e mais tarde passar para todos os mercados que envolvam compra de bilhetes para eventos. Porém ainda não existe nada implementado, não passando apenas de uma ideia e de um *website*.

NFT.Kred permite aos utilizadores criar bilhetes em formato de *NFTs* mantendo todo o controlo sobre o mesmo, nomeadamente na definição da data de expiração do bilhete e inserção da marca da organização no mesmo. Funciona através de um modelo de subscrição que oferece possibilidade de gerar mais bilhetes e suporte conforme a subscrição escolhida. Opera na *blockchain Polygon* ou *Ethereum* e os *NFTs* gerados seguem o standard *ERC-721* (ver Secção 3.6) oferecendo assim interoperabilidade na *blockchain*. Este serviço não se limita apenas a transformar bilhetes em *NFTs*, qualquer tipo de objeto pode ser ligado a um *NFT* desde que o utilizador crie um link para o recurso e o adicione no momento da criação do *NFT* [12].

TicketMint é uma plataforma semelhante às anteriores, porém está focada em

emitir bilhetes à base de *NFTs* para concertos e eventos virtuais dentro do *Metaverse* (universo digital maioritariamente construído em realidade virtual). Embora seja esta a visão da plataforma, a sua implementação vai ser semelhante ao que se está a estudar neste documento. No seu *website*, porém, não se encontra qualquer tipo de informação técnica. Apenas informa que pertence a uma empresa chamada *SmartLedger* e apresenta apenas as vantagens e problemas resolvidos com a solução [28].

Digital Twin, da *GET Protocol*, é uma plataforma que permite através de uma *API* criar representações dos bilhetes existentes em *NFTs*. O objetivo desta plataforma não é criar de raiz uma solução de bilhética, mas sim implementar estas representações em *NFTs* por cima da infraestrutura já existente. Tem uma excelente documentação e utiliza a *framework Polygon*. Porém, esta solução apenas é representativa dos bilhetes reais e não os substitui logo servem apenas como uma "memória" para quem os comprou [43].

3.8.3 Comparação das soluções

Como é demonstrado, alguns projetos de bilhética em *NFTs* estão num estado bastante inicial, o que é normal visto que a tecnologia ainda está numa fase inicial. Na Tabela 3.4 é apresentada uma comparação entre as soluções exploradas.

Podemos ver que as soluções baseadas em bilhetes NFT oferecem ainda poucas funcionalidades em relação às plataformas de bilhetes tradicionais. Plataformas como a *Bol* e *Ticketline* têm uma infraestrutura bastante mais sólida que foi naturalmente implementada com o tempo através de contratos com salas de espetáculos e revendedores de bilhetes. As opções de pagamento também são uma clara vantagem para este tipo de plataformas, porém, perdem significativamente no controlo de mercados secundários de bilhetes e na garantia de autenticidade de bilhetes que procuram compensar com serviços de apoio ao cliente.

	Bol	Ticketline	EventGenius	Centaurify	NFT.Kred	TicketMint	Digital Twin	Sala-Z
Bilhetes *NFT*	Não	Não	Não	Sim	Sim	Sim	Sim	Sim
Bilhetes por email	Sim	Sim	Sim	Não	Não	Não	Não	Sim
Pagamento com Cartão Crédito/Débito	Sim	Sim	Sim	Sim (através de Cex/Dex)	Não	Não	Não	Sim
Pagamento com MBWay	Sim	Sim	Não	Não	Não	Não	Não	Sim
Pagamento com PayPal	Sim (>30€)	Sim	Sim	Não	Não	Não	Não	Sim
Ver eventos	Sim	Sim	Sim	Não	Sim	Sim	Não	Sim
Controlo do mercado secundário	Não	Não	Não	Sim	Sim	Não	Não	Não
Pontos fortes	No mercado há 25 anos, contratos com lojas físicas conhecidas	Tecnologia TicketNet e TicketNetAcessos, Linha telefónica 24/7 e vários pontos físicos de compra	Plataforma bastante completa com várias soluções para eventos	Bom plano e design	Onboarding para pagamentos com crypto, 0% taxas de transação	-	Boa API para geração dos bilhetes	Possibilidade de escolha de emissão de bilhete *NFT* ou email
Pontos fracos	Interface do utilizador confusa e saturada	-	Muito geral e não aceita pagamentos locais	Não implementado estando numa fase inicial	Implementação generalizada sem possibilidade de customização da plataforma em si	Plataforma em estado muito inicial sem qualquer funcionalidade implementada	Representação dos bilhetes em *NFT* e não realmente os bilhetes em si	Fase inicial de desenvolvimento

Tabela 3.4: Comparação entre as soluções de sistemas de bilhética

3.9 Tecnologias de *Front end* e *Back end*

Nesta secção vão ser exploradas tecnologias de *front end* e *back end* para o desenvolvimento do *software*, analisando a sua taxa de uso e diferenças entre cada tecnologia. Este estudo vai fundamentar as escolhas realizadas no capítulo de decisões tomadas, embora com algumas restrições.

3.9.1 *Front end*

Existem várias escolhas para o cenário de *front end*. De todas as opções foram escolhidas algumas por apresentarem grandes vantagens ao nível de popularidade, apoio da comunidade e facilidade de desenvolvimento.

- **React**: *Framework* de *Javascript/Typescript* baseada em componentes desenvolvida e mantida pela empresa *Facebook*. É bastante popular devido à sua facilidade de aprendizagem e bom desempenho [38]. Uma das principais funcionalidades desta *framework* é a *virtual* DOM com *one-way data binding* que permite operações no documento com uma maior rapidez. Porém, tem algumas desvantagens como documentação relativamente pior que outras e complexidade na aprendizagem da sintaxe *JSX* para a criação dos componentes.

- **Vue.js**: Com mais de 700 000 *websites* desenvolvidos com esta *framework* de *Javascript/Typescript*, *Vue.js* é uma *framework* fácil de utilizar devido à sua documentação e comunidade. Foi criada por *Evan You* que também entrou no desenvolvimento de uma outra *framework* bastante popular chamada *Angular*. Apresenta vantagens como virtual *DOM* com *two-way data binding*, o seu tamanho é relativamente inferior aos seus competidores e apresenta uma sintaxe fácil. Porém é bastante recente (2013) logo, ainda não tem tanto apoio da comunidade que se reflete na falta de *plugins*. Tem também as suas limitações para projetos de larga escala [38].

- **Angular**: Também chamada de *Angular* 2+, é uma *framework* de *TypeScript* bastante popular desenvolvida pela *Google*. Tal como *Vue.js*, suporta *two-way binding* que permite sincronização imediata entre a *view* e o *model* na arquitetura *MVC* (*Model View Controller*). Outra vantagem importante desta *framework* é funcionalidade de *directives* que permitem ao *developer* programar comportamentos especiais no *DOM* tornando possível criar *HTML* dinâmico. Porém, é bastante difícil de implementar e o tamanho da *framework* é grande comparativamente aos seus competidores [38].

Na Tabela 3.5 é feita a comparação entre as três *frameworks* escolhidas.

	React	Vue.js	Angular
Ano de lançamento	2010	2013	2014
Linguagem	JavaScript/Typescript e JSX sintaxe	JavaScript/Typescript	TypeScript
Vantagens	Integração fácil com outras JavaScript libraries. Apoiada pelo Facebook	Pequenas dimensões e fácil aprendizagem.	Facilmente testável. Componentes reutilizáveis. Boa comunidade.
Desvantagens	Documentação pouco elaborada. Complexidade de aprendizagem da sintaxe JSX	Comunidade limitada de developers. Aplicação limitada a projetos de larga escala. Bastante recente.	Aprendizagem difícil e grandes dimensões.

Tabela 3.5: Comparação das tecnologias Front end

3.9.2 *Back end*

Tal como no *front end*, o *back end* também apresenta várias opções para desenvolvimento. Para a escolha de um *back end*, tem que se ter em conta vários aspetos como a dimensão do projeto, a arquitetura do projeto e as capacidades da tecnologia. Foram retiradas quatro opções com base na popularidade e uso [36].

- **Spring Boot**: *Framework lightweight open-source* escrita em *Java* lançada em 2002. É utilizada para criar aplicações *stand-alone* e prontas para produção baseadas em *Spring*. Esta *framework* facilita a configuração *Extensible Markup Language* (XML) complexa de um projeto em *Spring* bem como na gestão de *endpoints* REST. Está pronta também para lidar com uma arquitetura de microsserviços [42].

- **Django**: Lançada em 2005, *Django* é uma das *frameworks open-source* escritas em *Python* mais populares. É baseada no princípio *Don't Repeat Yourself* (DRY) tornando assim possível a reutilização de código. É bastante escalável e personalizável e tem um bom apoio da comunidade e muito boa documentação [42].

- **Laravel**: *Laravel* é uma *framework open-source* escrita em *PHP* lançada em 2011. Esta *framework* é bastante utilizada devido à facilidade de implementação de *blogs*, notícias e *e-commerce* nesta plataforma. Tem o seu próprio *CLI*, sistema de migração de base de dados e suporta código *PHP* normal dentro da implementação. Também conta com um grande apoio da comunidade e boa documentação [42].

- **Quarkus**: *Quarkus* é uma *framework open-source*, *Kubernetes native* aprimorada para *GraalVM* e *OpenJDK HotSpot*. É escrita em *Java*, tal como *Spring Boot*, no entanto, oferece vantagens a nível de velocidade e gestão de memória. *Quarkus* está desenvolvida para se integrar em ambientes *serverless*, *containerized* e *cloud*. Tem outras vantagens como ser construído por cima de *standards enterprise* como *CDI* ou *Jax-rs* e tempos de desenvolvimento mais rápidos através de *hot reloads*. Porém tem também algumas desvantagens como ser relativamente novo (2018) que consequentemente implica

não existirem tantos utilizadores o que leva a pouco apoio da comunidade, e não suportar todos os standards de *EE* como por exemplo, *Enterprise Java Beans* (EJB) [31].

Na Tabela 3.6 é feita a comparação entre as quatro *frameworks* estudadas.

	Spring Boot	Django	Laravel	Quarkus
Ano de lançamento	2002	2005	2011	2018
Linguagem	Java	Python	PHP	Java
Vantagens	Facilita a configuração XML de Spring. Microservices-ready. Gestão de Rest Endpoints.	Escalável e personalizável. Desenvolvimento rápido. Bom apoio da comunidade e boa documentação.	Bom apoio da comunidade e boa documentação. Própria CLI. Sistema de migração de base de dados.	Bom desempenho em termos de startup e gestão de memória. Integração fácil em serverless, cloud ou containers
Desvantagens	Cria muitas dependências que por vezes não são utilizadas. Difícil migrar de Spring para Spring Boot.	Arquitetura muito monolítica. Muitas funcionalidades baseadas no ORM do Django.	Relativamente nova. Qualidade é por vezes diferente. Upgrades podem ser problemáticos.	Framework bastante nova. Falta de suporte a alguns EE standards.

Tabela 3.6: Comparação das tecnologias Back end

3.10 Decisões tomadas

Nesta secção vão ser expostas as escolhas feitas pelo aluno para cada tecnologia a utilizar no desenvolvimento do projeto, de acordo com o estudo feito e relatado neste capítulo.

Para o *front end*, o aluno decidiu optar por *Vue.js*. *Vue.js* é uma *framework* que é bastante popular e consegue as vantagens do *Angular* com um menor esforço de aprendizagem. Dito isto, também existe a oportunidade de manter a tecnologia já utilizada no projeto visto que assim o aluno tem mais apoio dos membros da equipa que estão familiarizados com o código implementado e conseguem fornecer ajuda mais rápida e eficaz.

Para o *back end*, o aluno teve total liberdade na escolha da tecnologia visto que o projeto segue uma arquitetura de microsserviços. O aluno escolheu a *framework* Quarkus que mantém o ambiente Java familiarizado por uma grande parte de colaboradores na empresa. *Quarkus* foi escolhido em vez de *Spring Boot* por questões de desempenho mencionadas na secção anterior (Ver Secção 3.9.2).

A *gateway* de pagamento escolhida foi a *EuPago* uma vez que tem uma boa documentação para a integração na plataforma e suporta mais métodos de pagamento de uma maneira simples e eficaz.

Em relação à tecnologia *blockchain*, o aluno optou pela plataforma **Polygon** para desenvolvimento dos *smart contracts* necessários, utilizando **Moralis** como plataforma de desenvolvimento de modo a facilitar a integração com o *front end* e aquisição dos dados de transações já agregados. *Polygon* é a melhor escolha devido às suas baixas taxas de transação simultaneamente oferecendo acesso a todo o ecossistema *Ethereum* que vem com um grande suporte da comunidade. *Moralis* foi escolhido por ter um grande apoio para a integração através de guias e uma boa documentação.

Capítulo 4

Requisitos

O planeamento da arquitetura de *software* necessita de uma boa definição dos requisitos a implementar. Para este efeito, foram levantados os requisitos das funcionalidades e especificados em forma de ***user stories***.

Neste capítulo vão ser expostas todas estas *user stories* escritas pelo *product owner* e pelo estagiário, derivar essas *user stories* para requisitos funcionais e elaborar os diagramas de caso de uso.

4.1 User stories

Para a especificação dos requisitos da plataforma, a empresa opta por descrever as funcionalidades de acordo com as necessidades do utilizador. Uma funcionalidade é descrita num formato fixo em que o objetivo é definir o tipo de utilizador que pode cumprir uma determinada ação com um certo objetivo em mente.

Uma *user story* é definida então do seguinte modo:

"Como um <tipo de utilizador> posso <ação> para que <objetivo>"

De seguida vão ser apresentadas as *user stories* para o sistema de bilhética e gestão de pagamentos da Sala-Z, porém, antes de passar à apresentação das mesmas é importante referir quais os tipos de utilizador que a plataforma vai passar a lidar. Também é necessário dar uma definição de *venue*. Uma *venue* é um local que está preparado para receber artistas de modo que estes sejam capazes de atuar diante um público. Com isto em mente, os utilizadores neste momento na plataforma são:

- **Administrador da Sala-Z** que lida com toda a gestão de utilizadores e espaços.
- **Utilizador de uma *Venue*** que lida com os eventos da *venue* atribuída.

A plataforma vai passar a integrar outros tipos de utilizador:

- **Administrador de *Venue*** que vai gerir toda a informação relacionada com os espaços e pode adicionar utilizadores como colaboradores de uma *venue*. O antigo utilizador de uma *venue* vai ser repartido em dois utilizadores sendo este um deles e o outro o colaborador de *venue*.
- **Colaborador de *Venue*** que pode alterar informação e participar na organização de eventos numa *venue*.
- **Administrador de artistas** que vai gerir toda a informação relacionada com os artistas na plataforma e pode adicionar utilizadores como colaboradores de artistas.
- **Colaborador de artista** que pode alterar informação sobre um artista.
- **Administrador do Sala-Z** que foi mencionado anteriormente mantendo todos os privilégios sobre a plataforma.
- **Utilizador não autenticado** que apenas tem acesso à informação pública do Sala-Z, conseguindo ver a lista de espaços, eventos, artistas e pode reservar bilhetes para os eventos.

Está planeado que o aluno trabalhe apenas com os utilizadores **Administrador de *Venue*** e **Utilizador não autenticado** sendo que os outros tipos de utilizadores vão ser implementados pelo outro estagiário no projeto.

Com o tipo de utilizadores definidos é possível especificar as *user stories*. Devido à grande quantidade de *user stories* produzidas, estas foram colocadas no Apêndice A por completo mostrando aqui algumas por motivos de exemplificação.

US-2: **Como um** utilizador não autenticado **quero** visualizar uma lista de destaques **para** saber quais os eventos mais procurados e interessantes na plataforma.

US-3: **Como um** utilizador não autenticado **quero** procurar por um evento **para** conseguir encontrar facilmente um evento específico.

US-4: **Como um** utilizador não autenticado **quero** filtrar os eventos disponíveis por tipo de evento **para** encontrar eventos do meu interesse. Quero filtrar por um ou mais tipos de evento em simultâneo, sendo estes:

- Evento gratuito
- Festival
- Concerto
- Lazer

É de salientar que as *Epic Stories* (ES) fornecidas fazem parte de um documento onde são especificadas funcionalidaeds para toda a plataforma. Logo, no Apêndice A as *Epic Stories* começam a partir da *ES-31* e vão até à *ES-37* que correspondem às funcionalidades do estágio do aluno.

4.2 Requisitos Funcionais

Tendo como base as *user stories* definidas na secção anterior, é possível levantar os requisitos funcionais da nova plataforma. Para classificar todos os requisitos foram atribuídos aos mesmos diferentes graus de prioridade. Estes graus foram definidos através do método de *MoSCoW* sendo apresentados de seguida.

- **Must Have** (M) – Prioridade máxima, tendo que estar presente no produto final.
- **Should Have** (S) – Prioridade média, indicando que o produto final deve ter a funcionalidade descrita.
- **Could Have** (C) – Prioridade baixa, indicando que é uma mais-valia se o produto final implementar a funcionalidade descrita.
- **Won't Have (W)** – Prioridade mínima, indicando que a funcionalidade descrita não vai ser implementada, porém, no futuro pode vir a ser.

Todos os requisitos para os objetivos do estágio estão representados na Tabela 4.1.

4.3 Requisitos Não Funcionais

Os atributos de qualidade de uma plataforma também são um fator importantíssimo para o sucesso de um produto. Um requisito não funcional é uma característica intrínseca de um sistema que define certas qualidades que o mesmo tem que ter, descrevendo também como é que o sistema se deve comportar. Para este sistema, o atributo de qualidade mais importante é o de segurança, pois uma aplicação tem que garantir um certo nível de segurança sempre que contenha funcionalidades que envolvam pagamentos e dados pessoais. De seguida estão descritos os requisitos não funcionais para a plataforma Sala-Z.

4.3.1 Segurança

Segurança é uma medida da habilidade de um sistema de proteger os dados e informação de acessos indevidos, continuando a fornecer acesso aos utilizadores e sistemas que estão autorizados para tal. Confidencialidade, integridade e disponibilidade são três qualidades de uma boa segurança. Visto que a plataforma vai lidar com *Personally Identifiable Information* (PII), o sistema tem que ter este requisito e cumprir com as normas do *General Data Protection Regulation* (GDPR). Na Tabela 4.2 temos descrito o cenário de segurança.

Tema	Requisito	Prioridade
Portal público	Ver lista de próximos eventos	(M)
	Ver lista de eventos destaque	(M)
	Procurar evento	(M)
	Filtrar eventos	(M)
	Ordenar eventos	(S)
	Ver evento	(M)
Compra de bilhetes	Selecionar evento	(M)
	Reservar lugares	(M)
	Eliminar lugares reservados	(M)
	Ver preço da reserva	(M)
	Pagamento dos bilhetes por cartão de crédito	(S)
	Pagamento dos bilhetes por MBWay	(S)
	Pagamento dos bilhetes por PayPal	(S)
	Pagamento dos bilhetes por cripto moedas	(C)
	Conectar crypto wallet	(C)
	Emitir fatura e enviar por email	(M)
	Autorização de recolha de dados	(M)
	Acesso aos termos e condições	(M)
	Ver resumo da reserva	(M)
	Editar informação da reserva	(M)
	Ver resumo da compra finalizada	(S)
Gestão dos bilhetes	Ver bilhetes comprados tradicionalmente no email	(M)
	Ver bilhetes comprados por *NFT* na crypto wallet	(S)
	Emitir bilhetes *NFT*	(M)
	Filtrar bilhetes *NFT*	(S)
	Ver estado dos bilhetes emitidos (Vendidos, por vender, ou revendidos)	(S)
	Ver mapa dos bilhetes vendidos	(C)
Gestão dos eventos	Adicionar eventos	(M)
	Editar eventos	(M)
	Ver últimas atuações no espaço	(C)
	Configurar tipos de bilhetes	(M)
	Configurar preço dos bilhetes	(M)
	Adicionar informação de custos e *revenue*	(S)
	Publicar um evento no portal público	(M)
	Adicionar colaboradores	(S)
	Remover colaboradores	(S)
	Ver lista de próximas tarefas	(C)
Estatísticas	Ver estatísticas de venda dos bilhetes	(S)
	Ver estatísticas das métricas dos eventos	(S)
	Ver atividade recente dos eventos	(S)
	Ver atividade recente de um evento	(S)
	Ver estatísticas relacionadas com os artistas	(S)

Tabela 4.1: Requisitos funcionais do sistema de bilhética e gestão de pagamento

Fonte	Outro sistema
Estímulo	Acesso não autorizado a dados privados
Artefacto	Base de dados
Ambiente	Completamente operacional
Resposta	Dados são protegidos
Medida da resposta	Precisão da deteção das tentativas

Tabela 4.2: Atributo de qualidade de segurança

Fonte	Desenvolvedor
Estímulo	Adição, modificação ou remoção de funcionalidades na plataforma
Artefacto	*Codebase* da plataforma
Ambiente	Desenvolvimento do *software* em qualquer fase
Resposta	A mudança é efetuada com sucesso
Medida da resposta	Tempo e custo da mudança

Tabela 4.3: Atributo de qualidade de modificabilidade

4.3.2 Modificabilidade

Modificabilidade é uma medida da habilidade de um sistema poder integrar, alterar ou retirar funcionalidades novas ou velhas, corrigir erros ou melhorar a segurança e a performance de uma maneira fácil com o menos risco e custos possíveis. Na Tabela 4.3 podemos ver o cenário para este atributo.

4.4 Diagramas de Caso de Uso

Após a realização dos requisitos, é possível gerar os diagramas de caso de uso. As funcionalidades a implementar vão utilizar dois tipos de utilizador: utilizador não autenticado e administrador de *venue*.

Na Figura 4.1 é apresentado o primeiro diagrama onde são especificadas as ações de um utilizador não autenticado no portal público. Neste local o utilizador deve conseguir ver a lista de eventos da plataforma podendo pesquisar, filtrar e ordenar a lista. Caso queira ver os detalhes de um evento pode fazê-lo. Também é possível ver outra lista de eventos que contém os eventos em destaque na plataforma.

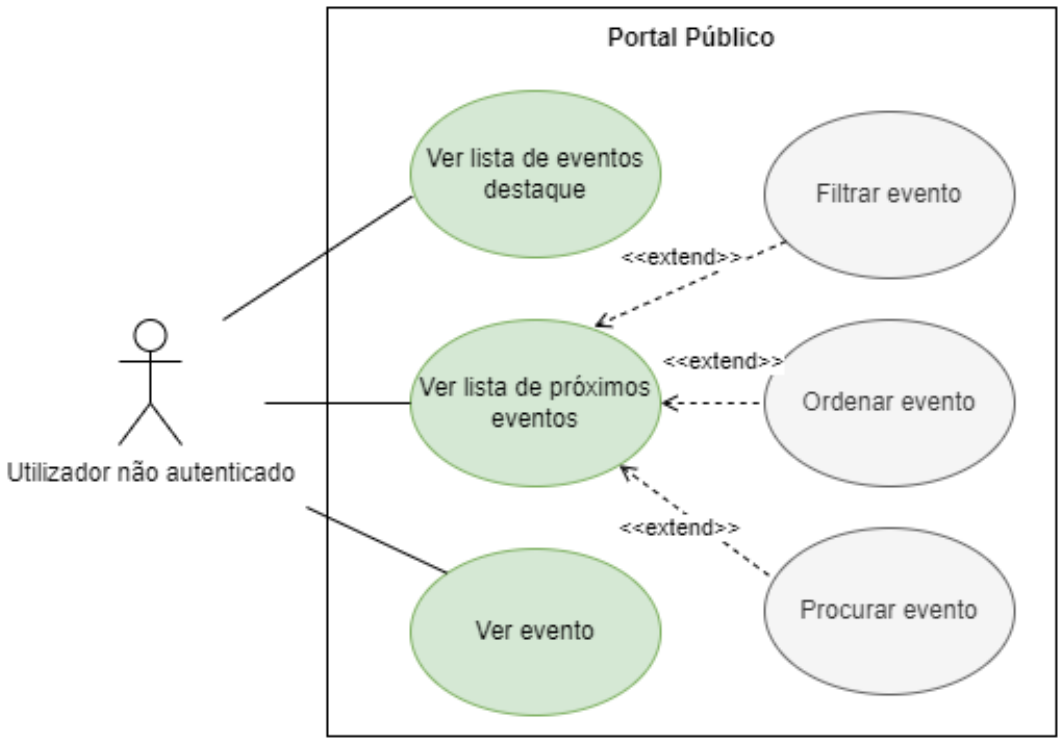


Figura 4.1: Diagrama de caso de uso 1: Portal público

Na Figura 4.2 é apresentado o segundo diagrama que descreve as ações relacionadas com a compra de bilhetes por parte de um utilizador não autenticado. Um utilizador vai poder selecionar um evento de modo a obter informação sobre o mesmo, reservar ou remover lugares durante a reserva sabendo sempre o preço da mesma. Após inserir os detalhes de faturação pode prosseguir para o pagamento tendo várias opções. Para os utilizadores que queiram bilhetes *NFT*, será possível conectar a *crypto wallet*. Por fim irá receber no *email* a fatura do bilhete e visualizar um resumo da compra na plataforma.

Na Figura 4.3 é apresentado o terceiro diagrama que refere às ações de gestão de bilhetes. O utilizador não autenticado vai poder ver os bilhetes que comprou na plataforma sejam estes bilhetes tradicionais via *email* ou *NFTs*. O administrador de *venue* autenticado poderá emitir os bilhetes para um evento, visualizar estes bilhetes e filtrá-los. Também terá acesso a estatísticas relevantes sobre os bilhetes emitidos conseguindo ver um mapa de bilhetes de modo a conseguir acompanhar a venda.

Na Figura 4.4 é apresentado o quarto diagrama que descreve a gestão de eventos. O administrador de *venue* autenticado irá poder adicionar e editar eventos, publicar um evento de modo que os utilizadores não autenticados consigam vê-los na plataforma e adicionar e remover colaboradores de evento. Além disto, também poderá definir os tipos e preços dos bilhetes para o evento. Para a gestão do evento ficar finalizada é necessário também adicionar os custos do evento no geral, bem como a receita, tendo em conta o *cachet* dos artistas.

Figura 4.2: Diagrama de caso de uso 2: Compra de bilhetes

Por último, na Figura 4.5 encontra-se o quinto diagrama que se refere à informação de estatísticas na plataforma. O administrador de *venue* autenticado terá um espaço com várias estatísticas em relação aos seus eventos como audiência, presença em eventos, lista de artistas ordenada por receita, tarefas do evento concluídas e por realizar, entre outros. Também haverá um espaço onde poderá ver a atividade recente de um evento como adição de *staff* e tarefas.

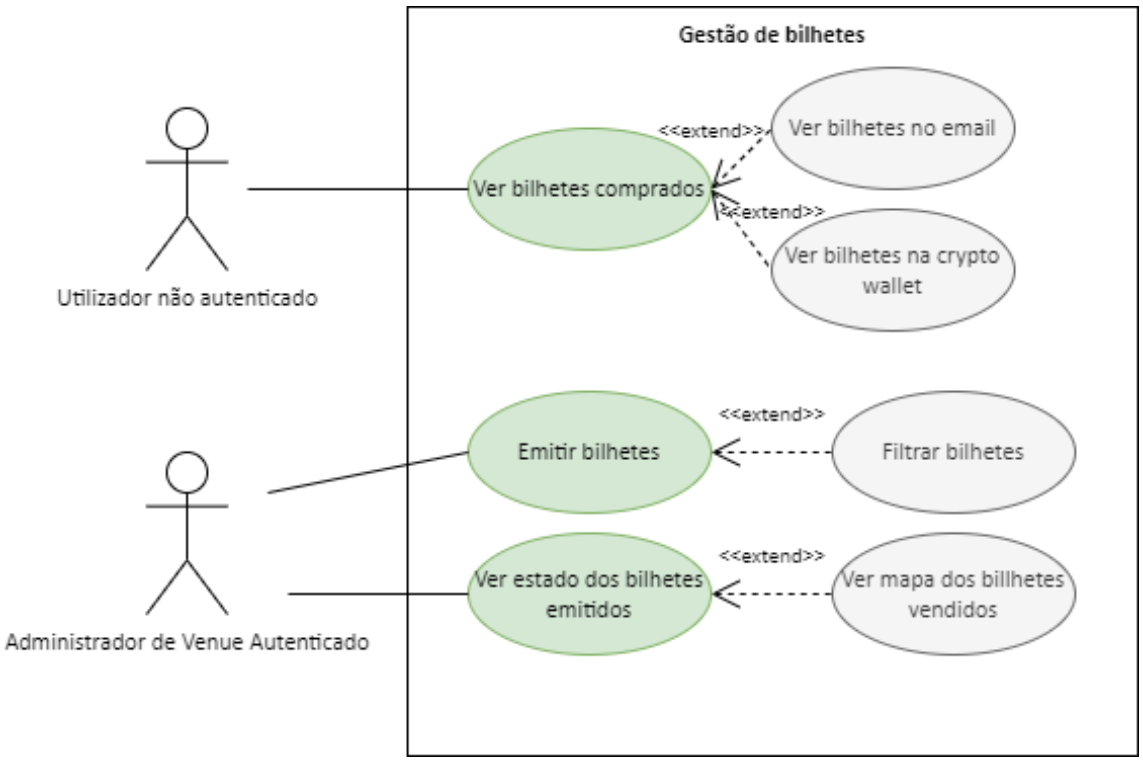


Figura 4.3: Diagrama de caso de uso 3: Gestão de bilhetes

Figura 4.4: Diagrama de caso de uso 4: Gestão de eventos

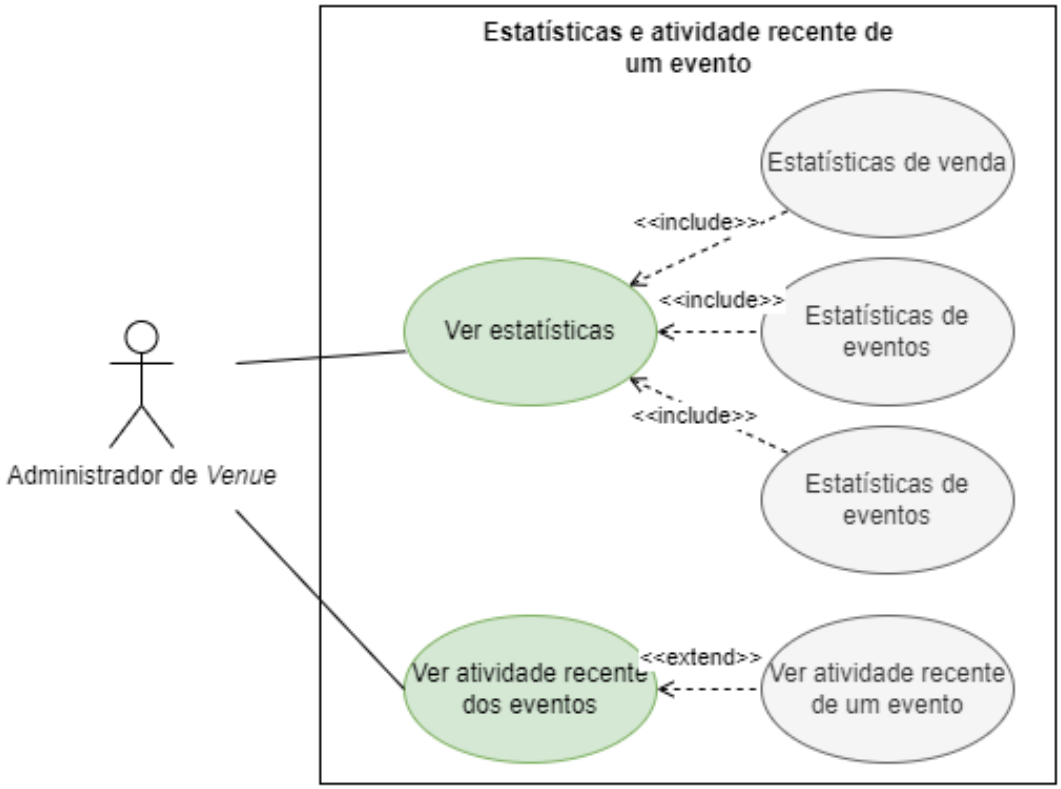


Figura 4.5: Diagrama de caso de uso 5: Estatísticas e atividade recente de um evento

Capítulo 5

Arquitetura de Software

Neste capítulo vai ser discutida a arquitetura de *software* proposta pelo aluno, de acordo com os requisitos funcionais produzidos e apresentados no capítulo anterior. A arquitetura foi realizada usando o modelo C4, por ser bastante intuitivo e por ter sido já previamente utilizado na definição da arquitetura da aplicação existente.

O Sala-Z, como informado previamente, não é um projeto novo tendo já funcionalidades implementadas como a parte de autenticação de utilizadores e gestão de salas de espetáculo. Nos diagramas que vão ser mostrados mais à frente, estão descritas as implementações existentes diferenciadas por cor. Também é importante referenciar que o estagiário trabalhou com outro estagiário da universidade no Sala-Z, porém com tarefas diferentes. Algumas das funcionalidades foram implementadas num microsserviço partilhado pelos dois. A comunicação entre microsserviços foi planeada e sincronizada entre os dois estagiários.

Uma vez que a plataforma tem objetivos de suportar eventos para todas as localidades de Portugal, estima-se que a plataforma possa ser utilizada por cerca de 1 milhão de portugueses, de acordo com as estatísticas recolhidas em 2020 pelo Instituto Nacional de Estatística [32]. Para suportar este tipo de atividade, a arquitetura tem que ter em conta a escalabilidade e performance.

Dito isto, a arquitetura vai seguir um modelo de microsserviços, uma vez que é uma arquitetura desenhada para ser escalável, e vai ser implementada através dos serviços disponibilizados pela *Amazon Web Services* (AWS). A AWS oferece uma grande variedade de serviços para construção de aplicações, sendo os que vão ser utilizados no decorrer do segundo semestre são:

- ***Elastic Compute Cloud* (EC2)** – Serviço *web* que disponibiliza capacidade computacional segura e redimensionável na *cloud*. É projetado para facilitar a computação em *cloud* na escala da web para os desenvolvedores [21]. Vai ser utilizado para alojar o back end da aplicação.

- **Cognito** – Serviço de autenticação da AWS que permite facilmente adicionar funcionalidades de *Sign-In*, *Sign-Up* e controlo de acesso para aplicações *web* ou *mobile* [20]. É já utilizada para a autenticação na plataforma.

- ***Simple Storage Service* (S3)** - Serviço de armazenamento de objetos que oferece escalabilidade, disponibilidade de dados, segurança e performance líderes do setor [24]. Vai ser utilizado para armazenar todo o conteúdo estático da plataforma.
- **API Gateway** – Serviço completamente gerido por si que facilita a construção de *REST APIs* e *WebSocket APIs*, bem como monitorização, modificação e segurança destas *APIs* [19]. Vai ser utilizado pelo front end para comunicar com o back end.
- ***Elastic Load Balancing* (ELB)** – Serviço que automaticamente distribui tráfego da aplicação para vários destinos numa ou mais *Availability Zones* (AZ) [22]. Já é utilizado pela aplicação existente para distribuir carga pelas várias instâncias EC2.
- ***Simple Email Service* (SES)** – Serviço de email eficiente, flexível e escalável que permite aos desenvolvedores enviar emails a partir de qualquer aplicação [25]. Vai ser utilizado para enviar os emails com os bilhetes e faturas.
- ***Relational Database Service* (RDS)** – Serviço que facilita a integração, operação e escalabilidade de uma base de dados relacional para ser utilizada pelas aplicações [23]. Vai ser utilizada para guardar a informação das compras dos utilizadores finais e eventos, bem como qualquer informação relevante que necessite de ser guardada.
- ***Simple Notification Service* (SNS)** – Serviço completamente gerido por si mesmo de *messaging* para tanto comunicação *Application-to-Application* (A2A) como comunicação *Application-to-Person* (A2P) [26]. Vai ser utilizada para a comunicação entre microsserviços na aplicação.
- ***Simple Queue Service* (SQS)** – Serviço completamente gerido por si mesmo de *queuing* que permite abstrair e escalar a comunicação entre microsserviços, sistemas distribuídos e aplicações *serverless* [27]. Vai ser utilizado no microsserviço *Notifications* para conseguir uma escalabilidade maior.

5.1 Modelo C4

O modelo C4 [29] é um modelo de arquitetura de *software* desenvolvido por Simon Brown com base em *Unified Modeling Language* (UML) e 4+1 *architectural view model*. Trata-se de repartir o software em quatro níveis e ir de um nível mais amplo para um nível mais detalhado. Os quatro níveis são: **contexto**, **containers**, **componente** e **código**. De seguida, vão ser detalhados os níveis e qual o foco de cada um.

5.1.1 Contexto

Este nível é o mais amplo dos níveis e é onde é apresentada a aplicação como um todo, descrevendo possíveis comunicações com sistemas externos. Também se

identificam os utilizadores da aplicação (ver Figura 5.1).

Para a aplicação, de acordo com os requisitos levantados, as funcionalidades vão utilizar dois tipos de utilizadores que são:

- **Utilizador não autenticado** – Utilizador que vai procurar por eventos e comprar bilhetes.
- **Utilizador de *venue* autenticado** – Utilizador que gere salas de concertos, eventos e emite os bilhetes na plataforma.

Blockchain, mais propriamente *Polygon* juntamente com *Moralis*, AWS e *EuPago* vão ser usados como sistemas externos.

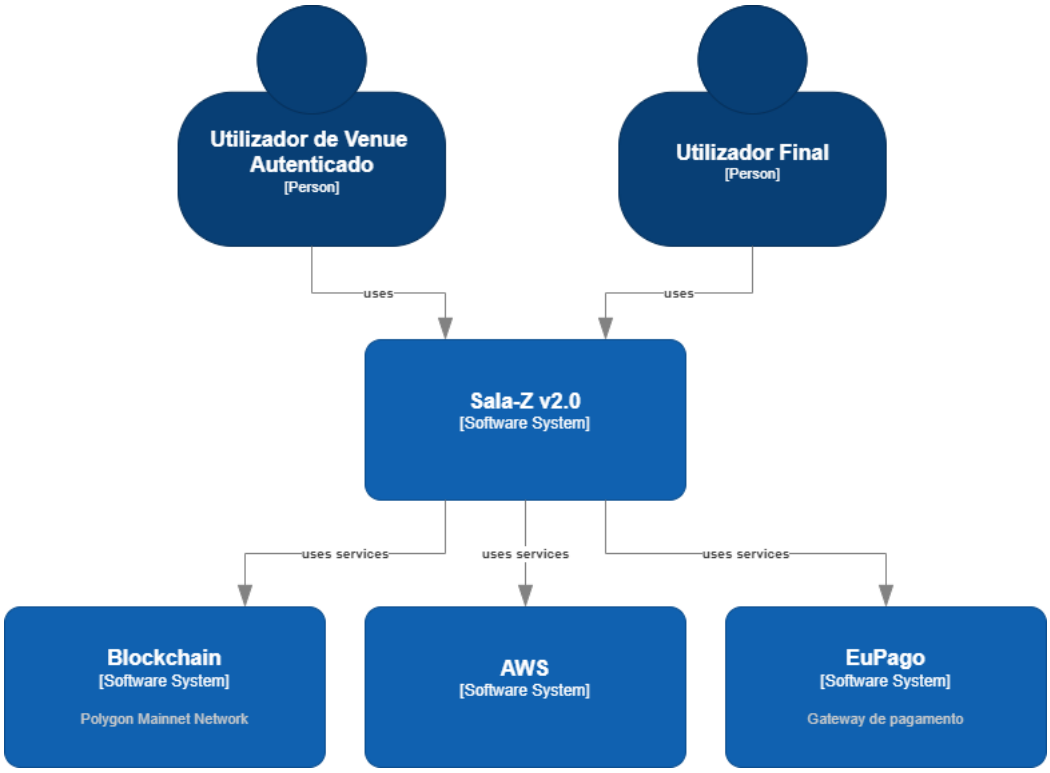


Figura 5.1: Arquitetura de software: Nível do Contexto

5.1.2 Containers

Neste nível é descrito em detalhe cada objeto do nível anterior, originando *containers*. Especifica-se também quais as relações entre os *containers* de cada sistema de modo a perceber como é que o sistema no todo funciona (ver Figura 5.2).

Os utilizadores têm acesso à *Single Page Application* que vai buscar os dados por *API* aos microsserviços em *Quarkus*. Como é mostrado na Figura 5.2, existe uma *API Gateway* que comunica com o ELB que redistribui a carga pelos vários microsserviços. Estes microsserviços têm cada um uma instância EC2. Neste diagrama são mostrados os microsserviços já existentes na plataforma a **branco**, os microsserviços novos a implementar pelo aluno a **verde**, e os microsserviços novos a

implementar com o estagiário que vai trabalhar simultaneamente no projeto a **roxo**.

O microsserviço ***Event Management*** é responsável por lidar com toda a informação relativa aos eventos como criar um evento, editar e adicionar colaboradores. Vai comunicar com o microsserviço ***Venue Management*** que já existe de modo a conseguir obter os dados relativos à *venue* do evento. Esta comunicação é feita através do serviço AWS SNS. Qualquer comunicação necessária com outros microsserviços também será permitida através deste serviço.

O microsserviço ***Ticket Management*** lida com toda a parte de geração de bilhetes, gestão de pagamento e emissão de faturas e criação de *NFTs*. Vai comunicar com o *Event Management*.

O microsserviço ***Notifications*** é responsável por emitir notificações na aplicação e enviar emails conforme as mensagens que receber do *Ticket Management*. Para este microsserviço, como vai estar em contacto com quase todos os microsserviços, ficou decidido ser implementado um *queuing service* oferecido pela AWS SQS que permite acumular as mensagens trocadas no SNS, oferecendo assim mais escalabilidade.

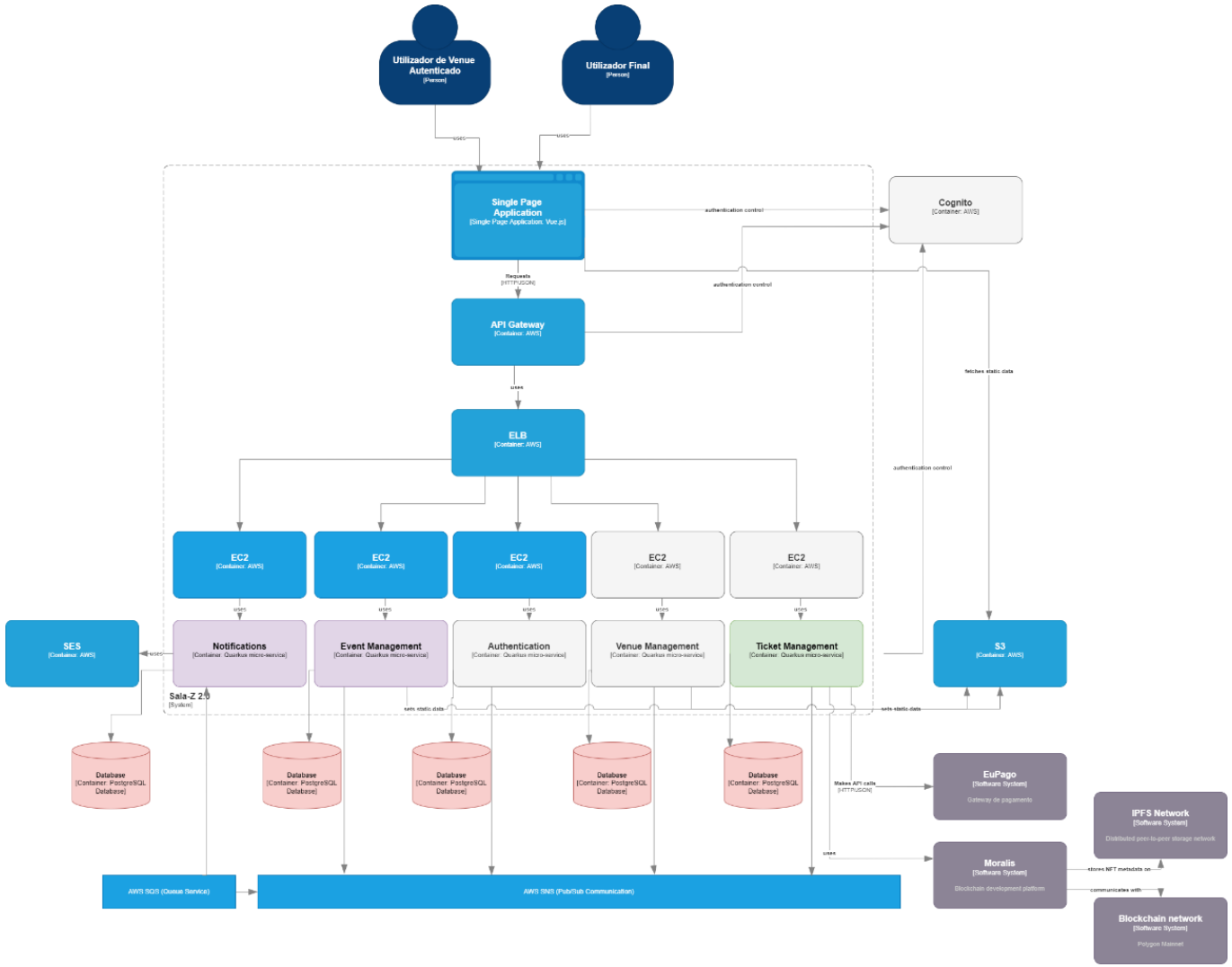


Figura 5.2: Arquitetura de software: Nível dos Containers

5.1.3 Componentes

Neste nível repartem-se os containers em componentes para perceber o funcionamento interno de cada *container*. Os microsserviços vão ser detalhados bem como as interações entre os sistemas externos.

No microsserviço ***Event Management*** vão existir quatro componentes. O componente de evento vai lidar mais com a parte de servir a informação para o *front end* tanto que está ligada ao SES para introduzir conteúdo estático. Informação como listagem de eventos e informação individual são exemplos da responsabilidade deste componente. O componente *Ticket Generation* vai ser responsável por gerar os lugares disponíveis dependendo da sala de espetáculo e do evento e produzir o mapa de lugares. O componente *Reservation* vai acompanhar todo o processo de escolha de lugares para um evento e por último, o componente *Statistics* vai produzir as estatísticas relacionadas com os eventos. Na Figura 5.3 podemos ver os componentes do *container Event Management*.

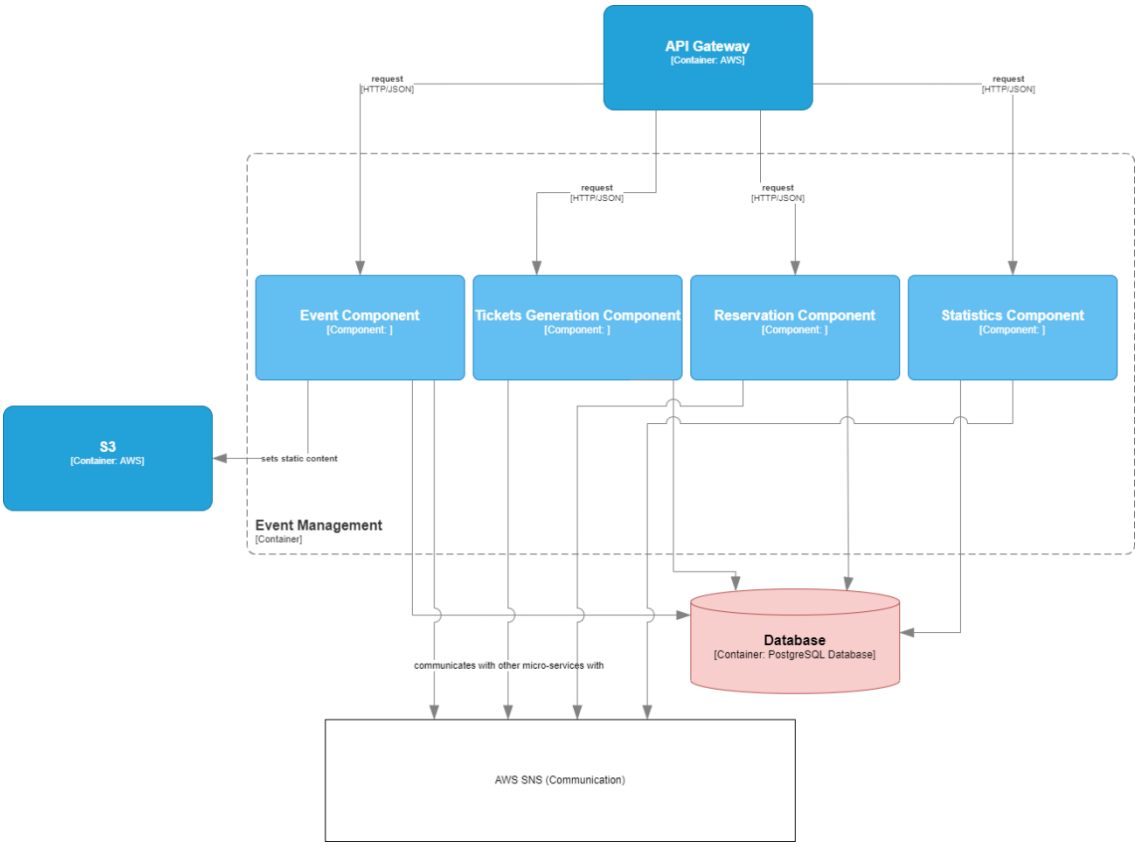


Figura 5.3: Arquitetura de software: Nível dos componentes (*Event Management*)

No microsserviço ***Ticket Management*** vão existir dois componentes. O componente *Payment* vai lidar com a gestão de pagamento de uma reserva para um evento e está responsável por comunicar com a API do *gateway* de pagamento. Após um pagamento completo com sucesso, este vai comunicar com o microsserviço *Notifications* para enviar os bilhetes ao comprador. Se o comprador optar por bilhetes *NFT*, o componente *NFT* entra em ação. Este está responsável pela emissão e gestão de bilhetes *NFT* e comunica com a plataforma de desenvolvimento

blockchain. Na Figura 5.4 podemos ver os componentes do container Ticket Management.

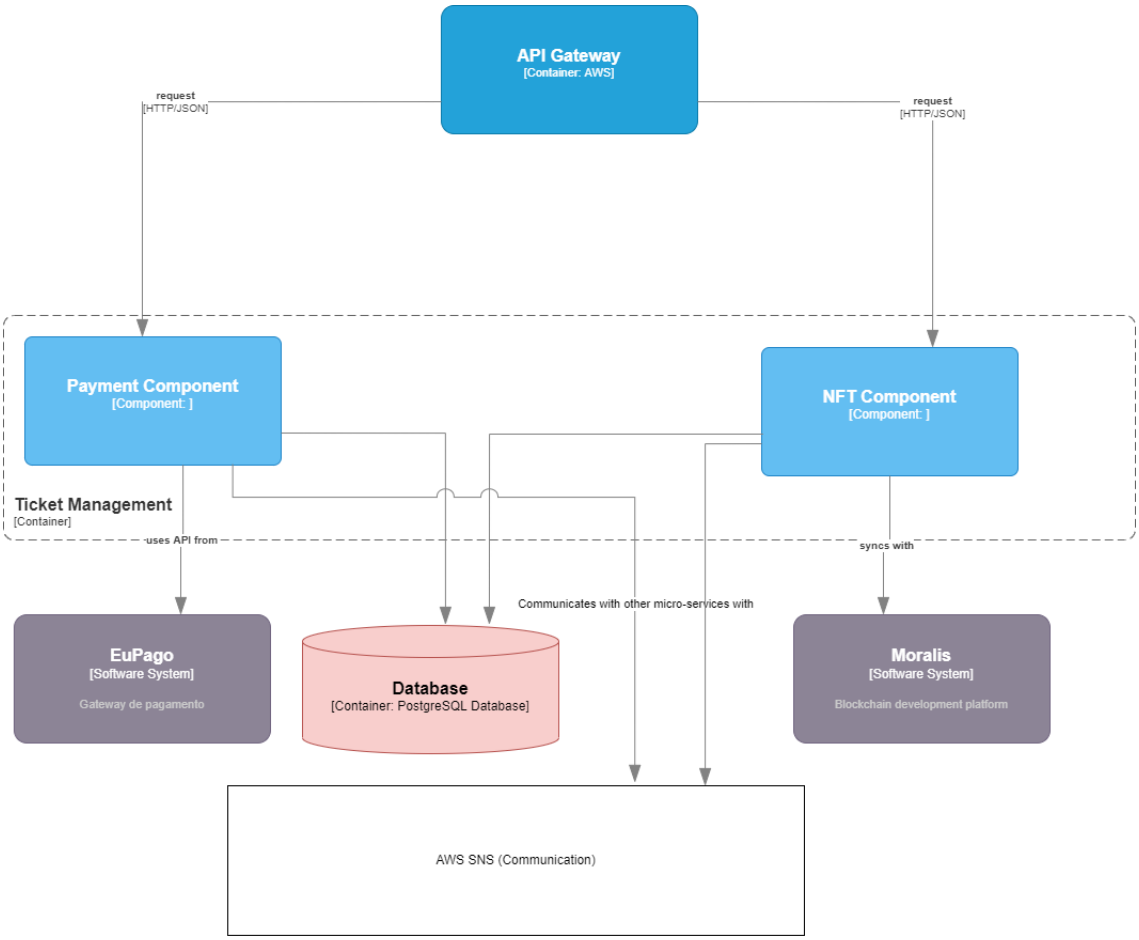


Figura 5.4: Arquitetura de software: Nível dos componentes (*Ticket Management*)

No microsserviço ***Notifications*** vão existir dois ou mais componentes. O componente que vai ser utilizado pelo aluno será o *Ticket Notifications* e estará responsável por escutar mensagens do microsserviço *Ticket Management* para saber se necessita de enviar emails com bilhetes através da ligação com a AWS SES. Este microsserviço vai ser partilhado com o outro estagiário e vai ter mais componentes para lidar com as notificações da aplicação. Na Figura 5.5 podemos ver os componentes do *container Notifications*.

5.1.4 Código

Visto que pela documentação do modelo C4 [29] este nível é opcional devido à grande necessidade de detalhe numa fase inicial do projeto, este nível não vai ser produzido pelas razões mencionadas.

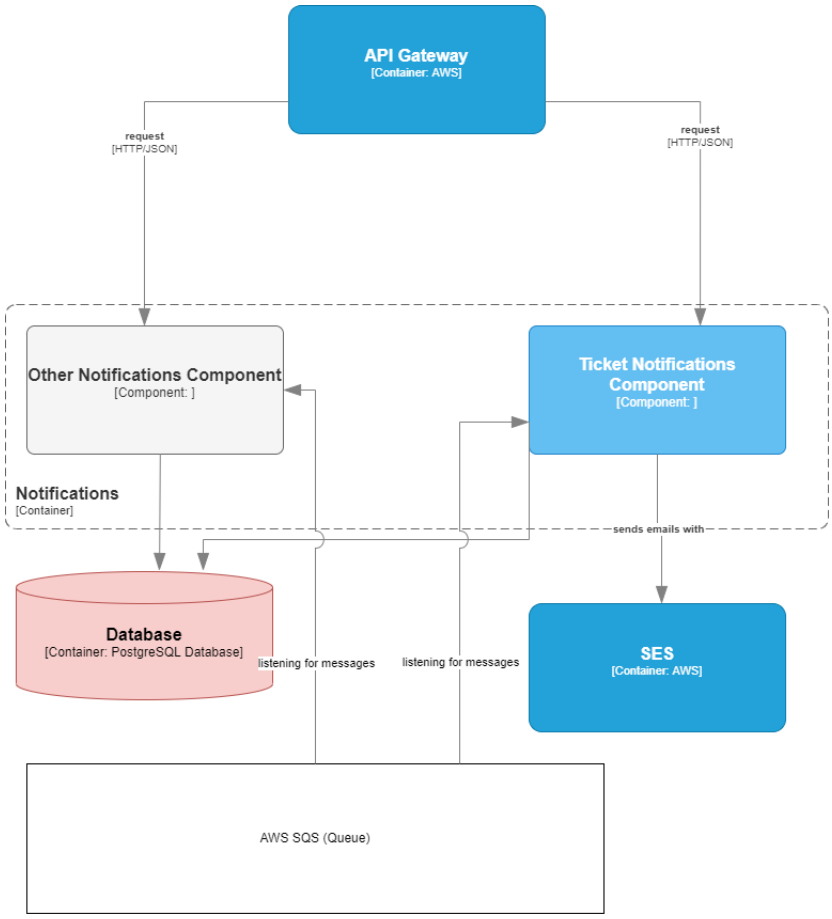


Figura 5.5: Arquitetura de software: Nível dos componentes (*Notifications*)

5.2 Diagrama Entidade-Relacionamento

De modo a perceber como os dados que vão estar presentes no projeto serão guardados e organizados foi elaborado um diagrama Entidade-Relacionamento (ER). Um diagrama ER é um tipo de *flow chart* que ilustra a maneira como as várias entidades no projeto se relacionam umas com as outras. Neste caso, estas entidades vão refletir os modelos ou tabelas da base de dados.

Podemos ver na Figura 5.6 três entidades (excluindo a entidade *Event* uma vez que não foi aluno a desenhar ou implementar esta tabela):

- **Ticket** - Esta entidade representa o bilhete na plataforma. Contém informação de localização do lugar no evento, o preço, se está reservado ou não e qual o link para a representação em *pdf* do bilhete no *bucket* s3 do sistema.
- **Buyer** - Esta entidade representa o comprador dos bilhetes na plataforma. Contém toda a informação pessoal do comprador como nome, email, número de telemóvel, nif e endereço. Também contém o método de pagamento utilizado para o pagamento da reserva na aplicação e campos adicionais com informação suplementar destas escolhas. Por fim, tem informação sobre a *crypto wallet* e se o comprador quer receber um *NFT* ou não.

- **Ticket Token Id** - Esta entidade é uma tabela de apoio à construção de *NFTs*. Cada *NFT* irá ter um *id* que representa o *id* real no *smart contract* (ver Secção 6.3.3). Desta maneira a *metadata* destes *ids* e os próprios *ids* ficam juntos em registos.

Tanto o *Ticket* como o *Buyer* têm uma relação de *One-To-Many* para o *Event* no sentido em que um evento pode ter zero ou mais bilhetes e zero ou mais bilhetes, um bilhete tem um e um só evento e um comprador tem um e um só evento. Entre um bilhete e um comprador existe também uma relação *One-To-Many* no sentido em que um bilhete não tem comprador ou tem apenas um e um comprador tem um ou mais bilhetes. Por fim, existe a mesma relação entre o *Ticket* e o *Ticket Token Id* no sentido em que um bilhete pode ter zero ou um registo na tabela *Ticket Token Id* e um registo *Ticket Token Id* pode ter zero ou mais bilhetes.

O leitor pode questionar porque é que não se pode simplesmente inserir os dois campos do *Ticket Token Id* (o *id* e o *ipfs_link* correspondente à *metadata*) em cada bilhete. A resposta é que ao permitir eventos sem lugares marcados na plataforma, esta informação seria replicada igualmente para cada bilhete desse evento uma vez que o *NFT* desse evento seria igual para todos os compradores, ou seja, teria o mesmo *id* no *smart contract* e consequentemente, a mesma *metadata*. Com esta razão, é justificável construir esta tabela para reduzir o número de campos iguais na entidade do bilhete.

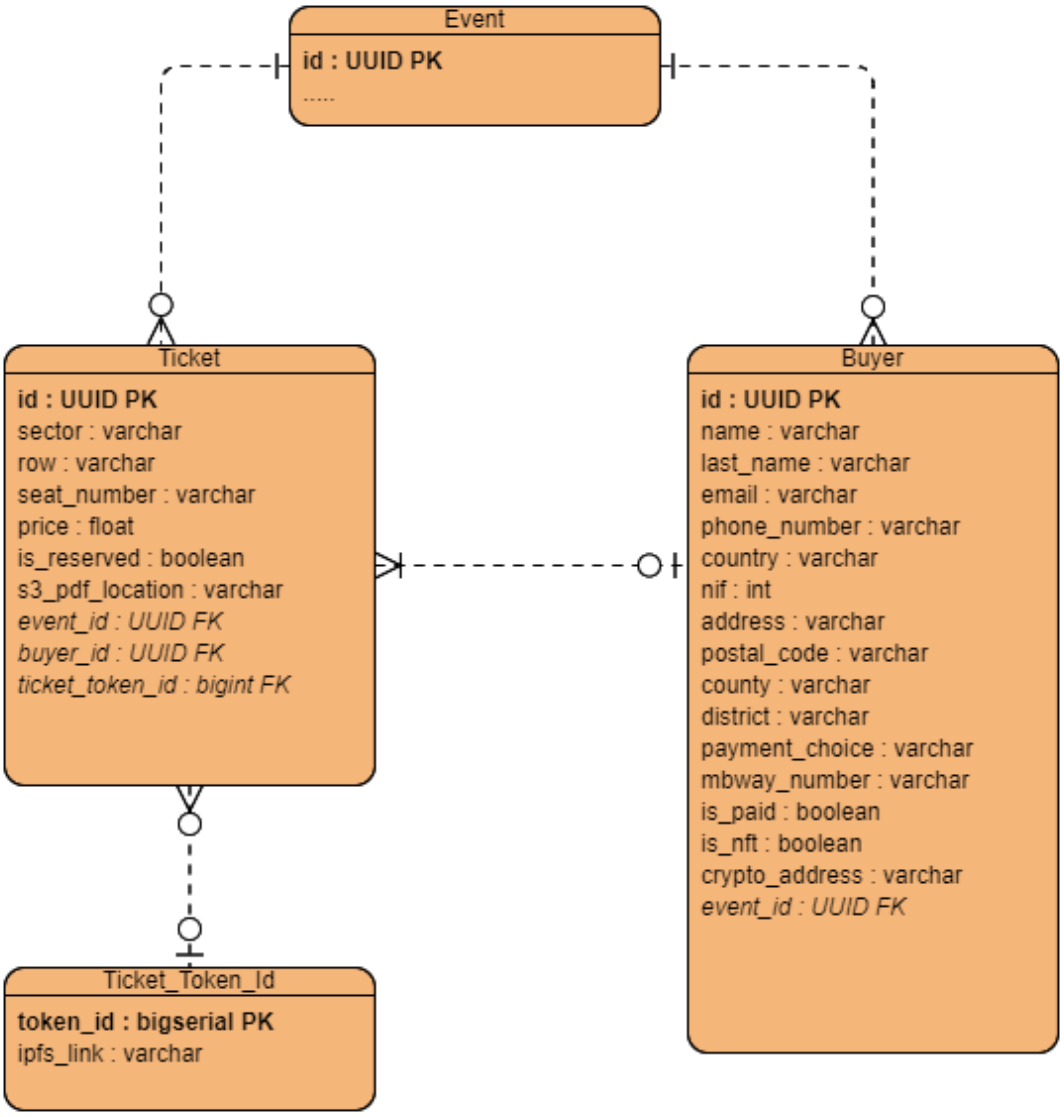


Figura 5.6: Diagrama Entidade-Relacionamento do sistema

Capítulo 6

Desenvolvimento

Neste capítulo é apresentada a fase de desenvolvimento do projeto, o estado do projeto atual face ao trabalho do aluno e aos requisitos levantados, os riscos materializados e os testes realizados.

6.1 Processo

O processo de desenvolvimento do projeto foi constituído por duas fases. É de salientar que todas as *mockups* das funcionalidades levantadas já estavam concluídas pela equipa de *design* do projeto, logo, o aluno só teve que desenvolver a funcionalidade de acordo com o design e com a especificação do *user story*.

A primeira fase foi o desenvolvimento da funcionalidade tratando da construção da interface bem como o suporte no *back end*. Após a conclusão da tarefa, o código seguia em *merge request* para o repositório oficial do projeto onde passava por uma fase de *code review* por parte de membros da empresa. Após aprovação, o novo código integrava o código existente e era marcado como *Ready to Test*. Este processo foi realizado na plataforma *GitLab* com auxílio da aplicação de gestão de projetos *Linear*, como foi mencionado no capítulo de planeamento.

A segunda fase foi a fase de testes da aplicação que ocorreu no penúltimo *sprint*. Cada uma das funcionalidades *Ready to Test* foram testadas e como resultado foi produzido uma tabela de *bugs* encontrados durante esta fase de testagem. Simultaneamente, estes erros foram corrigidos para dar a funcionalidade como completa.

6.2 Estrutura do código

A estrutura do código é um ponto bastante importante em qualquer projeto uma vez que influencia diretamente a qualidade de manutenção do mesmo. Como o Sala-Z já era um projeto existente, o aluno tentou adaptar-se sempre que possível à estrutura e práticas existentes no projeto.

6.2.1 Front end

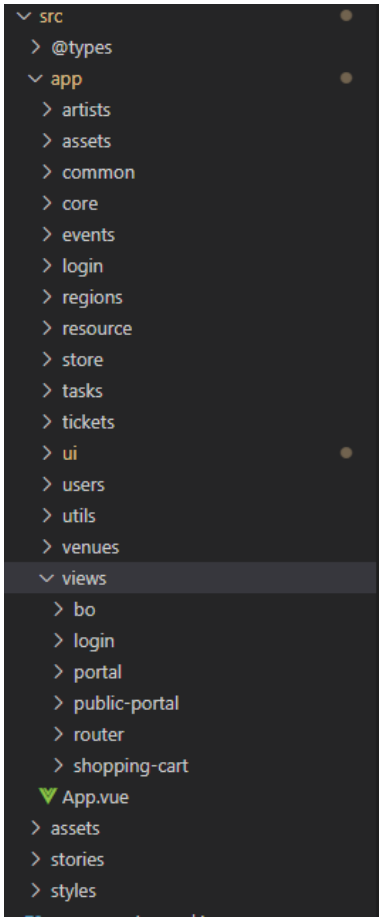


Figura 6.1: Estrutura do código do *front end*

O *front end* foi implementado em *Vue.js* na versão 2.6 com *Typescript* de modo a manter melhor controlo sobre as estruturas de dados do projeto. A aplicação divide-se sobretudo em três pastas principais:

- /src/assets
- /src/styles
- /src/app

Na pasta **assets** encontram-se todos os ficheiros estáticos da aplicação como imagens e videos, na pasta **styles** encontram-se todas as *stylesheets* globais ao projeto como variáveis de cores, tipos de letra, tamanhos e outro tipo de variáveis ligadas a *styling*. É de salientar que o projeto utiliza *SCSS* que é uma versão melhorada de *CSS* que é compilada para *CSS* puro e oferece grandes vantagens a nível de sintaxe bem como performance. Por fim, a pasta **app** armazena o código *Vue* da aplicação. Aqui também existe uma sub estrutura de código:

- app/common
- app/core
- app/ui
- app/views

A sub pasta **common** contém código reutilizável comum a todos os componentes, a pasta **core** tem as respostas e *requests* mapeados para os serviços do *back end*, a pasta **ui** contém todos os componentes *Vue* reutilizáveis na aplicação como botões e *cards* e por último, a sub pasta **views** contém as páginas da aplicação por onde o utilizador navega dentro da aplicação. Na Figura 6.1 consegue ver-se a estrutura em formato de árvore.

6.2.2 Back end

Para demonstrar a estrutura do *back end* vai ser partilhado a estrutura do microsserviço de *Ticket Management* uma vez que foi o microsserviço em que o aluno desenvolveu mais. O código está dividido em 4 grandes *packages* de *java*, como podemos ver na Figura 6.2:

- **api** - Onde estão os *resources* da API, isto é, as classes que definem os *endpoints* da API.

- **data** - Onde estão as entidades da API que fazem a ligação direta às tabelas da base de dados. Também contém classes *Data Transfer Objects* (DTO) que servem para comunicação de respostas e *requests* entre *front end* e *back end*.

- **external.services** - Onde estão interfaces para serviços externos ao microsserviço, nomeadamente, a REST API do PayPal e o microsserviço de Artistas da aplicação.

- **services** - Onde estão as classes intermédias de processamento da lógica do microsserviço e verificação de permissões do remetente do *request*. Estão entre as classes JAX-RS (endpoints) e os repositórios de dados da base de dados.

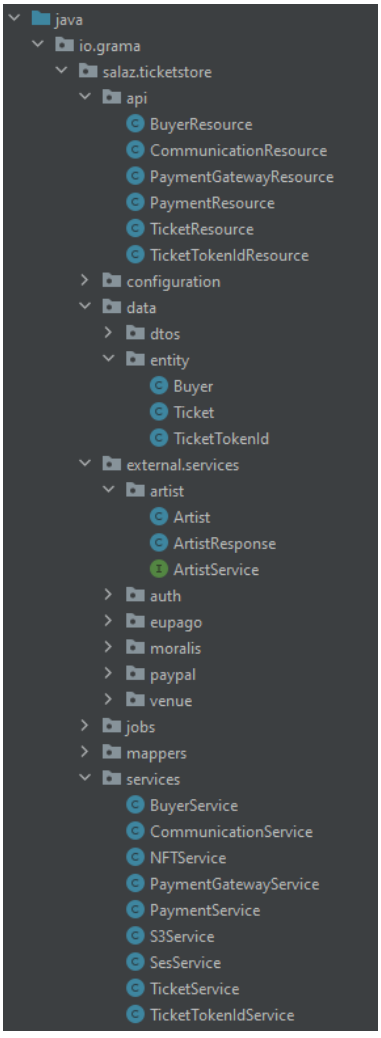


Figura 6.2: Estrutura do código do microsserviço *Ticket Management* (*back end*)

6.3 Funcionalidades

De acordo com os requisitos e a arquitetura, o aluno desenvolveu uma grande parte das funcionalidades previstas contudo, algumas mudanças ocorreram tal como previsto numa metodologia *agile*. De uma maneira geral, o aluno desenvolveu uma plataforma de reserva e pagamento de bilhetes para eventos, com a opção de receber uma representação do bilhete em formato *NFT*, recebendo também um email com as informações da reserva e os bilhetes. Na Tabela 6.1 podemos ver os requisitos que foram levantados no primeiro semestre e o seu estado atual de desenvolvimento.

Apesar do objetivo primordial do estágio ter sido alcançado, algumas das funcionalidades previstas não foram desenvolvidas. Isto deveu-se ao facto da estimativa do trabalho necessário ter sido bastante otimista, tendo em conta a experiência do aluno e da própria carga do trabalho. O aluno ficou responsável por desenvolver tanto o o *front end* numa *framework* que lhe era desconhecida, como o *back end* nas mesmas condições, acrescentando ainda a parte de *NFTs* e *blockchain* o que consumiu imenso tempo de aprendizagem.

Outra potencial causa para a má estimação do esforço do projeto foi a decisão do aluno implementar o microsserviço no paradigma **reativo** ou **assíncrono**. Este paradigma tornou o desenvolvimento significativamente mais complexo do que se tivesse considerado uma abordagem clássica imperativa. Porém, tendo em conta as vantagens de se implementar desta maneira, nomeadamente performance, o aluno juntamente com a empresa, decidiram que esta seria a melhor abordagem a adotar.

6.3.1 Portal Público

Na versão anterior da plataforma Sala-Z, não existia qualquer tipo de espaço público para utilizadores não autenticados sendo que um utilizador da plataforma era logo reencaminhado para a página de *login*. O aluno teve então de criar este espaço tendo em conta o propósito do mesmo: levar utilizadores não autenticados a escolher um evento e eventualmente comprar bilhete como podemos ver na Figura 6.3.

Também é possível filtrar os eventos por tipo de evento, por artista, por localidade e pesquisar por título do evento (ver Figura 6.4). Por último, esta página também contém um botão para redirecionar utilizadores autenticados ao *login* para aceder ao *backoffice*.

Tema	Requisito	Desenvolvido
Portal público	Ver lista de próximos eventos	Sim
	Ver lista de eventos destaque	Sim
	Procurar evento	Sim
	Filtrar eventos	Sim
	Ordenar eventos	Não
	Ver evento	Sim
Compra de bilhetes	Selecionar evento	Sim
	Reservar lugares	Sim
	Eliminar lugares reservados	Sim
	Ver preço da reserva	Sim
	Pagamento dos bilhetes por cartão de crédito	Não
	Pagamento dos bilhetes por MBWay	Não
	Pagamento dos bilhetes por PayPal	Sim
	Pagamento dos bilhetes por cripto moedas	Não
	Conectar crypto wallet	Não
	Emitir fatura e enviar por email	Sim
	Autorização de recolha de dados	Sim
	Acesso aos termos e condições	Sim
	Ver resumo da reserva	Sim
	Editar informação da reserva	Sim
	Ver resumo da compra finalizada	Sim
Gestão dos bilhetes	Ver bilhetes comprados tradicionalmente no email	Sim
	Ver bilhetes comprados por *NFT* na crypto wallet	Não
	Emitir bilhetes *NFT*	Sim
	Filtrar bilhetes *NFT*	Não
	Ver estado dos bilhetes emitidos (Vendidos, por vender, ou revendidos)	Não
	Ver mapa dos bilhetes vendidos	Não
Gestão dos eventos	Adicionar eventos	Sim
	Editar eventos	Sim
	Ver últimas atuações no espaço	Sim
	Configurar tipos de bilhetes	Não
	Configurar preço dos bilhetes	Não
	Adicionar informação de custos e *revenue*	Não
	Publicar um evento no portal público	Sim
	Adicionar colaboradores	Sim
	Remover colaboradores	Sim
	Ver lista de próximas tarefas	Não
Estatísticas	Ver estatísticas de venda dos bilhetes	Não
	Ver estatísticas das métricas dos eventos	Não
	Ver atividade recente dos eventos	Não
	Ver atividade recente de um evento	Não
	Ver estatísticas relacionadas com os artistas	Não

Tabela 6.1: Estado atual dos requisitos levantados para o sistema de bilhética do Sala-Z

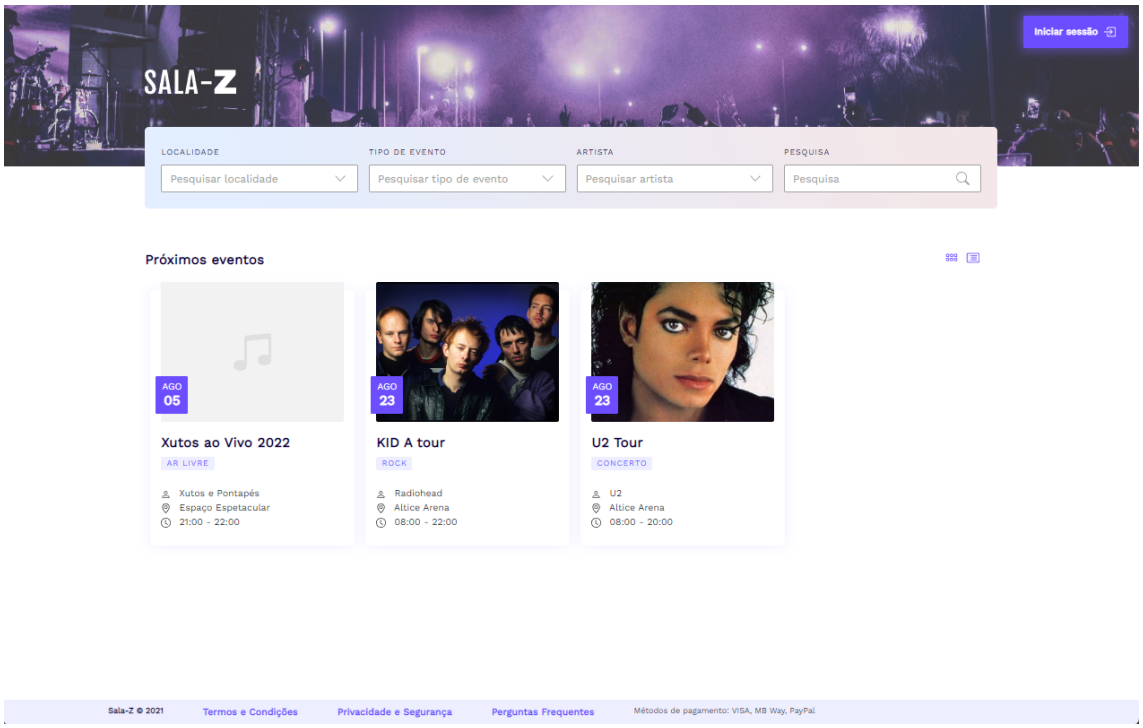


Figura 6.3: Portal público da plataforma Sala-Z

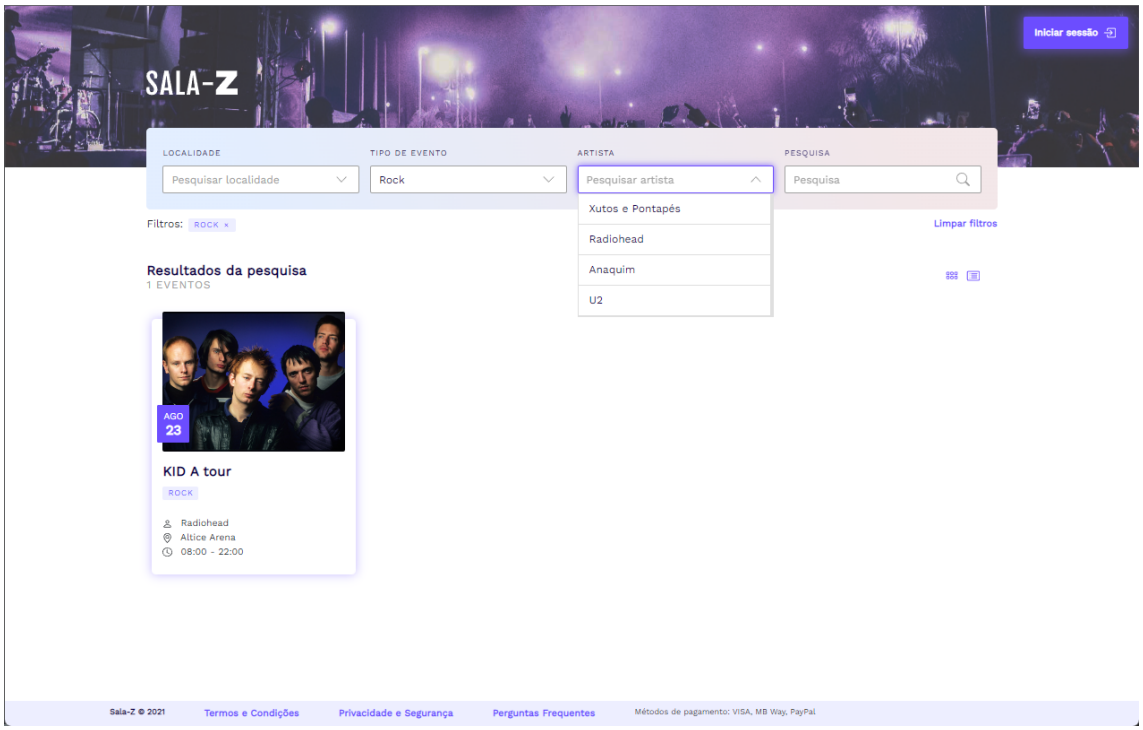


Figura 6.4: Portal público da plataforma Sala-Z (filtros)

Detalhes de um evento

Ao clicar num evento, o utilizador é levado para a página de detalhes de um evento (ver Figura 6.5). Aqui tem informação mais detalhada sobre o evento, mostrando detalhes sobre o local, capacidade do evento, restrição de idades e

informação sobre o promotor. Também também um menu que leva o utilizador a comprar bilhetes para esse evento dando já informação sobre os preços.

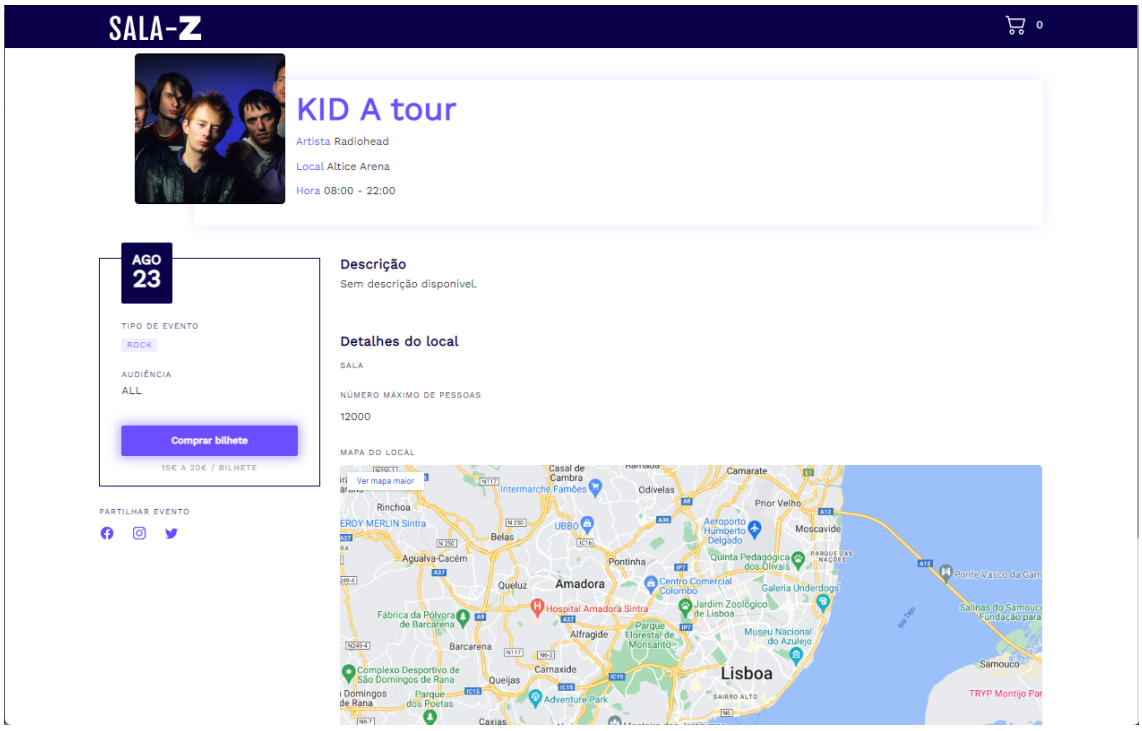


Figura 6.5: Página de detalhes de um evento

6.3.2 Compra de bilhetes

Assim que um utilizador clica no botão de comprar bilhetes, este é redirecionado para a página do carrinho de compras (ver Figura 6.6). Aqui foi implementada uma única página com componentes em fragmentos de modo a melhorar a performance não tendo assim necessidade de redirecionamentos no *front end*. No primeiro fragmento o utilizador pode escolher os bilhetes que quer através do formulário e assim que os dados estejam corretos e não ultrapassar o limite máximo de bilhetes por compra ou o número de bilhetes disponíveis para o evento, o botão de avançar fica disponível e pode avançar para o próximo fragmento.

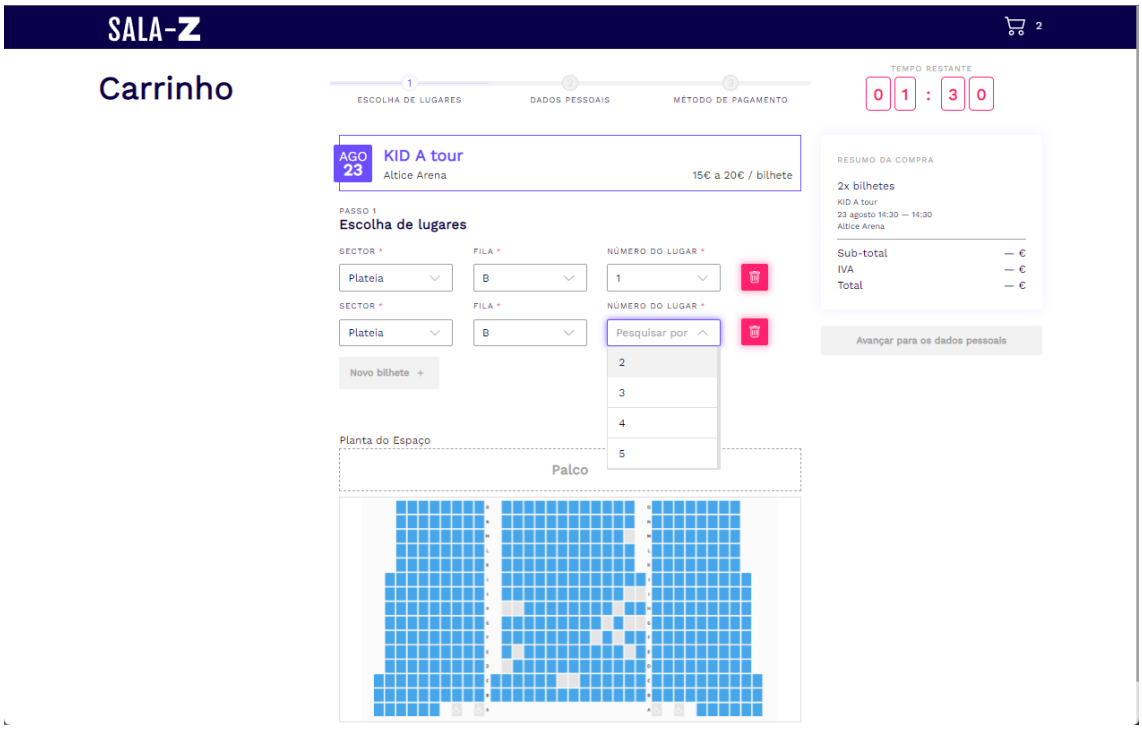


Figura 6.6: Página de detalhes de um evento

No segundo fragmento, é pedido ao utilizador que insira os seus dados pessoais de modo a prosseguir com a compra (ver Figura 6.7 e 6.8). No formulário consta toda a informação neccessária levantada nos requisitos. Este formulário está devidamente validado utilizando o módulo de *Vue* chamado *Vuelidate* que facilita todo o trabalho de validação como a definição de campos obrigatórios e validações condicionais. Aqui também é pedido o endereço da carteira de *crypto* do utilizador caso queira receber o bilhete como *NFT*. Este tem que ser um endereço válido que esteja apto para receber *NFTs* na rede *Polygon* ou *Ethereum*. Por fim, assim que as validações estejam todas corretas, o utilizador pode avançar para o fragmento de pagamento.

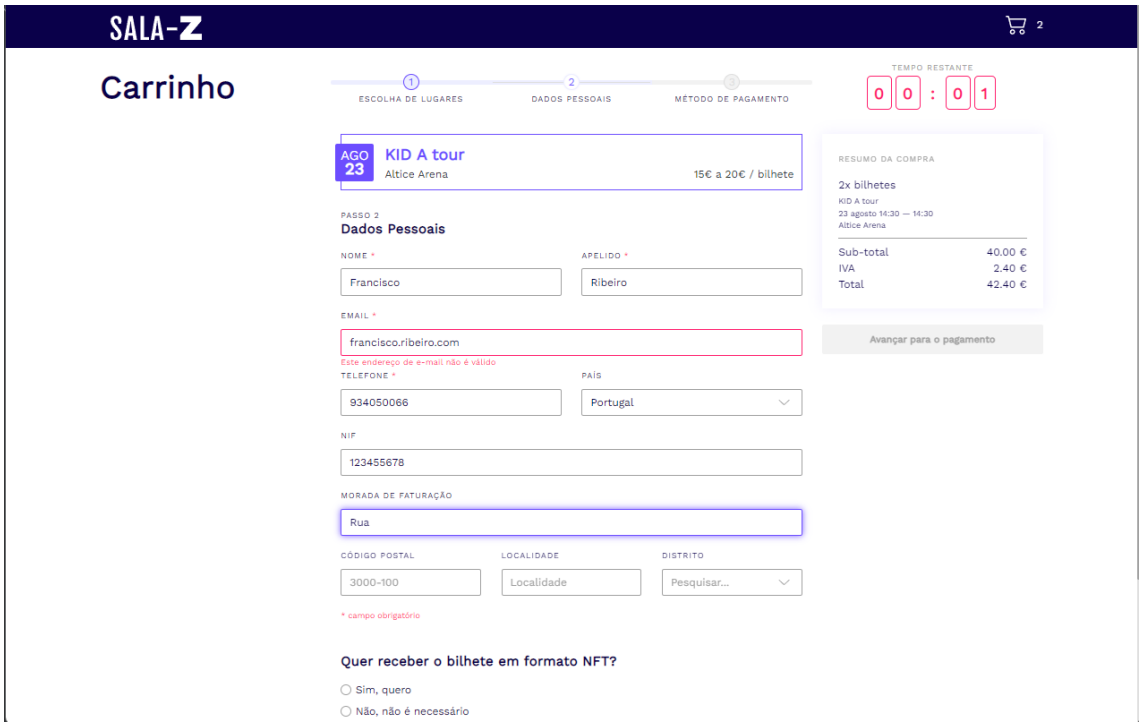


Figura 6.7: Formulário de informação pessoal no processo de reserva de bilhetes

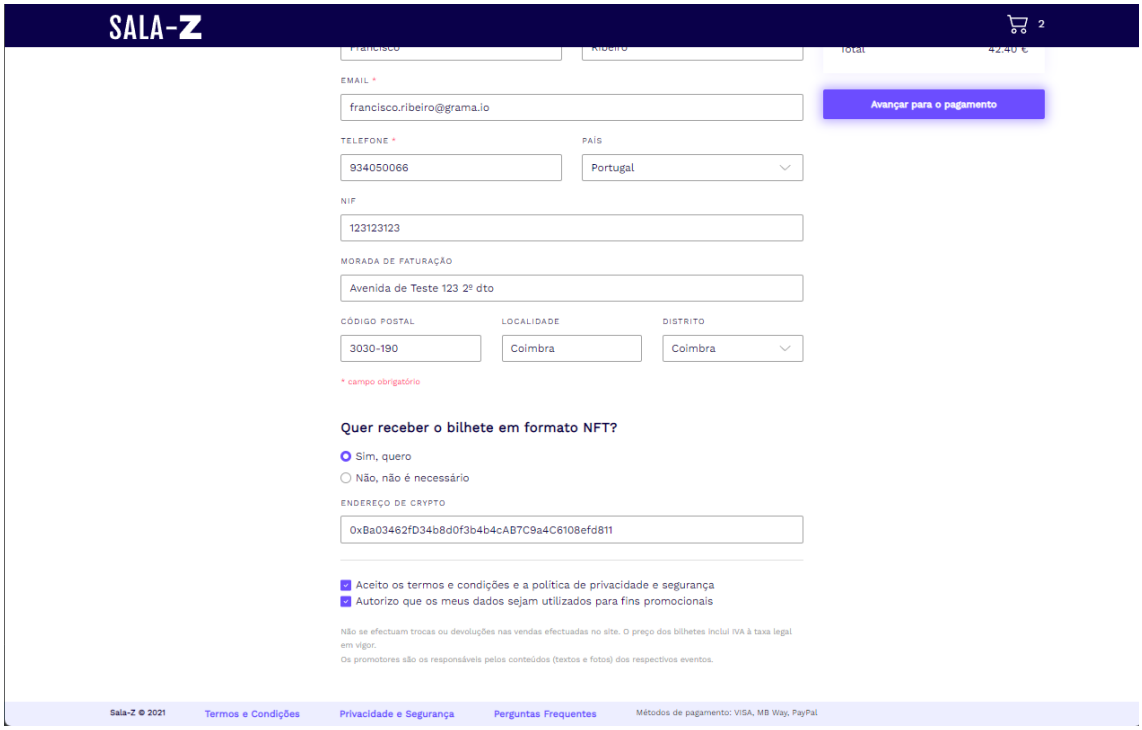


Figura 6.8: Formulário de informação pessoal no processo de reserva de bilhetes (continuação)

Neste último fragmento, é apresentado ao utilizador vários métodos de pagamento, porém, no estado atual da plataforma, apenas o método *PayPal* se encontra completamente implementado e testado. Isto deveu-se ao facto da comunicação entre a empresa e o prestador do serviço de *gateway* de pagamento *EuPago*

não ter sido estabelecida a tempo. A aplicação está pronta para este tipo de pagamentos (Referências multibanco, MBWay e Cartão de crédito) no entanto, o aluno não conseguiu testar estas funcionalidades por falta de acesso ao serviço completo. Na Figura 6.9 e 6.10 podemos ver as opções de escolha sendo que as últimas duas (Cartão de Crédito e *PayPal*) levam o utilizador para uma página externa de modo a inserir os dados de pagamento num formulário seguro. As primeiras duas opções não requerem ações fora da aplicação logo ainda passam por uma página de confirmação (ver Figura 6.11 e 6.12).

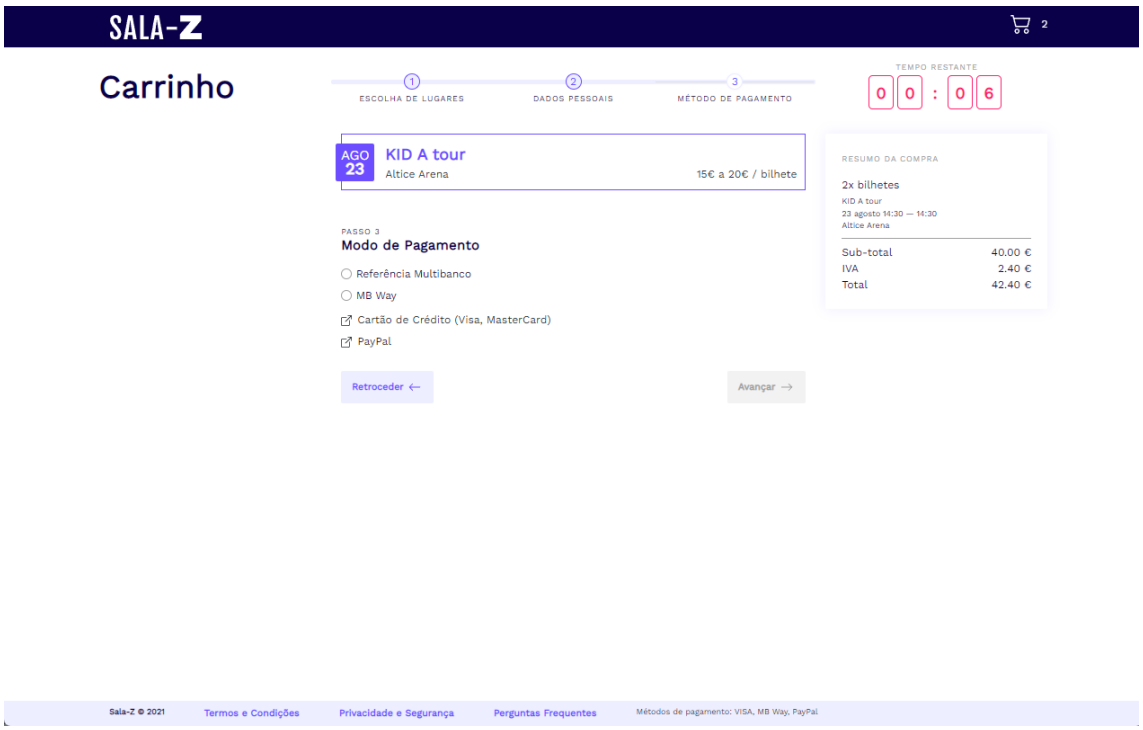


Figura 6.9: Métodos de pagamento da plataforma Sala-Z

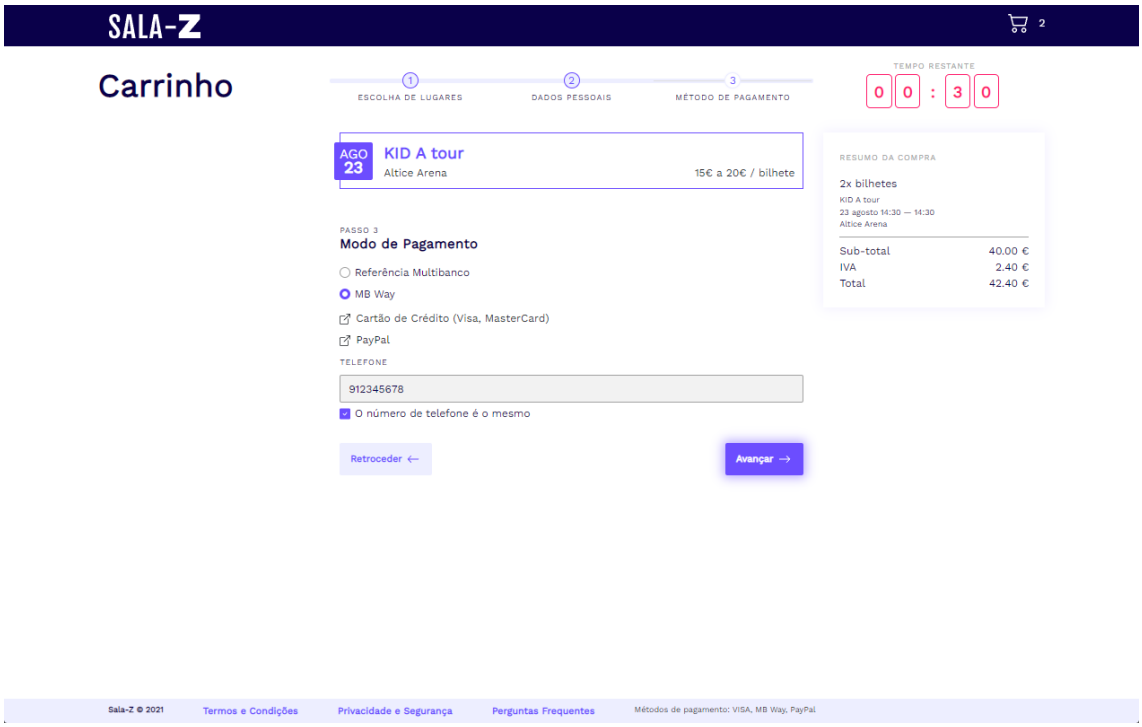


Figura 6.10: Métodos de pagamento da plataforma Sala-Z (MBWay)

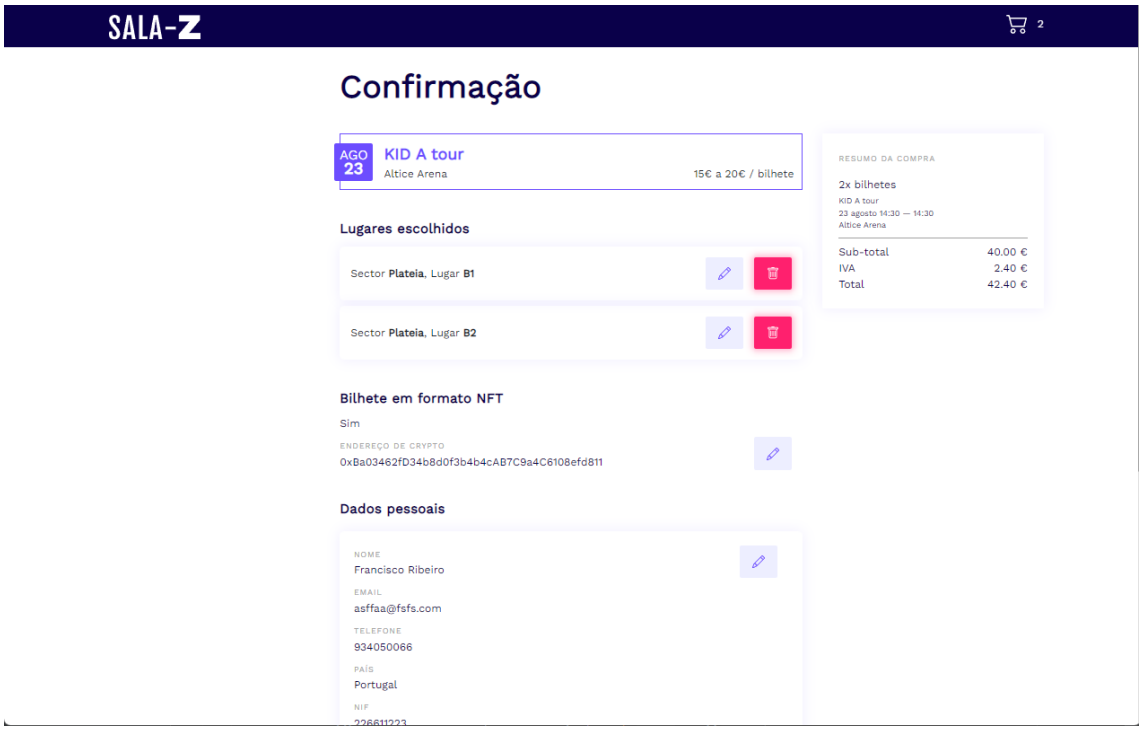


Figura 6.11: Ecrã de confirmação dos dados da reserva

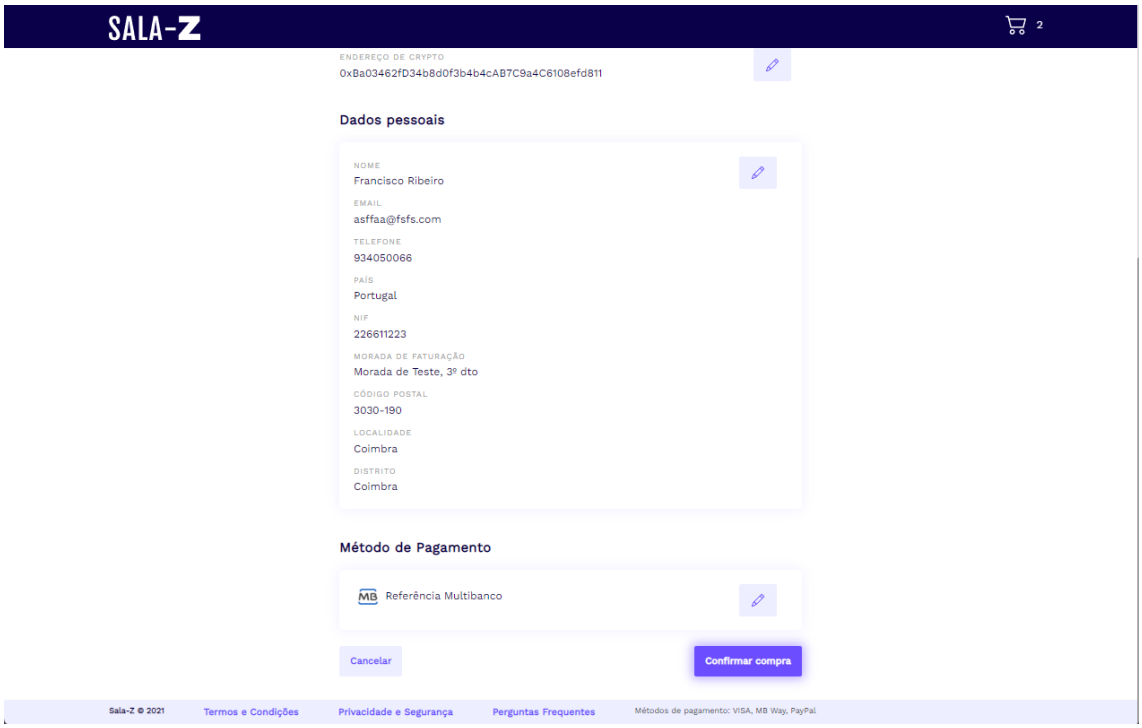


Figura 6.12: Ecrã de confirmação dos dados da reserva (continuação)

Na Figura 6.13 também podemos ver o ecrã de geração de referências multibanco para o pagamento utilizando o serviço *EuPago* no entanto, como previamente mencionado, estas referências são referências de teste do ambiente de testes do serviço.

Figura 6.13: Dados de referência Multibanco para o pagamento de uma reserva

Para o método de pagamento implementado e testado (*PayPal*) o utilizador é levado para um formulário externo onde lhe é pedido que faça o *login* com a sua conta *PayPal* e autorize o pagamento da reserva (ver Figura 6.14). Ao aceitar, o serviço foi configurado para informar o microsserviço *Ticket Management* de que o utilizador confirmou o pagamento através de uma função **callback**. É enviado ao microsserviço o *id* do utilizador e após este sinal, é marcado na base de dados que a reserva foi paga e os bilhetes dessa reserva já não estão disponíveis.

Por último é enviado um email com os dados da reserva ao utilizador através do serviço SES da AWS. Na arquitetura inicial, o que estava previsto era ter sido criado outro microsserviço que estaria encarregue de enviar os emails aos utilizadores após receber uma mensagem do microsserviço *Ticket Management*. Embora esta comunicação tenha sido construída através de uma *queue* SNS e SQS, este microsserviço não chegou a ser implementado devido a falta de tempo, sendo que esta funcionalidade foi implementada diretamente no microsserviço *Ticket Management*.

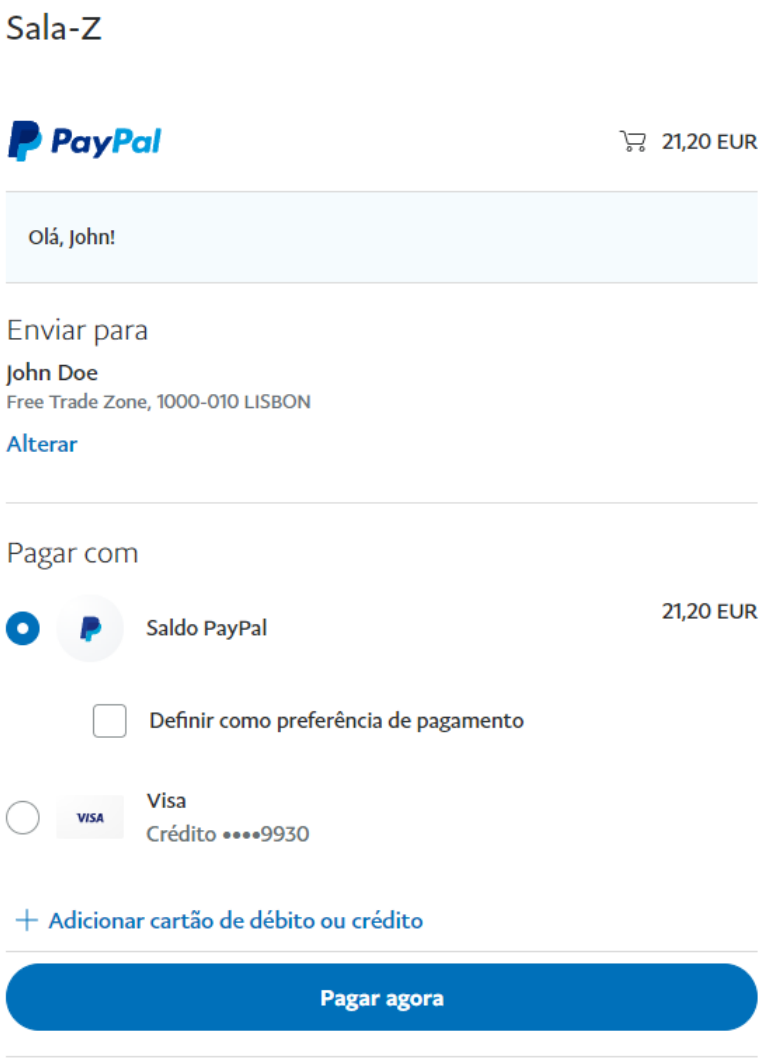


Figura 6.14: *Form* externo de pagamento do serviço *PayPal*

Enquanto o utilizador insere os seus dados na página externa, o *front end* encontra-se numa página de espera onde é feito um *pooling* de 5 a 5 segundos a verificar se o registo da reserva na base de dados já se encontra com a *flag* que indica que esta reserva foi paga (ver Figura 6.15). Assim que o *callback* tenha sido bem sucedido, durante este processo esta *flag* vai mudar de valor desbloqueando assim a página final da reserva no *front end*.

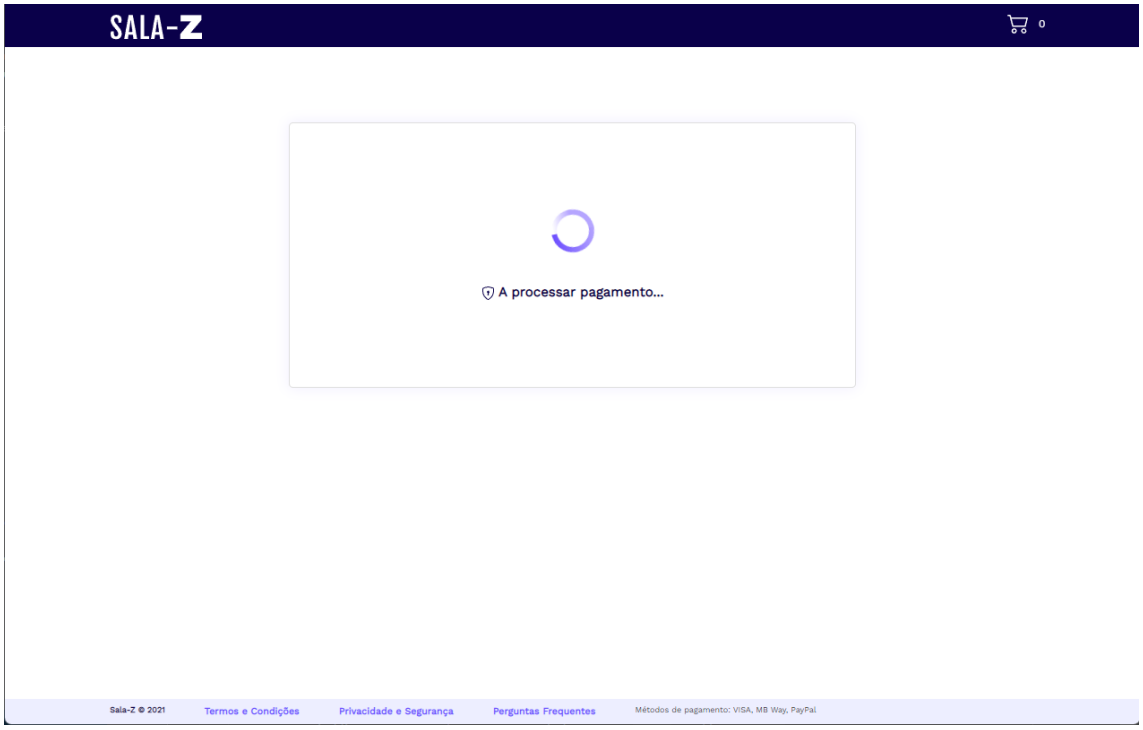


Figura 6.15: Página de espera pelo pagamento de uma reserva

Na Figura 6.16 podemos ver a página de resumo da reserva bem sucedida. São mostrados os bilhetes comprados incluindo a opção de transferir os bilhetes. Embora possa transferir os bilhetes pela plataforma, o utilizador recebe o email da reserva tal como foi mencionado anteriormente. Nas Figuras 6.17 e 6.18 pode-se ver o email que um utilizar deve receber após a conclusão bem sucedida do pagamento de um reserva na plataforma. Este email foi contruído através de um *template* predefinido no *back end* utilizando o motor de *templates Qute* [15]. Este motor permite construir *templates* para qualquer tipo de documentos e permite a substituição de valores por objetos *Java*, *conditional rendering* e está bem integrado no ambiente *Quarkus* o que permitiu uma integração fácil e sem problemas.

Figura 6.16: Página final de resumo da reserva

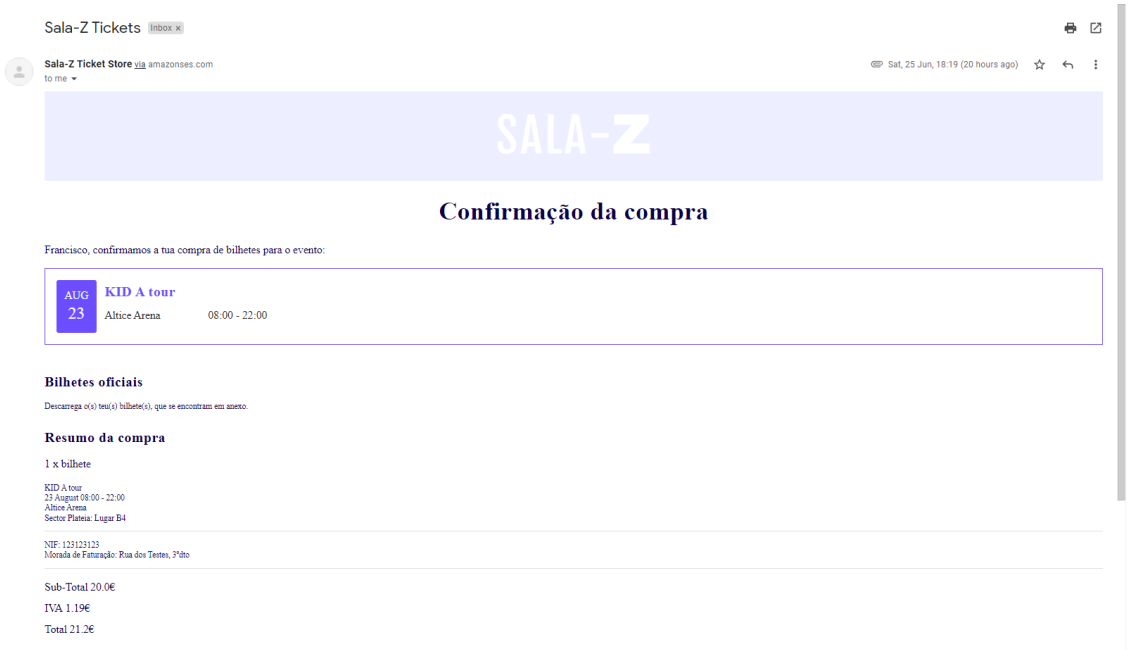


Figura 6.17: Email recebido após a conclusão bem sucedida do pagamento de uma reserva

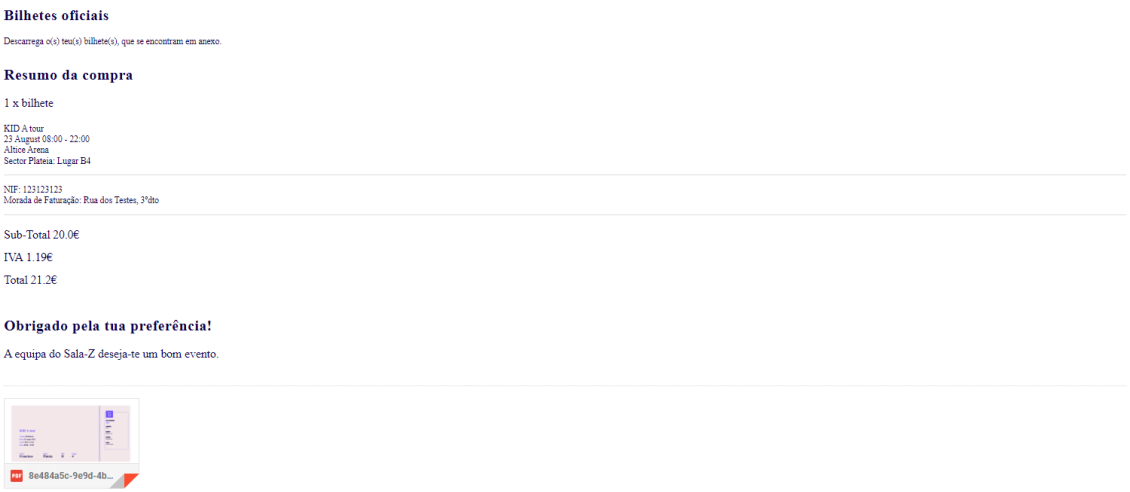


Figura 6.18: Email recebido após a conclusão bem sucedida do pagamento de uma reserva (continuação)

6.3.3 Bilhetes *NFT*

Neste sistema de pagamento de bilhetes para eventos foi implementada a opção de receber um *NFT* como bilhete para o evento. Infelizmente, devido à falta de tempo, o aluno não conseguiu implementar uma solução na plataforma para a visualização destes *NFTs* mas, no entanto, estes *NFTs* estão a ser criados e enviados para o endereço do utilizador caso seja esta a sua opção.

Foi desenvolvido o *smart contract* referente à construção dos *NFTs* e houve uma grande diferença para a implementação prevista. Foi utilizada a norma ERC 1155 (ver Secção 3.6.3) para a criação dos *NFTs* em vez da norma estudada previamente. Esta mudança foi realmente necessária devido ao facto de poderem haver eventos com bilhetes sem lugares marcados. Cada pessoa que for ao evento tem direito a um *NFT* desse evento caso tenha pago o bilhete, no entanto, esse tipo de *NFT* vai ser igual para os outros compradores, coisa que seria impossível de realizar utilizando a norma antiga. O *smart contract* é relativamente simples uma vez que se utilizou *libraries* de *smart contracts* que implementam estas normas de um repositório bastante conhecido e utilizado mundialmente chamado *OpenZeppelin*. *OpenZeppelin* fornece produtos para construir, automatizar e operar aplicações descentralizadas [13] e um destes produtos é a implementação segura e auditada da norma ERC 1155 em *Solidity* para ser utilizada em *smart contracts*. Estas implementações são gratuitas e estão disponíveis no repositório de *Github* da *OpenZeppelin*.

Na Figura 6.19 podemos ver o código do *smart contract* implementado. Utiliza duas normas implementadas pelo *OpenZeppelin*: ERC 1155 e *Ownable*. Foram criadas várias funções que vão ser descritas:

- **mint** - É apenas um *wrapper* da função original da norma ERC 1155 que leva como parâmetro o endereço da carteira para onde se quer criar o *NFT*, o id do *NFT* dentro do contrato, e a quantidade de *NFTs* desse tipo que se quer

criar.

- **mintWithUri** - É uma função de *mint* igual à anterior com a diferença de que leva também como parâmetro o URI da metadata desta nova classe de *NFTs*. Ao utilizar esta função, todos os *NFTs* criados e por criar com o id passado também como parâmetro, terão a mesma *metadata* que está alojada no URI fornecido.
- **setUri** - É uma função *wrapper* da função original *setUri* da norma ERC 1155 que permite alterar o URI dos *NFTs*.
- **mintBatch** - É uma função *wrapper* da função original *mintBatch* da norma ERC 1155 que permite criar *NFTs* de diferentes ids na mesma chamada de função, otimizando assim as chamadas ao contrato.
- **burn** - É uma função *wrapper* da função original *burn* da norma ERC 1155 que permite **apenas** ao detentor do *NFT* destruir a quantidade desejada de um certo tipo de *NFT* identificado pelo id passado como parâmetro.

```
1  //SPDX-License-Identifier: UNLICENSED
2  pragma solidity ^0.8.0;
3
4  // Import 1155 ERC from OpenZeppelin
5  import "@openzeppelin/contracts/token/ERC1155/ERC1155.sol";
6  import "@openzeppelin/contracts/access/Ownable.sol";
7
8  contract NFTContract is ERC1155, Ownable {
9      constructor() ERC1155("") { }
10
11     function setURI(string memory newuri) private onlyOwner {
12         _setURI(newuri);
13     }
14
15     function mintWithUri(address account, uint256 id, uint256 amount, string memory uri)
16         public
17         onlyOwner
18     {
19         setURI(uri);
20         _mint(account, id, amount, "");
21     }
22
23     function mint(address account, uint256 id, uint256 amount)
24         public
25         onlyOwner
26     {
27         _mint(account, id, amount, "");
28     }
29
30     //
31     function mintBatch(address to, uint256[] memory ids, uint256[] memory amounts)
32         public
33         onlyOwner
34     {
35         _mintBatch(to, ids, amounts, "");
36     }
37
38     // Only NFT owner can burn their tokens. Amount checking already processed by internal _burn function
39     function burn(address account, uint256 id, uint256 amount) public {
40         require(msg.sender == account);
41         _burn(account, id, amount);
42     }
43 }
```

Figura 6.19: *Smart Contract* dos *NFTs* da plataforma Sala-Z

Certas funções estão apenas disponíveis para o endereço que criou o contrato. *mintWithUri*, *mint* e *mintBatch* são funções que têm um ***modifier*** (palavra após a definição da assinatura da função) que permite verificar certas condições antes de correr a própria função. Neste caso, o *modifier* é o ***onlyOwner*** proveniente da *library Ownable* da implementação *OpenZeppelin* que não deixa executar a função caso o endereço que tenha chamado a função não seja o endereço que criou o contrato.

Deployment

Com o contrato finalizado, este teve que ser *deployed* na rede *blockchain* de modo a estar acessível para os utilizadores. Para tal, vários passos tiveram de ser realizados. O primeiro passo passou então por criar um endereço oficial da empresa de modo a ficar encarregue de manter os fundos necessários e ser o criador do contrato e, para isso, foi criado através do serviço de criação e gestão de *wallets Ethereum* chamada *Metamask* [11]. De seguida, com um endereço público e privado gerado, foi necessário configurar a rede de testes *Polygon Mumbai*, que permite realizar testes aos contratos antes de os criar na rede real, e inserir fundos falsos para essa rede de testes de modo a conseguir "pagar"as taxas de *deployement* do contrato.

Ao concluir o *deployment*, o contrato está acessível através de um endereço esepcífico ao contrato. O *deplyoment* foi feito através da ferramenta ***Remix*** que é um IDE específico para construção e *deployment* de *smart contracts* na rede *Ethereum*.

Integração no *back end*

Por último, foi necessário integrar no *back end* um meio de comunicação entre este, escrito em *Java*, e o próprio *smart contract*. Para tal, foi utilizado a *library* ***Web3J***. *Web3j* é uma *library* modular, reativa e *type safe* de *Java* e *Android* para trabalhar com *smart contracts* e integrar comunicação com *nodes* na rede *Ethereum* [18]. Esta tecnologia também traz um *Command Line Interface* (CLI) que permite gerar automaticamente classes em *Java* que servem como interfaces no código para comunicar com o *smart contract* compilado. Deste modo, é possível por exemplo chamar a função *mintWithUri* diretamente no código *Java* e receber a resposta, garantindo claro que a chamada esteja assinada criptográficamente. Na Figura 6.20 conseguimos ver um excerto de código que utiliza esta classe gerada e cria um novo bilhete *NFT* assincronamente.

```
tokenIdService.findTicketTokenById(tokenId).map(ticketTokenId -> {
        Web3j web3j = Web3j.build(new HttpService(rpcNode));
        Credentials credentials = Credentials.create(privateKey);
        TransactionManager transactionManager = new RawTransactionManager(web3j, credentials, chainId);

        NFTContract contract = NFTContract.load(contractAddress, web3j, transactionManager, new DefaultGasProvider());

        return new IpfsLinkAndContractReturnType(ticketTokenId.ipfsLink, contract);
    })
    .chain(returnType -> Uni.createFrom().completionStage(returnType.contract.mintWithUri(destAddress, BigInteger.valueOf(tokenId), BigInteger.ONE, returnType.ipfsLink).sendAsync()))
    .subscribe().with(item -> {
        if (item.isStatusOK()) {
            log.info("Smart contract transaction went ok!");
        } else {
            log.error("Smart contract transaction error!");
        }
    });
```

Figura 6.20: Excerto de código do serviço *NFTService* responsável por criar um novo *NFT* de um bilhete

Uma breve explicação do código começa por iniciar o objeto Web3j que se liga

ao *node* da rede *Ethereum* e iniciar um objeto *Credentials* que representa a maneira como vão ser assinadas as chamadas das funções. Neste caso é guardada e utilizada a chave privada da conta criada anteriormente. De seguida é carregada a classe gerada a partir do *smart contract* criado através da função *load* e a partir daí pode-se utilizar este objeto para chamar as funções desse *smart contract*.

IPFS e Moralis

Todos os *NFTs* criados na plataforma tiveram a sua *metadata* criada na rede IPFS de modo a garantir imutabilidade da mesma após a criação (ver Secção 3.7.1). Para tal, utilizou-se o serviço *Moralis* porque fornecia uma maneira fácil e eficaz de armazenar conteúdo nesta rede. Estava previsto usar *Moralis* para interagir com o *smart contract* e obter informação sobre os *NFTs* criados, porém, por falta de tempo como já foi mencionado, esta parte do estágio ficará para trabalho futuro. Também não se utilizou *Moralis* no *back end* uma vez que este serviço está especificamente feito para suportar a construção quase exclusiva de *Dapps* (Decentralized Apps), o que implica serem aplicações com ausência quase total de um *back end*, logo, havia pouca informação sobre a integração deste serviço em ambiente *Java*. Na Figura 6.21 podemos ver a única utilização no projeto do serviço *Moralis* para o armazenamento da *metadata* dos *NFTs*.

```java
@Transactional(SUPPORTS)
public String ipfsUpload(@Valid Ticket ticket) {
    // 1: Create json metadata
    // 2: Encode64 it
    // 3: Create name
    // 4: Create MoralisFileRequest object
    // 5: Save link to Ticket object and return link

    EventResponse eventResponse = venueService.get().getEvent(ticket.eventId);

    log.debug("Building NFT Metadata for ticket " + ticket.getId());
    NFTMetadata nftMetadata = new NFTMetadata();
    nftMetadata.setRow(ticket.getRow() != null ? ticket.getRow() : "");
    nftMetadata.setSeatNumber(ticket.getSeatNumber() != null ? ticket.getSeatNumber() : "");
    nftMetadata.setSector(ticket.getSector() != null ? ticket.getSector() : "");
    nftMetadata.setName(eventResponse.getPayload().getName() + " - " + nftMetadata.getRow() + nftMetadata.getSeatNumber());
    nftMetadata.setImage(ticket.getS3PdfLocation());
    nftMetadata.setDescription("Seat number: " + nftMetadata.getSeatNumber() + " Row: " + nftMetadata.getRow() + " Sector: " + nftMetadata.getSector());
    log.debug("NFT Metadata built -> " + nftMetadata);

    String encodedJson = null;
    try {
        String jsonStr = mapper.writeValueAsString(nftMetadata);
        // Moralis requires the json metadata encoded in Base64 probably to build the unmatched URI generated according to content
        encodedJson = Base64.getEncoder().encodeToString(jsonStr.getBytes());
        log.debug("Json metadata encoded with Base64: " + encodedJson);

    } catch (JsonProcessingException e) {
        log.error("Error parsing nftMetadata to Json on ipfsUpload(). -> " + e.getMessage());
        throw new ServerErrorException("Error uploading NFT Metadata to Moralis", Response.Status.INTERNAL_SERVER_ERROR);
    }

    String fileName = "tickets/" + UtilFunctions.padLeftZeros(ticket.getTicketTokenId().toString(), length: 64) + ".json";
    log.debug("IPFS resource location set to: " + fileName);


    MoralisFileRequest fileRequest = new MoralisFileRequest(fileName, encodedJson);

    List<MoralisFileResponse> response = moralisService.get().uploadJsonMetadata(List.of(fileRequest), restApiKey);
    log.debug("NFT Metadata resource location available at: " + response.get(0).path);
    return response.get(0).path;
}
```

Figura 6.21: Excerto de código da função de upload de *metadata* de um *NFT*

Para concluir esta secção, na Figura 6.22 podemos ver o *NFT* de um bilhete na plataforma *OpenSea* onde é possível ver todos os *NFTs* de um certo *address* nas redes reais *blockchain* ou, neste caso, na rede de testes.

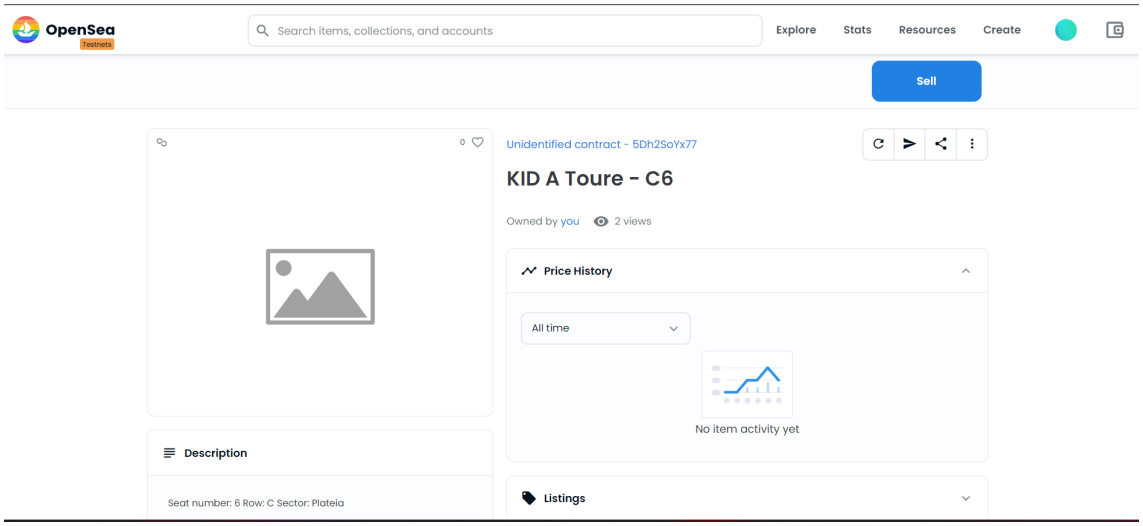


Figura 6.22: *NFT* de um bilhete visualizado na plataforma de visualização de *NFTs OpenSea*

6.4 API

Na secção anterior foi falado maioritariamente das funcionalidades implementadas vistas do lado do utilizador no *front end*. Nesta secção vai ser mostrado o mapa da API construída no *back end* mais propriamente do microsserviço *Ticket Management*. O microsserviço está dividido em seis categorias de *endpoints* cada uma com o seu propósito.

Foi implementado no microsserviço uma ferramenta para ajudar na visualização dos *endpoints* chamada ***SwaggerUI***. Esta ferramenta fornece um espaço onde se podem ver todos os *endpoints* e seus caminhos do microsserviço, os *schemas* das respostas e *requests* de cada *endpoint*, capacidade de testar cada *endpoint* nesse espaço e apresenta documentação específica a cada *endpoint*. É uma ferramenta bastante útil quando se quer ter um mapa organizado de todas as funcionalidades do *back end*. Este espaço encontra-se disponível no URL *"q/swagger-ui/"* do servidor que está a hospedar o microsserviço.

6.4.1 Buyer

Esta categoria contém os *endpoints Create Read Update and Delete* (CRUD) relativos à gestão de um *buyer*. Na Figura 6.23 podemos ver os diferentes *endpoints* juntamente com o método do request HTTP.

Figura 6.23: *Endpoints* da categoria *Buyer*

O aluno baseou-se na convenção REST [16] para a nomenclatura e utilização dos métodos HTTP durante a contrução dos *endpoints*.

6.4.2 Communication

Esta categoria oferece *endpoints* para a comunicação entre microsserviços (ver Figura 6.24). O *endpoint* GET vai buscar uma lista atualizada da mensagens na fila SQS específica para o microsserviço enquanto que o POST publica uma mensagem para o SNS de modo a que outros microsserviços recebam atualizações. O *endpoint* POST do email é responsável por enviar o email com os bilhetes de uma reserva para o comprador.

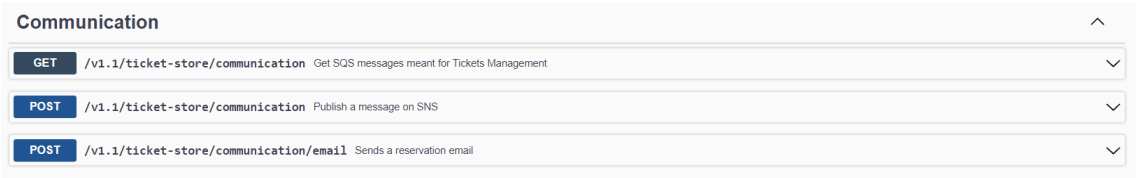


Figura 6.24: *Endpoints* da categoria *Communication*

6.4.3 Payment

Esta categoria oferece *endpoints* para o pagamento de reservas de bilhetes de acordo com o seu método de pagamento (ver Figura 6.25). O *endpoint capture* é o *endpoint* que está definido como *callback* na gateway de pagamentos *PayPal*. Assim que um utilizador complete o pagamento no *form* externo da aplicação, este endpoint é chamado pelo sistema *PayPal* de modo a oficializar a compra.

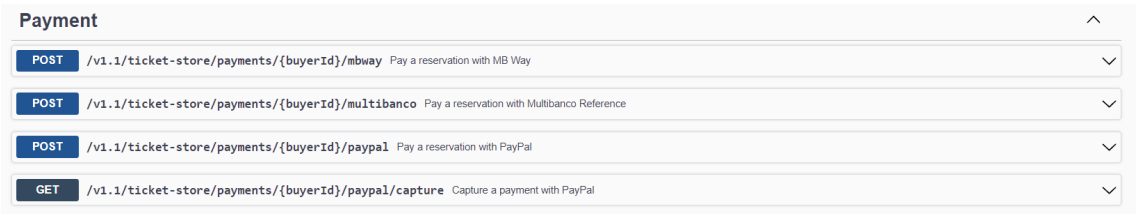


Figura 6.25: *Endpoints* da categoria *Payment*

6.4.4 PaymentGateway

Esta categoria oferece *endpoints* maioritariamente de teste para a comunicação direta entre o microsserviço e os sistemas externos de pagamento (ver Figura 6.26). Têm os mesmos *endpoints* que os anteriores, com a adição de um *endpoint* para pagamentos com cartão de crédito, mas não são utilizados pelo *front end*. Pode-se reparar na diferença de que nos *endpoints* anteriores, o URL leva como parâmetro um *id* de um *buyer*, logo, estes *endpoints* preparam primeiro a informação do *front end* antes de comunicar com a *gateway* de pagamento.

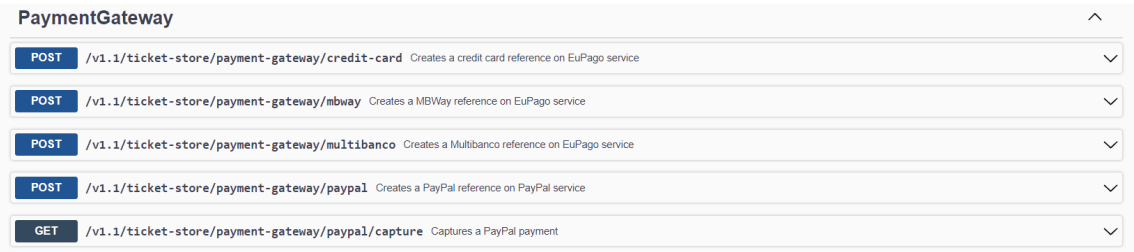


Figura 6.26: *Endpoints* da categoria *PaymentGateway*

6.4.5 Ticket

Esta categoria oferece *endpoints* para operações CRUD da entidade que representa os bilhetes na aplicação (ver Figura 6.27). Tem alguns *endpoints* que implementam operações mais específicas com o intuito de reduzir o número de chamadas feitas à API ou reduzir a computação feita pelo *front end*, nomeadamente, *"tickets/event/eventId/available"* que retorna apenas bilhetes de um evento que não estejam reservados.

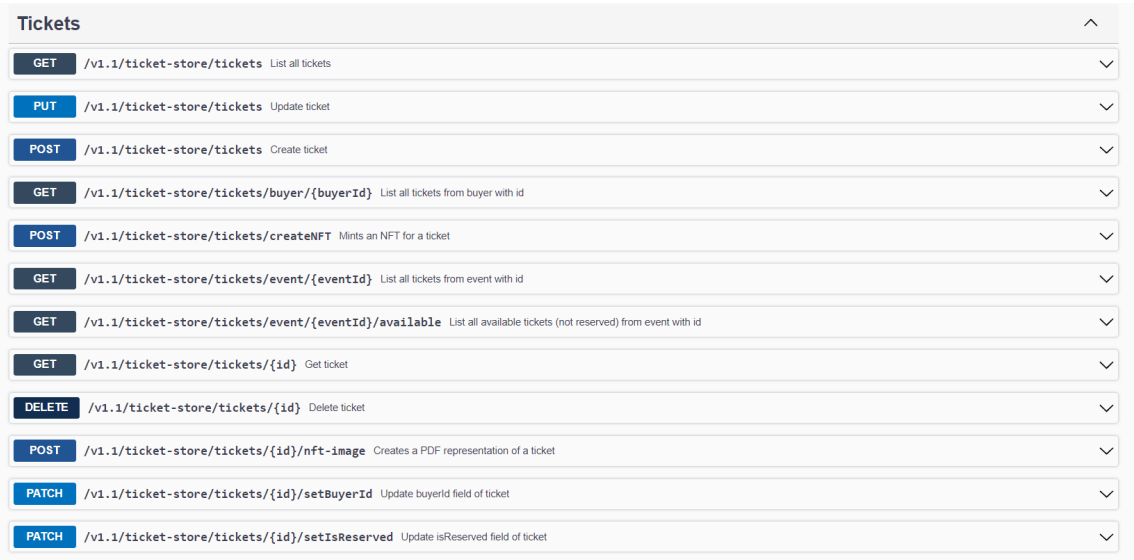


Figura 6.27: *Endpoints* da categoria *Ticket*

6.4.6 Token

Por fim, temos a categoria que oferece apenas um *endpoint* de teste para a criação manual de registos da entidade *TicketTokenId* (ver Figura 6.28). Esta entidade fornecia a ligação entre os bilhetes e o *id* no *smart contract* referente aos *NFTs*. Este *endpoint* também não é utilizado pelo *front end* e serviu apenas para razões de testagem durante o desenvolvimento.

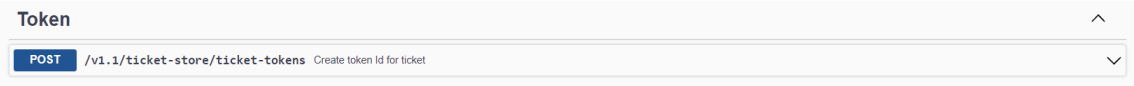


Figura 6.28: *Endpoints* da categoria *Token*

6.5 Arquitetura Real

Na Secção 5.1 foi descrita a arquitetura de software planeada para o projeto na fase inicial do mesmo, porém, como em todos os projetos, esta tem que se adaptar de modo a acomodar eventuais mudanças e imprevistos no decorrer do projeto. Como tal, é apresentada de seguida as mudanças e suas justificações na arquitetura de software da plataforma Sala-Z.

No nível de contexto, houve ligeiras mudanças para a integração do sistema externo *PayPal* e na remoção dos tipos de utilizador ficando apenas com um único utilizador. Podemos ver o diagrama atualizado na Figura 6.29.

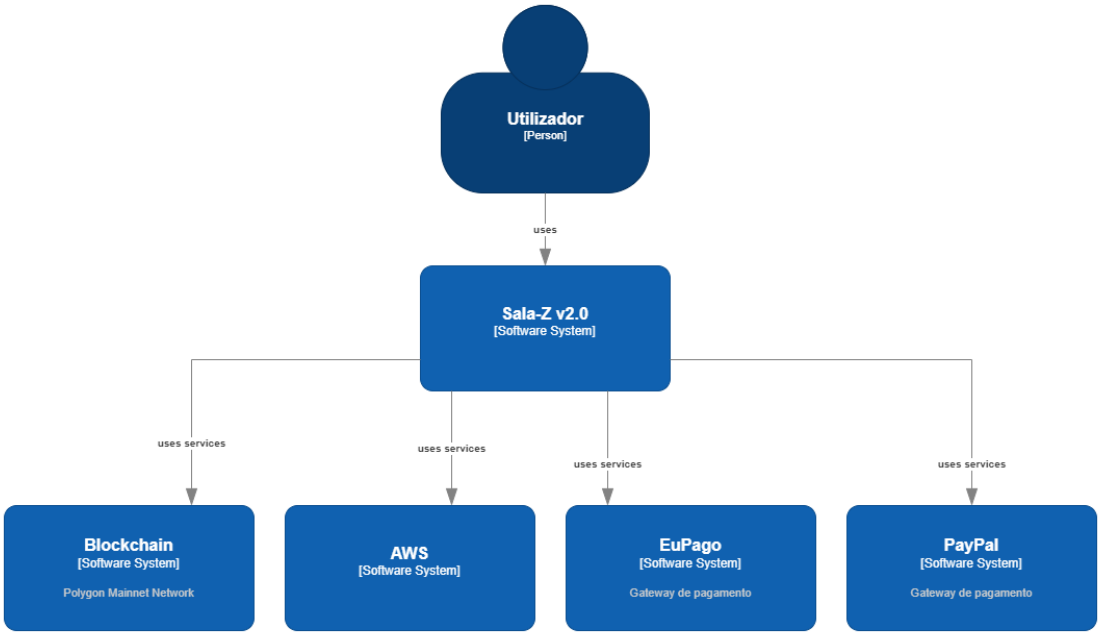


Figura 6.29: Nível de contexto do modelo C4 final

No nível de *containers* do modelo C4 o diagrama atualizado do estado final da plataforma encontra-se descrito na Figura 6.30

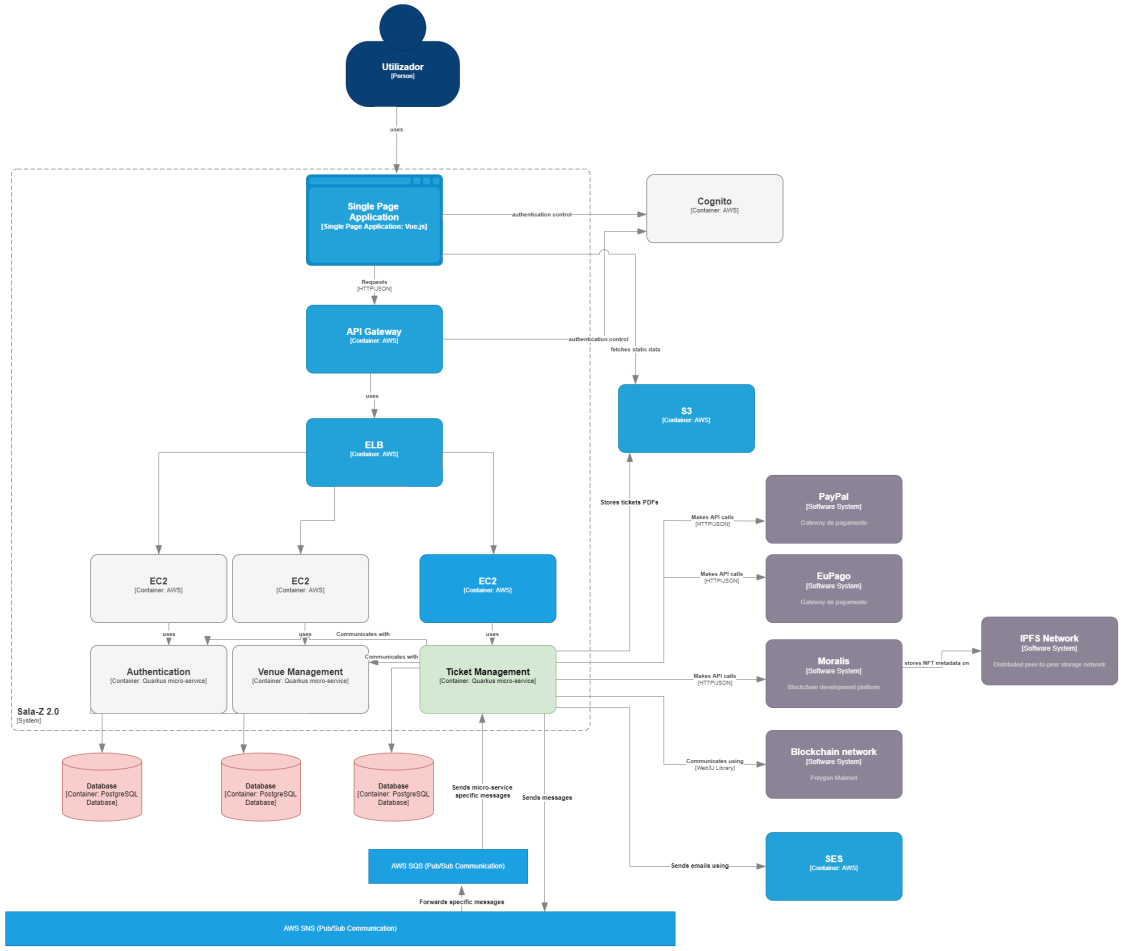


Figura 6.30: Nível de *containers* do modelo C4 final

Uma vez que o microsserviço de notificações não foi implementado, o microsserviço *Ticket Management* ficou encarregue de enviar os emails com os bilhetes. A comunicação entre serviços ficou implementada porém faltou o suporte nos outros microsserviços da plataforma. O microsserviço consegue publicar mensagens para o serviço SNS da AWS e consegue receber mensagens do mesmo através de uma fila própria SQS que filtra e recebe primeiro as mensagens do sistema direcionadas a esse microsserviço. Por último, acrescentou-se o serviço externo *PayPal* e ligou-se diretamente a comunicação com a *blockchain* através da *library Web3j* ao microsserviço.

No nível de componentes do microsserviço *Ticket Management* também houve alterações para suportar as novas responsabilidades, estando descrito na Figura 6.31.

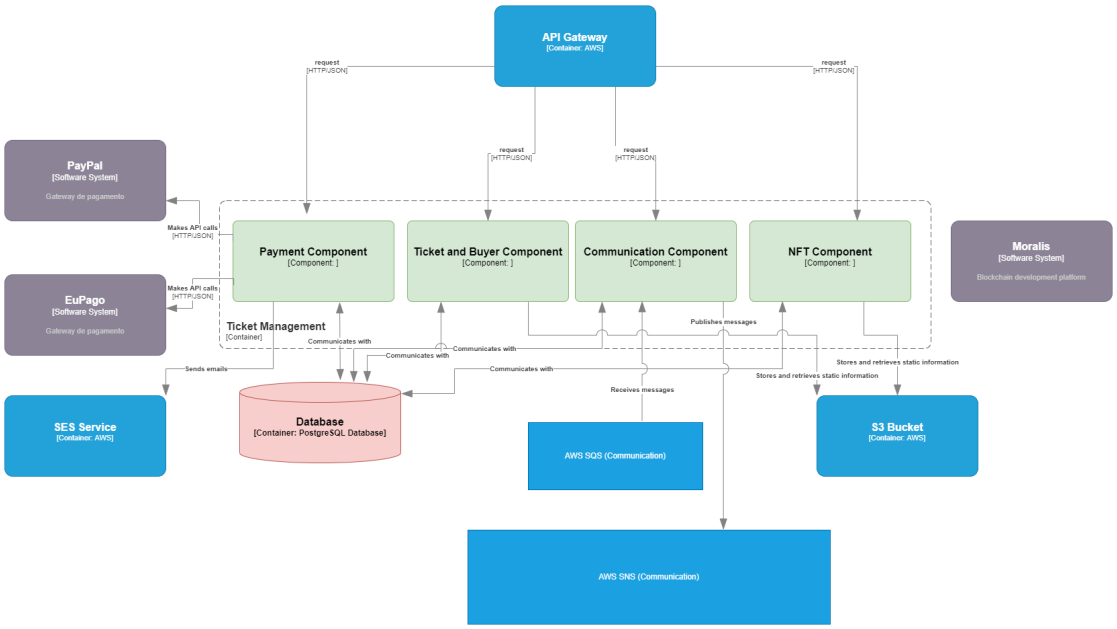


Figura 6.31: Nível de componentes do *Ticket Management* do modelo C4 final

Foram acrescentados mais dois componentes: *Ticket* e *Buyer Component* que tratam das ações CRUD das entidades *Ticket* e *Buyer*, e o componente *Communication* que trata da comunicação entre microsserviços descrita acima. Também foi incluído os serviços externos SES e *Paypal*.

6.6 Riscos Materializados

É importante perceber se as estratégias de mitigação dos riscos definidos na Secção 2.3 foram realmente eficazes a combater a eventual materialização destes riscos. Para tal, é feita esta análise na presente secção mencionando de seguida os riscos materializados.

- **Tecnologias** - A experiência do aluno nas tecnologias específicas impactou bastante no tempo de desenvolvimento do projeto. Mesmo com a estratégia

de mitigação posta em prática, o aluno sentiu algumas dificuldades durante o desenvolvimento após o período inicial alocado. Contudo, o planeamento inicial contribuiu para combater as consequências deste risco.

- **Tecnologias recentes** - A estratégia de mitigação para este risco provou-se bastante eficaz visto que ocorreram alterações a certos aspetos da tecnologia como nomeadamente a mudança da norma ERC 752 para ERC 1155, o que tornou possível cumprir com os requisitos do sistema. O aluno conseguiu adaptar-se facilmente às mudanças na tecnologia conseguindo mitigar com sucesso este risco.

- **Trabalho em conjunto** - Visto que o trabalho do aluno ficou a depender de algumas funcionalidades do colega de estágio este risco materializou-se. Porém, através das reuniões e chamadas na plataforma de comunicação, as duas entidades conseguiram integrar as suas próprias funcionalidades sem um acréscimo de tempo demasiado impactante de desenvolvimento.

Capítulo 7

Testes

De modo a validar o trabalho feito durante o semestre, é necessário definir e aplicar uma metodologia eficaz de testes. Neste capítulo é exposta a metodologia utilizada pelo aluno e os seus resultados.

7.1 Testes funcionais

O aluno realizou testes funcionais na plataforma percorrendo todas as funcionalidades implementadas na plataforma em ambiente de desenvolvimento, produzindo assim um artefacto presente no Apêndice B. Um exemplo de um registo desta tabela está demonstrado na Tabela 7.1.

User Story	Cenário	Descrição	Passos	Resultado esperado	Cumpriu o esperado?
US-1	Ter acesso à lista dos próximos eventos	Verificar se o utilizador consegue ver os próximos eventos e apenas os próximos eventos	- Entrar na plataforma e ver a lista de eventos	Ver todos os eventos na plataforma que estejam no futuro	Sim

Tabela 7.1: Excerto da tabela dos testes funcionais aplicados à plataforma

Ao realizar estes testes o aluno também inseriu numa outra tabela os *bugs* para os resolver mais facilmente. A Figura 7.1 mostra um excerto desta tabela com exemplos de *bugs* encontrados.

Esta tabela tinha cerca de trinta *bugs* encontrados e cerca de 80% dos erros foram

Id	Bug Title	Summary	Reproduction	Environment	Status
1	Login button label incorrect	Login button label incorrect when user is already logged in and in public portal	Log in and return to public portal page	LOCAL & DEV	Fixed
2	Bad redirection after venue creation	User is redirected to public portal after successfuly creating a venue when it shouldn't	Log in as admin, create a space and submit	LOCAL & DEV	Fixed
3	Audience is static	On event full details, the audience is set to all but I believe there is data to make this dynamic on db.	Public portal and click on an event, and check the data on left side	LOCAL & DEV	Fixed
4	INFINITE a -INFINITE label	On event full details, when there is no tickets generated for an event, the label presents text that's not user friendly. Should present something else (and possibly prevent user to click buy)	Public portal and click on an event, and check the data on leftside when there are no available tickets	LOCAL & DEV	Fixed
5	Share event buttons not functional	Social network buttons on event full details are not implemented and do not cause any action	Public portal and click on an event	LOCAL & DEV	Future Fix
6	Static countdown	Countdown in shopping cart is static and was not implemented	Shopping cart	LOCAL & DEV	Fixed
7	Event venue plant image is static	The image for the disposition of the event's venue is static and does not correspons to the actual layout.	Shopping cart	LOCAL & DEV	Future Fix
8	Event banner displays same INFINITY a -INFINITY label	Same as bug #4 only on shopping cart's event banner	Shopping cart	LOCAL & DEV	Fixed
9	"Privacidade e Segurança" and "Perguntas Frequentes" options on footer unavailable	These pages were not implemented therefore there is nothing to show	Footer	LOCAL & DEV	Future Fix
10	"Termos e Condições" page does not have production data	It has placeholder data instead of the actual terms and conditions	Terms and conditions page on footer	LOCAL & DEV	Future Fix
11	Prices labels show NaN when there is no ticket selected	When in the page for the first time, the UI selects a blank ticket that does not have data to show. Since tickets can have different prices there is no default price to show so it shows a NaN label	Shopping cart (Shopping Summary at right side)	LOCAL & DEV	Fixed
12	When user is logged in, ticket retrieval does not work on shopping cart	It does not show the tickets for that event. Probably has something to do with filter interceptor	Shopping cart of an event that has tickets on db	LOCAL & DEV	Fixed
13	Search events button does not have function	When cart is empty, a button shows up to search for events. This should redirect the user to public portal	Shopping cart and delete the first blank ticket	LOCAL & DEV	Fixed
14	Infinite ticket selection	User can infinitely click on "Novo Bilhete" button to generate new form. Although interface prevents user from moving forward in the process, UI should prevent button click if there is a form that is not complete	Shopping cart	LOCAL & DEV	Fixed
15	"Novo bilhete" is clickable when there is no other ticket left to buy on event	This button should only be enabled if there is tickets to select on the form	Shopping cart	LOCAL & DEV	Fixed

Figura 7.1: Excerto da tabela de *bugs* encontrados durante a fase de testes da plataforma

corrigidos. Infelizmente, devido a falta de tempo, esta fase de testes acabou prematuramente ficando a faltar testes mais formais nomeadamente, **unit testing** e **integration testing**. Parte do problema também se deveu ao planeamento ter sido muito otimista, visto que apenas um *sprint* de duas semanas dedicado aos testes não foi suficiente para a identificação e correção dos erros vindos dos mesmos.

7.2 Testes não funcionais

Para assegurar o cumprimento dos requisitos não funcionais descritos na Secção 4.3, o aluno teve que aplicar testes à plataforma que medissem estes atributos ou garantir através da arquitetura que estes atributos estão presentes.

Para o atributo de **segurança**, a arquitetura foi desenvolvida de modo a suportar a utilização de autenticação através do serviço *Cognito* da AWS. Em cima desta autenticação, na plataforma são utilizados *Roles* para cada utilizador dando assim permissões diferentes de acordo com a autorização desses *roles*. É utilizado ***JSON Web Tokens* (JWT)** para este objetivo. Porém, só utilizar este mecanismo para verificar o cumprimento do requisito é insuficiente logo, a plataforma foi testada para as vulnerabilidades mais comuns de uma plataforma *web*. Na Figura 7.2 é demonstrado os resultados do teste de um ataque *SQL Injection* realizado através da ferramenta **OWASP ZAP** e na Figura 7.3 os resultados de um ataque *Cross Site Scripting* (XSS).

Para o atributo de **modificabilidade** não foram precisos realizar quaisquer testes uma vez que pode ser assegurado pela arquitetura e implementação. Ao implementar uma arquitetura de microsserviços, o sistema está preparado para lidar com eventuais mudanças no código como novas funcionalidades. O sistema *Pub/-Sub* de filas de mensagens implementado no microsserviço *Ticket Management* também contribui para o desacoplamento entre serviços, o que torna o sistema mais independente reduzindo as chamadas diretas de API entre microsserviços.

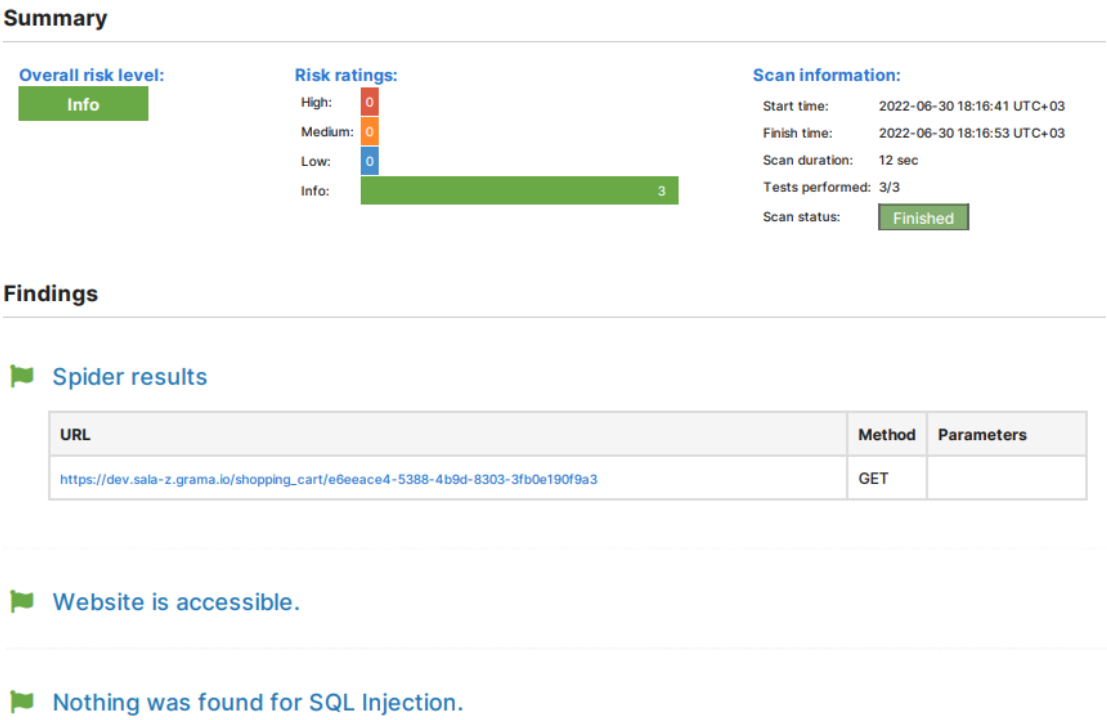


Figura 7.2: Resultados do teste de um ataque *SQL Injection*

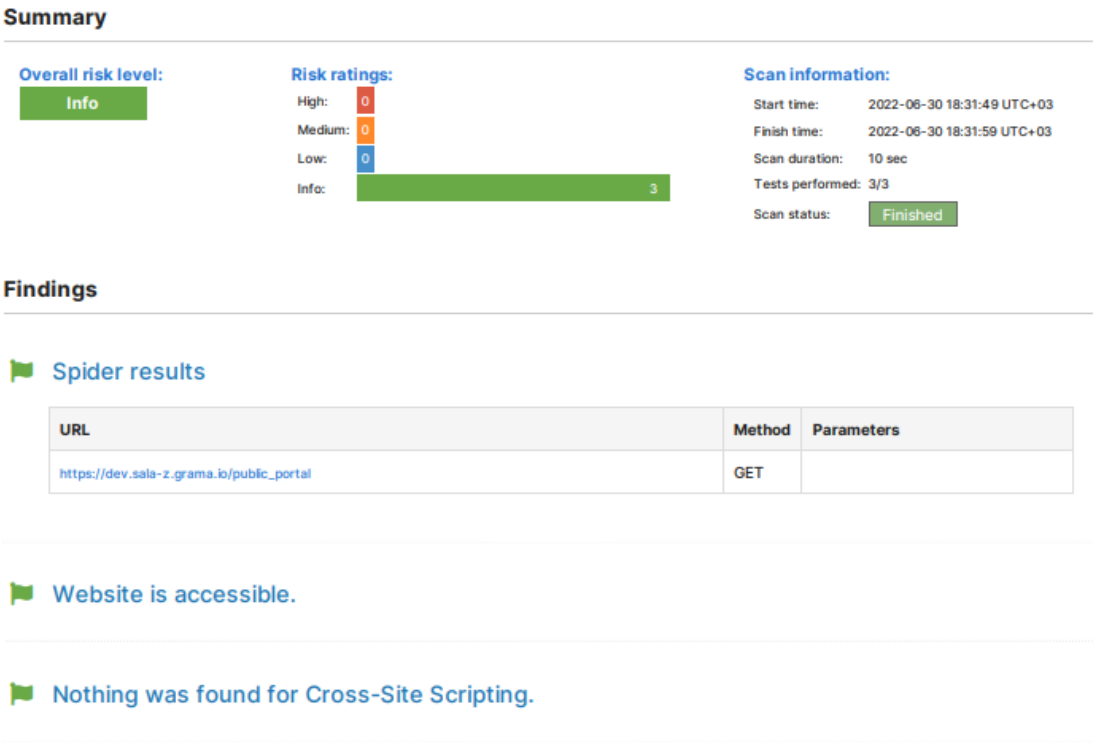


Figura 7.3: Resultados do teste de um ataque *XSS*

Capítulo 8

Conclusão

Neste capítulo vai ser recapitulada a proposta do estágio, os objetivos alcançados do primeiro semestre, os requisitos cumpridos no desenvolvimento do segundo semestre, um resumo do trabalho do segundo semestre e o trabalho futuro.

O aluno vai desenvolver um sistema de bilhética e gestão de pagamento para a aplicação interna da empresa Grama, Sala-Z. Este sistema vai permitir o pagamento de bilhetes para eventos da plataforma e possibilitar a emissão de *NFTs* numa *blockchain* de modo a garantir autenticidade e transparência ao cliente.

Durante o primeiro semestre, o aluno realizou o planeamento, requisitos e toda a análise das tecnologias a serem utilizadas e desenvolveu uma arquitetura para a integração destas novas funcionalidades.

No segundo semestre o aluno ficou responsável por criar um plano de trabalho e desenvolver o produto em questão cumprindo com maior parte dos objetivos esperados. Através de uma metodologia *agile*, o aluno desenvolveu cerca de 95% dos requisitos *Must have*, 25% dos requisitos *Should have*, e 20% dos requisitos *Could have*. Para avaliar a implementação das funcionalidades, o aluno realizou testes funcionais à plataforma de onde resultou um artefacto de listagem de *bugs* em que o mesmo resolveu maior parte deles. De seguida também avaliou os atributos de qualidade onde conseguiu determinar que o sistema consegue oferecer os atributos de segurança dos dados e modificabilidade.

O principal obstáculo na resolução do projeto foi claramente a falta de tempo. A carga do trabalho bem como a natureza das tecnologias envolvidas consumiram imenso tempo de aprendizagem. Contudo, o aluno adquiriu bastantes conhecimentos em vários aspetos de desenvolvimento de aplicações web:

- A construção de microsserviços em ambiente Java.
- Desenvolvimento de *front end* em *Javascript*.
- Utilização de serviços AWS como S3, SES, SNS, SQS e gestão dos mesmos na consola da AWS.
- Construção de ficheiros PDFs a partir de *templates* para envio como anexo em *emails*.

- *Deployment* de uma aplicação web em infraestrutura AWS.
- Conhecimento aprofundado da tecnologia *blockchain*
- Construção de *NFTs* numa rede *blockchain*

8.1 Trabalho futuro

Após a conclusão do estágio, o aluno notou que existe ainda trabalho para conseguir disponibilizar o próprio sistema de bilhética ao público. Mais atributos de qualidade e testes na plataforma têm que ser empregados de modo a conseguir garantir um produto seguro e estável pronto a ser utilizado por vários utilizadores em simultâneo.

Alguns requisitos planeados ficaram por implementar nomeadamente a transição de uma comunicação direta entre microsserviços para uma comunicação à base de mensagens em filas. Esta comunicação terá de ser implementada para todos os microsserviços. Outro requisito importante passará por implementar funcionalidades de *backoffice* para a geração dos bilhetes consoante a estrutura das salas dos espetáculos .

A integração de um utilizador comprador de bilhetes oficial pode ser um rumo da plataforma. Por enquanto, um espetador não tem qualquer registo permanente no Sala-Z e no futuro poderia ter um espaço próprio onde possa ver eventos do seu interesse, eventos a que já tenha ido e visualização e troca dos *NFTs* dos bilhetes que adquiriu.

Em relação à tecnologia *blockchain* existem funcionalidades bastante interessantes para o futuro. A possibilidade do utilizador poder pagar bilhetes com cripto moedas diretamente na plataforma sem qualquer tipo de entidade central. O desenvolvimento de um *smart contract* mais complexo de modo a descentralizar cada vez mais a plataforma. Um mercado secundário de revenda de bilhetes implementado diretamente na plataforma seria também algo bastante interessante de implementar.

References

[1] Bol - comprar bilhetes para espectáculos, . URL https://www.bol.pt/.

[2] Centaurify - the nft ticketing cryptocurrency of music, . URL https://www.centaurify.com/.

[3] Eip-1155: Multi token standard, . URL https://eips.ethereum.org/EIPS/eip-1155.

[4] Eip-721: Non-fungible token standard, . URL https://eips.ethereum.org/EIPS/eip-721.

[5] Live ethereum tps data, . URL https://ethtps.info/.

[6] Ethereum (eth) average transaction fees - messari, . URL https://messari.io/asset/ethereum/chart/txn-fee-avg.

[7] Event genius | tecnologia de bilheteira e cashless para ponto de venda, . URL https://pt.eventgenius.live/.

[8] Flow operators - flow documentation - flow blockchain, . URL https://docs.onflow.org/faq/operators/.

[9] Key concepts - flow documentation, . URL https://docs.onflow.org/flow-token/concepts/.

[10] Mint an nft with ipfs | ipfs docs, . URL https://docs.ipfs.io/how-to/mint-nfts-with-ipfs/#a-short-introduction-to-nfts.

[11] Openzeppelin, . URL https://www.openzeppelin.com/.

[12] Engage your community in your own nft marketplace | whitelabel nft hub by nft.kred, . URL https://www.nft.kred/.

[13] The crypto wallet for defi, web3 dapps and nfts | metamask, . URL https://metamask.io/.

[14] Polkadot coin, . URL https://cryptonews.com/coins/dot-polkadot/.

[15] Quarkus - qute reference guide, . URL https://quarkus.io/guides/qute-reference.

[16] Rest resource naming guide, . URL https://restfulapi.net/resource-naming/.

[17] Ticketline - bilhetes para eventos, . URL https://ticketline.sapo.pt/.

[18] Web3j, . URL https://docs.web3j.io/4.8.7/.

[19] Amazon. Amazon api gateway | api management | amazon web services, . URL https://aws.amazon.com/api-gateway/.

[20] Amazon. Amazon cognito - simple and secure user sign up & sign in | amazon web services (aws), . URL https://aws.amazon.com/cognito/.

[21] Amazon. Elastic compute cloud - amazon ec2 - aws, . URL https://aws.amazon.com/pt/ec2/.

[22] Amazon. Elastic load balancing, . URL https://aws.amazon.com/elasticloadbalancing/.

[23] Amazon. Amazon rds | cloud relational database | amazon web services, . URL https://aws.amazon.com/rds/.

[24] Amazon. Armazenamento s3 - simple storage service - amazon web services, . URL https://aws.amazon.com/pt/s3/.

[25] Amazon. Amazon simple email service | cloud email service | amazon web services, . URL https://aws.amazon.com/ses/.

[26] Amazon. Amazon simple notification service (sns) | messaging service | aws, . URL https://aws.amazon.com/sns/.

[27] Amazon. Amazon sqs | message queuing service | aws, . URL https://aws.amazon.com/sqs/.

[28] Autor anónimo. Ticketmint, 2021. URL https://ticketmint.org/.

[29] Simon Brown. The c4 model for visualising software architecture. URL https://c4model.com/.

[30] Amadeu Silveira Campanelli and Fernando Silva Parreiras. Agile methods tailoring – a systematic literature review. *Journal of Systems and Software*, 110:85–100, 2015. ISSN 0164-1212. doi: https://doi.org/10.1016/j.jss.2015.08.035. URL https://www.sciencedirect.com/science/article/pii/S0164121215001843.

[31] Sebastian Daschner. Thoughts on quarkus, 4 2019. URL https://dzone.com/articles/thoughts-on-quarkus.

[32] Instituto Nacional de Estatística. Bilhetes vendidos de espetáculos ao vivo (n.º) em portugal, 2020. URL https://www.ine.pt/xportal/xmain?xpid=INE&xpgid=ine_indicadores&indOcorrCod=0007576&contexto=bd&selTab=tab2&xlang=pt.

[33] Monika di Angelo and Gernot Salzer. Wallet contracts on ethereum – identification, types, usage, and profiles. 1 2020.

[34] emerchantpay. What is a payment gateway and how does it work?, 7 2019. URL https://www.emerchantpay.com/insights/what-is-a-payment-gateway-and-how-does-it-work/.

[35] EuPago. eupago - instituição de pagamento. URL https://www.eupago.pt/index.

[36] Github. framework · github topics. URL https://github.com/topics/framework.

[37] Linda John. How to create a cryptocurrency wallet? - easy guide. URL https://readwrite.com/2021/09/21/how-to-create-a-cryptocurrency-wallet/.

[38] Dawid Karczewski. What are the best frontend frameworks to use in 2021?, 4 2021. URL https://www.ideamotive.co/blog/best-frontend-frameworks.

[39] Shafaq Naheed Khan, Faiza Loukil, Chirine Ghedira-Guegan, Elhadj Benkhelifa, and Anoud Bani-Hani. Blockchain smart contracts: Applications, challenges, and future trends. *Peer-to-Peer Networking and Applications*, 14(5):2901–2925, April 2021. doi: 10.1007/s12083-021-01127-0. URL https://doi.org/10.1007/s12083-021-01127-0.

[40] Minjeong Kim, Yujin Kwon, and Yongdae Kim. Is stellar as secure as you think? pages 377–385. IEEE, 6 2019. ISBN 978-1-7281-3026-2. doi: 10.1109/EuroSPW.2019.00048.

[41] Polygon. Polygon | ethereum's internet of blockchains. URL https://polygon.technology/.

[42] Sumanta Pramanick. Top backend web frameworks 2021, 2 2021. URL https://www.kelltontech.com/kellton-tech-blog/top-7-backend-web-development-frameworks-in-2021.

[43] GET Protocol. The future of ticketing | get protocol. URL https://www.get-protocol.io/.

[44] Ferdinand Regner and André Schweizer. Nfts in practice-non-fungible tokens as core component of a blockchain-based event ticketing application blockchain-based process ecosystems view project wissenschaftscampus e-commerce view project. 2019. URL https://www.researchgate.net/publication/336057493.

[45] Nikita Savchenko. Decentralized applications architecture: Back end, security and design patterns. 4 2019. URL https://www.freecodecamp.org/news/how-to-design-a-secure-backend-for-your-decentralized-application-9541b5d8bd

[46] Sibs. Sibs payment gateway - sibs api market. URL https://www.sibsapimarket.com/sibs-payment-gateway/.

[47] Rajeev Sobti and Geetha Ganesan. Cryptographic hash functions: A review. *International Journal of Computer Science Issues, ISSN (Online): 1694-0814*, Vol 9:461 – 479, 03 2012.

[48] Stripe. Online payment processing for internet businesses - stripe. URL `https://stripe.com/en-pt`.

[49] Qin Wang, Rujia Li, Qi Wang, and Shiping Chen. Non-fungible token (nft): Overview, evaluation, opportunities and challenges. 5 2021. URL `http://arxiv.org/abs/2105.07447`.

[50] Dylan Yaga, Peter Mell, Nik Roby, and Karen Scarfone. Blockchain technology overview. 10 2018. doi: 10.6028/NIST.IR.8202.

[51] Anatoly Yakovenko. Solana: A new architecture for a high performance blockchain v0.8.13.

[52] Arthur Gervais ETH Zurich, Ghassan O Karame, Karl Wüst ETH Zurich, Eth Zurich, and Hubert Ritzdorf ETH Zurich. On the security and performance of proof of work blockchains vasileios glykantzis srdjan capkun. URL `https://bitcoin.org/en/developer-reference#data-messages`.

Appendices

Apêndice A

User Stories

ES-31: Como um utilizador não autenticado quero ter acesso à lista de eventos disponíveis para conseguir facilmente visualizar a informação dos eventos e comprar bilhetes.

US-1: Como um utilizador não autenticado quero visualizar a lista de próximos eventos para saber quais os eventos disponíveis para a compra de bilhetes.

US-2: Como um utilizador não autenticado quero visualizar uma lista de destaques para saber quais os eventos mais procurados e interessantes na plataforma.

US-3: Como um utilizador não autenticado quero procurar por um evento para conseguir encontrar facilmente um evento específico.

US-4: Como um utilizador não autenticado quero filtrar os eventos disponíveis por tipo de evento para encontrar eventos do meu interesse. Quero filtrar por um ou mais tipos de evento em simultâneo, sendo estes:

- Evento gratuito
- Festival
- Concerto
- Lazer

US-5: Como um utilizador não autenticado quero filtrar os eventos disponíveis por localidade para encontrar eventos nas áreas geográficas perto de mim. Quero filtrar por uma ou mais localidades em simultâneo, sendo estes os vários distritos de Portugal.

US-6: Como um utilizador não autenticado quero filtrar os eventos disponíveis por data para encontrar eventos a decorrerem num determinado intervalo de tempo.

US-7: Como um utilizador não autenticado quero ordenar os eventos disponíveis por data para obter uma lista ordenada cronologicamente. A ordenação é feita ascendente ou descendentemente.

US-8: Como um utilizador não autenticado quero consultar os detalhes de um evento para ficar informado sobre a informação adicional do mesmo. Quero ver informação sobre:

- Título
- Artista
- Local e Hora
- Detalhes do local e mapa
- Descrição
- Número máximo de pessoas
- Restrições de idade
- Tipo de evento
- Promotor
- Eventos semelhantes

ES-31: Como um utilizador não autenticado quero comprar bilhetes fácil e rapidamente para um evento para conseguir estar presente nos eventos em que estou interessado.

US-9: Como um utilizador não autenticado quero escolher o evento para realizar a compra do bilhete e comparecer no mesmo.

US-10: Como um utilizador não autenticado quero reservar o(s) meu(s) lugar(es) em eventos que seja possível a marcação de lugares para conseguir assistir aos mesmos sem problemas. Conforme o sítio, quero escolher o setor (Plateia ou Balcão), a fila e o número do lugar.

US-11: Como um utilizador não autenticado quero eliminar uma reserva de um lugar para refletir os meus interesses e possibilidades.

US-12: Como um utilizador não autenticado quero ver um resumo da(s) reserva(s) a ser(em) realizada(s) para acompanhar o estado da minha compra e o valor com IVA até ao momento.

US-13: Como um utilizador não autenticado quero reservar lugar(es) em eventos que não tenham lugares marcados para assegurar o meu acesso ao evento.

US-14: Como um utilizador não autenticado quero reservar lugar(es) em eventos que apenas tenham a escolha de um tipo de lugar para assegurar o meu acesso de acordo com as minhas preferências e informar-me das limitações de quantidade de lugares. As zonas/tipos escolhidos pelo utilizador de venue autenticado (Ex. Normal, Backstage, VIP, ...).

US-15: Como um utilizador não autenticado quero reservar lugar(es) em eventos escolhendo os lugares de acordo com as opções da venue para assegurar o

meu acesso tendo em conta as condições da venue. No caso de camarotes quero escolher o camarote, no caso de quadrado matinée quero escolher a zona, etc.

US-16: Como um utilizador não autenticado quero visualizar o preço de cada bilhete conforme o lugar escolhido para ter em conta as diferentes opções de lugares disponíveis.

US-17: Como um utilizador não autenticado quero preencher todos os meus dados de faturação para receber a fatura do pagamento do(s) meu(s) bilhete(s). Estes dados são:

- Nome e Apelido
- Email
- Telefone
- País
- NIF
- Morada de faturação
- Código Postal
- Localidade e Distrito

US-18: Como um utilizador não autenticado quero escolher o meu modo de pagamento para efetuar a compra do(s) bilhete(s). Os modos de pagamento que quero ter disponíveis são:

- Cartão de débito/crédito (Visa, Mastercard)
- MBWay

US-19: Como um utilizador não autenticado quero conectar a minha crypto wallet à aplicação para tomar posse dos bilhetes que adquiro.

US-20: Como um utilizador não autenticado quero autorizar ou não que os meus dados sejam utilizados para fins promocionais para ter controlo sobre os meus próprios dados.

US-21: Como um utilizador não autenticado quero ter acesso aos termos e condições e à política de privacidade e segurança da aplicação para me informar sobre o funcionamento legal da aplicação.

US-22: Como um utilizador não autenticado quero ver um resumo da minha compra com todos os dados que inseri para verificar se está tudo correto antes de avançar com a compra.

US-23: Como um utilizador não autenticado quero apagar um lugar antes de confirmar o meu pagamento para que no caso de haver um imprevisto, não tenha que realizar todo o processo de compra de bilhete para apenas remover um lugar.

US-24: Como um utilizador não autenticado quero editar a minha informação de faturação ou pagamento antes de confirmar o pagamento para corrigir eventuais erros nos dados que inseri.

US-25: Como um utilizador não autenticado quero editar o(s) lugar(es) escolhido(s) antes de confirmar o pagamento para não ter que repetir todo o processo de reserva de bilhete e inserção dos dados de faturação e pagamento.

US-26: Como um utilizador não autenticado quero receber no meu email a fatura dos meus bilhetes para ter provas que comprei os meus bilhetes.

US-27: Como um utilizador não autenticado quero visualizar um resumo da minha compra finalizada com detalhes sobre os bilhetes para ter a certeza de que não houve erros durante o pagamento.

ES-33: Como um utilizador não autenticado quero ter acesso fácil a todos os meus bilhetes comprados para conseguir mostrá-los durante o evento, guardá-los no final do evento e poderem ser verificados sobre a autenticidade dos mesmos.

US-28: Como um utilizador não autenticado quero visualizar todos os meus bilhetes comprados na minha crypto wallet para garantir que os bilhetes estão em minha posse.

US-29: Como um utilizador não autenticado quero revender os meus bilhetes num mercado secundário para não perder dinheiro no caso em que não possa estar presente no evento.

US-30: Como um utilizador não autenticado quero utilizar a minha crypto wallet durante o acesso ao evento para validar os meus bilhetes para esse evento.

US-31: Como um utilizador não autenticado quero manter os *NFTs* na minha crypto wallet após o evento para ter uma lembrança ou possivelmente vender a prova em como estive nesse evento.

ES-34: Como um utilizador de venue autenticado quero ter acesso fácil a todos os meus bilhetes emitidos para provar a autenticidade dos meus bilhetes e manter observação dos bilhetes revendidos.

US-32: Como um utilizador de venue autenticado quero emitir um número de bilhetes de acordo com a capacidade da venue para garantir que não existem mais bilhetes do que é suposto.

US-33: Como um utilizador de venue autenticado quero filtrar os bilhetes emitidos de acordo com o evento para ter uma maior organização no meu espaço de bilhetes emitidos.

US-34: Como um utilizador de venue autenticado quero ver quantos bilhetes foram comprados e quantos estão por vender para acompanhar o progresso de vendas de um evento.

US-35: Como um utilizador de venue autenticado quero poder cancelar o evento e consequentemente reemitir novos bilhetes *NFTs* para ajustar potenciais emergências relacionadas com o artista, não responsabilizando os compradores.

US-36: Como um utilizador de venue autenticado quero ver quantos e quais os bilhetes que foram revendidos para acompanhar o estado da revenda de bilhetes.

ES-35: Como um utilizador de venue autenticado quero publicar o meu evento no portal público do Sala-Z para que os utilizadores finais possam ver os meus eventos e comprar bilhetes.

US-37: Como um utilizador de venue autenticado quero selecionar a data do meu evento num calendário para definir a data tendo em conta as datas dos meus outros eventos.

US-38: Como um utilizador de venue autenticado quero visualizar as últimas atuações no espaço do meu evento para ver se o espaço é adequado aos meus objetivos.

US-39: Como um utilizador de venue autenticado quero configurar os meus bilhetes para poder vender bilhetes com a informação correta. Quero definir a seguinte informação:

- Tipo de sala
- Restrições de idade
- Lotação
- Custo do bilhete

US-40: Como um utilizador de venue autenticado quero escolher a tipologia da sala do evento para emitir bilhetes de acordo com os lugares da sala. Estes tipos podem ser:

- Sala com setores, filas e lugares
- Sala sem lugares marcados
- Sala dividida em zonas ou espaços

US-41: Como um utilizador de venue autenticado quero definir o preço dos bilhetes e lotação máxima para a área desse tipo de bilhete de acordo com a tipologia da sala do evento para distinguir os possíveis diferentes lugares na sala.

US-42: Como um utilizador de venue autenticado quero anexar ficheiros referentes ao evento para guardar documentos importantes que sirvam de apoio à organização.

US-43: Como um utilizador de venue autenticado quero inserir informação relacionada com a parte monetária da organização do evento para me organizar melhor sobre os custos e receitas do evento. Quero inserir informação sobre:

- Gastos do evento
- Receitas do evento

- Cachet dos artistas

US-44: Como um utilizador de venue autenticado quero editar um evento que criei para ajustar alterações que possam surgir. Os campos que quero alterar podem ser:

- Nome do evento
- Espaço
- Descrição
- Estado do evento
- Tags

US-45: Como um utilizador de venue autenticado quero tornar um evento público para permitir aos utilizadores finais verem o meu evento.

US-46: Como um utilizador de venue autenticado quero adicionar colaboradores ao evento para estes estarem a par de toda a informação relacionada.

ES-36: Como um utilizador de venue autenticado quero gerir os bilhetes vendidos para o meu evento para conseguir facilmente entender o estado atual da venda de bilhetes.

US-47: Como um utilizador de venue autenticado quero ver uma lista de bilhetes vendidos para o meu evento para organizar melhor o evento tendo em conta a lotação.

US-48: Como um utilizador de venue autenticado quero ver um mapa da sala com os lugares reservados (nos eventos em que seja possível) para saber visualmente os lugares reservados e os lugares por reservar.

US-49: Como um utilizador de venue autenticado quero ver estatísticas relevantes sobre a venda dos meus bilhetes para perceber se os eventos estão a ser lucrativos ou não. Estas estatísticas podem incluir:

- Percentagem de bilhetes vendidos para um evento
- Percentagem de bilhetes vendidos de todos os meus eventos (média)
- Percentagem de bilhetes revendidos no mercado secundário para um evento
- Percentagem de bilhetes revendidos no mercado secundário para todos os meus eventos (média)
- Bilhetes vendidos por zona/área de um evento (nos eventos em que seja possível)
- Lista ordenada por sucesso de eventos (eventos com maior percentagem de venda de bilhetes)

ES-37: Como um utilizador de venue autenticado quero ver informação relevante sobre o meu evento para facilmente ter em conta as métricas dos resultados dos eventos.

US-50: Como um utilizador de venue autenticado quero ver um calendário com os meus eventos para planear visualmente o decorrer dos mesmos.

US-51: Como um utilizador de venue autenticado quero ver uma lista dos meus próximos eventos para acompanhar e atribuir recursos consoante os eventos mais próximos.

US-52: Como um utilizador de venue autenticado quero ver a atividade recente dos meus eventos para acompanhar os processos de cada evento. Nesta atividade recente quero ver informação como edição de eventos, tarefas concluídas, material adicionado ao inventário, comentários e todas as ações significantes na plataforma.

US-53: Como um utilizador de venue autenticado quero ver uma lista de próximas tarefas dos meus eventos para ter uma maior organização nos processos do evento.

US-54: Como um utilizador de venue autenticado quero ver estatísticas como a audiência, novos espaços de atuação e presença em eventos para estar informado sobre os resultados dos meus eventos. Quero ter a opção de ver estas estatísticas por semana, mês, ano ou desde o início. Quero também filtrar estas estatísticas por artista.

US-55: Como um utilizador de venue autenticado quero ver um resumo de atividade por evento para ter uma maior organização nos processos do evento. Quero um resumo de notificações e de tarefas. Quero ver o número de tarefas concluídas, em progresso, por fazer e expiradas num gráfico.

US-56: Como um utilizador de venue autenticado quero ver um gráfico com as receitas e despesas de um evento para ver facilmente se o evento foi lucrativo ou não.

US-57: Como um utilizador de venue autenticado quero ver uma lista de artistas com dados dos eventos dos artistas para acompanhar o desempenho dos mesmos. Os dados relevantes sobre os artistas são:

- Audiência
- Receita
- Última atuação
- Número de atuações

Apêndice B

Testes funcionais

User story	Cenário	Descrição	Passos	Resultado esperado	Cumpriu o esperado?
US-1	Ter acesso à lista dos próximos eventos	Verificar se o utilizador consegue ver os próximos eventos e apenas os próximos eventos.	- Entrar na plataforma e ver a lista de eventos	Ver todos os eventos na plataforma que estejam no futuro	Sim
US-2	Ter acesso à lista de destaques	Não desenvolvido			
US-3	Procurar por um evento	Verificar se o utilizador consegue pesquisar por eventos através de 1 ou mais palavras chave	- Entrar no portal público - Pesquisar por um evento	Encontrar o evento ou eventos que contenham as palavras-chave inseridas	Sim
US-4	Filtrar os eventos por tipo	Verificar se o utilizador consegue filtrar os eventos por tipo de evento	- Entrar no portal público - Filtrar um evento por tipo de evento através do dropdown	Aparecerem apenas eventos filtrados pelo tipo de evento escolhido.	Sim
US-5	Filtrar os eventos por localidade	Verificar se o utilizador consegue filtrar os eventos por localidade	- Entrar no portal público -Filtrar um evento por localidade através do dropdown	Aparecerem apenas eventos filtrados pela localidade	Sim
US-6	Filtrar os eventos por data	Não desenvolvido			
US-7	Ordenar os eventos por data	Não desenvolvido			
US-8	Consultar os detalhes de um evento	Utilizador vê toda a informação de um evento	- Entrar no portal público - Selecionar um evento	Ver a informação de um evento: título, artista, local e hora, detalhes do local e mapa, descrição, número máximo de pessoas, restrição de idades, tipo de evento, promotor e eventos semelhantes	Sim
US-9	Escolher o evento para compra de bilhetes	Utilizador consegue selecionar um evento para a compra de bilhetes	- Entrar no portal público - Selecionar um evento - Avançar para a compra de bilhetes	O utilizador avança para a escolha de bilhetes do evento para a reserva	Sim
US-10	Reservar lugares marcados	Utilizador consegue selecionar bilhetes com lugares marcados para a reserva	- Navegar até ao carrinho de compras - Selecionar bilhetes	O utilizador vê os bilhetes a serem adicionados ao carrinho de compras	Sim
US-11	Eliminar lugares marcados	Utilizador consegue eliminar lugares na reserva que tenha inserido por engano	- Navegar até ao carrinho de compras - Selecionar bilhetes - Eliminar um bilhete	O utilizador vê o bilhete escolhido a ser eliminado da reserva	Sim
US-12	Ver um resumo da reserva	Utilizador consegue ver um resumo da reserva com informação sobre a reserva	- Navegar até ao carrinho de compras - Selecionar bilhetes	O utilizador vê dados importantes da reserva como: o número de bilhetes reservados e o seu preço, título do evento, espaço e horário	Sim
US-13	Reservar lugares não marcados	Utilizador consegue selecionar bilhetes com lugares não marcados para a reserva	- Navegar até ao carrinho de compras - Selecionar bilhetes	O utilizador vê os bilhetes a serem adicionados ao carrinho de compras	Sim
US-14	Reservar lugares com tipos de lugares	Não desenvolvido			
US-15	Reservar lugares de acordo com as opções da venue	Não desenvolvido			

US-16	Visualizar o preço dos bilhetes	Utilizador consegue ver o preço de um bilhete na reserva	- Navegar até ao carrinho de compras - Selecionar o bilhete - Ver o preço do bilhete	O utilizador vê o preço do bilhete no resumo da reserva	Sim
US-17	Preencher dados de faturação	Utilizador pode preencher os dados de faturação para a reserva	- Navegar até ao carrinho de compras - Selecionar bilhetes - Avançar para a secção dos dados pessoais	O utilizador insere todos os dados especificados no *user story* 17 e apenas depois é que pode avançar para a fase seguinte	Sim
US-18	Escolher o método de pagamento	Utilizador pode escolher o método de pagamento para a reserva	- Navegar até ao carrinho de compras - Selecionar bilhetes -Avançar para a fase de dados pessoais -Inserir os dados necessários - Avançar para a fase de pagamento	O utilizador pode selecionar o método de pagamento preferível	Sim
US-19	Conectar *crypto wallet*	Não desenvolvido			
US-20	Autorização de recolha de dados	O utilizador autoriza que os dados que inseriu sejam utilizados para fins promocionais	- Navegar até ao carrinho de compras - Selecionar bilhetes - Avançar para a fase de dados pessoais	O utilizador pode escolher se aceita ou não a utilização dos seus dados através da *checkbox* na área de dados pessoais	Sim
US-21	Acesso aos termos e condições e política de privacidade	O utilizador pode a qualquer momento aceder aos termos e condições da plataforma, bem como à política de privacidade	- Entrar na plataforma - Aceder às ligações no *footer* da aplicação	O utilizador é redirecionado para a página com as informações legais que necessita	Sim
US-22	Resumo da compra	No final da inserção de todos os dados, o utilizador pode ver a informação da reserva antes de a completar	- Navegar até ao carrinho de compras - Selecionar bilhetes -Avançar para a fase de dados pessoais -Inserir os dados necessários - Avançar para a fase de pagamento - Ver o resumo	O utilizador consegue ver toda a informação da reserva antes de a completar	Sim
US-23	Apagar lugares no resumo da compra	O utilizador consegue apagar lugares marcados durante o resumo da compra	- Navegar até ao carrinho de compras - Selecionar bilhetes -Avançar para a fase de dados pessoais -Inserir os dados necessários - Avançar para a fase de pagamento - Ver o resumo	O utilizador é redirecionado para a secção de reserva de lugares do carrinho de compras	Sim

US-24	Alteração de dados pessoais no resumo da compra	O utilizador consegue alterar dados pessoais durante o resumo da compra	- Navegar até ao carrinho de compras - Selecionar bilhetes -Avançar para a fase de dados pessoais -Inserir os dados necessários - Avançar para a fase de pagamento - Ver o resumo	O utilizador é redirecionado para a secção de dados pessoais do carrinho de compras	Sim
US-25	Editar lugares no resumo da compra	O utilizador consegue apagar lugares marcados durante o resumo da compra	- Navegar até ao carrinho de compras - Selecionar bilhetes -Avançar para a fase de dados pessoais -Inserir os dados necessários - Avançar para a fase de pagamento - Ver o resumo	O utilizador é redirecionado para a secção de reserva de lugares do carrinho de compras	Sim
US-26	Receber fatura no email	Após o pagamento, o utilizador recebe no email a fatura da reserva	- Concluir o processo de reserva de bilhetes	O utilizador recebe no email inserido a fatura da reserva	Sim
US-27	Ver resumo da reserva completada	Após o pagamento, o utilizador vê um resumo da reserva que adquiriu	- Concluir o processo de reserva de bilhetes	O utilizador é redirecionado para a página de sucesso de uma reserva	Sim
US-28	Ver o bilhetes na *crypto wallet*	Não desenvolvido			
US-29	Revender bilhetes NFT	Não desenvolvido			
US-30	Utilização da *wallet* para autenticação no evento	Não aplicável para teste			
US-31	Manter NFTs na *wallet*	O utilizador mantém os NFTs que adquiriu durante a compra	- Concluir o processo de reserva de bilhetes	Se não revender, estes NFTs continuam na posse do utilizador que realizou a compra	Sim
US-32	Emitir bilhetes para um evento	Não desenvolvido			
US-33	Filtrar bilhetes emitidos				
US-34	Visualizar os bilhetes emitidos	Não desenvolvido			
US-35	Publicar eventos no portal público	Um utilizador de venue é capaz de publicar um evento no portal público da plataforma	- Fazer login como administrador de venue - Criar um evento numa venue	Ao criar o evento, este fica disponível no portal público da plataforma	Sim
US-36	Ver o estado da revendo dos bilhetes	Não desenvolvido			
US-37	Selecionar a data do novo evento	Um administrador de venue consegue registar a data de um evento tendo em conta as datas dos eventos registados	- Fazer login como administrador de venue - Criar um evento numa venue	Consegue ver os eventos num calendário durante a criação	Sim
US-38	Visualizar últimas atuações	Não desenvolvido			
US-39	Configuração dos bilhetes	Não desenvolvido			
US-40	Definir tipologia da sala do evento	Não desenvolvido			
US-41	Definir preço dos bilhetes				
US-42	Anexar ficheiros referentes ao evento	Não desenvolvido			
US-43	Definir informação monetária do evento	Não desenvolvido			

US-44	Editar um evento	Como administrador de *venue*, pode editar um evento que tenha criado	- Fazer login como administrador de venue - Criar um evento numa venue - Editar um evento	Consegue alterar informações dos eventos como: nome do evento, espaço e descrição.	Sim
US-45	Tornar evento público	Como administrador de venue, pode tornar um evento público	- Fazer login como administrador de venue - Criar/Editar um evento numa venue	Consegue tornar o evento público estando assim disponível no portal público	Sim
US-46	Adicionar colaboradores	Não desenvolvido			
US-47	Ver lista de bilhetes vendidos	Não desenvolvido			
US-48	Ver mapa de lugares reservados	Não desenvolvido			
US-49	Ver estatisticas de venda	Não desenvolvido			
US-50	Ver um calendário com os eventos	Como administrador de *venue*, pode ver um calendário com os seus eventos	- Fazer login como administrador de venue - Ver eventos	Consegue ver os eventos num calendário	Sim
US-51	Ver lista dos próximos eventos	Como administrador de venue, pode ver uma lista dos próximos eventos	- Fazer login como administrador de venue - Ver eventos	Vê uma lista com os eventos que criou ordenados por data e filtrados pela data do evento	Sim
US-52	Ver atividade recente	Não desenvolvido			
US-53	Próximas tarefas	Não desenvolvido			
US-54	Estatatísticas do evento	Não desenvolvido			
US-55	Resumo de atividade de um evento	Não desenvolvido			
US-56	Ver gráfico receitas/despesas	Não desenvolvido			
US-57	Ver lista dos artistas com dados dos eventos	Não desenvolvido			