UNIVERSIDADE D
COIMBRA

João Miguel Branco da Silva

Plataforma de Criação e Gestão de Mapas de Salas de Espetáculo

VOLUME 1

Dissertação no âmbito do Mestrado em Engenharia Informática, especialização em Engenharia de Software orientada pelo Professor Doutor Raúl André Brajczewski Barbosa e pela Mestre Marta Mercier e apresentada à Faculdade de Ciências e Tecnologia / Departamento de Engenharia Informática.

Junho de 2021

Esta página é intencionalmente deixada em branco.

Agradecimentos

Em primeiro lugar, gostaria de agradecer à minha família, em especial aos meus pais e irmã, por todo o apoio e paciência durante os anos em que frequentei a Universidade.

Queria também agradecer especialmente à minha orientadora, Marta Mercier, por toda a motivação e ajuda durante a realização do estágio e também pelas revisões de código e da dissertação. Gostaria também de agradecer ao Gonçalo Amaral por todas as revisões e sugestões durante o desenvolvimento do projeto. Estou também muito grato, a todos os membros da The Loop Company, em especial à equipa de desenvolvimento da Ticketline, por me terem acolhido e dado a oportunidade de realizar o estágio curricular.

Ao meu orientador, Professor Raúl Barbosa, gostaria de agradecer todas as sugestões e todo conhecimento transmitido durante o decorrer do estágio curricular.

Finalmente, gostaria de agradecer a todos os meus amigos por todo o apoio e por me ajudarem a atingir este marco importante.

Esta página é intencionalmente deixada em branco.

Resumo

A Ticketline, uma das maiores empresas de bilhética nacional, possui um sistema complexo de várias plataformas para suportar toda a sua operação. Muitas destas plataformas estão ultrapassadas e, visto que a exigência dos clientes aumentou muito nos últimos anos,é necessário substituir as plataformas por outras mais adequadas às necessidades atuais.

O objetivo deste projeto é a criação de uma plataforma de criação e gestão de mapas de salas de espetáculo para a Ticketline. A nova plataforma, desenvolvida em Ruby on Rails, será dividida em diversos componentes com funcionalidades e objetivos específicos. Em primeiro lugar, será desenvolvida uma aplicação web para gerir mapas, que será responsável pelas operações CRUD e associação dos mapas a salas de espetáculo. Em adição, será desenvolvido um componente para importar os mapas da plataforma antiga e fazer a sua tradução para o novo modelo de dados, para que possam ser apresentados num widget que permitirá visualizar os mapas das salas e que poderá ser integrado em plataformas da Ticketline. Finalmente, será desenvolvido um componente que deverá permitir adicionar zonas e lugares aos mapas das salas, de forma interativa.

O presente documento descreve o desenvolvimento desta plataforma, seguindo uma metodologia de software. Este projeto seguirá práticas de engenharia de software, obtidas durante a frequência do autor no mestrado de Engenharia Informática, e detalha as fases de planeamento, levantamento do estado da arte, levantamento de requisitos e arquitetura do sistema, e também desenvolvimento e validação do sistema.

Palavras-Chave

Internet, Bilhética, Mapas Interativos, Salas de Espetáculo, Gestão, Eventos, Aplicação Web, API

Esta página é intencionalmente deixada em branco.

Abstract

Ticketline is one of the biggest ticketing companies in Portugal and therefore possesses a complex system of multiple platforms to support its operation. Since client demands have increased in the past few years, it is now necessary to replace those platforms with more adequate ones capable of fulfilling of all those demands.

The goal of this project is to develop a platform capable of creating and managing seat maps. The new platform will be developed in Ruby on Rails and divided into components with different and specific objectives. The first component is a web application capable of managing seat maps. Also, it should be responsible for all CRUD operations and associations of seat maps to rooms. The second component will import the old seat maps and translate them to the new data model so that it is possible to transition from the old plataform to the new one. The third component will be a widget for rendering the room maps. It should be possible to integrate this component in any Ticketline platform that needs it. Finally, the last component will be a seat map editor capable of drawing zones and seats that should be interactively added to the map.

This document describes the development of this platform following a software development methodology. The document will also follow practices of software engineering acquired by the author during his master's degree and detail the phases of planning, state of the art, system requirements, system architecture and also system development and validation.

Keywords

Internet, Ticketing, Interactive Seat Maps, Show Venues, Management, Events, Web Application, API

Esta página é intencionalmente deixada em branco.

Conteúdo

Esta página é intencionalmente deixada em branco.

Acrónimos

API Application Programming Interface. 23

CRUD Create, Read, Update and Delete. 23

JWT JSON Web Token. 51

ORM Object-Relational Mapping. 11

REST Representational State Transfer. 23

SMTP Simple Mail Transfer Protocol. 23

SOAP Simple Object Access Protocol. 23

SQL Structured Query Language. 11

WSDL Web Services Description Language. 23

XML Extensible Markup Language. 23

Esta página é intencionalmente deixada em branco.

Lista de Figuras

Esta página é intencionalmente deixada em branco.

Lista de Tabelas

Esta página é intencionalmente deixada em branco.

Capítulo 1

Introdução

O projeto aqui apresentado é o desenvolvimento de uma ferramenta de criação e gestão de mapas de salas de espetáculo desenvolvida por um aluno do Mestrado de Engenharia Informática no âmbito da disciplina de Dissertação/Estágio em Engenharia de Software na Faculdade de Ciências e Tecnologias da Universidade de Coimbra. A ferramenta será desenvolvida durante o estágio na The Loop Company sob orientação do Professor Doutor Raúl Barbosa, do Departamento de Engenharia Informática, e Mestre Marta Mercier, da The Loop Company.

1.1 The Loop Company

A The Loop Company é uma empresa que se iniciou em 2016 com o lançamento do projeto Book in Loop, uma plataforma inovadora de reutilização de manuais escolares. Inicialmente, o grande foco da empresa era o desenvolvimento de soluções que incentivassem a economia circular. No entanto, a necessidade de desenvolver plataformas para suportar estes produtos levou à criação de uma equipa com elevadas capacidades nas áreas de ciência de dados e tecnologia. Desde então, a empresa passou a desenvolver soluções tecnológicas para clientes externos, como é o caso da Ticketline. Apesar desta transição, as preocupações ambientais mantiveram-se, assim como a vontade de tornar a economia mais circular.

1.2 Contexto e Motivação

O cliente deste projeto é a Ticketline, a maior empresa de venda de bilhetes a nível nacional. Fundada há mais de 20 anos, apresenta uma grande rede de pontos de venda e um website que permitem aos seus clientes adquirir os seus ingressos onde e quando desejarem.

O sistema que suporta toda esta operação é complexo e envolve uma série de plataformas que interagem entre si. Existem plataformas para gestão e administração como a plataforma de gestão de eventos e gestão de salas. Há plataformas específicas para venda de bilhetes como o website e pontos de venda e, finalmente, está disponível uma área para que os promotores possam consultar detalhes e estatísticas dos seus eventos que se encontrem à venda.

Apesar de existirem todas estas plataformas, o foco deste projeto incidirá sobre a plataforma de gestão de salas, que permite criar e gerir mapas de salas de espetáculo. Adicional-

mente, é responsável por fornecer as representações dos mapas das salas de espetáculo ao website e aos POS, para que o cliente possa escolher o lugar que quer reservar, e também à plataforma de gestão de eventos, para que possa ser feita a configuração individual de cada evento.

Estas plataformas foram desenvolvidas quando a empresa nasceu e, desde então, as necessidades aumentaram consideravelmente, assim como a exigência dos clientes relativamente ao comércio online. Por essa razão, a Ticketline decidiu que as suas plataformas necessitavam de renovação e a The Loop Company foi a empresa escolhida para a execução desta tarefa.

1.3 Objetivo

O objetivo deste projeto é desenvolver uma nova plataforma que substitua o software utilizado atualmente para criar e gerir mapas de salas de espetáculo.

A plataforma será uma aplicação web e deverá ser integrada com outras plataformas para fornecer os mapas. Adicionalmente, deverá existir um ferramenta que importe e traduza o modelo das salas antigas para a nova plataforma. Em maior detalhe, a plataforma pode ser descrita da seguinte forma:

A aplicação web será a forma dos gestores de salas acederem às funcionalidades desta plataforma. Aqui, os gestores poderão consultar as informações relativas aos locais dos eventos e respectivas salas de espetáculo. Em adição, será possível desenhar mapas e associá-los às salas. A criação dos mapas será realizada através de um editor que permitirá incluir os componentes que compõem a sala de forma interativa.

Para integrar os mapas das salas nas outras plataformas, será criado um widget que, suportado por uma API, permitirá renderizar o mapa e realizar uma série de operações. O widget deverá permitir consultar a disponibilidade dos lugares em tempo real e selecionar os lugares para que seja possível efetuar reservas. Nas plataformas onde for integrado, o widget servirá como forma de selecionar lugares para que as plataformas possam realizar as operações que necessitam dessa informação.

Finalmente, a ferramenta de tradução deve permitir traduzir a informação presente nas bases de dados do sistema atual para modelos de dados que o novo sistema consiga interpretar e utilizar.

As funcionalidades suportadas pela plataforma serão descritas em maior detalhe no capítulo dos Requisitos.

Esta página é intencionalmente deixada em branco.

Capítulo 2

Planeamento

Neste capítulo são descritas as tarefas a realizar durante o projeto e o seu enquadramento temporal. Seguidamente, são analisados os riscos inerentes a este projeto e definidas estratégias de mitigação. Finalmente, são analisadas as ferramentas de desenvolvimento e descritas as ferramentas de apoio ao projeto.

2.1 Tarefas

Esta secção descreve a lista de tarefas a realizar durante o projeto, após uma breve análise da proposta de estágio curricular a realizar na The Loop Company.

2.1.1 Primeiro Semestre

O plano de trabalho para o primeiro semestre consiste em estudar o problema, levantar requisitos junto do cliente e desenhar a arquitetura do sistema.

Planeamento

Na tarefa de planeamento é feita toda a preparação necessária para iniciar o projeto. É escolhida uma metodologia de desenvolvimento de software e são definidas as tarefas a realizar. É produzido um diagrama de Gantt com a organização das tarefas pelos semestres.

Levantamento do Estado da Arte

A tarefa de levantamento do estado da arte implica a análise da plataforma de gestão de salas utilizada atualmente, de ferramentas que permitem a criação de mapas interativos de salas de espetáculo e de web services para realizar a integração da plataforma. Os estudos realizados nesta tarefa permitem entender melhor o contexto deste projeto e tomar decisões mais informadas.

Levantamento de Requisitos e Restrições

A tarefa de levantamento de requisitos e restrições envolve reunir com o cliente para obter

uma descrição do sistema a implementar. Nesta tarefa, são produzidas as histórias de utilizador e identificados atributos de qualidade e restrições.

Design do Sistema

A tarefa de design do sistema implica realizar a descrição da plataforma através de diagramas que permitam entender o funcionamento do sistema.

Desenvolvimento do relatório intermédio

Visto que este projeto é realizado no âmbito de uma tese de mestrado em Engenharia de Software, no final do primeiro semestre é necessário submeter um relatório sobre o trabalho desenvolvido.

Preparação da apresentação/defesa intermédia

Além da entrega do relatório intermédio, é necessário apresentar e defender o trabalho realizado perante um júri constituído por professores do Departamento de Engenharia de Informática.

Estudo do desenvolvimento de aplicações em Ruby on Rails e React

Como forma de preparação para o desenvolvimento da plataforma, no final do primeiro semestre são realizados tutoriais de Ruby on Rails e React para aprofundar o conhecimento sobre estas *frameworks*.

2.1.2 Segundo Semestre

No segundo semestre, o plano de trabalho consiste em desenvolver a plataforma, realizar testes ao sistema e implementar eventuais melhorias e/ou correções que possam ser necessárias. No final do semestre, é feita a entrega da plataforma ao cliente.

Desenvolvimento da Plataforma de Gestão de Salas

O desenvolvimento da plataforma de gestão de salas consiste no desenvolvimento do painel de gestão de salas, responsável pela gestão das informações dos locais, salas e mapas, e do editor, responsável pela criação dos mapas de salas de forma interativa. Implica também a construção de um widget e API para integrar os mapas nas outras plataformas e de uma ferramenta que permita realizar a tradução do conteúdo presente nas bases de dados do sistema de gestão de salas antigo para modelos de dados que o novo sistema consiga interpretar.

Testes à Plataforma de Gestão de Salas

Nesta tarefa, são realizados testes individuais às várias funcionalidades implementadas, testes de integração entre as várias plataformas e testes de sistema.

Melhorias e/ou correções à Plataforma de Gestão de Salas

Nesta fase, os problemas detetados durante a tarefa anterior são corrigidos e são implementadas eventuais melhorias que a equipa de desenvolvimento considere possível.

Desenvolvimento do relatório final

Tal como no primeiro semestre, no final do segundo semestre é necessário submeter um relatório com a descrição do projeto desenvolvido durante o estágio na The Loop Company.

Preparação da apresentação/defesa final

Além da submissão do relatório, é necessário apresentar e defender o trabalho realizado perante um júri constituído por professores do Departamento de Engenharia Informática.

2.2 Planeamento Temporal

Nesta secção, as tarefas são divididas em subtarefas e são apresentadas estimativas temporais para a duração destas tarefas através de diagramas de Gantt. Posteriormente, é realizada uma comparação entre as estimativas e a duração real das tarefas e são justificadas eventuais inconformidades.

2.2.1 Plano do Primeiro Semestre

As figuras 2.1 e 2.2 apresentam o enquadramento temporal das tarefas do primeiro semestre. As primeiras tarefas a realizar no primeiro semestre servem para a escolha de metodologia, divisão de tarefas e definição da estrutura do relatório. De seguida, é feito o levantamento do estado da arte que consiste na análise da plataforma de gestão de salas da Ticketline, de *Web Services* e de ferramentas de criação de mapas interativos. Esta tarefa prolonga-se até à segunda semana de novembro. A fase seguinte inicia-se com uma reunião com

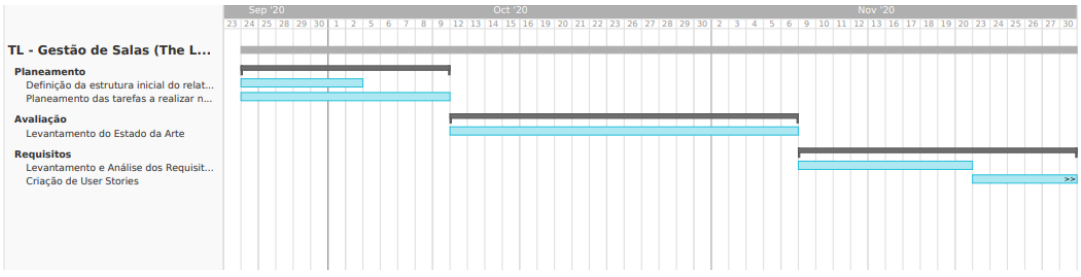


Figura 2.1: Diagrama de Gantt do plano do primeiro semestre - Parte 1

cliente tendo em vista o levantamento dos casos de uso do sistema. Seguem-se as tarefas de análise e identificação dos requisitos, e criação de histórias de utilizador, que se estendem até à primeira semana do mês de dezembro. Posteriormente, são descritos os atributos de qualidade e as restrições inerentes a este projeto.

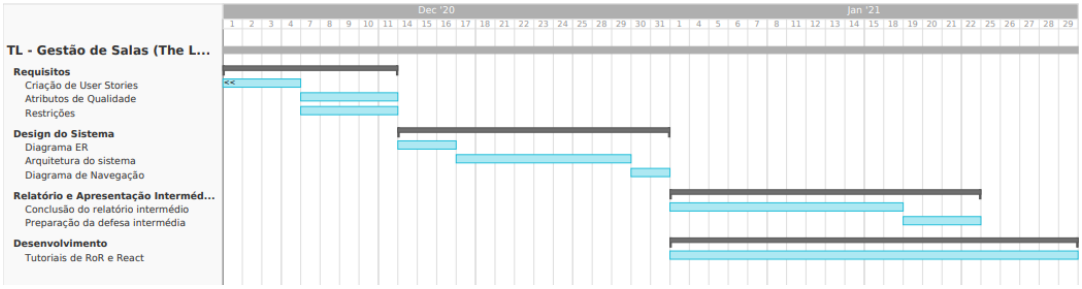


Figura 2.2: Diagrama de Gantt do plano do primeiro semestre - Parte 2

Nas últimas três semanas do mês de dezembro decorre a fase de design do sistema que se inicia com a descrição do modelo de dados através de um diagrama Entidade-Relacionamento. Nas semanas seguintes segue-se a criação dos diagramas de arquitetura e de navegação.

Nas duas primeiras semanas de janeiro é finalizado o relatório intermédio, e, na semana seguinte, é preparada uma apresentação para a defesa intermédia. Em paralelo, são realizados tutoriais de Ruby on Rails e React como forma de preparação para o segundo semestre.

2.2.2 Resultado do Primeiro Semestre

Durante o primeiro semestre ocorreram alguns desvios temporais em relação ao plano estabelecido no início do semestre. A tarefa de levantamento e análise de requisitos sofreu um atraso pois a previsão efetuada era claramente otimista tendo em conta a quantidade de requisitos deste projeto. Após reunir com o cliente, foi necessário recorrer ao feedback iterativo do modelo Waterfall e realizar algumas correções nos requisitos. Estas correções estenderam-se até à primeira semana de janeiro. Devido a este atraso, foi necessário condensar as tarefas de descrição da arquitetura do sistema na segunda semana de janeiro. Nesta fase, foram marcadas várias reuniões para definir a arquitetura do sistema mais rapidamente que permitiram cumprir o plano estipulado para o primeiro semestre.

2.2.3 Plano do Segundo Semestre

O segundo semestre destina-se ao desenvolvimento da plataforma e à realização dos testes e correções necessárias, como pode ser consultado nas figuras 2.3 e 2.4.

A primeira tarefa é o desenvolvimento do painel de gestão(*Dashboard*) que permite gerir os locais e salas de espetáculo e tem a duração de três semanas. Segue-se o desenvolvimento do tradutor do modelo de dados até à primeira semana do mês de março.

2.2.4 Resultado do Segundo Semestre

Durante o segundo semestre também ocorreram alguns desvios temporais. Em primeiro lugar, o desenvolvimento do dashboard estendeu-se duas semanas mais que o proposto. Isto deve-se ao facto do autor ainda não estar totalmente familiarizado com a linguagem Ruby on Rails. O atraso no desenvolvimento do deste componente fez com que a data de início do desenvolvimento dos restantes componentes fosse adiado também duas semanas. Por essa razão, a tarefa de testes e correções passou a ter apenas a duração de duas sema-

nas. Durante estas semanas o autor focou-se maioritariamente na resolução de problemas detectados nas revisões de código.

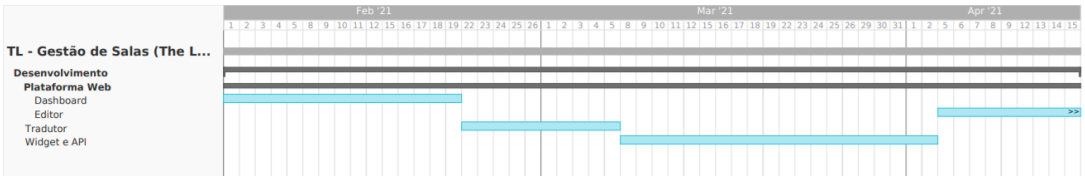


Figura 2.3: Diagrama de Gantt do plano do segundo semestre - Parte 1

Nas quatro semanas seguintes é desenvolvido o widget de visualização de salas e a API que o suporta. A última tarefa de desenvolvimento é a criação do editor para desenhar os mapas de salas de forma interativa. Esta é a tarefa mais demorada e dura até metade do mês de maio.

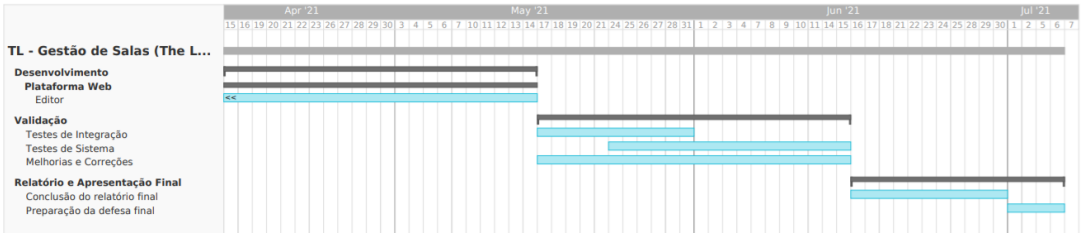


Figura 2.4: Diagrama de Gantt do plano do segundo semestre - Parte 2

Seguidamente, inicia-se a fase de testes que contempla a realização de testes de integração e de sistema e são efetuadas as correções e melhorias que sejam necessárias. As tarefas de testes e melhorias duram até meio do mês de junho e, posteriormente, inicia-se a conclusão do relatório final de estágio. Depois da entrega do relatório é preparada uma apresentação para a defesa final que ocorre durante o mês de julho.

2.3 Metodologia

Após analisar a disposição de tarefas apresentada na proposta de estágio da The Loop Company, existe uma clara divisão de tarefas a realizar nos dois semestres. No primeiro semestre as tarefas consistem na análise e planeamento do projeto, levantamento do estado da arte, levantamento e análise de requisitos e design do sistema. O segundo semestre é composto por tarefas de desenvolvimento, testes e entrega ao cliente. Tendo em conta esta distribuição, o modelo Waterfall, presente na figura 2.5, foi a metodologia escolhida para este projeto.

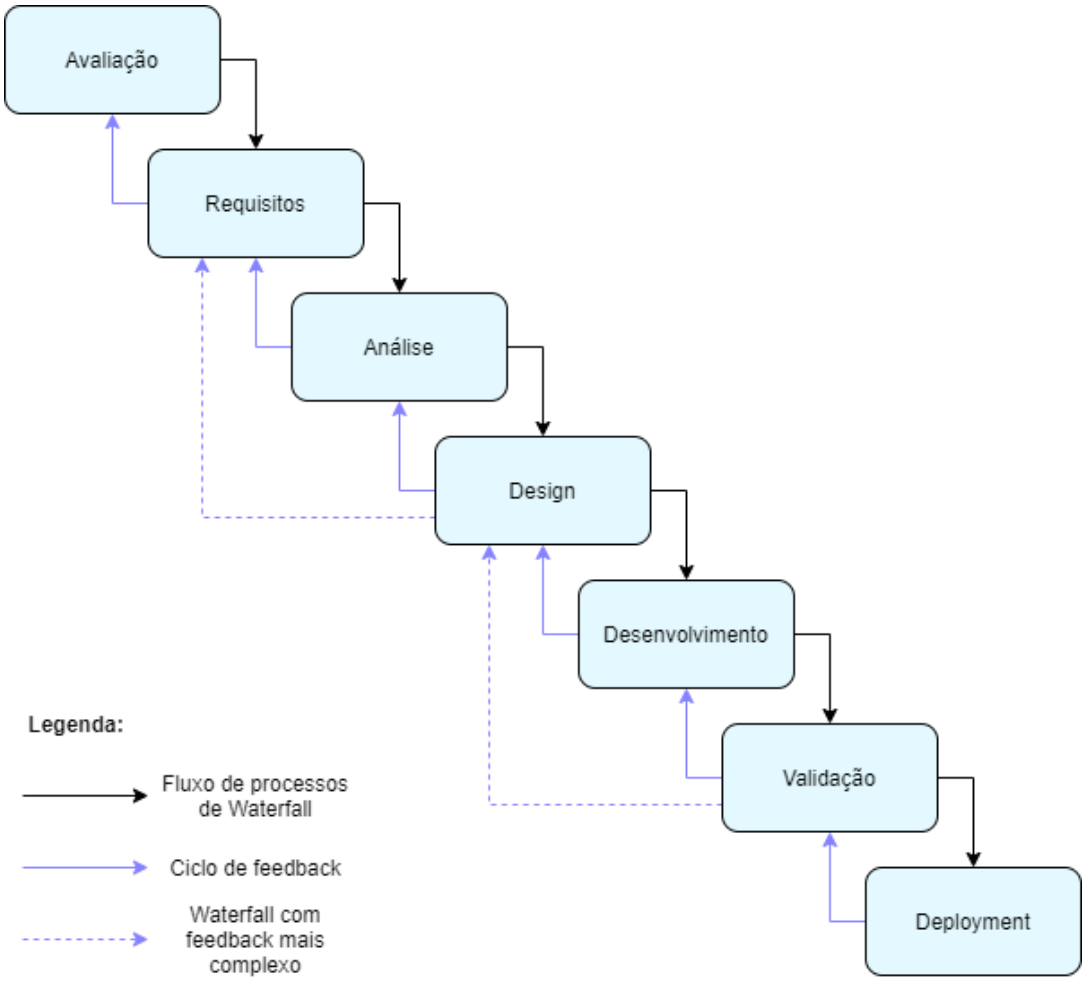


Figura 2.5: Modelo Waterfall com feedback iterativo [1]

O modelo Waterfall é uma metodologia de desenvolvimento de software que foi introduzida em 1970 por Winston Walker Royce. Neste modelo, as tarefas são divididas em Requisitos, Design, Desenvolvimento, Validação e Deployment e o progresso deve ser feito de forma sequencial, ou seja, só se avança para fase seguinte assim que a fase anterior esteja concluída. Em alguns casos, como no caso deste projeto, é adicionada uma fase de Avaliação devido à necessidade de realizar um estudo do Estado da Arte no início do projeto.

Apesar do declínio de popularidade, o modelo Waterfall ainda é bastante utilizado em diversos projetos. No entanto, a dificuldade em lidar com mudanças pode ser um problema visto que quando se trabalha com um cliente, como é o caso deste projeto, é natural existirem alterações no decorrer do projeto. Por essa razão, será importante estabilizar os requisitos e manter o cliente envolvido nessa fase, de forma a minimizar o número de mudanças durante o projeto e, consequentemente, evitar atrasos.

2.4 Gestão de Riscos

O processo de desenvolvimento de software envolve muitos fatores e, por essa razão, é necessário identificar os riscos que podem impedir o sucesso do projeto e definir estratégias que permitam diminuir a sua probabilidade.

Na tabela 2.1, são identificados riscos inerentes a este projeto e é-lhes atribuída uma

probabilidade de ocorrência e o impacto que podem ter nos objetivos definidos para este estágio.

Tabela 2.1: Tabela de Riscos

ID	Risco	Probabilidade	Impacto
1	Contágio por COVID-19 que cause atrasos no projeto.	Baixa	Baixo
2	Estabilização dos requisitos leva a atrasos no projeto.	Média	Médio
3	A aprendizagem das tecnologias demora mais tempo do que o esperado.	Baixa	Baixo
4	Testes revelam a necessidade de muitas correções	Baixa	Médio
5	Dificuldades na integração com outras plataformas	Média	Alta
6	Dependência de outras plataformas pode causar atrasos no desenvolvimento	Média	Médio
7	O modelo de dados antigo não permite uma tradução completa para o novo modelo de dados	Médio	Médio

As estratégias de mitigação para gerir os riscos identificados são as seguintes:

1. Seguir as indicações da Direção Geral de Saúde para evitar um contágio por COVID-19.
2. Alocar mais tempo para o levantamento de requisitos.
3. Reservar tempo para a aprendizagem das tecnologias no primeiro semestre.
4. Deixar uma margem para a execução de correções até ao fim do projeto.
5. Escolher tecnologias adequadas para integração de sistemas.
6. Desenvolver outros funcionalidades enquanto os sistemas não estiverem prontos.
7. Em último caso, criar um sistema semelhante ao atual para os mapas que forem importados.

2.5 Ferramentas

Nesta secção, são analisadas as ferramentas utilizadas para o desenvolvimento deste projeto e são descritas outras ferramentas importantes para a organização do trabalho e comunicação com a equipa.

2.5.1 Ferramentas de Desenvolvimento

Para o desenvolvimento de uma aplicação web existem, neste momento, muitas linguagens e *frameworks* que facilitam o trabalho dos programadores ao disponibilizarem bibliotecas que auxiliam a implementação de arquiteturas comuns a este tipo de projetos.

Para este projeto, a equipa da The Loop Company já tinha realizado estudos prévios e decidiu que a plataforma seria desenvolvida em Ruby on Rails e PostgreSQL seria o SGBD

utilizado. Adicionalmente, para implementar o *front-end* do editor de salas foi escolhida a biblioteca React, devido à necessidade de maior interatividade neste componente da plataforma. Assim sendo, é importante realizar um estudo sobre estas ferramentas para entender melhor os seus princípios de funcionamento.

Back-End

Ruby on Rails[4] é uma *framework* de desenvolvimento de aplicações web que utiliza a linguagem Ruby. Foi desenhada com a intenção de facilitar o desenvolvimento destas aplicações ao fazer previsões sobre o que os programadores necessitam para começar a desenvolver. Adicionalmente, permite aos programadores escrever menos código para atingir o mesmo ou mais do que outras linguagens e *frameworks*.

A sua filosofia baseia-se em dois princípios[5]: Não se repita (*Don't Repeat Yourself*) e Convenção em vez de Configuração (*Convention Over Configuration*). O primeiro princípio indica que cada pedaço de conhecimento deve ter uma representação única, confiável e sem ambiguidades dentro de um sistema. O segundo princípio afirma que, ao contrário de muitas *frameworks*, a Ruby on Rails não necessita de configurações extensas e detalhadas, pois já existe uma configuração utilizada por defeito.

A *framework* Ruby on Rails segue o padrão MVC, ou Modelo-Vista-Controlador, muito utilizado para o desenvolvimento de aplicações web. O padrão MVC pode ser dividido em três partes:

O Modelo é responsável pela gestão e manipulação da informação, a Vista controla a forma de apresentação da informação ao utilizador e, por último, o Controlador interpreta os pedidos do utilizador e gere a lógica entre Modelo e Vista, para poder responder de forma apropriada. As interações entre estes componentes podem ser visualizadas na figura 2.6.

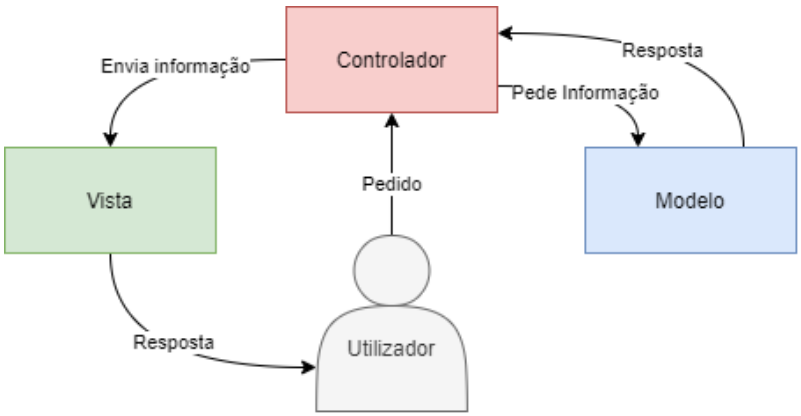


Figura 2.6: Arquitetura Modelo-Vista-Controlador [2]

A *framework* Ruby on Rails disponibiliza bibliotecas que facilitam a implementação desta arquitetura, sendo elas:

- A biblioteca **Active Record**[6], que facilita a criação e utilização de objetos cuja informação necessita de ser guardada numa base de dados. É uma implementação do padrão Active Record[7] que utiliza técnicas de *Object-Relational Mapping (ORM)* para evitar a necessidade de escrita de métodos SQL.
- A biblioteca **Action View**[8], responsável por converter a informação transmitida pelo Controlador para respostas aos pedidos feitos pelo utilizador.

- A biblioteca **Action Controller**[9], responsável por fazer pedidos ao Modelo por informações específicas e organizar essas informações da forma que a Vista necessita para apresentar ao utilizador.

Em adição, existe uma grande quantidade de outras bibliotecas que se encontram disponíveis na RubyGems[10].

Front-End

O desenvolvimento do front-end de um aplicação web implica, habitualmente, a utilização de três linguagens: HTML, CSS e JavaScript. A junção destas três linguagens é o que possibilita a criação de páginas estruturadas, estilizadas e interativas.

HTML, ou Hypertext Markup Language, é a linguagem de marcação utilizada para criar páginas web. Permite descrever a estrutura e organização de uma página web através da adição de elementos predefinidos que indicam ao browser como deve exibir o conteúdo ao utilizador[11].

CSS, ou Cascading Style Sheets, é uma linguagem que permite estilizar um documento HTML ao indicar como os vários elementos que compõem a página devem ser apresentados[12].

JavaScript é a linguagem utilizada para criar e controlar conteúdo dinâmico dentro de um website. Através desta linguagem é possível invocar métodos de uma API ou modificar informações dentro de uma página sem ter de a atualizar manualmente[13].

Como o componente de edição necessita de um elevado nível de interatividade, foi escolhida a biblioteca React em vez de JavaScript puro. A principal vantagem da utilização desta biblioteca é a facilidade de atualizar a interface gráfica cada vez que existem alterações nos dados.

React é uma biblioteca de JavaScript utilizada para construir interfaces de utilizador. Essas interfaces são construídas através da junção de componentes independentes, isolados e reutilizáveis. Adicionalmente, existem estados que permitem atualizar a interface assim que são alterados. Esta arquitetura permite tornar o processo de desenvolvimento de *front-End* mais organizado e eficiente.

Sistema de Gestão de Base de Dados

Um Sistema de Gestão de Base de Dados é um software desenhado para guardar, retirar, definir, e gerir informações numa base de dados[14]. Habitualmente, estes sistemas incorporam o modelo de dados relacional que permite a definição de relações entre os dados através de valores de dados em tabelas, garantindo maior independência entre os dados.

A escolha de PostgreSQL[15] justifica-se com o facto de ser *open-source* e muito popular na comunidade de programadores de Rails, onde se inclui a The Loop Company que utiliza habitualmente este SGBD.

2.5.2 Ferramentas de Apoio ao Projeto

Overleaf Editor de LaTeX online que permite a partilha de documentos. Neste projeto, é utilizado para desenvolver o relatório de estágio.

Trello Ferramenta de gestão de projetos online. É utilizada neste projeto para organização das tarefas relativas ao desenvolvimento do relatório.

TeamGantt Ferramenta online para planeamento temporal de projetos. Neste projeto, é

utilizada para produzir diagramas de Gantt.

Microsoft Teams Plataforma de comunicação e colaboração que oferece funcionalidades de chat, videoconferência e partilha de ficheiros. Foi a aplicação escolhida para comunicar com a orientadora da tese e restantes membros da The Loop Company, visto ser a aplicação utilizada atualmente na empresa.

Microsoft 365 Pack de aplicações online de escritório que permite criar e partilhar documentos com colegas de equipa.

ClickUp Ferramenta de gestão de projetos online. É a ferramenta utilizada pela The Loop Company para a gestão de projetos de software.

GitLab Gestor de repositórios de software, baseado em git, utilizado pela The Loop Company nos seus projetos.

Visual Studio Code Editor de código utilizado pelo autor neste projeto.

Capítulo 3

Estado da Arte

Neste capítulo é realizado um levantamento e análise do estado da arte. Visto que o objetivo deste projeto é desenvolver uma solução para substituir um sistema já existente, o primeiro item a ser analisado é a plataforma de gestão de salas utilizada atualmente pela Ticketline. Seguidamente, são avaliadas algumas ferramentas que facilitam a produção de mapas de salas de espetáculo para entender se podem ser utilizadas ou se é necessário desenvolver a plataforma a partir do zero. Finalmente, são analisados *Web Services* para entender qual é a melhor solução para a integração dos mapas das salas nas plataformas que necessitam.

3.1 Plataforma de Gestão de Salas da Ticketline

Nesta secção, são analisadas as funcionalidades que a plataforma de gestão de salas da Ticketline oferece e são identificadas as limitações e problemas que surgem da utilização desta ferramenta. Esta análise visa identificar as funcionalidades que o sistema atual oferece para um melhor entendimento relativamente ao contexto deste projeto. Adicionalmente, são analisadas as limitações e problemas deste software para que os mesmos possam ser evitados no design da nova solução.

3.1.1 Funcionalidades

Numa primeira análise, o software parece algo complexo e bastante antiquado. A sua estrutura está dividida em três páginas mas existem dois componentes que são comuns a todas as páginas da aplicação, sendo o primeiro uma listagem dos locais presentes na plataforma. Esta lista apresenta um formato de árvore em que os locais podem ser selecionados, para apresentar os detalhes do local, ou expandidos, para apresentar as salas do local selecionado. As salas, por sua vez, podem também ser selecionadas para apresentar a página de detalhes da sala ou expandidas para apresentar a lista de zonas que compõem a sala. É possível, posteriormente, selecionar uma zona para apresentar a página de detalhes da zona.

O segundo componente é uma barra que permite ao utilizador realizar cinco operações: criar uma nova zona, incluir uma zona numa sala, confirmar as alterações realizadas, excluir uma zona ou colocar a lista de salas no seu estado inicial. Na figura 3.1, podemos visualizar a página inicial da plataforma que é, também, a página de detalhes de um local.

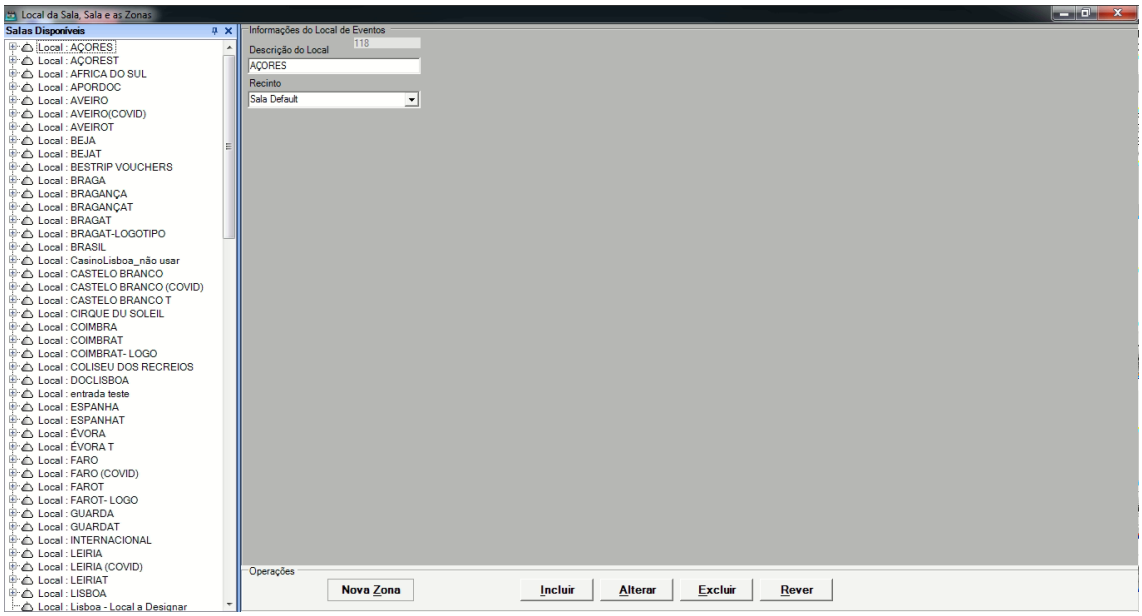


Figura 3.1: Software Ticketline - Página Inicial/Detalhes de Local

A página de detalhes de uma sala, na figura 3.2, permite editar as informações da sala como o nome da sala, morada, contactos e outras informações relevantes. Adicionalmente, é possível definir a imagem do mapa da sala que será apresentada no website. É também apresentado um esboço do mapa da sala com a representação das zonas através de retângulos.

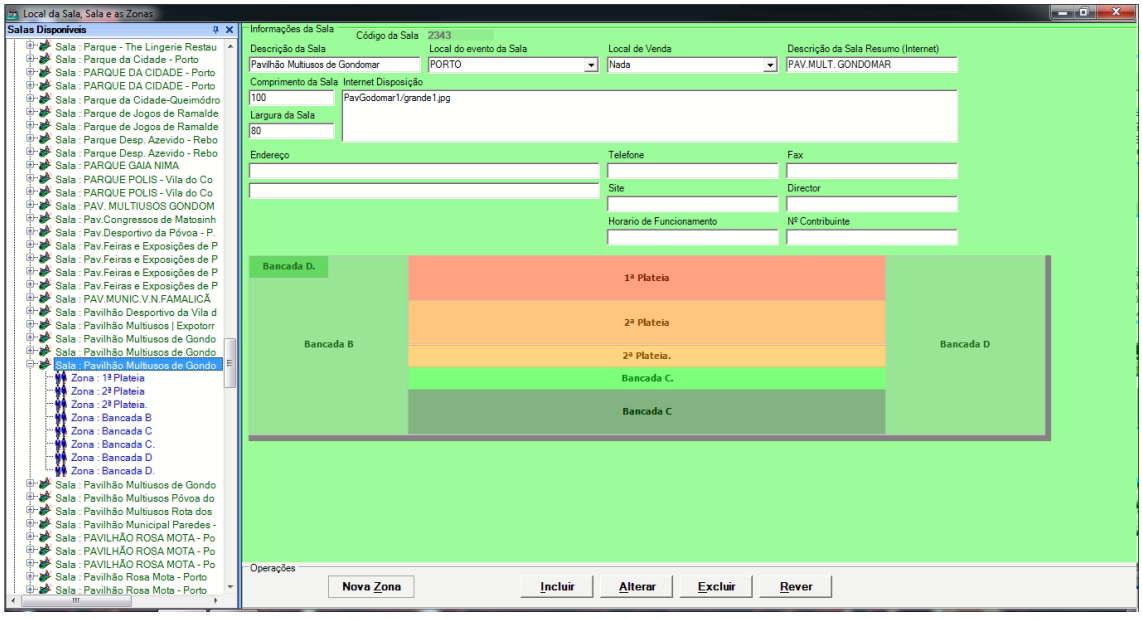


Figura 3.2: Software Ticketline - Página de Detalhes de Sala

Na página de detalhes da zona, figura 3.3, o utilizador pode definir o nome da zona, a cor da zona, a porta de entrada e o tipo de espetáculo. É também nesta página que é feito o mapeamento dos lugares da zona através de uma matriz que o utilizador pode configurar para que possa representar, de forma aproximada, a disposição dos lugares na sala. A plataforma oferece a possibilidade de definir o número de filas e colunas da matriz, anular células para representar obstáculos e o formato da zona e, adicionalmente, definir

corredores verticais ao indicar a coluna que antecede o corredor. É também possível definir a *label* do primeiro lugar e escolher entre uma série de algoritmos de numeração para que os restantes lugares sejam numerados de forma automática.

Nesta página é também definido um polígono, em HTML, que irá ser sobreposto à imagem fornecida nas configurações da sala. Desta forma, o cliente pode aceder ao mapa de lugares da zona ao clicar em cima da zona na imagem. É também definido um retângulo, e o seu tamanho e posição, que será apresentado no esboço presente na página de detalhes da sala.

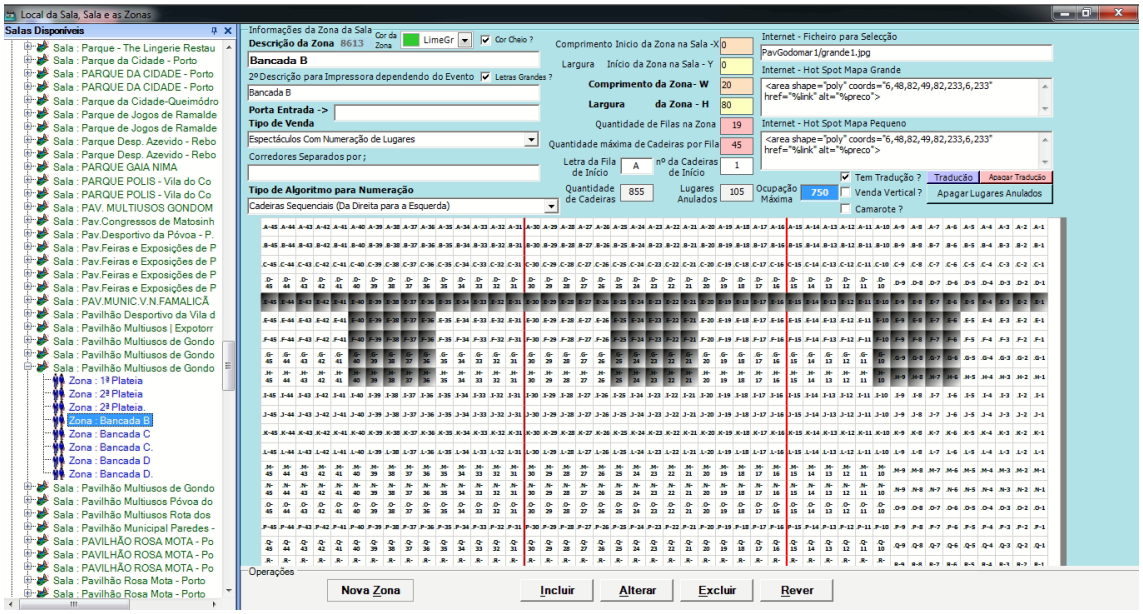
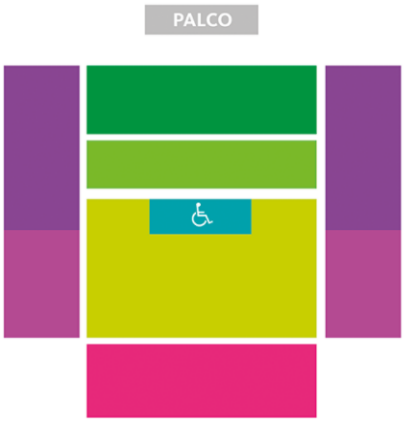


Figura 3.3: Software Ticketline - Página de Detalhes de Zona

Na figura 3.4, é possível visualizar o mapa da sala exportado para o website da Ticketline. Neste mapa, é possível visualizar a topologia da sala e clicar nas zonas do mapa para abrir o mapa individual da zona, caso tenha lugares marcados.



ZONA	DESCONTOS	VISIBILIDADE	MARCAÇÃO	PREÇO
Plateia Vip	-	-		35,00€
Bancada B	-	-		20,00€
Bancada B - Palco	-	-		25,00€
Bancada C	-	-		18,00€
Bancada D	-	-		20,00€
Bancada D - Palco	-	-		25,00€
Mobilidade Reduzida	-	-	não	15,00€
Plateia A	-	-		28,00€
Plateia De Pé	-	-	não	13,00€

Figura 3.4: Software Ticketline - Mapa da Sala

O mapa individual da zona, na figura 3.5, é possível identificar os lugares ocupados através de uma legenda de cores que facilita a interpretação da informação presente no mapa. Adicionalmente, é possível selecionar lugares para efetuar reservas.

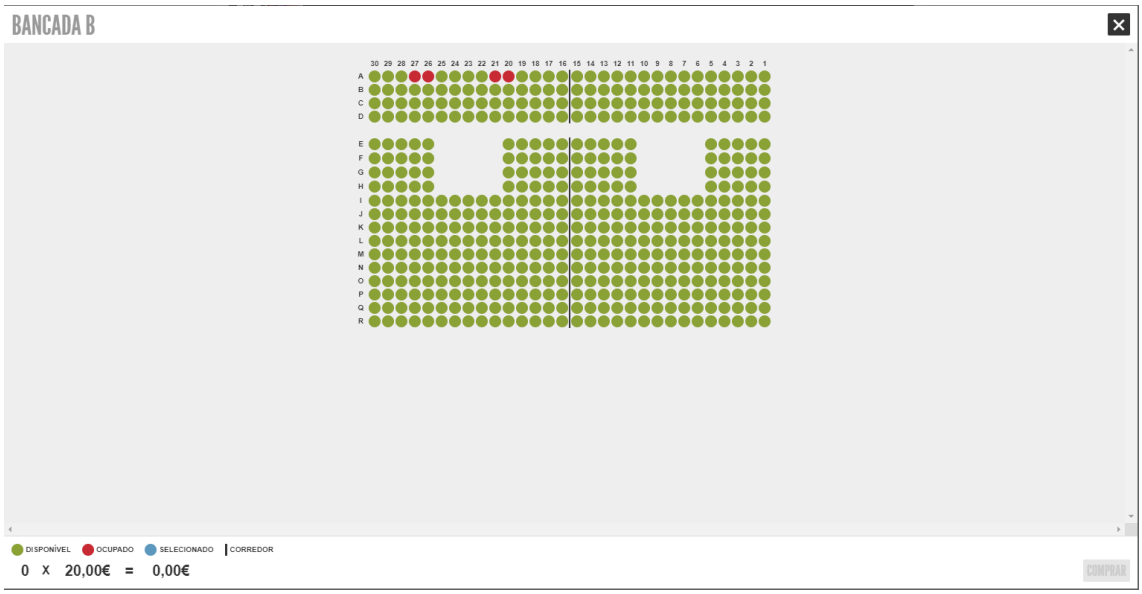


Figura 3.5: Software Ticketline - Mapa da Zona

3.1.2 Problemas e Limitações

Apesar do software cumprir o seu propósito, as tecnologias utilizadas para desenvolver têm várias limitações que, por sua vez, causam problemas na utilização deste software. Por essa razão, nesta subsecção serão identificados e analisados alguns problemas desta plataforma.

Interface

Em primeiro lugar, a interface é desatualizada e algo complexa. Apesar da estrutura do software ser muito simples, com apenas três páginas distintas, a informação presente nas páginas pode tornar-se confusa pois não existe grande organização. Em adição, muitos campos não apresentam uma linguagem suficientemente clara para se entender o seu propósito.

Representação de Lugares

Outro problema é a representação das zonas utilizar o formato de matriz, que não é suficientemente flexível para apresentar formatos de zona que não sejam retângulos. Por exemplo, na figura 3.6, está representada uma zona que tem uma forma curva que não é refletida no respetivo mapa da zona, presente na figura 3.7.

Pouca Automação

Alguns processos de criação de mapas poderiam ser automatizados para facilitar as tarefas dos gestores. Por exemplo, a necessidade de definir manualmente um polígono HTML para sobrepor uma imagem poderia ser eliminada utilizando tecnologias mais recentes. Outro problema, que está também relacionado com a forma de representação em matriz, é a necessidade de desativar lugares para representar obstáculos ou tentar aproximar a zona ao seu formato real. Este processo pode tornar-se exaustivo e seria evitado com outro tipo de representação mais apropriada.

Visualização dos Mapas

Finalmente, os mapas produzidos podem tornar-se confusos e, de certa forma, enganadores. Por exemplo, como é possível visualizar nas figuras 3.6 e 3.7, devido a não existirem

referências do palco na página da zona, o cliente pode ficar confuso e não conseguir avaliar bem o posicionamento de um lugar em relação ao palco.

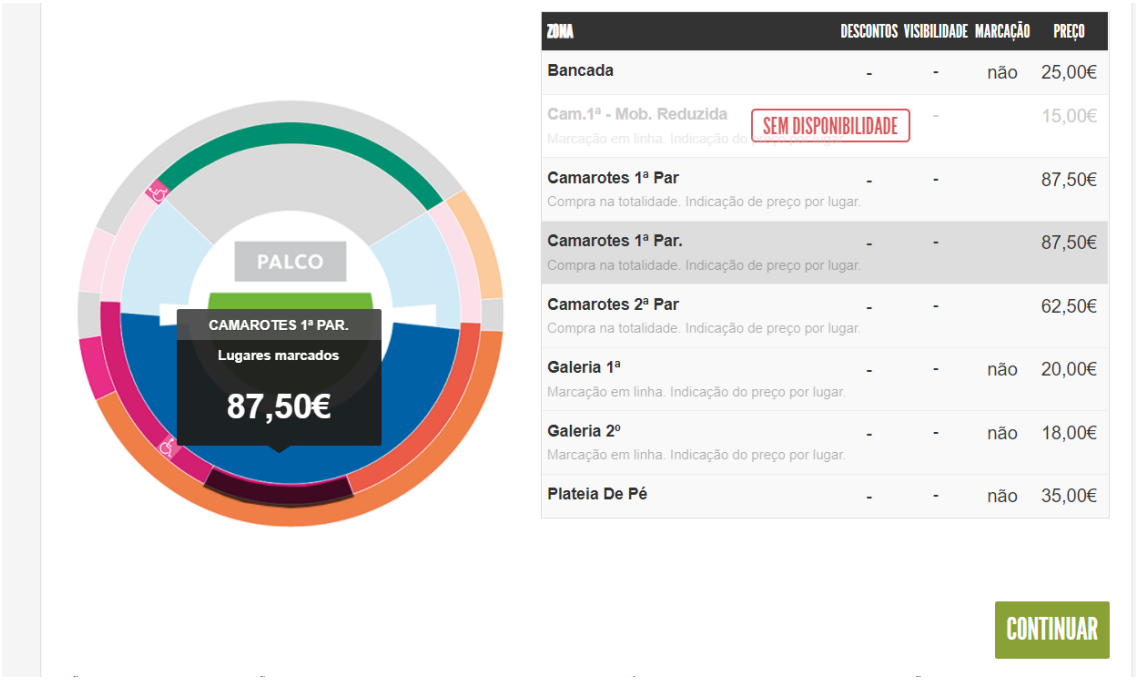


Figura 3.6: Software Ticketline - Zona Curvada

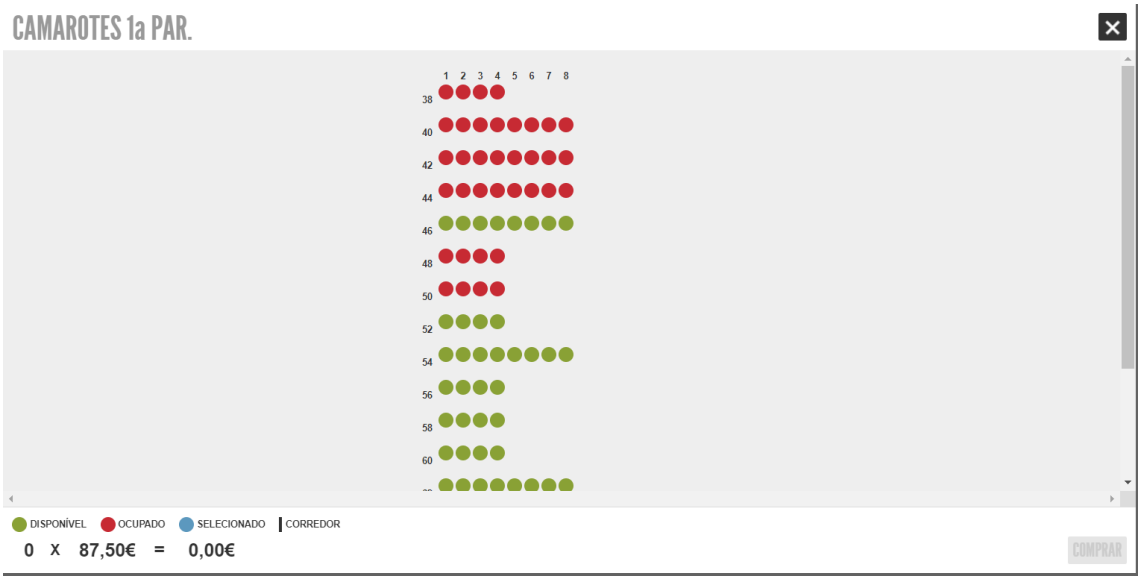


Figura 3.7: Software Ticketline - Mapa de Lugares da zona semicircular

3.2 Ferramentas de Criação de Mapas Interativos para Salas de Espetáculo

Esta secção será iniciada com uma breve explicação sobre o funcionamento de mapas interativos. De seguida, serão analisadas ferramentas que permitem criar mapas interativos para salas de espetáculo. No final, será realizada uma comparação para entender se alguma destas soluções poderá ser utilizada para facilitar o desenvolvimento deste projeto.

3.2.1 Mapas Interativos

Os mapas interativos são, habitualmente, uma representação da vista aérea da sala de espetáculos. Esta forma de representação permite aos clientes entender melhor a topologia da sala e, consequentemente, avaliar a qualidade dos lugares. Os mapas são interativos pois permitem ao utilizador clicar, fazer zoom e explorar o mapa através da utilização do seu rato ou smartphone. Habitualmente, estes mapas são constituídos por várias zonas que o utilizador pode selecionar ou ampliar para conseguir visualizar as filas de lugares[16].

O desenvolvimento de mapas interativos com uma interface apelativa e intuitiva é um desafio complexo. No entanto, existem várias ferramentas que permitem criar e renderizar estes mapas através dos seus editores. Posteriormente, estes mapas podem ser integrados em websites e ser utilizados por clientes para efetuar reservas de bilhetes.

3.2.2 Softjourn Venue Mapping Tool

A Softjourn[17] é uma empresa que presta serviços de tecnologia, especialmente nas áreas de tecnologia financeira, cartões e pagamentos, entretenimento e média e bilhética. No início de 2020, anunciaram a introdução de uma nova ferramenta[18] de criação de mapas de salas de espetáculo.

No seu website, a Softjourn indica que esta ferramenta possibilita a criação de mapas com um design intuitivo, sendo possível identificar facilmente zonas, obstáculos e portas. Informam também que estes componentes possuem um elevado nível de personalização pois permitem alterar a cor, forma e designação. A figura 3.8 mostra o editor de salas desta ferramenta.

Em termos de integração, é necessário hospedar a ferramenta no lado do cliente, algo que, segundo a Softjourn, diminui problemas de conectividade e aumenta a performance, fiabilidade e segurança.

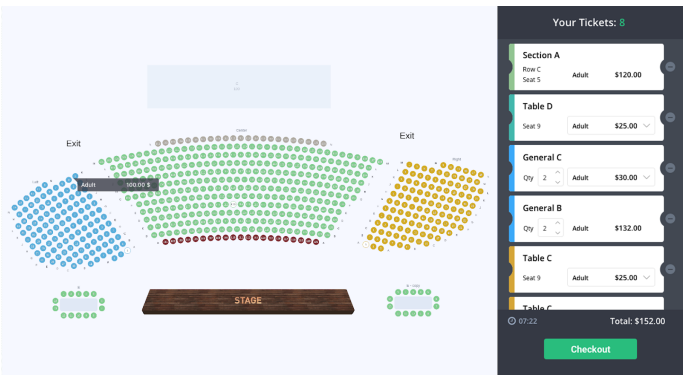


Figura 3.8: Softjourn Venue Mapping Tool - Editor de Mapas

3.2.3 Seatmap.pro

A Seatmap.pro[19] é uma ferramenta de criação de mapas de salas de espetáculo que está dividida em três partes: um editor de mapas, um sistema de reservas e um renderizador de mapas. O sistema de reservas não será analisado pois não faz parte do âmbito deste projeto.

No website, é indicado que o editor de mapas de salas, presente na figura 3.9, fornece um grande número de funcionalidades. Nesta ferramenta, o utilizador pode criar zonas ao definir o número de filas e colunas que compõem a zona. A partir desse ponto, é possível aplicar uma série de transformações como rodar, curvar e esticar. Através destas transformações é possível aproximar o mapa da sala à topologia da sala. Além da representação de uma secção, este software disponibiliza representações de mesas e polígonos aos quais se podem associar lugares.

A Seatmap.pro permite escolher entre realizar a hospedagem da ferramenta localmente ou aceder às funcionalidades através do seu website. Posteriormente, é possível integrar esta ferramenta com outros sistemas através da API da ferramenta. Para facilitar este processo, a empresa disponibiliza toda a documentação relevante no seu website.

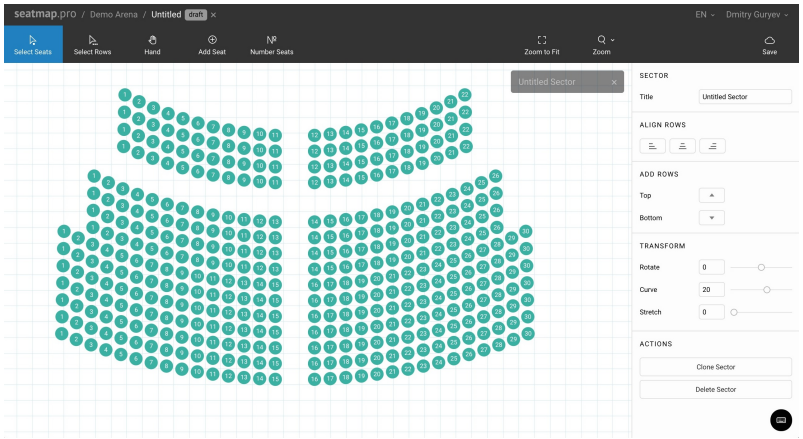


Figura 3.9: Seatmap.pro - Editor de Mapas

3.2.4 Seats.io

A ferramenta Seats.io[20] destaca-se pela facilidade de integração dos vários componentes, sendo apenas necessário incluir um *script* de JavaScript numa página web e implementar os métodos da sua API. A documentação com a descrição do processo de integração é disponibilizada na página da plataforma.

A ferramenta pode ser resumida em três grandes módulos: um editor de mapas, um gestor de eventos e um renderizador para apresentar o mapa ao cliente. O gestor de eventos não será analisado pois não faz parte do âmbito deste projeto.

O editor de salas, presente na figura 3.10, oferece aos seus utilizadores um elevado nível de personalização, sendo possível adicionar e manipular polígonos para criar zonas e obstáculos. Adicionalmente, existem representações de mesas e expositores que oferecem maior variedade no momento da criação do mapa. É também disponibilizada uma biblioteca dos mapas das salas mais populares que podem ser importados e que permitem evitar a tarefa de criação desses mapas.

Finalmente, o renderizador de mapas fornece ao cliente uma experiência imersiva e visualmente apelativa. O mapa da sala é apresentado na sua totalidade e o utilizador pode navegar e interagir com o mapa para obter mais informações sobre os lugares e efetuar reservas.

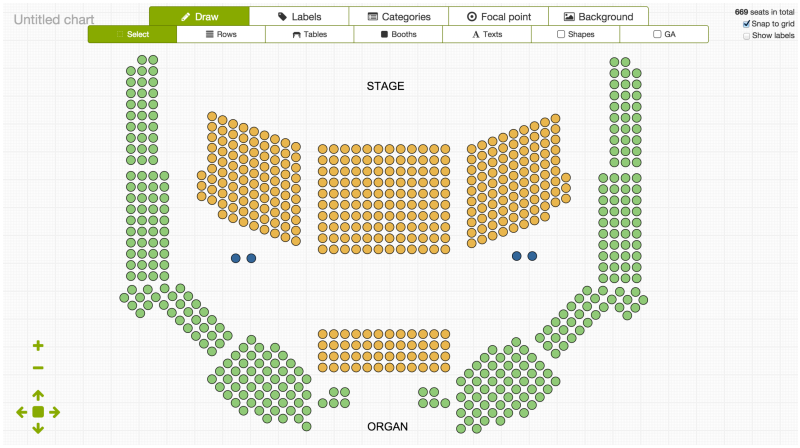


Figura 3.10: Seats.io - Editor de Mapas

3.2.5 Seatics Maps

A Seatics Maps[21] é uma solução de mapas interativos que pode ser integrada através de uma API. Esta ferramenta permite ao utilizador controlar a interface da plataforma de criação de salas. Atualmente existem duas opções: utilizar uma interface pré-construída que fornece várias opções de configuração ou construir uma interface de raiz. Uma visualização da interface pré-construída está presente na figura 3.11.

Segundo a Seatics Maps, os mapas produzidos por esta ferramenta destacam-se pelo design responsivo e preparado para mobile e também pela inclusão de fotografias da vista de cada zona.

A documentação desta ferramenta não se encontra disponível, sendo necessário pedir uma demonstração para obter mais informações.

Figura 3.11: Seatics - Editor de Mapas

3.2.6 Comparações e Decisão

Nesta subsecção é realizada uma comparação entre as ferramentas analisadas. Visto que a Seatics Maps não apresenta informação suficiente sobre as suas funcionalidades, não foi considerada como opção para este projeto e, consequentemente, não foi incluída na comparação.

Os critérios utilizados nesta comparação e os respetivos resultados podem ser consultados na tabela 4.1.

Tabela 3.1: Comparação de ferramentas de criação de mapas de salas

Ferramenta	Softjourn VMT	Seatmap.pro	Seats.io
Editor de Salas	Completo	Muito Completo	Muito Completo
Documentação	-	Muito Completa	Muito Completa

Editor de Salas Os editores de salas destas plataformas são muito poderosos mas, no entanto, aqueles que se destacam são o editor da ferramenta Seats.io[22] e Seatmap.pro[23] que suportam todos os elementos necessários a este projeto. O editor da Softjourn VMT[24] é completo e oferece quase tantas funcionalidades como os outros dois, mas, no entanto, não oferece outro tipo de representações que não sejam zonas de lugares. Por exemplo, a representação de mesas não é suportada nesta ferramenta e é um dos requisitos deste projeto.

Documentação Enquanto a documentação das APIs da Seats.io[25] e Seatmap.pro[26] são muito completas e estão disponíveis nos websites, a Softjourn VMT não disponibiliza qualquer tipo de documentação.

Apesar da Seatmap.pro e Seats.io serem muito completas, a decisão final foi não utilizar qualquer ferramenta e desenvolver um sistema de raiz. Estas ferramentas permitem desenvolver mapas interativos muito capazes e que satisfazem muitos requisitos deste projetos mas, no entanto, a sua falta de personalização e flexibilidade fazem com que alguns requisitos não sejam implementáveis. Por exemplo, o sistema atual da Ticketline suporta vários tipos de algoritmos para numeração de lugares que são utilizados nas salas portuguesas e que não são suportados por estas ferramentas. Esta informação é depois utilizada nos bilhetes para que os clientes possam identificar facilmente o seu lugar na sala. Outro exemplo seria o facto de não ser possível adicionar novas funcionalidades visto que o sistema estaria dependente de outra entidade.

Resumindo, a utilização destas ferramentas levaria à necessidade de desistir de algumas funcionalidades que são importantes para o cliente. Por essa razão, o desenvolvimento de um sistema ajustado às necessidades do cliente é a única solução, até ao momento, capaz de implementar todos os requisitos deste projeto.

3.3 *Web Services*

Os *Web Services* são, por definição, aplicações ou fontes de informação que estão acessíveis através de um protocolo web, HTTP ou HTTPS, e que permitem comunicar e trocar informação na internet. Ao contrário de aplicações web, os *Web Services* são desenhados

para comunicar com outras aplicações em vez de comunicar com utilizadores. Uma funcionalidade muito importante dos *Web Services* é a capacidade das aplicações comunicarem entre si, independentemente da linguagem em que foram escritas.

Existem vários tipos de *Web Services* mas os mais utilizados na construção de aplicações web são os *SOAP Web Services* e *RESTful Web Services*.

SOAP Web Services

SOAP, ou *Simple Object Access Protocol*, é um protocolo baseado em XML para aceder a *Web Services* via HTTP ou SMTP. Utiliza documentos WSDL como forma de distribuir o modelo de descrição de um *Web Service*, que indica como devem ser os pedidos e respostas a este serviço.

RESTful Web Services

REST, ou *Representational State Transfer*, é um estilo de arquitetura de software em que cada URL único representa um objeto. Os *RESTful Web Services* são leves, escaláveis e de fácil manutenção, sendo habitualmente utilizados para a criação de APIs para aplicações web. Utilizam HTTP e suportam diversos métodos como GET, POST, PUT ou DELETE, facilitando a construção de serviços orientados a operações CRUD.

Comparações e Decisão

Ao contrário dos *SOAP Web Services*, os *RESTful Web Services* não funcionam bem para sistemas que necessitem de transmitir a sua informação de forma segura e que necessitem de realizar operações *stateful* mas, no entanto, esse não é o caso deste projeto. A informação enviada para o *Widget* não é confidencial, pois consistirá apenas no mapa da sala que está disponível no website, e não há necessidade de realizar operações *stateful*, visto que a maioria dos pedidos será apenas para receber ou atualizar informação. Em adição, os *RESTful Web Services* são mais rápidos, eficientes e não estão limitados a um tipo de conteúdo das mensagens.

A decisão caiu então na utilização de *RESTful Web Services* para realizar a comunicação com o *Widget* presente nas outras plataformas.

Capítulo 4

Requisitos do Sistema

Neste capítulo são apresentados os requisitos que têm de ser cumpridos e as restrições que têm de ser respeitadas. Em primeiro lugar são identificados os requisitos funcionais deste sistema e é-lhes atribuída uma prioridade. De seguida, são criadas histórias de utilizador para descrever melhor o funcionamento do sistema do ponto de vista do utilizador. Posteriormente, são identificados os atributos de qualidade deste sistema e as restrições impostas à realização deste projeto.

4.1 Requisitos Funcionais

Nesta secção são listados os requisitos funcionais identificados após reunir com o cliente. Estes requisitos encontram-se priorizados relativamente à sua relevância para o projeto utilizando o método de priorização de requisitos MoSCoW[27], que define a prioridade dos requisitos da seguinte forma:

- *Must Have* (M) - requisitos que são absolutamente necessários para o sucesso do projeto.
- *Should Have* (S) - requisitos que não são vitais mas adicionam valor significativo.
- *Could Have* (C) - requisitos que não são necessários mas têm um pequeno impacto nos resultados do projeto.

Dashboard

Na tabela 4.1 estão apresentados os requisitos funcionais da aplicação web que será o *dashboard* da plataforma de gestão de salas.

Tabela 4.1: Requisitos Funcionais - Dashboard

Requisitos Funcionais		**Prioridade**
Autenticação	Autenticar Utilizadores	M
Utilizador	Visualizar perfil	S
Venues	Listar Venues	M
	Pesquisar Venues	M
Salas	Listar Salas associadas a uma Venue	M
	Pesquisar Salas associadas a uma Venue	C
Layouts	Adicionar/Editar Layouts	M
	Eliminar Layouts	M
	Duplicar Layouts	M
	Recuperar Layouts eliminados (até 30 dias)	S
	Associar Layouts a uma Venue/Sala	M
	Listar Layouts associados a uma Venue/Sala	M
	Listar Layouts recentemente modificados	S
	Listar Layouts eliminados	S
	Listar Layouts importados	S
	Pesquisar Layouts associados a uma Venue/Sala	M
	Pesquisar Layouts recentemente modificados	S
	Pesquisar Layouts eliminados	S
	Pesquisar Layouts importados	S
	Mostrar alterações feitas a um Layout	S

Widget de Visualização de Salas

Na tabela 5.7 são listados os requisitos referentes ao *widget* de visualização de salas que será integrado no website, POS e plataforma de gestão de eventos da Ticketline.

Tabela 4.2: Requisitos Funcionais - *Widget*

Requisitos Funcionais		**Prioridade**
Lugares	Selecionar lugares	M
	Apresentar informações sobre lugares (tags)	S
	Alterar a disponibilidade dos lugares em tempo real	M
	Clicar sobre uma zona, setor ou anel para expandir	M
Grupos de Lugares	Apresentar a disponibilidade dos lugares	M
	Apresentar zonas sem lugares marcados	M
	Fazer *zoom* sem perder a visão global da sala	M
	Apresentar a lista de zonas, setores ou anéis	S

Editor de Mapas

A tabela 5.6 mostra os requisitos do componente de edição de mapas de salas.

Tabela 4.3: Requisitos Funcionais - Editor

Requisitos Funcionais		**Prioridade**
Guardar	Guardar Alterações	M
	Guardar Alterações Automaticamente	S
Zonas, Setores e Bancadas	Adicionar/editar zonas, setores e bancadas	M
	Eliminar zonas, setores e bancadas	M
	Selecionar zonas, setores e bancadas	M
	Mover zonas, setores e bancadas	M
Lugares	Adicionar/Editar Lugares	M
	Eliminar lugares	M
	Selecionar Lugares	M
	Mover Lugares	M
	Atribuir números a lugares	M
	Atribuir *tags* a lugares (mobilidade ou visibilidade reduzida e VIP)	S
	Associar lugares a zonas, setores ou bancadas	M
Conjuntos de Lugares (Retângulos, Semicírculos, Anéis e Mesas)	Adicionar/Editar Conjuntos de Lugares	M
	Eliminar Conjuntos de Lugares	M
	Mover Conjuntos de Lugares	M
	Atribuir *labels*/números a filas e lugares de acordo com algoritmos	S
	Atribuir *tags* a Conjuntos de Lugares	S
	Curvar, esticar ou rodar Conjuntos de Lugares	S
Formas Geométricas	Adicionar/Editar formas geométricas	S
	Eliminar formas geométricas	S
	Mover formas geométricas	S
	Selecionar formas geométricas	S
	Adicionar tipos a formas geométricas (Palco, Obstáculo, Bar, WC, Portas, Outro)	S
	Associar formas geométricas a zonas, setores ou bancadas	S
Outros	Bloquear edições simultâneas a um *layout*	S
	Replicar alterações feitas no Layout base noutros Layouts	C
	Importar uma imagem para o fundo da sala	S
	Importar um ficheiro CSV para gerar o rascunho da sala	C

4.2 Histórias de Utilizador

As histórias de utilizador[28] são explicações gerais e informais de um requisito de software através da perspetiva do utilizador. Habitualmente, as histórias de utilizador são utilizadas em metodologias ágeis mas, no entanto, também podem ser utilizadas noutras metodologias como forma de completar os requisitos identificados.

As histórias de utilizador são expressas numa frase simples e têm, habitualmente, a seguinte estrutura[28]:

Como [tipo de utilizador], **quero poder** [intenção] **para que** [razão].

A estrutura de uma História de Utilizador foca-se em três partes principais:

Tipo de utilizador: O cargo do utilizador que quer realizar uma ação.

Intenção: A ação que o utilizador quer realizar.

Razão: O objetivo do utilizador ao realizar a ação.

4.2.1 Utilizador

1. Autenticação
   (a) **Como** gestor **quero** poder fazer login no sistema **para que** possa autenticar-me na plataforma.
   (b) **Como** gestor **quero** poder fazer logout no sistema **para que** possa remover a minha autenticação na plataforma.
2. Perfil
   (a) **Como** gestor **quero** poder visualizar o meu perfil **para que** possa ver as minhas informações pessoais.
   (b) **Como** gestor **quero** poder editar as minhas informações pessoais (nome, contactos, email, password) **para que** as possa manter atualizadas.

4.2.2 Painel de Gestão/*Dashboard*

1. Locais
   (a) **Como** gestor **quero** poder pesquisar por um local pelo nome ou id **para que** possa encontrar rapidamente um local que eu já conheço.
   (b) **Como** gestor **quero** poder selecionar um local **para que** possa ver as informações e salas associadas.
   (c) **Como** gestor **quero** poder filtrar locais por distrito **para que** possa encontrar os locais de um distrito específico mais rapidamente.
   (d) **Como** gestor **quero** poder ordenar os locais pelo nome, de forma alfabética e inversa, **para que** possa encontrar mais rapidamente um local que eu já conheço.

2. Salas

   (a) **Como** gestor **quero** poder pesquisar uma sala pelo nome e id **para que** possa encontrar rapidamente uma sala que eu já conheço.

   (b) **Como** gestor **quero** poder selecionar uma sala **para que** possa ver as informações e *layouts* associados.

   (c) **Como** gestor **quero** poder ordenar as salas pelo nome, de forma alfabética e inversa, **para que** possa encontrar mais rapidamente uma sala que eu já conheço.

   (d) **Como** gestor **quero** poder associar *layouts* a uma sala **para que** possam ser listados.

3. *Layouts*

   (a) **Como** gestor **quero** poder pesquisar por um *layout* pelo nome, id ou sala/local, **para que** possa encontrar rapidamente um *layout* que já conheço.

   (b) **Como** gestor **quero** poder filtrar os *layouts* por estado (publicado/rascunho) **para que** possa encontrar rapidamente um *layout* cujo estado já conheço.

   (c) **Como** gestor **quero** poder ordenar os *layouts* por nome e data de modificação **para que** possa encontrar o *layout* que procuro mais facilmente.

   (d) **Como** gestor **quero** poder selecionar um *layout* **para que** possa ver as informações e o mapa do *layout*.

   (e) **Como** gestor **quero** poder adicionar um *layout* a uma sala **para que** possa ser listado.

   (f) **Como** gestor **quero** poder adicionar ou editar informações de um *layout*, tais como:

      i. O nome **para que** os gestores possam diferenciar o *layout*.

      ii. A capacidade máxima **para que** se possa definir o número máximo de bilhetes do evento.

      iii. O estado **para que** possa definir se o *layout* está pronto para ser utilizado num evento.

   (g) **Como** gestor **quero** poder definir um *layout* como *layout* base de uma sala **para que** este seja utilizado na construção de novos *layouts*.

   (h) **Como** gestor **quero** poder visualizar o mapa do *layout* **para que** possa distinguir visualmente os *layouts*.

   (i) **Como** gestor **quero** poder selecionar os *layouts* afetados por uma edição ao *layout* base **para que** possa reproduzir as alterações feitas no *layout* base sem ter de editar outros *layouts*.

   (j) **Como** gestor **quero** poder duplicar um *layout* **para que** possa utilizar um *layout* já criado.

   (k) **Como** gestor **quero** poder ver um histórico de modificações **para que** possa analisar quando um *layout* foi editado e por quem.

   (l) **Como** gestor **quero** poder eliminar um *layout* **para que** deixe de ser listado na sala e passe a ser listado numa lixeira.

   (m) **Como** gestor **quero** poder visualizar os eventos associados a um *layout* **para que** possa saber que eventos estão a utilizar ou utilizaram este *layout*.

4. Lixeira de *Layouts*
   (a) **Como** gestor **quero** poder pesquisar por um *layout* que tenha sido apagado no espaço de 30 dias **para que** possa encontrar um *layout* apagado que já conheço.
   (b) **Como** gestor **quero** poder selecionar um *layout* que tenha sido apagado **para que** possa ver as suas informações e mapa.
   (c) **Como** gestor **quero** poder filtrar os *layouts*, por estado ou data de modificação, que tenham sido apagados num espaço de 30 dias **para que** os possa encontrar mais rapidamente.
   (d) **Como** gestor **quero** poder recuperar um *layout* num espaço de 30 dias **para que** volte a ser listado.
5. *Layouts* Recentemente Modificados
   (a) **Como** gestor **quero** poder selecionar um *layout* recentemente modificado **para que** possa ver as suas informações e mapa.
   (b) **Como** gestor **quero** poder pesquisar por um *layout* recentemente modificado pelo nome e id, **para que** os posso encontrar rapidamente.
   (c) **Como** gestor **quero** poder filtrar os *layouts* que tenham sido editados recentemente **para que** os possa encontrar rapidamente.

4.2.3 Editor

1. Geral
   (a) **Como** gestor **quero** poder editar o mapa da sala **para que** possa alterar a disposição dos vários componentes, tais como:
      i. Lugares;
      ii. Obstáculos;
      iii. Portas;
      iv. Palcos;
      v. Zonas/Setores/Anéis;
      vi. WCs.
   (b) **Como** gestor **quero** poder importar uma imagem **para que** possa ser utilizada como fundo do mapa.
   (c) **Como** gestor **quero** poder importar um ficheiro csv **para que** possa gerar um rascunho do mapa da sala.
2. Lugares
   (a) **Como** gestor **quero** poder adicionar lugares a uma posição especifica **para que** possam aparecer no mapa.
   (b) **Como** gestor **quero** poder adicionar e editar informações de um lugar, tais como:
      i. A *label* **para que** o cliente possa saber o número do lugar.
      ii. As *tags* **para que** o cliente possa saber se o lugar é de mobilidade reduzida, visibilidade reduzida ou VIP.
      iii. O estado **para que** o cliente possa saber se o lugar pode ser reservado.
      iv. O formato **para que** o cliente possa distinguir os tipos de lugares.

(c) **Como** gestor **quero** poder mover lugares **para que** possa definir a sua posição no mapa.

(d) **Como** gestor **quero** poder eliminar lugares **para que** deixem de aparecer no mapa.

(e) **Como** gestor **quero** poder escolher um algoritmo de *labeling* **para que** possa atribuir *labels* a vários lugares. Esses algoritmos são:

i. Sequencial com a numeração a iniciar-se da esquerda para a direita.
ii. Sequencial com a numeração a iniciar-se da direita para a esquerda.
iii. Só números pares com a numeração a iniciar-se da esquerda para a direita.
iv. Só números pares com a numeração a iniciar-se da direta para a esquerda.
v. Só números ímpares com a numeração a iniciar-se da esquerda para a direita.
vi. Só números ímpares com a numeração a iniciar-se da direta para a esquerda.
vii. Números pares no lado esquerdo e ímpares no lado direito com os números maiores na parte exterior e menores na parte interior.
viii. Números pares no lado esquerdo e ímpares no lado direito com os números menores na parte exterior e maiores na parte interior.
ix. Números pares no lado direito e ímpares no lado esquerdo com os números maiores na parte exterior e menores na parte interior.
x. Pares no lado direito e ímpares no lado esquerdo com os números menores na parte exterior e maiores na parte interior.

(f) **Como** gestor **quero** poder alterar o alinhamento dos lugares (esquerda, centro, direita) **para que** o cliente possa entender o formato da plateia.

3. Zonas/Setores/Anéis

(a) **Como** gestor **quero** poder adicionar zonas, setores e anéis **para que** possam aparecer no mapa.

(b) **Como** gestor **quero** poder adicionar e editar informações de uma zonas, setores ou anéis, tais como:

i. O nome **para que** o cliente possa diferenciar as zonas.
ii. O nome abreviado **para que** possa definir o nome que aparece no bilhete do cliente.
iii. A porta de entrada/saída **para que** o cliente possa saber a porta pela qual deve entrar e sair da sala.
iv. A cor **para que** possa distinguir as várias zonas, setores e anéis visualmente.
v. O tipo **para que** o cliente possa saber se existem lugares marcados naquela zona, setor ou anel.
vi. Os lugares, caso existam lugares marcados, **para que** se possa saber quais são os lugares que pertencem à zona, setor ou anel.

(c) **Como** gestor **quero** poder eliminar uma zona, setor e anel **para que** deixe de aparecer listada/o.

(d) **Como** gestor **quero** poder listar as zonas, setores e anéis **para que** os possa selecionar individualmente.

(e) **Como** gestor **quero** poder adicionar zonas a um setor para que possam aparecer numa camada inferior ao setor no mapa da sala.

(f) **Como** gestor **quero** poder fazer transformações a uma zona, setor ou anel, tais como:

i. Rodar **para que** possa alterar a orientação da zona, setor ou anel no mapa.
ii. Distorcer **para que** possa alterar o formato da zona, setor ou anel.
iii. Curvar **para que** possa alterar o formato da zona, setor ou anel.

4. Mesas
   (a) **Como** gestor **quero** poder adicionar mesas **para que** possam aparecer no mapa.
   (b) **Como** gestor **quero** poder adicionar e editar informações a uma mesa, tais como:
      i. O nome **para que** possa diferenciar as mesas.
      ii. Os lugares **para que** se possa perceber quais são os lugares pertencentes à mesa.
      iii. O tipo **para que** o cliente possa saber se os lugares podem ser comprados individualmente ou se têm de ser comprados na totalidade.
   (c) **Como** gestor **quero** poder rodar uma mesa **para que** alterar a orientação da mesa no mapa.

5. Formas Geométricas
   (a) **Como** gestor **quero** poder adicionar formas geométricas **para que** possam aparecer no mapa.
   (b) **Como** gestor **quero** poder adicionar e editar informações de uma forma geométrica, tais como:
      i. A posição **para que** o cliente possa visualizar a sua localização no mapa.
      ii. O tipo (porta, WC, obstáculo, zonas sem lugares marcados) **para que** o cliente possa saber o que representa a forma geométrica no mapa.
      iii. O formato **para que** o cliente possa associar esse a forma geométrica ao objeto que representa.
   (c) **Como** gestor **quero** poder eliminar uma forma geométrica **para que** deixe de aparecer no mapa.
   (d) **Como** gestor **quero** poder fazer transformações a uma forma geométrica, tais como:
      i. Rodar **para que** possa alterar a orientação da forma geométrica no mapa.
      ii. Distorcer **para que** possa alterar o formato da forma geométrica.
      iii. Curvar **para que** possa alterar o formato da forma geométrica.

4.2.4 *Widget*

1. Utilizador
   (a) **Como** utilizador **quero** poder visualizar o mapa completo da sala **para que** possa entender topologia da sala.
   (b) **Como** utilizador **quero** poder fazer zoom no mapa da sala **para que** possa navegar facilmente pelo mapa da sala.
   (c) **Como** utilizador **quero** poder clicar, ou fazer zoom, sobre uma zona/setor/anel **para que** possa ver a camada abaixo dessa zona/setor/anel.
   (d) **Como** utilizador **quero** poder ver a disponibilidade dos lugares em tempo real **para que** possa saber a ocupação da sala.

(e) **Como** utilizador **quero** poder selecionar lugares **para que** possa escolher os lugares sobre os quais desejo realizar uma operação.

(f) **Como** utilizador **quero** poder ver informações sobre os lugares, tais como:

i. O preço de cada lugar **para que** possa tomar uma decisão informada.

ii. As *tags* **para que** possa saber se o lugar é de visibilidade reduzida, mobilidade reduzida ou VIP.

2. Cliente

(a) **Como** cliente **quero** poder ver as zonas/setores/anéis numa lista **para que** possa selecionar uma zona/setor/anel sem ter de procurar no mapa.

(b) **Como** cliente **quero** poder selecionar uma zona/setor/anel na lista **para que** possa ver a camada abaixo dessa zona/setor/anel.

4.3 Atributos de Qualidade

Os atributos de qualidade, ou requisitos não-funcionais, são propriedades testáveis e mensuráveis de um sistema que são utilizadas para avaliar se um sistema satisfaz as necessidades dos seus *stakeholders*. Nesta secção, são descritos os atributos de qualidade deste sistema através da criação de cenários para cada atributo.

4.3.1 Interoperabilidade

Interoperabilidade é a habilidade de sistemas de software diferentes comunicarem e operarem entre si. Neste projeto, a interoperabilidade é um atributo de qualidade essencial, visto que o objetivo principal desta plataforma é a criação de *layouts* que podem ser integrados noutras plataformas.

Cenário: Um sistema com o *widget* integrado faz um pedido para obter um mapa de uma sala.

Tabela 4.4: Atributo de Qualidade - Interoperabilidade

Origem do Estímulo	Sistema com o *widget* integrado
Estímulo	Sistema faz um pedido para obter um mapa de uma sala
Ambiente	Funcionamento normal
Artefacto	Servidor
Resposta	O sistema responde ao pedido com o mapa da sala.

4.3.2 Segurança

A segurança é um atributo de qualidade importante para esta plataforma devido à necessidade de evitar que existam manipulações de dados indesejadas que possam comprometer o bom funcionamento do sistema.

Cenário: O utilizador tenta realizar um pedido, como aceder à página de detalhes de um layout, sem estar autenticado na plataforma.

Tabela 4.5: Atributo de Qualidade - Segurança

Origem do Estímulo	Utilizador não autenticado
Estímulo	O utilizador tenta aceder à página de detalhes de um layout
Ambiente	Funcionamento Normal
Artefacto	Servidor
Resposta	O sistema verifica se o utilizador está autenticado e redireciona para a página de *login*

4.3.3 Usabilidade

A usabilidade é uma medida de quão bem um utilizador consegue realizar uma tarefa para atingir um determinado objetivo. É uma qualidade essencial que qualquer aplicação web deve ter para manter os seus utilizadores satisfeitos. Estudos[29] indicam que algumas métricas que podem ser utilizadas para medir a usabilidade de um sistema consistem na percentagem de sucesso na realização das tarefas, tempo de realização da tarefa, percentagem de erro e satisfação dos utilizadores.

Cenário: Utilizador a desenhar um mapa para uma sala através do editor.

Tabela 4.6: Atributo de Qualidade - Usabilidade

Origem do Estímulo	Utilizador
Estímulo	O utilizador a desenhar um mapa para a sala
Ambiente	Funcionamento Normal
Artefacto	Editor de Mapas
Resposta	O utilizador consegue desenhar o mapa da sala
Métricas da Resposta	O utilizador deve ter uma percentagem de sucesso igual ou superior a 50% na realização das tarefas necessárias para construir um mapa para uma sala

4.3.4 Performance

A performance é um atributo essencial para qualquer aplicação web nos dias de hoje, visto que os utilizadores não estão dispostos a esperar por respostas às suas ações no website. Por essa razão, é importante que o sistema seja capaz de responder passado um tempo apropriado. Alguns estudos[30] revelam que 0.1s é o tempo máximo para que os utilizadores considerem que as suas ações estão a ser executadas instantaneamente. Adicionalmente, um tempo de resposta de 1s fá-los aperceberem-se do atraso na resposta mas mantém a sensação que estão a navegar livremente. Finalmente, um atraso de 10s de resposta faz com que os utilizadores desistam da tarefa que estavam a tentar executar.

Cenário: O sistema recebe um pedido, proveniente de um *widget*, para expandir um grupo de lugares e responde dentro de um limite de tempo adequado.

Tabela 4.7: Atributo de Qualidade - Performance

Origem do Estímulo	*Widget*
Estímulo	*Widget* envia pedido HTTP
Ambiente	Funcionamento Normal
Artefacto	Servidor
Resposta	O sistema responde ao pedido com sucesso
Métricas da Resposta	O tempo de resposta não excede 1 segundo

4.4 Restrições

Nesta secção, são identificadas as restrições que devem ser respeitadas durante o desenvolvimento deste projeto.

Tabela 4.8: Restrição Técnica RT01: Ruby on Rails

ID	RT01
Título	Ruby on Rails
Origem	The Loop Company
Ambiente	Desenvolvimento de Software
Objetivo	Facilitar a manutenção e inclusão de novas funcionalidades no software
Descrição	Nos projetos desenvolvidos pela The Loop Company é utilizada a *framework* Ruby on Rails. Por essa razão, e para facilitar a inclusão de novas funcionalidades no futuro, a sua utilização é considerada obrigatória.

Tabela 4.9: Restrição Técnica RT02: PostgreSQL

ID	RT02
Título	PostgreSQL
Origem	The Loop Company
Ambiente	Desenvolvimento de Software
Objetivo	Facilitar a manutenção do software
Descrição	O PostgreSQL é o sistema de gestão de bases de dados utilizado pela The Loop Company nos seus projetos. Por essa razão, e para facilitar a manutenção do sistema, a sua utilização é considerada obrigatória neste projeto.

Capítulo 5

Descrição do Sistema

Neste capítulo é descrito o sistema a implementar através da criação de diagramas de arquitetura que representam módulos e as conexões entre eles, e entidades e respetivas relações. Adicionalmente, é apresentado o esquema e organização das páginas, através de *wireframes*, e o fluxo da aplicação, num diagrama de navegação.

5.1 Diagrama Entidade-Relacionamento

Os diagramas Entidade-Relacionamento permitem visualizar a organização da informação e as conexões entre as várias entidades que compõem o sistema. Na figura 5.1, é apresentado o diagrama produzido para este projeto que pode ser visualizado em maior escala na secção dos apêndices.

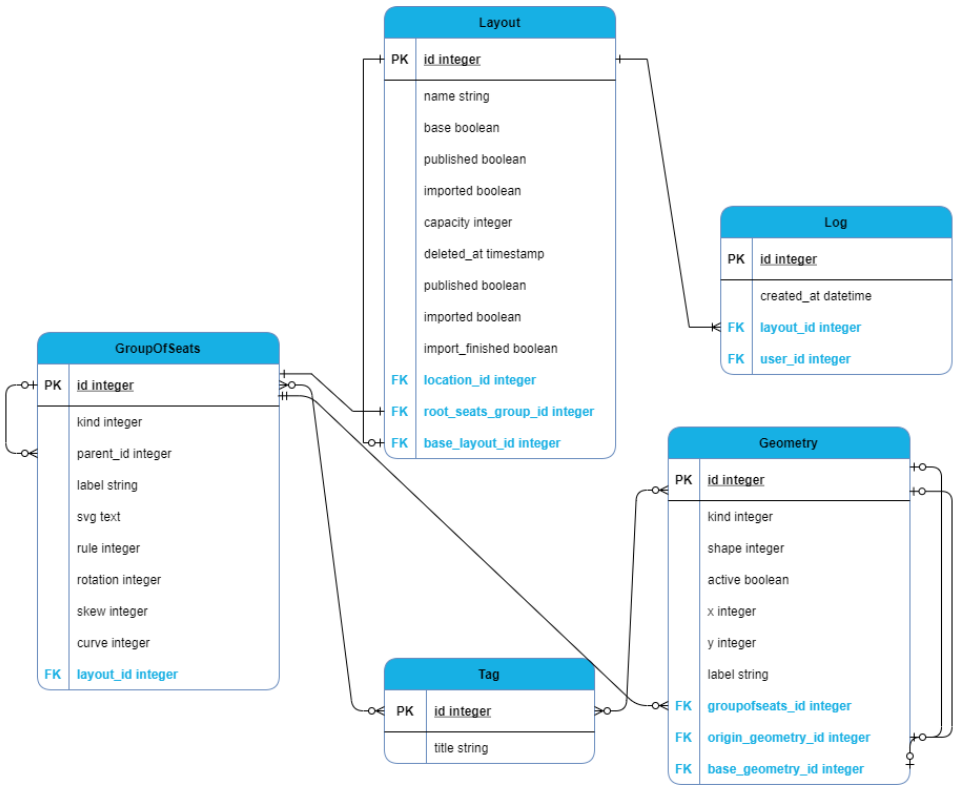


Figura 5.1: Diagrama Entidade-Relacionamento

As entidades presentes neste diagrama podem ser brevemente descritas da seguinte forma:

Layout

A entidade *Layout* guarda as informações relativas a um mapa de uma sala. Esta entidade tem as seguintes ligações:

- Um *layout* tem um ou mais grupos de lugares. (Ligação 1-para-N com a entidade *GroupOfSeats*)
- Um *layout* tem um ou mais registos de alteração. (Ligação 1-para-N com a entidade *Log*)

GroupOfSeats

A representação *GroupOfSeats* foi baseada num artigo[3] presente no website da ferramenta Seatmap.pro.

A entidade *GroupOfSeats* representa um grupo de lugares e descreve a organização do mapa recorrendo a uma representação em árvore. Todas as salas têm pelo menos um grupo de lugares que representa a raiz da sala. Esse grupo de lugares pode depois ser decomposto em outros grupos de lugares ou numa lista de lugares, representados pela entidade *Geometry*. As folhas desta árvore são sempre lugares ou grupos de lugares que não têm lugares marcados.

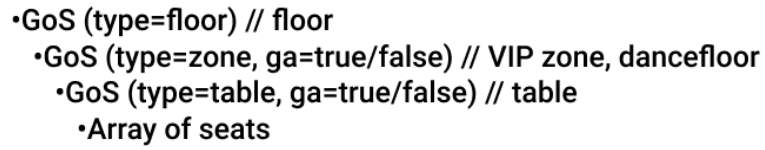


Figura 5.2: Representação de uma sala utilizando a entidade *GroupOfSeats*[3]

A grande vantagem de armazenar dados desta forma é a facilidade de integração de novas estruturas de organização da sala. A figura 5.2, mostra um exemplo de como pode ser representada uma sala que contenha zonas com mesas.

A entidade *GroupOfSeats* tem as seguintes ligações:

- Um *GroupOfSeats* pode ter um grupo de lugares pai. (Ligação 0-para-N com a entidade *GroupOfSeats*)
- Um *GroupOfSeats* pode ter zero ou mais rótulos. (Ligação N-para-N com a entidade *Tag*)
- Um *GroupOfSeats* pode ter zero ou mais lugares ou portas. (Ligação 0-para-N com a entidade *Geometry*)

Geometry

A entidade *Geometry* representa uma geometria que pode ser um lugar ou porta. Esta entidade tem as seguintes ligações:

- Uma geometria pode ter zero ou mais rótulos. (Ligação N-para-N com a entidade *Tag*)

- Uma geometria, do tipo lugar, pode ter uma indicação da geometria de origem caso tenha sido duplicada. (Ligação 0-para-1 com a entidade *Geometry*)
- Uma geometria pode ter uma indicação da geometria base caso tenha sido criada a partir do *layout* base. (Ligação 0-para-1 com a entidade *Geometry*)

Tag

A entidade *Tag* representa os rótulos como mobilidade reduzida, visibilidade reduzida, VIP, e outros rótulos que poderão ser úteis.

Log

A entidade *Log* representa os registos das alterações feitas a um *layout*.

5.2 Arquitetura do Sistema

Nesta secção é apresentada a arquitetura de sistema deste projeto, seguindo o modelo C4 para visualização de arquiteturas de software[31]. A arquitetura segue o padrão MVC, ou Modelo-Vista-Controlador, visto que o sistema será desenvolvido em Ruby on Rails.

Todos os diagramas de arquitetura do sistema podem ser consultados em maior escala na secção dos apêndices.

O diagrama de contexto do sistema, na figura 5.3, apresenta uma visão do panorama do sistema com pouco detalhe. Nesta visão é apresentada a plataforma de gestão de salas e as entidades ou sistemas que comunicam com este software.

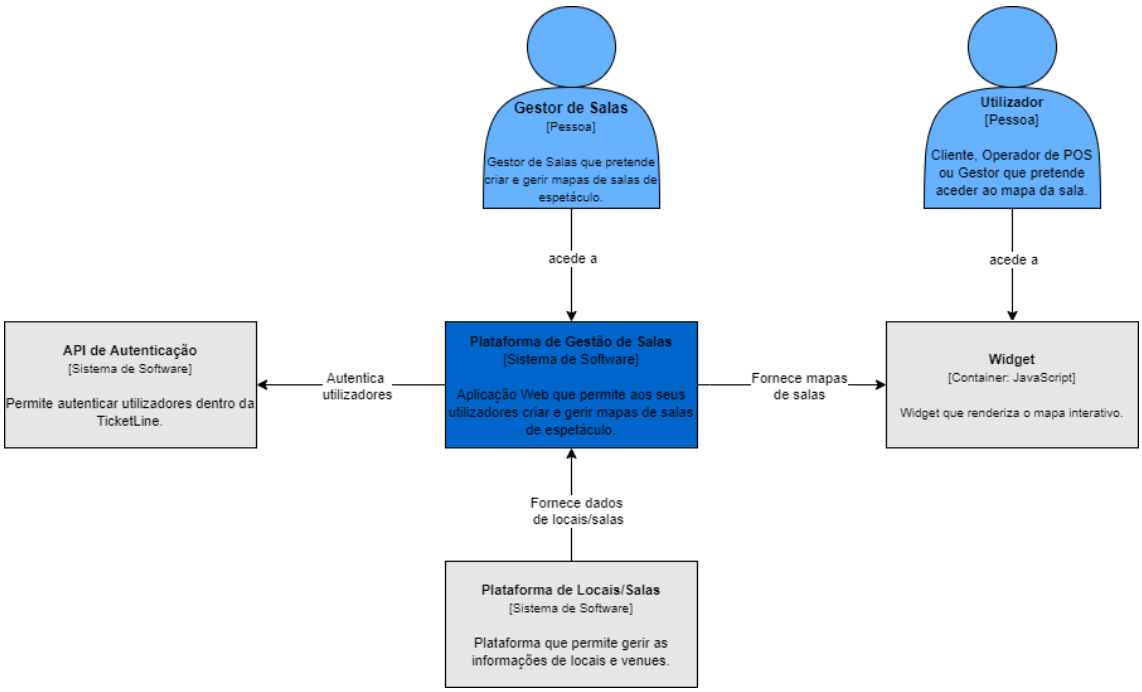


Figura 5.3: Diagrama de Contexto do Sistema

Os sistemas de software que interagem com a plataforma de gestão de salas são:

- API de Autenticação, para autenticar os utilizadores da plataforma de gestão de salas. É utilizada pelas plataformas internas da TicketLine e permite que os utilizadores possam utilizar as mesmas credenciais em todos estes sistemas.
- API de Locais/Salas, para obter as informações relativas a Locais e Salas que são necessárias para a plataforma de gestão de salas.
- *Widget*, para representar visualmente o mapa das salas de espetáculo em outras plataformas.

O diagrama de *containers* do sistema, na figura 5.4, mostra uma decomposição da plataforma de gestão de salas nos seus blocos principais. Aqui é possível visualizar as três camadas do padrão Modelo-Vista-Controlador, as conexões entre elas e entre as restantes entidades e sistemas de software.

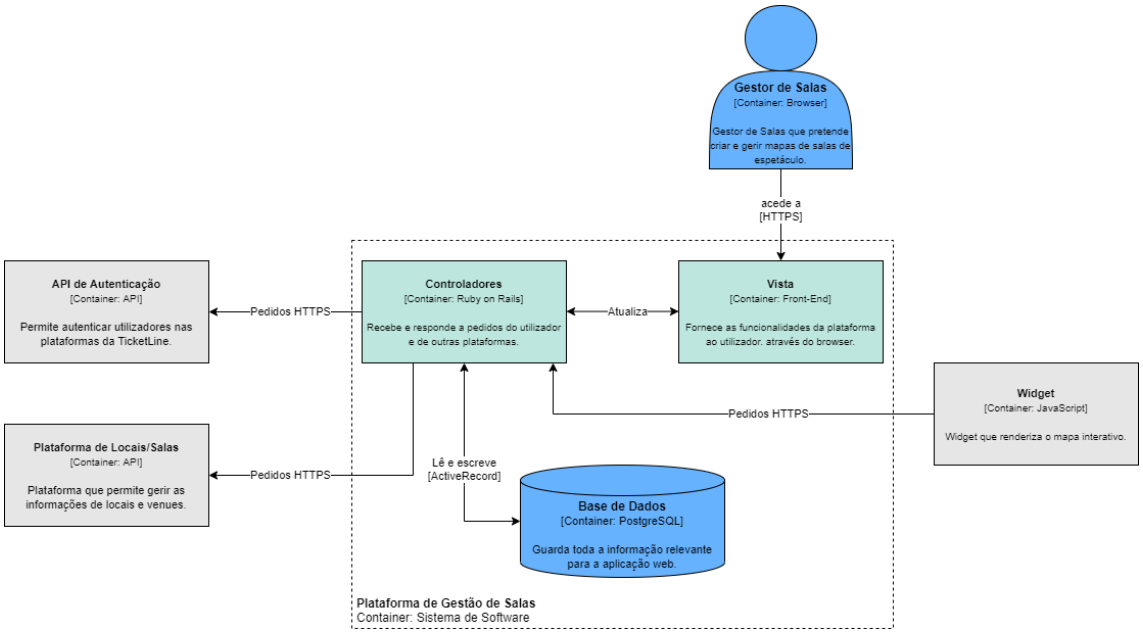


Figura 5.4: Diagrama de *Containers* do Sistema

O diagrama de componentes do sistema, na figura 5.5, apresenta a decomposição do *container* dos controladores em componentes mais pequenos e descreve, mais detalhadamente, a divisão das responsabilidades entre eles.

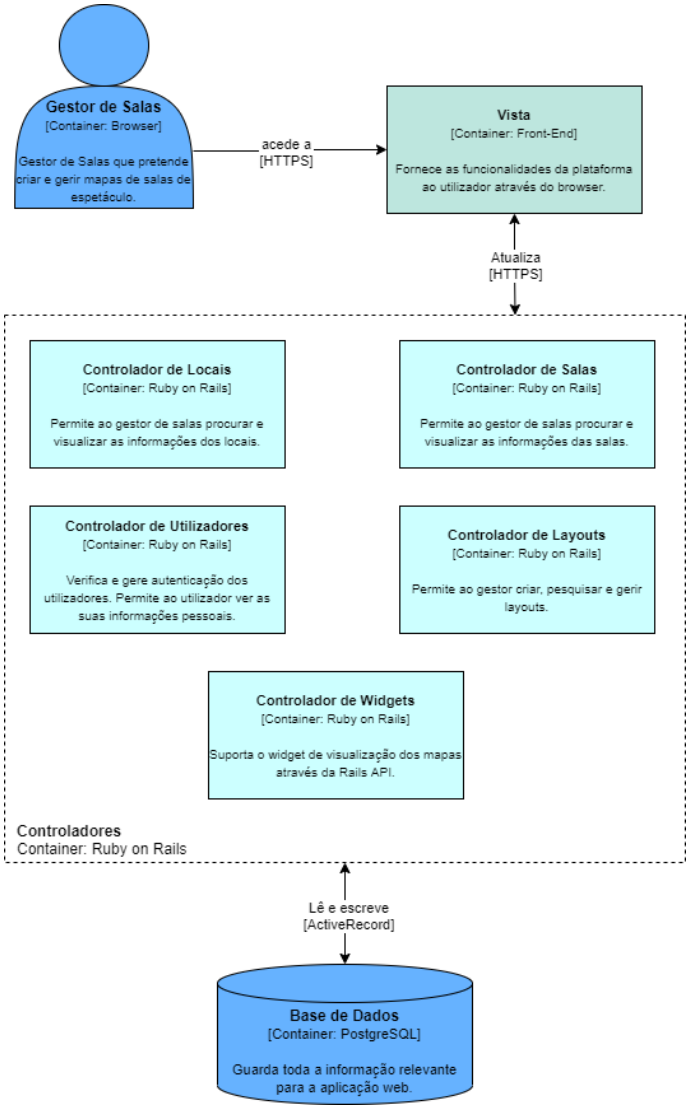


Figura 5.5: Diagrama de Componentes do Sistema - Controladores

5.3 *Wireframes*

Os *Wireframes*[32] são esboços simples que demonstram como o conteúdo das páginas está estruturado e organizado. O principal objetivo é validar e estruturar ideias sem preocupações de design e estilos. Durante o desenvolvimento da arquitetura do sistema, o autor e a equipa de design da The Loop Company analisaram os requisitos deste projeto e planearam a estrutura das páginas de acordo com as suas necessidades. Posteriormente, tendo como base as decisões tomadas nas reuniões anteriores, a equipa de design criou uma série de Wireframes para apresentar ao cliente, que aprovou resultado final. A lista completa de Wireframes pode ser consultada nos apêndices do relatório.

A figura 5.6 mostra a página inicial da plataforma onde o utilizador pode visualizar e procurar os *layouts* recentemente modificados. Adicionalmente, o utilizador pode criar um novo *layout* através do atalho presente nesta página.

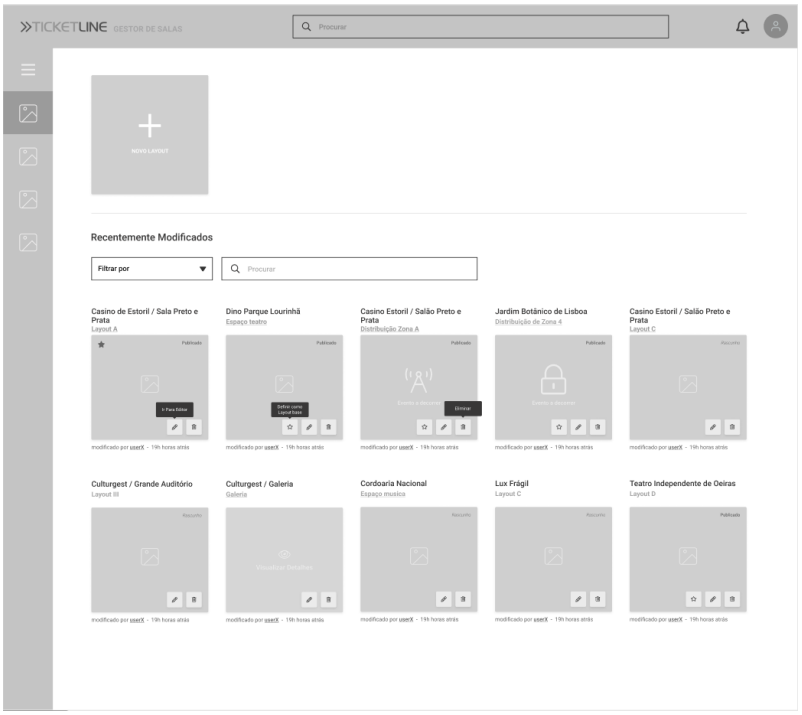


Figura 5.6: Exemplo de *Wireframe*: Página Inicial

A página de detalhes de um local, na figura 5.7, mostra as informações de um local como o nome, id e lista de salas associadas a este local. Em adição, o utilizador pode pesquisar por salas dentro do local.

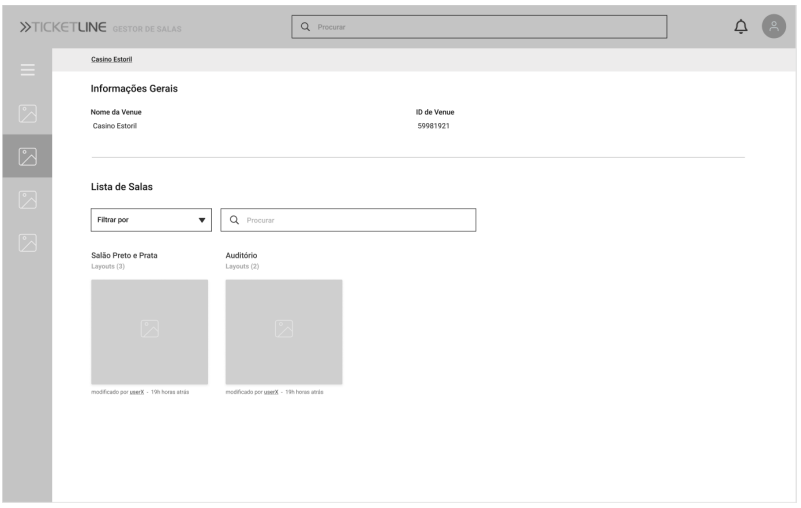


Figura 5.7: Exemplo de *Wireframe*: Página de Local

Na figura 5.8 é apresentada a página de lista de locais onde o utilizador pode procurar e selecionar um local.

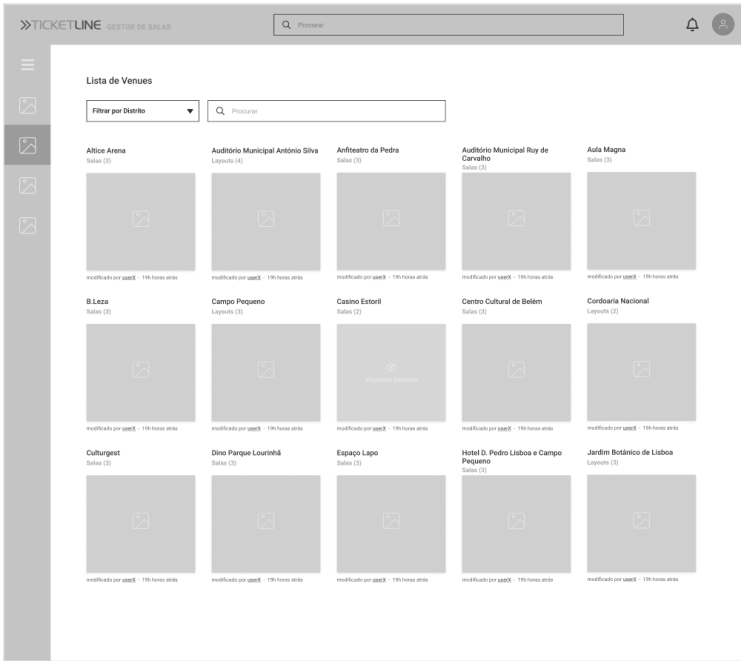


Figura 5.8: Exemplo de *Wireframe*: Lista de Locais

A figura 5.9 mostra a página de detalhes de uma sala, onde o utilizador pode ver o nome, id, capacidade e o *layout* base. Adicionalmente, o utilizador pode pesquisar por outros *layouts* que estejam associados a esta sala ou criar um novo *layout*.

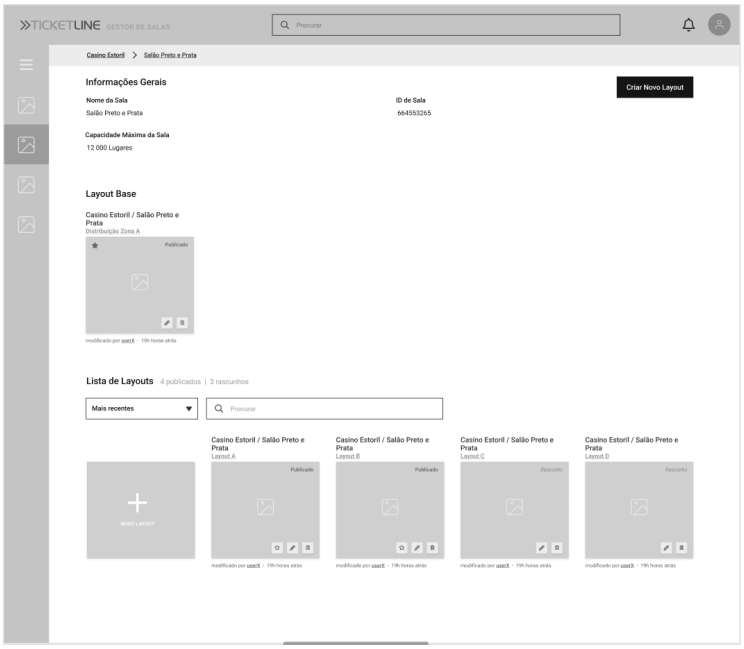


Figura 5.9: Exemplo de *Wireframe*: Página de Sala

Na figura 5.10 é possível visualizar a página de detalhes de um *layout*. Permite ao utilizador visualizar as informações de um *layout* como o nome, id, eventos associados, estado, lotação máxima e registo de alterações. A figura mostra também as operações que o utilizador pode realizar dentro de um *layout*.

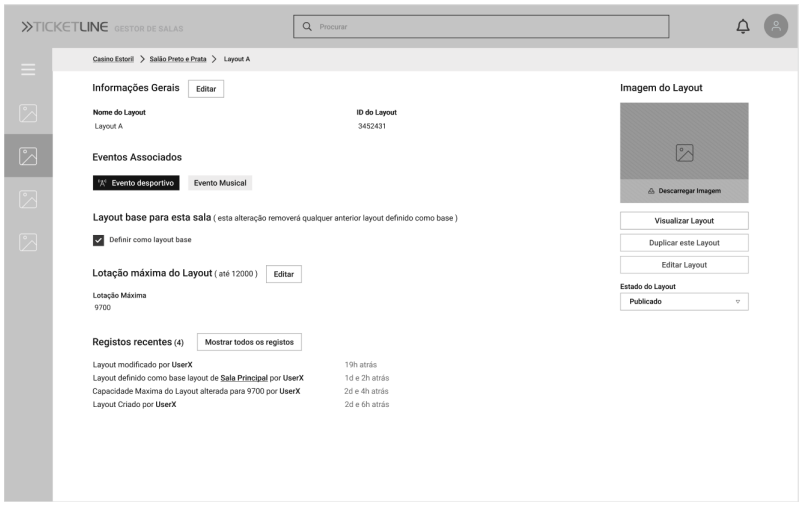


Figura 5.10: Exemplo de *Wireframe*: Página de *Layout*

A lixeira de *layouts*, na figura 5.11, permite ao utilizador procurar por um *layout* que tenha sido apagado e, posteriormente, recuperá-lo ou apagá-lo permanentemente.

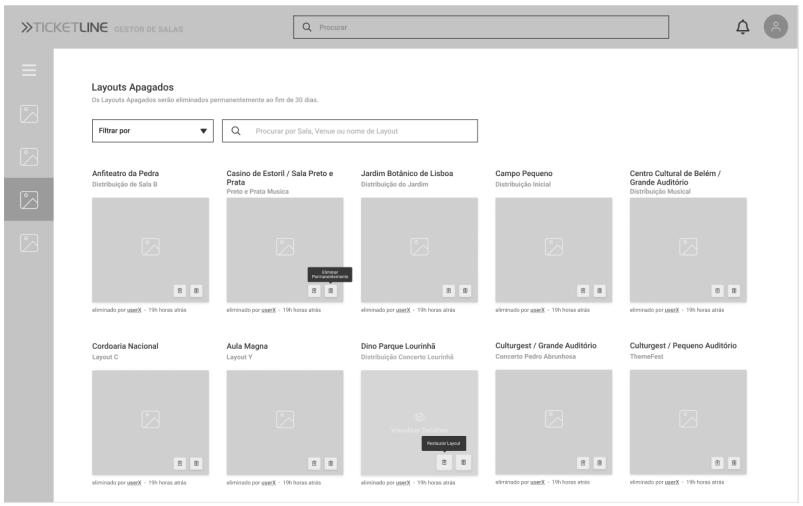


Figura 5.11: Exemplo de *Wireframe*: Lixeira de *Layouts*

Na figura 5.12 é possível ver o perfil do utilizador que contém informações como nome, contactos, email, password e as últimas ações feitas pelo utilizador.

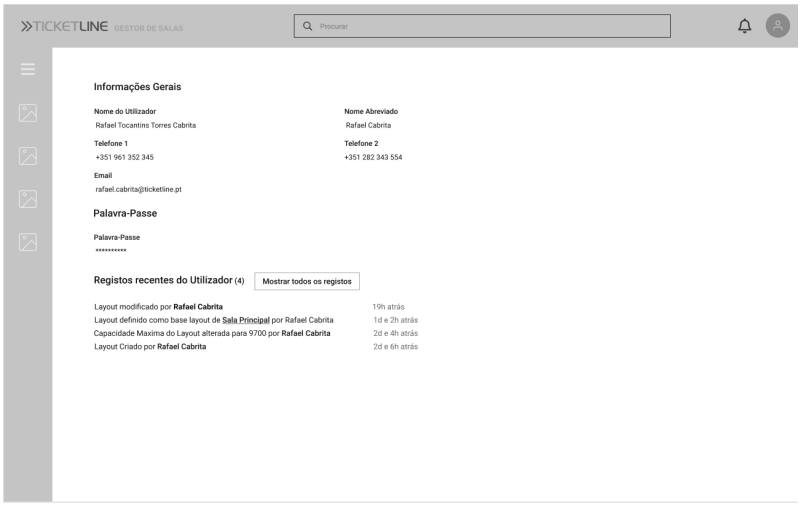


Figura 5.12: Exemplo de *Wireframe*: Perfil do Utilizador

A figura 5.13 mostra o editor de *layouts* da plataforma e as operações que o utilizador pode realizar na criação do mapa de uma sala.

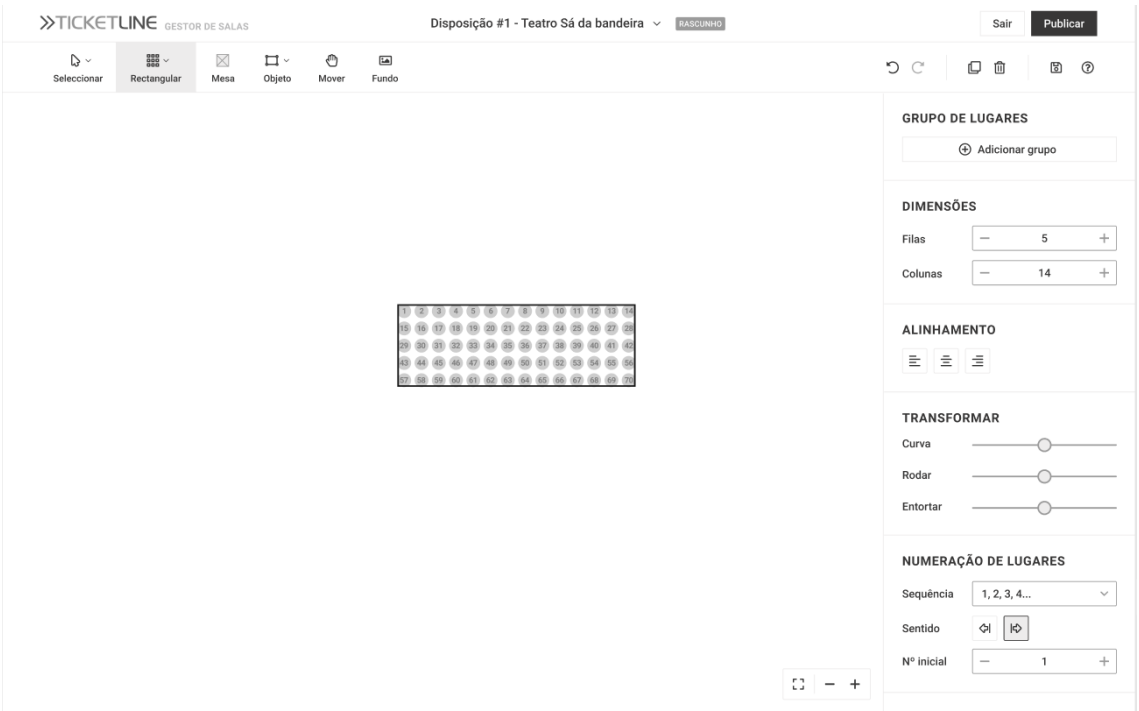


Figura 5.13: Exemplo de *Wireframe*: Editor de *Layouts*

5.4 Diagrama de Navegação

Um diagrama de navegação é uma mapa das conexões entre as várias páginas de um website.

Na figura 5.14 pode ser consultado o diagrama de navegação da Plataforma de Gestão de Salas.

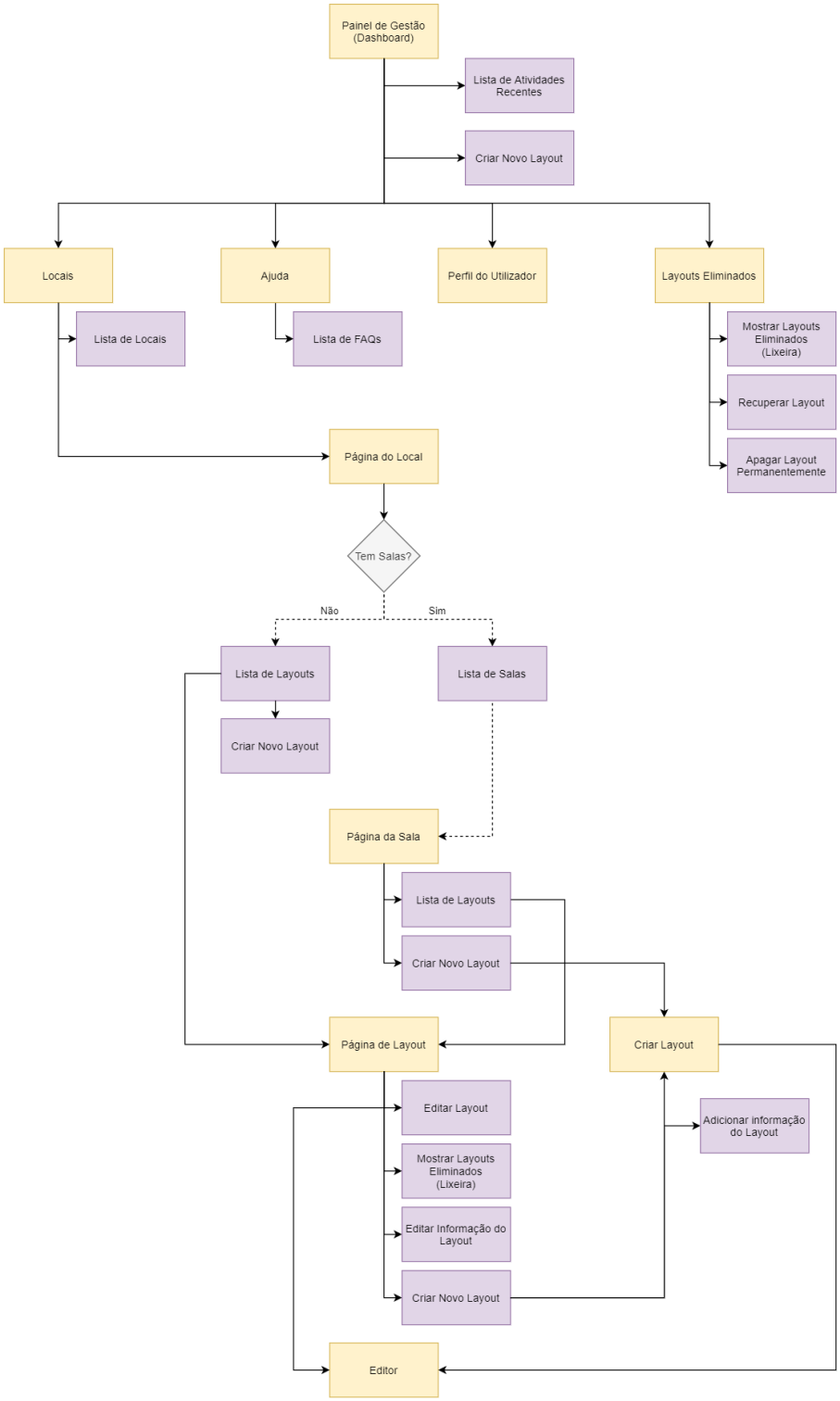


Figura 5.14: Diagrama de Navegação da Plataforma

Capítulo 6

Desenvolvimento do Sistema

Este capítulo explora o processo e o plano de desenvolvimento deste projeto. Em adição, é descrita a implementação dos vários componentes da plataforma de gestão de salas.

6.1 Processo de Desenvolvimento

Esta secção descreve o processo utilizado pela equipa na implementação deste projeto.

6.1.1 Organização da Equipa

Durante o projeto, o autor foi incluído na equipa que estava a desenvolver várias soluções para a Ticketline e, por essa razão, esteve presente em reuniões diárias com os restantes membros da equipa. Nestas reuniões era pedido que cada elemento fizesse um ponto de situação sobre as tarefas que estava a implementar para que os restantes membros pudessem estar a par do progresso de cada projeto e, principalmente, para que a gestora de projeto pudesse monitorizar o progresso de cada projeto. Adicionalmente, estas reuniões permitiram também o esclarecimento de dúvidas e discussão técnica entre os membros da equipa.

Além destas reuniões, foram também agendadas reuniões, habitualmente duas vezes por semana, com o *Tech Lead* da equipa para realizar o planeamento técnico das várias funcionalidades, nomeadamente sobre a organização do código e configuração de eventuais bibliotecas necessárias. Em caso de necessidade, estas reuniões serviam também para esclarecer dúvidas que surgiram durante a fase de desenvolvimento.

6.1.2 Organização de Tarefas

O início do desenvolvimento de um novo componente era marcado pelo agendamento de uma reunião com a orientadora e o *Tech Lead* para planear a sua implementação e criar tarefas para dividir e organizar todo o processo. Caso fosse necessário, devido à maior complexidade de implementação, poderiam ser também discutidas questões técnicas como a organização do código ou bibliotecas necessárias.

Para organizar as tarefas durante o projeto foi utilizado o ClickUp[33], um software que permite criar quadros de tarefas e dividi-las de acordo com o seu estado de desenvolvimento. Estes quadros podem depois ser partilhados com a equipa para que cada elemento possa

consultar as tarefas que tem de realizar e atualizar o seu progresso. Um exemplo do quadro do ClickUp, durante a fase de desenvolvimento, pode ser consultado na figura 6.1.

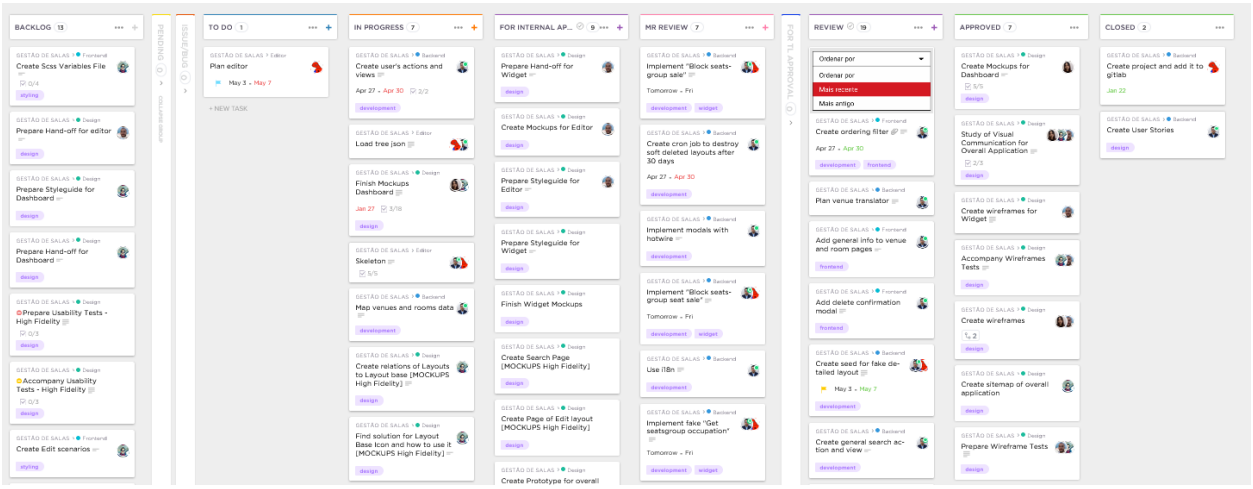


Figura 6.1: Exemplo do quadro do ClickUp durante a fase de desenvolvimento

Backlog - Lista de tarefas que ainda não foram realizadas.

Pending - Lista de tarefas que estão dependentes de desenvolvimentos da Ticketline.

Issue/Bug - O código foi revisto e continha bugs ou outros problemas.

To Do - Tarefas a realizar num futuro próximo.

In Progress - Tarefas que estão a ser realizadas.

MR Review - Código que aguarda revisão para ser *merged* com o *branch* de desenvolvimento.

Review - Tarefas que aguardam revisão da gestora de projetos.

Closed - Tarefas terminadas.

Quando o autor iniciava o desenvolvimento de uma funcionalidade, mudava o cartão da respetiva tarefa da coluna "*To Do*" para "*In Progress*" para sinalizar os orientadores que tinha começado a trabalhar naquela funcionalidade. No momento em que a tarefa estava concluída, era criado um *Merge Request* para o *branch* de desenvolvimento e o cartão da tarefa era mudado para a coluna de "*MR Review*". Na descrição do *Merge Request* eram indicadas as funcionalidades adicionadas, alteradas ou removidas e era feita a associação à tarefa criada no ClickUp. O *Merge Request* criado pelo autor era posteriormente revisto pelo *Tech Lead* para garantir que o código estava de acordo com o esperado. Se fossem necessárias alterações, o cartão era movido para a coluna de "*Issue/Bug*" e era feita uma descrição das alterações necessárias para que o autor as pudesse implementar. Quando o código era aprovado, a tarefa passava para a coluna de "*Review*"e ficava a aguardar aprovação por parte da gestora de projeto que poderia aceitar o resultado e concluir a tarefa ou indicar novas alterações que fossem necessárias.

6.1.3 Verificação do Código

Como foi referido na subsecção acima, o código desenvolvido pelo autor, tal como todo o código produzido na The Loop Company, é revisto por membros seniores para tentar diminuir a ocorrência de erros no código e garantir que as boas práticas de *Ruby on Rails* estão a ser cumpridas.

Em adição a este método de revisão, foram também adicionadas algumas ferramentas que faziam verificações antes de cada *commit*, tais como:

- Rubocop[34] - verificador de estilo e formatação de código baseado no guia de estilos de Ruby on Rails mantido pela comunidade.
- Brakeman[35] - ferramenta de análise de código que procura vulnerabilidades de segurança em aplicações na linguagem Ruby on Rails.
- Bundle-Audit[36] - ferramenta que procura vulnerabilidades nas bibliotecas de Ruby utilizadas no projeto.
- Yarn-Audit[37] - ferramenta que procura vulnerabilidades nas bibliotecas de javascript que foram importadas pelo Yarn[38].

6.2 Plano de Desenvolvimento

A presente secção descreve os objetivos, em termos de requisitos implementados, que se planeia atingir na realização deste projeto.

6.2.1 Objetivos

O objetivo deste projeto é desenvolver um Produto Viável Mínimo que, neste caso, consiste na implementação dos requisitos que foram considerados obrigatórios na tarefa de priorização de requisitos. No entanto, visto que algumas funcionalidades estão dependentes de outras APIs que ainda estão em desenvolvimento na Ticketline, existe o risco de não ser possível implementar algumas funcionalidades.

Relativamente ao plano de desenvolvimento da plataforma, em adição à priorização realizada, foi acordado, em reunião com os restantes membros da equipa e com o cliente, que o desenvolvimento se iniciaria com a implementação das funcionalidades do *dashboard*, seguido do Tradutor, posteriormente o *widget* e, finalmente, o Editor de *layouts*. No desenvolvimento destes componentes serão sempre implementados em primeiro lugar os requisitos com maior índice de prioridade.

6.2.2 Requisitos Funcionais Implementados

Nesta subsecção são apresentados os requisitos funcionais implementados na fase de desenvolvimento do projeto. No caso de existirem requisitos não implementados são fornecidas justificações para o sucedido. As tabelas apresentadas abaixo mostram os requisitos funcionais identificados no capítulo 4 com indicação da sua prioridade e sucesso da implementação.

As prioridades estão representadas de acordo com a seguinte legenda:

- *Must Have* (M) - requisitos que são absolutamente necessários para o sucesso do projeto.
- *Should Have* (S) - requisitos que não são vitais mas adicionam valor significativo.
- *Could Have* (C) - requisitos que não são necessários mas têm um pequeno impacto nos resultados do projeto.

Dashboard

Como é possível verificar na tabela 6.1, a maioria das funcionalidades planeadas para o *dashboard* foi implementada com sucesso. No entanto, existiram dois requisitos que não foram implementados e que, apesar de não serem considerados obrigatórios, poderiam acrescentar algum valor ao produto final. A razão para a não implementação destes requisitos foi não existirem, até à data, *endpoints* nas API's da Ticketline que possibilitem o desenvolvimento e integração destas funcionalidades.

Tabela 6.1: Requisitos Funcionais - *dashboard*

Requisitos Funcionais		**Prioridade**	**Concluído**
Autenticação	Autenticar Utilizadores	M	Sim
Utilizador	Visualizar perfil	S	Não
Venues	Listar Venues	M	Sim
	Pesquisar Venues	M	Sim
Salas	Listar Salas associadas a uma Venue	M	Sim
	Pesquisar Salas associadas a uma Venue	C	Não
layouts	Adicionar/Editar layouts	M	Sim
	Eliminar layouts	M	Sim
	Duplicar layouts	M	Sim
	Recuperar layouts eliminados (até 30 dias)	S	Sim
	Associar layouts a uma Venue/Sala	M	Sim
	Listar layouts associados a uma Venue/Sala	M	Sim
	Listar layouts recentemente modificados	S	Sim
	Listar layouts eliminados	S	Sim
	Listar layouts importados	S	Sim
	Pesquisar layouts associados a uma Venue/Sala	M	Sim
	Pesquisar layouts recentemente modificados	S	Sim
	Pesquisar layouts eliminados	S	Sim
	Pesquisar layouts importados	S	Sim
	Mostrar alterações feitas a um layout	S	Sim

Widget de Visualização de Salas

Na tabela 6.2 é possível verificar que não foram cumpridos dois requisitos relativos ao widget. Como são dois requisitos que não foram considerados obrigatórios na fase de priorização, foi decidido que seria mais prioritário avançar imediatamente para o desenvolvimento do componente de edição, para que fosse possível implementar todos os requisitos obrigatórios, e assim cumprir os objetivos traçados para o desenvolvimento da plataforma.

Tabela 6.2: Requisitos Funcionais - *widget*

Requisitos Funcionais		**Prioridade**	**Concluído**
Lugares	Selecionar lugares	M	Sim
	Apresentar informações sobre lugares (tags)	S	Não
	Alterar a disponibilidade dos lugares em tempo real	M	Sim
	Clicar sobre uma zona, setor ou anel para expandir	M	Sim
Grupos de Lugares	Apresentar a disponibilidade dos lugares	M	Sim
	Apresentar zonas sem lugares marcados	M	Sim
	Fazer *zoom* sem perder a visão global da sala	M	Sim
	Apresentar a lista de zonas, setores ou anéis	S	Não

Editor de Mapas

Relativamente ao editor, como é visível na tabela **??**, não foram cumpridos sete requisitos não obrigatórios. Dois dos requisitos, um obrigatório e outro não obrigatório, foram cumpridos de forma parcial. Neste caso, foi considerado que seria importante passar à fase de testes para que o planeamento do segundo semestre pudesse ser cumprido.

Em relação aos requisitos implementados parcialmente, é importante considerar que:

(1) O único tipo de conjunto de lugares implementado foi o retângulo. Ainda assim, é possível criar semicírculos através da junção do retângulo e de operações de adição de lugares singulares.

(2) Foi apenas implementada a operação rotação, devido a ter sido considerado que seria a opção fundamental para combater uma das principais limitações da plataforma anterior: a impossibilidade de saber a posição dos lugares em relação ao palco.

Tabela 6.3: Requisitos Funcionais - Editor

Requisitos Funcionais		**Prioridade**	**Concluído**
Guardar	Guardar Alterações	M	Sim
	Guardar Alterações Automaticamente	S	Não
Zonas, Setores e Bancadas	Adicionar/editar zonas, setores e bancadas	M	Sim
	Eliminar zonas, setores e bancadas	M	Sim
	Selecionar zonas, setores e bancadas	M	Sim
	Mover zonas, setores e bancadas	M	Sim
Lugares	Adicionar/Editar Lugares	M	Sim
	Eliminar lugares	M	Sim
	Selecionar Lugares	M	Sim
	Mover Lugares	M	Sim
	Atribuir números a lugares	M	Sim
	Atribuir *tags* a lugares (mobilidade ou visibilidade reduzida e VIP)	S	Não
	Associar lugares a zonas, setores ou bancadas	M	Sim
Conjuntos de Lugares (Retângulos, Semicírculos, Anéis e Mesas)	Adicionar/Editar Conjuntos de Lugares	M	Parcialmente(1)
	Eliminar Conjuntos de Lugares	M	Sim
	Mover Conjuntos de Lugares	M	Sim
	Atribuir *labels*/números a filas e lugares de acordo com algoritmos	S	Sim
	Atribuir *tags* a Conjuntos de Lugares	S	Não
	Curvar, esticar ou rodar Conjuntos de Lugares	S	Parcialmente(2)
Formas Geométricas	Adicionar/Editar formas geométricas	S	Sim
	Eliminar formas geométricas	S	Sim
	Mover formas geométricas	S	Sim
	Selecionar formas geométricas	S	Sim
	Adicionar tipos a formas geométricas (Palco, Obstáculo, Bar, WC, Portas, Outro)	S	Sim
	Associar formas geométricas a zonas, setores ou bancadas	S	Sim
Outros	Bloquear edições simultâneas a um *layout*	S	Não
	Replicar alterações feitas no layout base noutros layouts	C	Não
	Importar uma imagem para o fundo da sala	S	Não
	Importar um ficheiro CSV para gerar o rascunho da sala	C	Não

6.3 Componentes

O propósito desta secção é descrever o processo de desenvolvimento de cada componente. São analisados também os objetivos de cada componente e, quando aplicável, são revelados os maiores problemas que surgiram durante a sua implementação e descritas as soluções adotadas para os ultrapassar.

6.3.1 *Dashboard*

Nesta subsecção são descritos os objetivos que o *dashboard* deve conseguir cumprir e é feita uma explicação, técnica e funcional, dos requisitos implementados.

Objetivo

O objetivo do desenvolvimento do *dashboard* é implementar um componente que permita criar e gerir novos *layouts* e associá-los a uma *venue* ou sala para que possam ser utilizados em eventos que ocorram nesses locais. Este componente serve também como ligação ao tradutor e editor de mapas, pois será através do *dashboard* que o utilizador poderá utilizá-los.

Descrição do Desenvolvimento

Do ponto de vista técnico, o *dashboard* consiste numa aplicação web desenvolvida em Ruby on Rails e, por essa razão, segue o modelo MVC. Isto significa que, de um modo geral, os requisitos implementados seguem o mesmo fluxo, isto é, o utilizador interage com a vista para efetuar um pedido a um controlador que irá consultar as informações presentes na base de dados, através da utilização dos modelos, para responder com uma nova vista.

Autenticação

A autenticação é um requisito importante do *dashboard* da Plataforma de Gestão de Salas, pois permite evitar que utilizadores não autorizados consigam aceder às funcionalidades da plataforma. Para gerir a autenticação na plataforma foi utilizada a *gem Warden*[39], que permite personalizar métodos de autenticação e serializar dados relevantes na sessão para saber que um utilizador está autenticado. Adicionalmente, possibilita redirecionar o utilizador para uma página específica cada vez que uma tentativa de autenticação falha.

No caso deste projeto, tal como foi referido na arquitetura do software, a autenticação é efetuada através de uma API disponibilizada pelo sistema de gestão de utilizadores da Ticketline. Esta API recebe uma combinação de nome de utilizador e palavra-passe, criada previamente no sistema de gestão de utilizadores da Ticketline, e retorna o identificador do utilizador, um JSON Web Token (JWT), um *refresh token* e a data de expiração do JWT. As informações retornadas pela API são serializadas na sessão para que possam ser acedidas posteriormente. Para garantir que um utilizador não-autenticado não possa aceder às funcionalidades da aplicação, é efetuada uma verificação para averiguar se o utilizador está autenticado que em caso de falha redireciona o utilizador para a página de *login* e em caso de sucesso autoriza o utilizador a realizar a ação pretendida.

Venues e Salas

Os requisitos relacionados com *venues* e salas englobam operações de listagem e pesquisa,

visto que estas entidades têm a sua própria plataforma de gestão onde são realizadas as restantes operações CRUD. Assim sendo, todos os requisitos implementados utilizam as APIs disponibilizadas por essa plataforma.

A página de *venues*, na figura 6.2, apresenta a lista total de *venues* disponíveis e permite que o utilizador pesquise *venues* pelo nome. Para não sobrecarregar as bases de dados foi utilizada a técnica de paginação disponibilizada pela API, que restringe o número de elementos retornados em cada pedido que, neste caso, foi limitado a 10 *venues* por pedido.

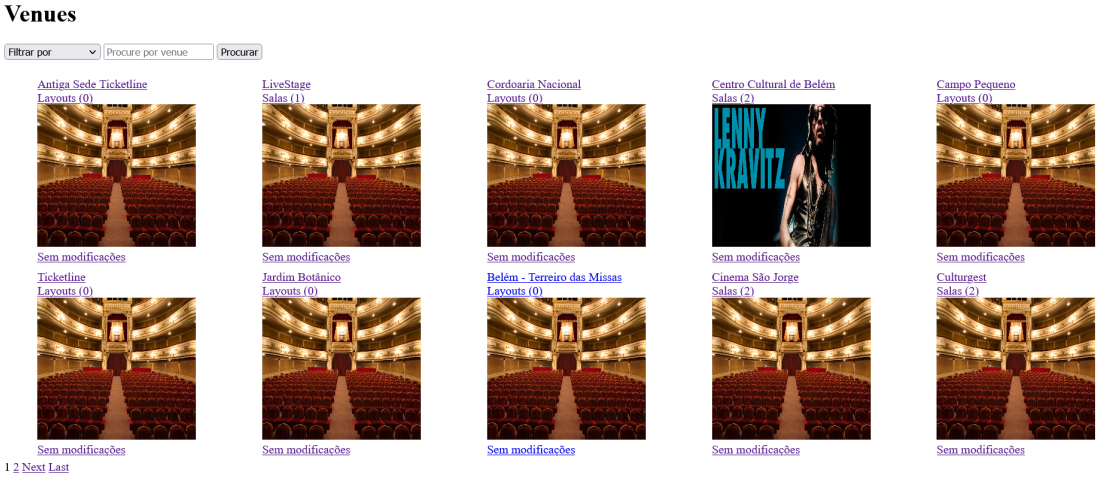


Figura 6.2: Página de *Venues*

A página de detalhes de uma *venue*, presente na figura 6.3, apresenta as informações gerais da *venues* e a lista de salas ou a lista de *layouts*, caso seja uma *venue* sem salas.

Figura 6.3: Página de detalhes da Venue

A página de detalhes de uma sala, figura 6.4, é semelhante à página da *venue*, pois apresenta também as informações gerais da sala e a lista de *layouts* onde o utilizador pode realizar pesquisas, filtrar os *layouts* publicados e em rascunho e ordenar por data da última modificação.

Figura 6.4: Página de detalhes da Sala

Layouts

Os *layouts* são a principal entidade gerida pelo *dashboard* e, por essa razão, é onde está condensada a maior parte das funcionalidades. De um modo geral, o *dashboard* implementa todos os métodos CRUD, sendo que existem algumas funcionalidades adicionais que permitem complementar estes métodos base.

A plataforma permite listar *layouts* de 4 maneiras diferentes, nomeadamente:

- *layouts* Associados a uma *Room/Venue* - os *layouts* que têm o campo "location_id" igual ao ID da *venue/room* e que não tenham sido eliminados.
- layouts Modificados Recentemente - os resultados distintos de um *left outer join*, pelo ID do *layout*, entre os *layouts* não eliminados e todos os registos de alterações que têm o ID do utilizador.
- layouts Eliminados - todos os *layouts* que têm o campo "*deleted_at*" diferente de *null*.
- layouts Importados - todos os *layouts* que têm o campo "*imported*" com o valor verdadeiro e que não tenham sido eliminados.

Estas operações de listagem são complementadas com pesquisas pelo nome do *layout*, que permitem filtrar e ordenar os resultados. Adicionalmente, é também suportada uma pesquisa geral de *layouts*, que procura pelo nome de todos os *layouts* que não tenham sido eliminados.

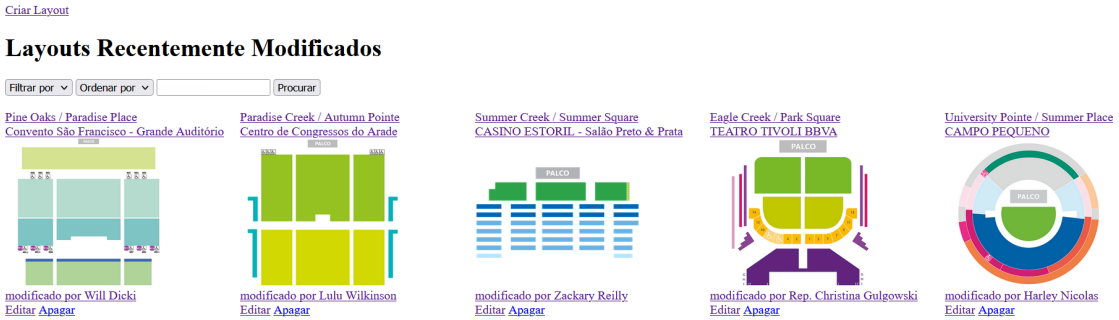


Figura 6.5: Exemplo de Listagem de *layouts*

A criação de *layouts* pode ser feita de duas formas distintas, ao preencher um formulário com os dados do novo *layout* ou ao duplicar um *layout* já existente. O formulário de criação do novo *layout* requer o nome do *layout*, a lotação, a *venue* ou sala a que será associado e a indicação sobre se será o *layout* base da sala.

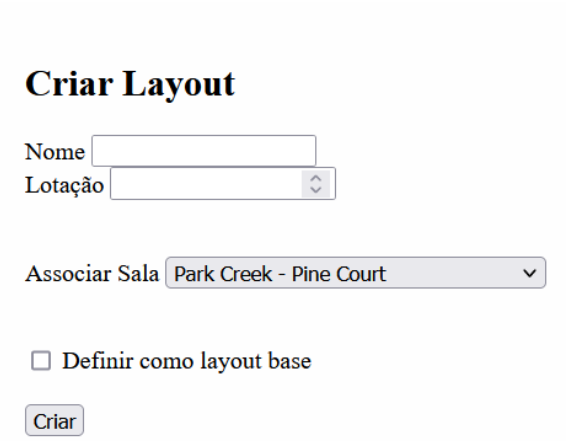


Figura 6.6: Formulário de Criação de um *layout*

A duplicação de um *layout* requer a *venue* ou sala a que um *layout* será associado, o nome do novo *layout* e a indicação sobre se será a o *layout* base da sala. Uma das preocupações no momento da duplicação de um *layout* é copiar todos os grupos de lugares e geometrias e não apenas o *layout*, para que futuras alterações no *layout* original não sejam reproduzidas nos *layouts* duplicados e vice-versa.

Figura 6.7: Formulário de Duplicação de um *layout*

A funcionalidade de eliminação de *layouts* ocorre em duas fases, em primeira instância, quando o utilizador elimina um *layout*, é atribuído um valor do campo "*deleted_at*" que consiste na data em que o *layout* foi eliminado. Desta forma, o *layout* passará a ser listado na lista de *layouts* eliminados e deixará de ser visível nas outras listas.

Quando o *layout* se encontra na lista de *layouts* eliminados, figura 6.8, o utilizador pode recuperá-lo, eliminá-lo permanentemente ou aguardar até que o *layout* seja eliminado automaticamente ao fim de 30 dias. Para recuperar um *layout* é atribuído o valor '*null*' ao campo "*deleted_at*", para que volte ao estado em que estava antes de ser eliminado. Para eliminar um *layout* permanentemente é necessário apagar todos os grupos de lugares e geometrias, diretamente ou indiretamente associados ao *layout*, e os registos de alterações desse *layout*.

Figura 6.8: Página de *layouts* eliminados

Para apagar os *layouts* ao fim de 30 dias, é utilizado um *Cron Job*, criado através da *gem* Sidekiq[40], que executa uma tarefa todos os dias à meia-noite e elimina os *layouts* cuja data presente no campo "*deleted_at*" tenha sido há mais de 30 dias.

Finalmente, a edição de *layouts* é realizada na página de detalhes do *layout* (figura 6.9). O utilizador pode alterar o nome, a lotação, o estado e a indicação de base da sala. Visto que apenas pode existir um *layout* base, quando um *layout* é definido como base, é importante remover o *layout* base atual do estatuto de base.

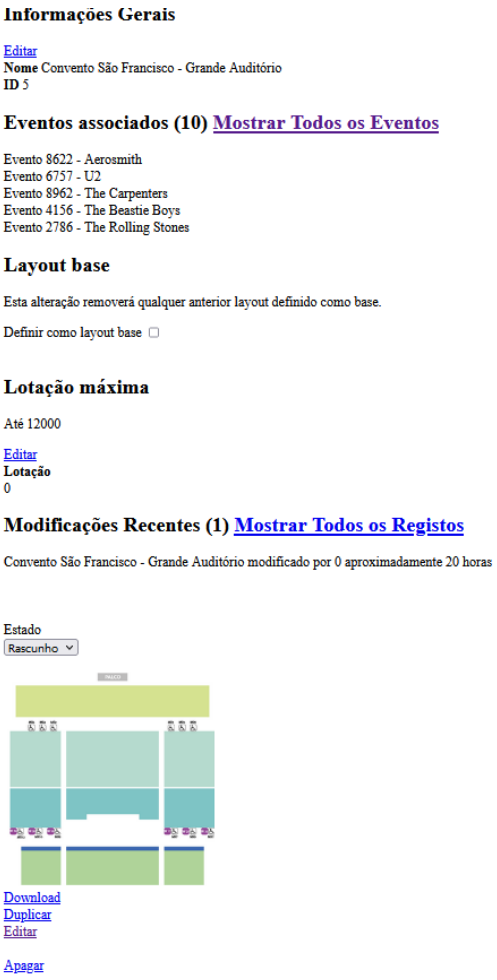


Figura 6.9: Página de detalhes do *layout*

Pesquisa

Para a pesquisa de *layouts* foi utilizada a *gem* pg_search[41] que facilita a pesquisa de texto no PostgreSQL[15] ao disponibilizar algoritmos que permitem lidar com erros ortográficos ou apresentar os resultados sem ser necessário escrever o nome completo do *layout*, neste caso, para obter o resultado que se procura. Em maior detalhe, os algoritmos utilizados neste projeto foram:

- O algoritmo *double metaphone*[42] que permite fazer a correspondência entre palavras que soam semelhantes mesmo que sejam escritas de forma bastante diferente.
- O algoritmo *trigram search*[43] que procura quantas *substrings* de 3 letras, ou trigramas, correspondem entre duas palavras. Desta forma é possível obter os resultados que se procura mesmo no caso de existirem erros ortográficos ou de digitação.

A biblioteca permite também configurar o peso que cada algoritmo tem em relação ao outro, sendo que, neste caso, se optou por dar um peso igual a cada algoritmo. Adicionalmente, foi também utilizada uma função que permite ignorar a acentuação das palavras durante a pesquisa.

Para evitar que a base de dados fosse sobrecarregada com pedidos muito extensos foi necessário limitar o número de resultados provenientes de uma pesquisa. Para isso, os resultados foram limitados a 10 itens através da utilização de um sistema de paginação criado pela *gem* Kaminari[44] que permite que os resultados de uma *query* sejam divididos em várias páginas.

6.3.2 Tradutor

Nesta subseção são descritos os objetivos que o tradutor deve conseguir cumprir, as decisões técnicas tomadas e os problemas enfrentados durante a fase de desenvolvimento do componente.

Objetivo

O objetivo do tradutor de salas é importar mapas de salas da plataforma antiga e traduzi-los para o novo modelo de dados. Desta forma, será possível utilizar todos os componentes da plataforma mesmo que ainda não tenham sido criados todos os mapas das salas através do novo editor de salas para que a transição possa ser feita de forma gradual sem o negócio deixar de funcionar.

Descrição do Desenvolvimento

Para entender a melhor forma de desenvolver o tradutor de salas, foi necessário estudar o funcionamento da plataforma antiga, e o respectivo modelo de dados, e mapear as informações necessárias para o novo modelo de dados.

Na plataforma antiga, os mapas das salas são representados por uma imagem que utiliza um mapa HTML que permite definir áreas da imagem clicáveis que contém uma ligação para um URL. Estas áreas são definidas através de conjuntos de coordenadas que estão persistidas na base de dados, assim como o URL para a imagem. No caso da nova plataforma, os mapas são construídos através da junção de imagens SVG que são manipuladas com uso de *javascript*, para permitir interactividade com o utilizador.

Por essa razão, o plano inicial para importar o mapa das zonas era criar polígonos SVG, através de *paths*, com as coordenadas presentes na base de dados. No entanto, verificou-se que as coordenadas de cada área não tinham sido recolhidas com o rigor necessário, porque não era necessário na implementação antiga, visto que as áreas HTML não eram visíveis para o utilizador. Essa limitação fez com que a solução passasse por aplicar uma lógica semelhante à usada na antiga plataforma, isto é, importar a imagem do fundo da sala e criar polígonos transparentes onde o utilizador pode clicar. Na figura 6.10, está representado o exemplo de um SVG , criado a partir das coordenadas presentes na plataforma antiga, que apresenta irregularidades entre zonas adjacentes.

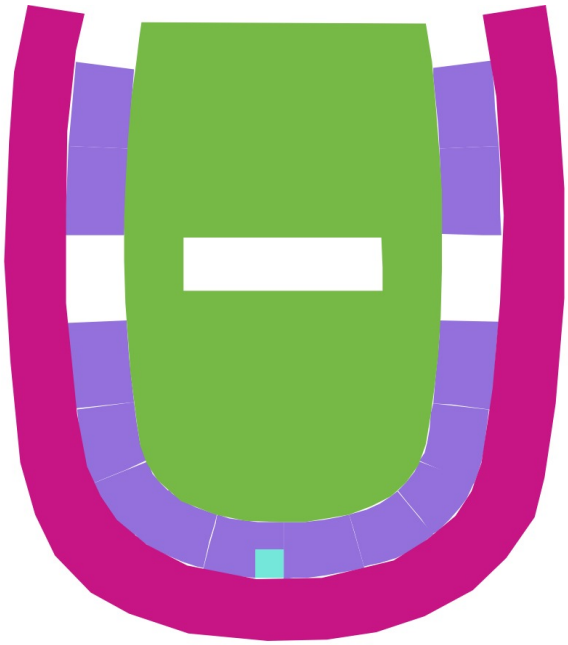


Figura 6.10: Exemplo de SVG criado com as coordenadas do sistema antigo

Posto isto, o próximo passo foi importar os mapas de lugares que, na plataforma antiga, eram implementados no formato de matriz. Para isso, seria necessário transformar o valor dos índices em coordenadas, x e y, relativamente ao SVG da zona a que pertenciam. Contudo, no caso em que os mapas de lugares necessitavam de ser rodados para ficarem dentro dos limites da zona, não era possível determinar o ângulo de rotação necessário pois não havia indicação da posição em relação ao palco.

Assim, a solução passou por utilizar uma solução semelhante à atual e apresentar os mapas de lugares dentro de um *pop-up* que é apresentado ao utilizador assim que faz um clique sobre uma grupo de lugares. Dentro deste *pop-up* existe um SVG onde são adicionados os SVGs de cada lugar da zona. As coordenadas de cada lugar obtêm-se através da multiplicação do índice pela soma do diâmetro e espaço de lugares para que os lugares fiquem nas suas linhas e colunas corretas e com um espaçamento entre eles.

Relativamente aos restantes dados necessários ao novo modelo, não foi necessário realizar quaisquer transformações visto que era possível fazer a correspondência entre os campos antigos e os novos.

Posto isto, é importante referir que a importação de *layouts* é feita no *dashboard*, através da submissão do formulário presente na figura 6.11. Este formulário inclui os campos necessários para a criação de um *layout* e também dois campos, *venue* e evento, que definem o mapa da sala a importar da plataforma antiga.

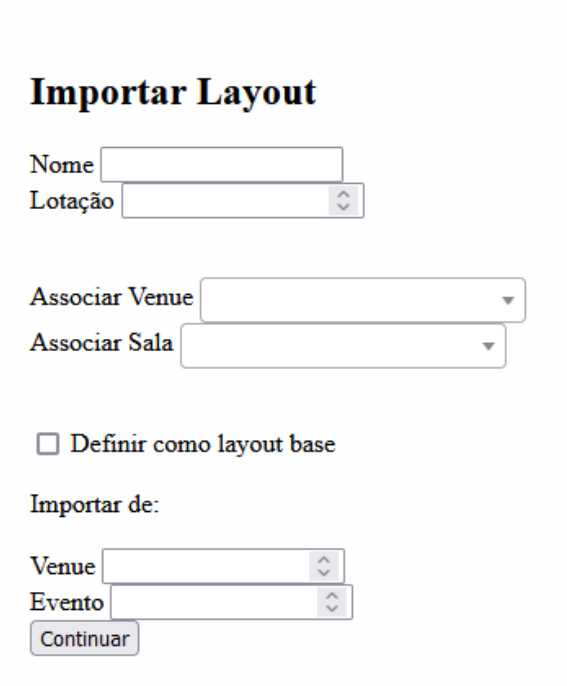


Figura 6.11: Formulário de Importação de *layouts*

Quando o formulário é submetido é agendado um *ActiveJob*, que permite realizar a importação de forma assíncrona. A opção pelo funcionamento assíncrono deve-se ao facto da importação ser uma tarefa demorada e que impediria o utilizador de realizar outras tarefas enquanto aguarda pela sua conclusão. Até o *layout* acabar de ser importado não é possível realizar operações que afetem os grupos de lugares e lugares pertencentes a este novo *layout*, pois poderiam gerar conflitos indesejados.

Dificuldades no Desenvolvimento

Durante o desenvolvimento deste componente surgiram algumas dificuldades, maioritariamente devido às diferenças no funcionamento do sistema atual e antigo.

Como foi referido na subsecção acima, o primeiro problema deve-se ao facto dos polígonos criados com as coordenadas presentes no sistema antigo terem sido criados apenas como uma camada invisível em que o utilizador pode clicar. Por essa razão, os polígonos apresentavam defeitos e poderiam existir espaços em branco entre zonas adjacentes.

Outro problema foi o facto de não existir, no modelo antigo, qualquer referência do palco. Assim, como não se sabe exatamente a localização do palco, não é possível orientar os lugares na sua direção. Adicionalmente, não existe qualquer referência à forma de cada polígono logo, não é possível garantir que os lugares são renderizados dentro dos limites do polígono, algo que poderia causar problemas de visualização. A solução para este problema foi adaptar o *widget* e utilizar, nos *layouts* importados, um modal para a visualização dos lugares. Esta solução será explorada em maior detalhe na secção seguinte.

6.3.3 *Widget*

Nesta subseção são descritos os objetivos que se pretendem atingir com o desenvolvido do *widget* de visualização de salas e é documentada a sua implementação.

Objetivo

O objetivo do *widget* de visualização de salas é apresentar o mapa da sala e permitir que o utilizador possa interagir com o mapa para realizar operações sobre lugares e grupos de lugares.

Descrição do Desenvolvimento

O *widget* de visualização de salas foi desenvolvido em *vanilla javascript* e é suportado por uma API disponibilizada na plataforma de gestão de salas. Para realizar a sua integração nas diversas plataformas, como o *website*, POS e plataforma de gestão de eventos, será adicionado um *iframe* às páginas que necessitarem com uma ligação ao URL onde o *widget* está *deployed*. Para inicializar o *widget* neste URL é necessário passar, por parâmetro, o ID do *layout* desejado. Um exemplo do mapa de uma sala apresentado no *widget* pode ser consultado na figura 6.12.

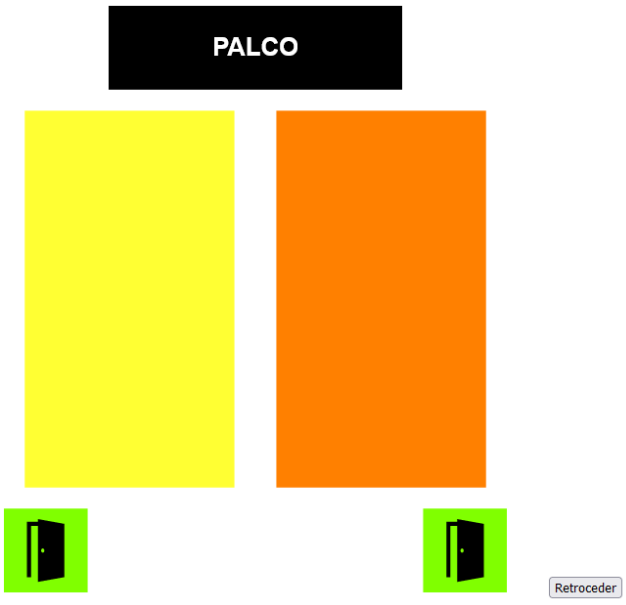


Figura 6.12: Exemplo do Mapa Inicial do *widget*

Para apresentar o mapa ao utilizador, o *widget* recorre à API para fazer um pedido que retorna a primeira camada do mapa, isto é, o grupo de lugares raiz da sala e todos os seus grupos de lugares descendentes. Seguidamente, é construído o SVG que representa o mapa da sala, que segue a seguinte estrutura:

```
<svg> // raiz da sala
    <svg></svg> // descendente 1
    <svg></svg>    // descendente 2
    <svg></svg>    // descendente 3
    (...)
</svg>
```

Como é possível verificar, a estrutura do SVG da sala respeita as relações presentes na base de dados, isto é, se um grupo de lugares descende de outro grupo de lugares então o seu SVG será adicionado dentro do SVG do seu grupo de lugares pai.

Assim que o mapa é renderizado, o utilizador pode clicar sobre os SVGs de cada grupo de lugares para os expandir. Quando isto acontece, é feito um novo pedido à plataforma de gestão de salas, para obter todos os grupos de lugares ou lugares que descendem desse grupo de lugares. Tal como foi referido, os SVGs desses grupos de lugares são inseridos dentro do SVG do grupo de lugares expandido. Na figura 6.13, é apresentado um exemplo de um grupo de lugares expandido cujos filhos são também grupos de lugares.

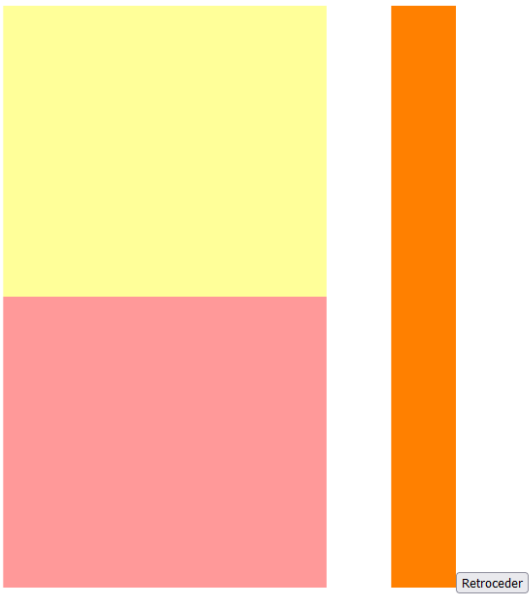


Figura 6.13: Exemplo de Grupo de Lugares Expandidos com Grupos de Lugares

No caso da camada seguinte ser constituída apenas por lugares, figura 6.14, além de ser necessário realizar o processo descrito acima para adicionar os SVGs de cada lugar, é necessário fazer outras operações extra. Em primeiro lugar, é feita uma conexão a um servidor de *WebSocket* que devolve a disponibilidade dos lugares dessa zona e atualiza, em tempo real, a disponibilidade de cada lugar. Adicionalmente, é necessário manter um pré-carrinho que retém os lugares que o utilizador seleccionou. Quando um lugar é selecionado, é enviado um pedido à API que bloqueia o lugar e difunde essa informação a todos os clientes que estejam ligados ao servidor *WebSocket*. Assim que um lugar é selecionado, o utilizador tem 10 minutos para concluir a sua compra ou então os lugares são novamente libertados. Caso o utilizador volte a clicar sobre um lugar selecionado, é feito um novo pedido para libertar esse lugar que será também difundido para os clientes ligados ao servidor *WebSocket*.

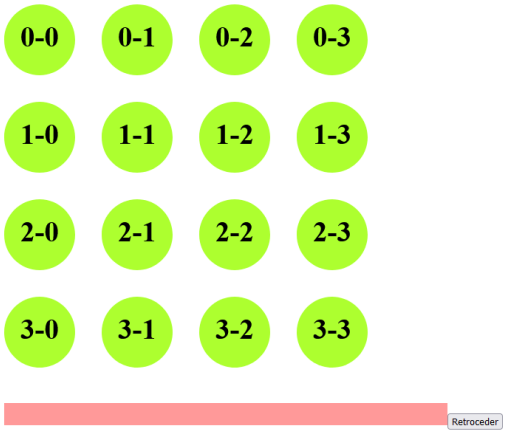


Figura 6.14: Exemplo de Grupo de Lugares Expandidos com Lugares

No caso dos *layouts* importados, tal como foi referido na secção anterior, os lugares são apresentados dentro de um modal como forma de resolver alguns problemas encontrados durante a tradução dos modelos de dados. Na figura 6.15, é apresentado um exemplo de uma zona pertencente a um *layout* importado.

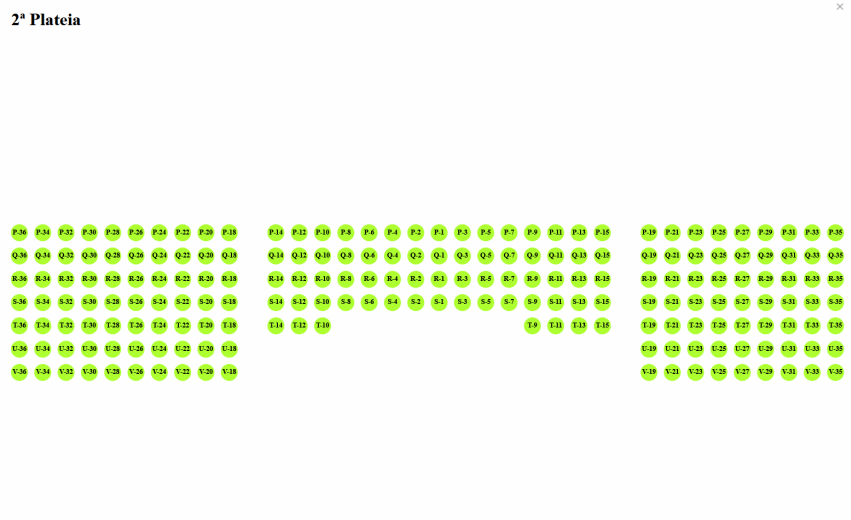


Figura 6.15: Exemplo de Modal de Lugares

Como os valores de x e y de cada lugar que são guardados na base de dados são os índices das linhas e colunas, é necessário transformá-los em coordenadas para que os lugares possam ser corretamente visualizados pelo utilizador. Para fazer essa transformação, esses índices são multiplicados pela soma do diâmetro e espaçamento de lugares. Dessa forma, como é possível visualizar na figura 6.15, os lugares são facilmente distinguíveis e os formatos são apresentados corretamente.

Atualização do Estado dos Lugares em Tempo Real

A atualização do estado dos lugares em tempo real é um dos requisitos obrigatórios deste projeto. O objetivo deste requisito é permitir aos clientes ter sempre uma visão atualizada dos lugares disponíveis e, ao mesmo tempo, garantir que um lugar selecionado não pode ser selecionado por outro utilizador em simultâneo.

Para manter a informação sobre os lugares bloqueados foi utilizado Redis[45], que é uma base de dados em memória do tipo chave-valor. Esta ferramenta suporta vários tipos de dados, sendo que para esta funcionalidade foram utilizados *hashes* que são mapas entre os campos e os valores de texto. Neste caso em particular, o *hash* irá conter todos os identificadores dos lugares bloqueados e estará associado a uma chave que será uma combinação entre o identificador da sessão e o identificador do grupo de lugares.

Cada vez que um utilizador selecionar um lugar será efetuado um pedido à plataforma de gestão de salas que irá verificar se o lugar já existe dentro do *hash* para aquela sessão e grupo de lugares e, caso não exista, é criado um novo campo com o identificador do lugar. O resultado desta operação será retornado para o *widget* que irá atualizar o estado do lugar consoante a resposta recebida. Se o utilizador voltar a selecionar o lugar já escolhido, é repetido o processo mas para desbloquear o lugar. Estas tarefas encontram-se definidas em *scripts* da linguagem Lua[46] que são executados pelo Redis de forma atómica, o que resolve quaisquer problemas de concorrência.

Para evitar que os lugares que foram bloqueados mas não reservados fiquem indisponíveis eternamente, foi criado outro *hash* que contém todos os lugares que aguardam a confirmação da reserva. Adicionalmente, foi implementada uma função que itera, a cada 10 segundos, sobre estes lugares e liberta aqueles cuja data de expiração tenha sido ultrapassada.

Como forma de propagar as alterações feitas no Redis para todos os utilizadores foi utilizada a biblioteca ActionCable[47], que permite criar canais para transmitir informação através de Web Sockets. Estes canais são criados utilizando o mesmo identificador utilizado no Redis, isto é, uma combinação entre o identificador da sessão e do grupo de lugares. Desta forma, cada vez que um utilizador expande um grupo de lugares para ver os lugares disponíveis numa sessão, é feita uma subscrição do respectivo canal e é recebida uma lista com todos os lugares bloqueados. Posteriormente, são recebidas atualizações, provenientes da plataforma de Gestão de Salas, cada vez que um lugar é bloqueado ou libertado. A subscrição ao canal é mantida enquanto o utilizador permanecer na vista de lugares do mesmo grupo de lugares.

6.3.4 Editor

Nesta subsecção são descritos os objetivos que o editor de salas deve cumprir e é feita uma explicação, técnica e funcional, dos requisitos implementados. Para terminar, é feita uma comparação entre os editores da antiga e da nova plataforma.

Objetivo

O objetivo do editor de salas é construir novos mapas de salas que utilizam a nova estrutura de salas e modelo de dados. Posteriormente, estes mapas podem ser utilizados no *widget* de visualização de salas para que os utilizadores possam interagir com eles nas diversas plataformas.

Descrição do Desenvolvimento

O componente de edição de mapas foi desenvolvido em React[48], devido à maior necessidade de interactividade com o utilizador. Adicionalmente, o autor decidiu que seria benéfico

implementar estilos durante o desenvolvimento deste componente, visto que facilitaria a testagem e validação do código durante o desenvolvimento. Para isso, foi acrescentado o CSS necessário, seguindo o *hand-off* disponibilizado pela equipa de design nos *mockups*.

Na figura 6.16, é apresentado o ecrã inicial do editor, que está dividido em vários componentes:

- uma barra de navegação superior, onde o utilizador pode selecionar operações que deseja executar;
- uma barra lateral esquerda, que apresenta a estrutura hierárquica dos grupos de lugares e organização da sala;
- uma barra lateral direita, que apresenta informações sobre o objeto selecionado e que permite, em alguns casos, realizar alterações a esse objeto;
- uma tela central que apresenta o mapa da sala com o qual o utilizador pode interagir para selecionar, inserir, eliminar ou editar os elementos constituintes da sala.

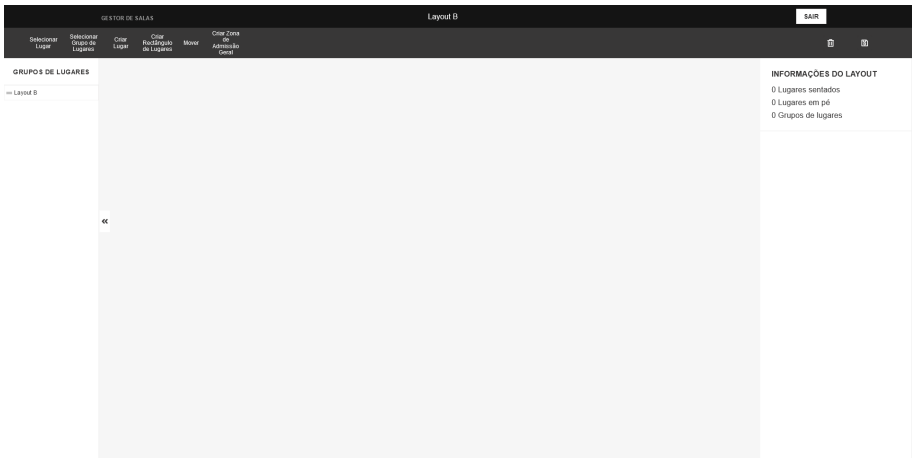


Figura 6.16: Página Inicial do Editor

Adicionalmente, existe ainda uma *Store* onde são guardadas variáveis que necessitam de ser mantidas durante o funcionamento do Editor, tais como a árvore de dados da sala, os objetos selecionados, a operação selecionada e a quantidade de grupos de lugares e lugares da sala. Em adição, este componente contém também funções responsáveis por manipular os dados presentes nestas variáveis.

De seguida, será descrito o funcionamento técnico do editor. Em primeiro lugar, visto que cada SVG tem o seu próprio sistema de coordenadas, é necessário entender como é feita a tradução do clique do rato nas coordenadas do ecrã do cliente (DOM) para o sistema de coordenadas de um SVG específico. Para esse efeito, foi utilizada a função presente na figura 6.17, extraída de um artigo sobre conversão de coordenadas SVG[49].

```
function svgPoint(element, x, y) {

  const pt = svg.createSVGPoint();
  pt.x = x;
  pt.y = y;

  return pt.matrixTransform( element.getScreenCTM().inverse() );

}
```

Figura 6.17: Função de tradução de coordenadas DOM para SVG

O primeiro passo desta função é criar um ponto SVG, utilizando o método *createSVGPoint*, e atribuir as coordenadas do evento a este ponto. De seguida, é aplicada uma transformação matricial, criada a partir da inversa da matriz que mapeia as unidades SVG para as coordenadas do ecrã, que transforma as coordenadas do ponto em coordenadas relativas à *viewbox* do SVG.

Quando o utilizador selecciona a opção de criar um lugar e existe um clique sobre a tela de visualização do mapa, é utilizada a função apresentada acima para obter o ponto em que o lugar deve ser adicionado. Posteriormente, é criado um nó, com todas as informações base de um lugar, que é adicionado à lista de descendentes do objeto selecionado. É também criado um elemento SVG, utilizando as coordenadas recolhidas anteriormente, que é adicionado ao SVG do grupo de lugares onde está inserido. Na notação SVG, um lugar é representado da seguinte forma:

```
<g data-node-type="seat" data-node-id="0">
   <circle cx="" cy="" r="8"></circle>
   <text x="" y=""></text>
</g>
```

Como é possível verificar, um lugar é representado por um círculo e contém um elemento de texto para apresentar a *label* do lugar. Estes elementos são agrupados com a utilização de um elemento g, que possui dois atributos: o tipo do elemento e o seu ID na lista de descendentes do grupo de lugares a que pertence. Na figura 6.18 é possível observar exemplos da visualização de lugares no mapa da sala.

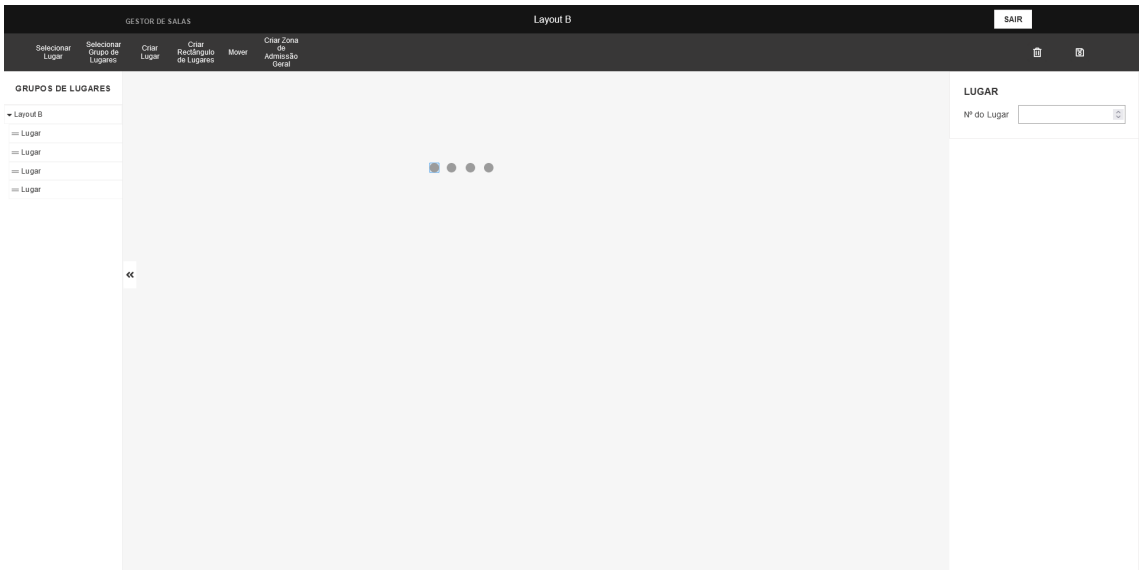


Figura 6.18: Exemplo de adição de lugares

Tal como está ilustrado na figura 6.19, quando um lugar é selecionado é apenas possível editar o seu número. Quando o número é editado, é necessário atualizar esse valor na árvore de dados da sala e também no mapa da sala, para que possa ser visível para o utilizador.

Figura 6.19: Exemplo de barra lateral direita de lugares

Além da adição de lugares de forma singular, é também suportada a adição de retângulos de lugares. Para isso, é necessário definir *EventListeners* para 3 eventos distintos: *mousedown*, *mousemove* e *mouseup*. Quando o utilizador carrega sobre a tela de visualização do mapa é capturado o evento *mousedown*, é recolhido um ponto inicial e iniciado o processo de desenho. De seguida, cada vez que existe um evento *mousemove*, é recolhido um novo ponto e é calculada a distância entre o ponto inicial e o ponto atual. O valor dessa diferença é dividido pela soma do diâmetro e do espaço entre lugares, que devolve o número de linhas e colunas que é possível adicionar. Se algum desses valores for diferente do número atual de linhas e colunas, são adicionadas ou removidas linhas e colunas até que o retângulo tenha as novas dimensões. De seguida, são atualizadas as dimensões do retângulo e o processo repete-se até existir um evento *mouseup*. Quanto isso acontece, atualizam-se as *labels* dos lugares e termina o processo de desenho. Em notação SVG, um rectângulo de lugares é representado da seguinte forma:

```
<g data-node-type="rect-seats" data-node-id="0">
   <g data-node-type="row" data-node-id="0">
        <g data-node-type="seat" data-node-id="0">(...)</g>
        <g data-node-type="seat" data-node-id="1">(...)</g>
        <g data-node-type="seat" data-node-id="2">(...)</g>
        <g data-node-type="seat" data-node-id="3">(...)</g>
   </g>
   <g data-node-type="row" data-node-id="1">(...)</g>
   <g data-node-type="row" data-node-id="2">(...)</g>
</g>
```

Um retângulo de lugares é um grupo de filas que, por sua vez, são um grupo de lugares. Esta divisão em filas, através da utilização de elementos do tipo g, permite que seja possível selecionar filas e realizar operações como mover ou apagar apenas uma fila de lugares e não todo o retângulo.

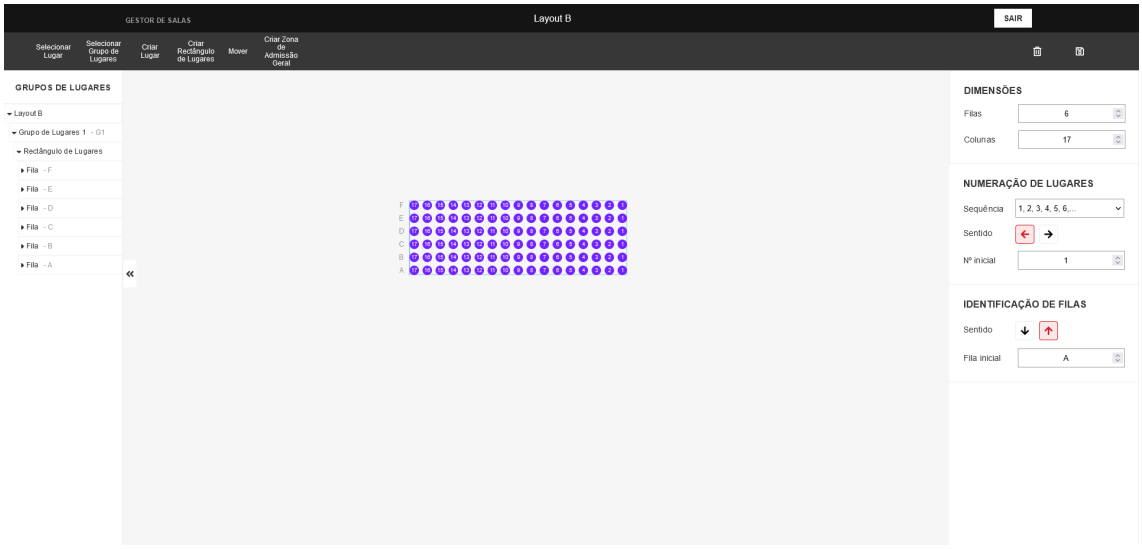


Figura 6.20: Exemplo de adição de rectângulo de lugares

Quando um rectângulo de lugares está selecionado, a barra lateral direita apresenta um estado semelhante ao apresentado na figura 6.21. Em primeiro lugar, é possível visualizar o número de filas e colunas que um rectângulo de lugares possui. Estes valores podem ser alterados, o que provoca modificações na árvore de dados da sala e no elemento SVG apresentado na tela de visualização. Também a barra lateral esquerda é atualizada com a nova estrutura da sala. Adicionalmente, é possível definir algoritmos para a numeração de lugares e filas. No caso dos lugares é possível escolher numeração de forma sequencial, apenas números pares e apenas números ímpares. É também possível definir um sentido e o número do primeiro lugar a ser numerado. No caso das filas, é apenas possível escolher o sentido de *labelling* e a *label* da primeira fila.

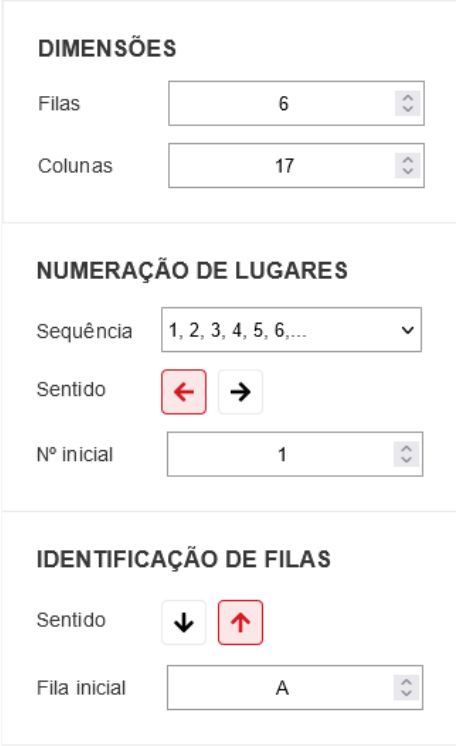


Figura 6.21: Exemplo de barra lateral direita do rectângulo de lugares

Finalmente, outra possibilidade é a adição de zonas sem lugares marcados. O processo de adição deste tipo de lugares é muito semelhante à adição de retângulos de lugares. Ou seja, quando há um evento *mousedown* sobre a tela de apresentação do mapa é capturado um ponto inicial. De seguida, cada vez que é capturado um evento *mousemove* é capturado um novo ponto e é calculada a diferença de altura e largura entre estes dois pontos. Caso as dimensões sejam diferentes, o retângulo que representa uma zona sem lugares marcados é redimensionado. Este processo é repetido até existir um evento *mouseup*, que marca o fim do desenho da nova zona.

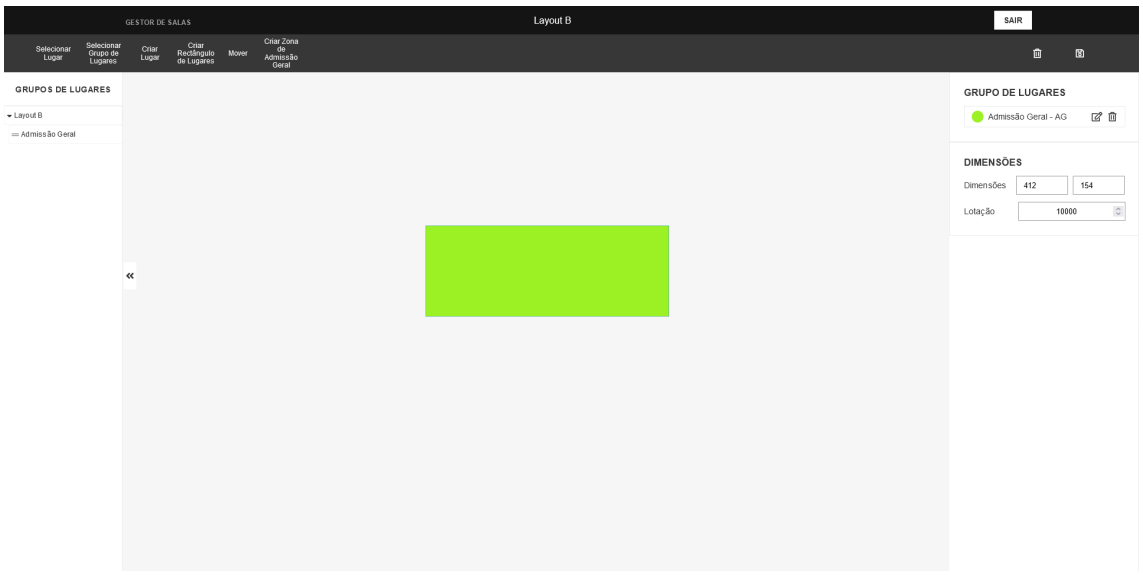


Figura 6.22: Exemplo de adição de zona sem lugares marcados

Quando uma zona sem lugares marcados está selecionada, a barra lateral direita apresenta dois formulários distintos, como é possível observar na figura 6.23. O primeiro formulário apresenta o nome da zona, nome abreviado da zona e também a sua cor. Quando estes valores são alterados, é necessário atualizar a árvore de dados da sala, o elemento na tela de visualização e também a barra lateral esquerda. Em adição, o segundo formulário permite visualizar as dimensões da zona e também a sua lotação. Tal como no formulário anterior, é possível editar esses valores, o que provoca modificações na árvore de dados e na tela de visualização, caso as dimensões sejam alteradas.

Figura 6.23: Exemplo de barra lateral direita de zona sem lugares marcados

Em adição a estas operações de criação e edição de lugares e grupos de lugares, existem também outras que são comuns a vários tipos de elementos, tais como mover, eliminar e gravar.

Para mover elementos no mapa da sala existem duas estratégias diferentes, dependendo do elemento que está selecionado. Em ambos os casos, a estratégia para calcular os valores da translação é semelhante à utilizada para calcular o tamanho de uma zona sem lugares marcados. Se o elemento selecionado for um grupo de lugares, são alterados os valores dos atributos x e y presentes no SVG. No caso do elemento selecionado ser um lugar ou rectângulo de lugares, é utilizado o atributo *transform*, suportado por elementos g. Neste caso, enquanto o utilizador está a mover o lugar ou rectângulo de lugares, o valor da translação é atualizado, o que faz com que todo o conteúdo do elemento g seja movido. Quando o utilizador termina de reposicionar o elemento, ou seja, quando é capturado um evento *mouseup*, são atualizadas as coordenadas de cada elemento que pertence a esse grupo e o atributo *transform* é removido.

No caso de eliminação de elementos, é necessário selecionar um elemento e clicar no botão de eliminar presente na barra de navegação superior. Quando este botão é carregado, o elemento selecionado é removido da árvore de dados da sala, da tela de visualização e também da barra lateral que mostra a estrutura da sala. Contudo, é necessário verificar se os elementos que serão eliminados já estão presentes na base de dados, isto é, se têm

um ID associado. Caso tenham, os elementos não são eliminados mas sim sinalizados como eliminados para que, quando é feita a gravação da árvore, esses elementos possam ser eliminados da base de dados.

Finalmente, para persistir os elementos na base de dados é utilizada uma API que recebe a árvore de dados da sala. A árvore é posteriormente percorrida e os seus nós são guardados, atualizados ou eliminados consoante os seus valores. Assim que todos os elementos estão atualizados, é retornada uma nova árvore atualizada com os IDs da base de dados e sem os elementos que foram eliminados.

6.3.5 Comparações e Conclusão

Terminado o desenvolvimento dos novos componentes, é tempo de aferir se as novas soluções resolvem os problemas identificados durante a análise da antiga plataforma.

Em primeiro lugar, um dos grandes benefícios desta plataforma em relação à sua antecessora é a sua organização em componentes distintos e com funções específicas. Dessa forma, é mais fácil para o utilizador entender os passos necessários para a realização de uma tarefa. Na plataforma antiga, o componente de edição e o *dashboard* funcionavam em conjunto, algo que tornava os menus muito complexos e com muita informação que era complicada de interpretar.

No caso do componente do *dashboard*, além da sua melhor organização, visível ao comparar as figuras 6.24 e 6.25. Existem também novas funcionalidades que facilitam as tarefas dos utilizadores. Por exemplo, a funcionalidade de duplicar mapas não era suportada na plataforma antiga, o que implicava que quando era necessário fazer a mínima alteração era necessário recriar a sala de novo. Na nova plataforma, o *dashboard* permite que esta tarefa seja realizada apenas com o preenchimento de um formulário. Adicionalmente, a funcionalidade de recuperação de *layouts* eliminados permite dar maior segurança ao utilizador, pois caso elimine um *layout* por acidente tem sempre a hipótese de reverter a sua ação. Na plataforma antiga, caso um mapa fosse eliminado por engano, era necessário recriar esse mapa desde o início. Finalmente, o facto de existir uma listagem de *layouts* que o utilizador modificou recentemente faz com que não seja necessário realizar várias pesquisas para retomar o seu trabalho, o que melhora a experiência do utilizador.

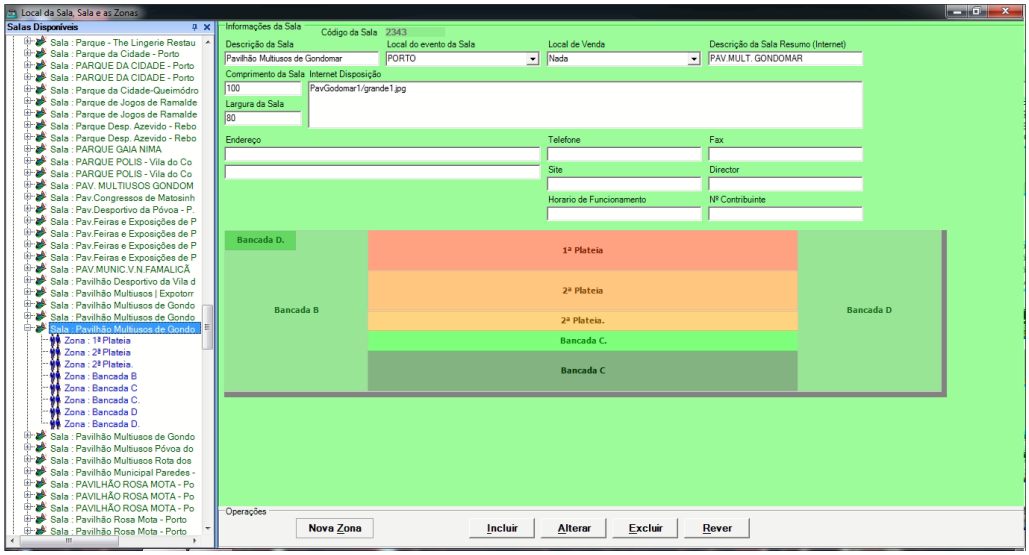


Figura 6.24: Exemplo da página da sala do *dashboard* antigo

Figura 6.25: Exemplo da página da sala no novo *dashboard*

Relativamente ao *widget*, é possível assinalar dois aspectos onde existiriam melhorias significativas. Primeiramente, os lugares deixaram de ser apresentados com um modal, figura 6.26, e passaram a ser mostrados diretamente no mapa, figura 6.27. Assim é possível perceber a localização dos lugares relativamente ao palco, algo que é essencial para avaliar a qualidade da localização do lugar. Isto faz com que seja possível entender melhor a planta da sala e a localização das diferentes zonas. Para além disso, a nova solução apresenta a disponibilidade dos lugares em tempo real. Dessa forma, o utilizador pode ter a garantia que um lugar que selecionou não será reservado por outro cliente até que o pagamento esteja concluído.

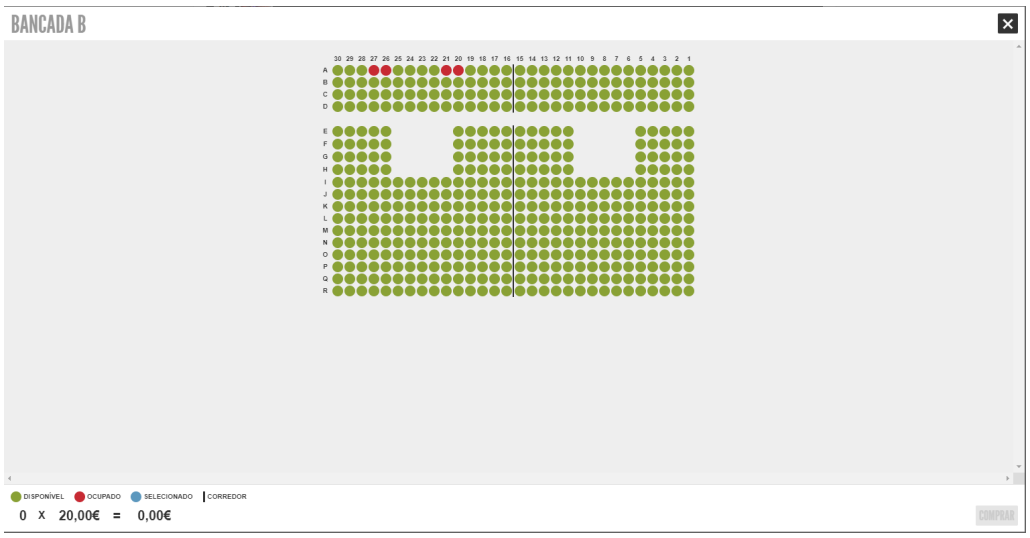


Figura 6.26: Exemplo de lugares apresentados no antigo widget

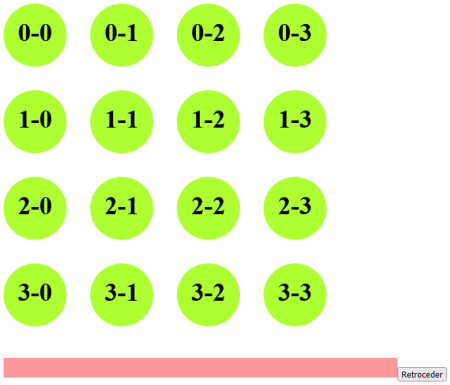


Figura 6.27: Exemplo de lugares apresentados no novo widget

Finalmente, em relação ao componente de edição, é possível identificar muitas melhorias. Como é possível observar ao comparar as figuras 6.28 e 6.29, a grande diferença no novo editor é a criação de zonas ser totalmente flexível e não limitada por uma estrutura de tabela fixa. Devido a esta diferença, foi possível resolver muitos dos problemas identificados no antigo editor. Em primeiro lugar, foi possível aumentar a interatividade da plataforma ao permitir que as salas passassem a ser criadas através de interações com a tela de visualização do mapa. Assim, o utilizador passa a ter maior liberdade para posicionar cada zona e tornar o mapa da sala mais próximo da realidade e criar uma melhor experiência de compra para o utilizador, que percebe exatamente onde se localiza o seu lugar. Adicionalmente, o novo editor apresenta menus mais simples e específicos aos elementos que estão selecionados, algo que facilita a aprendizagem do utilizador.

Em termos funcionais, o novo editor é também muito mais completo que o da antiga plataforma. Por exemplo, agora existe um maior número de elementos que se podem adicionar, sendo eles: lugares individuais, zonas sem lugares marcados, retângulos de lugares e palcos. Adicionalmente, é possível realizar várias novas operações sobre estes elementos, como reposicionar os elementos para que possam ficar na localização ideal. É também possível aplicar rotações aos grupos de lugares para que possam ficar orientados para o palco, se assim for o caso.

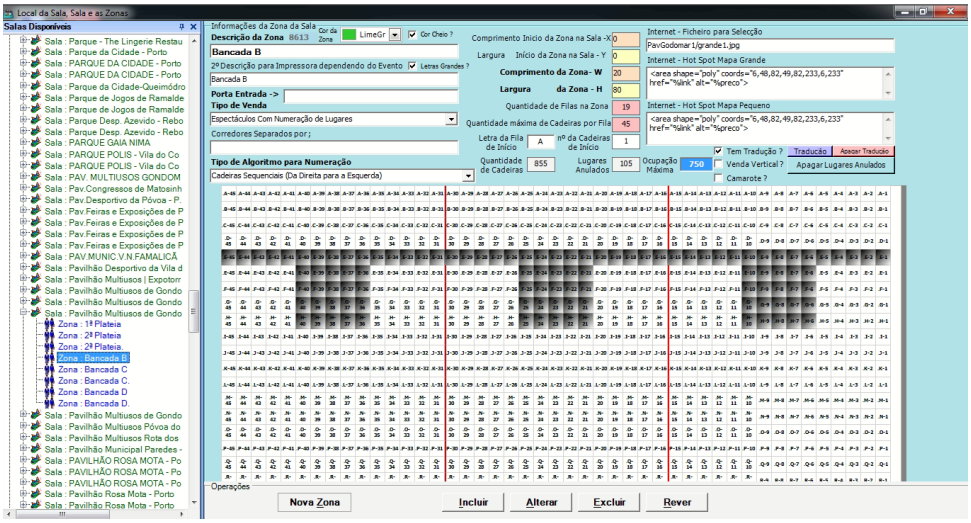


Figura 6.28: Exemplo da criação de uma sala no editor antigo

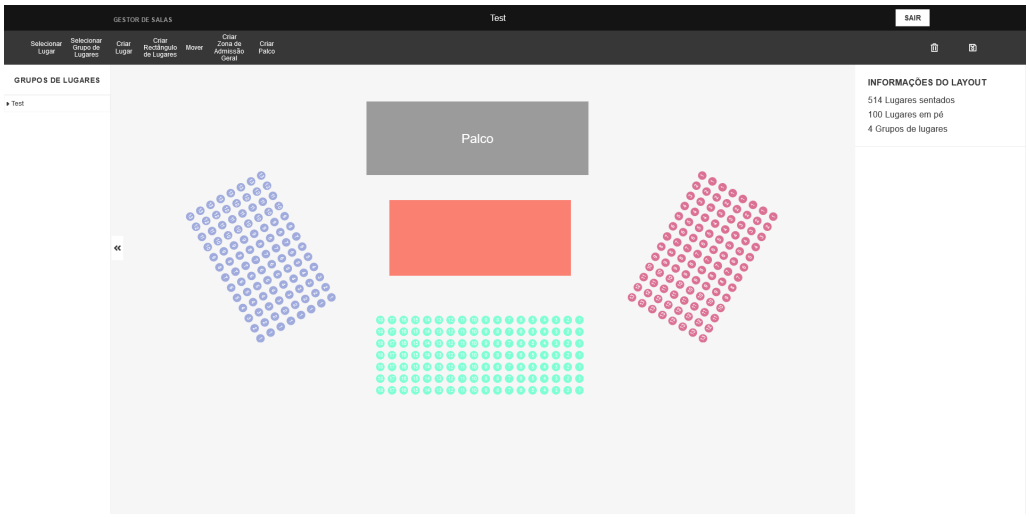


Figura 6.29: Exemplo da criação de uma sala no novo editor

Em suma, a nova plataforma oferece maior organização e um maior número de funcionalidades aos utilizadores. Para além disso, resolve muitos problemas que foram identificados na análise à antiga plataforma e que se tornavam obstáculos para os utilizadores. Assim sendo, e apesar de existir espaço para melhorias e novas funcionalidades, é possível concluir que todos os componentes da plataforma cumprem o seu objetivo e tornam esta plataforma, em termos funcionais, superior à plataforma antiga.

Capítulo 7

Testes de Sistema

Este capítulo descreve os testes funcionais realizados na plataforma de gestão de salas.

7.1 Plano de Testes

O plano de testes para a Plataforma de Gestão de Salas consistiu em realizar testes funcionais, de unidade e de integração, aos vários componentes da plataforma. No caso do *Dashboard* e Tradutor, o objetivo era testar todos os métodos que tiverem sido desenvolvidos na linguagem Ruby. Nos casos do Editor e *Widget*, como são maioritariamente desenvolvidos em javascript, o plano era testar as APIs que suportam estes componentes.

7.2 *Unit Testing*

Unit Testing é a testagem individual de módulos para garantir que os resultados que produzem são corretos. Estes testes foram realizados manualmente pelo autor durante a fase de desenvolvimento de cada funcionalidade e, posteriormente, através de testes automáticos na fase de validação da plataforma.

Os testes automáticos foram realizados com a utilização da ferramenta RSpec[50] que permite criar casos de testes e executar várias verificações para garantir a validação do resultado do teste.

A técnica escolhida para testar a aplicação foi *statement coverage*, que é uma técnica de *white box testing* que implica a execução de todas as linhas do código pelo menos uma vez. O objetivo desta técnica é cobrir todos os caminhos possíveis do código e garantir que todas as linhas sejam testadas.

Para aplicar esta técnica, foram criados, manualmente, casos de teste para cada função. Posteriormente, os testes foram executados e avaliados de forma automática com a ferramenta RSpec[50].

Para avaliar a cobertura dos testes unitários foi utilizada a ferramenta de análise SimpleCov[51]. Na figura 7.1, é possível verificar que o projeto teve uma cobertura de 87.74%, que, apesar de não chegar ao objetivo de 100% de cobertura da técnica de *statement coverage*, é um resultado que pode ser considerado satisfatório.

Figura 7.1: Cobertura do código depois de correr todos os casos de teste

As razões que não permitiram uma cobertura total devem-se ao facto da fase de testes ter coincidido com algumas revisões de código que revelaram a necessidade de correções. O autor preferiu dar prioridade a estas correções pois entendeu que não seria benéfico testar código que teria de ser alterado. Por essa razão, não foi possível cumprir todos os testes planeados em tempo útil.

7.3 *Integration Testing*

Integration Testing é a combinação e testagem de diferentes módulos para garantir que funcionam corretamente quando interagem entre si. Esta fase ocorre depois dos módulos serem testados individualmente.

No caso da plataforma de gestão de salas, foram realizados testes de integração manuais quando existia interação com outras plataformas da Ticketline. Em primeiro lugar, foi testada a importação de um mapa da antiga plataforma de gestão de salas e, posteriormente, a sua apresentação no *widget* de visualização de salas.

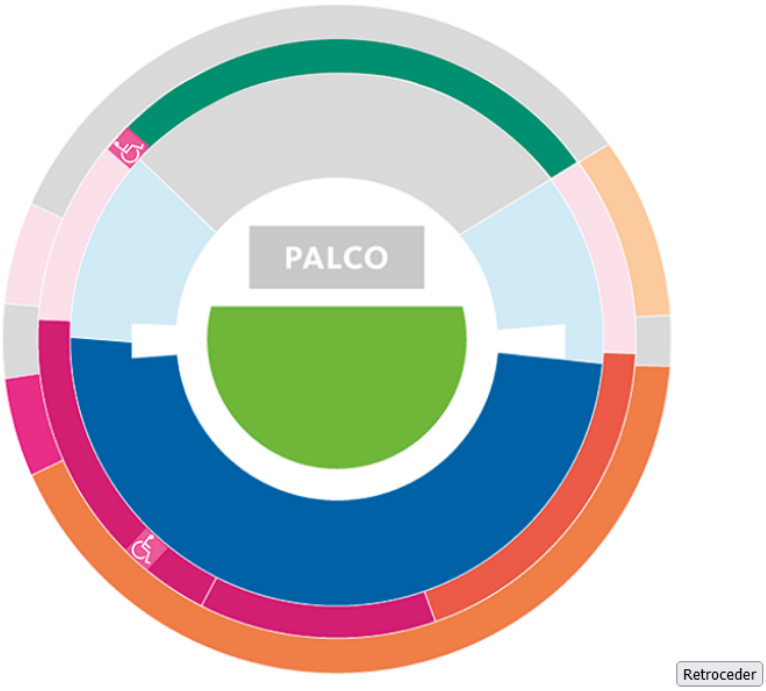


Figura 7.2: Exemplo da representação de uma sala importada no *widget*

Em adição, foi também testada a inclusão do *widget* noutras plataformas da Ticketline. Para realizar esse teste, o *widget* foi incluído no novo website da Ticketline como é possível verificar na figura 7.3.

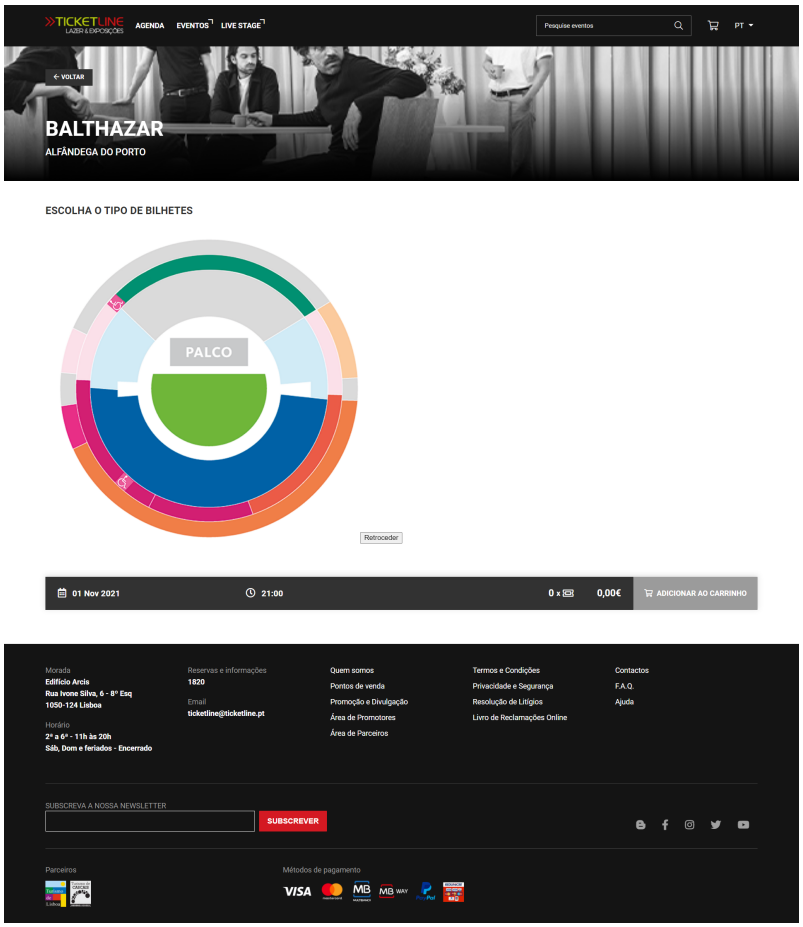


Figura 7.3: Exemplo do *widget* integrado no website da Ticketline

Finalmente, como a autenticação na plataforma de gestão de salas é realizada por uma API da Ticketline, o último teste de integração consistiu em tentar autenticar um utilizador na plataforma ao utilizar credenciais válidas e inválidas.

7.4 Conclusão

A realização dos testes, quer através de revisões ou de forma automatizada, revelou a necessidade de realizar algumas correções no código. Essas correções foram realizadas para tornar as plataformas mais robustas e menos susceptíveis a falhas quando utilizadas pelos utilizadores.

Com a realização dos testes de integração foi possível concluir que as plataformas desenvolvidas podem ser integradas nos sistemas Ticketline, interagindo corretamente com as APIs internas e também com outras plataformas.

Capítulo 8

Conclusão

A possibilidade de ser integrado numa equipa e trabalhar num projeto de desenvolvimento de software foi uma experiência muito valiosa pois permitiu melhorar competências e consolidar as matérias exploradas durante as unidades curriculares do Mestrado em Engenharia Informática.

Este projeto iniciou-se com uma fase de planeamento onde foi escolhida uma metodologia de desenvolvimento de software e foram definidas as tarefas a realizar durante o semestre de acordo com a metodologia.

Seguiu-se uma fase de levantamento do estado da arte onde foi analisada a Plataforma de Gestão de Salas utilizada atualmente como forma de identificar as limitações e problemas que os utilizadores da plataforma enfrentam atualmente. Adicionalmente, foram analisadas ferramentas de criação de mapas de salas para entender se poderia ser utilizada uma destas ferramentas para realizar a criação dos mapas de salas e a sua integração noutras plataformas. Finalmente, foram analisados e comparados Web Services para decidir o tipo de Web Service que iria ser utilizado no desenvolvimento do projeto.

Na fase seguinte foi feita a identificação e análise dos requisitos funcionais e atributos de qualidade após reuniões com o cliente. Nesta fase foram também identificadas as restrições impostas pela The Loop Company para o desenvolvimento deste projeto. A presença de um cliente real foi um desafio pois implicou não só interpretar os desejos do cliente como também gerir as suas expectativas para garantir que a dimensão e complexidade do projeto se adequavam ao tempo disponível.

Para concluir o primeiro semestre, foi feita uma descrição do sistema para entender como os requisitos iriam ser cumpridos. A descrição foi feita através da representação do modelo de dados num diagrama entidade-relacionamento, da criação dos diagramas de arquitetura do sistema, da descrição da organização das páginas com o auxílio de wireframes e da criação de um diagrama de navegação para entender o fluxo da aplicação.

Durante o primeiro semestre, os grandes desafios foram conseguir identificar corretamente os requisitos explicados pelo cliente. Adicionalmente, existiram algumas dificuldades em contribuir com sugestões nas reuniões para definir a arquitetura e funcionamento do sistema.

No segundo semestre iniciou-se a fase de desenvolvimento dos vários componentes que compõem a plataforma. Para cada componente, foram implementados os requisitos de maior prioridade, cumprindo assim o objetivo de desenvolver um Produto Viável Mínimo durante este projeto. Esta fase de desenvolvimento foi, sem dúvida, a mais desafiante pois

implicou o contacto com novas tecnologias e também uma equipa de desenvolvimento. Em especial, desenvolver o componente de edição de salas foi uma grande experiência pois a realização de muita pesquisa de tecnologias com que o autor não estava muito familiarizado.

Finalmente, na fase de validação do sistema foram realizados testes funcionais à plataforma que permitiram identificar alguns problemas e evitar que esses erros ocorram quando a plataforma for entregue ao cliente.

O principal foco do trabalho futuro neste projeto deve ser o cumprimento dos restantes requisitos identificados para enriquecer o valor desta plataforma. Por exemplo, no editor de salas não foram implementados muitos requisitos 'Should Have', como a adição de novos formatos de zona e adição de mesas, que podem trazer maior valor para o cliente. A adição de estilos para as várias páginas do Dashboard deve também ser uma prioridade. Finalmente, devem também ser completados os testes funcionais para cumprir o objetivo de *statement coverage* e devem ser realizados testes não funcionais, assim que a plataforma estiver pronta para ser entregue ao cliente.

O estágio foi muito positivo para o autor pois permitiu experienciar como é trabalhar num grande projeto de engenharia de software. A possibilidade de fazer parte deste projeto desde o seu início fez com que o autor adquirisse novos conhecimentos e tivesse a experiência de contactar com um cliente real. Finalmente, a oportunidade de estar integrado numa equipa de desenvolvimento de software fez com que o autor melhorasse competências essenciais para um engenheiro de software.

Referências

[1] Nayan Ruparelia. Software development lifecycle models. *ACM SIGSOFT Software Engineering Notes*, 2010.

[2] Joseph Spinelli. Mvc overview. https://medium.com/@joespinelli_6190/mvc-model-view-controller-ef878e2fd6f5.

[3] Maria Kobelyatskaya. Group of Seats (GoS) as the universal standard in seating data storage. https://seatmap.pro/blog/group-of-seats-standard/.

[4] Ruby on rails. https://rubyonrails.org/.

[5] Getting started with rails. https://guides.rubyonrails.org/getting_started.html.

[6] Active record basics. https://guides.rubyonrails.org/active_record_basics.html.

[7] Martin Fowler. P of EAA: Active Record. https://www.martinfowler.com/eaaCatalog/activeRecord.html.

[8] Action view overview. https://guides.rubyonrails.org/action_view_overview.html.

[9] Action controller overview. https://guides.rubyonrails.org/action_controller_overview.html.

[10] RubyGems.org | O host de gems da sua comunidade. https://rubygems.org/.

[11] Introduction to HTML. https://www.w3schools.com/html/html_intro.asp.

[12] CSS Introduction. https://www.w3schools.com/css/css_intro.asp.

[13] O que é JavaScript? | MDN. https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript/First_steps/O_que_e_JavaScript.

[14] What is a DBMS? Definition and FAQs. https://www.omnisci.com/technical-glossary/dbms/.

[15] Postgresql: The world's most advanced open source database. https://www.postgresql.org/.

[16] The mind-blowing benefits of using an interactive seat map. https://seatedly.com/interactive-seat-map/.

[17] Softjourn. https://softjourn.com/about-us.

[18] Softjourn venue mapping tool. https://softjourn.com/expertise/social-distancing-checkerboard-seating-algorithm.

[19] Seatmap.pro. https://seatmap.pro/.

[20] Seats.io. https://www.seats.io.

[21] Seatics maps. https://seatics.com/.

[22] Seats.io - features. https://www.seats.io/features#designer.

[23] Features - seatmap.pro. https://seatmap.pro/docs/.

[24] Venue mapping and reserved seating software - softjourn, inc. https://softjourn.com/blog/article/venue-mapping-and-seat-selection-tool.

[25] Seats.io - documentation. https://docs.seats.io/docs.

[26] Seatmap.pro integration guide. https://seatmap.pro/features.html.

[27] What is MoSCoW Prioritization? Overview of the MoSCoW Method. https://www.productplan.com/glossary/moscow-prioritization/.

[28] Max Rehkopf. User Stories | Examples and Template. https://www.atlassian.com/agile/project-management/user-stories.

[29] Jakob Nielsen. Usability 101: Introduction to usability. https://www.nngroup.com/articles/usability-101-introduction-to-usability/.

[30] Jakob Nielsen. Response times: The 3 important limits. https://www.nngroup.com/articles/response-times-3-important-limits/.

[31] The c4 model for visualising software architecture. https://c4model.com/.

[32] Peldi Guilizzoni. What Are Wireframes? | Wireframing Academy | Balsamiq. https://balsamiq.com/learn/articles/what-are-wireframes/.

[33] ClickUp. https://www.clickup.com/.

[34] Rubocop. https://rubocop.org/.

[35] Brakeman. https://brakemanscanner.org/.

[36] Bundle-Audit. https://github.com/rubysec/bundler-audit.

[37] Yarn Audit. https://classic.yarnpkg.com/en/docs/cli/audit/.

[38] Yarn. https://yarnpkg.com/.

[39] Warden. https://github.com/wardencommunity/warden.

[40] Sidekiq. https://sidekiq.org/.

[41] PgSearch. https://github.com/Casecommons/pg_search.

[42] PostgreSQL: Documentation: 9.1: fuzzystrmatch. https://www.postgresql.org/docs/9.1/fuzzystrmatch.html.

[43] PostgreSQL: Documentation: 9.6: pg_trgm. https://www.postgresql.org/docs/9.6/pgtrgm.html.

[44] Kaminari. https://github.com/kaminari/kaminari.

[45] Redis. https://redis.io/.

[46] The Programming Language of Lua. https://www.lua.org/.

[47] Action Cable Overview - Ruby on Rails Guides. https://guides.rubyonrails.org/action_cable_overview.html.

[48] React. https://reactjs.org/.

[49] Craig Buckler. How to Translate from DOM to SVG Coordinates and Back Again. https://www.sitepoint.com/how-to-translate-from-dom-to-svg-coordinates-and-back-again/.

[50] RSpec: Behaviour Driven Development for Ruby. https://rspec.info/.

[51] Simplecov. https://github.com/simplecov-ruby/simplecov.

[52] Ticketline - quem somos. https://ticketline.sapo.pt/pagina/quemsomos.

[53] Chuks Opia. How to set up a ruby on rails project with a react frontend. https://www.digitalocean.com/community/tutorials/how-to-set-up-a-ruby-on-rails-project-with-a-react-frontend.

[54] Paulo Merson Liam O'Brien, Len Bass. Quality Attributes and Service-Oriented Architectures. Technical report, Software Engineering Institute, Carnegie Mellon, 2005.

[55] Jakob Nielsen. Success rate: The simplest usability metric. https://www.nngroup.com/articles/success-rate-the-simplest-usability-metric/.

[56] SOAP Vs. REST: Difference between Web API Services.

[57] Understanding SOAP vs REST: Basics and Differences. https://smartbear.com/blog/test-and-monitor/soap-vs-rest-whats-the-difference/.

[58] GitLab.

[59] HTML map tag. https://www.w3schools.com/tags/tag_map.asp.

[60] HTML area tag. https://www.w3schools.com/tags/tag_area.asp.

Apêndices

Esta página é intencionalmente deixada em branco.

Apêndice A - Diagramas de Gantt

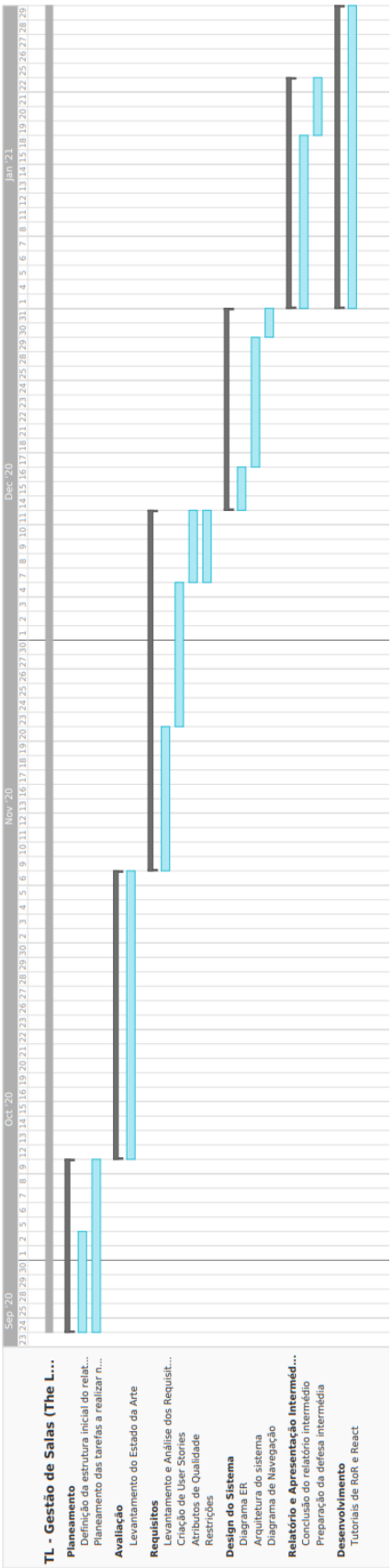


Figura 1: Diagrama de Gantt do plano do primeiro semestre

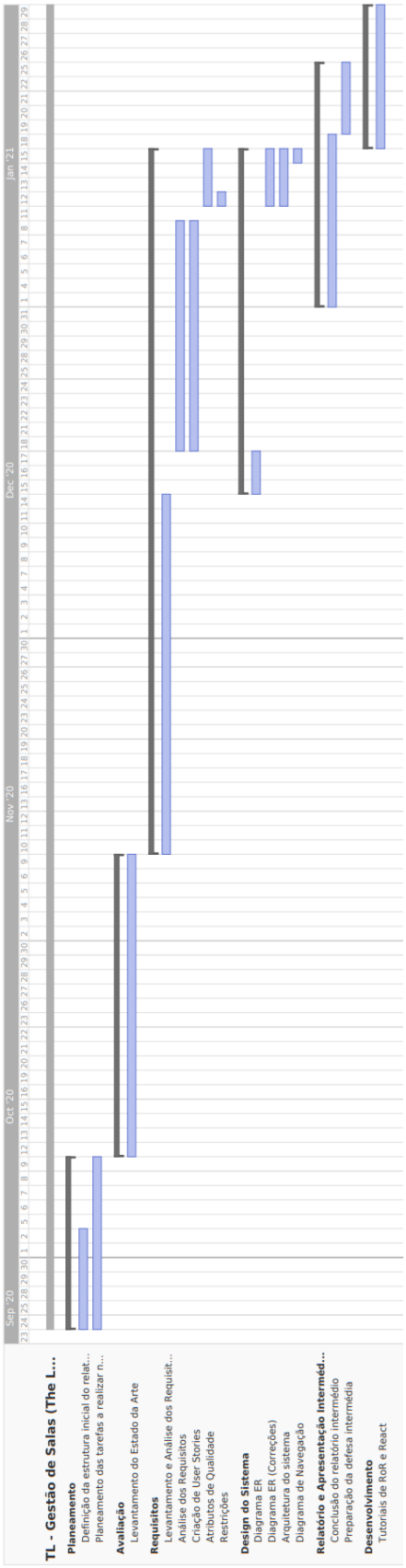


Figura 2: Diagrama de Gantt do resultado do primeiro semestre

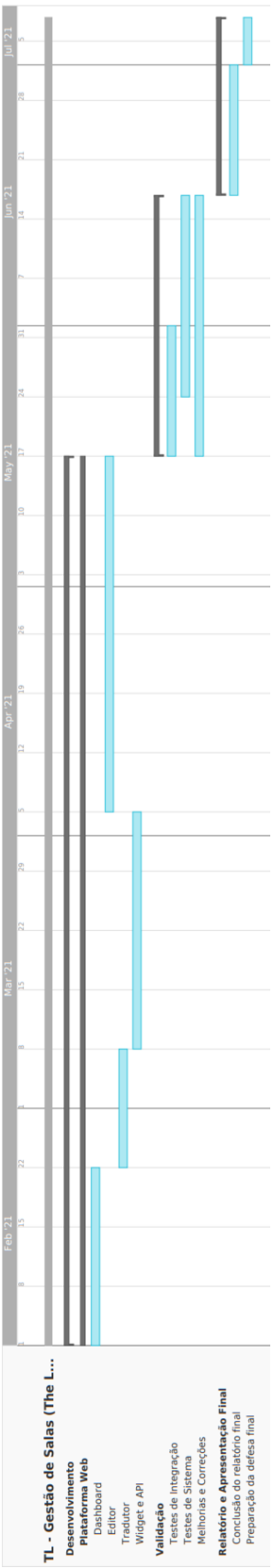


Figura 3: Diagrama de Gantt do plano do segundo semestre

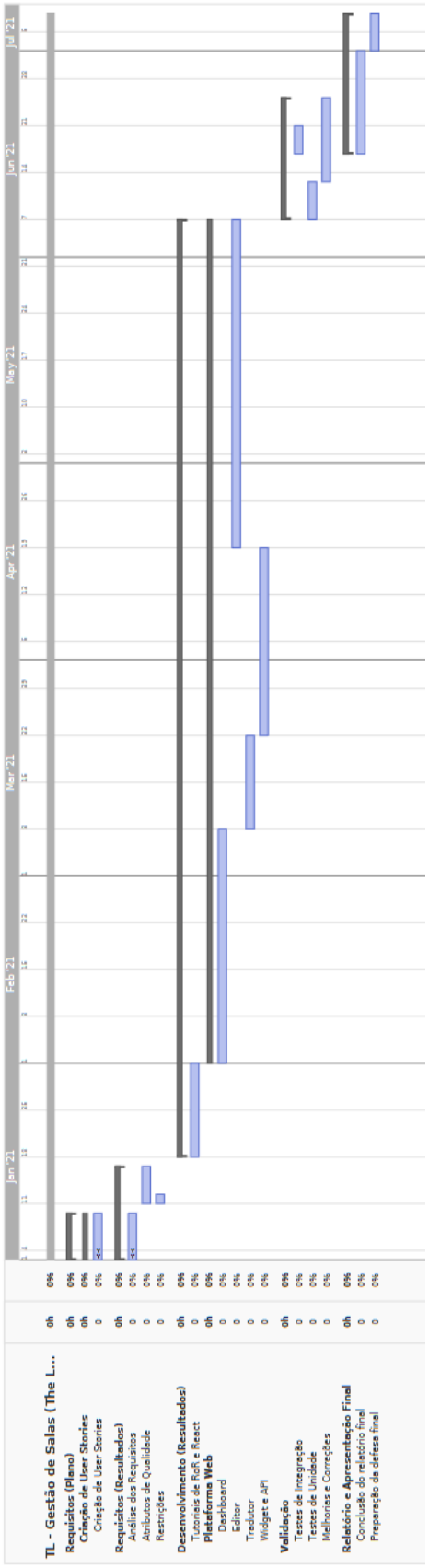


Figura 4: Diagrama de Gantt do resultado do segundo semestre

Esta página é intencionalmente deixada em branco.

Apêndice B - Arquitetura do Sistema

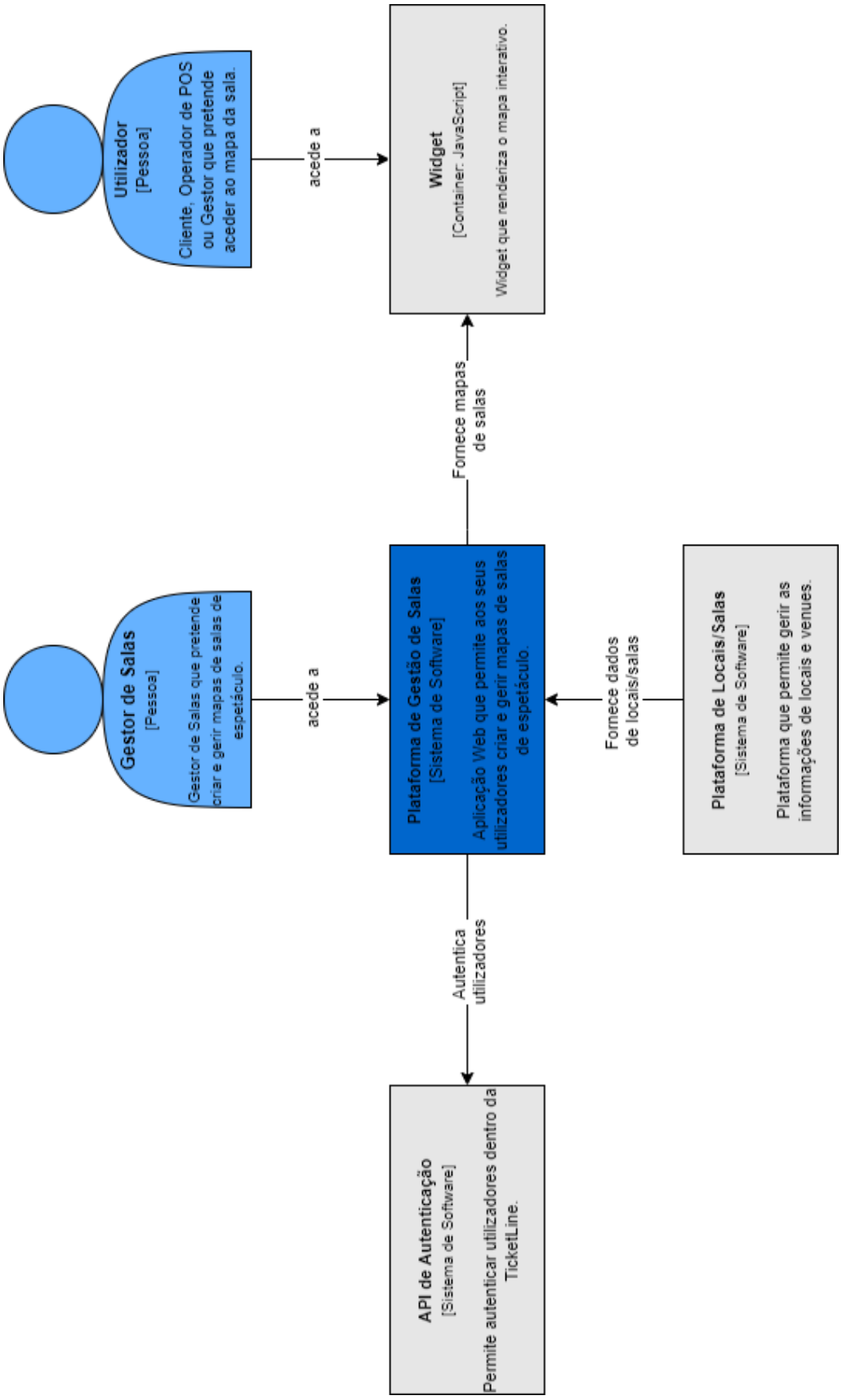


Figura 5: Diagrama de Contexto do Sistema

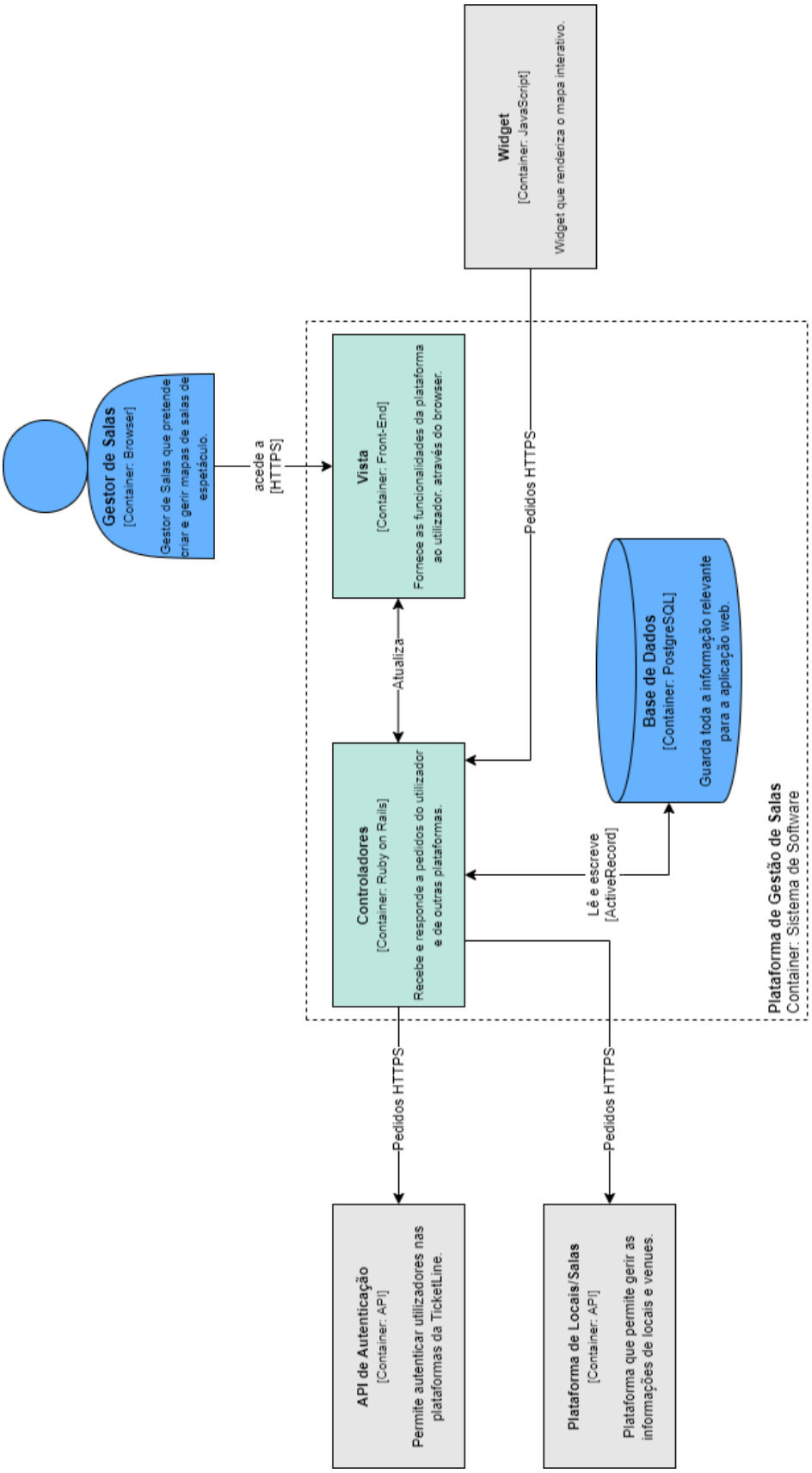


Figura 6: Diagrama de *Containers* do Sistema

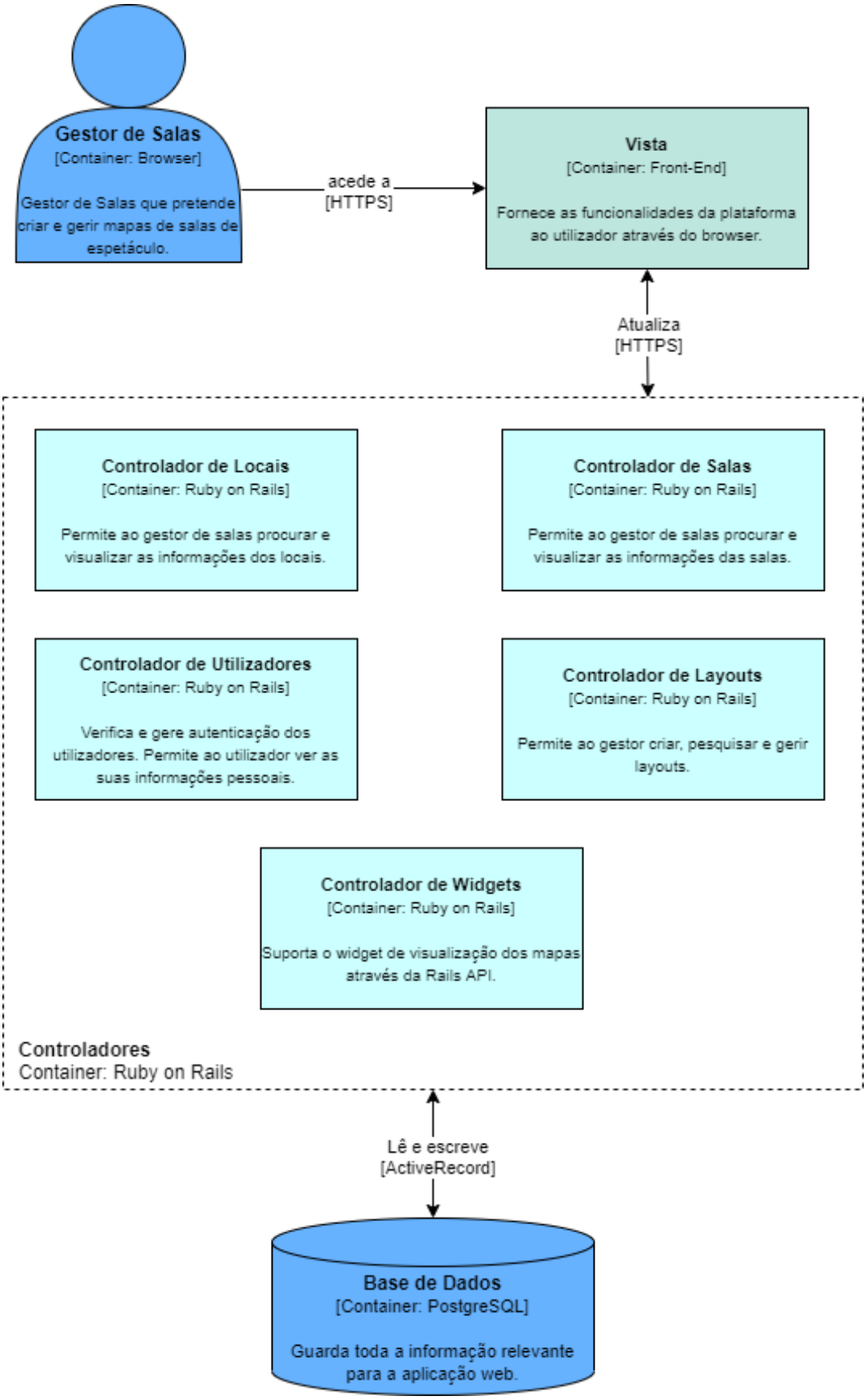


Figura 7: Diagrama de Componentes do Sistema - Controladores

Apêndice C - Diagrama Entidade-Relacionamento

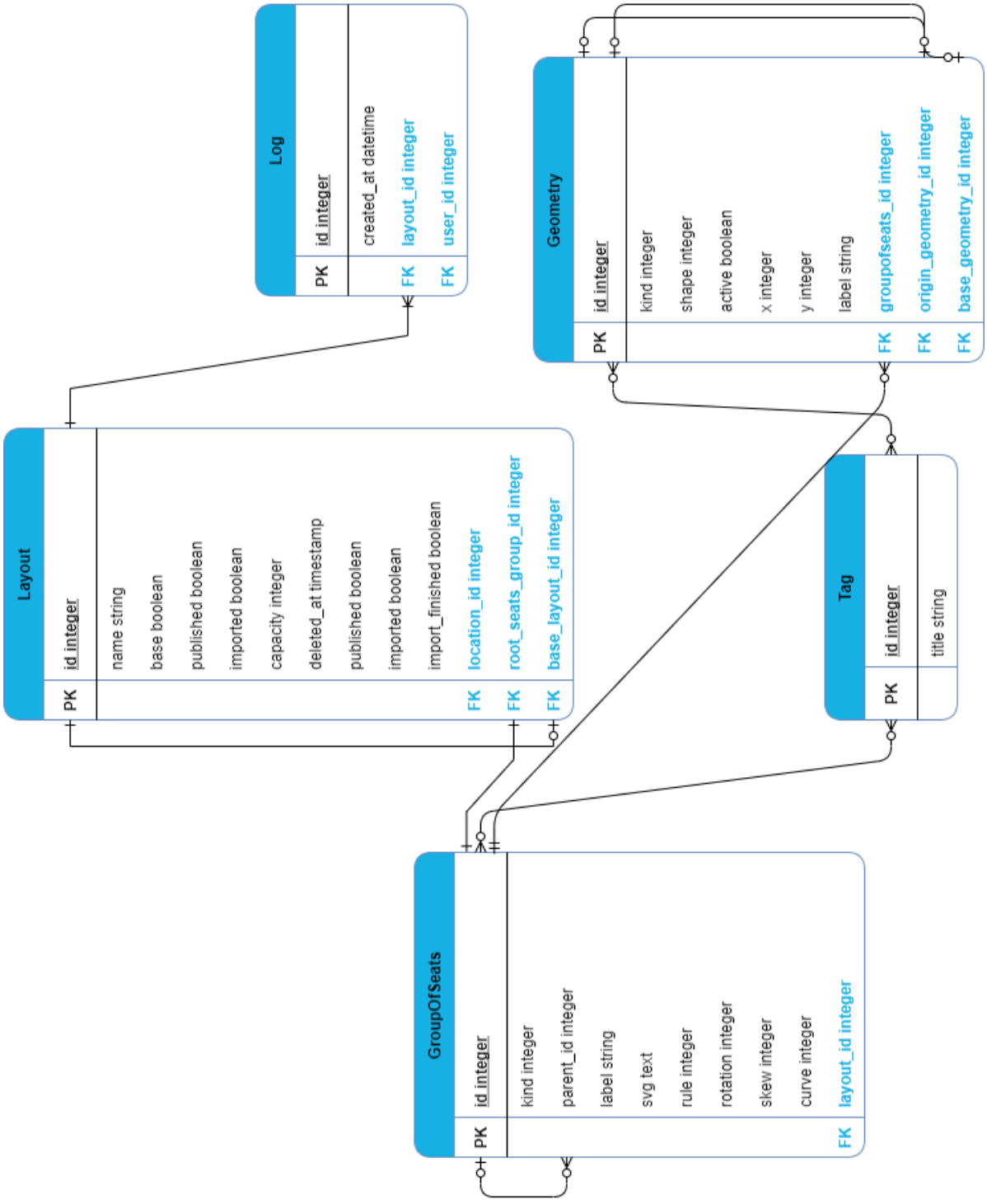


Figura 8: Diagrama de Entidade-Relacionamento

Apêndice D - *Wireframes*

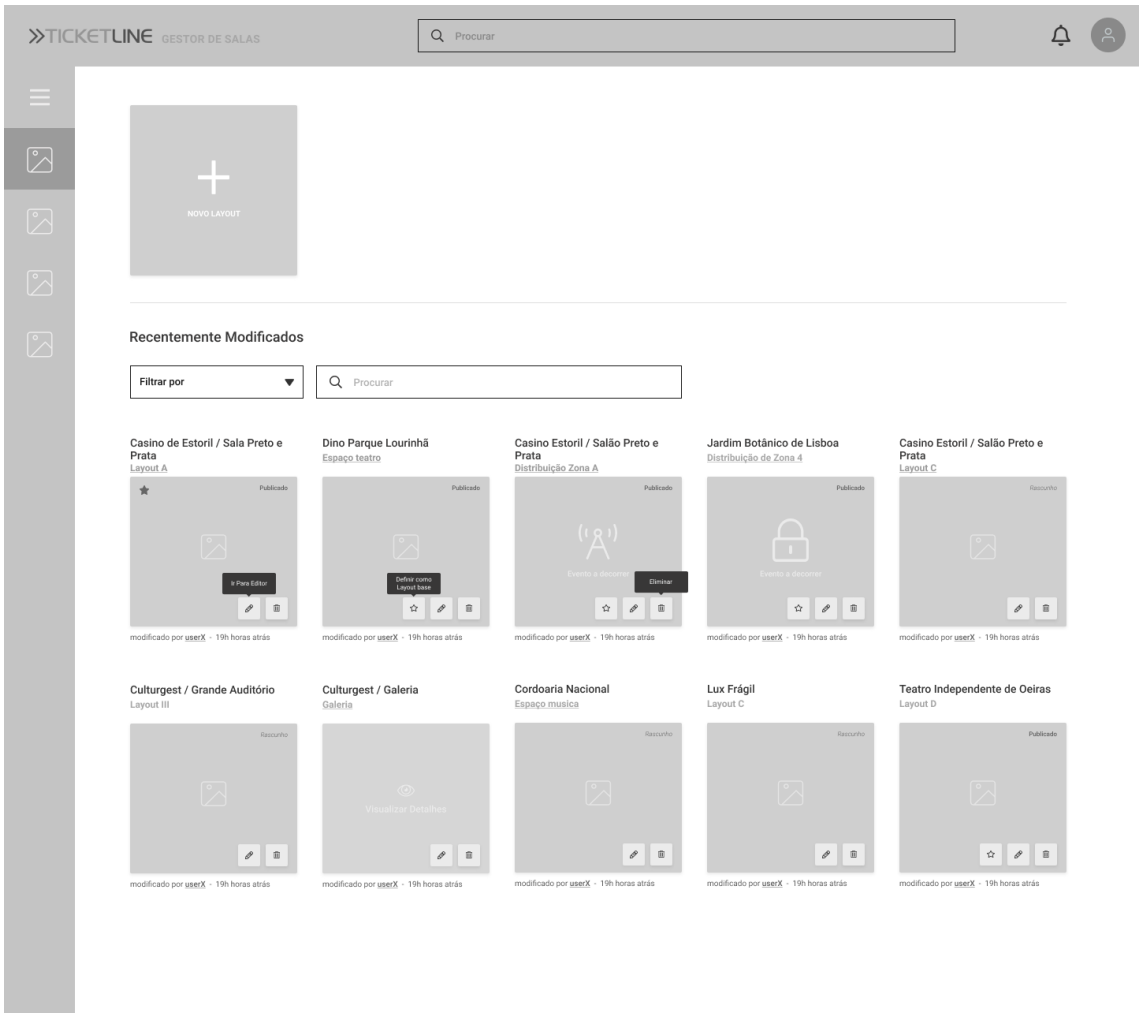


Figura 9: Wireframe: Layouts Recentemente Modificados

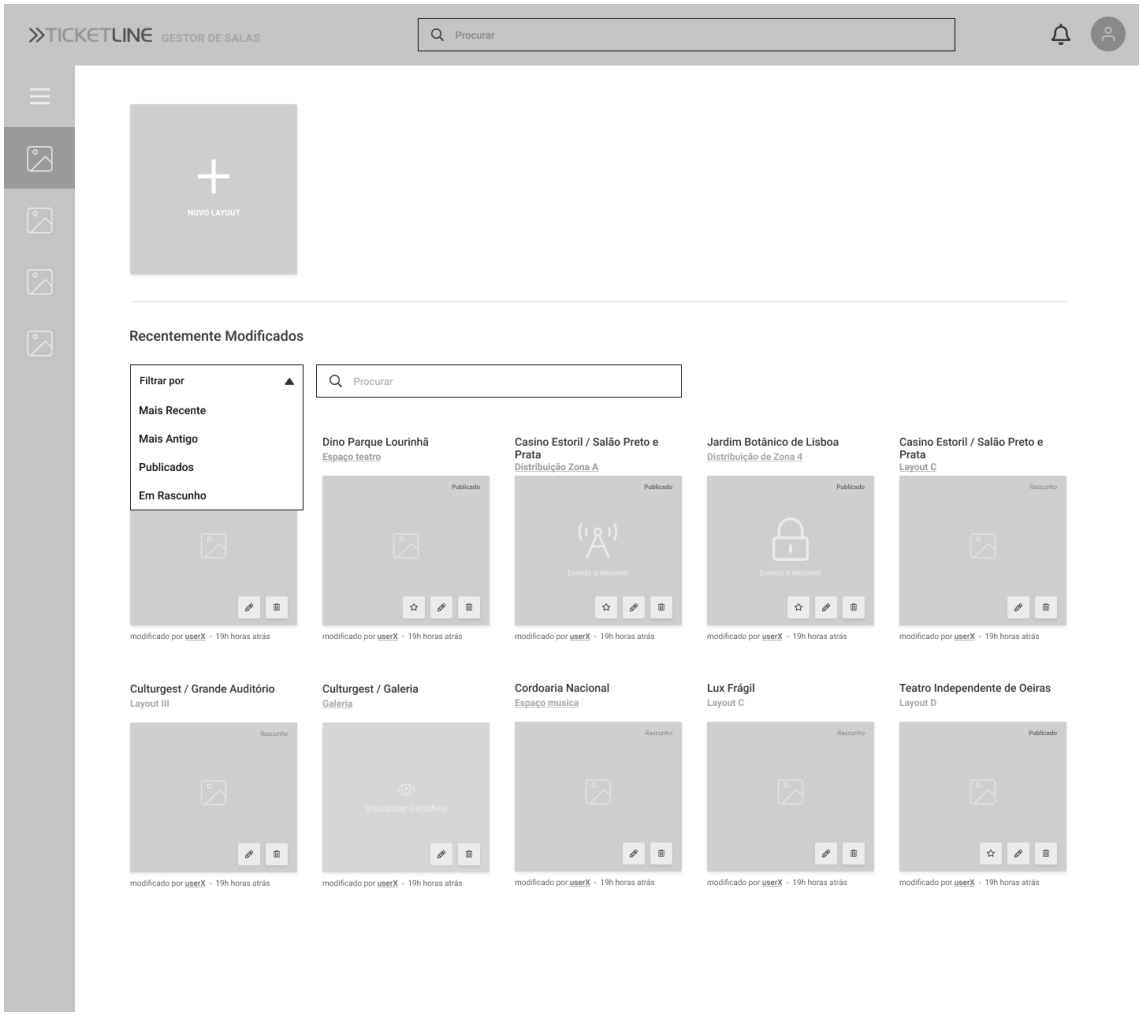


Figura 10: Wireframe: Filtro de Layouts Recentemente Modificados

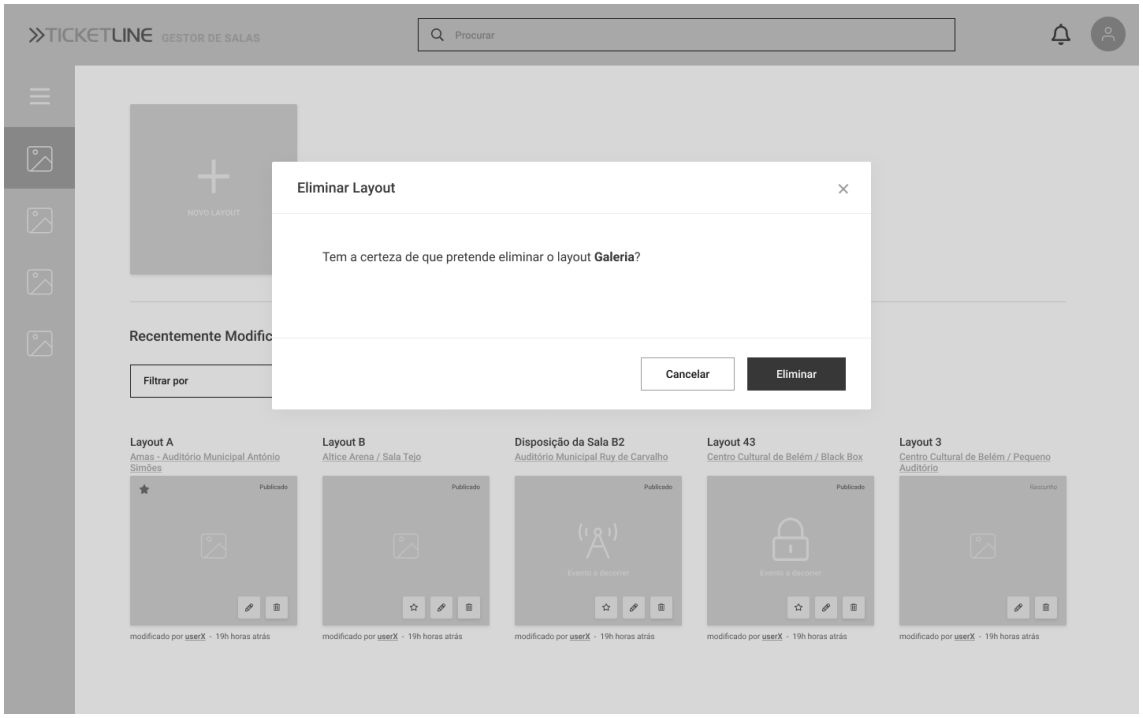


Figura 11: Wireframe: Eliminar Layout

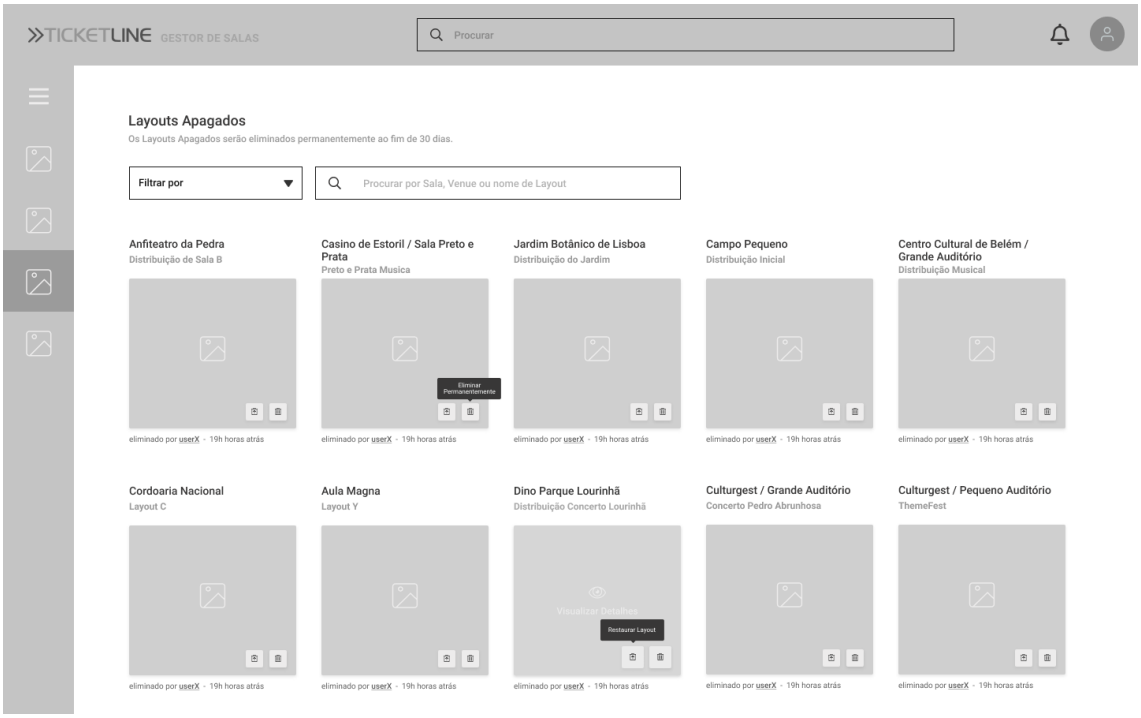


Figura 12: Wireframe: Layouts Apagados

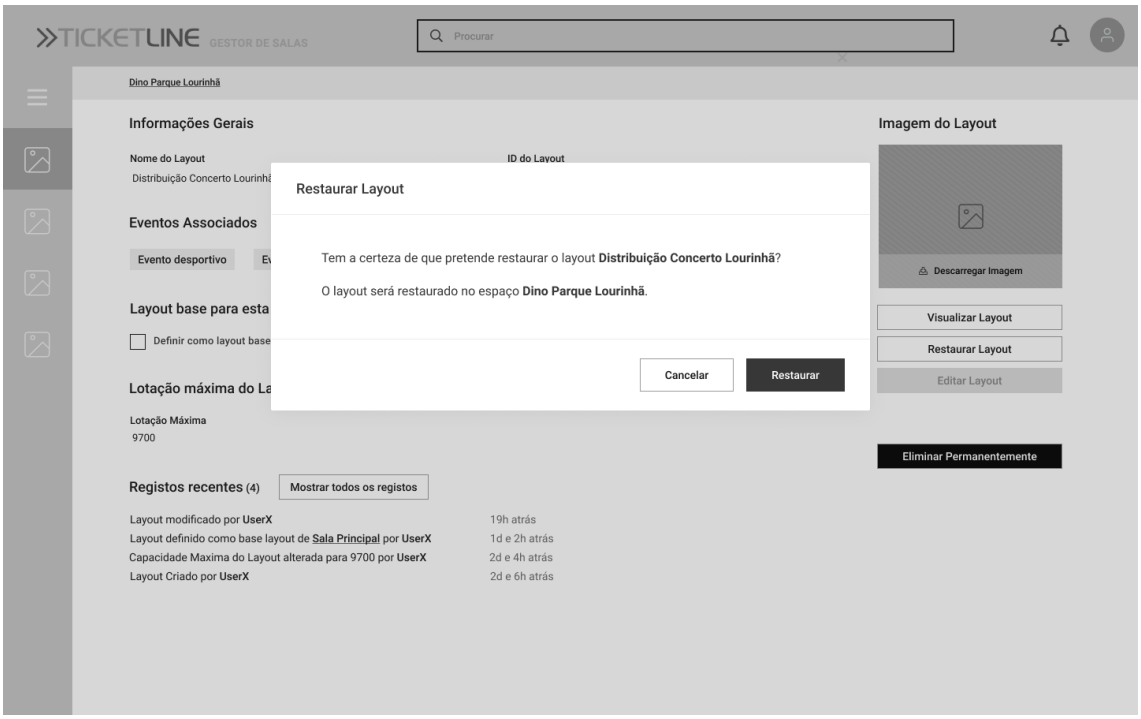


Figura 13: Wireframe: Restaurar Layout

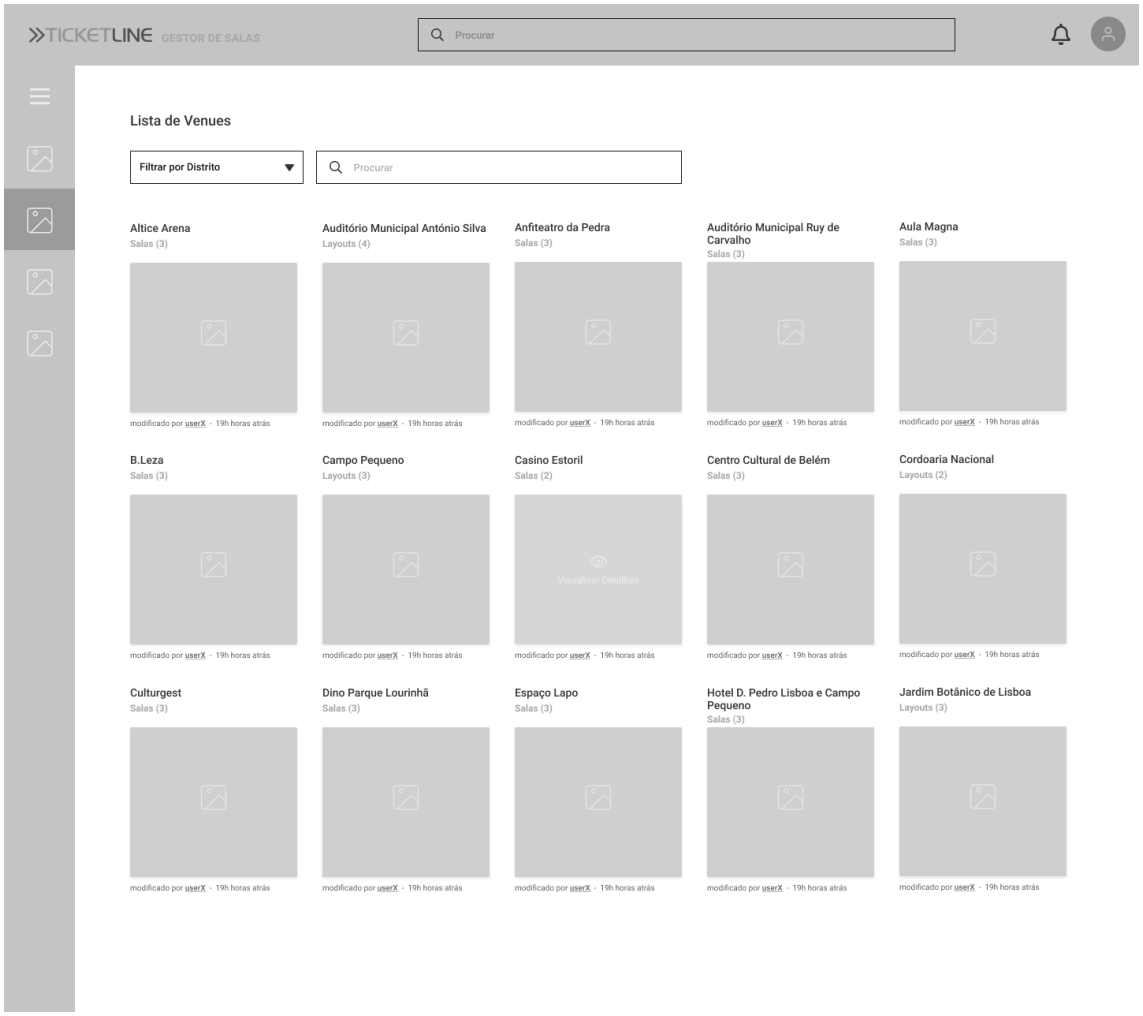


Figura 14: Wireframe: Lista de Locais

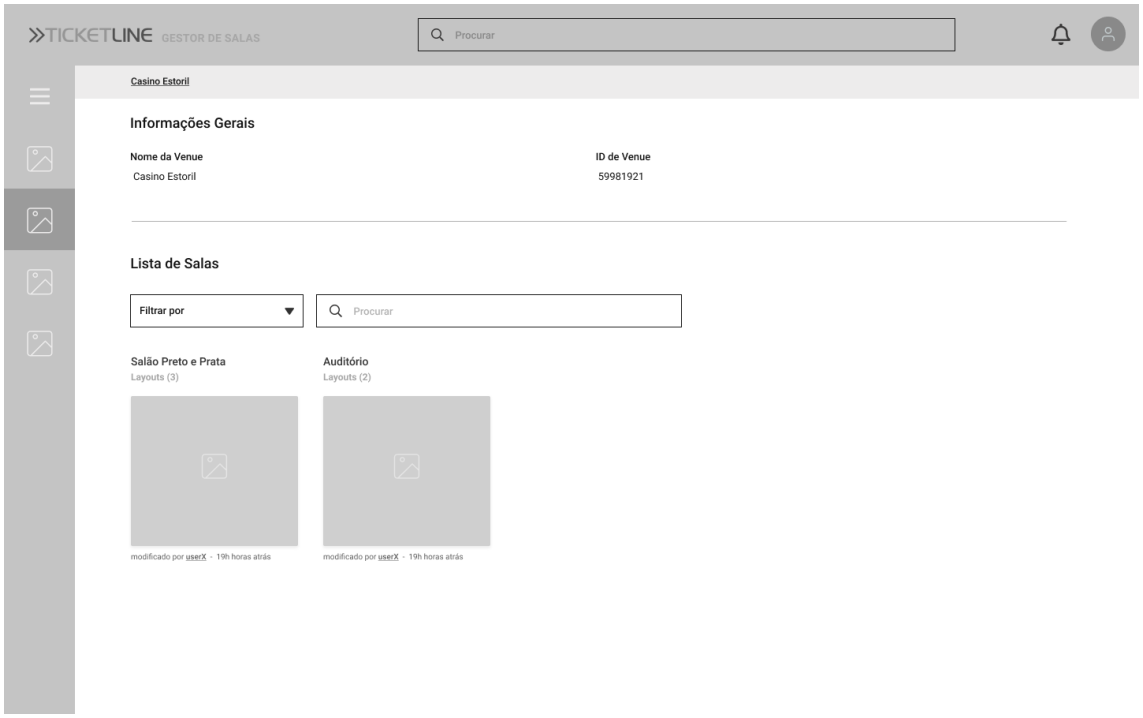


Figura 15: Wireframe: Página do Local

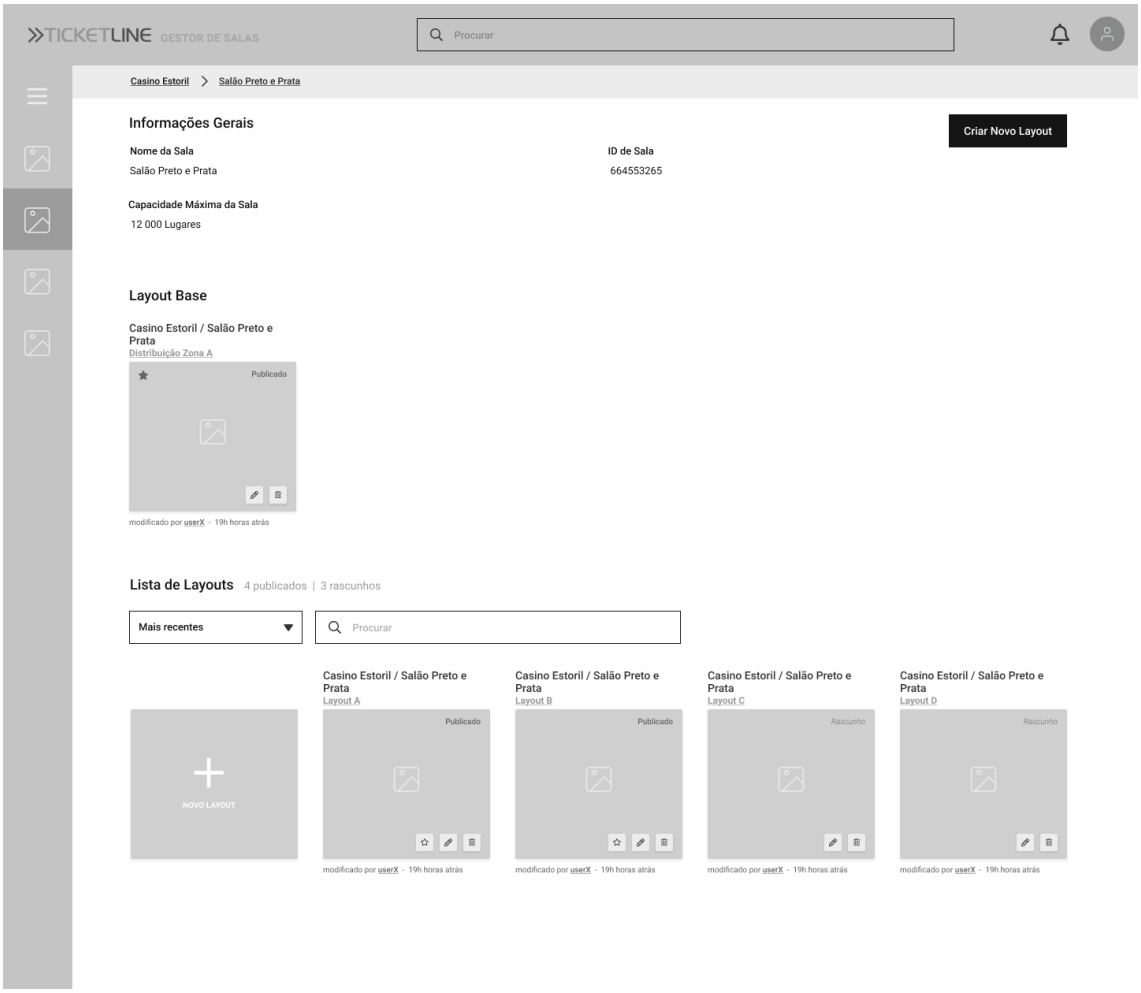


Figura 16: Wireframe: Página da Sala

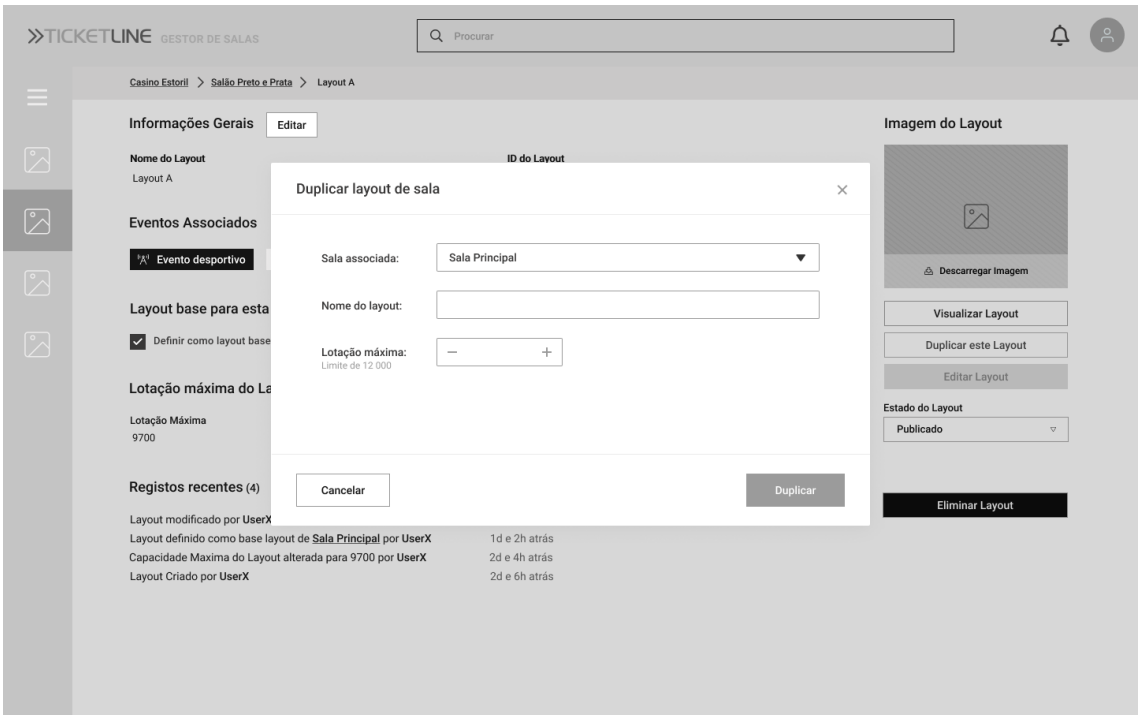


Figura 17: Wireframe: Duplicar Layout

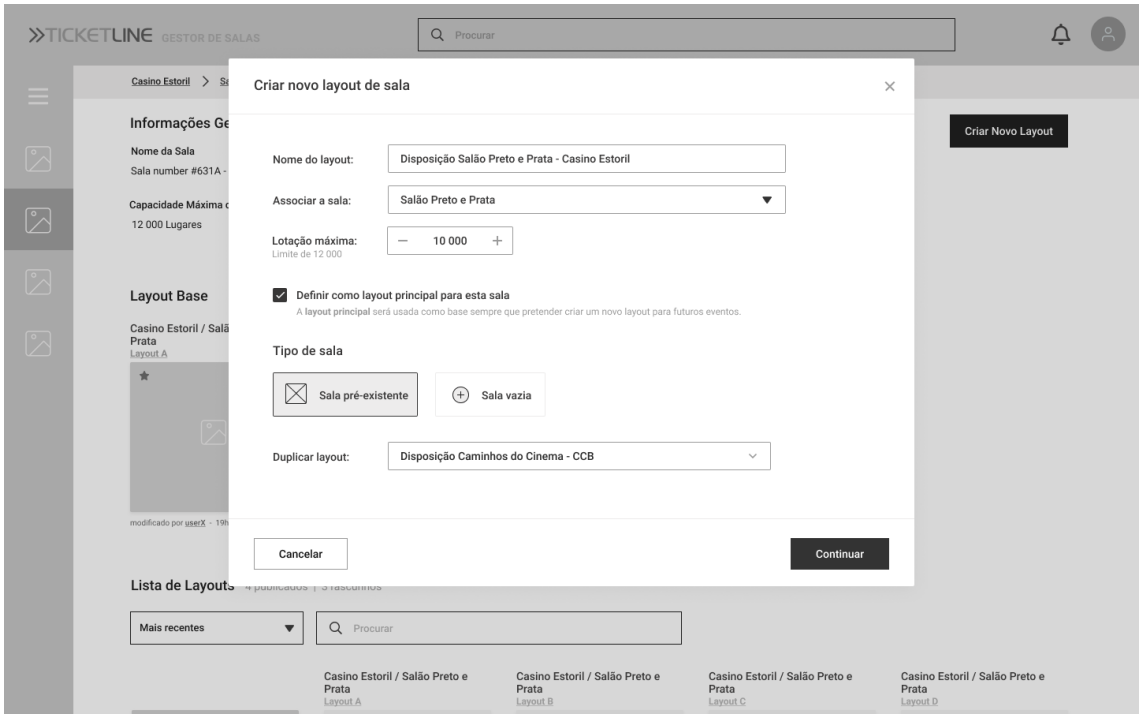


Figura 18: Wireframe: Criar novo Layout

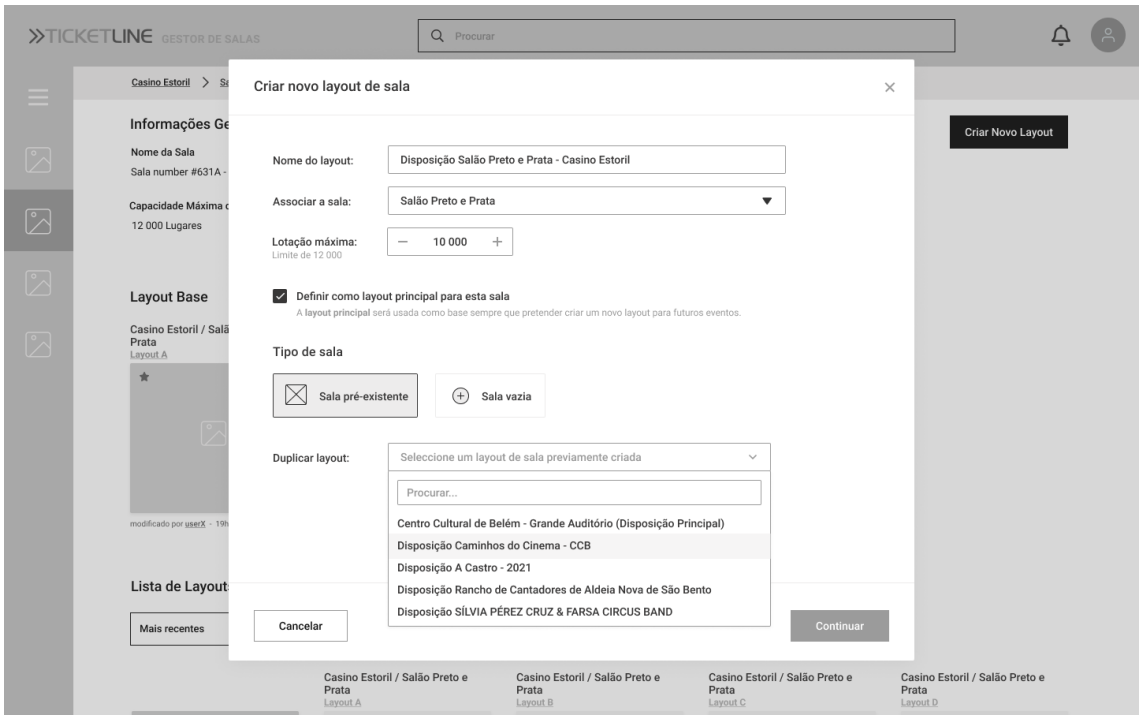


Figura 19: Wireframe: Criar novo Layout

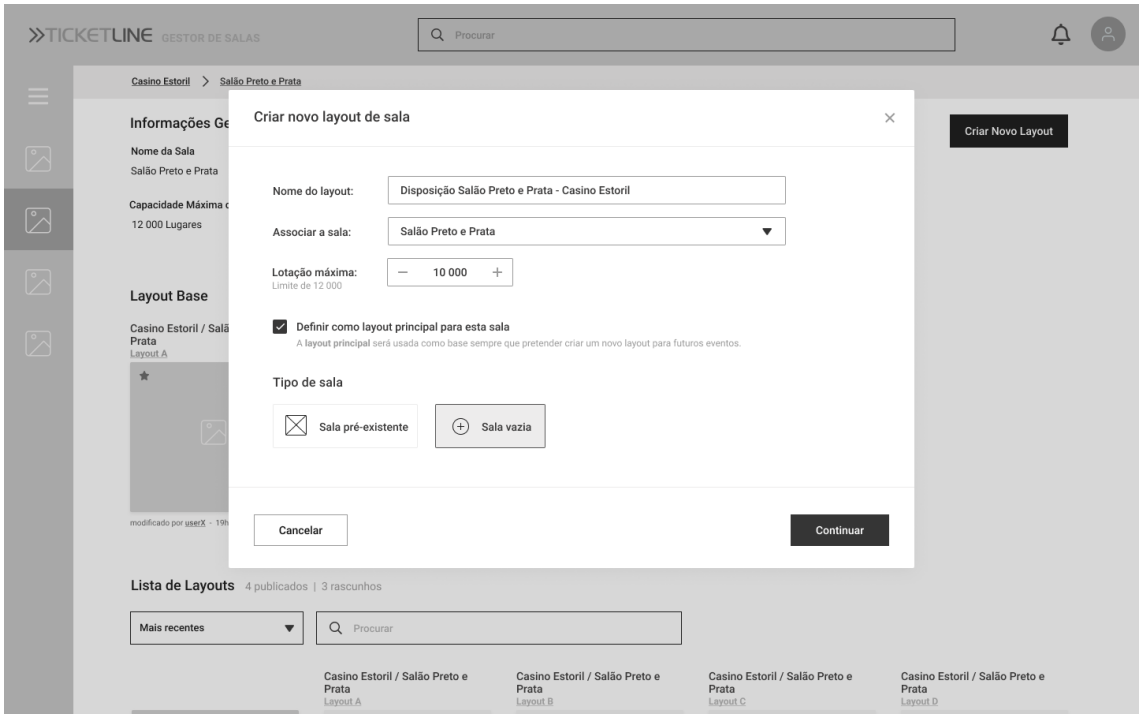


Figura 20: Wireframe: Criar novo Layout

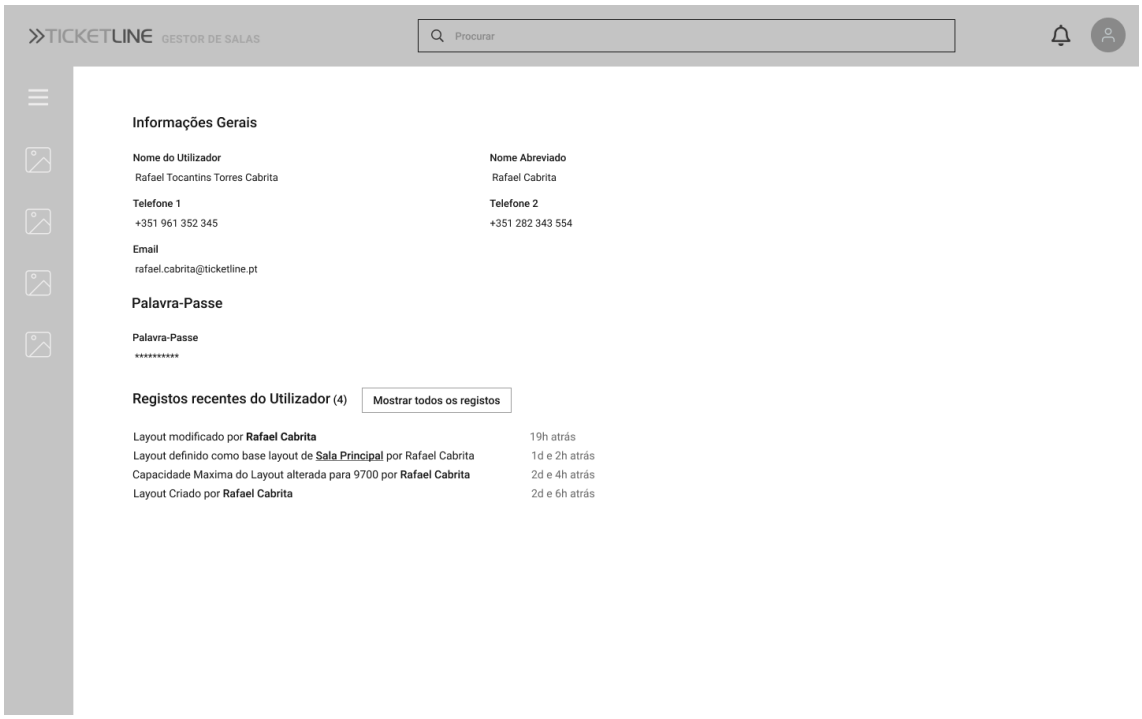


Figura 21: Wireframe: Perfil do Utilizador

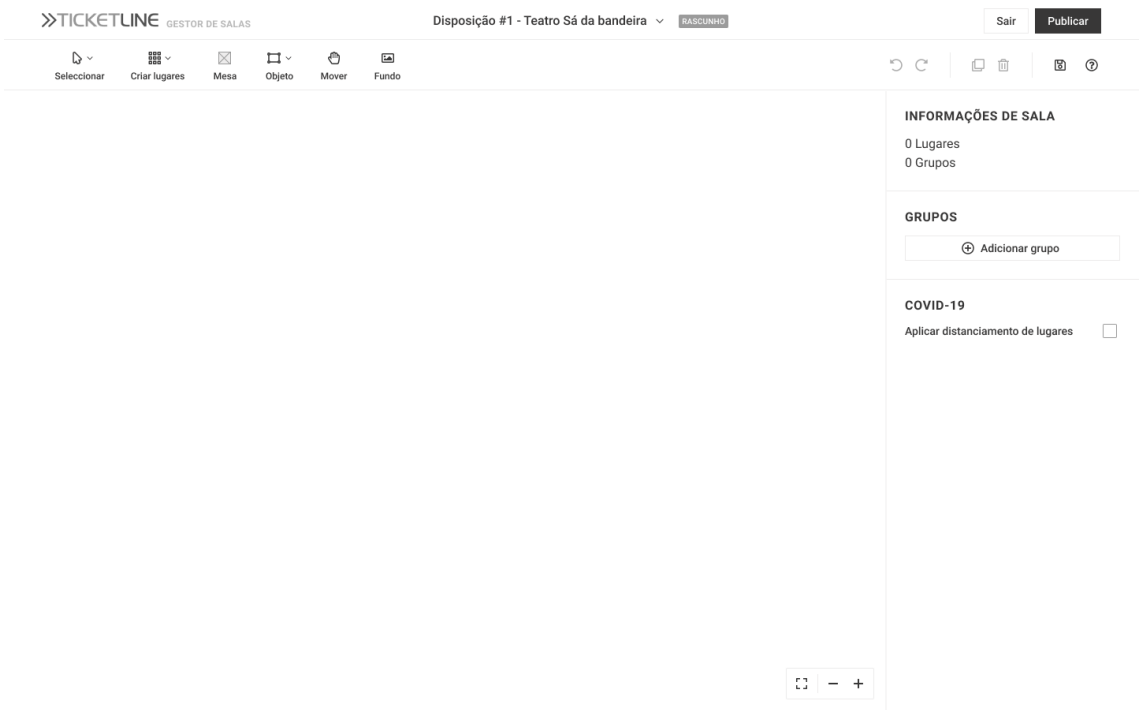


Figura 22: Wireframe: Página Inicial do Editor

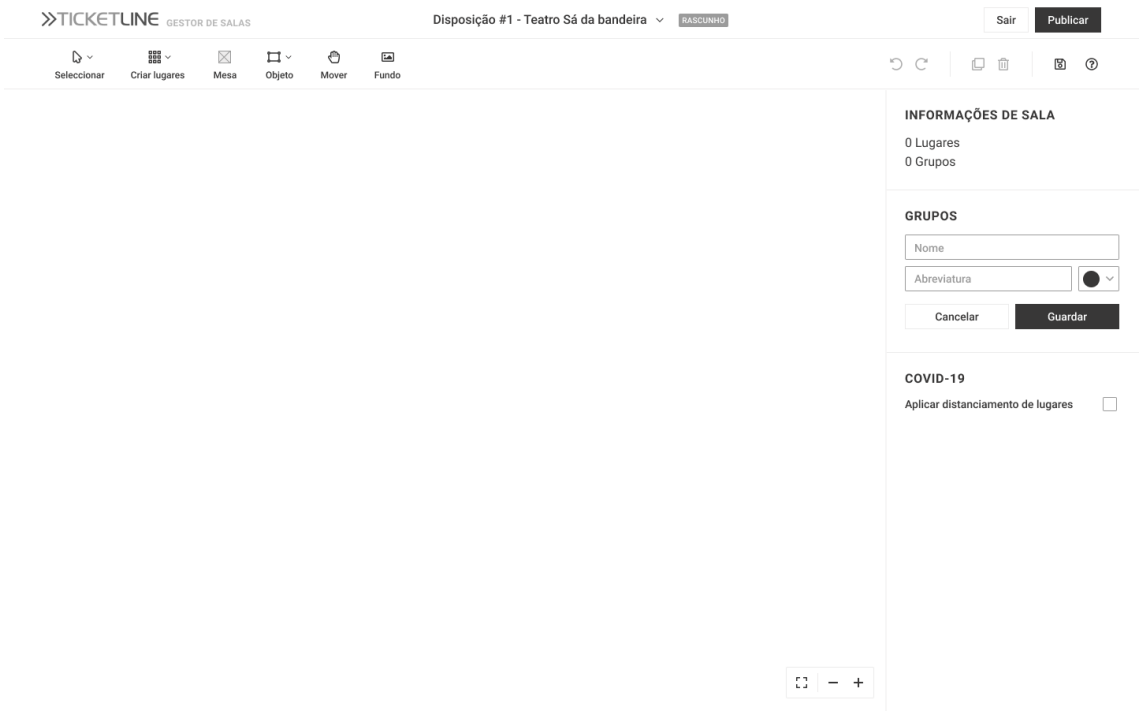


Figura 23: Wireframe: Criar Grupo de Lugares

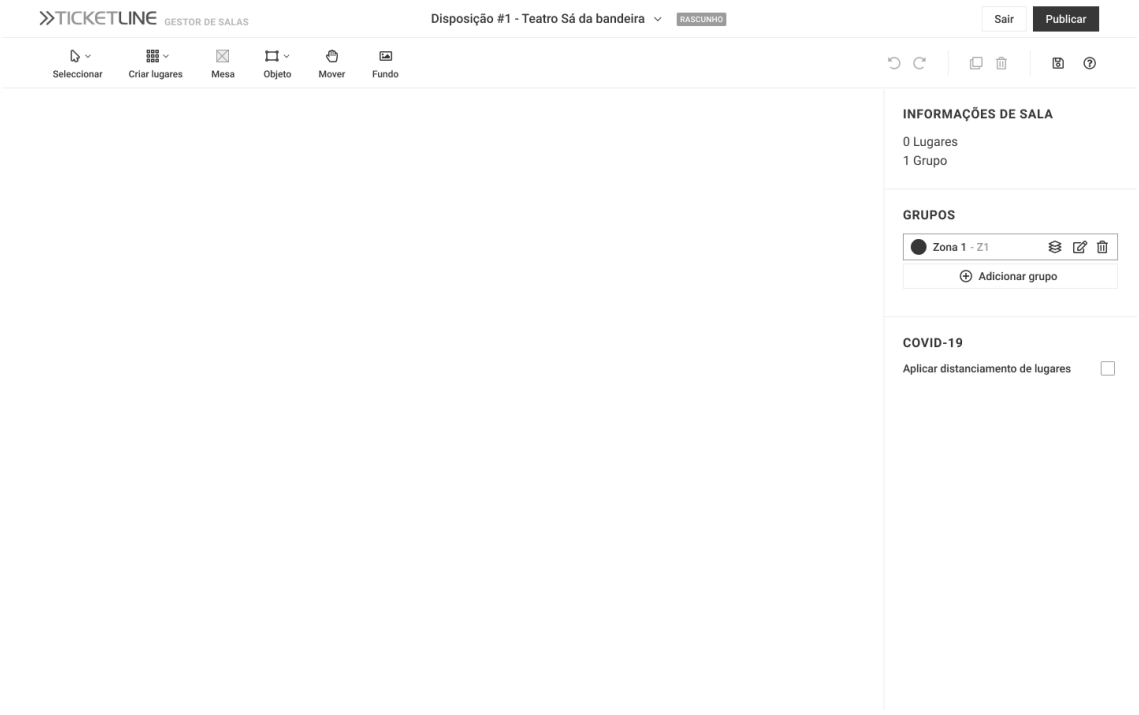


Figura 24: Wireframe: Criar Grupo de Lugares

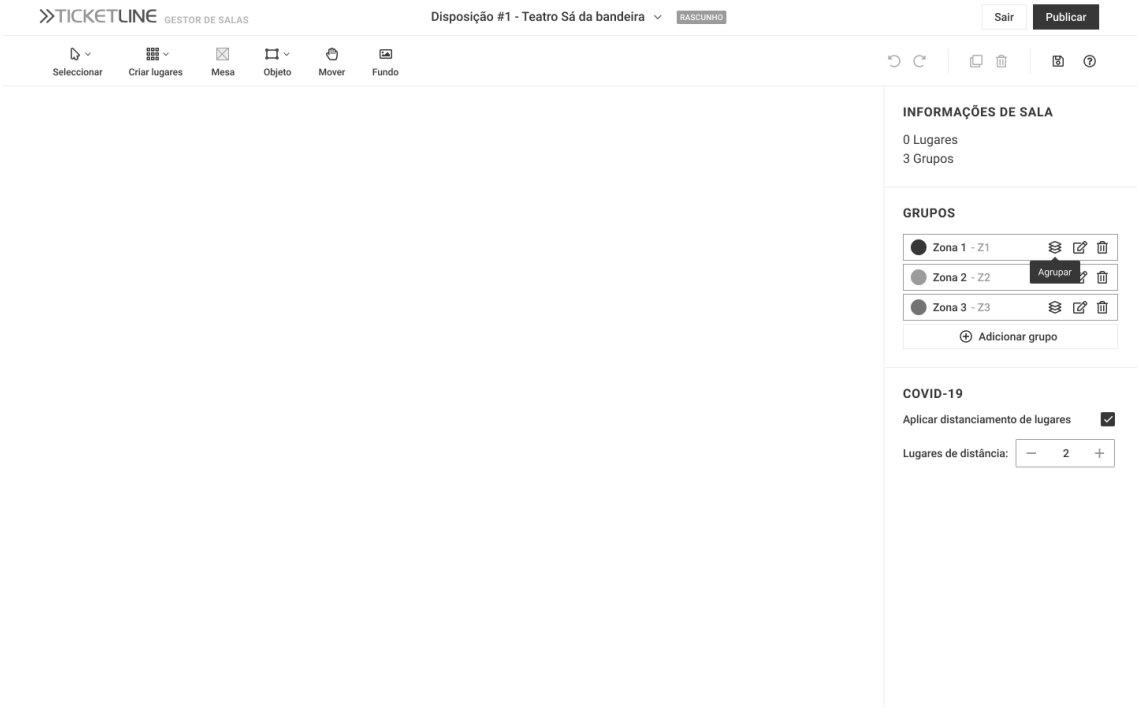


Figura 25: Wireframe: Agrupar Grupos de Lugares

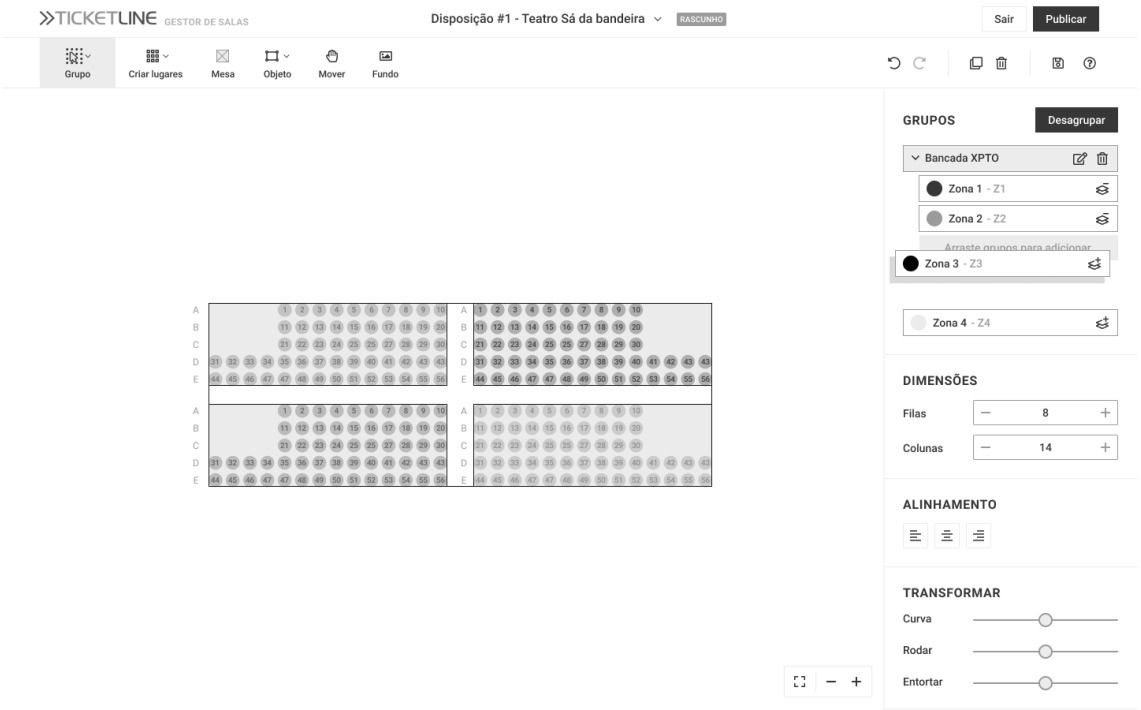


Figura 26: Wireframe: Agrupar Grupos de Lugares

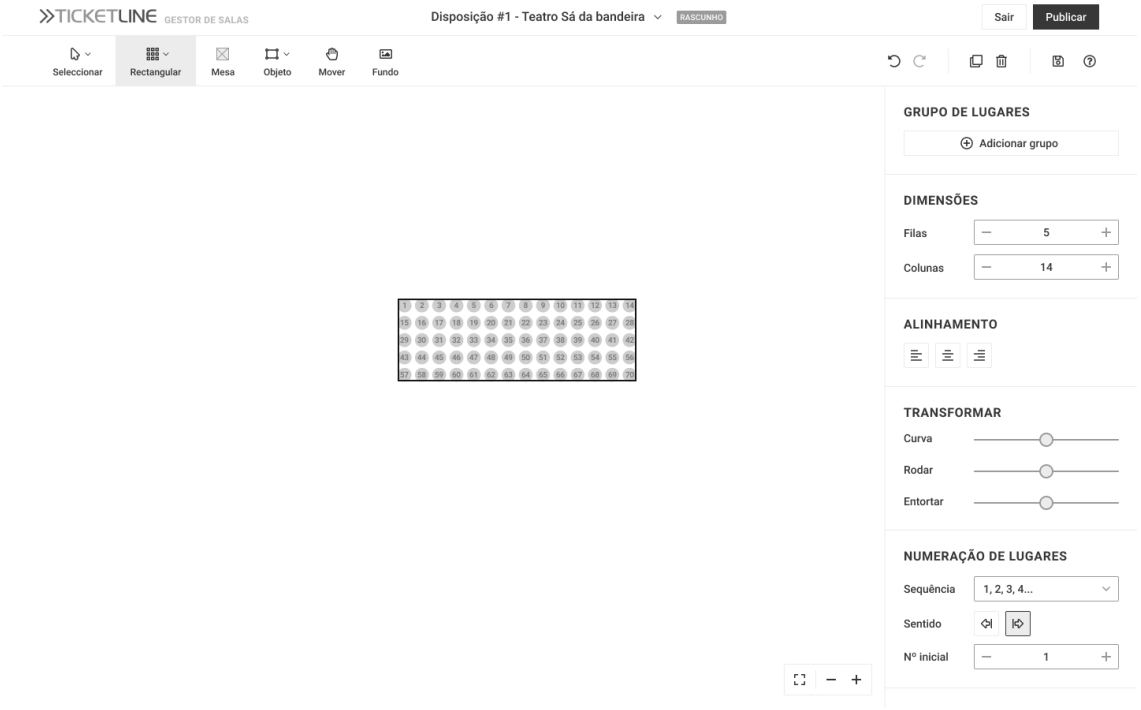


Figura 27: Wireframe: Adicionar Lugares

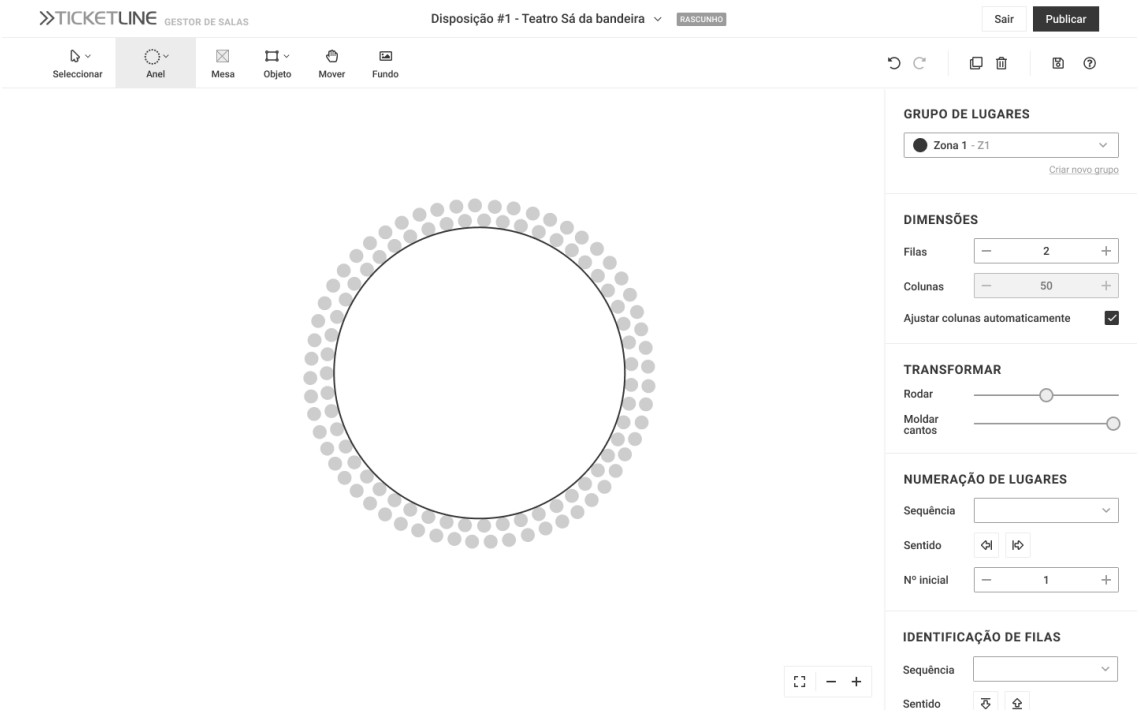


Figura 28: Wireframe: Adicionar Lugares em Anel

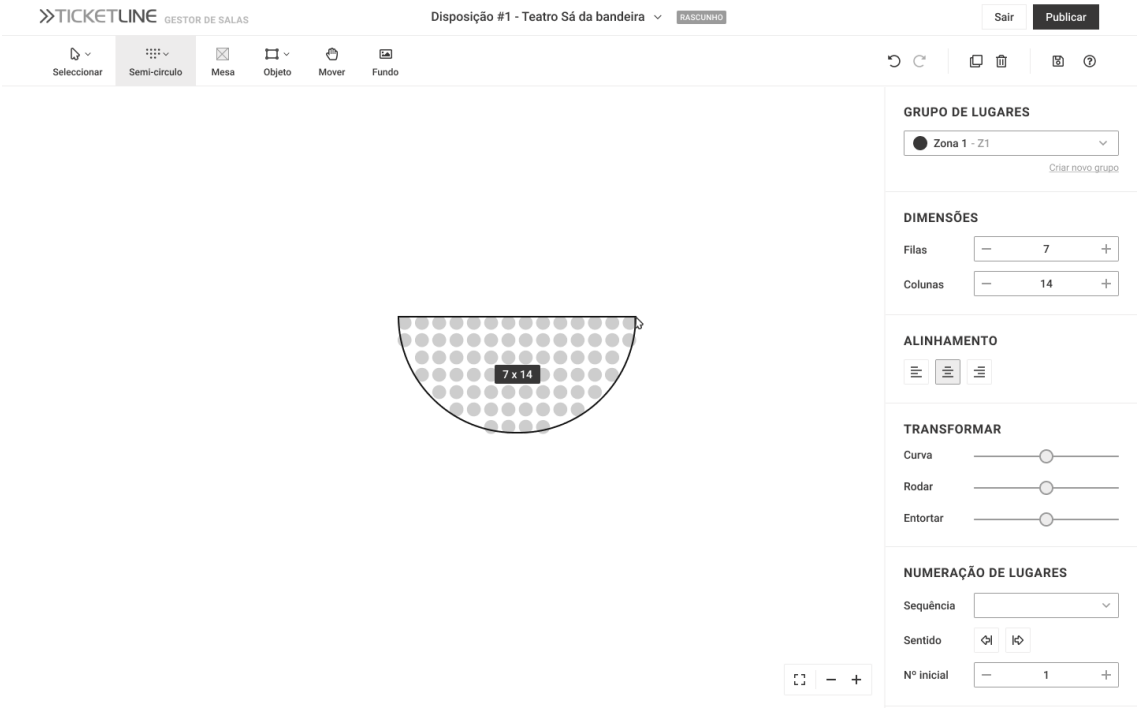


Figura 29: Wireframe: Adicionar Lugares em Semicírculo

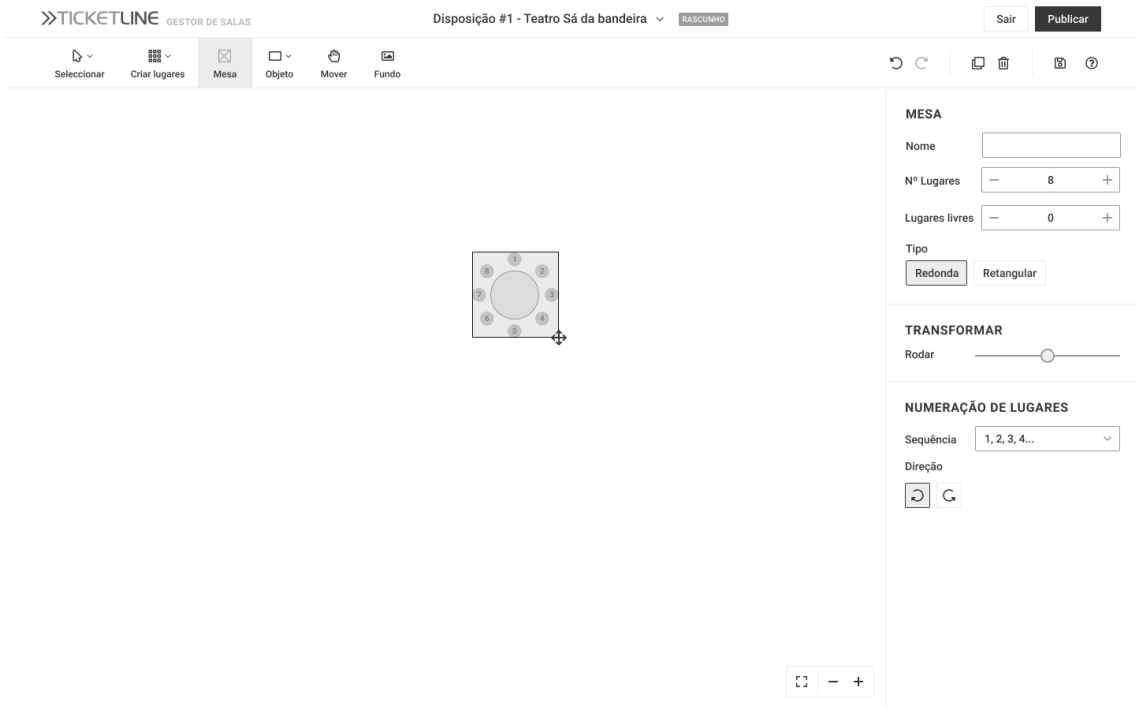


Figura 30: Wireframe: Criar Mesas

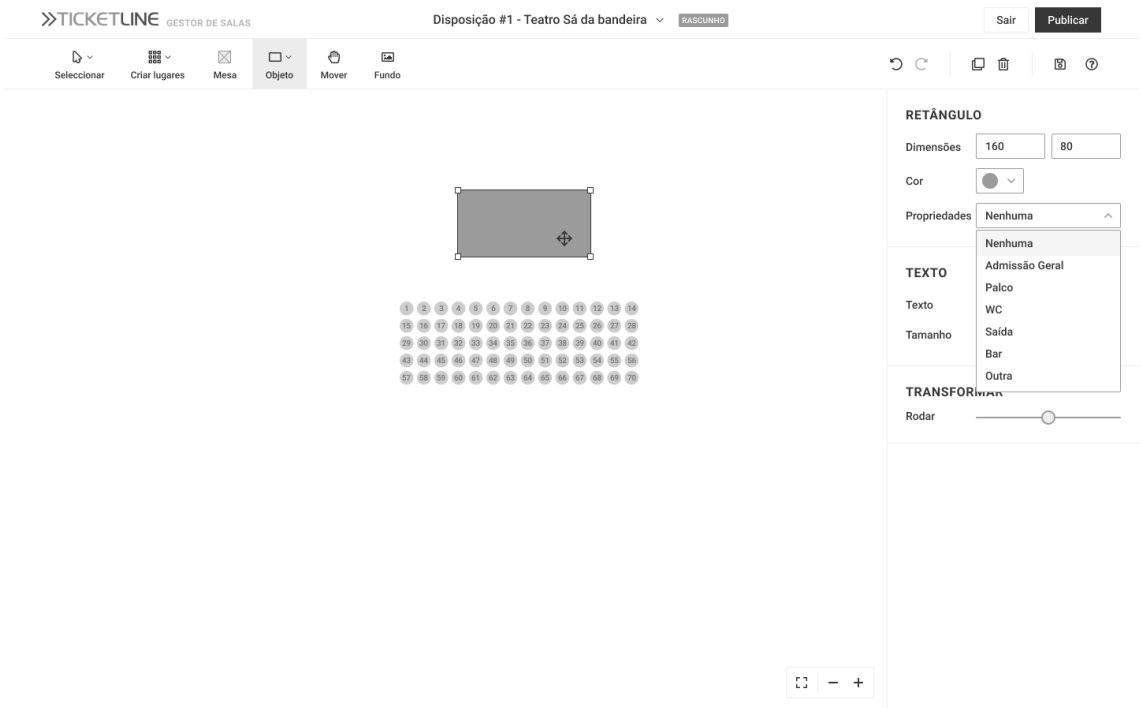


Figura 31: Wireframe: Criar Formas Geométricas

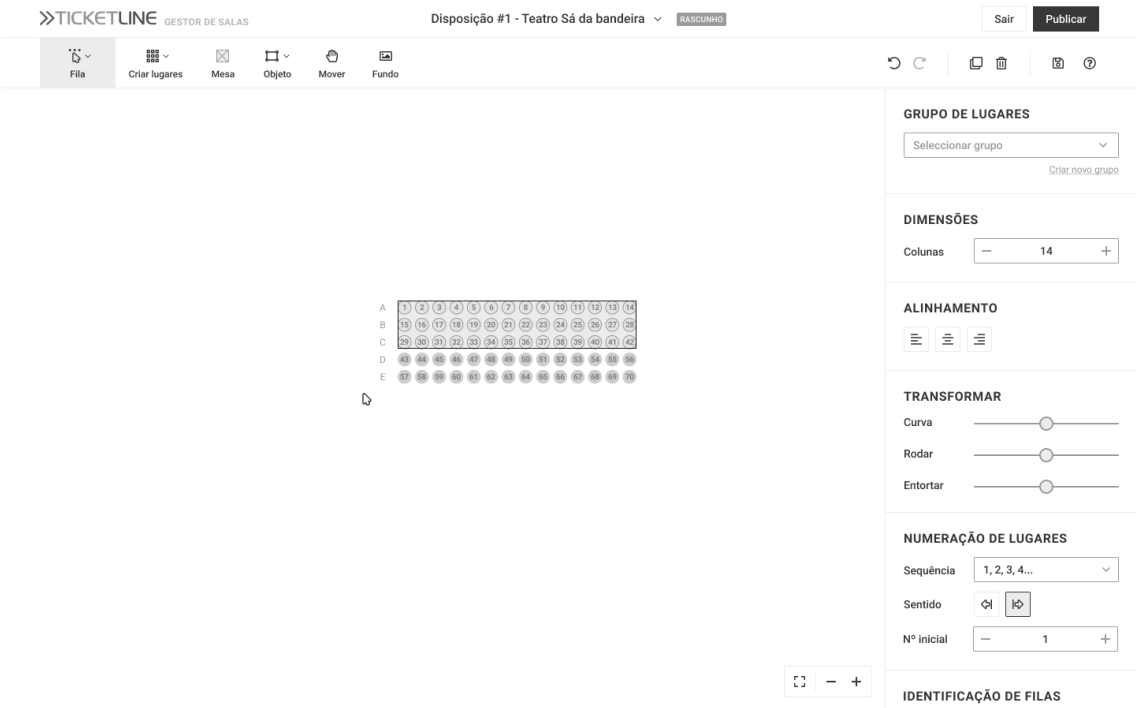


Figura 32: Wireframe: Selecionar Filas de Lugares

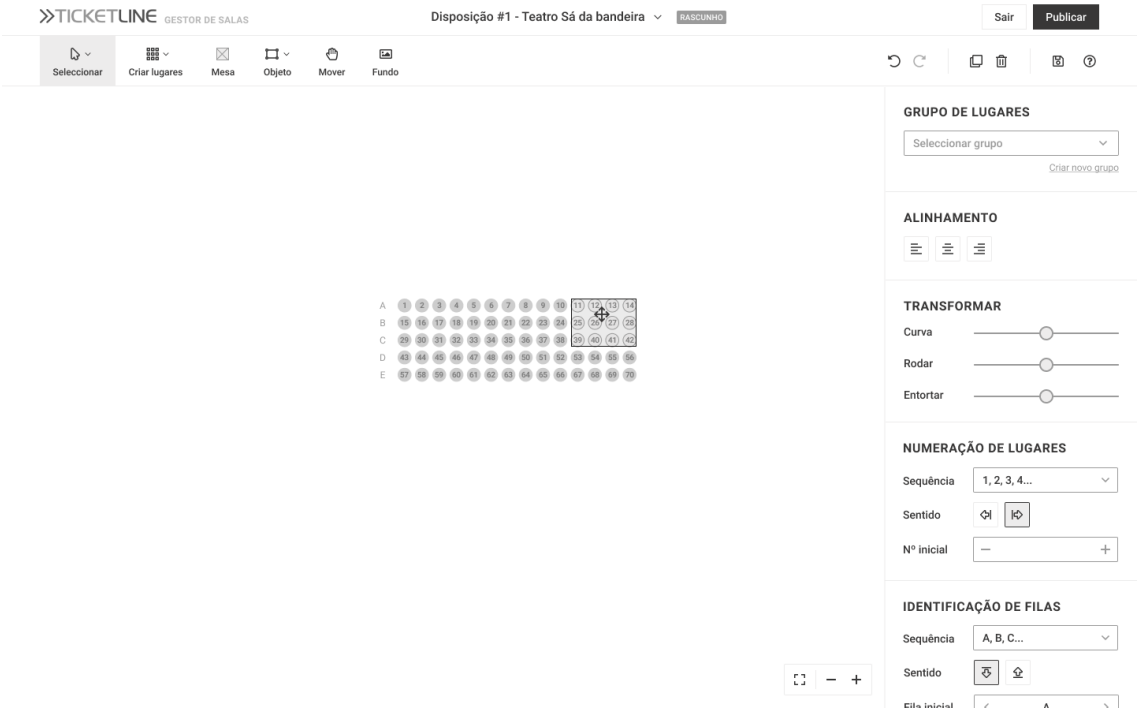


Figura 33: Wireframe: Selecionar Lugares Coletivamente

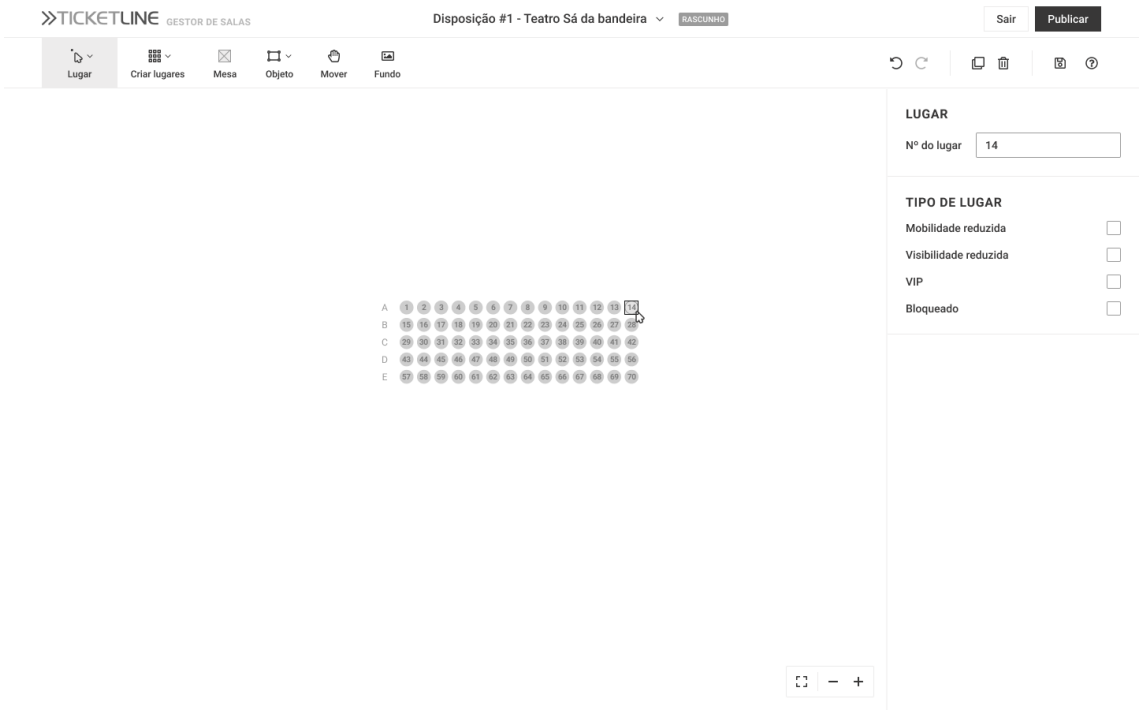


Figura 34: Wireframe: Selecionar Lugar Individualmente

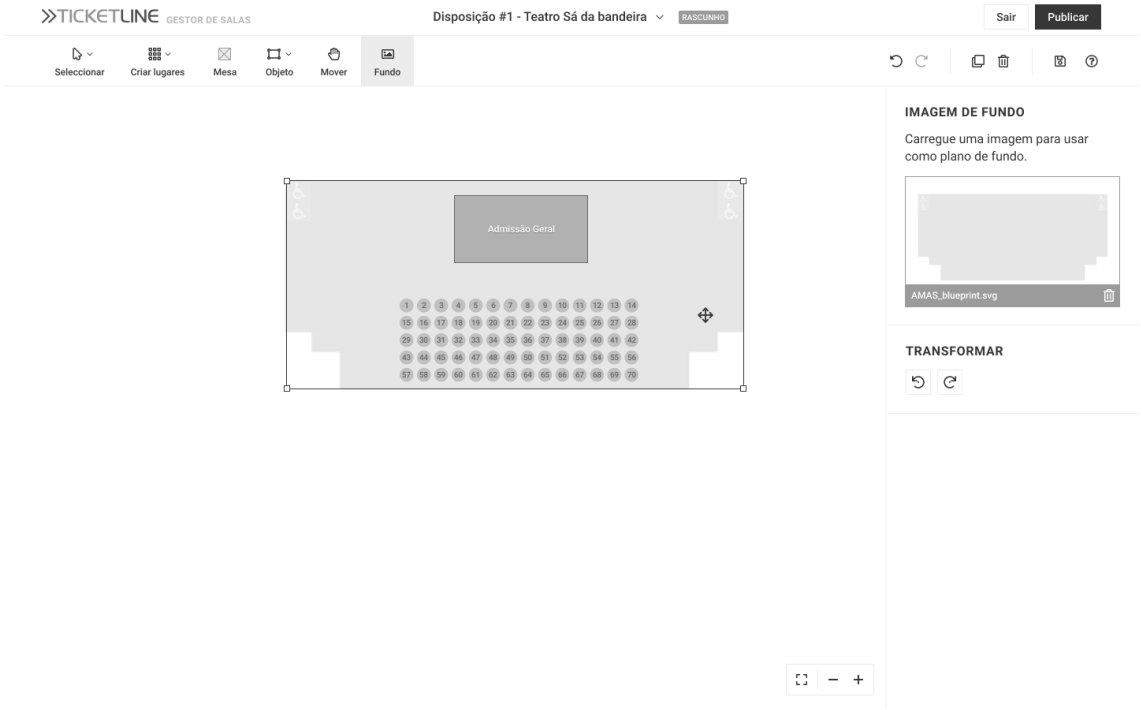


Figura 35: Wireframe: Importar Fundo da Sala

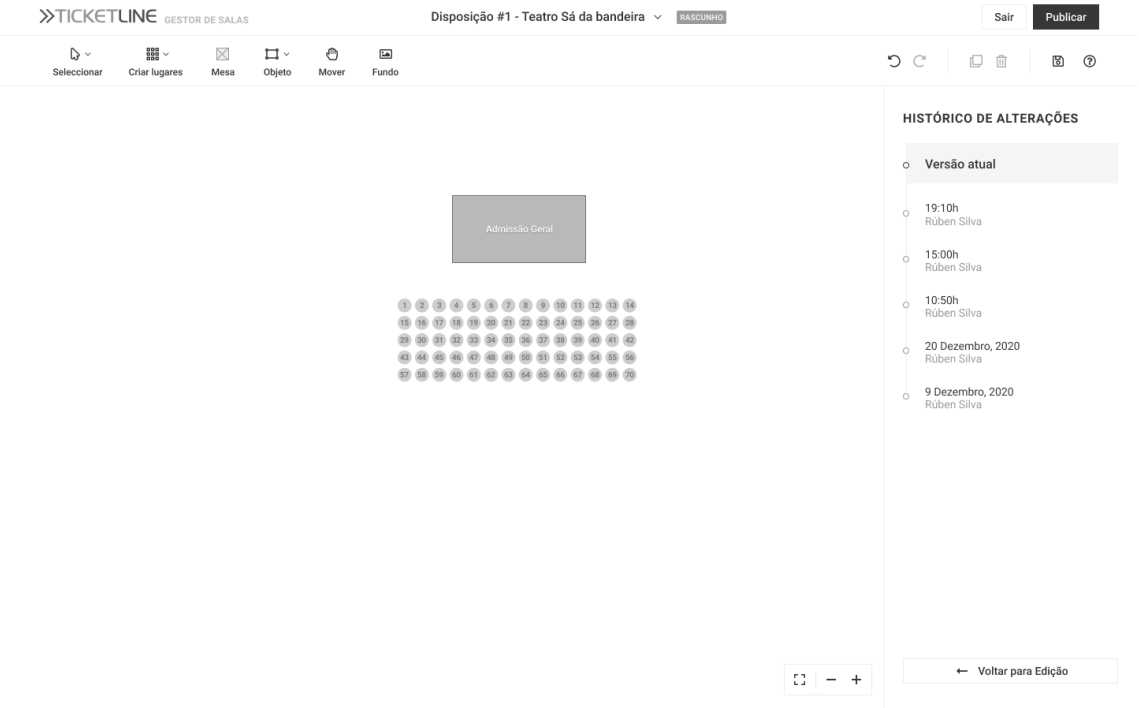


Figura 36: Wireframe: Histórico de Alterações