APLICATIVO WEB PARA CRIAÇÃO E EDIÇÃO DE LAYOUTS TRIDIMENSIONAIS DE ARMAZÉNS

João Paulo Freire dos Santos
Graduando em Tecnologia em Banco de Dados pela FATEC Bauru
joao.santos401@fatec.sp.gov.br

Orientador: Prof. Dr. Alexandre Galvani

RESUMO
No Brasil, o comércio eletrônico segue em constante crescimento, intensificando a necessidade de armazéns logísticos eficientes, especialmente para pequenos e médios negócios. No entanto, a falta de ferramentas acessíveis para planejar layouts de armazéns, aliada ao alto custo e complexidade de softwares como AutoCAD, torna a otimização do espaço e do fluxo operacional um desafio formidável. A presente iniciativa propõe o desenvolvimento de uma aplicação web para criação e edição de layouts de armazéns em visualização 3D, utilizando o banco de dados MongoDB para gerenciamento de dados espaciais e a biblioteca React com Three.js para interatividade de interface. A metodologia inclui revisão bibliográfica, modelagem de banco com JSON, desenvolvimento da interface e testes de desempenho e usabilidade. Espera-se que a aplicação contribua para a redução de custos operacionais e melhora da eficiência logística, beneficiando pequenas empresas e contribuindo para o avanço de tecnologias NoSQL na área de logística.

Palavras-chave: Logística; Armazém; MongoDB; Banco de Dados; React; Visualização 3D

1 INTRODUÇÃO

O setor logístico ocupa um papel estratégico no funcionamento das empresas modernas, especialmente diante do crescimento contínuo do comércio eletrônico. No Brasil, esse fenômeno tem ganhado força e impulsionado transformações importantes na forma como produtos são armazenados, movimentados e entregues aos consumidores. De acordo com dados da Associação Brasileira de Comércio Eletrônico (ABComm), estima-se que o e-commerce brasileiro atinja um faturamento superior a 200 bilhões de reais até o ano de 2025. Esse cenário exige soluções cada vez mais eficientes para o planejamento e a operação logística, o que inclui, de forma essencial, a organização de armazéns.

A forma como o espaço físico de um armazém é planejado influencia diretamente na eficiência das operações logísticas. Um layout inadequado pode causar desperdício de espaço, aumento no tempo de movimentação de mercadorias e dificuldades no atendimento de picos de demanda. Ainda assim, pequenas e médias empresas enfrentam grandes obstáculos ao tentarem melhorar o planejamento de seus armazéns. Entre os principais desafios estão a falta de ferramentas acessíveis para simulação de layouts e o alto custo de softwares especializados, como AutoCAD, que muitas vezes exigem conhecimento técnico avançado para sua utilização ou não

têm uma ferramenta interna que sirva a esse propósito de criação de armazéns.

Nesse contexto, este artigo propõe o desenvolvimento de uma aplicação web voltada à criação e edição de layouts de armazéns com visualização tridimensional (3D). A proposta utiliza tecnologias modernas e amplamente utilizadas no mercado, como o banco de dados NoSQL e MongoDB, para o armazenamento e manipulação de dados espaciais, e a biblioteca React, para a construção de uma interface gráfica interativa e de fácil utilização. A renderização em 3D será implementada por meio da biblioteca Three.js, permitindo ao usuário visualizar, editar e testar diferentes configurações de layout em tempo real. Além de fornecer uma ferramenta gratuita e acessível, a aplicação pretende contribuir para a digitalização e a profissionalização de processos logísticos em empresas que não possuem recursos para investir em soluções comerciais complexas. Do ponto de vista acadêmico, o projeto explora o uso conjunto de tecnologias web modernas e bancos de dados não relacionais aplicados à logística, proporcionando um avanço no campo de sistemas de informação voltados para a organização espacial. Através disso, pretende-se colaborar com a modernização da logística nacional e com a criação de alternativas viáveis para um setor cada vez mais exigido pelo avanço do comércio digital.

2 REFERENCIAL TEÓRICO

2.1 Comparativo entre os Bancos de Dados MYSQL e MONGODB

O artigo desenvolvido por Calasans de Souza e Oliveira (2019) explora as características dos sistemas gerenciadores de bancos de dados (SGBDs) MySQL, um sistema relacional, e MongoDB, um não-relacional, destacando seus conceitos teóricos e vantagens para o mercado. No documento, O MySQL é apresentado como um banco de dados de código aberto, com alta velocidade, segurança e compatibilidade com SQL, enquanto o MongoDB é descrito por sua escalabilidade, flexibilidade e orientação a documentos, ideal para grandes volumes de dados. A metodologia utilizada é qualitativa e comparativa, baseada em revisão bibliográfica de fontes como Silva (2001) e Abraham, Korth e Sudarshan (2012). As principais descobertas e conclusões indicam que o MongoDB é mais indicado para aplicações que exigem manipulação de dados massivos e escalabilidade horizontal. Este estudo é relevante para o atual projeto ainda em desenvolvimento por fornecer uma base comparativa entre modelos relacionais e não-relacionais, justificando a seleção do MongoDB para o desenvolvimento.

2.2 INTEGRATION OF INTERACTIVE 3D MODELS INTO REACT-BASED APPLICATION

O trabalho desenvolvido por Smelov (2024) investiga a integração de interfaces 3D interativas em aplicações web baseadas em React, explorando ferramentas que utilizam WebGL, como Three.js, React Three Fiber (R3F), Spline, Babylon.js e P5.js. Os principais conceitos teóricos abordados incluem fundamentos de modelos 3D (vértices, arestas, faces, shaders, iluminação e câmeras) e as características do React (Virtual DOM, componentes e hooks), que otimizam a renderização de elementos dinâmicos em uma página da Web. A metodologia combina uma análise comparativa qualitativa das ferramentas 3D, baseada em revisão bibliográfica, com um desenvolvimento prático-experimental, no qual foi desenvolvido um aplicativo Web usando as tecnologias React, R3F e Spline, a fim de testar funções interativas como

animações, shaders e física. A conclusão retirada do experimento ressalta que R3F é a solução mais eficiente para integrar 3D em React, oferecendo desempenho e facilidade de uso, enquanto Spline é ideal para designs visuais intuitivos. Um estudo relevante para o desenvolvimento do atual TG, uma vez que o projeto também se trata do desenvolvimento de um aplicativo Web usando tecnologias de WbGL.

2.3 PROJETO DE LAYOUTS DE ARMAZÉNS E MELHORIA DOS PROCESSOS LOGÍSTICOS

O trabalho de Costa (2020) consiste em uma investigação sobre a otimização de layouts e processos logísticos na Amorim Cork Flooring, S.A., com foco em armazéns de LVT (Luxury Vinyl Tiles) e produto acabado. Os principais conceitos teóricos incluem gestão de estoques (modelos determinísticos e estocásticos, Análise ABC e XYZ), desenho de armazéns (fluxos, sistemas de armazenagem, picking) e indicadores de desempenho (rotação e cobertura de estoque). A metodologia é prático-analítica, estruturada em cinco fases: familiarização com a empresa, revisão bibliográfica, análise de dados com ferramentas como Análise ABC/XYZ e Diagrama de Ishikawa, proposta de layouts via software CAD e avaliação parcial das soluções. As descobertas destacam que layouts bem projetados, aliados a gestão visual e automação, melhoram a eficiência operacional e reduzem custos, com propostas como localizações fixas e equipamentos de manuseio. Este estudo possui também certa relevância para a base teórica do atual projeto a ser realizado pelo, por fornecer fundamentos sobre logística de armazenagem.

3 DESENVOLVIMENTO

3.1 MATERIAIS E MÉTODOS

Nesta seção são descritos os recursos tecnológicos utilizados, bem como as etapas metodológicas adotadas para o desenvolvimento do aplicativo web. O atual projeto foi construído com foco na utilização de tecnologias modernas e acessíveis, buscando equilibrar desempenho, facilidade de uso e aplicabilidade ao contexto da logística de armazéns.

3.1.1 ABORDAGEM METODOLÓGICA

O desenvolvimento do presente sistema seguiu uma abordagem incremental, cuja implementação e testes foram realizados em ciclos curtos. Cada funcionalidade foi desenvolvida, validada isoladamente e depois integrada ao ambiente geral. O fluxo de trabalho foi dividido em três fases principais, respectivamente:

Desenvolvimento do Front-End: criação da interface 3D interativa utilizando React, Three.js e React Three Fiber, priorizando a visualização e manipulação dos racks (porta-paletes).
Desenvolvimento do Back-End: estruturação do servidor Node.js com Express e integração ao banco de dados MongoDB, para persistência dos layouts criados pelo usuário.
Integração e Testes: conexão entre as duas camadas por meio de uma API RESTful, seguida de testes com o Postman e no ambiente local de desenvolvimento.
Durante a execução de cada etapa, adotou-se um modelo de prototipagem evolutiva, em que as funcionalidades eram aprimoradas conforme o comportamento esperado

do usuário era validado no ambiente 3D.

O planejamento consistiu em desenvolver primeiro o Front-End, para que os valores e informações em relação a estruturas de armazenagem, bem como suas medidas, posições e tipos, se tornassem mais palpáveis e compreensíveis. As principais tecnologias utilizadas nessa primeira etapa incluem JavaScript, Vite, bibliotecas React, Three.js e React Three Fiber, as quais serão explicadas posteriormente.

3.1.2 TECNOLOGIAS UTILIZADAS

O projeto foi desenvolvido em JavaScript (JS), uma linguagem de programação conhecida por sua tipagem dinâmica e multiparadigma. Isso significa que é possível usar JS para suportar várias abordagens de programação, tal como orientação a objetos, programação funcional, procedural, etc.. Ela é suportada por todos os navegadores modernos, o que a torna amplamente utilizada para desenvolvimento web. bem como por ambientes de servidor como Node.js. Entre suas características destacam-se a versatilidade de execução, tanto no cliente (navegador) quanto no servidor, alta interatividade, manipulação do DOM e compatibilidade com APIs web modernas. A documentação da Mozilla (MDN) afirma que “JavaScript é uma linguagem leve, interpretada e baseada em objetos com funções de primeira classe, e conhecida como a linguagem de script para páginas Web, mas usada também em vários outros ambientes sem browser, tais como Node.js, Apache CouchDB e Adobe Acrobat.”

Vite é um moderno kit de ferramentas para desenvolvimento frontend que tem como foco a velocidade de inicialização e recarga do servidor de desenvolvimento de maneira eficiente. Ele serve arquivos à medida que são requisitados (*on-demand*), em vez de recompilar todo o projeto a cada alteração, o que reduz o tempo de espera no desenvolvimento. Vite também oferece suporte imediato para JSX, CSS, TypeScript e outras linguagens ou transpiladores sem configurações pesadas.

React é uma biblioteca JavaScript para construção de interfaces de usuário, especialmente aquelas que exigem atualização dinâmica do DOM. Seu modelo de componentes facilita a modularização e a reutilização de partes da interface. O React mantém uma representação virtual do DOM, o que permite otimizar atualizações e renderização, minimizando manipulações desnecessárias do DOM real. Além disso, React permite integração com diversas bibliotecas externas, facilitando estender funcionalidades, gerenciar estado, roteamento, etc.

Three.js é uma biblioteca JavaScript para renderização de gráficos 3D em navegadores via WebGL. Seu uso permite trabalhos com geometria, iluminação, câmeras, texturas, sombras, entre outros elementos visuais, oferecendo baixo nível de controle sobre as cenas gráficas.

React Three Fiber (R3F) é um renderizador do React para Three.js, permitindo que cenas e recursos 3D possam ser definidos em JavaScript XML (JSX), tornando possível também gerenciar ciclos de vida, estados e componentes do React, abstraindo boa parte da complexidade de trabalhar diretamente com Three.js.

O Node.js, de acordo com a documentação oficial (NODE.JS FOUNDATION, 2025), é um ambiente de execução JavaScript multiplataforma e de código aberto, projetado

para criar aplicações web escaláveis no lado do servidor. Sua principal característica é a permissão do uso de JavaScript tanto no cliente quanto no servidor, eliminando qualquer necessidade de migração para linguagens diferentes entre as camadas da aplicação.

O Express, como consta em sua documentação, “é um framework de aplicações web Node.js minimalista e flexível que oferece um conjunto robusto de recursos para aplicações web e mobile" [tradução automática] (EXPRESS.JS FOUNDATION, 2025)

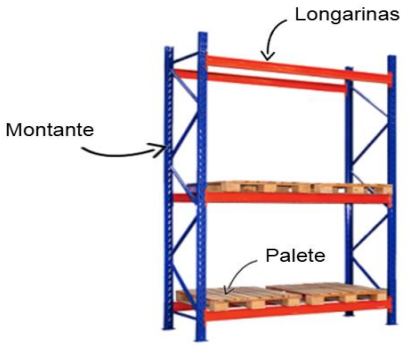
O MongoDB é um banco de dados NoSQL orientado a documentos, e segundo o seu manual oficial (MONGODB INC., 2025), é projetado como uma alternativa à bancos de dados relacionais para aplicações modernas, permitindo que os desenvolvedores armazenem documentos avançados semelhantes a JSON que mapeiam naturalmente os objetos usados em seus códigos.
Por seu formato de documento, sem esquemas rígidos, torna-se mais fácil gravar estruturas de dados não primitivas como arrays e objetos, fazendo do MongoDB uma excelente escolha para armazenamento dos layouts 3D gerados pelo usuário.

3.1.3 CONTEXTO APLICADO: ESTRUTURAS PORTANTES

As principais estruturas presentes em um armazém são os porta-paletes, que por definição são estruturas metálicas usadas para armazenar e organizar cargas paletizadas de forma vertical, aproveitando o espaço do armazém e otimizando a movimentação de mercadorias. São conhecidos por sua versatilidade e simplicidade, facilitando o acesso direto a cada palete, e são usados amplamente em indústrias, galpões e centros de distribuição.

Eles são compostos basicamente por duas partes principais: os montantes, que são as estruturas verticais, e as longarinas, que são vigas horizontais fixadas nos montantes (geralmente são destacadas por cores quentes, como laranja, por exemplo) sobre as quais os paletes são apoiados, distribuindo assim o peso da carga.

Figura 1: Estruturas em um porta-palete



Fonte: Adaptado de COMAQ PARANÁ (2024).

Foi decidido que para recriar fileiras de porta-paletes com React, seria necessário criar um modelo 3D para representar os montantes e paralelepípedos coloridos para representar as longarinas, sendo estas últimas criadas a partir de componentes geométricos nativos do React Three Fiber. O modelo tridimensional dos montantes foi criado com o auxílio do Tripo 3D Studio, uma ferramenta Web gratuita que utiliza recursos de Inteligência Artificial para transformar imagens em objetos 3D texturizados, dispensando assim o trabalho inicial de usar softwares de modelagem mais complexos.

3.1.4 CONFIGURAÇÃO DO AMBIENTE DE DESENVOLVIMENTO

Para inicialização do Front-End do projeto, foi utilizado o Node.js como ambiente de execução JavaScript e o NPM como gerenciador de pacotes. A criação da estrutura-base foi realizada por meio do comando:

```
npm create vite@latest tcc
```

Durante o processo de configuração, foi selecionado o template React, fornecido pelo próprio Vite, por sua leveza e suporte nativo a módulos ECMAScript. Essa escolha permitiu o desenvolvimento de uma aplicação modular com recarga automática, permitindo uma facilidade maior para testes e ajustes em tempo real, sem precisar ficar recarregando o ambiente.

Em seguida, foram instaladas as bibliotecas necessárias para a renderização tridimensional e para navegação da aplicação:

```
npm install three @react-three/fiber @react-three/drei react-router-dom
```

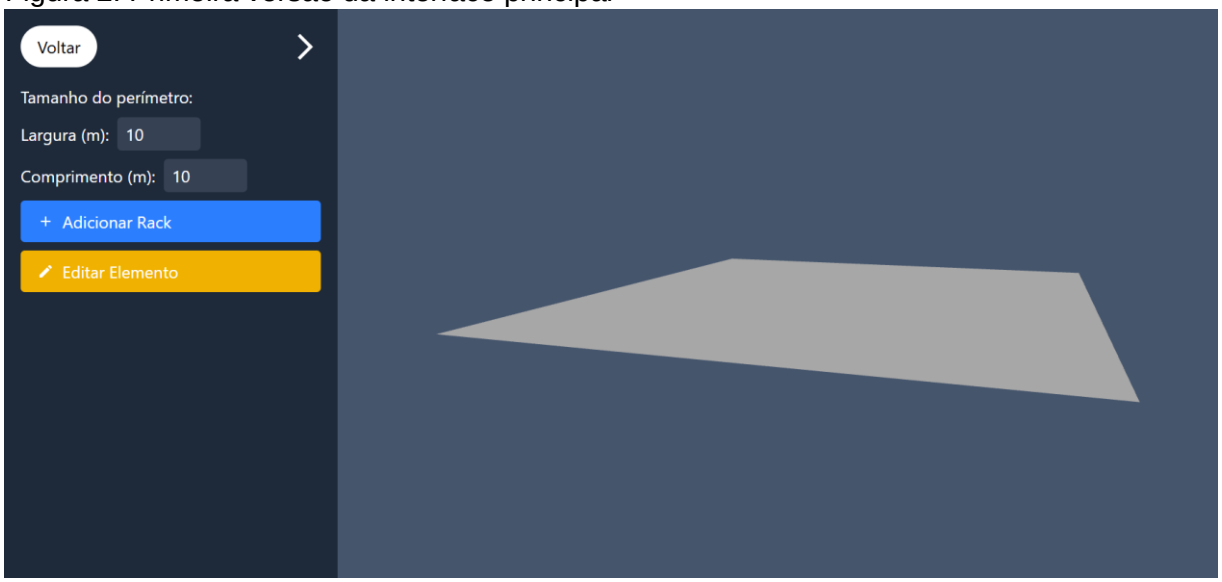
A inicialização do ambiente de desenvolvimento foi realizada localmente, através do comando:

```
npm run dev
```

Esse processo disponibiliza um servidor hot-reloading local acessível via navegador, permitindo a renderização imediata da interface, assim como os testes de cada implementação em tempo real, sem a necessidade de recarregar a página manualmente no navegador.

Inicialmente, para uma pré-visualização da tela principal, foi criado o Canvas, que é o componente fundamental para a exibição da cena 3D, e foi adicionado um plano cinza, representando o “chão” do armazém, com largura e comprimento ajustáveis através de inputs correspondentes a essas medidas em um menu lateral retrátil. Foram também posicionados dois botões para serem configurados e usados nas etapas posteriores.

Figura 2: Primeira versão da interface principal



Fonte: Elaborado pelo autor (2025).

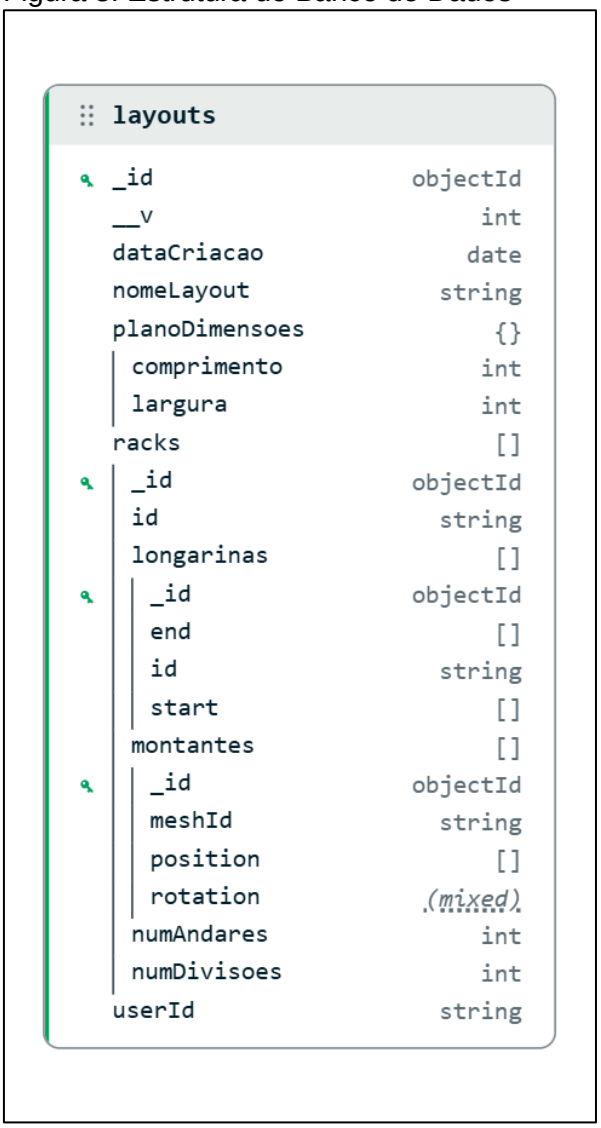
3.1.5 INTEGRAÇÃO COM O BACK-END

A camada de persistência de dados foi implementada com Node.js, Express e MongoDB Atlas (versão Cloud do MongoDB), permitindo o armazenamento e recuperação de layouts criados pelo usuário.

Para operações de criação e leitura de layouts, foram configuradas rotas locais RESTful usando os métodos GET para ler e recuperar dados, POST para criar layouts, DELETE para remoção, e POST para atualizar dados de layouts que já estejam salvos.

A estrutura de armazenamento dos layouts no MongoDB foi organizada em documentos no formato mostrado na figura abaixo, onde cada layout contém metadados gerais, as dimensões do plano e uma lista de racks, compostos por seus respectivos montantes e longarinas.

Figura 3: Estrutura do Banco de Dados



Fonte: Elaborado pelo autor com base em dados do MongoDB Compass (2025).

3.2 RESULTADOS E DISCUSSÃO

O desenvolvimento do aplicativo web foi estruturado em etapas funcionais, refletindo a evolução natural do projeto e seu fluxo de uso. Considerando que o foco principal foi a criação de um editor tridimensional interativo para o planejamento de layouts de armazéns, a versão final apresenta um ambiente 3D interativo que permitia a criação e manipulação de localização de fileiras inteiras de porta-paletes em escala real.

3.2.1 IMPLEMENTAÇÃO DA MECÂNICA DE CRIAÇÃO DE FILEIRAS

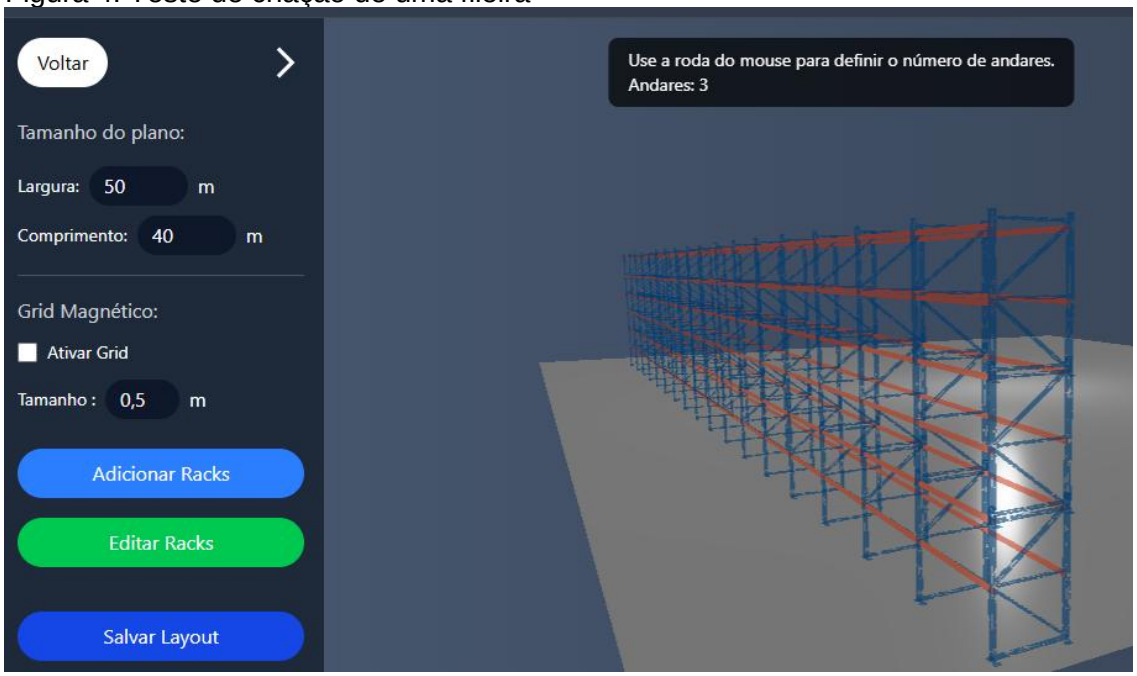
A primeira funcionalidade desenvolvida foi a mecânica de criação de fileiras de racks, que constitui a base de toda a estrutura do editor 3D. Essa etapa foi implementada no componente nomeado de CorteDeLaco, sendo responsável por controlar o posicionamento dos montantes, o alinhamento automático das longarinas e a definição do número de divisões e andares.

O processo de criação foi estruturado como uma máquina de estados, conduzindo o

usuário de forma progressiva por quatro fases:

Colocar Abertura: O usuário define o primeiro ponto da fileira em qualquer lugar do plano, que representa o início da estrutura.
Colocar Fechamento: O segundo ponto é posicionado de maneira que este sempre siga um alinhamento linear em relação ao primeiro, para impedir fileiras diagonais, determinando a direção e o comprimento da fileira.
Corte de Laço: Inspirado em uma mecânica presente no Blender, um software de modelagem, o sistema calcula automaticamente as divisões intermediárias para adicionar os outros montantes da fileira, exibindo uma pré-visualização deles. O número de divisões pode ser ajustável com a roda do mouse. Assim que o ajuste estiver de acordo com o cliente, ele pode confirmar isso com um clique, indo para o próximo estado.
Definir Andares: O número de andares da fileira é configurado pelo usuário, também com a roda do mouse, gerando pré-visualizações dos montantes e longarinas em alturas proporcionais, sendo confirmado também com um clique do mouse, encerrando assim a criação de uma fileira inteira.

Figura 4: Teste de criação de uma fileira



Fonte: Elaborado pelo autor (2025)

A interação com o ambiente 3D é mediada por um raycaster do Three.js, responsável por projetar um raio virtual a partir da posição do cursor e detectar colisões com o plano de referência. Essa técnica permite calcular com precisão a posição tridimensional do mouse sobre o plano de trabalho, permitindo que os elementos tridimensionais sejam adicionados exatamente nessa posição.
Além disso, foi implementado um sistema de grid magnético, com escala ajustável em metros, garantindo o alinhamento exato dos elementos conforme a malha de referência ativa e conforme a distância selecionada pelo usuário. Também foi pensado na criação de mensagens informativas no canto superior do editor, com o objetivo de

orientar o usuário em cada etapa da construção.
A criação dessa mecânica foi essencial para permitir a criação modular e escalável de layouts de armazéns, servindo como base para as etapas seguintes.

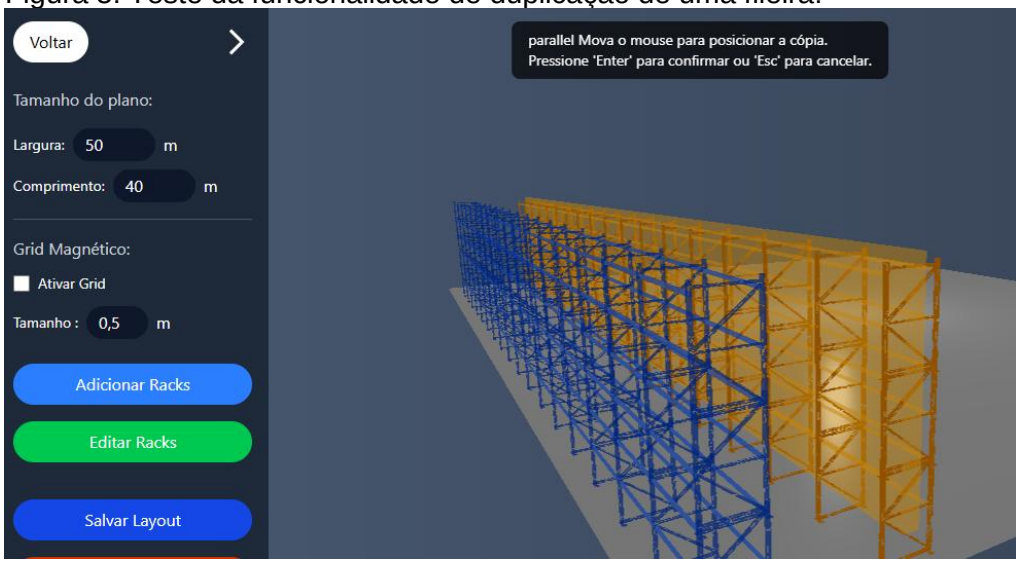
3.2.2 IMPLEMENTAÇÃO DA MECÂNICA DE SELEÇÃO

O objetivo dessa mecânica foi permitir que o usuário clicasse sobre qualquer parte de uma fileira de porta-paletes para selecioná-lo. A ideia era fazer com que ao clicar sobre uma fileira, esta seria destacada visualmente, mudando de cor de para laranja, indicando que o modo de seleção está ativo.
Um obstáculo inicial envolvia o próprio Raycaster, que por ser muito preciso, dificultava a seleção, visto que uma fileira pode possuir muitas geometrias, então o estado de seleção não era ativado, simplesmente porque o raio que partia da posição do mouse, ainda que conseguisse detectar colisões com os elementos geométricos da fileira (montantes e longarinas), não os compreendia como uma única malha a ser selecionada, podendo desencadear conflitos na ativação do estado de seleção. Para resolver esse problema, foi preciso implementar uma técnica similar ao triggering, que consiste no uso de uma malha 3D para executar ações, como se fosse um gatilho. No contexto do projeto, foi feito um ajuste no código-fonte para que fossem adicionados paralelepípedos transparentes envolvendo cada fileira assim que o modo de seleção fosse ativado. Com isso, o Raycaster precisava apenas detectar colisões com esses gatilhos, garantindo seleção estável e intuitiva.
Também foi implementado um listener para a tecla Esc, possibilitando a desmarcação imediata do objeto. Essa mecânica de interação trouxe maior clareza visual e controle sobre os elementos dos racks, além de preparar o terreno para futuras possibilidades de operações de edição, como variações de cor e mudanças para outras estruturas de armazenagem presentes no mundo da logística.

3.2.3 IMPLEMENTAÇÃO DA MECÂNICA DE EDIÇÃO

Essa etapa consistiu no progresso da mecânica de seleção, a qual permite que a fileira selecionada possa também ser submetida a operações básicas de edição, como mudança de posição, rotação, duplicação e exclusão da fileira selecionada.

Figura 5: Teste da funcionalidade de duplicação de uma fileira:



Fonte: Elaborado pelo autor (2025)

3.2.4 AJUSTES ESTRUTURAIS E OTIMIZAÇÃO DE COMPONENTES

Com a mecânica de seleção funcionando, a etapa seguinte consistiu em padronizar a comunicação entre os componentes 3D.
Foram revisados os componentes de renderização dos modelos dos montantes (Montante.jsx) e da geometria das longarinas (Longarina.jsx) para garantir que ambos compartilhassem corretamente o identificador rackId via propriedade userData.
Além disso, foram otimizadas as renderizações condicionais no componente CorteDeLaco.jsx, que gerencia a pré-visualização de racks durante o processo de criação. Essa otimização reduziu o consumo de memória e tornou o editor mais fluido, mesmo com múltiplas estruturas simultâneas na cena.

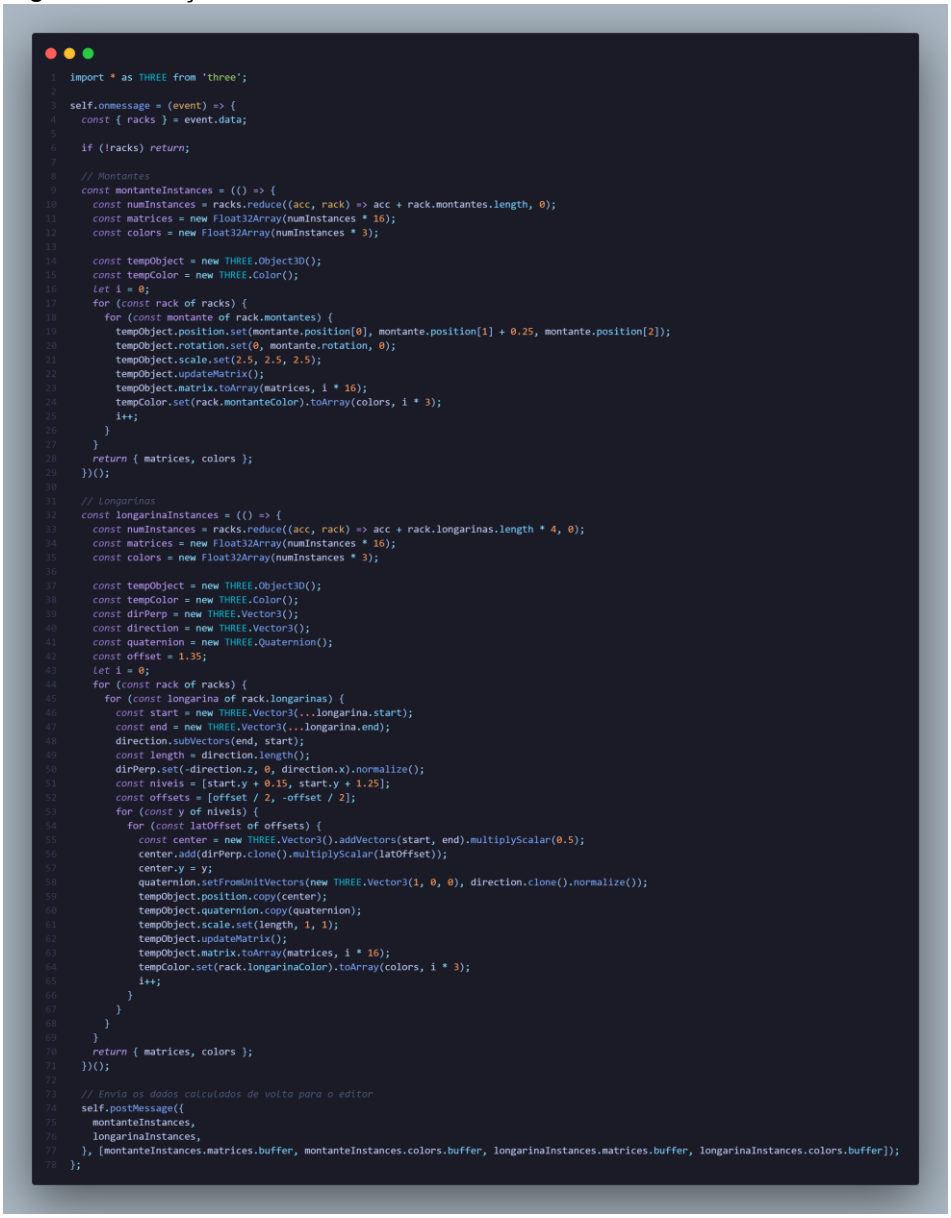
3.2.5 RENDERIZAÇÃO DE ALTA PERFORMANCE

A principal preocupação para o funcionamento dessa aplicação foi garantir a permanência da fluidez da interação e renderização, mesmo com múltiplos racks.
A forma tradicional de renderizar 1000 montantes seria a realização de 1000 eventos conhecidos como *draw calls* (chamadas de desenho) para a GPU. Cada uma dessas chamadas vai causando sobrecargas de comunicação entre a CPU e a GPU, tornando-se rapidamente em um gargalo, desencadeando quedas drásticas de FPS e por fim, o travamento total das funcionalidades do aplicativo. Para resolver esse problema foi preciso empregar um conjunto de procedimentos avançados de otimização.

Para a primeira técnica foi criado um componente separado chamado de InstancedRenderer, onde foi usado o componente InstancedMesh, nativo do React Three Fiber, que permite instanciamento para renderização. Ou seja, ao invés de fazer uma chamada individual para cada montante e longarina adicionados à cena, é possível, através desse componente, fazer apenas uma única chamada por tipo de objeto (uma chamada para todos os montantes, e outra para todas as longarinas), usando as listas de posições, rotações e cores atribuídos a esses elementos.

Outra técnica usada foi a criação de um web worker separado apenas para a lógica que envolvia o cálculo pesado das cores das instâncias e das matrizes de transformação, responsáveis por combinar posições, rotações e escalas dos objetos 3D em entidades matemáticas únicas. Assim que essas matrizes e cores são calculados, são armazenados em Float32Array. Seus ArrayBuffers subjacentes são por sua vez transferidos de volta para a interface principal (o componente do editor) usando postMessage com a lista de transferables. Isso é quase instantâneo, transferindo apenas a propriedade da memória e evitando a serialização e cópia dos dados. O componente InstancedRenderer.jsx recebe então todos esses arrays calculados nesse web worker e os aplica diretamente aos atributos instanceMatrix e instanceColor de InstancedMesh. A GPU então cuida do restante, desenhando todos os objetos de forma paralela e eficiente.

Figura 6: Criação do Web-Worker



```
import * as THREE from 'three';

self.onmessage = (event) => {
  const { racks } = event.data;

  if (!racks) return;

  // Montantes
  const montanteInstances = (() => {
    const numInstances = racks.reduce((acc, rack) => acc + rack.montantes.length, 0);
    const matrices = new Float32Array(numInstances * 16);
    const colors = new Float32Array(numInstances * 3);

    const tempObject = new THREE.Object3D();
    const tempColor = new THREE.Color();
    let i = 0;
    for (const rack of racks) {
      for (const montante of rack.montantes) {
        tempObject.position.set(montante.position[0], montante.position[1] + 0.25, montante.position[2]);
        tempObject.rotation.set(0, montante.rotation, 0);
        tempObject.scale.set(2.5, 2.5, 2.5);
        tempObject.updateMatrix();
        tempObject.matrix.toArray(matrices, i * 16);
        tempColor.set(rack.montanteColor).toArray(colors, i * 3);
        i++;
      }
    }
    return { matrices, colors };
  })();

  // Longarinas
  const longarinaInstances = (() => {
    const numInstances = racks.reduce((acc, rack) => acc + rack.longarinas.length * 4, 0);
    const matrices = new Float32Array(numInstances * 16);
    const colors = new Float32Array(numInstances * 3);

    const tempObject = new THREE.Object3D();
    const tempColor = new THREE.Color();
    const dirPerp = new THREE.Vector3();
    const direction = new THREE.Vector3();
    const quaternion = new THREE.Quaternion();
    const offset = 1.35;
    let i = 0;
    for (const rack of racks) {
      for (const longarina of rack.longarinas) {
        const start = new THREE.Vector3(...longarina.start);
        const end = new THREE.Vector3(...longarina.end);
        direction.subVectors(end, start);
        const length = direction.length();
        dirPerp.set(-direction.z, 0, direction.x).normalize();
        const niveis = [start.y + 0.15, start.y + 1.25];
        const offsets = [offset / 2, -offset / 2];
        for (const y of niveis) {
          for (const latOffset of offsets) {
            const center = new THREE.Vector3().addVectors(start, end).multiplyScalar(0.5);
            center.add(dirPerp.clone().multiplyScalar(latOffset));
            center.y = y;
            quaternion.setFromUnitVectors(new THREE.Vector3(1, 0, 0), direction.clone().normalize());
            tempObject.position.copy(center);
            tempObject.quaternion.copy(quaternion);
            tempObject.scale.set(length, 1, 1);
            tempObject.updateMatrix();
            tempObject.matrix.toArray(matrices, i * 16);
            tempColor.set(rack.longarinaColor).toArray(colors, i * 3);
            i++;
          }
        }
      }
    }
    return { matrices, colors };
  })();

  // Envia os dados calculados de volta para o editor
  self.postMessage({
    montanteInstances,
    longarinaInstances,
  }, [montanteInstances.matrices.buffer, montanteInstances.colors.buffer, longarinaInstances.matrices.buffer, longarinaInstances.colors.buffer]);
};
```

Fonte: Elaborado pelo autor (2025)

Outra abordagem foi a memoização de componentes, feita também em um componente separado do editor principal, envolvido em React.memo. Isso instrui o React a comparar as props com as da renderização anterior. Se estas não mudaram, o React simplesmente pula a fase de renderização para este componente e seus filhos, economizando poder de processamento da CPU.

A última e definitiva metodologia de otimização envolveu mudanças na geometria do próprio modelo 3D do montante. É inegável que, quanto mais polígonos ou faces forem necessários para compor um modelo 3D, mais pesado será para a GPU processá-los e renderizá-los. E o então modelo gerado pelo Tripo 3D Studio era composto por cerca de 421.188 polígonos. Felizmente, o Blender possui um modificador conhecido como *Decimate*, que reduz o número de polígonos de qualquer

malha, preservando ao máximo sua silhueta. Graças à essa ferramenta, o número de faces do modelo atual foi reduzido para 2895, deixando-o mais leve para renderizações massivas e sucessivas em qualquer navegador.

3.2.6 POLIMENTO DO SISTEMA DE CRIAÇÃO DE RACKS

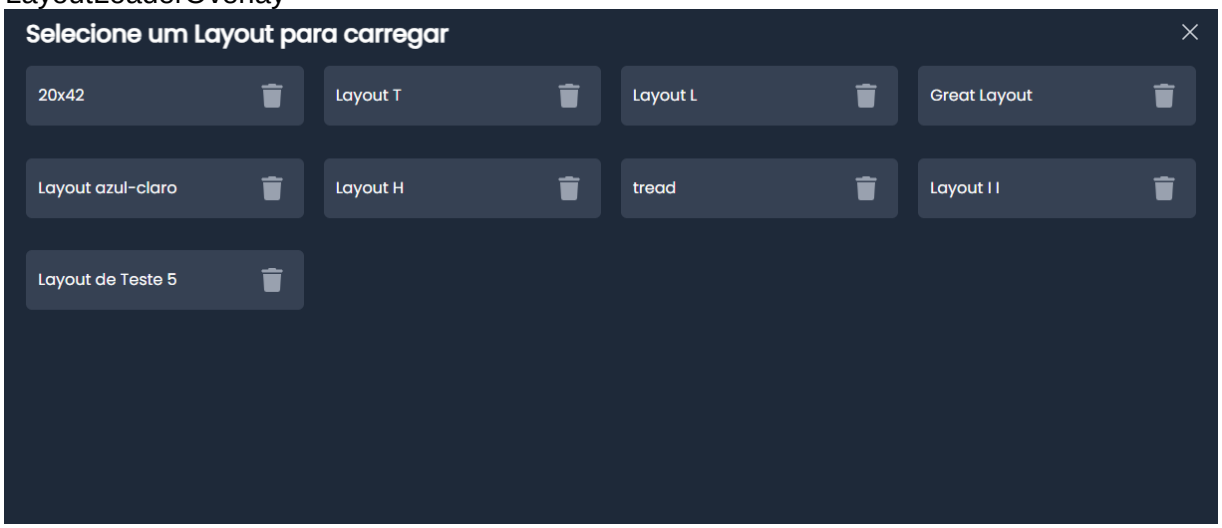
Nesta fase, foi finalizado o ciclo completo de construção das fileiras de porta-paletes. O componente CorteDeLaco foi separado da thread principal para garantir clareza no código e foi aprimorado para gerar automaticamente montantes intermediários e múltiplos andares, simulando fileiras completas de racks.

Foi implementado o ajuste dinâmico de divisões e andares por meio do scroll do mouse, tornando o processo de criação mais natural e eficiente. Além disso, o sistema passou a gerar identificadores únicos (UUID) para cada rack criado, através de uma função chamada de generateUUID, presente no MathUtils do Three.js, um objeto de utilitários para operações matemáticas que vão além do objeto padrão Math do JavaScript. Essa geração de IDs únicos permitiu a identificação e o controle individual das estruturas.

3.2.7 PERSISTÊNCIA DE DADOS (SALVAR E CARREGAR)

Para garantir a continuidade do trabalho do usuário, foi implementado um sistema de persistência baseado em API RESTful conectada ao MongoDB. Foram criados mais dois botões no menu lateral do editor principal: O botão "Salvar Layout", responsável por enviar ao servidor, através do método POST, as dimensões do plano, racks e os metadados de todas as malhas 3D contidas na cena, e o botão "Carregar Layout", que realiza uma requisição GET para recuperar os layouts disponíveis, que são listados em uma interface separada em um componente chamado de LayoutLoaderOverlay.jsx.

Figura 7: Listagem dos layouts salvos no Banco de dados através do componente LayoutLoaderOverlay



Fonte: Elaborado pelo autor (2025)

3.2.7.1 FUNCIONAMENTO DO BANCO DE DADOS

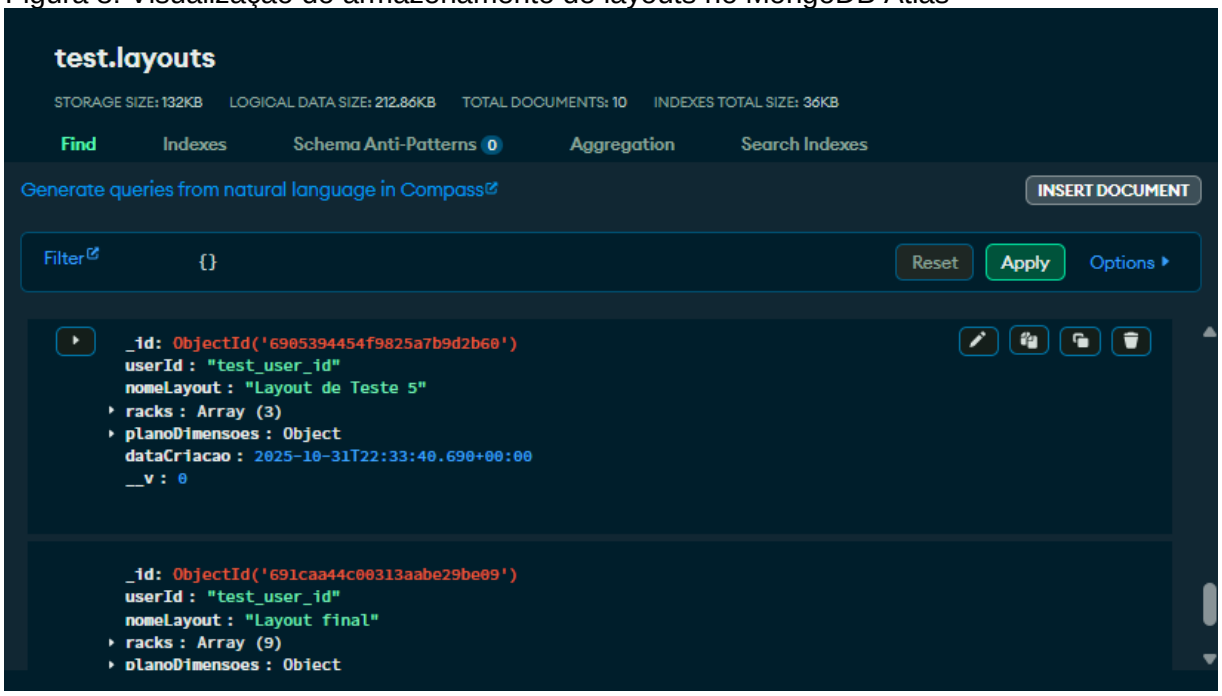
Armazenar uma cena 3D completa, com dezenas ou centenas de grupos contendo mais dezenas ou centenas de modelos tridimensionais foi possível apenas porque o MongoDB trabalha nativamente com documentos JSON, cuja estrutura se integra perfeitamente com o formato usado internamente pelo Three.js e pelo React para representação de objetos tridimensionais.

No Three.js, um objeto 3D é essencialmente uma estrutura contendo inúmeras informações, como posição (coordenadas nos eixos x, y e z), rotação, escala, identificadores, geometria, material, cor, e dados adicionais (userData), e no caso das fileiras de porta paletes, não é diferente. Cada objeto tridimensional na aplicação, seja um montante, uma longarina ou um rack completo, é composto por todas essas propriedades e parâmetros necessários para reconstrução da cena.
Conforme descreve a documentação do MDN, JavaScript é uma linguagem baseada na definição de objetos compostos por propriedades e métodos, que servem como unidades fundamentais de organização de dados e comportamento (Mozilla Foundation, 2025). Dessa forma, qualquer elemento criado em bibliotecas gráficas desenvolvidas em JavaScript sempre será representado internamente como um objeto.

Como esses dados são representados originalmente como objetos JavaScript, eles podem ser convertidos para JSON de forma direta com risco praticamente nulo de perda de informações, eliminando a necessidade de conversões complexas ou serializações específicas.

O MongoDB, por sua natureza orientada a documentos, permite armazenar estruturas de dados altamente hierárquicas, ou seja, documentos que contêm listas e subdocumentos aninhados. Essa característica é particularmente adequada para representar uma cena 3D, que também é uma estrutura hierárquica composta por elementos agrupados. Um único rack, por exemplo, contém diversos montantes e longarinas, com suas próprias propriedades internas. Em bancos relacionais tradicionais, essa representação exigiria múltiplas tabelas e chaves estrangeiras. Já no MongoDB, essa complexidade praticamente desaparece, permitindo que toda a estrutura seja armazenada em um único documento, o que se mostrou extremamente eficaz para este projeto.

Figura 8: Visualização do armazenamento de layouts no MongoDB Atlas



Fonte: Elaborado pelo autor (2025)

A utilização do Mongoose contribuiu para tornar esse processo mais robusto e organizado. Por meio da criação de quatro schemas no back-end, foi possível criar modelos que representam exatamente a estrutura dos objetos tridimensionais utilizados no front-end, funcionando como “espelhos” lógicos para a composição da cena 3D, validando automaticamente o formato dos dados recebidos, garantindo que posições, rotações, e os demais atributos estejam no padrão esperado antes de serem persistidos no banco de dados.

Figura 8: Código-fonte dos schemas no back-end

```
import mongoose from "mongoose";

const MontanteSchema = new mongoose.Schema({
    meshId: String, position: [Number], rotation: Number
});
const LongarinaSchema = new mongoose.Schema({
    id: String, start: [Number], end: [Number]
});
const RackSchema = new mongoose.Schema({
    id: String,
    color: String,
    numDivisoes: Number,
    numAndares: Number,
    montantes: [MontanteSchema],
    longarinas: [LongarinaSchema],
});
const LayoutSchema = new mongoose.Schema({
    userId: String,
    nomeLayout: String,
    dataCriacao: {type: Date, default: Date.now},
    racks: [RackSchema],
    planoDimensoes: {
        largura: {type: Number},
        comprimento: {type: Number}
    }
});
export default mongoose.model("Layout", LayoutSchema);
```

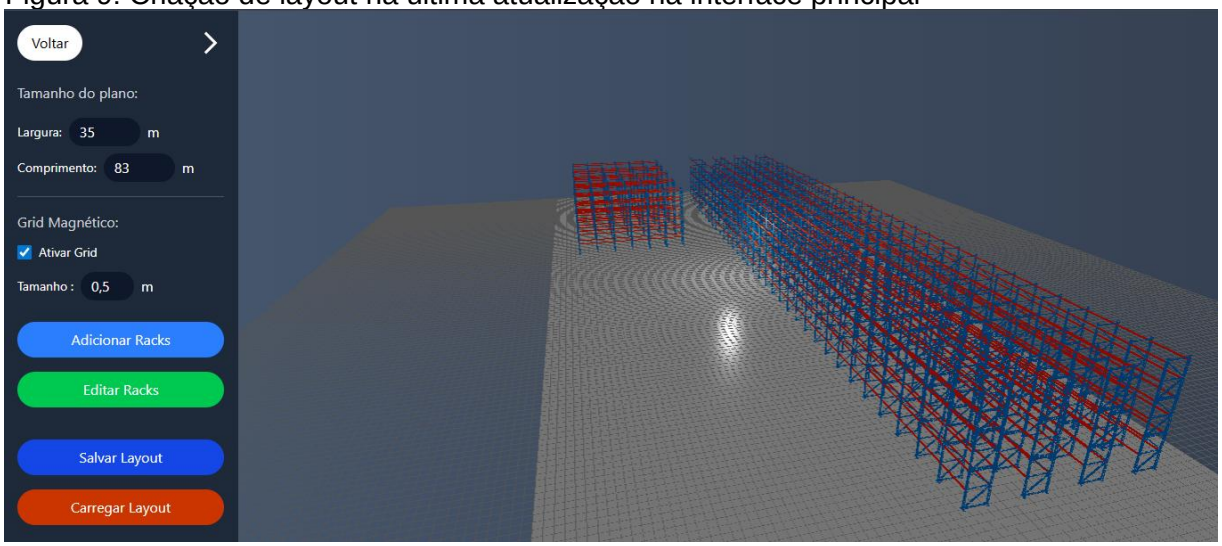
Fonte: Elaborado pelo autor (2025).

Outro fator fundamental é que o projeto não armazena arquivos tridimensionais completos, como modelos GLB ou GLTF. Ao invés disso, o sistema salva apenas os dados lógicos dos modelos, como coordenadas, quantidades, divisões, andares, e número de fileiras. Ao carregar um layout salvo, o React e o Three.js utilizam essas informações para reconstruir dinamicamente a cena, recriando as geometrias diretamente no navegador. Essa abordagem garante que os layouts permaneçam leves, rápidos de carregar e fáceis de versionar.

4. CONSIDERAÇÕES FINAIS

O desenvolvimento do presente projeto permitiu demonstrar a viabilidade da criação de um editor tridimensional interativo voltado ao planejamento de layouts de armazéns, com foco na modelagem e manipulação de fileiras de porta-paletes em ambiente web. Ao longo do processo, foi possível integrar tecnologias modernas como React, JavaScript e MongoDB de forma coesa, resultando em um sistema funcional capaz de criar, editar, visualizar e armazenar representações tridimensionais de estruturas completas de armazenagem em banco de dados.

Figura 9: Criação de layout na última atualização na interface principal



Fonte: Elaborado pelo autor (2025).

O aplicativo desenvolvido contemplou plenamente o objetivo principal: oferecer uma ferramenta digital capaz de representar, de maneira clara e precisa, elementos estruturais encontrados em operações logísticas reais. A implementação da mecânica de criação de fileiras, somada às funcionalidades de seleção, edição e duplicação, constituiu um ambiente flexível e intuitivo, permitindo que o usuário construa e reorganize layouts com precisão. Além disso, a adoção de técnicas avançadas de otimização, como renderização instanciada e uso de Web Workers, viabilizou a manipulação de grandes quantidades de objetos tridimensionais sem comprometer o desempenho da aplicação.

Outro resultado significativo foi a implementação do sistema de persistência de dados por meio de uma API REST integrada ao MongoDB. Tal recurso permitiu que layouts completos fossem armazenados e posteriormente reconstruídos a partir de documentos JSON, demonstrando que bancos orientados a documentos são os melhores para representar estruturas hierárquicas complexas, como cenas tridimensionais. A construção de schemas específicos no back-end proporcionou organização, coerência e segurança ao processo de armazenamento.

Embora o sistema desenvolvido seja funcional e apresente grande potencial de uso prático, algumas limitações foram identificadas. A aplicação ainda não possui um nome oficial ou uma página inicial, e como utiliza WebGL, pode apresentar problemas severos de inicialização ou renderização em versões muito inferiores de navegadores, e o editor, no momento, não contempla a criação ou outras interações com diferentes tipos de estruturas de armazenagem, como mezaninos, estantes convencionais,

cantiléver ou porta-bobinas, e não possui sistema de autenticação para uso individualizado por diferentes usuários. Contudo, essas limitações não representam falhas, mas sim oportunidades de expansão, considerando que a arquitetura adotada permite incorporar outros modelos 3D, novas rotinas de edição e novos fluxos de interação sem necessidade de reescrever o núcleo do sistema.

O aplicativo, portanto, cumpre seu propósito acadêmico ao demonstrar integração entre áreas distintas da computação e relevância prática no contexto da logística, além de estabelecer bases sólidas para evoluções futuras, permitindo que o sistema possa se tornar, com continuidade e refinamento, uma ferramenta robusta que permita o apoio de planejamentos de armazéns.

Assim, o trabalho realizado não apenas comprova a aplicabilidade da proposta inicial, mas também evidencia o grande potencial da computação gráfica aliada ao uso de bancos de dados não-relacionais e tecnologias web modernas como instrumentos para apoiar processos logísticos, oferecendo ganhos de precisão, agilidade e confiabilidade na tomada de decisões relacionadas ao uso do espaço físico.

REFERÊNCIAS

CALASANS DE SOUZA, Elaine; OLIVEIRA, Marcus Rogério de. **Comparativo entre os bancos de dados MySQL e MongoDB: quando o MongoDB é indicado para o desenvolvimento de uma aplicação**. Interface Tecnológica, v. 16, n. 2, p. 38-48, 2019. DOI: 10.31510/infa.v16i2.664.

SMELOV, Aleksandr. **Integration of interactive 3D models into React-based application**. 2024. 47 f. Bachelor Thesis (Bachelor of Engineering, Automation Engineering) - Seinäjoki University of Applied Sciences, Seinäjoki, 2024.

COSTA, Nuno Daniel Oliveira. **Projeto de layouts de armazéns e melhoria dos processos logísticos**. 2020. 111 f. Relatório de Projeto (Mestrado em Engenharia e Gestão Industrial) - Departamento de Economia, Gestão, Engenharia Industrial e Turismo, Universidade de Aveiro, Aveiro, 2020.

MOZILLA FOUNDATION. **JavaScript**. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/JavaScript. Acesso em: 7 out. 2025.

COMAQ PARANÁ. **Porta-pallets.** 2024? [Imagem]. Disponível em: https://comaqparana.com.br/estruturas/porta-pallets/porta-pallets. Acesso em: 17 nov. 2025.

THREE.js. **Documentation**. Disponível em: https://threejs.org/docs/. Acesso em: 18 nov. 2025.

NODE.js. **Página inicial.** Disponível em: https://nodejs.org/pt. Acesso em: 18 nov. 2025.

EXPRESS. **Express - Node.js web application framework.** Disponível em:

https://expressjs.com/. Acesso em: 18 nov. 2025.

MONGODB. **Documentação do MongoDB**. Disponível em: https://www.mongodb.com/pt-br/docs/. Acesso em: 18 nov. 2025.