INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SUL-RIO-GRANDENSE
Campus Visconde da Graça

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SUL-RIO-GRANDENSE – CÂMPUS PELOTAS - VISCONDE DA GRAÇA
CURSO TÉCNICO EM DESENVOLVIMENTO DE SISTEMAS

HYPERSHELL: UM TERMINAL WEB SIMULADO

Cristian Camillo Buaez

Pelotas, 2025.

Cristian Camillo Buaez

HYPERSHELL: UM TERMINAL WEB SIMULADO

Projeto de Desenvolvimento de Sistema apresentado como requisito na disciplina de Metodologia da Pesquisa II do curso Técnico em Desenvolvimento de Sistemas, do Instituto Federal Sul-Rio-Grandense – *Campus* Pelotas - Visconde da Graça.

Prof(a): Verlani Timm Hinz.

Pelotas, 2025.

SUMÁRIO

1. INTRODUÇÃO..... 4
2. TEMA..... 4
3. MOTIVAÇÕES..... 4
4. OBJETIVOS..... 5
4.1. OBJETIVO GERAL..... 5
4.2. OBJETIVOS ESPECÍFICOS..... 5
5. ESPECIFICAÇÃO DE REQUISITOS..... 5
5.1. MÉTODOS DE ESPECIFICAÇÃO DE REQUISITOS..... 6
5.2. REQUISITOS FUNCIONAIS..... 6
5.3. REQUISITOS NÃO FUNCIONAIS..... 7
6. MODELAGEM..... 8
6.1. MODELAGEM DE CASOS DE USO..... 8
6.2. MODELAGEM CONCEITUAL DO BANCO DE DADOS..... 10
6.3. MODELAGEM LÓGICA DO BANCO DE DADOS..... 11
7. TECNOLOGIAS UTILIZADAS..... 12
8. DESCRIÇÃO DO SISTEMA..... 13
8.1. Tela Inicial do terminal..... 13
8.2. Listagem de Comandos utilizando o comando “help”..... 14
8.3. Listagem de Diretórios e Arquivos através do comando “ls”..... 14
8.4. Visualização simplificada do IP fixo com o comando “ifconfig”..... 15
9. CONSIDERAÇÕES FINAIS..... 16
10. REFERÊNCIAS..... 17
Apêndice:..... 18
Estrutura do Componente Principal(Terminal.vue)..... 18
Estrutura SQL(Criação da base no PgAdmin)..... 22

RESUMO

O HyperShell é um terminal web simulado desenvolvido com o intuito de despertar o interesse dos usuários pelo ambiente de linha de comando. O projeto busca transmitir a sensação de imersão e controle característica dos terminais reais, resgatando a curiosidade e o fascínio por essa forma de interação direta com o sistema. A proposta nasceu do apreço pessoal do desenvolvedor por interfaces de terminal e da vontade de compartilhar essa experiência de maneira acessível e envolvente. Em um cenário onde as interfaces gráficas dominam, o HyperShell se destaca por valorizar a simplicidade e a essência da comunicação textual entre usuário e máquina. A aplicação foi desenvolvida utilizando tecnologias modernas, como Vue.js, Node.js, PostgreSQL e Xterm.js, que em conjunto tornam possível a simulação realista de um terminal interativo no navegador. Mais do que um sistema funcional, o HyperShell representa uma forma de expressão técnica e criativa, voltada a despertar a curiosidade e o interesse por esse universo.

Palavras-chave: Terminal Web, Interface de Linha de Comando (CLI), Xterm.js, PostgreSQL, Vue.js

1. INTRODUÇÃO

Nos tempos atuais, a Tecnologia da Informação se tornou cada vez mais visual, interativa e automatizada. Interfaces gráficas dominam o cenário do desenvolvimento e do aprendizado em computação, tornando a experiência do usuário mais acessível e intuitiva. Contudo, por trás das cores e layouts existe um universo poderoso e incrível, as *Command Line Interfaces* (CLI).

Os terminais, muitas vezes visto como uma ferramenta antiga ou complexa, é na verdade uma das formas mais diretas e expressivas de interação entre o ser humano e o computador. Ele desperta a curiosidade de quem busca compreender o funcionamento real dos sistemas, promovendo um contato mais próximo com a lógica e o controle que sustentam o ambiente digital.

O Hypershell surgiu através da ideia de resgatar esse fascínio pelo terminal e de apresentar ao usuário a arte das interfaces de comando. A proposta foi desenvolver um terminal web simulado, acessível e imersivo.

2. TEMA

Terminal Virtual com elementos gerais compostos em Interface de Linha de Comando (CLI).

3. MOTIVAÇÕES

O desenvolvimento do HyperShell surgiu do interesse pessoal recente do desenvolvedor por interfaces de linha de comando, e busca compartilhar este universo com outros usuários. Este projeto busca trazer uma experiência prática e interativa, capaz de despertar a curiosidade por meio da linha de comando.

4. OBJETIVOS

4.1. OBJETIVO GERAL

Desenvolver um ambiente sandbox que fomente o interesse em simuladores de terminal como também proporcionar uma experiência que introduza conceitos essenciais de uma Interface de Linha de Comando.

4.2. OBJETIVOS ESPECÍFICOS

Para atingir o objetivo geral, devem ser contemplados os seguintes objetivos específicos:

- liberar descrição de uso de cada ferramenta disponível no sistema;
- integrar o Xterm.js como base de desenvolvimento do ambiente;
- proporcionar uma experiência dinâmica onde no decorrer da aplicação será possível interagir com diversas ferramentas de navegação e uso diário

5. ESPECIFICAÇÃO DE REQUISITOS

O levantamento de requisitos funciona como o alicerce indispensável para qualquer iniciativa de desenvolvimento ou projeto. É nesta fase crucial que se mergulha na investigação e compreensão das dores, desejos e restrições dos futuros usuários e demais partes interessadas. O processo envolve um diálogo ativo e análise criteriosa para transformar ideias abstratas e necessidades latentes em um conjunto claro e funcional de especificações, que servirão como mapa para a construção da solução desejada.

5.1. MÉTODOS DE ESPECIFICAÇÃO DE REQUISITOS

Conforme analisado em diferentes projetos similares ao HyperShell, alguns requisitos são essenciais para o desenvolvimento da aplicação e para que a mesma possa trazer as principais funcionalidades para uma melhor experiência ao usuário.

O levantamento de requisitos se deu pela análise destes projetos já realizados pela comunidade, com objetivo de alcançar uma simulação fiel visualmente aos terminais convencionais de sistemas operacionais de distribuições Linux.

Dentre os atributos essenciais de todos os projetos estava a fluidez, responsividade visual, persistência de navegação, além dos típicos comandos que são a base de toda Interface de Linha de Comando.

5.2. REQUISITOS FUNCIONAIS

Os requisitos funcionais descrevem as ações e comportamentos que o sistema deve executar para atender às necessidades do usuário. Eles definem o que o sistema faz, representando suas funcionalidades principais, como criação de diretórios, manipulação de arquivos ou execução de comandos. Esses requisitos orientam diretamente o desenvolvimento, pois especificam as operações essenciais que o usuário poderá realizar dentro do sistema.

Tabela 1: Requisitos Funcionais

RF1	Gerenciar Diretórios	O sistema deve permitir a listagem e criação de diretórios no sistema.
RF2	Visualizar Conteúdo	O sistema deve permitir que o usuário visualize o conteúdo de arquivos em diferentes diretórios.
RF3	Gerenciar Arquivos	O sistema deve permitir que o usuário crie e exclua arquivos dentro de diretórios no sistema.

RF4	Gerenciar IP	O sistema deve permitir que o usuário edite seu próprio endereço IP.
RF5	Gerenciar Usuários	O sistema deve permitir a criação e exclusão de usuários.
RF6	Utilizar Comandos	O sistema deve permitir que o usuário possa acessar todos os comandos disponíveis e utilizá-los a qualquer momento.

Fonte: Autoral

5.3. REQUISITOS NÃO FUNCIONAIS

Os requisitos não funcionais definem como o sistema deve se comportar, estabelecendo características relacionadas à qualidade e ao desempenho. Eles abrangem aspectos como segurança, usabilidade, disponibilidade e confiabilidade. Diferentemente dos requisitos funcionais, eles não descrevem funções específicas, mas sim propriedades que garantem uma experiência adequada e eficiente ao usuário e ao ambiente em que o sistema opera.

Tabela 2: Requisitos não Funcionais

RNF1	Desempenho Local	A aplicação deve ser executada de forma fluida e responsiva em máquinas com especificações mínimas, garantindo baixa latência na resposta aos comandos.
RNF2	Disponibilidade	A aplicação deve estar disponível para uso pelo menos 99% do tempo, evitando quedas frequentes ou períodos de indisponibilidade.
RNF3	Confiabilidade	A aplicação deve manter a integridade do ambiente simulado mesmo após várias interações, prevenindo bugs ou comportamentos inesperados.

RNF4	Usabilidade	A interface deve ser intuitiva e acessível mesmo para usuários iniciantes, com dicas visuais e sugestões de comandos quando necessário.
RNF5	Manutenibilidade	O código da aplicação deve ser modular, bem documentado e fácil de manter e atualizar, permitindo a adição de novos comandos ou funcionalidades no futuro.
RNF6	Estabilidade	A aplicação deve manter a estabilidade mesmo se ocorrerem erros pontuais (ex: falha de rede, comando mal formatado), garantindo que o usuário não perca seu progresso.

Fonte: Autoral

6. MODELAGEM

6.1. MODELAGEM DE CASOS DE USO

A modelagem de casos de uso tem como objetivo representar as interações entre o usuário e o sistema, descrevendo de forma clara quais funcionalidades são acessadas e em que contexto elas ocorrem. Por meio de diagramas e descrições textuais, essa modelagem ajuda a identificar os atores envolvidos, seus objetivos e os fluxos principais de execução, oferecendo uma visão geral do comportamento esperado do sistema, conforme demonstra a Figura 1.

ROOT: este é o ator inicial do sistema. Ele representa qualquer pessoa que acessa o terminal web, um usuário comum. É o responsável por interagir diretamente com o sistema para executar diversas ações. “Root” vem de referência simbólica ao usuário principal dos sistemas baseados em Unix/Linux, que possui acesso total a todos os comandos e diretórios.

Funções do Ator:

Gerenciar Usuários:

Descrição: Root pode criar e deletar usuários dentro do sistema.

Executar Comandos:

Descrição: Root pode digitar comandos no terminal para realizar diversas operações.

Gerenciar Diretório:

Descrição: Root tem permissão para navegar por qualquer diretório no sistema.

Gerenciar Arquivos:

Descrição: Root pode criar ou remover quaisquer arquivos dentro de diretórios no sistema.

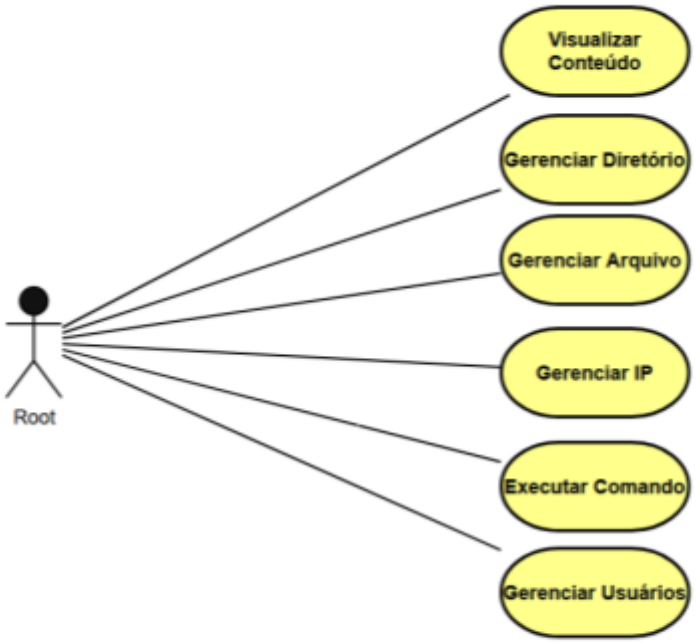
Gerenciar IP:

Descrição: Root pode visualizar o endereço IP e definir IP fixo/dinâmico.

Visualizar Conteúdo:

Descrição: Root pode visualizar o conteúdo de arquivos existentes no sistema através do comando cat.

Figura 1 - Modelo de Caso de Uso

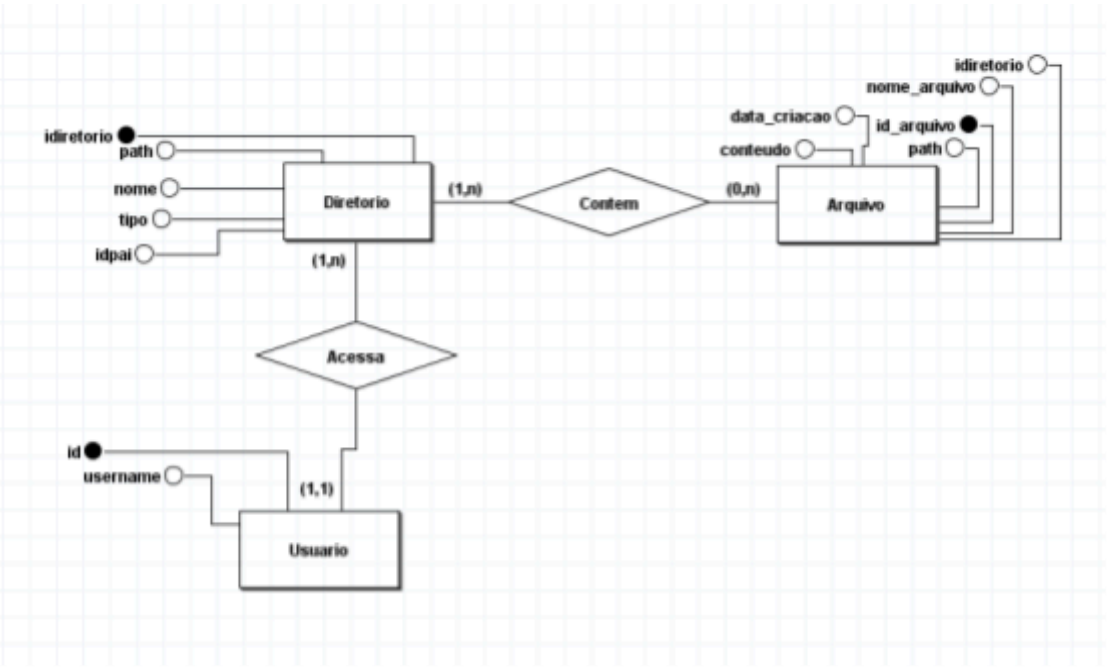


Fonte: Autoral

6.2. MODELAGEM CONCEITUAL DO BANCO DE DADOS

O modelo relacional tem como objetivo organizar os dados do sistema em estruturas chamadas tabelas, possibilitando uma forma clara e consistente de armazenar informações. Conforme apresentado na Figura 2, esse modelo fornece uma visão estruturada das entidades e de suas conexões dentro do banco de dados, servindo como fundamento para a implementação lógica e física.

Figura 2 - Modelo Conceitual

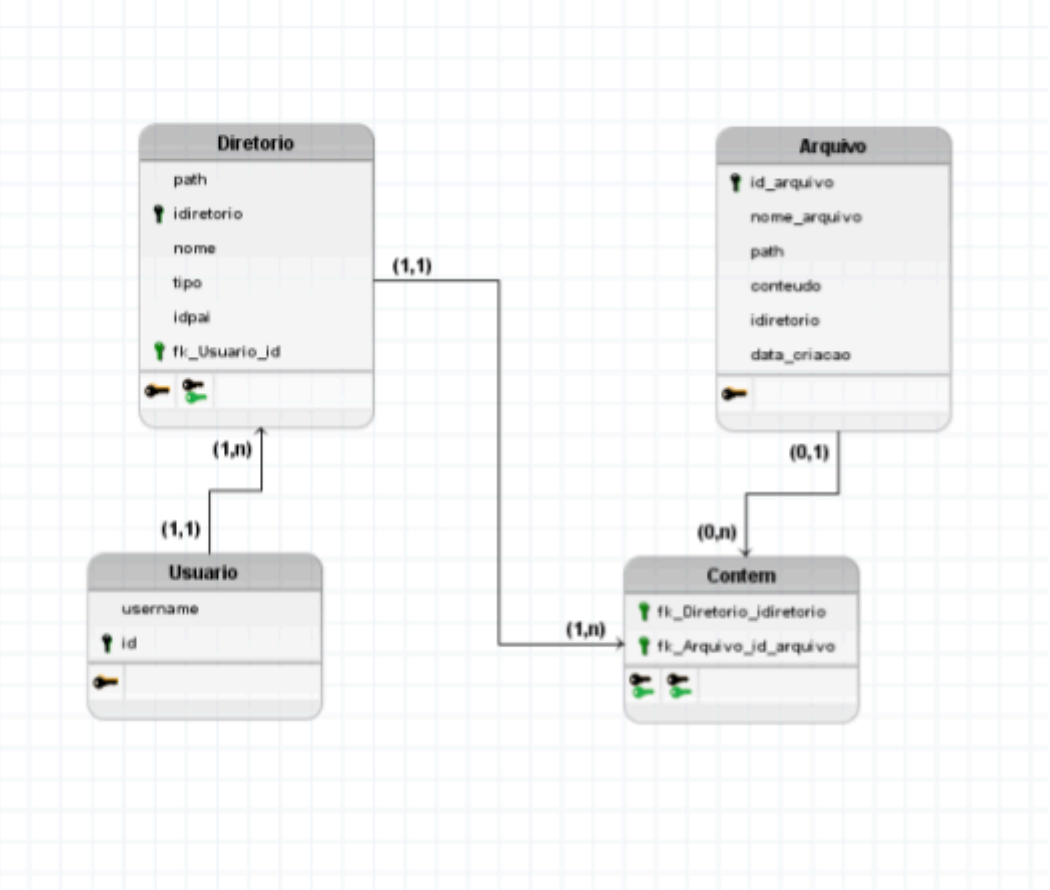


Fonte: Autoral

6.3. MODELAGEM LÓGICA DO BANCO DE DADOS

O modelo lógico tem como objetivo transformar a representação conceitual em uma estrutura mais próxima do que será efetivamente implementado no banco de dados. Assim como mostra na Figura 3, o modelo lógico descreve de forma detalhada como os dados serão organizados internamente, funcionando como uma ponte entre a visão conceitual e a criação do banco de dados propriamente dito.

Figura 3 - Modelo Conceitual



Fonte: Autoral

7. TECNOLOGIAS UTILIZADAS

O sistema foi desenvolvido utilizando ferramentas robustas e eficientes para facilitar o processo evolutivo de produção, segue a listagem das tecnologias fundamentais:

- Vue.js é o framework principal para o funcionamento da aplicação, ótimo para construção de modelos single-page(página única) que permite a criação de novos componentes que podem servir como janelas que sobrepõem o componente principal para expor uma interface nova. Além disso, sua reatividade facilita toda parte de atualização dos elementos, tornando automático;

- JavaScript atua como a base da lógica de interação no *frontend*. Dentro dos arquivos .vue, foi empregado para controlar eventos, capturar comandos digitados no terminal. Além disso, o JavaScript é responsável pela integração direta com o Xterm.js, permitindo que a digitação, impressão e atualização da interface ocorra de forma natural e responsiva;

- Xterm.js foi escolhido como biblioteca central para emular o terminal dentro da aplicação web. Ele oferece suporte a renderização rápida, manipulação de texto, histórico de comandos e compatibilidade com WebSockets. Isso possibilita a criação de um terminal realista, aproximando a experiência aos consoles utilizados em sistemas Unix;

- O WebSocket foi utilizado para viabilizar comunicação bidirecional em tempo real entre o navegador e o servidor. Essa tecnologia é essencial para garantir que cada comando digitado no terminal seja enviado instantaneamente ao backend, retornando a resposta sem recarregar a página. A API de WebSockets cria uma conexão persistente, permitindo um fluxo contínuo de informações e garantindo baixa latência na simulação do ambiente;

- PostgreSQL foi adotado como sistema de gerenciamento de banco de dados para estruturar as pastas e usuários. Sua robustez, suporte a chaves estrangeiras e flexibilidade permitem modelar diretórios hierárquicos e relacionamentos complexos;

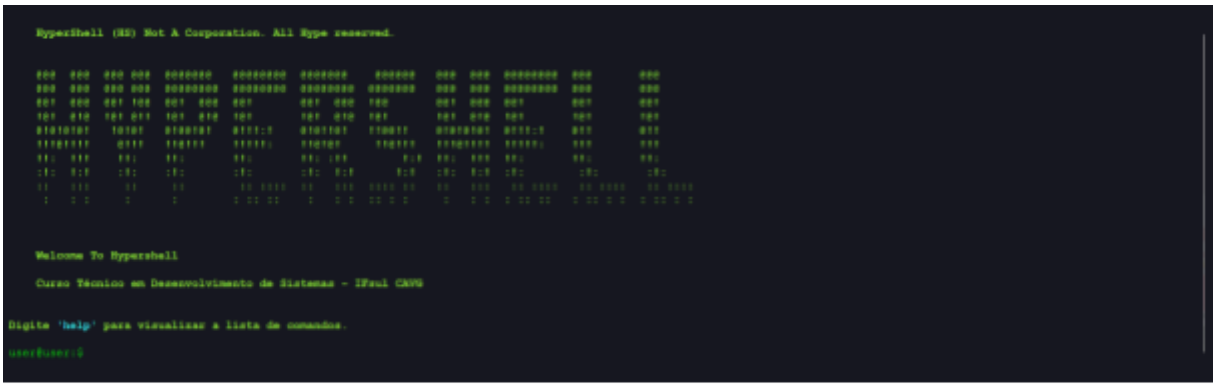
- Node.js foi utilizado para implementar o backend da aplicação. Ele atua como servidor responsável por receber os comandos enviados pelo terminal e processá-los. A natureza assíncrona e orientada a eventos do Node.js torna-o adequado para cenários onde muitas operações pequenas ocorrem em sequência, como interpretar comandos, manipular dados e fazer consultas ao banco.

8. DESCRIÇÃO DO SISTEMA

8.1. Tela Inicial do terminal

- Primeira tela (Figura 4) apresenta o banner de boas vindas do sistema quando o usuário conecta ao terminal.

Figura 4 - Tela inicial



Fonte: Autoral

8.2. Listagem de Comandos utilizando o comando “help”

- A Figura 5 apresenta a listagem de comandos disponíveis no sistema, todos com suas respectivas descrições.

Figura 5 - Comando help

Fonte: Autoral

- banner: Exibe um banner textual no terminal, geralmente usado para mensagens ou identificação do sistema.

- cat: Exibe o conteúdo de arquivos. Também pode ser usado para visualizar informações do sistema, como cat /etc/passwd, que lista os usuários cadastrados.

- cd: Permite a navegação entre diretórios.

- clear: Limpa a tela do terminal.

- deluser: Exclui um usuário do sistema.

- github: Acessa o GitHub do criador do sistema.

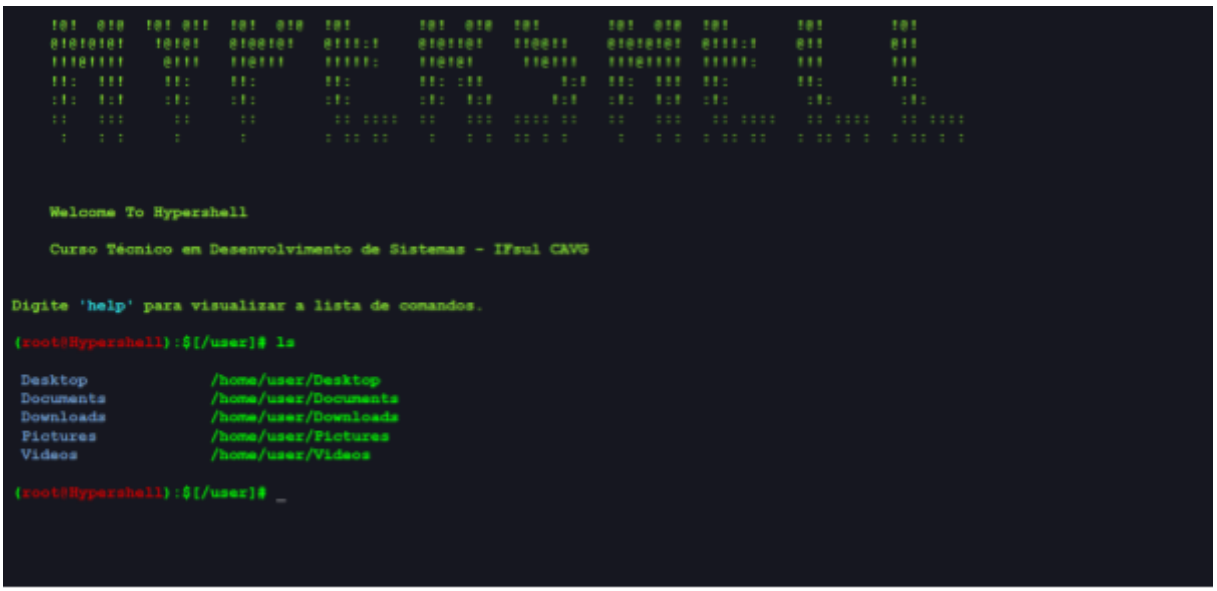
- help: Exibe a lista de comandos disponíveis no sistema.
- ifconfig: Visualizar e alterar IP.

- linkedin: Direciona para o perfil LinkedIn do criador do sistema.
- ls: Lista arquivos e diretórios no caminho atual.
- msfconsole: Interface para execução de scripts e atividades de pentest.
- pwd: Exibe o diretório atual.
- sherlock: Ferramenta de OSINT para buscar usuários em redes sociais.
- useradd: Adicionar novos usuários ao sistema.
- whoami: Exibe o nome do usuário atual.

8.3. Listagem de Diretórios e Arquivos através do comando “ls”

- A Figura 6 apresenta a listagem de diretórios e arquivos acessíveis no caminho atual do usuário, ao acessar pela primeira vez, o usuário está no diretório /user por padrão.

Figura 6 - Comando ls

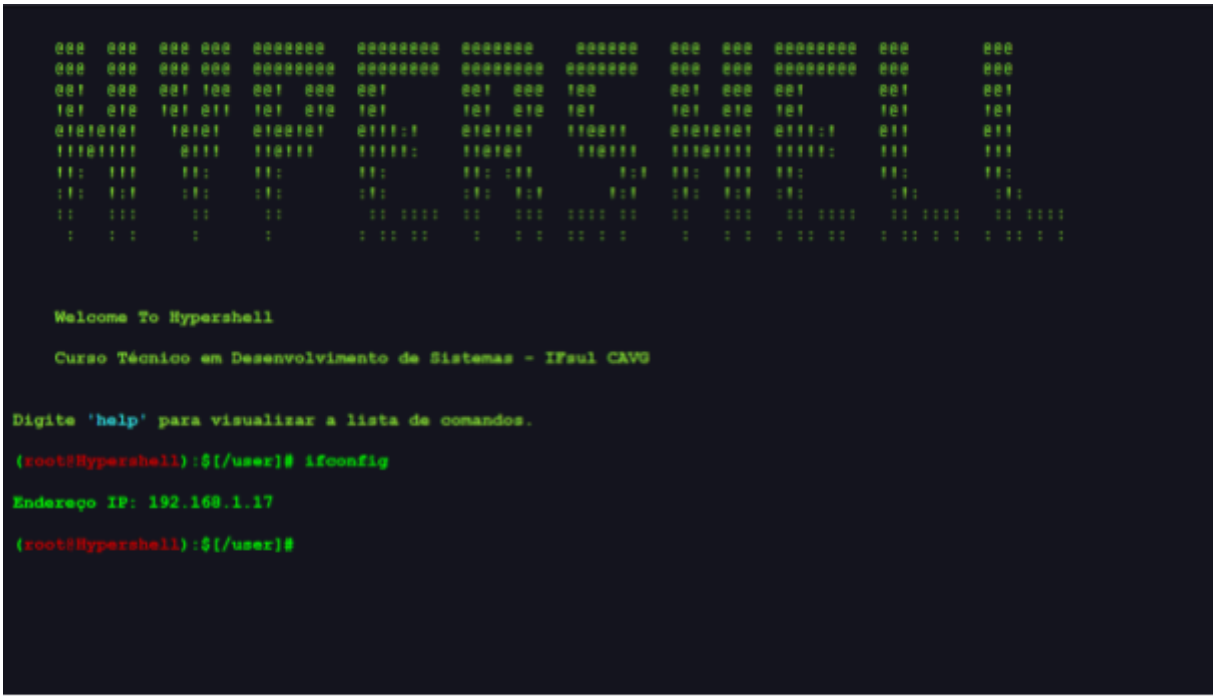


Fonte: Autoral

8.4. Visualização simplificada do IP fixo com o comando “ifconfig”

- A Figura 7 mostra a execução do comando “ifconfig”, que permite visualizar seu endereço IP e alterá-lo quando quiser, basta adicionar o novo endereço após o comando.

Figura 7 - Comando ifconfig



Fonte: Autoral

9. CONSIDERAÇÕES FINAIS

No decorrer do desenvolvimento do Trabalho de Conclusão de Curso, foram encontrados diversos desafios, sobretudo no curto espaço de tempo disponível para a execução do projeto. A construção da base revelou-se particularmente exigente, gerando uma sensação inicial de pressão e dúvida quanto à conclusão final. Contudo, foi nesta mesma etapa onde foi compreendido a importância esmagadora de um padrão de desenvolvimento em um sistema completo, foi com esse mesmo novo entendimento que tornou-se possível manter atualizações constantes após a primeira fase.

Dentre os demais obstáculos na aplicação do tema, conseguir uma utilidade para o banco de dados tornou limitante a escalabilidade do código e em conclusão desnecessário para futuros projetos relacionados, visto que não é necessário o armazenamento de muitas informações relevantes.

Para concluir, este foi meu primeiro projeto de desenvolvimento de um Shell, uma ferramenta que alia robustez técnica a uma experiência de uso fluida e intuitiva diretamente no navegador. Com este sistema, reacendeu-se o desejo de aprender sobre novas tecnologias e distribuições linux, ao longo do contato com diversos projetos similares e a descoberta de uma comunidade estabelecida que desenvolve idéias incríveis em torno da criação e manipulação de Interfaces de Linha de Comando.

10. REFERÊNCIAS

OCCUPYTHEWEB. Linux Basics for Hackers: Getting Started with Networking, Scripting, and Security in Kali. No Starch Press. 2018. Disponível em: https://kea.nu/files/textbooks/humblesec/linuxbasicsforhackers.pdf.

INSTITUTO FEDERAL SUL-RIO-GRANDENSE. Introdução ao Modelo de Entidade Relacionamento. Disponível em: apnp.ifsul.edu.br/pluginfile.php/1437285/mod_resource/content/2/BD_cap2.pdf Acesso em 18 de novembro de 2025

XTERM. Xterm.js: A terminal for the Web. Disponível em: https://github.com/xtermjs/xterm.js. Acesso em: 20 de novembro de 2025.

FOREST KNIGHT. FK CODES. Disponível em: https://fkcodes.com/. Acesso em: 20 de novembro de 2025.

Apêndice:

Estrutura do Componente Principal (Terminal.vue)

```
<template>
<div ref="HyperShell" class="terminal-container"></div>
</template>

<script setup>
import { ref, onMounted, onUnmounted } from 'vue';
import { FitAddon } from 'xterm-addon-fit';
import { Terminal } from 'xterm';
import 'xterm/css/xterm.css';
import banner from '../../assets/MainBanner.txt?raw';
import smallBanner from '../../assets/SmallBanner.txt?raw';
import { criaHandler, handleMessage } from '../../utils/messageHandler.js';
import ws from '../../utils/initWs';
// import { initTerminal } from '../../utils/initTerminalEvent.js';

const HyperShell = ref(null);
let term;
let fitAddon;

let resizeTimeout;

const atualizaInput = (newVal) => {
    userInput = newVal;
  };



const canFitBanner = (cols) => {
  const bannerLines = banner.split('\n');
  const maxLineLength = Math.max(...bannerLines.map(line => line.length));
```

```
    return cols >= maxLineLength;
  };

  const time = new Date()

  const displayBanner = () => {
    const cols = term.cols;

    if (canFitBanner(cols)) {
      term.write(`\x1b[1;32m${time}\x1b[0m`)
      term.write(`\x1b[1;32m${banner}\x1b[0m \r\n`);
        term.writeln(`\x1b[1;32mDigite \x1b[36m'help'\x1b[1;32m para visualizar a lista de comandos.\x1b[0m`)
    } else {
      term.write(`\x1b[1;32m${smallBanner}\x1b[0m `);
        term.writeln(`\x1b[1;32mDigite \x1b[36m'help'\x1b[1;32m para visualizar a lista de comandos.\x1b[0m`)
    }


  };

  onMounted(() => {
    term = new Terminal({
      cursorBlink: true,
      cursorStyle: "underline",
      cursorInactiveStyle: "block",
      fontSize: 16,
      fontWeight: 900,
      rightClickSelectsWord: true,
      convertEol: true,
      theme: {
        background: '#161720',
        foreground: '#00FF00',
```

```
    selectionBackground: '#2c2e3d'
  }
});

fitAddon = new FitAddon();
term.loadAddon(fitAddon);


term.open(HyperShell.value);
fitAddon.fit();
term.focus();

displayBanner();

let userInput = '';

const handlers = criaHandler(term, displayBanner);

ws.onmessage = (m) => {
   const parsed = JSON.parse(m.data);

  handleMessage(m, handlers);
};



term.onData((data) => {

  if (data === '\x1b[A') return
  if (data === '\x1b[B') return
  if (data === '\x1b[C') return
  if (data === '\x1b[D') return

  if (data === '\r') {
```

```
      term.write("\r\n\n");
      ws.send(userInput);
      userInput = '';
    } else if (data === '\u007F') {
      if (userInput.length > 0) {
        userInput = userInput.slice(0, -1);
        term.write('\b \b');
      }
    } else {
      userInput += data;
      term.write(data);
    }
  });

  const handleResize = () => {
    clearTimeout(resizeTimeout);
    resizeTimeout = setTimeout(() => {
      fitAddon.fit();
    }, 100);
  };

  window.addEventListener('resize', handleResize);

  onUnmounted(() => {
    window.removeEventListener('resize', handleResize);
    if (ws) ws.close();
    if (term) term.dispose();
  });
});
</script>

<style scoped>
.terminal-container {
  width: 100%;
```

```
  height: 100%;
  padding: 10px;
  box-sizing: border-box;
  box-sizing: border-box;
  zoom: 1.25;
}
</style>
```

Estrutura SQL (Criação da base no PgAdmin)

Base de dados: “Terminal”

– Estrutura da tabela de usuários

```
CREATE TABLE IF NOT EXISTS public.usuarios
(
    id integer NOT NULL DEFAULT nextval('usuarios_id_seq'::regclass),
        username character varying(100) COLLATE pg_catalog."default" NOT NULL,
                data_criacao timestamp without time zone DEFAULT CURRENT_TIMESTAMP,
    CONSTRAINT usuarios_pkey PRIMARY KEY (id)
)
```

– Estrutura da tabela de diretórios

```
CREATE TABLE IF NOT EXISTS public.diretorio
(
    idiretorio integer NOT NULL,
    nome character varying COLLATE pg_catalog."default" NOT NULL,
    path character varying(100) COLLATE pg_catalog."default" NOT NULL,
    tipo character varying(10) COLLATE pg_catalog."default",
    idpai integer,
```

```
    CONSTRAINT pk_dir_idiretorio PRIMARY KEY (idiretorio),
    CONSTRAINT un_dir_nome_pai UNIQUE (nome, idpai),
    CONSTRAINT fk_dir_idpai FOREIGN KEY (idpai)
        REFERENCES public.diretorio (idiretorio) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE CASCADE
)
```

– Estrutura da tabela de arquivos

```
CREATE TABLE IF NOT EXISTS public.arquivo
(
    id_arquivo integer NOT NULL GENERATED BY DEFAULT AS IDENTITY ( INCREMENT 1 START 1 MINVALUE 1 MAXVALUE 2147483647 CACHE 1 ),
    nome_arquivo character varying(255) COLLATE pg_catalog."default" NOT NULL,
    path character varying(500) COLLATE pg_catalog."default" NOT NULL,
    conteudo text COLLATE pg_catalog."default" NOT NULL,
        data_criacao timestamp without time zone DEFAULT CURRENT_TIMESTAMP,
    idiretorio integer,
    CONSTRAINT pk_arq_idarquivo PRIMARY KEY (id_arquivo),
    CONSTRAINT fk_arquivo_idiretorio FOREIGN KEY (idiretorio)
        REFERENCES public.diretorio (idiretorio) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)
```