CENTRO PAULA SOUZA

ETEC PHILADELPHO GOUVÊA NETTO

Curso Técnico em Desenvolvimento de Sistemas Mtec PI

Pietro Furlanetto Gonçalez

P2PCraft: Um método diferente de hospedar servidores

São José do Rio Preto

2024

Pietro Furlanetto Gonçalez

P2PCraft: Um método diferente de hospedar servidores

Trabalho de Conclusão de Curso apresentado ao Curso Técnico de Desenvolvimento de Sistemas, orientado pela Prof.ª Camila Brandão Fantozzi, como requisito parcial para obtenção do título de técnico em Desenvolvimento de Sistemas.

São José do Rio Preto

2024

Apliquei o conhecimento desenvolvido pelos professores para criar este trabalho, que

reflete o aprendizado e a dedicação ao longo desses três anos de curso.

AGRADECIMENTOS

Primeiramente, gostaria de agradecer minha família pelo apoio nos momentos difíceis durante a criação deste trabalho.

Aos professores, agradeço pela paciência, pelos ensinamentos e pela sabedoria repassada ao longo dos três anos do curso.

Agradeço também aos meus amigos, que me trouxeram alegria nos momentos mais sombrios e por me aconselhar nas mais variadas situações.

“O verdadeiro progresso é aquele que coloca a tecnologia ao alcance de todos e não apenas dos privilegiados; é aquele que gera benefícios e oportunidades, e não barreiras e desigualdades”

Anonimo.

RESUMO

A P2PCraft é um sistema de hospedagem descentralizado para servidores de Minecraft, criado para pequenos grupos de amigos que buscam uma solução gratuita e de qualidade, sem filas, travamentos ou alta latência, problemas comuns nas opções gratuitas do mercado. Composto por um *site* e um *mod* de Minecraft, o sistema permite que um dos jogadores atue como *host*, com o *mod* gerenciando a hospedagem, conexão e salvamento de dados. A ideia é proporcionar uma alternativa de hospedagem que combina desempenho e simplicidade para pequenos servidores, sem custo para o usuário.

Palavras chave: hospedagem, Minecraft, descentralizado, *mod,* gratuito.

ABSTRACT

P2PCraft is a decentralized Minecraft hosting system, developed to be used by small groups of friends that search for good quality, lag free, queue free alternatives for Minecraft hosting. It has a website and a Minecraft mod, by installing the mod, our system enables the player to become the host of the server by. The main idea of the project is to give an alternative for those who search for a simplistic and free Minecraft hosting.

Keywords: Minecraft, hosting, decentralized, free, mod

LISTAGEM DE TABELAS

LISTAGEM DE IMAGENS

SUMÁRIO

INTRODUÇÃO

1.1 Jogos online

Desde de criança eu sempre gostei de jogos, particularmente aqueles jogos que você pode jogar com vários amigos, os jogos *online*. Hoje em dia a maioria dos jogos lançados são *online*, é muito normal você hoje instalar um jogo e ver que tem o modo *online*. Mas nem sempre foi assim.

1.2 Conexões de *internet* antigas

1.2.1 *L.A.N*

Antigamente, digo próximos dos anos 2000, era muito difícil jogar com seus amigos, os jogos multijogadores da época usavam conexão chamada *L.A.N.* (*local area network*) e para essa conexão funcionar, todos os jogadores precisam estar conectados na mesma *internet*. Então era normal você ir a um estabelecimento chamado *L.A.N. house*, para poder jogar.

Imagem 1 - *L.A.N House* em 2001

Fonte: VG247, 2024.

1.2.2 *Dial* Up *Internet*

Um outro método bastante popular que se assemelha ao de hoje em dia é o *Dial up internet*, a famosa *internet* discada, onde você ou o seu computador, precisava literalmente discar em um telefone para conectar. Isso era no mínimo rudimentar, os problemas eram muitos, eles iam desde alguém telefonar para sua casa até um barulho chato que fazia ao tentar a conexão.

Conforme o tempo passou, a *internet* obviamente ficou mais rápida e os métodos de conexão se aprimoraram. Hoje a grande maioria dos sistemas usam a conexão cliente-servidor, onde uma única máquina ou computador, recebe, processa e espalha os dados dos clientes ou usuários. E nos jogos isso não é diferente. Jogos como Counter Strike, League of Legends ou Valorant usam esse tipo de conexão.

1.2.3 *Peer-to-Peer*

Mas também existe um outro tipo de conexão muito usado chamado *peer-to-peer*, ou *P2P*, onde o servidor não é uma máquina por região, ou por país, mas sim o conjunto de máquinas dos jogadores, ou seja, o servidor é descentralizado, os próprios jogadores são o servidor, não há um computador na região que controla os dados, mas sim, os computadores dos jogadores de uma sessão. Exemplos de jogos que usam esse tipo de conexão são: *GTA 5 Online, Brawl Starts e Among Us*.

Na teoria, o melhor tipo de conexão para um jogo *online* é a cliente-servidor, pois ela apresenta menor latência, e uma taxa de perda de dados menor comparado a *peer-to-peer*, mas isso vem com um alto custo. O preço tanto de aquisição, quanto de manter um servidor, ou seja, um mega computador, é simplesmente muito alto para até mesmo as maiores empresas do setor de jogos *online* como a Rockstar (criadora do GTA 5 *online*). Então o que muitos programadores acabam fazendo é usar a máquina dos jogadores para hospedar o servidor a um custo mínimo.

1.3 Servidores da Comunidade

E também, existe uma terceira opção, os chamados servidores da comunidade. Que é quando um jogo deixa que os jogadores hospedam seus servidores de forma

centralizada, ou seja, o tipo de conexão é servidor-cliente, mas quem hospeda os servidores não é a empresa que fez o jogo, mas sim uma pessoa comunidade do jogo, que tem o dinheiro para bancar um servidor.

Um dos jogos mais famosos que usam dessa opção é o Minecraft, onde você tem servidores como o da *hypixel*, que bancam o custo de ter uma conexão servidor-cliente, enquanto a *Mojang* (criadora do *Minecraft*) não precisa pagar nada com isso.

Então esse é o melhor dos dois mundos né? A empresa não paga nada, e os jogadores possuem a melhor conexão possível. A resposta é não, não é bem assim. O alto custo da conexão cliente-servidor ainda existe, e o problema agora é que (no caso do *Minecraft*) o jogador não consegue ter um mapa para si, ele não tem controle dos arquivos internos do mapa, ou seja, é como se ele tivesse alugado um livro, você pode ler o que tem nele, mas não pode fazer o que quiser com ele.

Para maioria dos jogadores isso está tudo bem, mas se você é daqueles que quer ter um mapa apenas para você e seus amigos jogarem ou é daqueles que gosta de “fuçar” nos comandos internos do jogo, você não consegue.

A única maneira de jogar Minecraft *online* e ao mesmo tempo, ter acesso total aos arquivos do mapa é hospedar um servidor, ou seja, se render ao alto custo. Mas não é o fim do mundo. Hoje, a maioria dos *notebooks* e até mesmo alguns celulares conseguem hospedar um servidor de Minecraft para pequenos grupos de amigos, por pequenos, eu quero dizer de 2 a 8 amigos.

A ideia de hospedar usando a conexão cliente-servidor em um computador pessoal de um dos jogadores, não é algo novo, existe uma ferramenta chamada Hamachi.

Porém, o Hamachi possui uma alta latência e também, um há outro problema inesperado, apenas um dos jogadores tem acesso ao mapa, isso significa que para todos jogarem na última versão do mapa, é necessário que apenas um jogador abra o mapa. Isso quer dizer que eu só posso jogar, quando algum dos meus colegas estiverem *online.* Isso é chateante para não dizer pior, mas ainda há uma outra opção.

1.4 O trabalho

Após muito discutido, foi decidido o tema do trabalho. Um sistema de *hosting* de servidores de Minecraft descentralizado. Esse tema foi escolhido pois além de ser um problema que o autor deste trabalho enfrenta, não há nenhum sistema deste tipo na *internet* nem gratuito e nem à venda.

O sistema foi feito para atender pequenos grupos de amigos (até 5 jogadores) que querem jogar juntos, porém não querem gastar em hospedagem de Minecraft, e não suportam as filas e os travamentos das alternativas gratuitas.

A P2PCraft é uma *host* de Minecraft que visa fornecer uma hospedagem de graça, com qualidade superior às demais hospedagens gratuitas.

O sistema conta com um *site* e um *mod* de Minecraft, que juntos funcionam em harmonia, para que o usuário tenha uma excelente experiência e facilidade na hora de criar, gerenciar, e hospedar o servidor.

O usuário poderá encontrar um *download* do *mod* no *site*, e futuramente na plataforma CurseForge, que é um disponibilizador confiável de *mods* de Minecraft.

2 FUNDAMENTAÇÃO TEÓRICA

Como já dito acima, atualmente não há nenhum sistema de *hosting* de Minecraft descentralizado, porém, mesmo sem competidores diretos, ainda há algumas empresas que podem impactar fortemente neste trabalho.

2.1 Aternos

A princípio, a empresa Aternos é a principal competidora, já que ela é uma das únicas empresas que fornecem gratuitamente uma *host* (porém centralizada).

Ela não possui nenhum tipo de limitação como o tempo que o servidor ficará *online*, quantidade de *players* simultaneamente ou até mesmo *mod* e *plugins*. Seu *site* possui um *design* simples, fácil de usar e está lotado de anúncios.

Durante o desenvolvimento do sistema, usei como fonte de inspiração o *site* do Aternos.

2.2 Falix e outras empresas

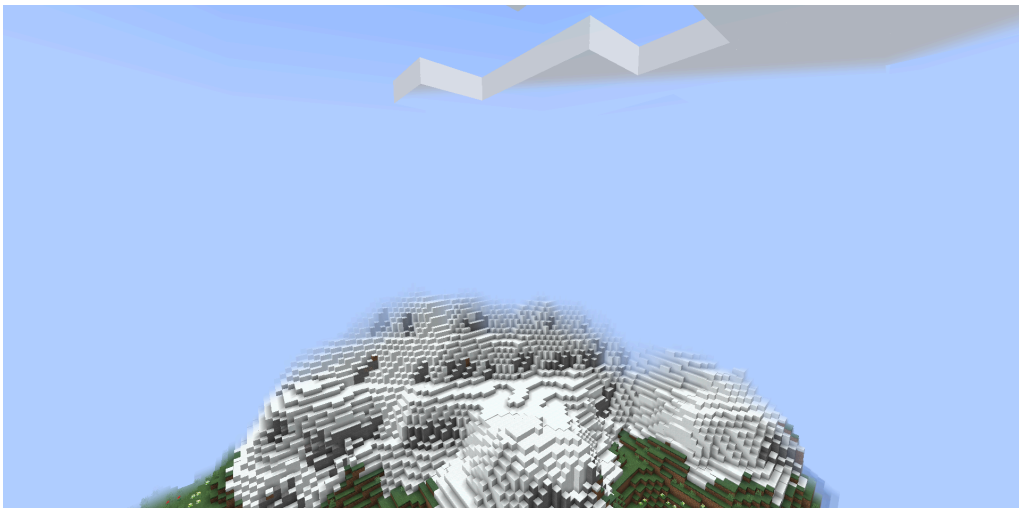
Um outro serviço de *hosting* gratuito que encontrei foi a Falix, assim como a Aternos, ela possui um *design* simples. Porém diferente da Aternos, a Falix possui também serviços de *V.P.S. (Virtual Private Server)* e *hosting* para outros jogos como Terraria ou Factorio.

Além dessas empresas, encontrei diversos *sites* como a Minehut que fornecem uma *host* porém com um tempo limitado, como se fosse uma “amostra grátis” do serviço deles. Mas como foi dito acima, a única empresa que fornece uma *host* grátis por tempo limitado é a Aternos.

Porém, após analisar a Aternos, conclui que ela possui diversas falhas no seu serviço tais como o *lag*, enormes filas, e a alta latência são os seus principais problemas.

O sistema tem como principal objetivo diminuir esses travamentos e acabar com a alta latência juntamente com as filas ao hospedar o servidor na maquina de seus próprios usuários.

Imagem 2 - Carregamento de chunk usando Aternos



Fonte : Do autor, 2024.

Imagem 3 - Carregamento de *chunk* usando P2PCraft

Fonte : Do autor, 2024.

3 DESENVOLVIMENTO

O *software* foi dividido em 3 partes, o sistema *web*, a *REST* A.P.I. (*Application Programming Interface)*, e o *mod* de Minecraft, essa seção irá explorar a fundo o funcionamento de cada uma dessas partes.

3.1 O *Mod*

O fluxo sistema envolve um dos jogadores ser a *host* do servidor, e para isso ser possível, várias implementações alternativas tiveram que ser criadas, uma delas é o funcionamento interno do *mod.* M*od* é basicamente um arquivo que quando injetado no Minecraft, altera o comportamento do jogo.

Em especial, a função do meu *mod,* é dentre várias, iniciar uma hospedagem, salvar o mapa periodicamente, e conectar no servidor.

3.1.1 Conectar ao servidor

Para se conectar ao servidor, o *mod* primeiramente deve identificar o endereço *I.P.* (*Internet Protocol)* do mesmo. Ele irá procurar pela máscara “p2pcraft.connect.xxxxxxx.xyz” onde “xxxxxxx” refere-se ao nome do servidor. Se o *I.P.* atende a essa máscara, e então o *mod* irá “perguntar” à *A.P.I.* se um *player* já está hospedando o servidor, se sim, ele irá seguir com a conexão, se não, ele irá iniciar o processo de hospedagem do servidor.

Para fazer a conexão, o usuário precisa primeiro, ter o *I.P.* verdadeiro do servidor, que pode ser encontrado ao consultar a *A.P.I.* Essa diferença entre *I.P.* fantasia e *I.P.* verdadeiro existe pois cada um dos dispositivos possui um endereço *I.P.* diferente, e como o jogador *host* pode mudar, esse I.P. também mudaria. Então a A.P.I. também faz esse indexamento entre *I.P.* fantasia e *I.P.* real, se assemelhando a uma *D.N.S. (Domain Name System).*

3.1.2 O processo de hospedagem

Para hospedar um servidor, o jogador precisa primeiramente ter os arquivos do servidor instalados em sua máquina. Onde esses arquivos são armazenados e como a instalação é feita será o tema da próxima subseção.

Após instalado o mapa, o *mod* irá procurar pôr um arquivo nomeado “server.jar”, este arquivo é responsável pôr abrir o servidor localmente. Após aberto, o *mod* deve então fazer algo chamado *T.C.P. (Transmission Control Protocol) Hole punching*, que faz com que o servidor que está local seja aberto à internet sem comprometer dados

importantes do usuário. Após isso, o *mod* notifica a A.P.I. que o servidor está *online* e fornece também o novo I.P. verdadeiro.

3.1.3 O salvamento

Para salvar o mapa, eu escolhi usar a ferramenta *Git,* ela é uma ferramenta muito usada para versionamento de repositórios. Uma função crucial para a escolha dessa ferramenta é como ela atualiza um repositório.

Basicamente, o *Git,* salva apenas as alterações dos arquivos, e não os arquivos em si, isso é muito útil para o salvamento do mapa, tendo em vista que uma vez carregado, os arquivos internos do mapa tendem a permanecer os mesmos.

Além disso, ao usar o *Git*, o sistema ganha acesso a serviços gratuitos como o *Github,* que está sendo usado para armazenar mapas, ou seja, o meu sistema não precisa pagar pelo armazenamento. Em contrapartida, o *Github* possui um limite máximo de 5 *Gigabytes* de armazenamento, o que é relativamente bastante para um servidor de *Minecraft* entre amigos.

3.2 *REST A.P.I.*

A *A.P.I.* é como se fosse o cérebro do sistema, ela é quem cria uma relação entre o banco de dados e os consumidores da usuários do sistema. É na *A.P.I.* onde encontramos todos os casos de uso, como *login,* registro de servidor, registro de cliente, informações do cliente, entre vários outros, a tabela 1 possui todos os casos de uso do sistema.

Tabela 1 - Casos de uso da A.P.I.

Código	Identificação	Classificação	Ator	Objetivo
RF01	Efetuar Login	Essencial	Usuário	O usuário precisa estar logado para criar e/ou gerenciar um servidor
RF02	Manter Servidor	Essencial	Usuário	Permite o usuário criar, alterar e deletar o servidor

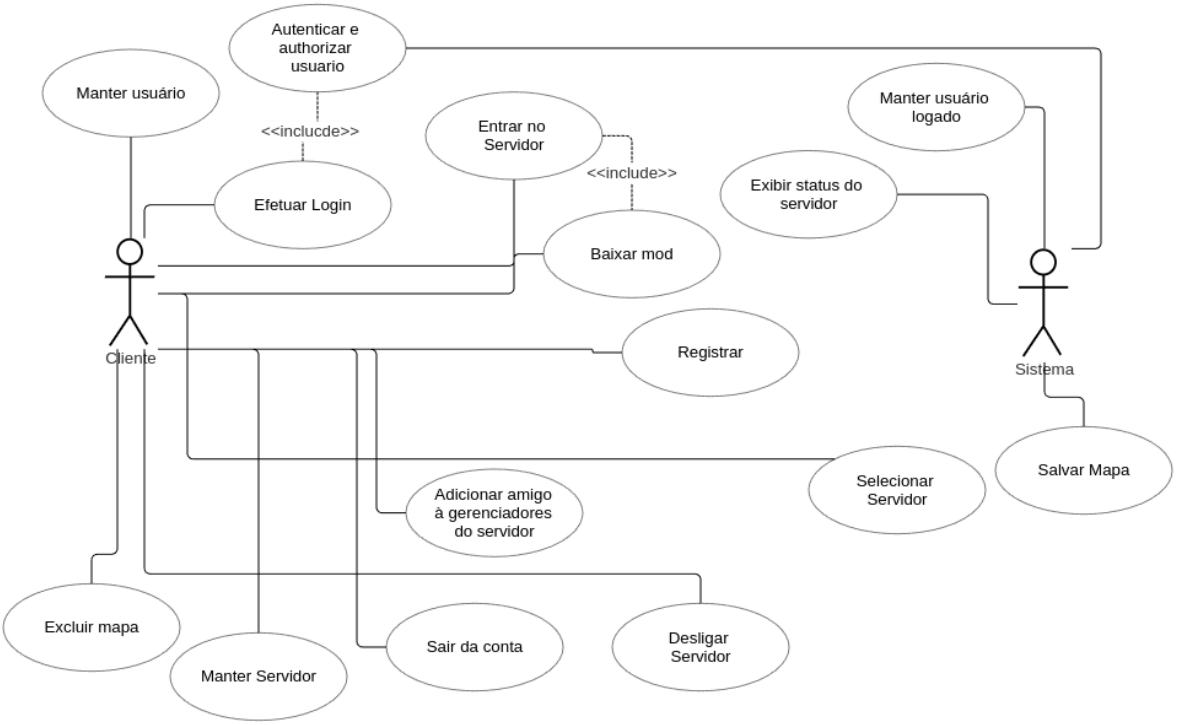
Código	Identificação	Classificação	Ator	Objetivo
RF03	Entrar no servidor	Essencial	Usuário	O *mod* deve fazer com que o usuário seja capaz de entrar no servidor.
RF04	Baixar Mod	Essencial	Usuário	O usuário deve ter acesso ao link de download do *mod*
RF05	Desligar servidor	Importante	Usuário	Permite desligar o servidor caso ninguém esteja jogando
RF06	Adicionar um amigo para gerenciar o servidor	Importante	Usuário	Permite ao usuário dar acesso de gerenciamento do servidor a um amigo
RF07	Salvar automaticamente o mapa	Importante	Sistema	Permite ao sistema salvar o mapa do servidor periodicamente
RF08	Manter Cliente	Essencial	Usuário	Permite o usuário criar, registrar e deletar, algumas informações de sua conta
RF11	Excluir mapa	Importante	Usuário	Permite o usuário deletar seu mapa
RF12	Baixar mapa	Importante	Usuário	Permite o usuário baixar o seu mapa
RF13	Exibir status do servidor	Importante	Sistema	Permite ao sistema, através de cores, exibir o status do servidor (online/offline)
RF14	Manter Logado	Importante	Sistema	Caso já tenha se logado, o sistema deve manter o login do usuário.

Código	Identificação	Classificação	Ator	Objetivo
RF15	Deslogar	Importante	Usuário	Permite o usuário deslogar
RF17	Selecionar servidor	Importante	Usuário	Permite o usuário selecionar um servidor para gerenciar
RNF 01	Autenticação e Autorização via token *J.W.T. (Json Web Token)*	Importante	Sistema	Permite o sistema usar os protocolos de autorização e autenticação usando *J.W.T.*

Fonte: De autoria própria, 2024

3.2.1 Caso de uso do sistema

Imagem 4 - Caso de uso do sistema



Fonte: De autoria própria, 2024

3.2.2 O fluxo esperado do cliente

O fluxo esperado é simples, primeiro o cliente registra sua conta, depois ele registra um servidor e o liga. E então o cliente instala o *mod*, abre seu Minecraft, coloca o *I.P.* do servidor e conecta, como se fosse um servidor normal.

Obviamente, cada implementação do caso de uso é diferente, mas existem algumas implementações que requerem um processo especial. Nas próximas subseções, irei entrar a fundo nesses casos de uso.

3.2.2.1 Registro de um servidor

Para registrar um servidor, um usuário precisa primeiramente estar devidamente logado. Além disso, algumas informações são necessárias para registrar o servidor, como a *seed* do mapa, o nome do servidor e a versão do mesmo.

A *seed* é opcional, ela pode ser qualquer sequência de letras. Ela é usada para a aleatoriedade na criação do mapa.

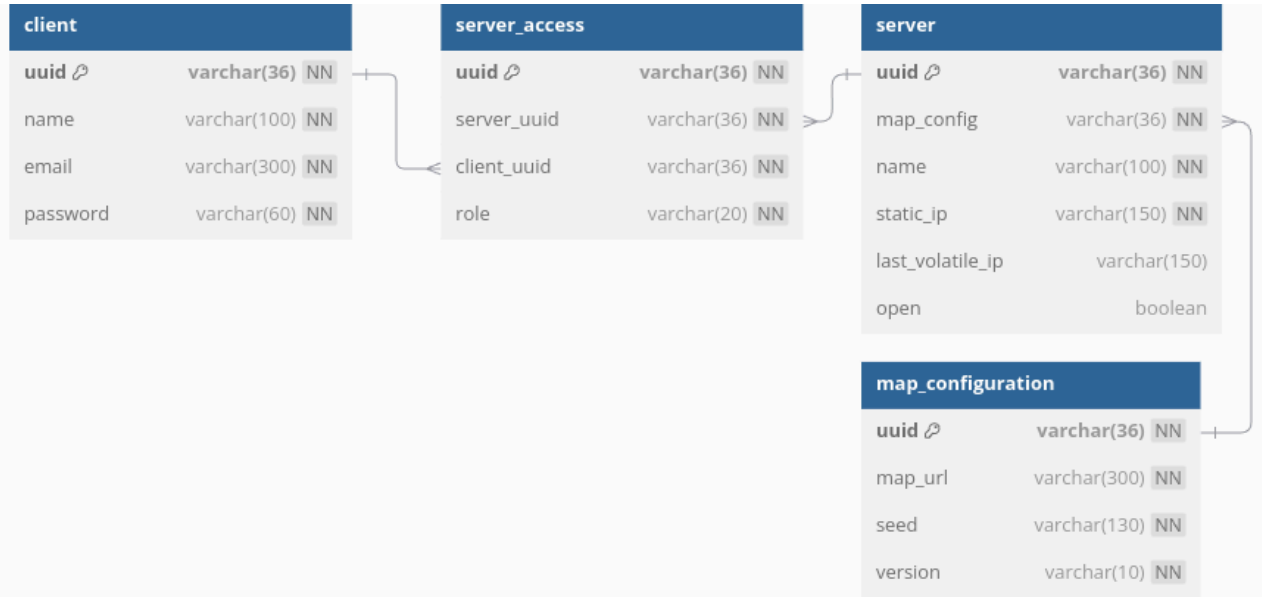
O nome do servidor é único e ele é quem define o *I.P.* do servidor, seguindo a máscara “p2pcraft.connect.xxxxxxx.xyz” onde “xxxxxxx” é o nome do servidor.

No momento da publicação desta monografia, apenas a versão 1.20.0 está disponível, mas novas versões já estão sendo desenvolvidas.

3.2.3 Banco de dados

Para guardar os dados dos clientes e dos servidores, eu escolhi usar um *R.D.B.M.S.* (*Relational Database Management System)* chamado MySql. Não há nenhuma justificativa pelo uso especifico do MySql, eu estou usando MySql simplesmente porque estou acostumado com ele. A imagem 3 mostra o diagrama relacional do banco de dados.

Imagem 5 - Diagrama relacional do banco de dados



Fonte: De auditoria própria, 2024.

3.3 Sistema *web*

Como eu quero aumentar ao máximo a compatibilidade e facilidade de acesso ao sistema, decidi criar um site para gerenciar algumas coisas do servidor, como, os

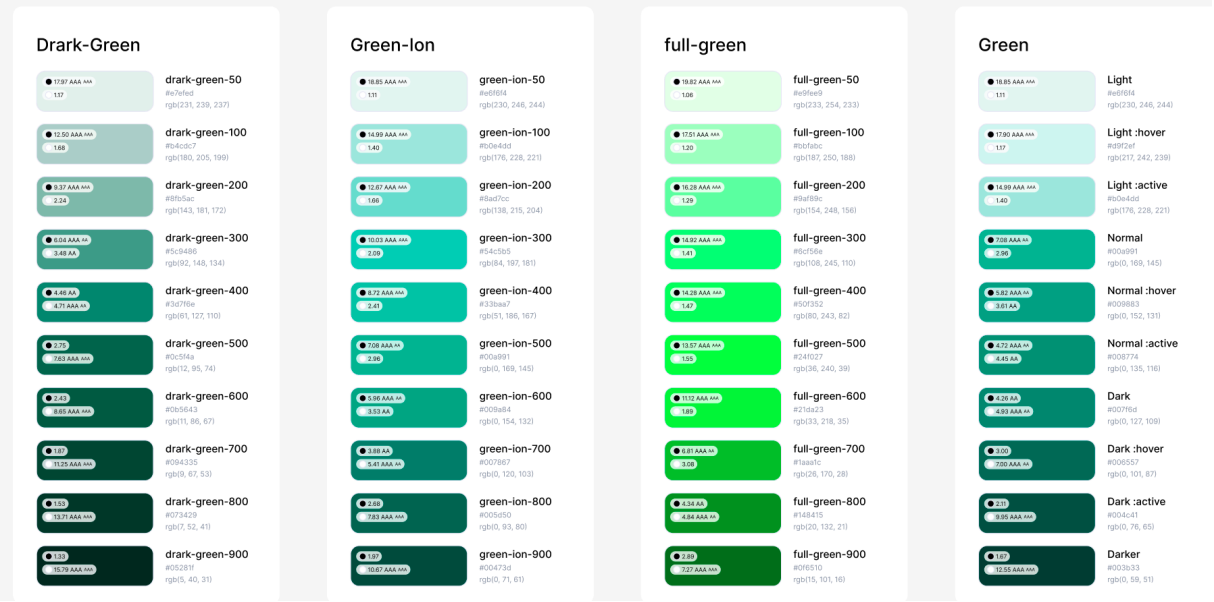
jogadores que podem ter acesso, os arquivos do mapa e também às configurações internas do servidor tais como dificuldade, quantidade de jogadores máximos, listas de jogadores permitidos, se é permitido usuários “piratas”, entre várias outras.

A partir disso, sabendo que os sites são carregados por navegadores, e que Minecraft foi feito em Java (uma linguagem que foi feita para funcionar em todo o sistema operacional), garantimos o acesso ao nosso sistema dentro de qualquer sistema operacional. Contudo, hospedar um servidor de Minecraft é uma tarefa que requer uma quantia considerável de processamento e por tanto, previmos que a quantidade mínima de *R.A.M.* (*Random Access Memory*) necessária para rodar o nosso sistema (caso você seja a *host*) seria de 1 a 2 *gigabytes* somados aos 4 *gigabytes* necessários para rodar o jogo.

3.3.1 Paleta de cores

Para criar a paleta de cores, refleti por um longo tempo sobre a teoria da psicologia das cores. Por fim, decidi usar a cor verde como base, pois ela reflete a liberdade, a renovação e a estabilidade. Além disso, a cor verde também é a cor usada na logo do Minecraft, e também é a cor do *mob* mais icônico do jogo, o *Creeper.* A imagem 4 ilustra todos os tons de verde do sistema.

Imagem 6 - Tons de verde do *frontend*



Fonte: De auditoria própria, 2024

3.3.2 Logo

A logo da P2PCraft foi inspirada na arquitetura de redes *peer-to-peer,* onde não há uma *host* principal (centralizada), e todos os jogadores, podem a qualquer momento se transformar na *host*.

Imagem 7 - Logo do sistema

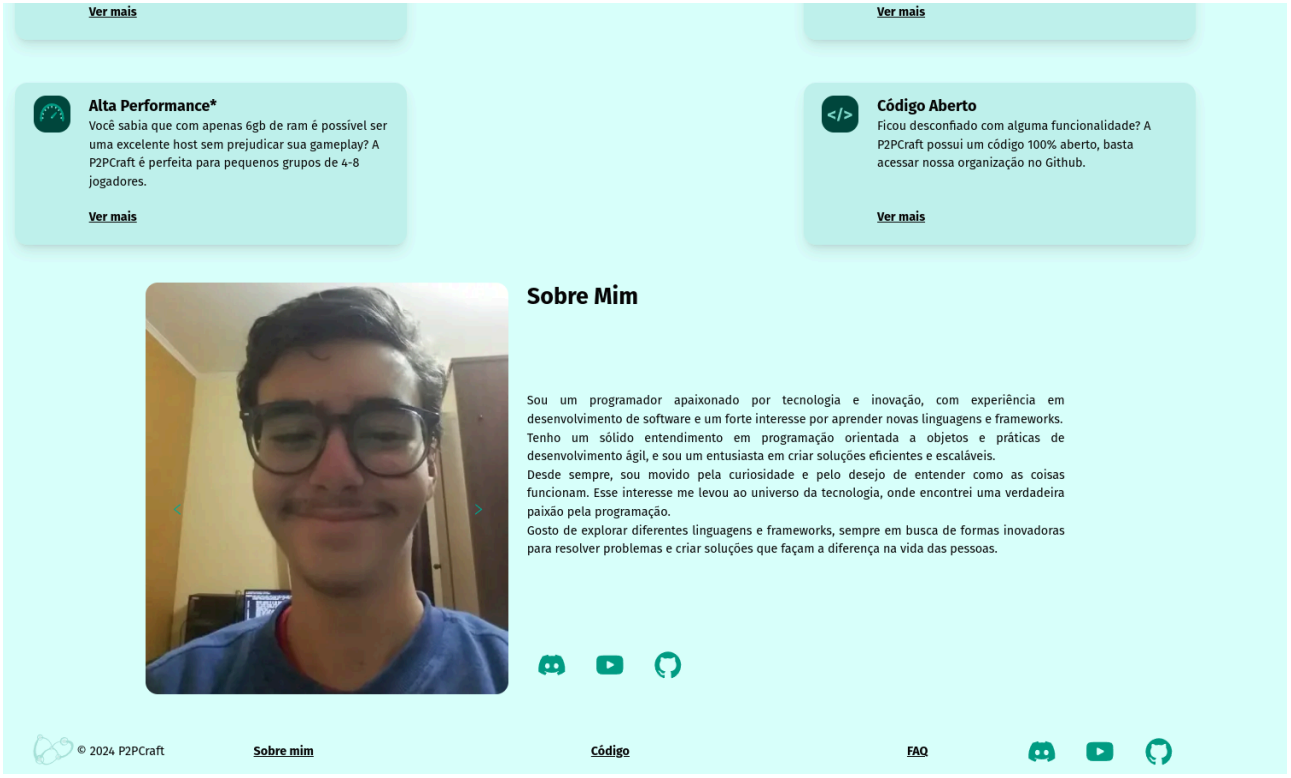
Fonte: De auditoria própria, 2024

3.3.3 *Landing page*

Imagem 8 - Landing page parte 1

Fonte: De auditoria própria, 2024

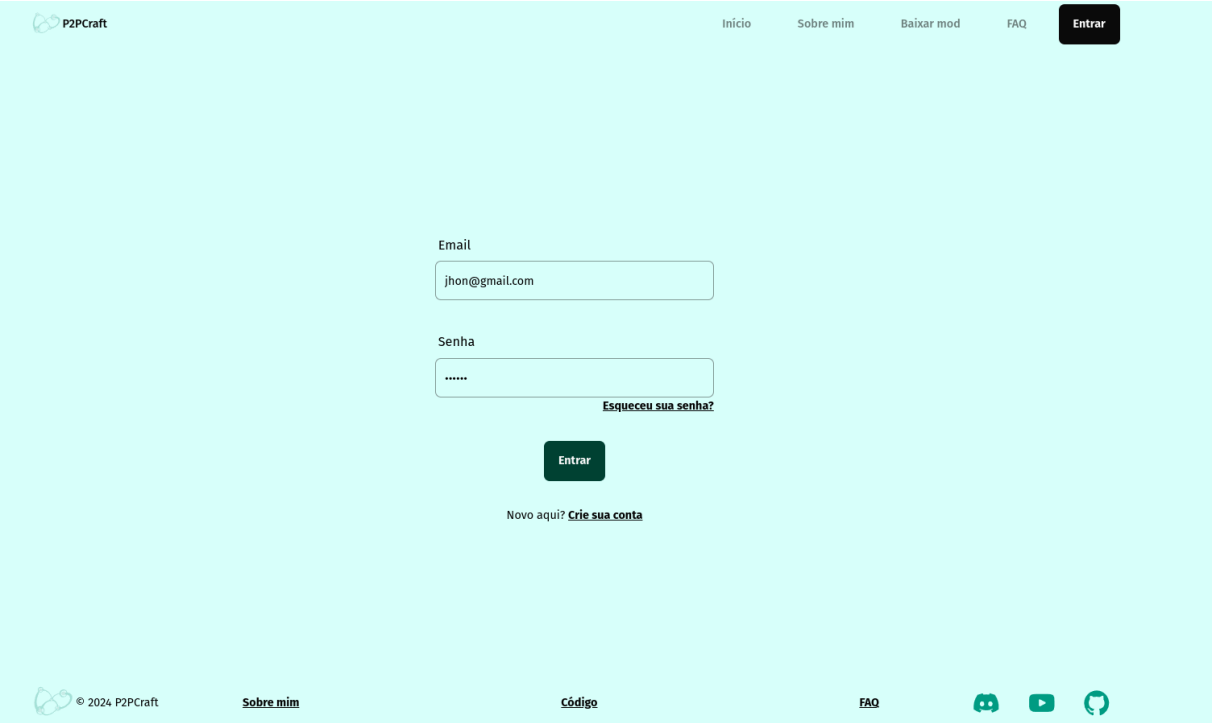
Imagem 9 - Landing page parte 2



Fonte: De auditoria própria, 2024

3.3.4 Página de *Login*

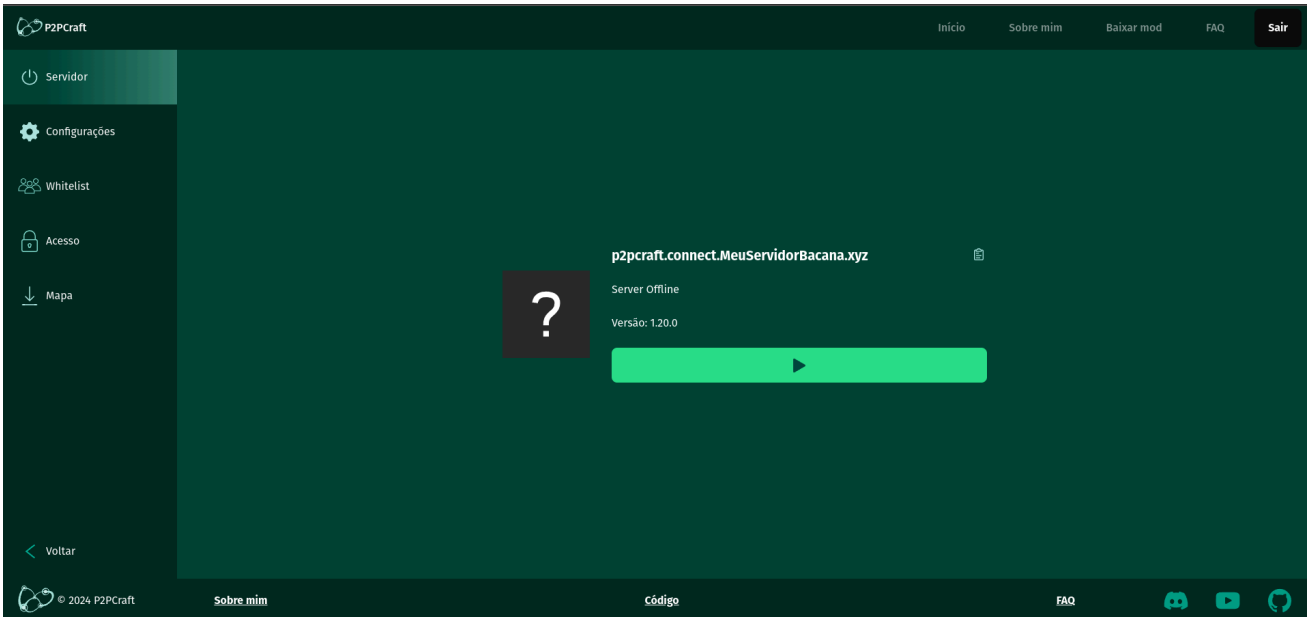
Imagem 10 - Página de Login



Fonte: De auditoria própria, 2024

3.3.5 Página inicial do servidor

Imagem 11 - Página inicial do servidor



Fonte: De auditoria própria, 2024

3.3.6 Página de configurações do servidor

Imagem 12 - Página de configurações do servidor

Fonte: De auditoria própria, 2024

4 AMBIENTES, FERRAMENTAS E LINGUAGEM

Para fazer o software, foram usadas diversas bibliotecas e *frameworks*, para apresentá-los, separei essa seção em 4 subseções, sendo elas, *Frontend, Backend, Devops* e ambiente de desenvolvimento.

4.1 *Frontend*

O *frontend* é a interface pela qual o usuário usa para se comunicar com a *A.P.I.*

Para fazer o *frontend* usei de início, NextJs*,* uma *framework* que usa React como base, essa foi uma escolha feita visando aumentar a produtividade e a qualidade final do código do site.

4.2 *Backend*

O nosso *backend* foi feito usando Java, utilizei a *framework* Spring Boot, e também diversas outras *A.P.I.s* como Jakarta Persistence para persistir dados no banco de dados, Spring Security para autenticação e validação de tokens, e Spring Validation para facilitar a validação dos dados enviados à nossa *A.P.I*.

Além disso, também será necessário que o usuário instale localmente um *mod* de Minecraft para além de hostear, comunicar com a nossa *A.P.I.* Tal *mod*, será desenvolvido também em Java usando inicialmente, a *A.P.I.* do Fabric. Vale ressaltar que o *mod* foi desenvolvido visando implementar uma versão também para o Forge para aumentar a compatibilidade de nosso sistema.

4.3 *Devops*

Para facilitar o desenvolvimento, iremos publicar todo o código no *Github*, e usar diversas tecnologias como *Git* para gerenciar o repositório localmente, e algumas ferramentas de terminal para aumentar a produtividade, tais como o asdf para gerenciar versionamento de ferramentas como Java, npm, pip. E também utilizei uma ferramenta própria chamada JPM para gerenciar dependências Java por meio do terminal.

4.3 Ambiente

Como estou usando o Docker (que emula qualquer tipo de ambiente), o sistema operacional que vou usar para o desenvolvimento não é tão importante para o trabalho, mas o autor do trabalho fez ele usando Linux pela praticidade.

5 CONCLUSÃO

O sistema da P2PCraft foi capaz de efetuar com sucesso a sua proposta inicial: fornecer uma *host* descentralizada de Minecraft, com qualidade superior ao das outras alternativas. Entretanto, há algumas melhorias e aprimoramentos que virão nas próximas versões do sistema, como: mais funcionalidades, maior desempenho, menor gasto de *R.A.M.* entre outros.

6 REFERÊNCIAS BIBLIOGRÁFICAS

Aternos Lags. Disponível em <https://support.aternos.org/hc/en-us/articles/12562176027549-Lags>. Acesso em 18 abr. 2024

Falix About Us. Disponível em <https://falixnodes.net/company>. Acesso em 2 mai. 2024

Minehut Plans. Disponível em <https://app.minehut.com/shop/plans>. Acesso em 2 mai 2024

Anna, **Minecraft System Requirements: Minimum and Recommended**. Disponível em: <https://www.minitool.com/backup-tips/Minecraft-system-requirements.htm>. Acesso em 18 de abr. de 2024.

Leandro Kovacs, **O que é P2P?** . Disponível em: <https://tecnoblog.net/responde/o-que-e-p2p-peer-to-peer>. Acesso em 1 nov. 2024

JUNIOR, E. **JPA – Java Persistence API. O que é ? Como Funciona ?**. Disponível em: <https://www.tidicas.com.br/?p=1864>. Acesso em: 18 abr. 2024.

NEVES, V. **React: o que é, como funciona e um Guia da biblioteca JS.** Disponível em: <https://www.alura.com.br/artigos/react-js>. Acesso em: 18 abr. 2024.

WIKIPEDIA CONTRIBUTORS. **Docker (software).** Disponível em: <https://en.wikipedia.org/wiki/Docker_(software)> .Acesso em: 18 abr. 2024.

WIKIPEDIA CONTRIBUTORS. **Git**. Disponível em: <https://en.wikipedia.org/wiki/Git>. Acesso em: 18 abr. 2024.

Andrei. **O Que é GitHub e Para Que é Usado?** Disponível em: <https://www.hostinger.com.br/tutoriais/o-que-github>. Acesso em: 18 abr. 2024.

Ana Paula, **O que é o Spring?** Disponível em: <https://www.treinaweb.com.br/blog/o-que-e-o-spring>. Acesso em: 18 abr. 2024.

ATLASSIAN. **What is Git: become a pro at Git with this guide | Atlassian Git Tutorial.** Disponível em: <https://www.atlassian.com/git/tutorials/what-is-git>. Acesso em 1 nov. 2024..

CLEMENTE, M. **Psicologia das cores: o que é e como usar no Marketing.** Disponível em : <https://rockcontent.com/br/blog/psicologia-das-cores/>. Acesso em 1 nov. 2024.

WIKIPEDIA CONTRIBUTORS. **Dial-up Internet access.** Disponível em: <https://en.wikipedia.org/wiki/Dial-up_Internet_access>.

CLOUDFLARE. **What is a LAN (local area network)?** Disponível em: <https://www.cloudflare.com/learning/network-layer/what-is-a-lan/>.

WIKIPEDIA CONTRIBUTORS. LAN gaming center. Disponível em: <https://en.wikipedia.org/wiki/LAN_gaming_center>.

APENDICE A - Script do BD

```
create database db_p2p;
use db_p2p;

create table client(
uuid varchar(36) primary key not null,
name varchar(100) not null,
email varchar(300) not null,
password varchar(60) not null
);

create table map_configuration(
        uuid varchar(36) primary key not null,
        map_url varchar(300) not null,
    seed varchar(130) not null,
    version varchar(10) not null

);
create table server(
uuid varchar(36) primary key not null,
map_config varchar(36) not null,
name varchar(100) not null,
static_ip varchar(150) not null,
last_volatile_ip varchar(150),
```

```
open boolean default false,
foreign key (map_config) references map_configuration(uuid)

);

create table server_access(
        uuid varchar(36) primary key not null,
    server_uuid varchar(36) not null,
    client_uuid varchar(36) not null,
    role varchar(20) not null,

    foreign key (client_uuid) references client(uuid),
    foreign key (server_uuid) references server(uuid)
);
```