



EquiTrade

Iuri Saad, João de Jesus, Pedro Henrique Soares Rosemberg e Priscila Andrade.



- Contexto do problema

01

-O mercado atual é fragmentado, dependendo de contatos diretos, redes sociais ou sites genéricos.

-Há limitações de alcance, desorganização das informações e falta de confiabilidade nas transações.



Motivação



Criadores de cavalos de raça,
vendedores e compradores
interessados no mercado equino.

Público-Alvo

- Criar um ambiente único que amplie a visibilidade dos criadores e ofereça praticidade aos compradores.
- Reforçar a segurança, transparência e credibilidade das negociações.





User Stories

PUBLICAÇÃO DE ANÚNCIOS (VENDEDOR)

"Como vendedor, quero cadastrar cavalos com detalhes (pedigree, fotos, vídeos e histórico) para atrair interessados."

BUSCA INTELIGENTE (COMPRADOR)

"Como comprador, quero filtrar animais por critérios (raça, preço, idade, localização) para encontrar o cavalo ideal.".

NEGOCIAÇÃO SEGURA (AMBOS)

"Como usuário, quero negociar através de um chat interno para manter a segurança e o registro da comunicação."

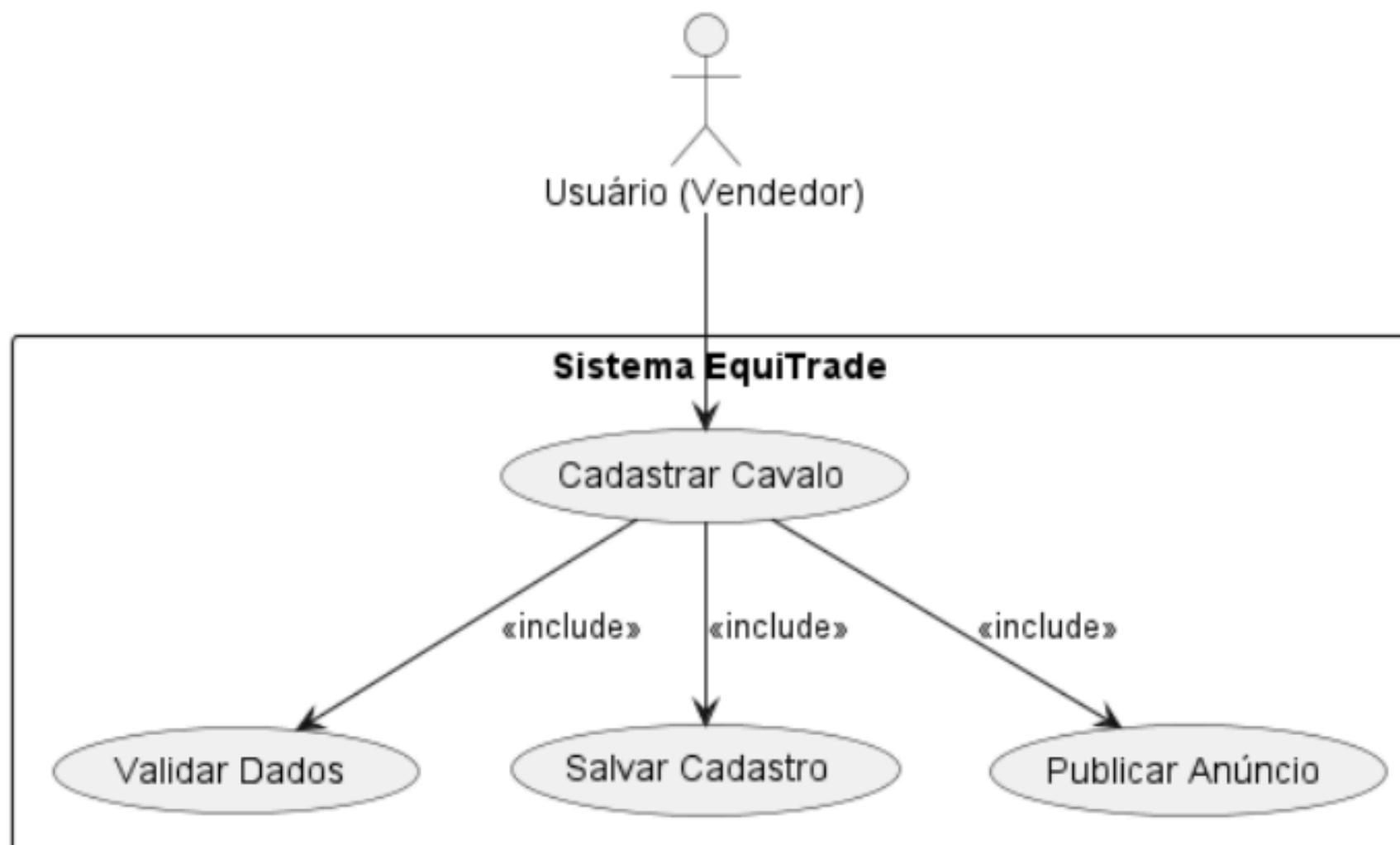
AVALIAÇÃO DE CONFIANÇA (COMPRADOR)

"Como comprador, quero avaliar o vendedor após a negociação para fortalecer a confiabilidade da plataforma."

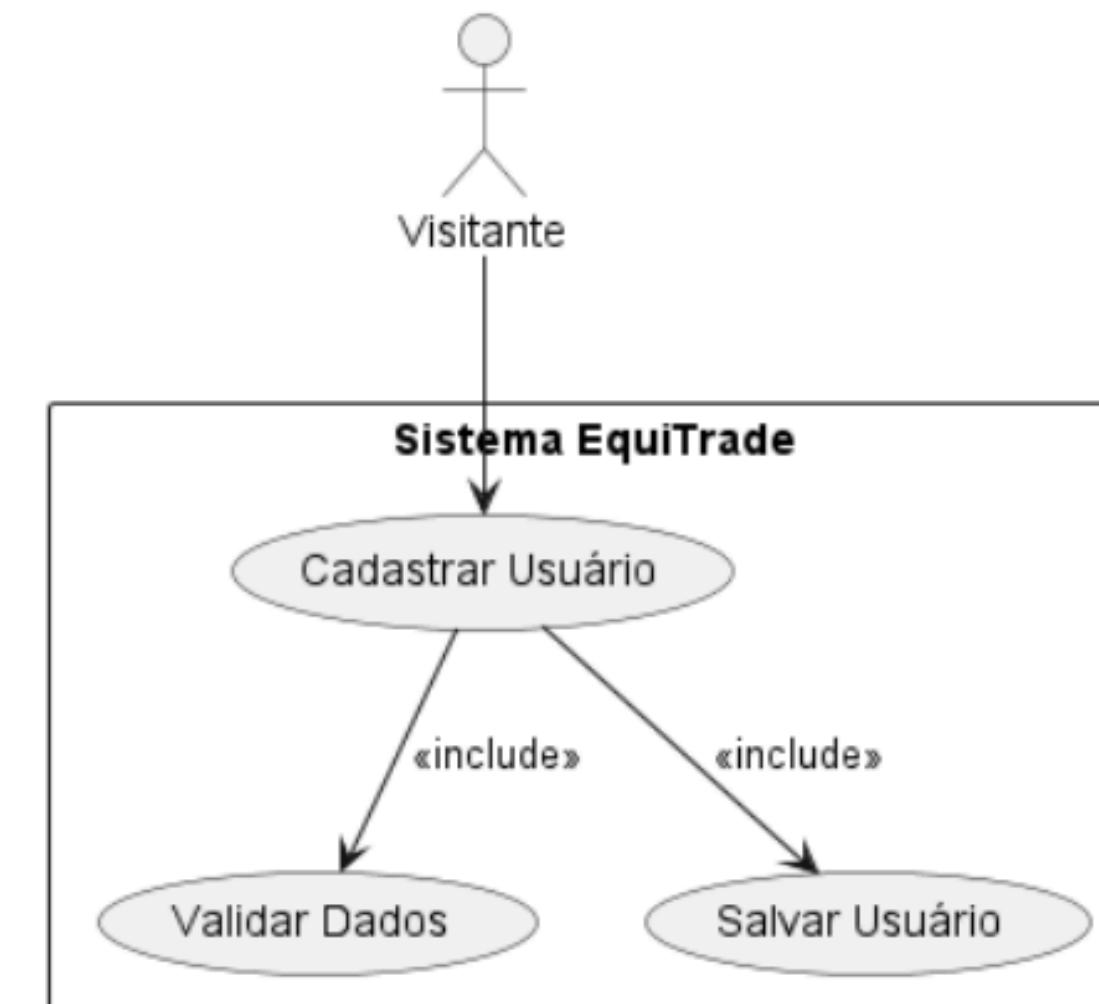


Diagramas de caso de Uso

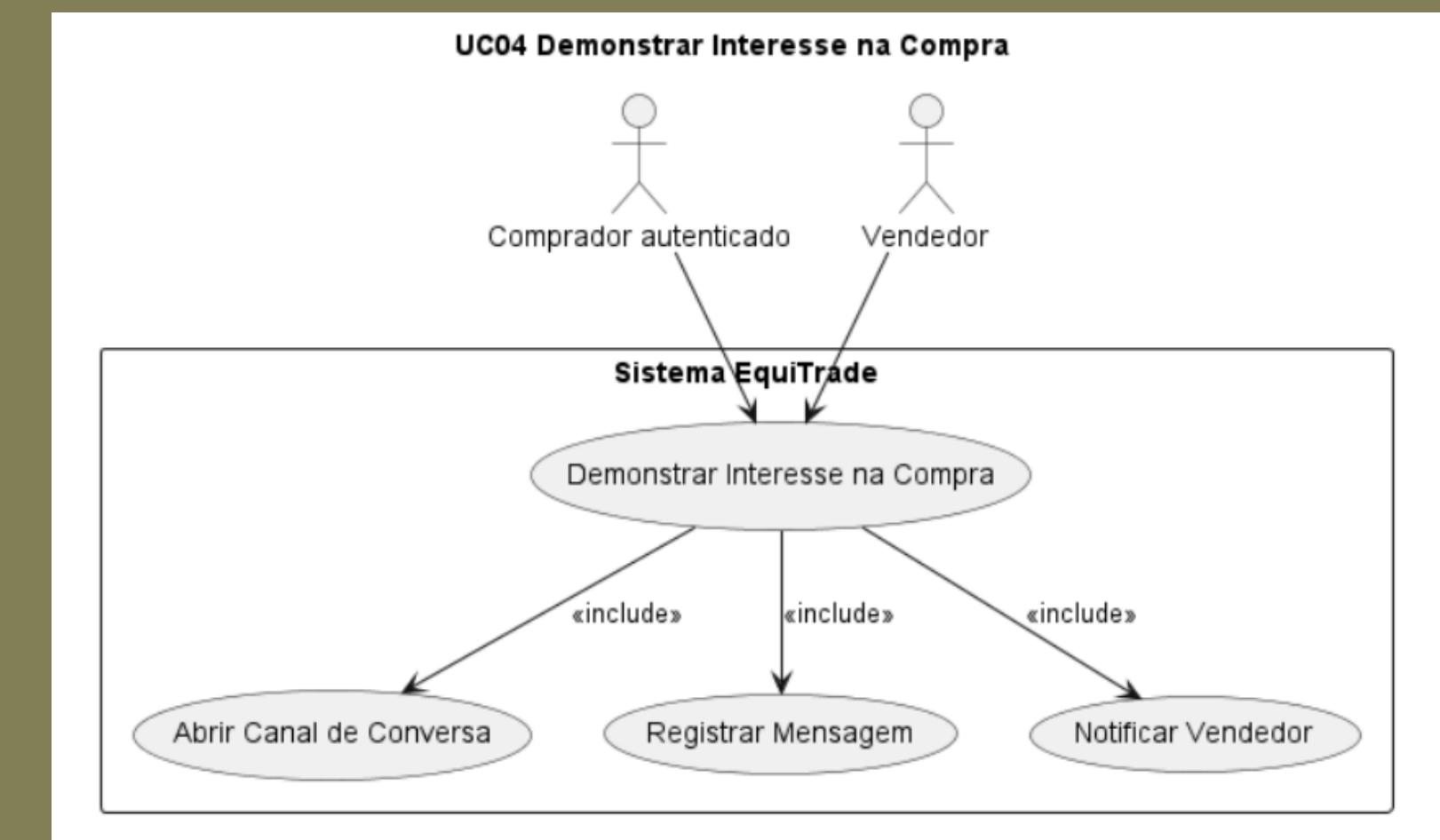
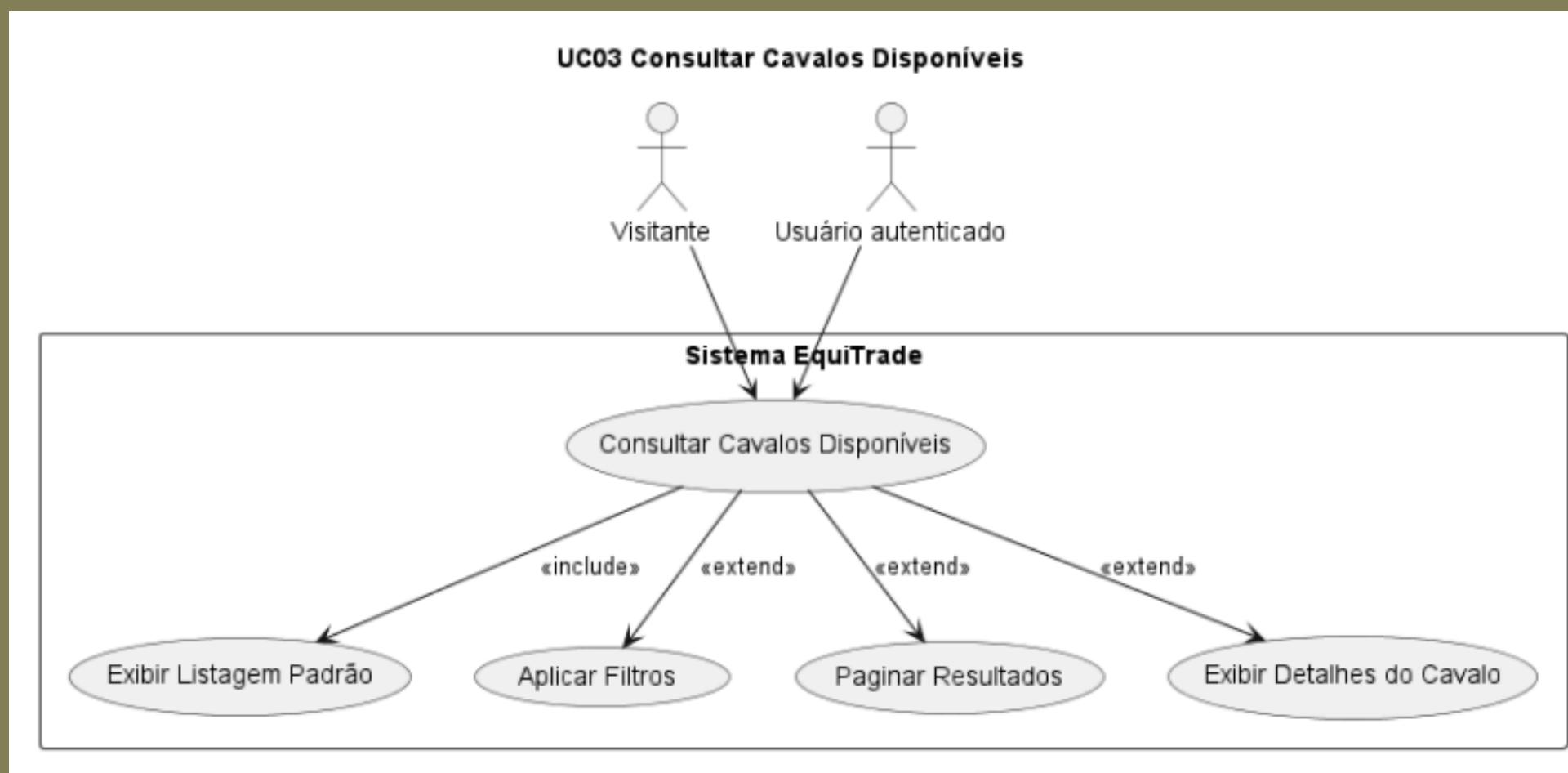
[UC01] – Cadastrar Cavalo

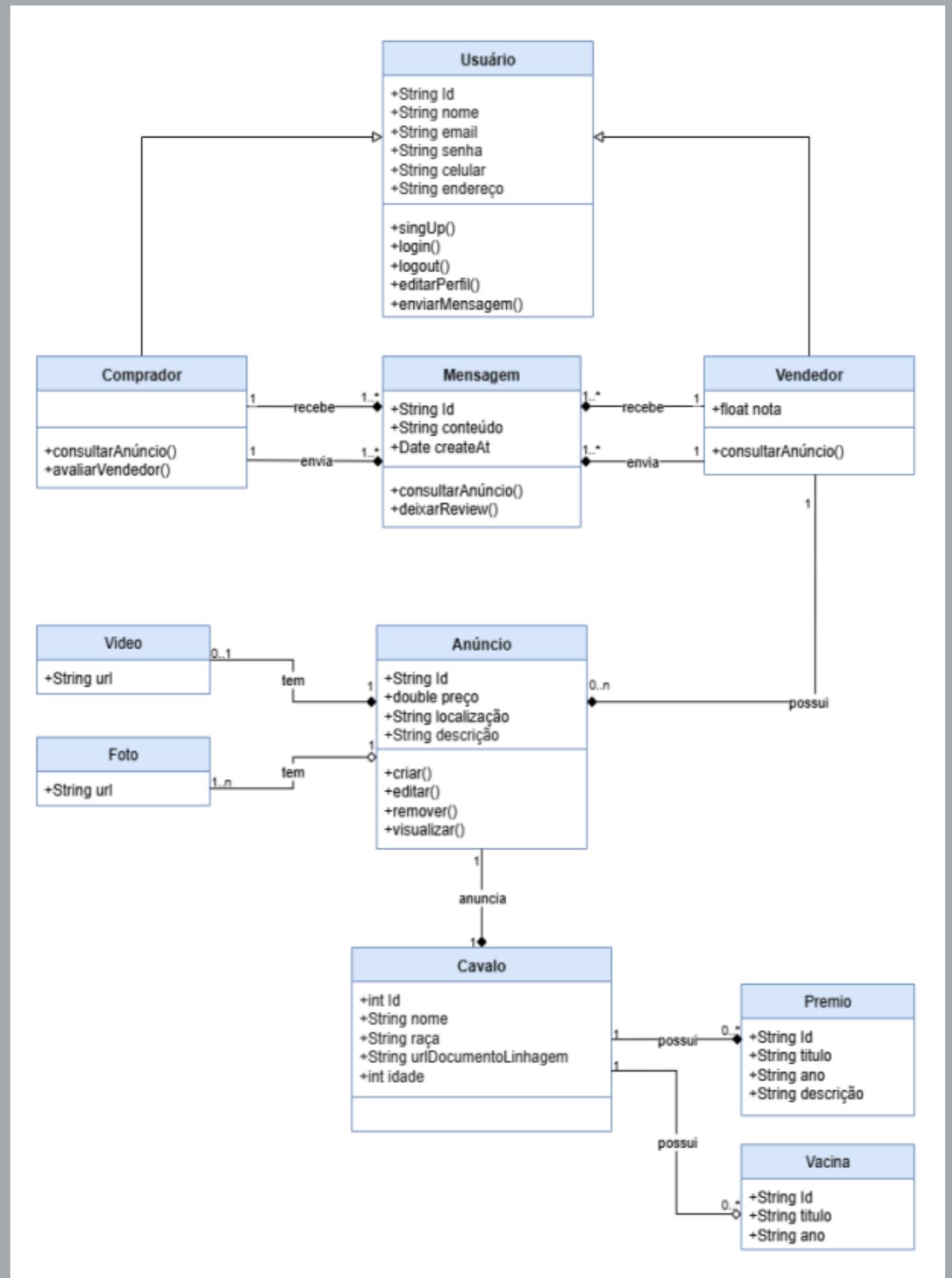


UC02 Cadastrar Usuário



Diagramas de caso de Uso





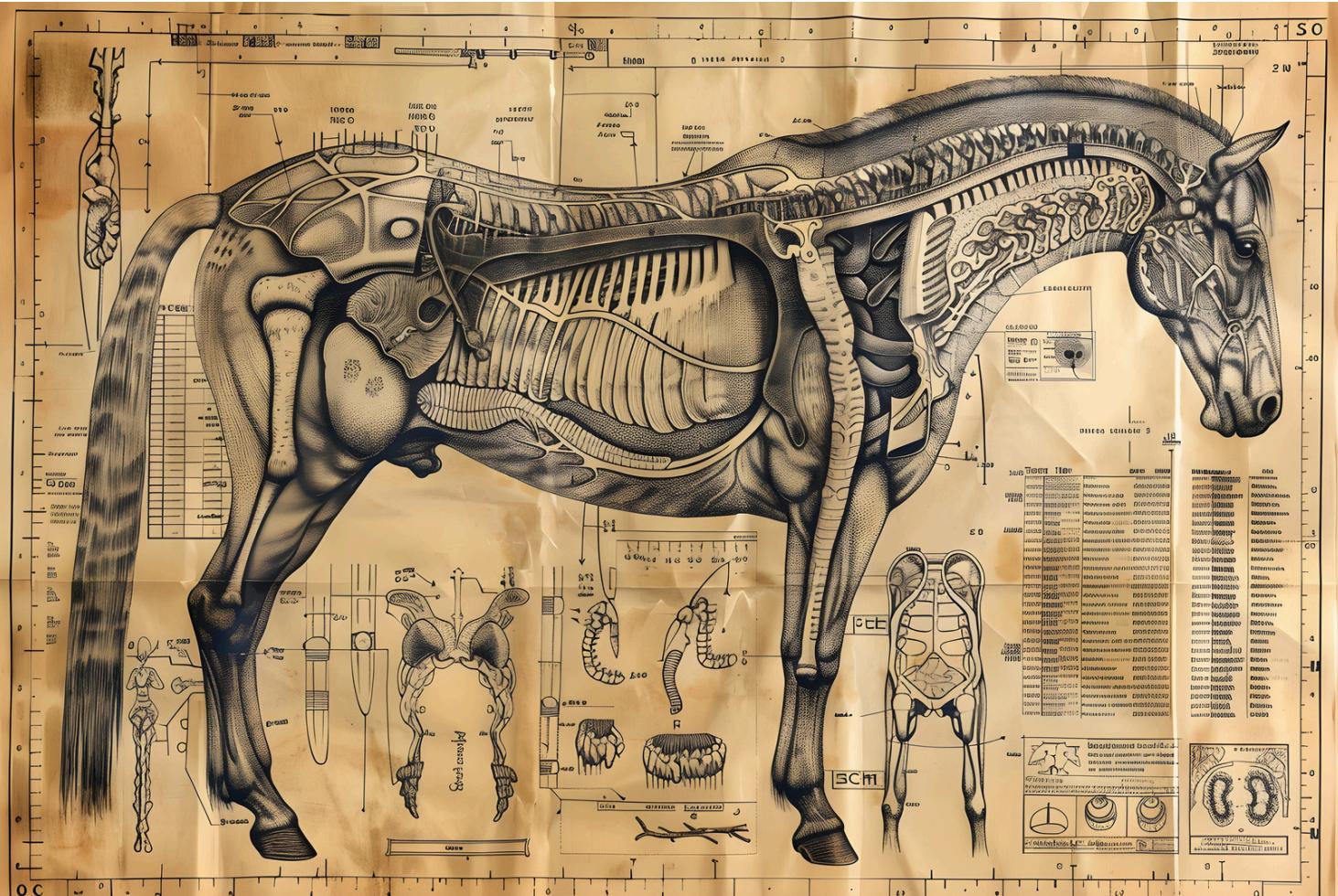
Arquitetura do Sistema

ESTILO ARQUITETURAL

- Arquitetura em Camadas com padrão Controller-Service-Repository
- Orientação a Serviços para módulos específicos

CAMADAS FUNDAMENTAIS

- Apresentação (Controllers) - Interação com o usuário
- Lógica de Negócio (Services) - Regras de negócio
- Acesso a Dados (Repositories) - Persistência de dados



Decisões que Garantem Qualidade

ESTILO ARQUITETURAL

- ✓ Separação de Responsabilidades - Cada camada com função definida
- ✓ Testabilidade - Testes unitários facilitados por camada
- ✓ Manutenibilidade - Alterações isoladas sem impacto lateral
- ✓ Escalabilidade - Crescimento gradual e controlado
- ✓ Modularização - Desenvolvimento paralelo de funcionalidades

Requisitos Não-Funcionais Atendidos

- Segurança: Senhas criptografadas (bcrypt)
- Performance: Respostas < 5s para listagens
- Portabilidade: Web e Mobile responsivos
- Compatibilidade: Chrome, Firefox, Edge



BACKEND

Implementação e Arquitetura

Tecnologias

- Node.js + Express
- TypeScript
- JWT
- TypeORM

Arquitetura em 3 camadas

- **Controller** → Validação e HTTP
- **Service** → Lógica de Negócio
- **Repository** → Acesso ao Banco



BACKEND

FLUXO: POST /API/CAVALOS

1. Router redireciona para o fluxo correto
2. Middleware valida JWT
3. Controller valida dados
4. Service aplica regras
5. Repository persiste no banco

1 - > cavaloRoutes.ts

```
router.post('/', authenticate, cavaloController.createCavalo.bind(cavaloController));
router.put('/:id', authenticate, cavaloController.updateCavalo.bind(cavaloController));
router.put('/:id/unavailable', authenticate, cavaloController.markCavaloAsUnavailable.bind(cavaloController));
router.put('/:id/available', authenticate, cavaloController.markCavaloAsAvailable.bind(cavaloController));
router.delete('/:id', authenticate, cavaloController.deleteCavalo.bind(cavaloController));
```

2- > CavaloController.ts

```
async createCavalo(req: Request, res: Response): Promise<void> {
  try {
    if (!req.user || !req.user.userId) {
      res.status(401).json({
        success: false,
        message: "Authentication required",
      });
      return;
    }

    const donoId = req.user.userId;
    const { nome, idade, raca, preco, descricao, disponivel, premios } = req.body;

    if (!nome || !idade || !raca || !preco) {
      res.status(400).json({
        success: false,
        message: "Nome, idade, raca, and preco are required",
      });
      return;
    }

    if (typeof idade !== 'number' || typeof preco !== 'number') {
      res.status(400).json({
        success: false,
        message: "Idade and preco must be numbers",
      });
      return;
    }

    const cavaloData: CreateCavaloDto = {
      nome,
      idade,
      raca,
      preco,
      descricao,
      disponivel,
      premios,
    };

    const cavalo = await this.cavaloService.createCavalo(cavaloData, donoId);
```



BACKEND

FLUXO: POST /API/CAVALOS

3- > CavaloService.ts

```
async createCavalo(cavaloData: CreateCavaloDto, donoId: string): Promise<CavaloDto>
  const dono = await this.cavaloRepository.findUserById(donoId);

  if (!dono) {
    throw new Error('Dono (owner) not found');
  }

  if (cavaloData.preco <= 0) throw new Error('Price must be greater than zero');
  if (cavaloData.idade <= 0 || cavaloData.idade > 50)
    throw new Error('Age must be between 1 and 50 years');

  const cavaloToCreate: Partial<Cavalo> & { type?: string } = {
    ...cavaloData,
    disponivel: cavaloData.disponivel ?? true,
    dono: dono,
    type: (cavaloData as any).type || 'Cavalo'
  };

  const cavalo = await this.cavaloRepository.create(cavaloToCreate);

  return this.toCavaloDto(cavalo);
}
```



4- > CavaloRepository.ts

```
async create(cavaloData: Partial<Cavalo>): Promise<Cavalo> {
  const cavalo = this.cavaloRepository.create(cavaloData);
  try {
    const savedCavalo = await this.cavaloRepository.save(cavalo);
    return savedCavalo;
  } catch (err) {
    throw err;
  }
}
```

BACKEND

CORS e Segurança

SEGURANÇA

- Origins controladas (dev: *, prod: específico)
- Headers: Content-Type, Authorization
- Methods: GET, POST, PUT, DELETE



JWT

- Tokens com expiração
- Senhas hasheadas (bcrypt)
- Validação em múltiplas camadas

BACKEND

FLUXO: AUTENTICAÇÃO

auth.middleware.ts

```
export const authenticate = (req: Request, res: Response, next: NextFunction): void => {
  try {
    const authHeader = req.headers.authorization;

    if (!authHeader) {
      res.status(401).json({
        success: false,
        message: 'Token não fornecido',
      });
      return;
    }

    const parts = authHeader.split(' ');

    if (parts.length !== 2) {
      res.status(401).json({
        success: false,
        message: 'Formato de token inválido',
      });
      return;
    }

    const [scheme, token] = parts;

    if (!token) {
      res.status(401).json({
        success: false,
        message: 'Token não fornecido',
      });
      return;
    }
  }
}
```

```
//testa se o token é válido
if (!scheme || !/^Bearer$/i.test(scheme)) {
  res.status(401).json({
    success: false,
    message: 'Token mal formatado',
  });
  return;
}

const jwtSecret = process.env.JWT_SECRET || 'your-secret-key-change-in-product';
const decoded = jwt.verify(token, jwtSecret) as any;

req.user = {
  userId: decoded.userId,
  email: decoded.email,
  tipo: decoded.tipo,
};

next();
} catch (error) {
  res.status(401).json({
    success: false,
    message: 'Token inválido ou expirado',
  });
}
```

🔒 Rotas protegidas

- **POST** /api/cavalos
- **PUT** /api/cavalos/:id
- **DELETE** /api/cavalos/:id
- **POST** /api/mensagens
- **GET** /api/mensagens/conversations

BACKEND

Qualidade

SEGURANÇA

Todas rotas da aplicação possuem documentação no Swagger completamente funcional com autenticação, pronto para teste em um futuro ambiente de staging.

EquiTrade API 1.0.0 OAS 3.0

API para plataforma de comércio de cavalos - EquiTrade

Contact EquiTrade Team

Servers
http://localhost:3000 - Development server

Anúncios Operações relacionadas ao gerenciamento de anúncios

GET	/api/anuncios	Buscar todos os anúncios
POST	/api/anuncios	Cadastrar novo anúncio
GET	/api/anuncios/{id}	Buscar anúncio por ID
PUT	/api/anuncios/{id}	Atualizar anúncio
DELETE	/api/anuncios/{id}	Excluir anúncio
GET	/api/anuncios/vendedor/{vendedorId}	Buscar anúncios por vendedor
PUT	/api/anuncios/{id}/inactive	Marcar anúncio como inativo
PUT	/api/anuncios/{id}/active	Marcar anúncio como ativo

Mensagens Operações relacionadas ao sistema de mensagens entre usuários

POST	/api/mensagens	Enviar mensagem
GET	/api/mensagens/{id}	Buscar mensagem por ID
DELETE	/api/mensagens/{id}	Excluir mensagem
GET	/api/mensagens/sent	Buscar mensagens enviadas
GET	/api/mensagens/received	Buscar mensagens recebidas
GET	/api/mensagens/conversations	Listar todas as conversas
GET	/api/mensagens/conversation/{userId}	Buscar conversa com usuário
Auth Autenticação e gerenciamento de sessão		
POST	/api/auth/login	Login de usuário
POST	/api/auth/register	Cadastro de novo usuário



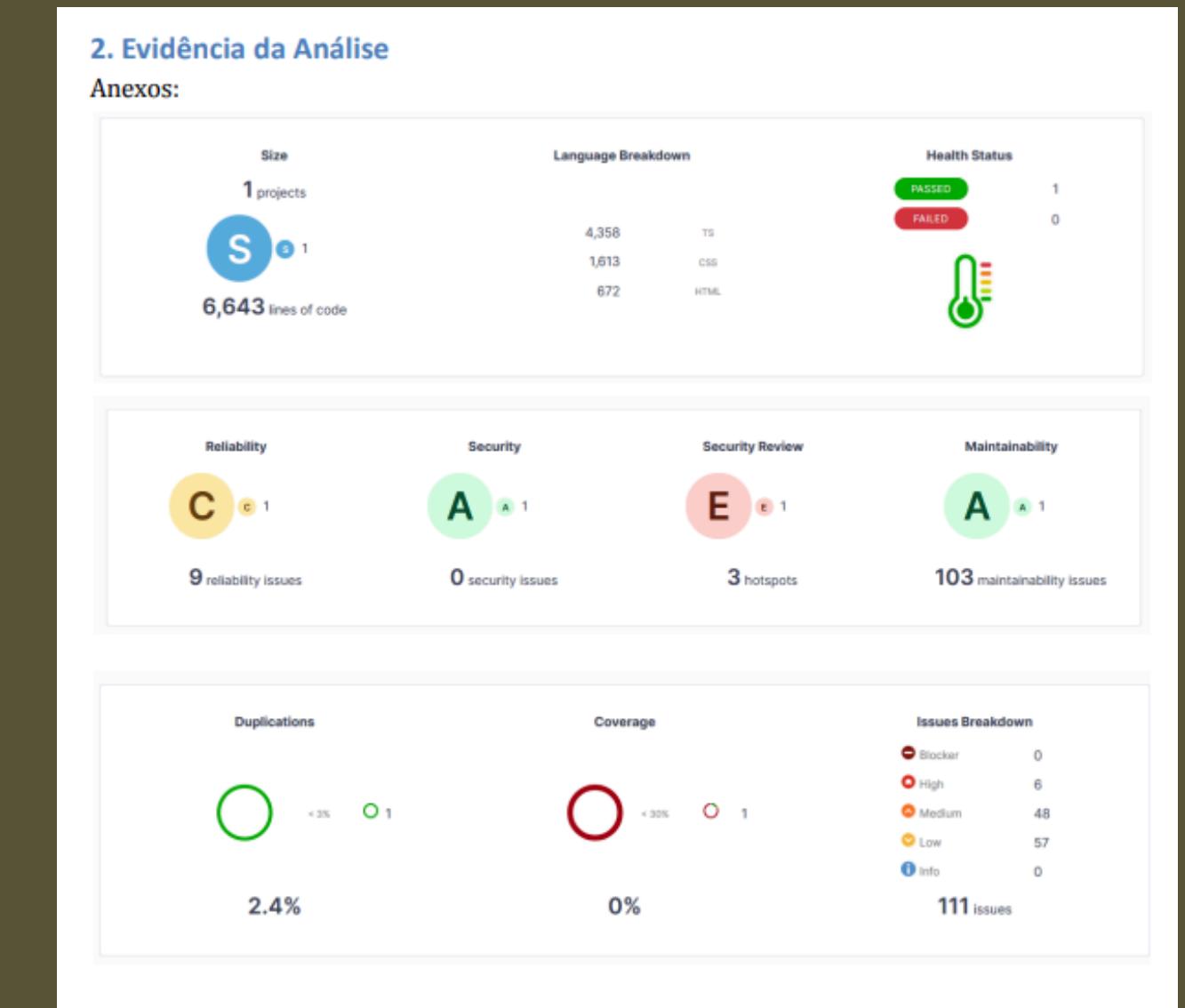
Testes e Qualidade

↗ ESTRATÉGIA DE TESTES

- Testes Unitarios
- Cavalos entity
- UserService
- UserController

↗ FERRAMENTAS UTILIZADAS

- supertest
- SonarQube





TECNOLOGIAS E FRAMEWORKS

- HTML + CSS + TypeScript no front-end
- Node.js + Express para criar as rotas HTTP
- TypeORM para mapear tabelas e acessar o banco
- Código organizado em controllers, services e repositories



Implementação (front-end)

PRINCIPAIS COMPONENTES

- Layout dividido em páginas: início, explorar, login/cadastro e área do usuário.
- Componentes reutilizáveis para cards de cavalos, formulários e navegação.
- Foco em simplicidade, leitura clara e fluxo intuitivo para o usuário.

FLUXO DE CHAMADAS ENTRE API → DOMÍNIO → PERSISTÊNCIA

- O front envia requisições REST para a API
- A API recebe a requisição e aciona os controllers, que validam e chamam os services correspondentes.
- Os services realizam a lógica de negócio e se comunicam com o banco de dados via Prisma/ORM.
- A resposta é devolvida em JSON para o front, que atualiza a interface.

Conclusões



Demonstração





Muito
obrigada por
assistir!