UNIVERSIDADE FEDERAL DE OURO PRETO

INSTITUTO DE CIÊNCIAS EXATAS E BIOLÓGICAS

DEPARTAMENTO DE COMPUTAÇÃO

PEDRO PARENTONI DE ALMEIDA

REPTILERECON - UMA APLICAÇÃO WEB PARA ANÁLISE DE SINAIS DE LAGARTOS

Ouro Preto
2026

PEDRO PARENTONI DE ALMEIDA

REPTILERECON - UMA APLICAÇÃO WEB PARA ANÁLISE DE SINAIS DE LAGARTOS

Monografia apresentada ao Curso de Ciência da Computação da Universidade Federal de Ouro Preto como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Jadson Castro Gertrudes

Ouro Preto
2026

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE OURO PRETO
REITORIA
INSTITUTO DE CIENCIAS EXATAS E BIOLOGICAS
DEPARTAMENTO DE COMPUTACAO

FOLHA DE APROVAÇÃO

Pedro Parentoni de Almeida

Reptilerecon - uma aplicação web para análise de sinais de lagartos

Monografia apresentada ao Curso de Ciência da Computação da Universidade Federal de Ouro Preto como requisito parcial para obtenção do título de Bacharel em Ciência da Computação

Aprovada em 24 de Fevereiro de 2026

Membros da banca

Jadson Castro Gertrudes (Orientador) - Doutor - Universidade Federal de Ouro Preto
Aline Norberta de Brito (Examinadora) - Doutora - Universidade Federal de Ouro Preto
Ricardo Pires Vidal Faustino (Examinador) - Bacharel - Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Ouro Preto

Jadson Castro Gertrudes, Orientador do trabalho, aprovou a versão final e autorizou seu depósito na Biblioteca Digital de Trabalhos de Conclusão de Curso da UFOP em 24/02/2026

Documento assinado eletronicamente por **Jadson Castro Gertrudes**, **PROFESSOR DE MAGISTERIO SUPERIOR**, em 19/03/2026, às 16:36, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site http://sei.ufop.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0 , informando o código verificador **1062457** e o código CRC **0A1A7849**.

R. Diogo de Vasconcelos, 122, - Bairro Pilar Ouro Preto/MG, CEP 35402-163
Telefone: 3135591692 - www.ufop.br

Dedico este trabalho à minha mãe, Karla Parentoni, por todo o apoio, amor e incentivo ao longo de toda a minha vida, e ao meu tio, Antônio Jose, que acreditou em mim e financiou meus estudos no Ensino Médio, tornando possível meu ingresso na universidade.

Agradecimentos

Agradeço ao meu orientador, Prof. Jadson Castro Gertrudes, pela orientação, paciência e dedicação ao longo de todo o desenvolvimento deste trabalho, bem como pelos ensinamentos e direcionamentos fundamentais para sua concretização.

Agradeço também a Ricardo Pires Vidal Faustino, por viabilizar as imagens que serviram de base para o desenvolvimento da interface do sistema.

Resumo

Com o avanço dos estudos sobre o comportamento animal, tornou-se necessária a superação dos métodos manuais de coleta e análise de dados, impulsionando o desenvolvimento de soluções computacionais mais robustas para observação, rastreamento e catalogação de características, reduzindo erros inerentes à intervenção humana. Este Trabalho de Conclusão de Curso apresenta o *Reptilerecon*, uma plataforma web desenvolvida para auxiliar no rastreamento automatizado dos movimentos de cabeceio de lagartos do gênero *Tropidurus*, utilizando o algoritmo YOLO para detecção e acompanhamento dos indivíduos em vídeo. O sistema foi implementado com arquitetura cliente-servidor, possuindo um *back-end* desenvolvido em Django, responsável pelo processamento, gerenciamento e disponibilização dos dados por meio de uma API REST. O *front-end* foi construído em Angular, proporcionando uma interface interativa e acessível aos usuários autenticados. A aplicação encontra-se plenamente funcional e está disponível em: <https://reptilerecon.csilab.ufop.br/>.

Palavras-chave: Django. YOLO. Tropidurus. Rastreamento de movimento. OAuth, Angular.

Abstract

With the advancement of studies in animal behavior, it has become necessary to overcome manual data collection and analysis methods, fostering the development of more robust computational solutions for observation, tracking, and characterization, thereby reducing errors inherent to human intervention. This Undergraduate Thesis presents *Reptilerecon*, a web-based platform developed to support the automated tracking of head-bobbing movements in lizards of the genus *Tropidurus*, using the YOLO algorithm for detection and tracking of individuals in video data.The system was implemented using a client-server architecture, featuring a Django-based *back-end* responsible for data processing, management, and exposure through a REST API. The *front-end* was developed in Angular, providing an interactive and accessible interface for authenticated users. The application is fully functional and available at: <https://reptilerecon.csilab.ufop.br/>.

Keywords: Django. YOLO. Tropidurus. Motion Tracking. OAuth,Angular.

Lista de Figuras

Lista de Tabelas

Sumário

1 Introdução

O gênero *Tropidurus* reúne espécies de lagartos amplamente distribuídas em ambientes variados do Brasil, desempenhando papel importante em seus ecossistemas como controladores de populações de insetos e como presas para diversos predadores. O estudo do comportamento desses animais, especialmente sinais visuais sutis como movimentos de cabeça, é fundamental para compreender aspectos ecológicos e comunicativos que influenciam sua sobrevivência e reprodução.

Entretanto, o aquecimento global tem causado alterações ambientais significativas, com aumento da temperatura média e redução das taxas de precipitação, afetando diretamente a fisiologia e o comportamento dos lagartos (PIANTONI, 2015). Além disso, estudos genéticos recentes apontam para alta diversidade críptica dentro do grupo *Tropidurus*, com limites entre espécies pouco resolvidos, o que ressalta a necessidade de dados comportamentais para melhor entender sua história evolutiva e suas interações ecológicas (WERNECK et al., 2015).

O monitoramento desses comportamentos ainda é majoritariamente realizado de forma manual, por meio da observação direta ou análise de vídeos, processos que demandam tempo significativo e estão sujeitos a vieses humanos (ZUERL et al., 2022). Essa limitação dificulta a obtenção de dados em larga escala e compromete a precisão necessária para estudos científicos rigorosos.

Nesse contexto, o framework desenvolvido por Zenobio et al. (2023) propôs o uso do algoritmo YOLO (*You Only Look Once*) para a detecção automática da cabeça de lagartos do gênero *Tropidurus* em vídeos, apresentando resultados promissores para o rastreamento de movimentos sutis. Entretanto, a ferramenta desenvolvida ainda não dispõe de uma plataforma acessível para que pesquisadores possam utilizar essa tecnologia de forma prática e eficiente.

Este trabalho propõe o desenvolvimento de uma plataforma web que integra o algoritmo YOLO proposto por Zenobio et al. (2023) para rastreamento automático dos movimentos de lagartos *Tropidurus*, com o objetivo de fornecer uma solução acessível e eficiente para pesquisadores da área biológica.

1.1 Justificativa

Os impactos do aquecimento global sobre espécies de répteis, em especial os lagartos do gênero *Tropidurus*, têm sido amplamente documentados, revelando alterações fisiológicas e comportamentais significativas (PIANTONI, 2015). Paralelamente, estudos genéticos recentes indicam a existência de diversidade críptica dentro do grupo, ressaltando a importância de dados comportamentais para compreender melhor suas interações ecológicas e evolutivas (WERNECK

et al., 2015).

No entanto, o monitoramento desses comportamentos, como movimentos sutis de cabeceio, ainda é realizado majoritariamente de forma manual, demandando tempo e estando sujeito a erros humanos. Essa limitação dificulta a análise em larga escala e compromete a precisão necessária para pesquisas voltadas à conservação da biodiversidade (ZUERL et al., 2022).

Diante desse cenário, o desenvolvimento de uma plataforma capaz de automatizar o rastreamento de movimentos em vídeos surge como uma solução viável e inovadora. Além de acelerar a coleta e análise de dados, tal ferramenta facilita o acesso de pesquisadores a tecnologias avançadas de visão computacional, aproximando as áreas de Biologia e Ciência da Computação.

A escolha deste tema também se justifica pelo interesse em explorar aplicações de aprendizado de máquina em problemas reais de conservação ambiental, contribuindo tanto para o avanço tecnológico quanto para a geração de conhecimento relevante para a preservação de espécies nativas.

1.2 Objetivos

Este trabalho tem como objetivo principal o desenvolvimento de uma plataforma web eficiente e acessível para o processamento e rastreamento automático de movimentos em vídeos de lagartos do gênero *Tropidurus*, utilizando um algoritmo pré-existente de aprendizado de máquina, o YOLO (*You Only Look Once*).

A plataforma tem como finalidade facilitar o trabalho de pesquisadores da área biológica, permitindo o upload simples de vídeos, o processamento automatizado para detecção de movimentos sutis, como o cabeceio, e a disponibilização dos dados resultantes de forma organizada e acessível.

Os objetivos específicos incluem:

- Integrar o algoritmo YOLO pré-existente proposto em (ZENOBIO et al., 2023) ao sistema, garantindo seu funcionamento eficiente dentro da plataforma.
- Desenvolver uma interface web intuitiva para usuários autenticados realizarem upload dos vídeos, visualizarem os resultados do rastreamento e exportarem os dados.

Com esses objetivos, a plataforma proposta visa apoiar pesquisas ecológicas e comportamentais, ampliando as ferramentas disponíveis para o monitoramento e conservação de espécies da fauna.

1.3 Organização do Trabalho

Este trabalho está organizado em capítulos que abordam de forma progressiva os aspectos fundamentais do desenvolvimento da plataforma para rastreamento automático de movimentos em lagartos *Tropidurus*. A estrutura adotada visa ser clara e sequencial, iniciando pela contextualização do tema e finalizando com as conclusões.

1.3.1 Estrutura da Monografia

Capítulo 1: Introdução.

Capítulo 2: Revisão Bibliográfica/ Embasamento Teórico (com o referencial teórico e trabalhos relacionados).

Capítulo 3: Metodologia ou Desenvolvimento (material e métodos).

Capítulo 4: Resultados e Discussões.

Capítulo 5: Conclusão (e trabalhos futuros).

2 Revisão Bibliográfica

2.1 Fundamentação Teórica

Esta seção fornece uma fundamentação teórica e explicações prévias sobre conceitos empregados ao longo do trabalho.

2.1.1 Tarefa de reconhecimento de sinais emitidos por lagartos

Em Furnas et al. (2019), destaca-se que os répteis são frequentemente muito mais difíceis de detectar, em parte devido ao seu comportamento tipicamente críptico e ao pequeno porte corporal. Além disso, a observação direta desses animais requer presença contínua no campo, muitas vezes sob condições adversas como calor extremo, vegetação densa e locais remotos, o que implica altos custos e limitações na cobertura temporal dos estudos. Nesse contexto, a estimativa de densidade e tamanho populacional foi realizada por meio de modelos estatísticos aplicados a dados obtidos em levantamentos visuais.

Adicionalmente, muitos métodos tradicionais para obtenção de dados sobre lagartos ainda são realizados de forma rústica. Por exemplo, em Ribeiro-Júnior, Gardner e Ávila-Pires (2006), é avaliada a eficácia da colocação de armadilhas de cola em diferentes locais para capturar amostras, ressaltando os desafios práticos e a necessidade de otimizar técnicas de amostragem para espécies de lagartos em ambientes complexos.

2.1.2 O Reptilerecon

O desenvolvimento da nova versão da aplicação Reptilerecon proposto neste trabalho teve como base versões anteriores da aplicação já existentes. O Reptilerecon é uma aplicação web projetada para ser segura, escalável e acessível em vários dispositivos, permitindo que usuários registrados e autenticados façam upload de vídeos. Esses vídeos são processados automaticamente com o objetivo de rastrear movimentos específicos de cabeceio de lagartos da espécie *Tropidurus*.

A motivação para o desenvolvimento da aplicação está ligada aos desafios contemporâneos enfrentados pela conservação da biodiversidade. O monitoramento de fauna, especialmente de espécies em ambientes naturais, demanda ferramentas que ofereçam precisão, eficiência e acessibilidade. Nesse contexto, o Reptilerecon se apresenta como uma solução que utiliza técnicas de visão computacional e aprendizado de máquina, especificamente utilizando o algoritmo YOLO (*You Only Look Once*), conhecido por sua precisão e velocidade em detecção de objetos em vídeos e imagens.

2.1.3 Django

Django é um framework de alto nível para desenvolvimento web, projetado para acelerar o processo de construção de aplicações, desde a concepção inicial até sua implementação final. Ele oferece uma série de funcionalidades integradas que facilitam o desenvolvimento de sistemas robustos e seguros. Entre seus recursos nativos, ele é comumente conhecido por suas capacidades de autenticação de usuários, gerenciamento de conteúdos, geração de sitemaps, suporte a feeds RSS, entre outros. Essas ferramentas permitem que os desenvolvedores foquem na lógica de negócio da aplicação, reduzindo significativamente o tempo de desenvolvimento.

Ele é projetado para auxiliar desenvolvedores na prevenção de falhas de segurança, como SQL Injection, Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF) e Clickjacking. Além disso, o sistema de autenticação de usuários fornecido pelo framework oferece uma abordagem segura para o gerenciamento de credenciais e controle de acesso(Django Software Foundation,).

A Figura 2.1 ilustra o fluxo básico de funcionamento do Django, baseado no padrão MTV (Model-Template-View), que é similar ao padrão MVC (Model-View-Controller) de outros frameworks. O fluxo pode ser detalhado da seguinte forma:

1. **HTTP Request:** Um usuário envia uma requisição ao servidor (por exemplo, acessando uma página web). Essa requisição chega ao Django e é processada pelo arquivo `urls.py`.

2. **URLs (`urls.py`):** O arquivo `urls.py` mapeia as URLs do site para funções ou classes específicas chamadas *views*. Ele encaminha a requisição HTTP para a view apropriada, dependendo da rota acessada pelo usuário.

3. **View (`views.py`):** A view processa a requisição, podendo interagir com o `model` para ler ou gravar dados no banco de dados. Ela também seleciona qual template será usado para gerar a resposta HTML e retorna uma `HTTP Response` para o usuário.

4. **Model (`models.py`):** O model representa a estrutura dos dados da aplicação (por exemplo, tabelas do banco de dados), permitindo que a view leia ou escreva informações de forma organizada.

5. **Template (`<filename>.html`):** Contém a estrutura HTML que será exibida ao usuário. A view envia os dados do model para o template, que preenche as informações dinâmicas.

6. **HTTP Response:** Após processar os dados e renderizar o template, a view envia a resposta HTTP de volta para o navegador, exibindo a página completa com os dados solicitados.

Resumo do fluxo: Usuário envia HTTP Request → `urls.py` escolhe a view correta → View interage com model e template → Template gera HTML → View retorna HTTP Response para o navegador.

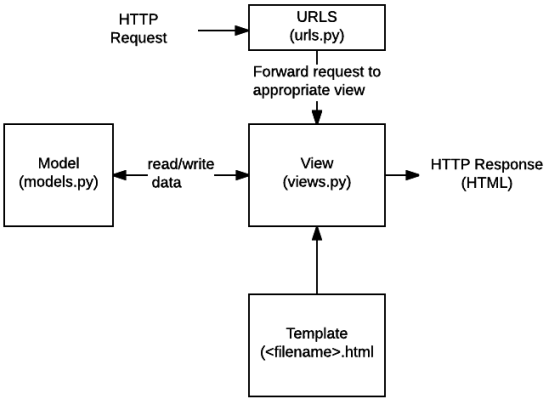

Figura 2.1 – Estrutura básica de um projeto Django. Fonte: Mozilla Developer Network.

Fonte: <https://developer.mozilla.org/en-US/docs/Learn_web_development/Extensions/Server-side/Django/Introduction>

2.1.4 Django Rest Framework

O Django Rest Framework é uma biblioteca poderosa para a construção de APIs web, oferecendo alto nível de personalização, ampla documentação e uma comunidade ativa. Seu uso é adotado por empresas reconhecidas internacionalmente, como Mozilla, Red Hat, Heroku e Eventbrite, o que evidencia sua robustez e confiabilidade (Encode OSS Ltd., 2025).

Todo o fluxo de uma requisição via Django Rest Framework segue um caminho bem definido:

- **Cliente / Frontend:** Tudo começa quando um cliente (como um navegador ou app) faz uma requisição HTTP para a API REST. Isso pode ser uma requisição GET, POST, PUT ou DELETE, dependendo da operação desejada.
- **URLs** (`urls.py`)**:** O Django recebe a requisição e passa pelo arquivo `urls.py`, que mapeia a URL requisitada para a view correta. O `urls.py` funciona como um "guia" que direciona a requisição ao ponto certo da aplicação.
- **Views** (`views.py`)**:** A view é responsável por processar a requisição. Ela pode interagir com os models para obter ou alterar dados no banco de dados. Após processar os dados, a view retorna uma resposta HTTP (normalmente em formato JSON no caso de APIs REST).
- **Models** (`models.py`)**:** O model representa a estrutura dos dados da aplicação. Ele abstrai as operações no banco de dados, permitindo que a view leia e escreva dados sem lidar diretamente com SQL.
- **Static URLs / Frontend Assets:** Arquivos estáticos, como CSS, JavaScript ou imagens, são servidos via static URLs. Eles podem complementar a interface do cliente que consome a API, mas não fazem parte do processamento principal de dados.

Resumo do fluxo: Cliente → `urls.py` escolhe a view correta → `views.py` interage com `models.py` e processa a lógica → `views.py` gera HTTP Response → Cliente.

Ou seja, o cliente envia uma requisição, a view processa usando os modelos de dados, e a resposta retorna para o cliente em JSON.

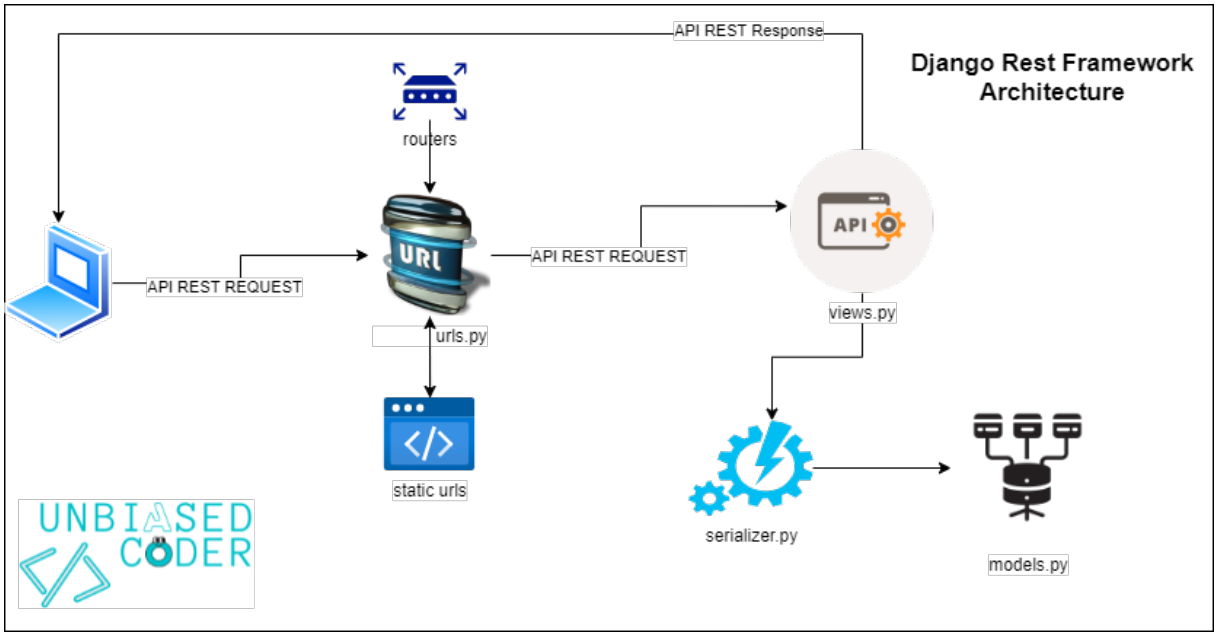

Figura 2.2 – Diagrama de fluxo do Django Rest Framework mostrando o caminho da requisição do cliente até a resposta.

Fonte: <https://unbiased-coder.com/setup-django-rest-framework/>

Enquanto o Django tradicional se concentra principalmente em gerar páginas HTML para o usuário final, o Django Rest Framework fornece endpoints que retornam dados estruturados, normalmente em formato JSON. Essa abordagem é ideal para aplicações, onde o front-end é implementado separadamente, utilizando frameworks como React, Angular ou Vue.js, que consomem a API para renderizar interfaces dinâmicas.

Dessa forma, o Django Rest Framework complementa o desenvolvimento front-end, separando a lógica de negócio e a apresentação, enquanto mantém a mesma segurança fornecida pelo Django tradicional.

2.1.5 Angular

Angular é um framework para desenvolvimento web voltado à construção de aplicações dinâmicas, eficientes e escaláveis. Mantido por uma equipe dedicada do Google, o framework oferece um amplo conjunto de ferramentas, bibliotecas e interfaces de programação de aplicações (APIs), que visam simplificar e otimizar o fluxo de trabalho no desenvolvimento front-end. Trata-se de uma reestruturação completa de sua versão anterior, o AngularJS, atualmente descontinuado. De acordo com a pesquisa anual Stack Overflow Developer Survey, Angular figura entre os

frameworks web mais utilizados pela comunidade de desenvolvedores. (JAIN; BHANSALI; MEHTA, 2014)

2.1.6 Modelo de deep learning *YOLO*

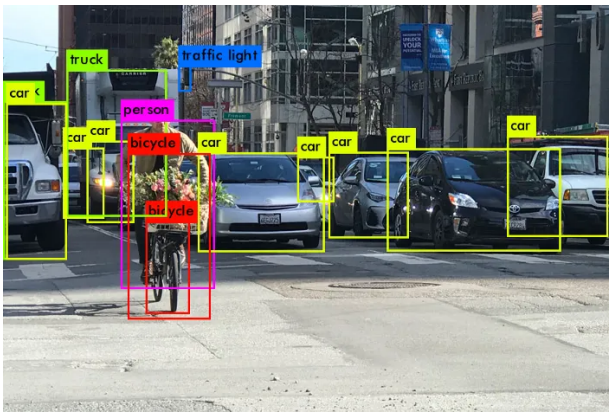

Figura 2.3 – Exemplo de funcionamento do YOLO.

Fonte: <https://medium.com/analytics-vidhya/yolo-explained-5b6f4564f31>

You Only Look Once (YOLO) é uma abordagem para a tarefa de detecção de objetos onde, diferente de métodos anteriores, que adaptavam classificadores para realizar a detecção, ele reformula o problema como uma tarefa de regressão, mapeando diretamente uma imagem para as caixas delimitadoras (bounding boxes), como demonstrado nas figuras 2.4 e 2.5 , e as probabilidades das classes associadas.

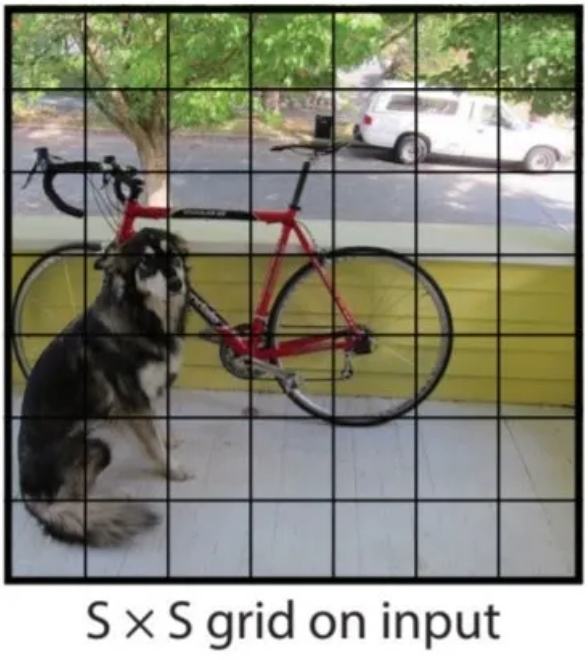

Figura 2.4 – Caixas delimitadoras do YOLO.

Fonte: <https://medium.com/analytics-vidhya/yolo-explained-5b6f4564f31>

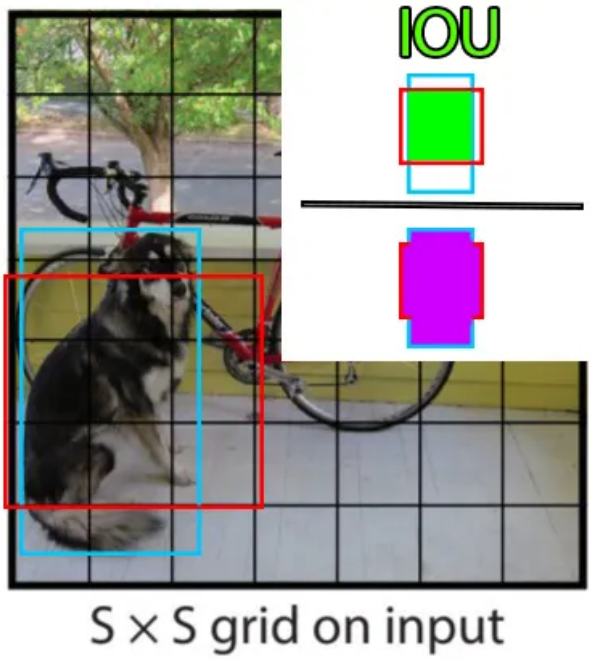

Figura 2.5 – Caixas delimitadoras do YOLO com predição. .

Fonte: <https://medium.com/analytics-vidhya/yolo-explained-5b6f4564f31>

A detecção é realizada por uma única rede neural que processa a imagem por completo em uma única avaliação, o que permite a otimização de ponta a ponta com foco no desempenho da detecção. Essa arquitetura unificada proporciona alta velocidade de processamento e mesmo que cometa mais erros de localização em relação a outros sistemas, o YOLO apresenta menor propensão a falsos positivos em áreas de fundo. Além disso, o modelo demonstra capacidade de generalização superior, superando outros métodos ao ser aplicado em domínios distintos, como imagens artísticas (REDMON et al., 2016).

2.1.7 OAuth 2.0

OAuth 2.0 é um protocolo de autorização que permite que uma aplicação obtenha acesso limitado a recursos protegidos em um servidor HTTP, sem precisar usar diretamente as credenciais do proprietário do recurso. No modelo tradicional de autenticação cliente-servidor, o cliente (aplicação) acessa recursos protegidos no servidor autenticando-se com as credenciais do usuário (proprietário do recurso), o que leva a diversos problemas, como a necessidade do compartilhamento e armazenamento de senhas por terceiros, riscos de segurança e falta de controle sobre os níveis de acesso e duração da autorização.

O protocolo OAuth 2.0 resolve essas questões ao introduzir uma camada de autorização que separa as funções do cliente e do proprietário do recurso. Em vez de compartilhar a senha do usuário, o cliente recebe um token de acesso emitido por um servidor de autorização, com a aprovação do proprietário do recurso. Esse token contém informações sobre o escopo, tempo

de validade e outras restrições de acesso, permitindo que o cliente acesse apenas os recursos autorizados, por um período determinado.

Por exemplo, um usuário pode conceder a um serviço de impressão acesso às suas fotos armazenadas em um serviço de compartilhamento, sem precisar compartilhar seu nome de usuário e senha com o serviço de impressão. O usuário autentica-se diretamente com um servidor confiável, que emite ao serviço de impressão um token de acesso específico para essa autorização.

O OAuth 2.0 foi projetado para ser usado com o protocolo HTTP e representa uma evolução do protocolo OAuth 1.0, trazendo melhorias de segurança, flexibilidade e usabilidade. (HARDT, 2012)

2.1.8 Uso do OAuth 2.0 pelo Google

O Google utiliza o protocolo OAuth 2.0 para gerenciar a autenticação e autorização de usuários em suas aplicações e serviços. Por meio dessa implementação, o Google permite que aplicativos de terceiros acessem recursos protegidos, como dados de perfil ou arquivos, sem a necessidade de compartilhar as credenciais do usuário, garantindo maior segurança e controle.

O fluxo de autorização do Google envolve a solicitação de permissões específicas por parte do aplicativo, a autenticação do usuário pelo Google, e a emissão de tokens de acesso que controlam o nível e o tempo de acesso concedidos (Google Developers, 2025).

Este mecanismo é amplamente adotado em serviços web modernos, facilitando a integração de aplicações com as plataformas do Google de forma segura e padronizada.

2.1.9 Nginx

O nginx (“engine x”) é um servidor web HTTP, proxy reverso, cache de conteúdo, balanceador de carga, servidor proxy TCP/UDP e servidor proxy de e-mail. Originalmente escrito por Igor Sysoev. Ele é conhecido pela flexibilidade e alto desempenho com baixo uso de recursos, sendo um dos servidores web mais populares do mundo.

Ele tem como algumas de suas funcionalidades: Servir arquivos estáticos e arquivos de índice, com suporte à autoindexação e cache de descritores de arquivos abertos; Atuar como proxy reverso com suporte a cache, balanceamento de carga e tolerância a falhas e suporte acelerado, com cache, para servidores FastCGI, uWSGI, SCGI e memcached, também com balanceamento de carga e tolerância a falhas.

Por conta de sua arquitetura eficiente e modular, o nginx é amplamente utilizado em ambientes de alta demanda, como servidores web de grande escala, aplicações em contêineres e infraestrutura baseada em computação em nuvem. (NGINX, Inc., 2025)

2.1.10 Gunicorn

O Gunicorn (“Green Unicorn”) é um servidor HTTP WSGI para Python voltado a sistemas UNIX, projetado para o ambiente de produção de aplicações web. Ele utiliza um modelo de *workers* do tipo *pre-fork*, sendo amplamente utilizado em conjunto com frameworks como Django e Flask, devido à sua simplicidade, bom desempenho e baixo consumo de recursos.

Algumas das suas principais funcionalidades são: A execução de aplicações web Python compatíveis com o padrão WSGI; O suporte a diferentes tipos de *workers*, permitindo adaptação a diferentes cargas de trabalho e o gerenciamento eficiente dos processos de *workers*, com inicialização e encerramento controlados.

Por conta da sua estabilidade e ampla adoção em ambientes reais de produção, o Gunicorn é frequentemente utilizado em conjunto com servidores de proxy reverso, como o nginx, gerando uma arquitetura consistente para aplicações web. (CHESNEAU, 2025)

2.1.11 Celery

O Celery é um sistema distribuído de filas de tarefas desenvolvido em Python, projetado para o processamento assíncrono e em tempo real de grandes volumes de trabalho. Ele permite a distribuição de tarefas entre múltiplos *workers*, possibilitando escalabilidade horizontal e alta disponibilidade.

Entre suas principais funcionalidades, destacam-se: A distribuição de tarefas por meio de filas, com comunicação baseada em mensagens entre clientes e *workers*; O suporte a múltiplos *brokers* de mensagens, como RabbitMQ e Redis e a execução concorrente de tarefas, com diferentes modelos de concorrência, incluindo multiprocessamento e multithreading.

Devido à sua flexibilidade, desempenho e ampla adoção em ambientes de produção, o Celery é amplamente utilizado para o processamento assíncrono, agendamento de tarefas e execução de trabalhos em segundo plano em aplicações web modernas. (Celery Project Contributors, 2025)

2.2 Trabalhos Relacionados

Piantoni (2015) analisou os efeitos do aquecimento global em populações do complexo *Tropidurus torquatus* no Brasil, demonstrando que o aumento da temperatura e a redução da precipitação afetam diretamente a fisiologia e o comportamento das espécies. Tais resultados reforçam a necessidade de ferramentas de monitoramento comportamental para avaliar como essas alterações ambientais influenciam os lagartos em seus habitats naturais.

De forma complementar, Werneck et al. (2015) investigaram a história biogeográfica e a diversidade genética do grupo *Tropidurus semitaeniatus*, revelando microendemismos e alta

diversidade críptica na Caatinga. Os autores destacam a carência de dados para delimitação das espécies e compreensão de suas relações evolutivas, indicando que novas abordagens, como o rastreamento automático de comportamento, podem contribuir significativamente para esse campo.

Nesse contexto, diversos frameworks têm sido desenvolvidos para o monitoramento automatizado do comportamento animal. O trabalho de Mönck et al. (2018) propõe o BioTracker, um software de visão computacional de código aberto projetado para análise de vídeo em diferentes condições. Sua arquitetura modular fornece funcionalidades básicas, como leitura e gravação de vídeos, sobreposições gráficas e interação por mouse e teclado, e permite a integração de algoritmos personalizados de rastreamento. Essa flexibilidade é especialmente útil para estudos que exigem soluções adaptadas, como o acompanhamento de cardumes ou de insetos em colônias.

De maneira semelhante, Romero-Ferrero et al. (2018) introduziram o idtracker.ai, capaz de rastrear até 100 indivíduos não marcados com alta precisão. O sistema combina duas redes neurais profundas: uma para identificar momentos de cruzamento ou contato entre animais e outra para a identificação individual adaptativa. Essa abordagem supera as limitações do idTracker original, restrito a grupos pequenos, ao implementar protocolos de treinamento progressivos que acumulam imagens de cada indivíduo, garantindo precisão mesmo em cenários de oclusão frequente.

O framework de Silva (2018), base para este estudo, aplica técnicas de Processamento Digital de Imagens e Aprendizado de Máquina para rastrear manualmente a cabeça de lagartos da espécie *Tropidurus*, a partir de templates definidos pelo pesquisador. Embora testes tenham confirmado sua eficácia, a necessidade de marcação manual introduz inconsistências, sobretudo em vídeos com movimentos rápidos e sutis, comprometendo a precisão dos gráficos gerados.

Para superar tais limitações, o trabalho de Zenobio et al. (2023) apresenta uma versão aprimorada do framework, que integra aprendizado profundo para a detecção automática da cabeça dos animais e algoritmos não supervisionados para identificação de padrões latentes nos sinais extraídos. Essa evolução permitiu análises mais eficientes em grandes volumes de dados, ampliando o escopo de aplicação da ferramenta e apresentando uma versão beta em um domínio(website) privado para que fosse possível testar o modelo, mesmo que sem quaisquer foco em segurança, autentificação de usuário e facilidade de uso geral.

Nessa mesma linha, Zuerl et al. (2022) propõem um framework baseado em aprendizado profundo para o monitoramento de ursos-polares em cativeiro. A solução utiliza o algoritmo YOLO para detecção e rastreamento, substituindo a observação manual tradicional, geralmente dependente de anotações de tratadores e sujeita a vieses e restrições de tempo, por medições objetivas e reprodutíveis. Além de fornecer dados contínuos e escaláveis, os autores destacam que a metodologia pode ser adaptada para outras espécies e ambientes, configurando-se como uma ferramenta promissora para estudos de bem-estar e conservação.

Esses trabalhos, em conjunto, evidenciam a relevância de sistemas automatizados de monitoramento animal, fornecendo a base para o desenvolvimento do Reptilerecon, que busca auxiliar na análise comportamental e na conservação de espécies de répteis.

3 Desenvolvimento

3.1 Visão geral do sistema

O sistema Reptilerecon é uma aplicação web destinada a pesquisadores que desejam analisar o comportamento de lagartos a partir de vídeos. Para isso, o sistema oferece uma interface amigável para o upload de vídeos, processamento dos dados e apresentação dos resultados do rastreamento.

A arquitetura do sistema é dividida em três camadas principais: o front-end, desenvolvido em Angular, que provê a interface para os usuários; o back-end, implementado com Django, Django REST Framework e Celery, responsável pelo processamento e lógica de negócio; e o banco de dados MySQL, que armazena as informações de usuários, vídeos e resultados.

O fluxo básico de uso do sistema inicia-se com o cadastro e autenticação do usuário, seguido pelo upload dos vídeos que são processados no servidor. Os resultados são disponibilizados para consulta via interface web.

A Figura 3.1 apresenta o Diagrama de Componentes do sistema, elaborado segundo a notação UML. O diagrama ilustra a organização estrutural da aplicação, destacando os principais componentes de software e suas dependências.

O objetivo do diagrama é representar a arquitetura geral do sistema e a forma como seus componentes interagem, não descrevendo fluxos de execução específicos, mas sim as relações estruturais entre os elementos.

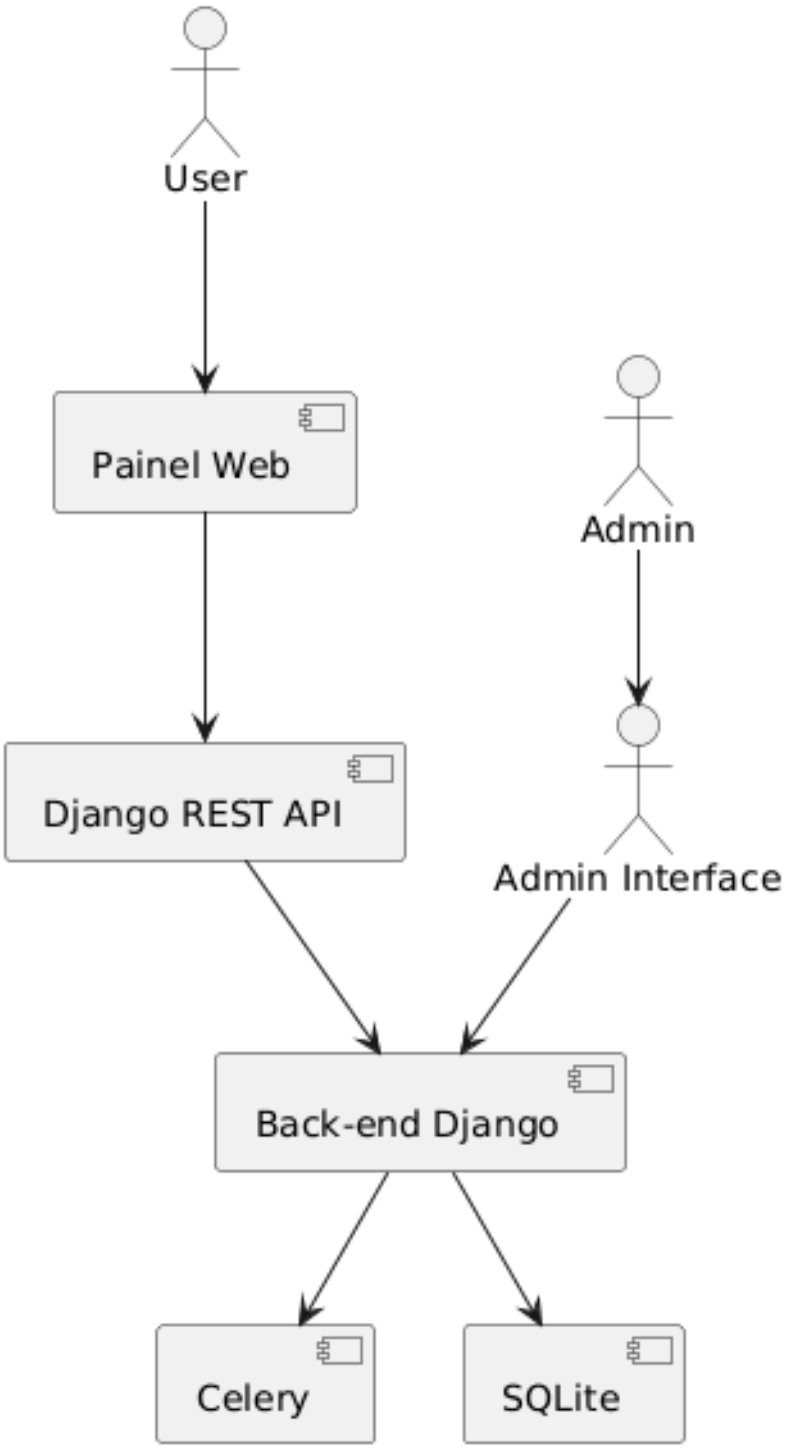

Figura 3.1 – Diagrama de Componentes do Sistema

Fonte: Elaborado pelo autor

3.2 Desenvolvimento do sistema

3.2.1 Front-end

O front-end da aplicação Reptilerecon foi desenvolvido utilizando o framework Angular, sendo hospedado e servido estaticamente por meio de um servidor web Nginx (NGINX, Inc., 2025) no domínio <https://reptilerecon.csilab.ufop.br/>. Essa camada da aplicação é responsável por oferecer uma interface intuitiva e acessível aos usuários, facilitando a interação com as funcionalidades do sistema. Para uma melhor compreensão da sua estrutura, a seção 3.2.1.1 detalha os principais componentes que compõem o front-end, enquanto a seção 3.2.1.2 descreve os serviços que suportam a comunicação e lógica da aplicação. Por fim, a seção 3.2.1.3 apresenta os caminhos de navegação que permitem o acesso às diferentes páginas disponíveis.

3.2.1.1 Componentes

- **RegisterComponent**: Facilita o registro de novos usuários, integrando processos de verificação de e-mail e configuração de autenticação de dois fatores.
- **LoginComponent**: Oferece um sistema de login robusto com suporte para autenticação de dois fatores, aumentando a segurança do usuário.
- **PasswordResetComponent**: Oferece um sistema para que o usuário seja capaz de resetar sua senha caso tenha vontade ou tenha perdido sua senha original.
- **UploadComponent**: Permite o upload seguro de vídeos, com verificações para garantir que os arquivos sejam legítimos e não comprometam o sistema.
- **VideoListComponent**: Mostra uma lista dos vídeos carregados pelo usuário, proporcionando uma interface simples para acessar e gerenciar os vídeos.
- **DeleteComponent**: Permite que o usuário delete de forma segura os vídeos, com verificações para garantir que os arquivos sejam legítimos e não comprometam o sistema.
- **VideoDetailComponent**: Exibe detalhes e análises dos vídeos processados, permitindo aos usuários visualizar os resultados do rastreamento.

3.2.1.2 Serviços

- **AuthService**: Administra todas as operações relacionadas à autenticação, incluindo registro, login e logout.
- **VideoService**: Responsável pela comunicação entre o frontend e o backend, gerenciando o upload de vídeos e a recuperação de dados de vídeos processados.
- **UserService**: Responsável pela comunicação entre frontend e backend gerenciando dados básicos do usuário como nome, email e senha.

3.2.1.3 Rotas

O sistema utiliza uma arquitetura baseada na separação entre frontend e backend, onde o frontend é desenvolvido em Angular e o backend em Django, comunicando-se principalmente por meio de rotas de API. A seguir são descritas as principais rotas do sistema.

- `/login`: Rota do frontend responsável pela autenticação do usuário no sistema.
- `/register`: Página de registro de novos usuários.
- `/home`: Página inicial do sistema após a autenticação.
- `/faq`: Página pública com informações frequentes sobre o funcionamento da plataforma.
- `/reset/:uid/:token`: Rota utilizada no processo de redefinição de senha, acessada a partir do link enviado por e-mail ao usuário.
- `/auth/callback`: Rota pública utilizada para receber e processar tokens de autenticação, realizando a validação e redirecionamento do usuário no frontend.
- `/accounts/redirect/`: Rota do backend responsável por redirecionar o usuário autenticado do Django para o frontend Angular, anexando o token de autenticação necessário para manter a sessão integrada entre as duas aplicações.
- `/api/`: Conjunto de rotas responsáveis pela comunicação entre frontend e backend. Essas rotas seguem o padrão REST e incluem funcionalidades como autenticação, gerenciamento de usuários, upload de vídeos, acompanhamento do progresso de processamento e atualização de dados do perfil do usuário.
- `/admin`: Interface administrativa do Django, utilizada exclusivamente por superusuários para gerenciamento direto dos dados do sistema. Por questões de segurança, essa rota é restrita ao acesso a partir da rede da Universidade Federal de Ouro Preto.

3.2.2 Back-end

O back-end da aplicação foi desenvolvido utilizando o framework Django (Django Software Foundation,), complementado pela extensão Django Rest Framework para a construção de uma API RESTful (Encode OSS Ltd., 2025). Essa camada é responsável pelo processamento das requisições, aplicação das regras de negócio e comunicação com o banco de dados.

Para garantir maior estabilidade, desempenho e tolerância a falhas em ambiente de produção, a aplicação Django é executada por meio do servidor de aplicação Gunicorn, atuando em conjunto com o servidor web Nginx, que funciona como proxy reverso. Isso garante um balanceamento eficiente das requisições e prevenção de erros causados por sobrecarga direta

no servidor de aplicação, contribuindo para a confiabilidade do sistema (NGINX, Inc., 2025; CHESNEAU, 2025).

Além disso, os processamentos e análise de vídeos submetidos pelos usuários são executados de forma assíncrona utilizando o framework Celery (Celery Project Contributors, 2025). Essa abordagem evita que operações de longa duração bloqueiem o servidor Gunicorn, reduzindo o risco de esgotamento de memória e garantindo que as requisições da API permaneçam responsivas mesmo sob alta demanda. O uso de processamento assíncrono também contribui para a escalabilidade do sistema, permitindo que tais tarefas sejam delegadas a workers dedicados.

Para detalhar a estrutura do back-end, a seção 3.2.2.1 apresenta os principais modelos de dados da aplicação, enquanto a seção 3.2.2.2 descreve os serializadores responsáveis pela conversão entre objetos Python e representações em formato JSON. A seção 3.2.2.3 aborda as views que implementam a lógica dos endpoints expostos, os quais são detalhados na seção 3.2.2.4. Por fim, a seção 3.2.2.5 descreve a estrutura do banco de dados utilizado e seus principais componentes.

3.2.2.1 Modelos

Os modelos do django são guardados no arquivo models.py do código. Eles tratam das diversas classes utilizadas no sistema.

- **ReptilerUser**: É uma extensão do usuário padrão do Django que será utlizada para guardar os usuários do sistema. A decisão de expandir o usuário base do Django foi feita para se aproveitar do modelo de autentificação padrão dele, assim como na possibilidade de utilizar o Google Oauth para a realização de Login (Google Developers, 2025).

  Um usuário Reptiler não é considerado como um administrador de sistema, portanto ele não tem acesso ao painel administrativo do Django.

- **Video**: Contém informações sobre os vídeos carregados, incluindo caminho do arquivo, data de upload e status do processamento. Ademais, cada vídeo tem um link com o usuário que o enviou, facilitando a listagem deles e o rastreio para identificar a origem.

3.2.2.2 Serializers

Essenciais para a conversão entre dados JSON e os modelos do Django, facilitando a serialização dos dados de usuários, vídeos e resultados.

3.2.2.3 Views

Fornecem a lógica por trás dos endpoints da API, tratando operações como registro, login, logout, upload de vídeos e recuperação de detalhes dos resultados.

3.2.2.4 Endpoints

Cada endpoint da API é detalhado para facilitar o acesso e manipulação dos dados pelo frontend.

3.2.2.5 Banco de Dados

O sistema utiliza o banco de dados relacional SQLite3, padrão do framework Django. A modelagem foi estruturada a partir dos modelos definidos na aplicação, sendo automaticamente convertida em tabelas relacionais por meio do sistema de *migrations* do Django.

De forma simplificada, a estrutura principal do sistema envolve:

- **auth_user**: Tabela padrão do Django responsável pelo armazenamento base das credenciais e informações essenciais de autenticação dos usuários.
- **reptilerUser**: Extensão do modelo *auth_user*, contendo informações adicionais específicas da aplicação.
- **videos**: Tabela responsável pelo registro dos vídeos enviados pelos usuários, mantendo associação com o respectivo *reptilerUser*.
- **django_site**: utilizada para gerenciamento de domínios e configurações de site.
- Conjunto de tabelas relacionadas ao sistema de autenticação social, como **socialaccount**, **socialapp**, **socialtoken** e **socialsites**.
- **auth_group** e tabelas de permissões associadas, responsáveis pelo controle de autorização e definição de privilégios no sistema.

A Figura 3.2 apresenta um Diagrama Relacional resumido da estrutura do banco de dados. O diagrama foi simplificado intencionalmente, ocultando algumas tabelas internas e auxiliares geradas automaticamente pelo Django, a fim de destacar o fluxo principal de relacionamento entre o usuário e os vídeos associados.

A simplificação do diagrama tem como objetivo evidenciar a relação central da aplicação: o vínculo entre o usuário autenticado (via *auth_user* e sua extensão *reptilerUser*) e os vídeos submetidos ao sistema para processamento.

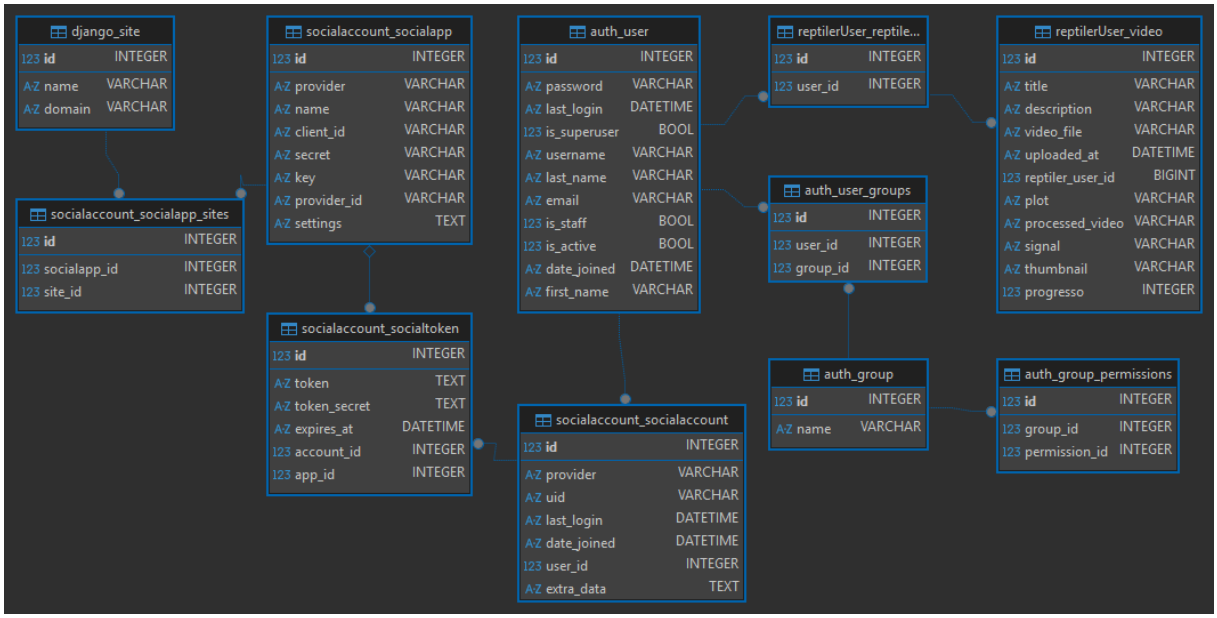

Figura 3.2 – Diagrama Relacional do banco de dados

Fonte: Elaborado pelo autor

3.2.3 Configuração do Ambiente de Produção

Para a implantação do sistema em ambiente de produção, foi adotada uma arquitetura baseada na separação de responsabilidades entre servidor web, servidor de aplicação e módulo de processamento assíncrono.

O front-end da aplicação, desenvolvido em Angular, é compilado para arquivos estáticos (HTML, CSS e JavaScript) e disponibilizado diretamente pelo servidor web Nginx (NGINX, Inc., 2025). Dessa forma, o Nginx atua como servidor de conteúdo estático, entregando a interface da aplicação aos usuários sem a necessidade de processamento adicional pelo back-end.

O back-end desenvolvido em Django (Django Software Foundation,) é executado por meio do servidor WSGI Gunicorn (CHESNEAU, 2025), responsável por gerenciar os processos da aplicação e atender às requisições HTTP internas. O Gunicorn opera escutando localmente na porta 8000, não sendo exposto diretamente à internet. As requisições destinadas à API REST, implementada com o Django REST Framework (Encode OSS Ltd., 2025), e ao painel administrativo são encaminhadas ao Gunicorn pelo Nginx por meio de proxy reverso.

O processamento de tarefas computacionalmente intensivas, como a análise de vídeos submetidos pelos usuários, é realizado de forma assíncrona utilizando o Celery (Celery Project Contributors, 2025). O Celery é executado como serviço independente do sistema operacional, configurado via *systemd*, garantindo inicialização automática e reinicialização em caso de falhas. O Django delega tarefas ao Celery, que as processa em segundo plano, evitando o bloqueio das requisições HTTP e melhorando a responsividade da aplicação.

Além disso, o Nginx também é responsável por servir arquivos de mídia gerados pelo

sistema e por implementar o redirecionamento automático de HTTP para HTTPS. A comunicação segura foi estabelecida por meio de certificados SSL obtidos via Let's Encrypt, garantindo criptografia das requisições externas.

O painel administrativo do Django (Django Software Foundation,) foi protegido por restrição de acesso baseada em endereço IP, permitindo acesso apenas à rede institucional da universidade, o que aumenta a segurança do sistema em ambiente de produção.

4 Resultados

4.1 Frontend

Esta sessão apresenta os resultados obtidos no desenvolvimento da aplicação, demonstrando todas as telas de usuário disponíveis no front end.

4.1.1 Home

A Figura 4.1 apresenta a págima base da aplicação, por meio da qual o usuário poderá acessar as telas de Login, Registro e FAQ, além de conter um link para o website do csilab, laboratório do qual o professor Jadson faz parte.

Figura 4.1 – Página Home — Front-end

Fonte: Elaborado pelo autor

4.1.2 Login

A Figura 4.2 apresenta a tela de login, por meio da qual o usuário poderá acessar suas informações e utilizar as funcionalidades do sistema. Nela também é possível retornar à página Home por meio do texto Reptilerecon, além de acessar a página de registro,realizar a autentificação com uma conta do google ou enviar um pedido de recuperação de senha para um email cadastrado.

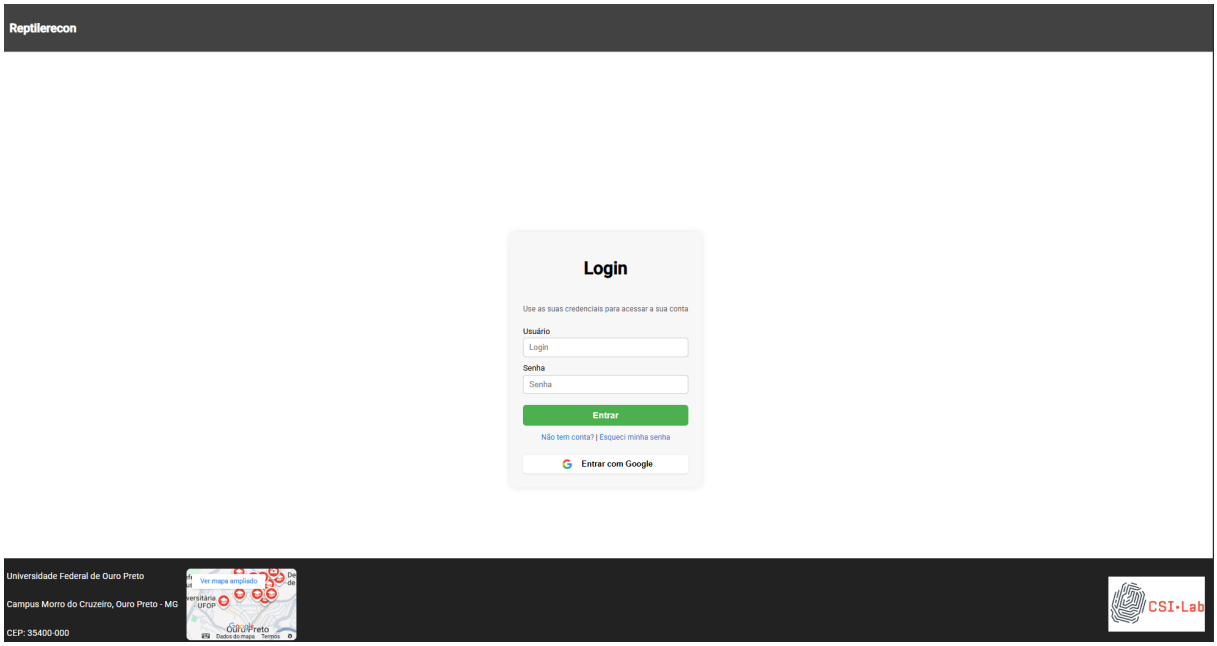

Figura 4.2 – Página de login — Front-end

Fonte: Elaborado pelo autor

4.1.3 Página de Registro

A Figura 4.3 apresentaa tela de registro de usuário do sistema *Repitilerecon* onde o usuário faz o cadastro para acessar as funcionalidades.

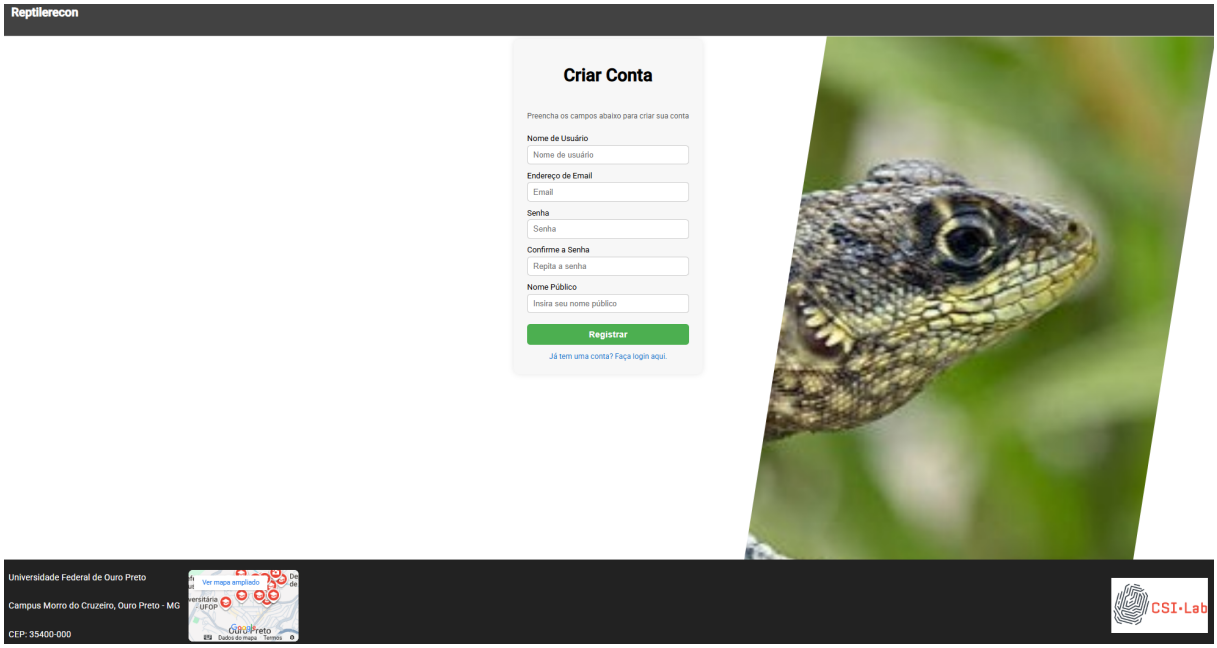

Figura 4.3 – Página de registro — Front-end

Fonte: Elaborado pelo autor

4.1.4 Página de Redefinição de senha

A Figura 4.4 apresenta a tela de redefenição de senha de usuário do sistema *Repitilerecon* onde o usuário faz a troca de sua senha antiga após acessar a página por um link enviado ao seu email.

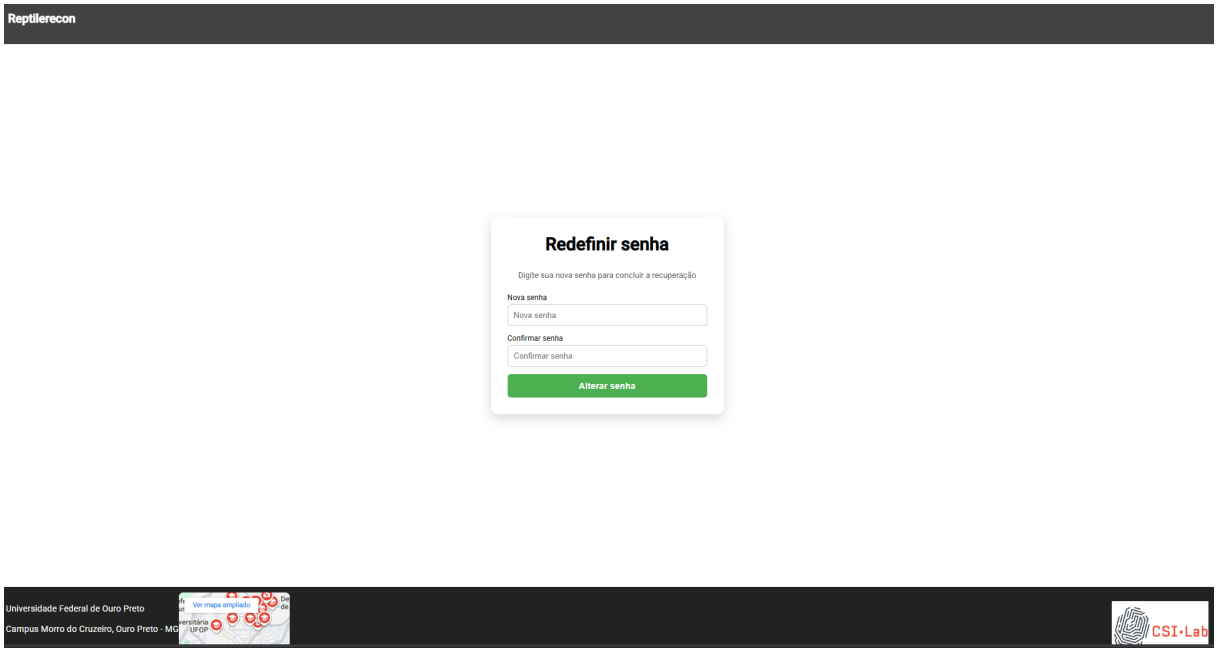

Figura 4.4 – Página de redefinição de senha — Front-end

Fonte: Elaborado pelo autor

4.1.5 Painel do Usuário

A Figura 4.5 apresenta o painel do usuário, acessado após a autenticação no sistema. Nele, o usuário pode visualizar seus dados pessoais, informações relevantes sobre os vídeos submetidos, além de acessar as funcionalidades de listagem e envio de novos vídeos.

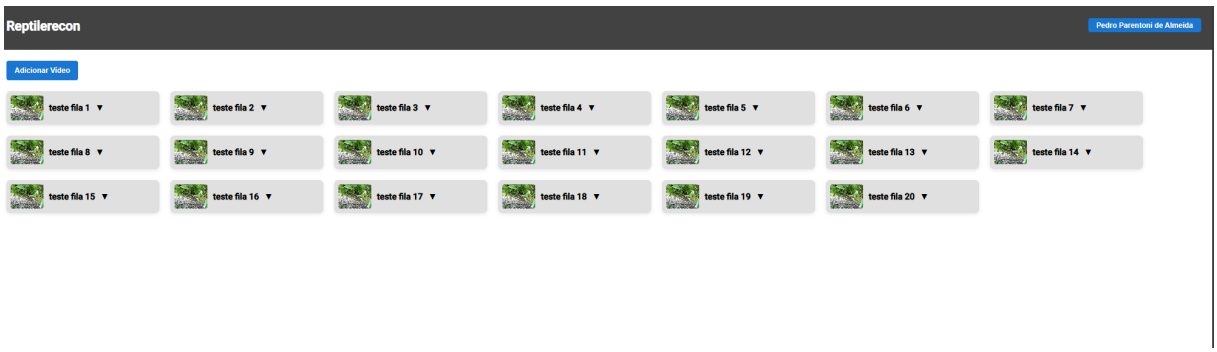

Figura 4.5 – Painel do usuário — Front-end

Fonte: Elaborado pelo autor

A Figura 4.6 apresenta a tela de envio de vídeos, também acessada a partir do painel do

usuário. Nessa interface, o usuário define o título e a descrição do vídeo a ser submetido para análise. Após o envio, o vídeo passa a integrar a lista exibida na tela de listagem.

Figura 4.6 – Tela de envio de vídeo — Front-end

Fonte: Elaborado pelo autor

A Figura 4.7 demonstra o modal de um vídeo processado, aberto ao clickar em uma caixa de vídeo no painel de usuário, onde o usuário tem acesso ao nome, ao vídeo processado e a descrição. Além disso nele se localizam os botões para excluir, abrir o grávico do cabeceio e também de baixar o sinais no formato CSV.

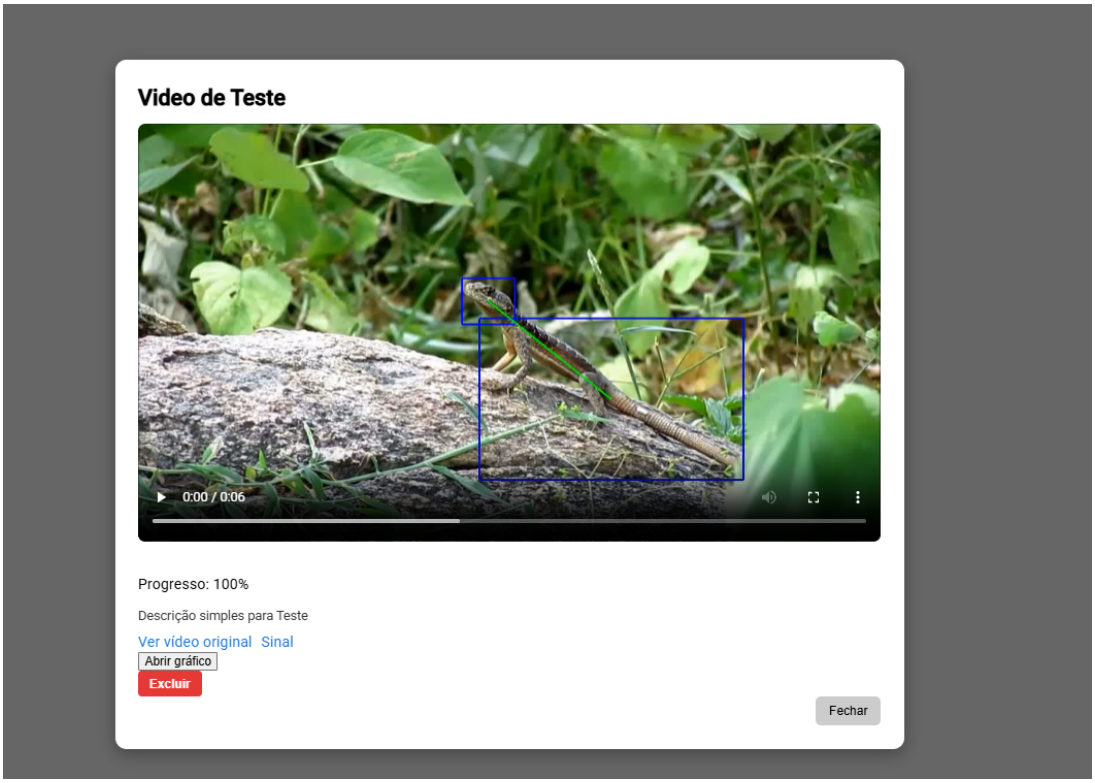

Figura 4.7 – Modal do acesso aos dados do vídeo processado

Fonte: Elaborado pelo autor

A Figura 4.8 demonstra o modal de acesso a dados do usuário, onde ele pode alterar seu nome no sistema ou sua senha.

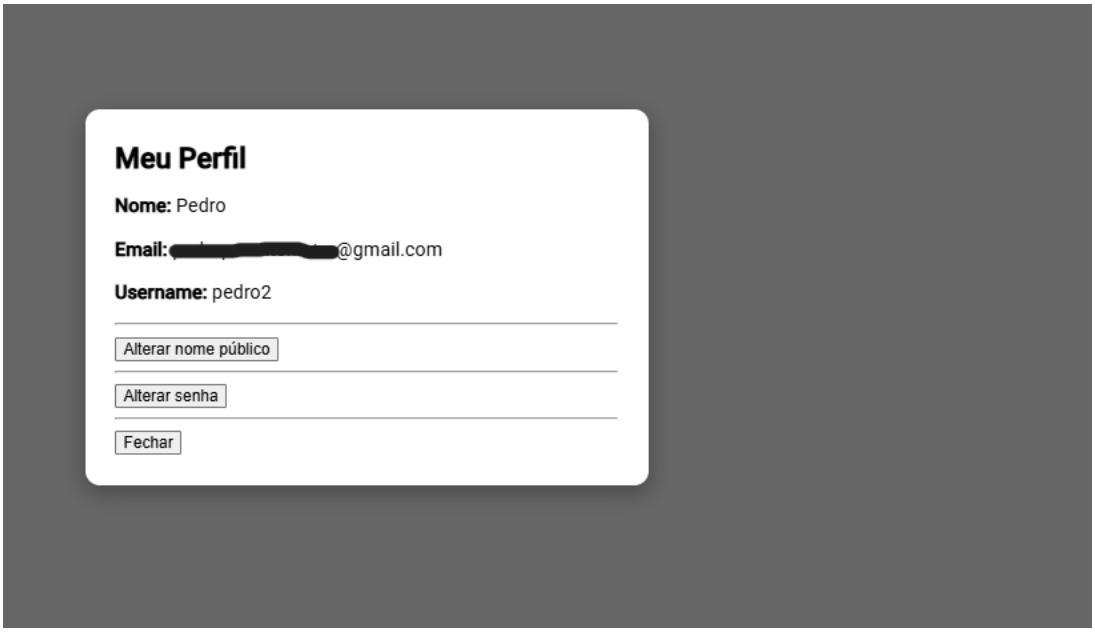

Figura 4.8 – Modal do acesso aos dados de usuário

Fonte: Elaborado pelo autor

4.2 Backend

Esta seção apresenta os resultados obtidos com o desenvolvimento da camada de backend do sistema, implementada com o framework Django. Essa camada é responsável pelo gerenciamento dos dados e pela disponibilização de serviços para a interface do usuário, garantindo integridade, segurança e consistência das informações processadas.

4.2.1 Arquitetura do Backend

O projeto segue a estrutura convencional do framework Django, composto pela pasta principal `mysite`, que concentra as configurações globais do sistema, e pela aplicação `reptilierUser`, na qual estão definidos os modelos e a lógica de negócio do sistema `Repitilerecon`. A Tabela 4.1 apresenta a estrutura de diretórios do projeto.

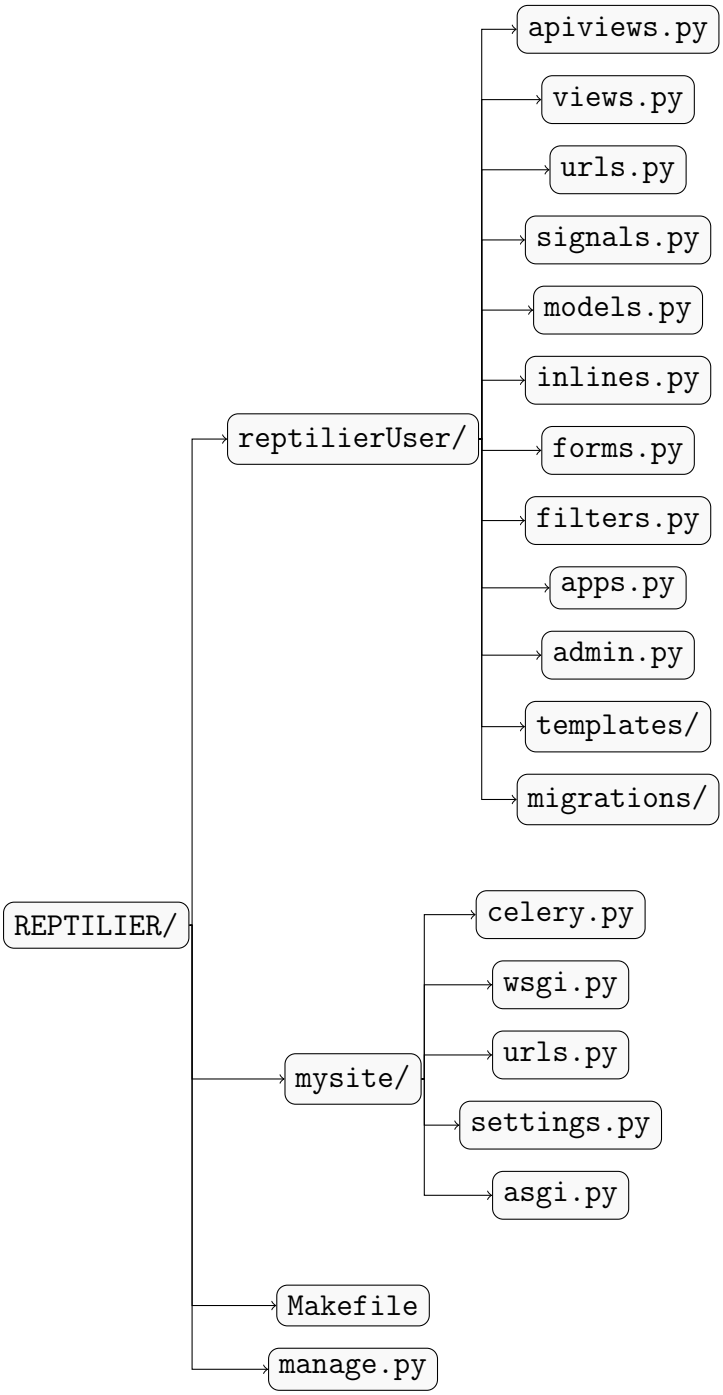

Tabela 4.1 – Estrutura de diretórios do projeto REPTILERECON

Fonte: Elaborado pelo autor

Ademais, na mesma pasta raiz de onde se encontra a pasta do projeto Django há o diretório que contem os arquivos responsáveis pelo frontend Angular que seguem a estrutura da Tabela 4.2

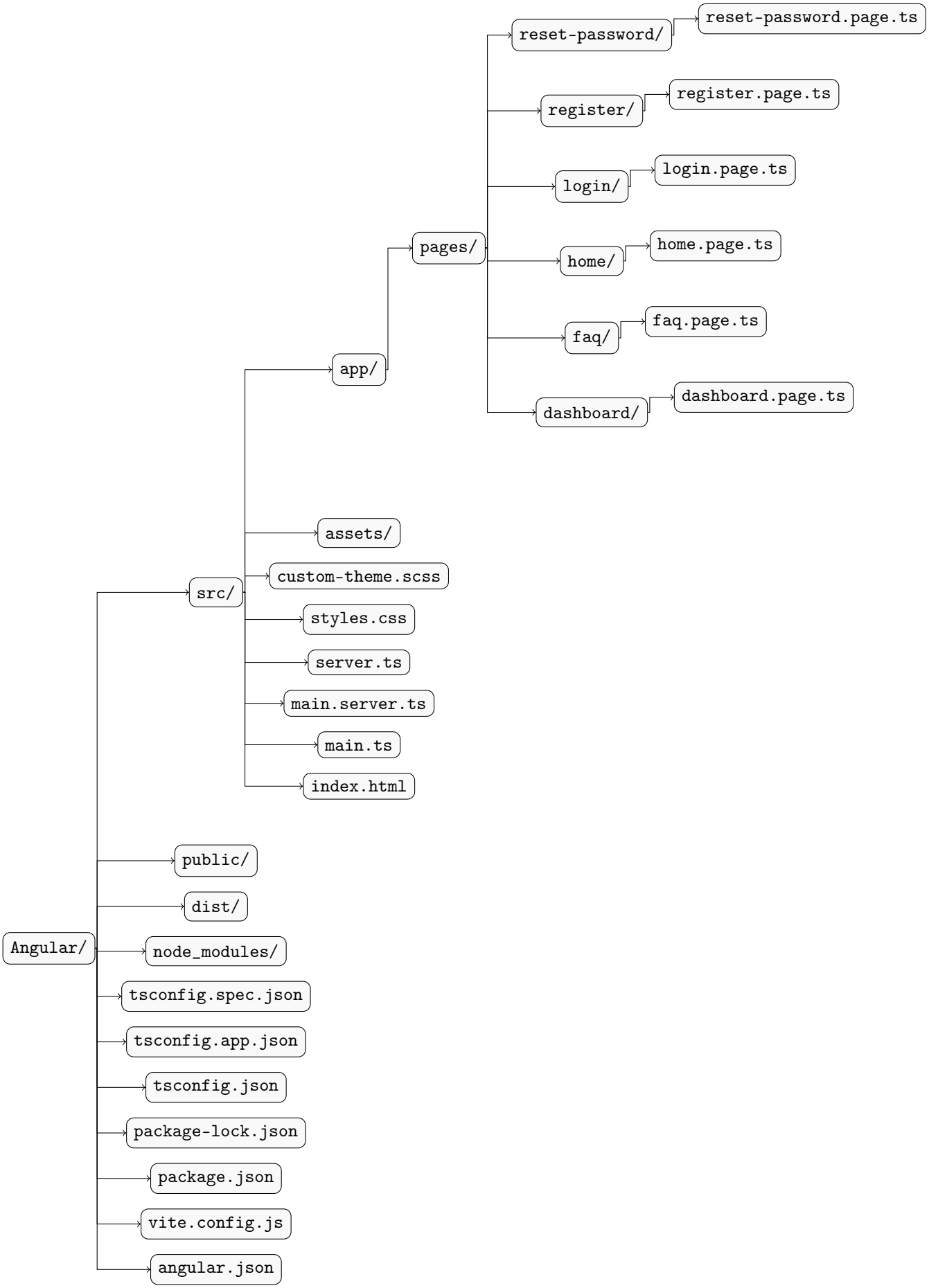

Tabela 4.2 – Estrutura de diretórios do frontend Angular

Fonte: Elaborado pelo autor

4.2.2 Sistema de Login de administrador

O sistema de login para administradores foi desenvolvido com o intuído de ser simples e direto visto que será acessado apenas por super usuários em situações ocasionais. Existem apenas duas formas de adquirir um login certificado para ele. Ou utilizando uma conta de adnistrador para registrar um super usuário na interface de admin, ou fazendo o registro de super usuário pelo terminal onde o servidor está rodando.

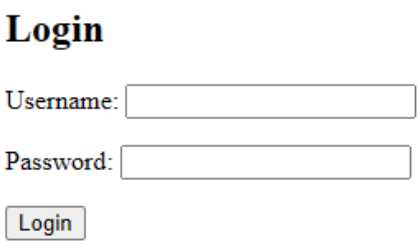

Figura 4.9 – Tela de login para administradores

Fonte: Elaborado pelo autor

4.2.3 Painel de Administrador

O painel de administração do Django oferece uma interface gráfica para gerenciamento direto dos dados do sistema. Por meio dessa ferramenta, o superusuário pode visualizar, criar, editar e excluir registros, além de gerenciar permissões e autenticações de usuários. A Figura 4.10 ilustra a tela principal do painel administrativo.

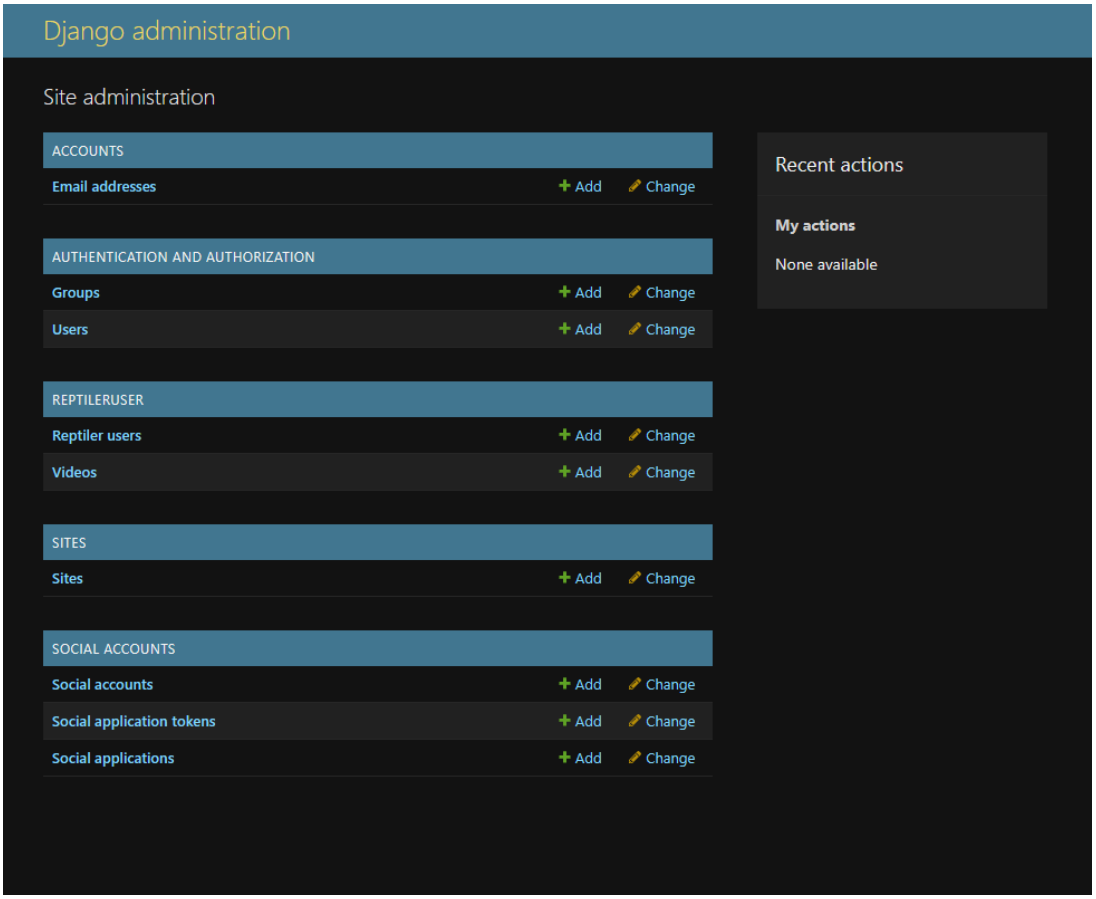

Figura 4.10 – Painel do Administrador - Django Back-end

Fonte: Elaborado pelo autor

4.2.3.1 Accounts

O módulo *Accounts* representa a seção do banco de dados destinada ao armazenamento dos endereços de e-mail registrados no sistema. Esse módulo é utilizado para gerenciar contas vinculadas a autenticações externas, facilitando a identificação dos usuários.

4.2.3.2 Authentication and Authorization

O módulo *Authentication and Authorization* gerencia usuários e grupos do sistema. Em *Groups* são armazenados agrupamentos com permissões específicas, enquanto *Users* reúne os usuários cadastrados no sistema. Neste projeto, foi implementado um filtro adicional para ocultar usuários criados automaticamente pelo sistema (*repitiler*) ou vinculados via OAuth do Google, conforme ilustrado na Figura 4.11.

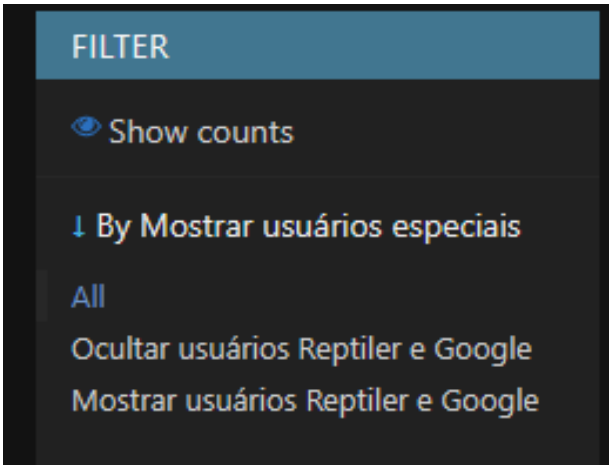

Figura 4.11 – Filtro de tipo de usuário em *Users* - Back-end Django

Fonte: Elaborado pelo autor

4.2.3.3 Sites

O módulo *Sites* é um recurso nativo do Django utilizado para gerenciar múltiplos domínios dentro de um mesmo projeto. No contexto do sistema `Repitilerecon`, esse módulo permanece com a configuração padrão, servindo como base para funcionalidades relacionadas à autenticação social e vinculação de domínios.

4.2.3.4 Social Accounts

O módulo *Social Accounts* é fornecido por bibliotecas adicionais do Django (como o `django-allauth`) e gerencia as contas vinculadas a provedores externos, como Google e Facebook. Neste sistema, ele foi utilizado para integrar o login via OAuth do Google, sem modificações além da configuração inicial.

4.2.3.5 Recent Actions

O Django fornece um histórico de ações de usuário que podem ser consultadas. O quadro na direita da imagem 4.10 demonstra isso. Lá ficam as ações tomadas pelo usuário para a consulta.

4.3 Testes do sistema

Os testes realizados no sistema `Reptilerecon` tiveram como objetivo validar o funcionamento de todas as funcionalidades implementadas, bem como avaliar a estabilidade e o desempenho da aplicação em um ambiente de execução real. Os testes foram conduzidos considerando tanto os aspectos funcionais quanto não funcionais do sistema.

4.3.1 Testes funcionais

Durante a fase de testes funcionais, todas as principais funcionalidades do sistema foram verificadas individualmente, constatando-se o correto funcionamento das seguintes operações:

- Autenticação de usuários por meio de login convencional;
- Registro de novos usuários;
- Recuperação de senha;
- Login administrativo;
- Autenticação via conta Google;
- Envio de vídeos para processamento;
- Exclusão de vídeos enviados;
- Alteração do nome de usuário por meio do menu de usuário;
- Alteração de senha do usuário autenticado (não relacionada ao fluxo de recuperação);
- Carregamento e exibição do dashboard com informações atualizadas.

Todos os fluxos citados foram testados com sucesso, não sendo identificadas falhas que comprometessem a utilização do sistema.

4.3.2 Processamento assíncrono e uso do Celery

Para evitar sobrecarga de memória e garantir a estabilidade do servidor de aplicação, foi utilizado o `Celery` como mecanismo de processamento assíncrono de vídeos. Essa abordagem impede que o servidor `Gunicorn` execute tarefas computacionalmente intensivas, o que poderia resultar em estouro de memória ou degradação do serviço.

Os vídeos enviados pelos usuários são organizados em uma fila de processamento gerenciada pelo `Celery`, sendo processados em lotes de dois vídeos simultaneamente. Essa estratégia foi adotada com o objetivo de limitar o consumo de memória do próprio serviço de filas, evitando que o `Celery` também exceda os recursos disponíveis no servidor.

4.3.3 Testes de processamento concorrente

Foram realizados testes de processamento envolvendo o envio de múltiplos vídeos ao sistema, com o objetivo de verificar o comportamento da aplicação diante de requisições simultâneas.

Em um dos cenários avaliados, foram enviados 20 vídeos consecutivos a partir de uma única conta de usuário. Em outro cenário, vídeos foram enviados simultaneamente a partir de diferentes contas. Em ambos os casos, todos os vídeos foram processados corretamente pelo servidor, sem falhas durante a execução.

Os testes foram conduzidos manualmente, acompanhando-se individualmente cada submissão. Para cada vídeo enviado, verificou-se sua completude, isto é, se todo o fluxo de processamento foi executado até a geração final dos resultados esperados.

Observou-se que o sistema respeitou a ordem de chegada das requisições no servidor, processando os vídeos sequencialmente conforme o recebimento, independentemente da conta de origem. Os resultados indicam que o sistema lida de maneira estável e organizada com múltiplas submissões simultâneas dentro do cenário testado.

4.3.4 Ambiente de execução e limitações de hardware

Os testes foram realizados em um servidor com hardware limitado, caracterizado por um processador Intel Core 2 Duo E7500 com dois núcleos, aproximadamente 4 GB de memória RAM e uso de memória swap. Devido a essas limitações, observou-se que o tempo de processamento de vídeos é relativamente elevado, o que configura uma restrição de desempenho imposta pelo hardware disponível, e não pelo software em si. Apesar disso, o sistema manteve-se funcional e estável durante todos os testes.

4.3.5 Arquitetura e estabilidade do sistema

A arquitetura do sistema é composta por um frontend desenvolvido em Angular, um backend em Django com API REST, processamento assíncrono via Celery e servidor web `Nginx`, responsável por gerenciar as requisições e servir os diferentes serviços.

Foram realizados testes de instabilidade simulando quedas inesperadas do servidor. Observou-se que, devido à configuração dos serviços com `Nginx` e `Gunicorn`, todos os componentes do sistema são reiniciados automaticamente assim que o servidor é ligado novamente. Dessa forma, em situações como queda de energia elétrica, o sistema retorna ao funcionamento normal sem necessidade de intervenção manual, salvo a necessidade de religar a máquina.

4.3.6 Considerações finais dos testes

Com base nos testes realizados, conclui-se que o sistema `Reptilerecon` atende aos requisitos funcionais propostos e apresenta estabilidade operacional, capaz de processar múltiplos vídeos de forma organizada mesmo sob limitações de hardware. O desempenho reduzido no processamento de vídeos é atribuído exclusivamente às restrições do ambiente de execução, não comprometendo a confiabilidade do sistema.

4.4 Exemplos de Uso

Esta seção apresenta um exemplo de utilização do sistema considerando sua finalidade principal: a obtenção e análise dos dados processados a partir de vídeos do comportamento de cabeceio de lagartos do gênero *Tropidurus*.

O fluxo descrito a seguir demonstra o percurso realizado pelo usuário desde o acesso à plataforma até a visualização dos sinais extraídos automaticamente pelo sistema, evidenciando como a ferramenta auxilia no tratamento e na interpretação dos dados comportamentais.

Autenticação do Usuário

O primeiro passo para utilização da plataforma consiste na autenticação do usuário. Conforme ilustrado na Figura 4.12, a interface de login permite o preenchimento dos campos de e-mail e senha para acesso à conta previamente cadastrada.

Além do método tradicional de autenticação, o sistema também disponibiliza a opção de login federado por meio de conta Google, simplificando o acesso e reduzindo a necessidade de gerenciamento adicional de credenciais.

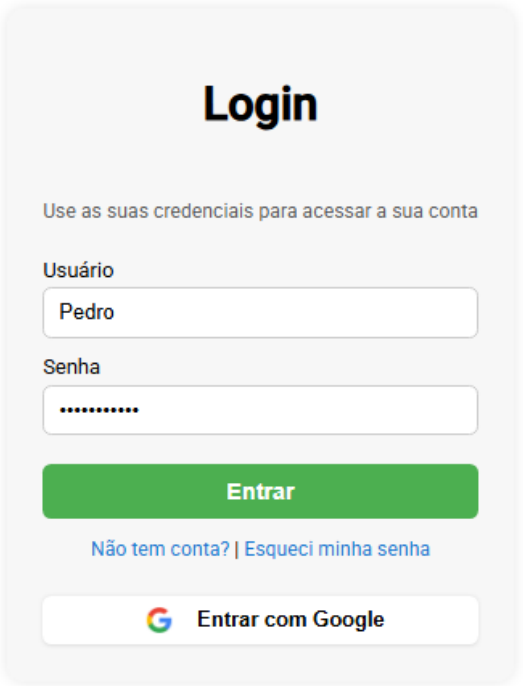

Figura 4.12 – Interface de autenticação do usuário

Fonte: Elaborado pelo autor

Acesso à Dashboard e Opção de Envio

Após a autenticação bem-sucedida, o usuário é direcionado à dashboard do sistema. Nessa interface, conforme apresentado na Figura 4.13, encontra-se no canto superior esquerdo o botão “Adicionar Vídeo”, responsável por iniciar o fluxo de submissão de um novo arquivo para análise.

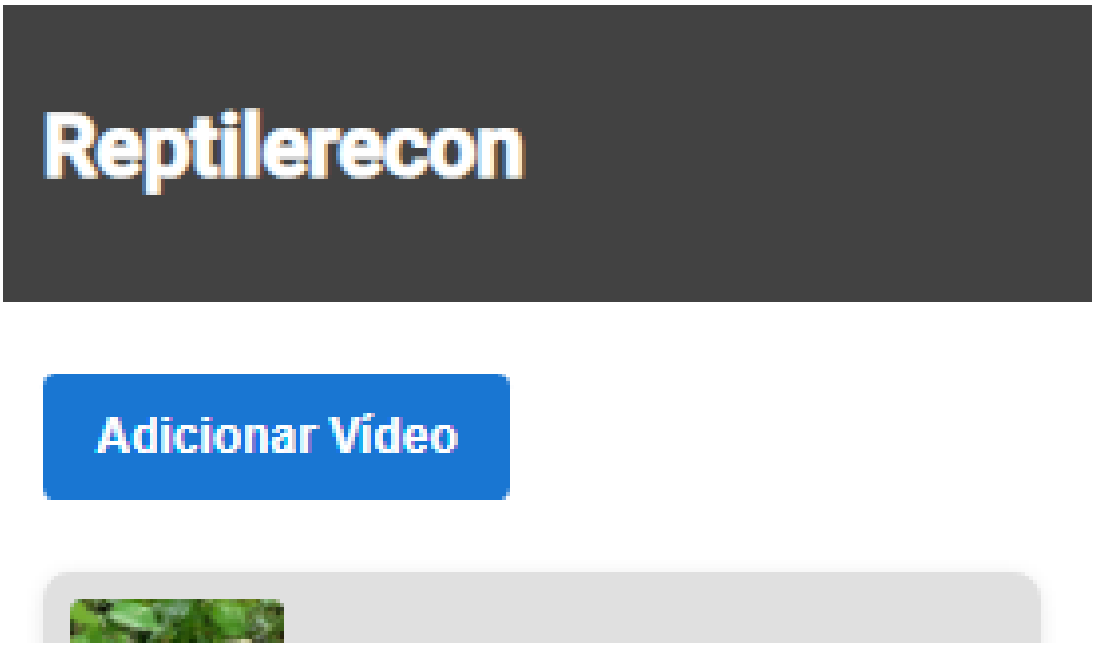

Figura 4.13 – Dashboard com botão de adicionar vídeo

Fonte: Elaborado pelo autor

Janela de Envio de Vídeo

Ao selecionar a opção “Adicionar Vídeo”, o sistema exibe a janela de submissão mostrada na Figura 4.14. Nessa interface, o usuário deve preencher as seguintes informações:

- Nome do vídeo (campo obrigatório para identificação);
- Seleção do arquivo de vídeo a ser enviado;
- Descrição opcional.

Após o preenchimento dos campos, o usuário deve clicar em “Enviar Vídeo”. O sistema realiza o upload do arquivo e inicia o processamento assíncrono. Durante esse período, o usuário aguarda alguns segundos até que a janela seja automaticamente fechada, indicando que o envio foi concluído com sucesso.

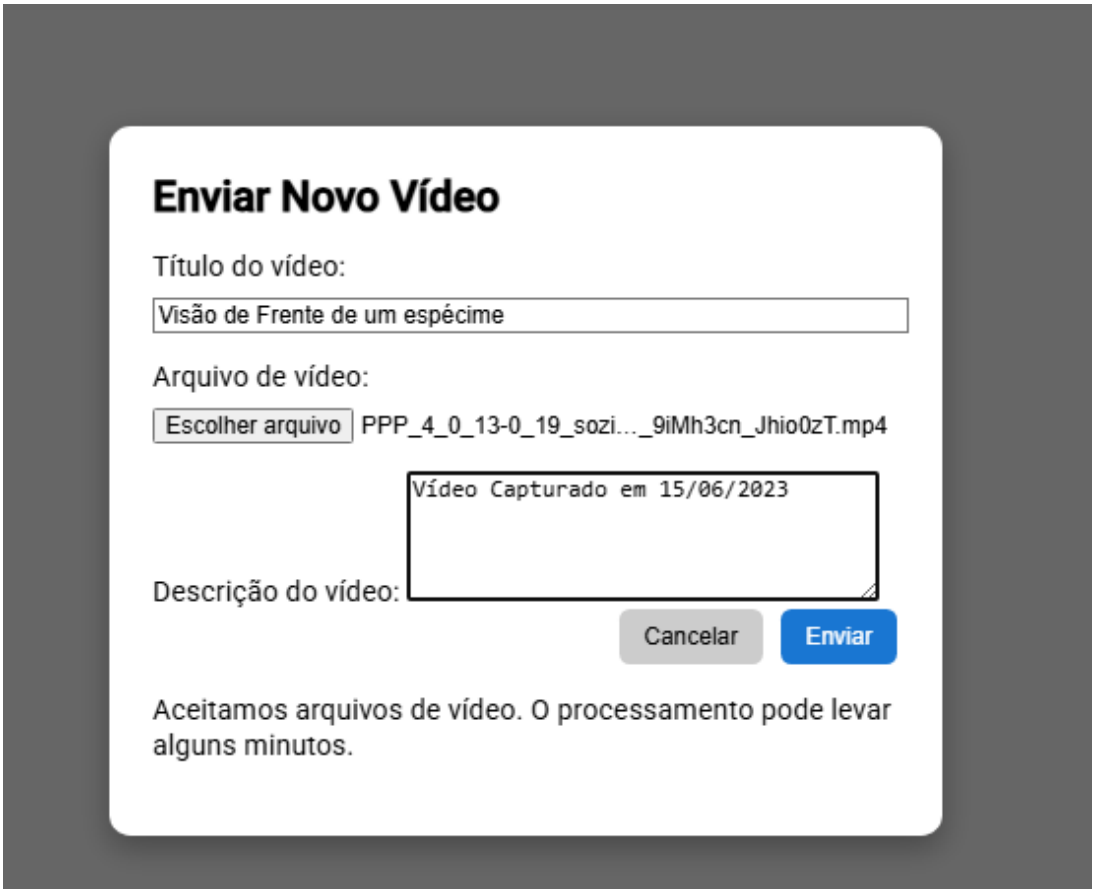

Figura 4.14 – Janela de envio de vídeo

Fonte: Elaborado pelo autor

Exibição do Vídeo na Dashboard

Concluído o envio, o vídeo passa a ser listado na dashboard do usuário, conforme ilustrado na Figura 4.15.

A partir dessa interface, o usuário pode selecionar um vídeo específico para acessar os detalhes do processamento.

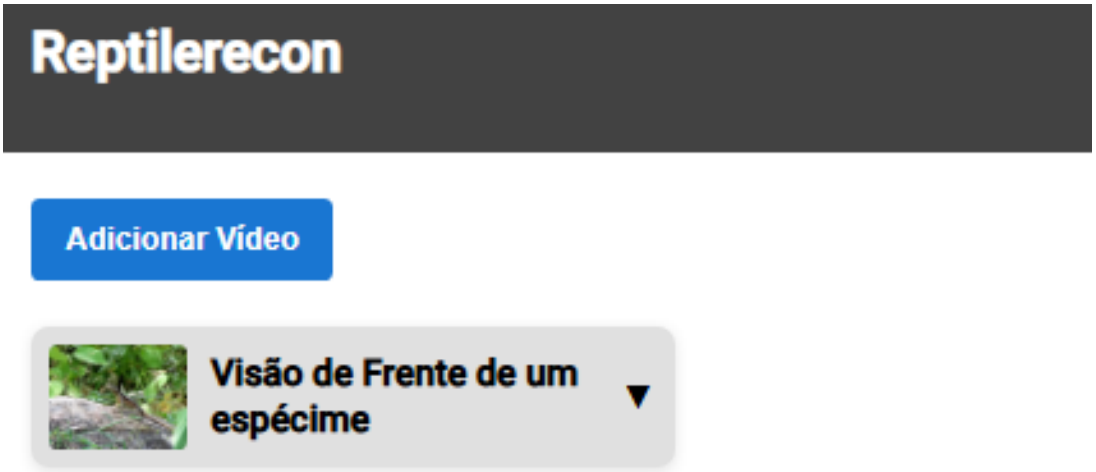

Figura 4.15 – Vídeo listado na dashboard do usuário

Fonte: Elaborado pelo autor

Visualização Detalhada do Vídeo Processado

Ao clicar na prévia do vídeo exibido na dashboard, o sistema abre a interface de visualização detalhada, apresentada na Figura 4.16. Nessa tela, são exibidas as informações relacionadas ao processamento do vídeo, bem como as opções de interação disponíveis ao usuário.

Entre as funcionalidades disponíveis nessa interface destacam-se:

- Exclusão do vídeo da base de dados;
- Acesso à visualização gráfica dos sinais extraídos.

Figura 4.16 – Tela de visualização detalhada do vídeo processado

Fonte: Elaborado pelo autor

Visualização do Gráfico de Sinais

Ao selecionar a opção “Abrir Gráfico” na tela de detalhes, o sistema apresenta a visualização gráfica dos sinais extraídos durante o processamento do vídeo, conforme ilustrado na Figura 4.17.

Essa interface permite ao usuário analisar os dados obtidos, possibilitando a interpretação visual dos padrões detectados pelo sistema.

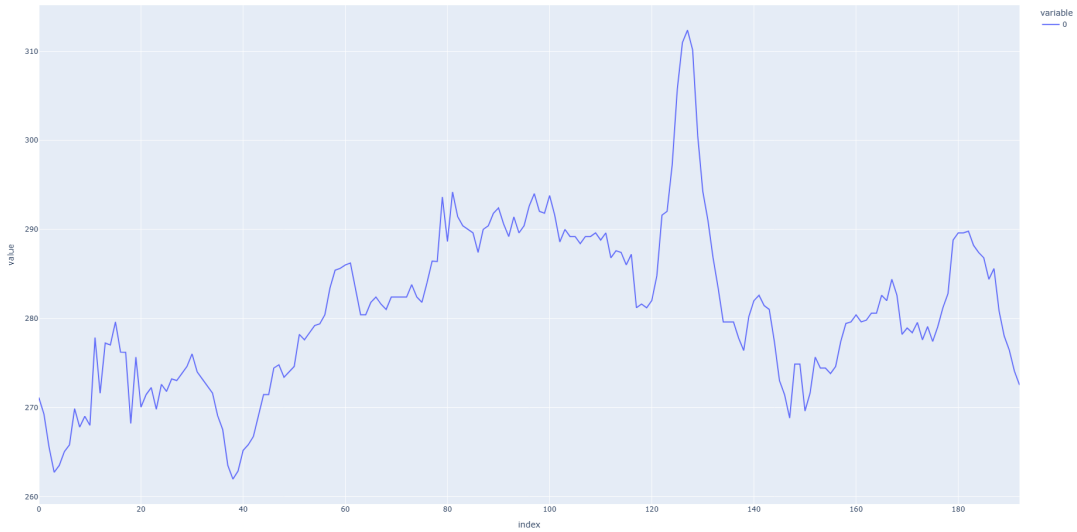

Figura 4.17 – Visualização gráfica dos sinais extraídos

Fonte: Elaborado pelo autor

5 Considerações Finais

Este trabalho apresentou o desenvolvimento do sistema *Reptilerecon*, concebido com o objetivo de disponibilizar, de forma acessível e confiável, um framework para análise de sinais visuais dos movimentos de cabeça em lagartos do gênero *Tropidurus*. O principal desafio enfrentado consistiu em adaptar e operacionalizar o framework proposto em Zenobio et al. (2023) em um sistema web funcional, robusto e adequado ao uso por pesquisadores da área visto que ele anteriormente era disponível apenas em um domínio privado exclusivamente para testes, sem apresentar questões como autentificação de usuário e consistência de serviços.

Para a superação desse desafio, foi adotada uma arquitetura baseada em tecnologias amplamente consolidadas. O backend foi desenvolvido utilizando o framework Django (Django Software Foundation,), enquanto o frontend foi implementado em Angular (JAIN; BHANSALI; MEHTA, 2014), com comunicação entre as camadas realizada por meio da Django REST Framework (Encode OSS Ltd., 2025). O processamento assíncrono dos vídeos foi conduzido com o auxílio do Celery (Celery Project Contributors, 2025), permitindo que tarefas computacionalmente intensivas fossem executadas de forma desacoplada do servidor de aplicação principal. Essa abordagem mostrou-se fundamental para garantir a estabilidade do sistema, evitando sobrecarga de memória no servidor *Gunicorn* (CHESNEAU, 2025).

O sistema implementa mecanismos completos de autenticação e gerenciamento de usuários, incluindo registro, login convencional, recuperação de senha, autenticação administrativa e login via Google OAuth (Google Developers, 2025). Além disso, foi realizada a extensão da classe padrão de usuários do Django para a criação do *RepitilerUser*, possibilitando maior flexibilidade no gerenciamento das informações. O banco de dados gerado pelo próprio Django foi utilizado, assegurando consistência e integridade dos dados armazenados.

Os testes realizados demonstraram que o *Reptilerecon* é capaz de processar múltiplos vídeos de forma estável e organizada, mesmo quando submetido a cenários de carga envolvendo diferentes usuários e grandes quantidades de vídeos. O uso de filas no Celery garantiu que os vídeos fossem processados respeitando a ordem de chegada, com controle do número de tarefas simultâneas, contribuindo para a previsibilidade do comportamento do sistema.

Ressalta-se que os testes foram conduzidos em um ambiente com limitações significativas de hardware, composto por uma máquina antiga, com recursos computacionais restritos. Como consequência, o tempo de processamento dos vídeos mostrou-se elevado. Ainda assim, o sistema manteve-se funcional e estável.

Atualmente, o sistema encontra-se disponível para acesso público no endereço <https://reptilerecon.csilab.ufop.br>, possibilitando sua utilização por pesquisadores interessados na análise de sinais visuais em estudos comportamentais.

Como perspectivas para melhorias futuras, destaca-se a possibilidade de adoção de uma arquitetura de processamento distribuído. Considerando que o serviço de processamento de vídeos via Celery é desacoplado do backend Django, torna-se viável a execução dessas tarefas em servidores externos com maior capacidade computacional. Nesse cenário, o sistema principal seria responsável apenas pelo envio dos vídeos para processamento e pelo recebimento dos resultados finais, reduzindo significativamente as limitações impostas pelo hardware atual e ampliando a escalabilidade da plataforma.

Referências

Celery Project Contributors. *Celery: Distributed Task Queue (Version 5.6.2)*. 2025. Disponível em: <https://docs.celeryq.dev/>.

CHESNEAU, B. *Gunicorn: Python WSGI HTTP Server for UNIX*. 2025. Disponível em: <https://gunicorn.org/>.

Django Software Foundation. *Django*. Disponível em: <https://djangoproject.com>.

Encode OSS Ltd. *Django REST Framework: Web APIs for Django*. 2025. Disponível em: <https://www.django-rest-framework.org/>.

FURNAS, B. J.; NEWTON, D. S.; CAPEHART, G. D.; BARROWS, C. W. Hierarchical distance sampling to estimate population sizes of common lizards across a desert ecoregion. *Ecology and Evolution*, v. 9, n. 6, p. 3046–3058, 2 2019. Disponível em: <https://doi.org/10.1002/ece3.4780>.

Google Developers. *Using OAuth 2.0 to Access Google APIs*. 2025. <https://developers.google.com/identity/protocols/oauth2?hl=pt-br>. Acesso em: 28 jul. 2025.

HARDT, D. *The OAuth 2.0 Authorization Framework*. [S.l.], 2012. Disponível em: <https://doi.org/10.17487/rfc6749>.

JAIN, N.; BHANSALI, A.; MEHTA, D. Angularjs: A modern mvc framework in javascript. *Journal of Global Research in Computer Science*, v. 5, n. 12, p. 17–23, 2014.

MÖNCK, H. J.; JÖRG, A.; FALKENHAUSEN, T. von; TANKE, J.; WILD, B.; DORMAGEN, D.; PIOTROWSKI, J.; WINKLMAYR, C.; BIERBACH, D.; LANDGRAF, T. *BioTracker: An Open-Source Computer Vision Framework for Visual Animal Tracking*. 2018. Disponível em: <https://arxiv.org/abs/1803.07985>.

NGINX, Inc. *nginx: High Performance Web Server and Reverse Proxy*. 2025. Disponível em: <https://nginx.org/>.

PIANTONI, C. *Efeitos do aquecimento global em populações do complexo de espécies Tropidurus torquatus (Squamata: Tropiduridae) no Brasil*. Tese (Doutorado) — Instituto de Biociências, University of São Paulo, São Paulo, 2015. Tese (Doutorado em Fisiologia Geral). Disponível em: <https://www.teses.usp.br/teses/disponiveis/41/41135/tde-08032016-160411/pt-br.php>.

REDMON, J.; DIVVALA, S.; GIRSHICK, R.; FARHADI, A. You only look once: Unified, real-time object detection. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. [S.l.: s.n.], 2016. p. 779–788.

RIBEIRO-JÚNIOR, M. A.; GARDNER, T. A.; ÁVILA-PIRES, T. C. S. THE EFFECTIVENESS OF GLUE TRAPS TO SAMPLE LIZARDS IN A TROPICAL RAINFOREST. *South American Journal of Herpetology*, Brazilian Society of Herpetology, v. 1, n. 2, p. 131 – 137, 2006. Disponível em: <https://doi.org/10.2994/1808-9798(2006)1[131:TEOGTT]2.0.CO;2>.

ROMERO-FERRERO, F.; BERGOMI, M. G.; HINZ, R.; HERAS, F. J. H.; POLAVIEJA, G. G. de. *idtracker.ai: Tracking all individuals in large collectives of unmarked animals*. 2018. Disponível em: <https://arxiv.org/abs/1803.04351>.

SILVA, A. O. *Framework para extração de sinais na comunicação visual de lagartos*. Ouro Preto, 2018. Monografia apresentada ao Curso de Bacharelado em Ciência da Computação como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

WERNECK, F. P.; LEITE, R. N.; GEURGAS, S. R.; RODRIGUES, M. T. Biogeographic history and cryptic diversity of saxicolous tropiduridae lizards endemic to the semiarid caatinga. *BMC Evolutionary Biology*, v. 15, n. 1, p. 94, 2015. Disponível em: <https://doi.org/10.1186/s12862-015-0368-3>.

ZENOBIO, J. G. F.; SILVA, P.; LUZ, E.; MOREIRA, G.; GALDINO, C.; GERTRUDES, J. C. Reptilerecon: Um arcabouço para extração e análise de sinais de lagartos. In: . [S.l.: s.n.], 2023. p. 154–166.

ZUERL, M.; STOLL, P.; BREHM, I.; RAAB, R.; ZANCA, D.; KABRI, S.; HAPPOLD, J.; NILLE, H.; PRECHTEL, K.; WUENSCH, S.; KRAUSE, M.; SEEGERER, S.; FERSEN, L. von; ESKOFIER, B. Automated video-based analysis framework for behavior monitoring of individual animals in zoos using deep learning—a study on polar bears. *Animals*, MDPI, v. 12, n. 6, p. 692, 2022. Disponível em: <https://doi.org/10.3390/ani12060692>.