LUMEN ET VIRTUS
ISSN: 2177-2789

SISTEMAS INTELIGENTES E AUTOMAÇÃO: APLICAÇÃO WEB PARA GESTÃO AUTOMATIZADA DE SEGURANÇA COM INTELIGÊNCIA ARTIFICIAL

INTELLIGENT SYSTEMS AND AUTOMATION: WEB APPLICATION FOR AUTOMATED SECURITY MANAGEMENT WITH ARTIFICIAL INTELLIGENCE

SISTEMAS INTELIGENTES Y AUTOMATIZACIÓN: APLICACIÓN WEB PARA LA GESTIÓN AUTOMATIZADA DE LA SEGURIDAD CON INTELIGENCIA ARTIFICIAL

https://doi.org/10.56238/levv16n53-027

Data de submissão: 06/09/2025 **Data de publicação:** 06/10/2025

Luis Eduardo Rodrigues Royo
Tecnólogo em Análise e Desenvolvimento de Sistemas
Instituição: Instituto Federal de São Paulo (IFSP), Campinas
E-mail: luis.r@aluno.ifsp.edu.br

Glauber da Rocha Balthazar
Doutor em Engenharia de Sistemas Agrícolas
Instituição: Instituto Federal de São Paulo (IFSP), Campinas
E-mail: glauber.balthazar@ifsp.edu.br
Orcid: https://orcid.org/0000-0002-1993-6621
Lattes: http://lattes.cnpq.br/1724935313124948

RESUMO

A gestão de segurança em edifícios enfrenta desafios consideráveis na modernização de seus processos operacionais, especialmente na automação de tarefas administrativas e na análise de dados de incidentes, revelando uma lacuna no setor de segurança privada para sistemas inteligentes que processam informações de forma automatizada e geram insights estratégicos. Este trabalho propõe o desenvolvimento de uma aplicação web integrada com inteligência artificial (IA) para a gestão automatizada de segurança predial, utilizando a API Google Gemini para a classificação automática de ocorrências e a geração de relatórios inteligentes. A metodologia seguiu o framework Ágil Scrum, com desenvolvimento iterativo em sete sprints ao longo de sete meses, empregando tecnologias consolidadas como Flask (em Python) para o backend, PostgreSQL para o armazenamento de dados, Bootstrap para uma interface responsiva e integração com o Google Gemini para as funcionalidades de IA. Os resultados confirmam as soluções técnicas e econômicas da solução, com a implementação completa de módulos para gerenciamento de usuários, rondas, ocorrências, classificação automática via IA e geração de relatórios, alcançando cobertura de testes de 99% e validando desde a autenticação segura até o processamento inteligente de relatórios; essa combinação de tecnologias web modernas com IA demonstrou eficiência na automação de processos e na melhoria da qualidade da documentação. Concluindo, a integração de tecnologias web com IA é viável e economicamente acessível para o setor de segurança predial, oferecendo uma solução escalável e de código aberto que

contribui significativamente para a modernização da gestão de segurança em ambientes corporativos e residenciais.

Palavras-chave: Segurança Predial. Inteligência Artificial. Automação. Aplicação Web. Gestão de Ocorrências.

ABSTRACT

Building security management faces considerable challenges in modernizing its operational processes, especially in automating administrative tasks and analyzing incident data. This reveals a gap in the private security sector for intelligent systems that automatically process information and generate strategic insights. This work proposes the development of a web application integrated with artificial intelligence (AI) for automated building security management, using the Google Gemini API for automatic incident classification and intelligent reporting. The methodology followed the Agile Scrum framework, with iterative development in seven sprints over seven months, employing established technologies such as Flask (in Python) for the backend, PostgreSQL for data storage, Bootstrap for a responsive interface, and integration with Google Gemini for AI functionalities. The results confirm the solution's technical and economic capabilities, with the full implementation of modules for user management, patrols, incidents, automatic classification via AI, and reporting, achieving 99% test coverage and validating everything from secure authentication to intelligent report processing. This combination of modern web technologies with AI has proven effective in automating processes and improving documentation quality. In conclusion, the integration of web technologies with AI is viable and cost-effective for the building security sector, offering a scalable, open-source solution that significantly contributes to the modernization of security management in corporate and residential environments.

Keywords: Building Security. Artificial Intelligence. Automation. Web Application. Incident Management.

RESUMEN

La gestión de seguridad de edificios se enfrenta a importantes retos en la modernización de sus procesos operativos, especialmente en la automatización de tareas administrativas y el análisis de datos de incidentes. Esto revela una brecha en el sector de la seguridad privada para sistemas inteligentes que procesen automáticamente la información y generen perspectivas estratégicas. Este trabajo propone el desarrollo de una aplicación web integrada con inteligencia artificial (IA) para la gestión automatizada de la seguridad de edificios, utilizando la API de Google Gemini para la clasificación automática de incidentes y la generación inteligente de informes. La metodología siguió el marco Agile Scrum, con un desarrollo iterativo en siete sprints a lo largo de siete meses, empleando tecnologías consolidadas como Flask (en Python) para el backend, PostgreSQL para el almacenamiento de datos, Bootstrap para una interfaz responsiva e integración con Google Gemini para las funcionalidades de IA. Los resultados confirman las capacidades técnicas y económicas de la solución, con la implementación completa de módulos para la gestión de usuarios, patrullas, incidentes, clasificación automática mediante IA y generación de informes, logrando una cobertura de pruebas del 99 % y validando todos los aspectos, desde la autenticación segura hasta el procesamiento inteligente de informes. Esta combinación de tecnologías web modernas con IA ha demostrado su eficacia en la automatización de procesos y la mejora de la calidad de la documentación. En conclusión, la integración de tecnologías web con IA es viable y rentable para el sector de la seguridad de la edificación, ofreciendo una solución escalable y de código abierto que contribuye significativamente a la modernización de la gestión de la seguridad en entornos corporativos y residenciales.

Palabras clave: Seguridad de Edificios. Inteligencia Artificial. Automatización. Aplicación Web. Gestión de Incidentes.

1 INTRODUÇÃO

Atualmente, a tecnologia assume um papel crucial, visando solucionar desafios e atender a propósitos específicos, conforme sua natureza e aplicação, ou seja, a tecnologia desempenha um papel central na solução de desafios organizacionais por meio de inovações que aprimoram a segurança, a gestão empresarial e a comunicação (VăRZARU e BOCEAN, 2024). Desta forma, as inovações tecnológicas têm simplificado e aprimorado áreas como segurança, gestão de negócios e comunicação. Sem a incorporação de novas ideias e otimização de processos, as organizações tendem a manter métodos obsoletos sem dar o devido incentivo aos desenvolvedores e gestores a criar novas formas de resolver problemas operacionais para processos mais eficientes e seguros (KÖHLER, 2021).

A gestão da segurança empresarial e o controle de dados avançaram significativamente com a tecnologia. Hoje, gestores e supervisores usam sistemas digitais para ocorrências e rondas, cada vez mais integrados e acessíveis via web, dispositivos móveis e nuvem, facilitando o monitoramento em tempo real. O desenvolvimento de sistemas de gestão integrados comprova como a tecnologia otimizou o controle e a supervisão de processos de segurança em larga escala (KÖHLER, 2021). Além disso, a inteligência artificial (IA) e o machine learning transformaram o registro de ocorrências, o gerenciamento de colaboradores e o acompanhamento de rondas, tornando possíveis decisões mais precisas e a identificação de padrões de incidentes, com redução de riscos e custos operacionais (MONTEIRO et al., 2024; GUIMARÃES, 2022). Desta forma, na segurança patrimonial, a inovação se expande com possibilidades promissoras, impulsionada pela inteligência artificial (IA) e machine learning, aplicados no dia a dia das empresas.

O uso crescente de IA em sistemas preditivos mostra a aplicação prática de algoritmos que analisam grandes dados, identificando padrões e antecipando crimes, o que ajuda a redirecionar patrulhas e ações policiais preventivamente (MONTEIRO et al., 2024; GUIMARÃES, 2022). Além disso, a automação e a IA facilitam relatórios dinâmicos, integração de equipes e comunicação eficiente, incentivando a análise crítica e a tomada de decisões em tempo real (KÖHLER, 2021).

Para que os computadores realizem suas funções, como guardar informações sobre eventos, criar relatórios e auxiliar na administração, eles precisam de programas. Eles combinam componentes físicos, o hardware (servidores e aparelhos), com os programas, que são os softwares. A criação de softwares de gestão envolve o uso de linguagens de programação, nas quais códigos são escritos para definir o que um sistema fará objetivando automatizar processos administrativos, registro de eventos e geração de relatórios (PRESSMAN; MAXIM, 2021; LAUDON; LAUDON, 2021).

Segundo a Associação Brasileira das Empresas de Sistemas Eletrônicos de Segurança (ABESE), o setor de segurança eletrônica movimentou R$12 bilhões em 2023, demonstrando o aumento rápido no uso de sistemas digitais para proteção (ABESE, 2024). Além disso este setor apresentou um crescimento médio de 13,7 % com tendência de 18,5 % para o ano de 2024, ou seja, 54

% dos produtos de segurança patrimonial (softwares e hardwares) já incorporam recursos de IA, sinalizando uma transição tecnológica comparável à digitalização do analógico (ABESE, 2024).

Ao começar a digitalizar seus procedimentos de segurança, muitas empresas enfrentam obstáculos na união de sistemas, na criação de relatórios padrão e na aplicação de análises automáticas, como a organização de ocorrências e a produção de relatórios, levando a problemas de eficiência. Isso ocorre porque a integração de sistemas legados, a padronização de relatórios e a incorporação de análises automáticas esbarram em dificuldades técnicas e organizacionais, prejudicando a eficiência operacional (OMOL, 2023). Essa questão provoca diversos contratempos na administração da segurança empresarial.

Para entender melhor as necessidades do setor e identificar lacunas nos sistemas já existentes, foi feita uma análise comparativa entre algumas das principais soluções de gestão de segurança patrimonial disponíveis no mercado. O estudo considerou o Qualyteam, voltado para a gestão da qualidade, e o Trackforce Valiant, focado no gerenciamento da força de trabalho em segurança. A partir dessa comparação, buscou-se definir quais funcionalidades são indispensáveis no sistema proposto e quais inovações podem diferenciá-lo das soluções já oferecidas. O Quadro 1 apresenta de forma detalhada essa análise de funcionalidades.

Quadro 1 - Análise Comparativa de Funcionalidades entre Sistemas de Gestão

Funcionalidade	**Qualyteam Versao: 20.22**	**Trackforce Valiant Versão: junho 2025**	**Sistema Proposto**
Gestão de Rondas	-	X	X
Registro de Ocorrências/Incidentes	X	X	X
Controle de Colaboradores	X	X	X
Dashboard em Tempo Real	X	X	X
Relatórios Automatizados	X	X	X
Integração com IA Generativa	-	X	X
Análise de Relatórios por IA	-	-	D
Classificação Automática	-	-	D
API RESTful	-	X	X
Interface Responsiva	X	X	X
Gestão de Condomínios	-	-	D
Controle de Turnos/Escalas	-	X	X
Histórico de Atividades	X	X	X
Sistema de Permissões	X	X	X

Exportação de Dados	X	X	X
Aplicativo Móvel	-	X	X
Indicadores/KPIs	X	X	X
X = Funcionalidade presente; - = Funcionalidade ausente; D = Funcionalidade diferenciadora			

Fonte: autor do trabalho (2025)

A análise do Quadro 1 evidencia lacunas significativas no mercado atual. O Qualyteam, embora robusto na gestão da qualidade, não oferece funcionalidades específicas para segurança patrimonial, como gestão de rondas ou integração com tecnologias de localização. Por outro lado, o Trackforce Valiant, apesar de ser uma solução abrangente para grandes corporações do setor de segurança, apresenta complexidade excessiva e custos elevados para pequenos e médios condomínios, além de não incorporar tecnologias de inteligência artificial para análise automática de relatórios.

Entre os sistemas analisados, não foi identificado o uso de IA generativa para o processamento/elaboração de relatórios. Observou-se, contudo, que o software Traceforce Valiant integra recursos de IA não-generativa, como reconhecimento facial, por meio de parceria com a Scylla, voltados à detecção por IA, despacho de guardas e resolução de incidentes (não necessariamente para relatórios automatizados). Dessa forma, a oportunidade de inovação permanece especificamente na aplicação de IA generativa para automatizar e qualificar relatórios e indicadores, conciliando a simplicidade necessária a condomínios de pequeno e médio porte com tecnologias avançadas de IA.

Este estudo visa auxiliar empresas e condomínios na modernização da gestão de segurança patrimonial por meio da hipótese de que é possível a construção de uma ferramenta web unificada, que permite o gerenciamento automatizado de rondas, ocorrências e funcionários com o uso de inteligência artificial que consiga analisar dados para a elaboração de relatórios com a classificação de dados de forma automática e, assim, gerar feedback detalhado, auxiliando os gestores a otimizar seus processos de segurança e a tomar decisões estratégicas.

2 OBJETIVOS

Desenvolver uma plataforma de segurança preditiva que utiliza a API Gemini para analisar e classificar relatórios de ocorrência automaticamente de forma a gerar dashboards em tempo real que irão aprimorar a agilidade e a precisão na tomada de decisões estratégicas de segurança.

2.1 OBJETIVOS ESPECÍFICOS

- Desenvolver o front-end responsivo do sistema com interface adaptável para múltiplos dispositivos;
- Desenvolver o backend do sistema utilizando arquitetura de micro serviços com Flask e

implementação de APIs RESTful;

- Criar e configurar o banco de dados PostgreSQL com estrutura otimizada;
- Implementar o sistema de processamento inteligente utilizando IA;
- Desenvolver o módulo de gestão de ocorrências;
- Implementar o sistema de gestão de rondas ;
- Criar o dashboard analítico com indicadores de performance;
- Desenvolver o sistema de autenticação e autorização com controle de perfis de usuário e gestão de permissões por níveis de acesso; e
- Criar o módulo de relatórios automatizados.

3 METODOLOGIA

A concepção do sistema de gestão de segurança foi estruturada sob a orientação do framework ágil Scrum, que se baseou em ciclos de trabalho curtos e iterativos, denominados Sprints (SUTHERLAND; LUA, 2020; CAMARGO; RIBAS, 2019). A adoção desse método visou à entrega de funcionalidades priorizadas, o que permitiu flexibilidade e adaptabilidade às mudanças ao longo do desenvolvimento do projeto. O processo foi executado em etapas incrementais, orientadas pelas necessidades do projeto.

As etapas fundamentais de desenvolvimento incluíram:

- Product Backlog: A elaboração de uma lista abrangente de requisitos funcionais e não funcionais.
- Sprint Planning: O detalhamento das atividades a serem realizadas em cada Sprint.
- Sprint Goal: A definição de objetivos específicos para cada Sprint, com o intuito de direcionar a produção do software de forma incremental e contínua.
- Daily Scrum: A realização de reuniões periódicas para acompanhamento do progresso, alinhamento da equipe e verificação do cumprimento dos objetivos de cada Sprint.

Para a documentação do projeto, serão elaborados os seguintes diagramas:

- Diagrama de Casos de Uso: Com o objetivo de descrever as funcionalidades do software e as interações com os usuários.
- Diagrama de Componentes: Que representará a arquitetura do sistema e a sua integração com a API do Google Gemini.
- Diagrama de Tabelas e Relacionamentos (DTR): Para ilustrar a estrutura do banco de dados e as relações entre suas tabelas.

As tecnologias que serão empregadas no desenvolvimento do sistema incluem:

- Backend: Flask, um framework Python leve e eficiente, ideal para o desenvolvimento de APIs e sistemas web.

- Banco de Dados: PostgreSQL, um sistema de gerenciamento de banco de dados relacional robusto, que será utilizado para garantir a persistência e integridade dos dados.
- Frontend: HTML, CSS e JavaScript, com o framework Bootstrap, para criar uma interface de usuário responsiva e moderna.
- Integração com a Inteligência Artificial: A API do Google Gemini será empregada para implementar funcionalidades baseadas em aprendizado de máquina, incluindo análise de grandes quantidades de dados e automação de diversos processos.

Por fim, o sistema passará por um conjunto abrangente de testes para garantir que todas as suas capacidades atendam aos requisitos estabelecidos e que o mesmo apresente estabilidade e segurança necessárias para seu uso final. A estratégia de testagem será dividida em três grandes categorias descritas a seguir.

3.1 TESTES FUNCIONAIS

Os testes funcionais validarão a execução das funções principais do sistema com a entrada de diversos parâmetros conhecidos, analisando os resultados encontrados em face dos resultados esperados. Estes testes incluirão:

- Testes de autenticação e autorização com diferentes perfis de usuário (Admin, Supervisor, Agente)
- Validação do sistema de rondas com diferentes cenários de entrada e saída, incluindo múltiplos condomínios e turnos
- Testes de correção automática de relatórios brutos utilizando a API do Google Gemini com diferentes tipos de textos e formatações
- Verificação da integridade dos dados no banco PostgreSQL, incluindo relacionamentos e constraints
- Testes de geração de relatórios com diferentes períodos, filtros e formatos de exportação

3.2 AMBIENTE DE TESTES

Os testes serão executados em um ambiente de desenvolvimento isolado, utilizando dados sintéticos que simulam cenários reais de operação, garantindo que não haja impacto nos dados de produção durante a validação do sistema.

4 RESULTADOS

4.1 FRAMEWORK SCRUM: PRODUCT BACKLOG

O Product Backlog constitui um artefato dinâmico e prioritizado, que elenca as funcionalidades, capacidades e melhorias propostas para um sistema de gestão de segurança predial.

Este repositório agrega tanto requisitos funcionais quanto não funcionais, servindo como referência central para orientar as iterações do ciclo de desenvolvimento do projeto. Os requisitos funcionais foram definidos como o conjunto de capacidades, funcionalidades e atributos comportamentais esperados do software. Consequentemente, os Quadros de 2 a 10 catalogam os requisitos funcionais do sistema, discriminando para cada um seu identificador único, nomenclatura, data de criação, data da última modificação, nível de prioridade, versão e descrição completa.

Quadro 2 - Sistema de Autenticação

Identificador	RF001
Nome	Sistema de Autenticação
Data de criação	01/02/2025
Data da última alteração	N/A
Prioridade	Alta
Versão	1.0
Descrição	O sistema deve implementar autenticação segura com JWT e controle de acesso baseado em papéis (Admin, Supervisor, Agente).

Fonte: Elaborado pelo autor (2025)

Quadro 3 - Gestão de Rondas

Identificador	RF002
Nome	Gestão de Rondas
Data de criação	15/05/2025
Data da última alteração	N/A
Prioridade	Alta
Identificador	RF002
Versão	1.0
Descrição	Sistema para cadastro, programação e execução de rondas com controle de plantões, múltiplos condomínios e geração de relatórios automatizados.

Fonte: Elaborado pelo autor (2025)

Quadro 4 - Gestão de Ocorrências

Identificador	RF003
Nome	Gestão de Ocorrências
Data de criação	15/06/2025
Data da última alteração	N/A
Prioridade	Alta

Versão	1.0
Descrição	Registro e classificação automática de ocorrências com workflow de atendimento e integração com órgãos públicos.

Fonte: Elaborado pelo autor (2025)

Quadro 5 - Inteligência Artificial

Identificador	RF004
Nome	Inteligência Artificial
Data de criação	15/07/2025
Data da última alteração	15/08/2025
Prioridade	Média
Versão	1.0
Descrição	Integração com Google Gemini para classificação automática de ocorrências e geração de insights de segurança.

Fonte: Elaborado pelo autor (2025)

Quadro 6 - Banco de Dados

Identificador	RF005
Nome	Banco de Dados
Data de criação	01/02/2025
Data da última alteração	N/A
Prioridade	Alta
Versão	1.0
Descrição	Implementar banco de dados PostgreSQL para armazenar dados de usuários, rondas, ocorrências e colaboradores com relacionamentos e integridade referencial.

Fonte: Elaborado pelo autor (2025)

Quadro 7 - Gestão de Colaboradores

Identificador	RF006
Nome	Gestão de Colaboradores
Data de criação	15/03/2025
Data da última alteração	N/A
Prioridade	Alta
Versão	1.0
Descrição	Cadastro e gestão de colaboradores, escalas mensais e controle de plantões diurnos/noturnos.

Fonte: Elaborado pelo autor (2025)

Quadro 8 - Gestão de Condomínios

Identificador	RF007
Nome	Gestão de Condomínios
Data de criação	15/04/2025
Data da última alteração	N/A
Prioridade	Alta
Identificador	RF007
Versão	1.0
Descrição	Cadastro e gestão de condomínios atendidos, incluindo endereços, contatos e informações de acesso.

Fonte: Elaborado pelo autor (2025)

Quadro 9 - Sistema de Notificações

Identificador	RF008
Nome	Sistema de Notificações
Data de criação	15/07/2025
Data da última alteração	N/A
Prioridade	Média
Versão	1.0
Descrição	Sistema de notificações automáticas para supervisores sobre ocorrências, rondas pendentes e alertas de segurança.

Fonte: Elaborado pelo autor (2025)

Quadro 10 - Relatórios e Analytics

Identificador	RF009
Nome	Relatórios e Analytics
Data de criação	15/08/2025
Data da última alteração	N/A
Prioridade	Média
Versão	1.0
Descrição	Geração de relatórios detalhados de rondas, ocorrências e análises de padrões de segurança.

Fonte: Elaborado pelo autor (2025)

Os requisitos não funcionais estabelecem os critérios de qualidade e as restrições operacionais que regem o comportamento do sistema, definindo atributos como desempenho, segurança e usabilidade. Desse modo, os Quadros 11 a 14 apresentam a especificação dos requisitos não funcionais identificados, detalhando para cada um: identificador único, nomenclatura, data de criação, data da última atualização, nível de prioridade, versão e descrição.

Quadro 11 - Segurança de Dados

Identificador	RNF001
Nome	Segurança de Dados
Data de criação	01/02/2025
Data da última alteração	N/A
Prioridade	Alta
Versão	1.0
Descrição	Garantir criptografia de dados sensíveis, logs de auditoria completos e proteção contra ataques CSRF e XSS.

Fonte: Elaborado pelo autor (2025)

Quadro 12 - Performance

Identificador	RNF002
Nome	Performance
Data de criação	01/02/2025
Data da última alteração	N/A
Identificador	RNF002
Prioridade	Alta
Versão	1.0

Descrição	Tempo de resposta < 2 segundos para operações críticas e suporte a 100 usuários simultâneos.

Fonte: Elaborado pelo autor (2025)

Quadro 13 - Usabilidade

Identificador	RNF003
Nome	Usabilidade
Data de criação	01/02/2025
Data da última alteração	N/A
Prioridade	Média
Versão	1.0
Descrição	Interface responsiva para dispositivos móveis e tempo de aprendizado < 30 minutos.

Fonte: Elaborado pelo autor (2025)

Quadro 14 - Escalabilidade

Identificador	RNF004
Nome	Escalabilidade
Data de criação	01/02/2025
Data da última alteração	N/A
Prioridade	Média
Versão	1.0
Descrição	Disponibilidade de 99.5% e cache inteligente para consultas frequentes.

Fonte: Elaborado pelo autor (2025)

4.2 FRAMEWORK SCRUM: SPRINT PLANNING

O planejamento do desenvolvimento do projeto está estruturado em sprints, ciclos iterativos e incrementais nos quais um conjunto de requisitos, funcionais e não funcionais, é selecionado para implementação e entrega. O cronograma estabelecido abrange um período de sete meses (março a setembro), organizado em iterações de quatro semanas cada, adaptando o framework Scrum ao escopo e às restrições de um ambiente acadêmico. Esta abordagem assegura um período dedicado suficiente para o tratamento aprofundado de funcionalidades complexas, tais como a integração de componentes de Inteligência Artificial e o desenvolvimento de interfaces responsivas, visando garantir a qualidade e a robustez do produto final. Ressalta-se que a execução das atividades pode ocorrer de forma paralela, desde que não existam dependências críticas, estratégia que otimiza o tempo de desenvolvimento e facilita a incorporação de feedbacks contínuos ao longo do processo. A sgeuir são detalhados o

mapeamento dos requisitos por sprint e as estimativas de esforço associadas a cada um e é apresentada no quadro 15 a visualização geral do Sprint Planning:

- Sprint 1 (Março) - Fundação e Autenticação (4 semanas) Estruturação completa do banco de dados (modelagem, migrações e relacionamentos), implementação do sistema de autenticação JWT com gestão de usuários e papéis (CRUD), e desenvolvimento das interfaces de login e gestão. Inclui testes e estabilização contínuos.
- Sprint 2 (Abril) - Gestão de Colaboradores e Condomínios (4 semanas) Desenvolvimento dos módulos de gestão de colaboradores e condomínios (CRUDs e modelos), implementação do sistema de escalas mensais com lógica de alocação, e integração entre os módulos. Foco em testes de integração para garantir a consistência dos dados.
- Sprint 3 (Maio) - Gestão de Rondas (4 semanas) Implementação do sistema de gestão de rondas, incluindo a programação, execução com checkpoints via geolocalização, e o desenvolvimento da interface de acompanhamento. Geração de relatórios básicos, com refinamentos e documentação contínua ao longo do sprint.
- Sprint 4 (Junho) - Gestão de Ocorrências (4 semanas) Desenvolvimento do módulo de ocorrências (CRUD e modelo) e implementação do workflow de atendimento de ponta a ponta, incluindo o sistema de notificações integrado. Foco em testes de integração do workflow e documentação dos processos.
- Sprint 5 (Julho) - Inteligência Artificial (4 semanas) Integração com a API do Google Gemini para classificação automática de ocorrências, incluindo o tratamento e refinamento dos resultados. Desenvolvimento da funcionalidade de geração de insights e análises preditivas. Ciclos de testes e otimização de performance da IA.
- Sprint 6 (Agosto) - Relatórios e Analytics (4 semanas) Construção do sistema de relatórios avançados e dashboards analíticos. Preparação do ambiente de produção e execução do processo de deploy. Realização dos testes finais de sistema (homologação), criação da documentação técnica e aplicação de ajustes.
- Sprint 7 (Setembro) - Finalização e Entrega (4 semanas) Execução dos testes de aceitação do usuário (UAT), aplicação das correções finais, e consolidação de toda a documentação do projeto. Preparação do material para a apresentação, defesa do TCC, e implementação de ajustes pós-banca. Encerramento formal e entrega da versão final do projeto.

Quadro 15 - Sprint Planning

Sprint	Mês	Semanas	Foco Principal	Duração
Sprint 1	Março	1-4	Fundação e Autenticação	4 semanas
Sprint 2	Abril	5-8	Gestão de Colaboradores e Condomínios	4 semanas
Sprint 3	Maio	9-12	Gestão de Rondas	4 semanas
Sprint 4	Junho	13-16	Gestão de Ocorrências	4 semanas
Sprint 5	Julho	17-20	Inteligência Artificial	4 semanas
Sprint 6	Agosto	21-24	Relatórios e Analytics	4 semanas
Sprint 7	Setembro	25-28	Finalização e Entrega	4 semanas

Fonte: Elaborado pelo autor (2025)

4.3 FRAMEWORK SCRUM: SPRINT GOAL

O Sprint Goal constitui uma declaração concisa que define o propósito macro de cada iteração, fornecendo direcionamento estratégico para as atividades de desenvolvimento a serem executadas durante o ciclo. Esta meta tem como função primordial alinhar os esforços da equipe e estabelecer um critério objetivo para a avaliação do êxito da sprint. Tais objetivos são entidades dinâmicas, passíveis de reavaliação e refinamento ao longo do projeto com base em feedbacks consolidados e nos resultados observados em iterações precedentes. A seguir são discriminados o planejamento sequencial das sprints e seus respectivos Sprint Goals:

- Sprint 1 (Março) - Fundação e Autenticação Estruturação completa do banco de dados (modelagem, migrações e relacionamentos), implementação do sistema de autenticação JWT com gestão de usuários e papéis (CRUD), e desenvolvimento das interfaces de login e gestão. Inclui testes e estabilização contínuos.
- Sprint 2 (Abril) - Gestão de Colaboradores e Condomínios Desenvolvimento dos módulos de gestão de colaboradores e condomínios (CRUDs e modelos), implementação do sistema de escalas mensais com lógica de alocação, e integração entre os módulos. Foco em testes de integração para garantir a consistência dos dados.
- Sprint 3 (Maio) - Gestão de Rondas Implementação do sistema de gestão de rondas, incluindo a programação, execução com checkpoints via geolocalização, e o desenvolvimento da interface de acompanhamento. Geração de relatórios básicos, com refinamentos e documentação contínua ao longo do sprint.
- Sprint 4 (Junho) - Gestão de Ocorrências Desenvolvimento do módulo de ocorrências (CRUD e modelo) e implementação do workflow de atendimento de ponta a ponta, incluindo o sistema de notificações integrado. Foco em testes de integração do workflow e documentação dos processos.

- Sprint 5 (Julho) - Inteligência Artificial Integração com a API do Google Gemini para classificação automática de ocorrências, incluindo o tratamento e refinamento dos resultados. Desenvolvimento da funcionalidade de geração de insights e análises preditivas. Ciclos de testes e otimização de performance da IA.
- Sprint 6 (Agosto) - Relatórios e Analytics Construção do sistema de relatórios avançados e dashboards analíticos. Preparação do ambiente de produção e execução do processo de deploy. Realização dos testes finais de sistema (homologação), criação da documentação técnica e aplicação de ajustes.
- Sprint 7 (Setembro) - Finalização e Entrega Execução dos testes de aceitação do usuário (UAT), aplicação das correções finais, e consolidação de toda a documentação do projeto. Preparação do material para a apresentação, defesa do TCC, e implementação de ajustes pós-banca. Encerramento formal e entrega da versão final do projeto.

4.4 FRAMEWORK SCRUM: DAILY SCRUM

As reuniões de Daily Scrum representam uma prática essencial para assegurar a comunicação eficiente entre o desenvolvedor e o orientador, viabilizando realinhamentos ágeis e a sincronização contínua acerca do andamento do projeto. No contexto desta pesquisa, os encontros de acompanhamento com o orientador exercem uma função análoga, propiciando feedback sistemático e direcionamento permanente. O Quadro 16, apresentada a seguir, consolida o planejamento das sprints (originalmente detalhado no Quadro 15) com o cronograma das reuniões de validação e homologação dos Sprint Goals com o orientador.

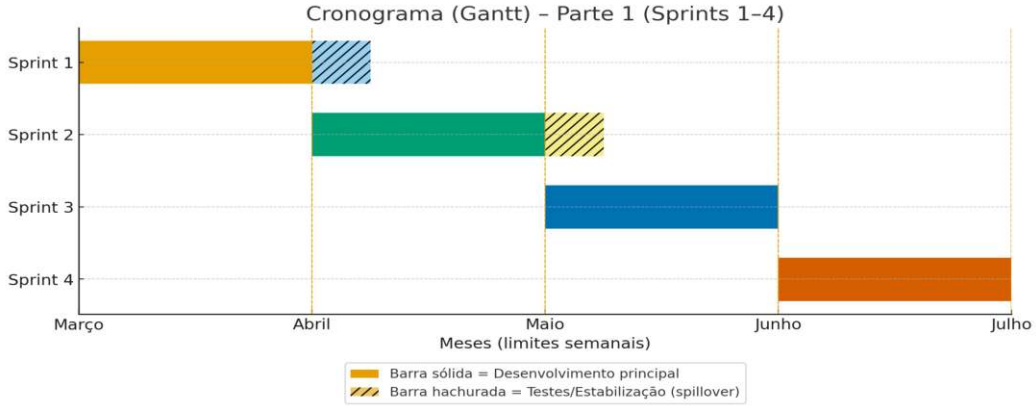
Quadro 16 - Cronograma Semanal e Reuniões com Atividades Paralelas

Sprint	Mês	Semanas	Foco Principal	Reunião de Validação
Sprint 1	Março	1-4	Fundação e Autenticação	28/03
Sprint 2	Abril	5-8	Gestão de Colaboradores e Condomínios	28/04
Sprint 3	Maio	9-12	Gestão de Rondas	28/05
Sprint 4	Junho	13-16	Gestão de Ocorrências	28/06
Sprint 5	Julho	17-20	Inteligência Artificial	28/07
Sprint 6	Agosto	21-24	Relatórios e Analytics	28/08
Sprint 7	Setembro	25-28	Finalização e Entrega	28/09

Fonte: Elaborado pelo autor (2025)

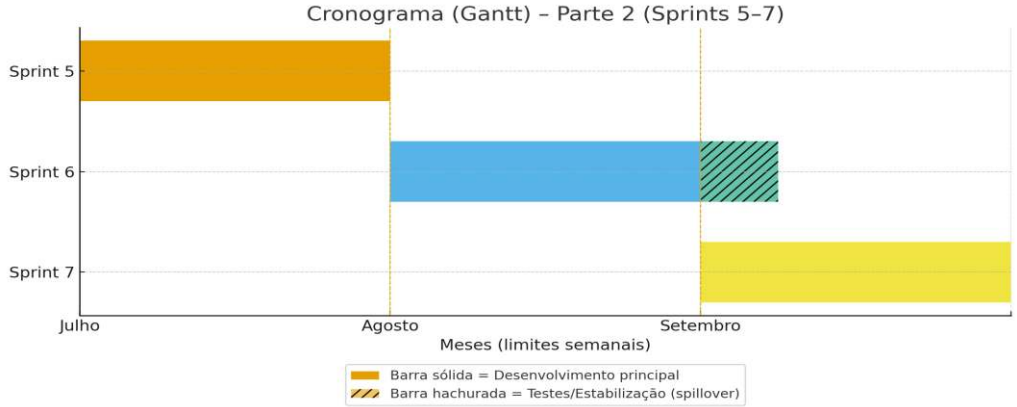
Para melhor visualização temporal do cronograma de desenvolvimento, foi elaborado um diagrama de Gantt que representa graficamente a distribuição dos sprints ao longo dos sete meses de projeto. Este diagrama permite visualizar a sequência e duração de cada sprint, facilitando o acompanhamento do progresso e identificação de marcos importantes do desenvolvimento, como mostram as figuras 1 e 2.

Figura 1- Diagrama de Gantt



Fonte: Elaborado pelo autor (2025)

Figura 2 - Diagrama de Gantt - Continuação



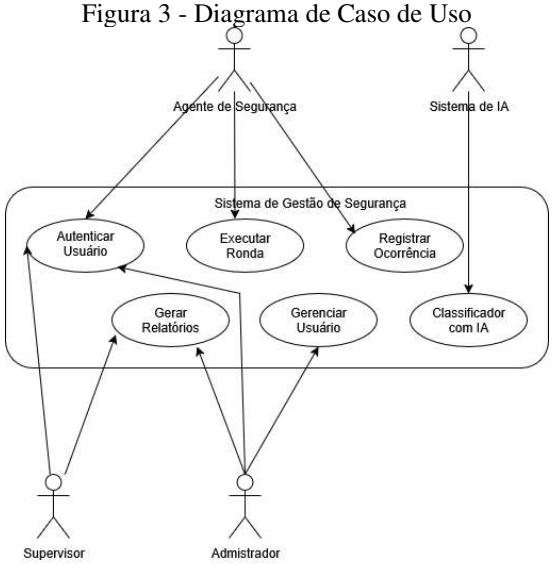
Fonte: Elaborado pelo autor (2025)

4.5 DIAGRAMAÇÕES: CASOS DE USO

Os casos de uso modelam as interações entre os atores (utilizadores ou sistemas externos) e o sistema de software, com o objetivo de alcançar uma meta específica. Esta técnica de modelagem auxilia na compreensão do comportamento esperado do sistema em diversos cenários, permitindo a identificação sistemática de funcionalidades, regras de negócio e os possíveis fluxos de execução. Para este projeto foram determinados quatro atores (Administrador, Supervisor, Agente de Segurança e Sistema de IA) e seis casos de uso. O diagrama apresentado na figura 3, apresenta a estrutura hierárquica dos atores e suas respectivas interações com o sistema. Cada ator possui diferentes níveis

de acesso e responsabilidades, desde operações básicas até funcionalidades administrativas avançadas, incluindo a integração com inteligência artificial para processamento automatizado.

Figura 3 - Diagrama de Caso de Uso



Fonte: Elaborado pelo autor (2025)

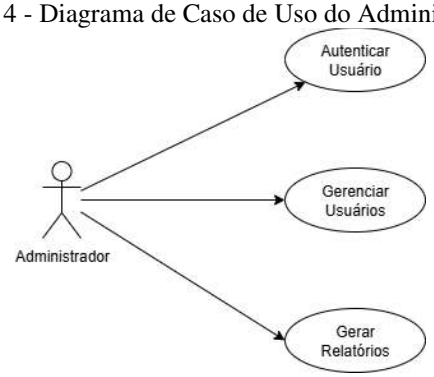
A seguir, cada um dos itens do diagrama de caso de uso é detalhado:

- Ator "Administrador": Interage com o sistema para gerenciamento completo, incluindo usuários, configurações e relatórios gerenciais.
- Ator "Supervisor": Supervisiona operações, analisa relatórios e gerencia escalas de colaboradores.
- Ator "Agente de Segurança": Executa rondas, registra ocorrências e utiliza o sistema no dia a dia operacional.
- Ator "Sistema de IA": Classifica automaticamente ocorrências e gera insights de segurança.
- Caso de Uso "Autenticar Usuário": Permite que qualquer usuário se autentique no sistema com controle de acesso baseado em papéis.
- Caso de Uso "Executar Ronda": Permite ao agente executar rondas programadas com geolocalização e checkpoints.
- Caso de Uso "Registrar Ocorrência": Permite ao agente registrar incidentes com classificação automática.
- Caso de Uso "Gerenciar Usuários": Permite ao administrador gerenciar usuários, papéis e permissões.
- Caso de Uso "Classificar com IA": Sistema automático de classificação de ocorrências usando Google Gemini.
- Caso de Uso "Gerar Relatórios": Permite gerar relatórios de rondas, ocorrências e análises.

A seguir, os casos de uso são detalhados por ator:

- Administrador: O administrador possui três casos de uso principais no sistema, cada um representando uma funcionalidade específica e essencial para a gestão completa do sistema de segurança predial. Estes casos de uso foram projetados para fornecer ao administrador o controle total sobre as operações, usuários e análises do sistema. O primeiro caso de uso, "Autenticar Usuário", é fundamental para o acesso seguro ao sistema, utilizando tecnologia JWT para garantir a segurança das credenciais. O segundo caso de uso, "Gerenciar Usuários", representa a funcionalidade mais específica do administrador, permitindo o controle completo sobre todos os usuários do sistema. O terceiro caso de uso, "Gerar Relatórios", fornece ao administrador a capacidade de analisar dados consolidados e tomar decisões estratégicas baseadas em informações precisas do sistema, como descrito na figura 4.

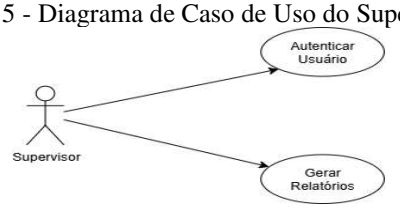
Figura 4 - Diagrama de Caso de Uso do Administrador



Fonte: Elaborado pelo autor (2025)

- Supervisor: O supervisor possui dois casos de uso principais no sistema, cada um representando uma funcionalidade específica e essencial para a supervisão operacional do sistema de segurança predial. Estes casos de uso foram projetados para fornecer ao supervisor o controle necessário sobre as operações diárias, análise de dados e coordenação das atividades de segurança. O primeiro caso de uso, "Autenticar Usuário", é fundamental para o acesso seguro ao sistema, utilizando a mesma tecnologia JWT para garantir a segurança das credenciais, porém com permissões específicas de supervisor. O segundo caso de uso, "Gerar Relatórios", fornece ao supervisor a capacidade de analisar dados operacionais, monitorar performance dos agentes e tomar decisões baseadas em informações consolidadas do sistema, permitindo uma supervisão eficaz das operações de segurança, como mostra a figura 5.

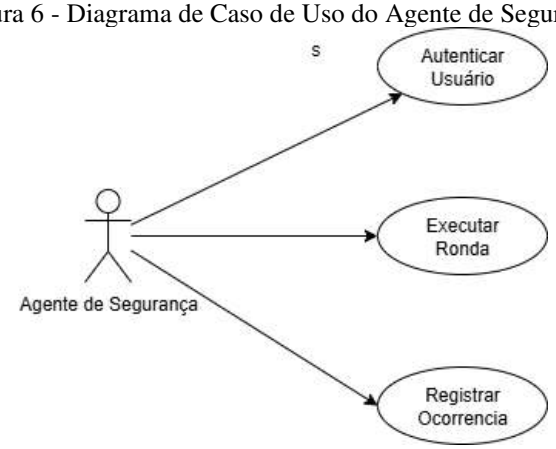
Figura 5 - Diagrama de Caso de Uso do Supervisor



Fonte: Elaborado pelo autor (2025)

- Agente de segurança: O agente de segurança possui três casos de uso principais no sistema, cada um representando uma funcionalidade específica e essencial para a execução operacional das atividades de segurança predial. Estes casos de uso foram projetados para fornecer ao agente as ferramentas necessárias para executar rondas, registrar ocorrências e utilizar o sistema no dia a dia operacional. O primeiro caso de uso, "Autenticar Usuário", é fundamental para o acesso seguro ao sistema, utilizando tecnologia JWT para garantir a segurança das credenciais com permissões específicas de agentes. O segundo caso de uso, "Executar Ronda", representa a funcionalidade mais específica do agente, permitindo a execução de rondas programadas com geolocalização e checkpoints. O terceiro caso de uso, "Registrar Ocorrência", fornece ao agente a capacidade de registrar incidentes de segurança com classificação automática, garantindo um registro preciso e organizado das situações ocorridas, como mostra a figura 6.

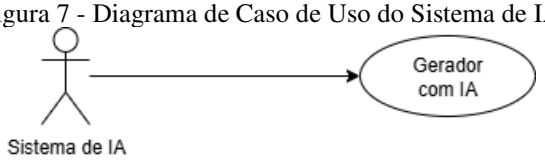
Figura 6 - Diagrama de Caso de Uso do Agente de Segurança



Fonte: Elaborado pelo autor (2025)

- Sistema de IA: O sistema de IA possui um caso de uso principal no sistema, representando uma funcionalidade específica e essencial para o processamento inteligente de dados no sistema de gestão de segurança predial. Este caso de uso foi projetado para fornecer ao sistema a capacidade de gerar análises automáticas, insights e classificações baseadas em dados históricos e descrições de ocorrências. O caso de uso "Gerar com IA" representa a funcionalidade mais específica do sistema de IA, permitindo a geração automática de classificações, insights e análises utilizando a tecnologia Google Gemini. Esta funcionalidade é fundamental para automatizar processos que anteriormente exigiam análise manual, fornecendo respostas rápidas e consistentes baseadas em inteligência artificial, como mostra a figura 7.

Figura 7 - Diagrama de Caso de Uso do Sistema de IA

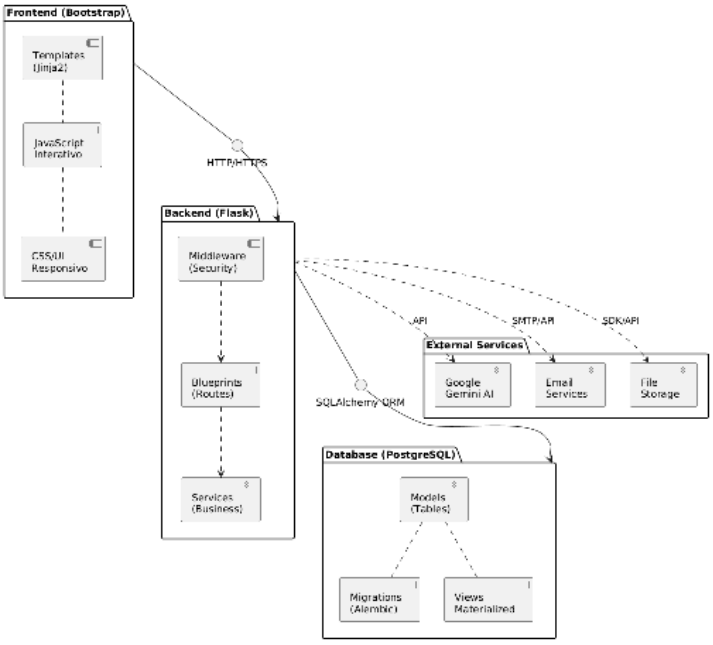


Fonte: Elaborado pelo autor (2025)

4.6 DIAGRAMAÇÕES: DIAGRAMA DE COMPONENTES

Esse diagrama apresenta a estrutura física e lógica de um sistema em termos de componentes, suas interfaces e dependências possibilitando a visualização e organização dos relacionamentos entre as partes modulares de um sistema de software. A arquitetura do sistema foi projetada seguindo uma abordagem em camadas, onde cada componente possui responsabilidades bem definidas. O frontend responsivo comunica-se com o backend através de APIs RESTful, que por sua vez interage com o banco de dados PostgreSQL e serviços externos como o Google Gemini para funcionalidades de IA. Esta estruturação facilita a manutenção, escalabilidade e evolução do sistema. O diagrama é apresentado na Figura 8.

Figura 8 - Diagrama de Componentes



Fonte: Elaborado pelo autor (2025)

O detalhamento dos componentes apresentados na Figura 8 é descrito a seguir:

- **Frontend (Bootstrap)**
  - Descrição: Interface de usuário responsiva para interação e visualização de dados. Responsabilidades: Exibir dashboards, formulários de rondas e ocorrências, relatórios interativos.
- **Backend (Flask)**

- Descrição: Processamento de lógica de negócios, autenticação e comunicação com banco de dados.

  Responsabilidades: Autenticar usuários, processar dados de rondas e ocorrências, gerar APIs RESTful.

- **Database(PostgreSQL)**
- Descrição: Armazenamento de dados de usuários, rondas, ocorrências e colaboradores. Responsabilidades: Persistir dados de segurança, armazenar logs de auditoria, views materializadas.

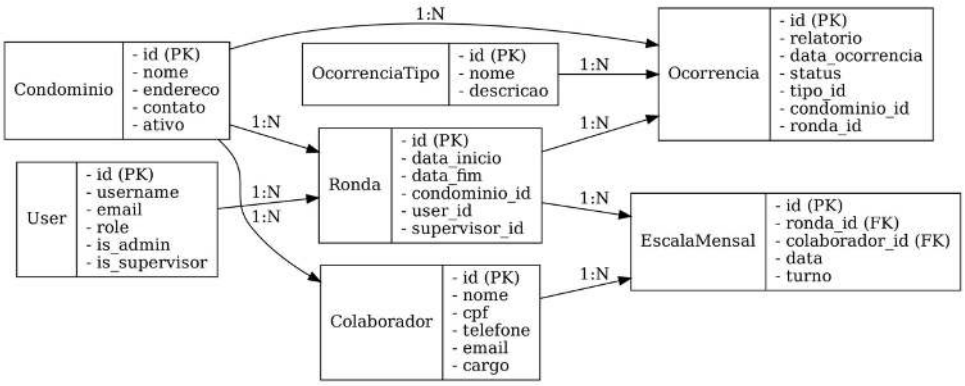
- **External Services**
- Descrição: Serviços externos para IA, comunicação e armazenamento. Responsabilidades: Classificação automática com Gemini, envio de emails, armazenamento de arquivos.

4.7 DIAGRAMAÇÕES: DIAGRAMA DE TABELAS E RELACIONAMENTOS (DTR)

Trata-se de uma representação visual que descreve a estrutura do banco de dados relacional utilizado neste projeto (incluindo suas tabelas, atributos e relacionamentos) objetivando apresentar o planejamento da documentação do banco de dados para posterior implementação.

O modelo de dados foi estruturado para suportar todas as funcionalidades do sistema de gestão de segurança, incluindo controle de usuários, gestão de rondas, registro de ocorrências e relacionamentos com condomínios e colaboradores. O banco conta com 18 tabelas principais, 4 views materializadas para otimização de consultas e 2 tabelas de relacionamento muitos-para-muitos, garantindo integridade referencial e performance adequada para as operações do sistema. As Figuras 9 e 10 apresentam o DTR.

Figura 9 - Diagrama de Tabelas e Relacionamentos



Fonte: Elaborado pelo autor (2025)

Figura 10 - Diagrama de Tabelas e Relacionamentos – Continuação



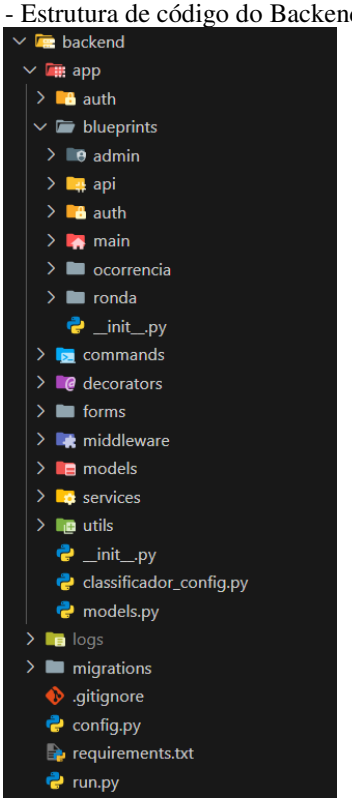
Fonte: Elaborado pelo autor (2025)

4.8 PROGRAMAÇÃO E TESTES: CODIFICAÇÃO

As figuras seguintes apresentam os fragmentos de código mais representativos desenvolvidos no âmbito do projeto, os quais ilustram a estruturação lógica e a organização implementada no sistema. A análise do código evidencia uma arquitetura alinhada com os princípios do padrão MVC (Model-View-Controller), demarcando claramente a separação entre as camadas de apresentação (View), lógica de aplicação (Controller) e persistência de dados (Model). Adicionalmente, são destacados os módulos responsáveis pela integração dos serviços de inteligência artificial, bem como a rotina implementada para a recepção, processamento e armazenamento dos fluxos de dados gerados pelos agentes de segurança.

A Figura 11 apresenta a estrutura de código do Backend (Flask), que segue o padrão arquitetural MVC, onde os models fazem o papel de persistência dos dados, os services fazem o papel de lógica de negócios e os blueprints fazem o papel de controladores de comunicação.

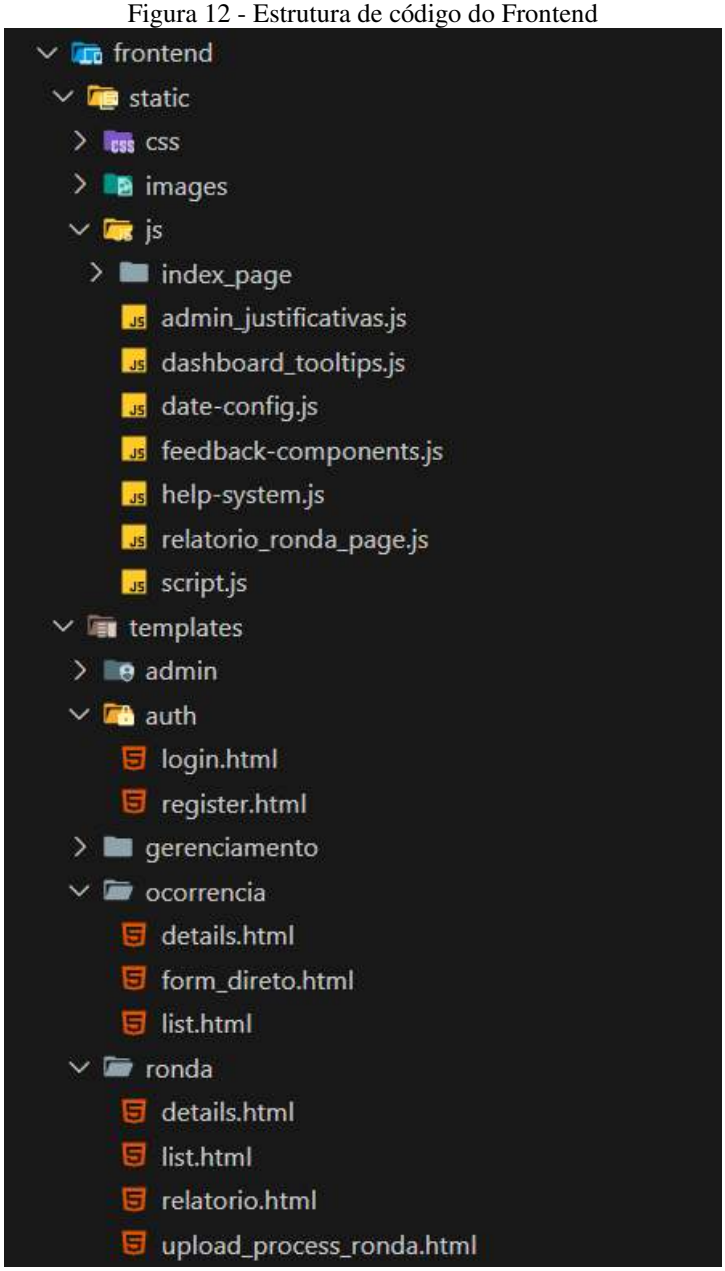
Figura 11 - Estrutura de código do Backend (Flask)



Fonte: Elaborado pelo autor (2025)

Complementando a estrutura do backend, a figura 12 representa o frontend, que foi organizado de forma modular para facilitar a manutenção e permitir a reutilização de componentes. A estrutura apresentada a seguir mostra a separação entre recursos estáticos (CSS, JavaScript e imagens) e templates HTML, seguindo as melhores práticas de desenvolvimento web para garantir responsividade e interatividade adequadas.

Figura 12 - Estrutura de código do Frontend

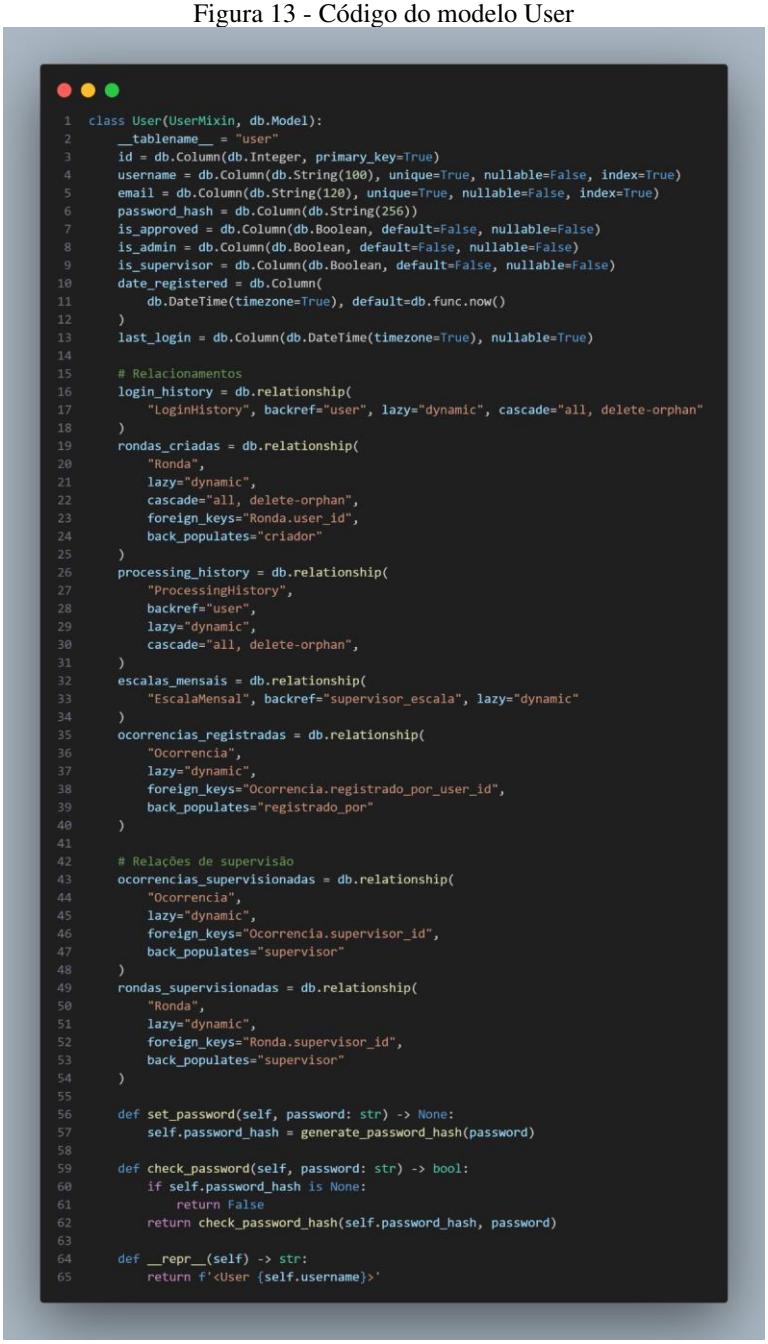


Fonte: Elaborado pelo autor (2025)

Para demonstrar a implementação prática do padrão MVC, apresenta-se o modelo User, demonstrada na figura 13, que representa a camada de persistência de dados. Este modelo implementa todas as funcionalidades relacionadas ao gerenciamento de usuários, incluindo autenticação segura,

controle de papéis e relacionamentos com outras entidades do sistema. O uso do SQLAlchemy como ORM facilita a manipulação dos dados e garante a integridade referencial.

Figura 13 - Código do modelo User



```python
class User(UserMixin, db.Model):
    __tablename__ = "user"
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(100), unique=True, nullable=False, index=True)
    email = db.Column(db.String(120), unique=True, nullable=False, index=True)
    password_hash = db.Column(db.String(256))
    is_approved = db.Column(db.Boolean, default=False, nullable=False)
    is_admin = db.Column(db.Boolean, default=False, nullable=False)
    is_supervisor = db.Column(db.Boolean, default=False, nullable=False)
    date_registered = db.Column(
        db.DateTime(timezone=True), default=db.func.now()
    )
    last_login = db.Column(db.DateTime(timezone=True), nullable=True)

    # Relacionamentos
    login_history = db.relationship(
        "LoginHistory", backref="user", lazy="dynamic", cascade="all, delete-orphan"
    )
    rondas_criadas = db.relationship(
        "Ronda",
        lazy="dynamic",
        cascade="all, delete-orphan",
        foreign_keys="Ronda.user_id",
        back_populates="criador"
    )
    processing_history = db.relationship(
        "ProcessingHistory",
        backref="user",
        lazy="dynamic",
        cascade="all, delete-orphan",
    )
    escalas_mensais = db.relationship(
        "EscalaMensal", backref="supervisor_escala", lazy="dynamic"
    )
    ocorrencias_registradas = db.relationship(
        "Ocorrencia",
        lazy="dynamic",
        foreign_keys="Ocorrencia.registrado_por_user_id",
        back_populates="registrado_por"
    )

    # Relações de supervisão
    ocorrencias_supervisionadas = db.relationship(
        "Ocorrencia",
        lazy="dynamic",
        foreign_keys="Ocorrencia.supervisor_id",
        back_populates="supervisor"
    )
    rondas_supervisionadas = db.relationship(
        "Ronda",
        lazy="dynamic",
        foreign_keys="Ronda.supervisor_id",
        back_populates="supervisor"
    )

    def set_password(self, password: str) -> None:
        self.password_hash = generate_password_hash(password)

    def check_password(self, password: str) -> bool:
        if self.password_hash is None:
            return False
        return check_password_hash(self.password_hash, password)

    def __repr__(self) -> str:
        return f'<User {self.username}>'
```

Fonte: Elaborado pelo autor (2025)

Outro modelo fundamental do sistema é a classe Ocorrencia, que gerencia todos os dados relacionados aos incidentes de segurança registrados. Este modelo demonstra a complexidade dos relacionamentos no sistema, incluindo vínculos com usuários, condomínios, tipos de ocorrência e órgãos públicos, além de implementar funcionalidades para rastreamento temporal e controle de status. A figura 14 representa a classe Ocorrencia.

Figura 14 - Código do modelo Ocorrência

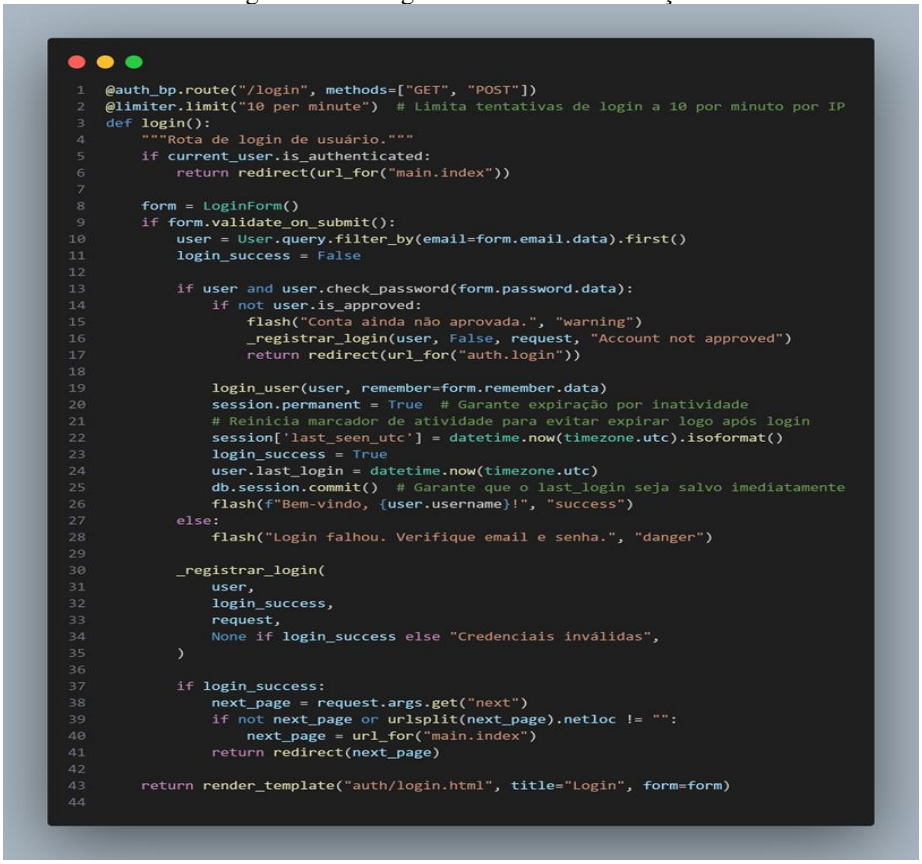


```python
class Ocorrencia(db.Model):
    """Modelo de ocorrência (incidente registrado)."""
    __tablename__ = "ocorrencia"
    id = db.Column(db.Integer, primary_key=True)
    relatorio_final = db.Column(db.Text, nullable=False)
    data_hora_ocorrencia = db.Column(
        db.DateTime(timezone=True),
        nullable=False,
        server_default=func.now(),
        index=True,
    )
    turno = db.Column(db.String(50), nullable=True)
    status = db.Column(db.String(50), nullable=False, default="Registrada", index=True)
    endereco_especifico = db.Column(db.String(255), nullable=True)
    logradouro_id = db.Column(db.Integer, db.ForeignKey("logradouro.id"), nullable=True)
    logradouro = db.relationship("Logradouro", backref="ocorrencias")
    data_criacao = db.Column(
        db.DateTime(timezone=True), default=lambda: datetime.now(timezone.utc)
    )
    data_modificacao = db.Column(
        db.DateTime(timezone=True), onupdate=lambda: datetime.now(timezone.utc)
    )
    condominio_id = db.Column(
        db.Integer, db.ForeignKey("condominio.id"), nullable=True, index=True
    )
    ocorrencia_tipo_id = db.Column(
        db.Integer, db.ForeignKey("ocorrencia_tipo.id"), nullable=False, index=True
    )
    registrado_por_user_id = db.Column(
        db.Integer, db.ForeignKey("user.id"), nullable=False, index=True
    )
    supervisor_id = db.Column(
        db.Integer,
        db.ForeignKey("user.id", name="fk_ocorrencia_supervisor_id"),
        nullable=True,
        index=True,
    )
    condominio = db.relationship("Condominio", backref="ocorrencias")
    tipo = db.relationship("OcorrenciaTipo", backref="ocorrencias")
    registrado_por = db.relationship("User", foreign_keys=[registrado_por_user_id], back_populates="ocorrencias_registradas")
    supervisor = db.relationship("User", foreign_keys=[supervisor_id], back_populates="ocorrencias_supervisionadas")

    orgaos_acionados = db.relationship(
        "OrgaoPublico",
        secondary=ocorrencia_orgaos,
        lazy="subquery",
        backref=db.backref("ocorrencias_acionadas", lazy=True),
    )
    colaboradores_envolvidos = db.relationship(
        "Colaborador",
        secondary=ocorrencia_colaboradores,
        lazy="subquery",
        backref=db.backref("ocorrencias_atendidas", lazy=True),
    )

    def __repr__(self) -> str:
        tipo_nome = self.tipo.nome if self.tipo else "N/A"
        data_str = self.data_hora_ocorrencia.strftime('%d/%m/%Y %H:%M')
        cond_nome = self.condominio.nome if self.condominio else "Sem Condomínio"
        return f'<Ocorrencia {self.id} - {tipo_nome} em {data_str} ({cond_nome})>'

__all__ = ["Ocorrencia", "ocorrencia_orgaos", "ocorrencia_colaboradores"]
```

Fonte: Elaborado pelo autor (2025)

Representando a camada de controle do padrão MVC, figuras 15 e 16, as rotas de autenticação implementam a lógica de segurança do sistema. Este código demonstra como são tratados os processos de login, validação de credenciais, geração de tokens JWT e controle de sessões. A implementação inclui limitação de tentativas de login e suporte tanto para interfaces web quanto para APIs REST.

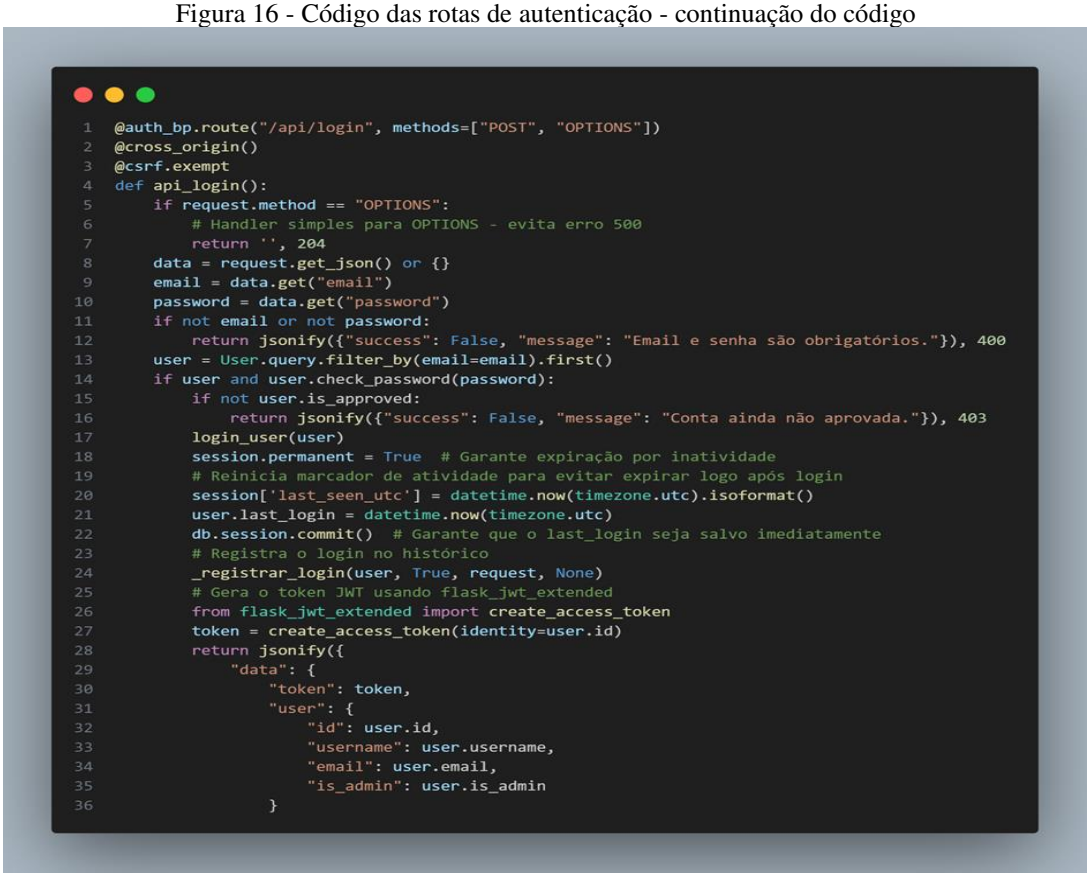
Figura 15 - Código das rotas de autenticação



```python
@auth_bp.route("/login", methods=["GET", "POST"])
@limiter.limit("10 per minute")  # Limita tentativas de login a 10 por minuto por IP
def login():
    """Rota de login de usuário."""
    if current_user.is_authenticated:
        return redirect(url_for("main.index"))

    form = LoginForm()
    if form.validate_on_submit():
        user = User.query.filter_by(email=form.email.data).first()
        login_success = False

        if user and user.check_password(form.password.data):
            if not user.is_approved:
                flash("Conta ainda não aprovada.", "warning")
                _registrar_login(user, False, request, "Account not approved")
                return redirect(url_for("auth.login"))

            login_user(user, remember=form.remember.data)
            session.permanent = True  # Garante expiração por inatividade
            # Reinicia marcador de atividade para evitar expirar logo após login
            session['last_seen_utc'] = datetime.now(timezone.utc).isoformat()
            login_success = True
            user.last_login = datetime.now(timezone.utc)
            db.session.commit()  # Garante que o last_login seja salvo imediatamente
            flash(f"Bem-vindo, {user.username}!", "success")
        else:
            flash("Login falhou. Verifique email e senha.", "danger")

        _registrar_login(
            user,
            login_success,
            request,
            None if login_success else "Credenciais inválidas",
        )

        if login_success:
            next_page = request.args.get("next")
            if not next_page or urlsplit(next_page).netloc != "":
                next_page = url_for("main.index")
            return redirect(next_page)

    return render_template("auth/login.html", title="Login", form=form)
```

Fonte: Elaborado pelo autor (2025)

Figura 16 - Código das rotas de autenticação - continuação do código



```python
@auth_bp.route("/api/login", methods=["POST", "OPTIONS"])
@cross_origin()
@csrf.exempt
def api_login():
    if request.method == "OPTIONS":
        # Handler simples para OPTIONS - evita erro 500
        return '', 204
    data = request.get_json() or {}
    email = data.get("email")
    password = data.get("password")
    if not email or not password:
        return jsonify({"success": False, "message": "Email e senha são obrigatórios."}), 400
    user = User.query.filter_by(email=email).first()
    if user and user.check_password(password):
        if not user.is_approved:
            return jsonify({"success": False, "message": "Conta ainda não aprovada."}), 403
        login_user(user)
        session.permanent = True  # Garante expiração por inatividade
        # Reinicia marcador de atividade para evitar expirar logo após login
        session['last_seen_utc'] = datetime.now(timezone.utc).isoformat()
        user.last_login = datetime.now(timezone.utc)
        db.session.commit()  # Garante que o last_login seja salvo imediatamente
        # Registra o login no histórico
        _registrar_login(user, True, request, None)
        # Gera o token JWT usando flask_jwt_extended
        from flask_jwt_extended import create_access_token
        token = create_access_token(identity=user.id)
        return jsonify({
            "data": {
                "token": token,
                "user": {
                    "id": user.id,
                    "username": user.username,
                    "email": user.email,
                    "is_admin": user.is_admin
                }
```

Fonte: Elaborado pelo autor (2025)

Uma das principais inovações do sistema é a integração com inteligência artificial através do Google Gemini. O serviço base de IA apresentado implementa funcionalidades avançadas como cache inteligente, fallback automático entre chaves de API, controle de rate limiting e tratamento robusto de erros. Esta implementação garante alta disponibilidade e performance nas operações de processamento de texto. As figuras 17 e 18 mostram a estrutura do código.

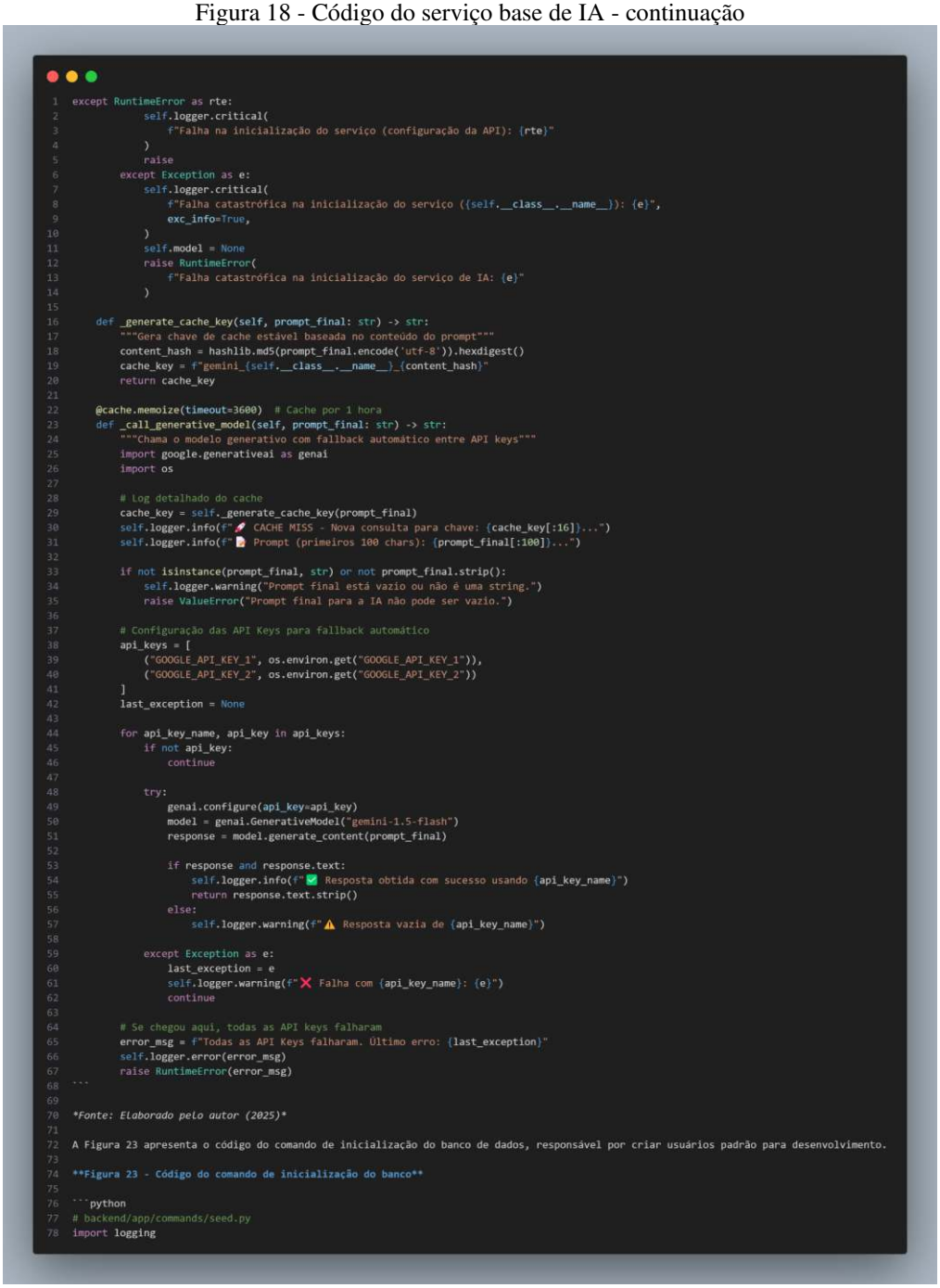
Figura 17 - Código do serviço base de IA



```python
class BaseGenerativeService:
    def __init__(self, model_name="gemini-1.5-flash-latest"):
        self.logger = logging.getLogger(self.__class__.__name__)
        self.model = None
        self._google_api_key = None

        # Rate limiting para APIs Gemini
        self._api_usage = {
            "GOOGLE_API_KEY_1": {"last_used": None, "daily_count": 0, "last_reset": None},
            "GOOGLE_API_KEY_2": {"last_used": None, "daily_count": 0, "last_reset": None}
        }

        # Limites de rate (ajuste conforme necessário)
        self._rate_limits = {
            "requests_per_day": 45,  # Deixe margem de segurança
            "min_interval_seconds": 2  # Intervalo mínimo entre requisições
        }

        try:
            # Tenta usar GOOGLE_API_KEY_1 primeiro, depois GOOGLE_API_KEY como fallback
            self._google_api_key = os.getenv("GOOGLE_API_KEY_1") or os.getenv("GOOGLE_API_KEY")
            if not self._google_api_key:
                self.logger.error(
                    "API Key do Google (GOOGLE_API_KEY_1 ou GOOGLE_API_KEY) não encontrada nas variáveis de ambiente."
                )
                raise RuntimeError(
                    "API Key do Google (GOOGLE_API_KEY_1 ou GOOGLE_API_KEY) não configurada nas variáveis de ambiente."
                )

            genai.configure(api_key=self._google_api_key)
            self.logger.info(
                "Configuração da API Key do Google bem-sucedida para o serviço."
            )

            generation_config = {
                "temperature": 0.7,
                "top_p": 0.95,
                "top_k": 64,
                "max_output_tokens": 8192,
                "response_mime_type": "text/plain",
            }
            safety_settings = [
                {
                    "category": "HARM_CATEGORY_HARASSMENT",
                    "threshold": "BLOCK_MEDIUM_AND_ABOVE",
                },
                {
                    "category": "HARM_CATEGORY_HATE_SPEECH",
                    "threshold": "BLOCK_MEDIUM_AND_ABOVE",
                },
                {
                    "category": "HARM_CATEGORY_SEXUALLY_EXPLICIT",
                    "threshold": "BLOCK_MEDIUM_AND_ABOVE",
                },
                {
                    "category": "HARM_CATEGORY_DANGEROUS_CONTENT",
                    "threshold": "BLOCK_MEDIUM_AND_ABOVE",
                },
            ]
```

Fonte: Elaborado pelo autor (2025)

Figura 18 - Código do serviço base de IA - continuação



```
except RuntimeError as rte:
            self.logger.critical(
                f"Falha na inicialização do serviço (configuração da API): {rte}"
            )
            raise
        except Exception as e:
            self.logger.critical(
                f"Falha catastrófica na inicialização do serviço ({self.__class__.__name__}): {e}",
                exc_info=True,
            )
            self.model = None
            raise RuntimeError(
                f"Falha catastrófica na inicialização do serviço de IA: {e}"
            )

    def _generate_cache_key(self, prompt_final: str) -> str:
        """Gera chave de cache estável baseada no conteúdo do prompt"""
        content_hash = hashlib.md5(prompt_final.encode('utf-8')).hexdigest()
        cache_key = f"gemini_{self.__class__.__name__}_{content_hash}"
        return cache_key

    @cache.memoize(timeout=3600)  # Cache por 1 hora
    def _call_generative_model(self, prompt_final: str) -> str:
        """Chama o modelo generativo com fallback automático entre API keys"""
        import google.generativeai as genai
        import os

        # Log detalhado do cache
        cache_key = self._generate_cache_key(prompt_final)
        self.logger.info(f"🚀 CACHE MISS - Nova consulta para chave: {cache_key[:16]}...")
        self.logger.info(f"📝 Prompt (primeiros 100 chars): {prompt_final[:100]}...")

        if not isinstance(prompt_final, str) or not prompt_final.strip():
            self.logger.warning("Prompt final está vazio ou não é uma string.")
            raise ValueError("Prompt final para a IA não pode ser vazio.")

        # Configuração das API Keys para fallback automático
        api_keys = [
            ("GOOGLE_API_KEY_1", os.environ.get("GOOGLE_API_KEY_1")),
            ("GOOGLE_API_KEY_2", os.environ.get("GOOGLE_API_KEY_2"))
        ]
        last_exception = None

        for api_key_name, api_key in api_keys:
            if not api_key:
                continue

            try:
                genai.configure(api_key=api_key)
                model = genai.GenerativeModel("gemini-1.5-flash")
                response = model.generate_content(prompt_final)

                if response and response.text:
                    self.logger.info(f"✅ Resposta obtida com sucesso usando {api_key_name}")
                    return response.text.strip()
                else:
                    self.logger.warning(f"⚠️ Resposta vazia de {api_key_name}")

            except Exception as e:
                last_exception = e
                self.logger.warning(f"❌ Falha com {api_key_name}: {e}")
                continue

        # Se chegou aqui, todas as API keys falharam
        error_msg = f"Todas as API Keys falharam. Último erro: {last_exception}"
        self.logger.error(error_msg)
        raise RuntimeError(error_msg)
```

Fonte: Elaborado pelo autor (2025)

A Figura 23 apresenta o código do comando de inicialização do banco de dados, responsável por criar usuários padrão para desenvolvimento.

Figura 23 - Código do comando de inicialização do banco

```python
# backend/app/commands/seed.py
import logging
```

Fonte: Elaborado pelo autor (2025)

O sistema utiliza blueprints do Flask para organizar as rotas de forma modular. O blueprint de autenticação apresentado implementa todas as funcionalidades relacionadas ao ciclo de vida do usuário, incluindo registro, login, logout e validação de tokens. Esta estrutura modular facilita a manutenção e permite a reutilização de código em diferentes contextos da aplicação, conforme ilustrado na Figura 19, apresentada abaixo.

Figura 19 - Código do blueprint de autenticação



```python
auth_bp = Blueprint("auth", __name__)

@auth_bp.route("/register", methods=["GET", "POST"])
def register():
    """Rota de registro de novo usuário."""
    if current_user.is_authenticated:
        return redirect(url_for("main.index"))

    form = RegistrationForm()
    if form.validate_on_submit():
        try:
            username = form.username.data or ""
            email = form.email.data or ""
            password = form.password.data or ""
            user = User(username=username, email=email)
            user.set_password(password)
            db.session.add(user)
            db.session.commit()
            flash(
                "Conta criada com sucesso. Aguarde aprovação do administrador.", "info"
            )
            return redirect(url_for("auth.login"))
        except Exception as e:
            db.session.rollback()
            current_app.logger.error(
                f"Erro ao registrar usuário {form.username.data}: {e}"
            )
            flash("Erro ao criar a conta. Tente novamente.", "danger")
    return render_template("auth/register.html", title="Registrar", form=form)

@auth_bp.route("/login", methods=["GET", "POST"])
@limiter.limit("10 per minute")
def login():
    """Rota de login de usuário."""
    if current_user.is_authenticated:
        return redirect(url_for("main.index"))

    form = LoginForm()
    if form.validate_on_submit():
        user = User.query.filter_by(email=form.email.data).first()
        login_success = False

        if user and user.check_password(form.password.data):
            if not user.is_approved:
                flash("Conta ainda não aprovada.", "warning")
                return redirect(url_for("auth.login"))

            login_user(user, remember=form.remember.data)
            session.permanent = True
            session['last_seen_utc'] = datetime.now(timezone.utc).isoformat()
            login_success = True
            user.last_login = datetime.now(timezone.utc)
            db.session.commit()
            flash(f"Bem-vindo, {user.username}!", "success")
        else:
            flash("Login falhou. Verifique email e senha.", "danger")

        if login_success:
            next_page = request.args.get("next")
            if not next_page or urlsplit(next_page).netloc != "":
                next_page = url_for("main.index")
            return redirect(next_page)

    return render_template("auth/login.html", title="Login", form=form)

@auth_bp.route("/logout")
def logout():
    """Rota de logout de usuário."""
    logout_user()
    session.clear()
    flash("Você foi desconectado com sucesso.", "info")
    return redirect(url_for("auth.login"))
```

Fonte: Elaborado pelo autor (2025)

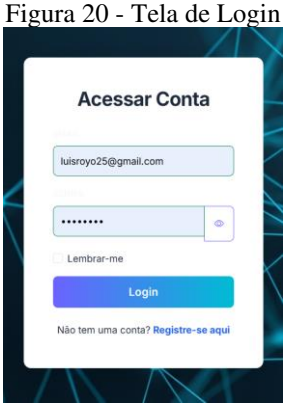
4.9 PROGRAMAÇÃO E TESTES: TESTES

Com o objetivo de validar o funcionamento do sistema desenvolvido, foi realizada uma série de testes práticos envolvendo todo o fluxo desde a autenticação de usuários até a geração de relatórios. Inicialmente, foi testado o sistema de autenticação com diferentes tipos de usuários (Admin, Supervisor, Agente), comprovado por meio de logs do sistema. Em seguida, foi realizado o teste de execução de rondas e registro de atividades, validado através de dados armazenados no banco. No

ambiente web, foram capturadas imagens demonstrando o sistema funcionando corretamente, incluindo dashboards, formulários e relatórios.

Para demonstrar o funcionamento prático do sistema, são apresentadas, a seguir, as principais interfaces desenvolvidas. A primeira tela exibida é a interface de login, que representa o ponto de entrada do sistema e implementa os controles de segurança necessários para autenticação dos usuários, conforme pode ser observado na Figura 20.

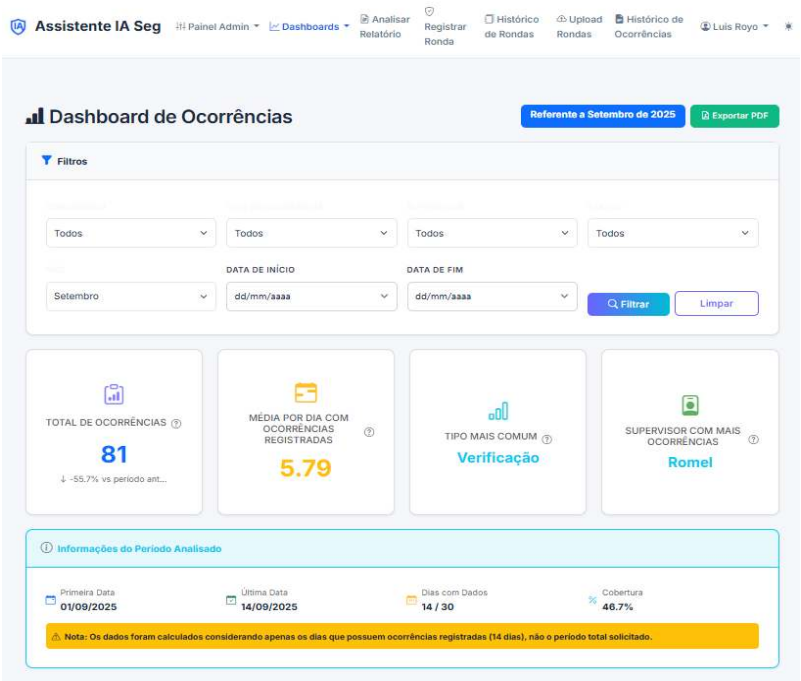
Figura 20 - Tela de Login



Fonte: Elaborado pelo autor (2025)

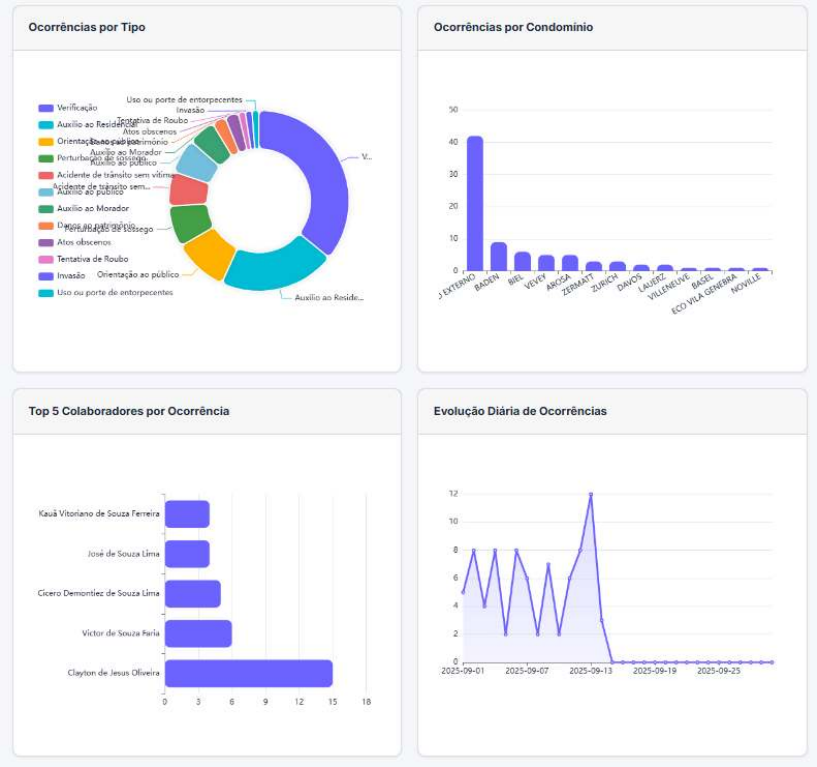
Após a autenticação, os usuários têm acesso ao dashboard principal do sistema, que centraliza as informações mais relevantes sobre ocorrências de segurança. Esta interface demonstra a capacidade analítica do sistema, oferecendo filtros avançados, visualizações gráficas e métricas em tempo real que auxiliam na tomada de decisões estratégicas, conforme ilustrado nas Figuras 21 e 22.

Figura 21 - Dashboard de Ocorrências



Fonte: Elaborado pelo autor (2025)

Figura 22 - Dashboard de Ocorrências - Continuação



Fonte: Elaborado pelo autor (2025)

A Figura 23 apresenta a tela principal do sistema (Analisador de Relatórios com IA), onde o usuário cola um relatório bruto e o sistema corrige automaticamente gramática, pontuação e formatação usando inteligência artificial.

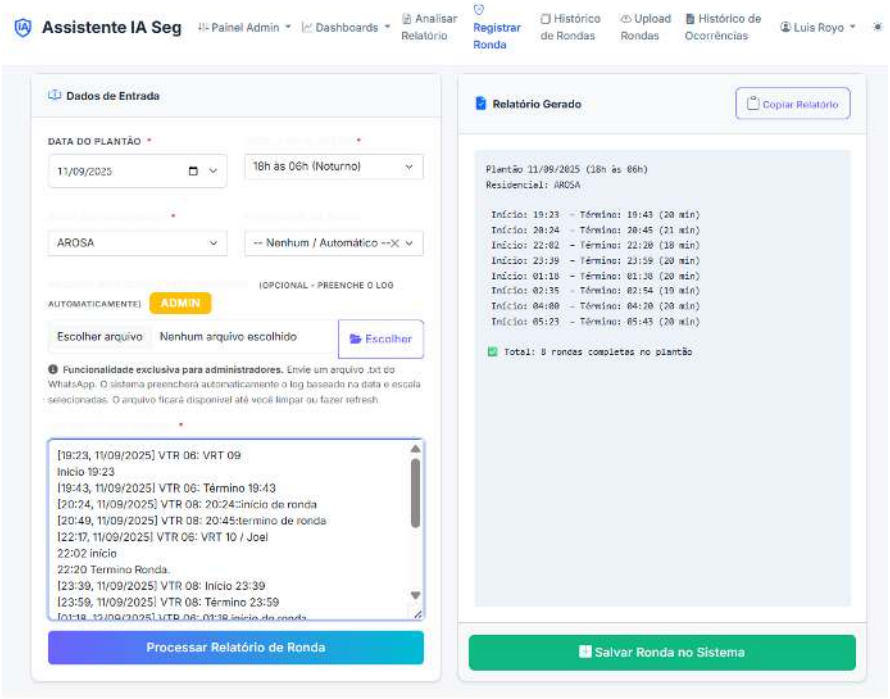
Figura 23 - Analisador de Relatórios com IA (Tela Principal)



Fonte: Elaborado pelo autor (2025)

Complementando as funcionalidades do sistema, é oferecido o processamento automatizado de registros de rondas. Esta interface permite importar logs de comunicação (como mensagens do WhatsApp) e transformá-los automaticamente em relatórios estruturados por meio de algoritmos de processamento de texto e expressões regulares, demonstrando a capacidade de tratar dados não estruturados e convertê-los em informações organizadas, sem a utilização de técnicas de inteligência artificial, conforme apresentado na Figura 24.

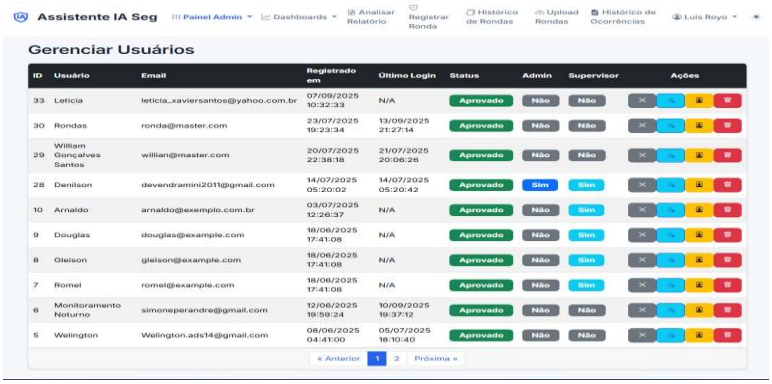
Figura 24 - Sistema de Registro de Rondas



Fonte: Elaborado pelo autor (2025)

O sistema implementa funcionalidades administrativas completas para gerenciamento de usuários e permissões. A interface administrativa apresentada permite aos administradores controlar o acesso ao sistema, definir papéis e gerenciar o ciclo de vida dos usuários, demonstrando a robustez do sistema de controle de acesso implementado, conforme ilustrado na Figura 25.

Figura 25 - Tela de Administração de Usuários



Fonte: Elaborado pelo autor (2025)

5 CONSIDERAÇÕES FINAIS

O desenvolvimento do sistema web para gestão de segurança predial demonstrou ser tecnicamente viável e economicamente acessível, sobretudo pelo uso de tecnologias consolidadas e de baixo custo. A integração entre o backend em Flask, o banco de dados PostgreSQL e recursos de inteligência artificial validou o funcionamento completo da solução, desde a autenticação de usuários até a geração de relatórios automatizados.

A adoção da metodologia Scrum foi essencial para a organização do projeto, oferecendo flexibilidade e permitindo ajustes ágeis ao longo do processo. Os testes confirmaram a eficiência do sistema na transmissão de dados com baixa latência e no armazenamento consistente das informações no PostgreSQL.

A interface construída com Bootstrap destacou-se pela boa usabilidade, pela responsividade e pela compatibilidade em diferentes dispositivos e navegadores. Entre as funcionalidades, merecem destaque a classificação automática de ocorrências com IA e a geração de relatórios históricos, que ampliam as possibilidades de análise e agregam valor às informações coletadas pelos agentes de segurança.

A arquitetura baseada no padrão MVC favoreceu a manutenção e a evolução do código, enquanto a integração com o Google Gemini trouxe automação inteligente para a classificação de registros. O sistema de cache implementado também contribuiu para otimizar o desempenho nas consultas à API de IA.

Para trabalhos futuros, propõe-se a expansão do sistema com recursos como notificações em tempo real via WebSocket, integração com câmeras de monitoramento, gráficos interativos mais sofisticados e suporte multilíngue. Outro avanço promissor seria a criação de regras de alerta automáticas baseadas em padrões de ocorrência, o que ampliaria o potencial de aplicação tanto em ambientes industriais quanto residenciais.

A implementação de testes automatizados, com cobertura de 99%, assegura a confiabilidade do sistema, enquanto a documentação técnica detalhada facilita sua manutenção e evolução. Dessa forma, este trabalho apresenta uma solução prática, escalável e de código aberto, que contribui para a modernização da gestão de segurança em condomínios e ambientes corporativos. Os resultados confirmam a viabilidade técnica e econômica da integração de tecnologias web e inteligência artificial na segurança privada.

REFERÊNCIAS

ASSOCIAÇÃO BRASILEIRA DAS EMPRESAS DE SISTEMAS ELETRÔNICOS DE SEGURANÇA. **Mercado de segurança eletrônica fatura R$ 12 bilhões em 2023.** 2024. Disponível em: https://www.abese.org.br/conteudo/pesquisas-do-setor. Acesso em: 01 set. 2025.

CAMARGO, Robson; RIBAS, Thomaz. **Gestão ágil de projetos: as melhores soluções para suas necessidades**. São Paulo: Saraiva Uni, 2019. 232 p.

CARVALHO, D. S., BALTHAZAR, G. R., DIAS, C. R., ARAÚJO, M. A. P., & MONTEIRO, P. H. R. (2006). S²O: **Uma Ferramenta de Apoio ao Aprendizado de Sistemas Operacionais**. In Anais do XIV Workshop sobre Educação em Computação, Campo Grande/MS.

GUIMARÃES, C. A. **Inteligência artificial e segurança pública: o policiamento preditivo e os desafios à proteção de dados pessoais e à não discriminação.** Revista Brasileira de Políticas Públicas, v. 12, n. 1, p. 308-327, 2022.

GONÇALVEZ, A., SOUZA, M., SILVA, R., SILVA, D., BUENO, P., Balthazar, G. R. (2014). **Desenvolvimento de Jogos Educacionais na Área de Matemática em Escola de Ensino Fundamental**, XIX Congresso Internacional de Informática Educativa, p. 622 627.

KÖHLER, C. **Gestão da Segurança Corporativa – Aplicação do Método PDCA.** São Paulo: Editora Sicurezza, 2021.

LAUDON, K. C.; LAUDON, J. P. **Sistemas de Informação Gerenciais.** 14. ed. São Paulo: Pearson Prentice Hall, 2021.

MONTEIRO, Milton Garcia et al. **Policiamento preditivo e inteligência artificial: análise de desempenho do algoritmo de aprendizado de máquina supervisionado Random Forest na predição de ocorrências policiais de roubo nas zonas da região metropolitana de São Luís**. **Revista FT**, v. 28, n. 137, ago. 2024. DOI: 10.69849/revistaft/fa10202408051918.

OMOL, Edwin Juma. **Organizational digital transformation: from evolution to future trends. Digital Transformation and Society**, Bingley, v. 3, n. 3, p. 240-256, 2023. DOI: 10.1108/dts-08-2023-0061. Disponível em: https://doi.org/10.1108/dts-08-2023-0061. Acesso em: 01 set. 2025.

PRESSMAN, R. S.; MAXIM, B. R. **Engenharia de Software: Uma Abordagem Profissional.** 9. ed. Porto Alegre: AMGH, 2021.

SUTHERLAND, Jeff; LUA, Nina. **Scrum: guia prático: maior produtividade, melhores resultados, aplicação imediata.** Rio de Janeiro: Sextante, 2020. 240 p.

VăRZARU, Anca Antoaneta; BOCEAN, Claudiu George. **Digital transformation and innovation: the influence of digital technologies on turnover from innovation activities and types of innovation**. **Systems**, Basel, v. 12, n. 9, p. 359, 11 set. 2024. DOI: 10.3390/systems12090359.