HiveCall: Uma colmeia de proteção em cada ligação

Trabalho de Conclusão de Curso

Sérgio Cerqueira Santos

Manoel Carvalho Marques Neto
Orientador

Instituto Federal da Bahia - IFBA
Curso de Análise e Desenvolvimento de Sistemas
Campus Salvador

Salvador, Bahia, Brasil
11 de Fevereiro de 2026

Sumário

1 Visão geral

1.1 Declaração do Problema

O crescimento de golpes e chamadas telefônicas fraudulentas no Brasil tem sido amplamente documentado por órgãos reguladores e instituições nacionais. De acordo com reportagens recentes e dados oficiais, o volume de chamadas automatizadas e indesejadas atingiu níveis elevados, impactando diretamente a privacidade, a segurança e a confiança dos usuários de telefonia móvel [1, 2].

Grande parte desses golpes utiliza técnicas de engenharia social, explorando fatores como urgência, medo e confiança para induzir as vítimas a fornecer dados pessoais ou realizar transações financeiras indevidas. Esse tipo de abordagem é recorrente em crimes digitais no contexto brasileiro, conforme apontam estudos e cartilhas de conscientização elaborados por entidades especializadas em segurança da informação [3].

Embora existam soluções comerciais voltadas à identificação e ao bloqueio de chamadas suspeitas, relatórios nacionais indicam que essas ferramentas, em geral, apresentam limitações relacionadas ao custo, à transparência no tratamento dos dados e à efetiva proteção da privacidade do usuário [4]. Além disso, muitas dessas soluções adotam mecanismos de bloqueio automático, o que pode resultar em falsos positivos e prejuízos à experiência do usuário.

Diante desse cenário, torna-se necessária a adoção de soluções acessíveis e colaborativas, que permitam a identificação progressiva de números suspeitos com base na participação dos próprios usuários. Modelos colaborativos de compartilhamento de informações têm sido incentivados por iniciativas brasileiras de governança da Internet como uma estratégia eficaz para aumentar a segurança e a conscientização digital [5].

Nesse contexto, o HiveCall propõe uma solução baseada em uma base de dados colaborativa para classificar ligações telefônicas e alertar usuários sobre possíveis golpes, promovendo maior segurança, privacidade e uso consciente dos serviços de telefonia.

1.2 Proposta de Solução de Software

O presente trabalho propõe o desenvolvimento do HiveCall, um aplicativo multiplataforma voltado à identificação e classificação de chamadas telefônicas suspeitas ou fraudulentas. O sistema baseia-se em uma base de dados colaborativa, na qual os próprios usuários podem registrar e consultar números reportados, contribuindo coletivamente para a construção de uma rede de proteção contra golpes telefônicos.

A solução adota uma arquitetura moderna e escalável, utilizando o framework Spring Boot no back-end e a linguagem Kotlin no desenvolvimento mobile. O MongoDB é empregado como banco de dados NoSQL, garantindo flexibilidade para armazenar e consultar grandes volumes de informações de forma eficiente. Além disso, a aplicação segue os princípios da arquitetura hexagonal e dos microsserviços, o que facilita a manutenção, a testabilidade e a evolução do

sistema.

A lógica de classificação dos números telefônicos é baseada em critérios objetivos definidos a partir das denúncias registradas na base de dados. Cada número possui um contador de ocorrências, que representa a quantidade de vezes em que foi reportado pelos usuários. A partir desse valor, o sistema atribui uma classificação ao número, conforme regras pré-estabelecidas.

De forma simplificada, a classificação dos números telefônicos ocorre com base na quantidade de denúncias registradas na base de dados colaborativa. Números sem registros prévios são considerados seguros, enquanto aqueles que atingem um volume superior a 20 denúncias são classificados como suspeitos. Neste caso, o sistema apenas alerta o usuário, permitindo que ele decida se deseja ou não realizar o bloqueio do número. Essa abordagem evita bloqueios automáticos imediatos e contribui para a redução de falsos positivos, garantindo maior controle e autonomia ao usuário.

Dessa forma, o HiveCall busca oferecer uma solução acessível e segura, que promova a privacidade do usuário e a colaboração comunitária, tornando o combate a fraudes telefônicas e ligações de spam mais efetivo.

1.3 Tecnologias adotadas

O desenvolvimento do HiveCall faz uso de um conjunto de tecnologias modernas que garantem segurança, escalabilidade, desempenho e facilidade de integração entre os diferentes componentes do sistema. A seguir, são apresentadas as principais ferramentas e tecnologias utilizadas:

- **IntelliJ IDEA IDE**: Ambiente de desenvolvimento integrado utilizado para codificação, depuração e execução dos projetos Java e Kotlin. Sua integração nativa com o Maven, Spring Boot e Git torna o desenvolvimento mais ágil e organizado;
- **Spring Boot**: Framework Java que simplifica a criação de aplicações *back-end* robustas, escaláveis e seguras. No HiveCall, ele é utilizado para estruturar os microsserviços, gerenciar dependências e facilitar a integração com o banco de dados e demais componentes da arquitetura;
- **Eureka Server**: Serviço de registro e descoberta utilizado para gerenciar os microsserviços da aplicação. Ele permite que os serviços se comuniquem de forma dinâmica e autônoma, eliminando a necessidade de configurações estáticas e facilitando a escalabilidade da solução;
- **API Gateway**: Responsável por centralizar o acesso aos diferentes microsserviços, garantindo segurança, controle de tráfego e roteamento eficiente das requisições. Além disso, o gateway simplifica a autenticação e o balanceamento de carga, otimizando a comunicação entre o aplicativo e os serviços internos;
- **Maven**: Ferramenta de automação de compilação e gerenciamento de dependências utilizada nos microsserviços do HiveCall. O Maven simplifica o processo de build, testes e empacotamento das aplicações Java, garantindo padronização e reprodutibilidade no

ambiente de desenvolvimento;

- **Firebase Authentication**: Serviço de autenticação gerenciado pelo Google, utilizado para o registro e login de usuários. Sua integração com o HiveCall garante uma autenticação segura e padronizada, baseada em *tokens JWT (JSON Web Token)*, reduzindo o risco de fraudes e simplificando o controle de acesso;
- **MongoDB**: Banco de dados NoSQL orientado a documentos, utilizado para armazenar os registros de chamadas e denúncias enviadas pelos usuários. Sua estrutura flexível permite lidar com grandes volumes de dados de forma eficiente e escalável;
- **MongoDB Compass**: Ferramenta gráfica oficial do MongoDB utilizada para visualizar, consultar e gerenciar coleções do banco de dados de forma prática e intuitiva durante o desenvolvimento e a fase de testes;
- **Android Studio**: Ambiente de desenvolvimento oficial para aplicações Android, utilizado na implementação do aplicativo móvel do HiveCall. Ele fornece um conjunto de ferramentas integradas para codificação, emulação, depuração e geração de pacotes de instalação (*APKs*), otimizando o fluxo de desenvolvimento.
- **Kotlin**: Linguagem moderna e expressiva utilizada no desenvolvimento do aplicativo móvel para Android. Sua compatibilidade com Java e suporte a recursos avançados tornam o código mais seguro e produtivo, além de otimizar o desempenho do aplicativo;

Em conjunto, essas tecnologias formam uma infraestrutura distribuída, segura e colaborativa, capaz de oferecer um sistema de classificação e denúncia de chamadas fraudulentas em tempo real, priorizando a privacidade dos usuários e a eficiência na comunicação entre serviços.

1.4 Trabalhos Relacionados

O aumento de chamadas telefônicas fraudulentas e o uso de robocalls têm motivado o desenvolvimento de diversas soluções voltadas à proteção e identificação de chamadas suspeitas. Entre as principais iniciativas existentes, destacam-se aplicativos como o Truecaller, Hiya e o Whoscall, que são amplamente utilizados para identificação de números desconhecidos e bloqueio de ligações indesejadas.

O Truecaller [6] é um dos aplicativos mais populares, oferecendo identificação de chamadas com base em um banco de dados global alimentado por usuários. No entanto, seu funcionamento depende da coleta extensiva de contatos e dados pessoais, o que levanta preocupações relacionadas à privacidade e ao uso de informações sensíveis. O Hiya [7], por sua vez, adota uma abordagem semelhante, com foco na detecção de spam telefônico, mas limita algumas de suas funcionalidades à versão paga. Já o Whoscall [8] apresenta boa precisão na classificação de chamadas, porém também exige permissões de acesso amplas e exibe anúncios em sua versão gratuita.

Diferentemente dessas soluções, o HiveCall propõe uma alternativa colaborativa, gratuita e com foco em privacidade, em que os usuários podem reportar números suspeitos sem a

necessidade de compartilhar seus contatos ou histórico de ligações. A aplicação é construída com uma arquitetura moderna baseada em microsserviços, utilizando Spring Boot, MongoDB e Firebase Authentication, o que garante maior escalabilidade, segurança e transparência na gestão dos dados.

Além disso, o HiveCall possui caráter puramente acadêmico e não possui fins lucrativos. O projeto foi desenvolvido com objetivos educacionais e científicos, visando a aplicação prática de conceitos de engenharia de software, arquitetura de sistemas e segurança da informação.

Tabela 1: Comparativo entre o HiveCall e soluções existentes

Característica	**HiveCall (proposta)**	**Truecaller** [6]	**Hiya** [7]	**Whoscall** [8]
Modelo de uso	Aplicativo gratuito e colaborativo	Aplicativo comercial (freemium)	Aplicativo comercial (freemium)	Aplicativo comercial (freemium)
Privacidade dos dados	Alta, não requer acesso a contatos ou dados pessoais	Baixa, requer acesso à lista de contatos	Moderada, coleta dados de uso e chamadas	Moderada, coleta dados para anúncios
Base de dados	Colaborativa e validada por usuários	Global, alimentada automaticamente	Global, mantida pela empresa	Global, alimentada por usuários
Código-fonte	Aberto (open source, foco acadêmico)	Fechado (proprietário)	Fechado (proprietário)	Fechado (proprietário)
Custo ao usuário	Gratuito	Gratuito com versão premium	Gratuito com versão premium	Gratuito com anúncios
Foco principal	Segurança, privacidade e colaboração	Identificação de chamadas e spam	Bloqueio de spam e segurança empresarial	Identificação e bloqueio de chamadas
Arquitetura técnica	Microsserviços e arquitetura hexagonal	Não especificada publicamente	Centralizada em servidores próprios	Centralizada em servidores próprios
Armazenamento de dados	MongoDB (NoSQL)	Servidores proprietários	Servidores proprietários	Servidores proprietários
Autenticação	Firebase + OAuth 2.0	Login via conta e telefone	Login via conta ou dispositivo	Login via número telefônico
Diferencial	Foco em privacidade, transparência e contribuição comunitária	Ampla base global de dados	Integração com operadoras	Forte presença em mercados asiáticos

2 Requisitos

2.1 Requisitos Funcionais

RF1 O sistema deve permitir que o usuário crie uma conta por meio do aplicativo, utilizando o *Firebase Authentication*;

RF2 O sistema deve validar o acesso do usuário através do *Firebase* e do protocolo *OAuth 2.0*, garantindo a segurança e o controle de sessão por meio de *tokens JWT*;

RF3 O sistema deve permitir que o usuário registre uma denúncia, informando o número telefônico;

RF4 Os usuários devem poder avaliar ou confirmar denúncias feitas por outros usuários, fortalecendo a base de dados colaborativa e aumentando a confiabilidade das informações;

RF5 O sistema deve classificar automaticamente as chamadas com base no histórico e frequência de denúncias, exibindo o nível de risco ao usuário antes do atendimento;

RF6 O aplicativo deve armazenar e permitir que o usuário visualize seu histórico de denúncias realizadas;

RF7 As requisições do aplicativo devem ser roteadas através do *API Gateway*, que será responsável por direcionar o tráfego para os microsserviços corretos e aplicar as políticas de segurança;

RF8 Todos os microsserviços devem ser registrados no *Eureka Server*, permitindo a descoberta automática e a comunicação dinâmica entre os componentes da aplicação;

RF9 O aplicativo deve exibir uma notificação ao usuário quando um número for classificado como potencialmente fraudulento, alertando-o antes de atender a chamada.

RF10 O sistema deverá permitir que o usuário bloqueie chamadas provenientes de números classificados como suspeitos ou spam, de acordo com sua vontade.

RF11 O sistema deverá permitir que o usuário desbloqueie números previamente bloqueados a qualquer momento, restabelecendo o recebimento de chamadas provenientes desses números.

2.2 Requisitos Não-Funcionais

RNF1 O sistema deve processar e responder às requisições do usuário em, no máximo, 2 segundos, garantindo uma experiência fluida e eficiente;

RNF2 O sistema deve manter uma disponibilidade mínima de 90% durante o período de operação, assegurando o acesso contínuo aos serviços essenciais;

RNF3 A arquitetura baseada em microsserviços deve permitir a expansão horizontal dos componentes, suportando o aumento no número de usuários e requisições sem degradação significativa de desempenho;

RNF4 O sistema deve utilizar autenticação baseada em *tokens JWT*, protegendo dados pessoais e credenciais dos usuários;

RNF5 A interface do aplicativo deve ser intuitiva, responsiva e acessível a usuários com diferentes níveis de familiaridade tecnológica;

RNF6 O aplicativo deve ser compatível com dispositivos Android a partir da versão 8.0 (Oreo) e com as principais resoluções de tela disponíveis no mercado;

RNF7 O código-fonte deve seguir boas práticas de desenvolvimento e padrões de arquitetura (como a arquitetura hexagonal), facilitando futuras atualizações e correções;

RNF8 O sistema deve garantir a integridade e consistência dos dados, mesmo em situações de falha parcial de algum microsserviço;

RNF9 O sistema deve ser construído utilizando o framework Spring boot com Java no back-end, linguagem Kotlin para o multiplataforma e maven para gerenciamento de dependências;

RNF10 O sistema deve ser projetado de forma modular, possibilitando a realização de testes unitários, de integração e de desempenho em cada componente da aplicação;

RNF11 Todo o código-fonte deve ser versionado em um repositório Git, garantindo rastreabilidade, histórico de alterações e colaboração entre desenvolvedores.

3 Design

3.1 Projeto UML

A modelagem do sistema HiveCall foi realizada utilizando a Linguagem de Modelagem Unificada (UML), com o objetivo de representar graficamente a estrutura, o comportamento e a organização arquitetural do software. A utilização de diagramas UML contribui para uma melhor compreensão do sistema, facilitando tanto a comunicação entre os envolvidos no projeto quanto a manutenção e evolução futura da aplicação.

Considerando as características do sistema, um backend distribuído baseado em microsserviços e uma aplicação mobile cliente, foram elaborados os seguintes diagramas:

- Diagrama de Casos de Uso;
- Diagrama de Classes;
- Diagrama de Sequência.

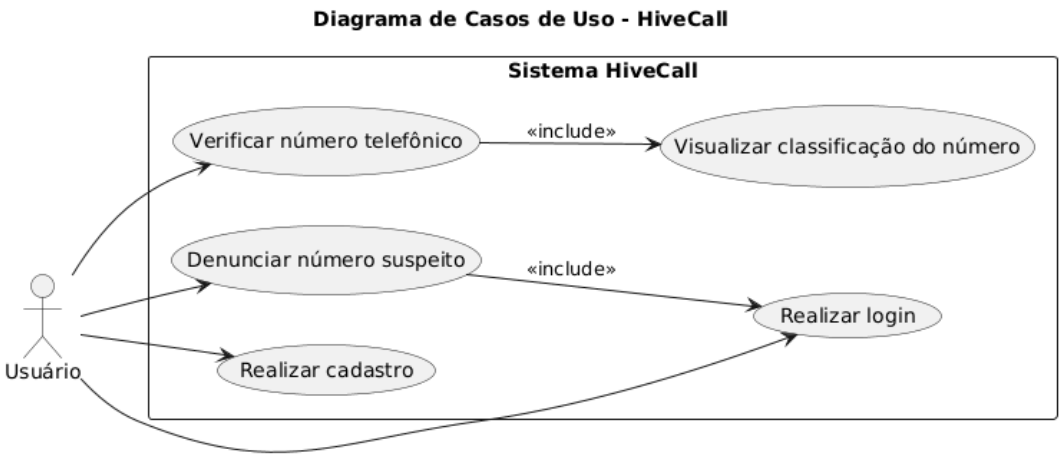

Figura 1: Diagrama de Casos de Uso

O Diagrama de Casos de Uso apresenta as principais funcionalidades disponibilizadas pelo sistema HiveCall sob a perspectiva do usuário. Ele evidencia as interações possíveis com o sistema, destacando a verificação de números telefônicos e o registro de denúncias como funcionalidades centrais, além dos mecanismos de autenticação necessários para ações restritas.

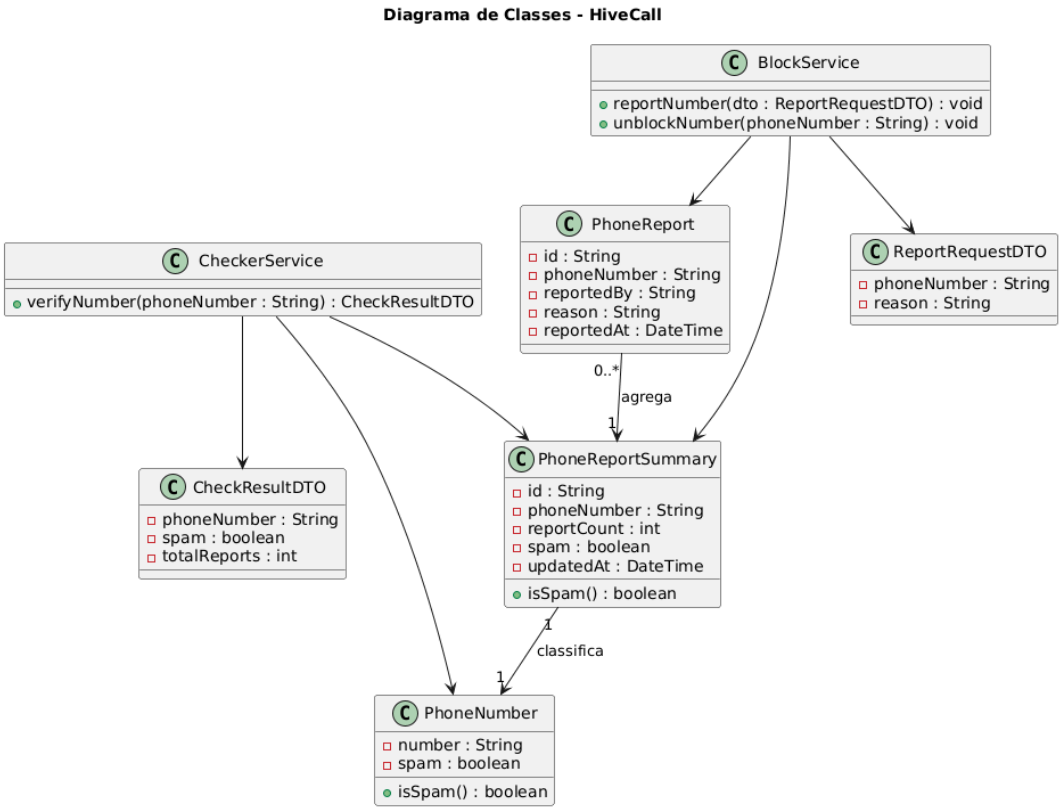

Figura 2: Diagrama de Classes

O Diagrama de Classes representa as principais estruturas do HiveCall, evidenciando

as classes de domínio responsáveis pelo registro e agregação de denúncias, bem como os serviços de aplicação que encapsulam as regras de negócio. O diagrama destaca ainda os DTOs utilizados para comunicação entre camadas, mantendo a separação entre domínio e infraestrutura.

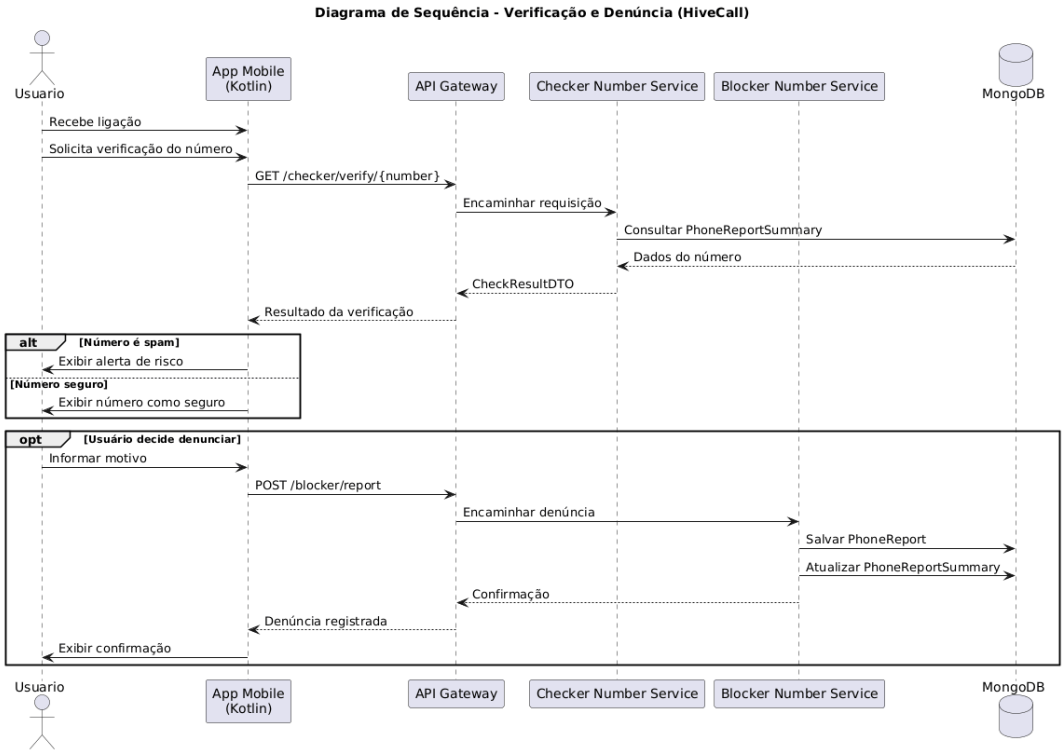

Figura 3: Diagrama de Sequência

O Diagrama de Sequência ilustra a interação entre o usuário, o aplicativo mobile e os microsserviços do backend durante a verificação de um número telefônico. O diagrama evidencia a comunicação via API Gateway e a consulta ao banco de dados, bem como o fluxo opcional de registro de denúncias.

3.2 Visão arquitetural

O HiveCall adota a Arquitetura de Microsserviços como estilo arquitetural principal, complementada pelo padrão de Arquitetura Hexagonal (Ports and Adapters) nos serviços responsáveis pela verificação de números (Checker Number Service) e pelo bloqueio ou denúncia de números (Blocker Number Service). Essa combinação foi escolhida por favorecer a escalabilidade da aplicação, a independência entre os componentes, a facilidade de manutenção e uma separação de responsabilidades mais nítida e organizada.

Na Arquitetura de Microsserviços, cada serviço do sistema possui uma responsabilidade bem definida e pode ser desenvolvido, implantado e escalado de forma independente. No HiveCall, os principais microsserviços são:

- Auth Service: responsável pela autenticação e autorização de usuários por meio do

Firebase Authentication;

- Blocker Number Service: responsável pelo registro, gerenciamento e agregação de denúncias de números telefônicos;
- Checker Number Service: responsável pela verificação pública de números telefônicos e retorno de sua classificação;
- API Gateway: que atua como ponto único de entrada do sistema, centralizando o roteamento de requisições e a validação de tokens JWT;
- Eureka Server: utilizado como mecanismo de descoberta de serviços.

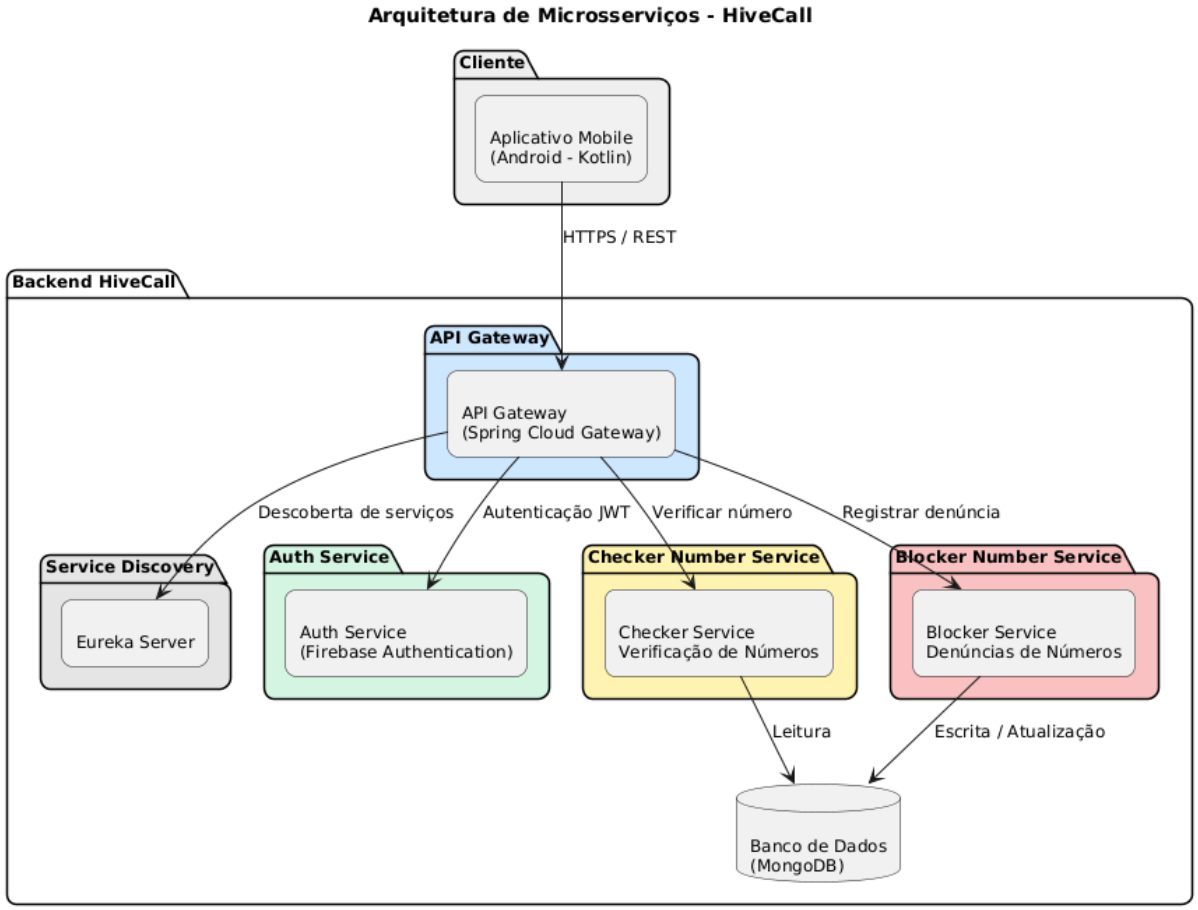

Figura 4: Diagrama Arquitetural em Microsserviços

O API Gateway desempenha um papel fundamental na arquitetura, pois desacopla o cliente mobile da topologia interna do backend, além de concentrar aspectos transversais como segurança e controle de acesso. A aplicação mobile, desenvolvida em Kotlin, interage exclusivamente com o Gateway, não possuindo acesso direto aos microsserviços internos.

Os serviços Blocker Number e Checker Number utilizam a Arquitetura Hexagonal, na qual o domínio da aplicação é isolado de detalhes de infraestrutura, como controladores REST e

acesso a banco de dados. Essa abordagem favorece a testabilidade, reduz o acoplamento entre regras de negócio e tecnologias externas e facilita a evolução do sistema ao longo do tempo.

Arquiteturalmente, o núcleo da aplicação é representado pelo domínio, que contém as entidades e DTOs responsáveis por modelar as regras de negócio. A camada de aplicação concentra os casos de uso, definidos por interfaces de entrada (*port.in*) e suas respectivas implementações (services), responsáveis por orquestrar o fluxo da lógica de negócio. As dependências externas são abstraídas por meio de portas de saída (*port.out*), que definem contratos para acesso a dados e integrações externas. As implementações concretas dessas portas encontram-se na camada de infraestrutura, por meio de adapters que utilizam tecnologias como Spring Data, controladores REST e outros mecanismos de integração.

Do ponto de vista técnico, o backend foi desenvolvido utilizando Java, Spring Boot e Spring Cloud. O banco de dados adotado é o MongoDB, cuja escolha se deu por sua flexibilidade de esquema e boa performance em cenários com alto volume de leitura.

Essa combinação arquitetural permite que o HiveCall seja uma aplicação escalável, modular e resiliente, adequada ao contexto de sistemas distribuídos modernos. Abaixo seguem exemplos da implementação da arquitetura hexagonal:

```java
public interface CheckerUseCase {
    CheckResultDTO verifyNumber(String number);
}
```

Listing 1: Interface de caso de uso

```java
public interface PhoneNumberRepositoryPort {
    Optional<PhoneReportSummary> findByPhoneNumber(String phoneNumber);
}
```

Listing 2: Porta de saída que abstrai o acesso a dados

```java
public class CheckResultDTO {
    private final String phoneNumber;
    private final boolean spam;
    private final int totalReports;

    public CheckResultDTO(String phoneNumber, boolean spam, int
        totalReports) {
        this.phoneNumber = phoneNumber;
        this.spam = spam;
        this.totalReports = totalReports;
    }

    public String getPhoneNumber() { return phoneNumber; }
    public boolean isSpam() { return spam; }
    public int getTotalReports() { return totalReports; }
}
```

Listing 3: DTO do domínio utilizado pelos casos de uso

```java
@Component
public class PhoneReportSummaryRepositoryAdapter implements
    PhoneNumberRepositoryPort {

    private final PhoneReportSummaryRepository repository;

    public
        PhoneReportSummaryRepositoryAdapter(PhoneReportSummaryRepository
        repository) {
        this.repository = repository;
    }

    @Override
    public Optional<PhoneReportSummary> findByPhoneNumber(String
        phoneNumber) {
        return repository.findByPhoneNumber(phoneNumber);
    }
}
```

Listing 4: Adapter da camada de infraestrutura que implementa a porta de saída

```java
@RestController
public class CheckerController {

    private final CheckerUseCase checkerUseCase;

    public CheckerController(CheckerUseCase checkerUseCase) {
        this.checkerUseCase = checkerUseCase;
    }

    @GetMapping("/check/{phone}")
    public CheckResultDTO check(@PathVariable String phone) {
        return checkerUseCase.verifyNumber(phone);
    }
}
```

Listing 5: Controller responsável por expor a API e consumir o caso de uso

```java
public class CheckerServiceUnitTest {

    @Test
    public void shouldReturnSummaryWhenFound() {
        PhoneNumberRepositoryPort repoMock =
            Mockito.mock(PhoneNumberRepositoryPort.class);

        PhoneReportSummary summary = PhoneReportSummary.builder()
                .phoneNumber("55")
                .reportCount(23)
                .spam(true)
                .updatedAt(LocalDateTime.now())
                .build();
```

```
13
14          Mockito.when(repoMock.findByPhoneNumber("55"))
15                  .thenReturn(Optional.of(summary));
16
17          CheckerService service = new CheckerService(repoMock);
18          CheckResultDTO result = service.verifyNumber("55");
19
20          assertThat(result.getPhoneNumber()).isEqualTo("55");
21          assertThat(result.isSpam()).isTrue();
22          assertThat(result.getTotalReports()).isEqualTo(23);
23      }
24  }
```

Listing 6: Exemplo de teste unitário isolando o serviço por meio de mock da porta de saída

3.3 Modelo de Banco de Dados

O HiveCall utiliza o MongoDB, um banco de dados NoSQL orientado a documentos, para armazenar as informações relacionadas às denúncias e classificações de números telefônicos. Apesar de não empregar um modelo relacional tradicional, foi elaborado um modelo de dados lógico e físico para representar as entidades principais do sistema e suas relações conceituais.

3.3.1 Modelo Lógico de Dados

No nível lógico, o sistema é composto pelas seguintes entidades principais:

- Usuário: representa o colaborador que realiza denúncias de números suspeitos. No sistema, a autenticação é realizada externamente via Firebase, sendo o usuário identificado principalmente por seu endereço de e-mail;
- Número de Telefone: entidade conceitual que representa o número telefônico a ser classificado;
- PhoneReport: representa uma denúncia individual realizada por um usuário, contendo informações como motivo da denúncia e data/hora do registro;
- PhoneReportSummary: representa o sumário agregado de denúncias de um número telefônico, armazenando o total de reports, o status de spam e um índice de confiança.

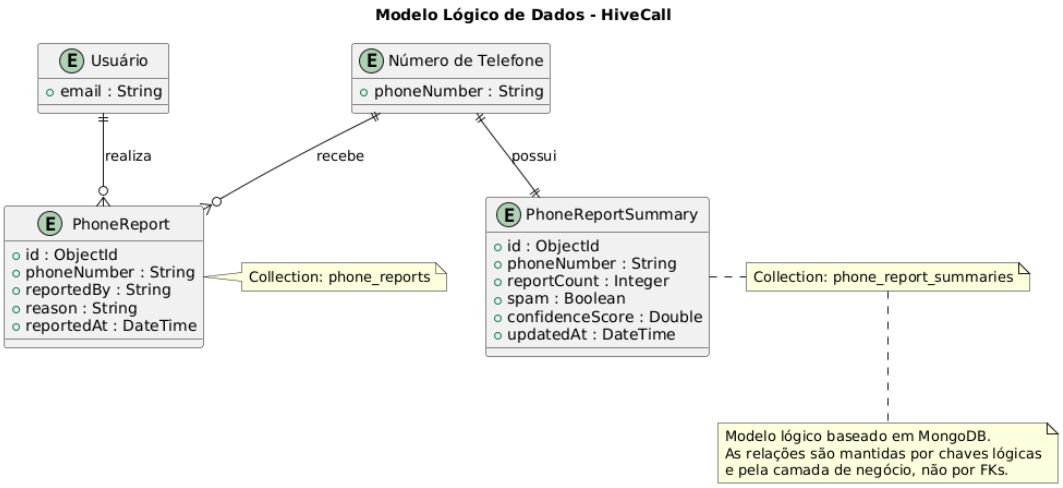

Figura 5: Modelo Lógico de Dados

As relações entre essas entidades são de natureza lógica, uma vez que o MongoDB não utiliza chaves estrangeiras. Um usuário pode realizar múltiplos reports, um número telefônico pode receber múltiplas denúncias, e cada número possui exatamente um sumário agregado.

3.3.2 Modelo Físico de Dados

No nível físico, os dados são armazenados no banco hivecall, utilizando as seguintes collections:

- phone_reports: armazena os reports individuais realizados pelos usuários;
- phone_report_summaries: armazena os dados agregados por número telefônico, otimizando consultas de verificação.

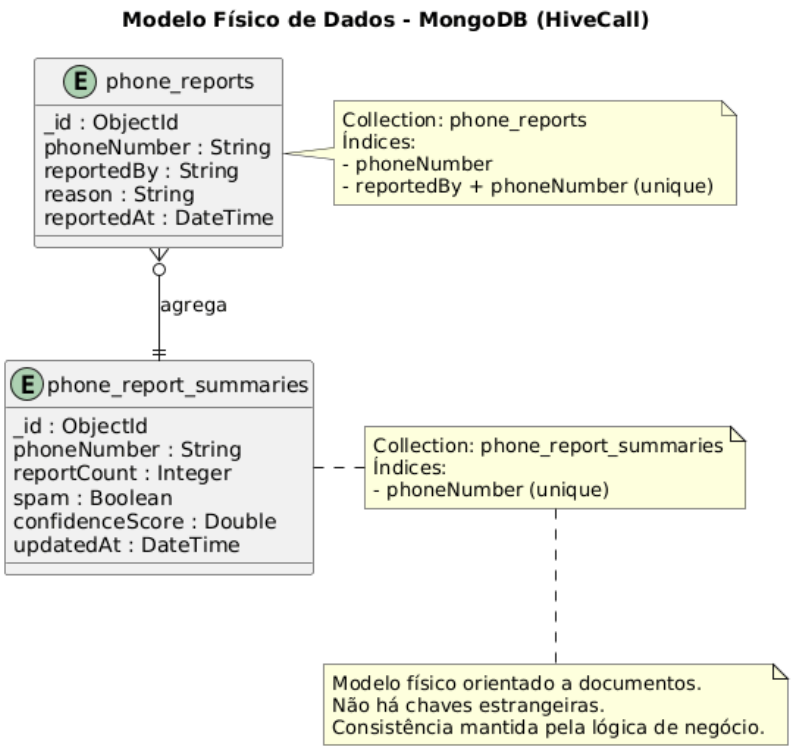

Figura 6: Modelo Físico de Dados

A integridade dos dados e a consistência entre reports individuais e seus respectivos sumários são garantidas pela lógica de negócio implementada no serviço Blocker Number, que atualiza automaticamente o sumário sempre que um *report* é criado ou removido. A escolha do MongoDB se mostrou adequada ao contexto do HiveCall, pois permite flexibilidade na estrutura dos documentos, facilidade de escalabilidade horizontal e desempenho eficiente em consultas frequentes, características essenciais para um sistema colaborativo e de consulta em tempo real.

4 Testes de Software

4.1 Projeto de Testes

Com o objetivo de assegurar a qualidade e a confiabilidade do sistema HiveCall, foram adotadas práticas de testes alinhadas às boas práticas de Engenharia de Software. Os testes foram aplicados principalmente sobre os microsserviços que compõem o backend da aplicação, garantindo que as regras de negócio e os fluxos principais operassem conforme o esperado.

No contexto da aplicação mobile multiplataforma, a validação dessa camada foi realizada por meio de testes manuais, utilizando emuladores com o objetivo de verificar o funcionamento correto da interface, a integração com o backend e a experiência do usuário. Adicionalmente, foram implementados testes automatizados com foco na validação de inputs de dados e na verificação de cenários em classificações de números durante chamadas telefônicas.

A estratégia de testes do backend foi dividida em dois cenários principais. No primeiro cenário, foram utilizadas ferramentas como o Postman e a documentação Swagger para a realização de testes manuais, possibilitando a validação dos endpoints expostos pelos microsserviços, bem como o fluxo completo das requisições.

No segundo cenário, foram desenvolvidos testes automatizados, com foco em testes unitários, considerados fundamentais para garantir a qualidade do código, prevenir erros e facilitar a manutenção do sistema. Os testes unitários permitem validar pequenas partes do código de forma isolada, como métodos e serviços, possibilitando a criação de múltiplos cenários de teste para uma mesma funcionalidade.

4.2 Exemplo de Teste Unitário

Abaixo é possível verificar um exemplo de teste unitário aplicado ao serviço de verificação de números do microsserviço *checker-number*. O teste valida dois cenários distintos: quando existe um resumo de denúncias previamente registrado para um número telefônico e quando não existe qualquer registro, garantindo que o serviço retorne os valores esperados em ambas as situações.

```
@ExtendWith(MockitoExtension.class)
public class CheckerServiceTest {

    @Mock
    private PhoneReportSummaryRepository summaryRepository;

    @Mock
    private PhoneNumberRepositoryPort repositoryPort;

    private CheckerService service;

    @BeforeEach
    public void setup() {
        service = new CheckerService(repositoryPort);
        ReflectionTestUtils.setField(service, "repository",
            summaryRepository);
    }

    @Test
    public void verifyNumberWhenSummaryExistsShouldReturnSpamInfo() {
        PhoneReportSummary summary = PhoneReportSummary.builder()
                .phoneNumber("123")
                .reportCount(21)
                .spam(true)
                .confidenceScore(1.0)
                .updatedAt(LocalDateTime.now())
                .build();

        given(summaryRepository.findByPhoneNumber("123"))
                .willReturn(Optional.of(summary));
```

```
30
31          CheckResultDTO result = service.verifyNumber("123");
32
33          assertThat(result.getPhoneNumber()).isEqualTo("123");
34          assertThat(result.isSpam()).isTrue();
35          assertThat(result.getTotalReports()).isEqualTo(21);
36      }
37
38      @Test
39      public void verifyNumberWhenNoSummaryShouldReturnDefault() {
40          given(summaryRepository.findByPhoneNumber("999"))
41                  .willReturn(Optional.empty());
42
43          CheckResultDTO result = service.verifyNumber("999");
44
45          assertThat(result.getPhoneNumber()).isEqualTo("999");
46          assertThat(result.isSpam()).isFalse();
47          assertThat(result.getTotalReports()).isEqualTo(0);
48      }
49  }
```

Listing 7: Teste unitário do serviço de verificação de números

5 Implantação

5.1 Projeto de Implantação

O sistema HiveCall foi projetado para funcionar em ambientes de computação em nuvem ou em servidores dedicados, utilizando uma arquitetura baseada em microsserviços. Para a implantação do backend, é necessário um servidor com sistema operacional Linux, Java Development Kit (JDK) versão 17 e MongoDB como sistema de banco de dados.

A aplicação mobile, desenvolvida de forma multiplataforma, executa em dispositivos móveis com sistema operacional Android ou iOS e se comunica com o backend por meio de APIs REST. A validação da aplicação mobile foi realizada através de testes manuais em emuladores e dispositivos físicos, garantindo a correta integração com os microsserviços.

6 Resultados Empíricos

Este capítulo apresenta os resultados obtidos a partir do desenvolvimento e da testagem do aplicativo *HiveCall*, cujo objetivo é identificar e classificar chamadas telefônicas como seguras, suspeitas ou spam, utilizando uma base de dados colaborativa.

6.1 Metodologia de Avaliação

A avaliação do HiveCall foi realizada por meio de testes funcionais em ambiente controlado. Foram utilizados números telefônicos fictícios e números conhecidos por realizarem chamadas

suspeitas, além de denúncias simuladas feitas por usuários de teste.

Os testes foram realizados em dispositivos Android, utilizando tanto emuladores quanto aparelhos físicos. O backend permaneceu ativo durante todo o processo, garantindo a consistência das respostas retornadas ao aplicativo mobile.

6.2 Coleta e Processamento dos Dados

Durante os testes, o sistema analisou as chamadas recebidas e realizou consultas à base de dados para verificar a existência de registros associados aos números. Para cada chamada, foram considerados:

- a existência de registros prévios do número;
- a quantidade de denúncias associadas;
- a classificação atribuída pelo sistema.

O processamento ocorre no momento da chamada, permitindo que o usuário receba um aviso imediato sobre o possível risco antes de atender a ligação.

6.3 Resultados Obtidos

Os resultados demonstraram que o HiveCall conseguiu identificar corretamente números previamente denunciados, exibindo alertas antes do atendimento da chamada. Números com a quantidade mínima de denúncias foram classificados como suspeitos. Observou-se também que a classificação dos números foi atualizada conforme novas denúncias foram sendo registradas, evidenciando o caráter colaborativo da aplicação.

A análise dos resultados indica que a solução proposta é eficaz para alertar os usuários sobre chamadas potencialmente perigosas, sem realizar bloqueios automáticos que possam gerar erros ou impactos negativos na experiência do usuário. Essa abordagem contribui para a redução de falsos positivos e oferece um maior controle ao usuário.

Além disso, a aplicação demonstrou eficiência nos processos de bloqueio e desbloqueio de números, realizados de forma manual pelo usuário. Durante os testes conduzidos em ambiente controlado, todos os números previamente denunciados puderam ser bloqueados e desbloqueados corretamente, sem a ocorrência de falhas funcionais.

6.4 Considerações Finais

Os resultados obtidos demonstram que o HiveCall cumpre o seu objetivo de classificar chamadas telefônicas de forma eficiente e de bloquear números caso o usuário deseje. O sistema se mostrou funcional e consistente, atendendo aos requisitos deste trabalho acadêmico, com possibilidade de evolução em estudos futuros.

7 Manual do Usuário

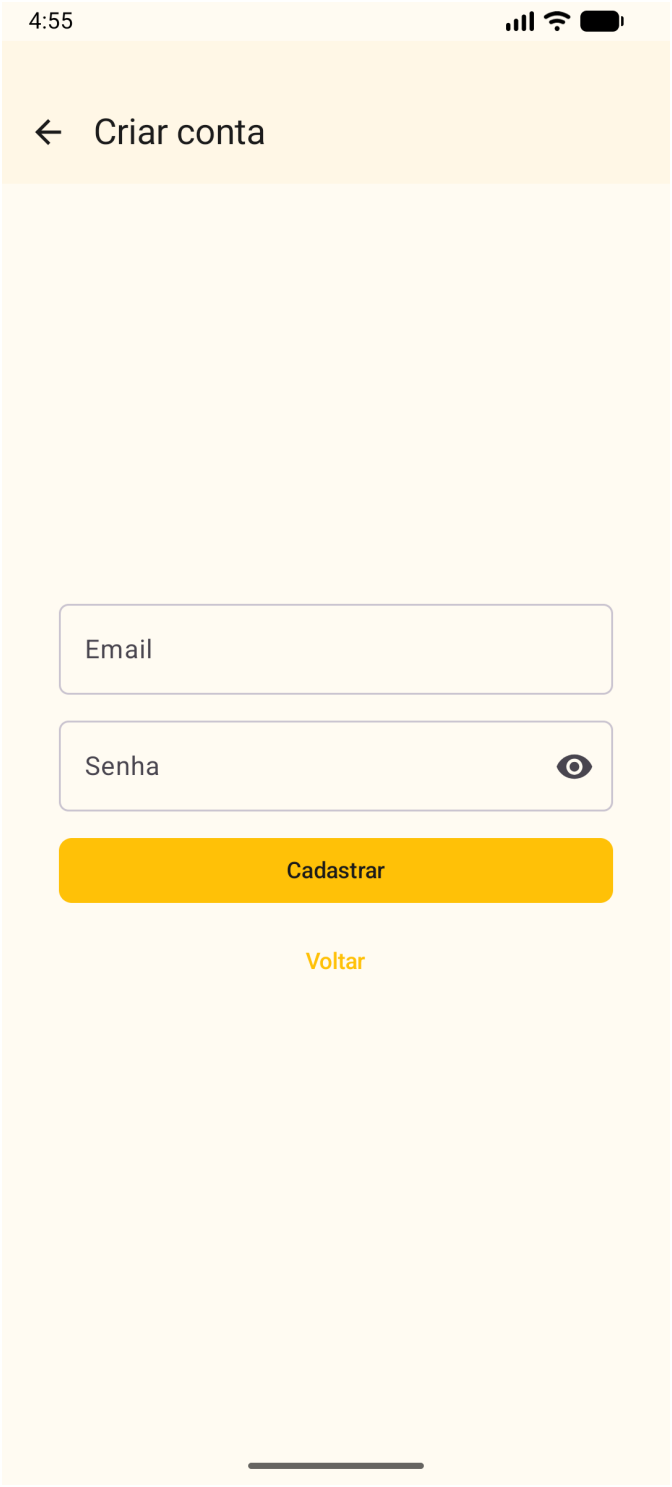

Figura 7: Criar nova conta

Para criar uma nova conta e acessar o aplicativo, o usuário deve informar um endereço de e-mail válido e uma senha nos campos correspondentes e, em seguida, selecionar a opção “Cadastrar”.

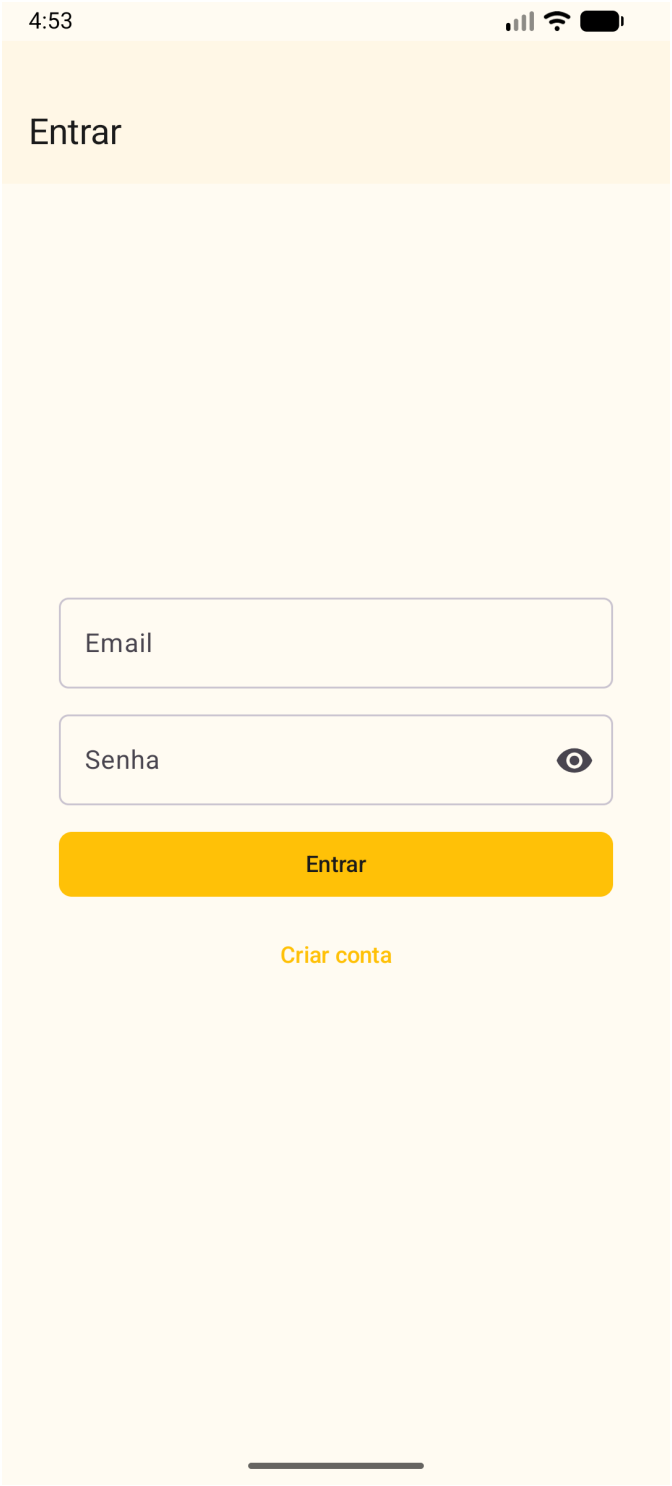

Figura 8: Criar nova conta

Para acessar a uma conta já existente, o usuário deve informar o endereço de e-mail e a senha previamente cadastrados e, em seguida, clicar no botão “Entrar” para realizar a autenticação no Hivecall.

Figura 9: Listar números

Após a realização do login com sucesso, o usuário poderá visualizar a lista de números telefônicos bloqueados associados à sua conta, bem como realizar ações de desbloqueio ou adicionar novos números à lista de bloqueio.

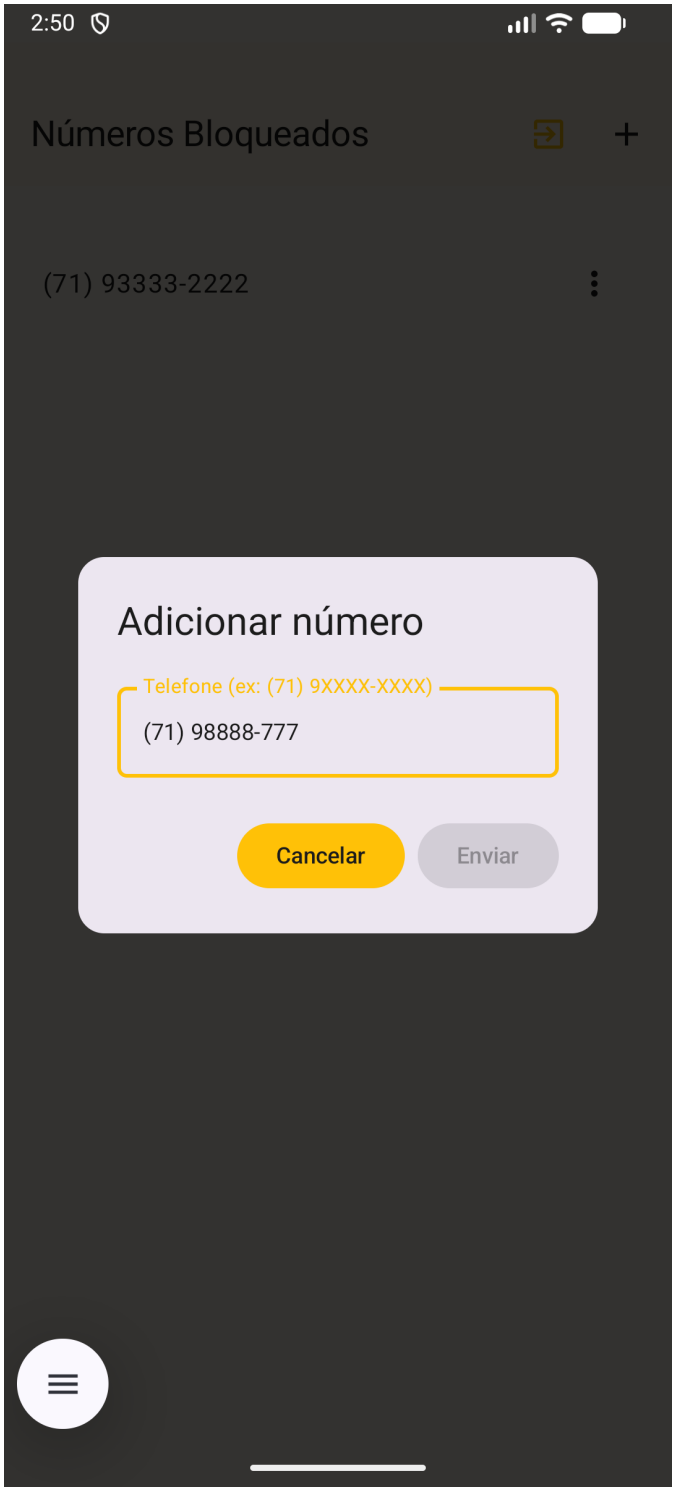

Figura 10: Bloquear número

Ao clicar no botão “+”, localizado no canto superior direito da tela, o usuário pode realizar o bloqueio de novos números telefônicos. Para garantir uma melhor experiência de usuário e a padronização dos números cadastrados, todos eles devem seguir o padrão brasileiro de comunicação: (XX) XXXXX-XXXX, cujo é automaticamente formatado durante a digitação. Após a inserção correta do número, o botão “Enviar” é habilitado, permitindo o envio das informações para a API e o armazenamento no banco de dados.

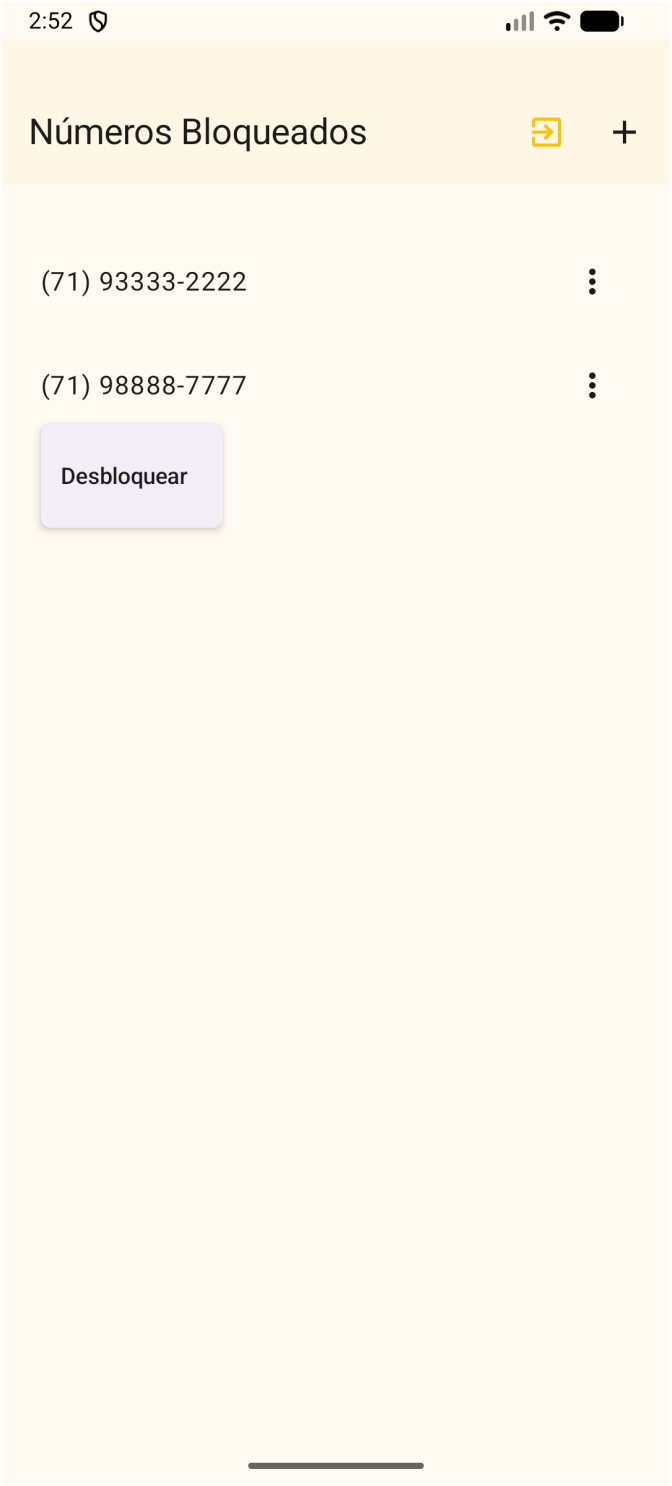

Figura 11: Desbloquear número

Na tela de listagem de números, também é possível realizar o desbloqueio de números previamente bloqueados. Para isso, basta clicar no ícone de três pontos localizado ao final do número desejado e, em seguida, selecionar a opção “Desbloquear”. O sistema efetuará automaticamente o desbloqueio do número selecionado.

Agradecimentos

Agradeço primeiramente à Deus, fonte de força e de inspiração para mim, cujo sem, nada seria. Sempre gostei de ficar no computador e, graças à Ele, tive a oportunidade de ingressar em um curso que justamente me identifico e gosto de trabalhar.

À minha mãe, que sempre foi tudo para mim, me oferecendo apoio incondicional, incentivo e muitas vezes realizando sacrifícios para que eu pudesse chegar até aqui. Mãe, graças a você, hoje posso realizar este trabalho.

À minha namorada por todo incentivo, carinho e amor nos momentos difíceis. Você sempre me motivou a seguir em frente e a nunca desistir. O seu apoio foi incondicional durante essa trajetória e certamente será em todas as fases da minha vida.

Aos professores e docentes do IFBA, em especial ao meu orientador Manoel Neto e à coordenadora do curso, Flávia Maristela, pelo compartilhamento de conhecimentos e pela dedicação que, juntos, me permitiram iniciar a trajetória profissional.

Aos meus familiares, amigos e colegas que, juntos, tornaram essa trajetória mais divertida e fácil, permitindo viver muitos momentos bons, de aprendizados e de companheirismo ao longo do curso.

Referências

1 G1 Fantástico. De testes a golpes: brasileiros recebem 10 bilhões de ligações feitas por robôs por mês. 2025. Disponível em: ⟨https://g1.globo.com/fantastico/noticia/2025/04/27/de-testes-a-golpes-brasileiros-recebem-10-bilhoes-de-ligacoes-feitas-por-robos-por-mes.ghtml⟩.

2 ANATEL. **Relatório sobre chamadas indesejadas e fraudulentas**. [S.l.], 2023.

3 CERT.br. **Cartilha de Segurança para Internet**. [S.l.], 2023.

4 CGI.br. **Privacidade e Proteção de Dados Pessoais**. [S.l.], 2022.

5 NIC.br. **Segurança na Internet no Brasil**. [S.l.], 2022.

6 Truecaller. **Truecaller: Identifique e bloqueie chamadas de spam**. 2025. Disponível em: ⟨https://www.truecaller.com/pt-br⟩.

7 Hiya. **Hiya: Proteção contra chamadas indesejadas e identificação de spam**. 2025. Disponível em: ⟨https://pt.hiya.com/⟩.

8 Whoscall. **Whoscall: Identificador e bloqueador de chamadas desconhecidas**. 2025. Disponível em: ⟨https://whoscall.com/pt⟩.