

**UNIVERSIDADE ESTADUAL PAULISTA "JÚLIO DE MESQUITA FILHO"**  
FACULDADE DE CIÊNCIAS - CAMPUS BAURU  
DEPARTAMENTO DE COMPUTAÇÃO  
BACHARELADO EM SISTEMAS DE INFORMAÇÃO

BRUNO JOSÉ PAPA

**APLICATIVO PARA PUBLICAÇÃO E ACOMPANHAMENTO DE  
ALERTAS DE SEGURANÇA PÚBLICA**

BAURU  
Março/2022

BRUNO JOSÉ PAPA

**APLICATIVO PARA PUBLICAÇÃO E ACOMPANHAMENTO DE  
ALERTAS DE SEGURANÇA PÚBLICA**

Trabalho de Conclusão de Curso do Curso de Bacharelado em Sistemas de Informação da Universidade Estadual Paulista “Júlio de Mesquita Filho”, Faculdade de Ciências, Campus Bauru.

Orientador: Profa. Dra. Roberta Spolon

BAURU  
Março/2022

Bruno José Papa   Aplicativo para publicação e acompanhamento de alertas de segurança pública/ Bruno José Papa. – Bauru, Março/2022-    29 p. : il.  
(algumas color.) ; 30 cm.

Orientador: Profa. Dra. Roberta Spolon

Trabalho de Conclusão de Curso – Universidade Estadual Paulista “Júlio de Mesquita Filho”

Faculdade de Ciências  
Sistemas de Informação, Março/2022.  
1. Tags 2. Para 3. A 4. Ficha 5. Catalográfica

Bruno José Papa

## **Aplicativo para publicação e acompanhamento de alertas de segurança pública**

Trabalho de Conclusão de Curso do Curso de Bacharelado em Sistemas de Informação da Universidade Estadual Paulista "Júlio de Mesquita Filho", Faculdade de Ciências, Campus Bauru.

Banca Examinadora

---

**Profa. Dra. Roberta Spolon**

Orientador

Departamento de Computação  
Faculdade de Ciências  
Universidade Estadual Paulista "Júlio de  
Mesquita Filho"

---

**Prof. Dr. José Remo Ferreira Brega**

Departamento de Computação  
Faculdade de Ciências  
Universidade Estadual Paulista "Júlio de  
Mesquita Filho"

---

**Profa. Dra. Márcia A. Zanolí Meira e  
Silva**

Departamento de Computação  
Faculdade de Ciências  
Universidade Estadual Paulista "Júlio de  
Mesquita Filho"

Bauru, \_\_\_\_\_ de \_\_\_\_\_ de \_\_\_\_\_.  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

# Resumo

O presente trabalho apresenta a análise, projeto, implementação e avaliação de uma aplicação móvel para a publicação e o acompanhamento de alertas relacionados à segurança pública no Brasil, o qual é capaz de manter as pessoas conscientes de eventuais perigos dos seus arredores em tempo real. Essas informações, provenientes pelas comunidades locais, podem ser cruciais para que as pessoas busquem se proteger e proteger uns aos outros. É exposto que a tecnologia, aliada com a colaboração entre usuários pela internet, pode também ser uma importante ferramenta de segurança. As principais funcionalidades do aplicativo desenvolvido são: rastreamento da localização do usuário, publicação de alertas com suporte ao envio de imagens e vídeos, e o disparo de notificações baseados em localização. É importante ressaltar que este trabalho não estimula que a justiça seja feita com as próprias mãos, uma vez que isso é responsabilidade dos órgãos públicos locais.

**Palavras-chave:** aplicação móvel, sistema colaborativo, engenharia de software

# Abstract

The current project presents the analyze, design, implementation and evaluation of a mobile application for the publication and monitoring of alerts related to public safety in Brazil, which is capable of keeping people aware of possible dangers in their surroundings in real time. This informations, provided by local communities, can be crucial for people to seek to protect themselves and each other. It is exposed that the technology, combined with collaboration between users over the internet, can also be an important security tool. The main features of the developed application are: user's location tracking, publishing alerts with images and videos uploads suports, and triggering location-based notifications. It is important to emphasize that this work does not encourage justice to be done with their own hands, since this is the responsibility of local public bodies.

**Keywords:** mobile application, collaborative system, software engineering.

# Listas de figuras

Figura 1 – Telas do aplicativo Citizen.	13
Figura 2 – Telas do aplicativo SP+ segura.	14
Figura 3 – Componentes físicos do sistema.	17
Figura 4 – Visão geral do sistema.	18
Figura 5 – Camadas de um módulo do servidor.	19
Figura 6 – Diagrama de classes da camada de domínio do módulo compartilhado do servidor.	20
Figura 7 – Entidades de domínio de cada módulo do servidor.	22
Figura 8 – Entradas e saídas de cada módulo do servidor.	23
Figura 9 – Interação detalhada entre cliente e servidor.	24

# Listas de abreviaturas e siglas

AWS	Amazon Web Services
HTTP	Hypertext Transfer Protocol
API	Application Programming Interface
DTO	Data Transfer Object
IaC	Infrastructure as Code
ORM	Object-Relational Mapping
SDK	Software Development Kit
REST	Representational State Transfer

# Sumário

<b>1</b>	<b>INTRODUÇÃO</b>	<b>10</b>
1.1	<b>Motivação</b>	<b>10</b>
1.2	<b>Solução</b>	<b>10</b>
1.3	<b>Desafios</b>	<b>11</b>
<b>2</b>	<b>SOLUÇÕES EXISTENTES E OBJETIVOS DA PROPOSTA</b>	<b>12</b>
2.1	<b>Soluções existentes</b>	<b>12</b>
2.1.1	Citizen	12
2.1.2	SP+ segura	12
2.1.3	Análise	12
2.2	<b>Objetivos</b>	<b>14</b>
<b>3</b>	<b>DETALHAMENTO DO PROBLEMA</b>	<b>15</b>
3.1	<b>Natureza do problema</b>	<b>15</b>
3.2	<b>Requisitos</b>	<b>15</b>
3.2.1	Requisitos funcionais	15
3.2.2	Requisitos não funcionais	15
<b>4</b>	<b>DESCRIÇÃO DO SISTEMA</b>	<b>17</b>
4.1	<b>Servidor</b>	<b>17</b>
4.1.1	Camadas de cada módulo	18
4.1.1.1	A camada de domínio	19
4.1.1.2	A camada de aplicação	20
4.1.1.3	A camada de adaptadores	21
4.1.1.4	A camada de infraestrutura	21
4.1.2	Módulos	21
4.1.3	Principais tecnologias	23
4.2	<b>Comunicação entre servidor e cliente</b>	<b>23</b>
4.2.0.0.1	GraphQL	23
4.2.0.0.2	Do cliente ao módulo do servidor	23
4.3	<b>Aplicativo</b>	<b>24</b>
4.3.1	Principais tecnologias	24
4.3.1.1	React Native	24
4.3.1.2	Relay	24
4.3.1.3	Print das telas	24

5	TECNOLOGIAS ESCOLHIDAS	25
6	CONCLUSÃO	27
	REFERÊNCIAS	28

# 1 Introdução

Os sistemas colaborativos revolucionaram a forma como as pessoas se relacionam. Eles permitem com que uma tarefa complexa seja dividida entre pequenas tarefas simples distribuídas entre várias pessoas, eles rompem as barreiras físicas para a viabilizar a conexão de pessoas até então desconhecidas entre si, eles possibilitam, através de um efeito de rede, a construção de um senso de comunidade, que é autossuficiente e orgânico.

Dentre esses sistemas, existem os sistemas conectadores, que são plataformas que fornecem toda a infraestrutura necessária para unir pessoas que são provedoras de um serviço às pessoas que são as consumidoras desse serviço, como aplicativos de transporte que permitem a busca por motoristas baseada na localização. Existem também os sistemas de colaboração em massa, do inglês *crowdsourcing*, que se beneficia ainda mais do poder da colaboração entre milhões de pessoas, como, por exemplo, o Wikipedia ([WIKIPEDIA, 2022](#)), a maior enciclopédia livre da internet.

A tecnologia, aliada à colaboração entre usuários, pode também ser uma importante ferramenta de segurança. Um grande exemplo de sucesso é o aplicativo Waze ([WAZE, 2022](#)), onde o usuário pode contribuir com alertas de acidentes de trânsito e com informações de rotas que estão congestionadas. Há, portanto, uma relação ganha-ganha para todas as partes envolvidas.

## 1.1 Motivação

O Brasil é um país onde as pessoas, sobretudo aquelas que residem em grandes centros urbanos, vivem constantemente com uma sensação de insegurança em decorrência do medo de assaltos, furtos, roubos e crimes em geral que, apesar de ser dever do Estado garantir a segurança pública, muitas vezes o tempo de resposta é muito alto. Somado à isso, catástrofes naturais são cada vez mais comuns em virtude do aquecimento global.

## 1.2 Solução

Tendo em vista a tendência do uso de sistemas facilitadores para a economia colaborativa, é apresentado neste trabalho uma solução para emponderar as comunidades locais a se protegerem e proteger uns aos outros usando a tecnologia como uma ferramenta de segurança. Por meio de um aplicativo móvel de celular, pessoas poderiam ser alertadas e alertar outras pessoas próximas sobre a ocorrência de crimes, tiroteios, incêndios, emergências, protestos, ruas interditadas, catástrofes naturais, entre outros incidentes que coloquem em risco a segurança

das pessoas.

### 1.3 Desafios

A implementação da solução possui desafios técnicos, como a implementação de um sistema de notificações e um mecanismo de rastreamento em tempo real da localização do usuário.

Além disso, a própria solução trás consigo muitos desafios. Qualquer dado que vem do usuário deve ser motivo de preocupação e, portanto, é preferível que eles sejam padronizados, validados, ou até sanitizados, se for o caso. A veracidade da informação fornecida por eles também é um risco, o usuário pode, com má intenção ou por engano, cometer injustiças sociais como calúnia, difamação, injúria racial, ou adicionar conteúdo explícito ou depreceativo. Portanto, além dos desafios técnicos, o maior desafio é o de oferecer formas para tentar mitigar esses problemas intrínsecos ao ser humano.

## 2 Soluções existentes e objetivos da proposta

### 2.1 Soluções existentes

#### 2.1.1 Citizen

O aplicativo *Citizen* ([APLICATIVO..., 2022a](#)), desenvolvido pela *sp0n, Inc.*, foi lançado originalmente em 2016 com o nome de *Vigilante* em alguns grandes centros dos Estados Unidos. Em junho de 2020, ele possuía cerca de 5 milhões de usuários ativos. Seu sucesso vem da sua robustez, usabilidade, velocidade e praticidade. Entre as principais funcionalidades destacam-se o envio de alertas de segurança baseado na localização em tempo real, o acompanhamento pelos usuários dos alertas que estão em andamento, a transmissão de vídeos ao vivo e a possibilidade de adicionar comentários. As suas notificações já ajudaram pessoas à evacuarem de prédios em chamas e ônibus escolares à escaparem de ataques terroristas.

A empresa dona do aplicativo, *sp0n, Inc.*, possui antenas de rádio nas cidades suportadas para que as chamadas telefônicas da polícia local sejam monitadoras, permitindo com que operadores especializados da empresa as filtrem e publiquem alertas no aplicativo. Em razão disso, o aplicativo *Citizen* tem sua atuação dependente dos órgãos públicos. Em agosto de 2020, ele atuava em apenas 60 cidades dos Estados Unidos.

A Figura 1 mostra algumas das principais telas do aplicativo.

#### 2.1.2 SP+ segura

O aplicativo *SP+ segura* ([APLICATIVO..., 2022b](#)) foi lançado pela Secretaria Municipal de Segurança Urbana de São Paulo em novembro de 2017. Com mais de 50 mil usuários, o aplicativo permite com que pessoas informem e sejam informadas de alertas em tempo real sobre episódios de risco em que pessoas se encontram ou presenciam.

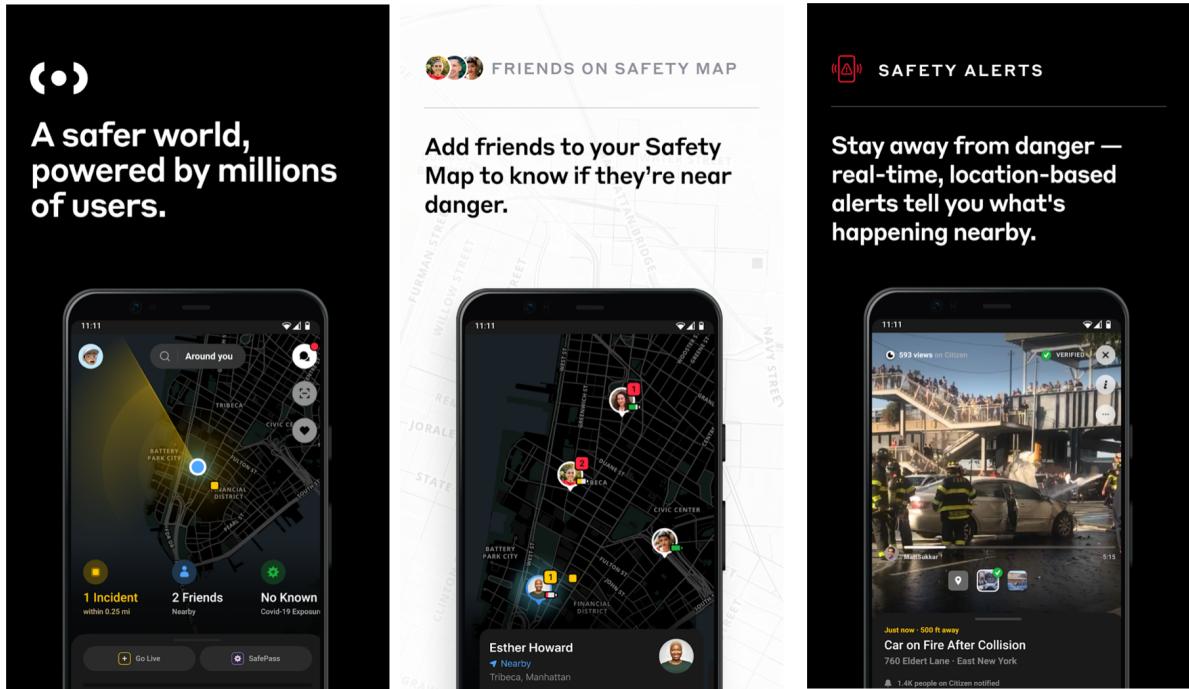
Ele oferece a opção de ligar para o orgão público responsável para acioná-lo quando for necessário. Porém, ele não oferece interação de chat entre usuários para eventuais discussões, e não suporta o envio de vídeos. Além disso, conta com muitas avaliações negativas na loja de aplicativos pelos usuários, com críticas à baixa qualidade.

A Figura 2 mostra as principais telas do aplicativo.

#### 2.1.3 Análise

Dada as soluções existentes descritas, nota-se que o *Citizen* é a maior referência do segmento no mundo. Porém, apesar de seu enorme sucesso, ele é restrito à apenas algumas

Figura 1 – Telas do aplicativo Citizen.



Fonte: Google Play.

cidades dos Estados Unidos e não apresenta previsão de expansão para o Brasil. O Brasil possui soluções para o problema, porém são limitadas, logo surgiu-se a oportunidade de oferecer uma solução alternativa.

Figura 2 – Telas do aplicativo SP+ segura.



Fonte: Google Play.

## 2.2 Objetivos

Este trabalho tem como objetivo apresentar o processo e as atividades para o desenvolvimento de uma aplicação móvel onde as pessoas possam se manter conscientes, em tempo real, de situações de perigo que eventualmente podem estar ocorrendo nas suas proximidades, como crimes, incêndios, ameaças, protestos, ruas interditadas, catástrofes naturais, entre outros. O aplicativo deve oferecer a infraestrutura para o desenvolvimento de um senso de comunidade. A integração dos alertas com os chamados das polícias locais não está incluso no escopo do projeto.

# 3 Detalhamento do problema

## 3.1 Natureza do problema

A segurança pública no Brasil é dever do Estado, porém o serviço oferecido não é eficiente. O Estado está distante do cotidiano do cidadão, o tempo de resposta dos chamados policiais muitas vezes não é rápido o bastante, entre outros.

Sendo assim, um ambiente online que ofereça a infraestrutura necessária para a construção de verdadeiras comunidades locais, onde a confiança é estabelecida com o tempo, com a ajuda de vídeos e images dos alertas que forem reportados, pode ajudar a amenizar o problema.

## 3.2 Requisitos

Dado a natureza do problema, uma solução para o problema deve atender os requisitos listados.

### 3.2.1 Requisitos funcionais

- a) Usuários devem ser capazes de se registrar e logar no sistema;
- b) Usuários devem ser capazes de publicar alertas, que devem suportar o upload de fotos e vídeos de curta duração;
- c) Usuários devem ser capazes de visualizar um mapa com os alertas mais recentes das suas proximidades;
- d) Usuários devem ser notificados quando alertas forem publicados nas suas proximidades; e
- e) Usuários devem ser capazes de definir qual o raio de proximidade em que ele está interessado, em metros.

### 3.2.2 Requisitos não funcionais

- a) o sistema deve ser seguro;
- b) o sistema deve ser altamente confiável, ou seja, qualquer foto ou vídeo carregado nunca deve ser perdido;
- c) os usuários do aplicativo devem ter uma experiência de uso em tempo real; e

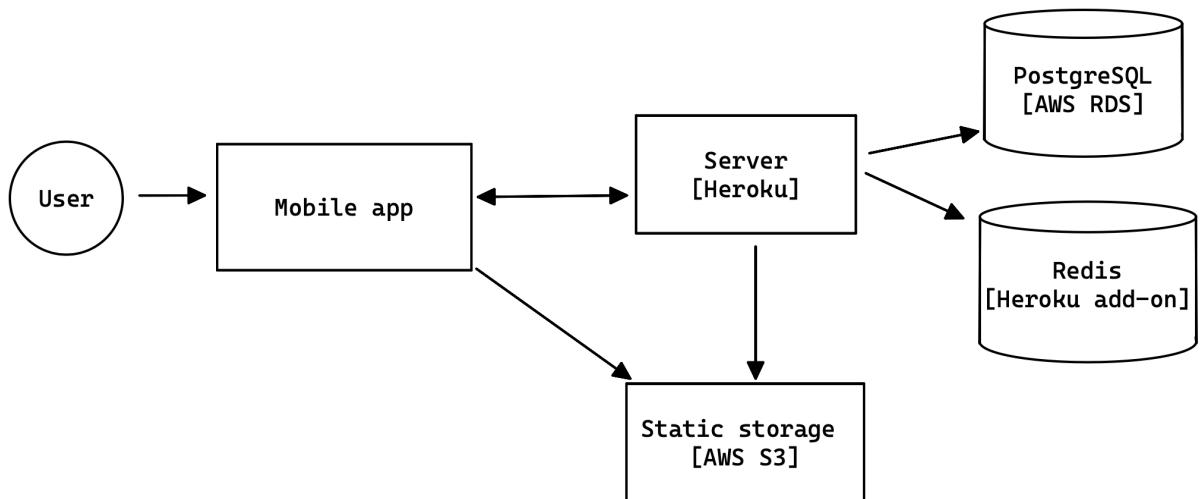
- d) o sistema deve suportar uma alta carga de leituras com latência mínima, sendo tolerável uma latência maior em escritas.

# 4 Descrição do sistema

O sistema como um todo segue uma arquitetura cliente-servidor, onde o usuário interage com uma aplicação cliente que, por sua vez, se comunica com um servidor através da internet utilizando algum protocolo de rede. A aplicação cliente, nesse caso, é um aplicativo para dispositivos móveis, enquanto o servidor é uma aplicação monolítica, ou seja, um serviço único que roda em um único processo. Foi considerado o desenvolvimento de vários microserviços, porém a arquitetura monolítica foi escolhida em virtude da simplicidade de desenvolvimento, facilidade de testes e de *deploy*.

A Figura 3 ilustra os componentes físicos do sistema.

Figura 3 – Componentes físicos do sistema.



Fonte: Elaborado pelo autor.

## 4.1 Servidor

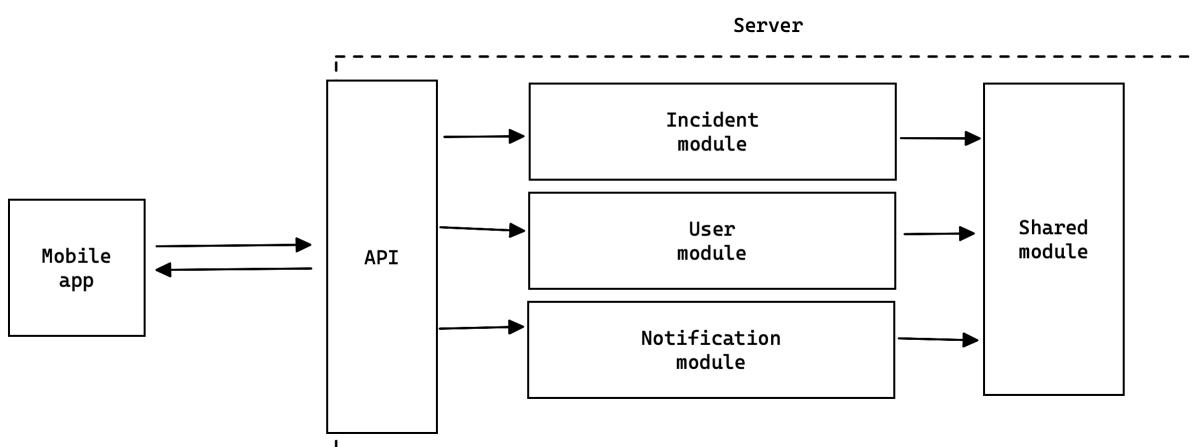
Os princípios do *domain-driven design*, termo cunhado inicialmente por Eric Evans ([EVANS, 2003](#)), e as ideias propostas por Uncle Bob em *Clean Architecture* ([BOB, 2017](#)) tiveram grande contribuição para o projeto do servidor, tendo em vista que a *separation of concerns* e a divisão da aplicação em camadas bem definidas com um rico modelo de domínio no centro permitem com que a complexidade do projeto cresça de forma manutenível ao longo do tempo.

A partir dos requisitos listados na Seção ??, se fez necessário identificar os diferentes subdomínios do sistema. Um módulo, também chamado de subdomínio, é um pedaço isolado

de código. Como ilustrado pela Figura 4, os seguintes módulos foram identificados:

- `user module`: responsável pelos usuários, gerenciamento de identidade, autenticação e autorização;
- `incident module`: responsável por todas as operações relacionadas aos alertas;
- `notification module`: responsável pelo envio de notificações aos dispositivos dos usuários;
- `shared module`: módulo global para reuso de código entre diferentes módulos.

Figura 4 – Visão geral do sistema.



Fonte: Elaborado pelo autor.

#### 4.1.1 Camadas de cada módulo

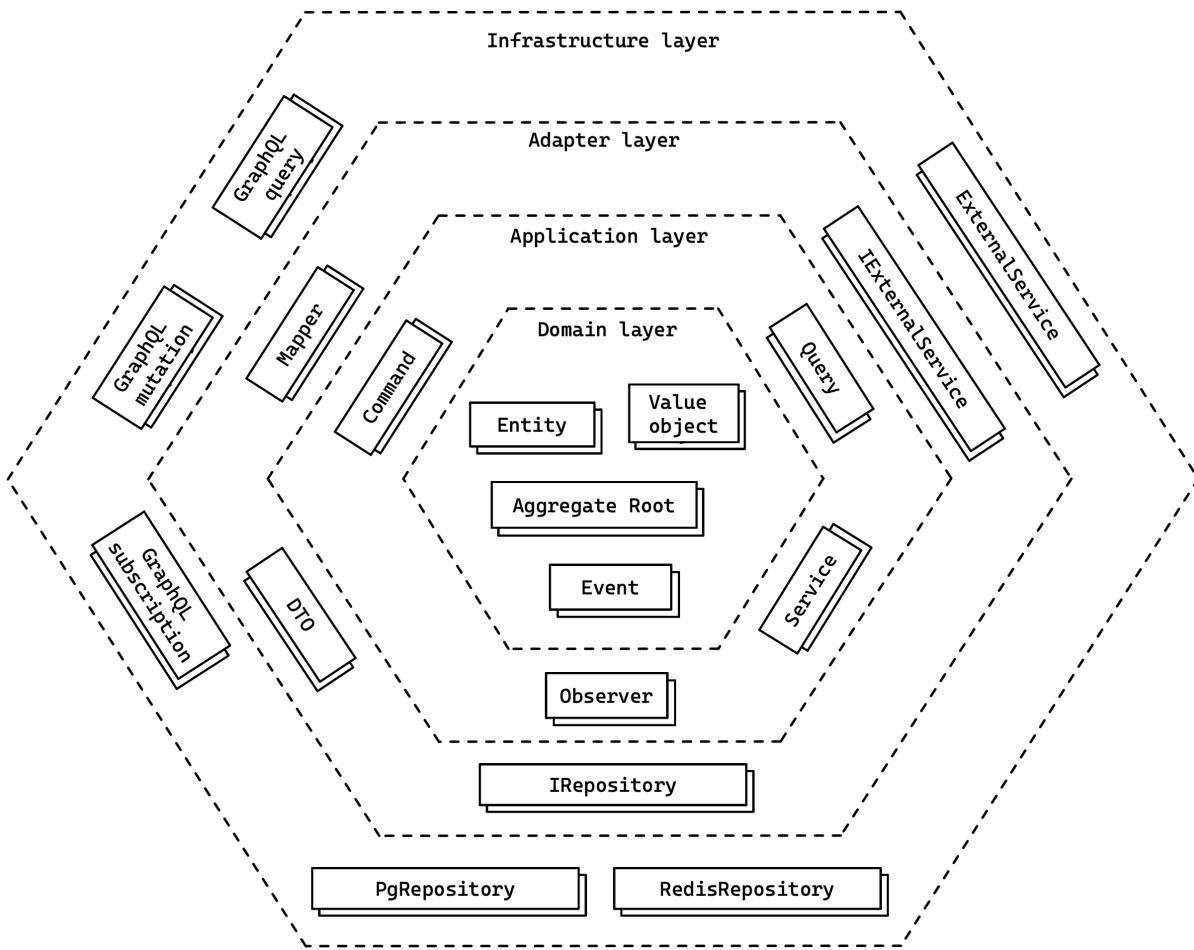
De acordo com a "arquitetura hexagonal" de Alistair Cockburn ([COCKBURN, 2005](#)), a camada mais interna da arquitetura deve manter a camada de aplicação e domínio, e fora dessa camada devem estar os adaptadores (ou portas). Os adaptadores são "uma maneira de conectar o mundo exterior ao mundo interior". A aplicação é escrita em cima das tecnologias específicas que existem no "mundo exterior", como bancos de dados, APIs externas e serviços em nuvem. Definindo adaptadores, as tecnologias específicas podem ser envolvidos com segurança pela aplicação, o que é chamado de inversão de dependência. Esse tipo de arquitetura trás inúmeros benefícios como:

- a) Atrasar a decisão sobre exatamente qual tipo de servidor web, banco de dados, serviços externos ou tecnologia de cache será escolhido até que seja absolutamente necessário decidir. Facilitando, por exemplo, que se faça uma implementação inicial em memória do bancos de dados para agilizar o desenvolvimento;

- b) Prioriza a escrita de código que pode ser facilmente testado usando injeção de dependência, minimizando o uso de dependências concretas que poderiam tornar o código não testável; e
- c) Ajusta o foco às coisas específicas da aplicação e do domínio.

Portanto, cada um dos módulos respeita a divisão em camadas descrita pela Figura 5, onde camadas mais externas dependem das camadas mais internas e, consequentemente, camadas mais internas não conhecem camadas mais externas.

Figura 5 – Camadas de um módulo do servidor.



Fonte: Elaborado pelo autor.

#### 4.1.1.1 A camada de domínio

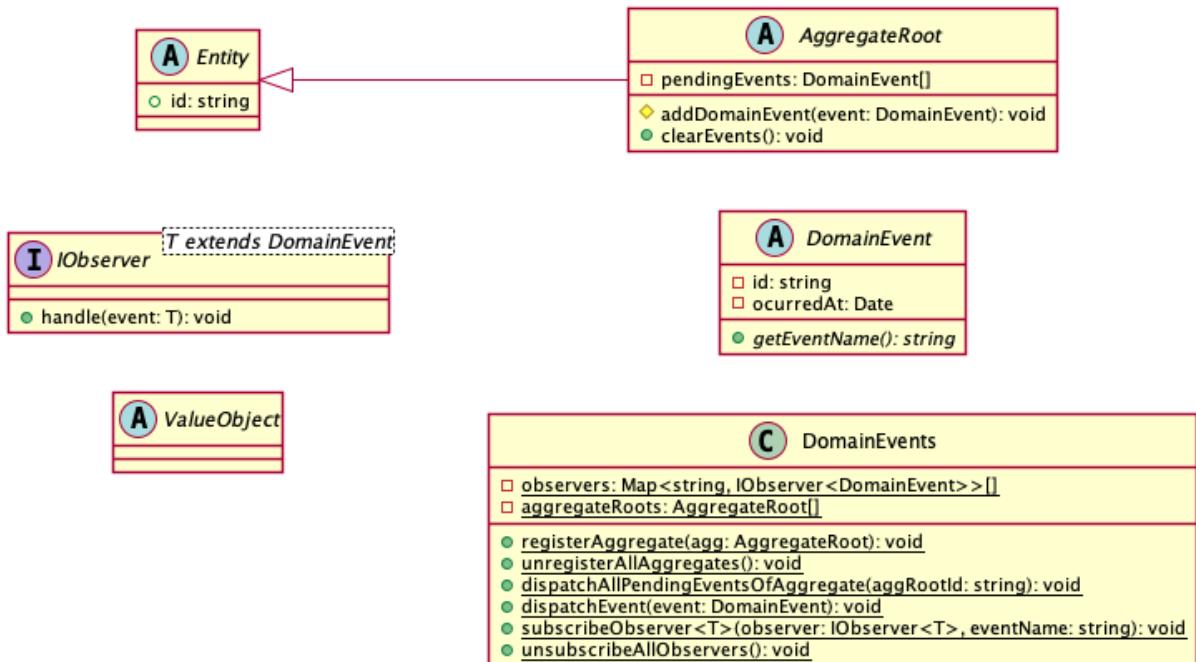
A domain layer é a camada mais interna. É a camada que contém tudo aquilo que é importante para o negócio e que está menos propensa a mudar, configurando-se como a camada mais estável dentre todas e a qual todas as outras dependem.

Para encapsular regras de validação, existem os value objects. Uma entity é um objeto que encarta um pequeno conjunto de regras de negócio.

Um aggregate root é um tipo específico de entity que pode emitir domain events quando algo relevante para o negócio ocorre, e para isso, ele armazena como estado os eventos que ainda não foram emitidos pela camada de infraestrutura. Ele é implementado pela principal entidade de um *cluster* de entities e value objects relacionados, os quais são tratados como uma única unidade de mudança.

A Figura 6 demonstra como essas classes abstratas são definidas no módulo shared, além da implementação da classe DomainEvents.

Figura 6 – Diagrama de classes da camada de domínio do módulo compartilhado do servidor.



Fonte: Elaborado pelo autor.

A classe `DomainEvents` é um *singleton*, ou seja, uma única instância global com tempo de vida vitalício em relação ao tempo de vida da aplicação. Ela é usada para encapsular o estado de quais `observers` estão interessados em ouvir pelos eventos emitidos por determinados aggregate roots. As instâncias dos aggregate roots inscritos contém domain events que serão dispatchados para seus `observers` quando a camada de infraestrutura persistir as alterações feitas.

#### 4.1.1.2 A camada de aplicação

A application layer contém os casos de uso, ou seja, principais funcionalidades, da aplicação. Em relação ao ambiente externo as entradas são os commands e queries, mas

essa camada também implementa `application services` e `observers`, que são funções que serão executadas quando um determinado evento de domínio ocorre.

O *Command-Query Separation (CQS)* é um padrão introduzido por Bertrand Meyer ([MEYER, 2000](#)) que afirma que um método é ou um `command` que executa uma ação ou uma `query` que retorna dados ao chamador, mas nunca ambos. Dessa forma os fluxos de operações que mudam o sistema (e geram efeitos colaterais) são separados daqueles que apenas requisitam dados ao sistema, tornando o código mais simples de entender e manter.

#### 4.1.1.3 A camada de adaptadores

A `adapter layer` contém abstrações para que a `application layer` possa interagir com a `infrastructure layer` sem depender dela, habilitando o que é chamado de inversão de dependência. Interfaces de repositórios que acessam bancos de dados, interfaces ou classes abstratadas que chamam APIs externas e mapeadores de objetos entre diferentes camadas (modelo de domínio, DTO, modelo de persistência) são exemplos do que pode estar nessa camada..

#### 4.1.1.4 A camada de infraestrutura

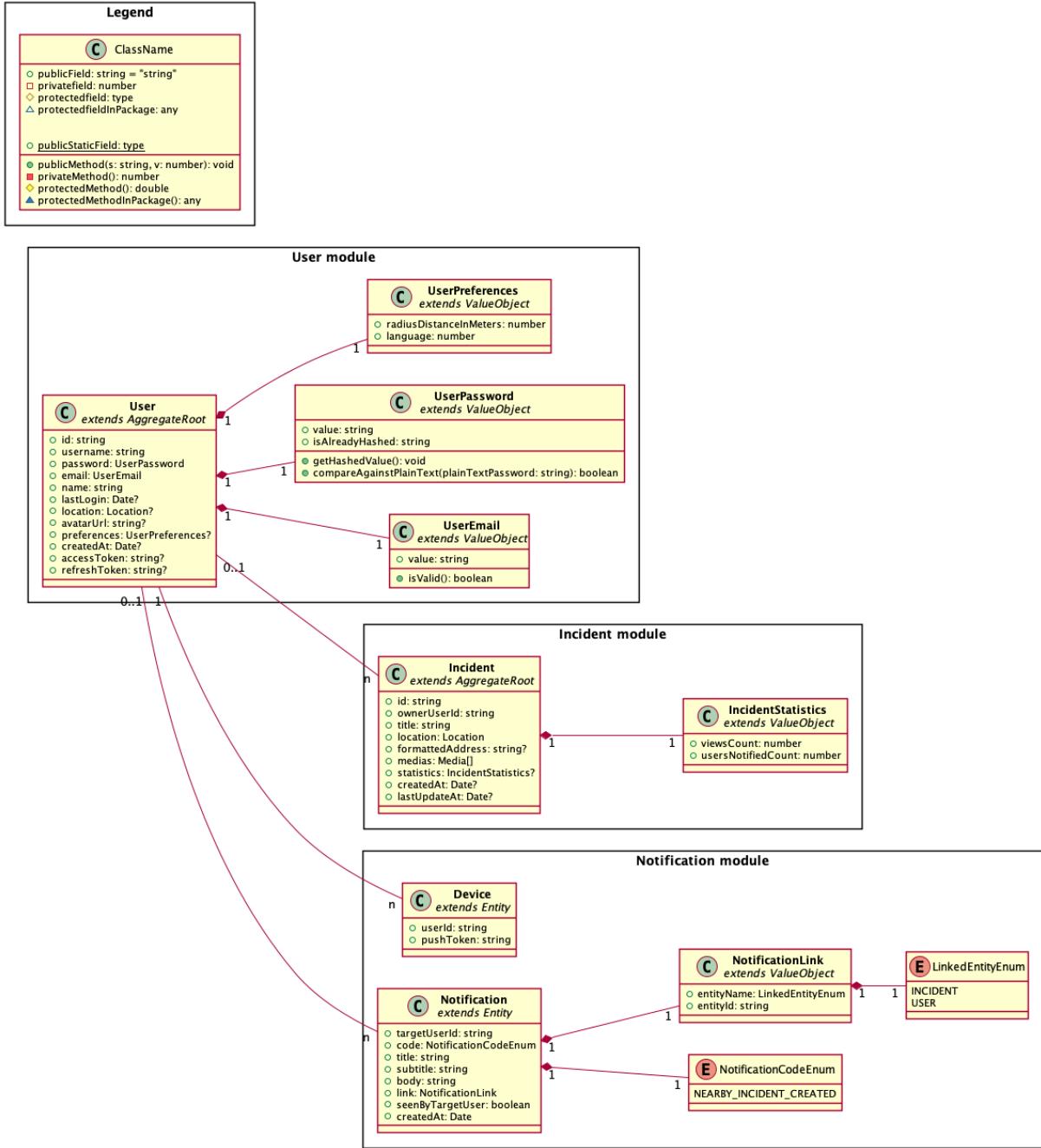
A `infrastructure layer` é a camada mais externa. Ela contém os detalhes da aplicação, os quais possuem maior chance de serem trocados por outras bibliotecas ou frameworks específicos ao decorrer do tempo. Isso inclui implementações concretas das abstrações definidas na `adapter layer` para que elas possam ser executadas em *runtime*, como serviços externos, repositórios para acesso à bancos de dados. Além disso, também contém lógicas de apresentação, como `HTTP endpoints` e `GraphQL operations`.

### 4.1.2 Módulos

Na sequência tem-se uma visão mais ampla de todos os módulos e como eles se inter-relacionam. Um módulo apenas deve se comunicar com outro via eventos de domínio, permitindo o desacoplamento entre eles.

A Figura 7 ilustra como as entidades de cada módulo estão definidas.

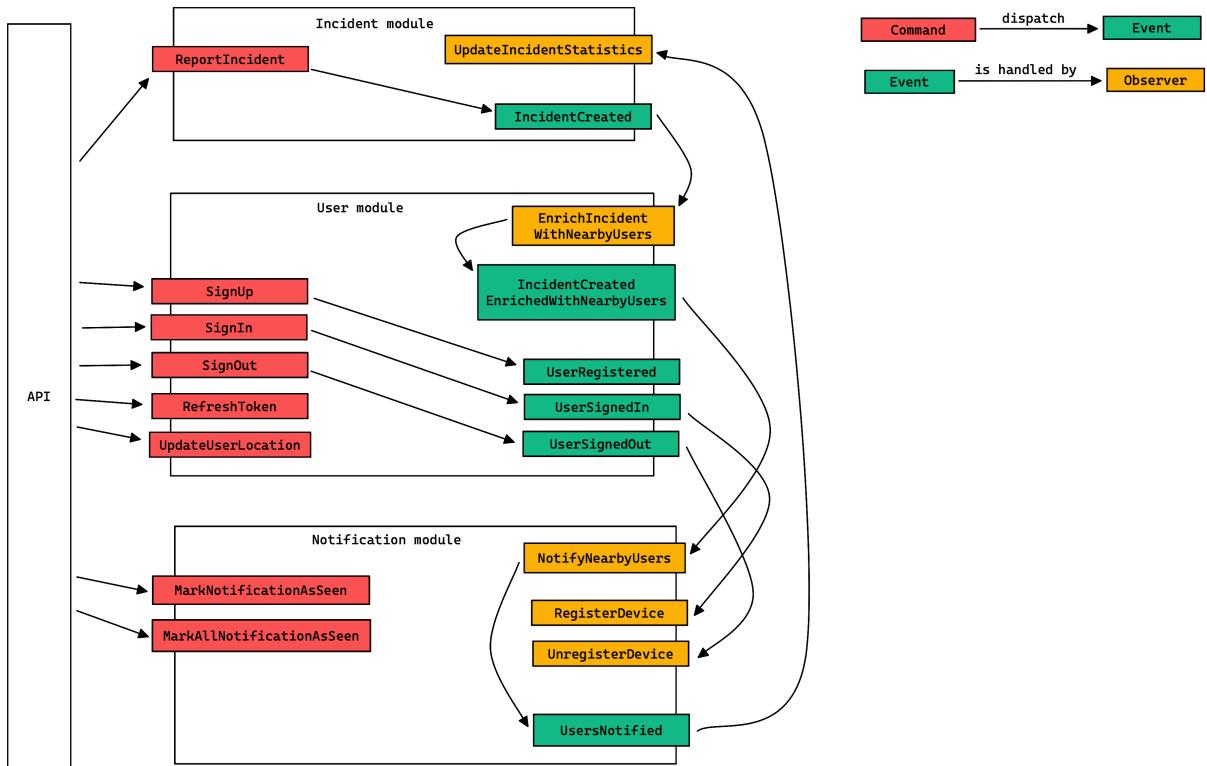
Figura 7 – Entidades de domínio de cada módulo do servidor.



Fonte: Elaborado pelo autor.

A Figura 8 ilustra as entradas e saídas de cada módulo.

Figura 8 – Entradas e saídas de cada módulo do servidor.



Fonte: Elaborado pelo autor.

#### 4.1.3 Principais tecnologias

Para a implementação do servidor, tecnologias específicas foram escolhidas e cada uma delas tiveram um motivador de escolha.

### 4.2 Comunicação entre servidor e cliente

#### 4.2.0.0.1 GraphQL

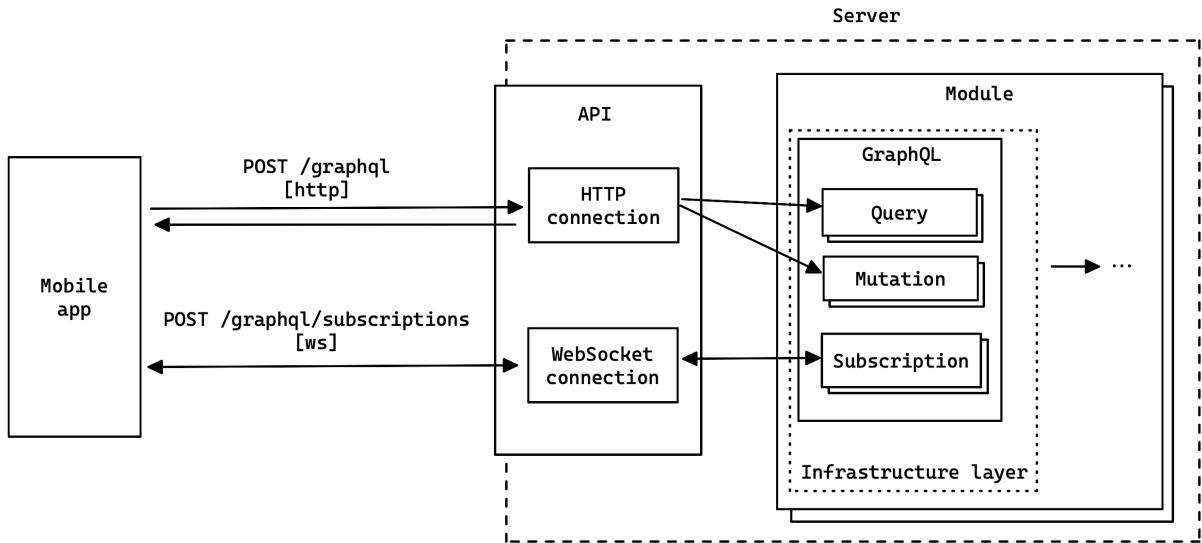
GraphQL ([GRAPHQL...](#), 2022) é uma linguagem de consulta

#### 4.2.0.0.2 Do cliente ao módulo do servidor

A Figura 9 ilustra de forma mais detalhada como o aplicativo se comunica com o servidor. Para GraphQL queries e GraphQL mutations há um fluxo síncrono de requisição-resposta onde é utilizado o protocolo HTTP, enquanto que para GraphQL subscriptions há um fluxo assíncrono e onde é estabelecida uma conexão persistente via o protocolo WebSocket

para que o servidor possa também enviar atualizações ao cliente sem que haja uma requisição prévia feita dele. Isso permite com que o usuário visualize os dados mais atualizados possíveis na tela. Alertas criados próximo à um usuário que está com o aplicativo aberto é um exemplo de atualização enviada apenas pelo servidor através de uma GraphQL subscription, a qual a aplicação cliente fica "ouvindo" desde o momento que é iniciada.

Figura 9 – Interação detalhada entre cliente e servidor.



Fonte: Elaborado pelo autor.

## 4.3 Aplicativo

### 4.3.1 Principais tecnologias

4.3.1.1 React Native

4.3.1.2 Relay

4.3.1.3 Print das telas

# 5 Tecnologias escolhidas

Para implementar o projeto descrito, as seguintes tecnologias foram adotadas:

- Redis ([REDIS...](#), 2022): banco de dados em memória de baixa latência. Utilizado para consultas eficientes por latitude e longitude e cache de tokens de sessão;
- PostgreSQL ([POSTGRESQL...](#), 2022): banco relacional. Utilizado para persistência dos dados dos usuários, alertas e notificações;
- AWS ([AMAZON...](#), 2022): provedor de serviços em nuvem. Foi utilizado o serviço *AWS S3* para armazenamento de objetos estáticos, como imagens e vídeos, e o serviço *AWS RDS* para o provisionamento do banco de dados PostgreSQL;
- Heroku ([HEROKU...](#), 2022): provedor de serviços em nuvem. Foi utilizado para o provisionamento do servidor e do banco de dados Redis;
- Javascript ([JAVASCRIPT...](#), 2022): linguagem de programação. Utilizada tanto no aplicativo quanto no servidor;
- Typescript ([TYPESCRIPT...](#), 2022): sistema de tipos para Javascript. Utilizado tanto no aplicativo quanto no servidor;
- GraphQL ([GRAPHQL...](#), 2022): especificação e linguagem de consulta para APIs;
- Terraform ([TERRAFORM...](#), 2022): ferramenta de *Infrastructure as Code (IaC)* para gerenciamento serviços em nuvem via arquivos de configuração declarativos;
- Docker ([DOCKER...](#), 2022): ferramenta de virtualização à nível do sistema operacional. Utilizado durante o desenvolvimento isolado através de containers;
- Node.js ([NODE...](#), 2022): *Javascript runtime* construído sobre o motor V8 do Chrome. Utilizado para a escrita do backend (servidor);
- Koa ([KOA...](#), 2022): framework para construção de servidor web no Node.js;
- Prisma ([PRISMA...](#), 2022): *Object-relational mapper (ORM)* para Node.js;
- Jest ([JEST...](#), 2022): framework para Node.js de testes unitários;
- Supertest ([SUPERTEST...](#), 2022): framework para Node.js de testes de integração;
- React ([REACT...](#), 2022a): biblioteca Javascript para a construção de interfaces de usuário de forma declarativa;

- React Native ([REACT..., 2022b](#)): framework híbrido para a construção de aplicações móveis que rodam em múltiplas plataformas (iOS e Android) utilizando React e Javascript.
- Expo ([EXPO..., 2022](#)): *React Native runtime*, SDK e ferramenta para melhor experiência de desenvolvimento com React Native;
- Relay ([RELAY..., 2022](#)): especificação GraphQL e framework para a busca de dados no React em APIs GraphQL com foco em performance;
- Recoil ([RECOIL..., 2022](#)): biblioteca para gerenciamento de estado global no React;
- React Navigation ([REACT..., 2022c](#)): biblioteca de navegação de telas no React Native.

## 6 Conclusão

Conclui-se que os requisitos funcionais e não funcionais do sistema foram atentidos. O usuário final é capaz de realizar as principais funções de negócio como publicar alertas e ser notificado por novos alertas nas proximidades. A implementação de cada nova funcionalidade do sistema ocorreu de forma intercalada entre o servidor e o aplicativo. A adoção do framework Relay ([RELAY..., 2022](#)) gerou uma alta curva de aprendizagem inicial, mas que foi compensada por ganhos em performance, flexibilidade e escalabilidade da aplicação cliente.

Em relação à validação do servidor, conclui-se que ele foi testado localmente repetidas vezes durante o processo de desenvolvimento com a ajuda do Docker ([DOCKER..., 2022](#)) para provisionamento local dos banco de dados Redis e PostgreSQL. Cada módulo do servidor foi testado com testes unitários - testes automatizados que devem testar uma única unidade de código, ignorando eventuais dependências externas. O aplicativo foi testado de forma manual e em um ambiente local.

Ainda existem, porém, dúvidas em relação à receptividade da opinião público ao projeto. Como o aplicativo não foi distribuído em lojas em razão dos custos e por se tratar, ainda, de um projeto prematuro, a ideia não foi testada com usuários reais. Considerando que o problema atacado - a segurança - é sensível a todos e a solução envolve a confiança entre pessoas anônimas, o maior desafio transcende as barreiras tecnológicas e está relacionado ao ser humano. O usuário pode, por exemplo, usar o aplicativo de forma não moderada ao agir como um “vigilante”, “reporter criminal” ou até mesmo como um “justiçiero”. Ele pode fornecer alertas equivocados, difamatórios, racistas ou mentirosos, sejam eles mal intencionados ou não.

O uso frequente do aplicativo pode levantar questões mais filosóficas aos usuários como: “É melhor ser totalmente informado de todos os crimes e emergências e ficar possivelmente paranoico, ou ser totalmente ignorante deles mas viver em constante perigo?”

# Referências

AMAZON Web Services documentation. 2022. Disponível em: <<https://aws.amazon.com>>. Acesso em 20 de fevereiro de 2022.

APLICATIVO Citizen. 2022. Disponível em: <[https://play.google.com/store/apps/details?id=sp0n.citizen&hl=pt\\_BR&gl=US](https://play.google.com/store/apps/details?id=sp0n.citizen&hl=pt_BR&gl=US)>. Acesso em 20 de fevereiro de 2022.

APLICATIVO SP+ segura. 2022. Disponível em: <[https://play.google.com/store/apps/details?id=com.policiapopular.spsegura&hl=pt\\_BR&gl=US](https://play.google.com/store/apps/details?id=com.policiapopular.spsegura&hl=pt_BR&gl=US)>. Acesso em 20 de fevereiro de 2022.

BOB, U. *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. [S.I.]: Pearson, 2017.

COCKBURN, A. *Hexagonal architecture*. [S.I.: s.n.], 2005. Disponível em: <<https://alistair.cockburn.us/hexagonal-architecture>>. Acesso em 20 de fevereiro de 2022.

DOCKER documentation. 2022. Disponível em: <<https://www.docker.com>>. Acesso em 20 de fevereiro de 2022.

EVANS, E. *Domain-Driven Design: Tackling Complexity in the Heart of Software*. [S.I.]: Addison-Wesley Professional, 2003.

EXPO documentation. 2022. Disponível em: <<https://expo.dev>>. Acesso em 20 de fevereiro de 2022.

GRAPHQL documentation. 2022. Disponível em: <<https://graphql.org>>. Acesso em 20 de fevereiro de 2022.

HEROKU documentation. 2022. Disponível em: <<http://heroku.com>>. Acesso em 20 de fevereiro de 2022.

JAVASCRIPT documentation. 2022. Disponível em: <<https://developer.mozilla.org/docs/Web/JavaScript>>. Acesso em 20 de fevereiro de 2022.

JEST documentation. 2022. Disponível em: <<https://jestjs.io>>. Acesso em 20 de fevereiro de 2022.

KOA documentation. 2022. Disponível em: <<https://koajs.com>>. Acesso em 20 de fevereiro de 2022.

MEYER, B. *Object Oriented Software Construction*. 2. ed. [S.I.]: Pearson College Div, 2000.

NODE.JS documentation. 2022. Disponível em: <<https://nodejs.org>>. Acesso em 20 de fevereiro de 2022.

POSTGRESQL documentation. 2022. Disponível em: <<https://www.postgresql.org>>. Acesso em 20 de fevereiro de 2022.

PRISMA documentation. 2022. Disponível em: <<https://www.prisma.io>>. Acesso em 20 de fevereiro de 2022.

REACT documentation. 2022. Disponível em: <<https://reactjs.org>>. Acesso em 20 de fevereiro de 2022.

REACT Native documentation. 2022. Disponível em: <<https://reactnative.dev>>. Acesso em 20 de fevereiro de 2022.

REACT Navigation documentation. 2022. Disponível em: <<https://reactnavigation.org>>. Acesso em 20 de fevereiro de 2022.

RECOIL documentation. 2022. Disponível em: <<https://recoiljs.org>>. Acesso em 20 de fevereiro de 2022.

REDIS documentation. 2022. Disponível em: <<https://redis.io>>. Acesso em 20 de fevereiro de 2022.

RELAY documentation. 2022. Disponível em: <<https://relay.dev>>. Acesso em 20 de fevereiro de 2022.

SUPERTEST documentation. 2022. Disponível em: <<https://www.npmjs.com/package/supertest>>. Acesso em 20 de fevereiro de 2022.

TERRAFORM documentation. 2022. Disponível em: <<https://www.terraform.io>>. Acesso em 20 de fevereiro de 2022.

TYPESCRIPT documentation. 2022. Disponível em: <<https://www.typescriptlang.org>>. Acesso em 20 de fevereiro de 2022.

WAZE. 2022. Disponível em: <<https://www.waze.com>>. Acesso em 20 de fevereiro de 2022.

WIKIPEDIA. 2022. Disponível em: <<https://pt.wikipedia.org>>. Acesso em 20 de fevereiro de 2022.