instituto superior de engenharia de lisboa

TASA - Theater Auto Silence App

Gonçalo Ribeiro
João Marques

Orientador: Artur Ferreira

Relatório do projeto realizado no âmbito de Projeto e Seminário
Licenciatura em Engenharia Informática e de Computadores

Junho de 2025

Instituto Superior de Engenharia de Lisboa

TASA - Theater Auto Silence App

48305 Gonçalo David Ferreira Ribeiro

48297 João Renato Vargas Marques

Orientadores: Artur Ferreira

Relatório do projeto realizado no âmbito de Projeto e Seminário
Licenciatura em Engenharia Informática e de Computadores

Junho de 2025

Resumo

Frequentemente, em ambientes públicos como teatros, cinemas, bibliotecas e espaços académicos, a interrupção causada pelos sons de dispositivos móveis constitui um problema social recorrente que afeta negativamente a experiência coletiva, causando perturbações indesejadas e quebras de concentração, além de representar uma forma de poluição sonora.

A nossa contribuição para solucionar este problema consiste no desenvolvimento da Theater Auto Silence App (TASA), uma aplicação móvel autónoma capaz de silenciar automaticamente dispositivos com base na localização geográfica do utilizador ou na sua participação em eventos previamente agendados, eliminando a necessidade de intervenção manual.

Assim, neste projeto apresenta-se uma solução que inclui diversos componentes, desde um sistema de geolocalização híbrida (combinando GPS, Wi-Fi e Bluetooth) para aumentar a precisão do posicionamento, até modos de silenciamento contextual que se adaptam a diferentes cenários. A solução é composta por uma integração com as agendas dos dispositivos móveis, uma API para acesso e armazenamento de dados, e uma interface gráfica intuitiva que facilita a interação do utilizador com a aplicação, permitindo uma experiência fluida e eficiente.

Palavras-chave: Silenciamento automático; Geolocalização híbrida; TASA; Poluição sonora; Aplicações móveis; Integração de agenda.

Abstract

Frequently, in public environments such as theaters, cinemas, libraries, and academic spaces, the interruption caused by mobile device sounds constitutes a recurring social problem that negatively affects the collective experience, causing unwanted disturbances and breaks in concentration, besides representing a form of noise pollution.

Our contribution to solving this problem consists of developing the Theater Auto Silence App (TASA), an autonomous mobile application capable of automatically silencing devices based on the user's geographical location or their participation in previously scheduled events, eliminating the need for manual intervention.

Thus, this project presents a solution that includes various components, from a hybrid geolocation system (combining GPS, Wi-Fi, and Bluetooth) to increase positioning accuracy, to contextual silencing modes that adapt to different scenarios. The solution comprises integration with mobile device calendars, a customized notification management system, an API for data access and storage, and an intuitive graphical interface that facilitates user interaction with the application, allowing for a fluid and efficient experience.

Keywords: Automatic silencing; Hybrid geolocation; TASA; Noise pollution; Mobile applications; Calendar integration

Índice

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Atualmente, os dispositivos móveis fazem parte do quotidiano das pessoas, sendo utilizados em diversas situações, tanto para fins profissionais como pessoais. A facilidade com que se recebe e envia informação, associada à constante ligação às redes digitais, tornou os telemóveis indispensáveis. No entanto, esta presença constante traz consigo desafios, como as interrupções sonoras inesperadas em momentos inapropriados, por exemplo, em reuniões, aulas ou eventos culturais.

Perante esta realidade, surge a necessidade de encontrar soluções que permitam ao utilizador manter a conveniência do seu dispositivo sem comprometer o respeito pelo contexto em que se encontra. Foi neste enquadramento que decidimos desenvolver uma aplicação móvel que automatize o processo de silenciamento do dispositivo com base em critérios como a localização e a calendarização de eventos.

Assim, apresentamos a TASA – *Theater Auto Silence App* –, uma aplicação que tem como finalidade ajudar os utilizadores a evitar interrupções sonoras indesejadas, oferecendo uma solução prática, automática e personalizável para situações do dia a dia em que o silêncio é essencial.

1.1 Problema

Com a utilização crescente de dispositivos móveis, tornou-se comum ocorrerem interrupções sonoras em locais e momentos inadequados, como durante reuniões de trabalho, aulas, bibliotecas ou eventos culturais. Apesar de os sistemas operativos permitirem definir perfis de som ou modos de silêncio, estas opções requerem, em geral, ação manual por parte do utilizador, o que leva frequentemente a esquecimentos ou configurações incorretas.

Este problema, embora aparentemente simples, pode causar situações embaraçosas, quebra de concentração e desrespeito por normas sociais. A ausência de uma solução automatizada, que integre contexto como localização e horários, continua a ser uma lacuna na maioria das aplicações existentes.

1.2 Objetivos do trabalho

O projeto tem como objetivos principais:

- Desenvolver uma aplicação Android que silencie automaticamente o telemóvel do utilizador com base em regras definidas por localização geográfica ou eventos previamente agendados no calendário;
- Permitir ao utilizador configurar regras, com opções de exceção;
- Garantir que a aplicação funciona de forma intuitiva, com consumo reduzido de bateria e compatibilidade com versões recentes do sistema operativo Android;

1.3 Trabalhos relacionados e solução proposta

Existem já no mercado algumas aplicações que permitem silenciar dispositivos móveis com base em horários fixos ou eventos de calendário (como o Google Calendar). No entanto, essas soluções são, na sua maioria, limitadas em termos de flexibilidade, personalização ou integração com a localização do utilizador.

A solução proposta, a aplicação TASA, visa colmatar estas limitações através de uma abordagem mais completa, combinando informação temporal e geográfica para decidir automaticamente quando o dispositivo deve ser colocado em silêncio.

1.4 Estrutura do relatório

Este relatório está organizado da seguinte forma:

- O **Capítulo ??** apresenta os trabalhos relacionados, incluindo aplicações existentes e soluções semelhantes;
- Por fim, o **Capítulo ??** apresenta as conclusões do trabalho e sugestões para possíveis melhorias futuras.

Capítulo 2

Enquadramento

Este capítulo tem como objetivo contextualizar o desenvolvimento da aplicação **TASA**, descrevendo o trabalho relacionado e identificando sistemas com funcionalidades semelhantes. A aplicação desenvolvida permite ao utilizador definir localizações geográficas e eventos com horário específico para silenciar automaticamente o dispositivo móvel, evitando interrupções indesejadas em momentos ou locais onde o silêncio é necessário.

2.1 Trabalho Relacionado

Com o aumento da utilização de dispositivos móveis em todas as vertentes do dia a dia, tornou-se comum a necessidade de silenciar o dispositivo em determinadas situações — como reuniões, aulas, consultas médicas ou eventos sociais. No entanto, este processo continua a ser, muitas vezes, manual e dependente da atenção do utilizador.

Para dar resposta a essa limitação, foi desenvolvida a aplicação **TASA**, que permite automatizar esse processo com base em dois critérios: **localização geográfica** e **intervalos de tempo definidos pelo utilizador**. Sempre que o utilizador entra numa zona previamente registada ou durante o período de um evento marcado, o dispositivo passa automaticamente para modo silencioso. Findo o evento ou ao sair da zona, o som é restaurado.

A aplicação foi implementada utilizando *Kotlin* e *Jetpack Compose*, tirando partido dos recursos disponibilizados pelo sistema operativo Android, como o acesso à localização, gestão do modo de som e integração com o calendário do dispositivo.

2.2 Sistemas Semelhantes

Atualmente, existem no mercado algumas aplicações que oferecem funcionalidades semelhantes, ainda que com abordagens diferentes. Entre as mais relevantes destacam-se:

- **Modo "Não Incomodar" (Android nativo)** – Permite ao utilizador configurar períodos de silêncio automáticos, mas não permite definir zonas geográficas para esse efeito, nem combina múltiplas regras de forma personalizada.

- **Tasker** – Aplicação de automação bastante completa que permite criar regras personalizadas, incluindo ações baseadas em localização e horário. No entanto, a sua complexidade e a necessidade de configurações detalhadas tornam-na menos acessível para utilizadores que procuram soluções simples e específicas.

- **MacroDroid** – Semelhante ao Tasker, permite criar "macros" com base em eventos, horários e localização. Ainda assim, exige configuração manual detalhada e não é focada exclusivamente na gestão do modo de som.

- **Llama (descontinuada)** – Era uma aplicação popular focada em perfis automáticos com base na localização, permitindo silenciar o dispositivo em zonas predefinidas. Apesar de já não estar disponível nas versões mais recentes do Android, foi uma referência importante neste tipo de funcionalidade e demonstrou o interesse por soluções de silenciamento automático.

A aplicação **TASA** propõe-se como uma solução mais simples e direta, centrada exclusivamente no **silenciamento automático do dispositivo**, combinando **localização** e **eventos temporais**. O seu principal diferencial está na facilidade de utilização e no foco específico, eliminando funcionalidades desnecessárias e tornando o processo de configuração rápido e intuitivo.

Capítulo 3

Solução Proposta

Neste capítulo, é apresentada em detalhe a solução desenvolvida para o projeto *TASA – Theater Auto Silence App*, com foco na arquitetura geral do sistema e nos principais componentes envolvidos. A descrição está organizada em quatro secções, cada uma dedicada a um componente específico da solução.

Na Secção 3.1, é descrita a abordagem de desenvolvimento adotada, incluindo a metodologia e as etapas seguidas ao longo do projeto. Segue-se a análise funcional e técnica na Secção **??**, onde se identificam os requisitos, constrangimentos e tecnologias escolhidas. A Secção **??** apresenta o projeto da solução, com destaque para a arquitetura, os modelos e a lógica de funcionamento. Por fim, a Secção **??** foca-se na implementação prática do sistema, abordando os detalhes técnicos, as dificuldades encontradas e os testes realizados.

A arquitetura geral do sistema proposto encontra-se representada na Figura 3.1. Esta ilustra a interação entre os principais componentes da aplicação, nomeadamente:

- **Frontend**: a interface com o utilizador, implementada em dispositivos móveis;
- **Backend**: o servidor responsável por processar lógica de negócio e comunicar com a base de dados;
- **Base de Dados**: sistema de armazenamento persistente que guarda informações relevantes, tais como dados do utilizador, as suas regras definidas e localizações associadas.

A comunicação entre os componentes é feita através de uma *Application Programming Interface* (API) privada, utilizando pedidos baseados no protocolo HTTP (*HyperText Transfer Protocol*).

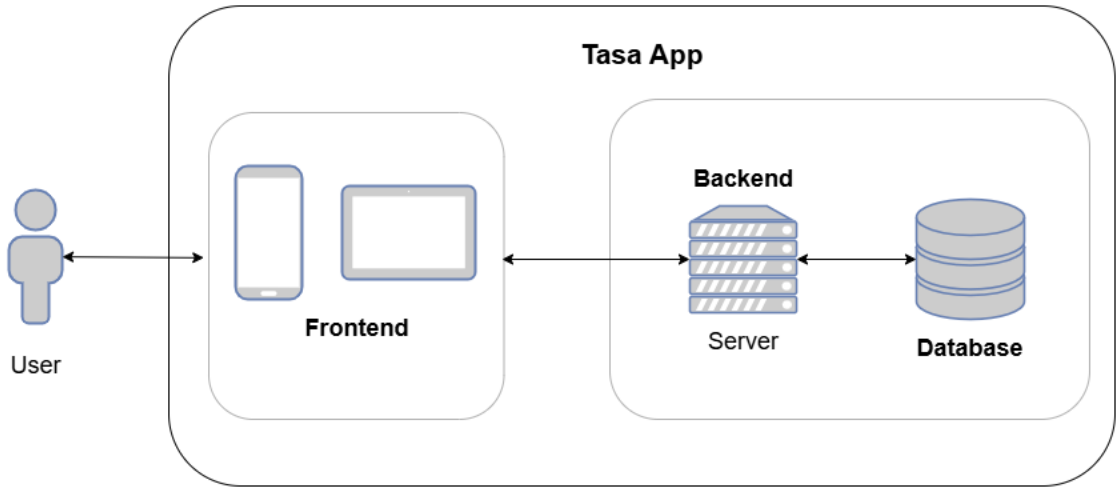

Figura 3.1: Diagrama geral da solução.

3.1 Base de dados

Para garantir a persistência dos dados resultantes da interação cliente-servidor foi necessário recorrer a uma base de dados. Escolheu-se o tipo de base de dados relacional em conjunto com o sistema de gestão de base de dados PostgreSQL, devido à necessidade de estruturação e ligação dos dados. Além das razões já mencionadas, o grupo possuía experiência prévia com PostgreSQL, o que evitou a necessidade de um período de aprendizagem.

3.1.1 Modelo Entidade-Associação

Desenvolveu-se o Modelo Entidade-Associação com o objetivo de representar as entidades, bem como as relações entre elas.

A Figura 3.2 representa o modelo de dados desenvolvido.

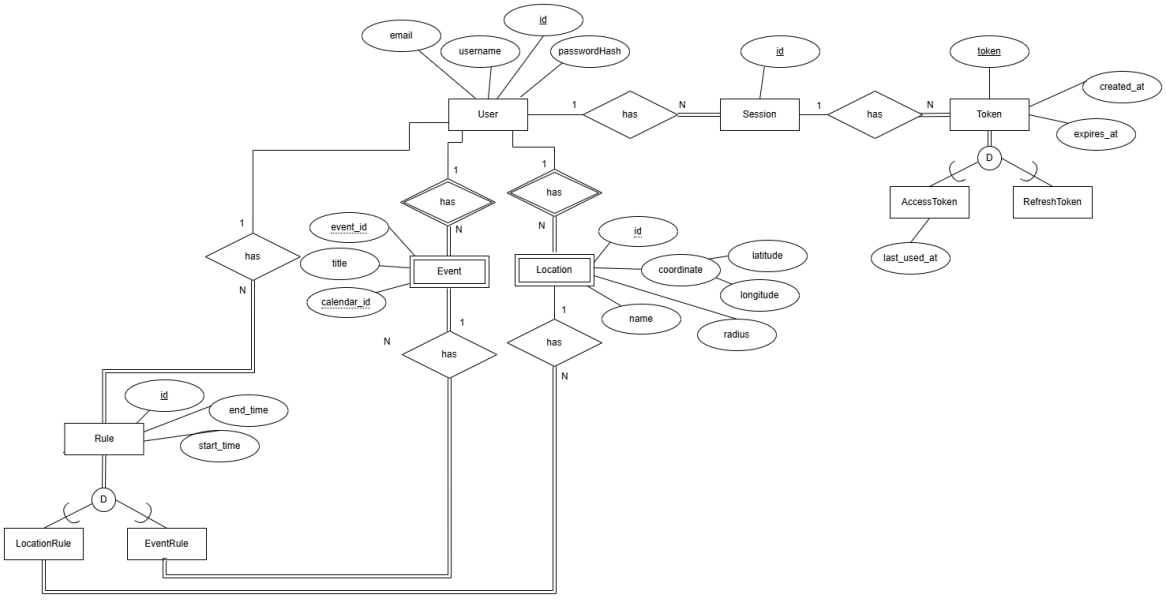

Figura 3.2: Modelo de dados da aplicação TASA.

3.1.2 Entidades

Definiram-se várias entidades que armazenam informações relativas aos utilizadores, às sessões de autenticação, aos eventos, às localizações, bem como às regras e exceções associadas ao processo de silenciamento. Abaixo encontra-se a descrição de cada entidade.

Entidade User – A entidade *User* representa os dados relativos a um utilizador da aplicação. Possui os atributos *id*, *username*, *email* e *passwordHash*. Cada utilizador é identificado por um identificador único, atribuído de forma sequencial, sendo o atributo *id* a chave primária desta entidade.

Entidade Session - A entidade *Session* representa uma sessão de um utilizador. Possui como único atributo um identificador que também é a chave primária da entidade, desta forma identificando a sessão.

Entidade Token – A entidade *Token* tem como objetivo representar um Token de um utilizador autenticado. Para esse efeito, a entidade apresenta os seguintes atributos: um *token* (que funciona como chave primária), a data de criação da sessão, a data da última utilização do token e a data de expiração. Nesta entidade, observa-se uma disjunção com relação total, de forma que a *Token* deve ser de um dos dois tipos: - **AccessToken**: representa o token obtido durante a autenticação do utilizador. - **RefreshToken**: representa o token que pode ser utilizado para obter uma nova sessão quando a atual se encontra prestes a expirar, sem que o utilizador tenha de se autenticar novamente.

Entidade Event – A entidade *Event* representa um evento selecionado pelo utilizador para definir uma regra de silenciamento. Esta entidade possui três atributos, sendo a chave primária composta pelos atributos *calendar_id* e *event_id*, o que permite identificar os even-

tos na agenda do telemóvel (uma vez que são extraídos o CALENDAR_ID e o EVENT_ID atribuídos pela agenda do Android, conforme indicado na documentação [1]. Adicionalmente, a entidade inclui o atributo *title*, referente ao título do evento.

Entidade Location – A entidade *Location* representa uma localização definida pelo utilizador. Esta contém os atributos *id*, a chave primária da entidade, *name* o nome definido pelo utilizador, *radius* o raio de abrangência da localização e *coordinate*, sendo este último um atributo composto constituído pelos atributos *latitude* e *longitude*.

Entidade Rule – A entidade *Rule* representa uma regra definida pelo utilizador que determina em que situação o silenciamento deverá ser aplicado. Esta entidade possui os atributos *id* (identificador atribuído de forma sequencial), *start_time* (data de início da regra), *end_time* (data de fim da regra).

Estas entidades constituem a base do modelo de dados da aplicação TASA, de acordo com os requisitos estabelecidos na proposta de projeto.

3.1.3 Relações entre as entidades

As entidades apresentadas no modelo da base de dados da aplicação encontram-se interligadas por diversas relações. Abaixo descrevem-se essas relações:

- **User–Session**: Existe uma relação de um-para-muitos (*1:N*) entre a entidade *User* e a entidade *Session*, o que significa que cada utilizador pode ter várias sessões de autenticação ativas (por exemplo, em diferentes dispositivos). A cada *Session* corresponde um e só um *User*.

- **Session–Token**: Cada sessão pode ter associados múltiplos tokens (*1:N*). Esta relação permite suportar mecanismos de autenticação modernos, onde são gerados tokens de acesso e de renovação para cada sessão ativa. Um *Token* está associado a uma só *Session*.

- **Token–AccessToken / RefreshToken**: A entidade *Token* está ligada a uma disjunção total e exclusiva (restrição *Disjoint*) entre os tipos *AccessToken* e *RefreshToken*, o que significa que cada token deve ser obrigatoriamente de um e apenas um destes dois tipos.

- **User–Event**: Existe uma relação de um-para-muitos (*1:N*) entre *User* e *Event*, permitindo armazenar os eventos das regras do utilizador.

- **User–Location**: Também se verifica uma relação de um-para-muitos (*1:N*) entre *User* e *Location*, indicando que um utilizador pode definir várias localizações relevantes.

- **User–Rule**: Cada utilizador pode criar múltiplas regras (*1:N*), sendo que cada regra pertence a um único utilizador.

- **Event–Rule (via EventRule)**: A relação entre *Event* e *Rule* é feita através da entidade derivada *EventRule*, o que representa uma especialização da entidade *Rule*. Esta ligação de um-para-muitos (*1:N*) entre as duas entidades, o que permite associar uma regra a um evento específico e um evento a múltiplas regras.

- **Location–Rule (via LocationRule)**: De forma semelhate, a entidade *LocationRule* especializa a entidade *Rule* e estabelece a ligação entre uma localização e a respetiva regra.

3.2 *Backend*

O *backend* constituído por um servidor que expõe uma API *Application Programming Interface*, este implementa a lógica de negócio e a persistência de dados. Disponibilizando recursos de forma a que o utilizador possa armazenar as suas informações e propagá-las para diferentes dispositivos. Este componente é responsável por realizar o acesso à base de dados.

Para o desenvolvimento do *backend*, foi escolhida a linguagem **Kotlin** por ser moderna, concisa, fortemente tipificada. A familiaridade prévia do grupo com Kotlin facilitou a sua adoção, eliminando a necessidade de um período de aprendizagem. A estrutura do servidor utiliza a framework **Spring MVC**,uma das grandes vantagens do Spring é a forma como abstrai os detalhes da comunicação HTTP: ao utilizar anotações simples, como por exemplo para mapear rotas ou parametrizar pedidos, o framework trata automaticamente da receção e processamento de pedidos HTTP, bem como da construção e envio das respetivas respostas. Além disso, é responsável pela injeção de dependências, evitando erros do programador. Para o acesso à base de dados, optou-se pela utilização da biblioteca **JDBI**, esta escolha permitiu maior controlo sobre as queries SQL. O grupo já tinha tido contacto com esta biblioteca o que acelerou o processo de implementação, uma vez que não foi necessário a aprendizagem.

A Figura 3.3 representa a arquitetura do *backend*.

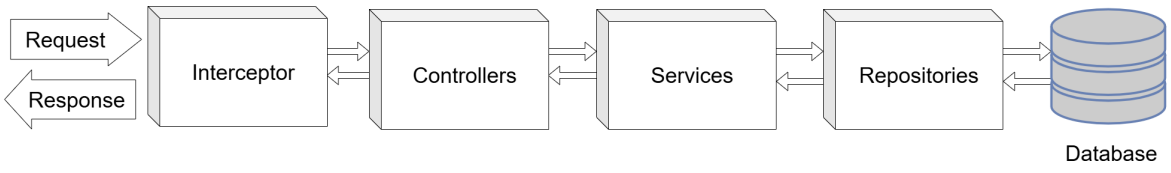

Figura 3.3: Arquitetura do *backend*.

O *backend* é constituído por vários módulos, com o objetivo de organizar e separar responsabilidades dentro do sistema.

A Figura 3.4 representa os diferentes módulos que constituem o *backend* do *backend*.

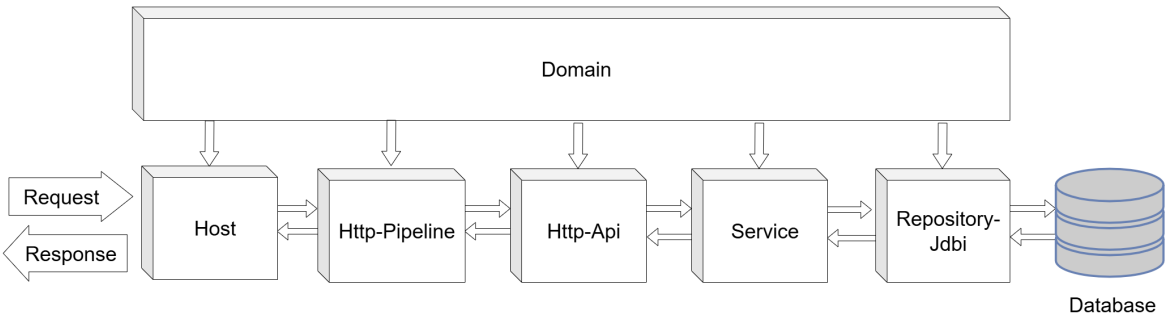

Figura 3.4: Diagrama de blocos do *backend*.

3.2.1 Módulo *Domain*

O módulo *Domain* contém todas as definições de tipos que são partilhadas entre os vários módulos do sistema e funções utilitárias. O principal objetivo deste módulo é promover a separação de responsabilidades e garantir a coesão do modelo de domínio.

Este módulo é independente dos restantes, ou seja, não possui dependências de outros módulos, o que o torna reutilizável e fácil de testar.

- **User**: Representa um utilizador.
- **AuthenticatedUser**: Representa um utilizador autenticado.
- **Event**: Representa um evento.
- **Location**: Representa uma localização.
- **PasswordValidationInfo**: Representa o hash da password do utilizador.
- **Rule**: Representa uma Regra.
- **RuleEvent**: Representa uma regra com um evento associado.
- **RuleLocation**: Representa uma regra com uma localização associada.
- **Session**: Representa uma sessão.
- **Sha256TokenEncoder**: Classe com funções de hash.
- **TokenValidationInfo**: Representa token.
- **UsersDomain**: Classe utilitária com funções de validação.
- **UsersDomainConfig**: Representa a configuração para a classe UsersDomain.

3.2.2 Módulo *Host*

O módulo *host* é responsável pela configuração e arranque da aplicação. Neste módulo são definidos os componentes essenciais ao funcionamento do servidor, como a ligação à base de dados, os mecanismos de autenticação, a documentação da API e a injeção de dependências.

As dependências da aplicação são definidas como *beans* através de métodos anotados com `@Bean`, permitindo que sejam geridas pelo *Spring*. Para lidar com a autenticação de utilizadores, foi desenvolvido um *HandlerInterceptor* e registado através da implementação da interface `WebMvcConfigurer`. A classe `PipelineConfigurer` trata da configuração deste mecanismo, adicionando o *interceptor* ao registo de interceptores da aplicação e registando também um *argument resolver* que permite injetar automaticamente a informação do utilizador autenticado nos controladores.

3.2.3 Módulo *Http-Api*

3.2.4 Módulo *Sevice*

3.2.5 Módulo *Repository*

3.2.6 Módulo *Repository-Jdbi*

3.3 *Frontend*

O componente *frontend* consiste numa aplicação móvel desenvolvida para o sistema operativo Android. Este ecossistema foi escolhido como alvo principal devido à sua ampla quota de mercado, bem como à experiência prévia do grupo no desenvolvimento de aplicações Android, adquirida em unidades curriculares do curso.

No desenvolvimento do *frontend* foram utilizadas várias tecnologias modernas do ecossistema Android. A linguagem escolhida foi *Kotlin*, garantindo uma maior consistência com o *backend* e facilitando a manutenção do projeto. Para a construção da interface gráfica, recorreu-se ao *Jetpack Compose*, uma *framework* declarativa que permite o desenvolvimento de interfaces modernas, responsivas e eficientes. A comunicação com o servidor foi implementada através do cliente HTTP *Ktor*, escolhido pela sua integração nativa com *Kotlin* e pela familiaridade do grupo com esta biblioteca. Estas escolhas tecnológicas, aliadas à experiência prévia do grupo, permitiram um desenvolvimento mais célere.

3.3.1 Casos de utilização

Nesta secção são apresentados os principais casos de utilização do sistema, com o objetivo de ilustrar as interações entre o utilizador e a aplicação. Cada caso de utilização descreve um cenário representativo das funcionalidades disponibilizadas, com o objetivo de demonstrar de forma clara os fluxos das operações. Foram definidos três casos de utilização que refletem os requisitos essenciais identificados durante a fase de análise: a criação de uma regra com base

num evento, a definição de regra de silenciamento com base numa localização e a configuração das exceções.

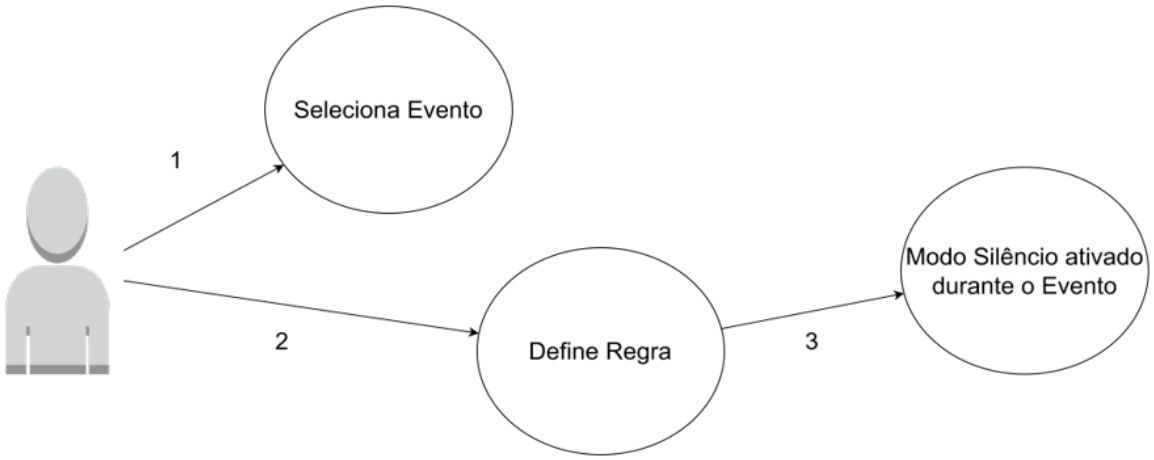

Figura 3.5: Caso de utilização 1.

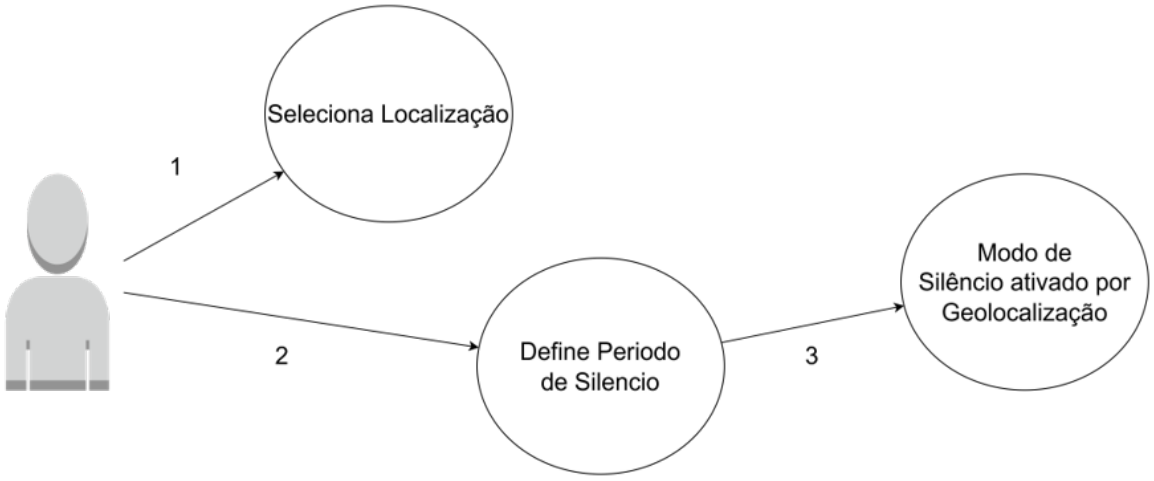

Figura 3.6: Caso de utilização 2.

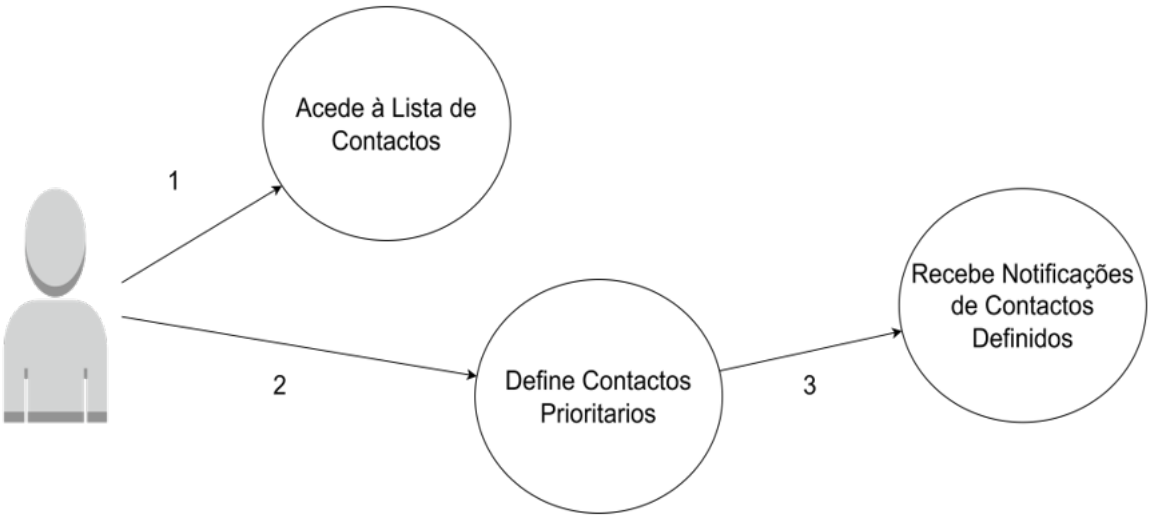

Figura 3.7: Caso de utilização 3.

3.3.2 Acesso à agenda do telemóvel

O acesso ao calendário na aplicação foi implementado através do *Calendar Provider*, um componente da *framework* Android concebido para permitir a leitura e manipulação centralizada dos dados de calendário armazenados no equipamento. Esta abordagem foi seleccionada devido à sua integração nativa com o sistema operativo, garantindo compatibilidade com diferentes versões do Android e com os diversos serviços de calendário (por exemplo, Google Calendar, Exchange, entre outros) sincronizados no equipamento do utilizador.

O *Calendar Provider* disponibiliza uma interface estruturada sobre uma base de dados interna, onde os dados são organizados em entidades como *calendários*, *eventos*, *lembretes* e *participantes*. A aplicação interage com este sistema através de consultas e operações de inserção/actualização, utilizando *URIs* normalizadas fornecidas pela classe `CalendarContract`. Esta estrutura permite, por exemplo, obter a lista de eventos futuros, identificar eventos recorrentes ou criar novos eventos em calendários específicos, tudo com elevado controlo sobre os dados descritivos associados (como horário, localização, descrição ou notificações).

Para garantir a segurança e a privacidade do utilizador, o acesso ao *Calendar Provider* está sujeito a um conjunto de permissões explícitas (`READ_CALENDAR` e `WRITE_CALENDAR`), exigindo não só a sua declaração no ficheiro `AndroidManifest.xml`, como também a sua autorização em tempo de execução a partir do Android 6.0 (API 23). A aplicação foi desenvolvida de forma a respeitar estas restrições, solicitando ao utilizador a autorização apenas quando estritamente necessário, promovendo uma utilização responsável dos seus dados pessoais.

A decisão de utilizar directamente o *Calendar Provider*, em detrimento de soluções externas ou integrações via *API web*, deve-se à necessidade de operar com dados locais e actualizados, mesmo em cenários sem ligação à Internet. Para além disso, esta abordagem assegura que os eventos gerados ou modificados pela aplicação ficam automaticamente visíveis em qualquer

outra aplicação de calendário presente no equipamento, promovendo a interoperabilidade e garantindo uma experiência de utilização consistente.

3.3.3 Silenciamento Automático

A funcionalidade de silenciamento automático da aplicação foi implementada com recurso ao `AlarmManager`[2], componente da framework Android concebido para agendar tarefas que devem ser executadas em momentos exatos. A escolha desta abordagem prende-se com a necessidade de garantir precisão temporal na execução da ação de silenciamento, mesmo em contextos de poupança de energia (modo Doze) [3]. O método `setExactAndAllowWhileIdle` foi utilizado para assegurar a execução mesmo quando o dispositivo se encontra em estado de inatividade. Embora alternativas como o `WorkManager`[4] ofereçam uma compatibilidade com diferentes versões do Android, estas são orientadas para tarefas recorrentes ou com tolerância a atrasos, o que não satisfaz os requisitos desta aplicação, que exige ativação precisa de alarmes com impacto direto na experiência do utilizador.

Para o silenciamento propriamente dito, optou-se por recorrer ao modo *Do Not Disturb* (DND)[5], através da API `NotificationManager`[6], em detrimento da utilização do `AudioManager`[2]. Esta decisão teve por base as alterações introduzidas a partir do Android 6.0 (API 23), onde a alteração direta do modo de toque com `AudioManager` pode, em certos dispositivos, ativar implicitamente o modo DND[7]. Esta ativação automática impede o acesso a funcionalidades essenciais, como. Além disso, nas versões mais recentes do Android, a capacidade de interceptar chamadas ou aceder à identificação do autor da chamada (nome e número) programaticamente deixou de ser possível por razões de privacidade. Ao utilizar o `NotificationManager`[6] e a sua API `setInterruptionFilter`[8], é possível silenciar e restaurar o estado do dispositivo de forma controlada, desde que a aplicação disponha da permissão necessária (`ACCESS_NOTIFICATION_POLICY`). Esta abordagem maximiza a compatibilidade com as diversas versões Android.

3.3.4 Localização

Capítulo 4

Ferramentas

Capítulo 5

Implementação

Capítulo 6

Testes

Este é o capítulo de testes. É possível forçar a inclusão de todas as referências com [0].

Modo de matemática em texto $x = ma^2$ e em equação (duas formas):

$$x = ma^2$$

$$x = ma^2 \tag{6.1}$$

Capítulo 7

Conclusões

Referências

[1] Google Developers. Calendar provider, . URL https://developer.android.com/identity/providers/calendar-provider?hl=pt-br. Acedido em: 06-Março-2025.

[2] Android Developers. Alarmmanager, . URL https://developer.android.com/reference/android/app/AlarmManager. Acedido em: 10-Maio-2025.

[3] Android Developers. Power management - doze and app standby, . URL https://developer.android.com/training/monitoring-device-state/doze-standby. Acedido em: 10-Maio-2025.

[4] Android Developers. Workmanager overview, . URL https://developer.android.com/topic/libraries/architecture/workmanager. Acedido em: 10-Maio-2025.

[5] Android Developers. Notification policy access - dnd permission, . URL https://developer.android.com/guide/topics/ui/notifiers/notifications#dnd-access. Acedido em: 12-Maio-2025.

[6] Android Developers. Notificationmanager - setinterruptionfilter, . URL https://developer.android.com/reference/android/app/NotificationManager#setInterruptionFilter(int). Acedido em: 11-Maio-2025.

[7] Google Issue Tracker. Audiomanager silent mode and dnd limitations. URL https://issuetracker.google.com/issues/37067997. Acedido em: 11-Maio-2025.

[8] Android Developers. setinterruptionfilter reference, . URL https://developer.android.com/reference/android/app/NotificationManager#setInterruptionFilter(int). Acedido em: 12-Maio-2025.

Apêndice A

Exemplo de apêndice

Este é o primeiro parágrafo do apêndice.

Etiam ac leo a risus tristique nonummy. Donec dignissim tincidunt nulla. Vestibulum rhoncus molestie odio. Sed lobortis, justo et pretium lobortis, mauris turpis condimentum augue, nec ultricies nibh arcu pretium enim. Nunc purus neque, placerat id, imperdiet sed, pellentesque nec, nisl. Vestibulum imperdiet neque non sem accumsan laoreet. In hac habitasse platea dictumst. Etiam condimentum facilisis libero. Suspendisse in elit quis nisl aliquam dapibus. Pellentesque auctor sapien. Sed egestas sapien nec lectus. Pellentesque vel dui vel neque bibendum viverra. Aliquam porttitor nisl nec pede. Proin mattis libero vel turpis. Donec rutrum mauris et libero. Proin euismod porta felis. Nam lobortis, metus quis elementum commodo, nunc lectus elementum mauris, eget vulputate ligula tellus eu neque. Vivamus eu dolor.

Nulla in ipsum. Praesent eros nulla, congue vitae, euismod ut, commodo a, wisi. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Aenean nonummy magna non leo. Sed felis erat, ullamcorper in, dictum non, ultricies ut, lectus. Proin vel arcu a odio lobortis euismod. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Proin ut est. Aliquam odio. Pellentesque massa turpis, cursus eu, euismod nec, tempor congue, nulla. Duis viverra gravida mauris. Cras tincidunt. Curabitur eros ligula, varius ut, pulvinar in, cursus faucibus, augue.

Nulla mattis luctus nulla. Duis commodo velit at leo. Aliquam vulputate magna et leo. Nam vestibulum ullamcorper leo. Vestibulum condimentum rutrum mauris. Donec id mauris. Morbi molestie justo et pede. Vivamus eget turpis sed nisl cursus tempor. Curabitur mollis sapien condimentum nunc. In wisi nisl, malesuada at, dignissim sit amet, lobortis in, odio. Aenean consequat arcu a ante. Pellentesque porta elit sit amet orci. Etiam at turpis nec elit ultricies imperdiet. Nulla facilisi. In hac habitasse platea dictumst. Suspendisse viverra aliquam risus. Nullam pede justo, molestie nonummy, scelerisque eu, facilisis vel, arcu.