VotoInformado
Grupo 3

Alexandre Santos, Bruno Correia, Daniel Carlos, Henrique Laia, Vasco Colaço

Projeto Coletivo de Programação em Dispositivos Móveis
Engenharia Informática

Orientador: Prof. Doutor Paulo Fazendeiro

Novembro de 2025

Resumo

A aplicação VotoInformado foi desenvolvida com o objetivo de fornecer aos eleitores informação fiável e acessível sobre candidatos, partidos, programas eleitorais e eventos políticos. Os objetivos centrais do projeto são centralizar a informação política, facilitar a comparação entre candidatos e aumentar o nível de literacia eleitoral da população. A solução consiste numa aplicação Android que inclui funcionalidades como listas de candidatos filtráveis, páginas de partidos, secção de notícias, sondagens, um questionário de posicionamento político e um sistema de criação de petições pelos utilizadores, suportado por uma API customizada.

Palavras-chave

VotoInformado, Aplicação Móvel, Android, Literacia Política, Eleições, Firebase

Índice

Resumo iii

Índice v

Lista de Figuras ix

Lista de Tabelas xi

Lista de Acrônimos xiii

1 Introdução 1

1.1 Enquadramento e Motivação 1

1.2 Objetivos 1

1.3 Estrutura do Relatório 1

2 Arquitetura e Tecnologias 3

2.1 Arquitetura do Sistema 3

2.2 Tecnologias Utilizadas 3

2.2.1 Backend (API Customizada) 3

2.2.2 Firebase (Legado/Autenticação) 4

2.2.3 Bibliotecas Externas 4

3 Funcionalidades 5

3.1 Autenticação e Perfil 5

3.2 Requisitos Funcionais 6

3.3 Consulta de Candidatos 6

3.4 Notícias em Tempo Real 7

3.5 Sondagens e Estatísticas 8

3.6 Datas Importantes 9

3.7 Notificações de Debates 10

3.8 Bússola Política 10
3.9 Petições Públicas 11

4 Implementação **13**

4.1 Migração para API Personalizada 13
4.2 Parsing de Notícias (RSS) 14
4.3 Gestão de Imagens 14
4.4 Adapters e RecyclerViews 14
4.5 Motor de Quiz e Visualização Gráfica 14
4.5.1 Modelo de Dados 14
4.5.2 CompassView 14
4.6 Sistema de Petições 15
4.6.1 Estrutura de Dados 15
4.6.2 Upload de Imagens 15
4.6.3 Eliminação de Petições 16
4.7 Resolução de Problemas e Otimizações 16
4.7.1 Ligação de Candidatos em Sondagens 16
4.7.2 Conectividade com API (Render Cold Start) 16
4.7.3 Carregamento de Imagens de Perfil 17
4.7.4 Upload de Imagens em Petições 17
4.7.5 Estabilidade na Votação de Debates 17

5 Resultados e Testes **19**

5.1 Estado Atual do Projeto 19
5.2 Testes Realizados 19
5.2.1 Testes Manuais 19
5.2.2 Testes Unitários e Instrumentados 19
5.3 Análise de Desempenho 20

6 Conclusão e Trabalho Futuro **21**

6.1 Conclusão 21

6.2 Trabalho Futuro 21

Bibliografia **23**

A Anexos **25**

A.1 Código Fonte Relevante 25

Glossário **33**

Lista de Figuras

3.1 Ecrã de Login da aplicação. 5
3.2 Lista de candidatos apresentada ao utilizador. 7
3.3 Feed de notícias políticas. 8
3.4 Visualização gráfica dos resultados de uma sondagem. 9
3.5 Exemplo de notificação recebida antes e aquando de um debate. 10
3.6 Resultado do teste da Bússola Política. 11
3.7 Lista de petições públicas criadas pelos utilizadores. 12

Lista de Tabelas

2.1 Bibliotecas externas utilizadas no projeto. 4

3.1 Requisitos Funcionais do VotoInformado. 6

5.1 Exemplo de casos de teste executados. 19

API	Application Programming Interface (Interface de Programação de Aplicações)
HTTP	Hypertext Transfer Protocol (Protocolo de Transferência de Hipertexto)
JSON	JavaScript Object Notation
JWT	JSON Web Token
ODM	Object-Document Mapping (Mapeamento Objeto-Documento)
REST	Representational State Transfer (Transferência de Estado Representacional)
SDK	Software Development Kit (Kit de Desenvolvimento de Software)
UI	User Interface (Interface do Utilizador)
UX	User Experience (Experiência do Utilizador)
XML	Extensible Markup Language (Linguagem de Marcação Extensível)
UBI	Universidade da Beira Interior

Capítulo 1

Introdução

[1]

1.1 Enquadramento e Motivação

O projeto **VotoInformado** surge no âmbito da unidade curricular de Programação de Dispositivos Móveis, com o intuito de combater a desinformação e o alheamento político. Numa era em que a informação é abundante mas nem sempre fidedigna, torna-se crucial fornecer aos cidadãos ferramentas que centralizem dados oficiais e notícias relevantes sobre o panorama político nacional.

A aplicação visa simplificar o acesso a informações sobre candidatos, partidos, sondagens e eventos eleitorais, de modo a promover uma participação cívica mais consciente e informada.

1.2 Objetivos

Os principais objetivos deste projeto são:

- Desenvolver uma aplicação Android nativa, robusta e intuitiva.
- Centralizar informações sobre candidatos e partidos políticos.
- Disponibilizar sondagens eleitorais com visualização gráfica de dados.
- Fornecer um feed de notícias políticas atualizado em tempo real.
- Implementar um calendário de datas importantes para o processo eleitoral.

1.3 Estrutura do Relatório

Este relatório está organizado da seguinte forma:

[1] Este relatório foi produzido utilizando o template LaTeX para teses e dissertações da Universidade da Beira Interior [1], uma versão não oficial mantida pela comunidade.

- O **Capítulo 2** descreve a arquitetura do sistema e as tecnologias utilizadas.
- O **Capítulo 3** detalha as funcionalidades implementadas na aplicação.
- O **Capítulo 4** aborda aspetos técnicos da implementação e desafios encontrados.
- O **Capítulo 5** apresenta os resultados obtidos e a validação da solução.
- O **Capítulo 6** apresenta as conclusões e sugestões para trabalho futuro.

Capítulo 2

Arquitetura e Tecnologias

Este capítulo descreve a arquitetura de software adotada para o desenvolvimento da aplicação VotoInformado, bem como as tecnologias e bibliotecas utilizadas.

2.1 Arquitetura do Sistema

A aplicação foi desenvolvida para o sistema operativo Android [2] e utiliza a linguagem Java. A estrutura do projeto segue os padrões recomendados pela Google e organiza o código em componentes lógicos para facilitar a manutenção e escalabilidade.

A arquitetura baseia-se na separação de responsabilidades:

- **Activities e Fragments**: Responsáveis pela interface de utilizador (UI) e interação com o utilizador. Exemplos incluem `HomeActivity`, `NoticiasFragment` e `CandidatosFragment`.
- **Adapters**: Fazem a ponte entre os dados e os componentes de visualização de listas (`RecyclerView`).
- **Modelos (Classes)**: Representam as entidades de dados, como `Candidato`, `Noticia` e `Sondagem`.
- **Utils/Helpers**: Classes utilitárias para acesso a dados e operações comuns.

2.2 Tecnologias Utilizadas

2.2.1 Backend (API Customizada)

O backend da aplicação, inicialmente projetado em Firebase, foi migrado para uma solução personalizada para responder a requisitos de maior complexidade técnica e controlo. A arquitetura atual utiliza:

- **Node.js & Express** [3, 4]: Servidor de aplicação que expõe uma API RESTful para comunicação com a aplicação móvel.
- **MongoDB** [5]: Base de dados NoSQL orientada a documentos, escolhida pela sua flexibilidade e escalabilidade.

- **Mongoose** [6]: Biblioteca de modelação de objetos (ODM) para interação estruturada com o MongoDB.

2.2.2 Firebase (Legado/Autenticação)

Embora a persistência de dados tenha sido migrada, o Firebase continua a desempenhar um papel auxiliar:

- **Firebase Authentication**: Mantido para gestão segura de identidades e sessões de utilizador.
- **Nota de Migração**: A transição do Firebase Firestore para a API personalizada foi motivada por uma sugestão pedagógica para aumentar a complexidade do projeto e demonstrar competências de desenvolvimento *full-stack*.

2.2.3 Bibliotecas Externas

Para enriquecer a funcionalidade da aplicação, foram integradas diversas bibliotecas *open--source*:

Tabela 2.1: Bibliotecas externas utilizadas no projeto.

Biblioteca	**Propósito**
Picasso	Carregamento e colocação em cache de imagens assíncronos.
MPAndroidChart	Criação de gráficos para visualização de sondagens.
CircleImageView	Exibição de imagens de perfil circulares.
Retrofit/Gson	(Legado) Análise sintática de dados JSON.

- **Picasso** [7]: Biblioteca poderosa para o carregamento e cache de imagens em Android. Resolve problemas complexos de gestão de memória e reciclagem de vistas em listas.
- **MPAndroidChart** [8]: Utilizada para a criação de gráficos interativos e visualmente apelativos na secção de sondagens.
- **CircleImageView**: Permite a exibição de imagens de perfil com formato circular de forma simples e eficiente.
- **Retrofit/Gson** [9] (Legado): Inicialmente utilizadas para análise sintática de JSON, foram substituídas pela integração direta com o Firestore, mas fizeram parte do processo de desenvolvimento.

Capítulo 3

Funcionalidades

A aplicação VotoInformado oferece um conjunto de funcionalidades desenhadas para informar o eleitor.

3.1 Autenticação e Perfil

A segurança e personalização são garantidas através de um sistema de autenticação robusto.

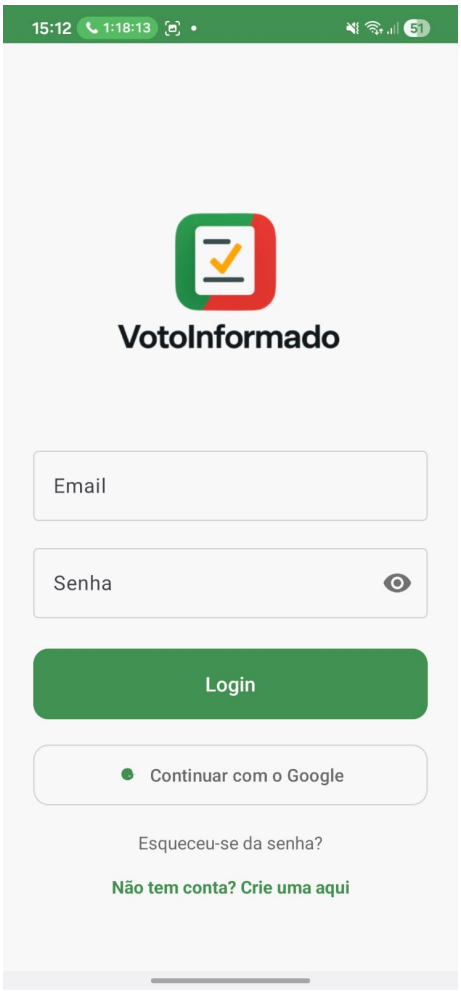

Figura 3.1: Ecrã de Login da aplicação.

- **Login e Registo**: Os utilizadores podem criar conta com email e password ou

utilizar a sua conta Google para um acesso mais rápido.

- **Gestão de Sessão**: A aplicação mantém a sessão do utilizador ativa, o que permite acesso direto sem necessidade de login constante.

3.2 Requisitos Funcionais

A tabela abaixo resume as principais funcionalidades implementadas.

Tabela 3.1: Requisitos Funcionais do VotoInformado.

ID	Funcionalidade	Descrição
RF01	Autenticação	Permitir login e registo via Email/Password e Google.
RF02	Listar Candidatos	Visualizar lista de candidatos com foto e partido.
RF03	Detalhe Candidato	Ver biografia, propostas e cargos de um candidato.
RF04	Feed Notícias	Ler notícias políticas atualizadas via RSS.
RF05	Sondagens	Visualizar gráficos de intenção de voto.

3.3 Consulta de Candidatos

Esta é uma das funcionalidades centrais da aplicação.

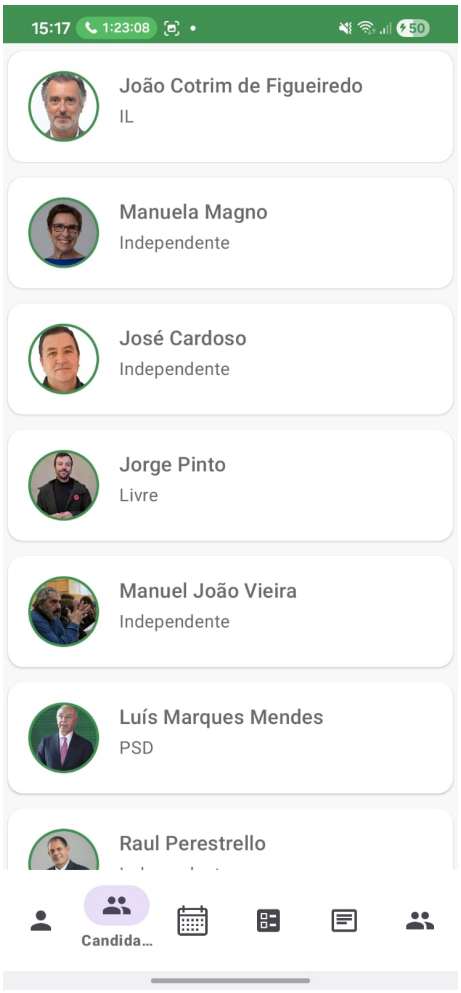

Figura 3.2: Lista de candidatos apresentada ao utilizador.

- **Listagem**: Apresenta uma lista de todos os candidatos registados na base de dados.
- **Detalhes**: Ao selecionar um candidato, o utilizador tem acesso a uma ficha detalhada que inclui:
  - Biografia e dados pessoais (idade, profissão).
  - Partido político e cargos desempenhados.
  - Lista de propostas eleitorais principais.

3.4 Notícias em Tempo Real

Para manter o utilizador atualizado sobre a atualidade política:

Figura 3.3: Feed de notícias políticas.

- **Feed RSS**: Integração com o feed de política da RTP Notícias.
- **Pesquisa**: Barra de pesquisa que permite filtrar notícias por palavras-chave em tempo real.
- **Visualização**: Abertura da notícia completa num navegador interno ou externo.

3.5 Sondagens e Estatísticas

A secção de sondagens permite visualizar as tendências de voto.

- **Gráficos**: Visualização clara dos resultados através de gráficos de barras.
- **Ficha Técnica**: Informação sobre a amostra, margem de erro e empresa responsável pela sondagem.

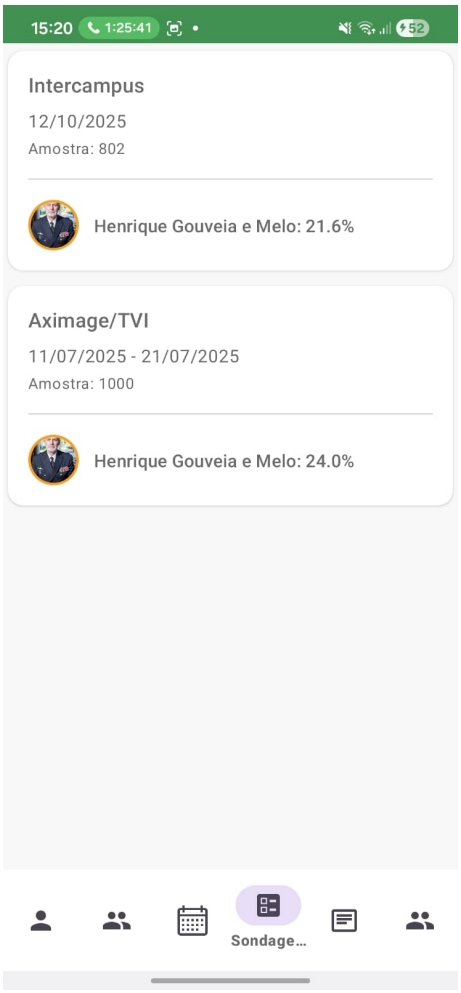

Figura 3.4: Visualização gráfica dos resultados de uma sondagem.

3.6 Datas Importantes

Um calendário eleitoral organizado por categorias que lista eventos cruciais. A interface foi reestruturada para utilizar um sistema de **abas (Tabs)**, o que permite ao utilizador filtrar facilmente entre:

- **Datas**: Prazos oficiais, dias de reflexão e o dia das eleições.
- **Debates**: Calendário de debates televisivos entre candidatos.
- **Entrevistas**: Agendamento de entrevistas aos candidatos.

Esta organização facilita a consulta e garante que o eleitor não perde prazos ou eventos importantes.

3.7 Notificações de Debates

Para assegurar que os eleitores acompanham os debates televisivos, a aplicação dispõe de um sistema de notificações.

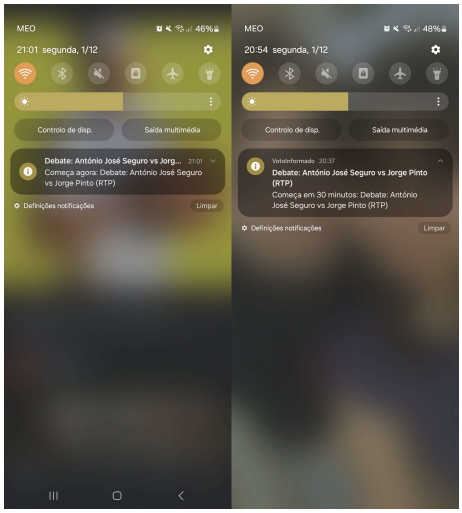

Figura 3.5: Exemplo de notificação recebida antes e aquando de um debate.

- **Alerta Antecipado**: O utilizador é notificado com antecedência sobre o início da transmissão.

- **Alerta ao Iniciar**: O utilizador é notificado imediatamente quando a transmissão começa.

- **Conteúdo**: A notificação indica os candidatos intervenientes e o canal televisivo.

3.8 Bússola Política

A Bússola Política é uma ferramenta interativa que permite ao utilizador descobrir o seu posicionamento político através de um questionário. Esta funcionalidade foi inspirada no conhecido website *The Political Compass* [10], adaptando o conceito para o contexto da aplicação.

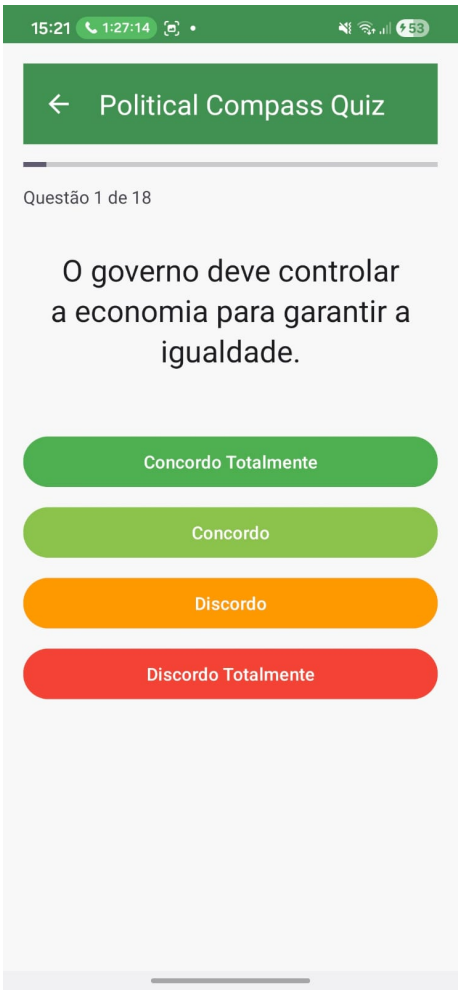

Figura 3.6: Resultado do teste da Bússola Política.

- **Questionário**: Um conjunto de perguntas sobre temas económicos e sociais, onde o utilizador expressa o seu grau de concordância.

- **Visualização**: O resultado é apresentado num gráfico bidimensional (Eixo Económico vs. Eixo Social), e compara a posição do utilizador com a de vários candidatos e partidos.

- **Partilhar**: O utilizador pode partilhar o resultado do teste como imagem através de um botão.

3.9 Petições Públicas

Para promover a participação cívica ativa, a aplicação inclui um sistema de petições públicas.

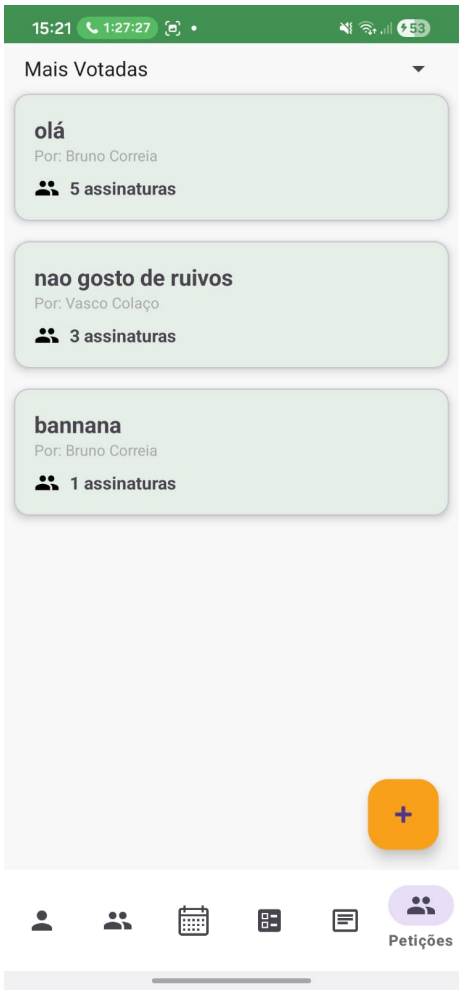

Figura 3.7: Lista de petições públicas criadas pelos utilizadores.

- **Criação**: Qualquer utilizador autenticado pode criar uma nova petição, na qual define um título, uma descrição e adiciona uma **imagem ilustrativa**.
- **Assinatura**: Os utilizadores podem apoiar causas ao assinar petições existentes.
- **Ordenação**: A lista de petições pode ser ordenada por popularidade (mais votadas) ou por data (mais recentes), o que facilita a descoberta de causas relevantes.

Capítulo 4

Implementação

Este capítulo aborda os desafios técnicos e as soluções de implementação adotadas durante o desenvolvimento.

4.1 Migração para API Personalizada

Inicialmente, o projeto utilizava o Firebase Firestore para persistência de dados diretamente na aplicação móvel. No entanto, em resposta a uma **sugestão pedagógica do docente** para aumentar a complexidade técnica do projeto e permitir uma solução mais personalizada, migrou-se para uma arquitetura com uma **API REST intermédia**.

Esta nova arquitetura consiste numa API desenvolvida em **Node.js**, que atua como *backend* para a aplicação Android. A API encontra-se alojada na plataforma **Render**, tirando partido do seu plano gratuito para disponibilizar os serviços publicamente. A API utiliza uma base de dados **MongoDB** para persiência de dados e comunica através da biblioteca **Mongoose**. A autenticação é gerida através de **JWT (JSON Web Tokens)**, o que garante uma sessão segura e independente do estado do servidor.

```
1  public interface ApiService {
2      @GET("api/candidates")
3      Call<List<Candidato>> getCandidates();
4
5      @POST("api/petitions")
6      Call<Peticao> createPetition(@Body Peticao peticao);
7
8      @Multipart
9      @POST("api/petitions/upload")
10     Call<Map<String, String>> uploadPetitionImage(@Part MultipartBody.Part
           image);
11 }
```

Listing 4.1: Definição da Interface Retrofit

4.2 Parsing de Notícias (RSS)

Para obter as notícias, foi implementada a classe `NoticiasFetcher`. Esta classe realiza uma requisição HTTP ao feed RSS da RTP e faz a análise sintática do XML resultante com o `DocumentBuilder`. Um desafio interessante foi a extração de imagens, que não vinham num campo explícito, mas sim embutidas na descrição HTML. Foi utilizada uma expressão regular (Regex) para extrair o atributo `src` das tags `<img>`.

4.3 Gestão de Imagens

As imagens das petições e dos candidatos são armazenadas localmente no servidor (*backend*) e servidas como ficheiros estáticos. A base de dados armazena apenas o URL relativo da imagem. Na aplicação Android, a biblioteca **Picasso** [7] é utilizada para carregar e exibir estas imagens de forma assíncrona, que gere automaticamente a transferência, cache e redimensionamento das imagens.

4.4 Adapters e RecyclerViews

A exibição de listas eficientes foi conseguida através da implementação de `RecyclerView.Adapter` personalizados. O padrão *ViewHolder* é utilizado para reciclar as vistas, o que garante um deslize suave mesmo com listas longas de candidatos ou notícias.

4.5 Motor de Quiz e Visualização Gráfica

A funcionalidade da Bússola Política envolveu o desenvolvimento de um motor de cálculo de pontuação e uma vista personalizada para o gráfico.

4.5.1 Modelo de Dados

A classe `Question` encapsula o texto da pergunta, o peso e o eixo a que pertence (Económico ou Social). A classe `CompassCandidate` estende a informação base de um candidato e adiciona as coordenadas (x, y) da sua posição política estimada.

4.5.2 CompassView

Para desenhar o gráfico, foi criada a classe `CompassView` que estende `View`.

- **onDraw()**: Este método é sobrescrito para desenhar os eixos, as quadrículas de fundo, e os pontos que representam o utilizador e os candidatos.

- **Transformação de Coordenadas**: As coordenadas lógicas (de -10 a +10) são convertidas para coordenadas de ecrã (pixels) tendo em conta a largura e altura da vista, o que garante que o gráfico se adapta a diferentes tamanhos de ecrã.

4.6 Sistema de Petições

O sistema de petições foi adaptado para consumir a nova API e mantém a autenticação de utilizadores via Firebase Auth (gerida pelo backend).

4.6.1 Estrutura de Dados

A classe `Peticao` foi atualizada para mapear os documentos MongoDB.

```
1 public class Peticao {
2     @SerializedName("_id")
3     private String id;
4     private String titulo;
5     private String descricao;
6     private List<String> assinaturas; // Lista de UIDs
7     private String imageUrl; // URL da imagem
8     // ... getters e setters
9 }
```

Listing 4.2: Estrutura simplificada da classe Peticao

4.6.2 Upload de Imagens

Para permitir o upload de imagens nas petições sem custos adicionais de serviços cloud, implementou-se um sistema de **armazenamento local** no servidor.

- **Backend**: Utiliza o middleware `multer` para receber ficheiros *multipart/form-data* e guardá-los numa pasta local `uploads/`. O servidor serve estes ficheiros estaticamente.

- **Android**: A atividade `CreatePeticaoActivity` seleciona a imagem da galeria, converte-a num ficheiro temporário e envia-a para o endpoint `/api/petitions/upload` antes de submeter os dados da petição.

4.6.3 Eliminação de Petições

Foi implementada a funcionalidade de eliminação de petições através do método HTTP `DELETE`. O endpoint `/api/petitions/:id` permite remover documentos da coleção `petitions` no MongoDB, o que garante a integridade dos dados. A aplicação Android expõe esta funcionalidade através da classe utilitária `DatabaseHelper`.

4.7 Resolução de Problemas e Otimizações

Durante a fase final de testes e integração, foram identificados e resolvidos três problemas críticos que afetavam a experiência do utilizador e a estabilidade da aplicação.

4.7.1 Ligação de Candidatos em Sondagens

Foi detetada uma inconsistência na ligação entre os resultados das sondagens e os perfis dos candidatos. O problema devia-se a uma discrepância nos identificadores: a base de dados utilizava IDs do MongoDB (`_id`), enquanto alguns resultados de sondagens referenciavam candidatos por um ID textual (e.g., "andre_pestana").

Solução: Implementou-se uma lógica de correspondência robusta nos adaptadores (`ResultadoSondagemAdapte` `SondagemAdapter`) e no `HomeFragment`. O algoritmo tenta agora corresponder o candidato sequencialmente por:

1. ID do MongoDB (`_id`).
2. ID textual personalizado (`id`).
3. Nome do candidato (como recurso final, insensível a maiúsculas/minúsculas).

4.7.2 Conectividade com API (Render Cold Start)

A API, alojada no plano gratuito do serviço Render, entra em modo de suspensão após períodos de inatividade. O tempo de arranque inicial ("cold start") excedia frequentemente o timeout padrão de 1.5 segundos definido no cliente Android, o que levava a aplicação a falhar silenciosamente para o URL local (`localhost`).

Solução: O timeout de conexão e leitura no `ApiClient` foi aumentado para **15 segundos**. Isto garante que a aplicação aguarda o tempo suficiente para que o servidor acorde, assegurando assim uma ligação bem-sucedida mesmo após inatividade.

4.7.3 Carregamento de Imagens de Perfil

Identificou-se que as imagens de perfil dos utilizadores não eram carregadas corretamente em dispositivos físicos. A causa raiz era dupla: o backend guardava URLs absolutos (e.g., `http://localhost:3000/...`) durante o registo, que são inacessíveis externamente, e os comentários não persistiam a foto do autor.

Solução:

- **Backend**: O controlador de autenticação foi atualizado para guardar apenas o **caminho relativo** da imagem (e.g., `/uploads/profiles/...`). Adicionalmente, a criação de comentários foi corrigida para persistir o URL da foto do autor.
- **Frontend**: Implementou-se uma lógica de validação no cliente (`HomeFragment`, `ComentarioAdapter`). A aplicação deteta agora URLs legados que contêm "localhost" ou "127.0.0.1", converte-os em caminhos relativos e prefixa dinamicamente o URL base correto da API (`ApiClient.getBaseUrl()`).

4.7.4 Upload de Imagens em Petições

Foi detetado que o upload de imagens falhava em certos dispositivos ou quando a imagem provinha de fontes específicas (como Google Photos), o que resultava em falhas na obtenção do tipo de ficheiro (MIME type) ou do nome original.

Solução: A lógica de seleção de ficheiros em `CreatePeticaoActivity` foi reescrita para ser mais robusta.

- **Resolução de Nomes**: Tenta-se extrair o nome real do ficheiro via `OpenableColumns` do Content Provider. Se falhar, gera-se um nome seguro.
- **Tipos de Ficheiro**: Implementou-se um mecanismo de *fallback* que assume o tipo JPEG caso o sistema não consiga determinar o tipo MIME, o que previne exceções do tipo `NullPointerException`.

4.7.5 Estabilidade na Votação de Debates

A funcionalidade de votação em debates apresentava instabilidade (falhas) quando os dados de votos continham inconsistências, especificamente quando o ID do candidato votado era nulo (dados corrompidos ou legados).

Solução: Adicionaram-se verificações de nulidade explícitas no método `processVotes` da `DebateVoteActivity`. O algoritmo ignora agora votos malformados em vez de interromper a execução, o que garante que a aplicação se mantém funcional mesmo com dados imperfeitos.

Capítulo 5

Resultados e Testes

5.1 Estado Atual do Projeto

A aplicação encontra-se num estado funcional estável, com todas as funcionalidades principais implementadas e operacionais. A navegação entre ecrãs é fluida e a integração com o Firebase responde com latência reduzida.

5.2 Testes Realizados

Para garantir a qualidade do software, foram realizadas várias etapas de verificação:

5.2.1 Testes Manuais

Foram realizados testes exaustivos em emuladores e dispositivos físicos para validar:

- O fluxo de registo e login.
- A correta visualização dos dados dos candidatos.
- A resiliência da aplicação a falhas de rede (tratamento de erros no carregamento de notícias).
- A adaptação da interface ao Modo Escuro.

Tabela 5.1: Exemplo de casos de teste executados.

ID	Ação	Resultado Esperado	Estado
CT01	Login com credenciais inválidas	Exibir mensagem de erro	Passou
CT02	Carregar lista de notícias	Exibir lista com títulos e imagens	Passou
CT03	Clicar em candidato	Abrir detalhe do candidato	Passou

5.2.2 Testes Unitários e Instrumentados

O projeto inclui uma estrutura para testes unitários (JUnit) e testes de interface (Espresso), permitindo a verificação automática de componentes críticos e fluxos de utilizador.

5.3 Análise de Desempenho

A utilização do *Android Profiler* permitiu identificar e corrigir fugas de memória, especialmente no carregamento de imagens, o que confirma a eficácia da introdução da biblioteca Picasso.

Capítulo 6

Conclusão e Trabalho Futuro

6.1 Conclusão

O projeto VotoInformado atingiu os seus objetivos principais, o que resultou numa aplicação móvel capaz de informar os eleitores de forma clara e acessível. A evolução da arquitetura para uma **API intermédia em Node.js** com base de dados **MongoDB**, provou ser uma decisão acertada, o que garante maior segurança, centralização da lógica e escalabilidade da solução.

A aprendizagem adquirida durante o desenvolvimento, nomeadamente na gestão de dependências, chamadas assíncronas e design de interfaces Android, foi valiosa para a equipa.

6.2 Trabalho Futuro

Apesar de funcional, a aplicação tem margem para evolução. Algumas sugestões para versões futuras incluem:

- **Notificações Push**: Alertar os utilizadores para novas sondagens ou notícias urgentes.
- **Comparador de Candidatos**: Uma ferramenta para colocar lado a lado as propostas de dois candidatos.
- **Gamificação**: Introduzir quizzes sobre política para incentivar a aprendizagem de forma lúdica.
- **Suporte Offline**: Melhorar a cache de dados para permitir a consulta básica mesmo sem ligação à internet.

Bibliografia

[1] M. C. da Silva Filho, "template-ubi-latex: Um modelo em LaTeX para a escrita de teses e dissertações da Universidade da Beira Interior (UBI)," https://github.com/manoelcampos/template-ubi-latex, 2016, acedido em 1 de Dezembro de 2025. 1

[2] Google Developers, "Android Developers Documentation," https://developer.android.com/, 2025. 3

[3] OpenJS Foundation, "Node.js," https://nodejs.org/, 2025. 3

[4] StrongLoop, IBM, and other contributors, "Express - Fast, unopinionated, minimalist web framework for Node.js," https://expressjs.com/, 2025. 3

[5] MongoDB, Inc., "MongoDB: The Developer Data Platform," https://www.mongodb.com/, 2025. 3

[6] Automattic, "Mongoose ODM," https://mongoosejs.com/, 2025. 4

[7] Square, Inc., "Picasso: A powerful image downloading and caching library for Android," https://square.github.io/picasso/, 2025. 4, 14

[8] P. Jahoda, "MPAndroidChart," https://github.com/PhilJay/MPAndroidChart, 2025. 4

[9] Square, Inc., "Retrofit: A type-safe HTTP client for Android and Java," https://square.github.io/retrofit/, 2025. 4

[10] Pace News Limited, "The Political Compass," https://www.politicalcompass.org/, 2025, acedido em 1 de Dezembro de 2025. 10

Apêndice A

Anexos

Esta secção contém excertos de código relevantes que detalham a implementação do sistema VotoInformado.

A.1 Código Fonte Relevante

Cliente da API - ApiClient.java

```
1  package pt.ubi.pdm.votoinformado.api;
2
3  import android.util.Log;
4  import java.io.IOException;
5  import java.net.HttpURLConnection;
6  import java.net.URL;
7  import java.util.concurrent.CountDownLatch;
8  import retrofit2.Retrofit;
9  import retrofit2.converter.gson.GsonConverterFactory;
10
11 public class ApiClient {
12     private static final String RENDER_URL = "https://api-votoinformado.
           onrender.com/";
13     private static final String LOCAL_URL = "http://10.0.2.2:3000/"; //
           Localhost for Android Emulator
14     private static final String LOCAL_IP_URL = "http://10.250.134.7:3000/"; //
           Local IP for Physical Device
15
16     private static volatile ApiClient instance;
17     private static final Object lock = new Object();
18     private final Retrofit retrofit;
19     private static String baseUrl;
20
21     private ApiClient(Retrofit retrofit) {
22         this.retrofit = retrofit;
23     }
24
25     private static boolean isUrlAvailable(String urlString) {
26         try {
27             URL url = new URL(urlString);
```

```
28              HttpURLConnection connection = (HttpURLConnection) url.
                    openConnection();
29              connection.setRequestMethod("HEAD");
30              connection.setConnectTimeout(15000); // 15 seconds timeout for
                    Render cold start
31              connection.setReadTimeout(15000);
32              connection.connect();
33              int responseCode = connection.getResponseCode();
34              return (responseCode >= 200 && responseCode < 300);
35          } catch (IOException e) {
36              return false;
37          }
38      }
39
40      public static ApiClient getInstance() {
41          if (instance == null) {
42              synchronized (lock) {
43                  if (instance == null) {
44                      CountDownLatch latch = new CountDownLatch(1);
45                      new Thread(() -> {
46                          if (isUrlAvailable(RENDER_URL)) {
47                              baseUrl = RENDER_URL;
48                          } else if (isUrlAvailable(LOCAL_IP_URL)) {
49                              baseUrl = LOCAL_IP_URL;
50                          } else {
51                              baseUrl = LOCAL_URL;
52                          }
53                          Log.d("ApiClient", "Using base URL: " + baseUrl);
54
55                          Retrofit retrofit = new Retrofit.Builder()
56                                  .baseUrl(baseUrl)
57                                  .addConverterFactory(GsonConverterFactory.
                                        create())
58                                  .build();
59
60                          instance = new ApiClient(retrofit);
61                          latch.countDown();
62                      }).start();
63
64                      try {
65                          latch.await(); // Block until initialization is
                                complete
66                      } catch (InterruptedException e) {
67                          Thread.currentThread().interrupt();
68                          // Handle error, maybe fallback to a default
69                          if (instance == null) {
```

```
70                              Log.e("ApiClient", "Initialization interrupted,
                                    falling back to local URL");
71                              baseUrl = LOCAL_URL;
72                              Retrofit retrofit = new Retrofit.Builder()
73                                      .baseUrl(baseUrl)
74                                      .addConverterFactory(GsonConverterFactory.
                                            create())
75                                      .build();
76                              instance = new ApiClient(retrofit);
77                          }
78                      }
79                  }
80              }
81          }
82          return instance;
83      }
84
85      public ApiService getApiService() {
86          return retrofit.create(ApiService.class);
87      }
88
89      public static String getBaseUrl() {
90          if (instance == null) {
91              getInstance(); // ensure initialization
92          }
93          return baseUrl;
94      }
95  }
```

Listing A.1: Cliente da API para comunicação com o servidor.

Atividade Principal - HomeActivity.java

```
1  package pt.ubi.pdm.votoinformado.activities;
2
3  import android.Manifest;
4  import android.content.Intent;
5  import android.content.pm.PackageManager;
6  import android.os.Build;
7  import android.os.Bundle;
8  import android.view.MenuItem;
9
10 import androidx.activity.result.ActivityResultLauncher;
11 import androidx.activity.result.contract.ActivityResultContracts;
12 import androidx.annotation.NonNull;
13 import androidx.appcompat.app.AppCompatActivity;
```

```
14 import androidx.core.content.ContextCompat;
15 import androidx.fragment.app.Fragment;
16 import androidx.work.ExistingPeriodicWorkPolicy;
17 import androidx.work.PeriodicWorkRequest;
18 import androidx.work.WorkManager;
19
20 import com.google.android.material.bottomnavigation.BottomNavigationView;
21
22 import java.util.concurrent.TimeUnit;
23
24 import pt.ubi.pdm.votoinformado.R;
25 import pt.ubi.pdm.votoinformado.activities.notificacoes.SyncDatesWorker;
26 import pt.ubi.pdm.votoinformado.fragments.CandidatosFragment;
27 import pt.ubi.pdm.votoinformado.fragments.ChooseEventTypeFragment;
28 import pt.ubi.pdm.votoinformado.fragments.HomeFragment;
29 import pt.ubi.pdm.votoinformado.fragments.NoticiasFragment;
30 import pt.ubi.pdm.votoinformado.fragments.PeticoesFragment;
31 import pt.ubi.pdm.votoinformado.fragments.SondagensFragment;
32
33 public class HomeActivity extends AppCompatActivity implements
      BottomNavigationView.OnNavigationItemSelectedListener {
34
35     // Launcher para pedir a permissão de notificação
36     private final ActivityResultLauncher<String> requestPermissionLauncher =
           registerForActivityResult(new ActivityResultContracts.RequestPermission
           (), isGranted -> {
37     });
38
39     @Override
40     protected void onCreate(Bundle savedInstanceState) {
41         super.onCreate(savedInstanceState);
42
43         // Check if user is logged in via shared preferences
44         android.content.SharedPreferences prefs = getSharedPreferences("
               user_session", MODE_PRIVATE);
45         String token = prefs.getString("auth_token", null);
46
47         if (token == null || token.isEmpty()) {
48             startActivity(new Intent(this, LoginActivity.class));
49             finish();
50             return;
51         }
52
53         setContentView(R.layout.activity_home);
54
55         BottomNavigationView bottomNav = findViewById(R.id.bottom_navigation);
56         bottomNav.setOnNavigationItemSelectedListener(this);
```

```
57
58          // Load the default fragment
59          if (savedInstanceState == null) {
60              loadFragment(new HomeFragment());
61          }
62
63          // Pede permissão de notificações (para Android 13+)
64          askNotificationPermission();
65
66          //funcao que vai tratar das notificacoes mesmo sem a necessidade de ter
                a aplicacao aberta
67          scheduleDateSync();
68      }
69
70      private void askNotificationPermission() {
71          // Apenas necessário para API 33+ (Android 13)
72          if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.TIRAMISU) {
73              if (ContextCompat.checkSelfPermission(this, Manifest.permission.
                    POST_NOTIFICATIONS) != PackageManager.PERMISSION_GRANTED) {
74                  // Pede diretamente a permissão
75                  requestPermissionLauncher.launch(Manifest.permission.
                        POST_NOTIFICATIONS);
76              }
77          }
78      }
79
80      private void scheduleDateSync() {
81          //criamos um pedido de trabalho periódico
82          PeriodicWorkRequest syncDatesRequest =
83                  new PeriodicWorkRequest.Builder(SyncDatesWorker.class, 24,
                        TimeUnit.HOURS)
84                          .build();                //executar a logica do
                                SyncDatesWorker.class a cada 24h
85
86          //entregamos o pedido de trabalho ao WorkManager
87          WorkManager.getInstance(this).enqueueUniquePeriodicWork(
88                  "syncDatesWork",
89                  ExistingPeriodicWorkPolicy.KEEP,
90                  syncDatesRequest);
91      }
92
93      private boolean loadFragment(Fragment fragment) {
94          if (fragment != null) {
95              getSupportFragmentManager().beginTransaction()
96                      .replace(R.id.fragment_container, fragment)
97                      .commit();
98              return true;
```

```
99          }
100         return false;
101     }
102
103     @Override
104     public boolean onNavigationItemSelected(@NonNull MenuItem item) {
105         Fragment fragment = null;
106
107         int itemId = item.getItemId();
108         if (itemId == R.id.nav_home) {
109             fragment = new HomeFragment();
110         } else if (itemId == R.id.nav_candidatos) {
111             fragment = new CandidatosFragment();
112         } else if (itemId == R.id.nav_eventos) {
113             fragment = new pt.ubi.pdm.votoinformado.fragments.
                    ImportantDatesHostFragment();
114         } else if (itemId == R.id.nav_sondagens) {
115             fragment = new SondagensFragment();
116         } else if (itemId == R.id.nav_noticias) {
117             fragment = new NoticiasFragment();
118         } else if (itemId == R.id.nav_peticoes) {
119             fragment = new PeticoesFragment();
120         }
121
122         return loadFragment(fragment);
123     }
124 }
```

Listing A.2: Atividade principal que gere a navegação e o fragmento inicial.

Modelo de Dados - Candidato.java

```
1  package pt.ubi.pdm.votoinformado.classes;
2
3  import android.annotation.SuppressLint;
4  import android.content.Context;
5  import android.util.Log;
6
7      // @Exclude removed
8
9  import java.io.Serializable;
10
11 import pt.ubi.pdm.votoinformado.R;
12
13 import com.google.gson.annotations.SerializedName;
14
```

```java
public class Candidato implements Serializable {

    @SerializedName("_id")
    private String id;

    @SerializedName("id")
    private String stringId;

    private String nome;
    private String partido;
    // private String fotoNome;
    private String profissao;
    private String cargosPrincipais;
    private String biografiaCurta;
    private String siteOficial;

    public Candidato() {
        // Construtor vazio necessário para o Firestore
    }

    // Getters and Setters with PropertyName annotations

    public String getId() { return id; }
    public void setId(String id) { this.id = id; }

    public String getStringId() { return stringId; }
    public void setStringId(String stringId) { this.stringId = stringId; }

    public String getNome() { return nome; }
    public void setNome(String nome) { this.nome = nome; }

    public String getPartido() { return partido; }
    public void setPartido(String partido) { this.partido = partido; }

    // fotoNome removed


    public String getProfissao() { return profissao; }
    public void setProfissao(String profissao) { this.profissao = profissao; }

    public String getCargosPrincipais() { return cargosPrincipais; }
    public void setCargosPrincipais(String cargosPrincipais) { this.
        cargosPrincipais = cargosPrincipais; }

    public String getBiografiaCurta() { return biografiaCurta; }
    public void setBiografiaCurta(String biografiaCurta) { this.biografiaCurta
        = biografiaCurta; }
```

```
60
61      public String getSiteOficial() { return siteOficial; }
62      public void setSiteOficial(String siteOficial) { this.siteOficial =
            siteOficial; }
63
64      @SerializedName("photoUrl")
65      private String photoUrl;
66
67      public String getPhotoUrl() {
68          return photoUrl;
69      }
70
71      public void setPhotoUrl(String photoUrl) {
72          this.photoUrl = photoUrl;
73      }
74  }
```

Listing A.3: Classe que representa o modelo de dados de um Candidato.

Glossário

API	*Application Programming Interface*. Uma interface que define interações entre múltiplas aplicações de software ou intermediários de hardware-software mistos. Permite que diferentes sistemas comuniquem entre si de forma padronizada.
Backend	A parte de um sistema ou aplicação de computador que não é diretamente acedida pelo utilizador, tipicamente responsável por armazenar e manipular dados, bem como pela lógica de negócio.
Frontend	A parte de um sistema ou aplicação de computador com a qual o utilizador interage diretamente. É sinónimo de interface de utilizador (UI).
Firebase	Uma plataforma desenvolvida pela Google para a criação de aplicações móveis e web. Fornece serviços como autenticação, bases de dados em tempo real, armazenamento e alojamento.
Git	Um sistema de controlo de versões distribuído para registar alterações no código-fonte durante o desenvolvimento de software.
JSON	*JavaScript Object Notation*. Um formato leve de troca de dados, fácil de ler e escrever para humanos e fácil de interpretar e gerar para as máquinas.
MongoDB	Um programa de base de dados orientado a documentos e multiplataforma. Classificado como uma base de dados NoSQL, o MongoDB utiliza documentos do tipo JSON com esquemas opcionais.
Node.js	Um ambiente de execução de JavaScript de código aberto, multiplataforma e de backend que funciona no motor V8 e executa código JavaScript fora de um navegador web.
REST	*Representational State Transfer*. Um estilo de arquitetura de software que define um conjunto de restrições a serem usadas para a criação de serviços Web.
Retrofit	Um cliente HTTP seguro para Android e Java, que facilita a comunicação com APIs REST.
SDK	*Software Development Kit*. Uma coleção de ferramentas de desenvolvimento de software num único pacote instalável.
UI	*User Interface* (Interface de Utilizador). O espaço onde ocorrem as interações entre humanos e máquinas.
UX	*User Experience* (Experiência do Utilizador). As emoções e atitudes de uma pessoa sobre a utilização de um determinado produto, sistema ou serviço.
LaTeX	Conjunto de macros para o processador de textos TeX, utilizado amplamente para a produção de textos matemáticos e científicos devido à sua alta qualidade tipográfica.