

Libruary

Desenvolvimento de Aplicações para Dispositivos Móveis

Licenciatura em Tecnologias da Informação

Nome do(s) autor(s):

Daniel Silva | 113900

Gabriel Cravo | 87689

Águeda | 09 de junho de 2024

Libruary

Desenvolvimento de Aplicações para Dispositivos Móveis

Licenciatura em Tecnologias da Informação

Nome do (s) autor(s):

Daniel Silva | 113900

Gabriel Cravo | 87689

Docente(s) responsável(is) da UC:

Gonçalo Marques

Águeda | 09 de junho de 2024

Resumo

O presente trabalho consiste na apresentação de uma proposta para a criação de uma aplicação móvel Android denominada "Libruary". Esta aplicação permite que os utilizadores armazenem livros do seu interesse nas suas contas pessoais e visualizem as avaliações desses livros, caso outros utilizadores já tenham deixado *reviews*.

Primeiramente, delimitou-se o escopo do problema com base no tema escolhido. Em seguida, realizou-se o planeamento do projeto, identificando e caracterizando os requisitos necessários, tanto funcionais quanto não funcionais. Posteriormente, partiu-se para o design do sistema, onde foram identificados e descritos os casos de uso, além da criação de protótipos de baixa e alta-fidelidade. Por fim, na fase de implementação, desenvolveu-se a camada de apresentação e a programação server-side.

O trabalho culmina com uma reflexão final acerca deste percurso ao longo do desenvolvimento da aplicação "Libruary", bem como algumas sugestões para colmatar lacunas identificadas no sistema.

Palavras-chave: aplicação, android, API, livro.

Abstract

This work consists of presenting a proposal for the creation of an Android mobile application called “Libruary”. This application allows users to store books of interest in their personal accounts and view the ratings of those books, if other users have already left reviews.

First, the scope of the problem was delimited based on the chosen topic. Next, the project was planned, identifying and characterizing the necessary requirements, both functional and non-functional. Next came system design, where use cases were identified and described, as well as the creation of low and high fidelity prototypes. Finally, in the implementation phase, the presentation layer and server-side programming were developed.

The work culminates with a final reflection on this journey through the development of the “Libruary” application, as well as some suggestions for filling gaps identified in the system.

Keywords: application, android, API, book.

Lista de siglas e abreviaturas

API - Application Programming Interface

IDE - Integrated development environment

JVM - Java virtual machine

UI - User interface

HTTP - Hypertext Transfer Protocol

Índice Geral

1.	Introdução.....	1
1.1	Enquadramento.....	2
2.	Levantamento do Estado de Arte.....	3
2.1	Aplicações relevantes.....	3
3.	Atividades e tarefas realizadas	6
3.1	<i>Milestones e Deliverables</i>	7
3.2	Metodologia Adotada.....	8
3.3	Operacionalização e gestão das atividades.....	9
4.	Prototipagem	10
4.1	Protótipo de baixa fidelidade	10
4.2	Protótipo de alta-fidelidade.....	11
5.	Especificação de requisitos	12
5.1	Requisitos funcionais.....	12
5.2	Restrições e requisitos não funcionais	13
6.	Casos de Utilização	14
6.1	Atores.....	14
6.2	Casos de Uso.....	15
6.2.1	Adicionar Livro	16
6.2.2	Eliminar Livro	16
6.2.3	Definir livro	17
6.2.4	Fazer <i>review</i>	17
6.2.5	Ver perfil	18
6.2.6	Editar <i>review</i>	18
6.2.7	Ver biblioteca pessoal	19
6.2.8	Criar conta	19
6.2.9	Autenticação	20
6.2.10	Receber ISBN do livro.....	20

7.	Análise de tecnologias	21
7.1	Android.....	21
7.2	Base de dados	24
7.3	API	25
7.4	Armazenamento, prototipagem e comunicação.....	28
8.	Desenvolvimento da solução	30
8.1	Diagramas da Base de dados	30
8.1.1	Modelo conceptual.....	30
8.1.2	Modelo relacional.....	31
8.2	Implementação da Base de dados	31
8.3	Implementação da API	31
8.4	Documentação da API.....	32
8.5	Desenvolvimento da aplicação	32
8.5.1	Data flow da navegação da aplicação.....	33
8.5.2	Data flow da API	34
8.5.2	Android Manifest	36
8.5.3	Shared Preferences	38
8.5.5	Interface da aplicação	38
9.	Análise dos resultados	45
9.1	Limitações do projeto	46
9.2	Sugestões para o futuro	47
10.	Conclusão.....	49
	Referências Bibliográficas	50
	Apêndices.....	52
	Apêndice 1. Script da API	53
	Apêndice 2. Script da Base de dados	61
	Apêndice 3. Ficheiro Postman API.....	62

Índice de Figuras

Figura 1 - Página inicial da aplicação Bookmory	4
Figura 2 - Aplicação Goodreads	5
Figura 3 - Protótipo de baixa fidelidade da página home e de registar utilizador.....	10
Figura 4 - Protótipo de baixa fidelidade da página do livro e de inserir livro	11
Figura 5 - Protótipo de alta fidelidade	11
Figura 6 - Diagrama de casos de uso	15
Figura 7 - Android Studio	21
Figura 8 - Exemplo do uso da ferramenta coroutine	22
Figura 9 - Dependências do android studio.....	23
Figura 10 - Aplicação PgAdmin.....	25
Figura 11 - Plataforma Vercel	27
Figura 12 - Página GitHub	28
Figura 13 - Aplicação Figma	29
Figura 14 - Aplicação Discord	29
Figura 15 - Modelo conceptual.....	30
Figura 16 - Modelo relacional.....	31
Figura 17 - Android Manifest.....	36
Figura 18 - Shared Preferences.....	38
Figura 19 - Interface de registo	38
Figura 20 - Interface de login	39
Figura 21 - Interface da activity principal.....	40
Figura 22 - Interface para adicionar o livro.....	40
Figura 23 - interface do processo de scan do ISBN do livro.....	41
Figura 24 - Interface da activity de detalhes do livro	42
Figura 25 - Interface para adicionar review do utilizador	42
Figura 26 - Interface do perfil do utilizador	43
Figura 27 - Interface das reviews feitas pelo utilizador.....	44
Figura 28 – Representação gráfica das percentagens dos requisitos cumpridos	46

Índice de Tabelas

Tabela 1 – Atividades	6
Tabela 2 - <i>Milestones</i>	7
Tabela 3 - <i>Deliverables</i>	7
Tabela 4 - Divisão de tarefas	9
Tabela 5 – Requisitos Funcionais	12
Tabela 6 – Requisitos não funcionais	13
Tabela 7 - Atores da aplicação	14
Tabela 8 - Data flow da API	34
Tabela 9 - Requisitos Cumpridos, não cumpridos e parcialmente cumpridos	45

1. Introdução

Este relatório descreve o desenvolvimento da aplicação móvel Android, realizada no âmbito da unidade curricular de Desenvolvimento de Aplicações para Dispositivos Móveis, lecionada pelo Professor Gonçalo Marques no segundo semestre do segundo ano da Licenciatura em Tecnologias da Informação da Escola Superior de Tecnologias e Gestão de Águeda.

Foi decidido então criar a aplicação "Libruary" que visa proporcionar aos utilizadores uma plataforma intuitiva e eficiente para armazenar e gerir livros de interesse, permitindo também a visualização de *reviews* deixadas por outros utilizadores.

Este relatório detalha as diversas etapas envolvidas no desenvolvimento da aplicação. Inicialmente, aborda-se o planeamento e a execução do trabalho, com uma descrição pormenorizada dos requisitos funcionais e não funcionais da aplicação. Em seguida, apresenta-se o levantamento do estado da arte, o esquema funcional e o modelo de requisitos. O processo de desenvolvimento foi estruturado em várias etapas, começando pela identificação do problema e o planeamento do projeto. Nesta fase inicial, foram delineados os requisitos funcionais e não funcionais da aplicação. Em seguida, o desenho do sistema envolveu a criação de casos de utilização, protótipos de baixa e alta-fidelidade, e o desenvolvimento do modelo de dados persistente.

A análise de resultados é fundamental, para assegurar a funcionalidade e a usabilidade da aplicação, sendo este igualmente abordado no relatório. A reflexão crítica e a conclusão fornecem uma avaliação do percurso de desenvolvimento da aplicação e propõem sugestões para futuras melhorias. Complementam o relatório a bibliografia e os anexos, que fornecem suporte adicional às informações apresentadas nos capítulos anteriores.

A aplicação "Libruary" pretende não apenas facilitar a organização pessoal de leituras, mas também fomentar uma comunidade de leitores que podem compartilhar as suas opiniões e descobertas literárias, contribuindo para uma experiência de leitura mais rica e interativa.

1.1 Enquadramento

No contexto atual, a organização e gestão pessoal de leituras enfrenta desafios significativos, especialmente com a crescente quantidade de livros disponíveis e a necessidade de compartilhar opiniões e descobertas literárias. Existem várias plataformas de renome que oferecem funcionalidades relacionadas, mas muitas vezes carecem de uma integração completa e de uma experiência de utilizador otimizada.

A aplicação móvel "Libruary" foi concebida para preencher esta lacuna, proporcionando aos utilizadores uma ferramenta intuitiva e eficiente para gerir as suas leituras. A nossa proposta é oferecer uma aplicação que não apenas armazene os livros de interesse dos utilizadores, mas também facilite a visualização de avaliações deixadas por outros leitores, criando uma comunidade dinâmica e interativa.

Com esta aplicação, pretendemos melhorar a experiência dos utilizadores ao fornecer uma interface visualmente atraente e bem organizada, que simplifique o processo de armazenamento e consulta de livros. Além disso, a aplicação visa reduzir a ocorrência de problemas comuns, aumentar a eficiência na gestão das leituras e elevar a satisfação geral dos utilizadores.

O desenvolvimento de "Libruary" foi orientado pela necessidade de uma solução prática e acessível que pudesse ser utilizada no dia a dia, respondendo eficazmente às necessidades dos leitores modernos. Assim, acreditamos que a nossa aplicação se pode tornar uma ferramenta indispensável para todos aqueles que desejam organizar e compartilhar as experiências literárias de maneira mais eficiente e satisfatória.

2. Levantamento do Estado de Arte

O levantamento do estado da arte é uma etapa crucial no desenvolvimento da aplicação "Libruary", pois permite uma compreensão aprofundada das tecnologias e soluções atuais no mercado de gestão de livros e plataformas de reviews. Este processo envolve a análise de diversas aplicações similares, identificando as suas principais funcionalidades, pontos fortes e limitações. Com base nessa análise, podemos incorporar as melhores práticas e evitar as falhas comuns, garantindo que "Libruary" ofereça uma experiência de utilizador superior e inovadora. Além disso, ao identificar tendências emergentes e tecnologias avançadas, asseguramos que a nossa aplicação esteja alinhada com os desenvolvimentos mais recentes, proporcionando uma ferramenta diferenciada e competitiva no mercado.

2.1 Aplicações relevantes

No contexto do levantamento do estado da arte, a análise de aplicações relevantes é essencial para entender as soluções existentes no mercado que oferecem funcionalidades semelhantes às da "Libruary". Este subtópico foca-se em examinar detalhadamente algumas das principais aplicações de gestão de livros e plataformas de reviews, destacando as suas características, pontos fortes e áreas de melhoria. Ao estudar estas aplicações, podemos identificar práticas bem-sucedidas e inovações que podem ser incorporadas na "Libruary". Esta análise comparativa vai fornecer-nos uma base sólida para desenvolver uma aplicação que atende às necessidades dos utilizadores.

Bookmory

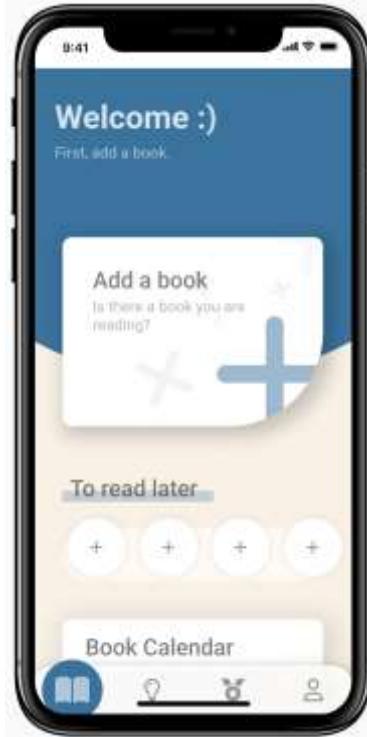


Figura 1 - Página inicial da aplicação Bookmory

A aplicação "Bookmory" destaca-se pelo seu layout simples e intuitivo, que facilita a navegação e o uso, mesmo para os utilizadores menos experientes com tecnologia. A interface é limpa e organizada, permitindo que estes encontrem facilmente as funcionalidades de que precisam sem se sentirem sobrecarregados. Outra funcionalidade notável é a capacidade de ler códigos de barra dos livros, o que simplifica a adição de novos títulos à biblioteca pessoal. Com apenas uma rápida digitalização, os utilizadores podem inserir automaticamente todas as informações relevantes sobre o livro, economizando tempo e garantindo precisão nos registos.

GoodReads



Figura 2 - Aplicação Goodreads

Com a Goodreads é salientado o catálogo de livros o que é uma parte essencial da plataforma, oferecendo aos utilizadores acesso a uma vasta seleção de livros de todos os géneros. Com milhões de livros disponíveis, os usuários podem facilmente encontrar obras que correspondam aos seus interesses específicos. Além disso, a capacidade de fazer reviews permite que os utilizadores partilhem as suas opiniões sobre os livros que leram, ajudando outros utilizadores a tomar decisões sobre o que ler. Essas reviews são visíveis para todos os utilizadores, criando uma comunidade de leitores onde as opiniões e recomendações são valorizadas e compartilhadas.

3. Atividades e tarefas realizadas

Este tópico descreve detalhadamente as atividades e tarefas realizadas ao longo do desenvolvimento da aplicação. É apresentada a metodologia adotada e os desafios enfrentados, bem como as soluções implementadas. Além disso, foi elaborada uma lista de *milestones* e *deliverables*, que serviu como guia para acompanhar o progresso do projeto e garantir que todos os objetivos fossem alcançados dentro dos prazos estabelecidos. Esta análise visa proporcionar uma visão clara e estruturada do percurso de desenvolvimento, destacando as principais conquistas e lições aprendidas ao longo do projeto.

Tabela 1 – Atividades

Id	Designação da atividade	Descrição genérica da atividade
A1	Criação da base de dados	Desenvolvimento do <i>script</i> da base de dados
A1.1	Criação do modelo conceptual e relacional	Criação dos diagramas do modelo conceptual e relacional
A2	Criação da API	Desenvolvimento do script da API
A2.1	Criação dos métodos da API	Realização dos métodos na API
A2.2	Integração da API com a base de dados	Estabelecimento da conexão da API com a base de dados
A3	Criação da aplicação android	Desenvolvimento da aplicação no sistema android
A3.1	Criação do protótipo de baixa fidelidade e alta-fidelidade	Elaboração do esboço de baixa fidelidade e produção de um modelo de alta-fidelidade
A3.2	Criação das atividades no android studio	Construção das atividades que iam ser utilizadas no android studio
A3.3	Implementação dos métodos das Atividades	Criação dos métodos de cada atividade.
A3.4	Integração das atividades com a API	Integração dos métodos de cada atividade com a API

3.1 Milestones e Deliverables

Os *milestones* e *deliverables* são fundamentais para a gestão do projeto. Este subtópico apresenta as etapas chave e as entregas principais do projeto. Os *milestones* marcam os pontos de verificação importantes, enquanto os *deliverables* representam os produtos tangíveis de cada fase, assegurando o progresso e a conclusão das metas dentro dos prazos estabelecidos.

Tabela 2 - *Milestones*

Id	Nome do milestone	Data esperada
B1	Criação da base de dados	16/05/2024
B2	Criação da API	20/05/2024
B3	API funcional	23/05/2024
B4	Criação das atividades no android studio	27/05/2024
B5	Integração com a API	30/05/2024
B6	Autenticação	30/05/2024
B7	Adicionar livro e remover livro	01/06/2024
B8	Fazer e editar review	04/06/2024
B9	Interface aperfeiçoada	06/06/2024
B10	Conclusão do relatório	09/06/2024

Tabela 3 - *Deliverables*

Id	Nome do deliverable	Data esperada	Tipo
C1	Relatório	09/06/2024	Documento detalhado com informações, análises e recomendações sobre o projeto
C2	Código da aplicação Android	09/06/2024	Código da aplicação desenvolvida para dispositivos móveis Android.
C3	Ficheiro Postman	09/06/2024	Ficheiro usado na plataforma Postman para testar APIs, contendo <i>requests</i> HTTP, testes e documentação.
C4	Código da API em Python	09/06/2024	Código em Python para implementar uma API.
C5	Script da base de dados	09/06/2024	Conjunto das operações da base de dados para a sua criação

3.2 Metodologia Adotada

O *Feature-driven development* (FDD), ou Desenvolvimento Orientado a Recursos em português, foi a metodologia de desenvolvimento de software escolhida para este projeto.

Ao optar pelo FDD, procurámos garantir um processo de desenvolvimento transparente, eficiente e focado nas nossas necessidades. Através da divisão do projeto em funcionalidades menores e administráveis, foi possível trabalhar de forma iterativa, ao testar e validar cada funcionalidade ativamente por nós. Essa abordagem colaborativa permitiu um ajuste do produto final, garantindo que ele atendesse às nossas expectativas de forma eficaz.

Além disso, o FDD promoveu um ambiente de trabalho dinâmico para a equipa de desenvolvimento. A comunicação constante entre os membros da equipe foi fundamental para o sucesso do projeto, permitindo a identificação e resolução rápida de problemas, além da otimização do tempo e dos recursos disponíveis.

3.3 Operacionalização e gestão das atividades

Desde que o enunciado foi publicado (a 6 de maio de 2024) foram dedicadas, em média, 21 horas por semana (3 horas por dia) a atividades extra-aula para o desenvolvimento do projeto, totalizando cerca de 105 horas. Além disso, os membros do grupo participaram nas 11 aulas semanais de 4 horas (44 horas no total), o que somado às 105 horas extra-aula totaliza 149 horas dedicadas ao projeto.

Abaixo encontra-se a tabela com a divisão de tarefas executada, a duração, bem como a responsabilidade de cada membro do grupo, ao longo do projeto.

Tabela 4 - Divisão de tarefas

#	Tarefa	Duração (dias)	Responsabilidade (%)
1	Levantamento do estado da arte	2	50 50
2	Identificação das funcionalidades do sistema	2	50 50
3	Identificação dos utilizadores do sistema	1	50 50
4	Identificação dos casos de utilização	1	50 50
5	Identificação dos requisitos do sistema	2	50 50
6	Criação do diagrama de casos de utilização	1	30 70
7	Descrição dos casos de utilização	2	30 70
8	Identificação de todas as <i>activities</i>	1	70 30
9	Produção do modelo de dados persistente	3	50 50
10	Produção do modelo estrutural	3	50 50
11	Protótipo de baixa fidelidade	1	40 60
12	Protótipo de alta fidelidade	3	40 60
13	Implementação da base de dados	2	60 40
14	Elaboração da API	7	60 40
15	Desenvolvimento da aplicação	15	50 50
16	Testes e correção de eventuais falhas	3	50 50
17	Análise de resultados e documentação	2	50 50
18	Conclusão da elaboração do relatório final	6	50 50

Daniel Silva

Gabriel Cravo

4. Prototipagem

A prototipagem permite a visualização e validação das funcionalidades e da interface de utilizador antes da implementação final. Este tópico apresenta os protótipos de baixa e alta-fidelidade criados durante o processo de design. Os protótipos de baixa fidelidade oferecem uma representação inicial e simplificada da interface, focando-se na estrutura e no fluxo de navegação. Os protótipos de alta-fidelidade apresentam um design mais detalhado e realista, aproximando-se da aparência e comportamento final da aplicação. Através destes protótipos, é possível identificar e corrigir problemas de usabilidade, garantindo que a aplicação final seja intuitiva e eficaz para os utilizadores.

4.1 Protótipo de baixa fidelidade

Este subtópico apresenta o protótipo desenhado a lápis, destacando os principais componentes e layout da interface de utilizador. As próximas imagens ilustram este protótipo, permitindo uma análise antecipada que facilita ajustes antes da criação de versões mais detalhadas.

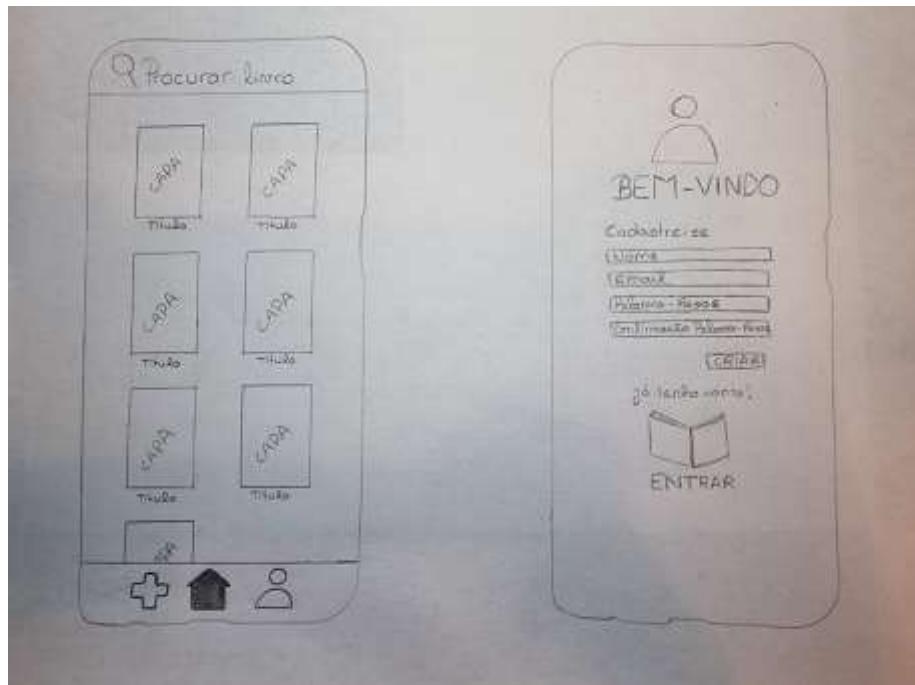


Figura 3 - Protótipo de baixa fidelidade da página home e de registar utilizador

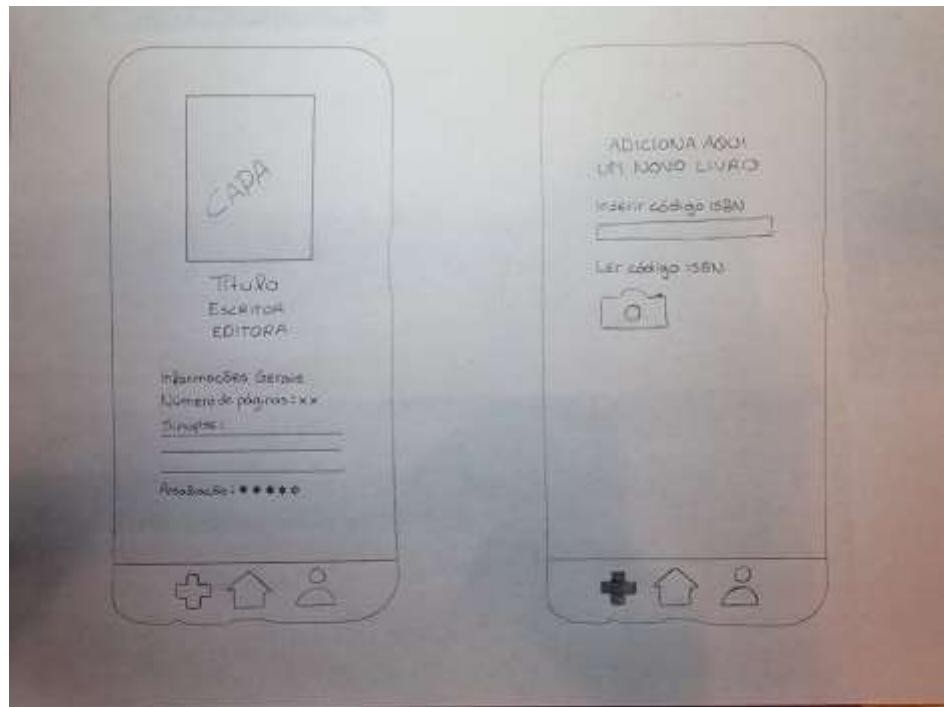


Figura 4 - Protótipo de baixa fidelidade da página do livro e de inserir livro

4.2 Protótipo de alta-fidelidade

Este subtópico apresenta o protótipo desenvolvido com a aplicação figma, destacando os elementos visuais e interativos que compõem a experiência do utilizador. A próxima imagem ilustra este protótipo, permitindo uma compreensão clara e precisa do design e da funcionalidade esperados na versão final da aplicação.

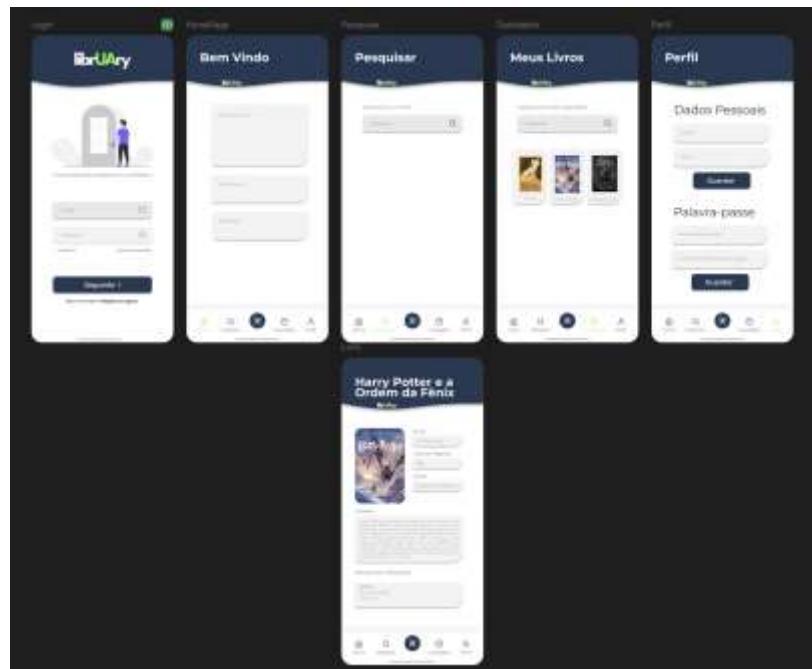


Figura 5 - Protótipo de alta fidelidade

5. Especificação de requisitos

A especificação de requisitos para a aplicação envolveu um processo de recolha de dados utilizando prototipagem, observação e análise de sistemas similares. A prototipagem permitiu criar representações visuais iniciais da aplicação, facilitando a validação dos requisitos pelos utilizadores. A observação forneceu insights sobre as interações dos utilizadores com aplicações de gestão de livros, identificando necessidades e comportamentos reais. A análise de sistemas similares ajudou a identificar funcionalidades essenciais e áreas de melhoria, garantindo que a plataforma android ofereça uma experiência eficiente e *user-friendly*.

5.1 Requisitos funcionais

Os requisitos funcionais definem as funcionalidades específicas que a aplicação deve possuir para atender às necessidades dos utilizadores. Estes requisitos detalham as operações e comportamentos que a aplicação deve executar. A definição clara dos requisitos funcionais é essencial para garantir que todas as funcionalidades críticas sejam implementadas, proporcionando uma experiência de utilizador eficiente e satisfatória.

Tabela 5 – Requisitos Funcionais

Ref ^a	Requisito funcional	Prioridade
RF. 1	Autenticação	Alta
RF. 2	Adicionar livro à biblioteca pessoal	Alta
RF. 3	Remover livro da biblioteca pessoal	Alta
RF. 4	Adicionar livro por código QR	Moderada
RF. 5	Adicionar <i>review</i> a um livro	Alta
RF. 6	Ver reviews de outros utilizadores	Alta
RF. 7	Criar conta	Alta
RF. 8	Alterar a <i>review</i> do livro	Moderada
RF. 9	Ver os livros da biblioteca pessoal	Alta
RF. 10	Gerir perfil	Moderada
RF. 11	Marcar conclusão de leitura do livro	Moderada
RF. 12	Estatísticas de livros lidos	Moderada
RF. 13	Alertas para ler	Baixa

5.2 Restrições e requisitos não funcionais

As restrições e os requisitos não funcionais são elementos essenciais que definem as condições e características que a aplicação deve cumprir além das suas funcionalidades básicas. Estes requisitos incluem aspectos como desempenho, segurança, usabilidade, compatibilidade e manutenção. As restrições, por sua vez, podem incluir limitações tecnológicas, orçamentais ou temporais que impactam o desenvolvimento e a implementação da aplicação.

Tabela 6 – Requisitos não funcionais

Ref ^a	Requisito não funcional	Tipo	Prioridade
RNF.1	Encriptação da password	Segurança	Alta
RNF.2	Utilização da API	Integração/Segurança	Alta
RNF.3	Aplicação rápida e intuitiva	Usabilidade	Alta
RNF.4	Backup da base de dados	Manutenção	Moderada
RNF.5	Utilização de tokens	Segurança	Alta

6. Casos de Utilização

Os casos de utilização são uma ferramenta essencial para descrever as interações entre os utilizadores e a aplicação "Librury". Este tópico apresenta os diferentes cenários em que os utilizadores interagem com a aplicação, detalhando as funcionalidades principais e os fluxos de trabalho esperados. Através dos casos de utilização, é possível compreender melhor as necessidades e expectativas dos utilizadores, identificando os requisitos específicos para cada funcionalidade. Esta análise detalhada permite assegurar que a aplicação atende a todos os requisitos dos utilizadores, ao proporcionar uma experiência de uso eficaz e intuitiva. Além disso, os casos de utilização servem como base para o desenvolvimento de testes, garantindo que cada funcionalidade seja implementada corretamente e funcione como previsto em situações reais.

6.1 Atores

No contexto dos casos de utilização, os atores representam os diferentes tipos de utilizadores e sistemas externos que interagem com a aplicação "Librury". Este subtópico identifica e descreve os principais atores envolvidos, detalhando as suas responsabilidades e expectativas (Tabela 6). A compreensão clara dos atores é fundamental para definir corretamente os requisitos e garantir que a aplicação atenda às necessidades de todos os envolvidos, proporcionando uma experiência de uso eficaz e satisfatória.

Tabela 7 - Atores da aplicação

Actor	Descrição
Utilizador	Pessoa capaz de adicionar livros e fazer <i>reviews</i> de livros.
GoogleBooksAPI	API que fornece dados dos livros que o utilizador adiciona.

6.2 Casos de Uso

Este subtópico apresenta o diagrama de casos de utilização da aplicação, ilustrando as interações entre os atores e as funcionalidades principais. Além do diagrama, serão fornecidas tabelas com a descrição detalhada de cada caso de uso, oferecendo uma visão clara dos principais fluxos de trabalho e requisitos funcionais para garantir que todas as necessidades dos utilizadores sejam atendidas.

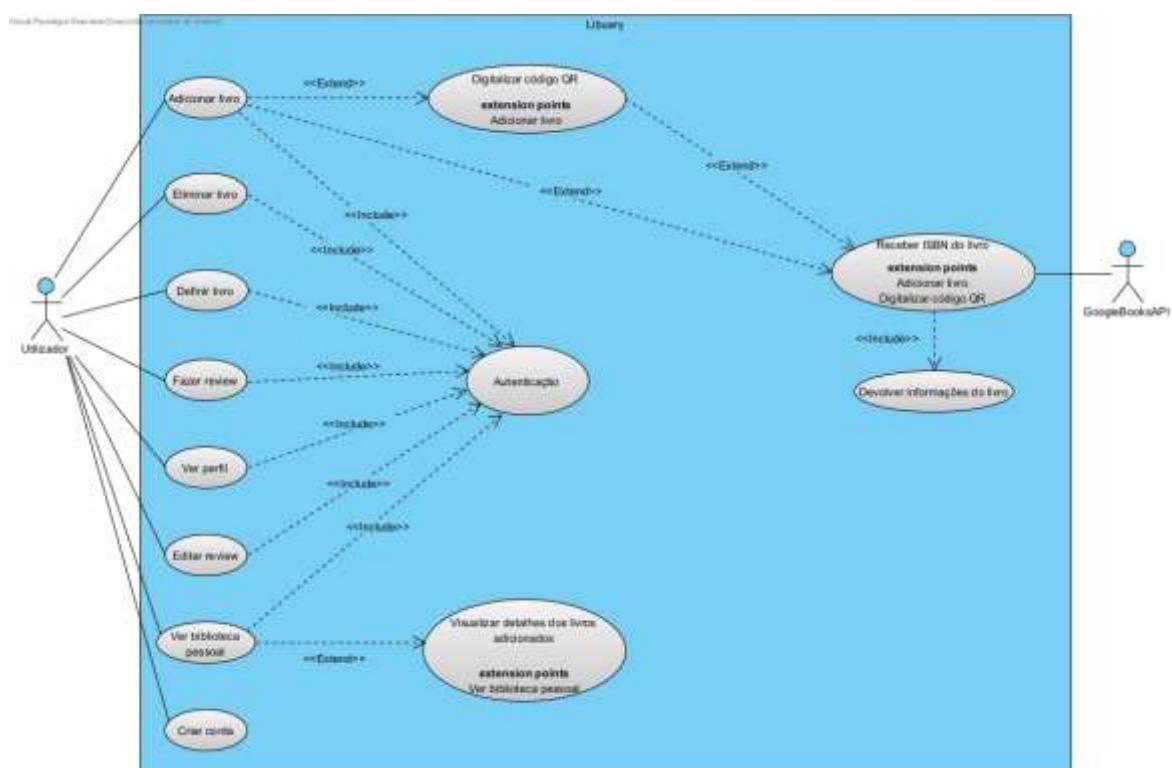


Figura 6 - Diagrama de casos de uso

6.2.1 Adicionar Livro

Nome:	Adicionar Livro
Atores:	Utilizador
Prioridade (1/3):	3
Finalidade:	Adicionar livro à biblioteca pessoal
Requisitos funcionais:	RF. 2 – Adicionar livro à biblioteca pessoal
Pré-condições:	Aplicação necessita de conta autenticada
Sumário:	O utilizador adiciona um livro manualmente inserindo o ISBN do livro ou a digitalizar o livro à biblioteca pessoal

Sequência típica dos eventos

Ações dos atores	Respostas do sistema
O ator utilizador introduz o ISBN do livro.	Sistema guarda o livro na biblioteca.

6.2.2 Eliminar Livro

Nome:	Eliminar Livro
Atores:	Utilizador
Prioridade (1/3):	3
Finalidade:	Eliminar o livro da biblioteca pessoal
Requisitos funcionais:	RF. 3 - Remover livro da biblioteca pessoal
Pré-condições:	Aplicação necessita de conta autenticada e de pelo menos um livro na biblioteca pessoal.
Sumário:	O utilizador elimina o livro da biblioteca pessoal selecionado.

Sequência típica dos eventos

Ações dos atores	Respostas do sistema
O ator utilizador remove o livro da sua biblioteca	Sistema elimina o livro da biblioteca pessoal do utilizador

6.2.3 Definir livro

Nome:	Definir livro
Atores:	Utilizador
Prioridade (1/3):	2
Finalidade:	Definir o livro como lido ou por ler
Requisitos funcionais:	RF.11 – Marcar conclusão de leitura do livro
Pré-condições:	Aplicação necessita de conta autenticada e de livros na biblioteca pessoal do utilizador.
Sumário:	O utilizador pode marcar como lido o livro que tem na biblioteca pessoal ou desmarcar.

Sequência típica dos eventos

Ações dos atores	Respostas do sistema
O ator utilizador marca o livro como lido	Sistema guarda o livro como lido.

6.2.4 Fazer review

Nome:	Fazer review
Atores:	Utilizador
Prioridade (1/3):	3
Finalidade:	Adicionar uma <i>review</i> de um livro da biblioteca pessoal
Requisitos funcionais:	RF. 5 – Adicionar <i>review</i> a um livro
Pré-condições:	Aplicação necessita de conta autenticada
Sumário:	O utilizador adiciona uma <i>review</i> a um livro da sua biblioteca pessoal tornando possível que aos outros utilizadores ver a sua <i>review</i> .

Sequência típica dos eventos

Ações dos atores	Respostas do sistema
O ator utilizador faz uma <i>review</i> do livro da sua coleção pessoal de livros.	O sistema guarda a <i>review</i> e mostra a <i>review</i> no livro caso o outro utilizador tenha o mesmo livro na biblioteca pessoal.

6.2.5 Ver perfil

Nome:	Ver perfil
Atores:	Utilizador
Prioridade (1/3):	3
Finalidade:	Ver as informações pessoais da conta
Requisitos funcionais:	RF. 10 – Gerir perfil
Pré-condições:	Aplicação necessita de conta autenticada
Sumário:	Utilizador vai à parte da parte do perfil na aplicação onde pode ver as suas informações pessoais

Sequência típica dos eventos

Ações dos atores	Respostas do sistema
O ator utilizador clica no botão de perfil.	O sistema redireciona para secção de perfil.

6.2.6 Editar review

Nome:	Editar review
Atores:	Utilizador
Prioridade (1/3):	3
Finalidade:	Editar a review que fez anteriormente num livro
Requisitos funcionais:	RF. 8 – Alterar review do livro
Pré-condições:	Aplicação necessita de conta autenticada e de uma review já feita de um livro
Sumário:	O utilizador edita uma review já feita de um livro da sua biblioteca

Sequência típica dos eventos

Ações dos atores	Respostas do sistema
O ator utilizador edita uma review já feita anteriormente	O sistema atualiza a review do livro

6.2.7 Ver biblioteca pessoal

Nome:	Ver biblioteca pessoal
Atores:	Utilizador
Prioridade (1/3):	3
Finalidade:	Ver os livros adicionados á biblioteca pessoal
Requisitos funcionais:	RF. 9 – Ver os livros da biblioteca pessoal
Pré-condições:	Aplicação necessita de conta autenticada e ter livros adicionados.
Sumário:	O utilizador consegue ver os livros que adicionou à sua biblioteca pessoal
Sequência típica dos eventos	
Ações dos atores	
Respostas do sistema	
O ator utilizador depois de adicionar um livro,	O sistema mostra o livro adicionado na sua biblioteca pessoal

6.2.8 Criar conta

Nome:	Criar conta
Atores:	Utilizador
Prioridade (1/3):	3
Finalidade:	Criação da conta para que o utilizador possa aceder às funcionalidades da aplicação
Requisitos funcionais:	RF. 7 – Criar conta
Pré-condições:	Ter a aplicação instalada
Sumário:	O utilizador cria uma conta para conseguir autenticar na aplicação
Sequência típica dos eventos	
Ações dos atores	
Respostas do sistema	
O ator utilizador cria uma conta	O sistema guarda as informações do utilizador de maneira que estejam seguras e que seja capaz depois de autenticar-se

6.2.9 Autenticação

Nome:	Autenticação
Atores:	Utilizador
Prioridade (1/3):	3
Finalidade:	Fazer <i>login</i> na aplicação
Requisitos funcionais:	RF. 1 – Autenticação
Pré-condições:	Ter conta na aplicação
Sumário:	O utilizador introduz as suas informações pessoais para conseguir fazer <i>login</i> na aplicação para conseguir utilizar as funcionalidades dela
Sequência típica dos eventos	
Ações dos atores	
Respostas do sistema	
O ator utilizador faz login na aplicação com as suas informações pessoais	O sistema redireciona o utilizador para a página inicial da aplicação

6.2.10 Receber ISBN do livro

Nome:	Receber ISBN do livro
Atores:	GoogleBooksAPI
Prioridade (1/3):	3
Finalidade:	Procurar informações do livro
Requisitos funcionais:	RF. 2 – Adicionar livro à biblioteca pessoal
Pré-condições:	Adicionar um livro á biblioteca pessoal
Sumário:	O utilizador ao adicionar o livro a aplicação utiliza a GoogleBooksAPI para ir procurar informações sobre o livro
Sequência típica dos eventos	
Ações dos atores	
Respostas do sistema	
O ator utilizador adiciona um livro	O sistema procura na API as informações para as guardar

7. Análise de tecnologias

O objetivo principal deste tópico é identificar as tecnologias que foram utilizadas no desenvolvimento de uma aplicação Android. Ao explorar em detalhe cada tecnologia utilizada, incluindo as específicas para o desenvolvimento Android. Esta análise detalhada visa fornecer uma visão clara do que foi utilizado no desenvolvimento da aplicação Android.

7.1 Android

Neste subtópico, detalhamos as tecnologias e ferramentas utilizadas para desenvolver a aplicação Android.

- **Android Studio:** O Android Studio é o ambiente de desenvolvimento integrado (IDE) oficial para o desenvolvimento de aplicativos Android. Ele é baseado no software JetBrains IntelliJ IDEA e foi projetado especificamente para o desenvolvimento Android, utilizado nas aulas de desenvolvimento de aplicações para dispositivos móveis, lecionada pelo professor Gonçalo Marques.

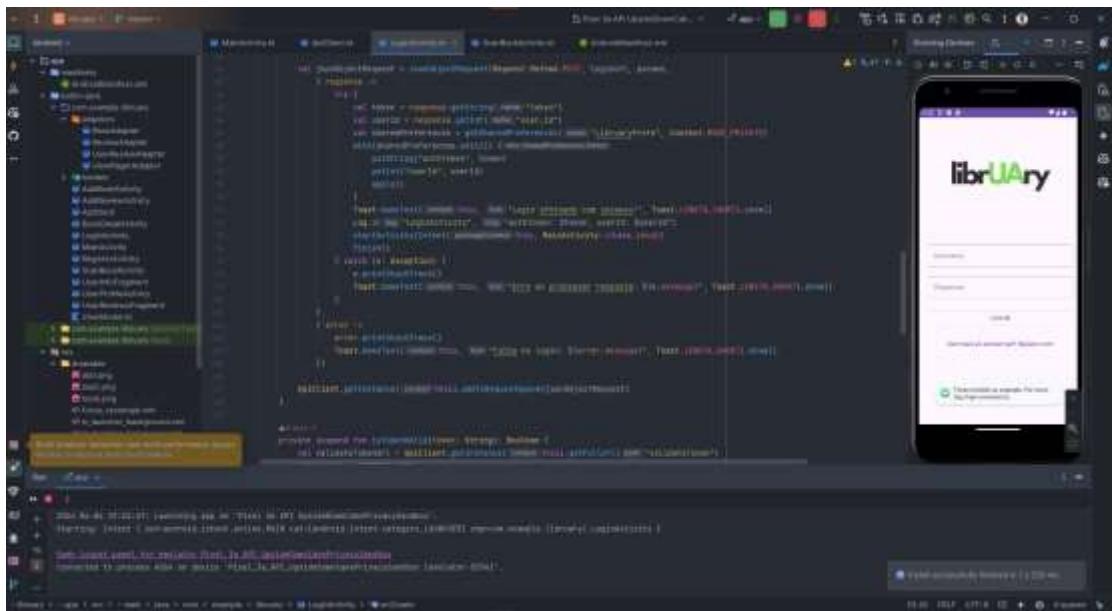
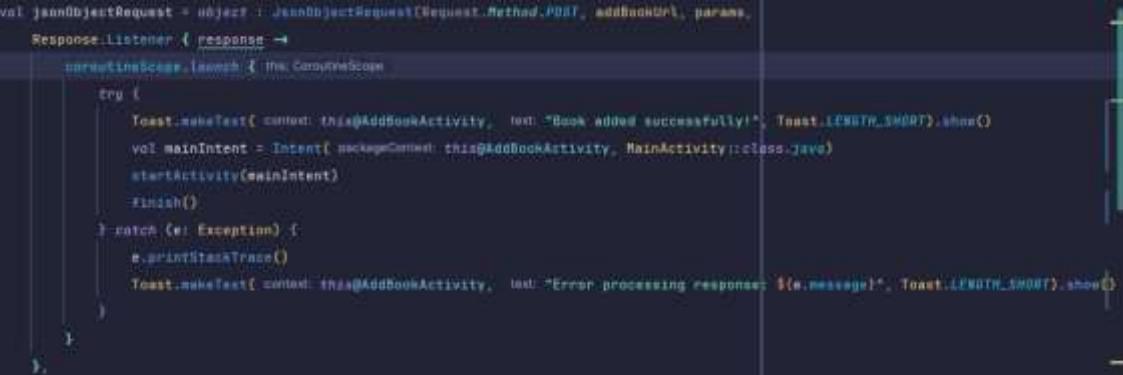


Figura 7 - Android Studio

- **Kotlin:** Kotlin é uma linguagem de programação, desenvolvida pela JetBrains. Projetada para ser utilizada no desenvolvimento de aplicativos Android, além de suportar outros ambientes como web e servidores. Esta linguagem facilita a escrita de código mais claro e menos propenso a erros. Utilizada nas aulas da unidade curricular.

Vale a pena ainda referir que com o requisito de utilizar a API para a transmissão de dados, o grupo optou por utilizar coroutines, que é uma ferramenta eficiente e simplificada para lidar com programação assíncrona e concorrente. Como uma funcionalidade default do Kotlin, a coroutines permitem a execução de *requests* HTTP sem bloquear a execução do restante do código, assegurando assim a responsividade da aplicação e melhorando também a experiência do utilizador.



```

val jsonObjectRequest = object : JsonObjectRequest(Request.Method.POST, addBookUrl, params,
    Response.Listener { response ->
        coroutineScope.launch { this.coroutineScope
            try {
                Toast.makeText(context, this@AddBookActivity, text: "Book added successfully!", Toast.LENGTH_SHORT).show()
                val mainIntent = Intent(packageName: this@AddBookActivity, MainActivity::class.java)
                startActivity(mainIntent)
                finish()
            } catch (e: Exception) {
                e.printStackTrace()
                Toast.makeText(context: this@AddBookActivity, text: "Error processing response: ${e.message}", Toast.LENGTH_SHORT).show()
            }
        }
    }
)

```

Figura 8 - Exemplo do uso da ferramenta coroutine

- **Gradle:** Gradle é uma ferramenta de automatização de *builds* utilizada em projetos de software na JVM, como Java e Kotlin. Ela torna automático a compilação, os testes e implementação, gera as dependências e usa uma linguagem de script flexível em Groovy ou Kotlin. O Gradle é amplamente utilizado pelo Android Studio, para gerir *builds* e as dependências de projetos Android. No IDE existe um ficheiro de configuração onde são especificadas as dependências do projeto, normalmente chamado build.gradle ou build.gradle.kts. As dependências são bibliotecas ou componentes necessários para o funcionamento do projeto, na seguinte imagem demonstra todas as dependências de nosso projeto.

```

1 dependencies { this: DependencyHandlerScope
2
3     implementation(libs.volley)
4     implementation(libs.androidx.annotation)
5     implementation(libs.androidx.lifecycle.livedata.ktx)
6     // CameraX core library using the latest version
7     val camerax_version = "1.4.0-beta01"
8     implementation("com.google.android.material:material:1.4.0")
9     implementation("androidx.camera:camera-core:${camerax_version}")
10    implementation("androidx.camera:camera-camera2:${camerax_version}")
11    implementation("androidx.camera:camera-lifecycle:${camerax_version}")
12    implementation("com.google.mlkit:barcode-scanning:0.2.0")
13    implementation("androidx.lifecycle:lifecycle-viewmodel-ktx:2.3.1")
14    implementation("androidx.lifecycle:lifecycle-runtime-ktx:2.3.1")
15    implementation("androidx.camera:camera-view:1.4.0-beta01")
16    implementation("androidx.camera:camera-extensions:1.4.0-beta01")
17    implementation ("com.android.volley:volley:1.2.1")
18
19    implementation("com.squareup.okhttp3:logging-interceptor:4.9.1")
20
21    implementation("com.github.bumptech.glide:glide:4.11.0")
22    annotationProcessor("com.github.bumptech.glide:compiler:4.11.0")
23
24    implementation(libs.androidx.core.ktx)
25    implementation(libs.androidx.appcompat)
26    implementation(libs.material)
27    implementation(libs.androidx.activity)
28    implementation(libs.androidx.constraintlayout)
29    implementation("androidx.recyclerview:recyclerview:1.1.0")
30
31    testImplementation(libs.junit)
32    androidTestImplementation(libs.androidx.junit)
33    androidTestImplementation(libs.androidx.espresso.core)
34
35    implementation ("androidx.lifecycle:lifecycle-viewmodel-ktx:2.3.1")
36    implementation ("androidx.paging:paging-runtime:3.0.0")
37

```

Figura 9 - Dependências do android studio

O grupo decidiu utilizar várias dependências, mas na questão de requisitos a API o grupo utilizou o volley, pois foi a biblioteca utilizada nas aulas e a que o grupo se sentiu mais à vontade para utilizar.

Volley: A biblioteca HTTP Volley é uma ferramenta desenvolvida pela Google para facilitar a comunicação de rede em aplicações Android. Ela simplifica a execução de pedidos HTTP entre outras funcionalidades.

Como o grupo foi ambicioso e quis a utilização da câmara para ler o código de barras dos livros, tivemos de utilizar duas dependências, CameraX e ML Kit.

Camex: CameraX é uma biblioteca da Jetpack desenvolvida pela google que facilita o desenvolvimento de aplicações que utilizam a câmara.

ML Kit: ML Kit é um conjunto de APIs do Firebase da google que traz modelos de machine learning para dentro das aplicações Android. A dependência barcode-scanning permite a leitura de códigos de barras e QR codes usando a câmera do dispositivo.

Para a UI o grupo utilizou a várias dependências para melhorar o visual da nossa aplicação, estas foram: O Material, a glide para a formatação das imagens dos livros recebidos da google e o paging para fazer a paginação das *reviews* dos utilizadores.

Material: Fornece componentes de UI e temas que seguem as diretrizes do Material, assim facilita a criação de interfaces modernas com a aparência visual recomendada pela Google.

Glide: Glide é uma biblioteca para carregamento e *caching* de imagens que simplifica o trabalho com imagens em aplicações Android.

Paging: A Biblioteca *Paging* facilita o carregamento de grandes volumes de dados paginados de forma eficiente.

7.2 Base de dados

- **PgAdmin:** Ele é projetado para atender às necessidades tanto de utilizadores iniciantes quanto experientes do Postgres, ao fornecer uma interface gráfica que simplifica a criação, manutenção e uso de dados na base de dados. O grupo optou por utilizar devido à experiência que obtivemos em projetos anteriores.

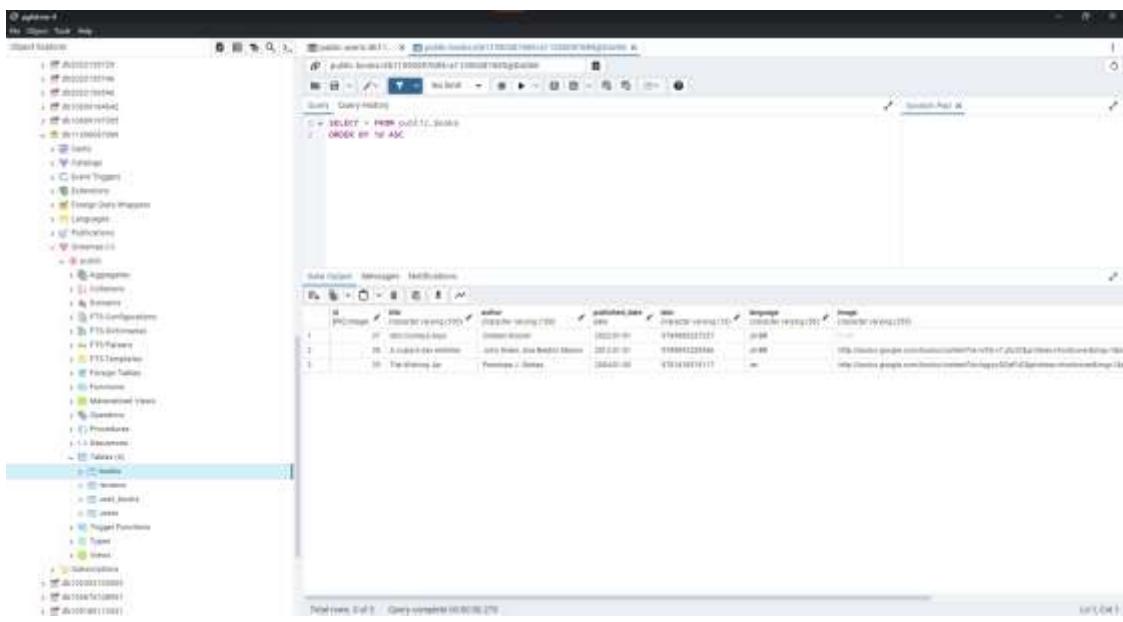


Figura 10 - Aplicação PgAdmin

- **PostgresSQL:** É um sistema de gestão de base de dados relacional de código aberto. Ele armazena dados em tabelas compostas por linhas e colunas, e os utilizadores podem gerir e consultar esses dados utilizando a Linguagem de Consulta Estruturada (SQL).

Foi um requisito no projeto fazer a encriptação da *password* do utilizador, então tal como lecionado na unidade curricular o grupo a utilizou a extenção do PostgreSQL, pgcrypto.

Pgcrypto: É uma extensão do PostgreSQL que fornece funções de criptografia e de hashing, permitindo a manipulação de dados sensíveis de forma segura.

7.3 API

Neste subtópico, apresentamos as tecnologias e ferramentas utilizadas para construir a API que suporta a aplicação.

O grupo construiu a API utilizando o vercel que é uma plataforma de hospedagem na *cloud* que simplifica o processo de colocar um site ou aplicativo web online. Para o grupo construir a API utilizou os seguintes passos:

Adicionar um Novo Projeto:

Fizemos login na conta do Vercel e fizemos um novo projeto.

Procurar pela Template Flask:

No momento de criar um projeto, fomos apresentados com várias opções de templates. Mas como foi lecionado na unidade curricular procuramos por "Flask" e selecionamos a template Flask.

O Flask permite criar APIs de forma mais fácil. Com ele é possível criar endpoints para receber e enviar dados entre diferentes partes da aplicação ou entre aplicações diferentes. É como um canal de comunicação para que as partes da plataforma possam trocar informações.

Criação de um Repositório no GitHub:

O Vercel integrasse bem com o GitHub para facilitar o processo de implementação, por isso é possível criar um repositório privado na plataforma web, o que foi o nosso caso.

Criar os Arquivos no Repositório:

No repositório do GitHub, foram criados os ficheiros necessários para o nosso projeto Flask. Estes são:

app.py: Este é o ficheiro principal da aplicação Flask, onde vai ser escrito o código para os *endpoints* e lógica destes.

requirements.txt: Aqui serve para listar todas as dependências do projeto, como o Flask e as outras bibliotecas que foram utilizadas na API, incluindo as suas versões.

vercel.json: Este ficheiro é utilizado pelo Vercel para perceber como construir e servir a aplicação Flask. Ele contém configurações importantes, como a versão do Vercel API que está utilizada e as rotas da aplicação.

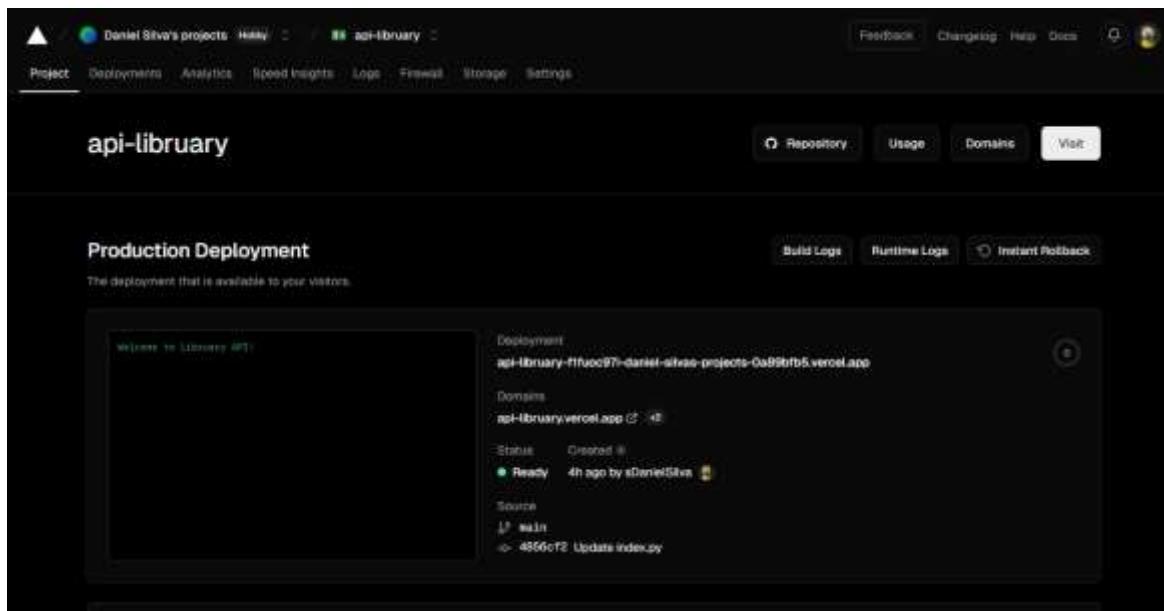


Figura 11 - Plataforma Vercel

Como referido anteriormente é necessário referir as dependências que utilizamos no ficheiro requirements.txt incluindo o flask para ser utilizado, neste caso utilizámos.

Flask>=2.0.0,<3.0.0

Para *requests* a google books API foi utilizado a dependência *requests*.

requests: *Requests* é uma biblioteca HTTP para Python, que facilita o envio de *requests* HTTP.

Para cumprir o requisito de segurança “utilização de *tokens*” para efetuar requests à API da aplicação utilizamos a dependência JWT.

PyJWT==2.3.0: PyJWT é uma biblioteca Python para trabalhar com tokens JWT.

Tal como foi lecionado na UC, a fim de aceder à informação que existe na base de dados na API, utilizamos o psycopg2:

psycopg2-binary==2.9.1: Psycopg2 é um adaptador de base de dados PostgreSQL para Python

7.4 Armazenamento, prototipagem e comunicação

- **GitHub:** O GitHub é uma plataforma de armazenamento de código-fonte que permite aos utilizadores armazenar e compartilhar os seus repositórios Git *online*. Além de fornecer uma interface gráfica do utilizador para o Git, o GitHub oferece recursos de colaboração, como rastreamento de problemas, *pull requests* e administração de projetos. Optámos por usar o GitHub no nosso projeto, pois o grupo já está familiarizado com essa ferramenta devido a projetos anteriores.

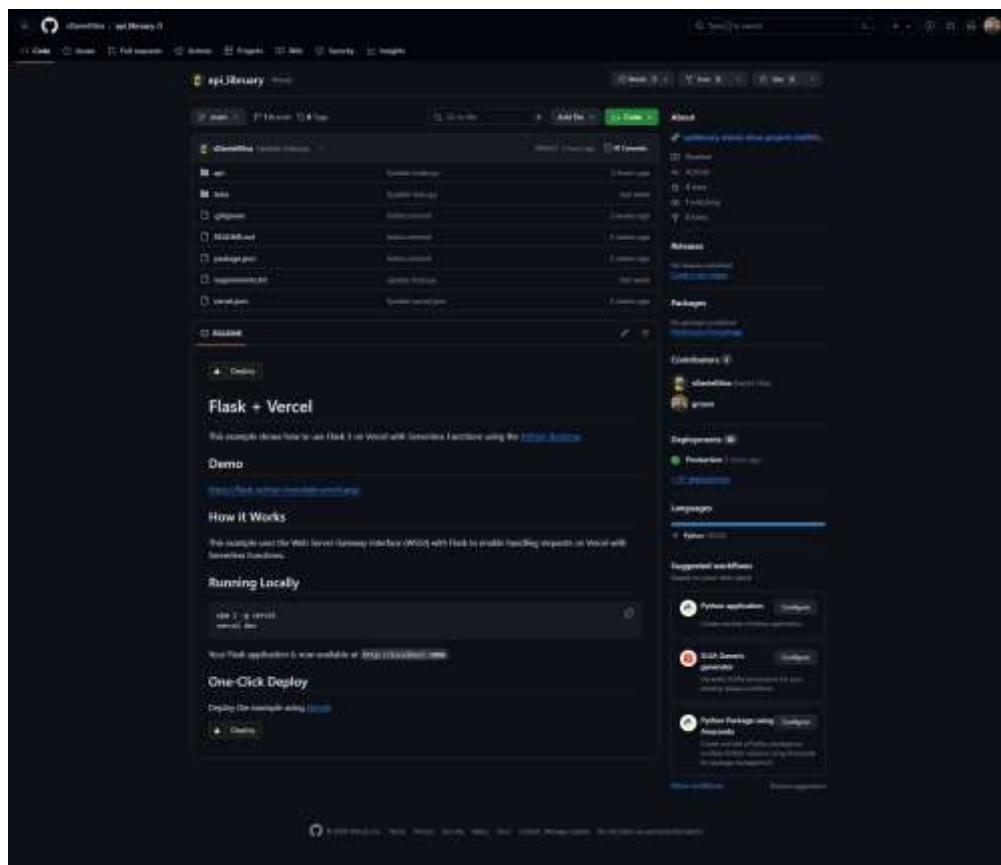


Figura 12 - Página GitHub

- **Git:** O Git é um sistema de controlo de versão distribuído. Ele permite que os desenvolvedores consigam fazer a gestão das alterações do código-fonte e projetos, mantendo um histórico completo das versões.

- **Figma:** O Figma é uma plataforma para construção de design de interfaces e protótipos. Ela permite criar produtos digitais, como sites e aplicações. Utilizada em unidades curriculares anteriores.

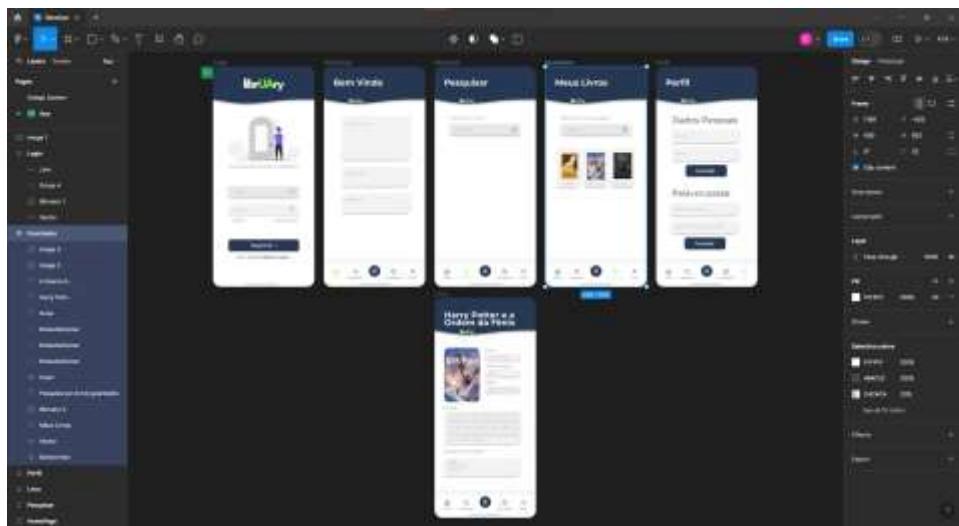


Figura 13 - Aplicação Figma

- **Discord:** O Discord é uma plataforma de comunicação que permite trocar mensagens de texto, fazer chamadas de áudio e vídeo. Esta aplicação oferece também a funcionalidade de partilha de ficheiros e a criação de servidores personalizados. Devido à sua acessibilidade e uso diário pelo grupo, foi selecionada para organizar e resolver questões relacionadas ao projeto.

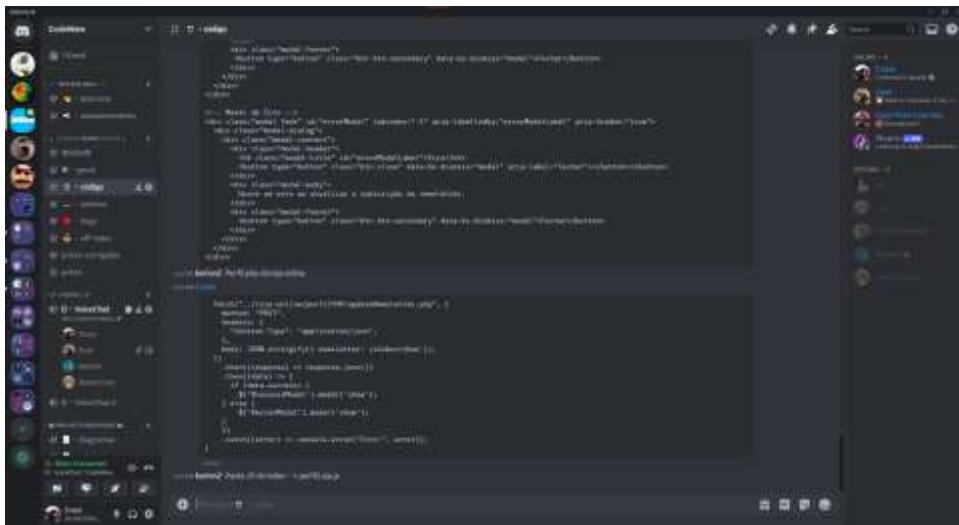


Figura 14 - Aplicação Discord

8. Desenvolvimento da solução

O tópico de Desenvolvimento da Solução descreve detalhadamente o processo de criação e implementação da aplicação. Este tópico está dividido em vários subtópicos que abordam os principais componentes técnicos do projeto.

8.1 Diagramas da Base de dados

Este subtópico apresenta os diagramas da base de dados, o diagrama conceptual e o relacional, ilustrando a estrutura que compõem o sistema. É importante referir que os esquemas estão funcionais, contudo o grupo percebeu muito perto da data-limite de entrega que não estão contruídos da melhor forma e estão de certeza sujeitos a alterações de desempenho, como utilizar uma tabela para juntar as duas tabelas *reviews* e *user_books*

8.1.1 Modelo conceptual

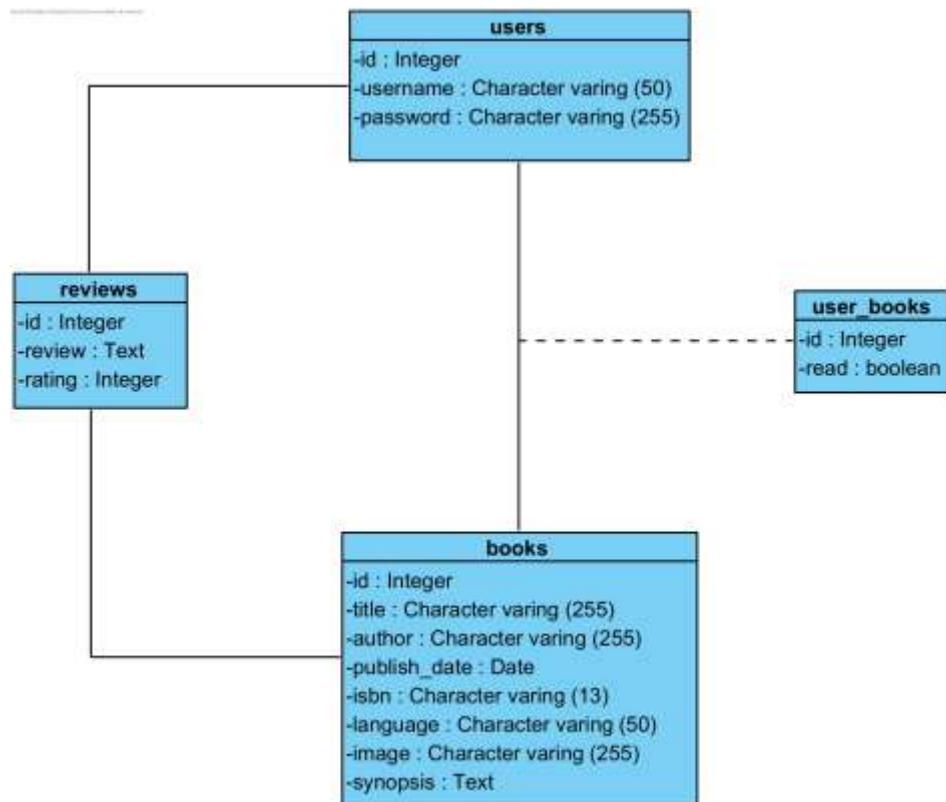


Figura 15 - Modelo conceptual

8.1.2 Modelo relacional

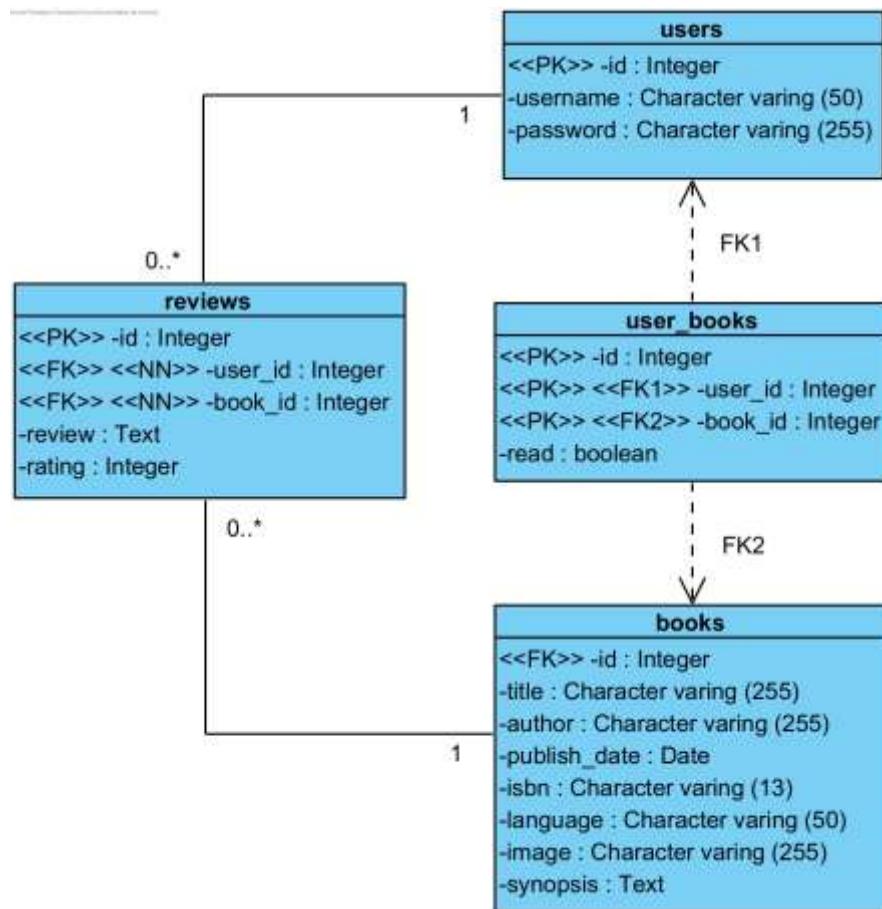


Figura 16 - Modelo relacional

8.2 Implementação da Base de dados

Aqui, detalhamos o *script* de implementação da base de dados, incluindo a criação das tabelas, definição de chaves primárias e estrangeiras, e a configuração geral do sistema de armazenamento de dados. Este está anexado como Apêndice 2.

8.3 Implementação da API

Este subtópico serve para referir o script desenvolvido da API, incluindo os endpoints criados, os métodos HTTP utilizados e a lógica implementada para permitir a comunicação entre o front-end e o back-end da aplicação. Este *script* está no apêndice 1.

8.4 Documentação da API

A documentação da API é essencial para garantir que outros desenvolvedores possam entender e utilizar a API de forma eficaz. Esta fornece uma visão detalhada dos *endpoints* disponíveis, os parâmetros necessários e os formatos de resposta esperados. Esta documentação está disponível no apêndice 3 e no seguinte url:

<https://documenter.getpostman.com/view/32992238/2sA3XJnREB>

8.5 Desenvolvimento da aplicação

Neste subtópico, apresentamos o desenvolvimento da aplicação Android "Librury". Detalhamos a criação da interface de utilizador e a implementação das funcionalidades principais. Esta seção abrange os desafios enfrentados e as soluções adotadas, proporcionando uma visão completa do processo de desenvolvimento da aplicação móvel.

8.5.1 Data flow da navegação da aplicação

O subtópico apresenta a estrutura de navegação entre as diferentes atividades da aplicação. Este diagrama de fluxo de dados ilustra como os utilizadores interagem com os botões e navegam de uma tela para outra, proporcionando uma visão clara do percurso do utilizador dentro da aplicação.



8.5.2 Data flow da API

Este data flow apresenta a interação entre as atividades da aplicação "Libruary" e os endpoints da API. Este fluxo de dados ilustra quais *endpoints* são acedidos por cada atividade, detalhando os dados enviados nas requisições (*requests*) e os dados recebidos nas respostas (*responses*). Através deste esquema, é possível compreender a comunicação entre o front-end e o back-end, garantindo que a troca de informações seja eficiente e segura. A tabela seguinte demonstra esta interação, destacando os *endpoints* utilizados, os parâmetros enviados e os dados retornados em cada caso.

Tabela 8 - Data flow da API

Activity	Endpoint	In (request)	Out (response)
MainActivity	/user_books/<int:user_id>	user_id	Dois arrays com as informações dos livros: books_to_read e books_read
	/book/<int:book_id>	book_id	Array com as informações do livro: id, título, autor, data de publicação, isbn, idioma, imagem da capa, número de páginas e a editora
	/user_rating/<int:user_id>/<int:book_id>	user_id book_id	Retorna a classificação do utilizador para aquele livro.
BookDetailActivity	/marc_book_as_read	book_id	Retorna uma mensagem positiva ou negativa dependendo de como o livro do utilizador estiver marcado
	/remove_book	action=delete book_id	Retorna uma mensagem se o livro foi removido ou não com sucesso

	/book/<int:book_id>	book_id	Array com as informações do livro: id, título, autor, data de publicação, isbn, idioma, imagem da capa, número de páginas, a editora e sinopse.
	/book_reviews/<int:book_id>	book_id	Retorna um array com todas as reviews, número total de reviews e o número total de páginas.
	/user_rating/<int:user_id>/<int:book_id>	user_id book_id	Retorna a classificação do utilizador para aquele livro.
	/user_read /<int:user_id>/<int:book_id>	user_id book_id	Retorna um boolean true ou false caso o utilizador tenha lido ou não o livro.
AddReviewActivity	/review	book_id review_text rating	Retorna se a review foi guardada com sucesso ou não.
LoginActivity	/login	username password	Retorna a mensagem se efetuou login com sucesso ou não e em conjunto os valores do token e do id do utilizador
	/validateToken	token	Retorna uma mensagem de validação do token.
UserProfileActivity	/profile/<int:user_id>	user_id	Retorna o username e todas a reviews feitas pelo utilizador.
RegisterActivity	/register	username password	Retorna a mensagem se o registo foi efetuado com sucesso ou não
AddBookActivity	/add_book	user_id book ISBN	Retorna a mensagem caso o livro for adicionado com sucesso ou não.
ScanBookActivity	/add_book	user_id book ISBN	Retorna a mensagem caso o livro for adicionado com sucesso ou não.

8.5.2 Android Manifest

O Android Manifest é um ficheiro importante para qualquer aplicação Android, pois define as configurações básicas e as permissões necessárias para o funcionamento adequado da aplicação. Ele também serve para declarar componentes como as atividades, além de especificar permissões que a aplicação precisa para aceder a recursos protegidos do dispositivo, como a câmara. Neste subtópico, explicamos o propósito do Android Manifest e detalhamos as implementações específicas realizadas para a aplicação. A próxima figura demonstra o conteúdo do Android Manifest utilizado na "Library", ao explicar as configurações e permissões.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="com.example.library">

    <uses-feature
        android:name="android.hardware.camera"
        android:required="false" />

    <uses-permission android:name="android.permission.CAMERA" />
    <uses-permission android:name="android.permission.INTERNET" />

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="library"
        android:networkSecurityConfig="@xml/network_security_config"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.Library"
        tools:targetApi="31">
        <activity android:name=".MainActivity" />
        <activity android:name=".BookDetailActivity" />
        <activity android:name=".AddReviewActivity" />
        <activity
            android:name=".LoginActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".RegisterActivity" />
        <activity android:name=".UserProfileActivity" />
        <activity android:name=".AddBookActivity" />
        <activity android:name=".ScanBookActivity" />
    </application>
</manifest>
```

Figura 17 - Android Manifest

O ficheiro `AndroidManifest.xml` como referido anteriormente define as configurações essenciais para o aplicativo Android. Aqui estão os detalhes dos principais pontos:

Utilização de Recursos:

Câmara: A aplicação pode usar a câmara, mas não é obrigatório (`android:required="false"`).

Permissões:

CAMERA: A aplicação pede permissão ao utilizador para utilizar a câmara.

INTERNET: A aplicação pede permissão para aceder à internet.

Atividades:

MainActivity: A atividade principal da aplicação.

BookDetailActivity: Atividade para os detalhes do livro.

AddReviewActivity: Atividade para adicionar uma *review*.

LoginActivity: Atividade de login, exportada para ser acessível externamente (`android:exported="true"`), definida como a atividade inicial com um *intent-filter* que especifica ações e categorias para o *launch* da aplicação.

RegisterActivity: Atividade para o registo de utilizador.

UserProfileActivity: Atividade de perfil do utilizador.

AddBookActivity: Atividade para adicionar um livro.

ScanBookActivity: Atividade para fazer scan de um livro.

8.5.3 Shared Preferences

A aplicação utiliza o mecanismo de Shared Preferences para armazenar de forma segura e eficiente dados essenciais para a persistência da sessão do utilizador. Na aplicação, as Shared Preferences são utilizadas para guardar apenas o token de autenticação e o id do utilizador, recebidos da API após o login. Este método de armazenamento permite que a aplicação mantenha o utilizador autenticado entre sessões, facilitando a experiência de uso contínuo. Na figura seguinte mostra a criação das variáveis do token de autenticação e do id do utilizador na *activity* de *login*.

```
val sharedpreferences = getSharedPreferences( name: "LibraryPrefs", Context.MODE_PRIVATE)
val userId = sharedpreferences.getInt( key: "userId", defValue: -1)
val authToken = sharedpreferences.getString( key: "authToken", defValue: "")
```

Figura 18 - Shared Preferences

8.5.5 Interface da aplicação

Este subtópico apresenta todas as *activities* principais da aplicação. Também vai detalhar os propósitos de cada uma com os elementos visuais finalizados de cada interface.

Activity register:



Figura 19 - Interface de registo

Este ecrã da aplicação serve para fazer o utilizador registar-se na aplicação para ter acesso às funcionalidades. O utilizar necessita de introduzir um *username* que não exista e uma password à escolha.

Login Activity:



Figura 20 - Interface de login

Nesta interface será possível ao utilizador efetuar o login na aplicação ao introduzir o *username* e a *password* atribuídos quando o utilizador fez o registo.

Main Activity:



Figura 21 - Interface da activity principal

Atividade inicial depois de o utilizador iniciar sessão na aplicação. Nesta página ele consegue ver os livros que adicionou à sua biblioteca pessoal, dividindo-os em lidos e por ler. Ainda nesta interface o utilizador ainda é capaz de aceder às funcionalidades de adicionar livro e navegar para o seu perfil.

Activity add_book:



Figura 22 - Interface para adicionar o livro

Nesta interface o utilizador é capaz de adicionar um livro à sua biblioteca pessoal. Como se pode verificar na figura 21, existem duas formas de o fazer, ou inserindo o isbn do livro manualmente ou ao digitalizar o código de barras do livro. Esta atividade ainda é possível para o utilizador voltar para a biblioteca pessoal ou verificar o seu perfil.

Activity Scan_book:



Figura 23 - interface do processo de scan do ISBN do livro

Nesta interface é possível ao utilizador ler o código de barras do livro como é exemplificado na figura 22, contudo o utilizador ainda tem a possibilidade de cancelar a leitura caso se tenha enganado.

Activity Book_details:



Figura 24 - Interface da activity de detalhes do livro

Nesta parte da aplicação o utilizador tem a capacidade de ver as informações do livro, bem como adicionar uma *review* ao livro ou caso de já tiver uma editá-la e marcar o livro como lido ou como não lido. Ainda é capaz de aceder às várias páginas *reviews* de outros utilizadores que o livro pode ter.

Activity Add_review:



Figura 25 - Interface para adicionar review do utilizador

O objetivo desta *activity* é efetuar a *review* do livro em que o utilizador em questão escolheu. Para isso ser feito o utilizador necessita de selecionar a quantidade de estrelas que pensa que o livro merece e adicionar uma justificação para o número de estrelas que deu.

Activity User_profile:

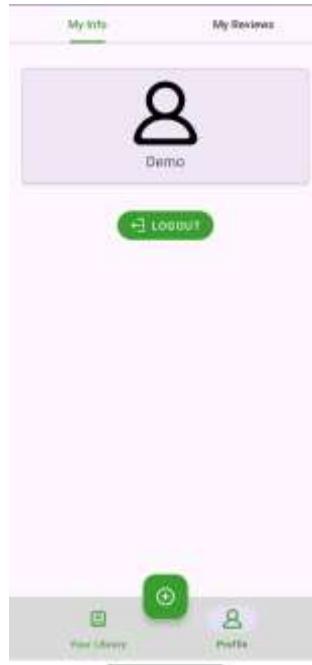


Figura 26 - Interface do perfil do utilizador

Nesta interface, demonstrada na figura 25, o utilizador tem acesso aos dados do seu perfil e a possibilidade de terminar sessão caso o queira fazer. Também tem a opção de rever as *reviews* feitas anteriormente.

Activity User_reviews:

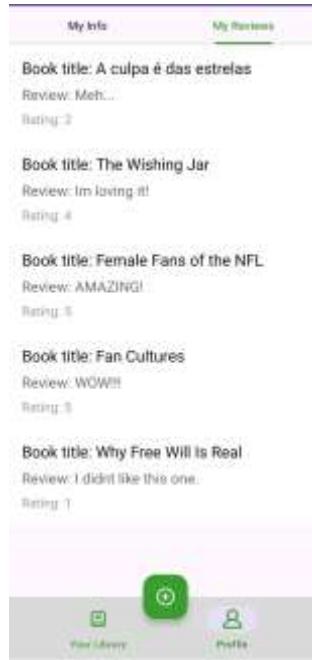


Figura 27 - Interface das reviews feitas pelo utilizador

Nesta parte o utilizador consegue ver o histórico das reviews feitas por ele ao longo da sua utilização na aplicação. É importante referir que o utilizador também consegue voltar para a biblioteca pessoal, assim como voltar a ver os dados pessoais no seu perfil ou adicionar um livro.

9. Análise dos resultados

O tópico de Análise dos Resultados apresenta uma avaliação detalhada do desempenho da aplicação após a sua implementação. Ao analisar a lista de requisitos previamente mencionada, verificou-se que foram levantados 18 requisitos no total, incluindo tanto funcionais quanto não funcionais. Em termos quantitativos, aproximadamente 77% dos requisitos iniciais foram totalmente cumpridos, cerca de 6% foram parcialmente implementados e 17% ficaram por implementar. Esta análise permite compreender melhor o impacto da aplicação sobre os utilizadores e garantir que ela atenda às expectativas de funcionalidade, usabilidade e eficiência.

Tabela 9 - Requisitos Cumpridos, não cumpridos e parcialmente cumpridos

Requisitos	Cumprido	Não cumprido	Parcialmente cumprido
RF. 1	X		
RF. 2	X		
RF. 3	X		
RF. 4	X		
RF. 5	X		
RF. 6	X		
RF. 7	X		
RF. 8	X		
RF. 9	X		
RF. 10			X
RF. 11	X		
RF. 12		X	
RF. 13		X	
RNF.1	X		
RNF.2	X		
RNF.3	X		
RNF.4		X	
RNF. 5	X		



Figura 28 – Representação gráfica das percentagens dos requisitos cumpridos

9.1 Limitações do projeto

Apesar dos esforços para desenvolver uma aplicação robusta e funcional, existem algumas limitações inerentes ao projeto que devem ser reconhecidas.

Dependência da API do Google Books: A aplicação depende fortemente da API do Google Books para obter informações detalhadas sobre os livros que ainda não se encontram na base de dados. Se a base de dados não tiver um livro específico nem a API do Google Books, a aplicação não será capaz de fornecer informações sobre esse livro. Além disso, quaisquer interrupções ou alterações na API do Google Books podem afetar a funcionalidade da nossa aplicação.

Funcionalidades Limitadas: A aplicação “Librury” oferece funcionalidades básicas de gestão de livros e revisões. No entanto, existem muitas outras funcionalidades mais avançadas oferecidas por outras aplicações semelhantes que não estão presentes na nossa aplicação. Por exemplo, a aplicação não suporta a criação de listas de leitura personalizadas, recomendações de livros baseadas nas preferências do utilizador, entre outros.

Escalabilidade: A aplicação foi testada a lidar com um número moderado de utilizadores e livros. No entanto, se o número de utilizadores ou livros aumentar significativamente, a aplicação pode enfrentar problemas de desempenho pois a escalabilidade da mesma não foi totalmente testada.

Estas limitações representam oportunidades para melhorias futuras e desenvolvimento contínuo da aplicação "Librury". Com o feedback dos utilizadores e com o passar do tempo, esperamos superar estas limitações e melhorar a aplicação.

9.2 Sugestões para o futuro

Este subtópico apresenta recomendações e ideias para futuras melhorias e expansões da aplicação. Estas sugestões visam não apenas resolver as limitações atuais, mas também enriquecer a experiência do utilizador e a funcionalidade da aplicação em versões futuras.

Calendário de Leitura: A implementação de um calendário de leitura permitiria o registo dos dias em que o utilizador lê. Esta funcionalidade proporcionaria uma visão clara dos hábitos de leitura e permitiria o acompanhamento do progresso mensal.

Meta Diária de Leitura: A definição de uma meta diária de leitura poderia funcionar como um desafio motivador. Seja por tempo, número de páginas, percentagem do livro ou até mesmo um livro inteiro por dia, esta funcionalidade poderia incentivar a leitura diária.

Organização de Livros por Coleções: A possibilidade de organizar livros em coleções personalizadas, incluindo os "favoritos", permitiria uma maior organização e personalização da biblioteca.

Pesquisa de Livros por Título ou Autor: A adição de livros à biblioteca poderia ser simplificada permitindo a pesquisa por título ou autor, em vez de apenas pelo ISBN. Esta funcionalidade tornaria o processo de adição de novos livros mais simples e agradável.

Exploração de Livros: A funcionalidade de explorar livros de diferentes géneros, tendências, entre outros, poderia expandir os horizontes literários. Esta funcionalidade poderia introduzir novos livros e autores que de outra forma poderiam passar despercebidos.

10. Conclusão

A elaboração da aplicação "Libruary" constituiu um desafio significativo, bem como uma oportunidade inestimável de aprendizagem no âmbito do desenvolvimento para dispositivos móveis. Este projeto permitiu-nos aprofundar os nossos conhecimentos na linguagem de programação Kotlin, na criação de bases de dados, na construção de APIs REST, bem como na utilização de APIs externas, como a do Google Books.

Adicionalmente, a colaboração em equipa foi um elemento crucial para o sucesso do projeto. Observou-se uma sinergia extremamente positiva entre os membros da equipa, sendo evidente a complementaridade e o apoio mútuo ao longo do projeto.

Contudo, nem todos os aspectos decorreram conforme planeado. Algumas funcionalidades, como a paginação de avaliações ou a leitura do ISBN através da câmara do telemóvel, exigiram mais tempo para serem implementadas do que o inicialmente previsto. Enfrentámos também alguns desafios inesperados, particularmente quando relacionados com aspectos muito específicos da linguagem Kotlin ou do design de interfaces para Android (utilizando Material Design), dado que foi a nossa primeira experiência com tais elementos. Apesar destes contratemplos, conseguimos adaptar-nos e superar estes desafios.

Em conclusão, o projeto "Libruary" foi uma experiência enriquecedora que proporcionou uma compreensão mais profunda do desenvolvimento de aplicações móveis e da gestão de projetos. As lições aprendidas serão certamente valiosas para projetos futuros.

Referências Bibliográficas

- [1] «Android Studio», *Android Developers*. Disponível em: <https://developer.android.com/studio>.
- [2] «Discord». Disponível em: <https://discord.com>.
- [3] «Figma». Disponível em: <https://www.figma.com/design/>.
- [4] «Firebase Documentation». Disponível em: <https://firebase.google.com/docs>.
- [5] «Flask Documentation». Disponível em: <https://flask.palletsprojects.com/en/3.0.x/>.
- [6] «Git». Disponível em: <https://www.git-scm.com/>.
- [7] «Github», *Github*. Disponível em: <https://github.com>.
- [8] «Glide». Disponível em: <https://bumptech.github.io/glide/>.
- [9] «Gradle Build Tool», *Gradle*, 31 de maio de 2024. Disponível em: <https://gradle.org/>.
- [10] «Jetpack libraries». Disponível em: <https://developer.android.com/jetpack/androidx/explorer>.
- [11] «JWT». Disponível em: <http://jwt.io/>.
- [12] «Kotlin Programming Language», *Kotlin*. Disponível em: <https://kotlinlang.org/>.
- [13] «Material Design», *Material Design*. Disponível em: [https://m3.material.io/](https://m3.material.io).
- [14] «Paging library», *Android Developers*. Disponível em: <https://developer.android.com/topic/libraries/architecture/paging/v3-overview>.
- [15] «pgAdmin». Disponível em: <https://www.pgadmin.org/>.
- [16] «Pgcrypto», *PostgreSQL Documentation*, 9 de maio de 2024. Disponível em: <https://www.postgresql.org/docs/16/pgcrypto.html>.
- [17] PostgreSQL Global Development, «PostgreSQL», *PostgreSQL*, 8 de junho de 2024. Disponível em: <https://www.postgresql.org/>.
- [18] F. D. Gregorio, «psycopg2». Disponível em: <https://psycopg.org/>.

- [19] «Vercel». Disponível em: <https://vercel.com/home>.
- [20] «Volley», *Volley*. Disponível em: <https://google.github.io/volley/>.

Apêndices

Apêndice 1. Script da API

```
1. import os
2. import datetime
3. from functools import wraps
4. import jwt
5. import requests
6. from flask import Flask, request, jsonify, send_from_directory
7. import psycopg2
8. from psycopg2.extras import DictCursor
9. from psycopg2.extensions import AsIs
10. from werkzeug.middleware.proxy_fix import ProxyFix
11.
12. app = Flask(__name__)
13. app.wsgi_app = ProxyFix(app.wsgi_app)
14. app.config['SECRET_KEY'] = os.environ.get('SECRET_KEY')
15.
16. conn = psycopg2.connect(
17.     dbname=os.environ.get('DB_NAME'),
18.     user=os.environ.get('DB_USER'),
19.     password=os.environ.get('DB_PASS'),
20.     host=os.environ.get('DB_HOST'),
21.     port=os.environ.get('DB_PORT')
22. )
23.
24. def token_required(f):
25.     @wraps(f)
26.     def decorated(*args, **kwargs):
27.         token = request.headers.get('x-access-token')
28.         if not token:
29.             return jsonify({'error': 'Token is missing!'}), 401
30.         try:
31.             data = jwt.decode(token, app.config['SECRET_KEY'], algorithms=['HS256'])
32.             with conn.cursor() as cur:
33.                 cur.execute("SELECT * FROM users WHERE id = %s", (data['user_id'],))
34.                 current_user = cur.fetchone()
35.                 if not current_user:
36.                     return jsonify({'error': 'Invalid token'}), 401
37.             except jwt.ExpiredSignatureError:
38.                 return jsonify({'error': 'Token has expired'}), 401
39.             except jwt.InvalidTokenError:
40.                 return jsonify({'error': 'Invalid token'}), 401
41.             return f(current_user, *args, **kwargs)
42.         return decorated
43.
44.
45. @app.route('/', methods = ["GET"])
46. def home():
47.     return "Welcome to library API!"
48.
49. @app.route('/favicon.ico')
50. def favicon():
51.     return send_from_directory(os.path.dirname(os.path.realpath(__file__)), 'favicon.ico',
mimetype='image/vnd.microsoft.icon')
52.
53. @app.route('/validateToken', methods=['POST'])
54. def validate_token():
55.     data = request.get_json()
56.     token = data.get('token')
57.
58.     if not token:
59.         return jsonify({'message': 'Token is missing'}), 400
60.
```

```

61.     try:
62.         decoded_token = jwt.decode(token, app.config['SECRET_KEY'], algorithms=['HS256'])
63.
64.         exp_date = datetime.datetime.fromtimestamp(decoded_token['exp'],
tz=datetime.timezone.utc)
65.
66.         if exp_date < datetime.datetime.now(datetime.timezone.utc):
67.             return jsonify({'is_valid': False, 'message': 'Token has expired'}), 401
68.
69.         return jsonify({'is_valid': True, 'message': 'Token is valid'})
70.     except jwt.ExpiredSignatureError:
71.         return jsonify({'is_valid': False, 'message': 'Token has expired'}), 401
72.     except jwt.InvalidTokenError:
73.         return jsonify({'is_valid': False, 'message': 'Invalid token'}), 401
74.
75. @app.route('/register', methods=['POST'])
76. def register():
77.     data = request.get_json()
78.     if not data:
79.         return jsonify({'message': 'No data provided'}), 400
80.
81.     username = data.get('username')
82.     password = data.get('password')
83.     if not username or not password:
84.         return jsonify({'message': 'Username and password are required'}), 400
85.
86.     try:
87.         with conn.cursor() as cur:
88.             cur.execute("SELECT * FROM users WHERE username = %s", (username,))
89.             existing_user = cur.fetchone()
90.     except Exception as e:
91.         print(f"Error: {e}")
92.     finally:
93.         conn.commit()
94.
95.     if existing_user:
96.         return jsonify({'message': 'Username already exists'}), 400
97.
98.     try:
99.         with conn.cursor() as cur:
100.             cur.execute("INSERT INTO users (username, password) VALUES (%s,
pgp_sym_encrypt(%s, %s))", (username, password, app.config['SECRET_KEY']))
101.             conn.commit()
102.             return jsonify({'message': 'Registered successfully!'})
103.     except Exception as e:
104.         return jsonify({'message': 'Registration failed', 'error': str(e)}), 500
105.
106. @app.route('/login', methods=['POST'])
107. def login():
108.     data = request.get_json()
109.     if not data:
110.         return jsonify({'message': 'No data provided'}), 400
111.
112.     username = data.get('username')
113.     password = data.get('password')
114.     if not username or not password:
115.         return jsonify({'message': 'Username and password are required'}), 400
116.
117.     with conn.cursor() as cur:
118.         cur.execute("SELECT id, username, pgp_sym_decrypt(password::bytea, %s) as password
FROM users WHERE username = %s", (app.config['SECRET_KEY'], username))
119.         user = cur.fetchone()
120.
121.     if not user or user[2] != password:
122.         return jsonify({'message': 'Login failed!'}), 401
123.
124.     token = jwt.encode({
125.         'user_id': user[0],
126.         'exp': datetime.datetime.utcnow() + datetime.timedelta(minutes=60)
127.     }, app.config['SECRET_KEY'], algorithm='HS256')

```

```

128.
129.     return jsonify({'message': 'Logged in successfully!', 'token': token, 'user_id': user[0]})  

130.
131. @app.route('/books', methods=['GET'])
132. def get_books():
133.     try:
134.         with conn.cursor(cursor_factory=DictCursor) as cur:
135.             cur.execute("SELECT * FROM books")
136.             books = cur.fetchall()
137.             output = [{  

138.                 'id': book['id'],  

139.                 'title': book['title'],  

140.                 'author': book['author'],  

141.                 'published_date': book['published_date'],  

142.                 'isbn': book['isbn'],  

143.                 'language': book['language'],  

144.                 'image': book['image'],  

145.                 'pages': book['pages'],  

146.                 'publisher': book['publisher']
147.             } for book in books]
148.             return jsonify({'books': output})
149.     except Exception as e:
150.         return jsonify({'message': 'Error fetching books', 'error': str(e)}), 500
151.
152. @app.route('/book/<int:book_id>', methods=['GET'])
153. def get_book(book_id):
154.     try:
155.         with conn.cursor(cursor_factory=DictCursor) as cur:
156.             cur.execute("SELECT * FROM books WHERE id = %s", (book_id,))
157.             book = cur.fetchone()
158.             if book is None:
159.                 return jsonify({'message': 'Book not found'}), 404
160.
161.             output = {
162.                 'id': book['id'],
163.                 'title': book['title'],
164.                 'author': book['author'],
165.                 'published_date': book['published_date'],
166.                 'isbn': book['isbn'],
167.                 'language': book['language'],
168.                 'image': book['image'],
169.                 'pages': book['pages'],
170.                 'publisher': book['publisher']
171.             }
172.
173.             return jsonify({'book': output})
174.     except Exception as e:
175.         return jsonify({'message': 'Error fetching book', 'error': str(e)}), 500
176.
177. @app.route('/review', methods=['POST'])
178. @token_required
179. def add_review(current_user):
180.     data = request.get_json()
181.     if not data:
182.         return jsonify({'message': 'No data provided'}), 400
183.
184.     book_id = data.get('book_id')
185.     review_text = data.get('review_text')
186.     rating = data.get('rating')
187.
188.     if not book_id or not review_text or rating is None:
189.         return jsonify({'message': 'Book ID, Review text, and Rating are required'}), 400
190.
191.     with conn.cursor() as cur:
192.         cur.execute("SELECT * FROM reviews WHERE book_id = %s AND user_id = %s", (book_id, current_user[0]))
193.         existing_review = cur.fetchone()
194.
195.         if existing_review:

```

```

196.         with conn.cursor() as cur:
197.             cur.execute("UPDATE reviews SET review = %s, rating = %s WHERE id = %s",
198.             (review_text, rating, existing_review[0]))
199.             message = 'Review updated successfully!'
200.         else:
201.             with conn.cursor() as cur:
202.                 cur.execute("INSERT INTO reviews (book_id, user_id, review, rating) VALUES
203.                 (%s, %s, %s, %s)", (book_id, current_user[0], review_text, rating))
204.             message = 'Review added successfully!'
205.     try:
206.         conn.commit()
207.     return jsonify({'message': message})
208. except Exception as e:
209.     return jsonify({'message': 'Error adding/updating review', 'error': str(e)}), 500
210.
211. @app.route('/profile/<int:user_id>', methods=['GET'])
212. @token_required
213. def get_profile(current_user, user_id):
214.     try:
215.         if current_user[0] != user_id:
216.             return jsonify({'message': 'Unauthorized'}), 403
217.
218.         with conn.cursor(cursor_factory=DictCursor) as cur:
219.             cur.execute("SELECT * FROM users WHERE id = %s", (user_id,))
220.             user = cur.fetchone()
221.
222.         if not user:
223.             return jsonify({'message': 'User not found'}), 404
224.
225.         with conn.cursor(cursor_factory=DictCursor) as cur:
226.             cur.execute("SELECT reviews.book_id, books.title, reviews.review, reviews.rating
227.             FROM reviews JOIN books ON reviews.book_id = books.id WHERE reviews.user_id = %s", (user_id,))
228.             reviews = cur.fetchall()
229.
230.             review_list = [{book_id: review['book_id'], book_title: review['title'],
231. 'review': review['review'], 'rating': review['rating']} for review in reviews]
232.             return jsonify({'username': user['username'], 'reviews': review_list})
233.     except Exception as e:
234.         return jsonify({'message': 'Error fetching profile', 'error': str(e)}), 500
235.
236. @app.route('/add_book', methods=['POST'])
237. @token_required
238. def add_book(current_user):
239.     data = request.get_json()
240.
241.     if not data:
242.         return jsonify({'message': 'No data provided'}), 400
243.
244.     isbn = data.get('isbn')
245.     user_id = data.get('user_id')
246.
247.     if not isbn or not user_id:
248.         return jsonify({'message': 'ISBN and user ID are required'}), 400
249.
250.     with conn.cursor(cursor_factory=DictCursor) as cur:
251.         cur.execute("SELECT * FROM books WHERE isbn = %s", (isbn,))
252.         book = cur.fetchone()
253.
254.     if not book:
255.         google_books_url = f'https://www.googleapis.com/books/v1/volumes?q=isbn:{isbn}'
256.         try:
257.             response = requests.get(google_books_url)
258.             response.raise_for_status()
259.             book_data = response.json().get('items', [])
260.
261.             book_info = book_data[0].get('volumeInfo', {})

```

```

262.         title = book_info.get('title', 'N/A')
263.         author = ', '.join(book_info.get('authors', []))
264.         language = book_info.get('language', 'N/A')
265.         image = book_info.get('imageLinks', {}).get('thumbnail')
266.         pages = book_info.get('pageCount')
267.         publisher = book_info.get('publisher')
268.
269.     def format_published_date(published_date):
270.         try:
271.             return datetime.datetime.strptime(published_date, '%Y-%m-%d').date()
272.         except ValueError:
273.             try:
274.                 return datetime.datetime.strptime(published_date, '%Y-%m').date().replace(day=1)
275.             except ValueError:
276.                 return datetime.datetime.strptime(published_date, '%Y-%m-%d').date().replace(day=1, month=1)
277.
278.     published_date = format_published_date(book_info.get('publishedDate', '1000'))
279.
280.     with conn.cursor() as cur:
281.         cur.execute("INSERT INTO books (title, author, published_date, isbn,
282. language, image, pages, publisher) VALUES (%s, %s, %s, %s, %s, %s, %s, %s) RETURNING id",
283. (title, author, published_date, isbn, language, image, pages, publisher))
284.         book_id = cur.fetchone()[0]
285.         conn.commit()
286.     except requests.exceptions.RequestException as e:
287.         app.logger.error(f'Error fetching book data from Google Books API: {e}')
288.     return jsonify({'message': 'Error fetching book data from Google Books API',
289. 'error': str(e)}), 500
290. else:
291.     book_id = book['id']
292.
293.     with conn.cursor() as cur:
294.         cur.execute("SELECT * FROM user_books WHERE user_id = %s AND book_id = %s",
295. (user_id, book_id))
296.         user_book = cur.fetchone()
297.
298.     if user_book:
299.         return jsonify({'message': 'Book already added to user library'}), 400
300.
301.     try:
302.         with conn.cursor() as cur:
303.             cur.execute("INSERT INTO user_books (user_id, book_id) VALUES (%s, %s)",
304. (user_id, book_id))
305.             conn.commit()
306.         return jsonify({'message': 'Book added to user library successfully!'})
307.     except Exception as e:
308.         app.logger.error(f'Error adding book to user library: {e}')
309.     return jsonify({'message': 'Error adding book to user library', 'error': str(e)}),
310. 500
311.
312. @app.route('/mark_book_as_read', methods=['POST'])
313. @token_required
314. def mark_book_as_read(current_user):
315.     data = request.get_json()
316.     book_id = data.get('book_id')
317.     if not book_id:
318.         return jsonify({'message': 'Book ID is required'}), 400
319.
320.     with conn.cursor() as cur:
321.         cur.execute("SELECT * FROM user_books WHERE user_id = %s AND book_id = %s",
322. (current_user[0], book_id))
323.         user_book = cur.fetchone()
324.
325.     if not user_book:
326.         return jsonify({'message': 'Book not found in user library'}), 404
327.
328.     try:
329.         with conn.cursor() as cur:
330.             cur.execute("UPDATE user_books SET read = TRUE WHERE user_id = %s AND book_id = %s",
331. (current_user[0], book_id))
332.             conn.commit()
333.         return jsonify({'message': 'Book marked as read successfully!'})
334.     except Exception as e:
335.         app.logger.error(f'Error marking book as read: {e}')
336.     return jsonify({'message': 'Error marking book as read', 'error': str(e)}),
337. 500

```

```

323.                 cur.execute("UPDATE user_books SET read = NOT read WHERE id = %s",
324. (user_book[0],))
325.             conn.commit()
326.             return jsonify({'message': 'Book read status toggled successfully!'})
327.         except Exception as e:
328.             return jsonify({'message': 'Error toggling book read status', 'error': str(e)}),
329.             500
330.     328.
331. @app.route('/user_read/<int:user_id>/<int:book_id>', methods=['GET'])
332. @token_required
333. def check_user_read_book(current_user, user_id, book_id):
334.     if current_user[0] != user_id:
335.         return jsonify({'message': 'Unauthorized'}), 403
336.     with conn.cursor() as cur:
337.         cur.execute("SELECT * FROM user_books WHERE user_id = %s AND book_id = %s AND read = true",
338. (user_id, book_id))
339.         read_book_entry = cur.fetchone()
340.         has_read = read_book_entry is not None
341.         return jsonify({'has_read': has_read}), 200
342.     340.
343. @app.route('/remove_book', methods=['POST'])
344. @token_required
345. def remove_book(current_user):
346.     data = request.get_json()
347.     if not data or 'action' not in data or data['action'] != 'delete':
348.         return jsonify({'message': 'Invalid action'}), 400
349.     book_id = data.get('book_id')
350.     if not book_id:
351.         return jsonify({'message': 'Book ID is required'}), 400
352.     with conn.cursor() as cur:
353.         cur.execute("SELECT * FROM user_books WHERE user_id = %s AND book_id = %s",
354. (current_user[0], book_id))
355.         user_book = cur.fetchone()
356.     if not user_book:
357.         return jsonify({'message': 'Book not found in user library'}), 404
358.     try:
359.         with conn.cursor() as cur:
360.             cur.execute("DELETE FROM user_books WHERE id = %s", (user_book[0],))
361.             conn.commit()
362.             return jsonify({'message': 'Book removed from user library successfully!'})
363.         except Exception as e:
364.             app.logger.error(f'Error removing book from user library: {e}')
365.             return jsonify({'message': 'Error removing book from user library', 'error': str(e)}),
366.             500
367.     365.
368.     366.
369. @app.route('/book_reviews/<int:book_id>', methods=['GET'])
370. @token_required
371. def get_book_reviews(current_user, book_id):
372.     page = request.args.get('page', 1, type=int)
373.     per_page = request.args.get('per_page', 5, type=int)
374.     with conn.cursor(cursor_factory=DictCursor) as cur:
375.         cur.execute("SELECT reviews.id, users.username, reviews.review, reviews.rating,
376. reviews.user_id FROM reviews INNER JOIN users ON reviews.user_id = users.id WHERE book_id = %s
377. ORDER BY id LIMIT %s OFFSET %s", (book_id, per_page, (page - 1) * per_page))
378.         reviews = cur.fetchall()
379.         review_list = [{id: review['id'], 'username': review['username'], 'review': review['review'],
380. 'rating': review['rating'], 'user_id': review['user_id']} for review in reviews]
381.     with conn.cursor() as cur:
382.         cur.execute("SELECT COUNT(*) FROM reviews WHERE book_id = %s", (book_id,))
383.         total_reviews = cur.fetchone()[0]
384.     return jsonify({
385.         'reviews': review_list,
386.         'total_reviews': total_reviews,

```

```

384.         'page': page,
385.         'pages': (total_reviews // per_page) + (total_reviews % per_page > 0)
386.     })
387.
388. @app.route('/user_rating/<int:user_id>/<int:book_id>', methods=['GET'])
389. @token_required
390. def get_user_rating(current_user, user_id, book_id):
391.     if current_user[0] != user_id:
392.         return jsonify({'message': 'Unauthorized'}), 403
393.
394.     with conn.cursor() as cur:
395.         cur.execute("SELECT rating FROM reviews WHERE user_id = %s AND book_id = %s",
396. (user_id, book_id))
397.         review = cur.fetchone()
398.
399.     if review is None:
400.         return jsonify({'rating': 0}), 200
401.
402.     return jsonify({'rating': review[0]}), 200
403.
404. @app.route('/user_books/<int:user_id>', methods=['GET'])
405. @token_required
406. def get_user_books(current_user, user_id):
407.     if current_user[0] != user_id:
408.         return jsonify({'message': 'Unauthorized'}), 403
409.
410.     try:
411.         with conn.cursor(cursor_factory=DictCursor) as cur:
412.             cur.execute("SELECT * FROM user_books JOIN books ON user_books.book_id = books.id WHERE user_id = %s AND read = FALSE", (user_id,))
413.             books_to_read = cur.fetchall()
414.             with conn.cursor(cursor_factory=DictCursor) as cur:
415.                 cur.execute("SELECT * FROM user_books JOIN books ON user_books.book_id = books.id WHERE user_id = %s AND read = TRUE", (user_id,))
416.                 books_read = cur.fetchall()
417.
418.             books_to_read_output = [
419.                 {
420.                     'book_id': user_book['book_id'],
421.                     'title': user_book['title'],
422.                     'author': user_book['author'],
423.                     'image': user_book['image']
424.                 } for user_book in books_to_read]
425.
426.             books_read_output = [
427.                 {
428.                     'book_id': user_book['book_id'],
429.                     'title': user_book['title'],
430.                     'author': user_book['author'],
431.                     'image': user_book['image']
432.                 } for user_book in books_read]
433.
434.             return jsonify({
435.                 'books_to_read': books_to_read_output,
436.                 'books_read': books_read_output
437.             })
438.         except Exception as e:
439.             return jsonify({'message': 'Error fetching user books', 'error': str(e)}), 500
440.
441. @app.errorhandler(404)
442. def not_found(error):
443.     return jsonify({'error': 'Not found'}), 404
444.
445. @app.errorhandler(400)
446. def bad_request(error):
447.     return jsonify({'error': 'Bad request'}), 400
448.
449. @app.errorhandler(500)
450. def server_error(error):
451.     return jsonify({'error': 'Server error'}), 500

```

```
451. if __name__ == '__main__':
452.     import logging
453.     logging.basicConfig(filename='api.log', level=logging.INFO)
454.     app.run(host='0.0.0.0')
455.
```

Apêndice 2. Script da Base de dados

```
1. CREATE TABLE public.user_books (
2.     id integer NOT NULL,
3.     user_id integer NOT NULL,
4.     book_id integer NOT NULL,
5.     read boolean DEFAULT false
6. );
7.
8. CREATE TABLE public.users (
9.     id integer NOT NULL,
10.    username character varying(50) NOT NULL,
11.    password character varying(1000) NOT NULL
12. );
13.
14. CREATE TABLE public.books (
15.     id integer NOT NULL,
16.     title character varying(255) NOT NULL,
17.     isbn character varying(13)
18. );
19.
20. CREATE TABLE public.reviews (
21.     id integer NOT NULL,
22.     book_id integer NOT NULL,
23.     user_id integer NOT NULL,
24.     review_text text,
25.     rating integer
26. );
27.
28. ALTER TABLE ONLY public.books
29.     ADD CONSTRAINT books_isbn_key UNIQUE (isbn);
30.
31. ALTER TABLE ONLY public.books
32.     ADD CONSTRAINT books_pkey PRIMARY KEY (id);
33.
34. ALTER TABLE ONLY public.reviews
35.     ADD CONSTRAINT reviews_pkey PRIMARY KEY (id);
36.
37. ALTER TABLE ONLY public.user_books
38.     ADD CONSTRAINT user_books_pkey PRIMARY KEY (id);
39.
40. ALTER TABLE ONLY public.user_books
41.     ADD CONSTRAINT user_books_user_id_book_id_key UNIQUE (user_id, book_id);
42.
43. ALTER TABLE ONLY public.users
44.     ADD CONSTRAINT users_pkey PRIMARY KEY (id);
45.
46. ALTER TABLE ONLY public.users
47.     ADD CONSTRAINT users_username_key UNIQUE (username);
48.
49. ALTER TABLE ONLY public.reviews
50.     ADD CONSTRAINT reviews_book_id_fkey FOREIGN KEY (book_id) REFERENCES public.books(id);
51.
52. ALTER TABLE ONLY public.reviews
53.     ADD CONSTRAINT reviews_user_id_fkey FOREIGN KEY (user_id) REFERENCES public.users(id);
54.
55. ALTER TABLE ONLY public.user_books
56.     ADD CONSTRAINT user_books_book_id_fkey FOREIGN KEY (book_id) REFERENCES public.books(id);
57.
58. ALTER TABLE ONLY public.user_books
59.     ADD CONSTRAINT user_books_user_id_fkey FOREIGN KEY (user_id) REFERENCES public.users(id);
60.
```

Apêndice 3. Ficheiro Postman API
