TRƯỜNG ĐẠI HỌC SÀI GÒN
KHOA CÔNG NGHỆ THÔNG TIN

PHÁT TRIỂN PHẦN MỀM MÃ NGUỒN MỞ

Xây ứng dụng di động

EBOOK trên Android và Python

GVHD: Từ Lãng Phiêu
SV: Đỗ Ngọc Lương Tuấn - 3120410583
Phạm Thanh Tuấn - 3120410589
Phạm Hào Vĩnh - 3120410618
Nguyễn Thanh Vũ - 3120410623

TP. HỒ CHÍ MINH, THÁNG 5/2024

DANH SÁCH THÀNH VIÊN NHÓM 27

MSSV	Họ và tên thành viên
3120410583	Đỗ Ngọc Lương Tuấn
3120410589	Phạm Thanh Tuấn
3120410618	Phạm Hào Vĩnh
3120410623	Nguyễn Thanh Vũ

Mục lục

Lời mở đầu

Trong thế giới hiện đại, sự phát triển của công nghệ thông tin đã trở thành một yếu tố quan trọng không thể thiếu đối với mọi lĩnh vực, bao gồm cả việc đọc sách. Công nghệ này không chỉ giúp cho việc đọc sách trở nên thuận tiện hơn mà còn mở ra một loạt các cơ hội mới cho người đọc.

Để tận dụng những ưu điểm của công nghệ, chúng em đã phát triển một ứng dụng đọc sách tiện lợi và linh hoạt. Ứng dụng này không chỉ cung cấp một thư viện sách đa dạng, mà còn tích hợp các tính năng tiện ích như đánh dấu trang, tìm kiếm nội dung, và gợi ý sách dựa trên sở thích cá nhân của người đọc. Ứng dụng đọc sách của chúng em không chỉ là một công cụ đơn thuần để đọc sách mà còn là một nguồn tài nguyên giáo dục và giải trí đa dạng, giúp cho việc đọc sách trở nên thú vị và tiện lợi hơn bao giờ hết.

Trong quá trình tìm hiểu, phân tích và thiết kế chương trình, dù đã rất cố gắng nhưng do với kiến thức có hạn cùng với thời gian khá ngắn nên nhóm không thể tránh khỏi những hạn chế và sai sót, rất mong nhận được sự đóng góp ý kiến của thầy.

Qua đây nhóm em cũng xin gửi lời cảm ơn chân thành đến thầy Từ Lãng Phiêu đã giúp đỡ nhóm em trong quá trình làm đề tài này.

1 CƠ SỞ LÝ THUYẾT

1.1 Ngôn ngữ Python

Python là ngôn ngữ lập trình máy tính bậc cao thường được sử dụng để xây dựng trang web và phần mềm, tự động hóa các tác vụ và tiến hành phân tích dữ liệu. Python là ngôn ngữ có mục đích chung, nghĩa là nó có thể được sử dụng để tạo nhiều chương trình khác nhau và không chuyên biệt cho bất kỳ vấn đề cụ thể nào.

Tính linh hoạt này, cùng với sự thân thiện với người mới bắt đầu, đã khiến nó trở thành một trong những ngôn ngữ lập trình được sử dụng nhiều nhất hiện nay. Một cuộc khảo sát được thực hiện bởi công ty phân tích ngành RedMonk cho thấy rằng đây là ngôn ngữ lập trình phổ biến thứ hai đối với các nhà phát triển vào năm 2021.

- Python được phát triển vào cuối những năm 1980 bởi Guido van Rossum tại Viện Nghiên cứu Quốc gia về Toán học và Khoa học Máy tính ở Hà Lan với tư cách là người kế thừa ngôn ngữ ABC có khả năng xử lý và giao tiếp ngoại lệ.
- Python có nguồn gốc từ các ngôn ngữ lập trình như ABC, Modula 3, small talk, Algol-68.
- Van Rossum đã chọn tên Python cho ngôn ngữ mới từ một chương trình truyền hình, Monty Python's Flying Circus.
- Trang Python là một tệp có phần mở rộng .py chứa có thể là sự kết hợp của Thẻ HTML và tập lệnh Python.
- Vào tháng 12 năm 1989, người sáng tạo đã phát triển trình thông dịch python đầu tiên như một sở thích, và sau đó vào ngày 16 tháng 10 năm 2000, Python 2.0 được phát hành với nhiều tính năng mới.
- Python là mã nguồn mở, có nghĩa là bất kỳ ai cũng có thể tải xuống miễn phí từ trang chủ và sử dụng nó để phát triển các chương trình. Mã nguồn của nó có thể được truy cập và sửa đổi theo yêu cầu trong dự án.
- Python có thể được sử dụng trên nhiều hệ điều hành máy tính khác nhau, chẳng hạn như Windows, macOS, Linux và Unix.

Ứng dụng của Python:

- Python thường được sử dụng để phát triển trang web và phần mềm, tự động hóa tác vụ, phân tích dữ liệu và trực quan hóa dữ liệu. Vì tương đối dễ học, Python đã được nhiều người không phải là lập trình viên như kế toán và nhà khoa học áp dụng cho nhiều công việc hàng ngày, chẳng hạn như tổ chức tài chính.
- Python đã trở thành một yếu tố chính trong khoa học dữ liệu, cho phép các nhà phân tích dữ liệu và các chuyên gia khác sử dụng ngôn ngữ này để thực hiện các phép tính thống kê phức tạp, tạo trực quan hóa dữ liệu, xây dựng thuật toán học máy, thao tác và phân tích dữ liệu cũng như hoàn thành các nhiệm vụ khác liên quan đến dữ liệu.
- Python thậm chí có thể được sử dụng bởi những người mới bắt đầu để tự động hóa các tác vụ đơn giản trên máy tính—chẳng hạn như đổi tên tệp, tìm và tải xuống nội dung trực tuyến hoặc gửi email hoặc văn bản theo khoảng thời gian mong muốn. Trong phát triển phần mềm, Python có thể hỗ trợ các tác vụ như kiểm soát bản dựng, theo dõi lỗi và thử nghiệm. Với Python, các nhà phát triển phần mềm có thể tự động kiểm tra các sản phẩm hoặc tính năng mới. Một số công cụ Python được sử dụng để kiểm thử phần mềm bao gồm Green và Requestium.

Đặc tính của Python: Python đang trở nên phổ biến trong cộng đồng lập trình nhờ có các đặc tính sau:

- Ngôn ngữ thông dịch: Python được xử lý trong thời gian chạy bởi Trình thông dịch Python.
- Ngôn ngữ hướng đối tượng: Nó hỗ trợ các tính năng và kỹ thuật lập trình hướng đối tượng.
- Ngôn ngữ dễ học: Python rất dễ học, đặc biệt là cho người mới bắt đầu.

- Cú pháp đơn giản: Việc hình thành cú pháp Python rất đơn giản và dễ hiểu, điều này cũng làm cho nó trở nên phổ biến.
- Dễ đọc: Mã nguồn Python được xác định rõ ràng và có thể nhìn thấy bằng mắt.
- Di động: Mã Python có thể chạy trên nhiều nền tảng phần cứng có cùng giao diện.
- Có thể cải tiến: Python cung cấp một cấu trúc cải tiến để hỗ trợ các chương trình lớn sau đó là shell-script.
- Có thể mở rộng: Người dùng có thể thêm các mô-đun cấp thấp vào trình thông dịch Python.

1.2 Các thư viện sử dụng

1.2.1 FastAPI

FastAPI là framework web hiện đại, hiệu suất cao được xây dựng dành riêng cho việc phát triển API RESTful với Python 3.7 trở lên. Nó dựa trên các tiêu chuẩn Python, giúp các nhà phát triển:

- Viết code ít hơn: FastAPI cung cấp nhiều tính năng được tích hợp sẵn, giúp mã code giảm thiểu sự lặp lại.
- Triển khai nhanh hơn: Nhờ vào hiệu suất cao và tối ưu hóa, FastAPI giúp triển khai API nhanh chóng và dễ dàng.
- Tài liệu đầy đủ: FastAPI tự động tạo tài liệu API toàn diện và dễ sử dụng, giúp người dùng dễ dàng hiểu và tích hợp API.
- Hỗ trợ mạnh mẽ: FastAPI hỗ trợ nhiều tính năng nâng cao như: Xác thực và ủy quyền; Xử lý lỗi;Kiểm tra mô hình; Tự động hóa tài liệu...

Ngoài ra , FastAPI còn có các điểm nổi bật như:

- Nhanh: FastAPI được đánh giá là một trong những framework Python nhanh nhất hiện nay, ngang ngửa với các ngôn ngữ như Node.js và Go.
- Dễ học và dễ sử dụng: FastAPI có cú pháp đơn giản, dễ hiểu và dễ học, ngay cả với những người mới bắt đầu.Hơn nữa ,FastAPI còn cung cấp nhiều công cụ và tính năng giúp việc phát triển API trở nên dễ dàng và hiệu quả hơn.
- Hiệu quả: FastAPI được tối ưu hóa để mang lại hiệu suất cao và sử dụng ít tài nguyên máy tính.
- Mạnh mẽ: FastAPI hỗ trợ nhiều tính năng nâng cao và có thể được sử dụng để xây dựng các API phức tạp.

1.2.2 Prisma Client Python

Prisma Client Python là một ORM (Object Relational Mapper) thế hệ mới, cung cấp giao diện lập trình mạnh mẽ và an toàn cho việc tương tác với cơ sở dữ liệu trong các ứng dụng Python. Nó được xây dựng dựa trên Prisma, một nền tảng phát triển cơ sở dữ liệu hiện đại, và mang đến nhiều lợi ích so với các ORM Python truyền thống:

- An toàn kiểu toàn diện: Prisma Client Python sử dụng hệ thống kiểu mạnh mẽ để đảm bảo tính chính xác của dữ liệu và ngăn chặn lỗi thời gian chạy. Nó tự động tạo ra các lớp Python tương ứng với các bảng và mối quan hệ trong lược đồ cơ sở dữ liệu của bạn, giúp bạn viết mã an toàn và dễ đọc hơn.
- Hỗ trợ đồng bộ và bất đồng bộ: Prisma Client Python hỗ trợ cả lập trình đồng bộ và bất đồng bộ, cho phép bạn linh hoạt lựa chọn cách thức truy cập dữ liệu phù hợp nhất với nhu cầu của mình. Điều này đặc biệt hữu ích cho các ứng dụng web hiệu suất cao và các ứng dụng sử dụng các API bất đồng bộ.
- Dễ dàng sử dụng: Prisma Client Python cung cấp API trực quan và dễ sử dụng, giúp bạn dễ dàng học cách sử dụng và tích hợp nó vào dự án của mình. Nó cũng đi kèm với tài liệu hướng dẫn chi tiết và các ví dụ để giúp bạn bắt đầu.
- Hỗ trợ nhiều cơ sở dữ liệu: Prisma Client Python hỗ trợ nhiều loại cơ sở dữ liệu phổ biến, bao gồm PostgreSQL, MySQL, MariaDB, SQLite và SQL Server. Điều này cho phép bạn sử dụng nó với nhiều dự án khác nhau mà không cần phải thay đổi mã của mình.

Prisma Client Python có thể được sử dụng cho nhiều trường hợp khác nhau, bao gồm:

- Tạo, đọc, cập nhật và xóa (CRUD) dữ liệu trong cơ sở dữ liệu
- Thực hiện các truy vấn phức tạp bằng ngôn ngữ truy vấn Prisma
- Tạo và quản lý các mối quan hệ giữa các bảng trong cơ sở dữ liệu
- Thực hiện các giao dịch cơ sở dữ liệu

1.2.3 Uvicorn Python

Uvicorn là một server ASGI (Asynchronous Server Gateway Interface) được thiết kế để tối ưu hóa hiệu suất cho các ứng dụng web Python sử dụng framework asychronous như Starlette và FastAPI. Nó nổi tiếng bởi tốc độ khởi động nhanh, khả năng xử lý nhiều request đồng thời và khả năng tích hợp liền mạch với hệ sinh thái Python hiện đại.

- Nhanh chóng: Uvicorn sử dụng engine dựa trên uvloop, tận dụng tối đa sức mạnh của các CPU hiện đại và hỗ trợ hiệu quả các kết nối mạng tốc độ cao.
- Hiệu quả: Uvicorn sử dụng ít tài nguyên hệ thống, cho phép nó xử lý nhiều request đồng thời với chi phí CPU và bộ nhớ thấp.
- Dễ sử dụng: Uvicorn cung cấp giao diện dòng lệnh đơn giản và dễ hiểu, giúp bạn dễ dàng khởi động và cấu hình server.
- Tích hợp liền mạch: Uvicorn tích hợp hoàn hảo với các framework web Python phổ biến như Starlette và FastAPI, cho phép bạn triển khai ứng dụng web của mình một cách nhanh chóng và dễ dàng.
- Hỗ trợ WebSocket: Uvicorn hỗ trợ giao thức WebSocket, cho phép bạn xây dựng các ứng dụng web thời gian thực tương tác.

1.3 Các công nghệ sử dụng

1.3.1 VSCode

Visual Studio Code (VSCode) là một trình biên tập mã nguồn mở và miễn phí được phát triển bởi Microsoft. Với giao diện người dùng đơn giản nhưng mạnh mẽ, VSCode đã trở thành một trong những công cụ phổ biến nhất trong cộng đồng phát triển phần mềm. Nó hỗ trợ đa nền tảng, có sẵn trên Windows, macOS và Linux, giúp các nhà phát triển dễ dàng chuyển đổi giữa các hệ điều hành mà không gặp phải rắc rối.

1.3.2 Xampp

XAMPP là một bản phân phối miễn phí và mã nguồn mở giúp người dùng dễ dàng tạo ra một môi trường phát triển web trên máy tính cá nhân. Tên gọi XAMPP là viết tắt của các công nghệ chính mà nó tích hợp: "X" đại diện cho hệ điều hành (cross-platform), "Apache", "MySQL", "PHP", và "Perl".

1.3.3 POST MAN

Postman là một ứng dụng desktop được sử dụng rộng rãi để phát triển, thử nghiệm và gỡ lỗi các API (Application Programming Interface). Với giao diện người dùng thân thiện và các tính năng mạnh mẽ, Postman là công cụ không thể thiếu đối với các nhà phát triển API và những người làm việc trong lĩnh vực phát triển phần mềm.

1.3.4 MySQL

MySQL là hệ quản trị cơ sở dữ liệu quan hệ (RDBMS) mã nguồn mở phổ biến nhất trên thế giới. Nó được sử dụng để lưu trữ, truy cập và quản lý dữ liệu cho các ứng dụng web và nhiều loại ứng dụng khác. MySQL hoạt động theo mô hình client-server, với máy chủ lưu trữ cơ sở dữ liệu và máy khách gửi các truy vấn để truy cập dữ liệu.

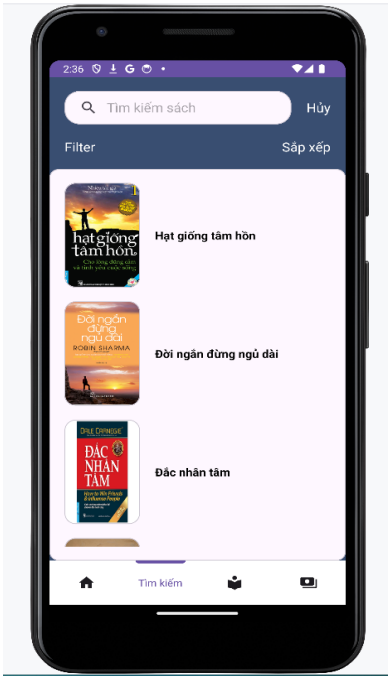
2 TỔNG QUAN GIAO DIỆN

Giao diện được thiết kế bằng Android bao gồm giao diện trang chủ ebook,người dùng,tác giả,audio để nghe sách nói.

Ngoài ra, còn có thanh navbar ở dưới cùng để tương tác cơ bản.

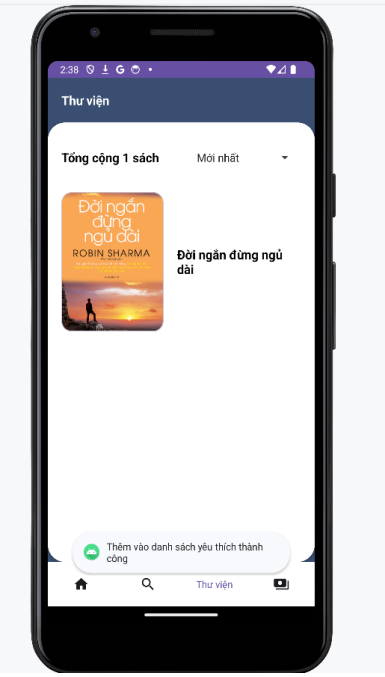
Một số giao diện front end

Hình 1. Trang chủ



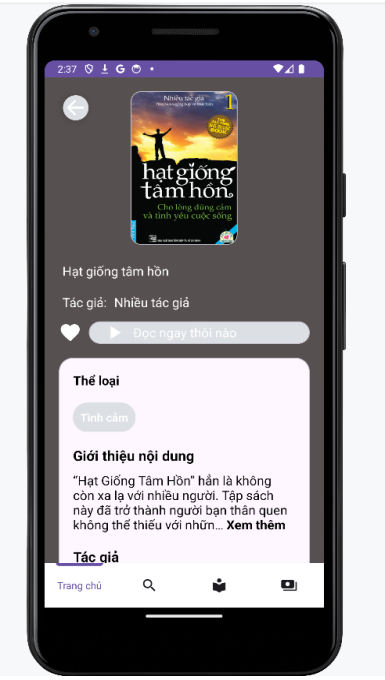
Hình 2. Tìm kiếm

Hình 3. Thư viện khi chưa có sách yêu thích



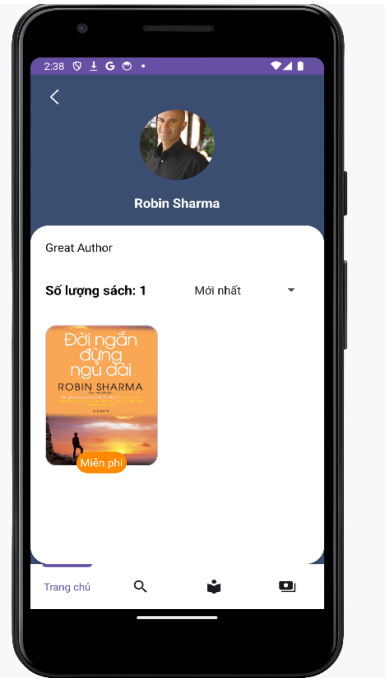
Hình 4. Thư viện khi đã có sách yêu thích

Hình 5.Tài khoản người dùng



Hình 6.Thông tin chi tiết sách

Hình 7.Giao diện sách nói



Hình 8.Tác giả sách

3 TỔNG QUAN CHỨC NĂNG CỦA SÁCH ĐIỆN TỬ EBOOK

3.1 Giới thiệu

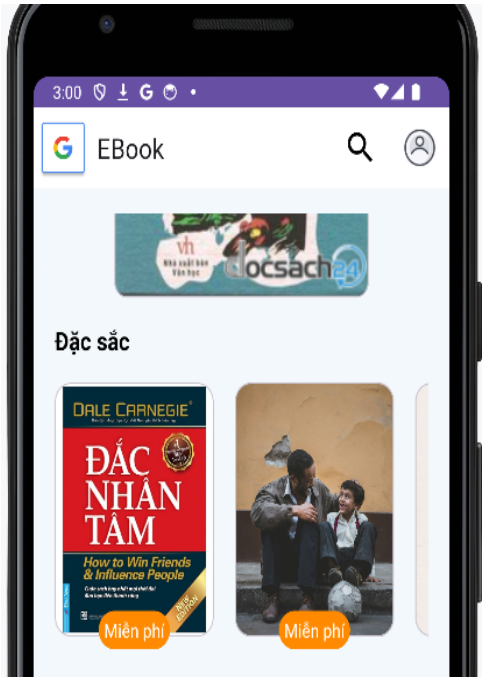
Nhu cầu đọc sách ngày càng cao trong xã hội, đặc biệt là giới trẻ. Tuy nhiên, sách truyền thống thường cồng kềnh, khó mang theo bên mình và tốn kém chi phí. Sách điện tử (Ebook) với những ưu điểm như nhỏ gọn, tiện lợi, giá rẻ và đa dạng về thể loại đã trở thành lựa chọn thay thế phù hợp.Vì thế xây dựng một ứng dụng đọc sách điện tử đa nền tảng là cần thiết để đáp ứng nhu cầu đọc sách của người dùng trên smartphone, máy tính bảng và máy tính cá nhân.

3.2 Các chức năng của sách điện tử Ebook

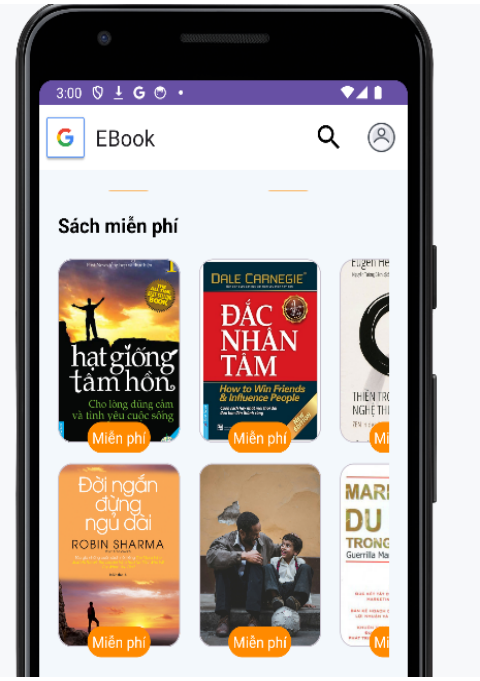
3.2.1 Trang chủ sách nói

Trang chủ sách nói cung cấp cái nhìn tổng quan về các sách nói phổ biến, mới phát hành và được đề xuất. Người dùng có thể dễ dàng tìm kiếm và duyệt qua các danh mục sách nói ngay từ trang chủ.Ngoài ra , người dùng có thể chọn sách muốn nghe ngay tại trang chủ.

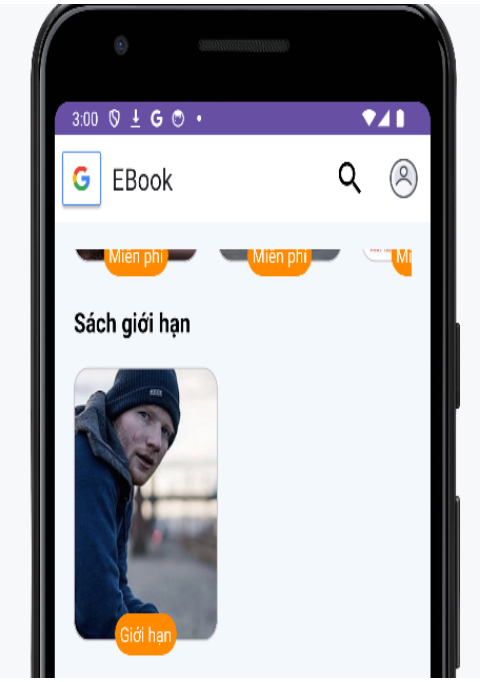
Một số danh mục tại trang chủ.



Hình 9 .Các loại sách đặc sắc



Hình 10 .Các loại sách miễn phí



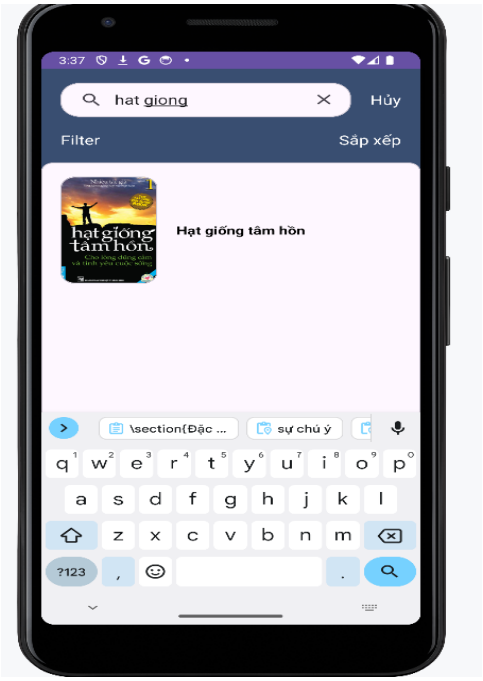
Hình 11 .Các loại sách giới hạn

Hình 12 .Các loại sách khuyên đọc và tác giả sách

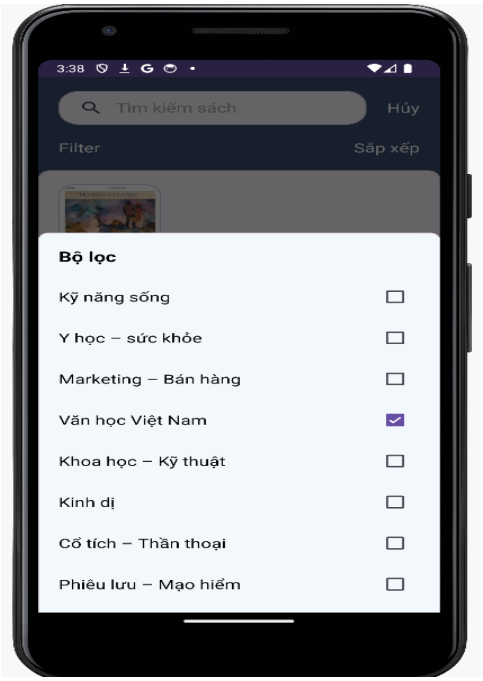
3.2.2 Tìm kiếm sách nói

Giao diện tìm kiếm sách cung cấp nhiều tính năng giúp người dùng dễ dàng tìm kiếm và khám phá sách yêu thích.Với thanh tìm kiếm với gợi ý thông minh giúp thu hẹp kết quả nhanh chóng; Bộ lọc theo thể loại sách; Sắp xếp theo tên sách, năm xuất bản, đánh giá.Sau khi tìm kiếm theo mong muốn thì ứng dụng sẽ hiển thị đầy đủ thông tin sách theo tìm kiếm.

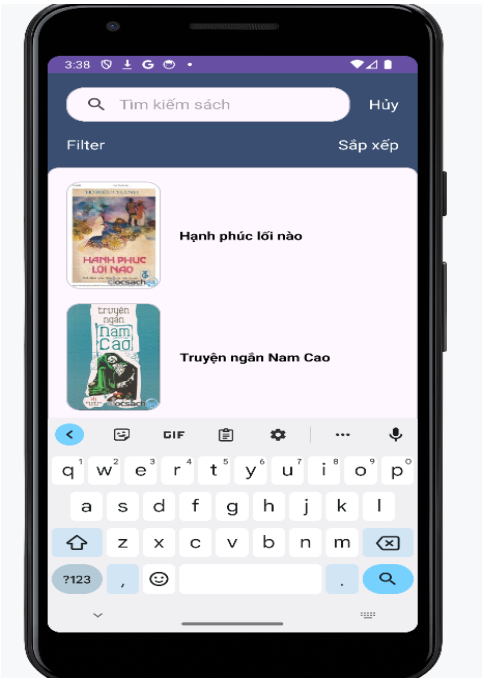
Một số giao diện trong tìm kiếm.



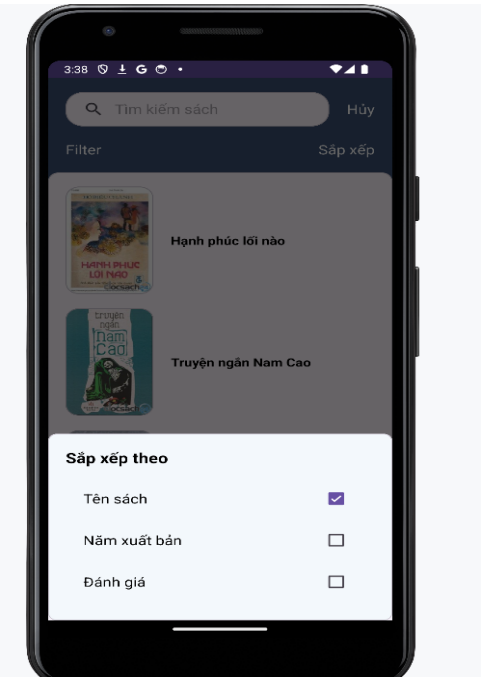
Hình 13 .Tìm kiếm theo tên sách



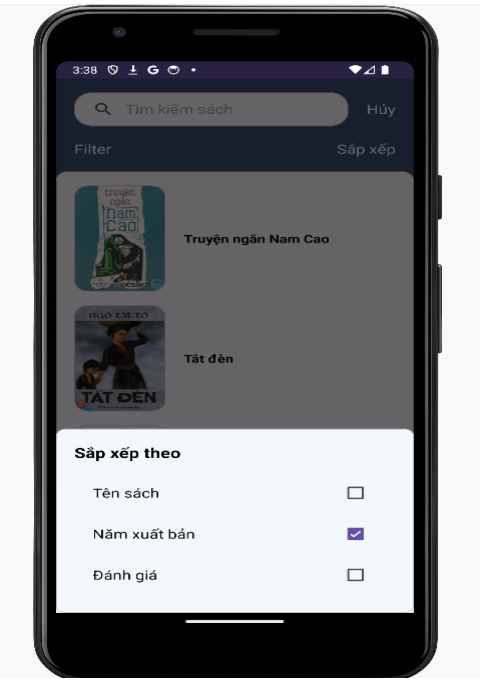
Hình 14 .Tìm kiếm theo bộ lọc



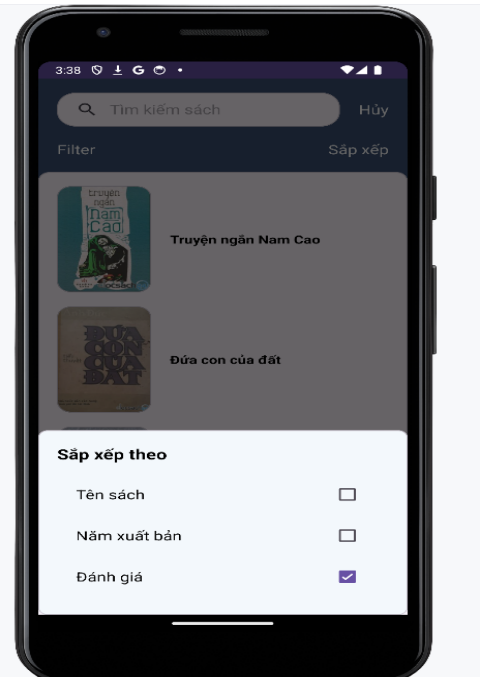
Hình 15 .Sau khi lọc theo thể loại sách



Hình 16 .Sắp xếp sách theo tên sách



Hình 17 .Sắp xếp sách theo năm xuất bản



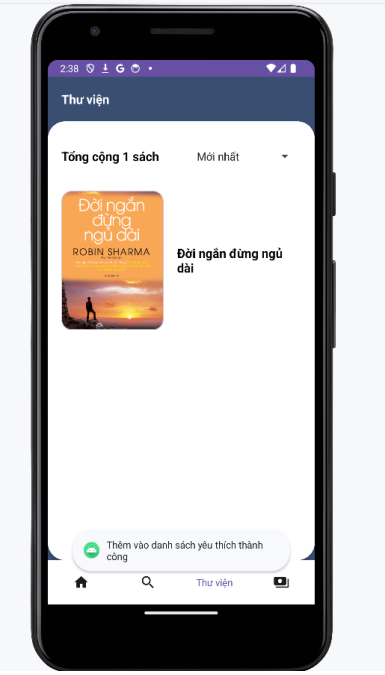
Hình 18 .Sắp xếp sách theo đánh giá

3.2.3 Thư viện yêu thích sách nói

Thư viện sách yêu thích là nơi lưu trữ những cuốn sách mà người dùng đã đánh dấu là yêu thích trên ứng dụng đọc sách điện tử. Thư viện giúp người dùng dễ dàng quản lý và truy cập những cuốn sách yêu thích nhất của mình, đồng thời cung cấp nhiều tính năng hữu ích như sắp xếp, tìm kiếm, đánh dấu trang, ghi chú, chia sẻ, ... Nhờ vậy, người dùng có thể tận hưởng việc đọc sách một cách hiệu quả và thú vị hơn.

Một số giao diện trong thư viện.

Hình 19 . Thư viện khi chưa có sách yêu thích



Hình 20 . Thư viện khi đã có sách yêu thích

3.2.4 Audio sách nói

Audio sách nói là tính năng cho phép người dùng nghe sách thay vì đọc trực tiếp trên ứng dụng, mang đến nhiều ưu điểm như:

- **Tiện lợi**: Nghe mọi lúc mọi nơi, khi làm việc nhà, lái xe hay tập thể dục.
- **Tiết kiệm thời gian**: Tiếp thu thông tin nhanh hơn so với đọc sách.
- **Dễ dàng tiếp cận**: Phù hợp cho người khiếm thị hoặc bận rộn.
- **Giải trí**: Thư giãn và giải trí như được nghe kể chuyện.

Audio sách nói có các tính năng như:

- Điều chỉnh tốc độ phát.
- Chế độ lặp lại

3.2.5 Thông tin chi tiết sách nói

Thông tin chi tiết sách nói cung cấp đầy đủ thông tin về một cuốn sách audio, giúp người dễ dàng hiểu về thông tin sách bao gồm: Tiêu đề, tác giả, thể loại, nhà xuất bản,tóm tắt nội dung của sách

3.2.6 Tác giả sách nói

Tác giả sách nói cung cấp các sách loại sách mà tác giả đã viết

3.2.7 Tài khoản người dùng

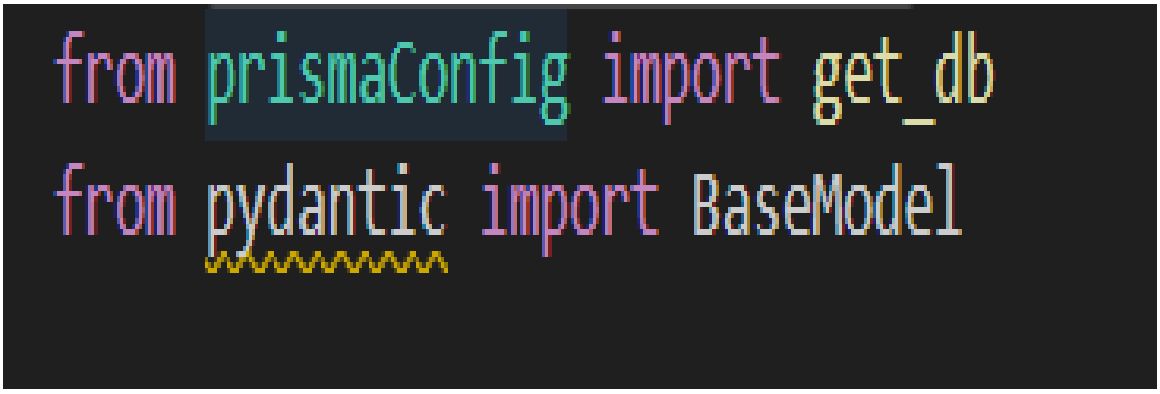
Tài khoản người dùng cung cấp các chức năng sau:

- **Thông tin cá nhân**: Tên, ảnh đại diện.
- **Nâng cấp**: Mua gói hội viên để truy cập kho sách độc quyền nghe audio chất lượng cao.
- **Hỗ trợ**: Liên hệ hỗ trợ, tìm hiểu về Ebook, chia sẻ ứng dụng.
- **Đăng xuất**: Thoát khỏi tài khoản.

3.3 Gọi API dưới backend bằng Python

Nhóm chúng em chọn framework FastAPI của Python để làm backend cho việc gọi API lấy dữ liệu cho ứng dụng.Framework này giúp nhóm có thể xử lý yêu cầu và lưu trữ sách dễ dàng.Ngoài ra , FastAPI còn cung cấp cho nhóm sự phản hồi từ hệ thống và quản lý có thể bảo vệ được dữ liệu của sách một cách an toàn.

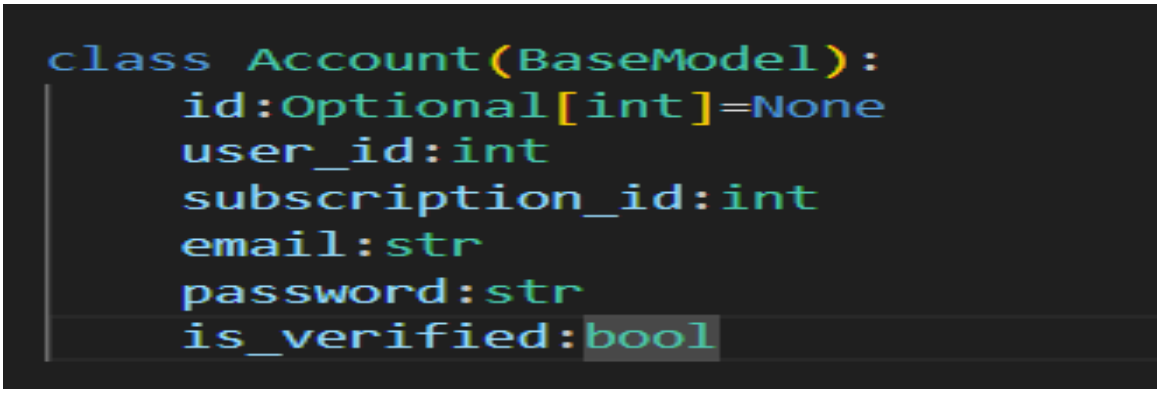
3.3.1 API Tài khoản người dùng



Thư viện sử dụng

get_db từ prismaConfig để kết nối cơ sở dữ liệu.

BaseModel từ Pydantic để định nghĩa mô hình Account.



Lớp Account

Định nghĩa cấu trúc của lớp Account bao gồm id,user_id,subscription_id,email,password và is_verified

```python
async def getAllAccount():
    prisma=await get_db()
    accounts=await prisma.account.find_many()
    return {
        "accounts":accounts
    }
```

Hàm getAllAccount()

Lấy tất cả tài khoản từ cơ sở dữ liệu.

```python
async def findByID(id:int):
    prisma=await get_db()
    account = await prisma.account.find_unique(
        where={
            "id":id
        }
    )
    return {
        "account":account
    }
```

Hàm findByID()

Tìm tài khoản theo ID.

```python
async def deleteByID(id:int):
    prisma=await get_db()
    foundAccount=await prisma.account.find_unique(where={
        "id":id
    })
    if foundAccount is None:
        return {
            "account":None
        }
    else:
        account= await prisma.account.delete(
        where={
            "id":id
        })
        return {
            "account":account
        }
```

Hàm deleteByID()

Xóa tài khoản nếu tồn tại, nếu không trả về None.

```python
async def updateByID(account:Account):
    prisma=await get_db()
    foundAccount=await prisma.account.find_unique(where={
        "id":account.id
    })
    if foundAccount is None:
        return{
            "account":None
        }
    else:
        accountUpdate= await prisma.account.update(where={
        "id":account.id
        },data=dict(account))
    return {
        "account":accountUpdate
    }
```

Hàm updateByID()

Cập nhật thông tin tài khoản nếu tồn tại, nếu không trả về None.

```python
async def findByEmail(email:str):
    prisma=await get_db()
    foundAccount=await prisma.account.find_first(where={
        "email":email
    })
    return {
        "account":foundAccount
    }
```

Hàm findByEmail()

Tìm tài khoản theo Email.

```python
async def signIn(email:str,password:str):
    prisma=await get_db()
    foundAccount=await prisma.account.find_first(where={
        "email":email,
        "password":password
    })
    return {
        "account":foundAccount
    }
```

Hàm signIn()

Kiểm tra đăng nhập bằng email và mật khẩu.

```python
async def create(account:Account):
    prisma=await get_db()
    createdAccount=await prisma.account.create(data=dict(account))
    return {
        "account":createdAccount
    }
```

Hàm create()

Tạo tài khoản mới với thông tin đã cung cấp.

3.3.2 API Người dùng

```python
from prismaConfig import get_db
from pydantic import BaseModel
from typing import Optional
```

Thư viện sử dụng

get_db từ prismaConfig để kết nối cơ sở dữ liệu.
BaseModel từ Pydantic để định nghĩa mô hình User.

Optional từ typing để chỉ định các trường có thể có giá trị None.

```python
class User(BaseModel):
    id:Optional[int]=None
    name:str
    phone_number:Optional[str]=None
    address:Optional[str]=None
```

Lớp User

Định nghĩa cấu trúc của lớp User bao gồm id,name,phone_number và address

```python
async def create(user: User):
    prisma = await get_db()
    createdUser = await prisma.user.create(data=dict(user))
    return {"user": createdUser}
```

Hàm create

Hàm này tạo một người dùng mới với thông tin cung cấp trong đối tượng User.

```python
async def delete(id: int):
    prisma = await get_db()

    user = await prisma.user.find_unique(where={"id": id})
    if user is None:
        return {"user": None}

    deletedUser = await prisma.user.delete(where={"id": id})
    return {"user": deletedUser}
```

Hàm delete

Hàm này xóa một người dùng nếu tồn tại, nếu không trả về None.

```python
async def update(user: User):
    prisma = await get_db()

    foundUser = await prisma.user.find_unique(
        where={"id": user.id}
    )
    if foundUser is None:
        return {"user": None}

    updatedUser = await prisma.user.update(
        where={"id": user.id}, data=dict(user)
    )
    return {"user": updatedUser}
```

Hàm update

Hàm này cập nhật thông tin người dùng nếu tồn tại, nếu không trả về None.

```python
async def findAll(limit: int = None, offset: int = None):
    prisma = await get_db()
    if limit is None:
        users = await prisma.user.find_many()
        return {"users": users}
    else:
        users = await prisma.user.find_many(take=limit, skip=offset)
        length = await prisma.user.count()
        return {"users": users, "length": length}
```

Hàm findAll

Hàm này lấy tất cả người dùng, có thể giới hạn số lượng trả về và sử dụng offset để phân trang. Nếu không có limit, trả về tất cả người dùng.

```python
async def findByID(id: int):
    prisma = await get_db()
    user = await prisma.user.find_unique(where={"id": id})
    return {"user": user}
```

Hàm findByID

Hàm này tìm người dùng theo ID.

```python
async def findByAccountID(id: int):
    prisma = await get_db()
    account = await prisma.account.find_unique(
        where={"id": id}, include={"User": True}
    )
    if account is None:
        return {
            "status":"Loi"
        }
    return {
        "user":account.User
    }
```

Hàm findByAccountID

Hàm này tìm người dùng dựa trên ID tài khoản liên kết. Nếu không tìm thấy tài khoản, trả về trạng thái lỗi.

3.3.3 API Sách nói

```
from prismaConfig import get_db
from pydantic import BaseModel
from typing import Optional

from enum import Enum
```

Thư viện sử dụng

get_db từ prismaConfig để kết nối cơ sở dữ liệu.
BaseModel từ Pydantic để định nghĩa mô hình User.

Optional từ typing để chỉ định các trường có thể có giá trị None.

Enum từ enum để định nghĩa loại sách.

```
class BookType(Enum):
    PREMIUM = "PREMIUM"
    NORMAL = "NORMAL"
```

Định nghĩa BookType

Định nghĩa các loại sách có thể có: PREMIUM và NORMAL.

```
class Book(BaseModel):
    id:Optional[int]=None
    name: str
    description: Optional[str] = None
    rating: int
    progress: float
    published_year: int
    image: Optional[str] = None
    language: str
    book_type: BookType
    src_audio: str
    lyric: str
```

Lớp Book

Định nghĩa cấu trúc của đối tượng Book.

```
async def findAll(limit:int=None,offset:int=None):
    prisma=await get_db()

    if limit is None:
        books=await prisma.book.find_many()
        return {
            "books":books
        }
    else:
        length=await prisma.book.count()
        books=await prisma.book.find_many(
            take=limit,
            skip=offset
        )
        return {
            "books":books,
            "length":length
        }
```

Hàm findAll

Hàm này lấy tất cả sách, có thể giới hạn số lượng trả về và sử dụng offset để phân trang. Nếu không có limit, trả về tất cả sách.

```
async def findByID(id:int):
    prisma=await get_db()
    book = await prisma.book.find_unique(
        where={
            "id":id
        }
    )
    return {
        "book":book
    }
```

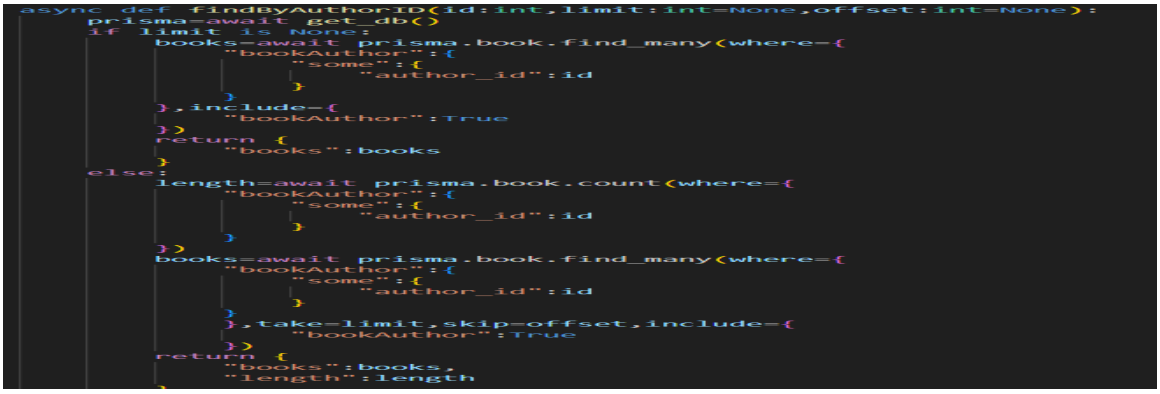
Hàm findByID

Hàm này tìm sách theo ID.

```
async def findByGenreID(id:int,limit:int=None,offset:int=None):
    prisma=await get_db()
    if limit is None:
        books=await prisma.book.find_many(where={
            "bookGenres":{
                "some":{
                    "genre_id":id
                }
            }
        },include={
            "bookGenres":True
        })
        return {
            "books":books
        }
    else:
        length=await prisma.book.count(where={
            "bookGenres":{
                "some":{
                    "genre_id":id
                }
            }
        })
        books=await prisma.book.find_many(where={
            },take=limit,skip=offset,include={
                "bookGenres":True
            })
        return {
            "books":books,
            "length":length
        }
```

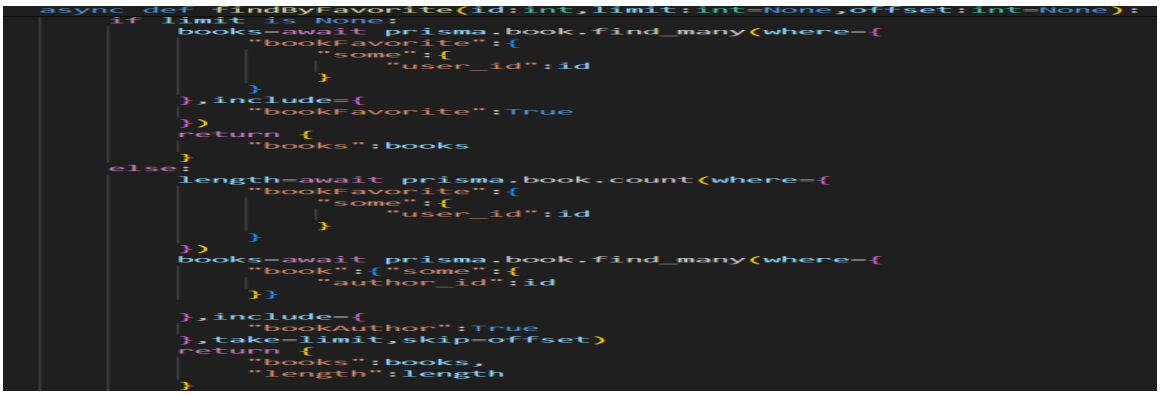
Hàm findByGenreID

Hàm này tìm sách theo ID thể loại, có thể giới hạn số lượng trả về và sử dụng offset để phân trang.



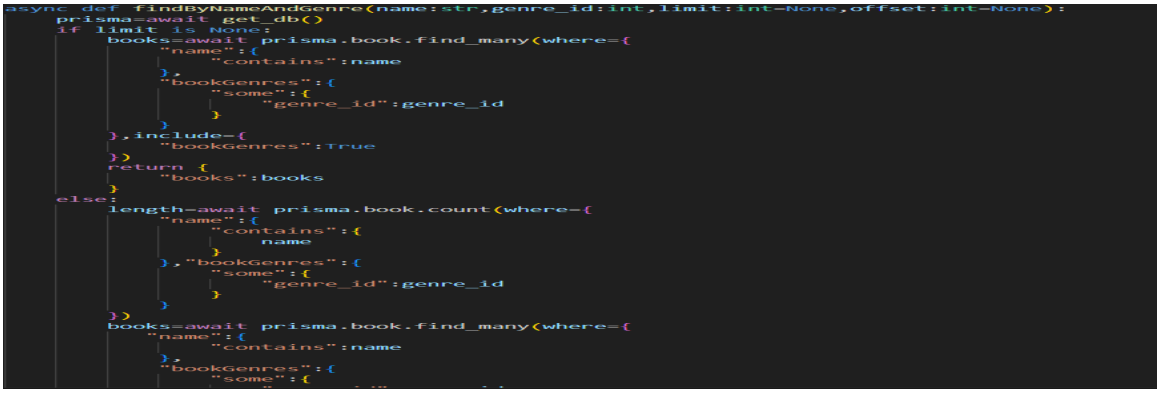
Hàm findByAuthorID

Hàm này tìm sách theo ID tác giả, có thể giới hạn số lượng trả về và sử dụng offset để phân trang.



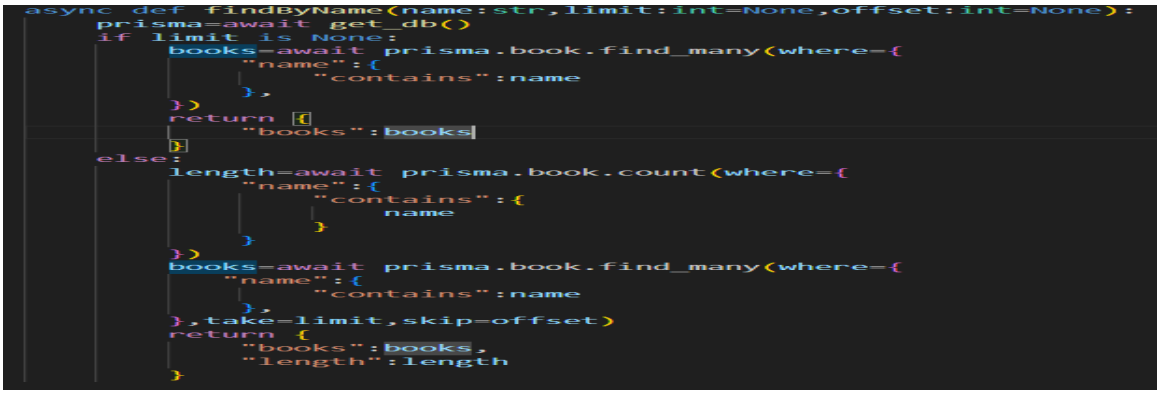
Hàm findByFavorite

Hàm này tìm sách theo ID người yêu thích, có thể giới hạn số lượng trả về và sử dụng offset để phân trang.



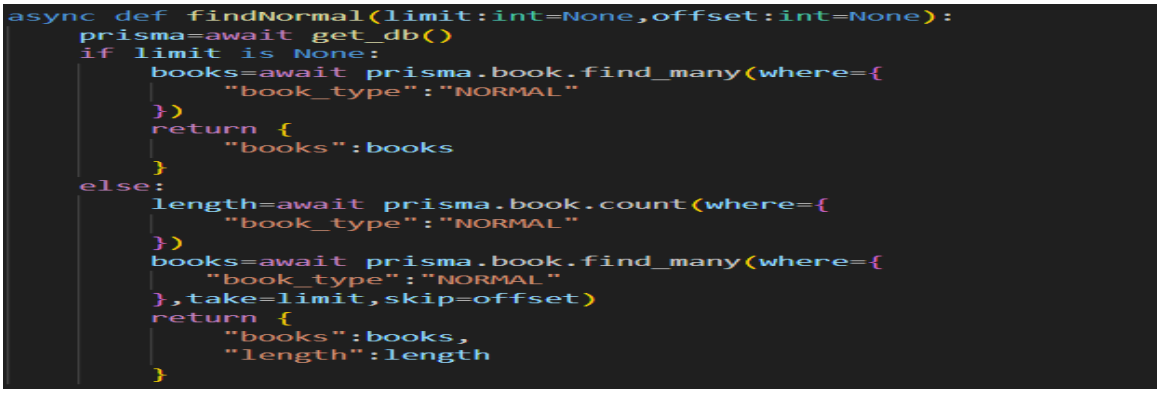
Hàm findByNameAndGenre

Hàm này tìm sách theo tên và ID thể loại, có thể giới hạn số lượng trả về và sử dụng offset để phân trang.



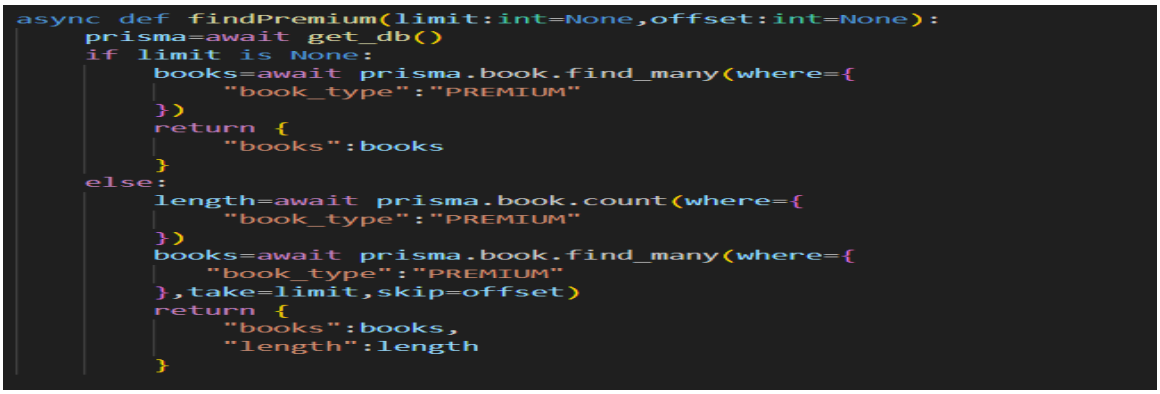
Hàm findByName

Hàm này tìm sách theo tên, có thể giới hạn số lượng trả về và sử dụng offset để phân trang.



Hàm findNormal

Hàm này tìm sách loại NORMAL, có thể giới hạn số lượng trả về và sử dụng offset để phân trang.



Hàm findPremium

Hàm này tìm sách loại PREMIUM, có thể giới hạn số lượng trả về và sử dụng offset để phân trang.

```python
async def findTopRating(limit:int=None,offset:int=None):
    prisma=await get_db()
    if limit is None:
        books=await prisma.book.find_many(order={
            "rating":"desc"
        })
        return {
            "books":books
        }
    else:
        length=await prisma.book.find_many(order={
            "rating":"desc"
        })
        books=await prisma.book.find_many(order={
            "rating":"desc"
        },take=limit,skip=offset)
        return {
            "books":books,
            "length":length
        }
```

Hàm findTopRating

Hàm này tìm sách theo xếp hạng cao nhất, có thể giới hạn số lượng trả về và sử dụng offset để phân trang.

```python
async def delete(id:int):
    prisma=await get_db()
    foundBook=await prisma.book.find_unique(where={
        "id":id
    })
    if foundBook is None:
        return {
            "book":None
        }
    else:
        deletedBook= await prisma.book.delete(
        where={
            "id":id
        })
        return {
            "book":deletedBook
        }
```

Hàm delete

Hàm này xóa sách theo ID nếu tồn tại, nếu không trả về None.

```python
async def update(book:Book):
    prisma=await get_db()
    foundBook=await prisma.book.find_unique(where={
        "id":book.id})
    if foundBook is None:
        return{
            "book":None
        }
    else:
        updatedBook= await prisma.book.update(where={
        "id":book.id
        },data=dict(book))
    return {
        "book":updatedBook
    }
```

Hàm update

Hàm này cập nhật thông tin sách nếu tồn tại, nếu không trả về None.

```python
async def create(book:Book):
    prisma=await get_db()
    createdBook=await prisma.book.create(data=dict(book))
    return {
        "book":createdBook
    }
```

Hàm create

Hàm này tạo một sách mới với thông tin cung cấp trong đối tượng Book.

3.3.4 API Thể loại sách

```python
async def create(genre: Genre):
    prisma = await get_db()
    createdGenre = await prisma.genre.create(data=dict(genre))
    return {"genre": createdGenre}


async def delete(id: int):
    prisma = await get_db()

    genre = await prisma.genre.find_unique(where={"id": id})
    if genre is None:
        return {"genre": None}

    deletedGenre = await prisma.genre.delete(where={"id": id})
    return {"genre": deletedGenre}


async def update(genre: Genre):
    prisma = await get_db()

    foundGenre = await prisma.genre.find_unique(where={"id": genre.id})
    if foundGenre is None:
        return {"genre": None}

    updatedGenre = await prisma.genre.update(where={"id": genre.id}, data=dict(genre))
    return {"genre": updatedGenre}
```

Các hàm CRUD

Hàm create nhận vào một thể loại (genre) và tạo một bản ghi mới trong cơ sở dữ liệu..

Hàm delete xóa một thể loại dựa trên id.Trước tiên kiểm tra xem thể loại có tồn tại hay không, sau đó thực hiện xóa nếu tồn tại.

Hàm update cập nhật thông tin của một thể loại dựa trên id.Trước tiên kiểm tra xem thể loại có tồn tại hay không, sau đó thực hiện cập nhật nếu tồn tại.

```python
async def findAll(limit: int = None, offset: int = None):
    prisma = await get_db()
    if limit is None:
        genres = await prisma.genre.find_many()
        return {"genres": genres}
    else:
        genres = await prisma.genre.find_many(take=limit, skip=offset)
        length = await prisma.genre.count()
        return {"genres": genres, "length": length}
```

Hàm findAll

Tìm và trả về tất cả các thể loại. Có thể giới hạn kết quả trả về bằng cách sử dụng limit và offset.Nếu không có limit, trả về tất cả thể loại. Nếu có, trả về số lượng thể loại theo limit và offset kèm theo tổng số lượng thể loại.

```python
async def findByID(id: int):
    prisma = await get_db()
    genre = await prisma.genre.find_unique(where={
        "id":id
    })
    return {"genre": genre}
```

Hàm findByID

Tìm và trả về một thể loại dựa trên id

```python
async def findByBookID(id:int,limit: int = None, offset: int = None):
    prisma = await get_db()
    if limit is None:
        genres = await prisma.genre.find_many(where={
            "bookGenres":{
                "some":{
                    "book_id":id
                }
            }
        },include={
            "bookGenres":True
        })
        return {"genres": genres}
    else:
        genres = await prisma.genre.find_many(where={
            "bookGenres":{
                "some":{
                    "book_id":id
                }
            }
        },take=limit,skip=offset,include={
            "bookGenres":True
        })
        length = await prisma.genre.count(where={
            "bookGenres":{
                "some":{
                    "book_id":id
                }
            }
        })
        return {"genres": genres, "length": length}
```

Hàm findByBookID

Tìm và trả về các thể loại liên quan đến một cuốn sách dựa trên book_id.Có thể giới hạn kết quả trả về bằng limit và offset. Bao gồm thông tin liên kết với bảng bookGenres.

3.3.5 API Tác giả sách nói

```
from prismaConfig import get_db
from pydantic import BaseModel
from typing import Optional
from fastapi import Request
```

Thư viện sử dụng

get_db từ prismaConfig để kết nối cơ sở dữ liệu.
BaseModel từ Pydantic để định nghĩa mô hình User.

Optional từ typing để chỉ định các trường có thể có giá trị None..

Request từ fastapi, mặc dù nó không được sử dụng trong đoạn code này.

```
class Author(BaseModel):
    id:Optional[int]=None
    name: str
    image: Optional[str] = None
    description: Optional[str] = None
```

Định nghĩa lớp Author

Author kế thừa từ BaseModel của Pydantic và định nghĩa các trường dữ liệu cho một tác giả. description là tùy chọn (Optional).

```python
async def create(author: Author):
    prisma = await get_db()
    createdAuthor = await prisma.author.create(data=dict(author))
    return {"author": createdAuthor}

async def delete(id: int):
    prisma = await get_db()

    author = await prisma.author.find_unique(where={"id": id})
    if author is None:
        return {"author": None}

    deletedAuthor = await prisma.author.delete(where={"id": id})
    return {"author": deletedAuthor}

async def update(author: Author):
    prisma = await get_db()

    foundAuthor = await prisma.author.find_unique(where={"id": author.id})
    if foundAuthor is None:
        return {"author": None}

    updatedAuthor = await prisma.author.update(
        where={"id": author.id}, data=dict(author)
    )
    return {"author": updatedAuthor}
```

Các hàm CRUD

Hàm create(author: Author): Hàm này tạo một tác giả mới trong cơ sở dữ liệu.Kết nối tới cơ sở dữ liệu.Tạo tác giả mới bằng cách chuyển đổi đối tượng Author thành dictionary.Trả về thông tin tác giả vừa được tạo.

Hàm delete(id: int): Hàm này xóa một tác giả dựa trên id cung cấp.Kết nối tới cơ sở dữ liệu.Tìm tác giả với id tương ứng.Nếu không tìm thấy, trả về None.Nếu tìm thấy, xóa tác giả và trả về thông tin tác giả vừa bị xóa.

Hàm update(author: Author): Hàm này cập nhật thông tin của một tác giả.Kết nối tới cơ sở dữ liệu.Tìm tác giả với id tương ứng. Nếu không tìm thấy, trả về None.Nếu tìm thấy, cập nhật thông tin và trả về thông tin tác giả vừa được cập nhật.

```python
async def findAll(limit: int, offset: int):
    prisma = await get_db()
    if limit is None:
        authors = await prisma.author.find_many()
        return {"authors": authors}
    else:
        authors = await prisma.author.find_many(take=limit, skip=offset)
        length = await prisma.author.count()
        return {"authors": authors, "length": length}
```

Hàm findAll

Hàm này tìm tất cả các tác giả với phân trang.Kết nối tới cơ sở dữ liệu.Nếu không cung cấp limit, trả về tất cả tác giả.Nếu cung cấp limit, trả về các tác giả theo giới hạn và độ lệch (offset) cùng với tổng số lượng tác giả.

```python
async def findByID(id: int):
    prisma = await get_db()
    author = await prisma.author.find_unique(where={"id": id})
    return {"author": author}
```

Hàm findByID

Hàm này tìm tác giả dựa trên id.Kết nối tới cơ sở dữ liệu.Trả về thông tin tác giả tương ứng với id.

```python
async def findByBookID(id: int):
    prisma = await get_db()
    bookAuthors = await prisma.bookauthor.find_many(
        where={"book_id": id}, include={"author": True}
    )
    authors = [bookAuthor.author for bookAuthor in bookAuthors]

    return {"authors": authors}
```

Hàm findBookByID

Hàm này tìm các tác giả của một quyển sách dựa trên book_id.Kết nối tới cơ sở dữ liệu.Tìm tất cả các liên kết bookauthor liên quan tới quyển sách và bao gồm thông tin tác giả.Trả về danh sách các tác giả liên quan tới quyển sách đó.

```python
async def findOneByBookID(id: int):
    prisma = await get_db()
    book = await prisma.book.find_unique(where={"id": id}, include={"bookAuthor": True})
    if book.bookAuthor.length == 0:
        return {"author": "Không rõ"}
    else:
        author = await prisma.author.find_first(
            where={"id": book.bookAuthor[0].author_id}
        )
        if author is None:
            return {"author": "Không rõ"}
        else:
            return {"author": author}
```

Hàm findOneBookByID

Hàm này tìm tác giả đầu tiên của một quyển sách dựa trên book_id.Kết nối tới cơ sở dữ liệu.Tìm thông tin quyển sách và bao gồm thông tin bookAuthor.Nếu không có tác giả, trả về "Không rõ". Nếu có, tìm tác giả đầu tiên và trả về thông tin của tác giả đó.

Tài liệu

[1] FastAPI "**link: https://fastapi.tiangolo.com/tutorial/**", *FastAPI*,lần truy cập cuối : 16/5.

[2] Python and Android "**link: https://realpython.com/lessons/package-android-app/**", *Python and Android*,lần truy cập cuối : 16/5.

[3] Prisma Client Python "**link: https://prisma-client-py.readthedocs.io/en/stable/**", *Prisma Client Python*,lần truy cập cuối : 16/5.

[4] Uvicorn "**link: https://www.uvicorn.org/**", *Uvicorn*,lần truy cập cuối : 16/5.