MÔ TẢ KỸ THUẬT VÀ NGHIỆP VỤ

Môn học: Lập trình trên thiết bị di động.

Nhóm: 13

Đề tài: Tạo ứng dụng đọc truyện trực tuyến.

I. Mô tả nghiệp vụ

Mô tả nghiệp vụ ứng dụng đọc truyện chữ gồm các chức năng sau:

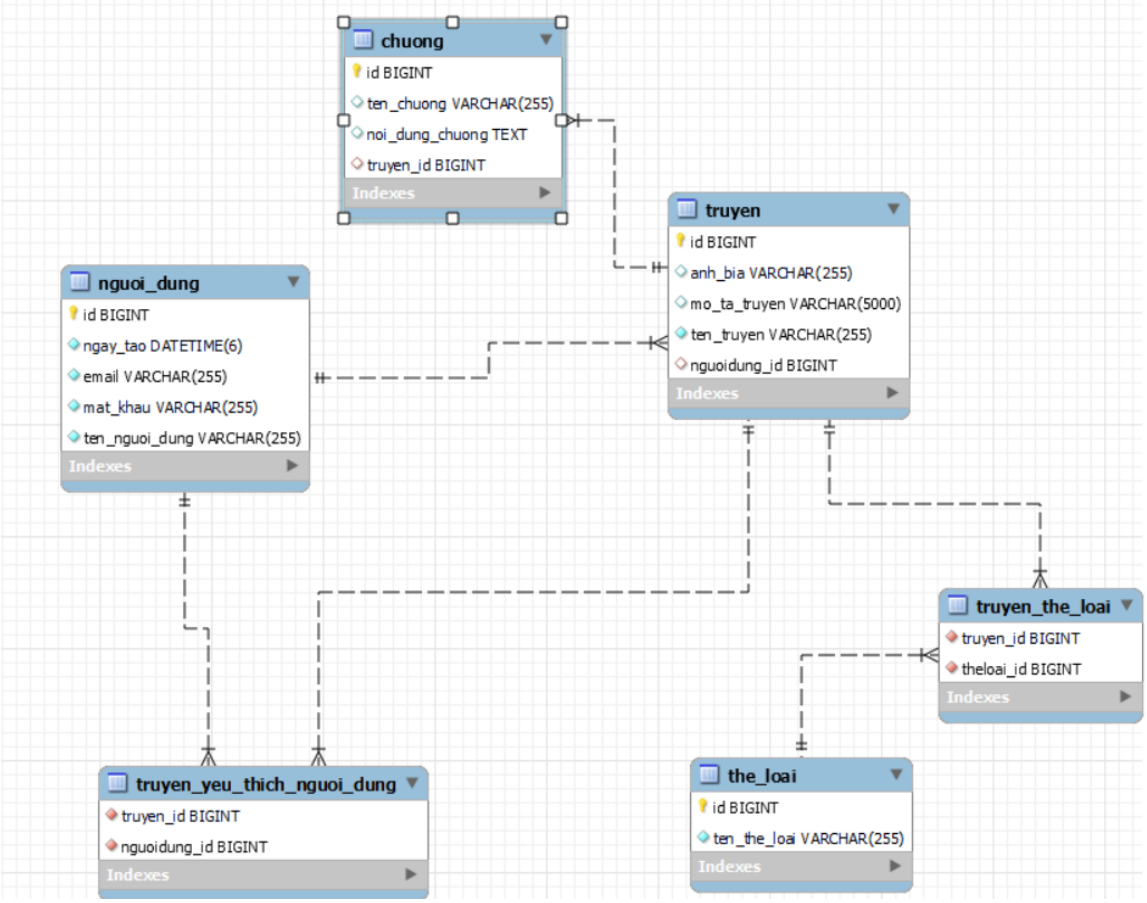
1. **Hiển thị các truyện theo thể loại và tác giả:**
   - Người dùng có thể xem danh sách truyện được sắp xếp theo các thể loại khác nhau.
   - Người dùng có thể tìm kiếm truyện theo tác giả.
2. **Hiển thị trang giới thiệu:**
   - Ứng dụng cung cấp trang giới thiệu giới thiệu về ứng dụng, mô tả chức năng chính và hướng dẫn sử dụng.
   - Trang giới thiệu có thể chứa thông tin về nhà phát triển, liên hệ, và các thông tin hữu ích khác.
3. **Thêm truyện vào mục yêu thích:**
   - Người dùng có thể thêm/xóa các truyện yêu thích trong danh sách yêu thích của họ.
   - Danh sách yêu thích sẽ cho phép người dùng dễ dàng truy cập và đọc lại các truyện mà họ quan tâm.
4. **Người dùng thêm truyện của mình vào:**
   - Người dùng có thể đăng ký tài khoản và đăng truyện của riêng họ lên ứng dụng.
   - Khi đăng truyện, người dùng cần cung cấp các thông tin như tên truyện, mô tả, thể loại, tác giả, nội dung chương, ảnh bìa,...
   - Truyện sau khi được đăng sẽ được hiển thị trong danh sách truyện của người dùng và có thể được tìm thấy bởi các người dùng khác.
5. **Đọc truyện:**
   - Người dùng có thể chọn một truyện và đọc các chương của truyện.
   - Trang đọc truyện cung cấp các chức năng như chọn chương, lật trang,...
6. **Chức năng khác:**
   - Đăng nhập và đăng ký tài khoản để quản lý truyện yêu thích và truyện do người dùng đăng.
   - Giao diện thân thiện, dễ sử dụng và tương thích trên nhiều thiết bị android.

II. Mô tả kỹ thuật

1. Cơ sở dữ liệu

Sử dụng hệ quản trị cơ sở dữ liệu MySQL, bao gồm các đối tượng sau:

- ‘Book’ và ‘Genre’ có mối quan hệ nhiều - nhiều (Many-to-Many): Một truyện (Book) có thể có nhiều thể loại (Genre), và một thể loại (Genre) cũng có thể có nhiều truyện (Book). Mối quan hệ này được biểu diễn thông qua bảng trung gian ‘truyen_the_loai’, mà cặp khóa chính của nó bao gồm các khóa chính của bảng ‘Book’ và bảng ‘Genre’.
- ‘Book’ và ‘User’ có mối quan hệ nhiều - một (Many-to-One): Một truyện (Book) được viết bởi một người dùng (User), nhưng một người dùng có thể viết nhiều truyện. Mối quan hệ này được biểu diễn bởi trường ‘User’ trong đối tượng ‘Book’, đại diện cho một đối tượng ‘User’.
- ‘User’ và ‘Book’ có mối quan hệ một - nhiều (One-to-Many): Mỗi người dùng (User) có thể đăng nhiều truyện (Book) trong danh sách truyện của họ, nhưng mỗi truyện chỉ thuộc về một người dùng. Mối quan hệ này được biểu diễn bởi trường ‘listBooks’ trong đối tượng ‘User’, đại diện cho danh sách các đối tượng ‘Book’ mà người dùng đó đã đăng.
- ‘User’ và ‘Genre’ không có mối quan hệ trực tiếp, nhưng thông qua mối quan hệ nhiều - nhiều giữa ‘Book’ và ‘Genre’, người dùng có thể truy cập các truyện thuộc mỗi thể loại.
- ‘Book’ và ‘User’ có mối quan hệ nhiều - nhiều (Many-to-Many): Một truyện (Book) có thể được nhiều người dùng (User) thêm vào mục yêu thích, và một người dùng cũng có thể thêm nhiều truyện vào mục yêu thích của họ. Mối quan hệ này được biểu diễn thông qua bảng trung gian ‘truyen_yeu_thich_nguoi_dung’, mà cặp khóa chính của nó bao gồm các khóa chính của bảng ‘Book’ và bảng ‘User’.
- ‘Book’ và ‘Chapter’ có mối quan hệ One-To-Many: Một truyện (Book) có thể có nhiều chương (Chapter), nhưng một chương chỉ thuộc một truyện. Mối quan hệ này được biểu diễn bởi trường ‘listChapter’ trong đối tượng ‘Book’, đại diện cho danh sách các đối tượng ‘Chapter’ thuộc một truyện.



2. Xử lý các chức năng

Công nghệ lựa chọn xử lý back-end:

- Ngôn ngữ lập trình: Java
- Framework: Spring Boot
- Cơ sở dữ liệu: MySQL
- Dịch vụ lưu trữ ảnh: Cloudinary
- Dịch vụ hiển thị ảnh bên client: Glide
- Thư viện hỗ trợ mã nguồn: Lombok
- Jakarta Persistence API (JPA): Để quản lý đối tượng và tương tác với cơ sở dữ liệu.

2.1.*Xây dựng API gồm các model: 'Book', 'User', 'Genre'*

- 'Book': Đại diện cho thông tin của một truyện chữ. Có các thuộc tính như nameBook (tên truyện), describe (mô tả truyện), content (nội dung của các chương), coverImg (link ảnh bìa truyện), listGenre (danh sách thể loại của truyện), user (người dùng đã đăng truyện), listUserPressingLove (danh sách người dùng thích truyện này).
- 'User': Đại diện cho thông tin của người dùng. Có các thuộc tính như userName (tên người dùng), password (mật khẩu), email, createdAt

(thời gian tạo tài khoản), listFavoriteBook (danh sách truyện mà người dùng thích), listBooks (danh sách truyện mà người dùng đã đăng).

- ‘Genre’: Đại diện cho thông tin của thể loại truyện. Có thuộc tính nameOfGenre (tên thể loại).
- Mỗi một đối tượng có một class Controller, một class Repository, một interface Service và một class ServiceImplement riêng để xử lý chức năng.
- Controllers:
- BookController: Xử lý các yêu cầu liên quan đến sách. Bao gồm tạo, cập nhật, xóa sách và các thao tác khác.
- GenreController: Xử lý các yêu cầu liên quan đến thể loại sách. Bao gồm tạo, cập nhật, xóa thể loại và liên kết với sách.
- UserController: Xử lý các yêu cầu liên quan đến người dùng. Bao gồm tạo, cập nhật, xóa người dùng, và quản lý sách yêu thích.
- Repositories:
- BookRepository: Giao tiếp với cơ sở dữ liệu liên quan đến sách.
- GenreRepository: Giao tiếp với cơ sở dữ liệu liên quan đến thể loại.
- UserRepository: Giao tiếp với cơ sở dữ liệu liên quan đến người dùng.
- Services:
- BookService: Chứa các phương thức xử lý logic liên quan đến sách. Bao gồm lấy danh sách sách, lấy sách theo ID, tạo sách mới, cập nhật sách và xóa sách.
- GenreService: Chứa các phương thức xử lý logic liên quan đến thể loại. Bao gồm lấy danh sách thể loại, lấy thể loại theo ID, tạo thể loại mới, cập nhật thể loại và xóa thể loại.
- UserService: Chứa các phương thức xử lý logic liên quan đến người dùng. Bao gồm lấy danh sách người dùng, lấy người dùng theo ID, tạo người dùng mới, cập nhật người dùng và xóa người dùng. Cũng bao gồm các phương thức quản lý sách yêu thích của người dùng.
- CoverImgService: Xử lý việc tải lên hình ảnh bìa sách lên Cloudinary.

2.2. Chức năng hiển thị truyện theo thể loại và tác giả

Back-end:

- Đầu tiên, đối với việc lấy truyện theo thể loại, ta phải cho mối quan hệ giữa đối tượng ‘Book’ và ‘Genre’ phải là Many - to - Many, ngoài ra cần lưu ý sử dụng @JsonIgnore để chỉ định những trường hoặc phương thức không nên được chuyển đổi thành JSON khi thực hiện quá trình serialization bởi nhiều mục đích (bảo mật, tránh gọi nhiều lần do ảnh hưởng bởi mối quan hệ giữa các đối tượng,...)
  Trong model Book:
  @ManyToMany

```
@JoinTable(
    name = "truyen_the_loai",
    joinColumns = @JoinColumn(name = "truyen_id"),
    inverseJoinColumns = @JoinColumn(name = "theloai_id")
)
private List<Genre> listGenre;
```

Trong model Genre:

```
@JsonIgnore
@ManyToMany(mappedBy = "listGenre")
private List<Book> listBook;
```

- Trong đoạn code trên @JsonIgnore có công dụng không cho trường listBooks được chuyển đổi thành JSON khi đối tượng 'Genre' được serialized hoặc trả về dưới dạng JSON thông qua các endpoint API. Điều này tránh cho trường này bị gọi nhiều lần do có quan hệ Many-to-Many giữa hai đối tượng 'Book' và 'Genre'.
- Đối với việc lấy truyện theo tác giả, ta xử lý như sau: Người đọc tìm kiếm truyện theo người dùng đăng, lúc này tác giả sẽ là đối tượng 'User', việc xử lý sẽ giống với tìm truyện theo thể loại.
- Người dùng ứng dụng gửi yêu cầu từ client lấy danh sách truyện theo thể loại hoặc tác giả tới các API tương ứng trong GenreController hoặc UserController.

Trong GenreController

```
@GetMapping("{id}/truyen/")
public List<Book> getBookByGenre(@PathVariable(value = "id")
Long id) {
  return genreService.getBookByGenre(id);
}
```

Trong UserController

```
@GetMapping("{id}/truyen/")
public List<Book> getBookByUser(@PathVariable(value = "id")
Long id) {
  return userService.getBookByUser(id);
}
```

- Controller gọi các phương thức tương ứng từ GenreService hoặc UserService để truy vấn dữ liệu từ cơ sở dữ liệu thông qua GenreRepository hoặc UserRepository.
  Trong GenreRepository
  ```
  @Query("SELECT t FROM Book t JOIN t.listGenre tl ON tl.id = ?1")
  List<Book> getBookByGenre(Long id);
  ```
  Trong UserRepository
  ```
  @Query("SELECT t FROM Book t JOIN t.user tg ON tg.id = ?1")
  List<Book> getBookByUser(Long id);
  ```

- GenreRepository hoặc UserRepository thực hiện các truy vấn liên quan đến dữ liệu truyện và thể loại/tác giả từ cơ sở dữ liệu.
  Trong UserServiceImpl
  ```
  @Override
  public List<Book> getBookByUser(Long id) {
    return userRepository.getBookByUser(id);
  }
  ```
  Trong GenreServiceImpl
  ```
  @Override
  public List<Book> getBookByGenre(Long id) {
    return genreRepository.getBookByGenre(id);
  }
  ```
- Dữ liệu trả về được đóng gói trong đối tượng JSON và trả về cho người dùng thông qua Controller.

Front-end:

- Để dễ kết nối với api phía server, ở bên phía client cũng tạo các Model Book, Chapter, Genre, User. Đồng thời tạo thêm interface APIService, chứa thư viện Retrofit, hỗ trợ cho việc giao tiếp với api phía server, và các phương thức như Get, Post, Delete dữ liệu. Phía client sẽ dùng phương thức enqueue để gửi yêu cầu đến server, và nhận phản hồi từ server về cho người dùng.
- 
```
public interface APIService {
    Gson gson = new GsonBuilder().create();
    APIService API_SERVICE = new Retrofit.Builder()
            .baseUrl("http://192.168.1.12:8080/api/")
            .addConverterFactory(GsonConverterFactory.create(gson))
            .build().create(APIService.class);
```

```
thanh huyen
@GET("truyen")
Call<List<Book>> getListBook();
1 usage  BNgan3C33
@GET("truyen/{id}")
Call<Book> getBookByID(@Path("id") long id);

2 usages  BNgan3C33
@GET("theloai/{genreId}/truyen/")
Call<List<Book>> getBookByGenre(@Path("genreId") long genreId);

4 usages  BNgan3C33 *
@GET("theloai")
Call<List<Genre>> getAllGenre();

2 usages  BNgan3C33
@GET("truyen/{bookID}/chuong/")
Call<List<Chapter>> getAllChaptersByBook(@Path("bookID") long bookID);

BNgan3C33
@GET("nguoidung")
Call<List<User>> getAllUser();

2 usages  BNgan3C33
@GET("nguoidung/{id}/truyenyeuthich/")
Call<List<Book>> getListFavoriteBookByUser(@Path("id") long userID);
2 usages  BNgan3C33
@GET("nguoidung/{id}/truyen/")
Call<List<Book>> getBookByUser(@Path("id") long userID);
```

```
private void getListBook() {
    BNgan3C33
    APIService.API_SERVICE.getListBook().enqueue(new Callback<List<Book>>() {
        BNgan3C33
        @Override
        public void onResponse(Call<List<Book>> call, Response<List<Book>> response) {
            if(response.isSuccessful()){
                listBook.addAll(response.body());
                BookAdapter bookAdapter = new BookAdapter(listBook);
                rcvBook.setAdapter(bookAdapter);
            }
        }

        BNgan3C33
        @Override
        public void onFailure(Call<List<Book>> call, Throwable t) {
            Log.d( tag: "Error", t.getMessage());
        }
    });
}
```

Danh sách book, genre, chapter sẽ được hiển thị bằng RecycleView, xây dựng Adapter để xử lý các đối tượng rồi đổ ra các RecycleView hiển thị cho người dùng thấy. Dữ liệu được lấy bằng các phương thức GET được xây dựng ở api phía server.

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                         Bundle savedInstanceState) {
    View view = inflater.inflate(R.layout.fragment_home, container,  attachToRoot: false);

    rcvBook = view.findViewById(R.id.rcv_book);
    rcvBook.setLayoutManager(new LinearLayoutManager(this.getContext(), LinearLayoutManager.VERTICAL,  reverseLayout: false));
    listBook = new ArrayList<>();
    getListBook();
    return view;
}

1 usage  BNgan3C33
private void getListBook() {
    BNgan3C33
    APIService.API_SERVICE.getListBook().enqueue(new Callback<List<Book>>() {
        BNgan3C33
        @Override
        public void onResponse(Call<List<Book>> call, Response<List<Book>> response) {
            if(response.isSuccessful()){
                listBook.addAll(response.body());
                BookAdapter bookAdapter = new BookAdapter(listBook);
                rcvBook.setAdapter(bookAdapter);
            }
        }

        BNgan3C33
        @Override
        public void onFailure(Call<List<Book>> call, Throwable t) {
            Log.d( tag: "Error", t.getMessage());
        }
    });
```

2.3. Chức năng thêm truyện vào mục yêu thích

Back-end:

- Ở chức năng này, ta cũng cần hai đối tượng 'Book' và 'User' có mối quan hệ Many-to-Many với nhau và phải chú ý @JsonIgnore lại các trường cho hợp lý.

  Trong model Book

```
@JsonIgnore
@ManyToMany
@JoinTable(
    name = "truyen_yeu_thich_nguoi_dung",
    joinColumns = @JoinColumn(name = "truyen_id"),
    inverseJoinColumns = @JoinColumn(name = "nguoidung_id")
)
private List<User> listUserPressingLove;
```

  :

  Trong model User:

```
@JsonIgnore
@ManyToMany(mappedBy = "listUserPressingLove")
private List<Book> listFavoriteBook;
```

- Trong trường hợp này, ta chọn che cả hai bên để bảo mật riêng tư cho người dùng.
- Người dùng ứng dụng gửi yêu cầu thêm truyện vào mục yêu thích hoặc xóa truyện khỏi mục yêu thích tới API trong UserController.

  Thêm truyện vào mục yêu thích:

```
@PostMapping("/{nguoidung_id}/yeuthich/{truyen_id}")
public ResponseEntity<Void> addBookInFavorites(
    @PathVariable(value = "nguoidung_id") Long userId,
    @PathVariable(value = "truyen_id") Long bookId) {
  try {
    userService.addBookInFavorites(userId, bookId);
    return ResponseEntity.ok().build();
  } catch (Exception e) {
    return ResponseEntity.badRequest().build();
  }
}
```

  Xóa truyện khỏi mục yêu thích:

```
@DeleteMapping("{userId}/xoayeuthich/{bookId}")
public ResponseEntity<String>
removeBookFromFavorites(@PathVariable(value = "bookId") Long
bookId,
```

```
                                    @PathVariable(value = "userId") Long userId) {
  try {
    userService.removeBookFromFavorites(bookId, userId);
    return ResponseEntity.ok("Đã xóa truyện khỏi mục yêu thích.");
  } catch (Exception e) {
    return ResponseEntity.badRequest().body("Xóa truyện khỏi mục yêu thích thất bại.");
  }
}
```

- Controller gọi phương thức addBookInFavorites/ removeBookFromFavorites từ UserService để xử lý.
- Trong UserService, sách và người dùng tương ứng được truy vấn từ cơ sở dữ liệu thông qua UserRepository.

Xử lý thêm truyện vào mục yêu thích:

```
public Book addBookInFavorites(Long userId, Long bookId) throws Exception {
    User user = userRepository.findById(userId)
        .orElseThrow(() -> new Exception("User not found: " + userId));

    Book book = bookRepository.findById(bookId)
        .orElseThrow(() -> new Exception("Book not found: " + bookId));
    if(!user.getListFavoriteBook().contains(book)){
//        System.out.println("[DEBUG] - User: " + user);
//        System.out.println("[DEBUG] - Book: " + book);
      user.getListFavoriteBook().add(book);
      book.getListUserPressingLove().add(user);
      userRepository.save(user);
    }
    return book;
  }
```

Xử lý xóa truyện khỏi mục yêu thích

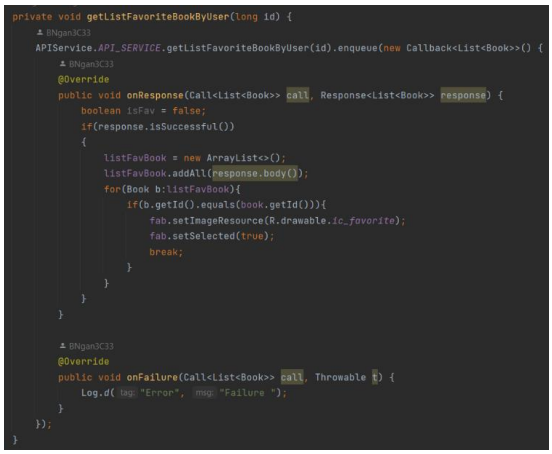
```
public Book removeBookFromFavorites(Long bookId, Long userId) throws Exception {
  User user = userRepository.findById(userId)
```

```
            .orElseThrow(() -> new Exception("User not found: " +
        userId));

        Book book = bookRepository.findById(bookId)
            .orElseThrow(() -> new Exception("Book not found: " +
        bookId));

        if (user.getListFavoriteBook().contains(book)) {
            System.out.println("[DEBUG] - User: " + user);
            System.out.println("[DEBUG] - Book: " + book);
            user.getListFavoriteBook().remove(book);
            book.getListUserPressingLove().remove(user);
            userRepository.save(user);
        } else{
            throw new Exception();
        }
        return book;
    }
```

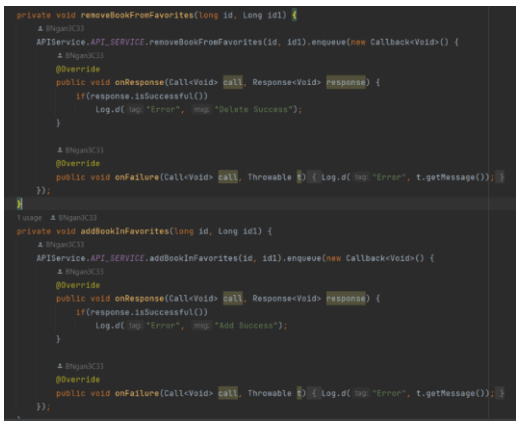
- Sau khi thêm, thông tin được cập nhật và lưu trong cơ sở dữ liệu.

Front-end:

- Khi người dùng chọn một truyện, phương thức getListFavoriteBookByUser sẽ trả về danh sách các truyện yêu thích của người dùng, rồi so sánh với truyện đang hiển thị, nếu đã nằm trong danh sách rồi thì sẽ hiển thị button yêu thích, ngược lại sẽ hiển thị button chưa yêu thích, và cho phép thêm vào danh sách yêu thích bằng cách chọn vào button, khi đó phương thức POST addBookInFavorites sẽ gửi yêu cầu đến cho server. Người dùng còn có thể xóa truyện khỏi danh sách bằng cách chọn tiếp vào button, phương thức Delete removeBookFromFavorites sẽ gửi yêu cầu xóa đến cho server.



```
private void getListFavoriteBookByUser(long id) {
    BNgan3C33
    APIService.API_SERVICE.getListFavoriteBookByUser(id).enqueue(new Callback<List<Book>>() {
        BNgan3C33
        @Override
        public void onResponse(Call<List<Book>> call, Response<List<Book>> response) {
            boolean isFav = false;
            if(response.isSuccessful())
            {
                listFavBook = new ArrayList<>();
                listFavBook.addAll(response.body());
                for(Book b:listFavBook){
                    if(b.getId().equals(book.getId())){
                        fab.setImageResource(R.drawable.ic_favorite);
                        fab.setSelected(true);
                        break;
                    }
                }
            }
        }

        BNgan3C33
        @Override
        public void onFailure(Call<List<Book>> call, Throwable t) {
            Log.d( tag: "Error",  msg: "Failure ");
        }
    });
}
```



```
private void removeBookFromFavorites(long id, Long id1) {
    BNgan3C33
    APIService.API_SERVICE.removeBookFromFavorites(id, id1).enqueue(new Callback<Void>() {
        BNgan3C33
        @Override
        public void onResponse(Call<Void> call, Response<Void> response) {
            if(response.isSuccessful())
                Log.d( tag: "Error",  msg: "Delete Success");
        }

        BNgan3C33
        @Override
        public void onFailure(Call<Void> call, Throwable t) { Log.d( tag: "Error", t.getMessage()); }
    });
}
1 usage   BNgan3C33
private void addBookInFavorites(long id, Long id1) {
    BNgan3C33
    APIService.API_SERVICE.addBookInFavorites(id, id1).enqueue(new Callback<Void>() {
        BNgan3C33
        @Override
        public void onResponse(Call<Void> call, Response<Void> response) {
            if(response.isSuccessful())
                Log.d( tag: "Error",  msg: "Add Success");
        }

        BNgan3C33
        @Override
        public void onFailure(Call<Void> call, Throwable t) { Log.d( tag: "Error", t.getMessage()); }
    });
```

2.4. Chức năng đăng truyện của người dùng

Back-end:

- Chức năng đăng truyện yêu cầu giữa 'User' và 'Book' phải có thêm một mối quan hệ One-to-Many. Để tránh gọi đệ quy ta cần @JsonIgnore các trường hợp lý.
- Người dùng ứng dụng gửi yêu cầu đăng truyện tới API trong BookController cùng với thông tin của cuốn sách.

```
@PostMapping("{id}/dang_truyen")
public ResponseEntity<?> postBook(@Valid @RequestBody Book book, @PathVariable(value = "id") Long userId) {
  Book bookMoi = bookService.postBook(book, userId);
  return ResponseEntity.status(HttpStatus.CREATED).body(bookMoi);
}
```

- Controller gọi phương thức postBook từ BookService để tạo một cuốn sách mới.

```
@Override
public Book postBook(Book book, Long userId) {
  System.out.println("[DEBUG] - START POST BOOK");
  User author = userRepository.findById(userId).get();
  book.setUser(author);
  Book createdBook = createBook(book);
  return createdBook;
}
```

  - Trong BookService, cuốn sách mới được tạo và lưu vào cơ sở dữ liệu thông qua BookRepository.

```
@Override
public Book createBook(Book book) {
  return bookRepository.save(book);
}
```

- Sau khi cuốn sách được đăng thành công, thông tin về cuốn sách được trả về cho người dùng.
- Ngoài ra, khi cho phép người dùng đăng tải thêm ảnh bìa minh họa, cần thêm một CoverImgService để xử lý tình huống này. Trong dự án này dịch vụ lưu trữ hình ảnh được sử dụng là Cloudinary.

```
@Configuration
public class CloudinaryConfig {

  @Bean
  public Cloudinary getCloudinary() {
```

```
        return new Cloudinary("cloudinary://717436577626688:kH1K6CFHeeOCP0DPZVn9yfurYvo@dlbp1jaju");
    }
}
```

- CoverImgService nên được xử lý như sau:

```
@Override
public Book postCoverImg(MultipartFile file, Long id) throws Exception {
    try {
        String url = coverImgService.uploadImage(file);
        Book book = bookRepository.findById(id).get();
        book.setCoverImg(url);
        bookRepository.save(book);
        return book;
    } catch (IOException e) {
        throw new Exception("Fail to upload image");
    }
}
```

Front-end:

Người dùng chọn vào biểu tượng đăng truyện trên thanh menu dưới, hệ thống kiểm tra xem người dùng đã đăng truyện nào chưa bằng cách dùng phương thức GET để lấy danh sách truyện của người dùng đã đăng, nếu tìm thấy thì sẽ hiển thị danh sách truyện đó, nếu không thì sẽ để trống.

```
private void getBookByUser(long id) {
    APIService.API_SERVICE.getBookByUser(id).enqueue(new Callback<List<Book>>() {
        @Override
        public void onResponse(Call<List<Book>> call, Response<List<Book>> response) {
            if(response.isSuccessful()){
                books.addAll(response.body());
                if(books.isEmpty()){
                    rcvPost.setVisibility(View.GONE);
                }
                else{
                    tv.setText("Các truyện đã đăng");
                    BookAdapter bookAdapter = new BookAdapter(books);
                    rcvPost.setLayoutManager(new LinearLayoutManager(getContext(), LinearLayoutManager.VERTICAL, reverseLayout: false));
                    rcvPost.setAdapter(bookAdapter);
                }
            }
        }
        @Override
        public void onFailure(Call<List<Book>> call, Throwable t) {
            Log.d( tag: "Error", t.getMessage());
        }
    });
}
```

Nếu có truyện đã đăng, người dùng có thể nhấn giữ vào truyện để hiển thị một dialog, hiển thị các chức năng cho người dùng lựa chọn muốn thực hiện như: thêm chương, sửa, xóa truyện, hoặc là đăng truyện mới.

```
holder.itemView.setOnLongClickListener(new View.OnLongClickListener() {
    BNgan3C33 *
    @Override
    public boolean onLongClick(View view) {
        AlertDialog.Builder builder = new AlertDialog.Builder(view.getContext());
        builder.setView(R.layout.dialog);
        AlertDialog dialog = builder.show();

        TextView update = dialog.findViewById(R.id.update);
        TextView delete = dialog.findViewById(R.id.delete);
        TextView add = dialog.findViewById(R.id.add);
        new *
```

Khi đăng truyện, người dùng sẽ nhập thông tin của truyện như tên truyện, mô tả truyện vào các EditText, chọn thể loại từ danh sách các thể loại (được lấy từ api khi dùng phương thức getAllGenre() rồi đưa vào spinner), và chọn ảnh từ thiết bị để làm ảnh bìa của truyện.

```
private void getListGenre() {
    BNgan3C33 *
    APIService.API_SERVICE.getAllGenre().enqueue(new Callback<List<Genre>>() {
    BNgan3C33 *
    @Override
    public void onResponse(Call<List<Genre>> call, Response<List<Genre>> response) {
        if (response.isSuccessful()) {
            genres.addAll(response.body());

            if (genres.isEmpty()) {
                Log.d( tag: "Error",  msg: "khong co listChapter");
            } else {
                SpinnerGenres adapter = new SpinnerGenres(getApplicationContext(), R.layout.spinner_genre, genres);
                spGenre.setAdapter(adapter);
                  spGenre.setSelection(0);
                new *
                spGenre.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
                    new *
                    @Override
                    public void onItemSelected(AdapterView<?> adapterView, View view, int i, long l) {
                        Genre g = (Genre) spGenre.getItemAtPosition(i);
                        mGenres.add(g);
                        edtGenre.setText(g.getNameOfGenre());
                    }
                    new *
                    @Override
                    public void onNothingSelected(AdapterView<?> adapterView) {

                    }
                });
```

Khi chọn vào khung ảnh để chọn ảnh, ứng dụng sẽ kiểm tra xem đã có quyền truy cập vào bộ nhớ của thiết bị. Nếu chưa thì thực hiện xin cấp quyền truy cập, rồi mở thư viện để người dùng chọn ảnh, sau đó chuyển thành file để có thể upload ảnh lên cloudinary.

```java
//kiem tra quyen va xin cap quyen
1 usage  BNgan3C33 *
private void onClickRequestPermission() {
    if (Build.VERSION.SDK_INT < Build.VERSION_CODES.M) {
        openGalery();
        return;}
    if (checkSelfPermission(Manifest.permission.READ_EXTERNAL_STORAGE) == PackageManager.PERMISSION_GRANTED) {
        openGalery();
    } else {
        String[] permission = {Manifest.permission.READ_EXTERNAL_STORAGE};
        requestPermissions(permission, REQUEST_CODE);
    }
}
//xac nhan ket qua xin cap quyen và mo thu vien
16 usages  BNgan3C33 *
@Override
public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[] grantResults) {
    super.onRequestPermissionsResult(requestCode, permissions, grantResults);
    if (requestCode == REQUEST_CODE) {
        if (grantResults.length > 0 && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
            openGalery();
        }
    }
}
//mo thu vien
3 usages  BNgan3C33
private void openGalery() {
    Intent intent = new Intent();
    intent.setType("image/*");
    intent.setAction(Intent.ACTION_GET_CONTENT);
    activityResultLauncher.launch(Intent.createChooser(intent,  title: "Select Image"));
```

```java
private final ActivityResultLauncher<Intent> activityResultLauncher = registerForActivityResult(new ActivityResultContracts.StartActivityForResu
    BNgan3C33
    @Override
    public void onActivityResult(ActivityResult result) {
        if (result.getResultCode() == Activity.RESULT_OK) {
            Intent data = result.getData();
            if (data == null) {
                return;
            }
            Uri uri = data.getData();
            mUri = uri;
            Log.d( tag: "Uri", mUri.getPath());
            try {
                Bitmap bitmap = MediaStore.Images.Media.getBitmap(getContentResolver(), uri);
                imgCoverBook.setImageBitmap(bitmap);
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
});
```

Khi nút Đăng truyện được chọn thì phương thức POST postBook cũng được thực hiện để đăng truyện lên hệ thống. Nếu đăng thành công thì phương thức POST postCoverImg cũng sẽ được thực hiện. Khi đó ảnh bìa của truyện cũng được tải lên hệ thống. Sau đó chuyển sang activity để thực hiện việc đăng chương

```java
btnSubmit.setOnClickListener(new View.OnClickListener() {
    new *
    @Override
    public void onClick(View view) {
        strName = edtNameBook.getText().toString();
        strDescribe = edtDescribe.getText().toString();
        book = new Book(strName, mGenres, strDescribe, user);
        postBook(book, user.getId());
    }
});
```

```
private void postBook(Book book, long id) {
    BNgan3C33 *
    APIService.API_SERVICE.postBook(book, id).enqueue(new Callback<Book>() {
        BNgan3C33 *
        @Override
        public void onResponse(Call<Book> call, Response<Book> response) {
            if (response.isSuccessful()) {
                Log.d( tag: "Error",  msg: "Post book Success");
                Book b = response.body();

                String url = RealPathUtil.getRealPath( context: PostBook.this, mUri);
                Log.d( tag: "Error", url);
                File file = new File(url);

                Log.d( tag: "File", file.toString());
                RequestBody requestBody = RequestBody.create(MediaType.parse( string: "image/*"), file);
                MultipartBody.Part body = MultipartBody.Part.createFormData( name: "file", file.getName(), requestBody);
                postCoverImg(body, b.getId());

                Intent intent = new Intent( packageContext: PostBook.this, PostChapter.class);
                intent.putExtra( name: "NewBook", b);
                startActivity(intent);
            } else {
                Log.d( tag: "Error",  msg: "Post book failure");
            }
        }
```

```
private void postCoverImg(MultipartBody.Part body, Long id) {
    BNgan3C33 *
    APIService.API_SERVICE.postCoverImg(body, id).enqueue(new Callback<Book>() {
        BNgan3C33 *
        @Override
        public void onResponse(Call<Book> call, Response<Book> response) {
            if (response.isSuccessful()) {
                Log.d( tag: "Error",  msg: "Success");
            } else {
                Log.d( tag: "Error",  msg: "Failure " + response);
            }
        }

        BNgan3C33
        @Override
        public void onFailure(Call<Book> call, Throwable t) {
            Log.d( tag: "Error",  msg: "Failure " + t.getMessage());
        }
    });
}
```

2.5. Các chức năng khác

Ngoài các chức năng chính được yêu cầu, chúng ta còn có các chức năng phụ cần được xử lý api như đăng nhập/đăng ký, hiển thị chapter truyện, cập nhật truyện, xóa truyện.

Đăng ký/đăng nhập

UserController:

```
@GetMapping("/find-by-email/{email}")
public ResponseEntity<?> findUserByEmail(@PathVariable(value = "email") String email) {
  User user = userService.findUserByEmail(email);
```

```
    if (user == null) {
      return ResponseEntity.notFound().build();
    }
    return ResponseEntity.ok(user);
  }

  @PostMapping(path = "/login")
  public ResponseEntity<?> loginAccount(@RequestBody User user)
  throws Exception {
    try {
      System.out.println("[DEBUG] - " + user);
      return
  ResponseEntity.ok().body(userService.loginAccount(user));
    } catch (Exception e){
      return ResponseEntity.badRequest().body(e.getMessage());
    }
  }

  UserRepository
  User findByEmail(String email);

  @Query("SELECT a FROM User a WHERE a.email = ?1 and
  a.password = ?2")
  Optional<User> loginAccount(String email, String password);

  UserService/UserServiceImpl
  public User findUserByEmail(String email) {
    return userRepository.findByEmail(email);
  }

  @Override
  public User loginAccount(User account) throws Exception {
    try {
      if (userRepository.findByEmail(account.getEmail()) != null) {
        System.out.println("[DEBUG] - " + account);
        Optional<User>                    accountEntity                    =
userRepository.loginAccount(account.getEmail(), account.getPassword());
        if (accountEntity.isPresent()) {
          return account;
        } else {
```

```
                throw new Exception("Wrong password!");
            }
        } else {
            throw new Exception("Email not found!");
        }
    } catch (Exception e) {
        throw new Exception(e.getMessage());
    }
}
```

Front-end:

Khi mở ứng dụng, một view hiện lên để người dùng nhập thông tin để đăng nhập, người dùng sẽ nhập email và password vào EditText. Khi nút đăng nhập được bấm, sự kiện setOnClickListener sẽ được thực hiện. Phương thức loginAccount được gọi, gửi email và password về cho hệ thống để hệ thống xử lý, sau đó trả về đầy đủ thông tin của người dùng đó.

```
btnLogin.setOnClickListener(new View.OnClickListener() {
    BNgan3C33 +1
    @Override
    public void onClick(View view) {
        strEmail = edtEmail.getText().toString();
        strPassword = edtPassWord.getText().toString();
        User us = new User(strPassword, strEmail, userName: "");
        Log.d( tag: "User", us.toString());

        loginAccount(us);
    }
```

Nếu đăng nhập thành công, Toast sẽ hiện lên để thông báo cho người dùng biết rồi dùng Intent để chuyển sang màn hình chính của ứng dụng. Đồng thời dùng SharePreferences để lưu thông tin cho ứng dụng biết. Thông tin ở đây bao gồm một biến isLogged, và một đối tượng user để cho ứng dụng biết, khi người dùng mở ứng dụng lần nữa thì không cần đăng nhập.

```
private void loginAccount(User us) {
    BNgan3C33
    APIService.API_SERVICE.loginAccount(us).enqueue(new Callback<User>() {
        BNgan3C33
        @Override
        public void onResponse(Call<User> call, Response<User> response) {
            if (response.isSuccessful()) {
                Toast.makeText(getView().getContext(), text: "Đăng nhập thành công", Toast.LENGTH_SHORT).show();
                User newUser = response.body();
                editor.putBoolean( s: "isLogged", b: true);
                Gson gson = new Gson();
                String userJson = gson.toJson(newUser);
                editor.putString( s: "user", userJson);
                editor.apply();
                Intent intent = new Intent(getActivity(), MainActivity.class);
                startActivity(intent);
            } else {
                Log.d( tag: "Error", response.toString());
            }
        }

        BNgan3C33
        @Override
        public void onFailure(Call<User> call, Throwable t) {
            Log.d( tag: "Error", msg: "Khong thuc hien");
        }
```

Khi người dùng bấm vào nút đăng ký, view hiện ra để người dùng nhập thông tin gồm email, tên đăng nhập, password.

Khi bấm nút đăng ký, ứng dụng sẽ gửi một phương thức findUserByEmail, để tìm xem đã có email nào trong cơ sở dữ liệu chưa, nếu có rồi thì báo lỗi cho người dùng.

```
private void findUserByEmail(String strEmail) {
    BNgan3C33 +1
    APIService.API_SERVICE.findUserByEmail(strEmail).enqueue(new Callback<User>() {
        BNgan3C33
        @Override
        public void onResponse(Call<User> call, Response<User> response) {
            if (response.isSuccessful()) {
                Log.d( tag: "Error", response.toString());
                user = response.body();
                Log.d( tag: "Error", user.toString());
                if (user != null) {
                    tvMessage.setVisibility(View.VISIBLE);
                    tvMessage.setText("Email đã tồn tại");
                }
            } else {
                User newUser = new User(strPassword, strEmail, strUserName);
                createUser(newUser);
                Toast.makeText(getView().getContext(), text: "Đăng ký thành công", Toast.LENGTH_SHORT).show();
                FragmentTransaction fragmentTransaction = getChildFragmentManager().beginTransaction();
                fragmentTransaction.replace(R.id.frm_register, new LoginFragment()).commit();
            }
        }
        BNgan3C33 +1
        @Override
        public void onFailure(Call<User> call, Throwable t) {
```

Nếu chưa thì sẽ gửi dữ liệu để lên hệ thống để tạo một người dùng mới bằng phương thức createUser(). Nếu đăng ký thành công, ứng dụng sẽ chuyển lại trang đăng nhập để người dùng đăng nhập vào ứng dụng.

Xử lý list chapter

Cần xử lý tương tự model Book. Ở đây chúng ta chỉ quan tâm đến cách tạo ra một chapter như thế nào.

ChapterController

```
@PostMapping("{bookId}")
public ResponseEntity<?> createChapter(@PathVariable(value = "bookId") Long bookId, @Valid @RequestBody Chapter chapter) {
    return ResponseEntity.ok(chapterService.createChapter(bookId,chapter));
}
```

Truy vấn trong BookRepository

```
@Query("SELECT c FROM Chapter c JOIN c.book b ON b.id = ?1")
List<Chapter> getAllChaptersByBook(Long id);
```

ChapterService/ChapterServiceImpl

```
@Override
public Chapter createChapter(Long bookId, Chapter chapter) {
    Book book = bookRepository.findById(bookId).get();
    chapter.setBook(book);
    chapterRepository.save(chapter);
    if(book.getListChapter() == null){
        book.setListChapter(new ArrayList<>());
    }
    book.getListChapter().add(chapter);
    bookRepository.save(book);
    return chapter;
}
```

Cập nhật truyện/chapter

ChapterController

```
@PutMapping("{bookId}/update-chapter/{chapterId}")
public ResponseEntity<Chapter> updateChapter(@PathVariable(value = "bookId") Long bookId, @PathVariable(value = "chapterId") Long chapterId,
                                             @Valid @RequestBody Chapter chapterDetails) throws Exception {
    return (ResponseEntity.ok(chapterService.updateChapter(bookId, chapterId, chapterDetails)));
}
```

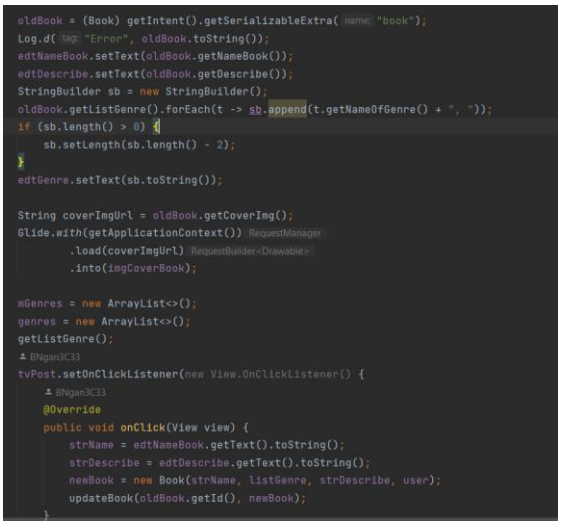
ChapterService/ChapterServiceImpl

```
@Override
public Chapter updateChapter(Long bookId, Long chapterId, Chapter chapterDetails) throws Exception {
    Chapter chapter = chapterRepository.findById(chapterId).orElseThrow(()-> new Exception("Chương này không tồn tại: " + chapterId));
        // So sánh và cập nhật nội dung chương nếu có thay đổi
    if (!Objects.equals(chapter.getContent(), chapterDetails.getContent())) {
        chapter.setContent(chapterDetails.getContent());
    }
    return chapterRepository.save(chapter);
}
```

Tương tự với truyện cũng làm theo cách này.

Front-end:

Khi muốn cập nhật truyện hoặc chương, người dùng chọn sẽ vào trang để viết truyện, nhấn giữ vào đối tượng. Nếu là truyện, sẽ có 3 lựa chọn là thêm chương, sửa truyện và xóa truyện. Nếu là chương, sẽ có hai lựa chọn là sửa và xóa chương. Khi chọn sửa, các thông tin cũ sẽ được cập nhật lại và gắn sẵn trên EditText, người dùng sẽ cập nhật. Để xác nhận cập nhật, người dùng sẽ bấm Đăng, khi đó phương thức PUT updateBook(đối với truyện) hoặc updateChapter(đối với chương). Sau đó quay lại trang chứa danh sách các truyện/chương đã đăng.



```
oldBook = (Book) getIntent().getSerializableExtra( name: "book");
Log.d( tag: "Error", oldBook.toString());
edtNameBook.setText(oldBook.getNameBook());
edtDescribe.setText(oldBook.getDescribe());
StringBuilder sb = new StringBuilder();
oldBook.getListGenre().forEach(t -> sb.append(t.getNameOfGenre() + ", "));
if (sb.length() > 0) {
    sb.setLength(sb.length() - 2);
}
edtGenre.setText(sb.toString());

String coverImgUrl = oldBook.getCoverImg();
Glide.with(getApplicationContext()) RequestManager
        .load(coverImgUrl) RequestBuilder<Drawable>
        .into(imgCoverBook);

mGenres = new ArrayList<>();
genres = new ArrayList<>();
getListGenre();
BNgan3C33
tvPost.setOnClickListener(new View.OnClickListener() {
    BNgan3C33
    @Override
    public void onClick(View view) {
        strName = edtNameBook.getText().toString();
        strDescribe = edtDescribe.getText().toString();
        newBook = new Book(strName, listGenre, strDescribe, user);
        updateBook(oldBook.getId(), newBook);
    }
```

```
private void updateBook(Long id, Book newBook) {
    BNgan3C33
    APIService.API_SERVICE.updateBook(id, newBook).enqueue(new Callback<Book>() {
        BNgan3C33
        @Override
        public void onResponse(Call<Book> call, Response<Book> response) {
            if (response.isSuccessful()) {
                Log.d( tag: "Error",  msg: "Update book success");
                Book newBook = response.body();
                postCoverImg(body, newBook.getId());
            } else {
                Log.d( tag: "Error",  msg: "Update book failure");
            }
        }

        1 usage  BNgan3C33
        private void postCoverImg(MultipartBody.Part body, Long id) {
            BNgan3C33
            APIService.API_SERVICE.postCoverImg(body, id).enqueue(new Callback<Book>() {
                BNgan3C33
                @Override
                public void onResponse(Call<Book> call, Response<Book> response) {
                    if (response.isSuccessful()) {
                        Log.d( tag: "Error",  msg: "Update image success");
                    } else {
                        Log.d( tag: "Error",  msg: "Update image failure");
```

Xóa truyện/chapter

ChapterController

```
@DeleteMapping("{bookId}/delete-chapter/{chapterId}")
public Map<String, Boolean> deleteChapter(@PathVariable(value = "bookId") Long bookId, @PathVariable(value = "chapterId") Long chapterId)
        throws Exception {
    return chapterService.deleteChapter(bookId, chapterId);
}
```

ChapterService/ChapterServiceImpl

```
@Override
public Map<String, Boolean> deleteChapter(Long bookId,Long chapterId) throws Exception {
    Chapter chapter = chapterRepository.findById(chapterId)
            .orElseThrow(() -> new Exception("Chương này không tồn tại: " + chapterId));

    chapterRepository.delete(chapter);
    Map<String, Boolean> response = new HashMap<>();
    response.put("deleted", Boolean.TRUE);
    return response;
```

}

Front-end:

Khi người dùng nhấn giữ vào truyện/chương trong trang viết truyện, sẽ hiện dialog, bấm chọn vào xóa. Phương thức DELETE deleteChapter, deleteBook sẽ được gửi lên hệ thống.

```
delete.setOnClickListener(new View.OnClickListener() {
    BNgan3C33
    @Override
    public void onClick(View view) {
        Log.d( tag: "Error", book.toString());
        deleteBook(book.getId());
        dialog.dismiss();
    }
```

2.6. Trang giới thiệu truyện

Front-end:

Trang sẽ chứa các thông tin của truyện như: Tên truyện, tên người đăng, tên tác giả, thể loại, số chương của truyện. Danh sách truyện được lấy từ phương thức GET getListBook(), sự kiện khi người dùng bấm vào từng truyện hoặc từng chương sẽ được xử lý trong Adapter, trong onBindViewHolder().

```
holder.itemView.setOnClickListener(new View.OnClickListener() {
    BNgan3C33
    @Override
    public void onClick(View view) {
        Intent intent = new Intent(view.getContext(), BookDetail.class);
        intent.putExtra( name: "book", book);
        view.getContext().startActivity(intent);
    }
});
```

Khi đó, thông tin của đối tượng được đóng gói lại bằng rồi chuyển sang trang thông tin riêng của nó bằng đối tượng Intent, Serializable.

```
book = (Book) getIntent().getSerializableExtra( name: "book");
Log.d( tag: "BookInfo", book.toString());
```

Khi chọn vào truyện, sẽ đưa đến trang thông tin của truyện, các thuộc tính như tên, tác giả, thể loại, mô tả, danh sách chương đều được gửi theo.

```
StringBuilder sb = new StringBuilder();
book.getListGenre().forEach(t -> sb.append(t.getNameOfGenre()+", "));
if (sb.length() > 0) {
    sb.setLength(sb.length() - 2);
}
  book.setId(id);
tvBookName.setText(book.getNameBook());
tvGenre.setText("Thể loại: "+ sb);
tvAuthor.setText("Tác giả: "+book.getUser().getUserName());
tvDescribe.setText(book.getDescribe());
Glide.with(getApplicationContext()).load(book.getCoverImg()).into(imgCover);
Glide.with(getApplicationContext()).load(book.getCoverImg()).into(imgBigCover);

listFavBook = new ArrayList<>();
getListFavoriteBookByUser(user.getId());
```

Khi chọn vào chương, sẽ đưa đến trang đọc truyện, để xem nội dung của chương truyện đó. Ảnh bìa của truyện dùng đối tượng ImageFilterView, đối tượng này có thể phóng to, giảm độ rõ của ảnh. Ngoài ra, đối tượng user được gọi hầu như là xuyên suốt quá trình, nên dùng SharePreferences để lưu trữ thông tin vào bộ nhớ của ứng dụng, đồng thời tiết kiệm thời gian đăng nhập.

```
SharedPreferences sharedPreferences = getApplicationContext().getSharedPreferences( s: "UserPref", Context.MODE_PRIVATE);
String userJson = sharedPreferences.getString( s: "user", s1: "");
if (userJson.isEmpty()) {
    Log.d( tag: "Error", msg: "userJson null");
}
else {
    Gson gson = new Gson();
    user = gson.fromJson(userJson, User.class);
}
```