**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.

1. **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.
7. **Mô tả kỹ thuật**
8. **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.

A screenshot of a computer

Description automatically generated

1. **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.
  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.
  1. ***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.
  + 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.
  1. ***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 bấm 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.
  1. ***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:*

* 1. ***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());

   }

}

**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.

**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;**

**}**

* 1. ***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.