

BIBLIOTHEK

KÜTÜPHANE
OTOMASYON
SİSTEMİ



KİTAPLARIN
DÜNYASINDA YOL
ARKADAŞINIZ

28.08.2024



Projenin Amacı ve Kapsamı

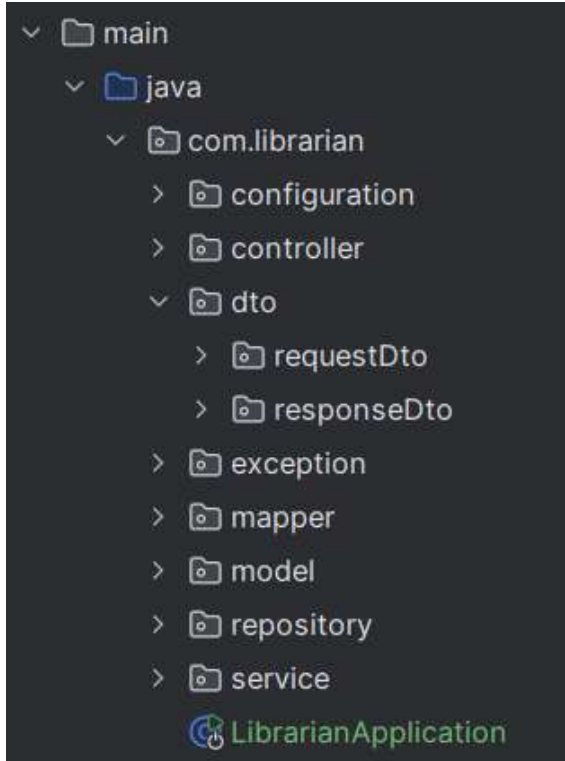
Amacı

Bibliothek, bir kütüphane yönetim sistemi olarak, kitapların, üyelerin ve ödünç işlemlerinin etkili bir şekilde yönetilmesini amaçlar.

Kapsamı

- Kitap Yönetimi: Kitap ekleme, güncelleme, silme işlemleri.
- Üye Yönetimi: Üye kayıtları, güncellemeler ve silme işlemleri.
- Ödünç ve Rezervasyon İşlemleri: Kitap ödünç alma, iade ve rezervasyon işlemleri.
- Geri Bildirim: Kullanıcılardan gelen geri bildirimlerin toplanması ve yönetilmesi.
- Raporlama: En çok alınan kitapları sıralama ve seçili id'deki rezervasyonu detaylı raporlama.

Kod Yapısı ve Bileşenler - Paketler



- **Configuration:** Uygulama yapılandırma ayarları.
- **Controller:** HTTP isteklerini işleyen sınıflar.
- **DTO:** Veri taşıma nesneleri.
- **Exception:** Hata işleme ve özelleştirilmiş istisnalar.
- **Mapper:** Veri dönüşüm işlemleri.
- **Model:** Veritabanı tablolarını temsil eden sınıflar, entityler.
- **Repository:** Veritabanı, CRUD ve JPA işlemleri.
- **Service:** Business ve uygulama içi işlemler.

Paketler – Model (Entities)

```
@Entity new *
@Data
@Table(name = "book")
public class Book {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(length = 36, nullable = false,
            unique = true)
    private UUID isbn;

    @Column(nullable = false)
    private String title;

    @Column(nullable = false)
    private String author;

    @Column(nullable = false)
    private String category;

    @Column(nullable = false)
    private int publicationYear;

    @OneToMany(mappedBy = "book")
    @JsonManagedReference
    private List<Feedback> feedbacks;

    private int timesLoaned;
}
```

Book Entity

```
@Entity new *
@Table(name = "member")
@Data
public class Member {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false)
    private String name;

    @Column(nullable = false)
    private String surname;

    @Column(nullable = false)
    private String email;

    @Column(nullable = false, name = "phone_number")
    private String phone_number;

    @Column(nullable = false, name = "birth_year")
    private int birth_year;

    @OneToMany(mappedBy = "member", cascade = CascadeType.ALL,
            orphanRemoval = true, fetch = FetchType.EAGER)
    @JsonManagedReference
    @JsonIgnore
    private List<Address> address;

    @OneToMany(mappedBy = "member", cascade = CascadeType.ALL,
            orphanRemoval = true)
    @JsonIgnore
    private List<Reservation> reservations;

    @OneToMany(mappedBy = "member")
    @JsonIgnore
    private List<Feedback> feedbacks;
}
```

Member Entity

```
@Data new *
@Entity
@Table(name = "reservation")
public class Reservation {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @ManyToOne(fetch = FetchType.EAGER)
    @JoinColumn(name = "book_id")
    private Book book;

    @ManyToOne
    @JoinColumn(name = "member_id")
    private Member member;

    @Column(name = "loan_date")
    @JsonFormat(shape = JsonFormat.Shape.STRING,
            pattern = "yyyy-MM-dd")
    private LocalDate reservationDate;

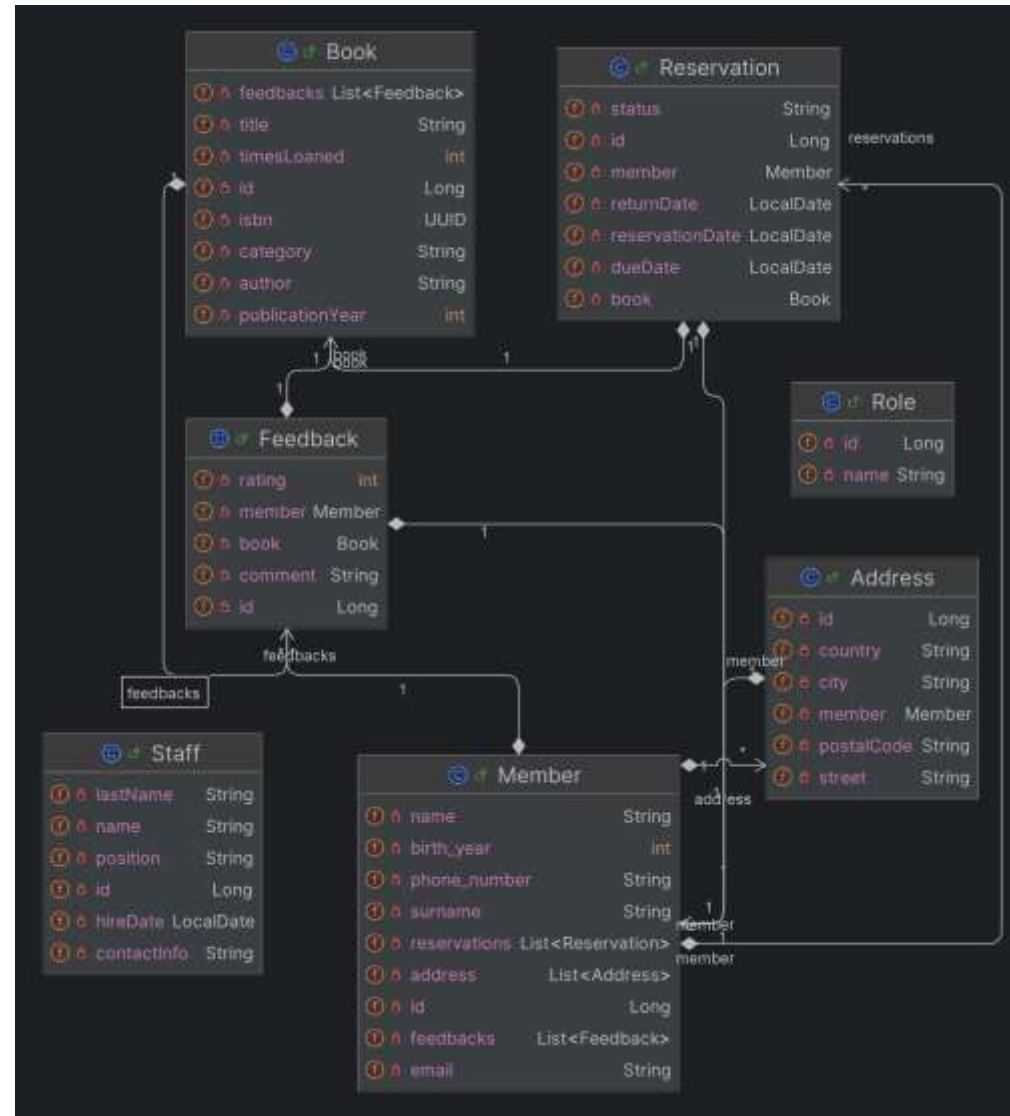
    @Column(name = "due_date")
    @JsonFormat(shape = JsonFormat.Shape.STRING,
            pattern = "yyyy-MM-dd")
    private LocalDate dueDate;

    @Column(name = "return_date")
    @JsonFormat(shape = JsonFormat.Shape.STRING,
            pattern = "yyyy-MM-dd")
    private LocalDate returnDate;

    @Column(nullable = false)
    private String status;
}
```

Reservation Entity

ER (Varlık-İlişki) Diyagramı



Member Entity - Kod Örnekleri

Controller
Layer

```
@PostMapping("/{memberId}/addReservation") new *
public ResponseEntity<String> addReservation(@PathVariable Long memberId,
                                             @RequestBody ReservationSaveRequestDto reservationSaveRequestDto) {

    try {
        boolean isAdded = memberService.addReservation(memberId, reservationSaveRequestDto);
        if (isAdded) {
            return ResponseEntity.ok( body "Reservation added successfully");
        } else {
            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("Failed to add reservation");
        }
    } catch (RuntimeException e) {
        return ResponseEntity.status(HttpStatus.NOT_FOUND).body(e.getMessage());
    }
}
```

```
@PostMapping("/{memberId}/addAddress") new *
public ResponseEntity<MemberGetResponseDto> addAddressToMember(
    @PathVariable Long memberId,
    @RequestBody Address address) {
    Member member = memberService.addAddressToMember(memberId, address);
    MemberGetResponseDto responseDto = modelMapper.map(member,
        MemberGetResponseDto.class);
    return ResponseEntity.ok(responseDto);
}
```

Service
Layer

```
@Transactional * 1 usage new *
public boolean addReservation(Long memberId, ReservationSaveRequestDto reservationSaveRequestDto) {
    Member member = memberRepository.findById(memberId)
        .orElseThrow(() -> new RuntimeException("Member not found"));

    Book book = bookRepository.findById(reservationSaveRequestDto.getBookId())
        .orElseThrow(() -> new RuntimeException("Book not found"));

    Reservation reservation = new Reservation();
    reservation.setBook(book);
    reservation.setMember(member);
    reservation.setReservationDate(reservationSaveRequestDto.getReservationDate());
    reservation.setDueDate(reservationSaveRequestDto.getDueDate());
    reservation.setReturnDate(reservationSaveRequestDto.getReturnDate());
    reservation.setStatus(reservationSaveRequestDto.getStatus());

    reservationRepository.save(reservation);
    return true;
}
```

```
@Transactional * 1 usage new *
public Member addAddressToMember(Long memberId, Address address) {
    Member member = memberRepository.findById(memberId)
        .orElseThrow(() -> new RuntimeException("Member not found"));

    member.getAddress().add(address);
    address.setMember(member);

    return memberRepository.save(member);
}
```

Kitap Rezervasyonu (Loan)

Adres Ekleme

* @Transactional, bir işlemdeki tüm veritabanı değişikliklerinin tutarlı bir şekilde gerçekleştirilmesini sağlar. İşlem sırasında bir hata oluşursa, yapılan değişiklikler geri alınır ve veri bütünlüğü korunur.

Raporlama

Constructor
Injection

```
@RestController new *
@RequestMapping("/report")
@RequiredArgsConstructor
public class ReportController {

    private final ReportService reportService;

    @GetMapping("/loans") new *
    public List<Reservation> getLoanReports() {
        return reportService.getLoanReports();
    }

    @GetMapping("/popular-books") new *
    public List<Book> getPopularBooks() {
        return reportService.getPopularBooks();
    }
}
```

Report Controller Layer

```
@Service 2 usages new *
@RequiredArgsConstructor
public class ReportService {

    private final ReservationRepository reservationRepository;
    private final BookRepository bookRepository;

    public List<Reservation> getLoanReports() { 1 usage new *
        return reservationRepository.findAll();
    }

    public List<Book> getPopularBooks() { 1 usage new *
        List<Reservation> reservations = reservationRepository.findAll();

        Map<Long, Long> bookLoanCounts = reservations.stream()
            .collect(Collectors.groupingBy(
                reservation -> reservation.getBook().getId(),
                Collectors.counting()));

        List<Book> books = bookRepository.findAll();
        for (Book book : books) {
            Long loanCount = bookLoanCounts.getOrDefault(book.getId(),
                defaultValue: 0L);
            book.setTimesLoaned(loanCount.intValue());
        }

        return books.stream()
            .sorted((b1, b2) -> Integer.compare(b2.getTimesLoaned(),
                b1.getTimesLoaned()))
            .limit(maxSize: 5)
            .collect(Collectors.toList());
    }
}
```

Report Service Layer

- Tüm rezerve işlem verilerini detaylı olarak raporlama
- Tüm kitap rezervasyonu verilerini çekme
- Kitapları özetleyerek her bir kitabın kaç kez ödünç alındığını (veya rezerve edildiğini) hesaplama
- Tüm kitap verilerini çekme
- En popüler (en çok rezerve edilen ve ödünç alınan) kitapları azalan şekilde sıralama.

Book Entity - Kod Örnekleri

Save
Function

```
@PostMapping("/{saveBook}") new *
public ResponseEntity<Boolean> saveBook(
    @RequestBody BookSaveRequestDto bookSaveRequestDto){
    Boolean savedBook = bookService.saveBook(bookSaveRequestDto);
    return new ResponseEntity<>(savedBook, HttpStatus.CREATED);
}
```

Update
Function

```
@PutMapping("/{updateById/{bookId}}") new *
public ResponseEntity<Book> updateBook(
    @PathVariable Long bookId,
    @RequestBody BookUpdateRequestDto bookUpdateRequestDto){

    Book updatedBook = bookService.updateBook(bookId,
        bookUpdateRequestDto);
    if (updatedBook != null) {
        return ResponseEntity.ok(updatedBook);
    } else {
        return ResponseEntity.notFound().build();
    }
}
```

Search
Function

```
@GetMapping("/{search}") new *
public List<Book> searchBooks(@RequestParam String title) {
    return bookService.searchBooksByTitle(title);
}
```

```
public boolean saveBook(BookSaveRequestDto bookSaveRequestDto){
    Book book = modelMapper.map(bookSaveRequestDto, Book.class);
    bookRepository.save(book);
    return true;
}
```

```
public Book updateBook(Long bookId, 1usage new *
    BookUpdateRequestDto bookUpdateRequestDto) {
    Book existingBook = bookRepository.findById(bookId).orElse( other: null);

    if (existingBook != null) {
        existingBook.setTitle(bookUpdateRequestDto.getTitle());
        existingBook.setAuthor(bookUpdateRequestDto.getAuthor());
        existingBook.setCategory(bookUpdateRequestDto.getCategory());
        return bookRepository.save(existingBook);
    } else {
        return null;
    }
}
```

```
public List<Book> searchBooksByTitle(String title) { 1usage new *
    return bookRepository.findByTitleContainingIgnoreCase(title);
}
```

Book Controller Layer

Book Service Layer

Feedback Sistemi

```
@PostMapping(Ⓜ"/add/{bookId}/{memberId}") new *
public FeedbackGetResponseDto addFeedback(
    @PathVariable Long bookId,
    @PathVariable Long memberId,
    @RequestBody FeedbackGetResponseDto feedbackDto) {
    return feedbackService.saveFeedback(bookId, memberId,
        feedbackDto);
}

@GetMapping(Ⓜ"/book/{bookId}") new *
public List<FeedbackGetResponseDto> getFeedbacksByBook(
    @PathVariable Long bookId) {
    return feedbackService.getFeedbacksByBook(bookId);
}

@GetMapping(Ⓜ"/member/{memberId}") new *
public List<FeedbackGetResponseDto> getFeedbacksByMember(
    @PathVariable Long memberId) {
    return feedbackService.getFeedbacksByMember(memberId);
}
```

Feedback Controller Layer

```
public FeedbackGetResponseDto saveFeedback(Long bookId, Long memberId, Usage new *
    FeedbackGetResponseDto feedbackDto) {
    Book book = bookRepository.findById(bookId)
        .orElseThrow(() -> new RuntimeException("Book not found"));
    Member member = memberRepository.findById(memberId)
        .orElseThrow(() -> new RuntimeException("Member not found"));

    Feedback feedback = feedbackMapper.toEntity(feedbackDto);
    feedback.setBook(book);
    feedback.setMember(member);
    Feedback savedFeedback = feedbackRepository.save(feedback);
    return feedbackMapper.toDto(savedFeedback);
}

public List<FeedbackGetResponseDto> getFeedbacksByBook(Long bookId) { Usage new *
    return feedbackRepository.findAll().stream() Stream<Feedback>
        .filter(fb -> fb.getBook().getId().equals(bookId))
        .map(feedbackMapper::toDto) Stream<FeedbackGetResponseDto>
        .collect(Collectors.toList());
}

public List<FeedbackGetResponseDto> getFeedbacksByMember(Long memberId) { Usage new *
    return feedbackRepository.findAll().stream() Stream<Feedback>
        .filter(fb -> fb.getMember().getId().equals(memberId))
        .map(feedbackMapper::toDto) Stream<FeedbackGetResponseDto>
        .collect(Collectors.toList());
}
```

Feedback Service Layer

➤ User feedback'i kaydetme

➤ Seçili kitabın
feedbacklerini
alma

➤ Seçili üyenin
yaptığı feed-
backleri alma

KAYNAK KODU VE DOKÜMANTASYONLAR

➤ Source Code: [gramchelle/Bibliothek](#)

➤ Requests Documentation:

Tüm cURL'ler ve örnek Request Body'ler aşağıdaki linkte temin edilmiştir.

<https://documenter.getpostman.com/view/37170721/2sAXjGcDyr>

➤ Cover Image Reference: [Open book](#)

A stack of books with an open book on top, surrounded by floating mathematical symbols like plus, minus, multiplication, division, and numbers.