Backend Design Doc

for Letterboxd-like film review and social media platform

System Architecture

Monolithic SpringBoot Application with layered architecture:

1. Presentation Layer (Controllers)
2. Service Layer (Business Logic)
3. Repository Layer (Data Access)
4. Domain Layer (Entities/Models)

Core Domain Models

1. User Management

```
User {
    Long id
    String username (unique)
    String email (unique)
    String password (encrypted)
    String displayName
    String bio
    String avatarUrl
    LocalDateTime joinedDate
    AccountStatus status (ACTIVE, SUSPENDED, DELETED)
    UserPreferences preferences
    Set<Role> roles
```

```
}

UserPreferences {
    Boolean privateWatchlist
    Boolean privateReviews
    Boolean showActivityFeed
    NotificationPreferences notifications
}
```

2. Film Catalog System

```
Film {
    Long id
    String title
    String originalTitle
    Integer releaseYear
    String language
    String country
    Integer durationMinutes
    String synopsis
    String posterUrl (uploaded to S3/cloud storage)
    String backdropUrl
    LocalDateTime addedDate
    FilmStatus status (PUBLISHED, DRAFT)
    Set<Genre> genres
    Set<Person> directors
    Set<Person> cast
    FilmMetadata metadata (avgRating, reviewCount, etc.)
}

Person {
    Long id
    String name
    String biography
    LocalDate birthDate
```

```
    LocalDate deathDate (nullable)
    String photoUrl
    PersonType type (DIRECTOR, ACTOR, WRITER, etc.)
    Set<Film> filmography
}

Genre {
    Long id
    String name
    String slug
}
```

3. Review & Rating System

```
Review {
    Long id
    User user
    Film film
    String content
    Integer rating (1-5, nullable)
    LocalDateTime reviewDate
    LocalDateTime modifiedDate
    Integer likesCount
    Set<User> likedBy
    ReviewStatus status (PUBLIC, PRIVATE, DELETED)
    Boolean containsSpoilers
}

Rating {
    Long id
    User user
    Film film
    Integer score (1-5)
    LocalDateTime ratedDate
    @Embeddable UserFilmKey (user_id, film_id) composite
```

```
key
}
```

4. Social Features

```
Follow {
    Long id
    User follower
    User following
    LocalDateTime followDate
}

Watchlist {
    Long id
    User user
    Film film
    LocalDateTime addedDate
    WatchStatus status (PLAN_TO_WATCH, WATCHING)
}

DiaryEntry {
    Long id
    User user
    Film film
    LocalDate watchDate
    LocalDateTime loggedDate
    Integer rewatchCount
    String notes
    Set<Tag> tags
}

List {
    Long id
    User creator
    String title
    String description
```

```
    Boolean isRanked
    LocalDateTime createdDate
    Integer likesCount
    Set<ListItem> items
    Set<User> likedBy
}

Comment {
    Long id
    User user
    String content
    LocalDateTime commentDate
    CommentableEntity target (polymorphic:
Review/List/Diary)
    Comment parentComment (nullable for replies)
}
```

5. Activity & Feed

```
Activity {
    Long id
    User actor
    ActivityType type (REVIEWED, RATED, WATCHED,
FOLLOWED, etc.)
    String targetType (FILM, USER, LIST, etc.)
    Long targetId
    LocalDateTime activityDate
    String metadata (JSON)
}

Feed {
    // Aggregates activities from followed users
    // Implemented as materialized view or real-time
aggregation
}
```

6. Tagging System

```
Tag {
    Long id
    String name
    String slug
    TagType type (GENRE, MOOD, ERA, etc.)
}

FilmTag {
    Film film
    Tag tag
    Integer count (usage frequency)
}
```

Service Layer Components

1. Film Management Services

- FilmService: CRUD operations, search, filtering
- PersonService: Manage directors/actors
- GenreService: Genre management
- FilmImportService: Manual/admin film entry with validation

2. User Interaction Services

- ReviewService: Create/update/delete reviews
- RatingService: Handle star ratings
- WatchlistService: Manage user watchlists

- DiaryService: Log film watches
- ListService: Create and manage film lists

3. Social Services

- FollowService: User following/unfollowing
- FeedService: Generate activity feeds
- NotificationService: In-app notifications
- CommentService: Comment management

4. Discovery & Recommendation Services

- RecommendationService: Collaborative filtering
- SearchService: Full-text search with ElasticSearch
- TrendingService: Calculate trending films

API Endpoints Design

A. Public APIs (No Auth Required)

```
GET /api/films                # Browse films with filters
GET /api/films/{id}           # Film details
GET /api/films/{id}/reviews   # Film reviews
GET /api/reviews/{id}         # Single review
GET /api/users/{username}     # Public profile
GET /api/search?q={query}     # Search films/users
```

B. User Authentication

```
POST /api/auth/register          # Register
POST /api/auth/login             # Login (JWT)
POST /api/auth/refresh           # Refresh token
POST /api/auth/logout            # Logout
```

C. User Actions (Require Auth)

```
# Reviews & Ratings
POST   /api/reviews              # Create review
PUT    /api/reviews/{id}         # Update review
DELETE /api/reviews/{id}         # Delete review
POST   /api/films/{id}/rate      # Rate film
POST   /api/reviews/{id}/like    # Like review

# Watchlist & Diary
POST   /api/watchlist            # Add to watchlist
GET    /api/me/watchlist         # My watchlist
POST   /api/diary/entries        # Log diary entry
GET    /api/me/diary             # My diary

# Social Features
POST   /api/users/{id}/follow    # Follow user
GET    /api/me/following         # Users I follow
GET    /api/me/followers         # My followers
POST   /api/lists                # Create list
GET    /api/me/feed              # Activity feed

# Profile Management
PUT    /api/me/profile           # Update profile
PUT    /api/me/preferences       # Update preferences
```

D. Admin APIs

```
# Film Management
POST   /api/admin/films           # Add film
PUT    /api/admin/films/{id}      # Update film
POST   /api/admin/persons         # Add person
POST   /api/admin/genres          # Add genre

# Moderation
GET    /api/admin/reports            # View reports
POST   /api/admin/reviews/{id}/hide # Hide review
PUT    /api/admin/users/{id}/status # Change user status
```

Database Design

Primary Database: PostgreSQL

- **Tables:** users, films, persons, reviews, ratings, follows, watchlist, diary_entries, lists, comments, activities, tags, genres
- **Indexes**: Composite indexes on frequently joined columns
- **Partitioning**: Activity table by date for older data

Search: ElasticSearch Integration

- Films index with title, synopsis, cast names
- Users index for search
- Reviews index for full-text search

Caching: Redis

- Film details cache
- User feed cache
- Trending films cache
- Session storage

Key Features Implementation

1. Film Entry System

- Admin panel for manual film entry
- Bulk upload via CSV/Excel
- Duplicate detection by (title + releaseYear)
- Image upload to cloud storage (S3/Cloudinary)

2. Recommendation Engine

- Collaborative filtering based on ratings
- Content-based filtering by genres/directors
- "Users who liked this also liked"
- Popular among friends

3. Activity Feed

- Real-time using WebSocket/Server-Sent Events
- Materialized view for performance
- Personalized feed algorithm

4. Search System

- ElasticSearch for films, users, reviews
- Faceted search by genre, year, rating
- Auto-complete suggestions

5. Moderation System

- User reporting system
- Automated content flagging
- Admin moderation queue
- Shadow banning capability

Security Design

Authentication & Authorization

- JWT-based authentication
- Role-based access control (USER, MODERATOR, ADMIN)
- OAuth2 for social login (optional)
- Rate limiting per endpoint

Data Protection

- Password encryption with BCrypt
- PII data encryption at rest
- GDPR compliance tools
- Data export functionality

Content Safety

- Profanity filter for user-generated content
- Image moderation (optional)
- Report abuse system

Performance Optimizations

1. Database Level

- Read replicas for heavy read operations
- Connection pooling (HikariCP)
- Query optimization with EXPLAIN
- Database partitioning for activity logs

2. Application Level

- Response caching with @Cacheable
- Pagination for all list endpoints
- Lazy loading with DTO projection
- Async processing for notifications

3. Infrastructure

- CDN for static assets (posters, avatars)
- Load balancing
- Horizontal scaling capability

Monitoring & Analytics

Metrics Collection

- User engagement metrics
- Film popularity trends
- Review sentiment analysis
- Daily active users

Admin Dashboard

- Site statistics
- User growth charts
- Content moderation tools
- System health monitoring

Deployment & DevOps

Containerization

- Docker containers
- Docker Compose for local development
- Kubernetes for production (optional)

CI/CD Pipeline

- Automated testing
- Blue-green deployment
- Database migration management

Monitoring Stack

- Spring Boot Actuator
- Prometheus for metrics
- Grafana dashboards
- Log aggregation (ELK Stack)

Scalability Considerations

Vertical Scaling

- Microservices split when needed:

    1. Film Service
    2. User Service
    3. Social Graph Service
    4. Feed Generation Service
    5. Search Service

Data Sharding

- User data by region
- Films by language/country
- Reviews by film ID

Message Queue

RabbitMQ/Kafka for async operations:

- Notification delivery
- Feed generation
- Analytics processing

Key Technical Decisions

1. **No external film APIs:** Manual curation with admin tools
2. **PostgreSQL**: For ACID compliance and JSON support
3. **JWT authentication**: Stateless and scalable
4. **ElasticSearch**: For complex search requirements
5. **Redis**: For caching and session management
6. **Event-driven architecture**: For real-time features
7. **Container-first approach**: For consistent deployments