InsightX Spring Boot Backend - Comprehensive Development Plan

Based on my thorough analysis of the design document, here's a detailed implementation plan:

Phase 1: Project Foundation & Infrastructure Setup

1.1 Project Initialization

- Create Spring Boot 3 project with Maven/Gradle
- Configure Java 17 as the baseline
- Set up multi-module structure if needed (optional but recommended for separation of concerns)
- Initialize Git repository with proper .gitignore

1.2 Core Dependencies

- Spring Boot Starter Web
- Spring Boot Starter Security (JWT authentication)
- Spring Boot Starter Data JPA
- PostgreSQL driver
- Redis (Lettuce client via Spring Data Redis)
- WebClient (Spring WebFlux for async HTTP calls to FastAPI)
- Lombok (reduce boilerplate)
- MapStruct (DTO-entity mapping)
- Validation API
- JWT library (jjwt or similar)
- SpringDoc OpenAPI (API documentation)

1.3 Environment Configuration

- Create application.yml/properties structure
- Define profiles: dev, staging, production
- Configure database connection pooling (HikariCP)
- Set up Redis connection with failover handling
- Configure JWT secret management (externalize via environment variables)
- Set up logging framework (SLF4J + Logback)

Phase 2: Database Schema Design

2.1 Core Entity Design

User Management:

- Users table (id, username, email, password_hash, region, created_at, updated_at)
- User preferences table (user_id, key, value) - flexible key-value storage
- User sessions/tokens table (optional, for token revocation)

Media State Tracking:

- Watched entries table (user_id, media_id, media_type, watched_date)
- Ratings table (user_id, media_id, media_type, rating, timestamp)
- Reviews table (id, user_id, media_id, media_type, review_text, created_at, updated_at)
- Bookmarks/saved items table (user_id, media_id, media_type, saved_at)

Derived Data:

- User taste profile cache table (user_id, profile_json, last_updated)
- Consider separate tables for genre preferences, theme affinities

2.2 Database Optimization

- Define appropriate indexes (user_id, media_id combinations)
- Set up composite keys where necessary
- Configure cascading rules
- Implement soft delete if needed
- Add database-level constraints

2.3 Migration Strategy

- Use Flyway or Liquibase for version-controlled migrations
- Create baseline schema
- Plan for incremental migrations

Phase 3: Security Layer Implementation

3.1 Authentication System

- JWT token generation service
- Token validation and parsing
- Refresh token mechanism
- Password encryption (BCrypt)

3.2 Authorization Framework

- Spring Security filter chain configuration
- JWT authentication filter
- Role-based access control (if needed for future admin features)
- Security context management

3.3 API Security

- CORS configuration for Flutter client
- Rate limiting per user/IP (consider using bucket4j)
- Request validation
- XSS and SQL injection protection

Phase 4: Core Domain Layer

4.1 Entity Layer

Design JPA entities for:

- User
- UserPreference
- WatchedEntry
- Rating
- Review
- Bookmark

Include proper relationships, cascade types, and fetch strategies

4.2 Repository Layer

Create Spring Data JPA repositories with:

- Custom query methods
- Pagination support
- Specification API for complex queries
- Projections for optimized data retrieval

4.3 Service Layer Architecture

Implement service classes for:

- UserService (registration, profile management)
- AuthenticationService (login, token management)
- WatchedService (mark as watched, get watch history)

- RatingService (submit/update ratings, get user ratings)
- ReviewService (CRUD operations for reviews)
- PreferenceService (manage user preferences and region)
- TasteProfileService (generate and cache taste profiles)

Phase 5: DTO & API Layer

5.1 DTO Design

Create request/response DTOs for:

- Authentication (LoginRequest, RegisterRequest, TokenResponse)
- User profile (UserProfileDTO, UpdateProfileRequest)
- Media interactions (WatchedRequest, RatingRequest, ReviewRequest)
- Preferences (PreferenceRequest, RegionUpdateRequest)

5.2 Controller Layer

Implement REST controllers:

- AuthController (/api/auth/*)
- UserController (/api/users/*)
- MediaStateController (/api/media/*)
- PreferenceController (/api/preferences/*)
- ReviewController (/api/reviews/*)

5.3 API Design Principles

- RESTful conventions
- Consistent response structure (success, error, data)
- Proper HTTP status codes
- Pagination for list endpoints
- API versioning strategy (URL path or header-based)

Phase 6: FastAPI Integration Layer

6.1 WebClient Configuration

- Configure non-blocking WebClient
- Set up connection pooling
- Implement timeout strategies

- Add retry logic with exponential backoff

6.2 FastAPI Service Interface

Create service to communicate with FastAPI for:

- Media metadata retrieval
- Recommendation requests
- Cross-media mapping
- Theme extraction
- AI explanation requests

6.3 Service-Level Authentication

- Implement internal service authentication mechanism
- API key or token-based auth between Spring Boot and FastAPI
- Request/response logging for debugging

Phase 7: Redis Integration

7.1 Cache Configuration

- Configure Redis connection pool
- Set up serialization/deserialization
- Implement cache key naming strategy
- Define TTL strategies per data type

7.2 Caching Strategy Implementation

Implement cache-aside pattern for:

- External API responses
- Watch provider lookups
- Recommendation results
- AI explanations
- Frequently accessed media details

7.3 Fault Tolerance

- Handle Redis unavailability gracefully
- Implement circuit breaker pattern
- Log cache misses for monitoring
- Ensure application functions without cache

Phase 8: Business Logic Implementation

8.1 Taste Profile Generation

- Algorithm to analyze user ratings
- Genre frequency calculation
- Theme affinity detection
- Weighting strategy based on recency
- Caching mechanism (PostgreSQL + Redis)

8.2 User Interaction Workflows

Complete flows for:

- Mark as watched with validation
- Rate content with business rules
- Write/edit/delete reviews
- Bookmark management

8.3 Regional Awareness

- Region storage and retrieval
- Region-based availability filtering
- Localization support structure

Phase 9: Cross-Cutting Concerns

9.1 Exception Handling

- Global exception handler (@ControllerAdvice)
- Custom exception hierarchy
- Consistent error response format
- Logging strategy for errors

9.2 Validation Framework

- Bean validation annotations
- Custom validators where needed
- Request validation
- Business rule validation

9.3 Audit & Logging

- Request/response logging interceptor
- Audit trail for critical operations
- Performance monitoring points
- Database query logging (dev environment)

9.4 API Documentation

- SpringDoc OpenAPI integration
- Swagger UI configuration
- Comprehensive endpoint documentation
- Request/response examples

Phase 10: Testing Strategy

10.1 Unit Testing

- Service layer tests with Mockito
- Repository tests with @DataJpaTest
- Utility and helper class tests
- Target 80%+ code coverage

10.2 Integration Testing

- Controller tests with MockMvc
- Security tests
- Database integration tests with Testcontainers
- Redis integration tests

10.3 API Contract Testing

- Test request/response contracts
- Validation testing
- Error scenario testing

Phase 11: Production Readiness

11.1 Configuration Management

- Externalize all configuration
- Environment-specific property files
- Secrets management strategy

11.2 Health & Monitoring

- Spring Boot Actuator setup
- Health check endpoints
- Metrics exposure (Prometheus format recommended)
- Custom health indicators (database, Redis, FastAPI)

11.3 Performance Optimization

- Database query optimization
- Connection pool tuning
- Implement database connection pooling best practices
- Response compression
- Lazy loading strategy

11.4 Deployment Preparation

- Dockerization (Dockerfile + docker-compose)
- CI/CD pipeline preparation
- Environment variable documentation
- Startup and shutdown hooks

Recommended Development Order

1. **Week 1-2:** Phase 1-3 (Foundation, Database, Security)
2. **Week 3-4:** Phase 4-5 (Domain Layer, API Layer)
3. **Week 5:** Phase 6-7 (FastAPI Integration, Redis)
4. **Week 6:** Phase 8 (Business Logic)
5. **Week 7:** Phase 9 (Cross-cutting Concerns)
6. **Week 8:** Phase 10-11 (Testing, Production Readiness)

Critical Design Considerations

1. **Stateless Design:** Keep Spring Boot stateless to allow horizontal scaling
2. **Idempotency:** Ensure critical operations are idempotent
3. **Transaction Management:** Proper transaction boundaries, especially for user state operations
4. **API Contract Stability:** Version APIs to avoid breaking Flutter client

5. **Graceful Degradation:** System should work even if Redis or FastAPI is temporarily unavailable
6. **Data Consistency:** Ensure PostgreSQL is single source of truth; Redis is cache only
7. **Security First:** Validate all inputs, secure all endpoints, encrypt sensitive data