📋 Informe Completo - Arquitectura Reddit Clone

🎯 Resumen Ejecutivo

Decisión Arquitectónica

- **Backend**: Arquitectura de Microservicios con Spring Boot 3.2
- **Frontend**: Angular 17+ con Standalone Components
- **Base de Datos**: Oracle Database 21c XE (una por microservicio)
- **Comunicación**: REST APIs con Feign Clients
- **Service Discovery**: Eureka Server
- **API Gateway**: Spring Cloud Gateway

🏗️ ARQUITECTURA BACKEND - MICROSERVICIOS

🌐 Componentes Principales

- **8 Microservicios principales**
- **API Gateway** para routing y seguridad
- **Service Discovery** con Eureka
- **Bases de datos Oracle independientes** por servicio

🗄️ Distribución de Bases de Datos

Servicio	Base de Datos	Puerto	Responsabilidad Principal
API Gateway	Ninguna (stateless)	8080	Enrutamiento y seguridad
Auth Service	auth_db	8081	Autenticación y autorización
User Service	user_db	8082	Gestión de perfiles
Community Service	community_db	8083	Gestión de comunidades
Post Service	post_db	8084	Gestión de posts
Comment Service	comment_db	8085	Sistema de comentarios
Vote Service	vote_db	8086	Sistema de votación
Notification Service	notification_db	8087	Notificaciones

🔧 DETALLE DE MICROSERVICIOS

1. API Gateway Service (Puerto 8080)

Base de Datos: Ninguna (stateless)

Responsabilidades:

- Punto de entrada único
- Enrutamiento de requests
- Rate limiting
- Load balancing
- CORS configuration
- Request/Response logging

Configuración de Rutas:

yaml

```yaml
spring:
  cloud:
    gateway:
      routes:
        - id: auth-service
          uri: lb://auth-service
          predicates:
            - Path=/api/auth/**
        - id: user-service
          uri: lb://user-service
          predicates:
            - Path=/api/users/**
        - id: community-service
          uri: lb://community-service
          predicates:
            - Path=/api/communities/**
        - id: post-service
          uri: lb://post-service
          predicates:
            - Path=/api/posts/**
        - id: comment-service
          uri: lb://comment-service
          predicates:
            - Path=/api/comments/**
        - id: vote-service
          uri: lb://vote-service
          predicates:
            - Path=/api/votes/**
        - id: notification-service
          uri: lb://notification-service
          predicates:
            - Path=/api/notifications/**
```

2. Auth Service (Puerto 8081)

Base de Datos: auth_db (Oracle)

Responsabilidades:

- Autenticación (login/signup)
- Autorización (JWT tokens)
- OAuth2 (Google)
- Password reset
- Refresh tokens
- User roles y permissions

Estructura del Proyecto:

```
auth-service/
├── src/main/java/com/redditclone/auth/
│   ├── AuthServiceApplication.java
│   ├── config/
│   │   ├── SecurityConfig.java
│   │   ├── JwtConfig.java
│   │   └── OAuth2Config.java
│   ├── controller/
│   │   ├── AuthController.java
│   │   └── OAuth2Controller.java
│   ├── service/
│   │   ├── AuthService.java
│   │   ├── JwtService.java
│   │   ├── RefreshTokenService.java
│   │   └── OAuth2Service.java
│   ├── model/
│   │   ├── User.java
│   │   ├── RefreshToken.java
│   │   ├── UserRole.java
│   │   └── OAuth2UserInfo.java
│   ├── dto/
│   │   ├── LoginRequest.java
│   │   ├── SignupRequest.java
│   │   ├── AuthResponse.java
│   │   ├── RefreshTokenRequest.java
│   │   └── PasswordResetRequest.java
│   ├── repository/
│   │   ├── UserRepository.java
│   │   └── RefreshTokenRepository.java
│   └── security/
│       ├── JwtAuthenticationFilter.java
│       ├── JwtAuthenticationEntryPoint.java
│       └── CustomUserDetailsService.java
│
```

Esquema de Base de Datos (auth_db):

sql

```sql
TABLES:
- users (id, username, email, password_hash, created_at, updated_at)
- refresh_tokens (id, user_id, token_hash, expires_at, created_at)
- oauth2_users (id, user_id, provider, provider_id, created_at)
- user_roles (id, user_id, role_name, created_at)
```

3. User Service (Puerto 8082)

Base de Datos: user_db (Oracle)

Responsabilidades:

- Gestión de perfiles de usuario
- Configuraciones de usuario
- Avatar y banner upload
- User stats (karma, awards)
- Followers/Following
- User preferences

Estructura del Proyecto:

```
user-service/
├── src/main/java/com/redditclone/user/
│   ├── UserServiceApplication.java
│   ├── controller/
│   │   ├── UserController.java
│   │   ├── UserProfileController.java
│   │   ├── UserSettingsController.java
│   │   └── UserStatsController.java
│   ├── service/
│   │   ├── UserService.java
│   │   ├── UserProfileService.java
│   │   ├── UserSettingsService.java
│   │   ├── AvatarService.java
│   │   └── UserStatsService.java
│   ├── model/
│   │   ├── User.java
│   │   ├── UserProfile.java
│   │   ├── UserSettings.java
│   │   ├── NotificationSettings.java
│   │   ├── UserStats.java
│   │   └── UserFollow.java
│   ├── dto/
│   │   ├── UserDTO.java
│   │   ├── UserProfileDTO.java
│   │   ├── UserStatsDTO.java
│   │   ├── UpdateProfileRequest.java
│   │   ├── UserSettingsDTO.java
│   │   └── AvatarUploadResponse.java
│   ├── repository/
│   │   ├── UserRepository.java
│   │   ├── UserProfileRepository.java
│   │   ├── UserSettingsRepository.java
│   │   ├── UserStatsRepository.java
│   │   └── UserFollowRepository.java
│   └── client/
│       └── AuthServiceClient.java
│
```

Esquema de Base de Datos (user_db):

```sql
TABLES:
- user_profiles (id, user_id, display_name, bio, location, avatar_url, banner_url, created_at, updated_at)
- user_settings (id, user_id, theme, language, timezone, privacy_settings, created_at, updated_at)
- notification_settings (id, user_id, email_notifications, push_notifications, frequency, dnd_start, dnd_end)
- user_stats (id, user_id, post_karma, comment_karma, total_posts, total_comments, awards_received)
- user_follows (id, follower_id, following_id, created_at)
```

4. Community Service (Puerto 8083)

Base de Datos: community_db (Oracle)

Responsabilidades:

- Gestión de comunidades/subreddits
- Memberships y roles
- Community settings y rules
- Moderator management
- Community stats
- Community discovery

Estructura del Proyecto:

```
community-service/
├── src/main/java/com/redditclone/community/
│   ├── CommunityServiceApplication.java
│   ├── controller/
│   │   ├── CommunityController.java
│   │   ├── CommunityMembershipController.java
│   │   ├── CommunityModerationController.java
│   │   └── CommunityRulesController.java
│   ├── service/
│   │   ├── CommunityService.java
│   │   ├── CommunityMembershipService.java
│   │   ├── CommunityModerationService.java
│   │   ├── CommunityRulesService.java
│   │   └── CommunityStatsService.java
│   ├── model/
│   │   ├── Community.java
│   │   ├── CommunityMembership.java
│   │   ├── CommunityRule.java
│   │   ├── CommunityModerator.java
│   │   ├── CommunityStats.java
│   │   └── CommunitySettings.java
│   ├── dto/
│   │   ├── CommunityDTO.java
│   │   ├── CreateCommunityRequest.java
│   │   ├── CommunityStatsDTO.java
│   │   ├── CommunityRuleDTO.java
│   │   ├── CommunityMembershipDTO.java
│   │   └── ModeratorDTO.java
│   ├── repository/
│   │   ├── CommunityRepository.java
│   │   ├── CommunityMembershipRepository.java
│   │   ├── CommunityRuleRepository.java
│   │   ├── CommunityModeratorRepository.java
│   │   └── CommunityStatsRepository.java
│   └── client/
│       ├── UserServiceClient.java
│       └── AuthServiceClient.java
│
```

Esquema de Base de Datos (community_db):

```sql
TABLES:
- communities (id, name, display_name, description, creator_id, member_count, created_at, updated_at)
- community_memberships (id, community_id, user_id, role, joined_at)
- community_rules (id, community_id, rule_number, title, description, created_at)
- community_moderators (id, community_id, user_id, permissions, appointed_at)
- community_stats (id, community_id, total_posts, total_comments, active_users, created_at)
- community_settings (id, community_id, is_private, allow_images, allow_videos, require_approval)
```

5. Post Service (Puerto 8084)

Base de Datos: post_db (Oracle)

Responsabilidades:

- Gestión de posts
- Post content (text, image, link)
- Post metadata
- Post categories/flairs
- Draft management
- Post search

Estructura del Proyecto:

```
post-service/
├── src/main/java/com/redditclone/post/
│   ├── PostServiceApplication.java
│   ├── controller/
│   │   ├── PostController.java
│   │   ├── PostDraftController.java
│   │   ├── PostSearchController.java
│   │   └── PostFlairController.java
│   ├── service/
│   │   ├── PostService.java
│   │   ├── PostDraftService.java
│   │   ├── PostSearchService.java
│   │   ├── PostFlairService.java
│   │   └── ImageUploadService.java
│   ├── model/
│   │   ├── Post.java
│   │   ├── PostContent.java
│   │   ├── PostFlair.java
│   │   ├── PostDraft.java
│   │   ├── PostImage.java
│   │   └── PostType.java (enum)
│   ├── dto/
│   │   ├── PostDTO.java
│   │   ├── CreatePostRequest.java
│   │   ├── UpdatePostRequest.java
│   │   ├── PostDraftDTO.java
│   │   ├── PostFlairDTO.java
│   │   └── ImageUploadResponse.java
│   ├── repository/
│   │   ├── PostRepository.java
│   │   ├── PostDraftRepository.java
│   │   ├── PostFlairRepository.java
│   │   └── PostImageRepository.java
│   └── client/
│       ├── UserServiceClient.java
│       ├── CommunityServiceClient.java
│       └── AuthServiceClient.java
│
```

Esquema de Base de Datos (post_db):

```sql
TABLES:
- posts (id, title, content, post_type, author_id, community_id, vote_count, comment_count, created_at, updated_at)
- post_content (id, post_id, content_type, content_url, content_text)
- post_flairs (id, community_id, name, color, background_color, created_at)
- post_flair_assignments (id, post_id, flair_id, assigned_at)
- post_drafts (id, user_id, title, content, community_id, created_at, updated_at)
- post_images (id, post_id, image_url, image_order, uploaded_at)
```

6. Comment Service (Puerto 8085)

Base de Datos: comment_db (Oracle)

Responsabilidades:

- Gestión de comentarios
- Sistema de comentarios anidados
- Reply management
- Comment threading
- Comment moderation

Estructura del Proyecto:

```
comment-service/
├── src/main/java/com/redditclone/comment/
│   ├── CommentServiceApplication.java
│   ├── controller/
│   │   ├── CommentController.java
│   │   ├── CommentThreadController.java
│   │   └── CommentModerationController.java
│   ├── service/
│   │   ├── CommentService.java
│   │   ├── CommentThreadService.java
│   │   ├── CommentModerationService.java
│   │   └── CommentTreeService.java
│   ├── model/
│   │   ├── Comment.java
│   │   ├── CommentThread.java
│   │   ├── CommentModeration.java
│   │   └── CommentStatus.java (enum)
│   ├── dto/
│   │   ├── CommentDTO.java
│   │   ├── CreateCommentRequest.java
│   │   ├── UpdateCommentRequest.java
│   │   ├── CommentThreadDTO.java
│   │   └── CommentTreeDTO.java
│   ├── repository/
│   │   ├── CommentRepository.java
│   │   ├── CommentThreadRepository.java
│   │   └── CommentModerationRepository.java
│   └── client/
│       ├── PostServiceClient.java
│       ├── UserServiceClient.java
│       └── AuthServiceClient.java
│
```

Esquema de Base de Datos (comment_db):

```sql
TABLES:
- comments (id, content, author_id, post_id, parent_comment_id, vote_count, depth, created_at, updated_at)
- comment_threads (id, post_id, root_comment_id, total_comments, created_at)
- comment_moderation (id, comment_id, moderator_id, action, reason, created_at)
```

7. Vote Service (Puerto 8086)

Base de Datos: vote_db (Oracle)

Responsabilidades:

- Sistema de voting (upvote/downvote)
- Vote calculations
- Karma management
- Vote history
- Vote analytics

Estructura del Proyecto:

```
vote-service/
├── src/main/java/com/redditclone/vote/
│   ├── VoteServiceApplication.java
│   ├── controller/
│   │   ├── VoteController.java
│   │   ├── KarmaController.java
│   │   └── VoteAnalyticsController.java
│   ├── service/
│   │   ├── VoteService.java
│   │   ├── KarmaService.java
│   │   ├── VoteCalculationService.java
│   │   └── VoteAnalyticsService.java
│   ├── model/
│   │   ├── Vote.java
│   │   ├── VoteType.java (enum)
│   │   ├── Karma.java
│   │   └── VoteHistory.java
│   ├── dto/
│   │   ├── VoteDTO.java
│   │   ├── VoteRequest.java
│   │   ├── KarmaDTO.java
│   │   └── VoteStatsDTO.java
│   ├── repository/
│   │   ├── VoteRepository.java
│   │   ├── KarmaRepository.java
│   │   └── VoteHistoryRepository.java
│   └── client/
│       ├── PostServiceClient.java
│       ├── CommentServiceClient.java
│       ├── UserServiceClient.java
│       └── AuthServiceClient.java
│
```

Esquema de Base de Datos (vote_db):

```sql
TABLES:
- votes (id, user_id, target_id, target_type, vote_type, created_at, updated_at)
- karma_history (id, user_id, karma_type, points_change, reason, created_at)
- vote_aggregations (id, target_id, target_type, upvotes, downvotes, score, updated_at)
```

8. Notification Service (Puerto 8087)

Base de Datos: notification_db (Oracle)

Responsabilidades:

- Sistema de notificaciones
- Email notifications
- Push notifications
- Notification preferences
- Real-time notifications (WebSocket)
- Notification history

Estructura del Proyecto:

```
notification-service/
├── src/main/java/com/redditclone/notification/
│   ├── NotificationServiceApplication.java
│   ├── controller/
│   │   ├── NotificationController.java
│   │   ├── NotificationPreferencesController.java
│   │   └── WebSocketController.java
│   ├── service/
│   │   ├── NotificationService.java
│   │   ├── EmailNotificationService.java
│   │   ├── PushNotificationService.java
│   │   ├── WebSocketNotificationService.java
│   │   └── NotificationPreferencesService.java
│   ├── model/
│   │   ├── Notification.java
│   │   ├── NotificationType.java (enum)
│   │   ├── NotificationPreferences.java
│   │   ├── EmailTemplate.java
│   │   └── NotificationStatus.java (enum)
│   ├── dto/
│   │   ├── NotificationDTO.java
│   │   ├── CreateNotificationRequest.java
│   │   ├── NotificationPreferencesDTO.java
│   │   └── EmailNotificationDTO.java
│   ├── repository/
│   │   ├── NotificationRepository.java
│   │   ├── NotificationPreferencesRepository.java
│   │   └── EmailTemplateRepository.java
│   ├── websocket/
│   │   ├── WebSocketConfig.java
│   │   ├── NotificationWebSocketHandler.java
│   │   └── WebSocketSessionManager.java
│   └── client/
│       ├── UserServiceClient.java
│       ├── PostServiceClient.java
│       ├── CommentServiceClient.java
│       └── AuthServiceClient.java
│
```

Esquema de Base de Datos (notification_db):

```sql
TABLES:
- notifications (id, user_id, type, title, message, read_status, created_at)
- notification_preferences (id, user_id, email_enabled, push_enabled, frequency, dnd_start, dnd_end)
- email_templates (id, template_name, subject, html_content, text_content, created_at)
- notification_history (id, notification_id, delivery_method, status, sent_at)
```

🔄 Service Discovery y Comunicación

Eureka Server

```yaml
# Eureka Server
eureka-server:
  port: 8761

# Cada microservicio se registra en Eureka
spring:
  application:
    name: auth-service
  eureka:
    client:
      service-url:
        defaultZone: http://localhost:8761/eureka
```

Tecnologías por Microservicio

Comunes a todos:

- Spring Boot 3.2
- Spring Data JPA
- Oracle Database 21c XE
- Flyway (migraciones)
- Docker (containerización)
- Maven (build tool)

Específicas:

- **API Gateway**: Spring Cloud Gateway, Eureka Client
- **Auth Service**: Spring Security, JWT, OAuth2
- **User Service**: Cloudinary (images), Redis (cache)

- **Community Service**: Redis (cache)
- **Post Service**: Cloudinary (images), Elasticsearch (search)
- **Comment Service**: Redis (cache para threading)
- **Vote Service**: Redis (cache para calculations)
- **Notification Service**: Spring WebSocket, SendGrid (email)

🅰 ARQUITECTURA FRONTEND - ANGULAR

🎯 Tecnologías Frontend

- **Angular 17+** (Standalone Components + Signals)
- **Angular Material** (UI Components)
- **Tailwind CSS** (Utility-first CSS)
- **NgRx** (State Management)
- **RxJS** (Reactive Programming)
- **TypeScript** (Strict mode)

📁 Estructura del Proyecto Angular

```
reddit-clone-frontend/
├── src/
│   ├── app/
│   │   ├── app.component.ts
│   │   ├── app.component.html
│   │   ├── app.component.scss
│   │   ├── app.config.ts
│   │   ├── app.routes.ts
│   │   │
│   │   ├── core/ ## 🔧 Core Module
│   │   │   ├── guards/
│   │   │   │   ├── auth.guard.ts
│   │   │   │   ├── guest.guard.ts
│   │   │   │   └── role.guard.ts
│   │   │   ├── interceptors/
│   │   │   │   ├── auth.interceptor.ts
│   │   │   │   ├── error.interceptor.ts
│   │   │   │   ├── loading.interceptor.ts
│   │   │   │   └── retry.interceptor.ts
│   │   │   ├── services/
│   │   │   │   ├── api.service.ts
│   │   │   │   ├── auth.service.ts
│   │   │   │   ├── loading.service.ts
│   │   │   │   ├── error.service.ts
│   │   │   │   ├── theme.service.ts
│   │   │   │   ├── websocket.service.ts
│   │   │   │   └── storage.service.ts
│   │   │   ├── models/
│   │   │   │   ├── api-response.interface.ts
│   │   │   │   ├── user.interface.ts
│   │   │   │   ├── auth.interface.ts
│   │   │   │   ├── pagination.interface.ts
│   │   │   │   └── error.interface.ts
│   │   │   └── utils/
│   │   │       ├── validators.ts
│   │   │       ├── date.utils.ts
│   │   │       ├── url.utils.ts
│   │   │       └── karma.utils.ts
│   │   │
│   │   ├── shared/ ## 🔄 Shared Module
│   │   │   ├── components/
│   │   │   │   ├── header/
│   │   │   │   │   ├── header.component.ts
│   │   │   │   │   ├── header.component.html
│   │   │   │   │   ├── header.component.scss
│   │   │   │   │   └── profile-dropdown/
│   │   │   │   │       ├── profile-dropdown.component.ts
│   │   │   │   │       ├── profile-dropdown.component.html
│   │   │   │   │       └── profile-dropdown.component.scss
│   │   │   │   ├── sidebar/
│   │   │   │   │   ├── sidebar.component.ts
│   │   │   │   │   ├── sidebar.component.html
│   │   │   │   │   ├── sidebar.component.scss
│   │   │   │   │   ├── community-list/
│   │   │   │   │   │   ├── community-list.component.ts
│   │   │   │   │   │   ├── community-list.component.html
│   │   │   │   │   │   └── community-list.component.scss
│   │   │   │   │   └── trending-topics/
│   │   │   │   │       ├── trending-topics.component.ts
│   │   │   │   │       ├── trending-topics.component.html
│   │   │   │   │       └── trending-topics.component.scss
│   │   │   │   ├── post-card/
│   │   │   │   │   ├── post-card.component.ts
│   │   │   │   │   ├── post-card.component.html
│   │   │   │   │   ├── post-card.component.scss
│   │   │   │   │   └── vote-buttons/
│   │   │   │   │       ├── vote-buttons.component.ts
│   │   │   │   │       ├── vote-buttons.component.html
│   │   │   │   │       └── vote-buttons.component.scss
│   │   │   │   ├── comment-tree/
│   │   │   │   │   ├── comment-tree.component.ts
│   │   │   │   │   ├── comment-tree.component.html
│   │   │   │   │   ├── comment-tree.component.scss
│   │   │   │   │   └── comment-item/
│   │   │   │   │       ├── comment-item.component.ts
│   │   │   │   │       ├── comment-item.component.html
│   │   │   │   │       └── comment-item.component.scss
│   │   │   │   ├── loading-spinner/
│   │   │   │   │   ├── loading-spinner.component.ts
│   │   │   │   │   ├── loading-spinner.component.html
│   │   │   │   │   └── loading-spinner.component.scss
│   │   │   │   ├── error-message/
│   │   │   │   │   ├── error-message.component.ts
│   │   │   │   │   ├── error-message.component.html
│   │   │   │   │   └── error-message.component.scss
│   │   │   │   ├── infinite-scroll/
│   │   │   │   │   ├── infinite-scroll.component.ts
│   │   │   │   │   ├── infinite-scroll.component.html
│   │   │   │   │   └── infinite-scroll.component.scss
│   │   │   │   ├── rich-text-editor/
│   │   │   │   │   ├── rich-text-editor.component.ts
│   │   │   │   │   ├── rich-text-editor.component.html
│   │   │   │   │   └── rich-text-editor.component.scss
│   │   │   │   └── image-upload/
│   │   │   │       ├── image-upload.component.ts
│   │   │   │       ├── image-upload.component.html
│   │   │   │       └── image-upload.component.scss
│   │   │   ├── directives/
│   │   │   │   ├── lazy-load.directive.ts
│   │   │   │   ├── auto-resize.directive.ts
│   │   │   │   ├── click-outside.directive.ts
│   │   │   │   └── highlight.directive.ts
│   │   │   ├── pipes/
│   │   │   │   ├── time-ago.pipe.ts
│   │   │   │   ├── karma-format.pipe.ts
│   │   │   │   ├── truncate.pipe.ts
│   │   │   │   ├── safe-html.pipe.ts
│   │   │   │   └── member-count.pipe.ts
│   │   │   └── layouts/
│   │   │       ├── main-layout/
│   │   │       │   ├── main-layout.component.ts
│   │   │       │   ├── main-layout.component.html
│   │   │       │   └── main-layout.component.scss
│   │   │       └── auth-layout/
│   │   │           ├── auth-layout.component.ts
│   │   │           ├── auth-layout.component.html
│   │   │           └── auth-layout.component.scss
│   │   │
│   │   ├── features/ ## 🎯 Feature Modules
│   │   │   │
│   │   │   ├── auth/ ## 🔐 Auth Module
│   │   │   │   ├── components/
│   │   │   │   │   ├── login/
│   │   │   │   │   │   ├── login.component.ts
│   │   │   │   │   │   ├── login.component.html
│   │   │   │   │   │   └── login.component.scss
│   │   │   │   │   ├── signup/
│   │   │   │   │   │   ├── signup.component.ts
│   │   │   │   │   │   ├── signup.component.html
│   │   │   │   │   │   └── signup.component.scss
│   │   │   │   │   ├── forgot-password/
│   │   │   │   │   │   ├── forgot-password.component.ts
│   │   │   │   │   │   ├── forgot-password.component.html
│   │   │   │   │   │   └── forgot-password.component.scss
│   │   │   │   │   └── oauth-callback/
│   │   │   │   │       ├── oauth-callback.component.ts
│   │   │   │   │       ├── oauth-callback.component.html
│   │   │   │   │       └── oauth-callback.component.scss
│   │   │   │   ├── services/
│   │   │   │   │   └── auth-api.service.ts
│   │   │   │   ├── models/
│   │   │   │   │   ├── login.interface.ts
│   │   │   │   │   ├── signup.interface.ts
│   │   │   │   │   └── auth-response.interface.ts
│   │   │   │   └── auth.routes.ts
│   │   │   │
│   │   │   ├── home/ ## 🏠 Home Module
│   │   │   │   ├── components/
│   │   │   │   │   ├── home-feed/
│   │   │   │   │   │   ├── home-feed.component.ts
│   │   │   │   │   │   ├── home-feed.component.html
│   │   │   │   │   │   └── home-feed.component.scss
│   │   │   │   │   ├── create-post-widget/
│   │   │   │   │   │   ├── create-post-widget.component.ts
│   │   │   │   │   │   ├── create-post-widget.component.html
│   │   │   │   │   │   └── create-post-widget.component.scss
│   │   │   │   │   ├── post-list/
│   │   │   │   │   │   ├── post-list.component.ts
│   │   │   │   │   │   ├── post-list.component.html
│   │   │   │   │   │   └── post-list.component.scss
│   │   │   │   │   └── feed-filters/
│   │   │   │   │       ├── feed-filters.component.ts
│   │   │   │   │       ├── feed-filters.component.html
│   │   │   │   │       └── feed-filters.component.scss
│   │   │   │   ├── services/
│   │   │   │   │   └── home-api.service.ts
│   │   │   │   └── home.routes.ts
│   │   │   │
│   │   │   ├── post/ ## 📝 Post Module
│   │   │   │   ├── components/
│   │   │   │   │   ├── post-detail/
│   │   │   │   │   │   ├── post-detail.component.ts
│   │   │   │   │   │   ├── post-detail.component.html
│   │   │   │   │   │   └── post-detail.component.scss
│   │   │   │   │   ├── create-post/
│   │   │   │   │   │   ├── create-post.component.ts
│   │   │   │   │   │   ├── create-post.component.html
│   │   │   │   │   │   ├── create-post.component.scss
│   │   │   │   │   │   ├── post-type-selector/
│   │   │   │   │   │   │   ├── post-type-selector.component.ts
│   │   │   │   │   │   │   ├── post-type-selector.component.html
│   │   │   │   │   │   │   └── post-type-selector.component.scss
│   │   │   │   │   │   ├── community-selector/
│   │   │   │   │   │   │   ├── community-selector.component.ts
│   │   │   │   │   │   │   ├── community-selector.component.html
│   │   │   │   │   │   │   └── community-selector.component.scss
│   │   │   │   │   │   └── tag-selector/
│   │   │   │   │   │       ├── tag-selector.component.ts
│   │   │   │   │   │       ├── tag-selector.component.html
│   │   │   │   │   │       └── tag-selector.component.scss
│   │   │   │   │   ├── post-actions/
│   │   │   │   │   │   ├── post-actions.component.ts
│   │   │   │   │   │   ├── post-actions.component.html
│   │   │   │   │   │   └── post-actions.component.scss
│   │   │   │   │   └── post-content/
│   │   │   │   │       ├── post-content.component.ts
│   │   │   │   │       ├── post-content.component.html
│   │   │   │   │       └── post-content.component.scss
│   │   │   │   ├── services/
│   │   │   │   │   └── post-api.service.ts
│   │   │   │   ├── models/
│   │   │   │   │   ├── post.interface.ts
│   │   │   │   │   ├── create-post.interface.ts
│   │   │   │   │   └── post-flair.interface.ts
│   │   │   │   └── post.routes.ts
│   │   │   │
│   │   │   ├── community/ ## 🏘️ Community Module
│   │   │   │   ├── components/
│   │   │   │   │   ├── community-page/
│   │   │   │   │   │   ├── community-page.component.ts
│   │   │   │   │   │   ├── community-page.component.html
│   │   │   │   │   │   └── community-page.component.scss
│   │   │   │   │   ├── community-header/
│   │   │   │   │   │   ├── community-header.component.ts
│   │   │   │   │   │   ├── community-header.component.html
│   │   │   │   │   │   └── community-header.component.scss
│   │   │   │   │   ├── community-nav/
│   │   │   │   │   │   ├── community-nav.component.ts
│   │   │   │   │   │   ├── community-nav.component.html
│   │   │   │   │   │   └── community-nav.component.scss
│   │   │   │   │   ├── community-posts/
│   │   │   │   │   │   ├── community-posts.component.ts
│   │   │   │   │   │   ├── community-posts.component.html
│   │   │   │   │   │   └── community-posts.component.scss
│   │   │   │   │   ├── community-about/
│   │   │   │   │   │   ├── community-about.component.ts
│   │   │   │   │   │   ├── community-about.component.html
│   │   │   │   │   │   └── community-about.component.scss
│   │   │   │   │   ├── community-rules/
│   │   │   │   │   │   ├── community-rules.component.ts
│   │   │   │   │   │   ├── community-rules.component.html
│   │   │   │   │   │   └── community-rules.component.scss
│   │   │   │   │   ├── community-moderators/
│   │   │   │   │   │   ├── community-moderators.component.ts
│   │   │   │   │   │   ├── community-moderators.component.html
│   │   │   │   │   │   └── community-moderators.component.scss
│   │   │   │   │   ├── community-sidebar/
│   │   │   │   │   │   ├── community-sidebar.component.ts
│   │   │   │   │   │   ├── community-sidebar.component.html
│   │   │   │   │   │   └── community-sidebar.component.scss
│   │   │   │   │   └── create-community/
│   │   │   │   │       ├── create-community.component.ts
│   │   │   │   │       ├── create-community.component.html
│   │   │   │   │       └── create-community.component.scss
│   │   │   │   ├── services/
│   │   │   │   │   └── community-api.service.ts
│   │   │   │   ├── models/
│   │   │   │   │   ├── community.interface.ts
│   │   │   │   │   ├── community-stats.interface.ts
│   │   │   │   │   ├── community-rule.interface.ts
│   │   │   │   │   └── moderator.interface.ts
│   │   │   │   └── community.routes.ts
│   │   │   │
│   │   │   ├── user/ ## 🧑🏿 User Module
│   │   │   │   ├── components/
│   │   │   │   │   ├── user-profile/
│   │   │   │   │   │   ├── user-profile.component.ts
│   │   │   │   │   │   ├── user-profile.component.html
│   │   │   │   │   │   └── user-profile.component.scss
│   │   │   │   │   ├── profile-header/
│   │   │   │   │   │   ├── profile-header.component.ts
│   │   │   │   │   │   ├── profile-header.component.html
│   │   │   │   │   │   └── profile-header.component.scss
│   │   │   │   │   ├── profile-nav/
│   │   │   │   │   │   ├── profile-nav.component.ts
│   │   │   │   │   │   ├── profile-nav.component.html
│   │   │   │   │   │   └── profile-nav.component.scss
│   │   │   │   │   ├── profile-posts/
│   │   │   │   │   │   ├── profile-posts.component.ts
│   │   │   │   │   │   ├── profile-posts.component.html
│   │   │   │   │   │   └── profile-posts.component.scss
│   │   │   │   │   ├── profile-comments/
│   │   │   │   │   │   ├── profile-comments.component.ts
│   │   │   │   │   │   ├── profile-comments.component.html
│   │   │   │   │   │   └── profile-comments.component.scss
│   │   │   │   │   ├── profile-saved/
│   │   │   │   │   │   ├── profile-saved.component.ts
│   │   │   │   │   │   ├── profile-saved.component.html
│   │   │   │   │   │   └── profile-saved.component.scss
│   │   │   │   │   ├── profile-awards/
│   │   │   │   │   │   ├── profile-awards.component.ts
│   │   │   │   │   │   ├── profile-awards.component.html
│   │   │   │   │   │   └── profile-awards.component.scss
│   │   │   │   │   ├── profile-stats/
│   │   │   │   │   │   ├── profile-stats.component.ts
│   │   │   │   │   │   ├── profile-stats.component.html
│   │   │   │   │   │   └── profile-stats.component.scss
│   │   │   │   │   └── profile-sidebar/
│   │   │   │   │       ├── profile-sidebar.component.ts
│   │   │   │   │       ├── profile-sidebar.component.html
│   │   │   │   │       └── profile-sidebar.component.scss
│   │   │   │   ├── services/
│   │   │   │   │   └── user-api.service.ts
│   │   │   │   ├── models/
│   │   │   │   │   ├── user-profile.interface.ts
│   │   │   │   │   ├── user-stats.interface.ts
│   │   │   │   │   └── user-award.interface.ts
│   │   │   │   └── user.routes.ts
│   │   │   │
│   │   │   ├── settings/ ## ⚙️ Settings Module
│   │   │   │   ├── components/
│   │   │   │   │   ├── settings-page/
│   │   │   │   │   │   ├── settings-page.component.ts
│   │   │   │   │   │   ├── settings-page.component.html
│   │   │   │   │   │   └── settings-page.component.scss
│   │   │   │   │   ├── settings-sidebar/
│   │   │   │   │   │   ├── settings-sidebar.component.ts
│   │   │   │   │   │   ├── settings-sidebar.component.html
│   │   │   │   │   │   └── settings-sidebar.component.scss
│   │   │   │   │   ├── account-settings/
│   │   │   │   │   │   ├── account-settings.component.ts
│   │   │   │   │   │   ├── account-settings.component.html
│   │   │   │   │   │   └── account-settings.component.scss
│   │   │   │   │   ├── privacy-settings/
│   │   │   │   │   │   ├── privacy-settings.component.ts
│   │   │   │   │   │   ├── privacy-settings.component.html
│   │   │   │   │   │   └── privacy-settings.component.scss
│   │   │   │   │   ├── notification-settings/
│   │   │   │   │   │   ├── notification-settings.component.ts
│   │   │   │   │   │   ├── notification-settings.component.html
│   │   │   │   │   │   └── notification-settings.component.scss
│   │   │   │   │   ├── theme-settings/
│   │   │   │   │   │   ├── theme-settings.component.ts
│   │   │   │   │   │   ├── theme-settings.component.html
│   │   │   │   │   │   └── theme-settings.component.scss
│   │   │   │   │   └── avatar-editor/
│   │   │   │   │       ├── avatar-editor.component.ts
│   │   │   │   │       ├── avatar-editor.component.html
│   │   │   │   │       └── avatar-editor.component.scss
│   │   │   │   ├── services/
│   │   │   │   │   └── settings-api.service.ts
│   │   │   │   ├── models/
│   │   │   │   │   ├── user-settings.interface.ts
│   │   │   │   │   ├── notification-settings.interface.ts
│   │   │   │   │   └── privacy-settings.interface.ts
│   │   │   │   └── settings.routes.ts
│   │   │   │
│   │   │   └── comment/ ## 💬 Comment Module
│   │   │       ├── components/
│   │   │       │   ├── comment-section/
│   │   │       │   │   ├── comment-section.component.ts
│   │   │       │   │   ├── comment-section.component.html
│   │   │       │   │   └── comment-section.component.scss
│   │   │       │   ├── comment-form/
│   │   │       │   │   ├── comment-form.component.ts
│   │   │       │   │   ├── comment-form.component.html
│   │   │       │   │   └── comment-form.component.scss
│   │   │       │   └── comment-sort/
│   │   │       │       ├── comment-sort.component.ts
│   │   │       │       ├── comment-sort.component.html
│   │   │       │       └── comment-sort.component.scss
│   │   │       ├── services/
│   │   │       │   └── comment-api.service.ts
│   │   │       ├── models/
│   │   │       │   ├── comment.interface.ts
│   │   │       │   └── comment-tree.interface.ts
│   │   │       └── comment.routes.ts
│   │   │
│   │   ├── store/ ## 🗄 NgRx Store
│   │   │   ├── auth/
│   │   │   │   ├── auth.actions.ts
│   │   │   │   ├── auth.reducer.ts
│   │   │   │   ├── auth.selectors.ts
│   │   │   │   ├── auth.effects.ts
│   │   │   │   └── auth.state.ts
│   │   │   ├── user/
│   │   │   │   ├── user.actions.ts
│   │   │   │   ├── user.reducer.ts
│   │   │   │   ├── user.selectors.ts
│   │   │   │   ├── user.effects.ts
│   │   │   │   └── user.state.ts
│   │   │   ├── post/
│   │   │   │   ├── post.actions.ts
│   │   │   │   ├── post.reducer.ts
│   │   │   │   ├── post.selectors.ts
│   │   │   │   ├── post.effects.ts
│   │   │   │   └── post.state.ts
│   │   │   ├── community/
│   │   │   │   ├── community.actions.ts
│   │   │   │   ├── community.reducer.ts
│   │   │   │   ├── community.selectors.ts
│   │   │   │   ├── community.effects.ts
│   │   │   │   └── community.state.ts
│   │   │   ├── comment/
│   │   │   │   ├── comment.actions.ts
│   │   │   │   ├── comment.reducer.ts
│   │   │   │   ├── comment.selectors.ts
│   │   │   │   ├── comment.effects.ts
│   │   │   │   └── comment.state.ts
│   │   │   ├── ui/
│   │   │   │   ├── ui.actions.ts
│   │   │   │   ├── ui.reducer.ts
│   │   │   │   ├── ui.selectors.ts
│   │   │   │   └── ui.state.ts
│   │   │   ├── app.reducer.ts
│   │   │   └── app.state.ts
│   │   │
│   │   └── environments/
│   │       ├── environment.ts
│   │       ├── environment.prod.ts
│   │       └── environment.staging.ts
│   │
│   ├── assets/
│   │   ├── icons/
│   │   ├── images/
│   │   ├── fonts/
│   │   └── scss/
│   │       ├── _variables.scss
│   │       ├── _mixins.scss
│   │       ├── _themes.scss
│   │       └── main.scss
│   │
│   ├── styles.scss
│   ├── main.ts
│   └── index.html
│
├── angular.json
├── package.json
├── tailwind.config.js
├── tsconfig.json
├── tsconfig.app.json
└── README.md
```

📨 Estructura de Routing

Main Routes (app.routes.ts)

typescript

```
export const routes: Routes = [
  {
    path: '',
    component: MainLayoutComponent,
    children: [
      { path: '', redirectTo: '/home', pathMatch: 'full' },
      { path: 'home', loadChildren: () => import('./features/home/home.routes') },
      { path: 'r/:communityName', loadChildren: () => import('./features/community/community.routes') },
      { path: 'user/:username', loadChildren: () => import('./features/user/user.routes') },
      { path: 'post/:id', loadChildren: () => import('./features/post/post.routes') },
      { path: 'submit', loadChildren: () => import('./features/post/post.routes') },
      { path: 'settings', loadChildren: () => import('./features/settings/settings.routes'), canActivate: [AuthGuard] },
    ]
  },
  {
    path: 'auth',
    component: AuthLayoutComponent,
    children: [
      { path: '', loadChildren: () => import('./features/auth/auth.routes') }
    ]
  },
  { path: '**', redirectTo: '/home' }
];
```

Community Routes (community.routes.ts)

typescript

```typescript
export const COMMUNITY_ROUTES: Routes = [
  {
    path: '',
    component: CommunityPageComponent,
    children: [
      { path: '', redirectTo: 'posts', pathMatch: 'full' },
      { path: 'posts', component: CommunityPostsComponent },
      { path: 'about', component: CommunityAboutComponent },
      { path: 'rules', component: CommunityRulesComponent },
      { path: 'moderators', component: CommunityModeratorsComponent }
    ]
  }
];
```

User Profile Routes (user.routes.ts)

typescript

```typescript
export const USER_ROUTES: Routes = [
  {
    path: '',
    component: UserProfileComponent,
    children: [
      { path: '', redirectTo: 'posts', pathMatch: 'full' },
      { path: 'posts', component: ProfilePostsComponent },
      { path: 'comments', component: ProfileCommentsComponent },
      { path: 'saved', component: ProfileSavedComponent, canActivate: [AuthGuard] },
      { path: 'awards', component: ProfileAwardsComponent }
    ]
  }
];
```

Settings Routes (settings.routes.ts)

typescript

```typescript
export const SETTINGS_ROUTES: Routes = [
  {
    path: '',
    component: SettingsPageComponent,
    children: [
      { path: '', redirectTo: 'account', pathMatch: 'full' },
      { path: 'account', component: AccountSettingsComponent },
      { path: 'privacy', component: PrivacySettingsComponent },
      { path: 'notifications', component: NotificationSettingsComponent },
      { path: 'theme', component: ThemeSettingsComponent }
    ]
  }
];
```

🎨 Mapeo de Componentes a Diseños SVG

1. Home Feed Component

SVG Plantilla: `reddit_home_design`

typescript

```typescript
@Component({
  selector: 'app-home-feed',
  standalone: true,
  imports: [CommonModule, PostListComponent, CreatePostWidgetComponent, SidebarComponent],
  template: `
    <div class="flex min-h-screen bg-gray-900">
      <!-- Sidebar Left -->
      <app-sidebar class="w-64 fixed left-0 top-16"></app-sidebar>

      <!-- Main Content -->
      <main class="flex-1 ml-64 mr-80 p-4">
        <app-create-post-widget class="mb-4"></app-create-post-widget>
        <app-post-list [posts]="posts$ | async"></app-post-list>
      </main>

      <!-- Sidebar Right -->
      <aside class="w-80 fixed right-0 top-16 p-4">
        <app-trending-topics></app-trending-topics>
      </aside>
    </div>
  `
})
```

2. Post Detail Component

SVG Plantilla: `reddit_post_detail`

typescript

```typescript
@Component({
  selector: 'app-post-detail',
  standalone: true,
  imports: [CommonModule, PostContentComponent, CommentSectionComponent, VoteButtonsComponent],
  template: `
    <div class="flex min-h-screen bg-gray-900">
      <main class="flex-1 max-w-4xl mx-auto p-4">
        <!-- Breadcrumb -->
        <nav class="mb-4 text-gray-400">
          <a [routerLink]="['/r', post.communityName]">r/{{post.communityName}}</a> > {{post.title}}
        </nav>

        <!-- Post Content -->
        <article class="bg-gray-800 rounded-lg p-6 mb-6">
          <div class="flex">
            <app-vote-buttons [targetId]="post.id" [voteCount]="post.voteCount"></app-vote-buttons>
            <app-post-content [post]="post" [showFullContent]="true"></app-post-content>
          </div>
        </article>

        <!-- Comments -->
        <app-comment-section [postId]="post.id"></app-comment-section>
      </main>

      <!-- Sidebar -->
      <aside class="w-80 p-4">
        <app-community-sidebar [communityName]="post.communityName"></app-community-sidebar>
      </aside>
    </div>
  `
})
```

3. Create Post Component

SVG Plantilla: `reddit_create_post`

typescript

```
@Component({
  selector: 'app-create-post',
  standalone: true,
  imports: [CommonModule, ReactiveFormsModule, RichTextEditorComponent, CommunitySelector],
  template: `
    <div class="min-h-screen bg-gray-900">
      <header class="bg-gray-900 border-b border-gray-700 p-4">
        <h1 class="text-2xl font-bold text-white">Create a post</h1>
      </header>

      <div class="flex max-w-6xl mx-auto p-4 gap-6">
        <main class="flex-1">
          <form [formGroup]="postForm" (ngSubmit)="onSubmit()" class="bg-gray-800 rounded-lg p-6">
            <!-- Community Selection -->
            <app-community-selector formControlName="communityId"></app-community-selector>

            <!-- Post Type Tabs -->
            <div class="flex gap-2 my-4">
              <button type="button" [class.bg-blue-600]="postType === 'text'" class="px-4 py-2 rounded-lg border">📝 P
              <button type="button" [class.bg-blue-600]="postType === 'image'" class="px-4 py-2 rounded-lg border">🖼
              <button type="button" [class.bg-blue-600]="postType === 'link'" class="px-4 py-2 rounded-lg border">🔗 Li
            </div>

            <!-- Title Input -->
            <input formControlName="title" placeholder="Title" class="w-full p-3 bg-gray-700 rounded-lg mb-4">

            <!-- Content Editor -->
            <app-rich-text-editor formControlName="content"></app-rich-text-editor>

            <!-- Tags -->
            <app-tag-selector formControlName="tags"></app-tag-selector>

            <!-- Submit -->
            <div class="flex justify-end gap-4 mt-6">
              <button type="button" class="px-6 py-2 border border-gray-600 rounded-lg">Save Draft</button>
              <button type="submit" class="px-6 py-2 bg-blue-600 text-white rounded-lg">Post</button>
            </div>
          </form>
        </main>

        <!-- Sidebar with posting guidelines -->
        <aside class="w-80">
          <div class="bg-gray-800 rounded-lg p-4">
            <h3 class="font-bold text-white mb-4">Posting to Reddit</h3>
            <!-- Guidelines content -->
          </div>
        </aside>
      </div>
    </div>
  `
})
```

4. Login Component

SVG Plantilla: reddit_login_page

typescript

```
@Component({
  selector: 'app-login',
  standalone: true,
  imports: [CommonModule, ReactiveFormsModule],
  template: `
    <div class="min-h-screen bg-black flex">
      <!-- Left side with features -->
      <div class="flex-1 flex flex-col justify-center items-center p-8">
        <h1 class="text-4xl font-bold text-orange-500 mb-4">Join the conversation</h1>
        <p class="text-gray-300 text-center mb-8">Connect with communities around your interests</p>

        <div class="space-y-4 w-full max-w-md">
          <div class="bg-gray-800 rounded-lg p-4 flex items-center">
            <div class="w-8 h-8 bg-green-500 rounded-full mr-3"></div>
            <div>
              <p class="text-white font-medium">Share your knowledge</p>
              <p class="text-gray-400 text-sm">Post and comment in communities</p>
            </div>
          </div>
          <!-- More features... -->
        </div>
      </div>

      <!-- Right side with login form -->
      <div class="flex items-center justify-center p-8">
        <div class="w-full max-w-md bg-gray-800 rounded-2xl p-8">
          <div class="text-center mb-6">
            <div class="w-16 h-16 bg-orange-500 rounded-full mx-auto mb-4 flex items-center justify-center">
              <span class="text-white text-2xl font-bold">R</span>
            </div>
            <h2 class="text-2xl font-bold text-white">Welcome back</h2>
            <p class="text-gray-400">Sign in to your account</p>
          </div>

          <form [formGroup]="loginForm" (ngSubmit)="onSubmit()">
            <div class="space-y-4">
              <input formControlName="email" type="email" placeholder="Email or Username" class="w-full p-3 bg-gray-7
              <input formControlName="password" type="password" placeholder="Password" class="w-full p-3 bg-gray-7(

              <div class="flex items-center justify-between">
                <label class="flex items-center text-gray-300">
                  <input type="checkbox" class="mr-2">
                  Remember me
                </label>
                <a href="#" class="text-blue-400 text-sm">Forgot password?</a>
              </div>
          <button type="submit" class="w-full bg-blue-600 text-white py-3 rounded-lg font-medium">Log In</button>

          <div class="text-center">
            <span class="text-gray-400">OR</span>
          </div>

          <button type="button" class="w-full border border-gray-600 text-white py-3 rounded-lg flex items-center just
            <span class="mr-2">G</span> Continue with Google
          </button>
        </div>
      </form>

      <p class="text-center text-gray-400 mt-6">
        Don't have an account?
        <a routerLink="/auth/signup" class="text-blue-400 font-medium">Sign Up</a>
      </p>
    </div>
  </div>
</div>
`
})
```

🔧 Servicios Principales del Frontend

Auth Service

typescript

```typescript
@Injectable({ providedIn: 'root' })
export class AuthService {
  private readonly API_URL = environment.apiUrl;
  private currentUserSubject = new BehaviorSubject<User | null>(null);
  public currentUser$ = this.currentUserSubject.asObservable();

  constructor(private http: HttpClient, private router: Router) {
    // Check for stored user on app init
    const storedUser = localStorage.getItem('currentUser');
    if (storedUser) {
      this.currentUserSubject.next(JSON.parse(storedUser));
    }
  }

  login(credentials: LoginRequest): Observable<AuthResponse> {
    return this.http.post<AuthResponse>(`${this.API_URL}/auth/login`, credentials)
      .pipe(
        tap(response => {
          localStorage.setItem('token', response.token);
          localStorage.setItem('currentUser', JSON.stringify(response.user));
          this.currentUserSubject.next(response.user);
        })
      );
  }

  signup(userData: SignupRequest): Observable<AuthResponse> {
    return this.http.post<AuthResponse>(`${this.API_URL}/auth/signup`, userData);
  }

  logout(): void {
    localStorage.removeItem('token');
    localStorage.removeItem('currentUser');
    this.currentUserSubject.next(null);
    this.router.navigate(['/auth/login']);
  }

  isAuthenticated(): boolean {
    return !!localStorage.getItem('token');
  }

  getCurrentUser(): User | null {
    return this.currentUserSubject.value;
  }
}
```

API Service

typescript

```typescript
@Injectable({ providedIn: 'root' })
export class ApiService {
  private readonly API_URL = environment.apiUrl;

  constructor(private http: HttpClient) {}

  // Generic HTTP methods
  get<T>(endpoint: string, params?: HttpParams): Observable<T> {
    return this.http.get<T>(`${this.API_URL}${endpoint}`, { params });
  }

  post<T>(endpoint: string, body: any): Observable<T> {
    return this.http.post<T>(`${this.API_URL}${endpoint}`, body);
  }

  put<T>(endpoint: string, body: any): Observable<T> {
    return this.http.put<T>(`${this.API_URL}${endpoint}`, body);
  }

  delete<T>(endpoint: string): Observable<T> {
    return this.http.delete<T>(`${this.API_URL}${endpoint}`);
  }

  // Specific API methods
  getPosts(page: number = 0, size: number = 10): Observable<PagedResponse<Post>> {
    const params = new HttpParams()
      .set('page', page.toString())
      .set('size', size.toString());
    return this.get<PagedResponse<Post>>('/posts', params);
  }

  getPost(id: string): Observable<Post> {
    return this.get<Post>(`/posts/${id}`);
  }

  createPost(post: CreatePostRequest): Observable<Post> {
    return this.post<Post>('/posts', post);
  }

  // Community methods
  getCommunity(name: string): Observable<Community> {
    return this.get<Community>(`/communities/${name}`);
  }

  getCommunityPosts(name: string, page: number = 0): Observable<PagedResponse<Post>> {
    const params = new HttpParams()
```

```
      .set('page', page.toString())
      .set('size', '10');
    return this.get<PagedResponse<Post>>(`/communities/${name}/posts`, params);
  }

  // User methods
  getUserProfile(username: string): Observable<UserProfile> {
    return this.get<UserProfile>(`/users/${username}`);
  }

  getUserPosts(username: string, page: number = 0): Observable<PagedResponse<Post>> {
    const params = new HttpParams()
      .set('page', page.toString())
      .set('size', '10');
    return this.get<PagedResponse<Post>>(`/users/${username}/posts`, params);
  }

  // Vote methods
  vote(targetId: string, targetType: 'post' | 'comment', voteType: 'upvote' | 'downvote'): Observable<VoteResponse> {
    return this.post<VoteResponse>('/votes', { targetId, targetType, voteType });
  }

  // Comment methods
  getComments(postId: string): Observable<Comment[]> {
    return this.get<Comment[]>(`/posts/${postId}/comments`);
  }

  createComment(comment: CreateCommentRequest): Observable<Comment> {
    return this.post<Comment>('/comments', comment);
  }
}
```

🎨 Configuración de Tailwind CSS

tailwind.config.js

javascript

```javascript
module.exports = {
  content: ['./src/**/*.{html,ts}'],
  darkMode: 'class',
  theme: {
    extend: {
      colors: {
        reddit: {
          orange: '#ff4500',
          blue: '#0079d3',
          dark: '#1a1a1b',
          gray: {
            100: '#f8f9fa',
            200: '#edeff1',
            300: '#d7dadc',
            400: '#878a8c',
            500: '#818384',
            600: '#343536',
            700: '#272729',
            800: '#1a1a1b',
            900: '#030303'
          }
        }
      },
      fontFamily: {
        sans: ['Inter', 'system-ui', 'sans-serif']
      },
      spacing: {
        '18': '4.5rem',
        '88': '22rem'
      }
    }
  },
  plugins: [
    require('@tailwindcss/forms'),
    require('@tailwindcss/typography')
  ]
}
```

Responsive Design Strategy

Breakpoints:

scss

```scss
// Mobile-first approach
$mobile: 640px; // sm
$tablet: 768px; // md
$desktop: 1024px; // lg
$large: 1280px; // xl

// Layout adjustments
@media (max-width: $tablet) {
  .sidebar { display: none; }
  .main-content { margin: 0; width: 100%; }
  .mobile-nav { display: block; }
}

@media (max-width: $mobile) {
  .post-card { padding: 1rem; }
  .header { height: 3rem; }
  .vote-buttons {
    flex-direction: row;
    gap: 0.5rem;
  }
}
```

🗄 State Management con NgRx

App State Structure

typescript

```typescript
export interface AppState {
  auth: AuthState;
  user: UserState;
  posts: PostState;
  communities: CommunityState;
  comments: CommentState;
  ui: UIState;
}

export interface AuthState {
  user: User | null;
  token: string | null;
  isAuthenticated: boolean;
  loading: boolean;
  error: string | null;
}

export interface PostState {
  posts: Post[];
  currentPost: Post | null;
  loading: boolean;
  error: string | null;
  pagination: {
    page: number;
    totalPages: number;
    hasMore: boolean;
  };
}
```

🧪 Estrategia de Testing

Component Testing Example

typescript

```typescript
describe('PostCardComponent', () => {
 let component: PostCardComponent;
 let fixture: ComponentFixture<PostCardComponent>;

 beforeEach(async () => {
  await TestBed.configureTestingModule({
   imports: [PostCardComponent, HttpClientTestingModule],
   providers: [provideMockStore()]
  }).compileComponents();

  fixture = TestBed.createComponent(PostCardComponent);
  component = fixture.componentInstance;
 });

 it('should display post content correctly', () => {
  const mockPost: Post = {
   id: '1',
   title: 'Test Post',
   content: 'Test Content',
   voteCount: 10,
   // ... more properties
  };

  component.post = mockPost;
  fixture.detectChanges();

  expect(fixture.nativeElement.querySelector('.post-title').textContent).toBe('Test Post');
 });
});
```

🚀 PLAN DE DESARROLLO INTEGRADO

📋 Estrategia de Desarrollo

- **Desarrollo en paralelo**: Backend + Frontend por funcionalidad
- **Pruebas continuas**: Integración después de cada fase
- **Iterativo e incremental**: Validar antes de continuar
- **MVP primero**: Funcionalidad básica funcionando end-to-end

🏗️ FASE 1: Infraestructura Base (Semana 1)

Backend - Fundación

Objetivo: Tener la infraestructura básica funcionando

1.1 Setup Inicial (Días 1-2)

- Eureka Server (Puerto 8761)
  - Configuración básica de Service Discovery
  - Docker compose para bases de datos Oracle
  - Configuración de red entre servicios
- API Gateway (Puerto 8080)
  - Spring Cloud Gateway básico
  - Configuración de CORS
  - Health checks
  - Routing básico preparado

1.2 Base de Datos (Días 3-4)

- Docker Compose para todas las BDs Oracle
- Scripts de creación de esquemas
- Configuración de conexiones
- Flyway setup para migraciones

Frontend - Setup

Objetivo: Estructura base de Angular funcionando

1.3 Proyecto Angular (Días 4-5)

- Setup inicial
  - Angular 17+ con standalone components
  - Tailwind CSS configurado
  - NgRx store setup básico
  - Estructura de carpetas según arquitectura
- Layouts básicos
  - Main layout component
  - Auth layout component
  - Header component básico
  - Routing principal configurado

1.4 Prueba de Integración

- Verificación: Angular puede conectar a API Gateway

- Health check: Endpoint básico funcionando
- CORS: Verificar comunicación frontend-backend

🔐 FASE 2: Autenticación Completa (Semana 2)

Backend - Auth Service

Objetivo: Sistema de autenticación JWT funcionando

2.1 Auth Service (Días 1-3)

- Microservicio Auth (Puerto 8081)
  - Base de datos auth_db con tablas de usuarios
  - Endpoints: `/api/auth/login`, `/api/auth/signup`
  - JWT token generation y validation
  - OAuth2 con Google (básico)
  - Password reset functionality
- API Gateway Integration
  - Routing para `/api/auth/**`
  - Security configuration

2.2 User Service Básico (Días 3-4)

- Microservicio User (Puerto 8082)
  - Base de datos user_db con perfiles básicos
  - Endpoint: `/api/users/profile` (obtener perfil actual)
  - Comunicación con Auth Service via Feign

Frontend - Auth Module

Objetivo: Login/Signup funcionando con JWT

2.3 Auth Frontend (Días 4-5)

- Auth Feature Module
  - Login component (según SVG `reddit_login_page`)
  - Signup component
  - Auth service con JWT handling
  - Auth guard para rutas protegidas
- NgRx Auth Store
  - Actions, reducers, effects para auth

    - Manejo de estado de usuario actual
    - Token storage en localStorage

2.4 Prueba de Integración

- Login/Logout: Flujo completo funcionando
- JWT: Token válido y renovación
- Guards: Protección de rutas funcionando
- User profile: Datos básicos del usuario

🏠 FASE 3: Home Feed Básico (Semana 3)

Backend - Posts Service

Objetivo: Crear y listar posts básicos

3.1 Post Service (Días 1-3)

- Microservicio Post (Puerto 8084)
    - Base de datos post_db con posts básicos
    - Endpoints:
        - `GET /api/posts` (lista paginada)
        - `POST /api/posts` (crear post)
        - `GET /api/posts/{id}` (post individual)
    - Solo posts de texto por ahora

3.2 Vote Service Básico (Días 3-4)

- Microservicio Vote (Puerto 8086)
    - Base de datos vote_db
    - Endpoints:
        - `POST /api/votes` (votar post)
        - `GET /api/votes/posts/{id}` (obtener votos)
    - Cálculo básico de score

Frontend - Home & Posts

Objetivo: Feed de posts con voting funcionando

3.3 Home Module (Días 4-5)

- Home Feature Module
    - Home feed component (según SVG `reddit_home_design`)

    - Post card component reutilizable
    - Vote buttons component
    - Infinite scroll básico
- NgRx Post Store
    - Actions y reducers para posts
    - Effects para cargar posts
    - Estado de paginación

3.4 Create Post Básico (Día 5)

- Create Post Component
    - Formulario básico (solo texto)
    - Validaciones
    - Navegación después de crear

3.5 Prueba de Integración

- CRUD Posts: Crear y listar posts
- Voting: Upvote/downvote funcionando
- Paginación: Cargar más posts
- Real-time: Votos se actualizan

🏘️ FASE 4: Comunidades Básicas (Semana 4)

Backend - Community Service

Objetivo: Crear y gestionar comunidades

4.1 Community Service (Días 1-3)

- Microservicio Community (Puerto 8083)
    - Base de datos community_db
    - Endpoints:
        - `GET /api/communities` (listar comunidades)
        - `POST /api/communities` (crear comunidad)
        - `GET /api/communities/{name}` (detalles comunidad)
        - `POST /api/communities/{name}/join` (unirse)

4.2 Posts + Communities Integration (Días 3-4)

- Modificar Post Service

    - Posts asociados a comunidades
    - Endpoint: `GET /api/communities/{name}/posts`
    - Validar permisos de posting

Frontend - Community Module

Objetivo: Navegación y gestión de comunidades

4.3 Community Frontend (Días 4-5)

- Community Feature Module
    - Community page component (según SVG `reddit_community_page`)
    - Community header y sidebar
    - Community posts list
    - Join/Leave functionality

4.4 Navigation Update (Día 5)

- Sidebar Component
    - Lista de comunidades suscritas
    - Navegación entre comunidades
    - Crear nueva comunidad

4.5 Prueba de Integración

- Communities: Crear, unirse, ver posts
- Posts in communities: Crear posts en comunidades específicas
- Navigation: Navegar entre home y comunidades

🧑🏿‍🦲 FASE 5: Perfiles de Usuario (Semana 5)

Backend - User Service Completo

Objetivo: Perfiles completos y estadísticas

5.1 User Service Enhancement (Días 1-3)

- Expandir User Service
    - Endpoints para perfil completo
    - User stats (karma, posts count, etc.)
    - Edición de perfil
    - Avatar upload (integración con Cloudinary)

5.2 User Posts & Stats (Días 3-4)

- Posts by User
  - Endpoint: `GET /api/users/{username}/posts`
  - User karma calculation
  - Integration con Vote Service

Frontend - User Profile

Objetivo: Perfiles de usuario completos

5.3 User Module (Días 4-5)

- User Feature Module
  - User profile component (según SVG `reddit_profile_page`)
  - Profile navigation (posts, comments, etc.)
  - Profile stats component
  - Edit profile functionality

5.4 Prueba de Integración

- User profiles: Ver perfiles completos
- User posts: Posts del usuario
- Stats: Karma y estadísticas correctas

💬 FASE 6: Sistema de Comentarios (Semana 6)

Backend - Comment Service

Objetivo: Comentarios anidados funcionando

6.1 Comment Service (Días 1-4)

- Microservicio Comment (Puerto 8085)
  - Base de datos comment_db
  - Endpoints:
    - `GET /api/posts/{id}/comments` (comentarios del post)
    - `POST /api/comments` (crear comentario)
    - `POST /api/comments/{id}/reply` (responder comentario)
  - Threading system para comentarios anidados

6.2 Comments + Votes Integration (Días 4-5)

- Vote Service Update

    - Votar comentarios
    - Karma por comentarios

Frontend - Comments System

Objetivo: Interfaz de comentarios anidados

6.3 Comment Module (Días 4-5)

- Comment Feature Module
    - Comment tree component
    - Comment item component (anidado)
    - Comment form component
    - Vote buttons para comentarios

6.4 Post Detail Enhancement (Día 5)

- Post Detail Update
    - Integrar comment section
    - Actualizar según SVG `reddit_post_detail`

6.5 Prueba de Integración

- Comments: Crear y mostrar comentarios
- Nested comments: Threading funcionando
- Comment voting: Votos en comentarios

🔔 FASE 7: Notificaciones (Semana 7)

Backend - Notification Service

Objetivo: Sistema de notificaciones básico

7.1 Notification Service (Días 1-4)

- Microservicio Notification (Puerto 8087)
    - Base de datos notification_db
    - WebSocket configuration
    - Email notifications (SendGrid)
    - Notification triggers

7.2 Real-time Integration (Días 4-5)

- WebSocket Setup

    - Real-time notifications
    - Integration con otros servicios

Frontend - Notifications

Objetivo: Notificaciones en tiempo real

7.3 Notifications Frontend (Días 4-5)

- Notification System
    - WebSocket service
    - Notification dropdown
    - Real-time updates

7.4 Prueba de Integración

- Real-time: Notificaciones en tiempo real
- Email: Notificaciones por email

🎨 FASE 8: Polish & Features Avanzadas (Semana 8)

Backend - Features Avanzadas

8.1 Advanced Features (Días 1-3)

- Image Upload: Posts con imágenes
- Search: Elasticsearch integration
- Moderation: Herramientas básicas

Frontend - Polish

8.2 Advanced Frontend (Días 3-5)

- Settings Module: Configuraciones completas
- Mobile Responsive: Optimización móvil
- PWA: Progressive Web App features
- Performance: Lazy loading, optimizaciones

8.3 Testing & Documentation (Día 5)

- Testing: Unit tests críticos
- Documentation: README y deployment guides

🛠️ Herramientas de Desarrollo

Monitoreo y Testing

- **Postman Collections**: Para testing de APIs
- **Docker Compose**: Desarrollo local completo
- **GitHub Actions**: CI/CD básico
- **Swagger**: Documentación automática de APIs

Base de Datos

- **Flyway**: Migraciones versionadas
- **Oracle DB**: Una instancia por microservicio
- **Redis**: Cache para sesiones y datos frecuentes

Desarrollo

- **Hot Reload**: Angular dev server + Spring Boot DevTools
- **Logs Centralizados**: ELK Stack básico
- **Environment Variables**: Configuración por ambiente

📊 Métricas de Éxito por Fase

Fase	Objetivo de Éxito
Fase 1-2	✅ Auth funcionando end-to-end
Fase 3	✅ Posts con voting funcionando
Fase 4	✅ Comunidades básicas operativas
Fase 5	✅ Perfiles de usuario completos
Fase 6	✅ Sistema de comentarios anidados
Fase 7	✅ Notificaciones en tiempo real
Fase 8	✅ Aplicación completa y optimizada

🎯 Conclusiones

Este informe detalla una arquitectura completa de microservicios para un clon de Reddit, incluyendo:

- **8 microservicios backend** especializados con Oracle Database
- **Frontend Angular 17+** con standalone components y NgRx
- **Plan de desarrollo de 8 semanas** con objetivos claros
- **Integración continua** entre frontend y backend
- **Escalabilidad** y **mantenibilidad** como principios base

La arquitectura está diseñada para ser un **proyecto de portafolio profesional** que demuestre conocimientos avanzados en desarrollo full-stack con tecnologías modernas.