Arquitectura de Software - Sistema de Finanzas Personales

Stack Tecnológico

Frontend

- **Framework:** Angular 19 (última versión estable - Nov 2024)
- **Lenguaje:** TypeScript 5.6+
- **Características principales:**
  - Standalone Components (por defecto en v19)
  - Signals para manejo de estado reactivo
  - Built-in Control Flow
  - Server-Side Rendering (SSR) opcional
  - Angular Material 19 para UI components

Backend

- **Framework:** Spring Boot 4.0.1 (última versión - Dic 2024)
- **Lenguaje:** Java 21 LTS
- **Seguridad:** Spring Security 6.4
- **Características:**
  - REST API con arquitectura RESTful
  - Autenticación JWT
  - Soporte para Virtual Threads (Java 21)
  - Native Image support con GraalVM

Base de Datos

- **Principal:** PostgreSQL 16+ o MySQL 8.0+
- **Cache:** Redis 7.0+ (opcional pero recomendado)
- **ORM:** Spring Data JPA con Hibernate 6.4

Arquitectura de Capas

1. Capa de Presentación (Frontend - Angular)

Módulos Principales:

Auth Module

- Login Component
- Register Component
- Password Recovery Component
- Auth Guards (protección de rutas)
- JWT Interceptor

Core Module

- Header/Navigation Component
- Sidebar Component

- Footer Component
- Services compartidos
- Models/Interfaces globales

Dashboard Module

- Home Component (resumen financiero)
- Widgets de gráficas (Chart.js o ngx-charts)
- Indicadores financieros clave

Gastos Module

- Lista de gastos diarios
- Formulario de registro de gastos
- Categorización de gastos
- Filtros y búsqueda
- Exportación a CSV/Excel

Tarjetas Module

- Gestión de tarjetas de crédito
- Gestión de tarjetas de débito
- Estados de cuenta
- Límites de crédito
- Fechas de corte y pago

Cuentas Module

- Cuentas de ahorro
- Movimientos de cuentas
- Transferencias entre cuentas
- Balance actual

Inversiones Module

- Portafolio de inversiones
- Rendimientos
- Gráficas de performance
- Integración con APIs externas (opcional)

Servicios Angular:

```typescript
// Ejemplos de servicios
- AuthService
- GastosService
- TarjetasService
- CuentasService
- InversionesService
- ReportsService
```

2. Capa de API Gateway (Opcional)

- **Nginx** o **Spring Cloud Gateway**
- Load Balancing
- Rate Limiting
- CORS configuration
- SSL/TLS termination

3. Capa de Backend (Spring Boot)

Estructura de paquetes recomendada:

```
com.tuempresa.finanzas
├── config
│   ├── SecurityConfig.java
│   ├── JwtConfig.java
│   └── CorsConfig.java
├── controller
│   ├── AuthController.java
│   ├── GastoController.java
│   ├── TarjetaController.java
│   ├── CuentaController.java
│   └── InversionController.java
├── service
│   ├── AuthService.java
│   ├── GastoService.java
│   ├── TarjetaService.java
│   ├── CuentaService.java
│   └── InversionService.java
├── repository
│   ├── UserRepository.java
│   ├── GastoRepository.java
│   ├── TarjetaRepository.java
│   ├── CuentaRepository.java
│   └── InversionRepository.java
├── model
│   ├── User.java
│   ├── Gasto.java
│   ├── Tarjeta.java
│   ├── Cuenta.java
│   └── Inversion.java
├── dto
│   ├── LoginRequest.java
│   ├── GastoDTO.java
│   └── ... (otros DTOs)
├── security
│   ├── JwtTokenProvider.java
│   ├── JwtAuthenticationFilter.java
│   └── CustomUserDetailsService.java
└── exception
    ├── GlobalExceptionHandler.java
    └── ResourceNotFoundException.java
```

Configuración Spring Security 6.4:

```
java
```

```
@Configuration
@EnableWebSecurity
@EnableMethodSecurity
public class SecurityConfig {

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) {
        http
            .csrf(csrf -> csrf.disable())
            .authorizeHttpRequests(auth -> auth
                .requestMatchers("/api/auth/**").permitAll()
                .anyRequest().authenticated()
            )
            .sessionManagement(session -> session
                .sessionCreationPolicy(SessionCreationPolicy.STATELESS)
            )
            .addFilterBefore(jwtAuthFilter,
                UsernamePasswordAuthenticationFilter.class);
        return http.build();
    }
}
```

4. Modelo de Datos (Entidades Principales)

User

- id (UUID)
- username
- email
- password (encriptado con BCrypt)
- nombres
- apellidos
- fechaCreacion
- ultimoAcceso

Gasto

- id
- userId (FK)
- concepto
- monto
- categoria
- fecha
- metodoPago
- notas

TarjetaCredito

- id
- userId (FK)
- nombreTarjeta
- banco

- numeroUltimos4Digitos
- limiteCredito
- saldoActual
- fechaCorte
- fechaPago
- tasaInteres

TarjetaDebito

- id
- userId (FK)
- nombreTarjeta
- banco
- numeroUltimos4Digitos
- saldoActual

CuentaAhorro

- id
- userId (FK)
- nombreCuenta
- banco
- numeroCuenta
- saldoActual
- tasaInteres
- fechaApertura

Inversion

- id
- userId (FK)
- tipoInversion (acciones, bonos, fondos, etc.)
- nombreInversion
- montoInvertido
- valorActual
- rendimiento
- fechaInversion

Seguridad

Autenticación JWT

1. Usuario envía credenciales (POST /api/auth/login)
2. Backend valida credenciales
3. Backend genera JWT token con tiempo de expiración
4. Frontend almacena token en localStorage/sessionStorage
5. Cada petición incluye token en header: `Authorization: Bearer <token>`

6. Backend valida token en cada request

Buenas prácticas de seguridad:

- Passwords hasheados con BCrypt (factor 12+)
- Tokens JWT con expiración (15-60 min)
- Refresh tokens para renovar sesión
- HTTPS obligatorio en producción
- CORS configurado correctamente
- Validación de entrada en frontend y backend
- Rate limiting para prevenir ataques
- SQL Injection protegido por JPA/Hibernate

APIs REST - Endpoints Principales

Auth

- POST `/api/auth/register` - Registro de usuario
- POST `/api/auth/login` - Login
- POST `/api/auth/refresh` - Refresh token
- POST `/api/auth/logout` - Cerrar sesión

Gastos

- GET `/api/gastos` - Listar gastos (con paginación)
- GET `/api/gastos/{id}` - Obtener gasto por ID
- POST `/api/gastos` - Crear gasto
- PUT `/api/gastos/{id}` - Actualizar gasto
- DELETE `/api/gastos/{id}` - Eliminar gasto
- GET `/api/gastos/stats` - Estadísticas de gastos

Tarjetas

- GET `/api/tarjetas/credito` - Listar tarjetas de crédito
- POST `/api/tarjetas/credito` - Crear tarjeta de crédito
- PUT `/api/tarjetas/credito/{id}` - Actualizar tarjeta
- DELETE `/api/tarjetas/credito/{id}` - Eliminar tarjeta
- GET `/api/tarjetas/debito` - Listar tarjetas de débito

Cuentas

- GET `/api/cuentas` - Listar cuentas
- POST `/api/cuentas` - Crear cuenta
- PUT `/api/cuentas/{id}` - Actualizar cuenta
- DELETE `/api/cuentas/{id}` - Eliminar cuenta
- GET `/api/cuentas/{id}/movimientos` - Movimientos de cuenta

Inversiones

- GET `/api/inversiones` - Listar inversiones

- POST `/api/inversiones` - Crear inversión
- PUT `/api/inversiones/{id}` - Actualizar inversión
- DELETE `/api/inversiones/{id}` - Eliminar inversión
- GET `/api/inversiones/rendimiento` - Reporte de rendimientos

Despliegue

Desarrollo

- Frontend: `ng serve` (puerto 4200)
- Backend: Spring Boot embedded server (puerto 8080)
- DB: Docker container con PostgreSQL

Producción

- **Frontend:**
  - Build: `ng build --configuration production`
  - Deploy: Netlify, Vercel, AWS S3 + CloudFront, o Nginx
- **Backend:**
  - Build: `./mvnw clean package` (JAR ejecutable)
  - Deploy: AWS EC2, Docker container, Kubernetes, Railway, Render
- **Base de Datos:**
  - AWS RDS PostgreSQL, Azure Database, Railway, o servidor dedicado

Contenedorización (Docker)

dockerfile

```
# Frontend Dockerfile
FROM node:20-alpine AS build
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
RUN npm run build

FROM nginx:alpine
COPY --from=build /app/dist/tu-app /usr/share/nginx/html
```

dockerfile

```
# Backend Dockerfile
FROM eclipse-temurin:21-jdk-alpine AS build
WORKDIR /app
COPY . .
RUN ./mvnw clean package -DskipTests

FROM eclipse-temurin:21-jre-alpine
WORKDIR /app
COPY --from=build /app/target/*.jar app.jar
EXPOSE 8080
ENTRYPOINT ["java", "-jar", "app.jar"]
```

Características Adicionales Recomendadas

1. **Reportes y Analytics:**
    - Dashboard con gráficas de gastos por categoría
    - Tendencias mensuales/anuales
    - Comparativas entre períodos
2. **Notificaciones:**
    - Email cuando se acerque fecha de pago de tarjeta
    - Alertas de gastos excesivos
    - Recordatorios personalizados
3. **Exportación de datos:**
    - Exportar a Excel/CSV
    - Generar PDFs de reportes
4. **Modo oscuro:**
    - Implementar tema claro/oscuro en Angular
5. **Internacionalización:**
    - Soporte multiidioma (español/inglés)
6. **Responsive Design:**
    - Mobile-first approach
    - Progressive Web App (PWA)

Próximos Pasos

1. Configurar proyecto Angular 19 con Angular CLI
2. Crear proyecto Spring Boot 4.0.1 con Spring Initializr
3. Configurar base de datos PostgreSQL
4. Implementar autenticación JWT
5. Desarrollar módulos uno por uno
6. Crear tests unitarios y de integración
7. Deployment en ambiente de desarrollo
8. Testing end-to-end
9. Deployment en producción

Notas importantes:

- Esta arquitectura es escalable y puede crecer según necesidades
- Considera implementar CI/CD con GitHub Actions o GitLab CI
- Usa control de versiones (Git) desde el inicio
- Documenta tu API con Swagger/OpenAPI
- Implementa logging con SLF4J y Logback
- Considera usar Spring Boot Actuator para monitoreo