



# Manual Técnico

Sistema de Gestión de Bienestar - Versión 0.0.1-SNAPSHOT  
Documentación técnica completa para desarrolladores

[Arquitectura](#)[Tecnologías](#)[Instalación](#)[Estructura](#)[API REST](#)[Seguridad](#)[Base de Datos](#)[Android](#)[Despliegue](#)

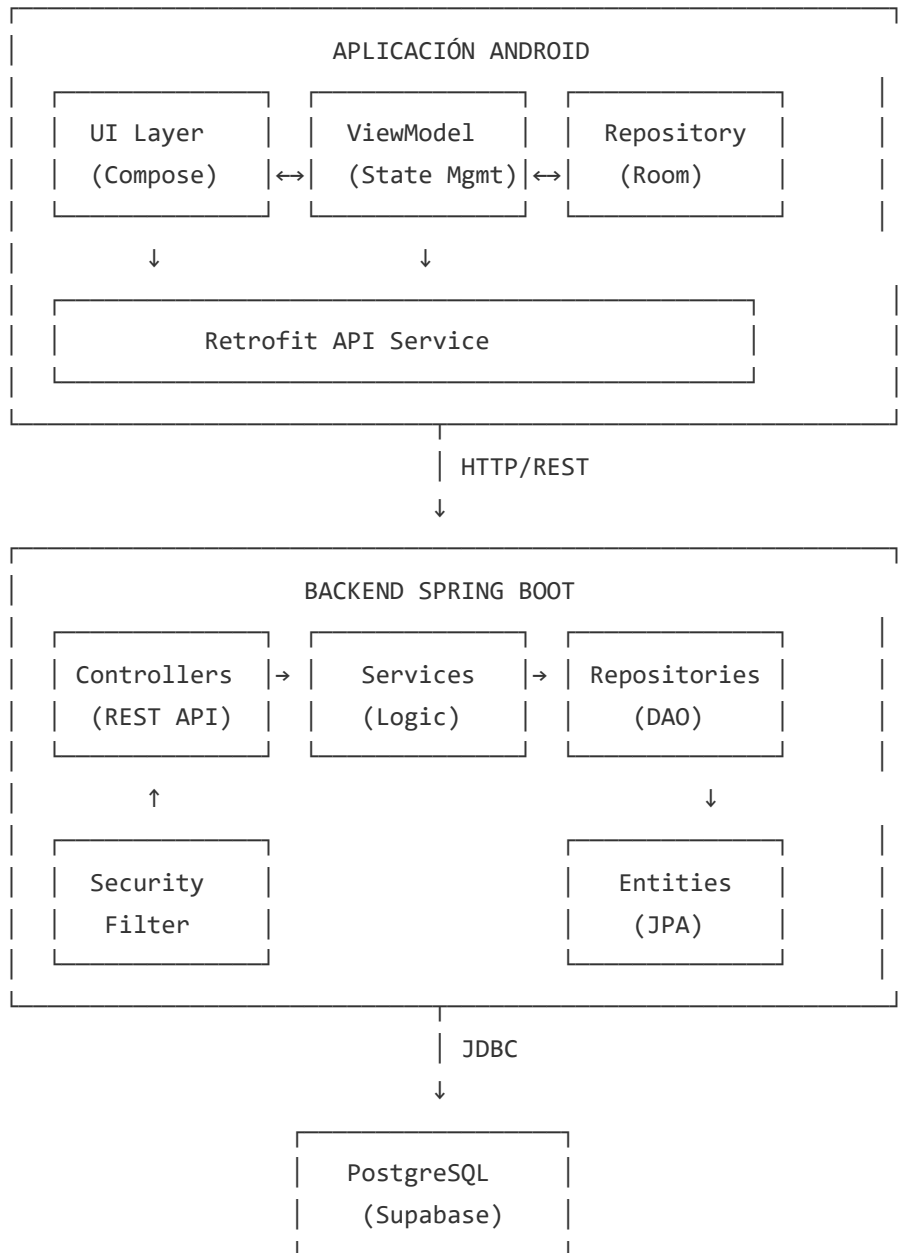
## Arquitectura del Sistema

### Arquitectura General

El sistema está compuesto por dos aplicaciones principales:

- **Backend:** API REST desarrollada con Spring Boot
- **Frontend Mobile:** Aplicación Android nativa con Kotlin y Jetpack Compose

#### Diagrama de Arquitectura



## Patrón MVC (Model-View-Controller)

Capa	Responsabilidad	Componentes
Model	Representación de datos	Entidades JPA, DTOs

Capa	Responsabilidad	Componentes
<b>View</b>	Presentación (UI)	JSON (API), Thymeleaf (Web), Jetpack Compose (Android)
<b>Controller</b>	Manejo de peticiones	@RestController, @Controller

## Principios SOLID Aplicados

- **S Single Responsibility:** Cada clase tiene una única responsabilidad
- **O Open/Closed:** Abierto a extensión, cerrado a modificación
- **L Liskov Substitution:** Las implementaciones son intercambiables
- **I Interface Segregation:** Interfaces específicas y pequeñas
- **D Dependency Inversion:** Dependencia de abstracciones, no implementaciones

## Stack Tecnológico

### Backend (Spring Boot)

Tecnología	Versión	Propósito
Java	17 LTS	Lenguaje de programación
Spring Boot	3.5.6	Framework principal
Spring Security	6.x	Autenticación y autorización
Spring Data JPA	3.x	Capa de persistencia

Tecnología	Versión	Propósito
Hibernate	6.6.x	ORM (Object-Relational Mapping)
PostgreSQL	17 (Supabase)	Base de datos relacional
Bean Validation	3.x	Validación de datos
SpringDoc OpenAPI	2.x	Documentación API (Swagger)
Maven	3.9.x	Gestión de dependencias

## Frontend Mobile (Android)

Tecnología	Versión	Propósito
Kotlin	1.9.x	Lenguaje de programación
Jetpack Compose	Latest	UI moderna declarativa
Material Design 3	Latest	Componentes UI
Retrofit	2.9.0	Cliente HTTP
OkHttp	4.12.0	Cliente HTTP de bajo nivel
Gson	2.10.1	Serialización JSON
Navigation Compose	2.7.7	Navegación entre pantallas
ViewModel	2.7.0	Gestión de estado
Coroutines	1.7.3	Programación asíncrona



# Instalación y Configuración

---

## Requisitos Previos

- JDK 17 o superior
- Maven 3.9.x
- PostgreSQL (o cuenta en Supabase)
- Android Studio (para desarrollo móvil)
- Git

## 1. Clonar el Repositorio

```
# Clonar proyecto
git clone https://github.com/Jesus2255/Proyecto-bienestar.git
cd Proyecto-bienestar
```

## 2. Configurar Base de Datos

Crear archivo `Bienestar/config/application-local.properties`:

```
# PostgreSQL Configuration
spring.datasource.url=jdbc:postgresql://[HOST]:[PORT]/[DATABASE]
spring.datasource.username=[USERNAME]
spring.datasource.password=[PASSWORD]

# JPA/Hibernate
spring.jpa.hibernate.ddl-auto=validate
spring.jpa.show-sql=false
spring.jpa.properties.hibernate.format_sql=true

# Configuración de pool de conexiones
```

```
spring.datasource.hikari.maximum-pool-size=10
spring.datasource.hikari.minimum-idle=5
```

⚠ **Importante:** No subir este archivo a Git. Ya está en `.gitignore`.

### 3. Compilar Backend

```
cd Bienestar
mvn clean install -DskipTests
mvn package
```

### 4. Ejecutar Backend

```
# Opción 1: Usando script (Windows)
run-local.cmd

# Opción 2: Con Maven
mvn spring-boot:run -Dspring-boot.run.profiles=local

# Opción 3: Con JAR
java -jar target/Bienestar-0.0.1-SNAPSHOT.jar --spring.profile
```

✅ **Servidor iniciado:** `http://localhost:8080`

### 5. Configurar Android

En `Bienestar`

`app/app/src/main/java/.../network/NetworkModule.kt` :

```
object NetworkModule {
    // Para emulador Android
    private const val BASE_URL = "http://10.0.2.2:8080/"
```

```
// Para dispositivo físico (usar IP de tu PC)
// private const val BASE_URL = "http://192.168.1.x:8080/"
}
```

## 6. Ejecutar App Android

1. Abrir Android Studio
2. File → Open → Seleccionar carpeta `Bienestar app`
3. Esperar sincronización de Gradle
4. Run → Run 'app'

## Estructura del Proyecto

### Backend (Spring Boot)

```
Bienestar/
├── src/main/java/com/bienestarproyect/Bienestar/
│   ├── controller/                # Controladores REST
│   │   ├── AuthController.java
│   │   ├── ClienteController.java
│   │   ├── CitaController.java
│   │   ├── ServicioController.java
│   │   ├── FacturaController.java
│   │   └── ValidationExceptionHandler.java
│   └── service/                   # Lógica de negocio
│       ├── ClienteService.java
│       ├── CitaService.java
│       ├── ServicioService.java
│       └── FacturaService.java
```

```
| | └─ UsuarioDetailsService.java
| |
| | └─ repository/          # Acceso a datos (DAO)
| |   └─ ClienteRepository.java
| |   └─ CitaRepository.java
| |   └─ ServicioRepository.java
| |   └─ FacturaRepository.java
| |   └─ UsuarioRepository.java
| |   └─ RoleRepository.java
| |
| | └─ entity/              # Entidades JPA
| |   └─ Cliente.java
| |   └─ Cita.java
| |   └─ Servicio.java
| |   └─ Factura.java
| |   └─ Usuario.java
| |   └─ Role.java
| |
| | └─ dto/                 # Data Transfer Objects
| |   └─ ClienteDTO.java
| |   └─ CitaDTO.java
| |   └─ FacturaDTO.java
| |   └─ UserInfoResponse.java
| |
| | └─ mapper/              # Conversión DTO ↔ Entity
| |   └─ DTOMapper.java
| |
| | └─ config/              # Configuración
| |   └─ WebConfig.java
| |
| | └─ migration/           # Migraciones de BD
| |   └─ SimpleMigrationRunner.java
| |
| | └─ SecurityConfig.java  # Configuración de seguridad
| | └─ DataInitializer.java # Datos iniciales
| | └─ BienestarApplication.java
| |
| └─ src/main/resources/
```



```

|   ├── application.properties
|   ├── application-dev.properties
|   ├── application-prod.properties
|   ├── db/migration/           # Scripts SQL
|   ├── static/                 # Recursos estáticos
|   └── templates/              # Plantillas Thymeleaf
|
└── pom.xml                     # Dependencias Maven

```

## Frontend Android

```

Bienestar app/
├── app/src/main/java/com/example/bienestarapp/
|   ├── ui/                      # Composables UI
|   |   ├── screens/
|   |   |   ├── LoginScreen.kt
|   |   |   ├── HomeScreen.kt
|   |   |   ├── ClientesScreen.kt
|   |   |   ├── ServiciosScreen.kt
|   |   |   └── CitasScreen.kt
|   |   └── components/
|   |       └── CommonUI.kt    # AppToolBar, etc.
|   |
|   ├── viewmodel/              # ViewModels
|   |   ├── LoginViewModel.kt
|   |   ├── AuthViewModel.kt
|   |   ├── ClientesViewModel.kt
|   |   ├── HomeViewModel.kt
|   |   └── CitasViewModel.kt
|   |
|   ├── network/                # Capa de red
|   |   ├── ApiService.kt      # Retrofit interface
|   |   └── NetworkModule.kt   # Configuración
|   |
|   ├── model/                  # Modelos de datos
|   |   └── Models.kt          # Cliente, Servicio, Cita, etc.
|   |
|   └──

```

```
| | └─ util/ # Utilidades
| |   └─ UserSession.kt # Gestión de sesión
| |
| | └─ MainActivity.kt # Activity principal
| |
| └─ app/src/main/res/
|   └─ values/ # Recursos
|       └─ AndroidManifest.xml
|
| └─ build.gradle.kts # Configuración Gradle
```



## Documentación API REST

---

■ **Swagger UI:** <http://localhost:8080/swagger-ui.html>

### Autenticación

#### POST /login

Autenticación de usuario. Retorna cookie de sesión (JSESSIONID).

```
// Request (form-urlencoded)
username=admin&password=1234

// Response 200 OK
{
  "message": "Login successful",
  "username": "admin"
}

// Response 401 Unauthorized
{
```

```
"error": "Authentication failed",  
"message": "Bad credentials"  
}
```

### **GET /api/auth/user-info**

Obtiene información del usuario autenticado (requiere sesión activa).

```
// Response 200 OK  
{  
  "success": true,  
  "username": "admin",  
  "role": "ROLE_ADMIN"  
}  
  
// Response 401 Unauthorized  
{  
  "success": false,  
  "username": null,  
  "role": null  
}
```

### **POST /logout**

Cierra la sesión del usuario.

```
// Response 200 OK  
{  
  "message": "Logout successful"  
}
```

## **Clientes**

### **GET /api/clientes**

Lista todos los clientes.

```
// Response 200 OK
[
  {
    "id": 1,
    "nombre": "Juan Pérez",
    "email": "juan@example.com",
    "telefono": "555-1234"
  }
]
```

### **GET /api/clientes/{id}**

Obtiene un cliente por ID.

### **POST /api/clientes**

Crea un nuevo cliente.

```
// Request Body
{
  "nombre": "María García",
  "email": "maria@example.com",
  "telefono": "555-5678"
}

// Response 200 OK
{
  "id": 2,
  "nombre": "María García",
  "email": "maria@example.com",
  "telefono": "555-5678"
}
```

### **PUT /api/clientes/{id}**

Actualiza un cliente existente.

### **DELETE /api/clientes/{id}**

Elimina un cliente.

## Servicios

### GET /api/servicios

Lista todos los servicios disponibles.

```
// Response 200 OK
[
  {
    "id": 1,
    "nombre": "Masaje Relajante",
    "descripcion": "Masaje de 60 minutos",
    "precio": 50.00
  }
]
```

## Citas

### POST /api/citas

Agenda una nueva cita.

```
// Request Body
{
  "clienteId": 1,
  "servicioId": 1,
  "fechaHora": "2025-10-30T14:00:00"
}

// Response 200 OK
{
  "id": 1,
  "cliente": { "id": 1, "nombre": "Juan Pérez" },
  "servicio": { "id": 1, "nombre": "Masaje Relajante" },
  "fechaHora": "2025-10-30T14:00:00",
  "estado": "AGENDADA"
}
```

## Códigos de Estado HTTP

Código	Significado	Uso
200	OK	Operación exitosa
400	Bad Request	Error de validación
401	Unauthorized	No autenticado
403	Forbidden	Sin permisos
404	Not Found	Recurso no encontrado
500	Internal Server Error	Error del servidor



## Seguridad

### Autenticación

El sistema utiliza **Spring Security** con autenticación basada en sesiones:

- **Tipo:** Form-based authentication
- **Sesión:** Cookie JSESSIONID
- **Encoder:** NoOpPasswordEncoder (⚠ Solo desarrollo)



**CRÍTICO:** En producción, cambiar `NoOpPasswordEncoder` por `BCryptPasswordEncoder`:

```
@Bean
public PasswordEncoder passwordEncoder() {
```

```
return new BCryptPasswordEncoder(12);  
}
```

## Autorización (RBAC)

Control de acceso basado en roles:

Rol	Permisos	Rutas Permitidas
ADMIN	Acceso total	/api/admin/**, /api/**
RECEPTIONIST	Gestión de citas y clientes	/api/**
CLIENT	Ver servicios, sus citas	/api/** (limitado)

## Validación de Datos

Validación automática con Bean Validation:

```
public class ClienteDTO {  
    @NotBlank(message = "Nombre obligatorio")  
    private String nombre;  
  
    @NotBlank(message = "Email obligatorio")  
    @Email(message = "Email inválido")  
    private String email;  
  
    @NotBlank(message = "Teléfono obligatorio")  
    private String telefono;  
}
```

## Protección CSRF

CSRF deshabilitado para API REST (clientes stateless):

```
http.csrf(csrf -> csrf.ignoringRequestMatchers(request -> {  
    String uri = request.getRequestURI();  
    return uri.startsWith("/api/") || uri.equals("/login");  
}));
```

## Usuarios Demo

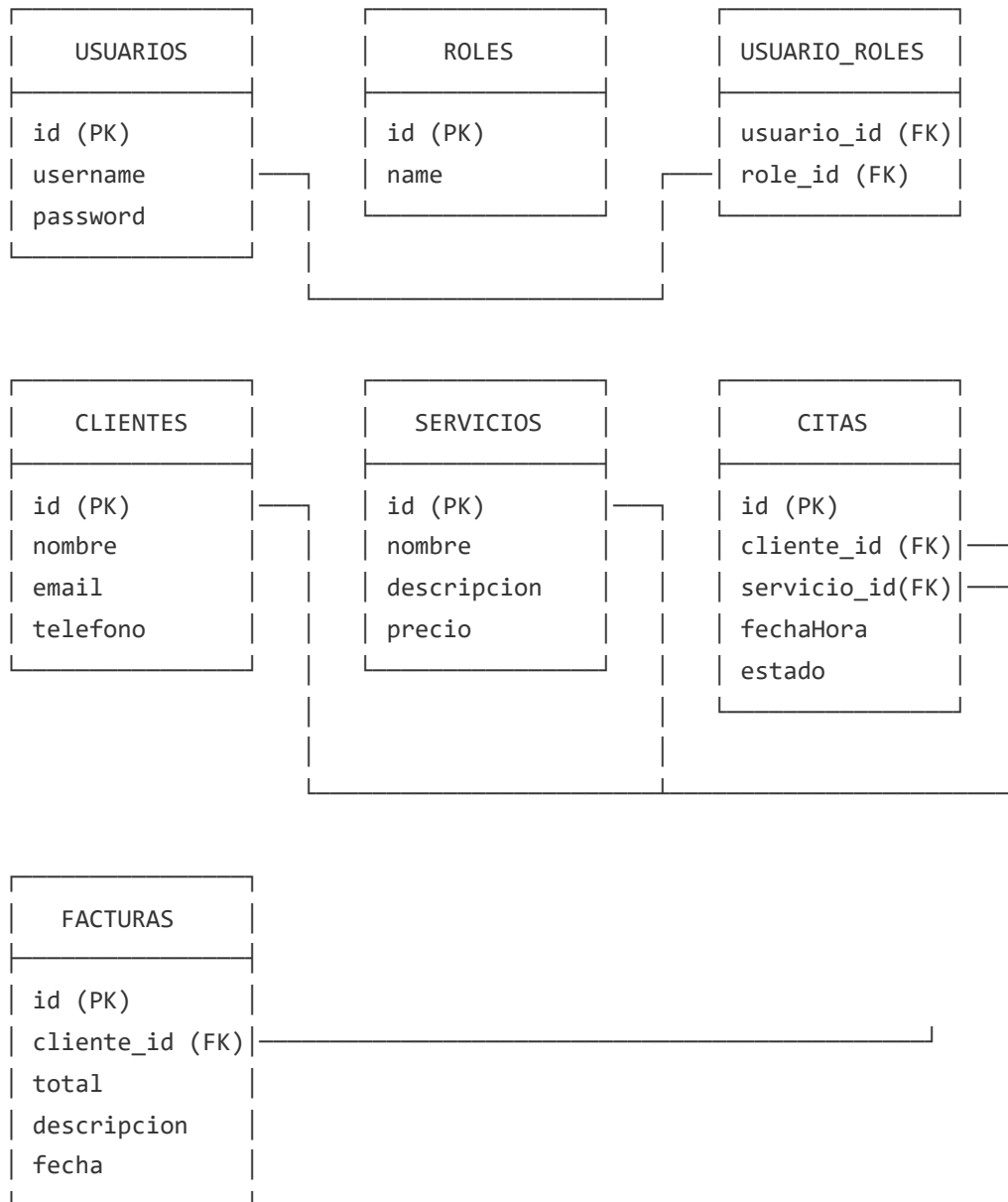
Usuario	Contraseña	Roles
admin	1234	ADMIN, RECEPTIONIST
client	1234	CLIENT



## Base de Datos

### Modelo de Datos





## Script de Creación

```
-- Usuarios y roles
CREATE TABLE usuarios (
  id BIGSERIAL PRIMARY KEY,
  username VARCHAR(255) NOT NULL UNIQUE,
```

```
password VARCHAR(255) NOT NULL
);

CREATE TABLE roles (
    id BIGSERIAL PRIMARY KEY,
    name VARCHAR(50) NOT NULL UNIQUE
);

CREATE TABLE usuario_roles (
    usuario_id BIGINT REFERENCES usuarios(id),
    role_id BIGINT REFERENCES roles(id),
    PRIMARY KEY (usuario_id, role_id)
);

-- Clientes
CREATE TABLE clientes (
    id BIGSERIAL PRIMARY KEY,
    nombre VARCHAR(255),
    email VARCHAR(255),
    telefono VARCHAR(100)
);

-- Servicios
CREATE TABLE servicios (
    id BIGSERIAL PRIMARY KEY,
    nombre VARCHAR(255),
    descripcion TEXT,
    precio NUMERIC(12,2)
);

-- Citas
CREATE TABLE citas (
    id BIGSERIAL PRIMARY KEY,
    cliente_id BIGINT REFERENCES clientes(id),
    servicio_id BIGINT REFERENCES servicios(id),
    fecha_hora TIMESTAMP,
    estado VARCHAR(50)
);
```

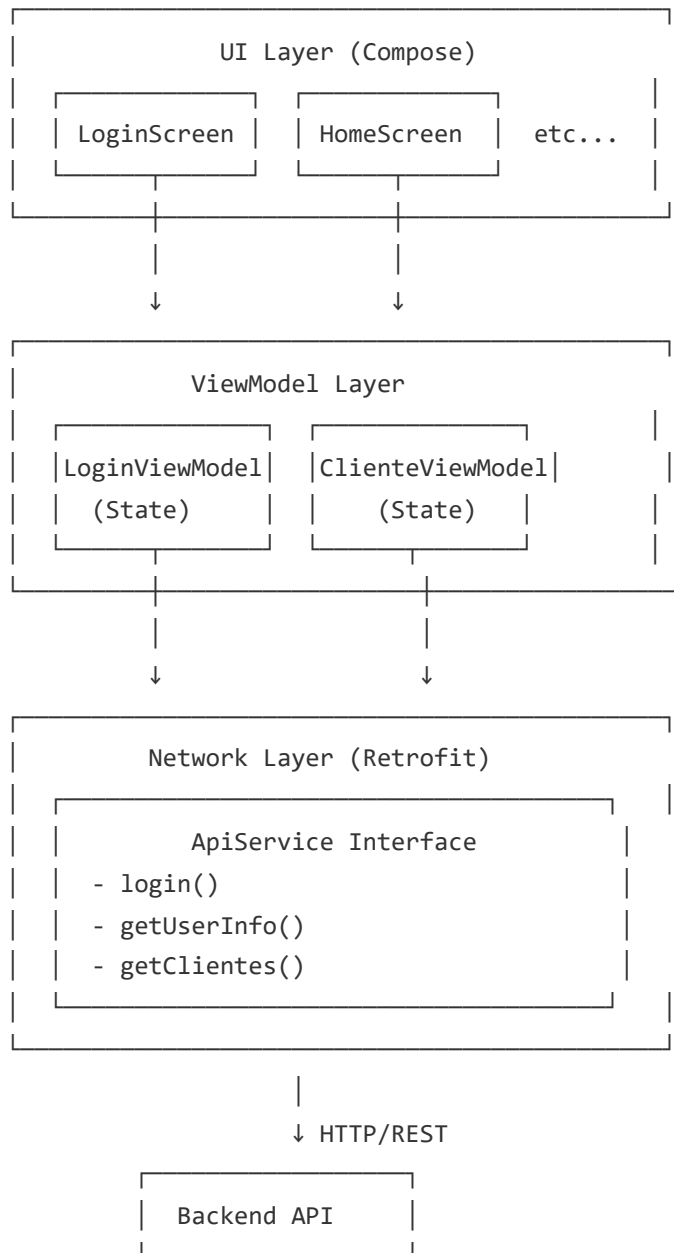
```
-- Facturas
CREATE TABLE facturas (
  id BIGSERIAL PRIMARY KEY,
  cliente_id BIGINT REFERENCES clientes(id),
  total NUMERIC(12,2),
  descripcion TEXT,
  fecha TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```



## Aplicación Android

---

### Arquitectura MVVM



## Componentes Principales

### 1. ApiService (Retrofit)

```
interface ApiService {  
    @FormUrlEncoded
```

```

@POST("login")
suspend fun login(
    @Field("username") username: String,
    @Field("password") password: String
): Response<LoginResponse>

@GET("api/auth/user-info")
suspend fun getUserInfo(): UserInfoResponse

@GET("api/clientes")
suspend fun getClientes(): List<Cliente>

@POST("api/clientes")
suspend fun createCliente(@Body cliente: Cliente): Cliente
}

```

## 2. ViewModel (State Management)

```

class LoginViewModel : ViewModel() {
    var username by mutableStateOf("")
    var password by mutableStateOf("")
    var loginStatus by mutableStateOf("")

    fun performLogin() {
        viewModelScope.launch {
            val response = NetworkModule.api.login(username, password)
            if (response.isSuccessful) {
                // Obtener rol del usuario
                val userInfo = NetworkModule.api.getUserInfo()
                UserSession.login(userInfo.username, userInfo.password)
            }
        }
    }
}

```

## 3. Composable UI

```

@Composable
fun LoginScreen(viewModel: LoginViewModel) {
    Column {
        OutlinedTextField(
            value = viewModel.username,
            onValueChange = { viewModel.username = it },
            label = { Text("Usuario") }
        )

        Button(onClick = { viewModel.performLogin() }) {
            Text("Iniciar Sesión")
        }
    }
}

```

## Gestión de Estado

Se utiliza `mutableStateOf` para reactividad automática:

- Cambios en el estado actualizan la UI automáticamente
- `viewModelScope.launch` para operaciones asíncronas
- Separación clara entre UI, lógica y datos



## Despliegue

---

### Backend en Producción

#### 1. Configurar Variables de Entorno


```

export SPRING_PROFILES_ACTIVE=prod
export DB_URL=jdbc:postgresql://[HOST]:[PORT]/[DB]

```

```
export DB_USERNAME=[USER]
export DB_PASSWORD=[PASS]
```

## 2. Cambiar PasswordEncoder

```
@Bean
public PasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder(12); //  Seguro
}
```

## 3. Compilar JAR

```
mvn clean package -DskipTests
# Genera: target/Bienestar-0.0.1-SNAPSHOT.jar
```

## 4. Ejecutar en Servidor

```
java -jar Bienestar-0.0.1-SNAPSHOT.jar \
    --spring.profiles.active=prod \
    --server.port=8080
```

# Aplicación Android

## 1. Cambiar BASE\_URL

```
// NetworkModule.kt
private const val BASE_URL = "https://tu-dominio.com/"
```

## 2. Generar APK Firmado

1. Build → Generate Signed Bundle / APK
2. Seleccionar "APK"
3. Crear/usar keystore
4. Build type: "release"
5. APK generado en: `app/release/app-release.apk`

# Checklist Pre-Producción

Item	Estado
✓ Cambiar NoOpPasswordEncoder a BCryptPasswordEncoder	Crítico
✓ No exponer stacktraces en respuestas HTTP	Crítico
✓ Configurar HTTPS/SSL	Crítico
✓ Variables de entorno para credenciales	Importante
✓ Logging apropiado (no debug en prod)	Importante
✓ Configurar pool de conexiones	Recomendado
✓ Pruebas de carga	Recomendado