

# MANUAL-GUIA

# PROYECTO FINAL

1ª PARTE.....	2
FASE 1: Preparar entorno con Docker + PostgreSQL .....	2
➤ Configurar PostgreSQL en Docker Desktop.....	2
FASE 2:Crear el proyecto Spring Boot en IntelliJ .....	5
➤ Configurar application.properties .....	6
➤ Crear las entidades .....	7
➤ Crear los Repositorios.....	9
➤ Crear los Controladores REST.....	10
➤ Crear la capa de servicios.....	12
FASE 3: Integrar Firebase Authentication en Spring Boot.....	14
➤ Crear un proyecto en Firebase .....	14
➤ Añadir dependencias Firebase en build.gradle .....	17
➤ Inicializar Firebase al arrancar la aplicación.....	18
➤ Crear un servicio para validar tokens de Firebase.....	18
➤ Crear clases de seguridad .....	19
➤ Crear un controlador para probar autenticación .....	20
FASE 4: Diseño Web con Thymeleaf .....	21
➤ Agregar dependencias de Thymeleaf en build.gradle .....	21
➤ Crear nuevos controladores para manejar Thymeleaf.....	21
➤ Creación de las vistas html.....	24
2ºPARTE.....	30
FASE 1: Crear proyecto en Android Studio .....	30
FASE 2: Modificaciones en Spring Boot para que los usuarios registrados en la base de datos puedan iniciar sesión desde la app Android.....	32
FASE 3: Seguir con el proyecto de Android Studio.....	33
➤ Crear Login desde cero .....	33
➤ Crear el menú de opciones (segunda pantalla).....	39
➤ Crear las diferentes opciones.....	43

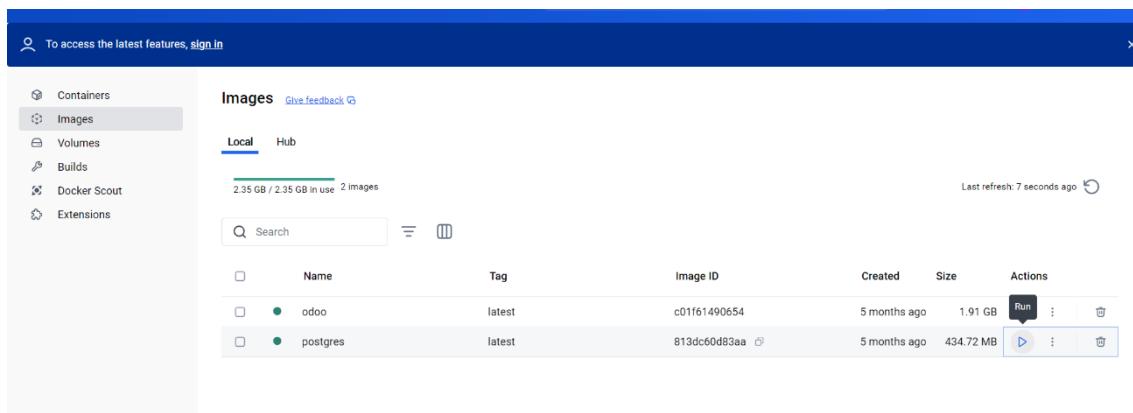
## 1ª PARTE

### FASE 1: Preparar entorno con Docker + PostgreSQL

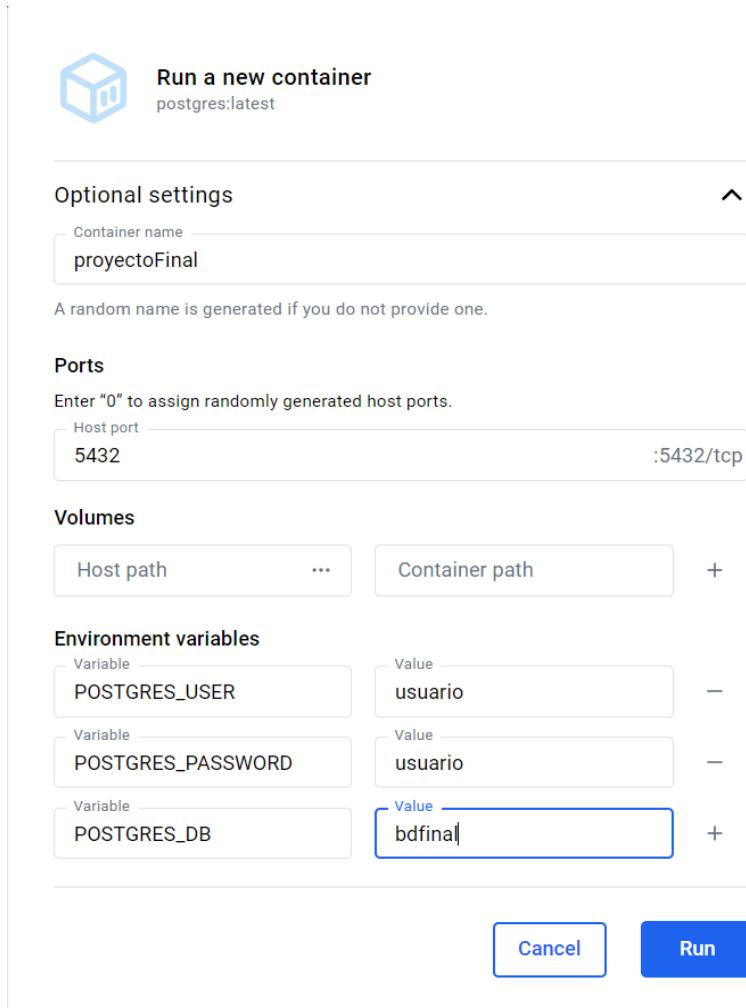
En esta fase vamos a tener una base de datos PostgreSQL funcionando con Docker para usarla desde Spring Boot y después desde Android.

#### ➤ Configurar PostgreSQL en Docker Desktop

- Abrir Docker Desktop
- Ir a la pestaña "Image"
  - Descargar la imagen de postgres.
  - Seleccionar la fecha RUN



- Se abrirá una nueva ventana donde tendremos que configurar nuestro postgres
  - En **Container Name**, ponemos “proyectoFinal”.
  - Configurar el host port a **5432**
  - Configurar las variables de entorno:
    - **POSTGRES\_USER** → usuario
    - **POSTGRES\_PASSWORD** → usuario
    - **POSTGRES\_DB** → bdfinal
  - Hacer clic en "**Run**"

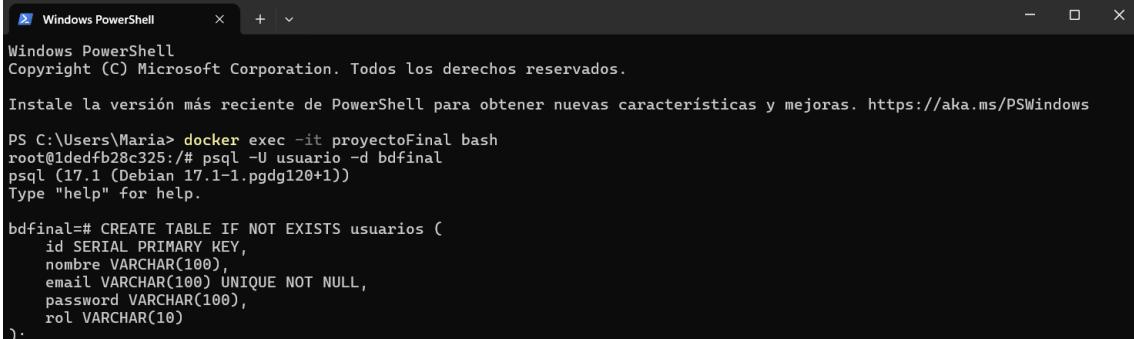


- Abrimos la terminal para crear las tablas
- Primero entramos dentro del contenedor (Docker) con este comando:
  - docker exec -it proyectoFinal bash
- Seguidamente ponemos el usuario y el nombre de la base de datos con este comando:
  - psql -U usuario -d bdfinal
- Y por último creamos las tablas:

```
CREATE TABLE IF NOT EXISTS usuarios (
    id SERIAL PRIMARY KEY,
    nombre VARCHAR(100),
    email VARCHAR(100) UNIQUE NOT NULL,
    password VARCHAR(100),
    rol VARCHAR(10)
);
```

```
CREATE TABLE IF NOT EXISTS servicios (
    nombre VARCHAR(100) PRIMARY KEY,
    precio NUMERIC(7,2) NOT NULL,
    tiempo INTEGER NOT NULL
);
```

```
CREATE TABLE IF NOT EXISTS citas (
    id SERIAL PRIMARY KEY,
    id_usuario INT,
    fecha TIMESTAMP,
    servicio VARCHAR(100) NOT NULL,
    FOREIGN KEY (id_usuario) REFERENCES usuarios(id),
    FOREIGN KEY (servicio) REFERENCES servicios(nombre)
);
```



```
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Instale la versión más reciente de PowerShell para obtener nuevas características y mejoras. https://aka.ms/PSWindows

PS C:\Users\Maria> docker exec -it proyectoFinal bash
root@1dedfb28c325:/# psql -U usuario -d bdffinal
psql (17.1 (Debian 17.1-1.pgdg120+1))
Type "help" for help.

bdffinal=# CREATE TABLE IF NOT EXISTS usuarios (
    id SERIAL PRIMARY KEY,
    nombre VARCHAR(100),
    email VARCHAR(100) UNIQUE NOT NULL,
    password VARCHAR(100),
    rol VARCHAR(10)
);

bdffinal=# CREATE TABLE IF NOT EXISTS servicios (
    nombre VARCHAR(100) PRIMARY KEY,
    precio NUMERIC(7,2) NOT NULL,
    tiempo INTEGER NOT NULL
);

bdffinal=# CREATE TABLE IF NOT EXISTS citas (
    id SERIAL PRIMARY KEY,
    id_usuario INT,
    fecha TIMESTAMP,
    servicio VARCHAR(100) NOT NULL,
    FOREIGN KEY (id_usuario) REFERENCES usuarios(id),
    FOREIGN KEY (servicio) REFERENCES servicios(nombre)
);

bdffinal=# INSERT INTO usuarios (nombre, email, password, rol) VALUES
('Ana García', 'ana@gmail.com', '123456', 'CLIENTE'),
('Luis Pérez', 'luis@gmail.com', 'abc123', 'CLIENTE');
```

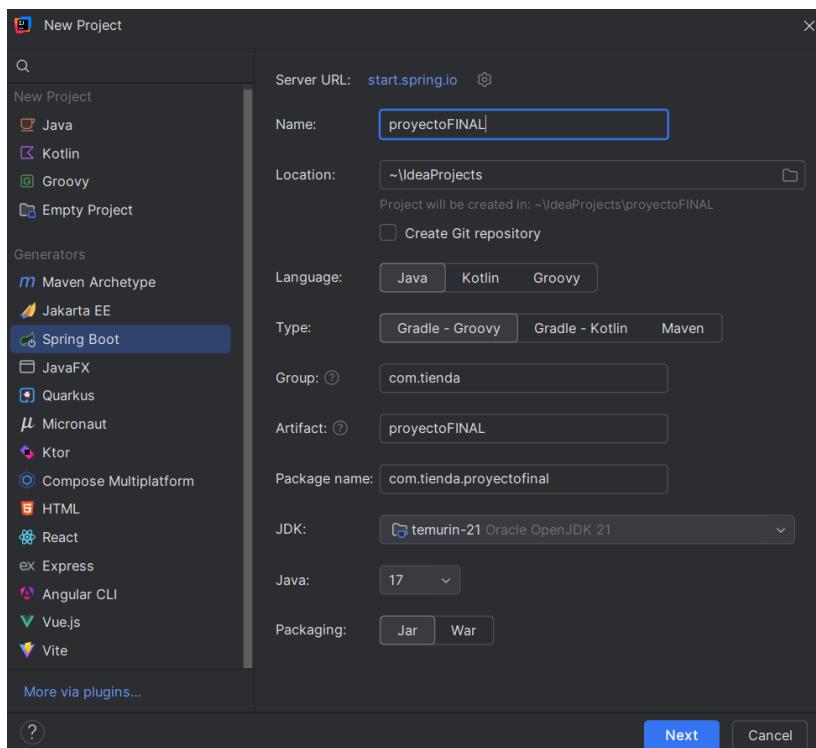
```
bdfinal=# INSERT INTO servicios (nombre, precio, tiempo) VALUES
('Corte de pelo',      12.00, 30),
('Lavado y peinado',   18.00, 25),
('Coloración básica',  35.00, 60);
INSERT 0 3
```

```
bdfinal=# INSERT INTO citas (id_usuario, fecha, servicio) VALUES
(1, '2025-05-20 10:00:00', 'Corte de pelo'),
(2, '2025-05-21 12:30:00', 'Coloración');
INSERT 0 2
```

## FASE 2: Crear el proyecto Spring Boot en IntelliJ

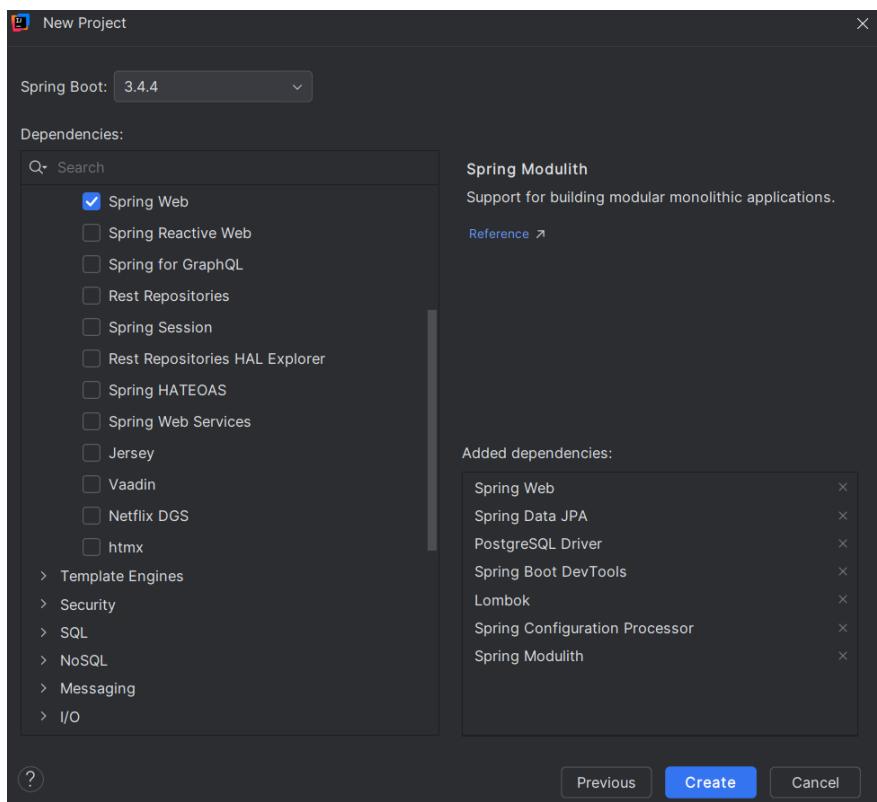
Spring Boot es una herramienta basada en el framework Spring que permite crear aplicaciones Java listas para producción de forma rápida y con mínima configuración. Su principal ventaja es que elimina la necesidad de configurar manualmente gran parte del proyecto, gracias a su autoconfiguración.

- Abrimos IntelliJ > **New Project**
- Seleccionamos **Spring Boot**
- Llenamos los datos:



- En "Dependencies" agregamos:

- Spring Web**
- Spring Data JPA**
- PostgreSQL Driver**
- Spring Boot DevTools**
- Lombok** (para evitar getters/setters)



- Creamos el proyecto y le damos a **Finish**.

## ➤ Configurar application.properties

- Vamos a este archivo:

src/main/resources/application.properties

- Y ponemos los datos de la base de datos que hemos creado antes:

```
spring.datasource.url=jdbc:postgresql://localhost:5432/bdfinal
spring.datasource.username=usuario
spring.datasource.password=usuario
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
```

The screenshot shows the IntelliJ IDEA interface. On the left, the project structure for 'proyectoFINAL' is visible, including 'gradle', '.idea', 'gradle', 'src' (containing 'main', 'java', 'resources' with 'static' and 'templates', and 'test'), and '.gitattributes'. On the right, the 'application.properties' file is open, displaying configuration properties for a Spring application.

```

spring.application.name=proyectoFINAL
spring.datasource.url=jdbc:postgresql://localhost:5432/bdfinal
spring.datasource.username=usuario
spring.datasource.password=usuario
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true

```

## ➤ Crear las entidades

- Dentro del paquete com.tienda.proyectofinal.model, creamos la clase Usuario:

The screenshot shows the IntelliJ IDEA interface with the 'Usuario.java' file open in the editor. The code defines a Java class 'Usuario' with annotations from the jakarta.persistence, jakarta.validation.constraints, and lombok Data packages. The class has fields for id, nombre, email, and password, each annotated with @Id, @GeneratedValue(strategy = GenerationType.IDENTITY), @NotBlank, @Email, @NotBlank, @Column(unique = true), @Size(min = 6), and @NotBlank respectively.

```

package com.tienda.proyectofinal.model;

import jakarta.persistence.*;
import lombok.Data;
import jakarta.validation.constraints.*;

@Entity
@Data
public class Usuario {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @NotBlank
    private String nombre;

    @Email
    @NotBlank
    @Column(unique = true)
    private String email;

    @Size(min = 6)
    private String password;

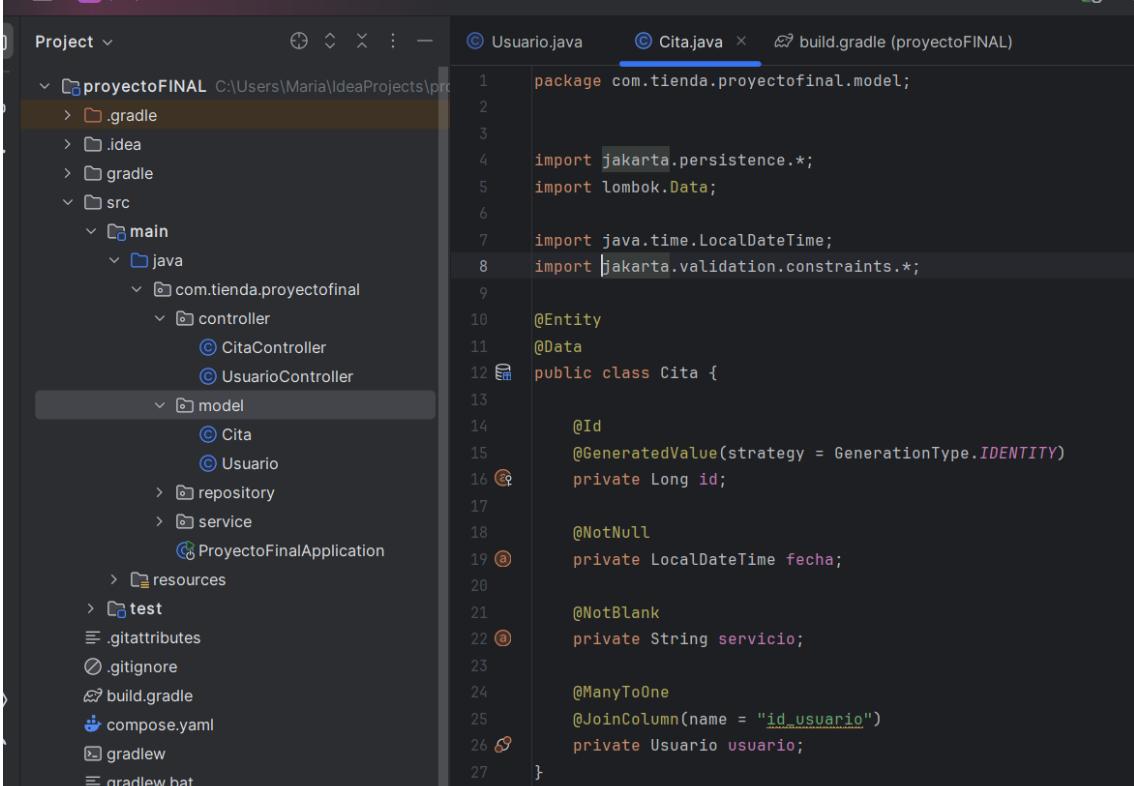
    @NotBlank
    private String rol;
}

```

Esta clase representa a los usuarios del sistema y está anotada con @Entity y @Table(name = "usuarios"), lo que indica que se mapea a una tabla llamada usuarios en la base de datos. Utiliza la anotación @Data para generar automáticamente los métodos get, set, toString, entre otros. Cada usuario tiene un id autogenerado, un nombre obligatorio (@NotBlank), un email único y

obligatorio validado con @Email, una password con un tamaño mínimo de 6 caracteres (validado con @Size) y un campo rol también obligatorio.

- En el mismo paquete creamos la clase Cita:



The screenshot shows the IntelliJ IDEA interface. On the left, the Project tool window displays the project structure for 'proyectoFINAL'. It includes a .gradle folder, an .idea folder, a gradle folder, a src folder containing main and test subfolders, and a resources folder. Under main, there are com.tienda.proyectofinal, controller, model, repository, service, and ProyectoFinalApplication. The model folder contains CitaController and UsuarioController, and the subfolder model contains Cita and Usuario. The right side of the screen shows the code editor for Cita.java. The code defines a class Cita with annotations for Entity, Data, and Id. It has fields for id (Long, @Id, @GeneratedValue(strategy = GenerationType.IDENTITY)), fecha (LocalDateTime, @NotNull), servicio (String, @NotBlank), and usuario (Usuario, @ManyToOne, @JoinColumn(name = "id\_usuario")).

```

package com.tienda.proyectofinal.model;

import jakarta.persistence.*;
import lombok.Data;
import java.time.LocalDateTime;
import jakarta.validation.constraints.*;

@Entity
@Data
public class Cita {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @NotNull
    private LocalDateTime fecha;

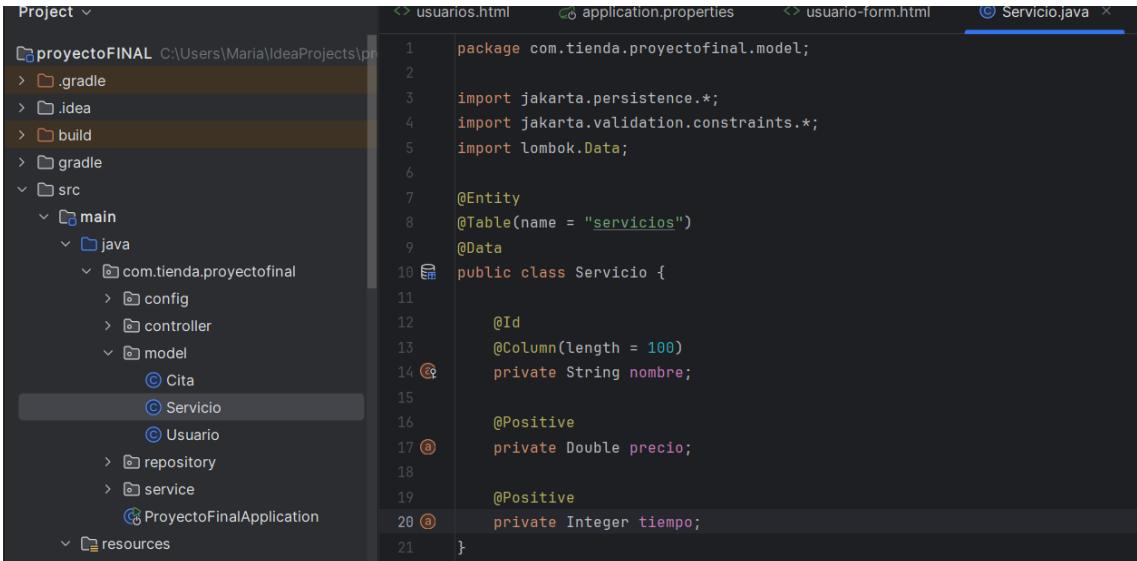
    @NotBlank
    private String servicio;

    @ManyToOne
    @JoinColumn(name = "id_usuario")
    private Usuario usuario;
}

```

La clase Cita representa las reservas o citas que los usuarios hacen para recibir un servicio. También está anotada con @Entity y @Table(name = "citas"). Contiene un id autogenerado como identificador principal. Cada cita está asociada a un Usuario mediante una relación @ManyToOne y una clave foránea llamada id\_usuario. La fecha y hora de la cita se almacenan en un campo fecha de tipo LocalDateTime, que no puede ser nulo (@NotNull). Además, cada cita está asociada a un Servicio también mediante @ManyToOne, usando como clave foránea el campo nombre del servicio

- Y por último creamos la clase Servicio:



The screenshot shows the IntelliJ IDEA interface. On the left, the project structure is displayed under 'Project'. It includes a .gradle folder, .idea folder, build folder, gradle folder, src folder (containing main/java/com/tienda/proyectofinal/config, controller, model, repository, service, and ProyectoFinalApplication), and resources folder. In the center, several tabs are open: usuarios.html, application.properties, usuario-form.html, and Servicio.java. The Servicio.java tab is active, showing the following Java code:

```

1 package com.tienda.proyectofinal.model;
2
3 import jakarta.persistence.*;
4 import jakarta.validation.constraints.*;
5 import lombok.Data;
6
7 @Entity
8 @Table(name = "servicios")
9 @Data
10 public class Servicio {
11
12     @Id
13     @Column(length = 100)
14     private String nombre;
15
16     @Positive
17     private Double precio;
18
19     @Positive
20     private Integer tiempo;
21 }

```

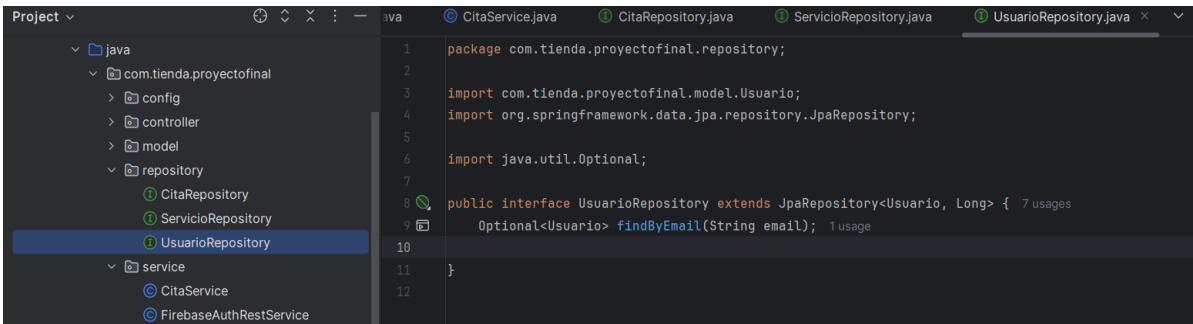
La clase Servicio representa los servicios que se ofrecen en la peluquería. Se mapea a la tabla servicios y también usa la anotación @Data. El campo nombre actúa como identificador (@Id) y tiene una longitud máxima de 100 caracteres. Cada servicio tiene un precio y un tiempo, ambos marcados con @Positive, lo que indica que deben ser valores mayores a cero.

## ➤ Crear los Repositorios

En Spring Boot, un repositorio es una interfaz que representa la capa de acceso a datos de tu aplicación. Se encarga de interactuar con la base de datos sin que tengas que escribir código SQL directamente. Se usa principalmente junto con Spring Data JPA (Persistencia), que genera automáticamente los métodos como findAll(), findById(), save(), deleteById() por lo que permite trabajar con bases de datos a través de objetos Java.

- Creamos una carpeta repository y dentro creamos la interfaz:

UsuarioRepository.java



The screenshot shows the IntelliJ IDEA interface. On the left, the project structure is displayed under 'Project'. It includes a java folder (containing com.tienda.proyectofinal/config, controller, model, repository, service, and FirebaseAuthRestService), and resources folder. In the center, several tabs are open: CitaService.java, CitaRepository.java, ServicioRepository.java, and UsuarioRepository.java. The UsuarioRepository.java tab is active, showing the following Java code:

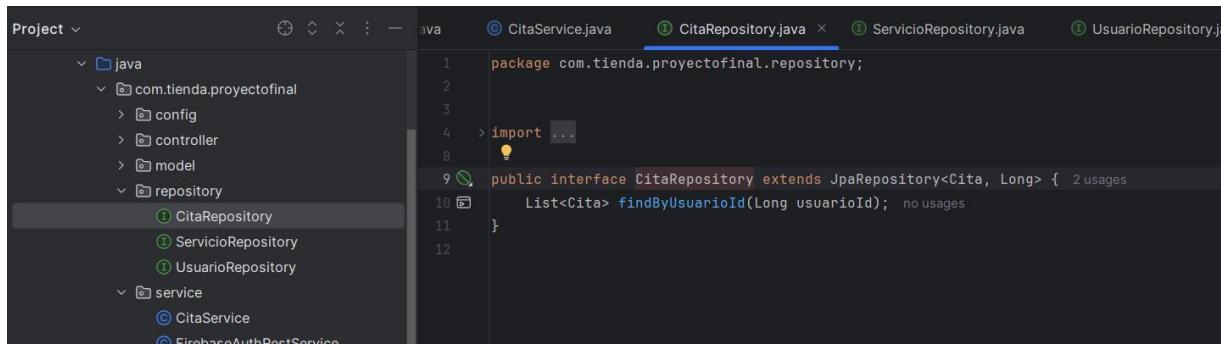
```

1 package com.tienda.proyectofinal.repository;
2
3 import com.tienda.proyectofinal.model.Usuario;
4 import org.springframework.data.jpa.repository.JpaRepository;
5
6 import java.util.Optional;
7
8 public interface UsuarioRepository extends JpaRepository<Usuario, Long> {
9     Optional<Usuario> findByEmail(String email);
10 }
11
12

```

- Creamos también la interfaz para cita:

### CitaRepository.java



```

package com.tienda.proyectofinal.repository;

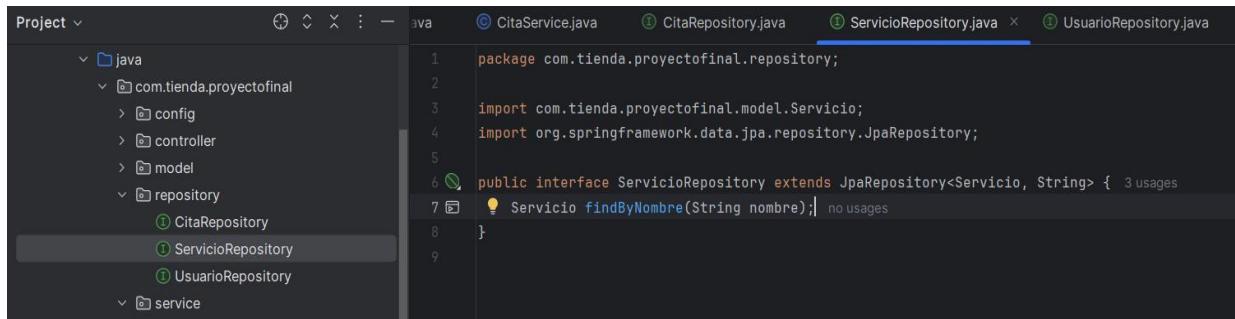
import org.springframework.data.jpa.repository.JpaRepository;
import com.tienda.proyectofinal.model.Cita;
import java.util.List;

public interface CitaRepository extends JpaRepository<Cita, Long> {
    List<Cita> findByUsuarioId(Long usuarioId);
}

```

- Y la interfaz para servicio:

### ServicioRepository.java



```

package com.tienda.proyectofinal.repository;

import com.tienda.proyectofinal.model.Servicio;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;

public interface ServicioRepository extends JpaRepository<Servicio, String> {
    Servicio findByNombre(@Param("nombre") String nombre);
}

```

## ➤ Crear los Controladores REST

Un controlador REST es una clase en Spring Boot que maneja las solicitudes HTTP (como GET, POST, PUT, DELETE) y devuelve respuestas en formato JSON o similar, en lugar de HTML.

- Creamos la carpeta controller  
En src/main/java/com/tienda/proyectofinal, creamos un nuevo paquete llamado controller
- Creamos UsuarioController.java

```

import ...
@RestController
@RequestMapping("/api/usuarios")
public class UsuarioController {

    @Autowired
    private UsuarioService usuarioService;

    @GetMapping
    public List<Usuario> getUsuarios() { return usuarioService.obtenerTodos(); }

    @PostMapping
    public ResponseEntity<Usuario> crearUsuario(@RequestBody Usuario usuario){
        return ResponseEntity.ok(usuarioService.guardar(usuario));
    }

    @DeleteMapping("/{id}")
    public void eliminarUsuario(@PathVariable Long id) { usuarioService.eliminar(id); }

    @PutMapping("/{id}")
    public ResponseEntity<Usuario> actualizarUsuario(@PathVariable Long id, @RequestBody Usuario usuario) {
        Usuario actualizado = usuarioService.actualizar(id, usuario);
        return ResponseEntity.ok(actualizado);
    }

    @GetMapping("/{id}")
    public ResponseEntity<Optional<Usuario>> obtenerUsuarioPorId(@PathVariable Long id) {
        return ResponseEntity.ok(usuarioService.obtenerPorId(id));
    }
}

```

- Creamos CitaController.java

```

@RestController
@RequestMapping("/api/citas")
public class CitaController {

    @Autowired
    private CitaService citaService;

    @GetMapping
    public ResponseEntity<List<Cita>> getCitas() {
        List<Cita> citas = citaService.obtenerTodas();
        return ResponseEntity.ok(citas);
    }

    @PostMapping
    public ResponseEntity<Cita> agendarCita(@Valid @RequestBody Cita cita) {
        try {
            Cita nuevaCita = citaService.guardar(cita);
            if (nuevaCita == null) {
                return ResponseEntity.badRequest().build(); // algo salió mal
            }
            System.out.println("RESPUESTA CITA => " + nuevaCita);
            return ResponseEntity.ok(nuevaCita);
        } catch (Exception e) {
            e.printStackTrace();
            return ResponseEntity.status(500).build();
        }
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<Void> eliminarCita(@PathVariable Long id) {
        try {
            citaService.eliminar(id);
            return ResponseEntity.noContent().build();
        } catch (Exception e) {
            return ResponseEntity.notFound().build();
        }
    }

    @PutMapping("/{id}")
    public ResponseEntity<Cita> actualizarCita(@PathVariable Long id, @Valid @RequestBody Cita cita) {
        try {
            Cita actualizada = citaService.actualizar(id, cita);
            return ResponseEntity.ok(actualizada);
        } catch (Exception e) {
            return ResponseEntity.notFound().build();
        }
    }

    @GetMapping("/usuario/{id}")
    public ResponseEntity<List<Cita>> getCitasPorUsuario(@PathVariable("id") Long usuarioId) {
        List<Cita> citas = citaService.obtenerCitasPorUsuario(usuarioId);
        return ResponseEntity.ok(citas);
    }
}

```

- Y creamos el último controlador ServicioController.java

The screenshot shows the IntelliJ IDEA interface with the project structure on the left and the code editor on the right. The code editor displays the `UsuarioController.java` file, which is annotated with `@RestController`. It contains methods for listing users, creating a user, deleting a user by ID, updating a user, and getting a user by ID. The controller interacts with a `UsuarioService` via `@Autowired`.

```

14
15     @RestController
16     @RequestMapping(value = "/api/usuarios")
17     public class UsuarioController {
18
19         @Autowired
20         private UsuarioService usuarioService;
21
22         @GetMapping
23         public List<Usuario> getUsuarios() { return usuarioService.obtenerTodos(); }
24
25         @PostMapping
26         public ResponseEntity<Usuario> crearUsuario(@RequestBody Usuario usuario) {
27             return ResponseEntity.ok(usuarioService.guardar(usuario));
28         }
29
30         @DeleteMapping("/{id}")
31         public void eliminarUsuario(@PathVariable Long id) { usuarioService.eliminar(id); }
32
33         @PutMapping("/{id}")
34         public ResponseEntity<Usuario> actualizarUsuario(@PathVariable Long id, @RequestBody Usuario usuario) {
35             Usuario actualizado = usuarioService.actualizar(id, usuario);
36             return ResponseEntity.ok(actualizado);
37         }
38         @GetMapping("/{id}")
39         public ResponseEntity<Optional<Usuario>> obtenerUsuarioPorId(@PathVariable Long id) {
40             return ResponseEntity.ok(usuarioService.obtenerPorId(id));
41         }
42     }
43
44
45
46
47

```

## ➤ Crear la capa de servicios

Un servicio en Spring Boot es una clase que contiene la lógica de negocio de la aplicación y se ubica entre el controlador (que recibe las peticiones) y el repositorio (que accede a la base de datos). Es decir, es donde se colocan las reglas, validaciones y procesos principales, separados del controlador y del repositorio y permite mantener el código limpio, organizado y reutilizable.

- Creamos el paquete service  
Dentro de `src/main/java/com/tienda/proyectofinal`, creamos una carpeta llamada Service
- Creamos `UsuarioService.java`

The screenshot shows the IntelliJ IDEA interface with the project structure on the left and the code editor on the right. The code editor displays the `UsuarioService.java` file, which is annotated with `@Service`. It contains methods for obtaining all users, getting a user by ID, saving a user, and deleting a user by ID. It uses a `UsuarioRepository` via `@Autowired`.

```

14
15     @Service
16     public class UsuarioService {
17
18         @Autowired
19         private UsuarioRepository usuarioRepository;
20
21         public List<Usuario> obtenerTodos() {
22             List<Usuario> usuarios = usuarioRepository.findAll();
23             System.out.println("Usuarios encontrados: " + usuarios.size());
24             return usuarios;
25         }
26
27         public Optional<Usuario> obtenerPorId(Long id) {
28             return usuarioRepository.findById(id);
29         }
30
31         public Usuario guardar(Usuario usuario) {
32             if (usuario.getId() == null) {
33                 // Usuario existente, buscar en la base de datos
34                 Usuario existente = usuarioRepository.findById(usuario.getId()).orElse(null);
35                 if (existente != null && (usuario.getPassword() == null || usuario.getPassword().isEmpty())) {
36                     usuario.setPassword(existente.getPassword());
37                 }
38             }
39             return usuarioRepository.save(usuario);
40         }
41
42         public void eliminar(Long id) { usuarioRepository.deleteById(id); }
43
44
45
46
47

```

```

41     public void eliminar(Long id) { usuarioRepository.deleteById(id); }
42
43     public Usuario actualizar(Long id, Usuario usuario) { 1 usage
44         Optional<Usuario> usuarioExistente = usuarioRepository.findById(id);
45         if (usuarioExistente.isPresent()) {
46             Usuario u = usuarioExistente.get();
47             u.setNombre(usuario.getNombre());
48             u.setEmail(usuario.getEmail());
49             u.setPassword(usuario.getPassword());
50             return usuarioRepository.save(u);
51         } else {
52             throw new ResponseStatusException(HttpStatus.NOT_FOUND, "Usuario no encontrado");
53         }
54     }
55
56 }
57
58 }
59
60

```

- Creamos CitaService.java

```

15     @Service 5 usages
16     public class CitaService {
17
18         @Autowired
19         private CitaRepository citaRepository;
20
21         @Autowired
22         private ServicioRepository servicioRepository;
23
24         public List<Cita> obtenerTodas() { 2 usages
25             List<Cita> citas = citaRepository.findAll();
26             System.out.println("Citas encontradas: " + citas.size());
27             return citas;
28         }
29
30         public Optional<Cita> obtenerPorId(Long id) { return citaRepository.findById(id); }
31
32
33         public Cita guardar(Cita cita) { 2 usages
34             String nombre = cita.getServicio().getNombre();
35             Servicio servicio = servicioRepository.findByNombre(nombre);
36
37             if (servicio == null) {
38                 throw new RuntimeException("Servicio no encontrado: " + nombre);
39             }
40
41             return citaRepository.save(cita);
42         }
43
44         public Cita guardar(Cita cita) { 2 usages
45             if (servicio == null) {
46                 throw new RuntimeException("Servicio no encontrado: " + nombre);
47             }
48
49             return citaRepository.save(cita);
50         }
51
52         public void eliminar(Long id) { 2 usages
53             citaRepository.deleteById(id);
54         }
55
56         public Cita actualizar(Long id, Cita citaActualizada) { 1 usage
57             return citaRepository.findById(id).map( Cita cita -> {
58                 cita.setFecha(citaActualizada.getFecha());
59                 cita.setServicio(citaActualizada.getServicio());
60                 cita.setUsuario(citaActualizada.getUsuario());
61                 return citaRepository.save(cita);
62             }).orElseThrow(() -> new RuntimeException("Cita no encontrada con id: " + id));
63         }
64
65         public List<Cita> obtenerCitasPorUsuario(Long usuarioId) { 1 usage
66             return citaRepository.findByUsuarioId(usuarioId);
67         }
68
69
70

```

- Creamos ServicioService.java

The screenshot shows a Java Spring Boot project named 'proyectoFINAL' in an IDE. The project structure on the left includes .gradle, .idea, build, gradle, and src folders. The src/main/java/com/tienda/proyectofinal directory contains config, controller, model, repository, service, and ProjetoFinalApplication classes. The service folder contains CitaService, FirebaseAuthRestService, FirebaseService, ServicioService, UsuarioService, and ProjetoFinalApplication classes. The code editor on the right displays the 'ServicioService.java' file with the following code:

```

import org.springframework.stereotype.Service;
import java.util.List;
@Service
public class ServicioService {
    private final ServicioRepository repo;
    public ServicioService(ServicioRepository repo) {
        this.repo = repo;
    }
    public List<Servicio> listar() {
        return repo.findAll();
    }
    public Servicio obtener(String nombre) {
        return repo.findById(nombre).orElseThrow();
    }
    public Servicio guardar(Servicio s) {
        return repo.save(s);
    }
    public void borrar(String nombre) {
        repo.deleteById(nombre);
    }
}

```

## FASE 3: Integrar Firebase Authentication en Spring Boot

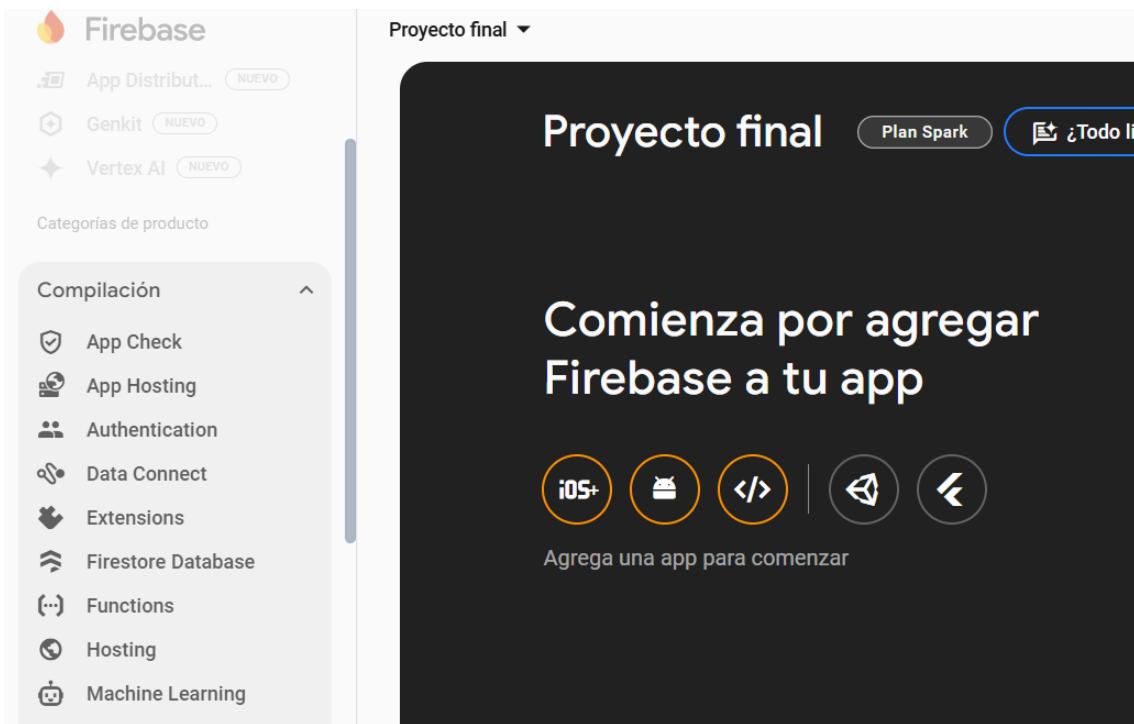
Integrar Firebase Authentication en una aplicación Spring Boot te permite usar el sistema de autenticación de Firebase (como login con email/contraseña, Google, Facebook, etc.) mientras sigues usando tu backend en Java.

### ➤ Crear un proyecto en Firebase

- Ir a <https://console.firebaseio.google.com/>



- Creamos un nuevo proyecto (Proyecto Final)
- Activamos **Authentication**:
  - Menú izquierdo → "Authentication" → "Empezar"



- En la pestaña **Métodos de inicio de sesión**
- Clic en **Agregar proveedor nuevo**
- Seleccionamos "**Correo electrónico/contraseña**".

- Generamos las credenciales:
  - Vamos a **Configuración**
  - Despu s vamos a la pesta a **Cuentas de Servicio**

The screenshot shows the 'Configuración de proyecto' (Project Configuration) screen in the Firebase console. The 'Cuentas de servicio' (Service Accounts) tab is active. On the left, there's a sidebar with a gear icon for settings and a 'NUEVA' (New) button. The main area has a title 'SDK de Firebase Admin' and a sub-section 'Secretos de la base de datos'. It lists 'Todas las cuentas de servicio' (All service accounts) and shows '2 cuentas de servicio' (2 service accounts). Below this, there's a section for 'Fragmento de configuración de SDK de Admin' (SDK Admin configuration snippet) with tabs for Node.js, Java, Python, and Go. The Java tab is selected, displaying the following code:

```

FileInputStream serviceAccount =
new FileInputStream("path/to/serviceAccountKey.json");

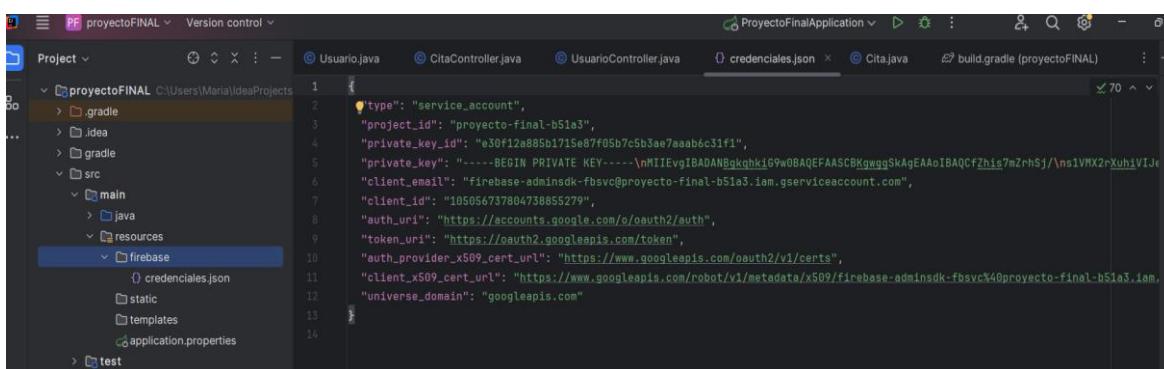
FirebaseOptions options = new FirebaseOptions.Builder()
.setCredentials(GoogleCredentials.fromStream(serviceAccount))
.build();

FirebaseApp.initializeApp(options);

```

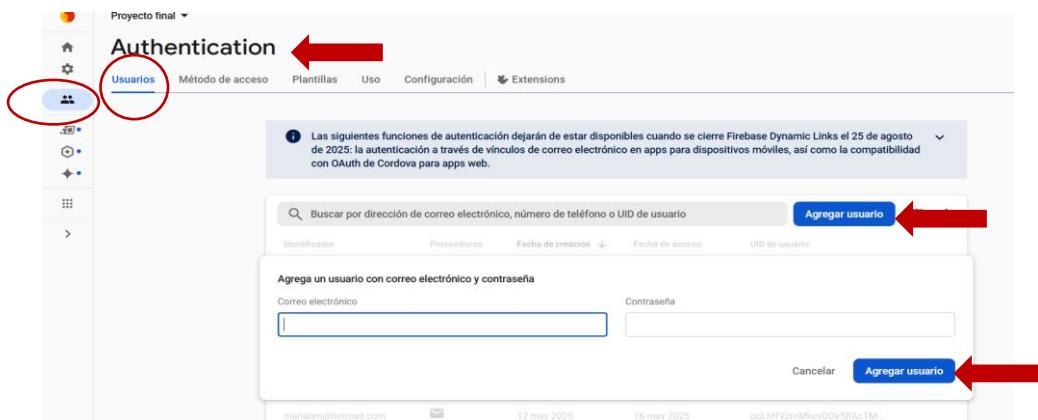
At the bottom right of this section is a blue 'Generar nueva clave privada' (Generate new private key) button.

- Seleccionamos la opción para Java
- Clic en **Generar nueva clave privada**
- Se descargará un archivo .json → Lo guardamos en el proyecto (`src/main/resources.firebaseio/credenciales.json`).



- Creamos el email y contraseña del administrador:
  - En el menú de la izquierda, clic en **Authentication**.
  - Luego, vamos a la pestaña **Usuarios** (arriba en la pantalla).
  - Clic en el botón “**Agregar usuario**”.

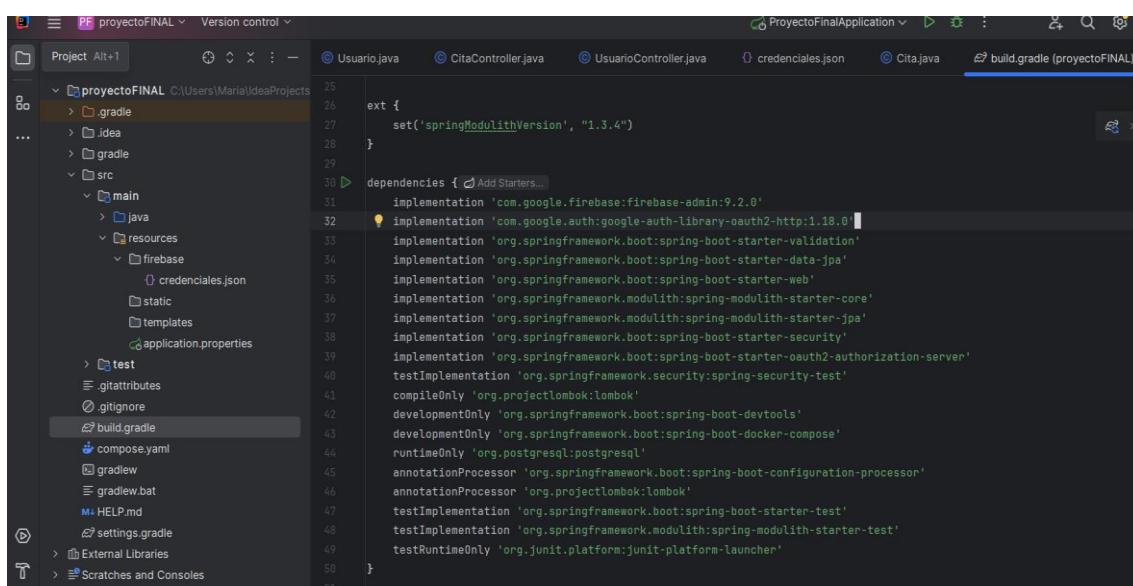
- Aparecerá un formulario:
  - Correo electrónico
  - Contraseña
- Clic en **Agregar usuario**



## ➤ Añadir dependencias Firebase en build.gradle

- Agregamos estas dependencias:

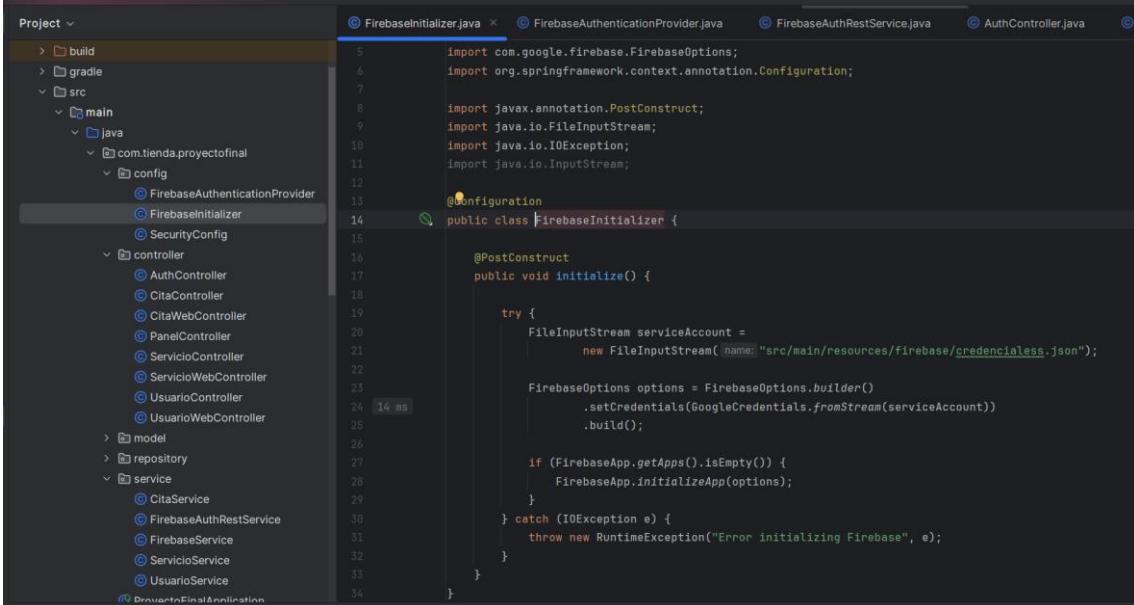
```
implementation 'com.google.firebaseio:firebase-admin:9.2.0'
implementation 'com.google.auth:google-auth-library-oauth2-
http:1.18.0'
implementation 'org.springframework.boot:spring-boot-starter-
security'
implementation 'org.springframework.boot:spring-boot-starter-
thymeleaf'
```



## ➤ Inicializar Firebase al arrancar la aplicación

Esta clase se encarga de inicializar la conexión con Firebase al arrancar la aplicación.

- Creamos la carpeta config
- Creamos dentro una clase FirebaseInitializer.java:



The screenshot shows the IntelliJ IDEA interface. On the left, the project structure is visible with packages like build, gradle, src, main, com.tienda.proyectofinal, config, controller, model, repository, service, and several controllers and services. The right pane displays the code for FirebaseInitializer.java:

```

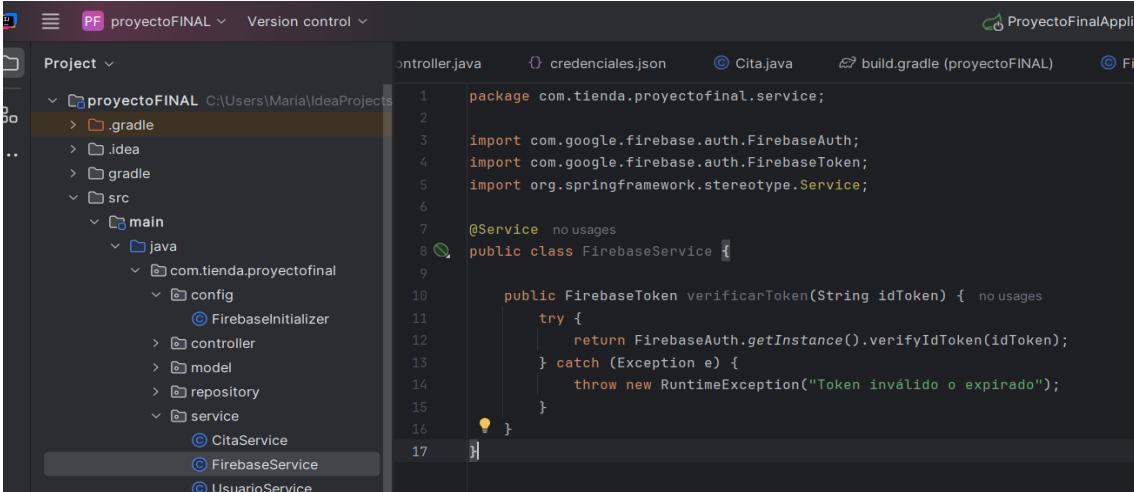
5 import com.google.firebaseio.FirebaseOptions;
6 import org.springframework.context.annotation.Configuration;
7
8 import javax.annotation.PostConstruct;
9 import java.io.FileInputStream;
10 import java.io.IOException;
11 import java.io.InputStream;
12
13 @Configuration
14 public class FirebaseInitializer {
15
16     @PostConstruct
17     public void initialize() {
18
19         try {
20             FileInputStream serviceAccount =
21                 new FileInputStream("src/main/resources/firebase/credencialess.json");
22
23             FirebaseOptions options = FirebaseOptions.builder()
24                 .setCredentials(GoogleCredentials.fromStream(serviceAccount))
25                 .build();
26
27             if (FirebaseApp.getApps().isEmpty()) {
28                 FirebaseApp.initializeApp(options);
29             }
30         } catch (IOException e) {
31             throw new RuntimeException("Error initializing Firebase", e);
32         }
33     }
34 }

```

## ➤ Crear un servicio para validar tokens de Firebase

Ahora, cuando el usuario se loguee, enviará un ID Token al backend, y el backend lo verificará.

- Creamos dentro del paquete service FirebaseService.java:



The screenshot shows the IntelliJ IDEA interface. On the left, the project structure is visible with packages like .gradle, .idea, gradle, src, main, com.tienda.proyectofinal, config, controller, model, repository, and service. The right pane displays the code for FirebaseService.java:

```

1 package com.tienda.proyectofinal.service;
2
3 import com.google.firebase.auth.FirebaseAuth;
4 import com.google.firebase.auth.FirebaseToken;
5 import org.springframework.stereotype.Service;
6
7 @Service no usages
8 public class FirebaseService {
9
10     public FirebaseToken verificarToken(String idToken) { no usages
11         try {
12             return FirebaseAuth.getInstance().verifyIdToken(idToken);
13         } catch (Exception e) {
14             throw new RuntimeException("Token inválido o expirado");
15         }
16     }
17 }

```

- Creamos FirebaseAuthRestService.java en service

El Admin SDK de Firebase no permite verificar contraseñas directamente (por seguridad), por eso necesitamos usar el API REST de Firebase Authentication para hacer login real con email y password.

The screenshot shows a Java project structure in an IDE. The project is named 'proyectoFINAL'. The 'src/main/java' package contains several classes: FirebaseAuthInitializer, SecurityConfig, CitaService, FirebaseAuthenticationProvider, FirebaseAuthRestService, FirebaseService, UsuarioService, and ProyectoFinalApplication. The 'FirebaseAuthRestService' class is selected and its code is displayed in the editor:

```

4 import org.springframework.stereotype.Service;
5 import org.springframework.web.client.RestTemplate;
6
7 import java.util.Collections;
8 import java.util.Map;
9
10 @Service no usages
11 public class FirebaseAuthRestService {
12
13     private final String API_KEY = "AIzaSyBtR0JjELDzNquW9EjhduIx08FU6RyoXpA"; // AQUI VA TU API KEY 1 usage
14     public boolean authenticate(String email, String password) { no usages
15         String url = "https://identitytoolkit.googleapis.com/v1/accounts:signInWithPassword?key=" + API_KEY;
16
17         Map<String, Object> request = Map.of(
18             "email", email,
19             "password", password,
20             "returnSecureToken", true
21         );
22
23         try {
24             RestTemplate restTemplate = new RestTemplate();
25             String response = restTemplate.postForObject(url, request, String.class);
26             JSONObject json = new JSONObject(response);
27
28             return json.has("idToken"); // login correcto
29         } catch (Exception e) {
30             return false; // login incorrecto
31         }
32     }
33 }

```

### ¿Dónde conseguir la API\_KEY?

En la consola de Firebase → Proyecto → Configuración → pestaña "General" → Sección "Tus apps" → Busca apiKey.

## ➤ Crear clases de seguridad

- Creamos la clase FirebaseAuthenticationProvider.java en config:

Es un proveedor de autenticación personalizado que integra Firebase Authentication (vía REST) con Spring Security. Implementa la interfaz AuthenticationProvider, lo cual permite que Spring Security lo use como parte del proceso de login.

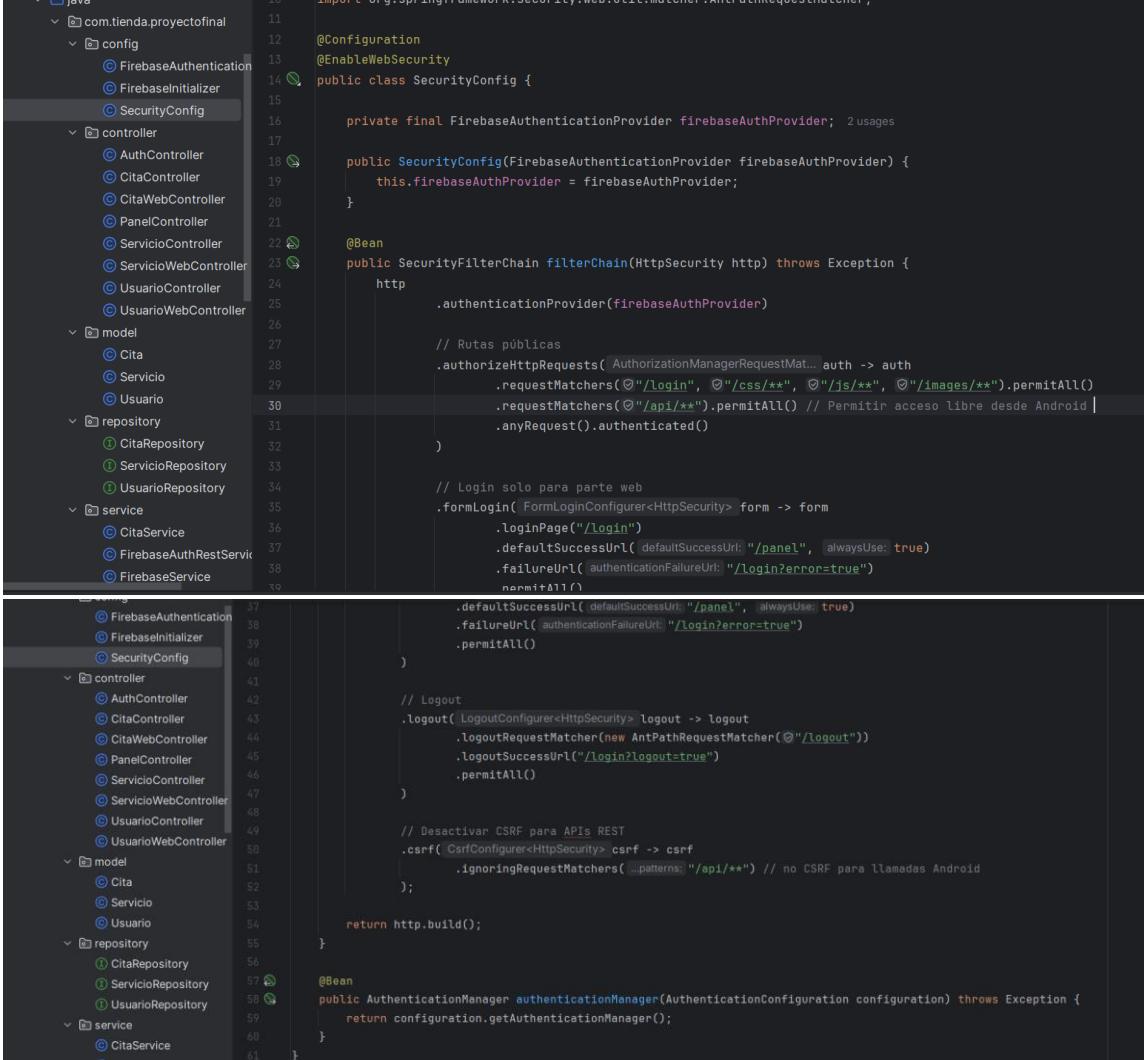
The screenshot shows the 'config' package in the 'src/main/java' directory. The 'FirebaseAuthenticationProvider' class is selected and its code is displayed in the editor:

```

14
15
16 @Component
17
18 public class FirebaseAuthenticationProvider implements AuthenticationProvider {
19
20     @Autowired
21     private FirebaseAuthRestService authRestService;
22
23     @Override
24     public Authentication authenticate(Authentication authentication) throws AuthenticationException {
25         String email = authentication.getName();
26         String password = authentication.getCredentials().toString();
27
28         if (authRestService.authenticate(email, password)) {
29             return new UsernamePasswordAuthenticationToken(
30                 email, password,
31                 Collections.singletonList(new SimpleGrantedAuthority( role: "USER" ))
32             );
33
34         }
35
36         throw new BadCredentialsException("Correo o contraseña inválidos");
37     }
38
39     @Override
40     public boolean supports(Class<?> authentication) {
41         return authentication.equals(UsernamePasswordAuthenticationToken.class);
42     }
43 }

```

- Después en la carpeta config también creamos SecurityConfig.java: Configura la seguridad de la aplicación en Spring Boot usando Spring Security. Define que el proveedor de autenticación será FirebaseAuthenticatorProvider, que permite validar usuarios con Firebase.



```

11  import org.springframework.security.config.annotation.web.builders.HttpSecurity;
12  import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
13  import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
14  import org.springframework.security.core.userdetails.UserDetailsService;
15
16  @Configuration
17  @EnableWebSecurity
18  public class SecurityConfig extends WebSecurityConfigurerAdapter {
19
20     private final FirebaseAuthenticatorProvider firebaseAuthProvider; 2 usages
21
22     public SecurityConfig(FirebaseAuthenticatorProvider firebaseAuthProvider) {
23         this.firebaseioAuthProvider = firebaseAuthProvider;
24     }
25
26     @Bean
27     public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
28         http
29             .authenticationProvider(firebaseAuthProvider)
30
31             // Rutas públicas
32             .authorizeHttpRequests( AuthorizationRequestMatcher auth -> auth
33                 .requestMatchers(@"/login", @"/css/**", @"/js/**", @"/images/**").permitAll()
34                 .requestMatchers(@"/api/**").permitAll() // Permitir acceso libre desde Android |
35                 .anyRequest().authenticated()
36
37             )
38
39             // Login solo para parte web
40             .formLogin( FormLoginConfigurer<HttpSecurity> form -> form
41                 .loginPage("/login")
42                 .defaultSuccessUrl( defaultSuccessUrl: "/panel", alwaysUse: true )
43                 .failureUrl( authenticationFailureUrl: "/login?error=true" )
44                 .normalizeUrl()
45
46                 .defaultSuccessUrl( defaultSuccessUrl: "/panel", alwaysUse: true )
47                 .failureUrl( authenticationFailureUrl: "/login?error=true" )
48                 .permitAll()
49
50             )
51
52             // Logout
53             .logout( LogoutConfigurer<HttpSecurity> logout -> logout
54                 .logoutRequestMatcher(new AntPathRequestMatcher(@"/logout"))
55                 .logoutSuccessUrl("/login?logout=true")
56                 .permitAll()
57
58             )
59
60             // Desactivar CSRF para APIs REST
61             .csrf( CsrfConfigurer<HttpSecurity> csrf -> csrf
62                 .ignoringRequestMatchers( ...patterns: "/api/**" ) // no CSRF para llamadas Android
63
64             );
65
66             return http.build();
67     }
68
69     @Bean
70     public AuthenticationManager authenticationManager(AuthenticationConfiguration configuration) throws Exception {
71         return configuration.getAuthenticationManager();
72     }
73
74 }

```

## ➤ Crear un controlador para probar autenticación

- Crea un controlador AuthController.java: AuthController sirve para validar si un token JWT de Firebase es auténtico y está activo. Este mecanismo se usa para proteger rutas de una API móvil o web y asegurarse de que el usuario está autenticado con Firebase.

```

1 package com.tienda.proyectofinal.controller;
2
3
4 import com.google.firebase.auth.FirebaseToken;
5 import com.tienda.proyectofinal.service.FirebaseService;
6 import org.springframework.web.bind.annotation.*;
7
8 @RestController
9 @RequestMapping("/auth")
10 public class AuthController {
11
12     private final FirebaseService firebaseService; 2 usages
13
14     public AuthController(FirebaseService firebaseService) {
15         this.firebaseioService = firebaseService;
16     }
17
18     @PostMapping("/verify")
19     public String verifyToken(@RequestHeader("Authorization") String authorizationHeader) {
20         String idToken = authorizationHeader.replace("Bearer ", "replacement");
21         FirebaseToken decodedToken = firebaseService.verificarToken(idToken);
22         return "Usuario verificado: " + decodedToken.getUid();
23     }
24 }

```

## FASE 4: Diseño Web con Thymeleaf

Thymeleaf es un motor de plantillas mucho más moderno y flexible que JSP, que se integra muy bien con Spring Boot. Thymeleaf te permite trabajar con vistas HTML y también puedes usarlo para generar dinámicamente contenido en el lado del servidor.

### ➤ Agregar dependencias de Thymeleaf en build.gradle

- Agregamos las siguientes dependencias en el `build.gradle` para habilitar Thymeleaf:

```

implementation 'org.springframework.boot:spring-boot-starter-web'
implementation 'org.springframework.boot:spring-boot-starter-thymeleaf'

```

### ➤ Crear nuevos controladores para manejar Thymeleaf

Para manejar las vistas con Thymeleaf, hay que crear controladores adicionales con la anotación `@Controller`, que será el encargado de renderizar las vistas HTML con Thymeleaf.

- Creamos `CitaWebController.java` en la carpeta controller

The screenshot shows a Java Spring application structure in an IDE. The project is named 'proyectoFINAL' and contains the following files:

- gradle**: build files.
- idea**: IDE configuration files.
- build**: build files.
- src**:
  - main**:
    - java**:
      - com.tienda.proyectofinal**:
        - config
        - controller (contains: AuthController, CitaController, CitaWebController, PanelController, ServicioController, ServicioWebController, UsuarioController, UsuarioWebController)
        - model
        - repository
        - service
        - ProyectoFinalApplication
  - resources**:
    - firebase
    - static
    - templates

**CitaWebController.java** code (partial view):

```

import org.springframework.web.bind.annotation.*;
import org.springframework.web.bind.annotation.*;

@Controller
@RequestMapping(@"/citas")
public class CitaWebController {

    private final CitaService citaService; 5 usages
    private final UsuarioService usuarioService; 4 usages
    private final ServicioService servicioService; 4 usages

    public CitaWebController(CitaService citaService,
                           UsuarioService usuarioService,
                           ServicioService servicioService) {
        this.citaService = citaService;
        this.usuarioService = usuarioService;
        this.servicioService = servicioService;
    }

    /* ----- LISTA ----- */
    @GetMapping(@"/")
    public String mostrarCitas(Model model) {
        model.addAttribute( attributeName: "citas", citaService.obtenerTodas());
        return "citas";
    }

    /* ----- NUEVA ----- */
    @GetMapping(@"/nueva")
    public String nueva(Model model) {
        model.addAttribute( attributeName: "cita", new Cita());
        model.addAttribute( attributeName: "usuarios", usuarioService.obtenerTodos());
    }

    public class CitaWebController {
        public String nueva(Model model) {
            model.addAttribute( attributeName: "usuarios", usuarioService.obtenerTodos());
            model.addAttribute( attributeName: "servicios", servicioService.listar());
            model.addAttribute( attributeName: "modo", attributeValue: "crear");
            return "cita-form";
        }

        /* ----- EDITAR ----- */
        @GetMapping(@"/editar/{id}")
        public String editar(@PathVariable Long id, Model model) {
            Cita cita = citaService.obtenerPorId(id)
                .orElseThrow(() -> new RuntimeException("Cita no encontrada id=" + id));
            model.addAttribute( attributeName: "cita", cita);
            model.addAttribute( attributeName: "usuarios", usuarioService.obtenerTodos());
            model.addAttribute( attributeName: "servicios", servicioService.listar());
            model.addAttribute( attributeName: "modo", attributeValue: "editar");
            return "cita-form";
        }

        /* ----- GUARDAR ----- */
        @PostMapping(@"/guardar")
        public String guardar(@Valid @ModelAttribute("cita") Cita cita,
                             BindingResult result, Model model) {
            if (result.hasErrors()) {
                model.addAttribute( attributeName: "usuarios", usuarioService.obtenerTodos());
                model.addAttribute( attributeName: "servicios", servicioService.listar());
                model.addAttribute( attributeName: "modo", (cita.getId() == null) ? "crear" : "editar");
                return "cita-form";
            }
            citaService.guardar(cita);
            return "redirect:/citas";
        }

        citaService.guardar(cita);
        return "redirect:/citas";
    }

    /* ----- ELIMINAR ----- */
    @PostMapping(@"/eliminar/{id}")
    public String eliminar(@PathVariable Long id) {
        citaService.eliminar(id);
        return "redirect:/citas";
    }
}

```

- También creamos UsuarioWebController.java en la misma carpeta

The screenshot shows the IntelliJ IDEA IDE with two panes. The left pane displays the project structure for 'proyectoFINAL' located at 'C:\Users\Maria\IdeaProjects\proyectoFINAL'. The right pane shows the code editor for the 'UsuarioWebController.java' file.

```

16  @Controller
17  @RequestMapping("/usuarios")
18  public class UsuarioWebController {
19
20      @Autowired
21      private UsuarioService usuarioService;
22
23      public UsuarioWebController(UsuarioService service) {
24          this.usuarioService = service;
25      }
26
27      /** LISTAR */
28      @GetMapping("/")
29      public String mostrarUsuarios(Model model) {
30          List<Usuario> usuarios = usuarioService.obtenerTodos();
31          System.out.println("Listado desde el controlador: " + usuarios.size());
32          model.addAttribute( attributeName: "usuarios", usuarios); // Pasa la lista de usuarios al modelo
33          return "usuarios"; // Nombre de la plantilla Thymeleaf (usuarios.html)
34      }
35
36      /** FORMULARIO CREAR */
37      @GetMapping("/nuevo")
38      public String formularioNuevo(Model model) {
39          model.addAttribute( attributeName: "usuario", new Usuario());
40          model.addAttribute( attributeName: "modo", attributeValue: "crear");
41          return "usuario-form";
42      }
43
44      /** FORMULARIO EDITAR */
45      @GetMapping("/editar/{id}")
46      public String formularioEditar(@PathVariable Long id, Model model) {
47          model.addAttribute( attributeName: "usuario", usuarioService.obtenerPorId(id));
48
49      }
50
51      /** GUARDAR (crear o actualizar) */
52      @PostMapping("/guardar")
53      public String guardar(@ModelAttribute @Valid Usuario usuario,
54                           BindingResult br,
55                           RedirectAttributes ra) {
56
57          if (br.hasErrors()) {
58              return "usuario-form";
59          }
60          usuarioService.guardar(usuario);
61          ra.addFlashAttribute( attributeName: "ok", attributeValue: "Usuario guardado");
62          return "redirect:/usuarios";
63
64      /** ELIMINAR */
65      @PostMapping("/eliminar/{id}")
66      public String eliminar(@PathVariable Long id, RedirectAttributes ra) {
67          usuarioService.eliminar(id);
68          ra.addFlashAttribute( attributeName: "ok", attributeValue: "Usuario eliminado");
69      }
70  }

```

- Y por último ServicioWebController.java

```

1 import org.springframework.web.bind.annotation.*;
2
3 @Controller
4 @RequestMapping("/servicios")
5 @RequiredArgsConstructor
6 public class ServicioWebController {
7
8     private final ServicioService servicioService;
9
10    @GetMapping()
11    public String lista(Model model){
12        model.addAttribute("servicios", servicioService.listar());
13        return "servicios";
14    }
15
16    @GetMapping("/nuevo")
17    public String nuevo(Model m){
18        m.addAttribute("servicio", new Servicio()); m.addAttribute("modo", "crear");
19        return "servicio-form";
20    }
21
22    @GetMapping("/{id}")
23    public String editar(@PathVariable String id, Model m){
24        m.addAttribute("servicio", servicioService.obtener(id));
25        m.addAttribute("modo", "editar"); return "servicio-form";
26    }
27
28    @PostMapping("/guardar")
29    public String guardar(@ModelAttribute("servicio") @Valid Servicio s, BindingResult br, Model m){
30        if(br.hasErrors()){
31            m.addAttribute("error", "Por favor, rellena todos los campos");
32        } else if(servicioService.guardar(s) == null) {
33            m.addAttribute("error", "El servicio ya existe");
34        } else {
35            servicioService.guardar(s);
36            return "redirect:/servicios";
37        }
38    }
39
40    @GetMapping("/{id}/borrar")
41    public String borrar(@PathVariable String id){ servicioService.borrar(id); return "redirect:/servicios"; }
42
43}
44
45
46
47
48
49
50

```

## ➤ Creación de las vistas html

- En main/resources/template creamos un html usuarios.htm:

```

1 <html xmlns:th="http://www.thymeleaf.org">
2     <body>
3         <h1>Usuarios</h1>
4         <div class="card">
5             <table>
6                 <thead>
7                     <tr>
8                         <th>Nombre</th>
9                         <th>Email</th>
10                        <th>Tipo</th>
11                        <th style="text-align:center;">Acciones</th>
12                    </tr>
13                </thead>
14                <tbody>
15                    <tr th:each="u : ${usuarios}">
16                        <td th:text="${u.nombre}">Nombre</td>
17                        <td th:text="${u.email}">email</td>
18                        <td th:text="${u.rol}">email</td>
19                        <td style="text-align:center;white-space:nowrap">
20                            <a href="#" class="btn blue" style="color:white; border-radius: 50%; padding: 5px 10px; font-size: 1em; margin-right: 10px;">NºSEditar</a>
21
22                            <form th:action="@{/usuarios/eliminar/" + ${u.id}]" method="post" style="display:inline">
23                                <input type="hidden" th:name="${_csrf.parameterName}" th:value="${_csrf.token}">
24                                <button type="submit" class="btn red" style="border-radius: 50%; padding: 5px 10px; font-size: 1em; color:white; background-color: #e74c3c; margin-right: 10px;">Borrar</button>
25                            </form>
26                        </td>
27                    </tr>
28                </tbody>
29            </table>
30        </div>
31
32        <div class="actions-bottom">
33            <a href="#" class="btn btn-volver" style="margin-right: 10px;">Volver</a>
34            <a href="#" class="btn btn-add" style="border-radius: 50%; padding: 5px 10px; font-size: 1em; color:white; background-color: #e74c3c; margin-right: 10px;">Nuevo usuario</a>
35        </div>
36    </body>
37</html>

```

- También creamos en el mismo sitio usuario-form.html que es el formulario para crear o editar los usuarios al cual se accede desde los botones de la pantalla usuarios.html

The screenshot shows the IntelliJ IDEA interface with the project structure on the left and the code editor on the right.

**Project Structure:**

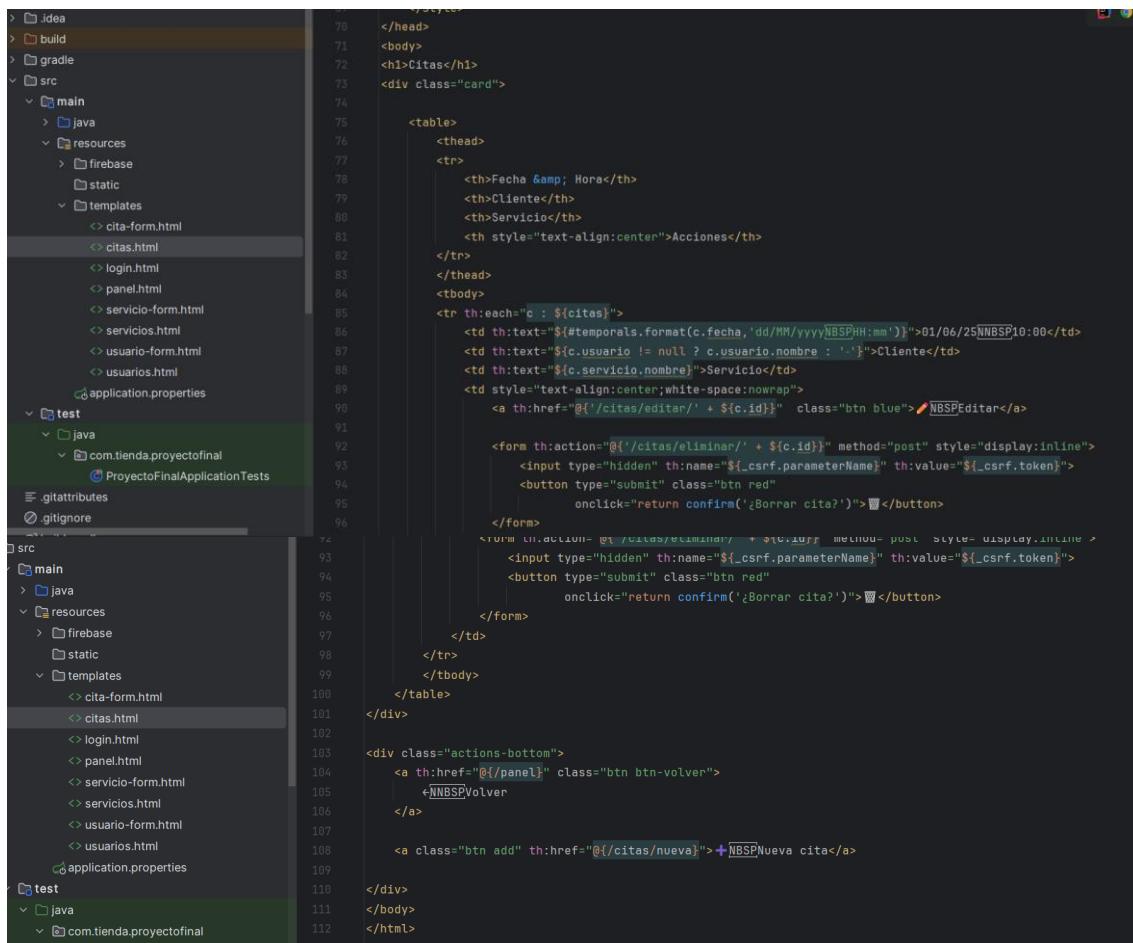
- src/main/resources/templates:** Contains files like cita-form.html, citas.html, login.html, panel.html, servicio-form.html, servicios.html, and usuario-form.html.
- src/test/java/com.tienda.proyectofinal:** Contains the ProyectoFinalApplicationTests class.
- application.properties**: Configuration properties.
- .gitattributes** and **.gitignore**: Version control files.

**Code Editor (usuario-form.html):**

```

1  <!DOCTYPE html>
2  <html xmlns:th="http://www.thymeleaf.org" lang="es">
3  <head>
4      <meta charset="UTF-8">
5      <title th:text="${modo == 'crear' ? 'Nuevo Usuario' : 'Editar Usuario'}>Formulario Usuario</title>
6
7      <!-- Bootstrap 5 (imprescindible para que .btn funcione!) -->
8      <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet">
9
10     <style>
11         body{font-family: 'Segoe UI', sans-serif;background:#eef2f7;display:flex;justify-content:center}
12
13         .btn-primary{background:#20d3b2; border:none}
14         .btn-primary:hover{background:#18a884}
15         .error{color:#d32f2f;font-size:.85rem; margin:-.5rem 0 1rem 0}
16     </style>
17 </head>
18 <body>
19     <div class="container py-4">
20         <div class="row justify-content-center">
21             <div class="col-md-6 col-lg-5">
22                 <div class="card p-4">
23
24                     <h3 class="text-center mb-3" th:text="${modo == 'crear' ? 'Nuevo Usuario' : 'Editar Usuario'}>Formulario Usuario</h3>
25
26                     <form th:action="@{/usuarios/guardar}" th:object="${usuario}" method="post">
27                         <input type="hidden" th:if="*{id}" th:field="*{id}"/>
28
29                         <!-- Nombre -->
30                         <div class="mb-3">
31                             <label class="form-label fw-semibold">Nombre</label>
32                             <input type="text" class="form-control" th:field="*{nombre}" required>
33                             <div class="error" th:errors="*{nombre}"></div>
34                         </div>
35
36                         <!-- Email -->
37                         <div class="mb-3">
38                             <label class="form-label fw-semibold">Email</label>
39                             <input type="email" class="form-control" th:field="*{email}" required>
40                             <div class="error" th:errors="*{email}"></div>
41                         </div>
42
43                         <!-- Password -->
44                         <div class="mb-3" th:if="${modo == 'crear'}">
45                             <label class="form-label fw-semibold">Contraseña</label>
46                             <input type="password" class="form-control" th:field="*{password}" required>
47                             <div class="error" th:errors="*{password}"></div>
48                         </div>
49
50                         <!-- Rol -->
51                         <div class="mb-3">
52
53                             <label class="form-label fw-semibold">Rol</label>
54                             <select class="form-select" th:field="*{rol}" required>
55                                 <option value="ADMIN">ADMIN</option>
56                                 <option value="TRABAJADOR">TRABAJADOR</option>
57                                 <option value="CLIENTE">CLIENTE</option>
58                             </select>
59                         </div>
60
61                         <!-- Botones -->
62                         <div class="d-flex justify-content-between">
63                             <a class="btn btn-outline-secondary" th:href="@{/usuarios}">Cancelar</a>
64                             <button type="submit" class="btn btn-primary">
65                                 <i class="bi bi-check-circle me-1"></i>
66                                 <span th:text="${modo == 'crear' ? 'Guardar' : 'Actualizar'}>Guardar</span>
67                             </button>
68                         </div>
69
70                     </form>
71
72                 </div>
73             </div>
74         </div>
75     </div>
76
77 
```

- De la misma forma creamos citas.html:



The screenshot shows the project structure on the left and the code for 'citas.html' on the right. The code is a Thymeleaf template for displaying a list of appointments. It includes a table with columns for Date & Time, Client, Service, and Actions (with a link to 'editar'). It also contains a form for deleting an appointment with a confirmation dialog.

```

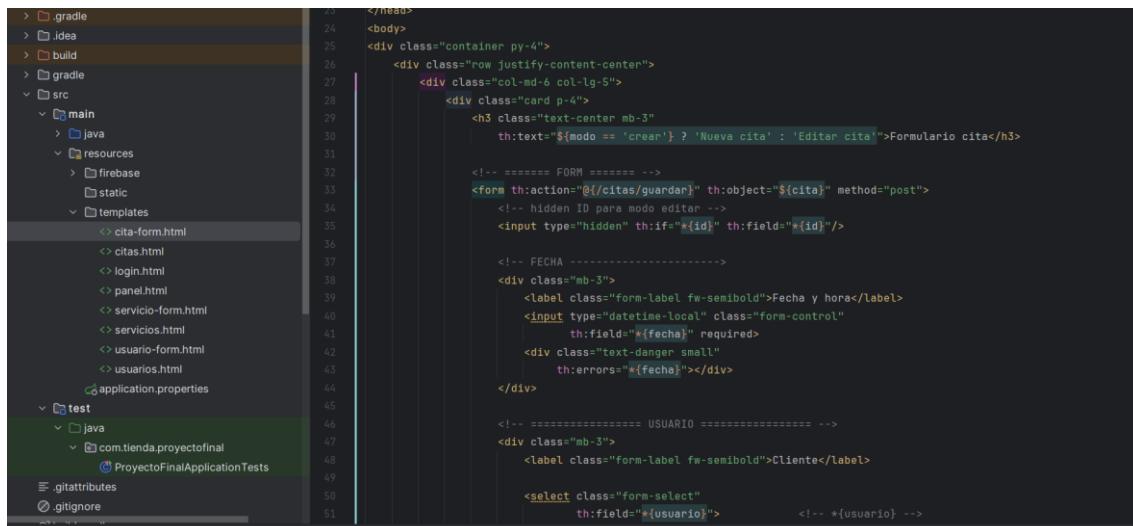
</head>
<body>
<h1>Citas</h1>
<div class="card">

    <table>
        <thead>
            <tr>
                <th>Fecha & Hora</th>
                <th>Cliente</th>
                <th>Servicio</th>
                <th style="text-align:center">Acciones</th>
            </tr>
        </thead>
        <tbody>
            <tr th:each="c : ${citas}">
                <td th:text="${temporals.format(c.fecha, 'dd/MM/yyyy HH:mm')}>01/06/25NBSF10:00</td>
                <td th:text="${c.usuario.nombre} != null ? c.usuario.nombre : '-'>Cliente</td>
                <td th:text="${c.servicio.nombre}">Servicio</td>
                <td style="text-align:center;white-space: nowrap">
                    <a th:href="@{/citas/editar/' + ${c.id}]" class="btn blue">NBSPEditar</a>
                    <form th:action="@{/citas/eliminar/' + ${c.id}]" method="post" style="display:inline">
                        <input type="hidden" th:name="${_csrf.parameterName}" th:value="${_csrf.token}">
                        <button type="submit" class="btn red" onclick="return confirm('¿Borrar cita?')">Borrar</button>
                    </form>
                    <form th:action="@{/citas/eliminar/' + ${c.id}]" method="post" style="display:inline">
                        <input type="hidden" th:name="${_csrf.parameterName}" th:value="${_csrf.token}">
                        <button type="submit" class="btn red" onclick="return confirm('¿Borrar cita?')">Borrar</button>
                    </form>
                </td>
            </tr>
        </tbody>
    </table>
</div>

<div class="actions-bottom">
    <a th:href="@{/panel}" class="btn btn-volver">Volver</a>
    <a class="btn add" th:href="@{/citas/nueva}">+ NBSPNueva cita</a>
</div>
</body>
</html>

```

- Y cita-form.html:

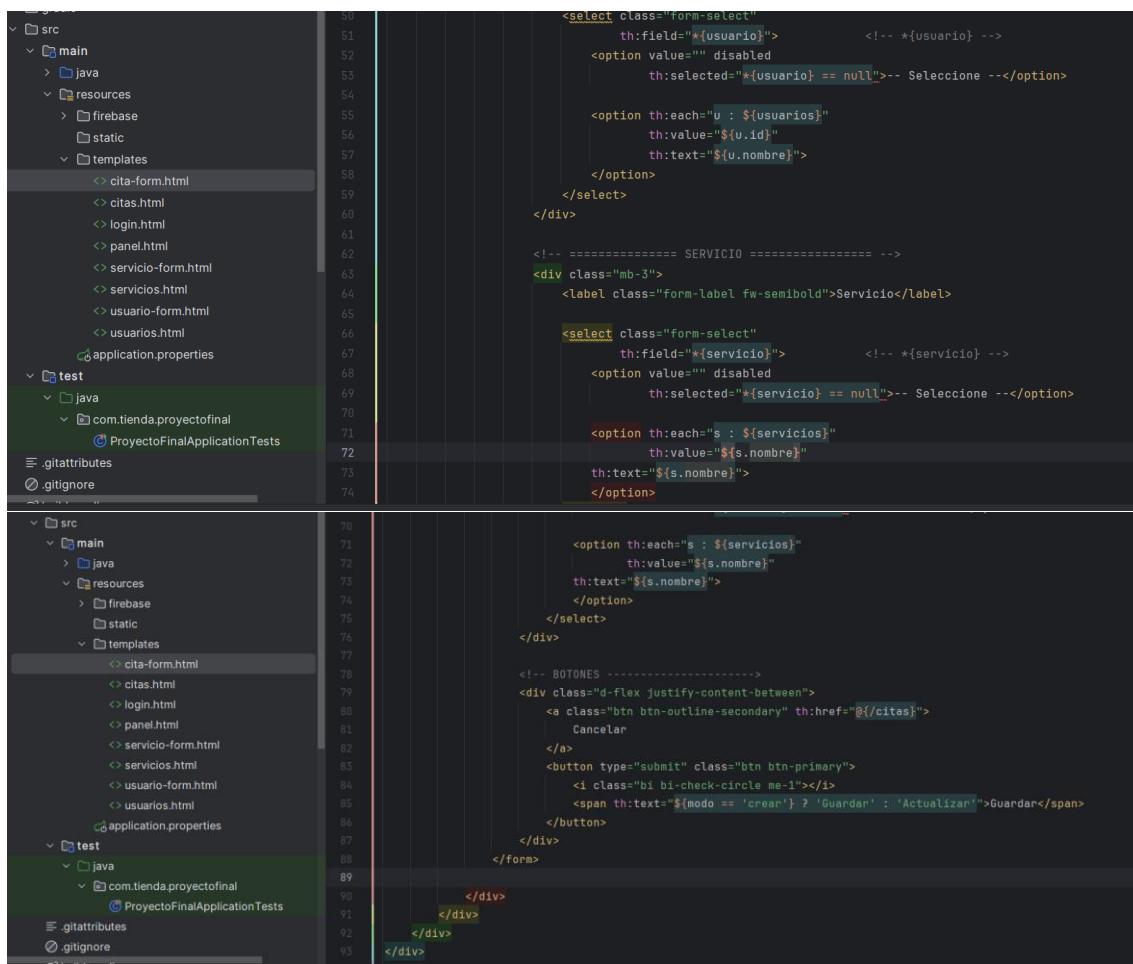


The screenshot shows the project structure on the left and the code for 'cita-form.html' on the right. This is a Thymeleaf template for a form to create or edit an appointment. It includes fields for date and time (using a datetime-local input), a client (select dropdown), and a service (select dropdown).

```

</head>
<body>
<div class="container py-4">
    <div class="row justify-content-center">
        <div class="col-md-6 col-lg-5">
            <div class="card p-4">
                <h3 class="text-center mb-3" th:text="${modo == 'crear' ? 'Nueva cita' : 'Editar cita'}>Formulario cita</h3>
                <!-- ===== FORM ===== -->
                <form th:action="@{/citas/guardar}" th:object="${cita}" method="post">
                    <!-- hidden ID para modo editar -->
                    <input type="hidden" th:if="#{id}" th:field="#{id}" />
                    <!-- FECHA ----->
                    <div class="mb-3">
                        <label class="form-label fw-semibold">Fecha y hora</label>
                        <input type="datetime-local" class="form-control" th:field="#{fecha}" required>
                        <div class="text-danger small" th:errors="*{fecha}"></div>
                    </div>
                    <!-- ===== USUARIO ===== -->
                    <div class="mb-3">
                        <label class="form-label fw-semibold">Cliente</label>
                        <select class="form-select" th:field="#{usuario}">
                            <!-- *{usuario} -->
                        </select>
                    </div>
                </form>
            </div>
        </div>
    </div>
</div>

```



The screenshot shows a file tree on the left and a code editor on the right. The file tree includes 'src/main/java', 'src/main/resources/firebase', 'src/main/templates', 'src/test/java', and test files like 'ProyectoFinalApplicationTests'. The code editor displays the 'servicios.html' template, which contains Thymeleaf syntax for dropdown menus and buttons.

```

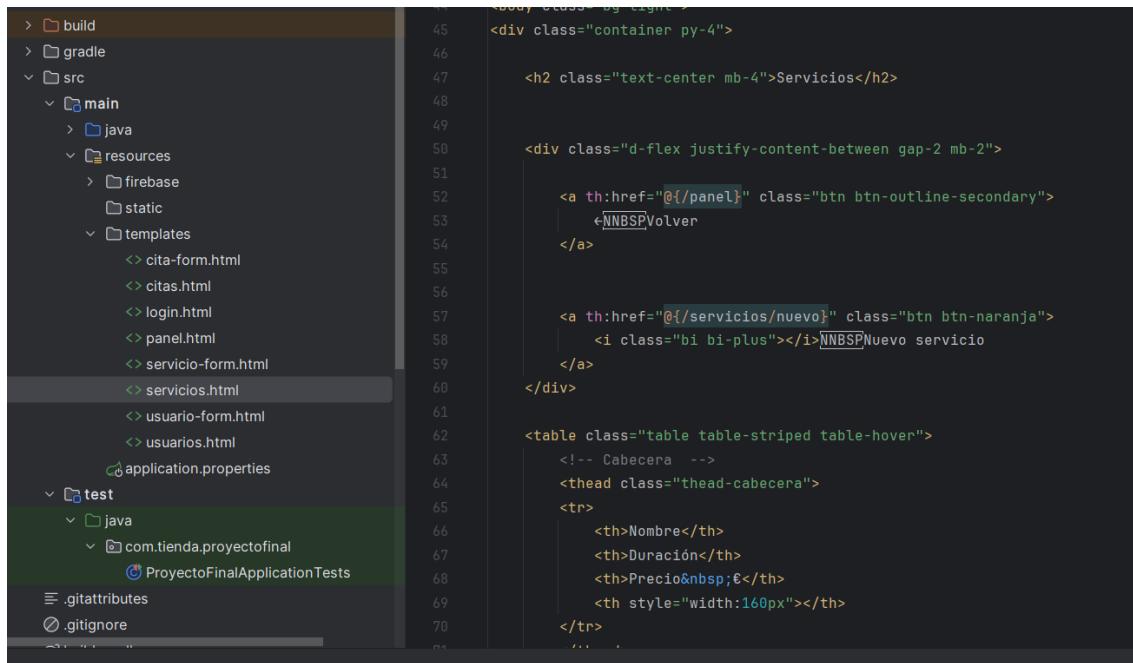
<select class="form-select" th:field="#{usuario}">
    <!-- *{usuario} -->
    <option value="" disabled th:selected="*{usuario} == null">-- Seleccione --</option>
    <option th:each="u : ${usuarios}" th:value="${u.id}" th:text="${u.nombre}">
    </option>
</select>

<!-- ===== SERVICIO ===== -->
<div class="mb-3">
    <label class="form-label fw-semibold">Servicio</label>
    <select class="form-select" th:field="#{servicio}">
        <!-- *{servicio} -->
        <option value="" disabled th:selected="*{servicio} == null">-- Seleccione --</option>
        <option th:each="s : ${servicios}" th:value="${s.nombre}" th:text="${s.nombre}">
        </option>
    </select>
</div>

<!-- BOTONES ----->
<div class="d-flex justify-content-between">
    <a class="btn btn-outline-secondary" th:href="@{/citas}">
        Cancelar
    </a>
    <button type="submit" class="btn btn-primary">
        <i class="bi bi-check-circle me-1"></i>
        <span th:text="${modo == 'crear'} ? 'Guardar' : 'Actualizar'">Guardar</span>
    </button>
</div>

```

- Hacemos lo mismo para servicios
- Creamos servicios.html:



The screenshot shows a file tree on the left and a code editor on the right. The file tree includes 'src/main/resources/firebase', 'src/main/templates', and 'src/test/java'. The code editor displays the 'servicios.html' template, which contains Thymeleaf syntax for a service creation form and a table.

```

<div class="container py-4">
    <h2 class="text-center mb-4">Servicios</h2>
    <div class="d-flex justify-content-between gap-2 mb-2">
        <a th:href="@{/panel}" class="btn btn-outline-secondary">
            &NBSP;Volver
        </a>
        <a th:href="@{/servicios/nuevo}" class="btn btn-naranja">
            <i class="bi bi-plus"></i>&NBSP;Nuevo servicio
        </a>
    </div>
    <table class="table table-striped table-hover">
        <!-- Cabecera -->
        <thead class="thead-cabecera">
            <tr>
                <th>Nombre</th>
                <th>Duración</th>
                <th>Precio&nbsp;€</th>
                <th style="width:160px;"></th>
            </tr>
            ...
        </thead>
        <tbody>
            ...
        </tbody>
    </table>

```

The screenshot shows a file explorer on the left and a code editor on the right. The file explorer displays a project structure with folders like 'gradle', 'src' (containing 'main' and 'test'), and files like 'application.properties'. The code editor shows a Thymeleaf template for displaying services. The template uses Thymeleaf syntax to iterate over a list of services and render their details. It includes buttons for editing and deleting services.

```

    </thead>
    <tbody>
        <tr th:each="s : ${servicios}">
            <td th:text="${s.nombre}"></td>
            <td th:text="${s.tiempo + ' min'}"></td>
            <td th:text="#{numbers.formatDecimal(s.precio,1,2)}"></td>
            <td class="text-end">
                <!-- Botón EDITAR (coral) -->
                <a th:href="@{/servicios/editar/" + ${s.nombre}]" class="btn btn-sm btn-edit me-1">
                     Editar
                </a>
                <!-- Botón BORRAR (mantiene rojo Bootstrap) -->
                <a th:href="@{/servicios/borrar/" + ${s.nombre}]" class="btn btn-sm btn-danger" onclick="return confirm('¿Eliminar servicio?');">
                     Borrar
                </a>
            </td>
        </tr>
    </tbody>
</table>

```

- Y servicio-form:

The screenshot shows a file explorer on the left and a code editor on the right. The file explorer displays a project structure with folders like 'gradle', 'src' (containing 'main' and 'test'), and files like 'application.properties'. The code editor shows a Thymeleaf template for creating or updating a service. It includes fields for name, duration, and price, and buttons for canceling or saving the changes.

```

    <!DOCTYPE html><html xmlns:th="http://www.thymeleaf.org">
    <head><meta charset="UTF-8"/><title th:text="${modo=='crear'?'Nuevo servicio':'Editar servicio'}"></title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet"/>
    </head>
    <body class="bg-light">
        <div class="container py-4">
            <div class="row justify-content-center">
                <div class="col-md-6 col-lg-5">
                    <div class="card p-4 shadow-sm">
                        <h3 class="text-center mb-3" th:text="${modo=='crear'?'Nuevo servicio':'Editar servicio'}"></h3>
                        <form th:action="@{/servicios/guardar}" th:object="${servicio}" method="post">
                            <!-- Nombre (PK) no editable en modo editar -->
                            <div class="mb-3">
                                <label class="form-label fw-semibold">Nombre</label>
                                <input type="text" class="form-control" th:field="*{nombre}" th:attr="readonly=${modo=='crear'}" required/>
                                <div class="text-danger small" th:errors="*{nombre}"></div>
                            </div>
                            <!-- Duración -->
                            <div class="mb-3">
                                <label class="form-label fw-semibold">Duración (min.)</label>
                                <input type="number" min="1" class="form-control" th:field="*{tiempo}" required/>
                                <div class="text-danger small" th:errors="*{tiempo}"></div>
                            </div>
                            <!-- Precio -->
                            <div class="mb-3">
                                <label class="form-label fw-semibold">Precio (€)</label>
                            </div>
                            <!-- Precio -->
                            <div class="mb-3">
                                <label class="form-label fw-semibold">Precio (€)</label>
                                <input type="number" step="0.01" min="0" class="form-control" th:field="*{precio}" required/>
                                <div class="text-danger small" th:errors="*{precio}"></div>
                            </div>
                            <!-- BOTONES -->
                            <div class="d-flex justify-content-between">
                                <a class="btn btn-outline-secondary" th:href="@{/servicios}">Cancelar</a>
                                <button class="btn btn-primary" type="submit">
                                    <i class="bi bi-check-circle me-1"></i>
                                    <span th:text="${modo=='crear'?'Guardar':'Actualizar'}"></span>
                                </button>
                            </div>
                        </form>
                    </div>
                </div>
            </div>
        </div>
    </body>
</html>

```

- Además, hemos creado una pantalla login.html donde el administrador introducirá el correo y la contraseña para autenticarse:

The screenshot shows the project structure on the left and the code editor on the right. The project structure includes resources, firebase, static, templates (with files like cita-form.html, citas.html, login.html, panel.html, servicio-form.html, servicios.html, usuario-form.html, usuarios.html), application.properties, test, and Java files under com.tienda.proyectofinal. The code editor displays the content of login.html, which contains form fields for email and password, and messages for incorrect credentials or logout.

```

    ...
    </head>
    <body>
    <form th:action="@{/login}" method="post">
        <h2>Iniciar sesión</h2>
        <label>Email:</label>
        <input type="email" name="username"/>
        <label>Contraseña:</label>
        <input type="password" name="password"/>
        <button type="submit">Entrar</button>
    </form>
    <div class="message" th:if="${param.error}">
        Usuario o contraseña incorrectos
    </div>
    <div class="message" th:if="${param.logout}" style="color: green;">
        Has cerrado sesión
    </div>
</body>
</html>

```

- Y también hemos creado un panel.html al que se accede inmediatamente después de autenticarse y es donde están los botones para acceder a los usuarios, citas y servicios además de cerrar sesión:

The screenshot shows the project structure on the left and the code editor on the right. The project structure is identical to the previous one. The code editor displays the content of panel.html, which includes a header with 'Bienvenido al panel', a main content section with links to Usuarios, Citas, and Servicios, and a footer with a logout form.

```

    ...
    <body>
    <header>
        <h1>Bienvenido al panel</h1>
    </header>

    <div class="main-content">
        <div class="links">
            <a class="tile" th:href="@{/usuarios}">Usuarios</a>
            <a class="tile" th:href="@{/citas}">Citas</a>
            <a class="tile" th:href="@{/servicios}">Servicios</a>
        </div>
    </div>

    <footer>
        <form th:action="@{/logout}" method="post" class="logout-form">
            <input type="hidden" th:name="${_csrf.parameterName}" th:value="${_csrf.token}">
            <button type="submit" class="logout-button">Cerrar sesión</button>
        </form>
    </footer>
</body>
</html>

```

- Como consecuencia de haber creado panel.html, tenemos que crear el controlador PanelController.java en la carpeta controller (no hace falta crear el controlador para login.html porque Spring Security proporciona automáticamente una página de login en la ruta /login)

The screenshot shows the project structure on the left and the code editor on the right. The project structure includes .gradle, .idea, build, gradle, and src (with main, java, and com.tienda.proyectofinal). The code editor displays the content of PanelController.java, which defines two methods: panel() mapped to /panel and login() mapped to /Login.

```

    ...
    package com.tienda.proyectofinal.controller;

    import org.springframework.stereotype.Controller;
    import org.springframework.web.bind.annotation.GetMapping;

    @Controller
    public class PanelController {

        @GetMapping("/panel")
        public String panel() {
            return "panel";
        }

        @GetMapping("/Login")
        public String login() {
            return "Login";
        }
    }

```

## 2ºPARTE

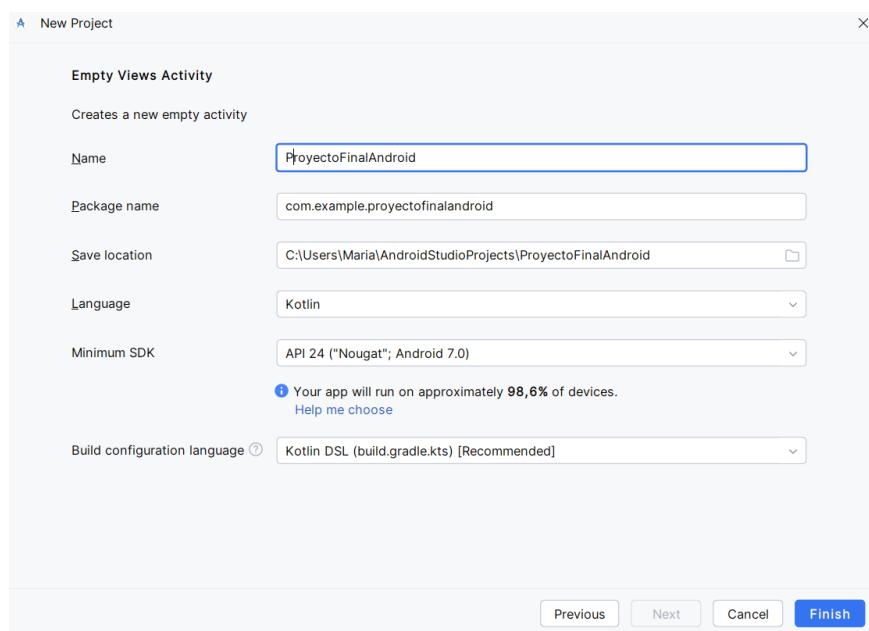
### FASE 1: Crear proyecto en Android Studio

- Abrimos Android Studio y seleccionamos:

File > New > New Project

- Elegimos "**Empty Views Activity**" y hacer clic en "Next".

- Rellenamos los campos



- Le damos a Finish
- Configurar dependencias necesarias en build.gradle:

```
1  plugins {
2      alias(libs.plugins.android.application)
3      alias(libs.plugins.kotlin.android)
4
5      id("com.google.gms.google-services") // Firebase
6  }
7
8  android {
9      namespace = "com.example.proyectofinalandroid"
10     compileSdk = 34
11
12     defaultConfig {
13         applicationId = "com.example.proyectofinalandroid"
14         minSdk = 24
15         targetSdk = 34
16         versionCode = 1
17         versionName = "1.0"
18
19         testInstrumentationRunner = "androidx.test.runner.AndroidJUnitRunner"
20
21         // Si usas Firebase Auth
22         multiDexEnabled = true
23     }
24
25     buildTypes {
26
27         buildTypes {
28             release {
29                 isMinifyEnabled = false
30                 proguardFiles(
31                     getDefaultProguardFile("proguard-android-optimize.txt"),
32                     "proguard-rules.pro"
33                 )
34             }
35         }
36     }
37
38     buildFeatures {
39         viewBinding = true
40     }
41
42     compileOptions {
43         sourceCompatibility = JavaVersion.VERSION_1_8
44         targetCompatibility = JavaVersion.VERSION_1_8
45     }
46     kotlinOptions {
47         jvmTarget = "1.8"
48     }
49 }
50
51 dependencies {
```

```

dependencies {

    implementation(libs.androidx.core.ktx)
    implementation(libs.androidx.appcompat)
    implementation(libs.material)
    implementation(libs.androidx.activity)
    implementation(libs.androidx.constraintlayout)
    testImplementation(libs.junit)
    androidTestImplementation(libs.androidx.junit)
    androidTestImplementation(libs.androidx.espresso.core)

    // Retrofit + Gson
    implementation("com.squareup.retrofit2:retrofit:2.9.0")
    implementation("com.squareup.retrofit2:converter-gson:2.9.0")

    // Firebase Auth
    implementation("com.google.firebaseio:firebase-auth-ktx")

    // Optional: Logging interceptor para debug de peticiones
    implementation("com.squareup.okhttp3:logging-interceptor:5.0.0-alpha.11")

    // Kotlin coroutines para trabajo en background
    implementation("org.jetbrains.kotlinx:kotlinx-coroutines-android:1.7.3")

}

```

## FASE 2: Modificaciones en Spring Boot para que los usuarios registrados en la base de datos puedan iniciar sesión desde la app Android.

- Añadimos a la interfaz de usuarios(repository) un método de búsqueda por email:

```

UsuarioRepository.java x ServicioService.java SecurityConfig.java <> login.html

1 package com.tienda.proyectofinal.repository;
2
3 import com.tienda.proyectofinal.model.Usuario;
4 import org.springframework.data.jpa.repository.JpaRepository;
5
6 import java.util.Optional;
7
8 public interface UsuarioRepository extends JpaRepository<Usuario, Long> {
9     Optional<Usuario> findByEmail(String email); no usages
10 }

```

- Modificamos el controlador de login personalizado AuthController.java:

```

public class AuthController {
    @Autowired
    private UsuarioRepository usuarioRepository;
    private final FirebaseService firebaseService; 2 usages

    public AuthController(FirebaseService firebaseService) {
        this.firebaseioService = firebaseService;
    }

    @PostMapping(@RequestMapping("/verify"))
    public String verifyToken(@RequestHeader("Authorization") String authorizationHeader) {
        String idToken = authorizationHeader.replace("target: " + Bearer, replacement: "");
        FirebaseToken decodedToken = firebaseService.verificarToken(idToken);
        return "Usuario verificado: " + decodedToken.getUid();
    }

    @PostMapping(@RequestMapping("/login"))
    public ResponseEntity<?> login(@RequestBody Usuario loginRequest) {
        Optional<Usuario> usuarioOpt = usuarioRepository.findByEmail(loginRequest.getEmail());

        if (usuarioOpt.isPresent()) {
            Usuario usuario = usuarioOpt.get();
            if (usuario.getPassword().equals(LoginRequest.getPassword())) {
                return ResponseEntity.ok(usuario);
            }
        }

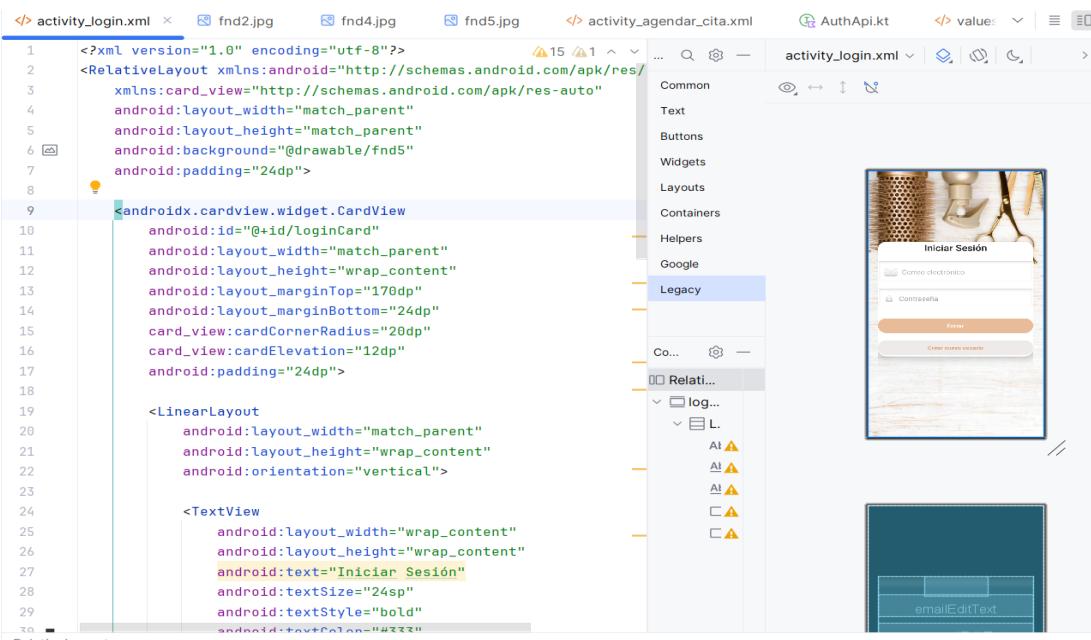
        return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body("Email o contraseña incorrectos");
    }
}

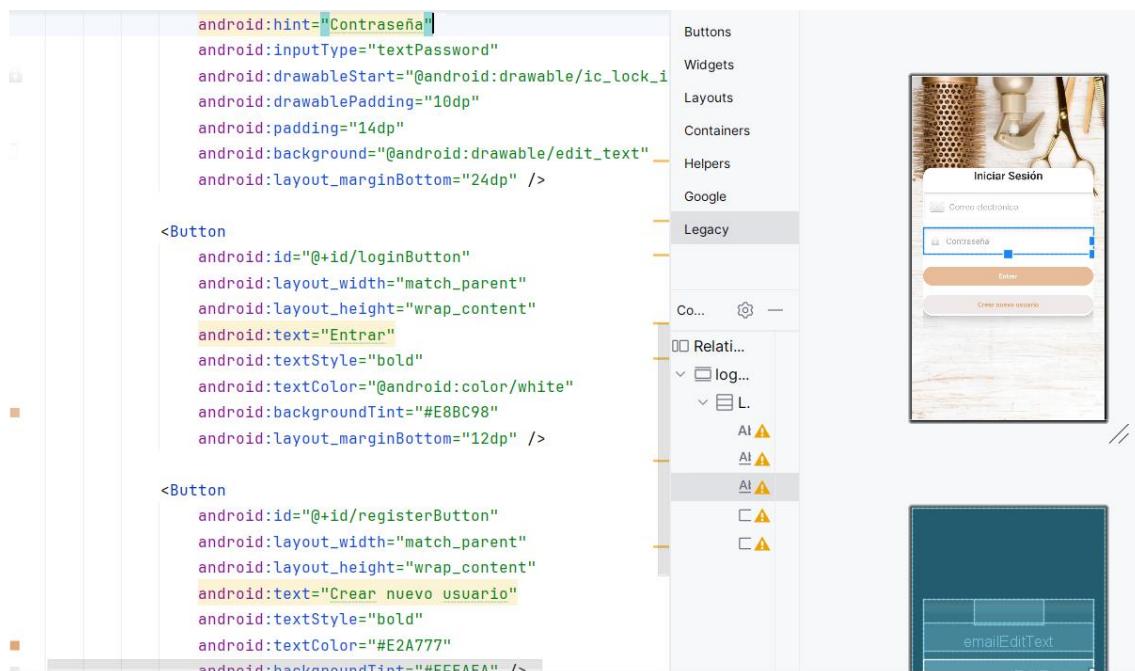
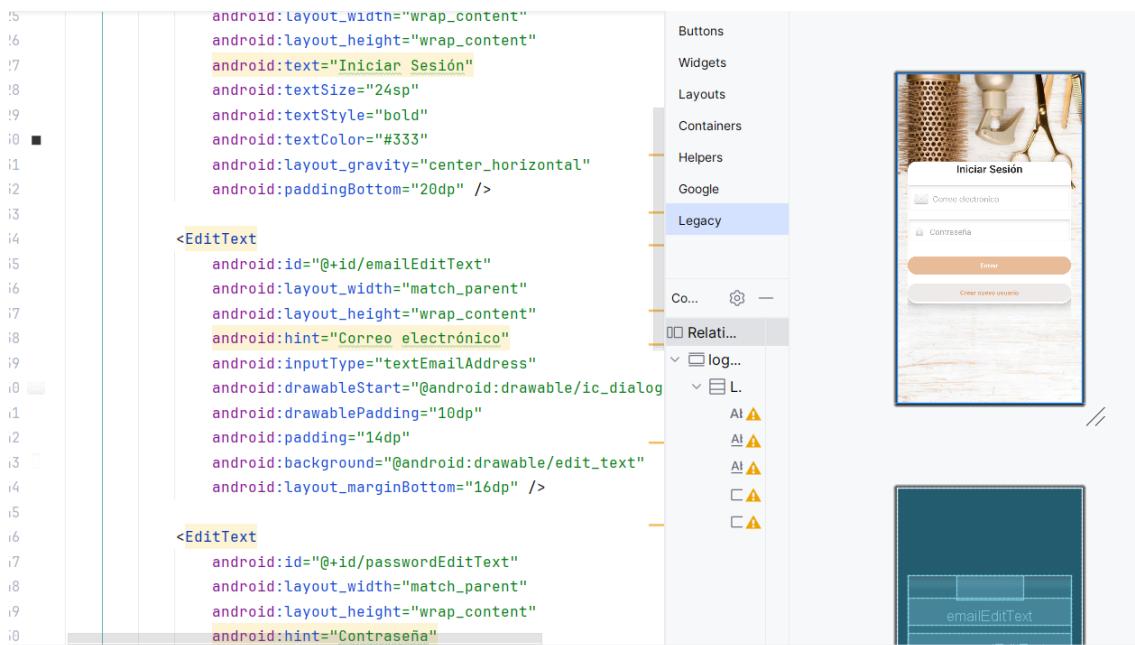
```

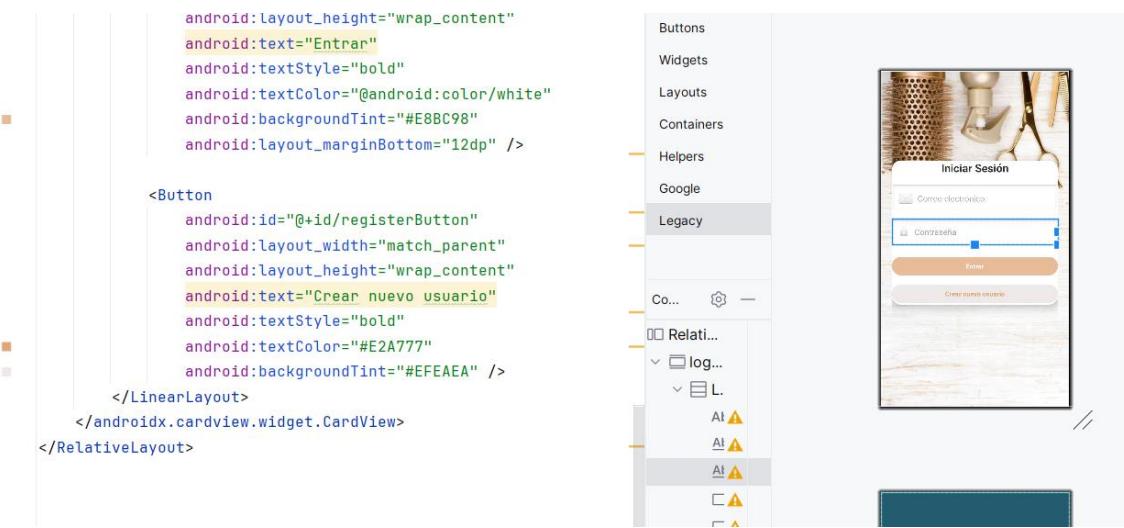
## FASE 3: Seguir con el proyecto de Android Studio

### ➤ Crear Login desde cero

- Primero creamos el layout en res/layout con el nombre de **activity\_login.xml**:







- Despues creamos la actividad **LoginActivity.kt**:

The screenshot shows the Android Studio code editor with the file `LoginActivity.kt` open. The code is written in Kotlin and sets up an `AppCompatActivity` to handle the login screen. It includes references to layout elements (`emailEditText`, `passwordEditText`, `loginButton`, `crearCuentaButton`) and an `AuthApi` interface. The code uses `OkHttpClient` with `HttpLoggingInterceptor` to handle network requests. The project structure on the left shows files like `AndroidManifest.xml`, `AuthApi.kt`, and various activity and adapter classes.

```

class LoginActivity : AppCompatActivity() {
    // Referencias a las vistas del layout
    private lateinit var emailEditText: EditText
    private lateinit var passwordEditText: EditText
    private lateinit var loginButton: Button
    private lateinit var crearCuentaButton: Button
    private lateinit var authApi: AuthApi

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        val screenSplash = installSplashScreen()
        Thread.sleep(500)
        screenSplash.setKeepOnScreenCondition(false)
        setContentView(R.layout.activity_login) // Conecta el layout XML

        // Inicializa los elementos visuales
        emailEditText = findViewById(R.id.emailEditText)
        passwordEditText = findViewById(R.id.passwordEditText)
        loginButton = findViewById(R.id.loginButton)
        crearCuentaButton = findViewById(R.id.registerButton)

        val logging = HttpLoggingInterceptor().apply {
            level = HttpLoggingInterceptor.Level.BODY
        }

        val client = OkHttpClient.Builder()
            .addInterceptor(logging)
            .build()

        // Configura Retrofit para conectar con tu backend
        val retrofit = Retrofit.Builder()
            .baseUrl("http://10.0.2.2:8080/api") // IP del host local en emulador
            .client(client)
            .addConverterFactory(GsonConverterFactory.create())
            .build()

        // Crea la instancia de la API
        authApi = retrofit.create(AuthApi::class.java)

        // Botón Login
        loginButton.setOnClickListener {
            val email = emailEditText.text.toString()
            val password = passwordEditText.text.toString()

            if (email.isBlank() || password.isBlank()) {
                Toast.makeText(context, "Completa todos los campos", Toast.LENGTH_SHORT).show()
                return@setOnClickListener
            }

            val usuario = Usuario(id = 0, nombre = "", email, password)

            authApi.login(usuario).enqueue(object : Callback<Usuario> {
                override fun onResponse(call: Call<Usuario>, response: Response<Usuario>) {
                    if (response.isSuccessful)
                }
            })
        }
    }
}

```

The screenshot shows the Android Studio interface. On the left, the project structure is visible under 'kotlin+java' and 'com.example.proyectoandroid'. On the right, a code editor displays the LoginActivity.kt file. The code is a Kotlin implementation of the Login API. It includes methods for logging in a user and handling errors. A specific section of the code is highlighted:

```

    authApi.login(usuario).enqueue(object : Callback<Usuario> {
        override fun onResponse(call: Call<Usuario>, response: Response<Usuario>) {
            if (response.isSuccessful) {
                val usuarioLogueado = response.body()
                Toast.makeText(context: this@LoginActivity, text: "Bienvenido ${usuarioLogueado?.nombre}", Toast.LENGTH_SHORT).show()

                val intent = Intent(packageContext: this@LoginActivity, MainActivity::class.java)
                intent.putExtra(name: "usuario", usuarioLogueado)
                startActivity(intent)
                finish()
            } else {
                Toast.makeText(context: this@LoginActivity, text: "Credenciales inválidas", Toast.LENGTH_SHORT).show()
            }
        }

        override fun onFailure(call: Call<Usuario>, t: Throwable) {
            Toast.makeText(context: this@LoginActivity, text: "Error de conexión", Toast.LENGTH_SHORT).show()
        }
    })
}

// Botón Crear Cuenta
crearCuentaButton.setOnClickListener {
    val intent = Intent(packageContext: this, CrearUsuarioActivity::class.java)
    startActivity(intent)
}
}

```

Cuando la actividad se inicia, se ejecuta el método `onCreate`, que realiza varias tareas clave:

Se lanza una pantalla de bienvenida (SplashScreen) que dura 0.5 segundos. Esto da una sensación más fluida al iniciar la app.

Carga el layout XML asociado a esta actividad, que contiene los campos de correo, contraseña y los botones de login y registro (`setContentView(R.layout.activity_login)`).

Se vinculan los elementos del XML con variables de Kotlin para poder interactuar con ellos (`emailEditText = findViewById(R.id.emailEditText)`).

Se configura Retrofit para conectarse con el backend que corre en el ordenador local `val retrofit = Retrofit.Builder().baseUrl("http://10.0.2.2:8080/api/")` (la IP 10.0.2.2 se usa en emuladores Android para referirse al localhost del PC).

Se añade un `HttpLoggingInterceptor` para mostrar en consola las peticiones/resuestas HTTP (útil para depurar).

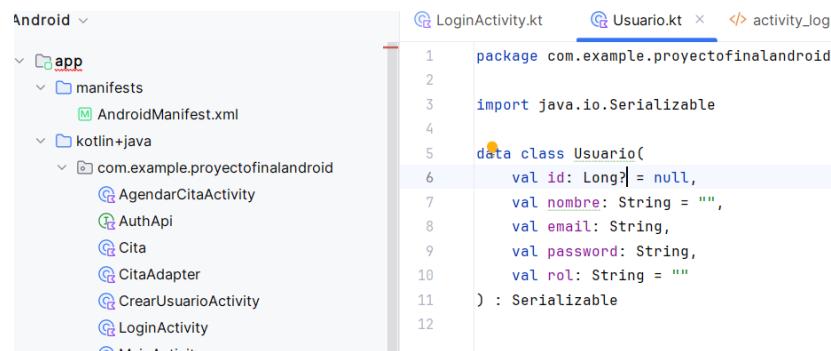
Se crea la instancia de AuthApi, la interfaz que define cómo comunicarse con el backend (por ejemplo, métodos como `login()` o `register()`).

Al hacer click en el botón Login (`loginButton.setOnClickListener`), se recogen los datos introducidos por el usuario, se valida que los campos no estén vacíos (`if (email.isBlank() || password.isBlank())`), se crea un objeto Usuario con los datos (`val usuario = Usuario(0, "", email, password)`) y se hace una llamada HTTP (`authApi.login(usuario).enqueue(...)`) para comprobar si el usuario existe y las credenciales son correctas. Si la respuesta es exitosa, se obtiene el usuario desde la respuesta (`val usuarioLogueado = response.body()`), se muestra un mensaje de bienvenida y se navega a la MainActivity, pasando el usuario como dato (`val intent = Intent(this@LoginActivity, MainActivity::class.java)` `intent.putExtra("usuario", usuarioLogueado)`). Si la respuesta falla o hay

error de red, se muestra un mensaje de error (credenciales incorrectas o problema de conexión).

Al hacer click en el botón crear cuenta (crearCuentaButton.setOnClickListener), redirige a la pantalla de registro (CrearUsuarioActivity) donde el usuario puede crear una cuenta nueva.

- Creamos la ‘data class’ que representa al usuario que se usará para enviar y recibir datos del backend **Usuario.kt** :

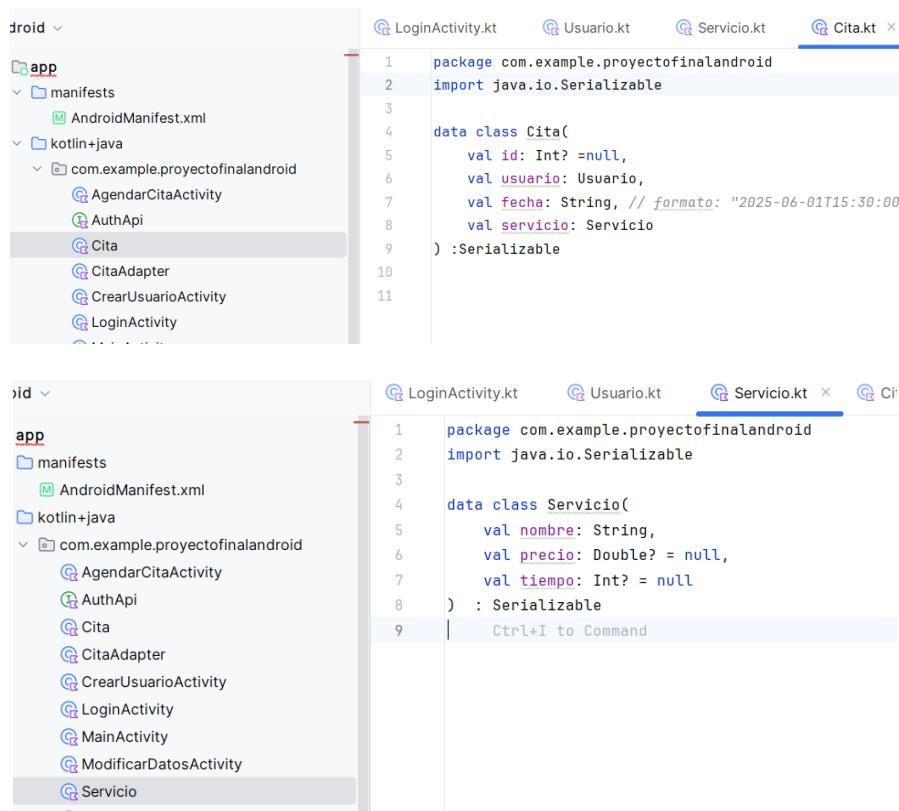


```

Android
app
  manifests
    AndroidManifest.xml
  kotlin+java
    com.example.proyectofinalandroid
      AgendarCitaActivity
      AuthApi
      Cita
      CitaAdapter
      CrearUsuarioActivity
      LoginActivity
      MainActivit
      Usuario.kt
        package com.example.proyectofinalandroid
        import java.io.Serializable
        data class Usuario(
          val id: Long? = null,
          val nombre: String = "",
          val email: String,
          val password: String,
          val rol: String = ""
        ) : Serializable
  
```

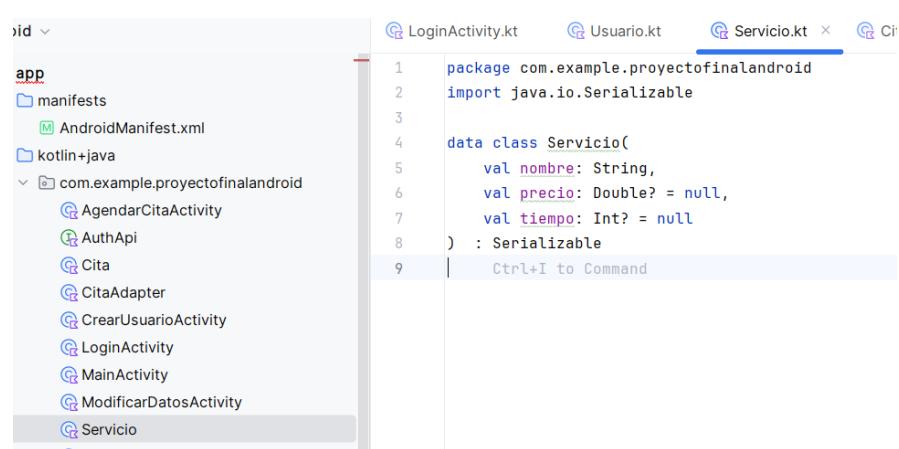
Se pone Serializable para que se pueda pasar un objeto completo (como Usuario) entre actividades en Android mediante un Intent.

- Creamos el resto de ‘data class’ **Servicio.kt** y **Cita.kt**:



```

droid
app
  manifests
    AndroidManifest.xml
  kotlin+java
    com.example.proyectofinalandroid
      AgendarCitaActivity
      AuthApi
      Cita
      CitaAdapter
      CrearUsuarioActivity
      LoginActivity
      MainActivit
      Servicio.kt
      Cita.kt
        package com.example.proyectofinalandroid
        import java.io.Serializable
        data class Cita(
          val id: Int? = null,
          val usuario: Usuario,
          val fecha: String, // formato: "2025-06-01T15:30:00"
          val servicio: Servicio
        ) : Serializable
  
```

```

id
app
  manifests
    AndroidManifest.xml
  kotlin+java
    com.example.proyectofinalandroid
      AgendarCitaActivity
      AuthApi
      Cita
      CitaAdapter
      CrearUsuarioActivity
      LoginActivity
      MainActivity
      ModificarDatosActivity
      Servicio.kt
        package com.example.proyectofinalandroid
        import java.io.Serializable
        data class Servicio(
          val nombre: String,
          val precio: Double? = null,
          val tiempo: Int? = null
        ) : Serializable
        Ctrl+I to Command
  
```

- También creamos la interfaz que define las peticiones HTTP que haremos con Retrofit **AuthApi.kt**:

```
// Interfaz que define las peticiones HTTP que haremos con Retrofit
interface AuthApi {
    // Llama al endpoint POST /api/auth/login enviando un objeto Usuario
    @POST("auth/login")
    fun login(@Body usuario: Usuario): Call<Usuario>

    // Llama al endpoint POST /api/usuarios para registrar un nuevo usuario
    @POST("usuarios")
    fun crearUsuario(@Body nuevoUsuario: Usuario): Call<Usuario>

    // Llama al endpoint PUT /api/usuarios/{id} enviando un objeto Usuario
    @PUT("usuarios/{id}")
    fun actualizarUsuario(@Path("id") id: Long?, @Body usuario: Usuario): Call<Usuario>

    // Llama al endpoint GET /api/servicios
    @GET("servicios")
    fun obtenerServicios(): Call<List<Servicio>>

    // Llama al endpoint POST /api/citas
    @POST("citas")
    fun agendarCita(@Body cita: Cita): Call<Cita>

    // Llama al endpoint GET /api/usuarios/{id}
    @GET("usuarios/{id}")
    fun obtenerUsuarioPorId(@Path("id") id: Long?): Call<Usuario>

    @GET("citas/usuario/{id}")
    fun obtenerCitasPorUsuario(@Path("id") idUsuario: Long?): Call<List<Cita>>

    @DELETE("citas/{id}")
    fun eliminarCita(@Path("id") id: Int?): Call<Void>

    @PUT("citas/{id}")
    fun editarCita(@Path("id") id: Int, @Body cita: Cita): Call<Cita>
}
```

Esta interfaz define todas las peticiones HTTP que la app Android realiza hacia el backend utilizando Retrofit. Cada método representa una llamada a un endpoint REST del backend desarrollado con Spring Boot.

- Modificamos `AndroidManifest.xml` para que inicie con la pantalla de `LoginActivity` y no con el `MainActivity` y añadimos los permisos para Internet:

```
2      <manifest xmlns:android="http://schemas.android.com/apk/res/android"
5          <!-- Permiso para usar Internet -->
6          <uses-permission android:name="android.permission.INTERNET"/>
7
8      <application
9          android:allowBackup="true"
10         android:dataExtractionRules="@xml/data_extraction_rules"
11         android:fullBackupContent="@xml/backup_rules"
12         android:icon="@mipmap/ic_launcher"
13         android:label="ProyectoFinalAndroid"
14         android:roundIcon="@mipmap/ic_launcher_round"
15         android:supportsRtl="true"
16         android:theme="@style/Theme.ProyectoFinalAndroid"
17         tools:targetApi="31">
18         <activity>
19             android:name=".LoginActivity"
20             android:exported="true" />
21             <intent-filter>
22                 <action android:name="android.intent.action.MAIN" />
23
24                 <category android:name="android.intent.category.LAUNCHER" />
25             </intent-filter>
26             <activity
27                 android:name=".MainActivity"
28                 android:exported="true">
29
30             </activity>
31         </application>
```

## ➤ Crear el menú de opciones (segunda pantalla)

- Modificamos **MainActivity.kt** para que al iniciar sesión el usuario, vea las diferentes opciones que tiene (modificar sus datos, agendar una cita, ver citas y cerrar sesión):

```

app
  manifests
    AndroidManifest.xml
  kotlin+java
    com.example.proyectofinalandroid
      AgendaCitaActivity
      AuthApi
      Cita
      CitaAdapter
      CrearUsuarioActivity
      LoginActivity
      MainActivity
      ModificarDatosActivity
      Servicio
      Usuario
      VerCitasActivity
    com.example.proyectofinalandroid (android)
      ExampleInstrumentedTest
    com.example.proyectofinalandroid (test)
    java (generated)
    res
      drawable
      layout
        activity_agendar_cita.xml
        activity_crear_usuario.xml
        activity_login.xml
        activity_main.xml
        activity_modificar_datos.xml
    CreateUsuarioActivity
    LoginActivity
    MainActivity
    ModificarDatosActivity
    Servicio
    Usuario
    VerCitasActivity
  com.example.proyectofinalandroid (android)
  ExampleInstrumentedTest
  com.example.proyectofinalandroid (test)
  java (generated)
  res
  drawable
  layout

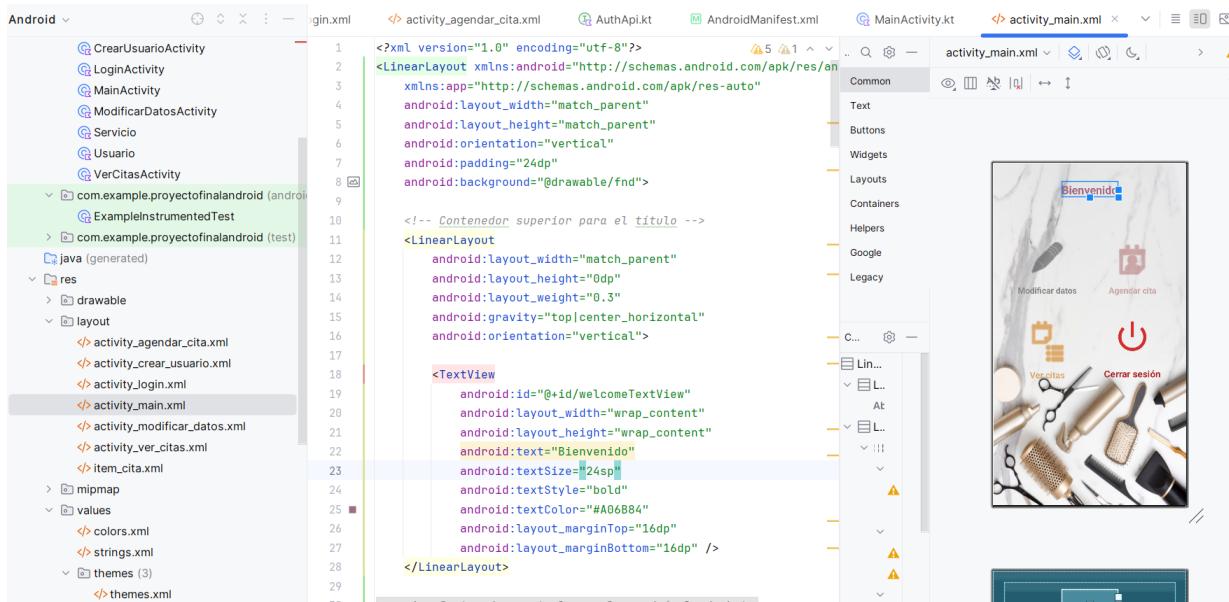
```

```

12
13 >< class MainActivity : AppCompatActivity() {
14
15
16
17     override fun onCreate(savedInstanceState: Bundle?) {
18         super.onCreate(savedInstanceState)
19         setContentView(R.layout.activity_main)
20
21         usuario = intent.getSerializableExtra( name: "usuario" ) as Usuario
22         findViewById<TextView>(R.id.welcomeTextView).text = "Bienvenido, ${usuario.nombre}"
23
24         findViewById<LinearLayout>(R.id.modificarDatosBtn).setOnClickListener {
25             //startActivity(Intent(this, ModificarDatosActivity::class.java))
26             val intent = Intent( packageContext: this, ModificarDatosActivity::class.java )
27             intent.putExtra( name: "usuario", usuario )
28             startActivity(intent)
29         }
30
31         findViewById<LinearLayout>(R.id.agendarCitaBtn).setOnClickListener {
32             //startActivity(Intent(this, AgendaCitaActivity::class.java))
33             val intent = Intent( packageContext: this, AgendaCitaActivity::class.java )
34             intent.putExtra( name: "usuario", usuario )
35             startActivity(intent)
36         }
37
38         findViewById<LinearLayout>(R.id.verCitasBtn).setOnClickListener {
39             val intent = Intent( packageContext: this, VerCitasActivity::class.java )
40             intent.putExtra( name: "usuario", usuario )
41             startActivity(intent)
42         }
43
44         findViewById<LinearLayout>(R.id.verCitasBtn).setOnClickListener {
45             val intent = Intent( packageContext: this, VerCitasActivity::class.java )
46             intent.putExtra( name: "usuario", usuario )
47             startActivity(intent)
48         }
49
50     }
51

```

- Y también modificamos su layout **activity\_main.xml**:



Project Structure:

```

com.example.proyectofinalandroid (android)
    |__ Servicio
    |__ Usuario
    |__ VerCitasActivity
    |__ com.example.proyectofinalandroid (android)
        |__ ExampleInstrumentedTest
    |__ com.example.proyectofinalandroid (test)
    |__ java (generated)
    |__ res
        |__ drawable
            |__ activity_agendar_cita.xml
            |__ activity_crear_usuario.xml
            |__ activity_login.xml
            |__ activity_main.xml
            |__ activity_modificar_datos.xml
            |__ activity_ver_citas.xml
            |__ item_cita.xml
        |__ mipmap
        |__ values
            |__ colors.xml
            |__ strings.xml
            |__ themes (3)
                |__ themes.xml
    |__ MoudularDatosActivity
        |__ Servicio
        |__ Usuario
        |__ VerCitasActivity
    |__ com.example.proyectofinalandroid (android)
        |__ ExampleInstrumentedTest
    |__ com.example.proyectofinalandroid (test)
    |__ java (generated)
    |__ drawable
    |__ layout
        |__ activity_agendar_cita.xml
        |__ activity_crear_usuario.xml
        |__ activity_login.xml
        |__ activity_main.xml
        |__ activity_modificar_datos.xml
        |__ activity_ver_citas.xml
        |__ item_cita.xml
    |__ mipmap
    |__ values
        |__ colors.xml
        |__ strings.xml
    |__ themes (3)
        |__ themes.xml
    |__ com.example.proyectofinalandroid (android)
        |__ ExampleInstrumentedTest
    |__ com.example.proyectofinalandroid (test)
    |__ java (generated)
    |__ res
        |__ drawable
        |__ layout
            |__ activity_agendar_cita.xml
            |__ activity_crear_usuario.xml
            |__ activity_login.xml
            |__ activity_main.xml
            |__ activity_modificar_datos.xml
            |__ activity_ver_citas.xml
            |__ item_cita.xml
        |__ mipmap
        |__ values
            |__ colors.xml
            |__ strings.xml
            |__ themes (3)
                |__ themes.xml (v31)
                |__ themes.xml (night)
        |__ xml
            |__ backup_rules.xml

```

XML Layout (activity\_main.xml):

```

<!-- Contenedor central para la cuadrícula de botones -->
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_weight="1.5"
    android:gravity="top"
    android:orientation="vertical">

    <GridLayout
        android:id="@+id/menuGrid"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:columnCount="2"
        android:rowCount="2"
        android:alignmentMode="alignMargins"
        android:columnOrderPreserved="false"
        android:useDefaultMargins="true">

        <!-- Botón 1 -->
        <LinearLayout
            android:id="@+id/modificarDatosBtn"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_columnWeight="1"
            android:orientation="vertical"
            android:gravity="center"
            android:padding="16dp"
            android:background="@android:color/transparent"
            android:elevation="4dp"
            android:layout_margin="8dp"
            android:clickable="true"
            android:focusable="true">

            <ImageView
                android:layout_width="80dp"
                android:layout_height="80dp"
                android:src="@android:drawable/ic_menu_edit"
                app:tint="#4D4B4C" />

            <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_marginTop="18dp"
                android:text="Modificar datos"
                android:textColor="#898889"
                android:textSize="18sp"
                android:textStyle="bold" />
        


Screenshot of the Android Studio interface showing the layout editor and code editor side-by-side.


```

Screenshot of the Android Studio interface showing the code editor and the design view for four screens of a mobile application. The code editor displays XML layout files for each screen, and the design view shows the visual representation of the UI with various icons and buttons.

```

<Image>
    android:layout_width="90dp"
    android:layout_height="80dp"
    android:src="@android:drawable/ic_menu_my_calendar"
    app:tint="#B87A7A" />

<TextView>
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Agendar cita"
    android:textColor="#CCAAAB"
    android:textSize="18sp"
    android:textStyle="bold"
    android:layout_marginTop="8dp" />

</LinearLayout>

<!-- Botón 3 -->
<LinearLayout>
    android:id="@+id/verCitasBtn"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:orientation="vertical"
    android:gravity="center"

    android:padding="16dp"
    android:background="@android:color/transparent"
    android:elevation="4dp"
    android:layout_margin="8dp"
    android:clickable="true"
    android:focusable="true">

    <ImageView>
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:src="@android:drawable/ic_menu_agenda"
        app:tint="#D17C17" />

    <TextView>
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Ver citas"
        android:textColor="#DCAAE" />

    <TextView>
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Ver citas"
        android:textColor="#DCAAE" />

</LinearLayout>

<!-- Botón 4 -->
<LinearLayout>
    android:id="@+id/cerrarSesionBtn"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:orientation="vertical"
    android:gravity="center"

    android:padding="16dp"
    android:background="@android:color/transparent"
    android:elevation="4dp"
    android:layout_margin="8dp"
    android:clickable="true"
    android:focusable="true">

    <ImageView>
        android:layout_width="80dp"
        android:layout_height="80dp"
        android:src="@android:drawable/ic_lock_power_off"
        app:tint="#D3F2F" />

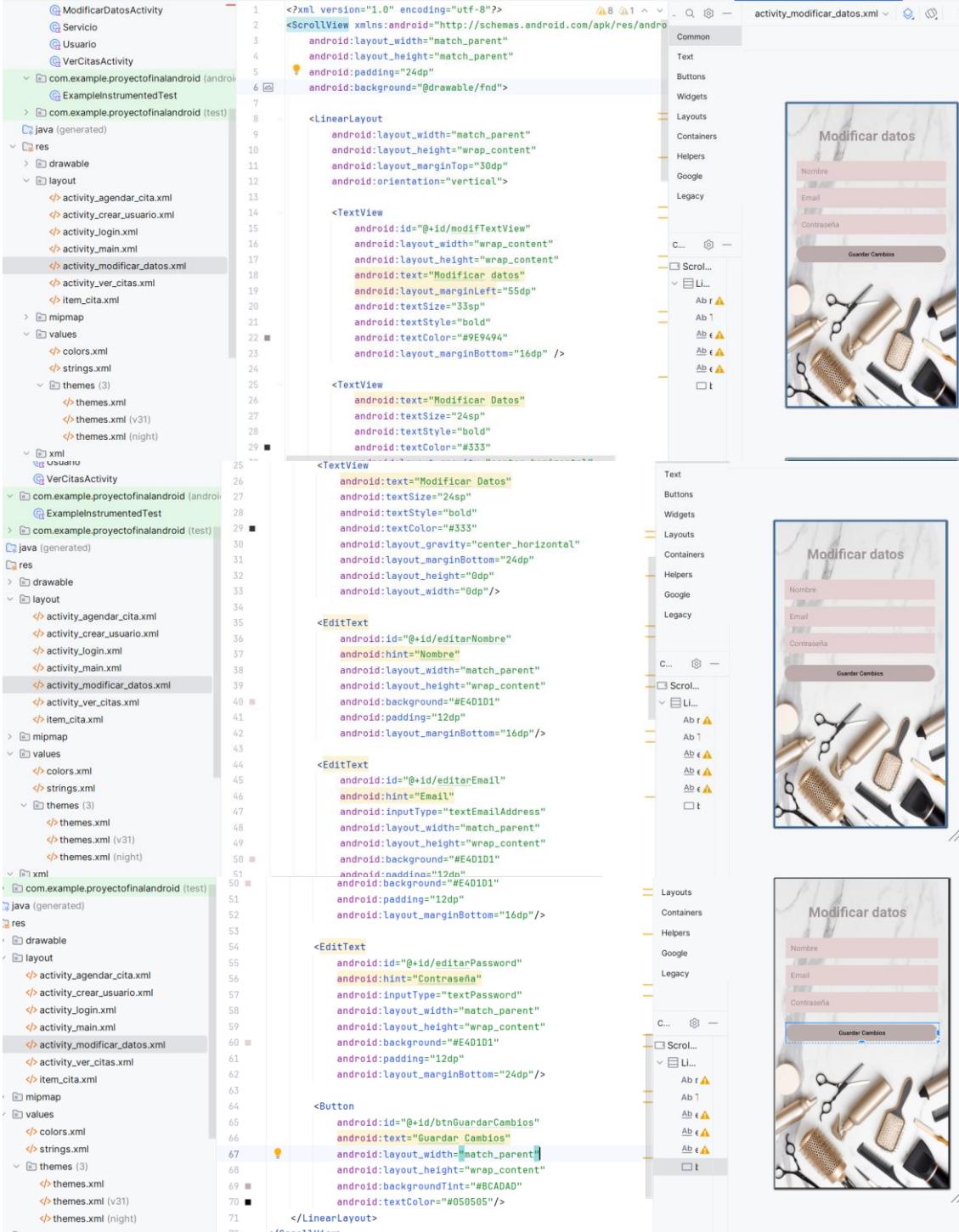
    <TextView>
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Cerrar sesión"
        android:textColor="#AB2A2A"
        android:textStyle="bold"
        android:textSize="20sp"
        android:layout_marginTop="25dp" />

    </LinearLayout>
</GridLayout>
</LinearLayout>
</LinearLayout>

```

## ➤ Crear las diferentes opciones

- Añadimos el layout de la opción de modificar datos del usuario **activity\_modificar\_datos.xml**:



```

<?xml version="1.0" encoding="utf-8"?>
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="24dp"
    android:background="@drawable/fondo">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="30dp"
        android:orientation="vertical">

        <TextView
            android:id="@+id/modifTextView"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Modificar datos"
            android:layout_marginLeft="55dp"
            android:textSize="33sp"
            android:textStyle="bold"
            android:textColor="#9E9494"
            android:layout_marginBottom="16dp" />

        <TextView
            android:text="Modificar Datos"
            android:textSize="24sp"
            android:textStyle="bold"
            android:textColor="#333333" />

        <EditText
            android:id="@+id/editarNombre"
            android:hint="Nombre"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:background="#E4D1D1"
            android:padding="12dp"
            android:layout_marginBottom="16dp" />

        <EditText
            android:id="@+id/editarEmail"
            android:hint="Email"
            android:inputType="textEmailAddress"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:background="#E4D1D1"
            android:padding="12dp"
            android:layout_marginBottom="16dp" />

        <EditText
            android:id="@+id/editarPassword"
            android:hint="Contraseña"
            android:inputType="textPassword"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:background="#E4D1D1"
            android:padding="12dp"
            android:layout_marginBottom="24dp" />

        <Button
            android:id="@+id/btnGuardarCambios"
            android:text="Guardar Cambios"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:background="#BCADAD"
            android:textColor="#050505" />

    </LinearLayout>
</ScrollView>

```

- Añadimos también su correspondiente activity **ModificarDatosActivity.kt** donde el usuario puede modificar sus datos (todos menos su ‘id’ y su ‘rol’):

The screenshot shows the Android Studio interface with the project structure on the left and the code editor on the right.

**Project Structure:**

- app
  - manifests
  - AndroidManifest.xml
  - kotlin+java
    - com.example.proyectofinalandroid
    - AgendarCitaActivity
    - AuthApi
    - Cita
    - CitaAdapter
    - CrearUsuarioActivity
    - LoginActivity
    - MainActivity
    - ModificarDatosActivity
    - Servicio
    - Usuario
    - VerCitasActivity
  - ExampleInstrumentedTest
  - com.example.proyectofinalandroid (test)
  - java (generated)
  - res
    - drawable
    - layout
      - activity\_agendar\_cita.xml
      - activity\_crear\_usuario.xml
      - activity\_login.xml
      - activity\_main.xml
      - activity\_modificar\_datos.xml

**Code Editor (ModificarDatosActivity.kt):**

```

15 >< class ModificarDatosActivity : AppCompatActivity() {
16     private lateinit var editarNombre: EditText
17     private lateinit var editarEmail: EditText
18     private lateinit var editarPassword: EditText
19     private lateinit var btnGuardarCambios: Button
20     private lateinit var authApi: AuthApi
21     private lateinit var usuario: Usuario
22
23     override fun onCreate(savedInstanceState: Bundle?) {
24         super.onCreate(savedInstanceState)
25         setContentView(R.layout.activity_modificar_datos)
26
27         editarNombre = findViewById(R.id.editarNombre)
28         editarEmail = findViewById(R.id.editarEmail)
29         editarPassword = findViewById(R.id.editarPassword)
30         btnGuardarCambios = findViewById(R.id.btnGuardarCambios)
31
32         val retrofit = Retrofit.Builder()
33             .baseUrl("http://10.0.2.2:8080/api/")
34             .addConverterFactory(GsonConverterFactory.create())
35             .build()
36
37         authApi = retrofit.create(AuthApi::class.java)
38
39         // Recuperar usuario del intent
40         val usuarioIntent = intent.getSerializableExtra("usuario") as Usuario
41         usuario = usuarioIntent
42
43         // Llamada al backend para obtener usuario actualizado
44         authApi.obtenerUsuarioPorId(usuario.id).enqueue(object : Callback<Usuario> {
45             override fun onResponse(call: Call<Usuario>, response: Response<Usuario>) {
46                 if (response.isSuccessful) {
47                     usuario = response.body()!!
48                     editarNombre.setText(usuario.nombre)
49                     editarEmail.setText(usuario.email)
50                     editarPassword.setText(usuario.password)
51                 } else {
52                     Toast.makeText(context, "Error al cargar datos", Toast.LENGTH_SHORT).show()
53                 }
54             }
55
56             override fun onFailure(call: Call<Usuario>, t: Throwable) {
57                 Toast.makeText(context, "Error de red al cargar datos", Toast.LENGTH_LONG).show()
58             }
59         })
60
61         btnGuardarCambios.setOnClickListener {
62             val nuevoNombre = editarNombre.text.toString()
63             val nuevoEmail = editarEmail.text.toString()
64             val nuevaPassword = editarPassword.text.toString()
65
66             val usuarioActualizado = Usuario(usuario.id, nuevoNombre, nuevoEmail, nuevaPassword)
67
68             authApi.actualizarUsuario(usuario.id, usuarioActualizado).enqueue(object : Callback<Usuario> {
69                 override fun onResponse(call: Call<Usuario>, response: Response<Usuario>) {
70                     if (response.isSuccessful) {
71                         authApi.actualizarUsuario(usuario.id, usuarioActualizado).enqueue(object : Callback<Usuario> {
72                             override fun onResponse(call: Call<Usuario>, response: Response<Usuario>) {
73                                 if (response.isSuccessful) {
74                                     val usuarioModificado = response.body()!!
75                                     Toast.makeText(context, "Datos actualizados correctamente", Toast.LENGTH_SHORT).show()
76                                     intent.putExtra("usuario", usuarioModificado)
77                                     startActivity(intent)
78                                     finish()
79                                 } else {
80                                     val responseBody = response.errorBody()?.string()
81                                     Log.e(tag = "MODIFICAR_DATOS", msg = "Error al actualizar: $responseBody")
82                                     Toast.makeText(context, "Error al actualizar", Toast.LENGTH_SHORT).show()
83                                 }
84                             }
85
86                             override fun onFailure(call: Call<Usuario>, t: Throwable) {
87                                 Log.e(tag = "MODIFICAR_DATOS", msg = "Error de red", t)
88                                 Toast.makeText(context, "Error de red: ${t.message}", Toast.LENGTH_LONG).show()
89                             }
90                         })
91                     }
92                 }
93             })
94         }
95     }
96 }

```

Al iniciarse, esta actividad carga la interfaz y establece conexión con el backend usando Retrofit. Recupera el objeto Usuario enviado desde otra actividad, luego realiza una petición GET al servidor para obtener los datos actualizados de ese usuario y mostrarlos en los campos de

texto. Cuando el usuario edita los datos y pulsa el botón "Guardar cambios", se envía una petición PUT al backend con la nueva información. Si la operación es exitosa, muestra un mensaje de confirmación, actualiza el intent con el nuevo usuario (para mostrar de nuevo la pantalla con los campos modificados, intent.putExtra("usuario", usuarioModificado) ) y finaliza la actividad.

- Creamos el layout de la opción agendar citas **activity\_agendar\_cita.xml**:

The screenshot shows the Android Studio interface with the XML code for `activity_agendar_cita.xml` on the left and three preview panes on the right. The XML code defines a ScrollView containing a Linear Layout with various TextViews, Spinners, Date Pickers, and Time Pickers, all styled with bold text and specific colors. The preview panes show the UI components arranged in a grid-like layout with a background image of hair styling tools.

```

<?xml version="1.0" encoding="utf-8"?>
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/fnd4"
    android:padding="24dp">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical">

        <TextView
            android:text="Selecciona un servicio"
            android:textSize="22sp"
            android:textStyle="bold"
            android:textColor="#03A4BC"
            android:layout_marginBottom="8dp"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />

        <Spinner
            android:id="@+id/spinnerServicios"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:backgroundTint="#078AB0"
            android:background="@android:drawable/btn_dropdown" />

        <TextView
            android:text="Elige la fecha"
            android:textSize="22sp"
            android:textStyle="bold"
            android:textColor="#03A4BC"
            android:layout_marginTop="24dp"
            android:layout_marginBottom="8dp"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />

        <DatePicker
            android:id="@+id/datePicker"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:theme="@style/MiEstiloDatePicker"
            android:calendarViewShown="true" />

        <TextView
            android:text="Elige la hora"
            android:textSize="22sp"
            android:textStyle="bold"
            android:textColor="#03A4BC" />

        <TimePicker
            android:id="@+id/timePicker"
            android:theme="@style/MiEstiloDatePicker"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />

        <Button
            android:id="@+id	btnAgendar"
            android:text="Agendar Cita"
            android:layout_marginTop="32dp"
            android:backgroundTint="#03A4BC"
            android:textColor="#FFF"
            android:layout_width="match_parent"
            android:layout_height="wrap_content" />

    </LinearLayout>
</ScrollView>

```

- Y también creamos su activity **AgendarCitaActivity.kt** donde el usuario puede seleccionar el servicio que desea en la fecha y hora que quiera:

The screenshot shows the Android Studio interface with the project structure on the left and the code editor on the right.

**Project Structure:**

- app** folder:
  - manifests**: Contains `AndroidManifest.xml`.
  - kotlin+java**: Contains the source code for the application.
  - com.example.proyectofinalandroid** package:
    - AgendarCitaActivity**: Selected in the tree view.
    - AuthApi**
    - Cita**
    - CitaAdapter**
    - CrearUsuarioActivity**
    - LoginActivity**
    - MainActivity**
    - ModificarDatosActivity**
    - Servicio**
    - Usuario**
    - VerCitasActivity**
  - com.example.proyectofinalandroid (android)** package:
    - ExampleInstrumentedTest**
  - com.example.proyectofinalandroid (test)** package:
    - java (generated)**
  - res** folder:
    - drawable**: Contains XML files for layouts: `activity_agendar_cita.xml`, `activity_crear_usuario.xml`, `activity_login.xml`, `activity_main.xml`, and `activity_modificar_datos.xml`.
    - layout**: Contains XML files for layouts: `activity_agendar_cita.xml`, `activity_crear_usuario.xml`, `activity_login.xml`, `activity_main.xml`, and `activity_modificar_datos.xml`.

**Code Editor (AgendarCitaActivity.kt):**

```

22 <> class AgendarCitaActivity : AppCompatActivity() {
23     private lateinit var spinnerServicios: Spinner
24     private lateinit var datePicker: DatePicker
25     private lateinit var timePicker: TimePicker
26     private lateinit var btnAgendar: Button
27
28     private lateinit var authApi: AuthApi
29     private var servicios: List<Servicio> = listOf()
30     private lateinit var usuario: Usuario
31     private var citaExistente: Cita? = null
32
33     @RequiresApi(Build.VERSION_CODES.O)
34     override fun onCreate(savedInstanceState: Bundle?) {
35         super.onCreate(savedInstanceState)
36         setContentView(R.layout.activity_agendar_cita)
37
38         spinnerServicios = findViewById(R.id.spinnerServicios)
39         datePicker = findViewById(R.id.datePicker)
40         timePicker = findViewById(R.id.timePicker)
41         btnAgendar = findViewById(R.id.btnAgendar)
42
43         usuario = intent.getSerializableExtra("usuario") as Usuario
44         citaExistente = intent.getSerializableExtra("cita") as? Cita
45
46         val retrofit = Retrofit.Builder()
47             .baseUrl("http://10.0.2.2:8080/api/")
48             .addConverterFactory(GsonConverterFactory.create())
49             .build()
50
51         authApi = retrofit.create(AuthApi::class.java)
52
53         cargarServicios()
54
55         btnAgendar.setOnClickListener {
56             if (citaExistente != null) {
57                 editarCita()
58             } else {
59                 agendarCita()
60             }
61         }
62     }
63
64     private fun cargarServicios() {
65         authApi.obtenerServicios().enqueue(object : Callback<List<Servicio>> {
66             @RequiresApi(Build.VERSION_CODES.O)
67             override fun onResponse(call: Call<List<Servicio>>, response: Response<List<Servicio>>) {
68                 if (response.isSuccessful) {
69                     servicios = response.body() ?: listOf()
70                     val listaservicios = servicios.map { "${it.nombre} - ${it.precio}€ - ${it.tiempo} min" }
71                     val adapter = ArrayAdapter(context: this@AgendarCitaActivity, android.R.layout.simple_spinner_dropdown_item, listaservicios)
72                     adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item)
73                     spinnerServicios.adapter = adapter
74                 }
75             }
76         })
77     }
78
79     // Cada selecciona adiciona una otra - noparamos los datos

```

```

    spinnerServicios.adapter = adapter

    // Si estamos editando una cita, precargamos los datos
    citaExistente?.let { cita ->
        val servicioIndex = servicios.indexOfFirst { it.nombre == cita.servicio.nombre }
        if (servicioIndex >= 0) spinnerServicios.setSelection(servicioIndex)

        val fechaHora = LocalDateTime.parse(cita.fecha)
        datePicker.updateDate(fechaHora.year, month: fechaHora.monthValue - 1, fechaHora.dayOfMonth)
        timePicker.hour = fechaHora.hour
        timePicker.minute = fechaHora.minute
    }

    btnAgendar.text = "Editar Cita"
}

override fun onFailure(call: Call<List<Servicio>>, t: Throwable) {
    Toast.makeText(context: this@AgendarCitaActivity, text: "Error al cargar servicios", Toast.LENGTH_SHORT).show()
}
}

@RequiresApi(Build.VERSION_CODES.O)
private fun agendarCita() {
    val fechaHora = obtenerFechaHoraSeleccionada()
    val nuevaCita = Cita(id: null, usuario, fechaHora, servicioSeleccionado)
    authApi.agendarCita(nuevaCita).enqueue(object : Callback<Cita> {
        override fun onResponse(call: Call<Cita>, response: Response<Cita>) {
            if (response.isSuccessful) {
                Toast.makeText(context: this@AgendarCitaActivity, text: "Cita agendada", Toast.LENGTH_SHORT).show()
                finish()
            } else {
                Toast.makeText(context: this@AgendarCitaActivity, text: "Error al agendar cita", Toast.LENGTH_SHORT).show()
            }
        }

        override fun onFailure(call: Call<Cita>, t: Throwable) {
            Log.e(tag: "AGENDAR_CITA", msg: "Error de red", t)
            Toast.makeText(context: this@AgendarCitaActivity, text: "Error de red: ${t.message}", Toast.LENGTH_LONG).show()
        }
    })
}

@RequiresApi(Build.VERSION_CODES.O)
private fun editarCita() {
    val fechaHora = obtenerFechaHoraSeleccionada()
    private fun editarCita() {
        val fechaHora = obtenerFechaHoraSeleccionada()
        val servicioSeleccionado = servicios[spinnerServicios.selectedItemPosition]
        val citaActualizada = Cita(citaExistente!!.id, usuario, fechaHora, servicioSeleccionado)

        authApi.editarCita(citaActualizada.id!!, citaActualizada).enqueue(object : Callback<Cita> {
            override fun onResponse(call: Call<Cita>, response: Response<Cita>) {
                if (response.isSuccessful) {
                    Toast.makeText(context: this@AgendarCitaActivity, text: "Cita editada", Toast.LENGTH_SHORT).show()
                    val intent = Intent(packageContext: this@AgendarCitaActivity, MainActivity::class.java)
                    intent.putExtra(name: "usuario", usuario)
                    startActivity(intent)
                    finish()
                } else {
                    Toast.makeText(context: this@AgendarCitaActivity, text: "Error al editar cita", Toast.LENGTH_SHORT).show()
                }
            }

            override fun onFailure(call: Call<Cita>, t: Throwable) {
                Toast.makeText(context: this@AgendarCitaActivity, text: "Error de conexión", Toast.LENGTH_SHORT).show()
            }
        })
    }

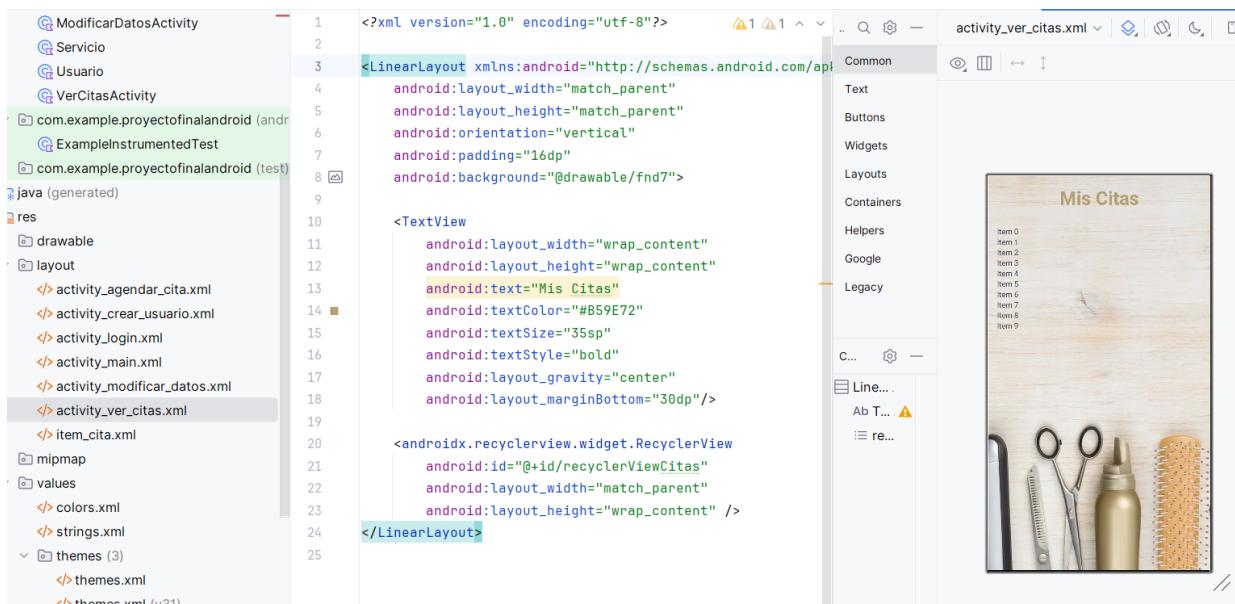
    @RequiresApi(Build.VERSION_CODES.O)
    private fun obtenerFechaHoraSeleccionada(): String {
        val day = datePicker.dayOfMonth
        val month = datePicker.month + 1
        val year = datePicker.year
    }

    @RequiresApi(Build.VERSION_CODES.O)
    private fun obtenerFechaHoraSeleccionada(): String {
        val day = datePicker.dayOfMonth
        val month = datePicker.month + 1
        val year = datePicker.year
        val hour = timePicker.hour
        val minute = timePicker.minute
        val fechaHora = LocalDateTime.of(year, month, day, hour, minute)
        return fechaHora.toString()
    }
}

```

La clase AgendarCitaActivity permite a un usuario crear o editar una cita en la app Android. Al iniciarse, establece la interfaz y conecta con el backend mediante Retrofit. Si recibe una cita desde el intent, entra en modo edición y precarga los datos en los campos correspondientes. Utiliza DatePicker, TimePicker y un Spinner para que el usuario seleccione fecha, hora y servicio. Al pulsar el botón, se agenda una nueva cita o se edita una existente según corresponda, enviando los datos al servidor mediante Retrofit. Si la operación es exitosa, muestra un mensaje y finaliza; en caso de error, informa al usuario. También carga dinámicamente la lista de servicios desde el backend y la muestra en el Spinner.

- Despues creamos el layout de ver citas **activity\_ver\_citas.xml** el cual tiene un RecyclerView que muestra una a una las citas



- Como consecuencia, tenemos que crear cada item de cita que mostrará el RecyclerView **item\_cita-xml**. Cada cita se podrá eliminar y/o editar

Screenshot of the Android Studio IDE showing the project structure and the XML code for the item\_cita.xml layout.

**Project Structure:**

- src
  - com.example.proyectofinalandroid
    - java (generated)
    - res
      - drawable
      - layout
        - activity\_agendar\_cita.xml
        - activity\_crear\_usuario.xml
        - activity\_login.xml
        - activity\_main.xml
        - activity\_modificar\_datos.xml
        - activity\_ver\_citas.xml
        - item\_cita.xml
      - mipmap
      - values
        - colors.xml
        - strings.xml
      - themes (3)
        - themes.xml
        - themes.xml (v31)
        - themes.xml (night)
    - xml
      - VerCitasActivity
  - com.example.proyectofinalandroid (androidTest)
    - ExampleInstrumentedTest
  - com.example.proyectofinalandroid (test)
    - java (generated)

**XML Code (item\_cita.xml):**

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:padding="12dp"
    android:background="#D0DFD0BF"
    android:layout_margin="8dp">

    <ImageView
        android:id="@+id/iconoServicio"
        android:layout_width="55dp"
        android:layout_height="55dp"
        android:src="@drawable/icono_pelu"
        android:contentDescription="Icono"
        android:layout_gravity="center_vertical"
        android:layout_marginEnd="20dp" />

    <LinearLayout
        android:layout_width="0dp"
        android:layout_weight="1"
        android:layout_height="wrap_content"
        android:orientation="vertical">

        <TextView
            android:id="@+id/fechaTextView"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Fecha"
            android:layout_weight="1"
            android:layout_height="wrap_content"
            android:text="Fecha"
            android:textStyle="bold"
            android:textSize="19sp"
            android:textColor="#333333" />

        <TextView
            android:id="@+id/servicioTextView"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textSize="17sp"
            android:text="Servicio"
            android:textColor="#555555" />

        <TextView
            android:id="@+id/precioTextView"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Precio"
            android:textSize="17sp"
            android:textColor="#555555" />

        <TextView
            android:id="@+id/tiempoTextView"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Tiempo"
            android:textSize="17sp"
            android:textColor="#555555" />
    


Design View (item_cita.xml):



The design view shows a horizontal layout with a central vertical linear layout containing four text views for Date, Service, Price, and Time, each with a bold title and standard text below. At the bottom of the central layout are two image buttons for Edit and Delete operations. A top navigation bar with icons for Fecha, Servicio, Precio, and Tiempo is also visible.


```

- Ahora creamos el activity de ver citas **VerCitasActivity.kt**

The screenshot shows the Android Studio interface with the project structure on the left and the code editor on the right.

**Project Structure:**

- app**
  - manifests**: Contains `AndroidManifest.xml`.
  - kotlin+java**
    - com.example.proyectofinalandroid**
      - `AgendarCitaActivity`
      - `AuthApi`
      - `Cita`
      - `CitaAdapter`
      - `CrearUsuarioActivity`
      - `LoginActivity`
      - `MainActivity`
      - `ModificarDatosActivity`
      - `Servicio`
      - `Usuario`
      - `VerCitasActivity` (selected)
    - `ExampleInstrumentedTest`
    - `com.example.proyectofinalandroid (test)`
    - java (generated)**
    - res**
      - drawable**
      - layout**
        - `activity_agendar_cita.xml`
        - `activity_crear_usuario.xml`
        - `activity_login.xml`
        - `activity_main.xml`
        - `activity_modificar_datos.xml`
  - AndroidManifest.xml**
  - kotlin+java**
    - com.example.proyectofinalandroid**
      - `AgendarCitaActivity`
      - `AuthApi`
      - `Cita`
      - `CitaAdapter`
      - `CrearUsuarioActivity`
      - `LoginActivity`
      - `MainActivity`
      - `ModificarDatosActivity`
      - `Servicio`
      - `Usuario`
      - `VerCitasActivity` (selected)
    - `ExampleInstrumentedTest`
    - `com.example.proyectofinalandroid (test)`
    - java (generated)**
    - res**
      - drawable**
      - layout**
        - `activity_agendar_cita.xml`
        - `activity_crear_usuario.xml`
        - `activity_login.xml`
        - `activity_main.xml`
        - `activity_modificar_datos.xml`

La clase `VerCitasActivity` muestra al usuario una lista de sus citas en un `RecyclerView`. Al iniciar, recupera el usuario desde el intent

y configura el adaptador CitaAdapter, que permite editar o eliminar citas. Se conecta al backend con Retrofit y carga las citas asociadas al usuario usando su ID. Si el usuario desea editar una cita, se lanza AgendarCitaActivity pasando la cita y el usuario. Si desea eliminarla, se envía una petición al backend; si tiene éxito, la lista se actualiza. La actividad informa mediante Toast en caso de éxito o error, tanto al cargar como al modificar las citas.

- Y creamos también el adaptador **CitaAdapter.kt**

```
manifests
└ AndroidManifest.xml

kotlin+java
com.example.proyectofinalandroid
    ↪ AgendarCitaActivity
    ↪ AuthApi
    ↪ Cita
    ↪ CitaAdapter
    ↪ CrearUsuarioActivity
    ↪ LoginActivity
    ↪ MainActivity
    ↪ ModificarDatosActivity
    ↪ Servicio
    ↪ Usuario
    ↪ VerCitasActivity
com.example.proyectofinalandroid (andr)
    ↪ ExampleInstrumentedTest
com.example.proyectofinalandroid (test)
java (generated)
res
    drawable
    layout
        <activity_agendar_cita.xml
        <activity_crear_usuario.xml
        <activity_login.xml
        <activity_main.xml
        <activity_modificar_datos.xml
            ↪ LoginActivity
            ↪ MainActivity
            ↪ ModificarDatosActivity
            ↪ Servicio
            ↪ Usuario
            ↪ VerCitasActivity
com.example.proyectofinalandroid (andr)
    ↪ ExampleInstrumentedTest
com.example.proyectofinalandroid (test)
java (generated)
res
    drawable
    layout
        </activity_agendar_cita.xml

8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49

class CitaAdapter(
    private var citas: List<Cita>,
    private val editar: (Cita) -> Unit,
    private val eliminar: (Cita) -> Unit
) : RecyclerView.Adapter<CitaAdapter.CitaViewHolder>() {

    class CitaViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {
        val fechaTextView: TextView = itemView.findViewById(R.id.fechaTextView)
        val servicioTextView: TextView = itemView.findViewById(R.id.servicioTextView)
        val precioTextView: TextView = itemView.findViewById(R.id.precioTextView)
        val tiempoTextView: TextView = itemView.findViewById(R.id.tiempoTextView)
        val btnEditar: View = itemView.findViewById(R.id.btnEditar)
        val btnEliminar: View = itemView.findViewById(R.id.btnEliminar)
    }

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): CitaViewHolder {
        val view = LayoutInflater.from(parent.context).inflate(R.layout.item_cita, parent, attachToRoot: false)
        return CitaViewHolder(view)
    }

    override fun onBindViewHolder(holder: CitaViewHolder, position: Int) {
        val cita = citas[position]

        val fechaFormateada = cita.fecha.replace( oldValue: "T", newValue: " ").substringBeforeLast( delimiter: ":")

        holder.fechaTextView.text = "Fecha: $fechaFormateada"
        holder.servicioTextView.text = "Servicio: ${cita.servicio.nombre}"
        holder.precioTextView.text = "Precio: ${cita.servicio.precio}€"
        holder.tiempoTextView.text = "Tiempo: ${cita.servicio.tiempo} minutos"

        holder.precioTextView.text = "Precio: ${cita.servicio.precio}€"
        holder.tiempoTextView.text = "Tiempo: ${cita.servicio.tiempo} minutos"

        holder.btnEditar.setOnClickListener { editar(cita) }
        holder.btnDelete.setOnClickListener { eliminar(cita) }
    }

    override fun getItemCount(): Int = citas.size

    fun actualizarCitas(nuevasCitas: List<Cita>) {
        citas = nuevasCitas
        notifyDataSetChanged()
    }
}
```

La clase CitaAdapter es un adaptador personalizado para un RecyclerView que muestra una lista de objetos Cita. En cada elemento se presentan datos como la fecha (formateada), el nombre del servicio, el precio y la duración. También incluye botones para editar o eliminar cada cita, ejecutando las funciones onEditar o onEliminar cuando se presionan. El método actualizarCitas permite actualizar la lista de citas y refrescar la vista con notifyDataSetChanged().

- Por último, creamos el layout de la opción crear usuario que tenemos en la pantalla login **activity\_crear\_usuario.xml**

Screenshot of the Android Studio IDE showing the project structure and the XML layout file for the 'activity\_crear\_usuario.xml' activity.

**Project Structure:**

- AuthApi
- Cita
- CitaAdapter
- CrearUsuarioActivity
- LoginActivity
- MainActivity
- ModificarDatosActivity
- Servicio
- Usuario
- VerCitasActivity
- com.example.proyectofinalandroid (andriodTest)
- ExampleInstrumentedTest
- java (generated)
- res
  - drawable
  - layout
    - activity\_agendar\_cita.xml
    - activity\_crear\_usuario.xml
    - activity\_login.xml
    - activity\_main.xml
    - activity\_modificar\_datos.xml
    - activity\_ver\_citas.xml
    - item\_cita.xml
  - mipmap
  - values
    - colors.xml
    - strings.xml
- values
  - colors.xml
  - strings.xml

**XML Layout File (activity\_crear\_usuario.xml):**

```

<?xml version="1.0" encoding="utf-8"?>
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:card_view="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/fnd5"
    android:fillViewport="true">

    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:padding="24dp">

        <androidx.cardview.widget.CardView
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_centerInParent="true"
            card_view:cardCornerRadius="20dp"
            card_view:cardElevation="10dp"
            android:background="#FFFFFF"
            android:padding="28dp">

            <LinearLayout
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:orientation="vertical"
                android:gravity="center_horizontal">

                <TextView
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:orientation="vertical"
                    android:gravity="center_horizontal">

                    <Textview
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content"
                        android:text="Registro de Usuario"
                        android:textSize="26sp"
                        android:textStyle="bold"
                        android:textColor="#222222"
                        android:paddingBottom="24dp" />

                    <EditText
                        android:id="@+id/nombreEditText"
                        android:layout_width="match_parent"
                        android:layout_height="50dp"
                        android:hint="Nombre completo"
                        android:drawableStart="@android:drawable/ic_menu_myplace"
                        android:drawablePadding="10dp"
                        android:paddingHorizontal="16dp"
                        android:background="#E00087"
                        android:layout_marginBottom="18dp"
                        android:textColor="#0000"
                        android:textColorHint="#0088" />

                    <EditText
                        android:id="@+id/emailEditText"
                        android:layout_width="match_parent"
                        android:layout_height="50dp"
                        android:hint="Correo electrónico"
                        android:inputType="textEmailAddress"
                        android:drawableStart="@android:drawable/ic_dialog_email"
                        android:drawablePadding="10dp"
                        android:paddingHorizontal="16dp"
                        android:background="#E00087"
                        android:layout_marginBottom="18dp"
                        android:textColor="#0000"
                        android:textColorHint="#0088" />

                    <EditText
                        android:id="@+id/passwordEditText"
                        android:layout_width="match_parent"
                        android:layout_height="50dp"
                        android:hint="Contraseña"
                        android:inputType="textPassword"
                        android:drawableStart="@android:drawable/ic_lock_idle_light"
                        android:layout_height="50dp"
                        android:hint="Contraseña"
                        android:inputType="textPassword"
                        android:background="#E00087"
                        android:layout_marginBottom="24dp"
                        android:textColor="#0000"
                        android:textColorHint="#0088" />

                    <Button
                        android:id="@+id/crearUsuarioButton"
                        android:layout_width="match_parent"
                        android:layout_height="50dp"
                        android:text="Crear Cuenta"
                        android:textAllCaps="false"
                        android:backgroundTint="#BFB29F"
                        android:textColor="#FFFFFF"
                        android:textSize="16sp" />

```

**Preview:** The preview shows a registration form titled "Registro de Usuario" with fields for Nombre completo, Correo electrónico, Contraseña, and a "Crear Cuenta" button. The background features a wooden texture with hairdressing tools like a comb and scissors.

- Creamos también su activity **CrearUsuarioActivity.kt**

The screenshot shows the Android Studio interface with the project structure on the left and the code editor on the right.

**Project Structure:**

- com.example.proyectofinalandroid
  - src
    - kotlin+java
    - com.example.proyectofinalandroid
      - AgendarCitaActivity
      - AuthApi
      - Cita
      - CitaAdapter
      - CrearUsuarioActivity
      - LoginActivity
      - MainActivity
      - ModificarDatosActivity
      - Servicio
      - Usuario
      - VerCitasActivity
    - com.example.proyectofinalandroid (andr)
      - ExampleInstrumentedTest
    - com.example.proyectofinalandroid (test)
  - java (generated)
  - res
  - layout
    - activity\_agendar\_cita.xml
    - activity\_crear\_usuario.xml
    - activity\_login.xml
    - activity\_main.xml
    - activity\_modificar\_datos.xml

**Code Editor (CrearUsuarioActivity.kt):**

```

12 >< class CreaUsuarioActivity : AppCompatActivity() {
13
14     private lateinit var nombreEditText: EditText
15     private lateinit var emailEditText: EditText
16     private lateinit var passwordEditText: EditText
17     private lateinit var registrarButton: Button
18     private lateinit var authApi: AuthApi
19
20     override fun onCreate(savedInstanceState: Bundle?) {
21         super.onCreate(savedInstanceState)
22         setContentView(R.layout.activity_crear_usuario)
23
24         nombreEditText = findViewById(R.id.nombreEditText)
25         emailEditText = findViewById(R.id.emailEditText)
26         passwordEditText = findViewById(R.id.passwordEditText)
27         registrarButton = findViewById(R.id.crearUsuarioButton)
28
29         val logging = HttpLoggingInterceptor().apply {
30             level = HttpLoggingInterceptor.Level.BODY
31         }
32
33         val client = OkHttpClient.Builder()
34             .addInterceptor(logging)
35             .build()
36
37         val retrofit = Retrofit.Builder()
38             .baseUrl("http://10.0.2.2:8080/api/")
39             .client(client)
40
41         val retrofit = Retrofit.Builder()
42             .baseUrl("http://10.0.2.2:8080/api/")
43             .client(client)
44             .addConverterFactory(GsonConverterFactory.create())
45             .build()
46
47         authApi = retrofit.create(AuthApi::class.java)
48
49         registrarButton.setOnClickListener {
50             val nombre = nombreEditText.text.toString()
51             val email = emailEditText.text.toString()
52             val password = passwordEditText.text.toString()
53
54             if (nombre.isBlank() || email.isBlank() || password.isBlank()) {
55                 Toast.makeText(context, "Completa todos los campos", Toast.LENGTH_SHORT).show()
56                 return@registerOnClickListener
57             }
58
59             val nuevoUsuario = Usuario( id: null, nombre, email, password, rol: "CLIENTE")
60
61             authApi.crearUsuario(nuevoUsuario).enqueue(object : Callback<Usuario> {
62                 override fun onResponse(call: Call<Usuario>, response: Response<Usuario>) {
63                     if (response.isSuccessful) {
64                         Toast.makeText(context, "Cuenta creada correctamente", Toast.LENGTH_LONG).show()
65                         finish()
66                     } else {
67                         Toast.makeText(context, "Error al registrar", Toast.LENGTH_SHORT).show()
68                     }
69                 }
70
71                 override fun onFailure(call: Call<Usuario>, t: Throwable) {
72                     Toast.makeText(context, "Error de conexión", Toast.LENGTH_SHORT).show()
73                 }
74             })
75         }
76     }
77 }

```

- Y con esto acabaríamos la aplicación Android y el proyecto.