

Ingeniería en Software 2

# Trabajo Integrador N°1

Gimnasio Sport

- Alvarez Lucía
- Olivares Agustín
- Padilla Lumelli

# Introducción


- Se desarrolló un sistema para la gestión integral de un Gimnasio.
- Incluye:
  - Socios
  - Empleados
  - Rutinas
  - Cuotas
  - Pagos
  - Envío de mails
- Metodología utilizada: RUP

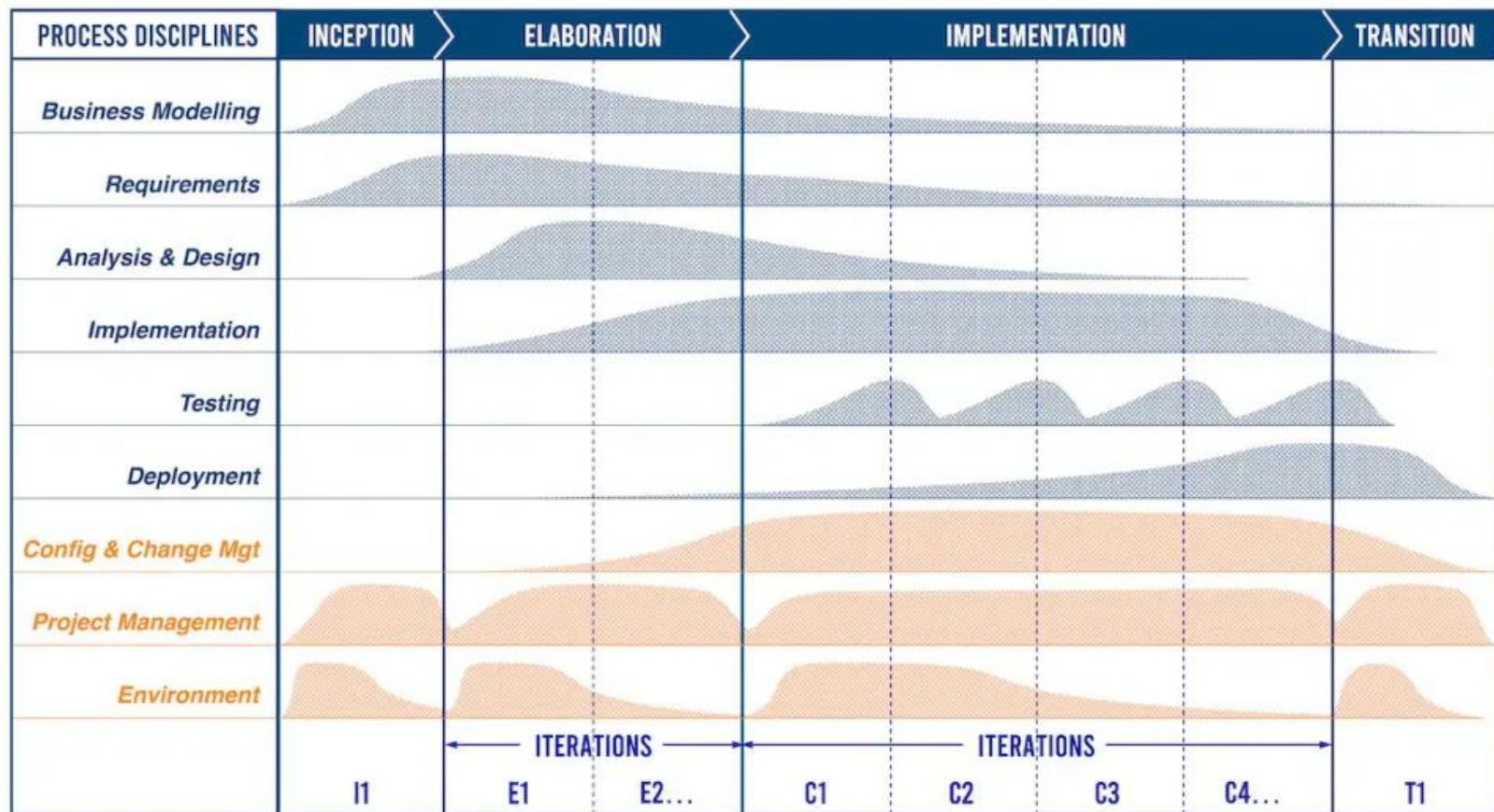


# RUP - ¿Qué es?

- Metodología de desarrollo Iterativa e Incremental.
- Divide el proyecto en fases claras.

## Fases:

- Inicio: Definir alcance, objetivos, casos de uso, riesgos iniciales.
  - Elaboración: Refinar requisitos, modelado y arquitectura. Diagramas.
  - Construcción: Implementación de funcionalidades.
  - Transición: Pruebas finales, despliegue y entrega al cliente.
- 



# RUP - Etapa de inicio

**Inicio:** Se define el alcance, objetivos, casos de uso, riesgos, estimaciones

Actores principales y objetivos:

Socio → ver rutinas, pagar cuotas, recibir notificaciones.

Profesor → crear y asignar rutinas.

Administrativo → gestionar usuarios, empleados, socios y cuotas.



# Casos de uso

## Críticos:

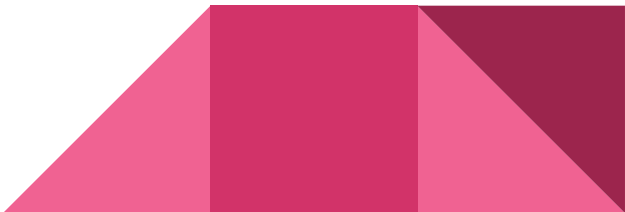
- ABM Valor Cuota y ABM Cuota Mensual
- Pago de cuota con su generación de factura y forma de pago
- **ABM Rutinas** ←
- Mostrar detalle de deuda y facturas pagadas de un mes
- Completar detalles rutinas realizadas, mostrar rutinas incumplidas, mostrar porcentaje cumplido de rutina
- Envío deuda, saludos de cumpleaños y cuota mensual por correo

## Prioridad media:

- ABM Empleados
- ABM Socios
- ABM Sucursal

## Prioridad baja:

- Login



# RUP - Etapa de diseño

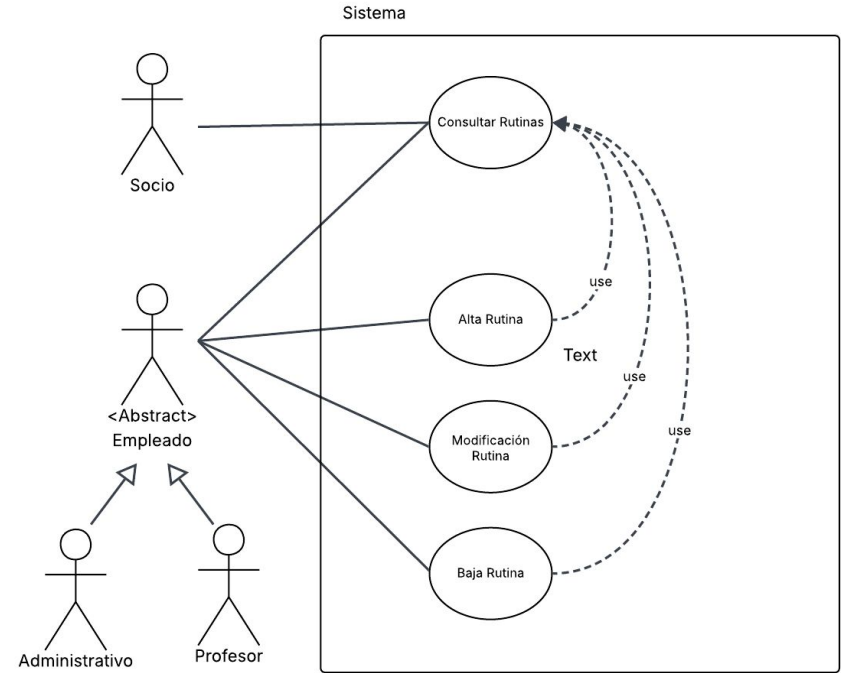
**Elaboración:** Realización de Diagramas.

- Diagrama de Caso de Uso
- Escenario de Caso de Uso
- Diagrama de Secuencia de Diseño

Además de un Prototipado del ABM Crítico: Rutinas



# Diagrama de Caso de Uso





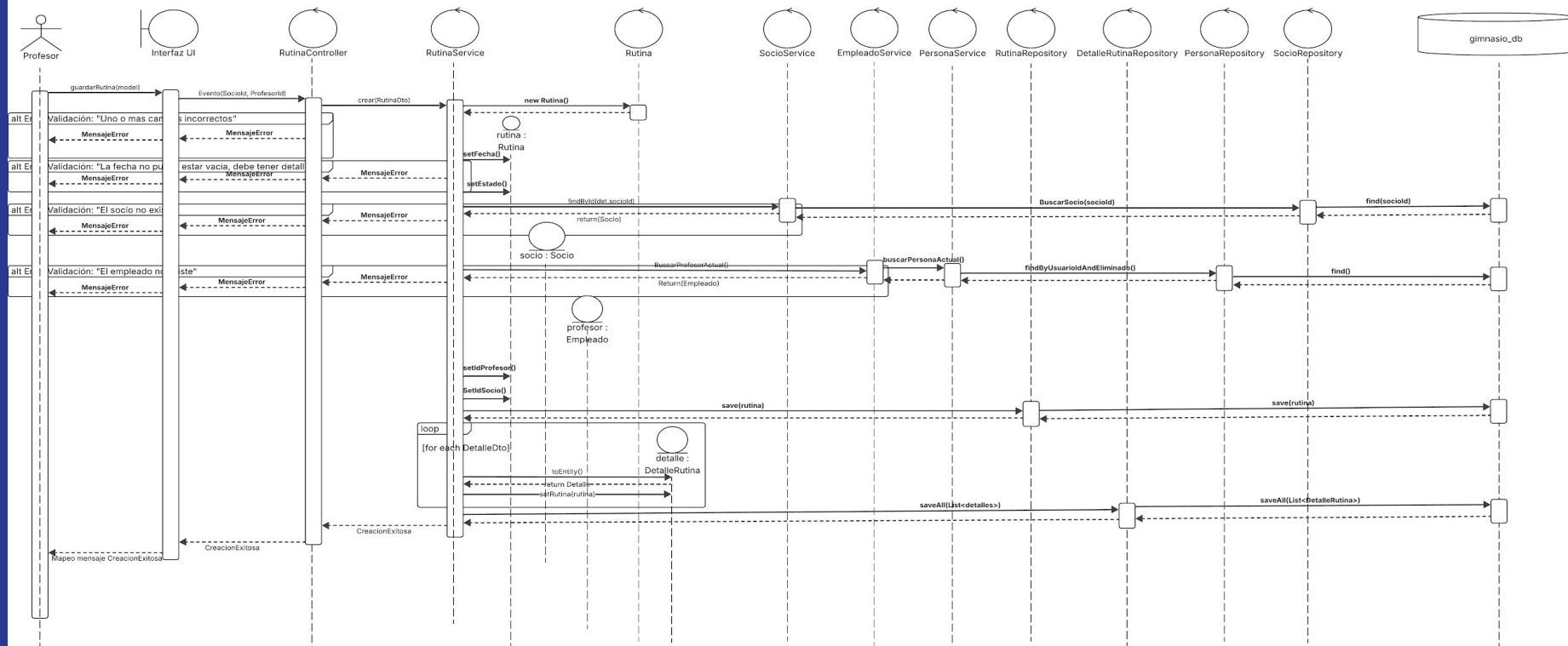
# Escenario de Caso de Uso

[https://drive.google.com/drive/folders/1GiD0L1fJzITqNIIdjex8HX-GPbk86\\_9Dw](https://drive.google.com/drive/folders/1GiD0L1fJzITqNIIdjex8HX-GPbk86_9Dw)

Caso de uso	Alta Rutina			ID:
Prioridad	Critica	Estimacion:		ID Pantalla prototipo
Precondicion	Existencia de Socio Registrado, Existencia de Profesor Registrado, Coincidencia entre las Sucursales de ambos			
Descripcion	El Profesor da de alta una Rutina junto con los Detalles para el Socio			
Escenario principal	Paso	Accion		
		1	El sistema muestra las rutinas existentes para cada socio a cargo	
		2	El profesor selecciona el socio	
		3	El profesor ingresa el estado de la rutina	
		4	El sistema carga fecha de inicio y de finalizacion por defecto	
		5	El profesor modifica las fechas (opcional)	
		6	El profesor ingresa los detalles de la rutina	
		7	El profesor confirma las configuraciones de la rutina	
		8	El sistema valida las configuraciones	
		9	El sistema crea la rutina	
Postcondicion				
Excepciones	5.1	Si la fecha termina antes de la creacion, Muestra mensaje de error y se vuelve al paso 5		
	6.1	Si la fecha del detalle no coincide con el periodo de la rutina, muestra mensaje de error y vuelve al paso 6		
	6.2	Si los campos de detalle tienen caracteres especiales, muestra mensaje de error y vuelve al paso 6		
Comentarios	Luego de crear la rutina, esta figura tambien para el socio y retorno al paso 1			

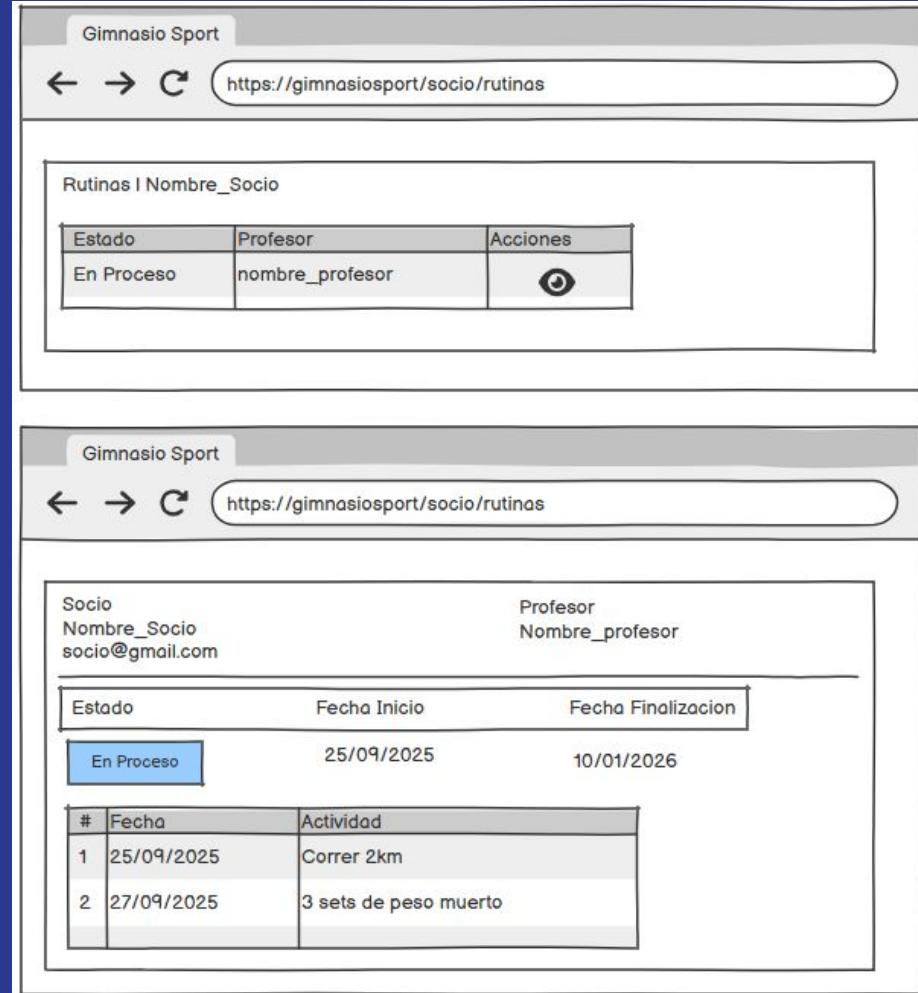
# Diagrama de Secuencia de Diseño

## Alta Rutina



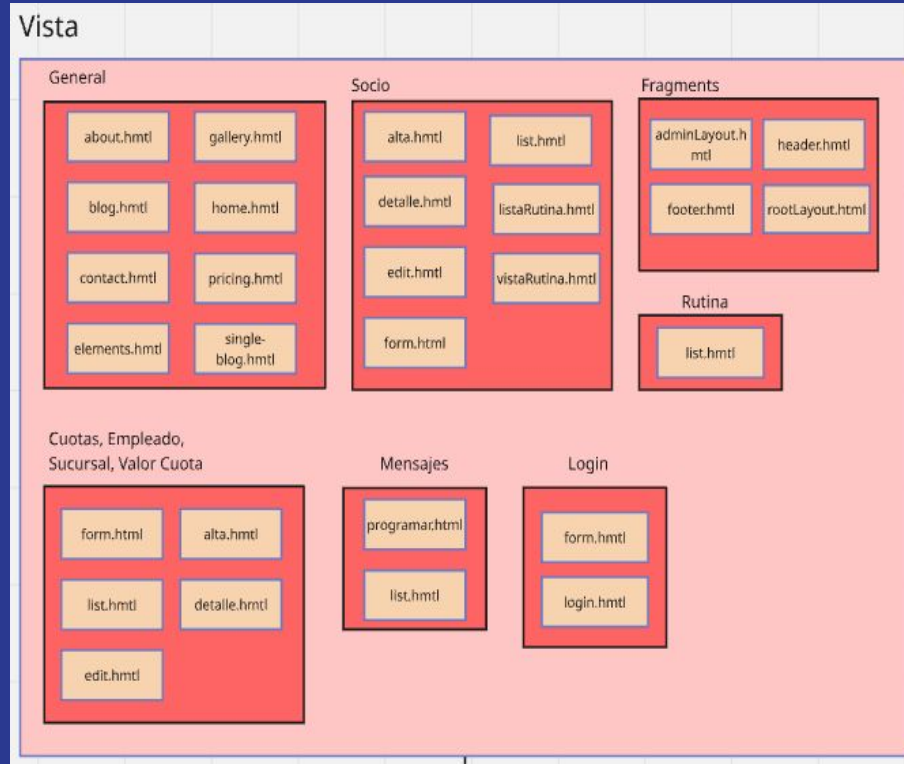
# Prototipado

<https://balsamiq.cloud/sx1kd8n/pxghah8/r3DB0>

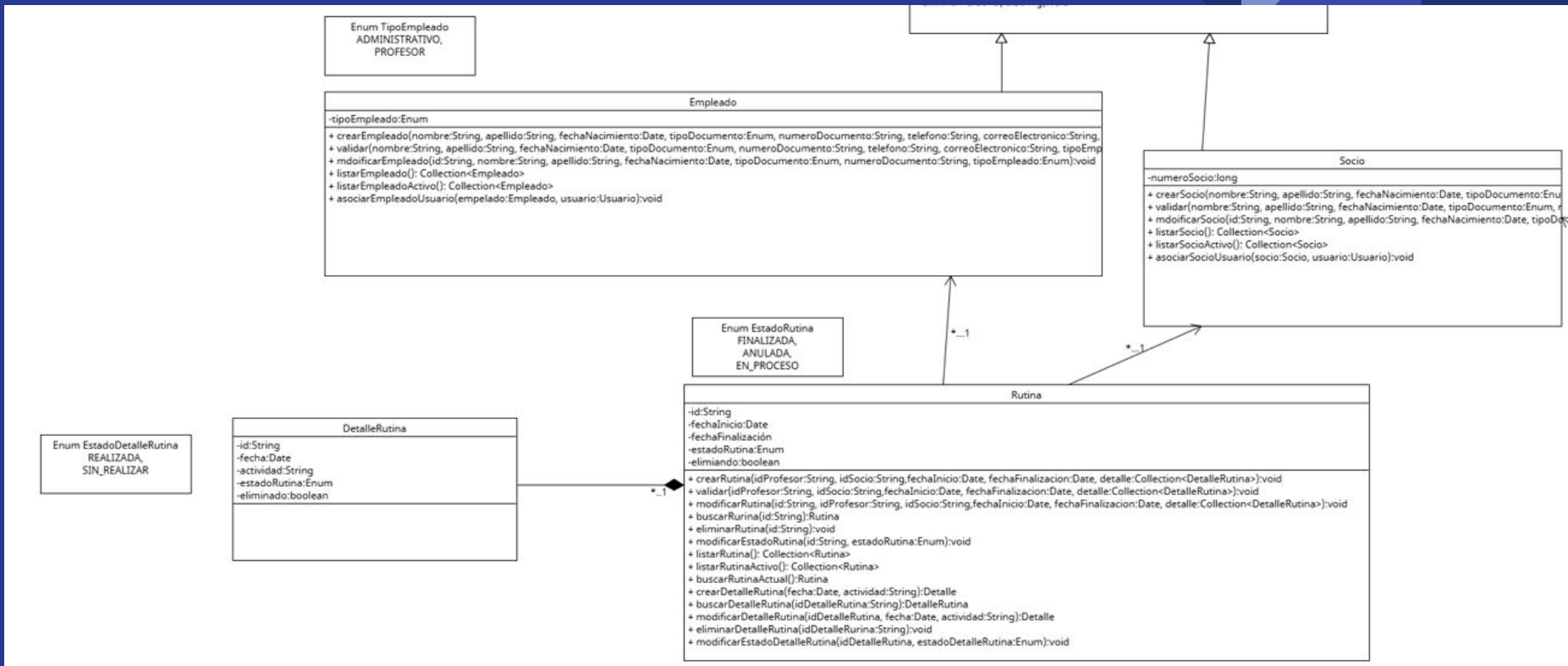


# Diagrama de Paquete de Software

[https://miro.com/app/board/uXjVJDg9C84=](https://miro.com/app/board/uXjVJDg9C84=/)

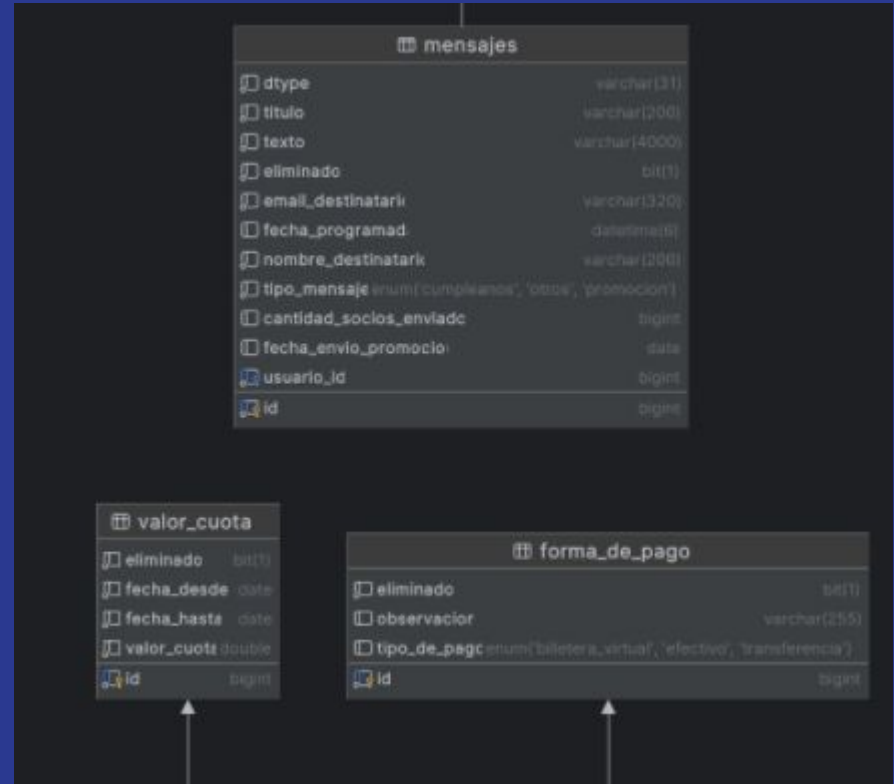


# Diagrama de Clases de Diseño

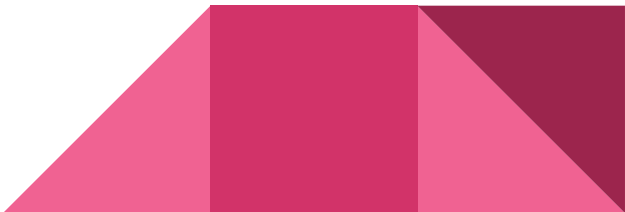


# Diagrama Entidad Relación

[https://miro.com/app/board/uXjVJDg9C84=](https://miro.com/app/board/uXjVJDg9C84=/)



# Requisitos No Funcionales

- Seguridad:
    - Autenticación de usuarios.
    - Roles restringen el acceso a funcionalidades no autorizadas.
    - Información sensible encriptada.
  - Usabilidad:
    - Interfaz intuitiva y clara
  - Mantenibilidad:
    - El código debe estar estructurado en capas
  - Escalabilidad
  - Compatibilidad
- 

# RUP - Etapa de construcción

**Construcción:** Se desarrolló en Spring Boot la mayoría de las funciones críticas (ABM de socios, pagos, rutinas).







# Demostración del sistema en uso



[gym.gpadilla.com](https://gym.gpadilla.com)

# Patrones Utilizados

- Inyección de Dependencias
- MVC
- Capas
- DTO

GRASP:

- Experto en información
- Creador
- Controlador
- Indirección
- Alta Cohesión / Bajo Acoplamiento
- Polimorfismo

GoF:

- Singleton
  - Factory Method
  - Proxy
-

# Inyección de dependencias

Ej. en Spring Boot

```
@Service
@RequiredArgsConstructor
public class UsuarioService {
    private final UsuarioRepository usuarioRepository;
    private final PasswordEncoder passwordEncoder;
    private final UsuarioMapper usuarioMapper;

    public Usuario crearUsuario(UsuarioCreateFormDTO formDto) {
        validarDatos(formDto.getClave(), formDto.getConfirmacionClave());

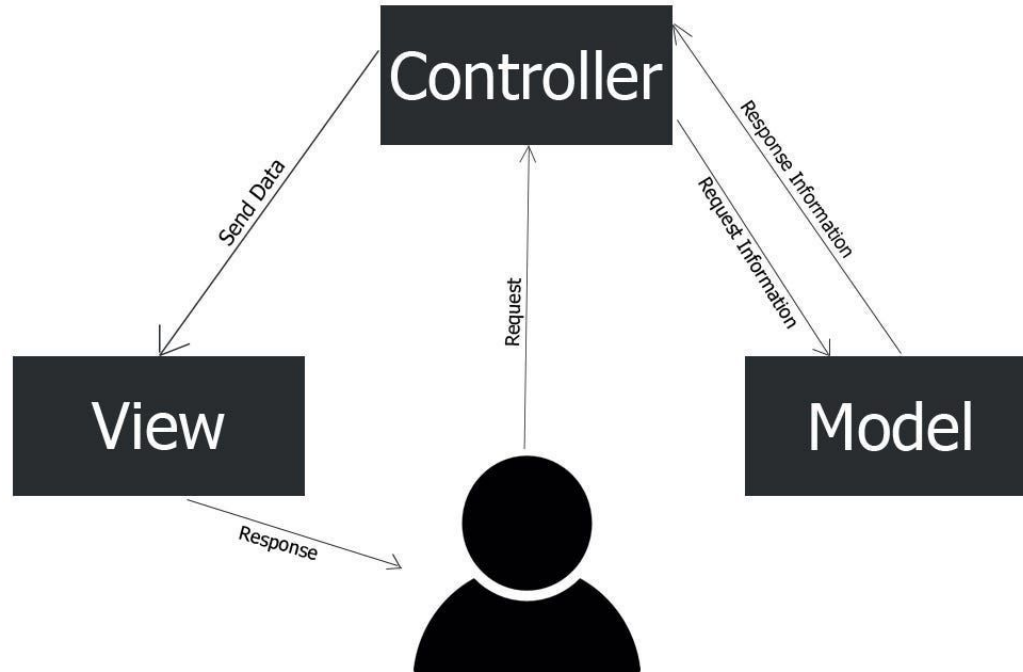
        if (usuarioRepository.existsByNombreUsuarioAndEliminadoFalse(formDto.getNombreUsuario()))
            throw new BusinessException("YaExiste.usuario.nombre");

        Usuario usuario = usuarioMapper.toEntity(formDto);
        String hashClave = passwordEncoder.encode(formDto.getClave());
        usuario.setClave(hashClave);

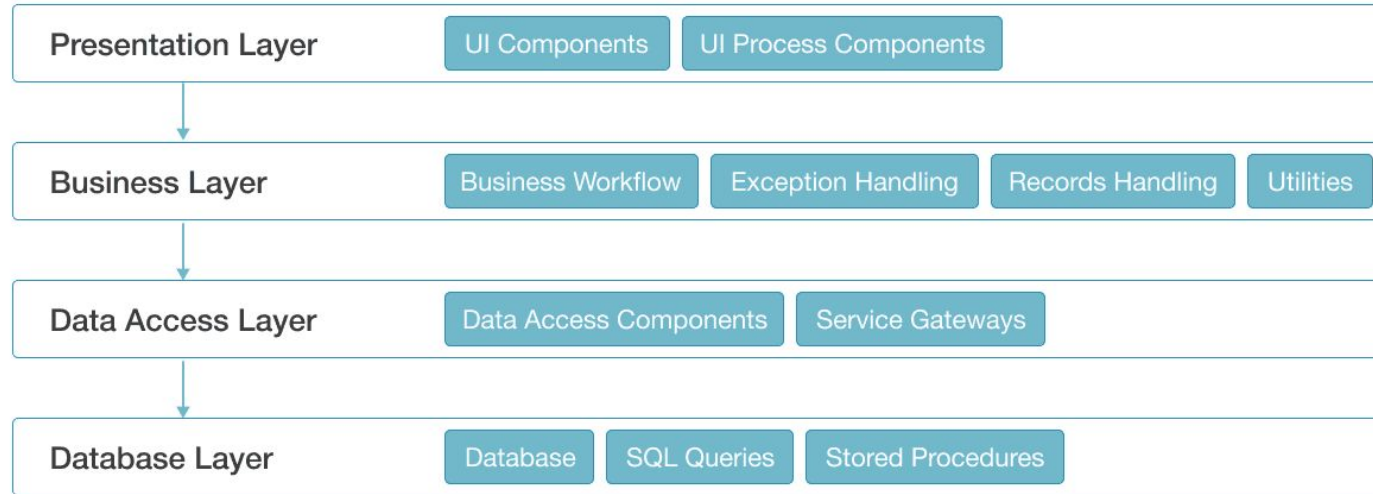
        return usuarioRepository.save(usuario);
    }
}
```

# MVC

## Model-View-Controller



# Capas



# DTO

```
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class SocioResumenDto {
    private Long id;
    private Long numeroSocio;
    private String nombre;
    private String apellido;
    private String correoElectronico;
    private String nombreSucursal;
}
```

```
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class RutinaDto {

    private Long id;

    @NotNull(message = "{NotNull.rutina.tipo}")
    private EstadoRutina tipo;

    @NotNull(message = "{NotNull.rutina.fechaInicio}")
    @DateTimeFormat(pattern = "yyyy-MM-dd")
    private LocalDate fechaInicio;

    @NotNull(message = "{NotNull.rutina.fechaFinalizacion}")
    @DateTimeFormat(pattern = "yyyy-MM-dd")
    private LocalDate fechaFinalizacion;

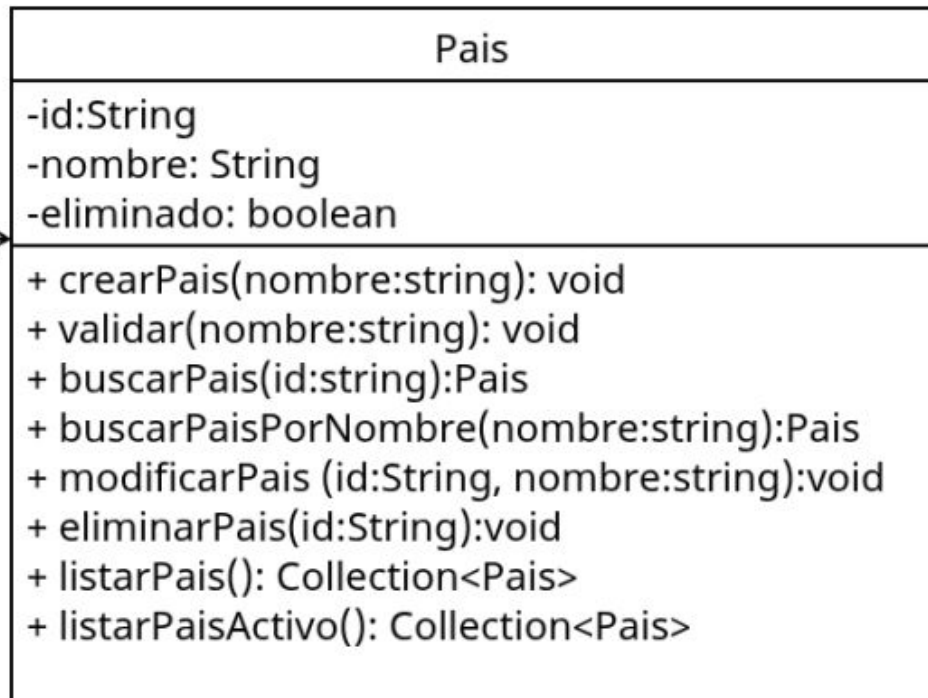
    @NotNull(message = "{NotNull.rutina.socio}")
    private Long socioId;
```

# Patrones GRASP

---



# Experto en información

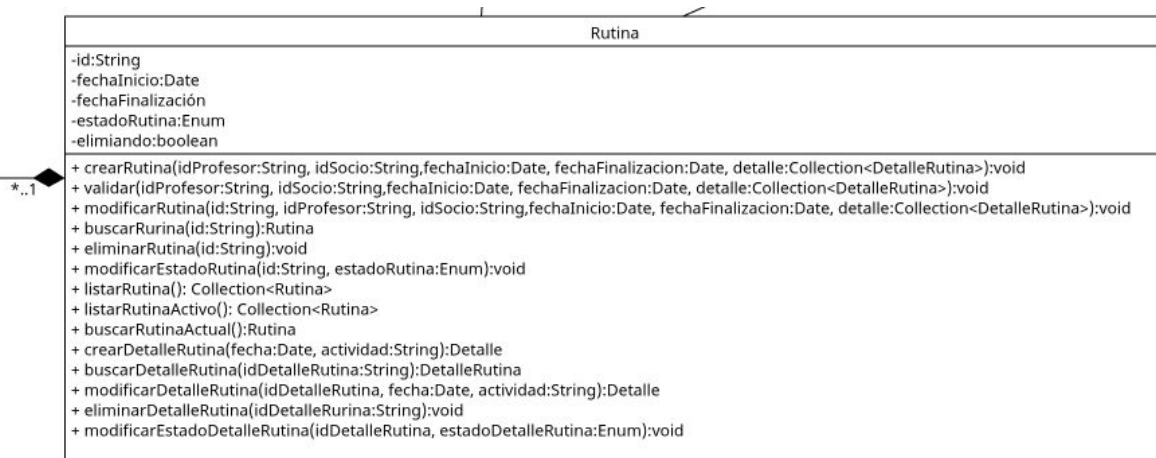
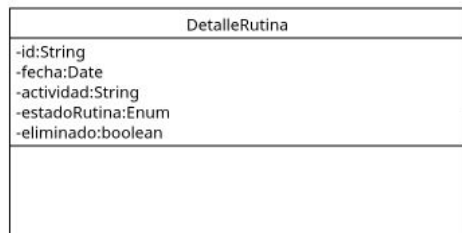


¿Cómo asignar responsabilidades a las clases?

Asignarlas a aquellas que tienen la información necesaria

# Creador

¿Quién debe crear a objetos de una clase?



# Controlador

¿Quién responde a los eventos de la interfaz?

```
@Controller
```

```
@RequiredArgsConstructor
```

```
public class SucursalController {
```

```
    private final SucursalService sucursalService;
```

```
    private final PaisService paisService;
```

```
@GetMapping("/sucursales")
```

```
public String listarSucursales(Model model) {
```

```
    try {
```

```
        return prepararVistaListaSucursales(model);
```

```
    } catch (BusinessException e) {
```

```
        model.addAttribute("msgError", e.getMessageKey());
```

```
    } catch (Exception e) {
```

```
        model.addAttribute("msgError", "error.sistema");
```

```
    }
```

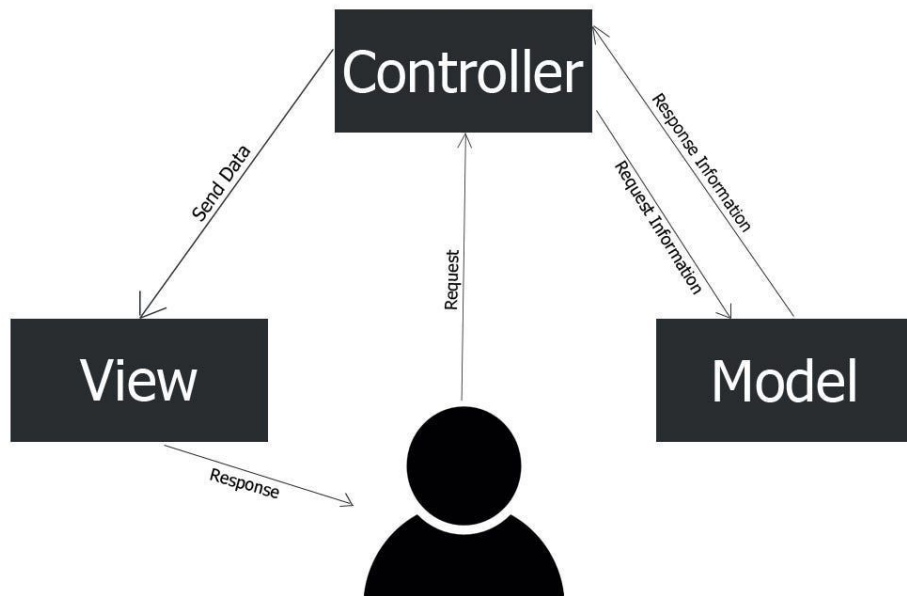
```
    return listView;
```

```
}
```

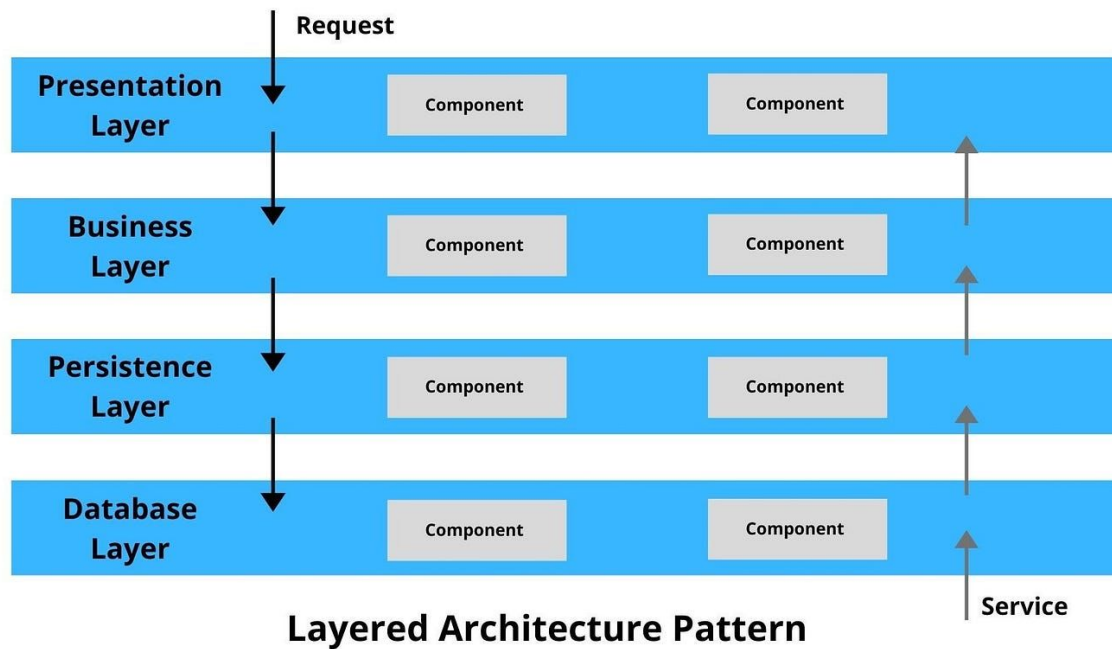
# Indirección

¿Cómo evitar acoplamiento directo entre clases? Ej. Patrón MVC

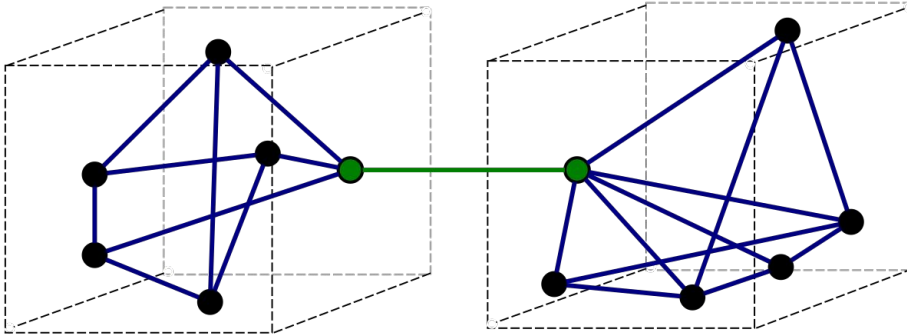
## Model-View-Controller



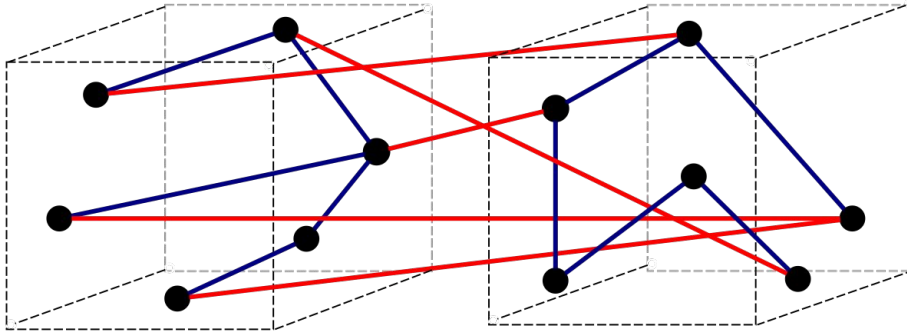
# Capas



# Alta Cohesión / Bajo Acoplamiento



a) Good (loose coupling, high cohesion)



b) Bad (high coupling, low cohesion)

# Polimorfismo

@Transactional

```
protected void crearPersona(Persona persona, PersonaCreateFormDTO formDto) {  
    UsuarioCreateFormDTO usuarioDto = formDto.getUsuario();  
    usuarioDto.setNombreUsuario(formDto.getCorreoElectronico());  
    Usuario usuario = usuarioService.crearUsuario(usuarioDto);  
  
    Direccion direccion = direccionService.crearDireccion(formDto.getDireccion());  
  
    Sucursal sucursal = sucursalService.buscarSucursal(formDto.getSucursalId());  
  
    personaMapper.updateEntityFromDto(formDto, persona);  
    persona.setSucursal(sucursal);  
    persona.setUsuario(usuario);  
    persona.setDireccion(direccion);  
    persona.setId(null);  
    persona.setEliminado(false);  
}
```

# Patrones GoF

---



# Singleton

Patrón Creacional. Utilizado por Spring.

```
@Service  
@RequiredArgsConstructor  
public class PaisService {
```



# Factory Method

Patrón creacional.  
Utilizado en Spring.

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

    @Autowired(required = false)
    private DevAutoLoginFilter devAutoLoginFilter;

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {

        http
            .csrf( CsrfConfigurer<HttpSecurity> csrf -> csrf.disable())
            .authorizeHttpRequests( AuthorizationManagerRequestMat... auth -> auth
                .requestMatchers(✓"/css/**", ✓"/fonts/**", ✓"/img/**", ✓"/js/"
                .requestMatchers(✓"/", ✓"/about", ✓"/blog", ✓"/contact", ✓"/"
                    ✓"gallery", ✓"pricing", ✓"single-blog", ✓"/error", ✓"/
                .requestMatchers(✓"/países/**").hasRole("ADMINISTRATIVO")
                .anyRequest().authenticated()
            )

        return http.build();
    }

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

# Proxy

Patrón estructural utilizado por Spring

```
@Transactional
```

```
public void modificarPais(PaisDto paisDto) {  
    Pais pais = buscarPais(paisDto.getId());  
  
    if (paisRepository.existsByNombreAndIdNotAndEliminadoFalse(paisDto.getNombre(), paisDto.getId()))  
        throw new BusinessException("YaExiste.pais.nombre");  
  
    paisMapper.updateEntityFromDto(paisDto, pais);  
    paisRepository.save(pais);  
}
```

# RUP - Etapa de transición

**Transición:** Aún pendiente, son las pruebas finales y el despliegue.



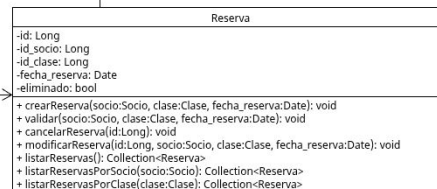
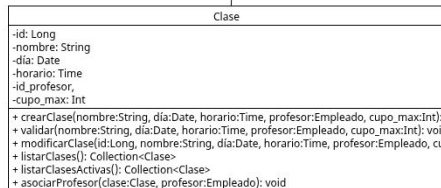
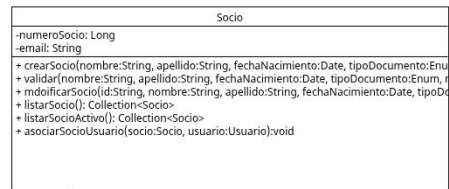
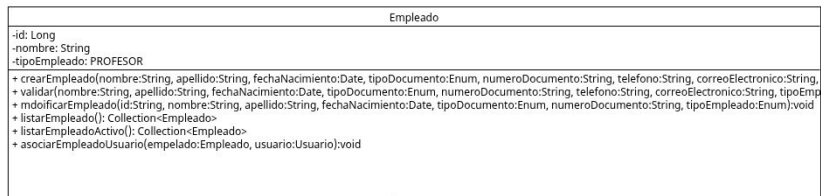
# Funcionalidad Propuesta

A pedido del cliente, se nos solicitó agregar nuevas funcionalidades al sistema. La propuesta fue de poder dictar clases especiales (Ej: Yoga, Boxeo), estas serán dictadas por un profesor, tendran un horario determinado y un cupo máximo de socios



# Diagrama Funcionalidad Propuesta

Enum TipoEmpleado  
ADMINISTRATIVO,  
PROFESOR



\*...1

\*...1

\*...1



¡Gracias!