UNIVERSIDAD AUTÓNOMA DE YUCATÁN

FACULTAD DE MATEMÁTICAS

ARQUITECTURAS DE SOFTWARE

PROYECTO INTEGRADOR

PROF. VICTOR HUGO MENENDEZ DIAZ

INTEGRANTES

CONTRERAS GAMBOA EMILIANO

DURAN VARELA ALEJANDRO MAGDIEL

JANSSEN AGUILAR HUGO DE JESÚS

Descripción de Funcionalidades

BACKEND

Configuraciones de entorno

Objetivo principal:

Ajustar las preferencias del entorno de producción y desarrollo.

Funcionalidades:

- Configuraciones de CORS para permitir el paso de solicitudes HTTP desde el dominio del front (ya sea desarrollo local o el que especifiquemos).
- Inicializar datos de prueba en la base de datos.
- Configuraciones de la cadena de filtros de seguridad.
- Definir el tipo de encriptación de contraseñas.
- Definir rutas permitidas para usuarios sin autenticación (es decir).

Manejo de usuarios y seguridad

Objetivo principal:

Gestionar la autenticación, autorización y control de acceso para cada tipo de usuario

Funcionalidades:

- Registrar nuevos usuarios en el sistema.
- Encriptar la contraseña de los usuarios.
- Login de usuarios.
- Logout de usuarios.
- Generación,validación e invalidación de tokens de sesión.
- Control de acceso basado en roles (ADMIN, CREADOR, USUARIO).

Templates

Objetivo principal:

Manejo de las funciones relacionadas con los templates

Funcionalidades:

- Creación de templates públicas o privadas (dependiendo del rol del usuario).
- Edición de templates privadas o privadas (dependiendo del rol del usuario).
- Eliminación de templates (Dependiendo del rol del usuario).
- Visualización de las templates según permisos.

- Listado de plantillas disponibles filtrado por rol del usuario autenticado.

Procesamiento de templates

Objetivo principal:

Procesar plantillas HTML estáticas convirtiéndolas en documentos dinámicos mediante la sustitución de placeholders.

Funcionalidades:

- Extraer los placeholders de la template estática con los nombres de los campos a introducir.
- Separar la información de los placeholders de la template
- Procesa una plantilla HTML reemplazando los placeholders con datos reales.
- Validar que los datos proporcionados coincidan con los placeholders requeridos.

Generación PDF

Objetivo principal:

Recibir los templates en HTML ya procesados, junto con los datos a introducir y generar el PDF para impresión.

Funcionalidades:

- Valida la entrada de la solicitud para generar un nuevo pdf.
- Busca y carga la template solicitada desde la base de datos.
- Verificar permisos de acceso a la plantilla antes de procesarla.
- Traduce de un HTML a documento PDF y lo devuelve como un array de bytes.

URL DE DIAGRAMA DE COMPONENTES

FRONTEND

Módulo 1: Autenticación y Gestión de Usuarios

Objetivo Principal: Gestionar el acceso seguro a la aplicación, permitiendo a los usuarios autenticarse con credenciales existentes o registrar nuevas cuentas con roles específicos, manteniendo la persistencia de la sesión.

Funcionalidades:

- **Inicio de Sesión:** Validación de credenciales (correo y contraseña) contra el servidor y almacenamiento seguro del token JWT recibido.
- **Registro de Usuarios:** Creación de nuevas cuentas solicitando nombre, correo, contraseña y rol (Usuario o Creador).
- **Validación de Formularios:** Verificación en tiempo real del formato de correo electrónico, campos obligatorios y coincidencia de contraseñas.
- **Persistencia de Sesión:** Almacenamiento local del token de sesión y rol del usuario para mantener el acceso activo al reiniciar la app.
- **Gestión de Estados de Vista:** Alternancia dinámica entre el formulario de inicio de sesión y el de registro dentro de la misma pantalla.

Módulo 2: Dashboard y Navegación

Objetivo Principal: Actuar como el contenedor principal de la aplicación post-login, orquestando la navegación entre las diferentes herramientas del sistema y gestionando el ciclo de vida de la sesión.

Funcionalidades:

- **Verificación de Sesión:** Comprobación automática de la existencia de un token válido al cargar la vista; si no existe, redirige al login.
- **Navegación por Menú:** Cambio dinámico del contenido principal entre tres estados: "Ver plantillas", "Crear plantilla" y "Mis plantillas".
- **Cierre de Sesión:** Eliminación de las credenciales locales (token JWT) y redirección segura a la pantalla de login.
- **Inyección de Dependencias:** Provisión del ViewModel específico (Provider) necesario para cada sub-vista seleccionada.

Módulo 3: Generación de Documentos

Objetivo Principal: Permitir al usuario seleccionar una plantilla, completar la información variable (campos simples o tablas dinámicas) y generar/descargar un documento PDF final.

Funcionalidades:

- **Visualización de Plantillas:** Despliegue de un carrusel con las plantillas disponibles obtenidas del servidor.

- **Generación Dinámica de Formularios:** Creación automática de campos de texto (TextControllers) basada en los placeholders detectados en la plantilla seleccionada.
- **Manejo de Tablas Dinámicas (Bucles):** Capacidad para añadir o eliminar filas dinámicamente para secciones repetitivas de la plantilla.
- **Previsualización de PDF:** Renderizado en pantalla del documento PDF generado a partir de los datos ingresados.
- **Descarga de Archivos:** Guardado local del documento PDF generado con un nombre único basado en la fecha y hora.

Módulo 4: Creación de Plantillas

Objetivo Principal: Proveer una interfaz para que los usuarios con permisos elevados (Creadores/Admins) puedan diseñar y guardar nuevas plantillas de documentos en el sistema.

Funcionalidades:

- **Edición de Plantilla:** Formularios para ingresar el nombre de la plantilla y su estructura en formato HTML/Texto.
- **Configuración de Visibilidad:** Opción para marcar una plantilla como "Pública" (visible para todos), habilitada solo para roles de Administrador o Creador.
- **Validación de Entrada:** Verificación de que el nombre y el contenido no estén vacíos antes de enviar.
- **Persistencia de Plantilla:** Envío de la nueva estructura al servidor para su almacenamiento.

Módulo 5: Gestión de Mis Plantillas (Edición)

Objetivo Principal: Administrar el portafolio personal de plantillas del usuario, permitiendo la modificación de contenido o la eliminación de plantillas privadas.

Funcionalidades:

- **Filtrado de Plantillas Privadas:** Carga y visualización exclusiva de las plantillas creadas por el usuario actual (no públicas).
- **Edición de Contenido:** Carga de los datos actuales de una plantilla en controladores de texto para su modificación y actualización.

- **Eliminación de Plantillas:** Funcionalidad para borrar permanentemente una plantilla, con validación para asegurar que no se borren plantillas públicas.
- **Manejo de Errores y Feedback:** Visualización de mensajes de error o éxito tras las operaciones de actualización o borrado.

Descripción de Componentes

Gestión de seguridad y autenticación
Descripción: Responsable de manejar todo el ciclo de autenticación, autorización y control de acceso basado en roles (RBAC). Gestiona el registro, login, logout de usuarios, así como la generación y validación de tokens JWT. **Dependencias con otros componentes:** Base de datos y componente de configuración. **Interfaces de salida:** • **AuthService:** Provee operaciones de registro • **TokenBlackListService:** Gestiona la invalidación de tokens • **UserRepository:** Acceso a operaciones CRUD de usuarios **Interfaces de entrada:** • **PasswordEncoder:** Requiere servicio de encriptación BCrypt • **DatabaseConnection:** Requiere conexión a base de datos H2 en local y PostgreSQL en base de datos de un servidor. **Artefactos:** **Clases:** AuthController.java, AuthService.java, JwtUtils.java, JwtFilter.java, SecurityConfig.java, CorsConfig.java, UserRepository.java, User.java, Role.java **Bibliotecas:** springbootstartersecurity, jjwtapi **Configuración:** application.properties
Gestión de plantillas
Descripción: Componente encargado del ciclo de vida completo de las plantillas HTML dinámicas (CRUD), aplica lógica de autorización basada en roles y propiedad. **Dependencias con otros componentes:** Componente de seguridad, Componente de procesamiento de plantillas, Base de datos. **Interfaces de salida:** • **TemplateService:** Provee operaciones CRUD

- **TemplateRepository:** Acceso a operaciones de persistencia de plantillas.

Interfaces de entrada:

- **TemplateProcessor.exctractPlaceholders():** Requiere servicio de extraccion de placeholders
- **AuthContext:** Requiere información del usuario autenticado
- **DatabaseConnection:**

Artefactos:

Clases: TemplateController.java, TemplateService.java, TemplateRepository.java, Template.java

Bibliotecas: Springbootstarterdatajpa

Recursos: Plantillas Html en resources/templates

Procesamiento de Plantillas

Descripción:

Componente especializado en el procesamiento de plantillas HTML utilizando el motor Mustache. Extrae placeholders y reemplaza marcadores con datos dinámicos.

Dependencias con otros componentes:

Ninguna, es un componente desacoplado

Interfaces de salida:

- **TemplateProcessor.extractPlaceholders():**
- **TemplateProcessor.processTemplate():**

Interfaces de entrada:

Ninguna

Artefactos:

Clases: TemplateProcessor.java

Bibliotecas: mustache (para compilación y procesamiento de plantillas)

Generación de PDF

Descripción:

Componente responsable de convertir plantillas HTML procesadas en documentos PDF descargables. Valida permisos, procesa datos y genera el archivo binario.

Dependencias con otros componentes:

Componente de gestión de plantillas, componente de procesamiento de plantillas y componente de seguridad.

Interfaces de salida:

- **PDFGenerationService.generatePdf():** Genera PDF a partir de template ID y datos
- **PDFController.generatePdf():** Endpoint REST para generación de PDF

Interfaces de entrada:

- **TemplateService.findById():** Requiere obtener plantilla por ID

- **TemplateService.validateTemplatePlaceholders():** Requiere validación de datos
- **TemplateProcessor.processTemplate():** Requiere procesamiento de HTML

Artefactos:

Clases: PdfController.java, PdfGenerationService.java, PdfGenerationRequest.java
Bibliotecas: flyingsaucerpdfopenpdf

Configuración del entorno

Descripción:
Componente de infraestructura que configura el entorno de ejecución, incluyendo CORS, base de datos, seguridad y datos iniciales (seeding).

Dependencias con otros componentes:
Todos los componentes

Interfaces de salida:

- **CorsConfigurationSource:** Configuración de CORS
- **SecurityFilterChain:** Cadena de filtros de seguridad
- **DataSeeder:** inicialización de datos de prueba

Interfaces de entrada:

- **Variables de entorno:** DB_URL, DB_USERNAME, DB_PASSWORD, JWT_SECRET

Artefactos:

Clases: SecurityConfig.java, CorsConfig.java, DataInitializer.java
Configuración: application.properties, .env.example
Bibliotecas: springbootdevtools, h2database, postgresql

Persistencia

Descripción:
Componente de persistencia que almacena usuarios, plantillas y relaciones. Soporta H2 (desarrollo) y PostgreSQL (producción).

Dependencias con otros componentes:
Ninguna

Interfaces de salida:

- **DatabaseConnection:** conexión JDB a la DB
- **EntityManager:** Gestión de entidades JPA

Interfaces de entrada:

- **Configuracion:** URL, username, password de conexion

Artefactos:

- **Bibliotecas:** springbootstarterdatajpa, HikariCP
- **Drivers:** h2 (para pruebas), postgresql (produccion)
- **Esquema:** autogenerado por JPA

URL al diagrama de Componentes:

Descripción de Clases

Para la descripción de las clases se hará uso de un listado de las mismas, donde cada uno se detallará las funciones y atributos:

- Nombre de la clase:
- Descripción:
- Dependencias con otras clases: Listar las asociaciones, nombre y descripción (¿para qué se tiene dicha asociación?)
- Atributos: Enumerarlas y adicionar el nombre, tipo, visibilidad, valor por omisión y descripción
- Funciones: Enumerarlas y adicionar el nombre, listado de argumentos con su tipo, valor de retorno, visibilidad, si es función pública mencionar el servicio que está implementando (componente e interface de salida) y descripción
- Añadir diagrama de clases

Entities

User

Descripción:

Entidad que representa a un usuario del sistema. Almacena información de autenticación, autorización y relación con sus plantillas.

Dependencias con otras clases:

- **Template** (Relación OneToMany): Un usuario puede poseer múltiples plantillas. Asociación bidireccional para gestionar propiedad de plantillas.
- **Role** (Enumeración): Define el nivel de permisos del usuario (ADMIN, CREADOR, USUARIO).

Atributos:

- **Id:** int, private, null, identificador unico, generado automáticamente
- **name**: String, private, null, nombre completo del usuario

- **email**: String, private, null, correo electrónico único del usuario (credencial de login)
- **password**: String, private, null, contraseña encriptada con BCrypt
- **role**: Role, private, null, rol del usuario (ADMIN, CREADOR, USUARIO)
- **templates**: List<Template>, private, null, lista de plantillas propiedad del usuario

Funciones:

- **User()**: sin argumentos, void, public, constructor por defecto requerido por JPA
- **getId()**: sin argumentos, Long, public, obtiene el ID del usuario
- **setId(Long id)**: Long id, void, public, establece el ID del usuario
- **getName()**: sin argumentos, String, public, obtiene el nombre del usuario
- **setName(String name)**: String name, void, public, establece el nombre del usuario
- **getEmail()**: sin argumentos, String, public, obtiene el email del usuario
- **setEmail(String email)**: String email, void, public, establece el email del usuario
- **getPassword()**: sin argumentos, String, public, obtiene la contraseña encriptada
- **setPassword(String password)**: String password, void, public, establece la contraseña
- **getRole()**: sin argumentos, Role, public, obtiene el rol del usuario
- **setRole(Role role)**: Role role, void, public, establece el rol del usuario
- **getTemplates()**: sin argumentos, List, public, obtiene lista de plantillas
- **setTemplates(List templates)**: List templates, void, public, establece lista de plantillas

Template

Descripción:

Representa una plantilla de documento HTML con placeholders dinámicos. Puede ser pública o privada según el propietario y su rol.

Dependencias con otras clases:

- **User**: Cada plantilla tiene un propietario. Asociación para validar permisos de edición/eliminación.
- **TemplateService**: Servicio que gestiona la lógica de negocio de las plantillas.

Atributos:

- **id**: Long, private, null, identificador único (PK) generado automáticamente
- **name**: String, private, null, nombre descriptivo de la plantilla

- **content**: String, private, null, contenido HTML con placeholders
- **owner**: User, private, null, usuario propietario de la plantilla
- **isPublic**: boolean, private, false, indica si la plantilla es pública o privada
- **placeholders**: List<String>, private, null, lista de nombres de placeholders extraídos del contenido

Funciones:

- **Template()**: sin argumentos, void, public, constructor por defecto requerido por JPA
- **getId()**: sin argumentos, Long, public, obtiene el ID de la plantilla
- **setId(Long id)**: Long id, void, public, establece el ID de la plantilla
- **getName()**: sin argumentos, String, public, obtiene el nombre de la plantilla
- **setName(String name)**: String name, void, public, establece el nombre de la plantilla
- **getContent()**: sin argumentos, String, public, obtiene el contenido HTML
- **setContent(String content)**: String content, void, public, establece el contenido HTML
- **getOwner()**: sin argumentos, User, public, obtiene el propietario de la plantilla
- **setOwner(User owner)**: User owner, void, public, establece el propietario
- **isPublic()**: sin argumentos, boolean, public, verifica si la plantilla es pública
- **setPublic(boolean isPublic)**: boolean isPublic, void, public, establece la visibilidad
- **getPlaceholders()**: sin argumentos, List<String>, public, obtiene lista de placeholders
- **setPlaceholders(List<String> placeholders)**: List<String> placeholders, void, public, establece lista de placeholders

Role (enum)

Descripción:
Enumeración que define los tres niveles de autorización en el sistema.

Valores:

- **ADMIN**: Administrador con acceso completo
- **CREADOR**: Usuario que crea plantillas públicas
- **USUARIO**: Usuario estándar con plantillas privadas

Services

AuthService

Descripción:
Servicio de autenticación y autorización de usuarios. Gestiona el registro, login, logout y validación de tokens JWT.

Dependencias con otras clases:

- **UserRepository**: Repositorio para operaciones CRUD de usuarios en la BD.
- **JwtUtils**: Utilidad para generación y validación de tokens JWT.
- **PasswordEncoder**: Servicio de Spring Security para encriptar contraseñas con BCrypt.
- **TokenBlacklistService**: Servicio para invalidar tokens al hacer logout.

Atributos:

- **userRepository**: UserRepository, private, inyección del repositorio de usuarios
- **jwtUtils**: JwtUtils, private, inyección de utilidad para manejo de JWT
- **passwordEncoder**: PasswordEncoder, private, inyección de encoder BCrypt
- **tokenBlacklistService**: TokenBlacklistService, private, inyección del servicio de blacklist

Funciones:

- **register(User user)**: User user, User, public, Componente: Gestión de Seguridad | Interface: IAuthService, registra nuevo usuario, encripta contraseña y asigna rol USUARIO por defecto
- **login(String email, String password)**: String email y String password, Map<String, Object>, public, Componente: Gestión de Seguridad | Interface: IAuthService, valida credenciales y retorna token JWT con datos del usuario
- **logout(String token)**: String token, void, public, Componente: Gestión de Seguridad | Interface: IAuthService, invalida el token agregándolo a la blacklist
- **getUserFromToken(String token)**: String token, User, public, Componente: Gestión de Seguridad | Interface: IAuthService, extrae y retorna el usuario asociado al token JWT

TemplateService

Descripción:
Servicio que implementa la lógica de negocio para gestionar plantillas.
Aplica reglas de autorización basadas en roles y propiedad.

Dependencias con otras clases:

- **TemplateRepository**: Repositorio para operaciones CRUD de plantillas en la BD.
- **TemplateProcessor**: Servicio para extraer placeholders del contenido HTML.
- **User**: Entidad para validar permisos del usuario autenticado.

Atributos:

- **templateRepository**: TemplateRepository, private, inyección del repositorio de plantillas
- **templateProcessor**: TemplateProcessor, private, inyección del procesador de plantillas

Funciones:

- **save(Template template, User authUser)**: Template, public, guarda plantilla aplicando lógica de roles.
- **findAllByRole(User authUser)**: User authUser, List<Template>, public, lista plantillas según rol.
- **findById(Long id, User authUser)**: Long id y User authUser, Template, public, busca plantilla por ID verificando permisos de lectura
- **update(Long id, Template details, User authUser)**: Long id, Template details y User authUser, Template, public, actualiza plantilla verificando que sea propietario o ADMIN
- **delete(Long id, User authUser)**: Long id y User authUser, void, public, elimina plantilla (públicas solo ADMIN, privadas propietario o ADMIN)
- **validateTemplatePlaceholders(Template template, Map<String, Object> data)**: Template template y Map<String, Object> data, void, public, valida que todos los placeholders requeridos estén en los datos

TemplateProcessor

Descripción:
Servicio para procesar plantillas HTML con el motor Mustache. Extrae placeholders y reemplaza marcadores con datos dinámicos.

Dependencias con otras clases:

- **MustacheFactory**: Biblioteca externa para compilación de plantillas Mustache.

Atributos:
Ninguno

Funciones:

- **extractPlaceholders(String templateContent)**: String templateContent, List<String>, public, Componente: Procesamiento de Plantillas | Interface: ITemplateProcessor, extrae nombres de placeholders {{variable}} usando Regex
- **processTemplate(String templateContent, Map<String, Object> data)**: String templateContent y Map<String, Object> data, String, public, procesa plantilla HTML reemplazando placeholders con datos reales usando Mustache

PdfGenerationService

Descripción:
Servicio responsable de convertir plantillas HTML procesadas en documentos PDF. Orquesta el proceso completo de generación.

Dependencias con otras clases:

- **TemplateRepository**: Repositorio para cargar plantillas desde la BD.
- **TemplateProcessor**: Servicio para procesar plantillas con datos.
- **ITextRenderer**: Biblioteca externa para conversión HTML a PDF.

Atributos:

- **templateRepository**: TemplateRepository, private, inyección del repositorio de plantillas
- **templateProcessor**: TemplateProcessor, private, inyección del procesador de plantillas

Funciones:

- **generatePdf(GenerationRequest request)**: GenerationRequest request, byte[], public, procesa y convierte a PDF
- **validateData(GenerationRequest request)**: GenerationRequest request, void, private, valida que el request tenga templateType y data no vacíos
- **loadTemplateByType(String templateType)**: String templateType, Template, public, carga plantilla desde la BD por su nombre
- **validatePlaceholders(Template template, Map<String, Object> data)**: Template template y Map<String, Object> data, void, private, valida que todos los placeholders estén presentes en los datos
- **convertHtmlToPdf(String html)**: String html, byte[], private, convierte HTML procesado a PDF usando Flying Saucer

TokenBlackListService

Descripción:
Servicio para gestionar tokens JWT invalidados (logout). Mantiene una lista en memoria de tokens bloqueados.

Dependencias con otras clases:
Ninguna

Atributos:

- **blacklistedTokens**: Set<String>, private, new HashSet<>(), conjunto de tokens invalidados

Funciones:

- **addToBlacklist(String token)**: String token, void, public, agrega token a la blacklist

- **isBlacklisted(String token)**: String token, boolean, public, verifica si un token está en la blacklist

Controllers

Authcontroller

Descripción:
Controlador REST que expone endpoints de autenticación y autorización.

Dependencias con otras clases:

- **AuthService**: Servicio de lógica de negocio para autenticación.
- **User**: Entidad para recibir datos de registro/login.

Atributos:

- **authService**: AuthService, private, inyección del servicio de autenticación

Funciones:

- **register(@RequestBody User user)**: User user, ResponseEntity<User>, public, Endpoint REST, POST /api/auth/register Registra nuevo usuario
- **login(@RequestBody Map<String, String> credentials)**: Map<String, String> credentials, ResponseEntity<Map<String, Object>>, public, Endpoint REST, POST /api/auth/login Inicia sesión y retorna JWT
- **logout(@RequestHeader("Authorization") String token)**: String token, ResponseEntity<String>, public, Endpoint REST, POST /api/auth/logout Invalida token JWT

TemplateController

Descripción:
Controlador REST que expone endpoints CRUD para gestión de plantillas.

Dependencias con otras clases:
TemplateService: Servicio de lógica de negocio para plantillas.
AuthService: Servicio para obtener usuario autenticado del token.

Atributos:

- **templateService**: TemplateService, private, inyección del servicio de plantillas
- **authService**: AuthService, private, inyección del servicio de autenticación

Funciones:

- **getAllTemplates(@RequestHeader("Authorization") String token)**: String token, ResponseEntity<List<Template>>, public, Endpoint REST, GET /api/templates Lista plantillas según rol

- **getTemplateById(@PathVariable Long id, @RequestHeader String token)**: Long id y String token, ResponseEntity<Template>, public, Endpoint REST, GET /api/templates/{id} Obtiene plantilla por ID
- **createTemplate(@RequestBody Template template, @RequestHeader String token)**: Template template y String token, ResponseEntity<Template>, public, Endpoint REST, POST /api/templates Crea nueva plantilla
- **updateTemplate(@PathVariable Long id, @RequestBody Template details, @RequestHeader String token)**: Long id, Template details y String token, ResponseEntity<Template>, public, Endpoint REST, PUT /api/templates/{id} Actualiza plantilla
- **deleteTemplate(@PathVariable Long id, @RequestHeader String token)**: Long id y String token, ResponseEntity<Void>, public, Endpoint REST, DELETE /api/templates/{id} Elimina plantilla

PdfController

Descripción:
Controlador REST que expone endpoint para generación de PDFs desde plantillas.

Dependencias con otras clases:

- **PdfGenerationService**: Servicio de lógica de negocio para generación de PDFs.
- **GenerationRequest**: DTO para recibir datos de generación.

Atributos:

- **pdfGenerationService**: PdfGenerationService, private, inyección del servicio de generación de PDFs

Funciones:

- **generatePdf(@RequestBody GenerationRequest request)**: GenerationRequest request, ResponseEntity<byte[]>, public, Endpoint REST, POST /api/pdf/generate Genera PDF desde plantilla y datos

Security

JwtUtils

Descripción:
Utilidad para generación, validación y extracción de información de tokens JWT.

Dependencias con otras clases:

- **User**: Entidad para extraer información del usuario del token.

Atributos:

- **SECRET_KEY**: String, private, clave secreta para firmar tokens (desde application.properties)

- **EXPIRATION_TIME**: long, private, tiempo de expiración del token en milisegundos

Funciones:

- **generateToken(String email)**: String email, String, public, genera token JWT con email como subject
- **validateToken(String token)**: String token, boolean, public, valida firma y expiración del token
- **getEmailFromToken(String token)**: String token, String, public, extrae el email del subject del token

JwtFilter

Descripción:
Filtro de Spring Security que intercepta peticiones HTTP y valida tokens JWT.

Dependencias con otras clases:

- **JwtUtils**: Utilidad para validar tokens.
- **UserRepository**: Repositorio para cargar usuario del token.
- **TokenBlacklistService**: Servicio para verificar tokens invalidados.

Atributos:

- **jwtUtils**: JwtUtils, private, inyección de utilidad JWT
- **userRepository**: UserRepository, private, inyección del repositorio de usuarios
- **tokenBlacklistService**: TokenBlacklistService, private, inyección del servicio de blacklist

Funciones:

- **doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain)**: HttpServletRequest request, HttpServletResponse response y FilterChain filterChain, void, protected, intercepta peticiones, valida JWT y autentica usuario en SecurityContext

Repositories

UserRepository

Descripción:
Repositorio para operaciones CRUD de usuarios. Extiende JpaRepository.

Funciones:

- **findByEmail(String email)**: String email, Optional<User>, public, busca usuario por email (query method de Spring Data JPA)

TemplateRepository

Descripción:
Repositorio para operaciones CRUD de plantillas. Extiende JpaRepository.

Funciones:

- **findByName(String name)**: String name, Optional<Template>, public, busca plantilla por nombre exacto
- **findByIsPublicTrue()**: sin argumentos, List<Template>, public, obtiene todas las plantillas públicas
- **findByIsPublicTrueOrOwner(User owner)**: User owner, List<Template>, public, obtiene plantillas públicas o propiedad del usuario especificado

URL DIAGRAMA DE CLASES

FRONTEND

1. Nombre de la clase: NewTemplateWidget

- **Descripción:** Widget de interfaz de usuario encargado de mostrar el formulario para la creación de una nueva plantilla. Permite ingresar nombre, contenido HTML y opciones de visibilidad (pública/privada) según el rol del usuario.
- **Dependencias con otras clases:**
  - NewTemplateViewModel: (Provider) Gestiona el estado de los campos de texto y la lógica de negocio para guardar la plantilla.
  - JwtKey: Utilizado para verificar el rol del usuario ('ADMIN' o 'CREADOR') y mostrar opciones adicionales.
  - ScaffoldMessenger: Utilizado para mostrar retroalimentación (SnackBar) al usuario sobre el éxito o error de la operación.
- **Atributos:**
  - key: Key?, Pública, null, Identificador del widget.
- **Funciones:**
  - build:
    - **Argumentos:** BuildContext context.
    - **Valor de retorno:** Widget.
    - **Visibilidad:** Pública (Override).
    - **Descripción:** Construye la interfaz gráfica que incluye los campos de texto para nombre y contenido, y el botón de guardar. Escucha los cambios del ViewModel.

2. **Nombre de la clase: MyTemplatesView**

- **Descripción:** Widget con estado (StatefulWidget) que presenta la vista de gestión de las plantillas privadas del usuario. Muestra una lista de plantillas y permite seleccionarlas para edición o eliminación.
- **Dependencias con otras clases:**
    - MyTemplatesViewModel: (Provider) Provee la lista de plantillas, gestiona la carga de datos y las operaciones CRUD.
    - _MyTemplatesViewState: Clase privada que maneja el ciclo de vida del widget y la interacción con el ViewModel.
- **Atributos:**
    - key: Key?, Pública, null, Identificador del widget.
- **Funciones:**
    - createState:
        - **Argumentos:** Ninguno.
        - **Valor de retorno:** State<MyTemplatesView>.
        - **Visibilidad:** Pública (Override).
        - **Descripción:** Crea la instancia del estado mutable _MyTemplatesViewState.

3. Nombre de la clase: LoginViewModel

- **Descripción:** Gestiona la lógica de negocio y el estado para la autenticación de usuarios (inicio de sesión y registro). Maneja los controladores de texto y validaciones de formularios.
- **Dependencias con otras clases:**
    - User: Modelo de datos utilizado para ejecutar las peticiones de login y registro.
    - EmailValidator: Utilería para validar el formato del correo electrónico.
- **Atributos:**
    - _inRegistrationState: bool, Privada, false, Indica si se está mostrando la vista de registro.
    - _formKey: GlobalKey<FormState>, Privada, GlobalKey(), Llave para identificar y validar el formulario.
    - _emailController: TextEditingController, Privada, Instancia nueva, Controlador del campo email.
    - _nameController: TextEditingController, Privada, Instancia nueva, Controlador del campo nombre.

- _passwordController: TextEditingController, Privada, Instancia nueva, Controlador del campo contraseña.
- _passConfirmController: TextEditingController, Privada, Instancia nueva, Controlador para confirmar contraseña.
- _isCreatorSelected: bool, Privada, false, Indica si el usuario seleccionó el rol de "Creador".

- **Funciones:**
  - login:
    - **Argumentos:** Ninguno.
    - **Valor de retorno:** Future<bool>.
    - **Visibilidad:** Pública.
    - **Descripción:** Valida el formulario e intenta iniciar sesión usando el modelo User.
  - createAccount:
    - **Argumentos:** Ninguno.
    - **Valor de retorno:** Future<bool>.
    - **Visibilidad:** Pública.
    - **Descripción:** Valida el formulario, configura el rol y registra un nuevo usuario.
  - validateEmail:
    - **Argumentos:** String? value.
    - **Valor de retorno:** bool.
    - **Visibilidad:** Pública.
    - **Descripción:** Verifica si el texto ingresado es un email válido.
  - validateText:
    - **Argumentos:** String? value.
    - **Valor de retorno:** bool.
    - **Visibilidad:** Pública.
    - **Descripción:** Verifica que el campo no sea nulo o vacío.
  - confirmPassword:
    - **Argumentos:** String? value.
    - **Valor de retorno:** bool.
    - **Visibilidad:** Pública.
    - **Descripción:** Verifica si la confirmación coincide con la contraseña original.
  - isInRegistrationState:
    - **Argumentos:** Ninguno.
    - **Valor de retorno:** bool.
    - **Visibilidad:** Pública.
    - **Descripción:** Getter para el estado de registro.
  - setRegistrationState:

        - **Argumentos:** bool state.
        - **Valor de retorno:** void.
        - **Visibilidad:** Pública.
        - **Descripción:** Cambia entre login/registro y limpia los formularios.

4. Nombre de la clase: RegisterWidget

- **Descripción:** Componente visual que renderiza el formulario de registro de usuario dentro de la vista de Login.
- **Dependencias con otras clases:**
    - LoginViewModel: (Provider) Provee los controladores de texto y funciones de validación.
- **Atributos:**
    - key: Key?, Pública, null, Identificador del widget.
- **Funciones:**
    - build:
        - **Argumentos:** BuildContext context.
        - **Valor de retorno:** Widget.
        - **Visibilidad:** Pública (Override).
        - **Descripción:** Construye el formulario con campos de Nombre, Correo, Contraseña, Confirmación y Dropdown de Rol.

5. Nombre de la clase: MyTemplatesViewModel

- **Descripción:** ViewModel encargado de la lógica de negocio para la gestión de "Mis Plantillas". Maneja la obtención, actualización y eliminación de plantillas privadas.
- **Dependencias con otras clases:**
    - TemplateList: Servicio de red/repositorio para obtener la lista de plantillas.
    - Template: Modelo de datos que representa una plantilla.
- **Atributos:**
    - _templateList: TemplateList, Privada, Instancia nueva, Servicio de plantillas.
    - _privateTemplates: List<Template>, Privada, [], Lista de plantillas filtradas (privadas).

- _selectedTemplate: Template?, Privada, null, Plantilla actualmente seleccionada para editar.
- _isLoading: bool, Privada, false, Estado de carga.
- _errorMessage: String, Privada, '', Mensaje de error para la UI.
- titleController: TextEditingController, Pública, Instancia nueva, Controlador para editar el título.
- contentController: TextEditingController, Pública, Instancia nueva, Controlador para editar el contenido.

- **Funciones:**
  - loadPrivateTemplates:
    - **Argumentos:** Ninguno.
    - **Valor de retorno:** Future<void>.
    - **Visibilidad:** Pública.
    - **Descripción:** Obtiene todas las plantillas del servicio y filtra las que no son públicas.
  - selectTemplate:
    - **Argumentos:** Template template.
    - **Valor de retorno:** void.
    - **Visibilidad:** Pública.
    - **Descripción:** Establece la plantilla seleccionada y llena los controladores de texto con sus datos.
  - clearSelection:
    - **Argumentos:** Ninguno.
    - **Valor de retorno:** void.
    - **Visibilidad:** Pública.
    - **Descripción:** Deselecciona la plantilla actual y limpia los controladores.
  - updateSelectedTemplate:
    - **Argumentos:** Ninguno.
    - **Valor de retorno:** Future<bool>.
    - **Visibilidad:** Pública.
    - **Descripción:** Valida y envía la actualización de la plantilla seleccionada al servicio.
  - deleteTemplate:
    - **Argumentos:** Template template.
    - **Valor de retorno:** Future<bool>.
    - **Visibilidad:** Pública.
    - **Descripción:** Solicita la eliminación de una plantilla específica.

6. Nombre de la clase: DashboardViewModel

- **Descripción:** Gestiona el estado global del Dashboard, controlando qué subvista se muestra actualmente (Lista, Crear o Editar).
- **Dependencias con otras clases:**
  - JwtKey: Utilizado para limpiar la sesión al cerrar sesión.
  - DashboardState: (Enum) Define los estados posibles de la vista.
- **Atributos:**
  - _dashState: DashboardState, Privada, DashboardState.lista, Estado actual de la vista del dashboard.
- **Funciones:**
  - getDashboardState:
    - **Argumentos:** Ninguno.
    - **Valor de retorno:** DashboardState.
    - **Visibilidad:** Pública.
    - **Descripción:** Retorna el estado actual de navegación.
  - setDashboardState:
    - **Argumentos:** DashboardState dashState.
    - **Valor de retorno:** void.
    - **Visibilidad:** Pública.
    - **Descripción:** Actualiza el estado de la vista y notifica a los oyentes.
  - logout:
    - **Argumentos:** Ninguno.
    - **Valor de retorno:** void.
    - **Visibilidad:** Pública.
    - **Descripción:** Reinicia el estado a 'lista' y limpia las credenciales JWT.

7. Nombre de la clase: Loginview

- **Descripción:** Vista principal (Scaffold) que contiene la lógica de presentación para el inicio de sesión. Actúa como contenedor para RegisterWidget o LoginWidget dependiendo del estado.
- **Dependencias con otras clases:**
  - LoginViewModel: Determina qué widget hijo mostrar.
  - RegisterWidget: Widget hijo mostrado en estado de registro.
  - LoginWidget: Widget hijo mostrado en estado de login.
- **Atributos:**
  - key: Key?, Pública, null, Identificador.

- **Funciones:**
    - build:
        - **Argumentos:** BuildContext context.
        - **Valor de retorno:** Widget.
        - **Visibilidad:** Pública (Override).
        - **Descripción:** Renderiza el fondo y la tarjeta central que alterna entre login y registro.

8. Nombre de la clase: LoginWidget

- **Descripción:** Componente visual que muestra el formulario de inicio de sesión (email y contraseña) y el logo de la aplicación.
- **Dependencias con otras clases:**
    - LoginViewModel: Gestiona la validación y la acción de login.
- **Atributos:**
    - key: Key?, Pública, null, Identificador.
- **Funciones:**
    - build:
        - **Argumentos:** BuildContext context.
        - **Valor de retorno:** Widget.
        - **Visibilidad:** Pública (Override).
        - **Descripción:** Construye la UI con campos de texto y botones para iniciar sesión o cambiar a registro.

9. Nombre de la clase: TemplateListWidget

- **Descripción:** Widget que compone la vista principal "Lista de Plantillas" en el dashboard. Organiza el carrusel de plantillas, la lista de campos y la vista previa del PDF.
- **Dependencias con otras clases:**
    - TemplateListViewModel: Provee los datos para los widgets hijos.
    - TemplatesCarousel: Widget hijo (carrusel).
    - FieldsList: Widget hijo (lista de campos).
    - PdfViewer: Widget hijo (visor PDF).
- **Atributos:**
    - key: Key?, Pública, null, Identificador.
- **Funciones:**
    - build:
        - **Argumentos:** BuildContext context.

        - **Valor de retorno:** Widget.
        - **Visibilidad:** Pública (Override).
        - **Descripción:** Estructura el layout principal en filas y columnas para mostrar las herramientas de generación de documentos.

10. Nombre de la clase: DashboardView

- **Descripción:** Vista principal de la aplicación postlogin. Configura la barra de navegación superior (AppBar) y gestiona el cambio dinámico del cuerpo de la página según la selección del menú.
- **Dependencias con otras clases:**
    - DashboardViewModel: Controla qué vista se renderiza en el cuerpo.
    - JwtKey: Verifica si existe una sesión activa al iniciar.
    - TemplateListViewModel, NewTemplateViewModel, MyTemplatesViewModel: ViewModels inyectados vía Provider para las subvistas.
    - TemplateListWidget, NewTemplateWidget, MyTemplatesView: Subvistas posibles.
- **Atributos:**
    - key: Key?, Pública, null, Identificador.
- **Funciones:**
    - build:
        - **Argumentos:** BuildContext context.
        - **Valor de retorno:** Widget.
        - **Visibilidad:** Pública (Override).
        - **Descripción:** Verifica el JWT, construye el AppBar con el menú de navegación y utiliza un switch para renderizar el widget correspondiente al estado del dashboard, inyectando el Provider necesario para cada caso.

11. Nombre de la clase: NewTemplateViewModel

- **Descripción:** ViewModel que gestiona el estado y la lógica para la creación de nuevas plantillas. Controla los campos de entrada y la comunicación con el repositorio de plantillas.
- **Dependencias con otras clases:**
    - TemplateList: Instancia del repositorio utilizada para enviar la petición de creación.

    - TextEditingController: Controladores para manejar la entrada de texto del usuario.
- **Atributos:**
    - templates: TemplateList, Pública, TemplateList(), Repositorio de plantillas.
    - _nameController: TextEditingController, Privada, Instancia nueva, Controlador del nombre.
    - _contentController: TextEditingController, Privada, Instancia nueva, Controlador del contenido HTML.
    - _isPublic: bool, Privada, false, Estado del checkbox de visibilidad.
- **Funciones:**
    - createTemplate:
        - **Argumentos:** Ninguno.
        - **Valor de retorno:** Future<bool>.
        - **Visibilidad:** Pública.
        - **Descripción:** Valida que los campos no estén vacíos y llama al repositorio para crear la plantilla.
    - setCreateAsPublic:
        - **Argumentos:** bool value.
        - **Valor de retorno:** void.
        - **Visibilidad:** Pública.
        - **Descripción:** Actualiza el estado de visibilidad pública/privada.
    - clear:
        - **Argumentos:** Ninguno.
        - **Valor de retorno:** void.
        - **Visibilidad:** Pública.
        - **Descripción:** Limpia los campos de texto.

12. Nombre de la clase: TemplateList

- **Descripción:** Clase repositorio encargada de la gestión de datos de las plantillas. Centraliza las llamadas a la API (CRUD y generación) y mantiene una lista local en caché de las plantillas.
- **Dependencias con otras clases:**
    - TemplateRequests: Capa de red para realizar peticiones HTTP.
    - Template: Modelo de datos para mapear las respuestas.
    - GenerationRequest: Modelo para empaquetar datos de generación.
- **Atributos:**

- _requests: TemplateRequests, Privada, TemplateRequests(), Instancia de la capa de red.
- _templateList: List<Template>, Privada, [], Lista caché de plantillas.

- **Funciones:**
  - getTemplateList:
    - **Argumentos:** Ninguno.
    - **Valor de retorno:** Future<List<Template>>.
    - **Visibilidad:** Pública.
    - **Descripción:** Retorna la lista de plantillas; si está vacía, solicita una actualización al servidor.
  - refreshTemplates:
    - **Argumentos:** Ninguno.
    - **Valor de retorno:** Future<List<Template>>.
    - **Visibilidad:** Pública.
    - **Descripción:** Fuerza una petición al servidor para actualizar la lista local.
  - getTemplateById:
    - **Argumentos:** int id.
    - **Valor de retorno:** Future<Template?>.
    - **Visibilidad:** Pública.
    - **Descripción:** Obtiene una plantilla específica por su ID.
  - createTemplate:
    - **Argumentos:** String name, String content, bool isPublic.
    - **Valor de retorno:** Future<bool>.
    - **Visibilidad:** Pública.
    - **Descripción:** Envía una solicitud para crear una nueva plantilla y refresca la lista local.
  - updateTemplate:
    - **Argumentos:** Template template, String name, String content, bool isPublic.
    - **Valor de retorno:** Future<bool>.
    - **Visibilidad:** Pública.
    - **Descripción:** Actualiza una plantilla existente y refresca la lista.
  - deleteTemplate:
    - **Argumentos:** Template template.
    - **Valor de retorno:** Future<bool>.
    - **Visibilidad:** Pública.
    - **Descripción:** Elimina una plantilla (si no es pública) y refresca la lista.
  - generateDocument:

        - **Argumentos:** String templateType, Map<String, Object> data.
        - **Valor de retorno:** Future<Uint8List?>.
        - **Visibilidad:** Pública.
        - **Descripción:** Solicita la generación del PDF enviando los datos dinámicos.

13. Nombre de la clase: TemplateListViewModel

- **Descripción:** ViewModel principal del Dashboard. Orquesta la interacción entre la lista de plantillas, la generación dinámica de campos de formulario (TemplateFieldsList) y la visualización del PDF (PdfDoc).
- **Dependencias con otras clases:**
    - TemplateList: Repositorio de datos.
    - PdfDoc: Gestión del documento PDF generado.
    - TemplateFieldsList: Gestión de los controladores de texto dinámicos.
    - Template: Modelo de la plantilla seleccionada.
- **Atributos:**
    - _templateList: TemplateList, Privada, Instancia nueva, Repositorio.
    - _pdfDoc: PdfDoc, Privada, Instancia nueva, Gestor de PDF.
    - _templateFields: TemplateFieldsList, Privada, Instancia nueva, Gestor de campos.
    - _selectedTemplate: Template?, Privada, null, Plantilla en uso.
    - _isLoading: bool, Privada, false, Estado de carga.
- **Funciones:**
    - selectTemplate:
        - **Argumentos:** Template template.
        - **Valor de retorno:** void.
        - **Visibilidad:** Pública.
        - **Descripción:** Selecciona una plantilla, extrae sus campos y genera los controladores necesarios en _templateFields.
    - generateDocument:
        - **Argumentos:** Ninguno.
        - **Valor de retorno:** Future<void>.
        - **Visibilidad:** Pública.
        - **Descripción:** Recolecta los datos de los campos, solicita la generación del PDF al repositorio y actualiza _pdfDoc.
    - downloadPdf:
        - **Argumentos:** Ninguno.

        - **Valor de retorno:** Future<void>.
        - **Visibilidad:** Pública.
        - **Descripción:** Genera el documento y activa la descarga del archivo.
    - deleteTemplate:
        - **Argumentos:** Template template.
        - **Valor de retorno:** Future<bool>.
        - **Visibilidad:** Pública.
        - **Descripción:** Elimina la plantilla y limpia la selección.

14. Nombre de la clase: User

- **Descripción:** Modelo de dominio que representa a un usuario del sistema. Incluye métodos auxiliares para ejecutar login y registro asociados a la instancia del usuario.
- **Dependencias con otras clases:**
    - UserRequests: Capa de red para autenticación.
- **Atributos:**
    - id: int?, Pública, null, Identificador único.
    - name: String?, Pública, null, Nombre completo.
    - email: String, Pública, (Requerido), Correo electrónico.
    - role: String?, Pública, null, Rol del usuario (ej. CREADOR).
- **Funciones:**
    - login:
        - **Argumentos:** String password.
        - **Valor de retorno:** Future<bool>.
        - **Visibilidad:** Pública.
        - **Descripción:** Ejecuta la petición de inicio de sesión usando el email del objeto.
    - register:
        - **Argumentos:** String password.
        - **Valor de retorno:** Future<bool>.
        - **Visibilidad:** Pública.
        - **Descripción:** Ejecuta la petición de registro enviando el objeto User actual.
    - fromJson:
        - **Argumentos:** Map<String, dynamic> json.
        - **Valor de retorno:** User.
        - **Visibilidad:** Pública (Factory).
        - **Descripción:** Crea una instancia desde un mapa JSON.

- toJson:
  - **Argumentos:** Ninguno.
  - **Valor de retorno:** Map<String, dynamic>.
  - **Visibilidad:** Pública.
  - **Descripción:** Serializa el objeto a JSON.

15. Nombre de la clase: TemplateFieldsList

- **Descripción:** Clase lógica (Helper) que administra dinámicamente los TextEditingController necesarios para una plantilla. Distingue entre campos simples y campos de bucle (tablas dinámicas).
- **Dependencias con otras clases:**
  - TemplateField: Define la estructura de cada campo.
  - TextEditingController: Controladores de UI.
- **Atributos:**
  - _simpleControllers: Map<String, TextEditingController>, Privada, {}, Mapa de controladores simples.
  - _loopControllers: Map<String, List<Map<String, TextEditingController>>>, Privada, {}, Mapa complejo para filas dinámicas.
  - _fields: List<TemplateField>, Privada, [], Lista de definiciones de campos.
- **Funciones:**
  - generateFields:
    - **Argumentos:** List<String> rawFields.
    - **Valor de retorno:** void.
    - **Visibilidad:** Pública.
    - **Descripción:** Analiza una lista de strings (ej. placeholders) y crea la estructura de campos y controladores correspondientes.
  - addDynamicRow:
    - **Argumentos:** String sectionName, List<String> childFields.
    - **Valor de retorno:** void.
    - **Visibilidad:** Pública.
    - **Descripción:** Añade un nuevo conjunto de controladores para una fila en una sección dinámica.
  - getFieldsAsMap:
    - **Argumentos:** Ninguno.
    - **Valor de retorno:** Map<String, Object>.
    - **Visibilidad:** Pública.

        - **Descripción:** Serializa el contenido de todos los controladores en un mapa estructurado para enviar al backend.
    - clear:
        - **Argumentos:** Ninguno.
        - **Valor de retorno:** void.
        - **Visibilidad:** Pública.
        - **Descripción:** Reinicia todos los mapas y listas.

16. Nombre de la clase: Template

- **Descripción:** Modelo de dominio que representa la estructura completa de una plantilla de documento.
- **Dependencias con otras clases:**
    - User: Propietario de la plantilla.
    - TemplateField: Campos/Placeholders contenidos en la plantilla.
- **Atributos:**
    - _id: int?, Privada, null, ID de base de datos.
    - _name: String, Privada, (Requerido), Nombre de la plantilla.
    - _content: String, Privada, (Requerido), Contenido (HTML/Texto).
    - _isPublic: bool, Privada, false, Visibilidad.
    - _owner: User?, Privada, null, Usuario creador.
    - _fields: List<TemplateField>, Privada, [], Lista de campos parseados.
- **Funciones:**
    - fromJson:
        - **Argumentos:** Map<String, dynamic> json.
        - **Valor de retorno:** Template.
        - **Visibilidad:** Pública (Factory).
        - **Descripción:** Crea una instancia, manejando claves variables (isPublic/public) y parseando la lista de placeholders.
    - toJson:
        - **Argumentos:** Ninguno.
        - **Valor de retorno:** Map<String, dynamic>.
        - **Visibilidad:** Pública.
        - **Descripción:** Serializa el objeto.

17. Nombre de la clase: GenerationRequest

- **Descripción:** DTO (Data Transfer Object) utilizado exclusivamente para estructurar el cuerpo de la petición de generación de documentos.
- **Dependencias con otras clases:** Ninguna.
- **Atributos:**
    - templateType: String, Pública, (Requerido), Nombre/Tipo de la plantilla.
    - data: Map<String, Object>, Pública, (Requerido), Datos para llenar la plantilla.
- **Funciones:**
    - toJson:
        - **Argumentos:** Ninguno.
        - **Valor de retorno:** Map<String, dynamic>.
        - **Visibilidad:** Pública.
        - **Descripción:** Convierte el objeto a un mapa JSON.

18. Nombre de la clase: TemplateField

- **Descripción:** Modelo que representa un campo individual dentro de una plantilla. Puede ser un campo simple o contener hijos si es dinámico.
- **Dependencias con otras clases:** Ninguna.
- **Atributos:**
    - _name: String, Privada, (Requerido), Nombre del campo.
    - isDynamic: bool, Pública, false, Indica si es una sección repetitiva.
    - _children: List<String>, Privada, [], Subcampos si es dinámico.
- **Funciones:**
    - getChildren:
        - **Argumentos:** Ninguno.
        - **Valor de retorno:** List<String>.
        - **Visibilidad:** Pública.
        - **Descripción:** Getter para la lista de hijos.
    - getName:
        - **Argumentos:** Ninguno.
        - **Valor de retorno:** String.
        - **Visibilidad:** Pública.
        - **Descripción:** Getter para el nombre.

19. Nombre de la clase: PdfDoc

- **Descripción:** Clase de utilidad para gestionar el ciclo de vida de un documento PDF en memoria, su visualización mediante un controlador y su descarga.
- **Dependencias con otras clases:**
  - PdfController: (Librería pdfx) Controla la renderización del PDF.
  - FileSaver: (Librería file_saver) Gestiona la descarga de archivos.
- **Atributos:**
  - _pdfController: PdfController?, Privada, null, Controlador del visor PDF.
  - _pdfBytes: Uint8List?, Privada, null, Datos binarios del archivo.
- **Funciones:**
  - getPdfController:
    - **Argumentos:** Ninguno.
    - **Valor de retorno:** PdfController?.
    - **Visibilidad:** Pública.
    - **Descripción:** Retorna el controlador actual.
  - setPdf:
    - **Argumentos:** Uint8List bytes.
    - **Valor de retorno:** void.
    - **Visibilidad:** Pública.
    - **Descripción:** Guarda los bytes y crea una nueva instancia de PdfController.
  - downloadPdf:
    - **Argumentos:** String templateName.
    - **Valor de retorno:** Future<void>.
    - **Visibilidad:** Pública.
    - **Descripción:** Guarda el archivo PDF en el dispositivo local con un nombre basado en la plantilla y la hora actual.
  - clear:
    - **Argumentos:** Ninguno.
    - **Valor de retorno:** void.
    - **Visibilidad:** Pública.
    - **Descripción:** Limpia el controlador y los bytes almacenados.

20. Nombre de la clase: HttpRequest

- **Descripción:** Clase abstracta base que encapsula la lógica para realizar peticiones HTTP a la API. Maneja la configuración de encabezados, la

gestión de tokens de autenticación (JWT) y el procesamiento estandarizado de respuestas y errores.

- **Dependencias con otras clases:**
  - JwtKey: Utilizado para obtener el token de sesión e inyectarlo en los encabezados Authorization.
  - http: (Librería externa) Motor para realizar las llamadas de red.
- **Atributos:**
  - baseUrl: String, Pública (final), 'https://dynadocs.onrender.com/api/', URL base del servidor.
- **Funciones:**
  - _getHeaders:
    - **Argumentos:** bool isAuth.
    - **Valor de retorno:** Future<Map<String, String>>.
    - **Visibilidad:** Privada.
    - **Descripción:** Construye los encabezados HTTP, añadiendo el token Bearer si isAuth es verdadero.
  - sendGetRequest:
    - **Argumentos:** String endpoint, bool auth (opcional).
    - **Valor de retorno:** Future<dynamic>.
    - **Visibilidad:** Pública.
    - **Descripción:** Ejecuta una petición GET y procesa la respuesta JSON.
  - sendPostRequest:
    - **Argumentos:** String endpoint, dynamic body, bool auth (opcional).
    - **Valor de retorno:** Future<dynamic>.
    - **Visibilidad:** Pública.
    - **Descripción:** Ejecuta una petición POST enviando un cuerpo JSON.
  - sendPostRequestBytes:
    - **Argumentos:** String endpoint, dynamic body, bool auth (opcional).
    - **Valor de retorno:** Future<List<int>>.
    - **Visibilidad:** Pública.
    - **Descripción:** Ejecuta un POST específico para respuestas binarias (ej. descarga de PDF) retornando los bytes crudos.
  - sendPutRequest:
    - **Argumentos:** String endpoint, dynamic body, bool auth (opcional).
    - **Valor de retorno:** Future<dynamic>.
    - **Visibilidad:** Pública.

        - **Descripción:** Ejecuta una petición PUT para actualizaciones.
    - sendDeleteRequest:
        - **Argumentos:** String endpoint, bool auth (opcional).
        - **Valor de retorno:** Future<dynamic>.
        - **Visibilidad:** Pública.
        - **Descripción:** Ejecuta una petición DELETE.
    - _processResponse:
        - **Argumentos:** http.Response response.
        - **Valor de retorno:** dynamic.
        - **Visibilidad:** Privada.
        - **Descripción:** Evalúa el código de estado; decodifica el JSON si es exitoso (200299) o lanza una excepción si falla.

21. Nombre de la clase: UserRequests

- **Descripción:** Subclase concreta de HttpRequest encargada de las operaciones de red relacionadas con la autenticación de usuarios (Login y Registro).
- **Dependencias con otras clases:**
    - HttpRequest: (Herencia) Provee los métodos de envío de datos.
    - JwtKey: Almacena el token y el rol recibidos tras un login exitoso.
    - User: Modelo utilizado para enviar los datos de registro.
- **Atributos:**
    - (Hereda baseUrl de HttpRequest).
- **Funciones:**
    - login:
        - **Argumentos:** String email, String password.
        - **Valor de retorno:** Future<bool>.
        - **Visibilidad:** Pública.
        - **Descripción:** Envía credenciales al endpoint de login. Si es exitoso, guarda el token y rol en JwtKey.
    - register:
        - **Argumentos:** User user, String password.
        - **Valor de retorno:** Future<bool>.
        - **Visibilidad:** Pública.
        - **Descripción:** Serializa el objeto User, añade la contraseña al cuerpo y envía la solicitud de registro.

22. Nombre de la clase: TemplateRequests

- **Descripción:** Subclase concreta de HttpRequest que maneja todas las operaciones CRUD (Crear, Leer, Actualizar, Borrar) y de generación de documentos para las plantillas.
- **Dependencias con otras clases:**
  - HttpRequest: (Herencia).
  - Template: Modelo para mapear las respuestas de la API.
  - GenerationRequest: Modelo para estructurar la petición de generación de PDF.
- **Atributos:**
  - (Hereda baseUrl de HttpRequest).
- **Funciones:**
  - getTemplateList:
    - **Argumentos:** Ninguno.
    - **Valor de retorno:** Future<List<Template>>.
    - **Visibilidad:** Pública.
    - **Descripción:** Obtiene la lista de plantillas y las convierte en objetos Template.
  - getTemplateById:
    - **Argumentos:** int id.
    - **Valor de retorno:** Future<Template>.
    - **Visibilidad:** Pública.
    - **Descripción:** Obtiene el detalle de una plantilla específica.
  - createTemplate:
    - **Argumentos:** String name, String content, bool isPublic.
    - **Valor de retorno:** Future<Template>.
    - **Visibilidad:** Pública.
    - **Descripción:** Envía los datos para crear una nueva plantilla.
  - generateDocument:
    - **Argumentos:** GenerationRequest request.
    - **Valor de retorno:** Future<Uint8List?>.
    - **Visibilidad:** Pública.
    - **Descripción:** Utiliza sendPostRequestBytes para enviar datos y recibir el archivo PDF generado en bytes.
  - deleteTemplate:
    - **Argumentos:** int id.
    - **Valor de retorno:** Future<bool>.
    - **Visibilidad:** Pública.
    - **Descripción:** Solicita la eliminación de una plantilla por ID.
  - updateTemplate:

        - **Argumentos:** int id, String name, String content, bool isPublic.
        - **Valor de retorno:** Future<bool>.
        - **Visibilidad:** Pública.
        - **Descripción:** Envía los datos modificados para actualizar una plantilla existente.

23. Nombre de la clase: JwtKey

- **Descripción:** Clase Singleton encargada de la persistencia y gestión segura del token de sesión (JWT) y el rol del usuario en el almacenamiento local del dispositivo.
- **Dependencias con otras clases:**
    - SharedPreferences: (Librería shared_preferences) Mecanismo de persistencia clavevalor.
- **Atributos:**
    - _instance: JwtKey, Privada (static), Instancia única del Singleton.
    - _token: String?, Privada, null, Token cargado en memoria.
    - _storageKey: String, Privada (static const), 'jwt_token', Clave de almacenamiento.
    - _roleKey: String, Privada (static const), 'user_role', Clave de almacenamiento para el rol.
- **Funciones:**
    - factory JwtKey:
        - **Argumentos:** Ninguno.
        - **Valor de retorno:** JwtKey.
        - **Visibilidad:** Pública.
        - **Descripción:** Constructor de fábrica para retornar la instancia única.
    - loadToken:
        - **Argumentos:** Ninguno.
        - **Valor de retorno:** Future<void>.
        - **Visibilidad:** Pública.
        - **Descripción:** Lee el token desde SharedPreferences y lo carga en la variable _token.
    - setJwtKeyAndRole:
        - **Argumentos:** String token, String role.
        - **Valor de retorno:** Future<void>.
        - **Visibilidad:** Pública.

        - **Descripción:** Guarda el token y el rol tanto en memoria como en almacenamiento persistente.
    - getJwtKey:
        - **Argumentos:** Ninguno.
        - **Valor de retorno:** Future<String?>.
        - **Visibilidad:** Pública.
        - **Descripción:** Refresca y retorna el token almacenado.
    - hasJwtKey:
        - **Argumentos:** Ninguno.
        - **Valor de retorno:** Future<bool>.
        - **Visibilidad:** Pública.
        - **Descripción:** Verifica asíncronamente si existe un token guardado.
    - getUserRole:
        - **Argumentos:** Ninguno.
        - **Valor de retorno:** Future<String?>.
        - **Visibilidad:** Pública.
        - **Descripción:** Retorna el rol del usuario almacenado.
    - hasKey:
        - **Argumentos:** Ninguno.
        - **Valor de retorno:** bool.
        - **Visibilidad:** Pública.
        - **Descripción:** Verificación síncrona rápida sobre la variable en memoria.
    - clear:
        - **Argumentos:** Ninguno.
        - **Valor de retorno:** Future<void>.
        - **Visibilidad:** Pública.
        - **Descripción:** Elimina el token de memoria y del almacenamiento, cerrando efectivamente la sesión local.

URL DIAGRAMA DE CLASE

Descripción de las secuencias

BackEnd

1. Autenticación de Usuario (Login)

Este diagrama ilustra el proceso de inicio de sesión de un usuario existente.

Nombre: Proceso de Login y Generación de JWT.

Descripción: El cliente envía credenciales (email y password). El sistema verifica si el usuario existe en la base de datos. Si existe, compara la contraseña en texto plano con la contraseña hasheada (usando PasswordEncoder).

Flujo Exitoso: Si las credenciales coinciden, JwtUtils genera un token firmado (HS256) y lo devuelve al cliente.

Flujo de Error: Se manejan excepciones si el usuario no existe o la contraseña es incorrecta.

Diagrama UML: Arquitecturas-de-software/Documentacion-proyecto-final/DiagramasDeSecuencia/BackEnd/LoginBackend.png at main · hjanssena/Arquitecturas-de-software

2. Registro de Usuario

Muestra cómo se da de alta un nuevo usuario en el sistema.

Nombre: Registro de Nuevo Usuario.

Descripción: El controlador recibe los datos del nuevo usuario. El servicio verifica primero si el email ya está registrado.

Lógica: Si el email es nuevo, se codifica la contraseña (encode(password)), se asigna un rol por defecto (si es nulo) y se guarda la entidad en la base de datos.

Respuesta: Retorna un estado 200/201 con mensaje de éxito. Si el email existe, lanza una RuntimeException.

Diagrama UML: Arquitecturas-de-software/Documentacion-proyecto-final/DiagramasDeSecuencia/BackEnd/RegistroBackend.png at main · hjanssena/Arquitecturas-de-software

3. Seguridad y Filtrado (Middleware)

Este es un diagrama crítico que muestra cómo se interceptan las peticiones para validar la seguridad antes de llegar a los controladores.

Nombre: Filtro de Seguridad JWT (JwtFilter).

Descripción: Intercepta cualquier petición HTTP protegida.

Extrae el token del header Authorization.

Verifica si el token está en la "Blacklist" (lista negra de tokens cerrados).

Valida la firma y expiración del token con JwtUtils.

Si es válido, carga los detalles del usuario desde la BD y establece la autenticación en el SecurityContext.

Si falla alguna validación, retorna 401 Unauthorized.

Diagrama UML: Arquitecturas-de-software/Documentacion-proyecto-final/DiagramasDeSecuencia/BackEnd/Seguridad(JwtFilter).png at main · hjanssena/Arquitecturas-de-software

4. Cierre de Sesión (Logout)

Detalla cómo se invalida una sesión activa de manera segura.

Nombre: Cierre de Sesión e Invalidación de Token.

Descripción: El usuario envía una petición de logout con su token. El sistema valida que el token sea auténtico y que no esté ya en la lista negra.

Acción: Llama a invalidateToken(jwt), lo que agrega el token a la tabla/cache de TokenBlacklistService. Esto asegura que ese token específico no pueda volver a usarse aunque no haya expirado por tiempo.

Diagrama UML: Arquitecturas-de-software/Documentacion-proyecto-final/DiagramasDeSecuencia/BackEnd/LogoutBackend.png at main · hjanssena/Arquitecturas-de-software

5. Creación de Plantilla (Template)

Muestra la lógica de negocio para guardar nuevas plantillas de documentos.

Nombre: Creación de Nueva Plantilla.

Descripción: Tras validar el token JWT, el sistema determina la visibilidad de la plantilla basándose en el rol del usuario:

Rol CREADOR: Puede establecer la plantilla como isPublic(true).

Rol USUARIO: Se fuerza a isPublic(false).

Procesamiento: Se utiliza TemplateProcessor para extraer automáticamente los "placeholders" (variables) del contenido HTML antes de guardar todo en la base de datos.

Diagrama UML: Arquitecturas-de-software/Documentacion-proyecto-final/DiagramasDeSecuencia/BackEnd/CrearTemplateBackend.png at main · hjanssena/Arquitecturas-de-software

6. Actualización de Plantilla

Describe el flujo para modificar una plantilla existente, incluyendo controles de permisos.

Nombre: Actualización de Plantilla y Verificación de Propiedad.

Descripción: El usuario intenta editar una plantilla por su ID.

Validación de Permisos: El servicio verifica si el usuario actual es el "Owner" (dueño) de la plantilla o si es un ADMIN. Si no lo es, retorna 403 Forbidden.

Lógica: Si tiene permiso, actualiza el nombre y contenido, vuelve a extraer los nuevos placeholders y guarda los cambios en la base de datos.

Diagrama UML: Arquitecturas-de-software/Documentacion-proyecto-final/DiagramasDeSecuencia/BackEnd/ActualizarTemplateBackend.png at main · hjanssena/Arquitecturas-de-software

7. Listado de Plantillas

Muestra cómo se recuperan las plantillas aplicando filtros según el usuario.

Nombre: Listado y Filtrado de Plantillas por Rol.

Descripción: Al solicitar la lista de plantillas, el TemplateService decide qué datos retornar basándose en quién pregunta:

ADMIN: Ejecuta findAll() (ve todo).

CREADOR: Ve las públicas (findByIsPublicTrue).

USUARIO: Ve las públicas Y las que él mismo creó (findByIsPublicTrueOrOwner).

Diagrama UML: Arquitecturas-de-software/Documentacion-proyecto-final/DiagramasDeSecuencia/BackEnd/ListarTemplatesBackend.png at main · hjanssena/Arquitecturas-de-software

8. Generación de PDF

El flujo final donde se utiliza una plantilla para crear un documento binario.

Nombre: Motor de Generación de PDF.

Descripción: El cliente solicita generar un PDF indicando el tipo de plantilla y los datos.

Carga la plantilla desde el repositorio.

Valida que los datos enviados coincidan con los placeholders requeridos.

Reemplaza las variables en el HTML (TemplateProcessor).

Convierte el HTML procesado a PDF usando ITextRenderer.

Retorna el archivo binario (byte[]) con los headers application/pdf.

Diagrama UML: Arquitecturas-de-software/Documentacion-proyecto-final/DiagramasDeSecuencia/BackEnd/GenerarPDFBackend.png at main · hjanssena/Arquitecturas-de-software

FrontEnd

1. Inicialización de la Aplicación

Este diagrama describe el proceso de arranque ("bootstrapping") de la aplicación móvil.

Nombre: Inicialización y Carga de Dependencias.

Descripción: El punto de entrada main.dart comienza la ejecución.

Intenta cargar un token JWT preexistente desde el almacenamiento local (JwtKey).

Ejecuta la aplicación (runApp).

Inicializa el gestor de estado (Provider), creando los ViewModels necesarios (LoginViewModel, DashboardViewModel).

Configura el sistema de rutas y establece la vista inicial (por defecto LoginView).

Diagrama UML: Arquitecturas-de-software/Documentacion-proyecto-final/DiagramasDeSecuencia/FrontEnd/InicializaciónFront.png at main · hjanssena/Arquitecturas-de-software

2. Autenticación (Login y Registro)

Detalla cómo el frontend maneja la entrada del usuario para iniciar sesión o crear una cuenta nueva.

Nombre: Flujo de Login y Registro de Usuario.

Descripción: El usuario interactúa con la LoginView. El LoginViewModel gestiona la lógica:

Login: Envía credenciales a la API (POST /login). Si es exitoso, guarda el token recibido y el rol en el singleton JwtKey (almacenamiento local) y navega al Dashboard.

Registro: Envía datos de nuevo usuario (POST /register). Si es exitoso, notifica a la vista para cambiar el estado visual a "modo Login" para que el usuario pueda ingresar.

Diagrama UML: Arquitecturas-de-software/Documentacion-proyecto-final/DiagramasDeSecuencia/FrontEnd/Autenticación(LoginRegistro).png at main · hjanssena/Arquitecturas-de-software

3. Gestión de Plantillas (Crear/Editar)

Muestra la lógica reutilizable para la gestión de plantillas, diferenciando entre crear una nueva o actualizar una existente.

Nombre: Creación y Edición de Plantillas.

Descripción: Desde la NewTemplateView, el usuario guarda los cambios. El NewTemplateViewModel decide la operación:

Crear: Si es nueva, llama a createTemplate que envía un POST a la API.

Editar: Si ya existe (tiene ID), llama a updateTemplate que envía un PUT.

Común: En ambos casos, se obtiene el token de JwtKey para autorizar la petición y, tras el éxito, se navega de regreso al Dashboard.

Diagrama UML: Arquitecturas-de-software/Documentacion-proyecto-final/DiagramasDeSecuencia/FrontEnd/(CrearEditar)Template.png at main · hjanssena/Arquitecturas-de-software

4. Listado de Plantillas

Describe cómo el frontend recupera y muestra la información segura desde el backend.

Nombre: Obtención y Renderizado de Lista de Plantillas.

Descripción: Al acceder al DashboardView, se solicita la lista de plantillas.

El TemplateListViewModel invoca a la capa de peticiones (TemplateRequests).

Se obtiene el token actual (JwtKey) y se adjunta al header.

Se realiza el GET /api/templates.

La respuesta JSON se mapea a una lista de objetos Template.

La vista renderiza los elementos (ej. en un carrusel).

Diagrama UML: Arquitecturas-de-software/Documentacion-proyecto-final/DiagramasDeSecuencia/FrontEnd/ListarTemplates.png at main · hjanssena/Arquitecturas-de-software

5. Generación de PDF

Ilustra el flujo de interacción para convertir una plantilla y datos dinámicos en un documento visualizable.

Nombre: Generación, Descarga y Visualización de PDF.

Descripción: El usuario selecciona una plantilla y llena los campos en el DynamicFieldWidget.

Al hacer clic en "Generar PDF", el GenerationViewModel envía la solicitud (POST).

La API backend procesa la plantilla y retorna bytes binarios.

El frontend recibe los datos como Uint8List (lista de enteros sin signo).

Estos bytes se pasan a un PDFWidget para visualizar el documento generado dentro de la app, permitiendo también su descarga.

Diagrama UML: Arquitecturas-de-software/Documentacion-proyecto-final/DiagramasDeSecuencia/FrontEnd/GenerarPDF.png at main · hjanssena/Arquitecturas-de-software

Descripción de caso de uso

URL AL VIDEO DE FUNCIONALIDADES

DESCRIPCIÓN DE ENTREGABLES. CODIGO FUENTE

Link al repositorio donde se puede visualizar el código fuente (Guiarse por el README.MD)

URL DEL REPOSITORIO