

GeekHub

Proyecto Integrado
GeekHub

Indice

1. Introducción	3
2. Plan de Empresa	3
2.1 Justificación	3
2.2 Nombre y Logo	3
2.3 Producto	4
2.4 Modelo de Negocio	4
2.5 Consumidores y Posibles Clientes	4
2.6 Competencia	5
2.7 Análisis DAFO	5
2.8 Publicidad y Promoción	6
3. Descripción detallada del sistema y listado de historias de usuario	7
4. Modelado y diseño	8
4.1 Prototipado en Figma de la aplicación	8
4.2 Diagrama de clases del modelo de dominio de la API.	8
5. Diseño	9
5.1 Diagrama de clases	9
6. Implementación	9

1. Introducción

Este proyecto consiste en una aplicación web desarrollada con SpringBoot y PostgreSQL para el backend, y Angular para el frontend. Está diseñada para conectar personas que comparten intereses comunes dentro de la cultura friki, permitiéndoles crear comunidades temáticas, compartir contenido y socializar en un entorno personalizado.

2. Plan de Empresa

2.1 Justificación

El modelo de negocio de GeekHub nace con el objetivo de cubrir las necesidades específicas de la comunidad friki, un nicho en crecimiento que busca plataformas especializadas para socializar y compartir intereses.

Actualmente, redes como Discord, Reddit o Facebook ofrecen soluciones parciales. Sin embargo, ninguna ofrece una experiencia social completa con funcionalidades centradas exclusivamente en la cultura friki.

GeekHub se diferencia por:

- Interfaz y funcionalidades exclusivas para la cultura friki.
- Algoritmos de conexión por intereses específicos.
- Integración de contenido multimedia.

2.2 Nombre y Logo

Nombre: GeekHub

- Sencillo, fácil de recordar y que transmite la esencia de la aplicación: ser un punto de encuentro para frikis.

Logo:

- Diseño moderno que combine un símbolo geek con elementos de conexión.
- Colores vivos como el morado, dirigidos al público joven y tecnológico.

2.3 Producto

¿Qué se ofrece?

Una aplicación web para:

- Crear perfiles temáticos.
- Compartir contenido multimedia.
- Conectar con personas con intereses comunes.

Interfaz:

- Intuitiva, moderna y atractiva.
- Filtros de búsqueda detallados según tipo de interés.

Finalidad:

- Facilitar la interacción social y fomentar comunidades dentro del nicho friki.

Necesidades que cubre:

- Soledad o aislamiento entre personas con gustos frikis.
- Necesidad de compartir contenido específico.
- Falta de una plataforma integral para esta comunidad.

2.4 Modelo de Negocio

Ingresos mediante:

- Publicidad segmentada: Anuncios sobre productos y eventos frikis.
- Colaboraciones y patrocinios: Con marcas y eventos del sector.

2.5 Consumidores y Posibles Clientes

Tipo de usuarios:

Jóvenes y adultos (15-40 años) aficionados a la cultura friki:

- Gamers, otakus, lectores de cómics, cosplayers, amantes del rol, ciencia ficción, etc.

Características:

- Activos en redes sociales.
- Buscan interacción y contenido especializado.
- Participan en eventos temáticos.

Ámbito geográfico:

- Inicialmente en el mercado hispanohablante.
- Posibilidad de expansión global, especialmente en zonas urbanas con alta presencia de cultura pop.

2.6 Competencia

- Discord: Fuerte en comunidades, pero no es una red social como tal.
- Reddit: Enfocado en foros temáticos, sin interacción social directa.
- Facebook Groups: Demasiado genérico.
- Apps de citas: Limitadas a relaciones románticas, no comunidad general.

2.7 Análisis DAFO

Fortalezas	Debilidades
Especialización en un nicho creciente.	Necesidad de una base amplia de usuarios.
Diseño y funcionalidades únicas.	Dependencia tecnológica y mantenimiento.



Comunidad comprometida.	Dificultad inicial de monetización.
-------------------------	-------------------------------------

Oportunidades	Amenazas
Crecimiento del mercado friki.	Competencia de plataformas consolidadas.
Interés en comunidades temáticas.	Cambios en tendencias culturales.
Posibilidad de internacionalización.	Saturación de apps sociales.

2.8 Publicidad y Promoción

Estrategia de promoción:

- Publicidad en redes sociales: Instagram, TikTok, YouTube.
- Colaboraciones con influencers del ámbito friki.
- Participación en eventos: ferias, convenciones, conferencias.
- Campañas en plataformas de gaming.

Coste:

- Presupuesto inicial moderado aprovechando marketing digital y colaboraciones.
- Posteriormente, mayor inversión en campañas pagadas y eventos propios.

3. Descripción detallada del sistema y listado de historias de usuario

La aplicación está dividida en dos partes: backend y frontend, siguiendo una arquitectura modular y escalable. Las principales funcionalidades del sistema son:

- **Login:** desde la pantalla principal permite iniciar sesión en una cuenta previamente creada añadiendo los credenciales y pulsando el botón de sign in o crear una cuenta en caso de no tenerla, pulsando crear cuenta y siendo redirigido a la pantalla de creación.
- **Registro:** el registro se divide en dos pasos, en el primer paso se introduce datos de la cuenta como nombre de usuario, correo electrónico y contraseña y en el paso dos datos personales del usuario. Una vez completos los dos formularios pasamos a la activación de la cuenta..
- **Activación de cuenta:** en esta pantalla debemos introducir un código que se ha enviado al correo electrónico proporcionado en el paso anterior para realizar la validación de la cuenta. Una vez confirmado el código volvemos al login para iniciar sesión con la cuenta ya creada.
- **Perfil:** al iniciar sesión llegamos al perfil, donde podemos ver nuestros datos y publicaciones, además de poder editarlos.
- **Publicaciones:** desde el menú lateral tenemos la opción de ir a la pantalla de posts, donde tenemos la posibilidad de subir publicaciones con contenido escrito o imágenes. Además de la pantalla propia desde home en la parte superior también podemos crear publicaciones.
- **Home:** en el menú principal aparecen las publicaciones de tus perfiles favoritos
- **Buscador:** buscador y filtros específicos para buscar perfiles por intereses, localización, género o edad.

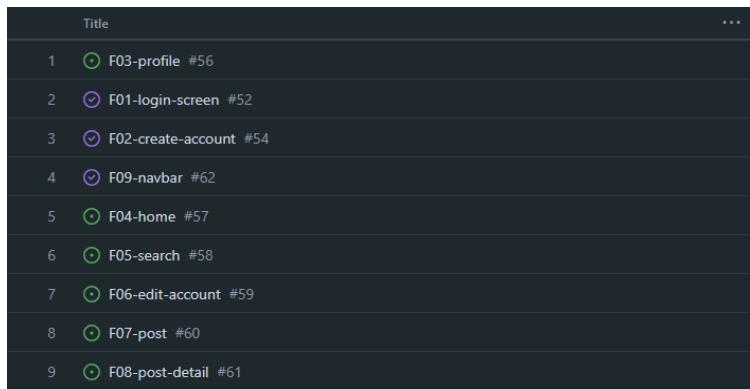
Las historias de usuario están divididas en dos projects de GitHub:

- **Backend:** <https://github.com/users/olgaaw/projects/4/views/1>

Title	...
1 ⚡ HU-6-eliminar-usuario #22	
2 ⚡ HU-18-filtro-multiple #31	
3 ⚡ HU-17-busqueda	
4 ⚡ HU-38-postman #50	
5 ⚡ HU-28-eliminar-interes #47	
6 ⚡ HU-20-eliminar-comentario-admin #45	
7 ⚡ HU-22-remove-like #43	
8 ⚡ HU-24-remove-comment #41	
9 ⚡ HU-10-eliminar-post #39	

Title	...
18 ⚡ HU-12-comment-post #18	
19 ⚡ HU-4-editar-usuario #14	
20 ⚡ HU-34-ver-todos-post-usuario #15	
21 ⚡ HU-1-conexiondb #1	
22 ⚡ HU-2-seguridad #2	
23 ⚡ HU-7-crear-post #7	
24 ⚡ HU-33-validar-crear-usuario #9	
25 ⚡ HU-32-iniciar-sesion #5	
26 ⚡ HU-3-crear-usuario #3	
27 ⚡ HU-10-eliminar-post #39	

- Frontend: <https://github.com/users/olgaaw/projects/7/views/1>

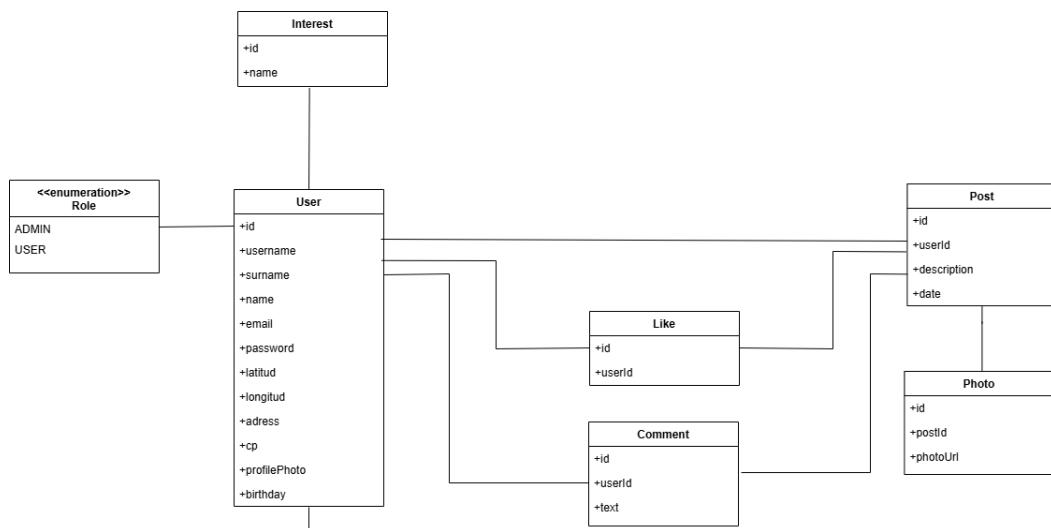


4. Modelado y diseño

4.1 Prototipado en Figma de la aplicación

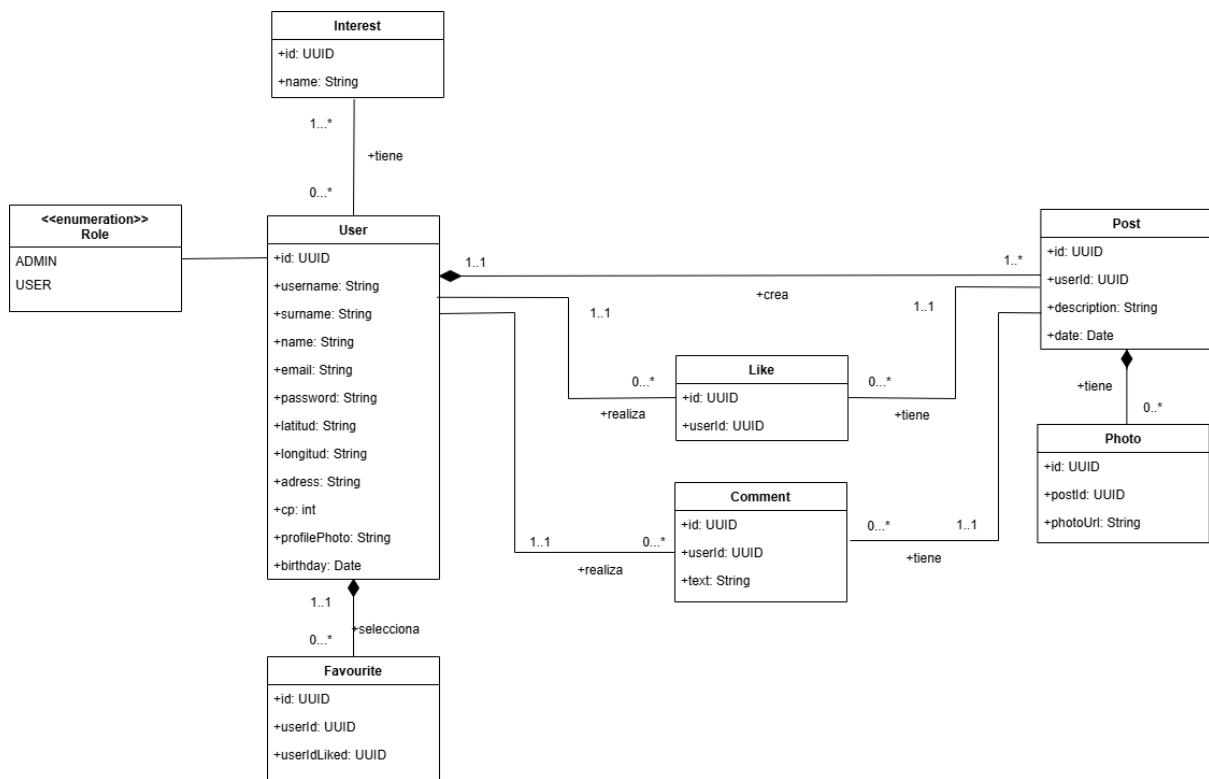
- Diseño web:
<https://www.figma.com/proto/A1Wx1f1BrY8gbem5pOfPvB/GeekHub-web?node-id=0-1&t=1GuLzO5vh0m6TinK-1>
- Diseño móvil:
<https://www.figma.com/proto/OcFYE6LZ5E8aH8oqUtctge/GeekHub?node-id=0-1&t=toKYKFbNnqpnFWI6-1>

4.2 Diagrama de clases del modelo de dominio de la API.



5. Diseño

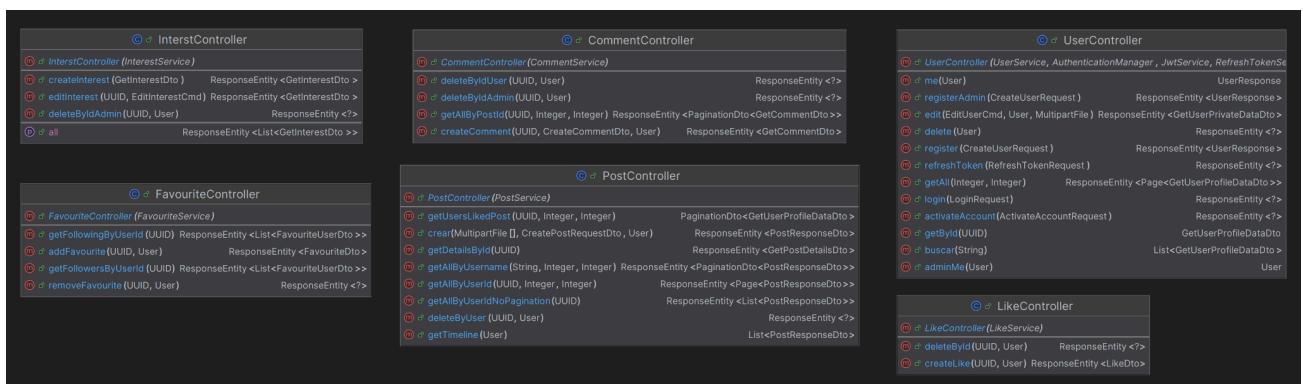
5.1 Diagrama de clases



6. Implementación

La estructura de la API REST está organizada en paquetes siguiendo una arquitectura por capas. Los paquetes principales son:

- **controller:** Contiene los controladores REST que gestionan las solicitudes HTTP. Cada clase en este paquete se encarga de exponer endpoints específicos del dominio de la aplicación y delegar la lógica al servicio correspondiente. Las clases principales son:
 - **CommentController:** Gestiona las operaciones relacionadas con los comentarios, como crear, listar o eliminar comentarios de publicaciones.
 - **FavouriteController:** Controla la funcionalidad para marcar usuarios como favoritos.
 - **InterestController:** Maneja las peticiones relacionadas con los intereses del usuario, permitiendo su asociación o modificación.
 - **LikeController:** Administra la lógica para añadir o quitar "me gusta" a publicaciones por parte de los usuarios.
 - **PostController:** Expone los endpoints relacionados con la creación, consulta, actualización y eliminación de publicaciones.
 - **UserController:** Gestiona la interacción con los usuarios del sistema, como el registro, login, actualización de perfil y recuperación de información personal.



- **dto:** Define los objetos de transferencia de datos (Data Transfer Objects), utilizados para estructurar las respuestas y peticiones del sistema.

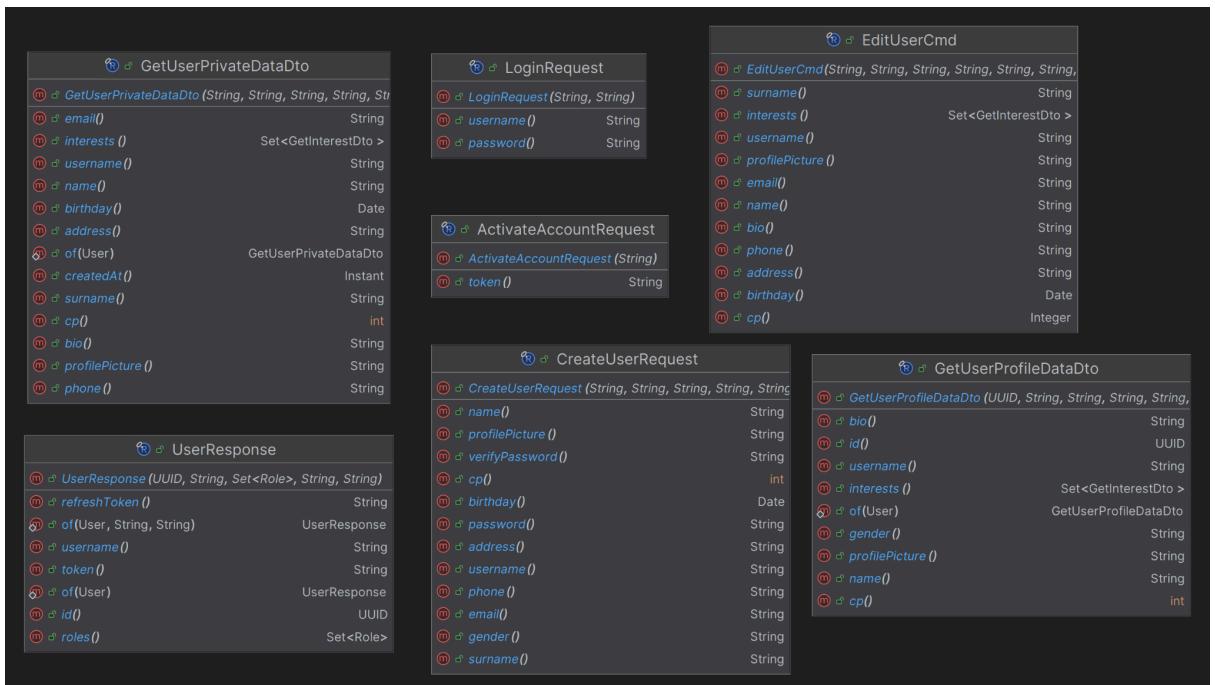
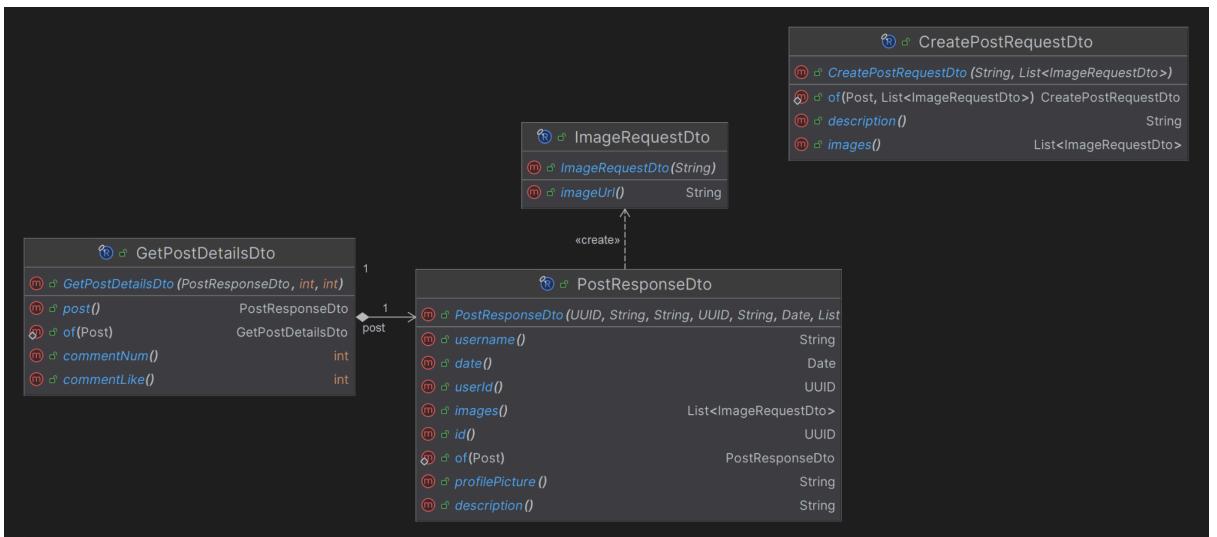
```

package com.salesianos.geekhub.dto.like
package com.salesianos.geekhub.dto.user
package com.salesianos.geekhub.dto.post
package com.salesianos.geekhub.dto.interest
package com.salesianos.geekhub.dto.favourite
package com.salesianos.geekhub.dto.comment

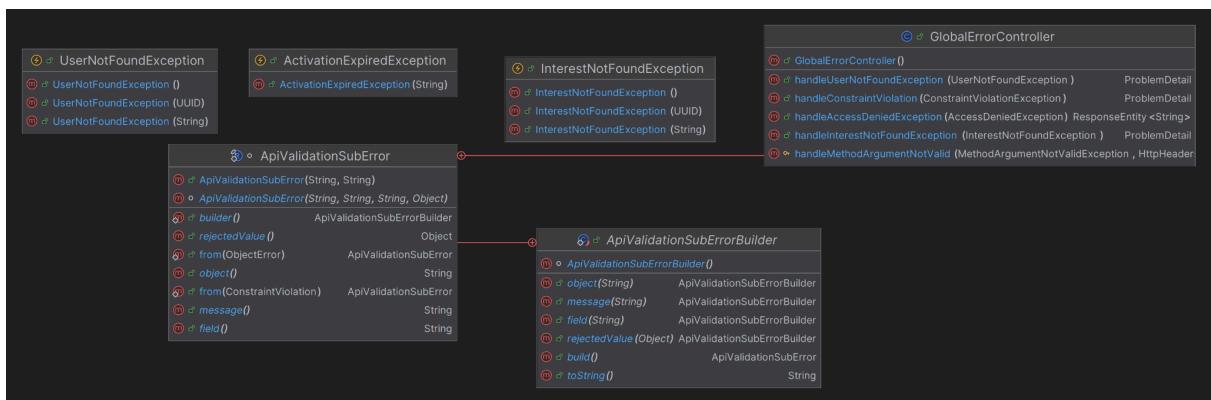
@Generated PaginationDto<T>
@Generated PaginationDto(int, int, long, int, List<T>)
@Generated paginasTotales() int
@Generated elementosEncontrados() long
@Generated of(Page<T>) PaginationDto<T>
@Generated numPagina() int
@Generated contenido() List<T>
@Generated tamanoPagina() int
  
```

CreateCommentDto	GetCommentDto
@ Generated CreateCommentDto(String) @ Generated content() String @ Generated of(Comment) CreateCommentDto	@ Generated GetCommentDto(UUID, UUID, String, String, String, Instant) @ Generated commentId() UUID @ Generated createdAt() Instant @ Generated username() String @ Generated userId() UUID @ Generated profilePicture() String @ Generated of(Comment) GetCommentDto @ Generated content() String

FavouriteDto	FavouriteUserDto
@ Generated FavouriteDto(UUID, String, String) @ Generated id() UUID @ Generated username() String @ Generated favouriteUsername() String @ Generated of(Favourite) FavouriteDto	@ Generated FavouriteUserDto(UUID, String, String, String) @ Generated of(User) FavouriteUserDto @ Generated profilePicture() String @ Generated name() String @ Generated username() String @ Generated id() UUID

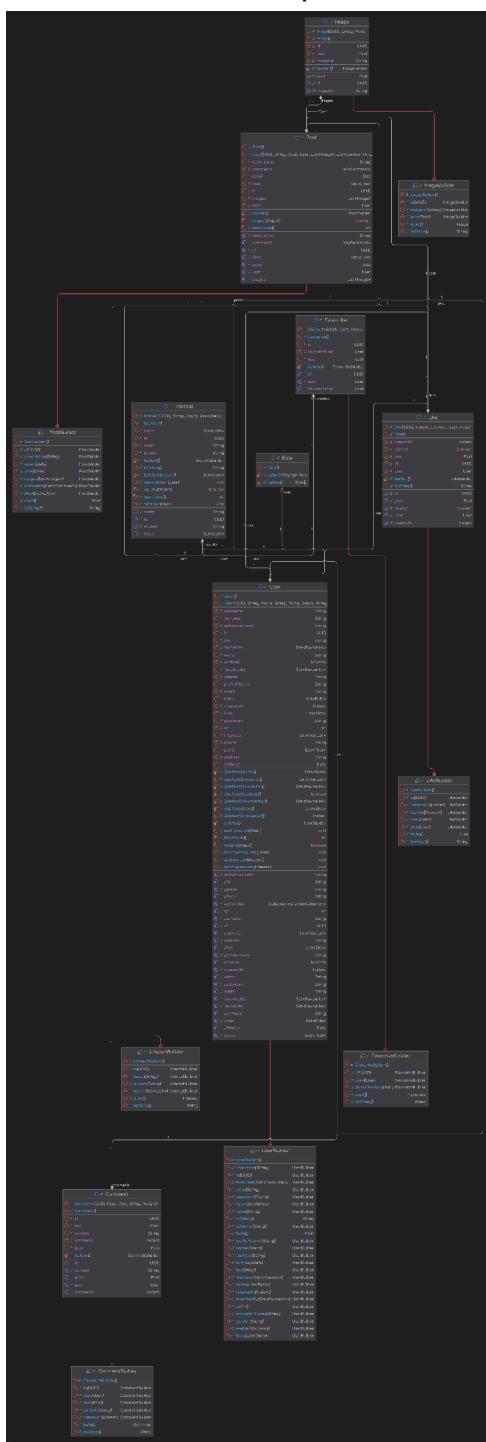


- **error:** Maneja los errores y excepciones personalizadas.



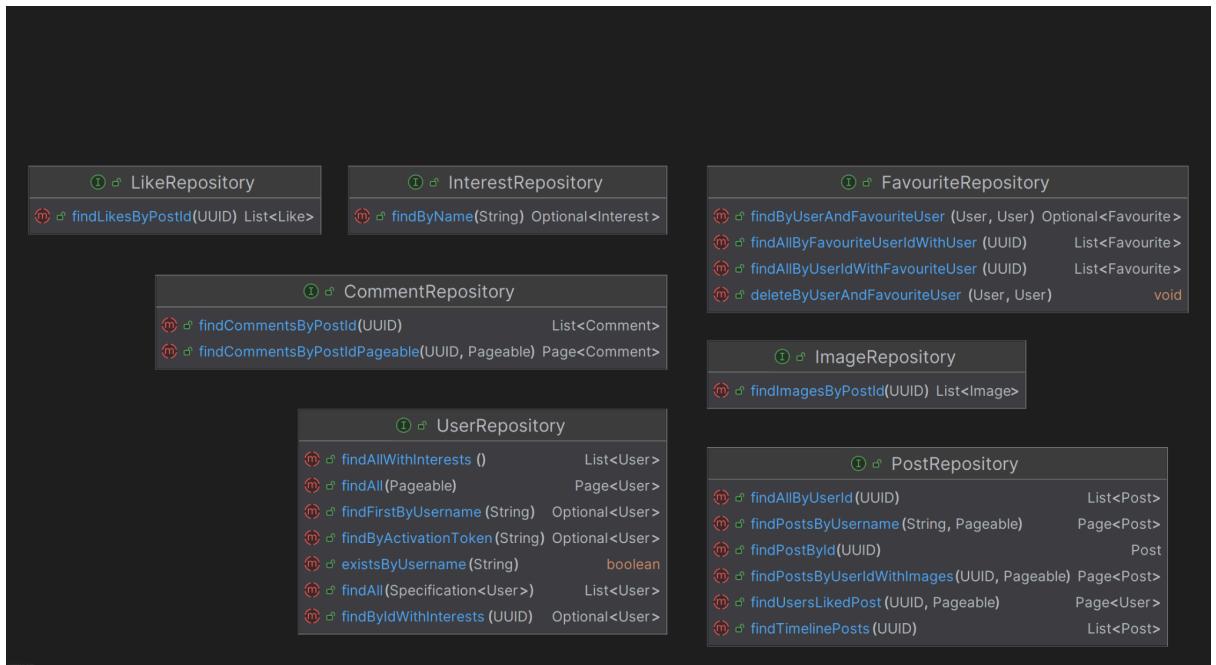
- **files:** Encapsula la lógica relacionada con la manipulación de archivos dentro del sistema. Se compone de controlador, servicio,dto, repositorio, excepciones y utils propios.

- **model:** Incluye las entidades del dominio que representan las estructuras de datos persistentes en la base de datos.

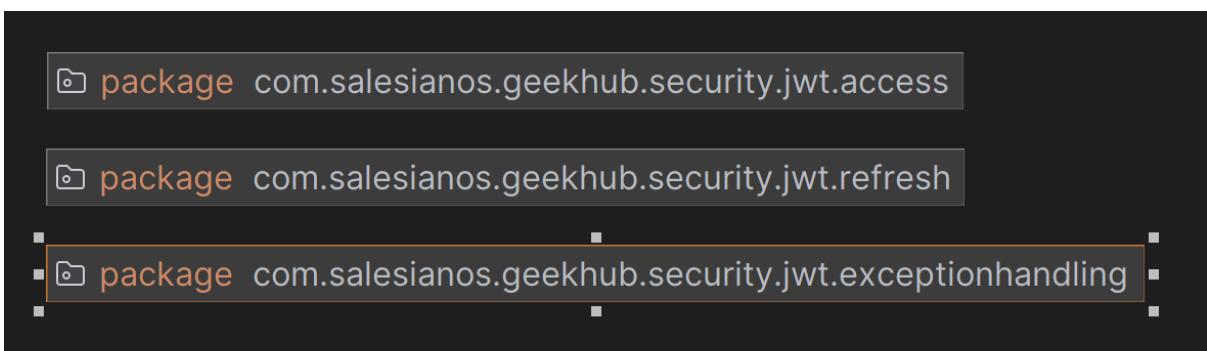


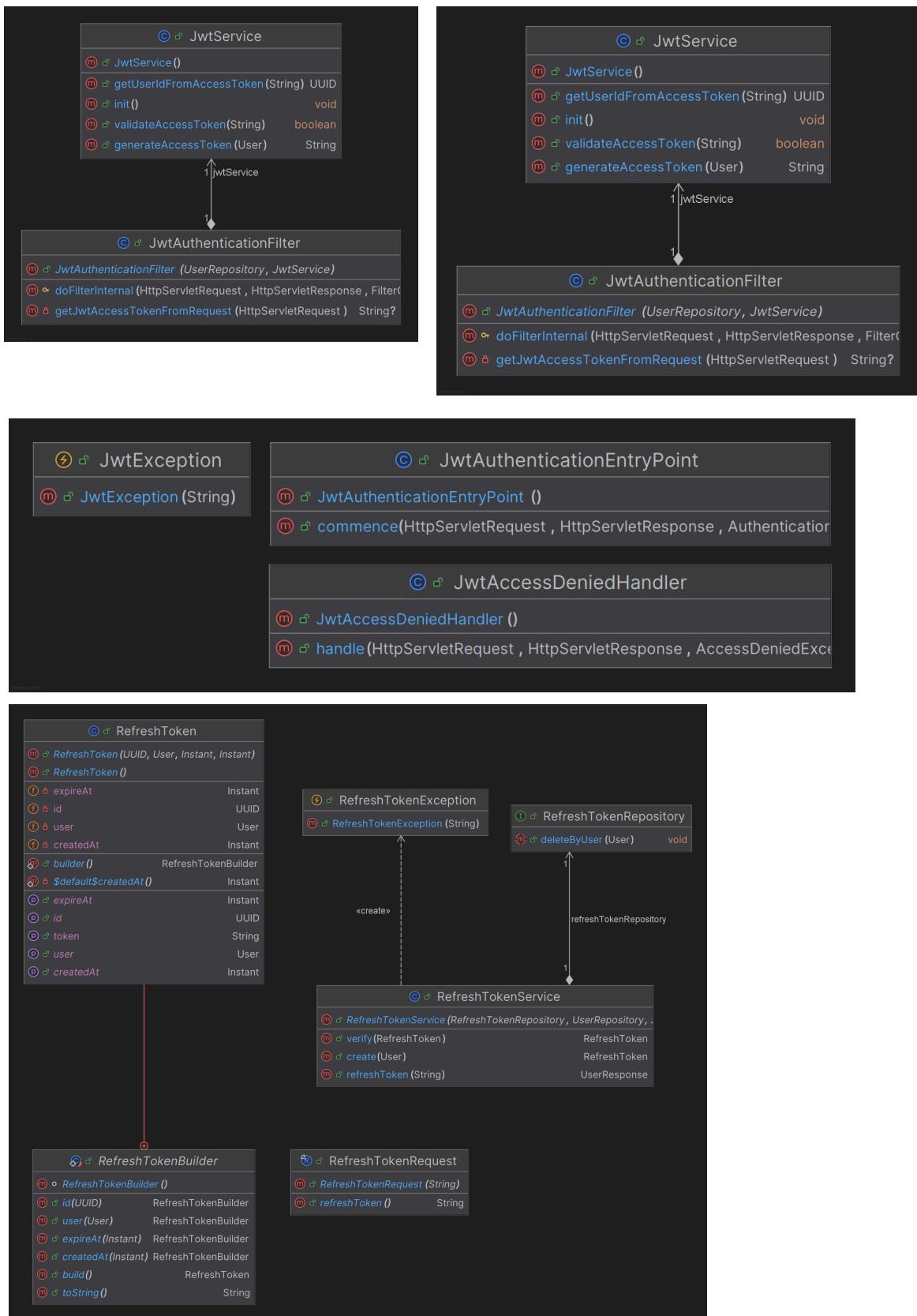
- **query**: Contiene clases para realizar consultas personalizadas o complejas sobre la base de datos.

- **repository:** Define las interfaces JPA que permiten el acceso a la base de datos, actuando como capa de persistencia.



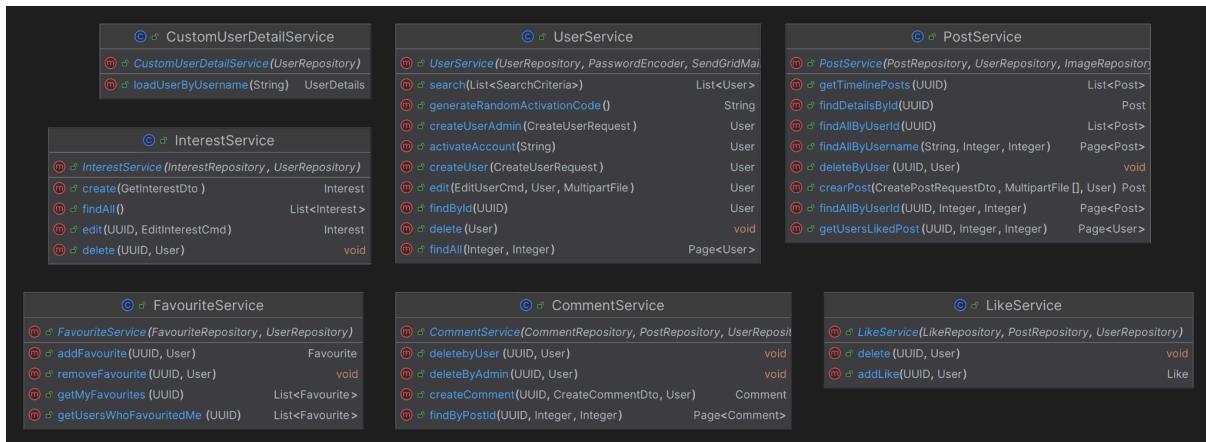
- **security:** Implementa la configuración y lógica de seguridad, incluyendo autenticación y control de acceso. Incluye:
 - **JwtAuthenticationFilter:** Filtro que intercepta peticiones y valida tokens JWT.
 - **SecurityConfig:** Configuración general de seguridad, roles, rutas protegidas, etc.
 - **UserDetailsServiceimpl:** Carga los detalles del usuario desde la base de datos para autenticación.



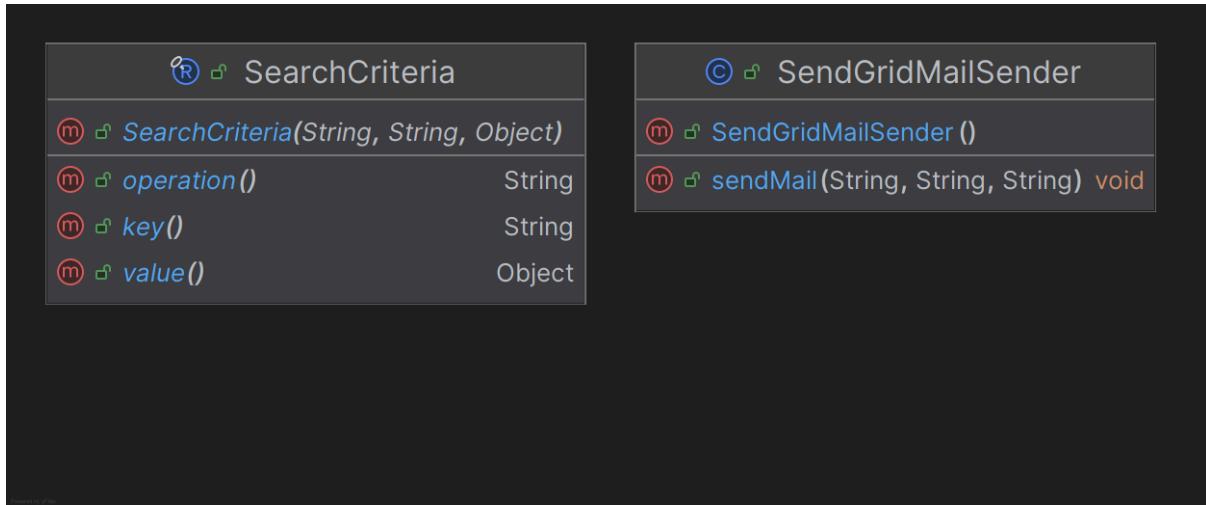


- **service:** Reúne la lógica de negocio principal, conectando controladores y repositorios. Cada clase implementa funciones específicas del dominio.
- Ejemplos:

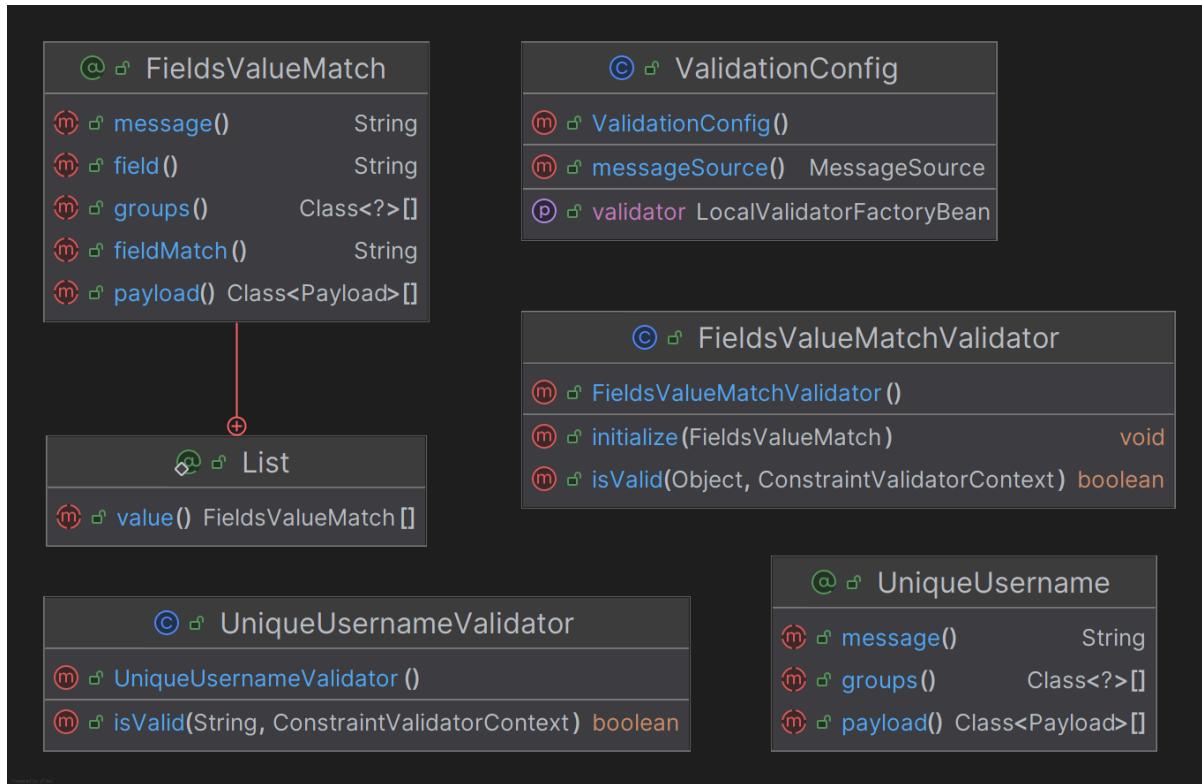
- **UserService:** Maneja registro, autenticación, y gestión de usuarios.
- **PostService, LikeService, CommentService,** etc.



- **util:** Proporciona clases y métodos de utilidad reutilizables en varias partes de la aplicación. Las clases principales son:
 - **SearchCriteria:** Permite construir filtros dinámicos para consultas personalizadas a la base de datos
 - **SendGridMailSender:** Encapsula la lógica para el envío de correos electrónicos a través del servicio SendGrid.

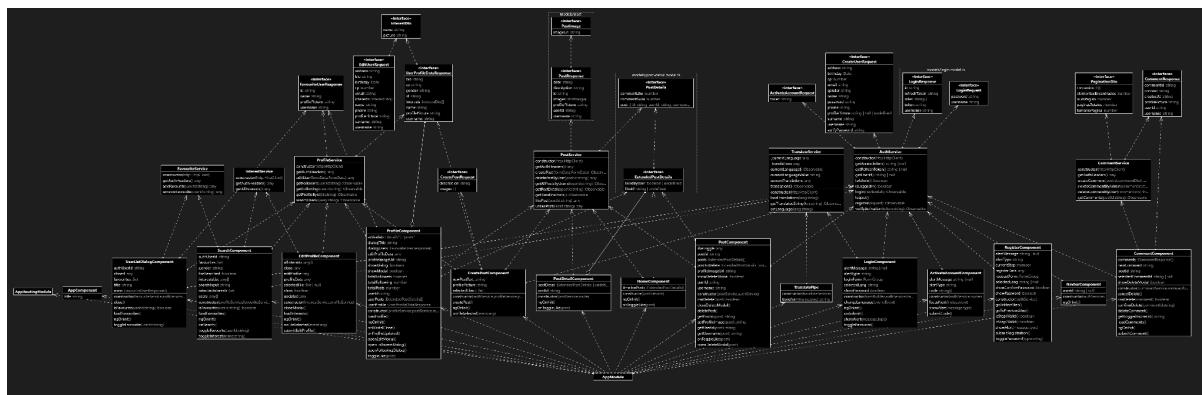


- **validation**: Agrupa los validadores personalizados.



- **uploads**: Es el paquete responsable de gestionar la subida de imágenes.

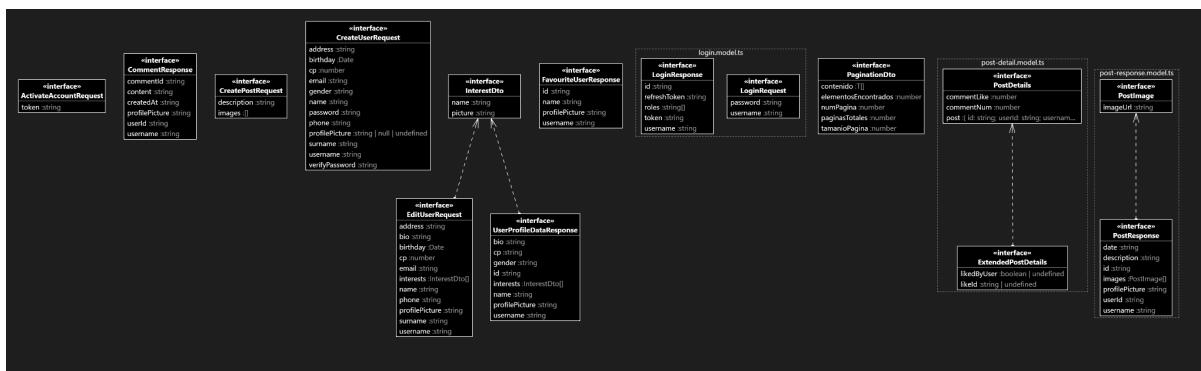
La aplicación web está desarrollada con Angular y organizada por separación de funcionalidades. A continuación se describen los principales paquetes o carpetas del proyecto:



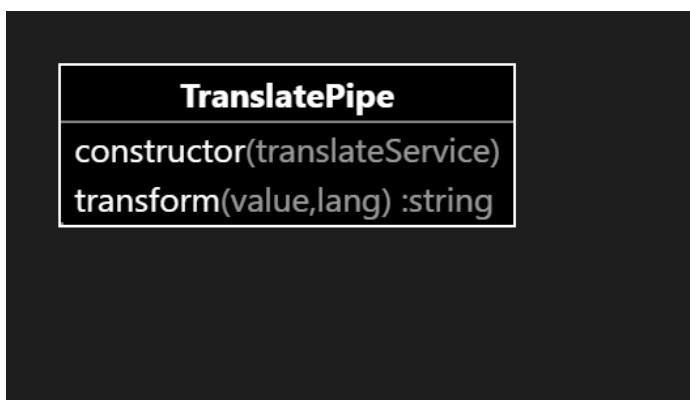
- **components**: Contiene los componentes de la interfaz de usuario. Cada componente representa una sección visual como register, edit profile, search, home o login.



- **models**: Define las interfaces que representan las entidades del sistema, como UserResponse, CreatePostRequest o ActivateAccountRequest.



- **pipes**: Incluye transformadores de datos personalizados, en este caso para la gestión de idiomas en la aplicación.



- **services:** Contiene los servicios responsables de consumir las APIs, gestionar autenticación y manejar datos

AuthService	CommentService	FavouriteService	InterestService	PostService	ProfileService	TranslateService
constructor(http:HttpClient) getAccessToken():string null getRefreshToken():string null getUserId():string null isAuthenticated():boolean isLoggedIn(credentials):Observable<User> logout() register(request):Observable<User> verifyActivation(tokenString):Observable<User>	constructor(authService:AuthService) getAuthHeaders():any getComments(postId:string):Observable<CommentData[]> createComment(postId:string,commentData):any deleteCommentByAdmin(commentId:string):any deleteCommentByUser(userId:string):any getComments(postId:string):Observable<CommentData[]>	constructor(http:HttpClient) getAuthHeaders():any createFavourite(postId:string):any deleteFavourite(userId:string):any addAllFavourites(userId:string):any getComments(postId:string):Observable<CommentData[]>	constructor(http:HttpClient) getAuthHeaders():any createCommentByAdmin(commentId:string):any deleteCommentByUser(userId:string):any getComments(postId:string):any	constructor(http:HttpClient) getAuthHeaders():any createPost(formData:FormData):Observable<PostData> deletePostByUser(postId:string):any getAllPosts():Observable<PostData[]> getPostDetails(postId:string):Observable<PostData> getFollowing(userId:string):Observable<User[]> getProfileById(id:string):Observable<User> likePost(postId:string):any unlikePost(postId:string):any	constructor(http:HttpClient) getAuthHeaders():any editUser(formData:FormData):any getUser(userId:string):Observable<User> getFollowers(userId:string):Observable<User[]> getProfile(userId:string):Observable<User>	,currentLanguage: any ,translations: any currentLanguage\$: Observable<string> currentLanguageValue: string currentLanguageName: string translations\$: Observable<any> constructor(http:HttpClient) loadTranslations(langString):Observable<any> getTranslatedString(keyString):Observable<any> setLanguage(lang:string)

- **shared:** Agrupa elementos reutilizables en distintos puntos de la aplicación: como comentarios o publicaciones.

CommentComponent	CreatePostComponent	NavbarComponent	PostComponent	UserListDialogComponent
comments :CommentResponse[] newComment: string postId: string selectedCommentId: string null showDeleteModal: boolean constructor(commentService:route.authService) cancelDelete() canDelete(comment):boolean confirmDelete(commentId:string) deleteComment() getLoggedInUserId():string loadComments() ngOnInit() submitComment()	newPostText: string profilePicture: string selectedFiles: File[] constructor(authService:profileService,p... createPost() ngOnInit() onFileSelected(event: any)	userId: string null constructor(authService) ngOnInit()	likeToggle: any postId: string posts: ExtendedPostDetails[] postToDelete: ExtendedPostDetails null profileImageUrl: string showDeleteModal: boolean userId: string username: string constructor(postService:authService) canDelete(post): boolean closeDeleteModal() deletePost() getPostId(post): string getProfileImage(post): string getUserId(post): string getUsername(post): string onToggleLike(post) openDeleteModal(post)	authUserId: string closed: any favourites: Set title: string users: FavouriteUserResponse[] constructor(favouriteService,authService)... close() isFavourite(userId: string): boolean loadFavourites() ngOnInit() toggleFavourite(userId: string)

- **environments:** Contiene la configuración específica para cada entorno, incluyendo URLs de la API y claves externas.
- **assets:** Carpeta para recursos estáticos como imágenes, íconos, fuentes, estilos externos o archivos JSON.