cookify

Documentación Cookify Kotlin App

Índice

ADAPTER

CarruselCrearPostAdapter

Descripción:

El `CarruselCrearPostAdapter` es un adaptador personalizado para el `RecyclerView` que muestra imágenes en un carrusel dentro de una funcionalidad de creación de posts en la aplicación Cookify.

Constructor:

- `CarruselCrearPostAdapter(images: List<Uri>)`
  - **Parámetros:**
    - `images`**: Lista de URIs que representan las imágenes a mostrar en el carrusel.**

Métodos:

1. `onCreateViewHolder(parent: ViewGroup, viewType: Int): ImageViewHolder`
   - **Descripción: Crea y devuelve una instancia de `ImageViewHolder`.**
   - **Parámetros:**
     - `parent`**: ViewGroup que contiene el RecyclerView.**
     - `viewType`**: Tipo de vista del elemento, en este caso no se utiliza.**
   - **Retorno: Instancia de `ImageViewHolder` asociada a la vista de cada elemento del RecyclerView.**
2. `onBindViewHolder(holder: ImageViewHolder, position: Int)`
   - **Descripción: Vincula los datos de la imagen en la posición `position` con la vista `holder`.**
   - **Parámetros:**
     - `holder`**: Instancia de `ImageViewHolder` que contiene la vista de cada elemento.**
     - `position`**: Posición del elemento dentro del RecyclerView.**
   - **Acción: Carga la imagen desde la URI utilizando Glide y la muestra en el `ImageView` asociado en `holder`.**
3. `getItemCount(): Int`
   - **Descripción: Devuelve el número total de elementos en la lista de imágenes.**
   - **Retorno: Número total de elementos en la lista de imágenes.**

Clase interna:

- `ImageViewHolder`
    - **Descripción: Representa la vista de cada elemento individual dentro del RecyclerView.**
    - **Propiedades:**
        - `imageView`**: ImageView que muestra la imagen dentro del elemento.**

CarruselMostrarPostAdapter

Descripción:

El `CarruselMostrarPostAdapter` es un adaptador personalizado para el `RecyclerView` que muestra imágenes en un carrusel para la visualización de posts en la aplicación Cookify.

Constructor:

- `CarruselMostrarPostAdapter(images: List<Bitmap>)`
    - **Parámetros:**
        - `images`: Lista de bitmaps que representan las imágenes a mostrar en el carrusel.

Métodos:

1. `onCreateViewHolder(parent: ViewGroup, viewType: Int): ImageViewHolder`
    - **Descripción:** Crea y devuelve una instancia de `ImageViewHolder`.
    - **Parámetros:**
        - `parent`: ViewGroup que contiene el RecyclerView.
        - `viewType`: Tipo de vista del elemento, en este caso no se utiliza.
    - **Retorno:** Instancia de `ImageViewHolder` asociada a la vista de cada elemento del RecyclerView.
2. `onBindViewHolder(holder: ImageViewHolder, position: Int)`
    - **Descripción:** Vincula los datos del bitmap en la posición `position` con la vista `holder`.
    - **Parámetros:**

        - `holder`: Instancia de `ImageViewHolder` que contiene la vista de cada elemento.
        - `position`: Posición del elemento dentro del RecyclerView.
    - **Acción:** Carga el bitmap en el `ImageView` asociado en `holder` utilizando Glide para la carga eficiente de imágenes.
3. `getItemCount(): Int`
    - **Descripción:** Devuelve el número total de elementos en la lista de imágenes.
    - **Retorno:** Número total de elementos en la lista de imágenes.

Clase interna:

- `ImageViewHolder`
    - **Descripción:** Representa la vista de cada elemento individual dentro del RecyclerView.
    - **Propiedades:**
        - `imageView`: ImageView que muestra la imagen dentro del elemento.

ComentariosAdapter

Descripción:

El `ComentariosAdapter` es un adaptador personalizado para el `RecyclerView` que muestra una lista de comentarios en la aplicación Cookify.

Constructor:

- `ComentariosAdapter(comentarios: List<Comentario>)`
    - **Parámetros:**
        - `comentarios`: Lista inicial de objetos de tipo `Comentario` que se mostrarán en el RecyclerView.

Métodos:

1. `onCreateViewHolder(parent: ViewGroup, viewType: Int): ComentarioViewHolder`
    - **Descripción:** Crea y devuelve una instancia de `ComentarioViewHolder`.
    - **Parámetros:**
        - `parent`: ViewGroup que contiene el RecyclerView.

        - `viewType`: Tipo de vista del elemento, en este caso no se utiliza.
    - **Retorno:** Instancia de `ComentarioViewHolder` asociada a la vista de cada elemento del RecyclerView.
2. `onBindViewHolder(holder: ComentarioViewHolder, position: Int)`
    - **Descripción:** Vincula los datos del comentario en la posición `position` con la vista `holder`.
    - **Parámetros:**
        - `holder`: Instancia de `ComentarioViewHolder` que contiene la vista de cada elemento.
        - `position`: Posición del elemento dentro del RecyclerView.
    - **Acción:** Llama al método `bind(comentario)` en `holder` para establecer los datos del comentario en los elementos de la vista.
3. `getItemCount(): Int`
    - **Descripción:** Devuelve el número total de elementos en la lista de comentarios.
    - **Retorno:** Número total de elementos en la lista de comentarios.
4. `actualizarComentarios(nuevosComentarios: List<Comentario>)`
    - **Descripción:** Actualiza la lista de comentarios con una nueva lista de comentarios y notifica al adaptador del cambio.
    - **Parámetros:**
        - `nuevosComentarios`: Nueva lista de objetos de tipo `Comentario` que se utilizará para actualizar los comentarios mostrados.
    - **Acción:** Asigna la lista `nuevosComentarios` a la lista `comentarios`, luego llama a `notifyDataSetChanged()` para actualizar el RecyclerView con los nuevos datos.

Clase interna:

- `ComentarioViewHolder`
    - **Descripción:** Representa la vista de cada elemento individual dentro del RecyclerView.
    - **Propiedades:**
        - `tvUsuario`: TextView que muestra el nombre del usuario que realizó el comentario.
        - `tvContenido`: TextView que muestra el contenido del comentario.
    - **Métodos:**

        - `bind(comentario: Comentario)`
            - **Descripción:** Establece los datos del comentario en los TextViews `tvUsuario` y `tvContenido`.
            - **Parámetros:**
                - `comentario`: Objeto de tipo `Comentario` que contiene los datos a mostrar en la vista del comentario.

MisPostsAdapter

Descripción:

El `MisPostsAdapter` es un adaptador personalizado para el `RecyclerView` que muestra una lista de posts creados por el usuario en la aplicación Cookify. Permite la visualización de imágenes decodificadas desde cadenas Base64 y proporciona funcionalidad para eliminar posts y ver detalles de cada post.

Constructor:

- `MisPostsAdapter(postList: List<Post>, context: Context)`
    - **Parámetros:**
        - `postList`: Lista de objetos de tipo `Post` que se mostrarán en el RecyclerView.
        - `context`: Contexto de la actividad o fragmento desde donde se instancia el adaptador.

Métodos:

1. `onCreateViewHolder(parent: ViewGroup, viewType: Int): PostViewHolder`
    - **Descripción:** Crea y devuelve una instancia de `PostViewHolder`.
    - **Parámetros:**
        - `parent`: ViewGroup que contiene el RecyclerView.
        - `viewType`: Tipo de vista del elemento, en este caso no se utiliza.
    - **Retorno:** Instancia de `PostViewHolder` asociada a la vista de cada elemento del RecyclerView.
2. `onBindViewHolder(holder: PostViewHolder, position: Int)`
    - **Descripción:** Vincula los datos del post en la posición `position` con la vista `holder`.

- **Parámetros:**
  - `holder`: Instancia de `PostViewHolder` que contiene la vista de cada elemento.
  - `position`: Posición del elemento dentro del RecyclerView.
- **Acción:** Decodifica la imagen base64 del post y carga la imagen en `imageViewPost` usando Glide. También define el comportamiento para la eliminación de posts y para abrir la actividad `VerPostActivity` al hacer clic en la imagen.

3. `getItemCount(): Int`
   - **Descripción:** Devuelve el número total de elementos en la lista de posts.
   - **Retorno:** Número total de elementos en la lista de posts.
4. `showDeleteConfirmationDialog(postId: Long)`
   - **Descripción:** Muestra un diálogo de confirmación para eliminar el post seleccionado.
   - **Parámetros:**
     - `postId`: ID del post que se desea eliminar.
   - **Acción:** Al confirmar la eliminación, llama al método `deletePost(postId)` para procesar la eliminación del post.
5. `deletePost(postId: Long)`
   - **Descripción:** Elimina el post del servidor utilizando `PostService.deleteUserPost()` y actualiza la lista de posts.
   - **Parámetros:**
     - `postId`: ID del post que se va a eliminar.
   - **Acción:** Actualiza la lista de posts eliminando el post correspondiente y notifica al adaptador del cambio mediante `notifyDataSetChanged()`.

Clase interna:

- `PostViewHolder`
  - **Descripción:** Representa la vista de cada elemento individual dentro del RecyclerView.
  - **Propiedades:**
    - `imageViewPost`: ImageView que muestra la imagen asociada al post.

PostAdapter

Descripción:

El `PostAdapter` es un adaptador personalizado para el `RecyclerView` que muestra una lista de posts en la aplicación Cookify. Cada elemento del RecyclerView representa un post individual y contiene detalles como la imagen del usuario, título, descripción, categoría, subcategoría y un carrusel de imágenes del post. Además, proporciona funcionalidad para ver el perfil del usuario, comentar en el post y denunciar posts que no sean relacionados con comida.

Constructor:

- **`PostAdapter(posts: MutableList<Post>)`**
  - **Parámetros:**
    - **`posts`: Lista mutable de objetos de tipo `Post` que se mostrarán en el RecyclerView.**

Métodos:

1. **`onCreateViewHolder(parent: ViewGroup, viewType: Int): PostViewHolder`**
   - **Descripción: Crea y devuelve una instancia de `PostViewHolder`.**
   - **Parámetros:**
     - **`parent`: ViewGroup que contiene el RecyclerView.**
     - **`viewType`: Tipo de vista del elemento, en este caso no se utiliza.**
   - **Retorno: Instancia de `PostViewHolder` asociada a la vista de cada elemento del RecyclerView.**
2. **`onBindViewHolder(holder: PostViewHolder, position: Int)`**
   - **Descripción: Vincula los datos del post en la posición `position` con la vista `holder`.**
   - **Parámetros:**
     - **`holder`: Instancia de `PostViewHolder` que contiene la vista de cada elemento.**
     - **`position`: Posición del elemento dentro del RecyclerView.**
   - **Acción: Configura todos los elementos visuales del post, incluyendo la imagen del usuario, título, descripción, categoría, subcategoría y el carrusel de imágenes. También define el comportamiento para la visualización del perfil del usuario, la apertura de la actividad de comentarios y la denuncia del post.**
3. **`getItemCount(): Int`**
   - **Descripción: Devuelve el número total de elementos en la lista de posts.**

    - **Retorno: Número total de elementos en la lista de posts.**
4. **`addPosts(newPosts: List<Post>)`**
    - **Descripción: Añade nuevos posts a la lista existente y notifica al adaptador del cambio.**
    - **Parámetros:**
        - **`newPosts`: Lista de nuevos objetos de tipo `Post` que se añadirán al RecyclerView.**
5. **`clearPosts()`**
    - **Descripción: Elimina todos los posts de la lista y notifica al adaptador del cambio.**

Clase interna:

- **`PostViewHolder`**
    - **Descripción: Representa la vista de cada elemento individual dentro del RecyclerView.**
    - **Propiedades:**
        - **`postImageView`: Componente personalizado para mostrar la imagen del usuario.**
        - **`postTitleTextView`: TextView para mostrar el título del post.**
        - **`postDescriptionTextView`: TextView para mostrar la descripción del post.**
        - **`postUserTextView`: TextView para mostrar el nombre del usuario que creó el post.**
        - **`postCategoryTextView`: TextView para mostrar la categoría del post.**
        - **`postSubCategoryTextView`: TextView para mostrar la subcategoría del post.**
        - **`postViewPager`: ViewPager2 para mostrar un carrusel de imágenes del post.**
        - **`ivComentar`: ImageView para permitir comentarios en el post.**
        - **`ivDenunciar`: ImageView para permitir la denuncia del post.**

Métodos privados:

- **`denunciarPost(post: Post)`**
    - **Descripción: Envía una denuncia del post al servidor si se considera que no es relacionado con comida.**
    - **Parámetros:**
        - **`post`: Objeto de tipo `Post` que se va a denunciar.**

RecetaApiAdapter

Descripción:

El `RecetaApiAdapter` es un adaptador para RecyclerView diseñado para mostrar una lista de recetas obtenidas de una API externa en la aplicación Cookify. Cada elemento del RecyclerView representa una receta individual y contiene una imagen de la receta, el nombre de la receta y un botón para ver detalles adicionales de la receta.

Constructor:

- **`RecetaApiAdapter(listaRecetas: List<Receta>)`**
  - **Parámetros:**
    - **`listaRecetas`: Lista de objetos de tipo `Receta` que se mostrarán en el RecyclerView.**

Métodos:

1. **`onCreateViewHolder(parent: ViewGroup, viewType: Int): RecetaViewHolder`**
   - **Descripción: Crea y devuelve una instancia de `RecetaViewHolder` que contiene las vistas de cada elemento del RecyclerView.**
   - **Parámetros:**
     - **`parent`: ViewGroup que contiene el RecyclerView.**
     - **`viewType`: Tipo de vista del elemento, en este caso no se utiliza.**
   - **Retorno: Instancia de `RecetaViewHolder` asociada a la vista de cada elemento del RecyclerView.**
2. **`onBindViewHolder(holder: RecetaViewHolder, position: Int)`**
   - **Descripción: Vincula los datos de la receta en la posición `position` con las vistas en `holder`.**
   - **Parámetros:**
     - **`holder`: Instancia de `RecetaViewHolder` que contiene la vista de cada elemento.**
     - **`position`: Posición del elemento dentro del RecyclerView.**
   - **Acción: Configura la imagen de la receta usando Glide, establece el nombre de la receta y configura el listener para el botón de ver detalles de la receta.**

3. `getItemCount(): Int`
    - **Descripción: Devuelve el número total de elementos en la lista de recetas.**
    - **Retorno: Número total de elementos en la lista de recetas.**

Clase interna:

- `RecetaViewHolder`
    - **Descripción: Representa la vista de cada elemento individual dentro del RecyclerView.**
    - **Propiedades:**
        - `ivRecetaApi`**: ImageView para mostrar la imagen de la receta.**
        - `tvRecetaApi`**: TextView para mostrar el nombre de la receta.**
        - `ivVerRecetaApi`**: ImageView para permitir al usuario ver detalles adicionales de la receta.**

UsuariosBuscadosAdapter

Descripción:

El `UsuariosBuscadosAdapter` es un adaptador para RecyclerView diseñado para mostrar una lista de usuarios buscados en la aplicación Cookify. Cada elemento del RecyclerView representa un usuario y contiene una imagen de perfil redondeada, el nombre del usuario y un elemento interactivo para ver más detalles del perfil del usuario.

Constructor:

- `UsuariosBuscadosAdapter(listaUsuarios: List<Usuario?>)`
    - **Parámetros:**
        - `listaUsuarios`**: Lista de objetos de tipo `Usuario` que se mostrarán en el RecyclerView. Puede contener elementos nulos.**

Métodos:

1. `onCreateViewHolder(parent: ViewGroup, viewType: Int): RecetaViewHolder`
    - **Descripción: Crea y devuelve una instancia de `RecetaViewHolder` que contiene las vistas de cada elemento del RecyclerView.**
    - **Parámetros:**

        - **`parent`: ViewGroup que contiene el RecyclerView.**
        - **`viewType`: Tipo de vista del elemento, en este caso no se utiliza.**
    - **Retorno: Instancia de `RecetaViewHolder` asociada a la vista de cada elemento del RecyclerView.**
2. **`onBindViewHolder(holder: RecetaViewHolder, position: Int)`**
    - **Descripción: Vincula los datos del usuario en la posición `position` con las vistas en `holder`.**
    - **Parámetros:**
        - **`holder`: Instancia de `RecetaViewHolder` que contiene la vista de cada elemento.**
        - **`position`: Posición del elemento dentro del RecyclerView.**
    - **Acción: Configura la imagen de perfil redondeada usando un servicio de decodificación base64 (`ServicioBase64`), establece el nombre del usuario y configura el listener para la vista del elemento, permitiendo ver más detalles del perfil del usuario.**
3. **`getItemCount(): Int`**
    - **Descripción: Devuelve el número total de elementos en la lista de usuarios.**
    - **Retorno: Número total de elementos en la lista de usuarios.**

Clase interna:

- **`RecetaViewHolder`**
    - **Descripción: Representa la vista de cada elemento individual dentro del RecyclerView.**
    - **Propiedades:**
        - **`ivImagenUsuario`: Componente de imagen redondeada para mostrar la imagen de perfil del usuario.**
        - **`tvNombreUsuario`: TextView para mostrar el nombre del usuario.**
        - **`ivNombreUsuario`: ImageView para permitir al usuario interactuar y ver más detalles del perfil del usuario.**

COMPONENTES

ComponenteImagenRedondeada

Descripción:

El `ComponenteImagenRedondeada` es una subclase de `ConstraintLayout` diseñada para encapsular la visualización de una imagen redondeada en la aplicación Cookify. Este componente personalizado utiliza un `ShapeableImageView` para mostrar imágenes redondeadas y proporciona métodos de inicialización para configurar la imagen predeterminada y obtener acceso al `ShapeableImageView` subyacente.

Constructores:

- `ComponenteImagenRedondeada(context: Context)`
  - **Descripción:** Constructor primario que llama a la inicialización del componente con un contexto.
  - **Parámetros:**
    - `context`: Contexto de la aplicación o actividad que utiliza este componente.
- `ComponenteImagenRedondeada(context: Context, attrs: AttributeSet?)`
  - **Descripción:** Constructor secundario que llama a la inicialización del componente con un contexto y atributos XML.
  - **Parámetros:**
    - `context`: Contexto de la aplicación o actividad que utiliza este componente.
    - `attrs`: Atributos XML del diseño que define configuraciones adicionales para el componente.
- `ComponenteImagenRedondeada(context: Context, attrs: AttributeSet?, defStyleAttr: Int)`
  - **Descripción:** Constructor secundario que llama a la inicialización del componente con un contexto, atributos XML y un estilo predeterminado.
  - **Parámetros:**
    - `context`: Contexto de la aplicación o actividad que utiliza este componente.
    - `attrs`: Atributos XML del diseño que define configuraciones adicionales para el componente.

        - `defStyleAttr`: Estilo predeterminado a aplicar al componente.

Métodos Públicos:

- `initialize(context: Context, attrs: AttributeSet?)`
    - **Descripción:** Método privado utilizado en los constructores para inflar el diseño XML del componente y obtener la referencia al `ShapeableImageView`.
    - **Parámetros:**
        - `context`: Contexto de la aplicación o actividad que utiliza este componente.
        - `attrs`: Atributos XML del diseño que define configuraciones adicionales para el componente.

Propiedades:

- `imagenEditable: ShapeableImageView`
    - **Descripción:** Componente `ShapeableImageView` que muestra la imagen redondeada dentro del `ComponenteImagenRedondeada`.

ComponenteModalSalir

Descripción:

`ComponenteModalSalir` es una subclase de `DialogFragment` que muestra un cuadro de diálogo modal para confirmar la salida de la aplicación. Este componente permite al usuario confirmar o cancelar la acción de salida, realizando acciones como cerrar la sesión del usuario y eliminar el token de autenticación almacenado.

Constructores:

- `ComponenteModalSalir()`
    - **Descripción:** Constructor por defecto que inicializa la instancia de `ComponenteModalSalir`.

Métodos Públicos:

- `onCreateDialog(savedInstanceState: Bundle?): Dialog`

- **Descripción:** Método sobreescrito de `DialogFragment` que crea y devuelve el cuadro de diálogo modal para confirmar la salida.
- **Parámetros:**
  - `savedInstanceState`: Instancia de `Bundle` que contiene datos previamente guardados del fragmento.
- **Devuelve:** Un objeto `Dialog` que representa el cuadro de diálogo modal creado.
- **Excepciones:** `IllegalStateException` si no se puede obtener la actividad asociada al fragmento.

Funcionalidad:

- El cuadro de diálogo muestra dos botones: "Confirmar" y "Cancelar".
- Al hacer clic en "Confirmar", se ejecuta la acción para cerrar la sesión del usuario y eliminar el token de autenticación.
- Al hacer clic en "Cancelar", se cierra el cuadro de diálogo sin realizar ninguna acción adicional.
- Después de cerrar la sesión y eliminar el token, redirige automáticamente a la actividad de inicio de sesión (`LoginActivity`).

ComponentesComentarios

Descripción:

`ComponentesComentarios` es una subclase de `BottomSheetDialogFragment` que muestra un fragmento de hoja inferior para gestionar comentarios asociados a un post específico. Permite a los usuarios ver los comentarios existentes, agregar nuevos comentarios y actualizar la lista de comentarios después de enviar uno nuevo.

Constructores:

- `ComponentesComentarios()`
  - **Descripción:** Constructor por defecto que inicializa la instancia de `ComponentesComentarios`.

Métodos Públicos:

- `onCreateDialog(savedInstanceState: Bundle?): Dialog`

- **Descripción:** Método sobreescrito de `BottomSheetDialogFragment` que crea y devuelve el diálogo de hoja inferior para la gestión de comentarios.
- **Parámetros:**
  - `savedInstanceState`: Instancia de `Bundle` que contiene datos previamente guardados del fragmento.
- **Devuelve:** Un objeto `Dialog` que representa el diálogo de hoja inferior creado.

Funcionalidad:

- Muestra un cuadro de diálogo de hoja inferior con una lista de comentarios asociados a un post específico.
- Permite a los usuarios agregar nuevos comentarios y ver los comentarios existentes.
- Utiliza `RecyclerView` junto con `ComentariosAdapter` para mostrar los comentarios de manera eficiente.
- Maneja la obtención de comentarios mediante llamadas a la API usando `PostService`.
- Administra la autenticación y autorización usando tokens de usuario mediante `TokenManagerService`.

CONTROLADORES

ControladorCarga

Descripción:

`ControladorCarga` es una clase auxiliar que proporciona funciones estáticas para gestionar la visibilidad de componentes de la interfaz de usuario, como un `ProgressBar` y un `ViewGroup`. Está diseñada para simplificar el manejo de la carga y la visibilidad de elementos en la interfaz de usuario durante operaciones asíncronas.

Funciones Estáticas:

- `showLoading(isLoading: Boolean, layout: ViewGroup, progressBar: ProgressBar)`
  - **Descripción:** Controla la visibilidad del `ProgressBar` y del `layout` basado en el estado de carga.

    - **Parámetros:**
        - `isLoading`: Booleano que indica si la carga está activa o no.
        - `layout`: `ViewGroup` que contiene los componentes de la interfaz de usuario.
        - `progressBar`: `ProgressBar` que muestra el estado de la carga.
    - **Uso:** Llama a esta función para mostrar u ocultar el `ProgressBar` y los componentes del `layout` según sea necesario durante la carga.

Funciones Privadas:

- `setAllChildrenGone()`
    - **Descripción:** Hace invisibles todos los componentes hijos dentro del `ViewGroup`.
    - **Uso:** Utilizada internamente por `showLoading()` para hacer invisibles todos los componentes del `layout` cuando se activa la carga.
- `setAllChildrenVisible()`
    - **Descripción:** Hace visibles todos los componentes hijos dentro del `ViewGroup`.
    - **Uso:** Utilizada internamente por `showLoading()` para hacer visibles todos los componentes del `layout` cuando se desactiva la carga.

MODELOS

Comentario

Descripción:

`Comentario` es una clase de datos que representa un comentario realizado por un usuario en un post específico dentro de la aplicación Cookify.

Atributos:

- `id`: Identificador único del comentario.
    - Tipo: `Long?` (nullable)
    - Descripción: Es el identificador único del comentario, que puede ser nulo si aún no ha sido asignado por el servidor.
- `post`: Post al que está asociado el comentario.
    - Tipo: `Post?` (nullable)

    - Descripción: Referencia al post al cual pertenece este comentario. Puede ser nulo si la referencia al post no está disponible o no es necesaria.
- `usuarioNombre`: Nombre del usuario que realizó el comentario.
    - Tipo: `String?` (nullable)
    - Descripción: Nombre del usuario que escribió este comentario.
- `contenido`: Contenido del comentario.
    - Tipo: `String?` (nullable)
    - Descripción: El texto del comentario escrito por el usuario.

Constructor:

- `Comentario(id: Long?, post: Post?, usuarioNombre: String?, contenido: String?)`
    - **Descripción:** Constructor primario que inicializa todos los atributos de la clase.

ImagenPost

Descripción:

`ImagenPost` es una clase de datos que representa una imagen en formato base64 asociada a un post en la aplicación Cookify.

Atributos:

- `imagenBase64`: Representa la imagen codificada en formato base64.
    - Tipo: `String`
    - Descripción: Es la representación de la imagen en formato base64, que puede ser decodificada para mostrar la imagen asociada al post.

Constructor:

- `ImagenPost(imagenBase64: String)`
    - **Descripción:** Constructor primario que inicializa el atributo `imagenBase64` con el valor proporcionado.

ListaCategorias

Descripción:

`ListaCategorias` es una clase de datos que encapsula una lista de objetos de tipo `Temporada`, representando las categorías de temporadas en la aplicación Cookify.

Atributos:

- `categories`: Lista de objetos de tipo `Temporada`.
    - Tipo: `List<Temporada>`
    - Descripción: Representa las categorías de temporadas disponibles en la aplicación Cookify.

Constructor:

- `ListaCategorias(categories: List<Temporada>)`
    - **Descripción:** Constructor primario que inicializa el atributo `categories` con la lista proporcionada de objetos `Temporada`.

ListaRecetaCompleta

Descripción:

`ListaRecetaCompleta` es una clase que encapsula una lista de objetos `RecetaCompleta`, representando una colección de recetas completas en la aplicación Cookify.

Atributos:

- `meals`: Lista de objetos de tipo `RecetaCompleta`.
    - Tipo: `List<RecetaCompleta>`
    - Descripción: Representa la lista de recetas completas disponibles en la aplicación Cookify.

Constructor:

- `ListaRecetaCompleta(meals: List<RecetaCompleta>)`
    - **Descripción:** Constructor primario que inicializa el atributo `meals` con la lista proporcionada de objetos `RecetaCompleta`.

ListaRecetas

Descripción:

`ListaRecetas` es una clase de datos que contiene una lista de objetos `Receta`, representando una colección de recetas en la aplicación Cookify.

Atributos:

- `meals`: Lista de objetos de tipo `Receta`.
    - Tipo: `List<Receta>`
    - Descripción: Representa la lista de recetas disponibles en la aplicación Cookify.

Constructor:

- `ListaRecetas(meals: List<Receta>)`
    - **Descripción:** Constructor primario que inicializa el atributo `meals` con la lista proporcionada de objetos `Receta`.

Post

Descripción:

`Post` es una clase de datos que encapsula los detalles de una publicación realizada por un usuario en la aplicación Cookify.

Atributos:

- `id`: Identificador único de la publicación.
    - Tipo: `Int`
    - Descripción: Identifica de manera única esta publicación dentro del sistema.
- `nombre`: Nombre o título de la publicación.
    - Tipo: `String`
    - Descripción: Representa el nombre o título asociado a esta publicación.
- `descripcion`: Descripción o contenido textual de la publicación.
    - Tipo: `String`

    - Descripción: Contiene el texto descriptivo o contenido de la publicación.
- `usuarioUid`: Identificador único del usuario que realizó la publicación.
    - Tipo: `String`
    - Descripción: Identifica de manera única al usuario que creó esta publicación.
- `usuarioNombre`: Nombre de usuario del autor de la publicación.
    - Tipo: `String`
    - Descripción: Nombre del usuario que realizó la publicación.
- `imagenes`: Lista de imágenes asociadas a la publicación.
    - Tipo: `List<ImagenPost>`
    - Descripción: Contiene las imágenes relacionadas con esta publicación en forma de objetos `ImagenPost`.
- `fechaDeCreacion`: Fecha y hora en que se creó la publicación.
    - Tipo: `String`
    - Descripción: Representa la fecha y hora en la que esta publicación fue creada.
- `categoria`: Categoría a la que pertenece la publicación.
    - Tipo: `String`
    - Descripción: Indica la categoría a la que pertenece esta publicación (por ejemplo, "Postre", "Principal", etc.).
- `receta`: Nombre o descripción breve de la receta asociada a la publicación.
    - Tipo: `String`
    - Descripción: Proporciona el nombre o una descripción breve de la receta relacionada con esta publicación.
- `comentarios`: Lista de comentarios realizados en la publicación.
    - Tipo: `List<Comentario>`
    - Descripción: Contiene los comentarios asociados a esta publicación en forma de objetos `Comentario`.
- `likes`: Lista de likes o me gusta recibidos por la publicación.
    - Tipo: `List<PostLike>`
    - Descripción: Representa los likes recibidos por esta publicación en forma de objetos `PostLike`.
- `reportes`: Lista de reportes o denuncias realizadas sobre la publicación.
    - Tipo: `List<Reporte>`
    - Descripción: Contiene los reportes o denuncias asociadas a esta publicación en forma de objetos `Reporte`.

Constructor:

- `Post(id: Int, nombre: String, descripcion: String, usuarioUid: String, usuarioNombre: String, imagenes: List<ImagenPost>, fechaDeCreacion: String, categoria: String, receta: String, comentarios: List<Comentario>, likes: List<PostLike>, reportes: List<Reporte>)`
    - **Descripción:** Constructor primario que inicializa todos los atributos de la clase `Post`.

PostResponse

Descripción:

`PostResponse` es una clase de datos que encapsula la respuesta de una solicitud que devuelve una lista de objetos `Post`.

Atributos:

- `content`: Lista de objetos `Post`.
    - Tipo: `List<Post>`
    - Descripción: Contiene la lista de publicaciones (`Post`) que se devuelve como parte de la respuesta.

Constructor:

- `PostResponse(content: List<Post>)`
    - **Descripción:** Constructor primario que inicializa la lista de publicaciones (`content`) dentro del objeto `PostResponse`.

Receta

Descripción:

`Receta` es una clase de datos que representa una receta con detalles básicos.

Atributos:

- `idMeal`: Identificador único de la receta.

    - Tipo: `String`
    - Descripción: Este atributo almacena el identificador único de la receta.
- `strMeal`: Nombre de la receta.
    - Tipo: `String`
    - Descripción: Este atributo almacena el nombre o título de la receta.
- `strMealThumb`: URL de la imagen de la receta.
    - Tipo: `String`
    - Descripción: Este atributo almacena la URL de la imagen que representa visualmente la receta.

Constructor:

- `Receta(idMeal: String, strMeal: String, strMealThumb: String)`
    - **Descripción:** Constructor primario que inicializa los atributos de la receta con los valores proporcionados.

RecetaCompleta

Descripción:

`RecetaCompleta` es una clase que representa una receta detallada con múltiples atributos relacionados con los ingredientes, instrucciones de preparación y metadatos adicionales.

Atributos:

- `idMeal`: Identificador único de la receta.
    - Tipo: `String`
    - Descripción: Este atributo almacena el identificador único de la receta.
- `strMeal`: Nombre de la receta.
    - Tipo: `String`
    - Descripción: Este atributo almacena el nombre o título de la receta.
- `strDrinkAlternate`: Alternativa de bebida.
    - Tipo: `String?`
    - Descripción: Este atributo puede almacenar una alternativa de bebida asociada con la receta.
- `strCategory`: Categoría de la receta.
    - Tipo: `String?`

    - Descripción: Este atributo puede almacenar la categoría a la que pertenece la receta.
- `strArea`: Área de origen de la receta.
    - Tipo: `String?`
    - Descripción: Este atributo puede almacenar el área geográfica o cultural de donde proviene la receta.
- `strInstructions`: Instrucciones de preparación.
    - Tipo: `String?`
    - Descripción: Este atributo almacena las instrucciones detalladas para preparar la receta.
- `strMealThumb`: URL de la imagen de la receta.
    - Tipo: `String`
    - Descripción: Este atributo almacena la URL de la imagen que representa visualmente la receta.
- `strTags`: Etiquetas relacionadas con la receta.
    - Tipo: `String?`
    - Descripción: Este atributo puede almacenar etiquetas o palabras clave asociadas con la receta.
- `strYoutube`: Enlace al video de YouTube relacionado con la receta.
    - Tipo: `String?`
    - Descripción: Este atributo almacena el enlace al video de YouTube que muestra la preparación de la receta.
- `strIngredient1` a `strIngredient20`: Ingredientes de la receta.
    - Tipo: `String?`
    - Descripción: Estos atributos almacenan los nombres de los ingredientes utilizados en la receta. Pueden haber hasta 20 ingredientes.
- `strMeasure1` a `strMeasure20`: Medidas de los ingredientes.
    - Tipo: `String?`
    - Descripción: Estos atributos almacenan las medidas o cantidades de los ingredientes correspondientes. Coinciden con los ingredientes por posición.
- `strSource`: Fuente de la receta.
    - Tipo: `String?`
    - Descripción: Este atributo puede almacenar la fuente de donde se obtuvo la receta.
- `strImageSource`: Fuente de la imagen de la receta.
    - Tipo: `String?`
    - Descripción: Este atributo almacena la fuente de la imagen de la receta.

- `strCreativeCommonsConfirmed`: Confirmación de Creative Commons.
    - Tipo: `String?`
    - Descripción: Este atributo puede almacenar información sobre la confirmación del uso de licencia Creative Commons.
- `dateModified`: Fecha de modificación de la receta.
    - Tipo: `String?`
    - Descripción: Este atributo almacena la fecha en la que se modificó por última vez la receta.

Constructor:

- `RecetaCompleta(idMeal: String, strMeal: String, ...)`
    - **Descripción:** Constructor primario que inicializa todos los atributos de la receta con los valores proporcionados.

Reporte

Descripción:

`Reporte` es una clase que encapsula la información de un reporte hecho por un usuario sobre un post específico en la aplicación.

Atributos:

- `id`: Identificador único del reporte.
    - Tipo: `Long?`
    - Descripción: Este atributo almacena el ID único del reporte, que puede ser `null` si aún no se ha asignado.
- `post`: Post reportado.
    - Tipo: `Post?`
    - Descripción: Este atributo representa el post al que está asociado el reporte. Puede ser `null` si no se ha especificado un post específico o si se utiliza un objeto `Post` incompleto.
- `usuarioNombre`: Nombre del usuario que realizó el reporte.
    - Tipo: `String?`
    - Descripción: Este atributo almacena el nombre del usuario que hizo el reporte. Puede ser `null` si no se ha proporcionado o se desconoce.
- `contenido`: Contenido del reporte.
    - Tipo: `String?`

    - Descripción: Este atributo almacena el contenido o la razón específica del reporte hecho por el usuario. Puede ser `null` si no se ha especificado.

Constructor:

- `Reporte(id: Long?, post: Post?, usuarioNombre: String?, contenido: String?)`
    - **Descripción:** Constructor primario que inicializa todos los atributos del reporte con los valores proporcionados.

Temporada

Descripción:

`Temporada` es una clase que encapsula la información de una categoría de temporada en la aplicación Cookify.

Atributos:

- `idCategory`: Identificador único de la categoría.
    - Tipo: `String`
    - Descripción: Este atributo almacena el ID único de la categoría de temporada.
- `strCategory`: Nombre de la categoría.
    - Tipo: `String`
    - Descripción: Este atributo representa el nombre de la categoría de temporada.
- `strCategoryThumb`: URL de la imagen de la categoría.
    - Tipo: `String`
    - Descripción: Este atributo almacena la URL de la imagen asociada a la categoría de temporada.
- `strCategoryDescription`: Descripción de la categoría.
    - Tipo: `String`
    - Descripción: Este atributo proporciona una descripción detallada de la categoría de temporada.

Implementación de `Serializable`:

La clase implementa la interfaz `Serializable`, lo que permite que los objetos de esta clase puedan ser serializados y enviados a través de la red o almacenados en archivos.

Constructor:

- `Temporada(idCategory: String, strCategory: String, strCategoryThumb: String, strCategoryDescription: String)`
    - **Descripción:** Constructor primario que inicializa todos los atributos de la temporada con los valores proporcionados.

Usuario

Descripción:

`Usuario` es una clase que encapsula la información básica de un usuario registrado en la aplicación Cookify.

Atributos:

- `id`: Identificador único del usuario.
    - Tipo: `Long`
    - Descripción: Este atributo almacena el ID único asignado al usuario en la base de datos.
- `firebaseUid`: UID del usuario en Firebase.
    - Tipo: `String`
    - Descripción: Este atributo contiene el identificador único del usuario en la plataforma Firebase.
- `email`: Correo electrónico del usuario.
    - Tipo: `String`
    - Descripción: Este atributo almacena la dirección de correo electrónico asociada al usuario.
- `name`: Nombre del usuario.
    - Tipo: `String`
    - Descripción: Este atributo representa el nombre completo o el nombre de usuario del usuario registrado.
- `photoUrl`: URL de la foto de perfil del usuario.

    - Tipo: `String`
    - Descripción: Este atributo contiene la URL de la imagen utilizada como foto de perfil del usuario.
- `description`: Descripción del usuario.
    - Tipo: `String`
    - Descripción: Este atributo proporciona información adicional o una breve descripción del usuario.
- `privacy`: Configuración de privacidad del usuario.
    - Tipo: `Boolean`
    - Descripción: Este atributo indica la configuración de privacidad del usuario, por ejemplo, si su perfil es privado o público.

Implementación de `Serializable`:

La clase implementa la interfaz `Serializable`, lo que permite que los objetos de esta clase puedan ser serializados y enviados a través de la red o almacenados en archivos.

Constructor:

- `Usuario(id: Long, firebaseUid: String, email: String, name: String, photoUrl: String, description: String, privacy: Boolean)`
    - **Descripción:** Constructor primario que inicializa todos los atributos del usuario con los valores proporcionados.

RETROFIT

Interfaz `PostApi`

Métodos:

1. `createPost`
    - **Descripción:** Crea una nueva publicación.
    - **Tipo de solicitud HTTP:** POST
    - **Ruta del Endpoint:** `/api/posts`
    - **Parámetros:**

        - `token`: Encabezado de autorización para autenticación.
        - `post`: Cuerpo de la solicitud que contiene la nueva publicación (`Post`).
    - **Respuesta:** `Call<Post>`: Llamada asíncrona que devuelve un objeto `Post` cuando se completa la solicitud.
2. **`getUserPosts`**
    - **Descripción:** Obtiene las publicaciones de un usuario específico.
    - **Tipo de solicitud HTTP:** GET
    - **Ruta del Endpoint:** `/api/posts/user`
    - **Parámetros:**
        - `token`: Encabezado de autorización para autenticación.
    - **Respuesta:** `Call<List<Post>>`: Llamada asíncrona que devuelve una lista de `Post` cuando se completa la solicitud.
3. **`getAllPosts`**
    - **Descripción:** Obtiene todas las publicaciones disponibles paginadas.
    - **Tipo de solicitud HTTP:** GET
    - **Ruta del Endpoint:** `/api/posts`
    - **Parámetros:**
        - `page`: Número de página para la paginación.
        - `size`: Tamaño de la página para la paginación.
    - **Respuesta:** `Call<PostResponse>`: Llamada asíncrona que devuelve un objeto `PostResponse` cuando se completa la solicitud. `PostResponse` contiene una lista de `Post`.
4. **`deletePost`**
    - **Descripción:** Elimina una publicación específica por su ID.
    - **Tipo de solicitud HTTP:** DELETE
    - **Ruta del Endpoint:** `/api/posts/{postId}`
    - **Parámetros:**
        - `postId`: ID de la publicación que se va a eliminar.
        - `token`: Encabezado de autorización para autenticación.
    - **Respuesta:** `Call<Void>`: Llamada asíncrona que no devuelve datos cuando se completa la solicitud.
5. **`getUserPosts`**
    - **Descripción:** Obtiene las publicaciones de un usuario específico por su nombre.
    - **Tipo de solicitud HTTP:** GET
    - **Ruta del Endpoint:** `/api/posts/user/{nombre}`
    - **Parámetros:**
        - `token`: Encabezado de autorización para autenticación.

        - `nombre`: Nombre del usuario del cual se obtienen las publicaciones.
    - **Respuesta:** `Call<List<Post>>`: Llamada asíncrona que devuelve una lista de `Post` cuando se completa la solicitud.
6. **`getPostById`**
    - **Descripción:** Obtiene una publicación específica por su ID.
    - **Tipo de solicitud HTTP:** GET
    - **Ruta del Endpoint:** `/api/posts/{postId}`
    - **Parámetros:**
        - `token`: Encabezado de autorización para autenticación.
        - `postId`: ID de la publicación que se va a obtener.
    - **Respuesta:** `Call<Post>`: Llamada asíncrona que devuelve un objeto `Post` cuando se completa la solicitud.
7. **`addComentarioToPost`**
    - **Descripción:** Agrega un comentario a una publicación específica.
    - **Tipo de solicitud HTTP:** POST
    - **Ruta del Endpoint:** `/api/posts/{postId}/comentarios`
    - **Parámetros:**
        - `postId`: ID de la publicación a la cual se va a agregar el comentario.
        - `token`: Encabezado de autorización para autenticación.
        - `comentario`: Cuerpo de la solicitud que contiene el nuevo comentario (`Comentario`).
    - **Respuesta:** `Call<Void>`: Llamada asíncrona que no devuelve datos cuando se completa la solicitud.
8. **`getComentariosByPostId`**
    - **Descripción:** Obtiene todos los comentarios asociados a una publicación específica.
    - **Tipo de solicitud HTTP:** GET
    - **Ruta del Endpoint:** `/api/posts/{postId}/comentarios`
    - **Parámetros:**
        - `postId`: ID de la publicación de la cual se obtienen los comentarios.
    - **Respuesta:** `Call<List<Comentario>>`: Llamada asíncrona que devuelve una lista de `Comentario` cuando se completa la solicitud.
9. **`addDenunciaToPost`**
    - **Descripción:** Agrega un reporte (denuncia) a una publicación específica.
    - **Tipo de solicitud HTTP:** POST
    - **Ruta del Endpoint:** `/api/posts/{postId}/denuncias`

- **Parámetros:**
  - `token`: Encabezado de autorización para autenticación.
  - `postId`: ID de la publicación a la cual se va a agregar el reporte.
  - `denuncia`: Cuerpo de la solicitud que contiene el reporte (`Reporte`).
- **Respuesta:** `Call<Void>`: Llamada asíncrona que no devuelve datos cuando se completa la solicitud.

Interfaz `RecetaApi`

Métodos:

1. **`getTemporadas`**
   - **Descripción:** Obtiene las categorías de temporadas disponibles.
   - **Tipo de solicitud HTTP:** GET
   - **Ruta del Endpoint:** `categories.php`
   - **Respuesta:** `Call<ListaCategorias>`: Llamada asíncrona que devuelve una lista de `Temporada` encapsulada en `ListaCategorias` cuando se completa la solicitud.
2. **`getRecetasPorCategoria`**
   - **Descripción:** Obtiene las recetas filtradas por una categoría específica.
   - **Tipo de solicitud HTTP:** GET
   - **Ruta del Endpoint:** `filter.php`
   - **Parámetros:**
     - `c`: Nombre de la categoría por la cual filtrar las recetas.
   - **Respuesta:** `Call<ListaRecetas>`: Llamada asíncrona que devuelve una lista de `Receta` encapsulada en `ListaRecetas` cuando se completa la solicitud.
3. **`getRecetaPorId`**
   - **Descripción:** Obtiene una receta específica por su ID.
   - **Tipo de solicitud HTTP:** GET
   - **Ruta del Endpoint:** `lookup.php`
   - **Parámetros:**
     - `i`: ID de la receta que se va a obtener.
   - **Respuesta:** `Call<ListaRecetaCompleta>`: Llamada asíncrona que devuelve una lista de `RecetaCompleta` encapsulada en `ListaRecetaCompleta` cuando se completa la solicitud.

Objeto Singleton `RetrofitClient`

Propiedades y Métodos:

1. `BASE_URL`
   - **Descripción:** URL base del servidor al que se realizarán las solicitudes.
   - **Valor:** `"https://coockify-docker-5-0-1.onrender.com/"`
2. `logger`
   - **Descripción:** Interceptor de logging HTTP para Retrofit, configurado para mostrar todos los logs del cuerpo de la solicitud y respuesta.
   - **Configuración:** Nivel de log establecido en `HttpLoggingInterceptor.Level.BODY`.
3. `client`
   - **Descripción:** Cliente OkHttp configurado con el interceptor de logging y tiempos de espera para conexiones, lecturas y escrituras.
   - **Configuración:** Conexión, lectura y escritura con un tiempo de espera de 30 segundos cada uno.
4. `retrofit`
   - **Descripción:** Objeto Retrofit que se encarga de construir y configurar la instancia de Retrofit.
   - **Configuración:**
     - URL base (`BASE_URL`).
     - Convertidor de JSON (`GsonConverterFactory`) para convertir JSON a objetos Kotlin.
     - Cliente OkHttp configurado (`client`).
5. `usuarioApi`
   - **Descripción:** Instancia del servicio `UsuarioApi` creada de forma perezosa (lazy) utilizando Retrofit.
   - **Uso:** Permite realizar solicitudes HTTP relacionadas con usuarios utilizando métodos definidos en `UsuarioApi`.
6. `postApi`
   - **Descripción:** Instancia del servicio `PostApi` creada de forma perezosa (lazy) utilizando Retrofit.
   - **Uso:** Permite realizar solicitudes HTTP relacionadas con publicaciones utilizando métodos definidos en `PostApi`.

Objeto Singleton `RetrofitClientTraductor`

Propiedades y Métodos:

1. `BASE_URL`
    - **Descripción:** URL base del servicio de traducción en línea (`https://libretranslate.de/`).
    - **Uso:** Especifica la dirección del servidor al que se realizarán las solicitudes HTTP para las traducciones.
2. `instance`
    - **Descripción:** Instancia única de Retrofit configurada para interactuar con el servicio de traducción.
    - **Configuración:**
        - Utiliza el patrón de inicialización perezosa (lazy initialization) para asegurar que solo se cree la instancia cuando sea necesaria.
        - Define la URL base (`BASE_URL`) para todas las solicitudes.
        - Agrega un convertidor Gson (`GsonConverterFactory.create()`) para manejar la conversión de JSON a objetos Kotlin.

Objeto Singleton `RetrofitExternalClient`

Propiedades y Métodos:

1. `retrofit`
    - **Descripción:** Instancia de Retrofit configurada para interactuar con el servicio externo que proporciona recetas de comidas (`https://www.themealdb.com/api/json/v1/1/`).
    - **Configuración:**
        - Utiliza el patrón de inicialización perezosa (lazy initialization) para asegurar que solo se cree la instancia cuando sea necesaria.
        - Define la URL base (`baseUrl`) para todas las solicitudes relacionadas con recetas de comidas.

        - Agrega un convertidor Gson (`GsonConverterFactory.create()`) para manejar la conversión de JSON a objetos Kotlin.

2. **`api`**
    - **Descripción:** Interfaz de API que define métodos para realizar solicitudes HTTP específicas al servicio externo de recetas de comidas.
    - **Configuración:**
        - Utiliza la instancia de Retrofit (`retrofit.create(RecetaApi::class.java)`) para crear una implementación de la interfaz `RecetaApi`, que contiene métodos para diferentes endpoints de la API de recetas.

Interfaz `UsuarioApi`

Métodos:

1. **`registerUser`**
    - **Tipo:** POST
    - **URL:** `api/usuarios/register`
    - **Descripción:** Registra un nuevo usuario.
    - **Parámetros:**
        - `token`: Encabezado de autorización para autenticación.
        - `request`: Objeto `Usuario` que contiene los datos del usuario a registrar.
    - **Respuesta:** Retorna un objeto `Usuario` representando el usuario registrado.

Ejemplo:
kotlin
Copiar código

```
@POST("api/usuarios/register")

fun registerUser(@Header("Authorization") token: String, @Body request: Usuario): Call<Usuario>
```

    -

2. **`getProfile`**
    - **Tipo:** GET
    - **URL:** `api/usuarios/profile`

- **Descripción:** Obtiene el perfil del usuario actual.
- **Parámetros:**
  - `token`: Encabezado de autorización para autenticación.
- **Respuesta:** Retorna un objeto `Usuario` representando el perfil del usuario.

Ejemplo:
kotlin
Copiar código

```
@GET("api/usuarios/profile")

fun getProfile(@Header("Authorization") token: String):
Call<Usuario>
```

3. **`updateProfile`**
   - **Tipo:** PUT
   - **URL:** `api/usuarios/profile`
   - **Descripción:** Actualiza el perfil del usuario actual.
   - **Parámetros:**
     - `token`: Encabezado de autorización para autenticación.
     - `request`: Objeto `Usuario` que contiene los datos actualizados del perfil.
   - **Respuesta:** Retorna un objeto `Usuario` representando el perfil actualizado del usuario.

Ejemplo:
kotlin
Copiar código

```
@PUT("api/usuarios/profile")

fun updateProfile(@Header("Authorization") token: String,
@Body request: Usuario): Call<Usuario>
```

4. **`searchUsers`**
   - **Tipo:** GET
   - **URL:** `api/usuarios/search/{name}`
   - **Descripción:** Busca usuarios por nombre.
   - **Parámetros:**
     - `name`: Nombre del usuario a buscar (se proporciona como parte de la URL).

- **Respuesta:** Retorna una lista de objetos `Usuario` que coinciden con el nombre buscado.

Ejemplo:
kotlin
Copiar código

```
@GET("api/usuarios/search/{name}")

fun searchUsers(@Path("name") name: String):
Call<List<Usuario>>
```

SERVICIOS

Funciones del `PostService`

1. **`getUserUid()`**
   - Retorna el UID del usuario actual autenticado mediante Firebase Authentication.
2. **`getToken(onTokenReceived: (String?) -> Unit, context: Context)`**
   - Obtiene el token de autenticación del usuario actual mediante Firebase Authentication.
   - Guarda el token en las preferencias compartidas (`TokenManagerService`).
   - Invoca el callback `onTokenReceived` con el token obtenido o `null` en caso de error.
3. **`crearPost(token: String, post: Post, onSuccess: (Post?) -> Unit, onFailure: (Throwable) -> Unit)`**
   - Crea una nueva publicación utilizando Retrofit.
   - Invoca el método `createPost` de `PostApi` con el token de autenticación y la publicación (`Post`) proporcionada.
   - En caso de éxito, invoca `onSuccess` con la publicación creada.
   - En caso de fallo, invoca `onFailure` con el error correspondiente.
4. **`postCreado(nombre: String, descripcion: String, usuarioUid: String, usuarioNombre: String, imagenes:`**

`List<ImagenPost>, fecha: String, categoria: String, receta: String): Post`

- Crea y retorna un objeto `Post` con los datos proporcionados.

5. **`getAllPosts(page: Int, size: Int, onSuccess: (List<Post>?) -> Unit, onFailure: (Throwable) -> Unit)`**
   - Obtiene todas las publicaciones paginadas utilizando Retrofit.
   - Invoca el método `getAllPosts` de `PostApi` con el número de página y tamaño de página especificados.
   - En caso de éxito, invoca `onSuccess` con la lista de publicaciones obtenida.
   - En caso de fallo, invoca `onFailure` con el error correspondiente.
6. **`getUserPosts(token: String, onSuccess: (List<Post>?) -> Unit, onFailure: (Throwable) -> Unit)`**
   - Obtiene las publicaciones del usuario actual utilizando Retrofit.
   - Invoca el método `getUserPosts` de `PostApi` con el token de autenticación.
   - En caso de éxito, invoca `onSuccess` con la lista de publicaciones del usuario.
   - En caso de fallo, invoca `onFailure` con el error correspondiente.
7. **`getUserPosts(token: String, nombre: String, onSuccess: (List<Post>?) -> Unit, onFailure: (Throwable) -> Unit)`**
   - Obtiene las publicaciones de un usuario específico utilizando Retrofit.
   - Invoca el método `getUserPosts` de `PostApi` con el token de autenticación y el nombre de usuario.
   - En caso de éxito, invoca `onSuccess` con la lista de publicaciones del usuario específico.
   - En caso de fallo, invoca `onFailure` con el error correspondiente.
8. **`deleteUserPost(token: String, postId: Long, onSuccess: () -> Unit, onFailure: (Throwable) -> Unit)`**
   - Elimina una publicación utilizando Retrofit.
   - Invoca el método `deletePost` de `PostApi` con el token de autenticación y el ID de la publicación.
   - En caso de éxito, invoca `onSuccess`.
   - En caso de fallo, invoca `onFailure` con el error correspondiente.
9. **`getPostById(token: String, postId: Long, onSuccess: (Post?) -> Unit, onFailure: (Throwable) -> Unit)`**
   - Obtiene una publicación por su ID utilizando Retrofit.
   - Invoca el método `getPostById` de `PostApi` con el token de autenticación y el ID de la publicación.
   - En caso de éxito, invoca `onSuccess` con la publicación obtenida.

    - En caso de fallo, invoca `onFailure` con el error correspondiente.
10. **`addComentarioToPost(postId: Long, token: String, comentario: Comentario, onSuccess: () -> Unit, onFailure: (Throwable) -> Unit)`**
    - Agrega un comentario a una publicación utilizando Retrofit.
    - Invoca el método `addComentarioToPost` de `PostApi` con el ID de la publicación, el token de autenticación y el comentario (`Comentario`) proporcionado.
    - En caso de éxito, invoca `onSuccess`.
    - En caso de fallo, invoca `onFailure` con el error correspondiente.
11. **`getComentariosByPostId(postId: Long, onSuccess: (List<Comentario>?) -> Unit, onFailure: (Throwable) -> Unit)`**
    - Obtiene comentarios de una publicación por su ID utilizando Retrofit.
    - Invoca el método `getComentariosByPostId` de `PostApi` con el ID de la publicación.
    - En caso de éxito, invoca `onSuccess` con la lista de comentarios obtenida.
    - En caso de fallo, invoca `onFailure` con el error correspondiente.
12. **`enviarDenuncia(token: String, postId: Long, denuncia: Reporte, onSuccess: () -> Unit, onFailure: (String) -> Unit)`**
    - Envía una denuncia sobre una publicación utilizando Retrofit.
    - Invoca el método `addDenunciaToPost` de `PostApi` con el token de autenticación, el ID de la publicación y la denuncia (`Reporte`) proporcionados.
    - En caso de éxito, invoca `onSuccess`.
    - En caso de fallo, invoca `onFailure` con el mensaje de error correspondiente.

Funciones del `ServicioBase64`

1. **`base64ToBitmap(base64String: String): Bitmap?`**
    - Convierte una cadena Base64 en un objeto `Bitmap`.
    - Decodifica la cadena Base64 a un array de bytes.
    - Utiliza `BitmapFactory.decodeByteArray` para obtener el `Bitmap` a partir de los bytes decodificados.

    - Maneja excepciones de `IllegalArgumentException` devolviendo `null` en caso de error.
2. **`bitmapToBase64(bitmap: Bitmap): String`**
    - Convierte un objeto `Bitmap` en una cadena Base64.
    - Utiliza `ByteArrayOutputStream` para comprimir el `Bitmap` en formato PNG.
    - Convierte el array de bytes resultante a una cadena Base64 utilizando `Base64.encodeToString`.
3. **`getCompressedBitmapFromUri(contexto: Context, uri: Uri): Bitmap`**
    - Obtiene un objeto `Bitmap` comprimido desde una `Uri` de una imagen.
    - Utiliza el `ContentResolver` para abrir un `InputStream` desde la `Uri`.
    - Utiliza `BitmapFactory.Options` para especificar opciones de compresión (`inSampleSize` reduce el tamaño del bitmap).
    - Decodifica el `InputStream` en un objeto `Bitmap` usando las opciones especificadas.
    - Cierra el `InputStream` después de la decodificación y retorna el `Bitmap` resultante.
4. **`resizeBitmap(bitmap: Bitmap, newWidth: Int, newHeight: Int): Bitmap`**
    - Redimensiona un objeto `Bitmap` a nuevas dimensiones especificadas.
    - Utiliza `Bitmap.createScaledBitmap` para redimensionar el bitmap al nuevo ancho y alto.
    - Preserva la relación de aspecto del bitmap estableciendo el parámetro `filter` como `true`.

`ServicioRegistro`

1. **`enviarDatosAlBackend`**
   Este método está diseñado para enviar los datos de un nuevo usuario al backend de Cookify para su registro.
    - **Parámetros:**
        - `token`: Token de autenticación necesario para realizar la solicitud al backend.
        - `usuario`: Objeto `Usuario` que contiene la información del nuevo usuario a registrar.

        - `destino`: Contexto de la actividad o servicio desde donde se llama a este método.
        - `currentUser`: Objeto `FirebaseUser` que representa al usuario actualmente autenticado en Firebase.
    - **Funcionamiento:**
        - Se obtiene la instancia de la interfaz `UsuarioApi` utilizando `RetrofitClient.usuarioApi`.
        - Se actualiza la URL de la foto del usuario usando `ImagenDefectoCodificada.getImagenDefecto()` antes de enviarla al backend.
        - Se crea una llamada (`Call<Usuario>`) usando `registerUser` de `UsuarioApi` con el token de autenticación y el objeto `usuario`.
        - Se realiza la solicitud asíncrona utilizando `enqueue` para manejar las respuestas asíncronamente.
        - En el `onResponse`, si la solicitud es exitosa (`response.isSuccessful`), se redirige al usuario a la pantalla de inicio de sesión (`LoginActivity`) usando un nuevo intent con `startActivity`.
        - En caso de fallo (`onFailure` o respuesta no exitosa), se elimina el usuario actual (`currentUser.delete()`) para manejar la situación de error.

`ServicioTemporada`

1. **`getTemporada`**
   Este método obtiene la lista de temporadas disponibles.
    - **Parámetros:**
        - `onSuccess`: Función de callback que se llama cuando se obtiene la lista de temporadas exitosamente.
    - **Funcionamiento:**
        - Se realiza una solicitud GET a través de Retrofit utilizando `RetrofitExternalClient.api.getTemporadas()`.
        - Se utiliza `enqueue` para manejar la respuesta de manera asíncrona.
        - En `onResponse`, si la respuesta es exitosa (`response.isSuccessful`), se obtiene la lista de temporadas

(`temps`) del cuerpo de la respuesta y se llama a `onSuccess` con esta lista.
        - En caso de fallo (`onFailure`), se imprime el stack trace del error (`t.printStackTrace()`).
2. **`getRecetasPorCategoria`**
   Este método obtiene la lista de recetas según una categoría específica.
    - **Parámetros:**
        - `categoria`: Nombre de la categoría para filtrar las recetas.
        - `onSuccess`: Función de callback que se llama cuando se obtiene la lista de recetas exitosamente.
    - **Funcionamiento:**
        - Se realiza una solicitud GET a través de Retrofit utilizando `RetrofitExternalClient.api.getRecetasPorCategoria(categoria)`.
        - Se utiliza `enqueue` para manejar la respuesta de manera asíncrona.
        - En `onResponse`, si la respuesta es exitosa (`response.isSuccessful`), se obtiene la lista de recetas (`listaRecetas`) del cuerpo de la respuesta y se llama a `onSuccess` con esta lista.
        - En caso de fallo (`onFailure`), se imprime el stack trace del error (`t.printStackTrace()`).
3. **`getRecetaPorId`**
   Este método obtiene los detalles completos de una receta específica según su ID.
    - **Parámetros:**
        - `id`: ID de la receta que se desea obtener.
        - `onSuccess`: Función de callback que se llama cuando se obtiene la receta exitosamente.
    - **Funcionamiento:**
        - Se realiza una solicitud GET a través de Retrofit utilizando `RetrofitExternalClient.api.getRecetaPorId(id)`.
        - Se utiliza `enqueue` para manejar la respuesta de manera asíncrona.
        - En `onResponse`, si la respuesta es exitosa (`response.isSuccessful`), se obtiene la receta (`listaRecetas`) del cuerpo de la respuesta y se llama a `onSuccess` con esta receta.
        - En caso de fallo (`onFailure`), se imprime el stack trace del error (`t.printStackTrace()`).

`TokenManagerService`

1. **Declaración de Constantes**
   - `PREF_NAME`: Nombre del archivo de preferencias compartidas donde se almacenará el token del usuario.
   - `KEY_USER_TOKEN`: Clave utilizada para almacenar y recuperar el token de usuario dentro de las preferencias compartidas.
2. **Inicialización de SharedPreferences**
   - `sharedPreferences`: Objeto de SharedPreferences inicializado en el contexto proporcionado (`Context.MODE_PRIVATE`), lo que significa que solo la aplicación puede acceder a estas preferencias.
3. **Método `saveUserToken`**
   - **Propósito:** Guarda el token de usuario en las preferencias compartidas.
   - **Parámetros:**
     - `token`: Token de usuario que se va a guardar.
   - **Funcionamiento:**
     - Utiliza el método `with(sharedPreferences.edit())` para iniciar una transacción de edición en las preferencias compartidas.
     - Llama a `putString(KEY_USER_TOKEN, token)` para almacenar el token bajo la clave `KEY_USER_TOKEN`.
     - Llama a `apply()` para aplicar los cambios de manera asincrónica y asegurar que se guarden correctamente.
4. **Método `getUserToken`**
   - **Propósito:** Obtiene el token de usuario almacenado previamente.
   - **Retorno:**
     - Devuelve el token de usuario como una cadena (`String`) o `null` si no se ha guardado ningún token.
5. **Método `clearUserToken`**
   - **Propósito:** Elimina el token de usuario almacenado.
   - **Funcionamiento:**
     - Utiliza el método `with(sharedPreferences.edit())` para iniciar una transacción de edición en las preferencias compartidas.
     - Llama a `remove(KEY_USER_TOKEN)` para eliminar el token almacenado bajo la clave `KEY_USER_TOKEN`.

        - Llama a `apply()` para aplicar los cambios de manera asincrónica y asegurar que se elimine correctamente el token.
6. **Método `hasUserToken`**
    - **Propósito:** Verifica si hay un token de usuario almacenado.
    - **Retorno:**
        - Devuelve `true` si existe un token de usuario almacenado en las preferencias compartidas, de lo contrario devuelve `false`.

`Traductor` y `TranslateResponse`

1. **`Traductor`**
    - `data class Traductor`: Define los datos necesarios para realizar una solicitud de traducción.
        - `q`: Cadena de texto que se va a traducir.
        - `source`: Idioma de origen del texto.
        - `target`: Idioma al cual se va a traducir el texto.
        - `format`: Formato del texto (por defecto es "text").
2. **`TranslateResponse`**
    - `data class TranslateResponse`: Define la estructura de la respuesta recibida después de realizar una solicitud de traducción.
        - `translatedText`: Texto traducido obtenido como respuesta.

`LibreTranslateService`

1. **Interfaz `LibreTranslateService`**
    - `@Headers("Content-Type: application/json")`: Encabezado de la solicitud HTTP que especifica el tipo de contenido como JSON.
    - `@POST("translate")`: Anotación que indica que se utilizará el método POST para enviar la solicitud al endpoint `/translate` del servicio LibreTranslate.
    - `fun translate(@Body request: Traductor): Call<TranslateResponse>`: Método que define la solicitud de traducción.
        - `@Body request: Traductor`: Parámetro que especifica el cuerpo de la solicitud como un objeto `Traductor`.
        - `Call<TranslateResponse>`: Devuelve un objeto `Call` que representa la solicitud HTTP asíncrona y la respuesta esperada es un objeto `TranslateResponse`.

`UserService` y sus métodos:

1. **`getToken`**
   - Método estático para obtener el token de autenticación del usuario actual.
   - Utiliza Firebase Authentication (`FirebaseAuth`) para obtener el token de usuario.
   - Guarda el token en las preferencias compartidas usando `TokenManagerService`.
   - Llama a `onTokenReceived` con el token obtenido o `null` si hay un error.
2. **`cargarPerfil`**
   - Método para obtener el perfil de usuario desde el servidor.
   - Hace una solicitud GET al endpoint `api/usuarios/profile` usando Retrofit y el token de autenticación.
   - Llama a `onSuccess` con el objeto `Usuario` obtenido si la solicitud es exitosa.
   - Llama a `onFailure` si hay algún error durante la solicitud.
3. **`usuarioModificado`**
   - Crea y devuelve un objeto `Usuario` con los datos modificados, incluyendo una imagen codificada en base64.
   - Es útil para preparar los datos del usuario antes de enviarlos al servidor para actualizar el perfil.
4. **`editarPerfil`**
   - Método para enviar una solicitud PUT para actualizar el perfil de usuario en el servidor.
   - Usa Retrofit para llamar al endpoint `api/usuarios/profile` con el token de autenticación y el objeto `Usuario` modificado.
   - Llama a `onSuccess` con el objeto `Usuario` actualizado si la solicitud es exitosa.
   - Llama a `onFailure` si hay algún error durante la solicitud.
5. **`searchUsers`**
   - Método para buscar usuarios por nombre en el servidor.
   - Hace una solicitud GET al endpoint `api/usuarios/search/{name}` usando Retrofit y el nombre proporcionado.
   - Llama a `onSuccess` con la lista de usuarios obtenidos si la solicitud es exitosa.
   - Llama a `onFailure` si hay algún error durante la solicitud.

(UI) BUSCADOR

Descripción general

El `BuscarFragment` es un fragmento de Android que muestra un campo de búsqueda para encontrar usuarios, un RecyclerView para listar los resultados de búsqueda y elementos visuales para indicar el estado de la búsqueda.

Funcionalidad clave

1. **Inicialización y Configuración Inicial**
    - `onCreateView`: Este método infla el diseño del fragmento (`FragmentBuscarBinding`), inicializa vistas como `RecyclerView`, `ProgressBar`, `TextView`, etc., y configura un `TextWatcher` para realizar la búsqueda después de un retardo de 1 segundo cuando se escribe en el campo de búsqueda.
    - `mostrarBuscador`: Método auxiliar para mostrar elementos visuales de búsqueda cuando se inicia una búsqueda.
2. **Búsqueda de Usuarios**
    - `cargarMiUsuario`: Utiliza `UserService.getToken` para obtener el token de autenticación del usuario actual y luego llama a `UserService.cargarPerfil` para cargar el perfil del usuario actual.
    - `cargarUsuarios`: Utiliza `UserService.searchUsers` para buscar usuarios por nombre según el texto ingresado en el campo de búsqueda. Filtra los usuarios permitidos (sin privacidad y diferente al usuario actual) y actualiza el `RecyclerView` con los resultados.
3. **Interfaz de Usuario**
    - `binding.searchEditText`, `binding.closeIcon`, `binding.searchIcon`: Elementos visuales para ingresar texto de búsqueda y controlar la interacción del usuario.
    - `binding.rvUsuarios`: RecyclerView para mostrar la lista de usuarios encontrados.
    - `binding.progressBar`, `binding.tvNoResultados`, `binding.ivNoResultados`: Elementos visuales para mostrar el estado de la carga y mensajes cuando no hay resultados.
4. **Manejo de Ciclo de Vida**

- `onDestroyView`: Limpia las referencias de vistas y elimina las llamadas pendientes al `searchRunnable` cuando se destruye la vista del fragmento.

`PerfilBuscadoActivity` es una actividad de Android que muestra el perfil de un usuario buscado, incluyendo su nombre, imagen de perfil redondeada y las publicaciones que ha realizado.

Funcionalidad Clave

1. **Configuración Inicial y UI**
   - `onCreate`: Este método configura la actividad al iniciarse. Se oculta la ActionBar, se establece el layout (`activity_perfil_buscado.xml`) y se ajustan los márgenes según las barras del sistema. También inicializa y configura las vistas como `TextView`, `ComponenteImagenRedondeada`, `RecyclerView`, `ImageView`, etc.
   - `ViewCompat.setOnApplyWindowInsetsListener`: Ajusta los márgenes del layout principal para evitar que las barras del sistema superpongan el contenido.
2. **Carga del Usuario Buscado**
   - `cargarUsuario`: Utiliza `UserService.searchUsers` para buscar un usuario por nombre. Configura la vista con el nombre del usuario y su imagen de perfil decodificada usando `ServicioBase64.base64ToBitmap`.
3. **Carga de Publicaciones del Usuario**
   - `cargarPostsUsuario`: Utiliza `PostService.getToken` para obtener el token de autenticación y `PostService.getUserPosts` para obtener las publicaciones del usuario buscado. Configura el RecyclerView (`recyclerViewPosts`) con un adaptador personalizado (`MisPostsAdapter`) que muestra las publicaciones.
4. **Interacción del Usuario**
   - `ivCerrarPerfil.setOnClickListener`: Cierra la actividad cuando el usuario presiona el botón de cerrar perfil (`ivCerrarPerfil`).
5. **Control de Carga**
   - `ControladorCarga.showLoading`: Métodos utilizados para mostrar y ocultar la barra de progreso (`pbCarga`) según el estado de la carga de datos.

(UI) CREAR

`CrearFragment`:

Descripción General

`CrearFragment` es un fragmento de Android que permite a los usuarios crear y publicar un nuevo post en la aplicación Cookify. Permite seleccionar múltiples imágenes, ingresar título y descripción, seleccionar categorías y subcategorías de comidas, y finalmente publicar el post.

Funcionalidad Clave

1. **Configuración Inicial y UI**
    - `onCreateView`: Este método infla y configura la vista del fragmento (`FragmentCrearBinding`). Configura los EditTexts para el título y descripción del post, ViewPager para mostrar las imágenes seleccionadas, Spinners para las categorías y subcategorías de comidas, y ImageView para abrir la cámara y cancelar la publicación.
2. **Selección de Imágenes**
    - `seleccionadorImagenes`: Utiliza `ActivityResultContracts.PickMultipleVisualMedia` para permitir al usuario seleccionar hasta 5 imágenes. Al seleccionar las imágenes, se actualiza el ViewPager con un adaptador personalizado (`CarruselCrearPostAdapter`) que muestra las imágenes seleccionadas.
3. **Creación y Publicación de Post**
    - `crearPublicacion`: Al hacer clic en el botón de publicar (`ivPublicar`), se obtiene el token de usuario usando `PostService.getToken`. Luego se llama a `PostService.crearPost` para enviar los datos del post al servidor, incluyendo título, descripción, imágenes, categoría y subcategoría seleccionadas.
4. **Rellenar Spinners Dinámicamente**
    - `rellenarSpinners`: Utiliza `ServicioTemporada.getTemporada` para obtener las categorías de temporada y configurar el Spinner (`spCategoria`). Dependiendo de la categoría seleccionada, habilita el Spinner de comidas (`spComida`) y lo llena dinámicamente con las comidas disponibles.

5. **Manejo de Errores**
    - `manejarErrores`: Valida que se haya ingresado un título, una descripción y se hayan seleccionado imágenes antes de permitir la publicación del post. Muestra un diálogo de error si alguna validación falla.
6. **Diálogo de Error**
    - `mostrarDialogoError`: Muestra un diálogo modal con un mensaje de error específico y un botón de confirmación.

(UI) INICIO

`DetalleCategoriaActivity` es una actividad de Android que muestra los detalles de una categoría de recetas específica. Utiliza `Glide` para cargar la imagen de la categoría y `RecyclerView` junto con `RecetaApiAdapter` para mostrar una lista de recetas pertenecientes a esa categoría.

Funcionalidad Clave

1. **Configuración Inicial y UI**
    - `onCreate`: Este método se llama al crear la actividad. Infla el diseño de la actividad (`R.layout.activity_detalle_categoria`), oculta la barra de acción (`supportActionBar?.hide()`), y obtiene la categoría seleccionada del intent utilizando `intent.getSerializableExtra("categoria") as Temporada`.
2. **Carga de Imagen y Nombre de la Categoría**
    - Utiliza `Glide` para cargar la imagen de la categoría (`categoria.strCategoryThumb`) en `ivImagen`.
    - Establece el nombre de la categoría (`categoria.strCategory`) en `tvNombre`.
3. **Configuración del RecyclerView**
    - `rvRecetasApi` se configura con un `LinearLayoutManager` para mostrar las recetas en una lista vertical.
    - `RecetaApiAdapter` se utiliza para adaptar y mostrar las recetas obtenidas a partir de `ServicioTemporada.getRecetasPorCategoria`.
4. **Manejo del Botón Cerrar**

- `ivCerrar.setOnClickListener`: Cierra la actividad cuando se hace clic en el icono de cierre (`ivCerrar`).

5. **Llamada al Servicio para Obtener Recetas**
   - `cargarRecetas()`: Utiliza `ServicioTemporada.getRecetasPorCategoria` para obtener las recetas específicas de la categoría seleccionada. Luego configura `recetaAdapter` con las recetas obtenidas y actualiza el `RecyclerView`.

`DetalleRecetaActivity` es una actividad de Android que muestra los detalles de una receta específica, incluyendo su nombre, imagen, lista de ingredientes, medidas, instrucciones de elaboración y un video de YouTube (si está disponible).

Funcionalidad Clave

1. **Configuración Inicial y UI**
   - `onCreate`: Este método se llama al crear la actividad. Infla el diseño de la actividad (`R.layout.activity_detalle_receta`), oculta la barra de acción (`supportActionBar?.hide()`), y configura el manejo de los márgenes según las barras del sistema utilizando `ViewCompat.setOnApplyWindowInsetsListener`.
2. **Carga de Datos de la Receta**
   - Utiliza `ServicioTemporada.getRecetaPorId` para obtener los detalles completos de la receta específica mediante su ID (`recetaId`).
   - Configura los diferentes `TextView` (`tvTituloReceta`, `tvElaboracion`, `tvI1` a `tvI20`, `tvC1` a `tvC20`) para mostrar el nombre de la receta, las instrucciones de elaboración, los ingredientes y sus medidas respectivas.
3. **Carga de Imagen de la Receta**
   - Utiliza Glide para cargar la imagen de la receta (`receta.strMealThumb`) en `ivReceta`.
4. **Configuración del Reproductor de YouTube**
   - Utiliza `YouTubePlayerView` para reproducir un video de YouTube asociado a la receta (`receta.strYoutube`). Si está disponible, extrae el ID del video y lo carga en el reproductor de YouTube.
5. **Manejo del Botón Cerrar**
   - `ivCerrarReceta.setOnClickListener`: Cierra la actividad cuando se hace clic en el icono de cierre (`ivCerrarReceta`).

El fragmento `InicioFragment` se encarga de gestionar la vista principal de la aplicación Cookify, donde se muestran las publicaciones de los usuarios y se permite la interacción con la cámara para tomar nuevas fotos.

Funcionalidades Implementadas

1. **Inicialización y Configuración de UI**
   - `onCreateView`: Se infla el diseño del fragmento (`FragmentInicioBinding`), se configuran los elementos visuales como el `SwipeRefreshLayout`, `RecyclerView`, `ProgressBar` y se registran los listeners para manejar el comportamiento de carga y refresco de datos.
2. **Carga de Publicaciones (`loadPosts`)**
   - Método que utiliza `PostService.getAllPosts` para obtener las publicaciones desde un servicio. Se maneja la paginación para cargar más publicaciones al llegar al final de la lista (`addOnScrollListener`).
3. **Refresco de Contenido (`refreshContent`)**
   - Simula la carga de datos nuevos cuando se realiza un gesto de deslizamiento hacia abajo en `SwipeRefreshLayout`.
4. **Acción de Tomar Foto (`takePhoto`)**
   - Utiliza `Intent(MediaStore.ACTION_IMAGE_CAPTURE)` para abrir la cámara del dispositivo y tomar una foto. Se manejan los permisos (`Manifest.permission.CAMERA`) y se utiliza `FileProvider` para generar una URI segura para la foto.
5. **Manejo de Resultados de Actividad (`getAction`)**
   - `ActivityResultContracts.StartActivityForResult` se utiliza para recibir el resultado de la actividad de la cámara (`intent`) y obtener la imagen capturada como un `Bitmap`.
6. **Gestión de Permisos (`onRequestPermissionsResult`)**
   - Método llamado cuando se solicitan permisos. Permite o deniega el acceso a la cámara y maneja el flujo de ejecución en consecuencia.

Explicación Detallada del Código

1. **Inicialización y Configuración de UI**
    - En `onCreate`, se configura la vista (`R.layout.activity_temporadas`) y se oculta la `ActionBar`.
    - Se configura un `ViewCompat.setOnApplyWindowInsetsListener` para ajustar el padding bajo la barra de sistema.
    - `ivCerrarTemporadas` es un `ImageView` utilizado para cerrar la actividad al hacer clic.
2. **Obtener Temporadas (`getTemporadas`)**
    - Utiliza `ServicioTemporada.getTemporada` para obtener una lista de temporadas (`temporadas.categories`).
    - Para cada temporada, se infla un `CardView` personalizado (`card_item.xml`) que contiene una imagen (`ImageView`) y un texto (`TextView`) para el nombre de la temporada.
    - Se utiliza Glide para cargar la imagen desde la URL (`temporada.strCategoryThumb`) en el `ImageView`.
    - Se llama a `translateText` para traducir el nombre de la temporada del inglés al español y establecer el texto en el `TextView`.
3. **Traducción de Texto (`translateText`)**
    - Utiliza Retrofit para realizar una solicitud HTTP a un servicio de traducción (`LibreTranslateService`).
    - Crea un objeto `Traductor` con el texto a traducir, el idioma de origen (`en` para inglés) y el idioma de destino (`es` para español).
    - En `onResponse`, si la solicitud es exitosa, establece el texto traducido en el `TextView`; de lo contrario, muestra el texto original.
    - En `onFailure`, establece el texto original en caso de error.
4. **Clic en Tarjeta (`cardView.setOnClickListener`)**
    - Al hacer clic en una tarjeta (`CardView`), se crea un intento para abrir `DetalleCategoriaActivity`, pasando la temporada seleccionada como un extra en el intento.
5. **Integración con Servicios y Bibliotecas**
    - Utiliza `ServicioTemporada` para obtener datos de temporadas.
    - Utiliza Glide para cargar imágenes desde URLs.
    - Utiliza Retrofit para realizar solicitudes HTTP a servicios de traducción.

(UI) PERFIL

Descripción Detallada del Código

1. **Inicialización de Variables**
    - Se inicializan todas las variables necesarias para manejar la interfaz de usuario y las operaciones relacionadas con la edición del perfil, como `base64` para almacenar la imagen en formato base64, `etEditarNombre` y `etEditarDescripcion` para los campos de nombre y descripción, `swEditarPrivacidad` para el interruptor de privacidad, `cpEditarImagen` para el componente de imagen redondeada, y otros elementos de UI como botones y vistas de carga.
2. **Configuración de la Interfaz de Usuario**
    - En `onCreate`, se configura la vista (`R.layout.activity_modificar_perfil`) y se oculta la barra de acción.
    - `ViewCompat.setOnApplyWindowInsetsListener` se utiliza para ajustar el padding bajo la barra de sistema.
    - Se inicializan todos los elementos de UI y se muestra una pantalla de carga utilizando `ControladorCarga.showLoading`.
3. **Refrescar Contenido (`refreshContent`)**
    - Se obtiene el token del usuario mediante `UserService.getToken`.
    - Con el token obtenido, se carga el perfil del usuario utilizando `UserService.cargarPerfil`. Si la operación es exitosa, se actualizan los campos de nombre, descripción, imagen y privacidad con los datos del usuario recuperados del servicio.
    - Si no se puede obtener el token o hay algún error en la operación, se muestra un mensaje de error.
4. **Seleccionador de Imágenes (`seleccionadorImagenes`)**
    - Se registra un `ActivityResultLauncher` para manejar la selección de imágenes utilizando `ActivityResultContracts.PickVisualMedia`.
    - Al seleccionar una imagen, se convierte en un `Bitmap`, se muestra en `cpEditarImagen` y se convierte a base64 utilizando `ServicioBase64`.
5. **Guardar Perfil (`guardarPerfil`)**
    - Al hacer clic en `btnGuardarPerfil`, se obtiene el token del usuario mediante `UserService.getToken`.

- Se utiliza `UserService.editarPerfil` para enviar la solicitud de edición del perfil, pasando la imagen en formato base64, el nombre, la descripción y la configuración de privacidad actualizados.
- Si la operación es exitosa, se finaliza la actividad y se muestra un mensaje de éxito; de lo contrario, se muestra un mensaje de error.

6. **Otros Métodos Utilitarios**
   - `getBitmapFromUri`: Convierte una `Uri` en un `Bitmap`.
   - `openImagePicker`: Abre el selector de imágenes utilizando `seleccionadorImagenes`.

Descripción Detallada del Código

1. **Inicialización de Variables y Elementos de UI**
   - Se inicializan todas las variables necesarias para manejar la interfaz de usuario y las operaciones relacionadas con el perfil del usuario. Esto incluye elementos como `base64` para la imagen en formato base64, `swipeRefreshLayout` para permitir la actualización mediante gestos de deslizamiento, y varios elementos de la interfaz como `imgRedondeada`, `etEditarNombre`, `ivPrivacidad`, etc.
   - Se configura el `RecyclerView` (`recyclerViewPosts`) con un `GridLayoutManager` de 3 columnas y se muestra una pantalla de carga inicial.
2. **Carga del Perfil (`cargarPerfil`)**
   - Se obtiene el token del usuario mediante `UserService.getToken`.
   - Se carga el perfil del usuario utilizando `UserService.cargarPerfil`. Si la operación es exitosa, se actualizan los elementos de la interfaz con los datos del usuario, incluida la imagen, nombre, descripción, número de publicaciones, etc.
   - Se cargan los posts del usuario utilizando `PostService.getUserPosts`. Si se obtienen los posts correctamente, se actualiza el adaptador del `RecyclerView` y se muestra el número de publicaciones; de lo contrario, se muestra un mensaje de error.
   - Se manejan errores mostrando mensajes adecuados y asegurándose de mostrar la barra de navegación inferior (`BottomNavigationView`) al finalizar la carga.
3. **Actualización de Contenido (`refreshContent`)**

- Cuando se realiza un gesto de deslizamiento hacia abajo en `swipeRefreshLayout`, se llama a este método para simular la actualización de contenido.
- Dentro de este método se llama nuevamente a `cargarPerfil` para actualizar los datos del perfil y de los posts.
- Se muestra un mensaje de "Contenido actualizado" y se detiene la animación de actualización.

4. **Mostrar Menú Emergente (`showPopupMenu`)**
   - Cuando se hace clic en `ivOpciones`, se muestra un menú emergente (`PopupMenu`) con dos opciones:
     - `option1`: Muestra un diálogo modal (`ComponenteModalSalir`) para confirmar si el usuario desea salir.
     - `option2`: Inicia la actividad `ModificarPerfilActivity` para permitir al usuario editar su perfil.
5. **Métodos Utilitarios**
   - `cargarImagen`: Carga una imagen en el componente de imagen redondeada a partir de una cadena base64.
   - `mensajeNoRecetas`: Muestra u oculta un mensaje indicando que el usuario no tiene recetas publicadas, dependiendo del número de recetas obtenidas.
6. **Ciclo de Vida**
   - Se asegura de limpiar y liberar recursos en `onDestroyView` al asignar `_binding` a `null`.

Descripción Detallada del Código

1. **Inicialización de Variables y Elementos de UI**
   - Se inicializan los elementos de la interfaz de usuario (`ImageView`, `TextView`, `ViewPager2`, etc.) necesarios para mostrar los detalles del post, así como para manejar eventos como cerrar el post (`ivCerrarPost`), comentar (`ivComnentar`) y denunciar (`ivDenunciar`).
   - Se configura la barra de progreso (`ProgressBar`) para mostrar y ocultar la carga mientras se obtienen los datos del post.
2. **Configuración de Eventos**

- **Cerrar Post (`ivCerrarPost`)**: Cierra la actividad actual al hacer clic en el ícono de cerrar.
- **Comentar (`ivComnentar`)**: Muestra un `BottomSheetDialogFragment` (`ComponentesComentarios`) para que los usuarios puedan agregar comentarios al post.
- **Denunciar (`ivDenunciar`)**: Muestra un cuadro de diálogo (`AlertDialog`) para que los usuarios puedan denunciar el post como contenido inapropiado.

3. **Métodos de Funcionalidad**
   - **`denunciarPost(post: Post)`**: Este método permite al usuario denunciar un post. Al confirmar la denuncia en el `AlertDialog`, se crea un objeto `Reporte` y se envía al servicio correspondiente (`PostService.enviarDenuncia`) para su procesamiento.
   - **`getPost(postId: Long)`**: Este método obtiene el post específico usando su `postId`. Utiliza `PostService.getPostById` para obtener los detalles del post y luego actualiza los elementos de la UI con la información correspondiente (nombre de usuario, categoría, título, descripción, imágenes, etc.). También configura el adaptador del `ViewPager2` (`postViewPager`) con las imágenes del post y muestra la imagen del usuario asociado al post.
4. **Integración con Servicios**
   - Se utilizan los servicios (`PostService` y `UserService`) para obtener datos del post y del usuario asociado.
   - Se manejan los tokens de autenticación para asegurar el acceso a los datos y realizar acciones como enviar denuncias.

LOGIN

Descripción Detallada del Código

1. **Inicialización y Selección de Componentes UI**
   - Se inicializan los elementos de la interfaz de usuario (`EditText`, `Button`, `TextView`) necesarios para el inicio de sesión.
   - Se configura el evento de clic para los botones de iniciar sesión (`btnIniciarSesion`) y registro (`btnRegistro`), así como para la opción de olvidar contraseña (`tvOlvidaContrasena`).
2. **Configuración de Eventos**

- **Iniciar Sesión (`btnIniciarSesion`)**: Verifica las credenciales ingresadas por el usuario utilizando Firebase Authentication (`signInWithEmailAndPassword`). Si las credenciales son válidas, redirige al usuario a la actividad principal (`MainActivity`).
- **Registro (`btnRegistro`)**: Inicia una nueva actividad para permitir al usuario registrarse (`RegistroActivity`).
- **Recuperar Contraseña (`tvOlvidaContrasena`)**: Envía un correo electrónico de restablecimiento de contraseña utilizando `sendPasswordResetEmail`.

3. **Validación de Entradas y Manejo de Errores**
   - **`manejarErrores(email: String, password: String)`:** Realiza validaciones simples en las entradas de correo electrónico y contraseña para mostrar errores si no cumplen con los criterios especificados.
4. **Firebase Authentication**
   - Utiliza `FirebaseAuth` para manejar el proceso de autenticación y recuperación de contraseña.
5. **Funciones Auxiliares**
   - **`emailValido(email: String)`:** Valida el formato básico de una dirección de correo electrónico utilizando una expresión regular simple.
   - **`mostrarDialogoError(contenido: String, boton: String)`:** Muestra un diálogo modal para informar al usuario sobre errores o mensajes importantes.

REGISTRO

Descripción Detallada del Código

1. **Inicialización y Selección de Componentes UI**
   - Se inicializan los elementos de la interfaz de usuario (`EditText`, `Button`) necesarios para el registro.
   - Se configuran los eventos de clic para los botones de registrarse (`btnRegistrarse`) y cancelar (`btnCancelar`).
2. **Configuración de Eventos**
   - **Registrar Usuario (`btnRegistrarse`)**: Valida las entradas del usuario y, si son válidas, intenta crear un nuevo usuario utilizando Firebase Authentication (`createUserWithEmailAndPassword`). Luego,

actualiza el perfil del usuario con su nombre utilizando `updateProfile`.
    - Si el registro es exitoso, se obtiene el token de ID del usuario y se envían los datos del usuario al backend utilizando Retrofit (`ServicioRegistro.enviarDatosAlBackend`).
3. **Manejo de Errores**
    - Se manejan varios tipos de excepciones que pueden ocurrir durante el proceso de registro, como contraseñas débiles, credenciales inválidas o correo electrónico ya en uso.
4. **Funciones Auxiliares**
    - **`manejarErrores(email: String, password: String, repPassword: String, nombre: String)`:** Realiza validaciones simples en las entradas del usuario para mostrar errores si no cumplen con los criterios especificados.
    - **`emailValido(email: String)`:** Utiliza una expresión regular para validar el formato básico de una dirección de correo electrónico.
    - **`mostrarDialogoError(contenido: String, boton: String)`:** Muestra un diálogo modal para informar al usuario sobre errores o mensajes importantes.

MAIN

Descripción Detallada del Código

1. **Configuración Inicial**
    - Se oculta la barra de acción (`supportActionBar?.hide()`) para personalizar la interfaz de usuario.
2. **Inicialización de Componentes**
    - Se infla el layout usando `ActivityMainBinding` para obtener referencias a los elementos de la interfaz de usuario definidos en `activity_main.xml`.
    - Se inicializa `FirebaseApp` para configurar Firebase en la aplicación.
3. **Configuración de Navigation Component**
    - Se obtiene el `NavController` del `nav_host_fragment_activity_main` para gestionar la navegación entre fragmentos.
    - `AppBarConfiguration` se utiliza para configurar las top-level destinations del Navigation Component junto con la barra de acción.

    - `setupActionBarWithNavController` configura la barra de acción para que refleje la navegación actual.
    - `setupWithNavController` configura `BottomNavigationView` para que funcione con el `NavController`, permitiendo la navegación mediante el menú inferior.
4. **Funciones Auxiliares**
    - `showBottomNavigationView()` y `hideBottomNavigationView()`: Habilitan o deshabilitan todos los elementos del menú en `BottomNavigationView`. Esto puede ser útil para bloquear la navegación cuando sea necesario, por ejemplo, durante ciertas operaciones o pantallas específicas.
5. **Manejo de `onBackPressed()`**
    - `onBackPressed()` está anulado para mostrar un diálogo de confirmación antes de cerrar la aplicación usando `mostrarDialogoCerrarApp()`.
6. **Diálogo de Confirmación**
    - `mostrarDialogoCerrarApp()` crea un `AlertDialog` simple que pregunta al usuario si desea cerrar la aplicación. Si el usuario confirma, `finishAffinity()` se llama para cerrar la aplicación por completo.

DEPENDENCIAS

1. androidx.swiperefreshlayout:swiperefreshlayout:1.1.0

Esta dependencia proporciona la capacidad de actualizar el contenido de una vista deslizando hacia abajo. Es útil para implementar la funcionalidad de "pull-to-refresh" en listas y otros tipos de vistas.

2. com.github.bumptech.glide:glide:4.12.0

Glide es una biblioteca de carga de imágenes rápida y eficiente para Android que maneja la descarga, almacenamiento en caché y visualización de imágenes, incluidas animaciones y transformaciones.

- **annotationProcessor("com.github.bumptech.glide:compiler:4.12.0")**: Esta es una dependencia de procesador de anotaciones necesaria para Glide que ayuda en la generación de código para cargar imágenes de manera eficiente.

3. Dependencias de Firebase

Firebase es una plataforma de desarrollo de aplicaciones móviles de Google que proporciona una variedad de servicios backend.

- **com.google.firebase:firebase-bom:33.0.0**: Esta dependencia declara un BOM (Bill of Materials) que facilita la gestión de versiones para las dependencias de Firebase en tu proyecto.
- **com.google.firebase**
  : Esta dependencia permite integrar Firebase Analytics para recopilar y analizar datos sobre el uso de tu aplicación.
- **com.google.firebase**
  : Esta dependencia facilita la integración de Firebase Authentication, proporcionando métodos Kotlin-friendly para autenticar usuarios.

4. Dependencias de Retrofit y OkHttp

Retrofit es una biblioteca de cliente HTTP para Android y Java que facilita la comunicación con servicios web mediante API RESTful.

- **com.squareup.retrofit2:retrofit:2.9.0**: Retrofit principal para la configuración de las solicitudes HTTP.
- **com.squareup.retrofit2:converter-gson:2.9.0**: Conversor Gson para Retrofit, que convierte automáticamente JSON a objetos Kotlin/Java.
- **com.squareup.okhttp3:logging-interceptor:4.9.0**: Interceptor de registro para OkHttp, útil para depurar y registrar las solicitudes HTTP y respuestas.

5. com.intuit.sdp:sdp-android:1.1.1

SDP (Scalable DP) es una biblioteca que facilita el uso de tamaños de dimensiones "responsive" en Android, permitiendo que las dimensiones se escalen automáticamente según la densidad de píxeles de la pantalla del dispositivo.

6. com.google.android.material:material:1.4.0

Material Components para Android es una biblioteca de diseño que proporciona estilos y componentes de UI siguiendo las directrices de Material Design de Google. Mejora la apariencia y la usabilidad de las aplicaciones Android.

7. Dependencias para funciones de traducción

Estas dependencias están relacionadas con las funcionalidades de traducción usando Firebase ML (Machine Learning).

- **com.google.firebase:firebase-ml-natural-language-translate-model:20.0.8**: Modelo de traducción de Firebase ML Natural Language.
- **com.google.firebase:firebase-ml-natural-language:22.0.1**: Biblioteca principal de Firebase ML Natural Language para funciones de procesamiento de lenguaje natural.

8. com.pierfrancescosoffritti.androidyoutubeplayer:core:11.0.1

Esta dependencia permite integrar un reproductor de videos de YouTube directamente en tu aplicación Android, lo que facilita la reproducción de videos de YouTube sin salir de la aplicación.