

Proyecto de investigación Flutter



Nombre del Proyecto: Wiki Valorant

Autor: Pablo Rodríguez Cascón

Fecha: 20/12/2024 - 31/01/2025

Índice

Proyecto de investigación Flutter.....	1
Índice.....	2
1. Introducción.....	3
2. Explicación del Proyecto.....	4
3. Descripción de las Pantallas.....	5
3.1. Pantalla de Inicio.....	5
3.2. Pantalla de Agentes.....	6
3.2.1. Pantalla de Detalles Agente.....	7
3.2.2. Pantalla de Ruleta Agentes.....	8
3.3. Pantalla de Armas.....	9
3.3.1. Pantalla de Detalles Armas.....	10
3.3.2. Pantalla de Skins Armas.....	11
3.3.3. Pantalla de Detalle Skins Armas.....	12
3.4. Pantalla de Escudos.....	13
3.4.1. Pantalla de Detalles Escudo.....	14
3.5. Pantalla de Mapas.....	15
3.5.1. Pantalla de Detalles Mapas.....	16
3.6. Pantalla de Amuletos.....	17
3.7. Pantalla de Modos de Juego.....	18
3.5. Pantalla de Detalles Modo de juego.....	19
3.8. Pantalla de Tarjetas de Jugador.....	20
3.9. Pantalla de Sprays (Grafitis).....	21
4. API Utilizada.....	21
5. Web Scraping.....	22
5.1 ¿Qué es el Web Scraping?.....	22
5.2 Implementación en la Aplicación.....	22
5.3 Implementación en la Aplicación.....	22
5.4 Cómo se extraen los datos.....	22
6. Nuevos Widgets y Dependencias Utilizadas.....	23
Widget 1: PageView.....	23
Widget 2: BorderedText.....	24
Widget 3: CachedNetworkImage.....	25
Widget 4: FortuneBar.....	26
Widget 5: AnimatedContainer.....	28
Widget 6: BoxShadow (API Flutter).....	29
Widget 7: AnimatedSwitcher.....	31
Dependencia 1: fluttericon.....	32
Dependencia 2: url_launcher.....	32
Dependencia 3: html.....	32

1. Introducción

Manual de usuario de la aplicación Wiki Valorant describe el desarrollo de la aplicación móvil en Flutter basada en la API pública de Valorant y la web de noticias Valorant. Se detallará el propósito de la aplicación, sus principales características, la estructura de sus pantallas y los widgets personalizados utilizados en el proyecto.

2. Explicación del Proyecto

La aplicación está diseñada para proporcionar información sobre el juego Valorant. Permite a los usuarios consultar datos sobre agentes, armas, mapas y habilidades dentro del juego. La interfaz está diseñada para ofrecer una experiencia fluida y accesible para los usuarios.

Las principales funcionalidades incluyen:

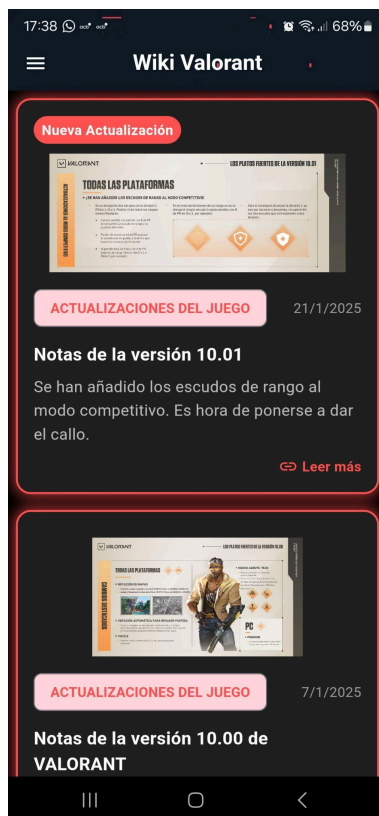
- Información sobre las actualizaciones de Valorant
- Listado de agentes con sus respectivas habilidades y roles.
- Información sobre mapas y armas disponibles en el juego.
- Búsqueda de contenido como amuletos, skins de armas, tarjetas de usuario...
- Interfaz atractiva.

3. Descripción de las Pantallas

3.1. Pantalla de Inicio

- Lista de las actualizaciones.
- Cada nota de versión actúa como un botón que al pulsar te abre el navegador con la nota de parche a la que pulsaste.
- Si la actualización es reciente de los últimos 15 días, aparecerá con un distintivo recuadro arriba que pone nueva actualización.

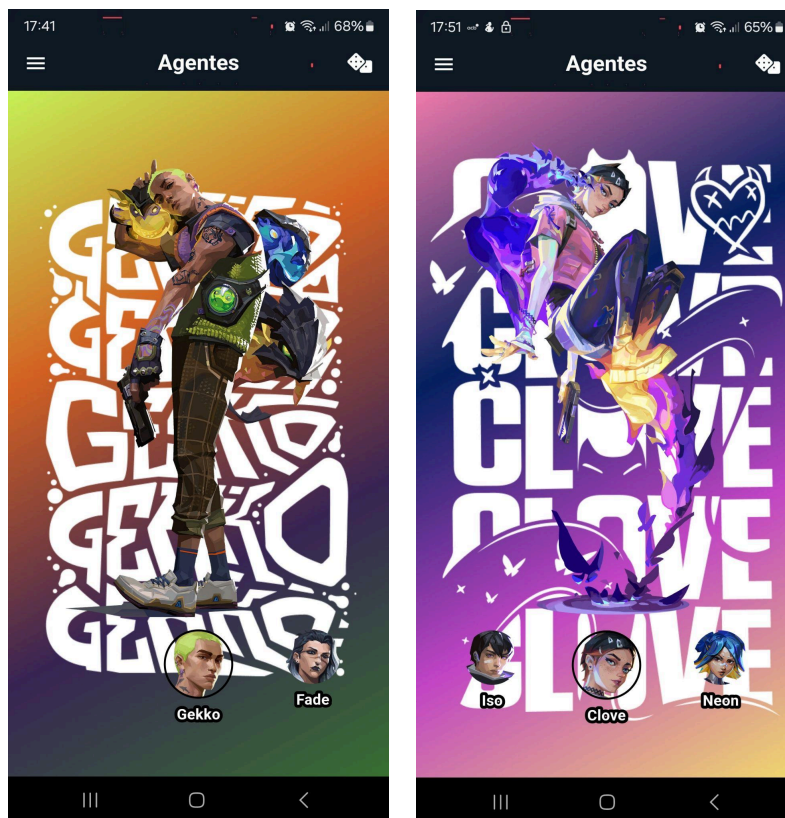
Imágenes:



3.2. Pantalla de Agentes

- Fondo dinámico que cambia los colores de fondo y las dos imágenes del nombre personalizado de fondo y la foto de cuerpo entero a las del agente seleccionado.
- Lista de todos los agentes disponibles en Valorant en forma de rueda para seleccionar uno en el fondo, tienes que desplazar la ruleta hasta ponerlo en el medio.
- Cada imagen de la ruleta hace la función de un botón que al pulsarlo te lleva a los detalles de agente. Que pulsaste
- Botón de arriba a la derecha en el appbar para abrir la ventana de ruleta agentes

Imágenes:



Descripción más detallada:

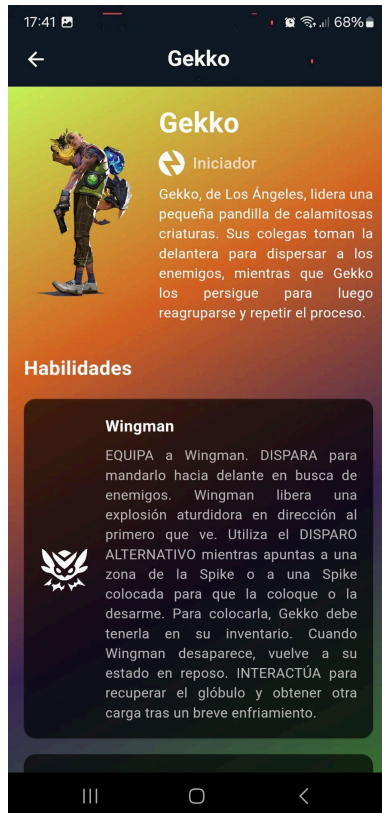
Para lograr el efecto de una lista dinámica en la que el fondo y las imágenes cambian en función del agente central seleccionado, utilicé un `PageView` junto con un `PageController` configurado con `viewportFraction: 0.3`. Esto permite que el avatar central sea el más destacado visualmente, mientras que los demás permanecen parcialmente visibles a los lados.

Cada vez que el usuario desplaza la lista (`onPageChanged`), se actualiza `_selectedIndex.value` con el índice del nuevo avatar central. Esto desencadena la reconstrucción de la interfaz, cambiando dinámicamente tanto el fondo como la imagen del agente seleccionado.

3.2.1. Pantalla de Detalles Agente

- Información detallada de cada agente con habilidades y rol proporcionado todo por la API.

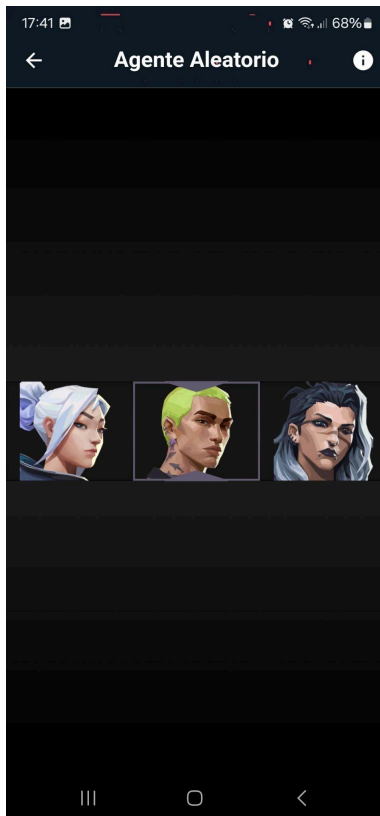
Imágenes:



3.2.2. Pantalla de Ruleta Agentes

- Ruleta de agentes salen las caras de los agentes en la ruleta. Al pulsar la ruleta esta gira y aleatoriamente selecciona un agente. Cuando sale un agente seleccionado, sale en un dialog con el fondo translúcido con la imagen del agente entero y su nombre debajo.
- El botón de la información del appbar te da indicaciones de como se utiliza la ruleta

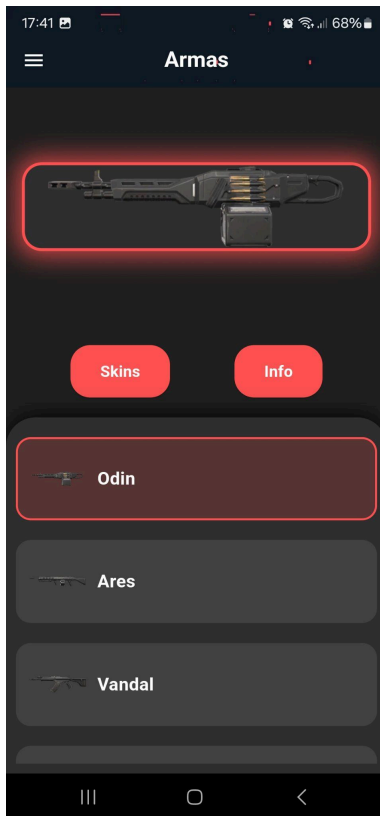
Imágenes:



3.3. Pantalla de Armas

- Catálogo de armas en Valorant.
- Imagen dinámica en la parte superior, que al seleccionar un arma de la lista inferior la imagen cambia a la del arma seleccionada.

Imágenes:



3.3.1. Pantalla de Detalles Armas

- Estadísticas de cada arma, como daño, cadencia de disparo, precisión...

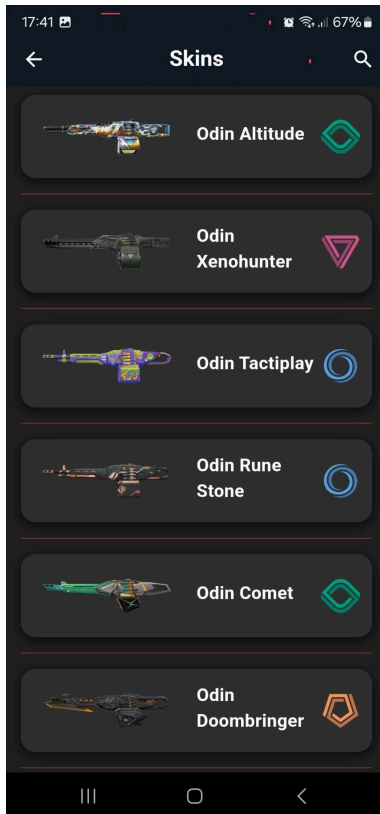
Imágenes:



3.3.2. Pantalla de Skins Armas

- Catálogo de skins del arma que seleccionaste.
- La imagen de su derecha es la rareza del arma. Esta se divide según lo que cueste las armas, las más comunes son las azules y verdes, luego siguen las moradas y amarillas y las más altas son las doradas
- Cuando pulsas en una skin se despliega otra ventana detalles de la skin.

Imágenes:



3.3.3. Pantalla de Detalle Skins Armas

- Imagen principal dinámica que se puede visualizar las distintas variantes de cada skin.
- Debajo de la imagen está el nombre y variante de cada arma.
- Dos IconButtons una flecha hacia la izquierda que si tiene una variable anterior te la mostrará y otra flecha hacia la derecha que si hay una variable posterior la mostrará en la imagen de arriba.

Imágenes:



3.4. Pantalla de Escudos

- Catálogo de escudos disponibles en la partida.

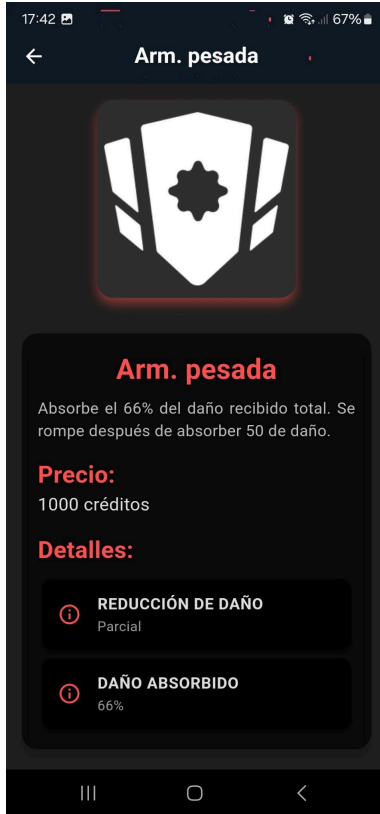
Imágenes:



3.4.1. Pantalla de Detalles Escudo

- Detalles sobre cada escudo.

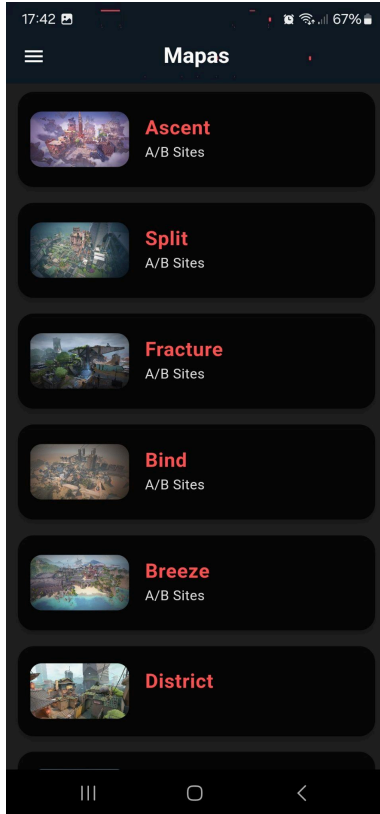
Imágenes:



3.5. Pantalla de Mapas

- Muestra una lista de los mapas disponibles en el juego.

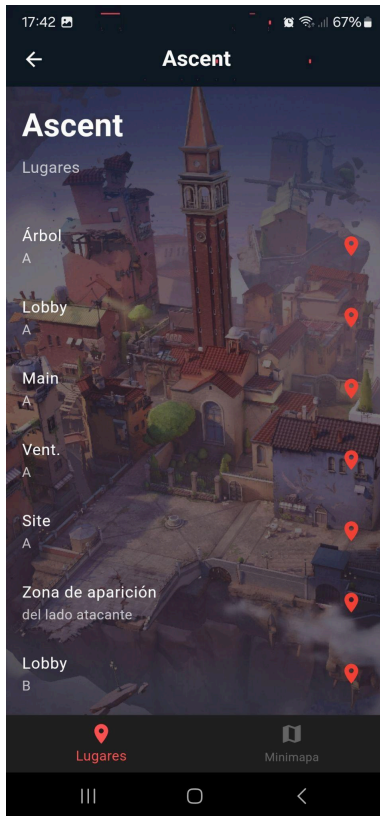
Imágenes:



3.5.1. Pantalla de Detalles Mapas

- Detalles sobre cada mapa, incluyendo su diseño y ubicaciones clave, se puede cambiar entre ellos por un navigatorbar.

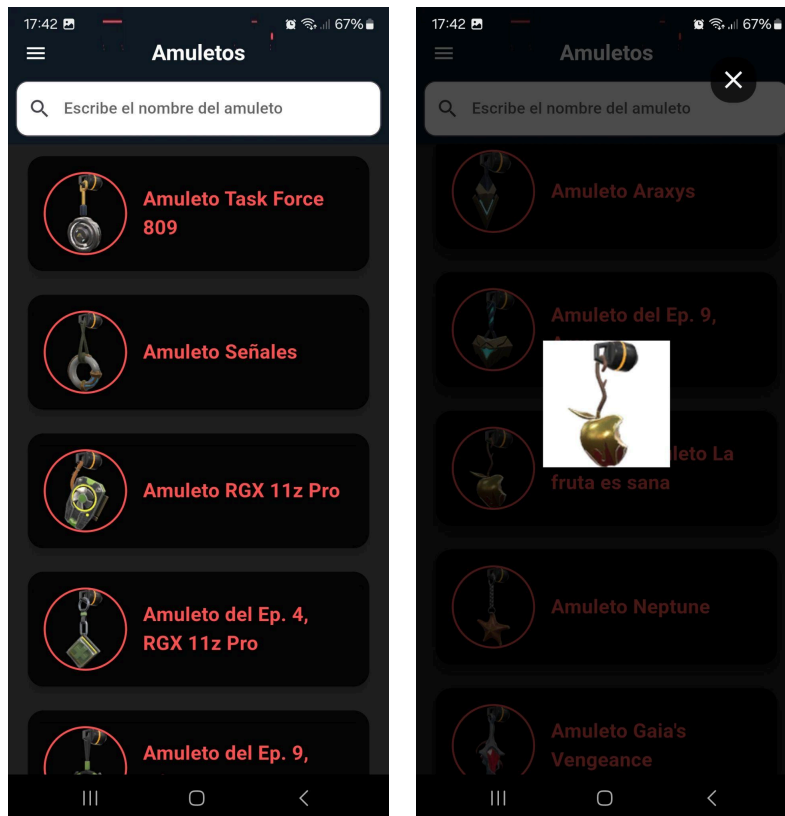
Imágenes:



3.6. Pantalla de Amuletos

- Muestra los amuletos que hay en el juego.
- Al pulsar sobre uno despliega un dialog que te muestra el amuleto.

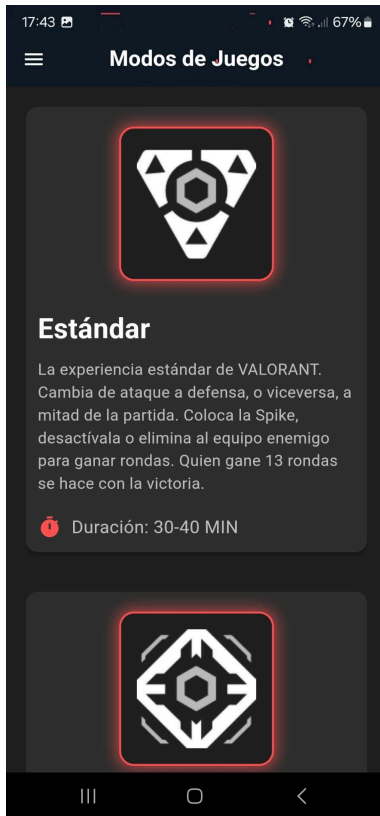
Imágenes:



3.7. Pantalla de Modos de Juego

- Muestra los modos de juego que están en el juego, una descripción de pequeña y su duración si es que la API la proporciona.

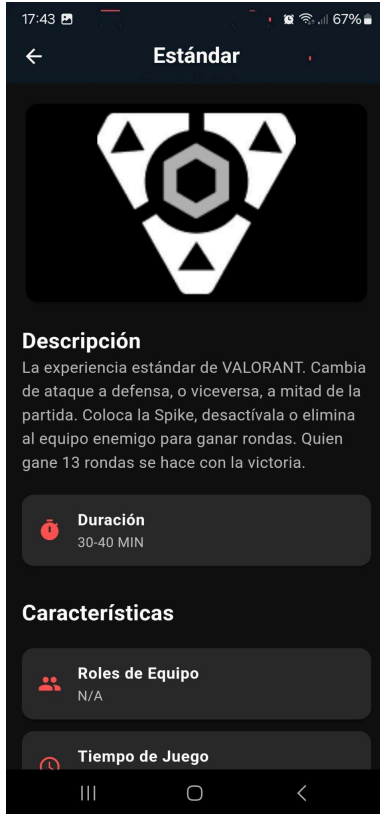
Imágenes:



3.5. Pantalla de Detalles Modo de juego

- Detalles sobre el modo de juego, incluyendo su descripción y características.

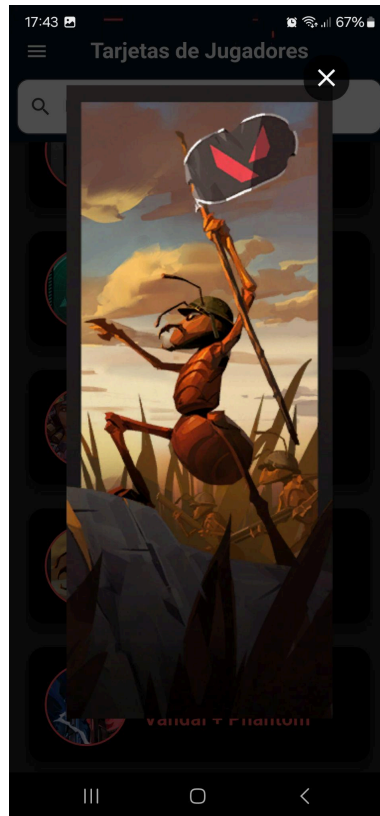
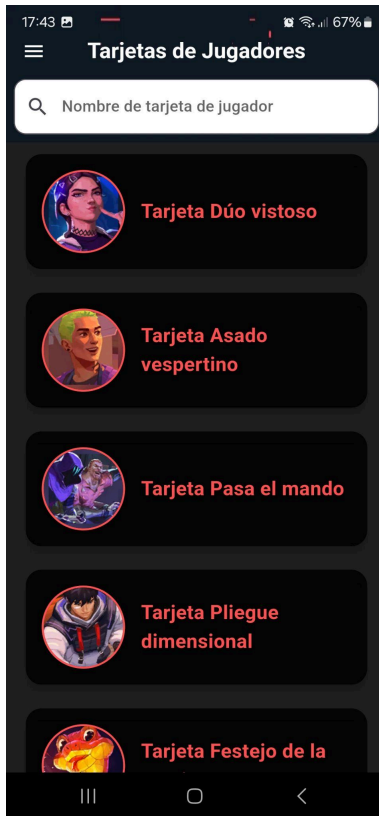
Imágenes:



3.8. Pantalla de Tarjetas de Jugador

- Muestra las tarjetas de jugador que hay en el juego.
- Al pulsar sobre uno despliega un dialog que te muestra la tarjeta de jugador completa.

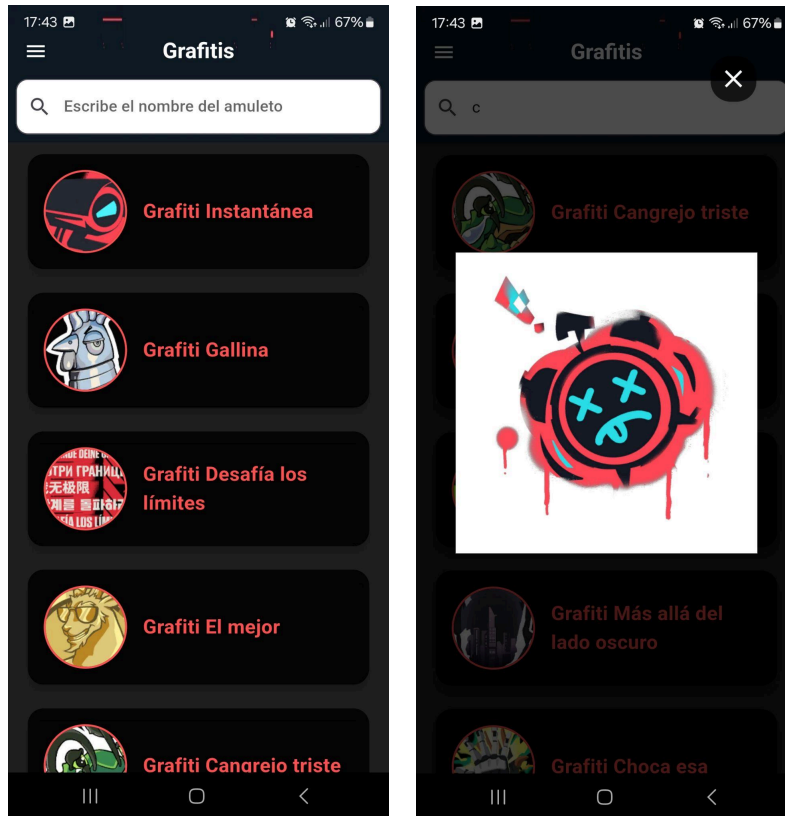
Imágenes:



3.9. Pantalla de Sprays (Grafitis)

- Muestra los Grafitis que hay en el juego.
- Al pulsar sobre uno despliega un dialog que te muestra el spray, ya sea una imagen o si tiene movimiento se verá de esta forma.

Imágenes:



4. API Utilizada

La aplicación se basa en la API pública de Valorant “<https://valorant-api.com/>”, la cual proporciona datos en tiempo real sobre los elementos del juego. La API permite obtener información detallada de los siguientes aspectos:

- Agentes y sus habilidades.
- Armas, estadísticas y skins.
- Mapas y ubicaciones.
- Amuletos, eventos...

5. Web Scraping

5.1 ¿Qué es el Web Scraping?

El Web Scraping es una técnica utilizada para extraer información de sitios web de forma automatizada. Consiste en obtener el código HTML de una página, analizarlo y extraer los datos relevantes para su uso en distintas aplicaciones. Es útil cuando no se dispone de una API oficial que facilite el acceso a la información.

5.2 Implementación en la Aplicación

En esta aplicación se ha utilizado Web Scraping para extraer información de las notas de parche de Valorant desde la URL "<https://playvalorant.com/es-es/news/tags/patch-notes/>". Esto permite mantener a los usuarios actualizados con los últimos cambios del juego sin necesidad de visitar la página manualmente.

5.3 Implementación en la Aplicación

Para la extracción de datos, se emplean las siguientes herramientas y bibliotecas en Flutter:

- http para realizar las solicitudes HTTP.
- html para analizar y extraer la información de la página web.

5.4 Cómo se extraen los datos

El proceso de scraping sigue los siguientes pasos:

1. **Realizar la solicitud HTTP:** Se envía una petición a la URL especificada para obtener el contenido de la página.
2. **Analizar el HTML:** Se utiliza la biblioteca html para estructurar el contenido y facilitar la extracción de datos.
3. **Extraer información relevante:** Se buscan elementos HTML que contienen los títulos, fechas y enlaces de las notas de parche (las imágenes son más complicadas de extraer, ya que cargan en "alt loading: lazy" haciendo que cargue después que la página, esto lo solucioné extrayendo la imagen de dentro de la información del enlace si no tiene, que hay veces que no tienen, esta utiliza una de assets).
4. **Se devuelven los datos:** A través de un provider se utilizan los datos que se adquieren de la página web

6. Nuevos Widgets y Dependencias Utilizadas

A continuación, se describen los widgets personalizados y las dependencias utilizadas en el proyecto que no fueron tratados en clase:

Widget 1: [PageView](https://api.flutter.dev/flutter/widgets/PageView-class.html)

Ubicación en el Proyecto:

Este está ubicado en el archivo **“agente_screen.dart”**

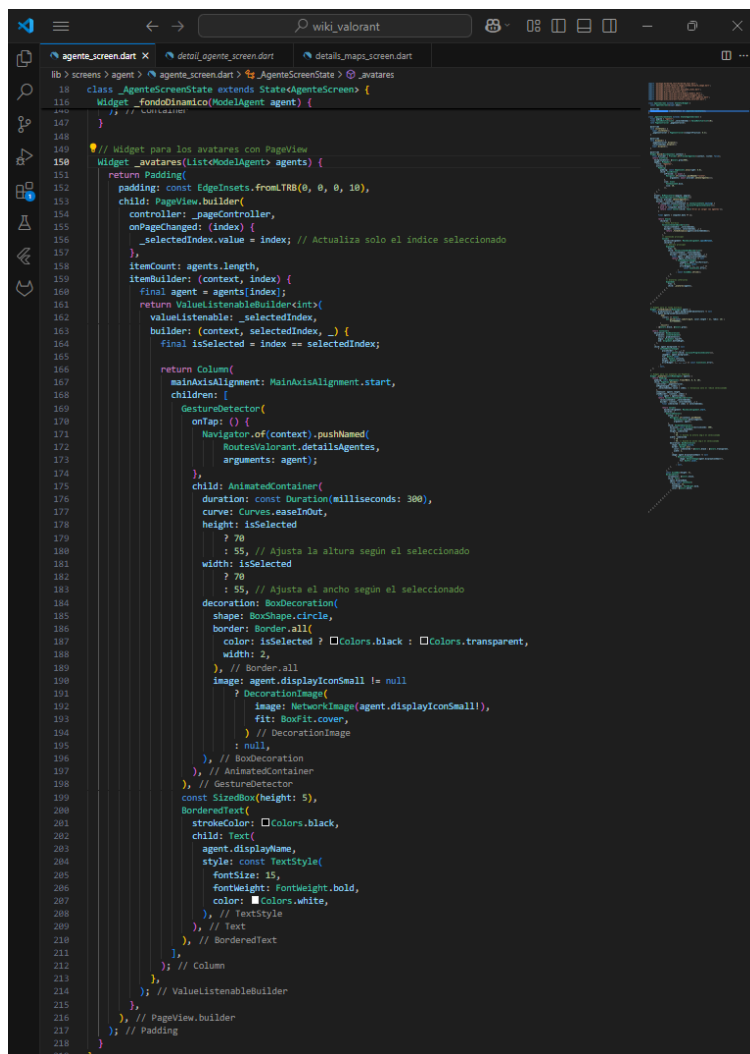
Descripción:

Una lista desplazable que funciona página por página. Esto tenía el propósito de simular una lista como una rueda que hace que cambie el fondo de la pantalla.

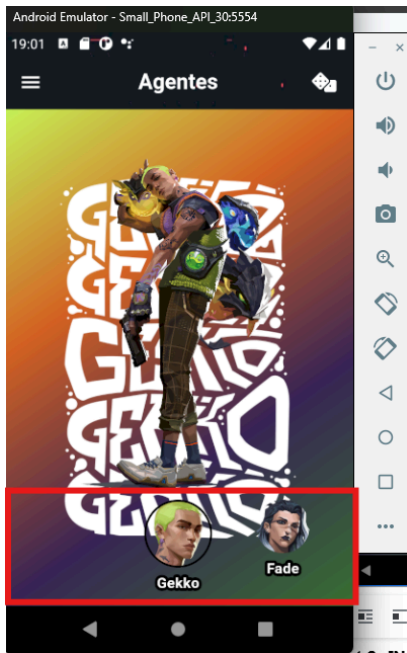
API Flutter:

<https://api.flutter.dev/flutter/widgets/PageView-class.html>

Captura en el proyecto:



Captura en funcionamiento:



Como lo descubrí:

Este lo encontré buscando por internet encontré el componente en la API [“https://api.flutter.dev/flutter/widgets/PageView-class.html”](https://api.flutter.dev/flutter/widgets/PageView-class.html), ya que necesitaba saber cuando cambiaba de posición y el que está en el medio, este widget se ajustaba a lo que necesitaba y para entender como funcionaba y utilizarlo como quería utilice de guía esta página [“https://docs.flutterflow.io/concepts/navigation/pageview/”](https://docs.flutterflow.io/concepts/navigation/pageview/).

Widget 2: BorderedText

Ubicación en el Proyecto:

Por ejemplo, está ubicado por ejemplo en el nombre de debajo de la foto de los avatares **“agente_screen.dart”**.

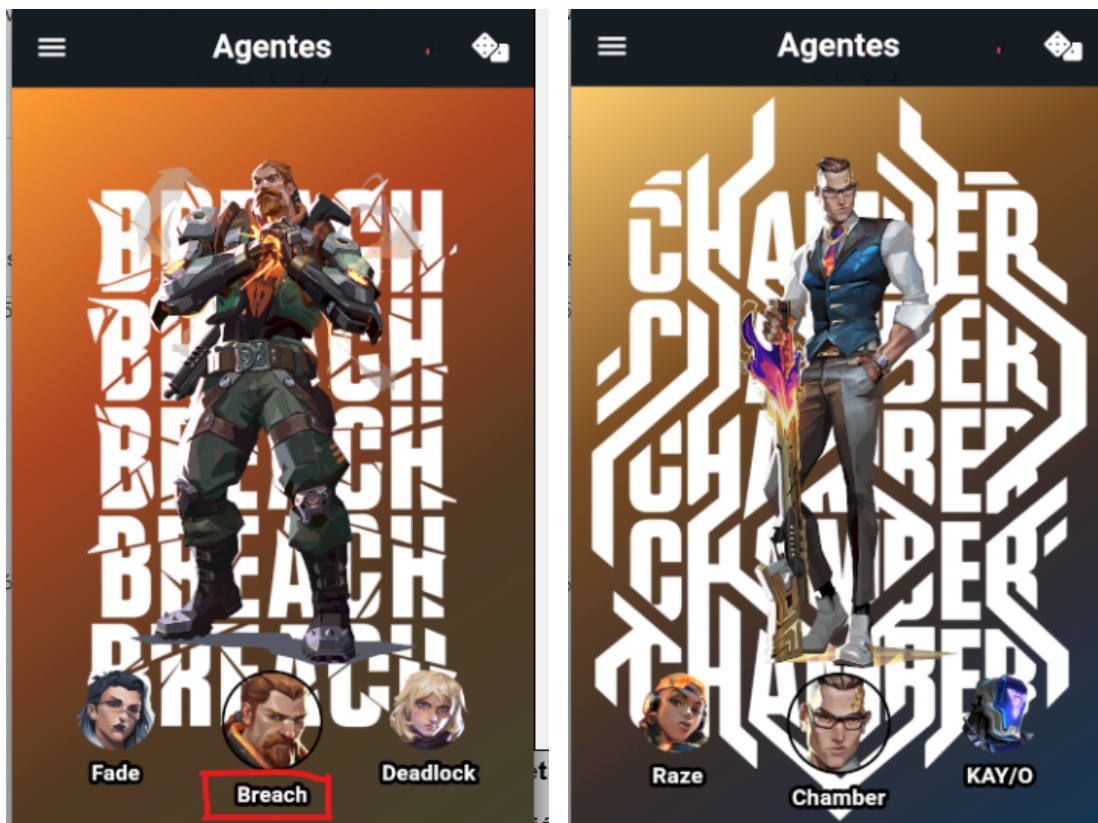
Descripción:

El propósito de utilizar esto fue que las letras se pudieran ver aunque el fondo sea claro o oscuro. Ejemplo en la pestaña agentes los colores del fondo cambian y esto deja paso a que pueda haber fondos claros y oscuros, con esto soluciono ese problema.

Captura en el proyecto:

```
199   const SizedBox(height: 5),
200   BorderedText(
201     strokeColor: Colors.black,
202     child: Text(
203       agent.displayName,
204       style: const TextStyle(
205         fontSize: 15,
206         fontWeight: FontWeight.bold,
207         color: Colors.white,
208       ), // TextStyle
209     ), // Text
210   ), // BorderedText
```

Captura en funcionamiento:



Como lo descubrí:

Sabía que lo habría en el "<https://pub.dev/>" lo busque directamente.

Widget 3: [CachedNetworkImage](#)

Ubicación en el Proyecto:

Lo he utilizado en varios lugares del proyecto, pero el principal es en lista de mapas "maps_screen.dart"

Descripción:

Permite cargar imágenes desde internet y almacenarlas en caché para mejorar el rendimiento de la aplicación. Su propósito en el proyecto es evitar la descarga repetitiva de imágenes, lo que mejora la fluidez de la interfaz y optimiza la experiencia del usuario.

Este enfoque surgió debido a que en la lista de mapas, la carga constante de imágenes ralentizaba la aplicación, generando una sensación de lentitud. Al utilizar `CachedNetworkImage`, las imágenes se almacenan localmente después de la primera carga, reduciendo el consumo de datos y mejorando la velocidad de navegación.

Captura en el proyecto:

```
71 ~ child: SizedBox(  
72 ~   width: 100,  
73 ~   child: CachedNetworkImage(  
74 ~     placeholder: (_, __) => const Center(  
75 ~       child: CircularProgressIndicator(  
76 ~         color: Colors.redAccent,  
77 ~       ), // CircularProgressIndicator  
78 ~     ), // Center  
79 ~     imageUrl: mapa.splash,  
80 ~     errorWidget: (_, __, ___) => Icon(  
81 ~       Icons.error,  
82 ~       color: Colors.redAccent,  
83 ~     ), // Icon  
84 ~   ), // CachedNetworkImage  
85 ~ ), // SizedBox  
86 ~ ), // ClipRRect
```

Captura en funcionamiento:

La captura en funcionamiento no se puede hacer porque afecta a la velocidad, no es nada estético.

Como lo descubrí:

Buscando en pub.dev “image” en el buscador me salió de primero esta opción y buscando información en internet (foros como Reddit

[“https://www.reddit.com/r/FlutterDev/comments/1bdling/cached_network_image_vs_image_network_which_also”](https://www.reddit.com/r/FlutterDev/comments/1bdling/cached_network_image_vs_image_network_which_also) y Stack Overflow).

Widget 4: [FortuneBar](#)

Ubicación en el Proyecto:

Por ejemplo, está ubicado en **“ruleta_agente_screen.dart”**.

Descripción:

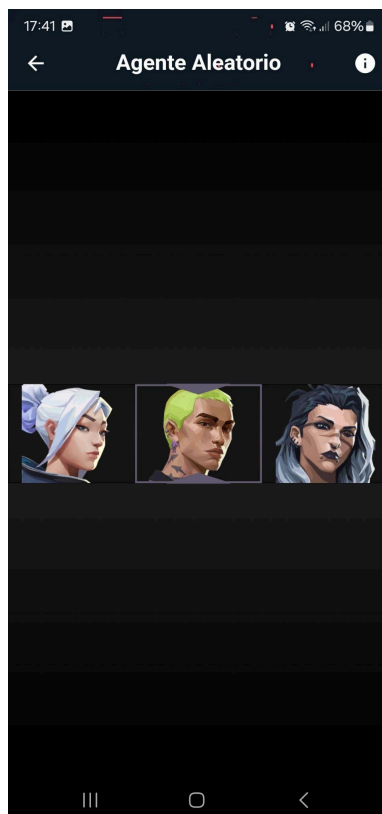
Es una librería de Flutter que permite crear un rueda de la fortuna interactiva. Esta rueda es un componente visual y animado que puedes usar en tu aplicación para seleccionar aleatoriamente entre varias opciones.

Al pulsar sobre ella y esta gira.

Captura en el proyecto:

```
97 class _RuedaAgentesState extends State<RuedaAgentes> {
63   Widget build(BuildContext context) {
75     child: FortuneBar(
76       duration: Duration(seconds: 3),
77       rotationCount: 14,
78       height: double.infinity,
79       animateFirst: false,
80       styleStrategy: UniformStyleStrategy(
81         color: Colors.black.withAlpha(70),
82         borderColor: Colors.black,
83       ), // UniformStyleStrategy
84       selected: controller.stream,
85       items: [
86         for (var agente in widget.list)
87           FortuneItem(
88             child: Image.network(agente.displayIcon!),
89           ), // FortuneItem
90       ],
91     onAnimationEnd: () {
92       // Mostrar el diálogo con el personaje seleccionado
93       if (agenteSeleccionado != null) {
94         showDialog(
95           context: context,
96           builder: (context) => GestureDetector(
97             onTap: () => Navigator.of(context).pop(),
98             child: Dialog(
99               backgroundColor: Colors.grey[800],
100               shape: RoundedRectangleBorder(
101                 borderRadius: BorderRadius.circular(16),
102                 side: BorderSide(
103                   color: Colors.redAccent,
104                   width: 2,
105                 ), // BorderSide
106               ), // RoundedRectangleBorder
107               child: Container(
108                 decoration: BoxDecoration(
109                   color: Colors.grey[800],
110                   borderRadius: BorderRadius.circular(16),
111                   boxShadow: [
```

Captura en funcionamiento:



Como lo descubrí:

Pues quería poner una opción en agente aleatorio y busque en pub.dev “fortune wheel”

Widget 5: [AnimatedContainer](#)

Ubicación en el Proyecto:

Por ejemplo, está en “*weapons_screen.dart*”.

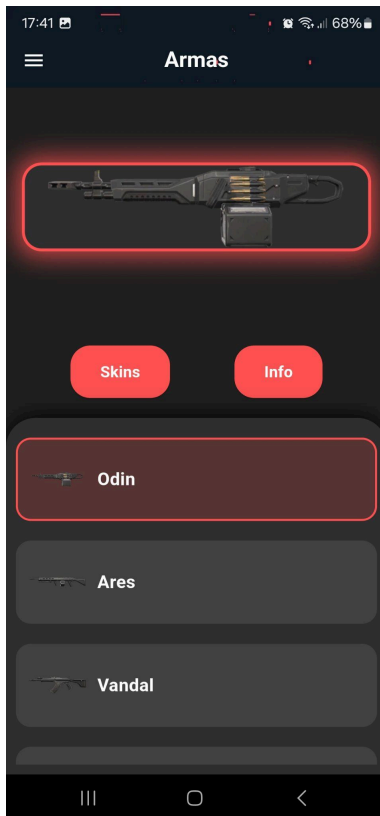
Descripción:

Es un widget que permite animar cambios en las propiedades de un Container, como su tamaño, color, borde, padding, etc., de manera fluida y con una transición suave.

Captura en el proyecto:

```
147 child: AnimatedContainer(  
148   duration: const Duration(milliseconds: 300),  
149   margin: const EdgeInsets.symmetric(vertical: 10.0),  
150   padding: const EdgeInsets.all(10.0),  
151   decoration: BoxDecoration(  
152     color: selectedIndex == index  
153       ? Colors.redAccent.withOpacity(0.2)  
154       : Colors.grey[800],  
155     borderRadius: BorderRadius.circular(15),  
156     border: Border.all(  
157       color: selectedIndex == index  
158         ? Colors.redAccent  
159         : Colors.transparent,  
160       width: 2,  
161     ), // Border.all  
162   ), // BoxDecoration  
163   child: Row(  
164     children: [  
165       ClipRect(  
166         borderRadius: BorderRadius.circular(10),  
167         child: Image.network(  
168           weapons![index].displayIcon,  
169           width: 60,  
170           height: 60,  
171           fit: BoxFit.contain,  
172         ), // Image.network  
173       ), // ClipRect  
174       const SizedBox(width: 10),  
175       Text(  
176         weapons![index].displayName,  
177         style: const TextStyle(  
178           color: Colors.white,  
179           fontSize: 18,  
180           fontWeight: FontWeight.bold,  
181         ), // TextStyle  
182       ), // Text  
183     ],
```

Captura en funcionamiento:



Como lo descubrí:

Cuando quería poner el InkWell estaba en su API y me gusto quise utilizarlo en alguna parte del proyecto

Widget 6: [BoxShadow \(API Flutter\)](#)

Ubicación en el Proyecto:

Por ejemplo, esto fue utilizado en la pantalla de inicio de la aplicación

Descripción:

es una clase que se utiliza para agregar sombras a un widget, específicamente en el contorno de los elementos que tienen una BoxDecoration, como un Container. Permite personalizar el aspecto de la sombra con parámetros como el color, el desplazamiento (offset), el difuminado y la extensión.

En mi caso lo utilicé para dar ese efecto neón de los bordes en varias ventanas

Captura en el proyecto:

```
72         color: Colors.grey[900],
73         border: Border.all(color: Colors.redAccent, width: 2), // Borde rojo
74         borderRadius: BorderRadius.circular(15),
75     },
76     Type: List<BoxShadow>
77     BoxShadow(
78         color: Colors.redAccent.withOpacity(0.7),
79         blurRadius: 12,
80         spreadRadius: 2,
81     ), // BoxShadow
82 ],
83 ), // BoxDecoration
```

Captura en funcionamiento:



Widget 7: [AnimatedSwitcher](#)

Ubicación en el Proyecto:

Se ubica por ejemplo en la pantalla de “`weapons_screen.dart`”.

Descripción:

Es un widget que permite realizar animaciones cuando el child de un widget cambia. Es útil cuando se desea mostrar diferentes widgets de manera fluida, con una transición animada entre ellos.

Captura en el proyecto:

```
74  ✓ child: AnimatedSwitcher(  
75      duration: const Duration(milliseconds: 500),  
76  ✓ child: Container(  
77      key: ValueKey<int>(selectedIndex),  
78      margin: const EdgeInsets.all(16.0),  
79  ✓ decoration: BoxDecoration(  
80      borderRadius: BorderRadius.circular(20),  
81  ✓ boxShadow: [  
82  ✓     BoxShadow(  
83          color: Colors.redAccent.withOpacity(0.8),  
84          blurRadius: 20,  
85          spreadRadius: 5,  
86          offset: const Offset(0, 0),  
87      ✓     ), // BoxShadow  
88      ],  
89      ), // BoxDecoration  
90  ✓ child: Container(  
91  ✓     decoration: BoxDecoration(  
92          borderRadius: BorderRadius.circular(20),  
93          color: Colors.grey[850],  
94  ✓     border: Border.all(  
95          color: Colors.redAccent,  
96          width: 3,  
97      ✓     ), // Border.all  
98      ), // BoxDecoration  
99  ✓ child: ClipRRect(  
100      borderRadius: BorderRadius.circular(17),  
101  ✓ child: Image.network(  
102      weapons![selectedIndex].displayIcon,  
103      fit: BoxFit.cover,  
104      ), // Image.network  
105      ), // ClipRRect  
106      ), // Container  
107      ), // Container  
108      ), // AnimatedSwitcher
```

Dependencia 1: [fluttericon](#)

Descripción:

Este trae más iconos para poner

Uso en el proyecto:

Fue utilizado en el Drawer

Como lo descubrí:

Lo descubrí buscando en internet "Flutter icons" y me salió esta [página web](#)

Dependencia 2: [url_launcher](#)

Descripción:

Es un paquete que permite lanzar URLs o abrir aplicaciones externas desde tu app Flutter.

Tuve que poner esto en el manifest para que funcionara

```
<queries>
  <intent>
    <action android:name="android.intent.action.VIEW" />
    <data android:scheme="https" />
  </intent>
</queries>
```

Uso en el proyecto:

En la pantalla inicial de cada parche puedes pulsar y abre el link

Como lo descubrí:

Lo descubrí a través de un foro [Stack Overflow](#)

Dependencia 3: html

Descripción:

Es un paquete que permite renderizar contenido HTML dentro de una aplicación Flutter.
Esta librería convierte el código HTML en widgets de Flutter

Uso en el proyecto:

La utilizo en "services_valorant_scrapping.dart"

Como lo descubrí:

Lo necesitaba para interpretar el código HTML que descargaba de internet para hacer el web scraping