**Documento Informe Aplicaciones Móviles**

**“RegistrAPP”**

***Fecha: [04/12/2024]***

**Integrantes:***Mauricio Carrión Torres.  
Daniel Román Álvarez.*

**Contenido**

Contenido

[Introducción 3](#_Toc184247431)

[Desarrollo 4](#_Toc184247432)

[1. Arquitectura de la Aplicación 4](#_Toc184247433)

[*Componentes de la Interfaz de Usuario (UI):* 4](#_Toc184247434)

[*Servicio de Almacenamiento (storage.service.ts):* 4](#_Toc184247435)

[*Manejo de Rutas(app.route.ts):* 5](#_Toc184247436)

[*Gestión de usuarios(user.service.ts):* 5](#_Toc184247437)

[2. Experiencia de Usuario (UX) y Usabilidad en Ionic con Angular: 6](#_Toc184247438)

[*Estructura de Navegación y Componentes de Ionic.* 6](#_Toc184247439)

[*Navegación en Vista:* 7](#_Toc184247440)

[*Tipografía y Paleta de Colores.* 8](#_Toc184247441)

[*Accesibilidad en la Aplicación.* 9](#_Toc184247442)

[*Adaptación a Tamaños de Pantalla: Diseño Responsivo.* 9](#_Toc184247443)

[3. Manejo del Estado y Persistencia de Datos en Ionic con angular 10](#_Toc184247444)

[4. Optimización y Navegación de la Aplicación en Ionic con Angular 13](#_Toc184247445)

[*Uso de ion-card-content para Desplazamiento Optimizado en Listas Grandes:* 13](#_Toc184247446)

[*Configuración de Rutas:* 14](#_Toc184247447)

[*Optimización de Recursos:* 14](#_Toc184247448)

[*Eliminación funcional en el Storage:* 15](#_Toc184247449)

[5. Manejo de API de Códigos QR en Ionic con Angular 16](#_Toc184247450)

[6. Pruebas Unitarias 22](#_Toc184247451)

[*Detalle Terminal VSC:* 22](#_Toc184247452)

[*Detalle Jasmine:* 22](#_Toc184247453)

[*Detalle Consola de Inspección Browser:* 23](#_Toc184247454)

[*P5. Verificar la creación del componente* 24](#_Toc184247455)

[*P6. Cargar datos exitosamente en ngOnInit* 24](#_Toc184247456)

[*P7: Creación exitosa del componente* 24](#_Toc184247457)

[*P8: Valor inicial de qrData* 24](#_Toc184247458)

[*P9: Generación del código QR al pulsar un botón* 24](#_Toc184247459)

[*P10: Verificar que el código QR no se muestra inicialmente* 25](#_Toc184247460)

[*P11: Actualización de createdCode al llamar a generateQRCode()* 25](#_Toc184247461)

[Conclusión 26](#_Toc184247462)

[Conclusión sobre las Pruebas Unitarias Realizadas 26](#_Toc184247463)

[Bibliografía 28](#_Toc184247464)

# Introducción

**RegistrApp** es una aplicación móvil diseñada para facilitar el registro de asistencia en entornos de educación superior, ofreciendo una solución digital práctica y precisa. Este proyecto surge como respuesta a la necesidad de contar con un método eficiente y centralizado para el seguimiento de la asistencia de estudiantes en universidades, centros de formación y otras instituciones académicas. La aplicación está orientada principalmente a dispositivos móviles, permitiendo a docentes y alumnos interactuar de manera intuitiva y rápida con la plataforma.

El principal objetivo de RegistrApp es digitalizar y automatizar el proceso de registro de asistencia, eliminando los métodos manuales tradicionales (como las listas en papel o las hojas de cálculo). A través de la aplicación, los docentes pueden marcar la asistencia de sus alumnos y los estudiantes pueden confirmar su presencia mediante códigos QR generados de manera segura.

**RegistrApp** está diseñada y desarrollada como una aplicación móvil multiplataforma utilizando tecnologías como **Ionic y Angular**. Esta elección permite que la aplicación funcione tanto en dispositivos Android como iOS sin necesidad de desarrollar versiones independientes, optimizando así los recursos de programación y mantenimiento.

# Desarrollo

# Arquitectura de la Aplicación

Para el desarrollo de **RegistrApp**, se ha empleado una arquitectura de Software **Modelo-Vista-Controlador (MVC)**. Esta arquitectura permite una clara separación entre la interfaz de usuario (UI), la lógica de negocio y la gestión de datos, facilitando la mantenibilidad y escalabilidad de la aplicación. A continuación, se detalla cómo se organizan los componentes de la aplicación según esta arquitectura.

## *Componentes de la Interfaz de Usuario (UI):*

En Ionic con Angular, la interfaz de usuario se divide en **componentes** que representan cada pantalla o elemento visual interactivo. Cada componente sigue una estructura de archivos típica en Angular, donde:

* **page.html**: Define el HTML de la vista para cada página o componente visual.
* **page.scss**: Contiene los estilos específicos del componente, adaptados para una interfaz móvil.
* **page.ts**: Actúa como el controlador, gestionando los datos y las interacciones del usuario.

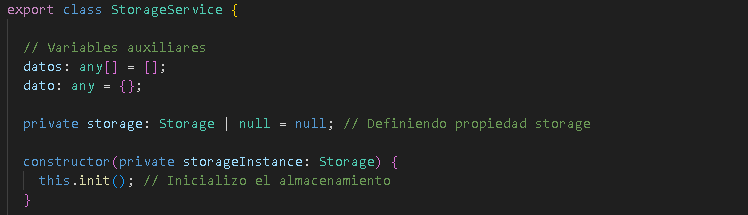


## 

## *Servicio de Almacenamiento (storage.service.ts):*

Utiliza la API de almacenamiento de Ionic para almacenar los datos de asistencia localmente en el dispositivo.

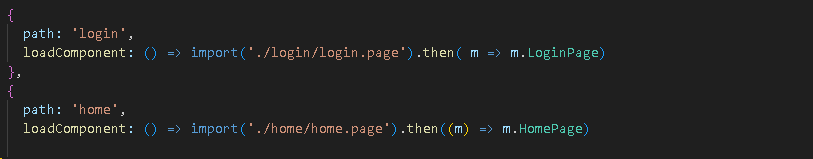




## *Manejo de Rutas(app.route.ts):*

La navegación en la aplicación se gestiona mediante el módulo de rutas de Angular, definido en app-routing.module.ts.

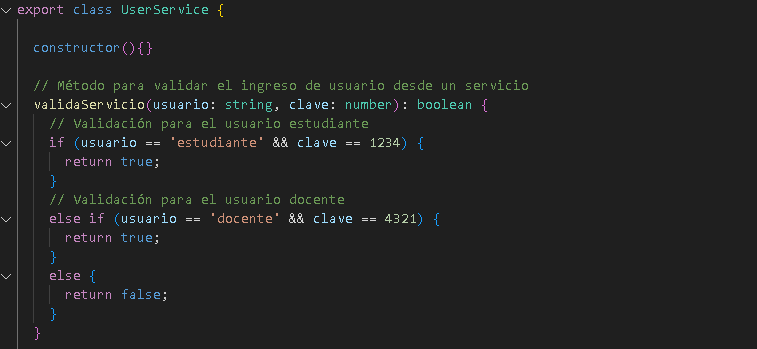




## *Gestión de usuarios(user.service.ts):*

Este servicio se encarga de las operaciones relacionadas con el usuario en la aplicación, tales como la autenticación.





# Experiencia de Usuario (UX) y Usabilidad en Ionic con Angular:

**RegistrApp** está diseñada con Ionic y Angular para ofrecer una experiencia de usuario fluida,accesible y adaptable en dispositivos móviles de diferentes tamaños y capacidades. La aplicación se centra en la usabilidad y accesibilidad para que estudiantes y docentes puedan gestionar la asistencia sin dificultades técnicas. A continuación, se detallan las estrategias y componentes clave empleados para optimizar la UX, la navegación, el rendimiento y la accesibilidad en la aplicación.

## *Estructura de Navegación y Componentes de Ionic.*

El componente de navegación **RouterModule y Router** permiten una transición suave entre vistas y la gestión de rutas de forma optimizada.

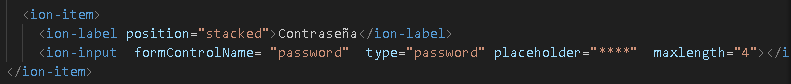


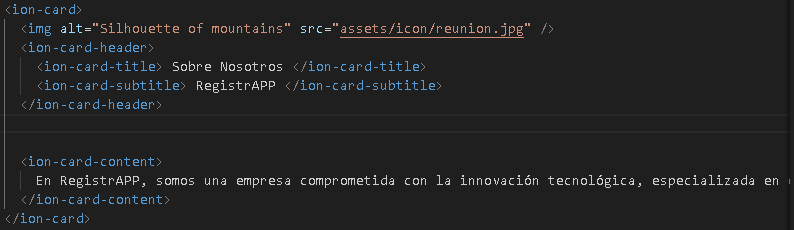
El componente **IonicModule** nos permite acceder a una gama de accesorios visuales que están diseñados para compatibilizar con la vista, tales como **Ion Button, Ion Input, Ion Item, Ion Card.**

****

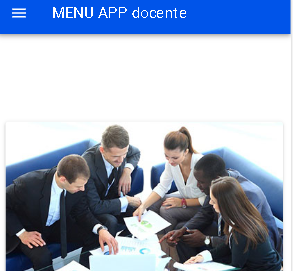
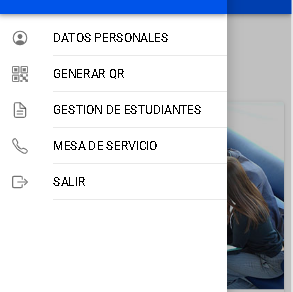
****

****

****

****

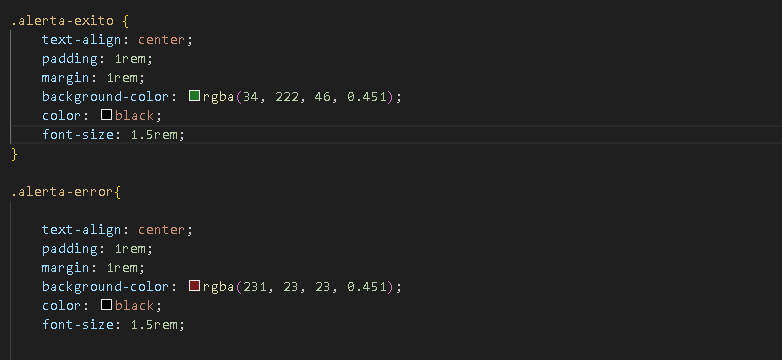
## *Navegación en Vista:*

**** 

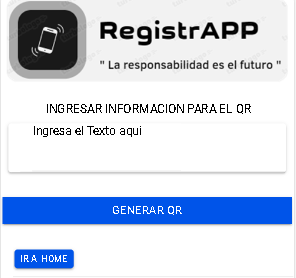
## *Tipografía y Paleta de Colores.*

**Tipografía:** Se seleccionaron fuentes legibles y modernas, con tamaños de texto ajustables mediante variables de CSS para mejorar la lectura en cualquier dispositivo.

**CSS en aplicación**:

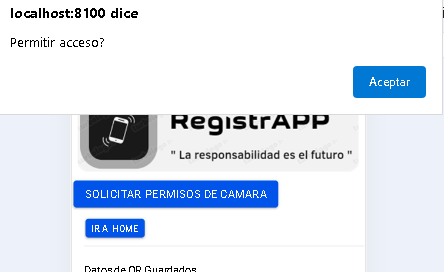
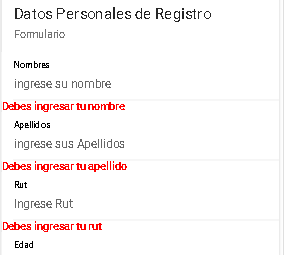


**Colores**: La paleta de colores sigue una línea profesional, con tonos neutros y colores de acento que destacan las opciones principales (por ejemplo, botones de acciones intermedias como azul). Se emplean tonos suaves para el fondo, asegurando un contraste adecuado para la accesibilidad.



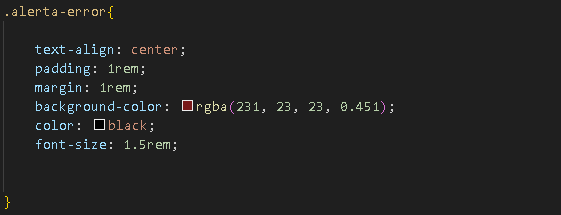
## *Accesibilidad en la Aplicación.*

**Mensajes de Retroalimentación:** Los mensajes de error y éxito son claros y concisos, con alertas visuales y en campos que indican las acciones necesarias o si alguna operación ha sido completada.

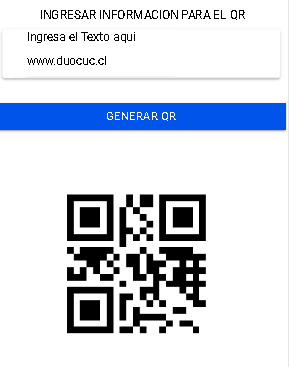
****

## *Adaptación a Tamaños de Pantalla: Diseño Responsivo.*

**Unidades Responsivas:** Las unidades CSS como el remse aplican para que los elementos se ajusten automáticamente a diferentes pantallas sin perder legibilidad.



**Pantallas Limpias y Foco en el Usuario**: Las páginas principales son libres de elementos innecesarios y están enfocadas en una sola tarea por pantalla, manteniendo al usuario centrado en la acción requerida (por ejemplo, solo se muestra el formulario de escaneo y botones funcionales a la aplicación).

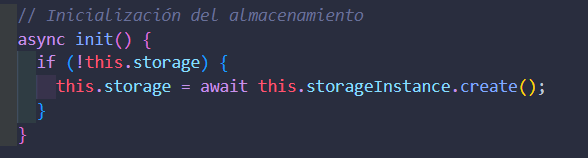


En general, **RegistrApp** proporciona una experiencia de usuario intuitiva, rápida y accesible. El uso de componentes nativos de Ionic , junto con una estructura de navegación modular, permite una navegación eficiente en distintos dispositivos. La cuidadosa selección de tipografía y colores, un diseño responsivo, y las consideraciones de accesibilidad hacen que la aplicación sea agradable de usar y accesible para todos los usuarios. Esta optimización general asegura que RegistrApp no solo sea funcional, sino también fácil de usar y atractiva para todos los miembros de la comunidad educativa.

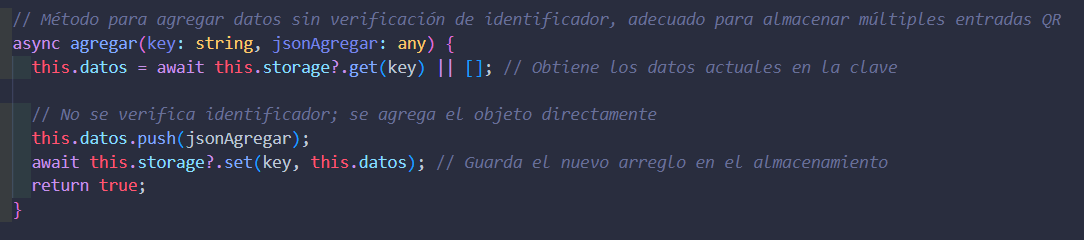
# Manejo del Estado y Persistencia de Datos en Ionic con angular

En nuestra aplicación **RegistrApp** con Ionic/Angular, la persistencia de datos y la gestión del estado se gestionan principalmente mediante el servicio “**storage.service.ts**”, que utiliza **@ionic/storage-angular** para almacenar y recuperar datos en un formato clave-valor. Esto permite que los datos permanezcan disponibles entre sesiones y que se puedan acceder y modificar de forma centralizada.

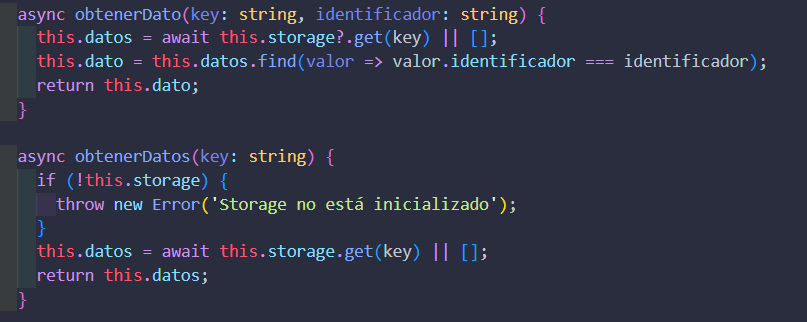
Como se aprecia a continuación en la serie de métodos, podemos evidenciar el manejo de persistencias y la gestión de estos según lo siguiente: En el método "**init**", el servicio inicializa el almacenamiento una vez que se crea la instancia de “**storage"** a través del constructor. Este paso es esencial para asegurarse de que el almacenamiento se encuentra listo para usarse:



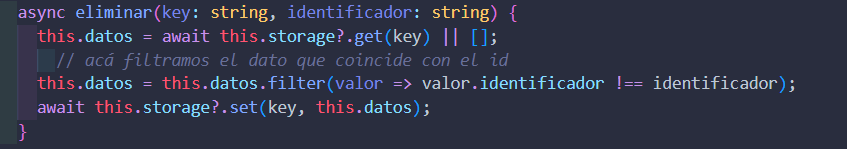
El método **agregar** se encarga de agregar nuevos objetos a una clave específica en el almacenamiento. Si no existen datos previamente asociados a esa clave, el servicio crea un nuevo arreglo vacío. Luego, el nuevo objeto se agrega a dicho arreglo:



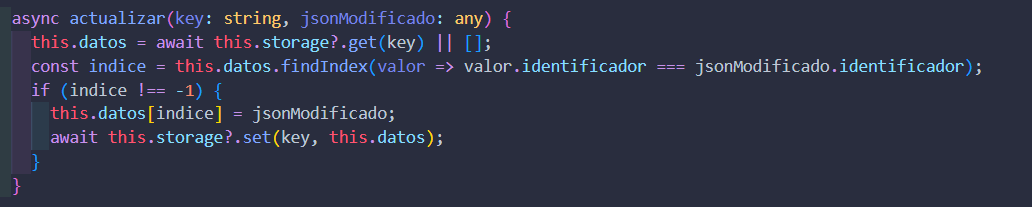
El método **obtenerDatos** obtiene todos los datos asociados a una clave específica, mientras que **obtenerDato** permite obtener un único dato buscando por su identificador (dentro de un arreglo de objetos). Esto es útil cuando se manejan múltiples entradas bajo la misma clave:



El método **eliminar** permite eliminar un dato específico, identificado por su clave única (identificador), dentro de un arreglo bajo una clave específica:

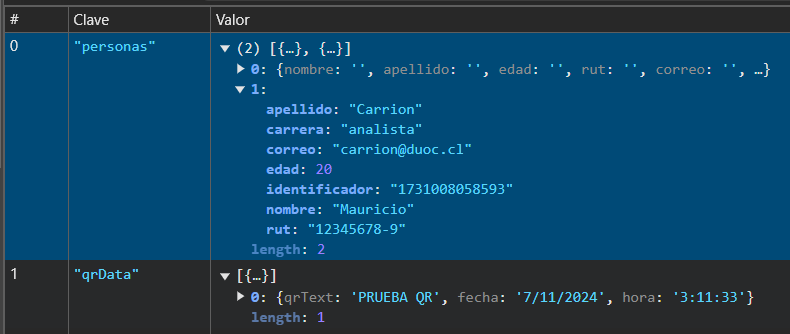


El método **actualizar** permite modificar un dato que ya existe. Se busca el objeto dentro del arreglo utilizando su identificador y se actualiza con los nuevos valores:



Respecto a la estructura de **Datos Clave/Valor**; en el contexto de este servicio, los datos se almacenan utilizando un enfoque de clave-valor. Cada clave puede tener un valor asociado que puede ser un arreglo de objetos.

A continuación, se ejemplifica la estructura de datos en la base:

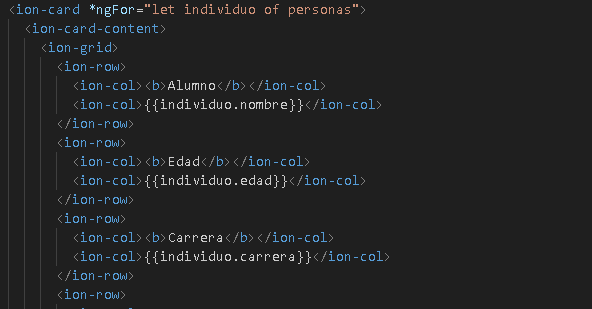


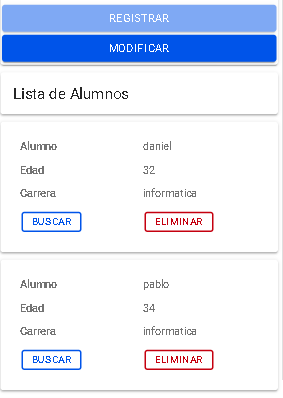
Como cierre de este punto, el uso de **StorageService** en combinación con el almacenamiento de **tipo clave-valor** permite gestionar de manera efectiva la persistencia de datos en la aplicación. Almacenar objetos bajo claves específicas facilita la organización y recuperación de datos críticos, como la información de los usuarios y las configuraciones de la app. Además, los métodos de agregar, actualizar, obtener y eliminar permiten una manipulación flexible de los datos.

# Optimización y Navegación de la Aplicación en Ionic con Angular

## *Uso de ion-card-content para Desplazamiento Optimizado en Listas Grandes:*

Esto se implementa mediante **\*ngFor** en el componente **ion-card-content**, mejorando así el rendimiento al mostrar listas extensas:

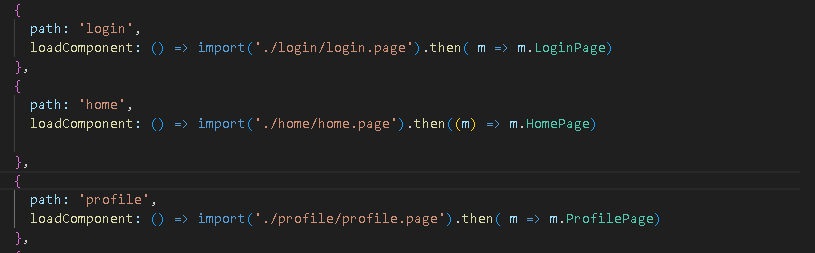




## *Configuración de Rutas:*

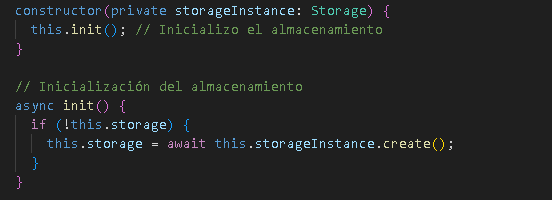
En el archivo de enrutado correspondiente(**app.routes.ts**) , se definen las rutas para cada pestaña, asegurando que cada pestaña cargue un módulo separado y esto nos ofrece una visión más efectiva del movimiento de vista en vista.





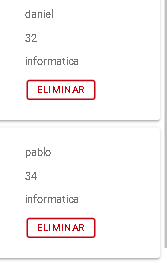
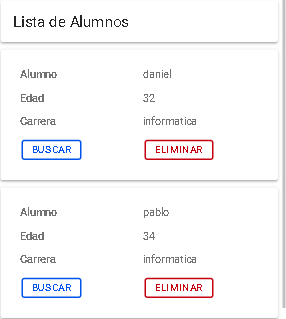
## *Optimización de Recursos:*

Uso de **Ionic Storage** para Caché Local,que sirve para Almacenar recursos o datos frecuentemente utilizados en la caché local permite que la aplicación acceda a ellos sin realizar múltiples llamadas a la API, reduciendo la carga del servidor y mejorando la velocidad.

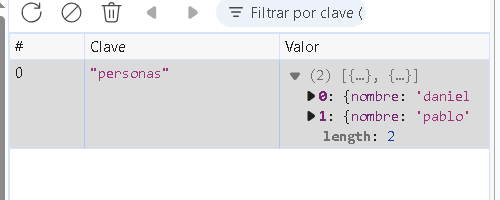


## *Eliminación funcional en el Storage:*

Uso de **Ionic Storage** para Caché Local, que sirve para almacenar recursos o datos frecuentemente utilizados en la caché local permite que la aplicación acceda a ellos, por lo que la funcionalidad del botón “**Eliminar**” sirve para liberar de información el **storage** para no se sature.



Lo cual se vería reflejado en el mismo **storage** de la aplicación:



# Manejo de API de Códigos QR en Ionic con Angular

Para la funcionalidad de **Generación de Códigos QR** nuestra aplicación usa la biblioteca **angularx-qrcode**, que permite crear códigos QR a partir de cualquier dato de entrada (como un texto o un identificador único).

Se debió instalar **angularx-qrcode** en el proyecto a través de la consola o terminal ingresando el siguiente comando:

.

Como parte de la configuración:

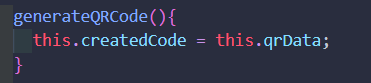
En el archivo **genera-qr.page.ts**, se importa el módulo **QRCodeModule** para integrarlo en la página.



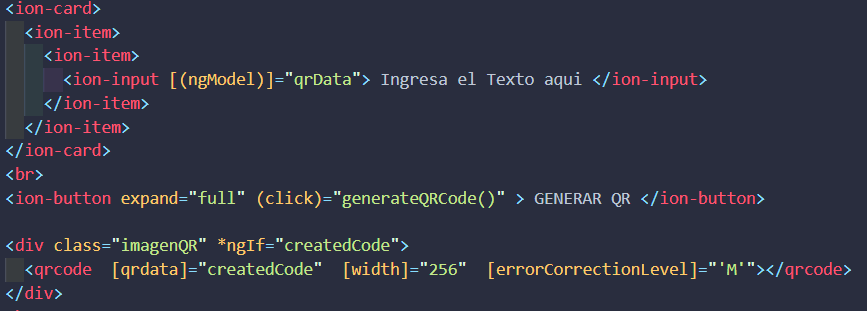
Se utilizó una variable llamada **qrData** para capturar la información ingresada por el usuario y **createdCode** para almacenar el código generado.



En la funcionalidad de Generar Código QR disponible en **genera-qr.page.ts** el método **generateQRCode()** toma el valor de **qrData** y lo asigna a **createdCode**, el cual se pasa al componente <**qrcode**> para crear el código QR.



El código HTML en el template **genera-qr.page.html** permite al usuario ingresar el texto a convertir en QR y muestra el código generado si **createdCode** contiene un valor.



Acá se ilustra el despliegue de la funcionalidad del código mediante la interfaz frontend para la generación del código Qr.



Para la funcionalidad de **Lectura de códigos QR**, nuestra aplicación utiliza **Html5QrcodeScanner** de la biblioteca **html5-qrcode**. Esta biblioteca permite escanear códigos QR desde la cámara del dispositivo.

Como parte de la configuración e implementación en confirmar.page.ts, se debió instalar la biblioteca **html5-qrcode** en el proyecto a través de la consola o terminal ingresando el siguiente comando:



Se importa la biblioteca **Html5QrcodeScanner:**

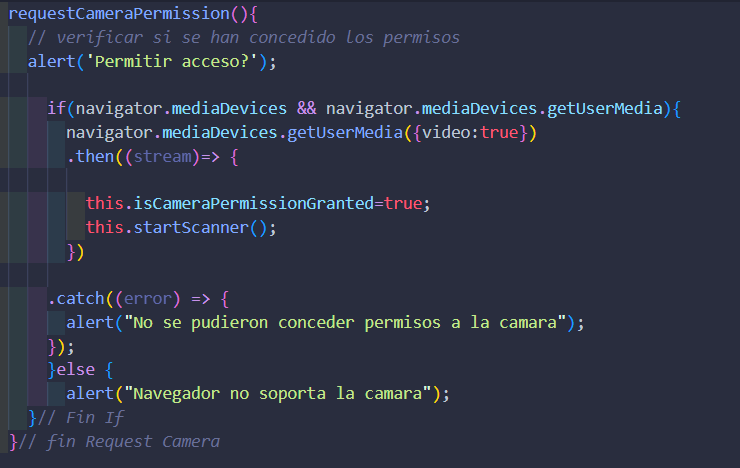


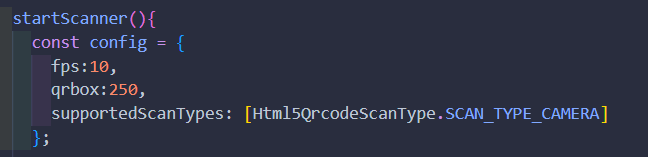
La lógica de la funcionalidad de escaneo de códigos Qr inicia con el método **requestCameraPermission()** solicitando permisos de cámara al usuario y luego llama a **startScanner(**) para iniciar el escáner si los permisos son concedidos.

Al iniciar el escaneo, **startScanner()**, se configura el escáner con **fps** y **qrbox** para optimizar el rendimiento y área de escaneo.

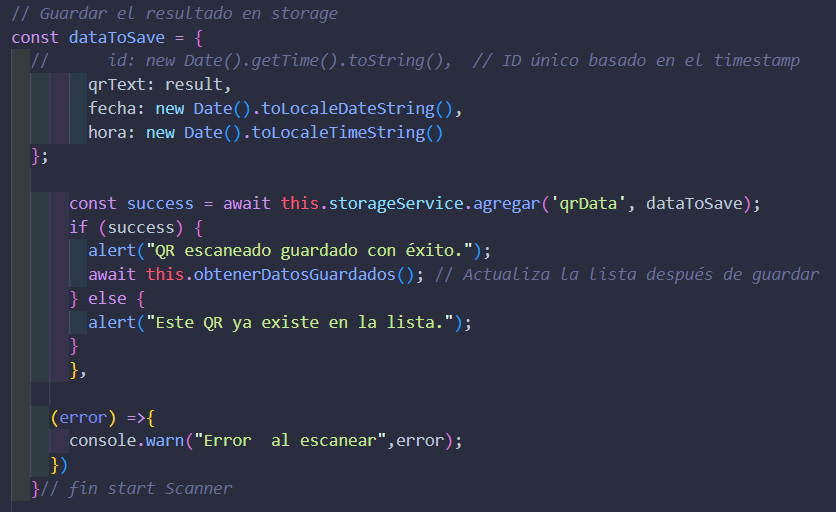
**Html5QrcodeScanner** se inicializa con un contenedor ("**reader**") y configuración de tipo de escaneo **(SCAN\_TYPE\_CAMERA)**.

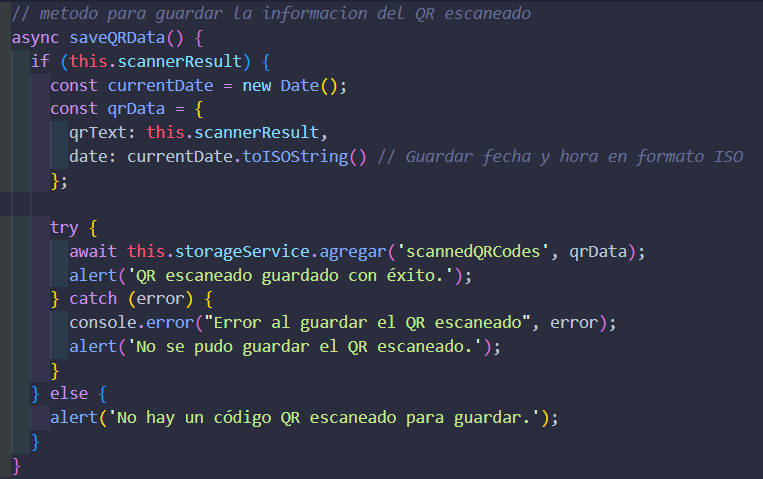
Se define como un manejador de resultados que procesa el valor escaneado y lo almacena usando el **StorageService**.



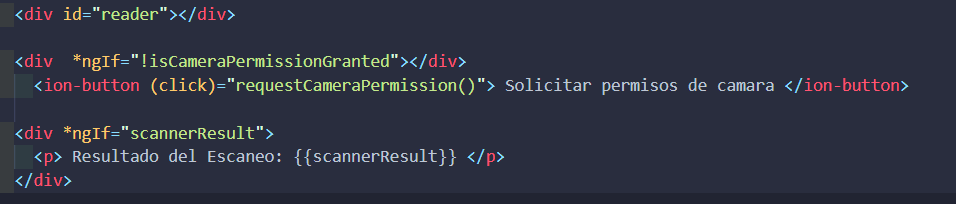


Cuando se escanea un código QR, el resultado se guarda en **scannerResult**, y se agrega un objeto con la información del escaneo (incluyendo fecha y hora) al almacenamiento local.

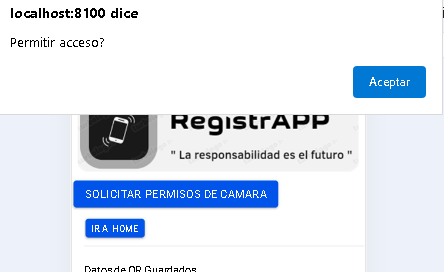




En el template (**confirmar.page.html)**, se incluyen los botones y mensajes necesarios para iniciar el escaneo y mostrar los resultados.



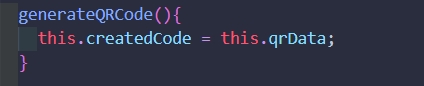
Acá se ilustra el despliegue de la funcionalidad de solicitud de permisos de cámara y escaneo de códigos QR mediante la interfaz frontend.

****

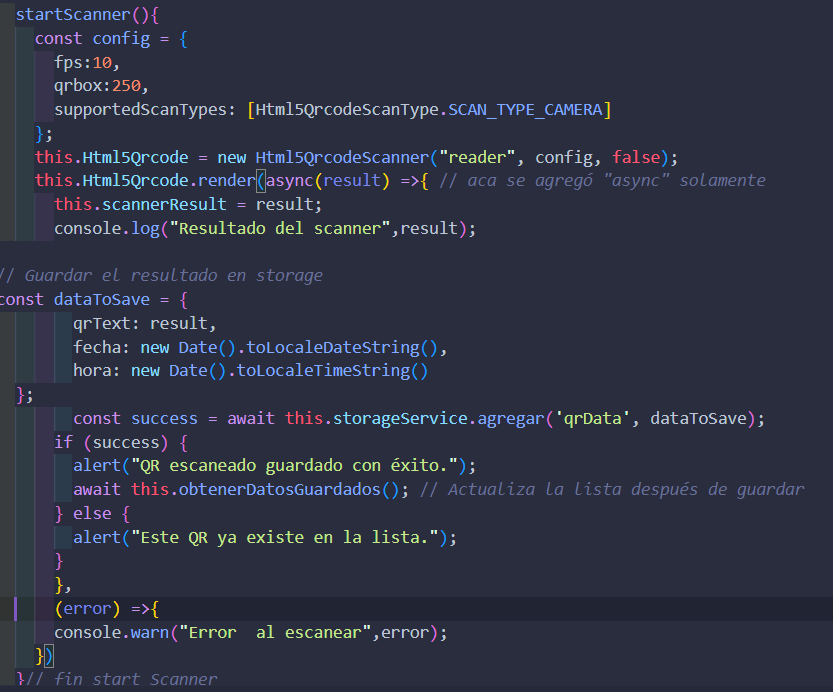
Conexión con Almacenamiento y Gestión de Datos.

Ambas funcionalidades están integradas con el almacenamiento local mediante el servicio **StorageService**:

* **Agregar Datos Escaneados**: Los datos de cada código QR escaneado se guardan con la clave "**qrData**". Cada entrada incluye el texto escaneado, la fecha y la hora.
* **Recuperación de Datos Guardados**: Los datos guardados se recuperan al iniciar **confirmar.page.ts** y se muestran en la lista de resultados de **confirmar.page.html**.
* Generación de Código QR:



* Iniciar Escáner:

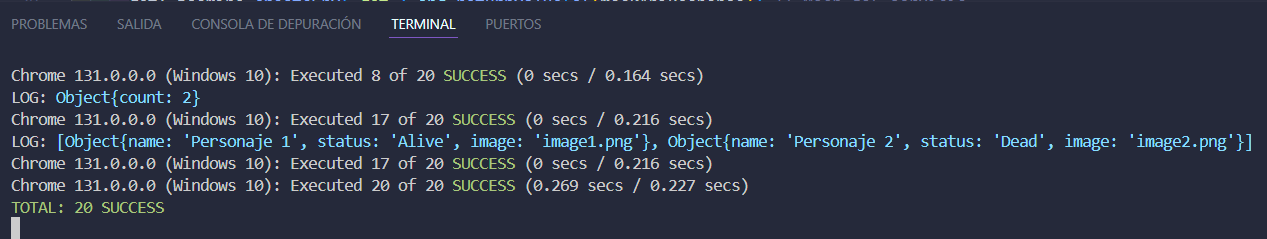


Este resumen abarca el flujo de generación y lectura de códigos QR en la aplicación, así como el almacenamiento de datos escaneados usando un servicio de almacenamiento.

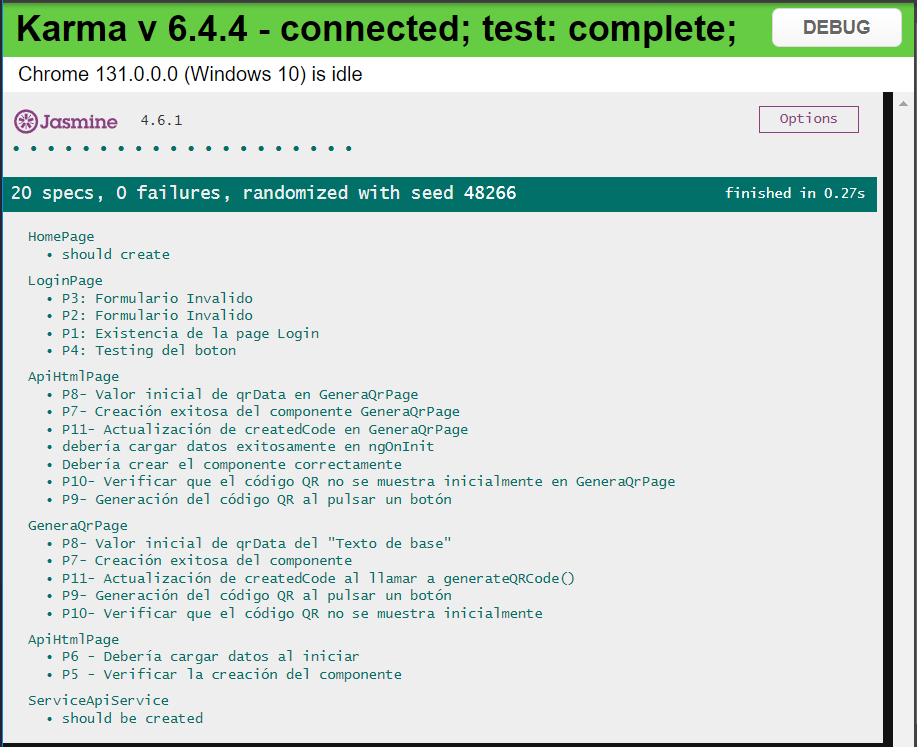
# Pruebas Unitarias

El inicio de las pruebas unitarias se dio como una etapa dentro del proceso de aprendizaje y desarrollo del equipo. Estas pruebas fueron diseñadas e implementadas con el objetivo de adquirir experiencia práctica en la evaluación de componentes y servicios dentro de aplicaciones desarrolladas con Ionic y Angular. Se eligió un proyecto diferente al principal para realizar estas pruebas, lo que permitió experimentar con un entorno controlado.

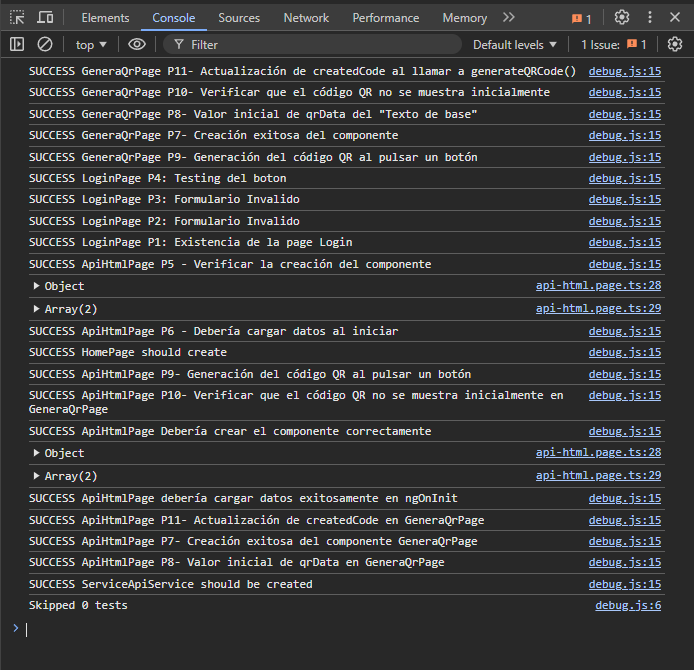
## *Detalle Terminal VSC:*



## *Detalle Jasmine:*



## *Detalle Consola de Inspección Browser:*



## *P5. Verificar la creación del componente*

* Objetivo: Confirmar que el componente se crea correctamente.
* Resultado esperado: La instancia del componente (*component*) no es null y se crea exitosamente.



## *P6. Cargar datos exitosamente en ngOnInit*

* Objetivo: Validar que el componente carga los datos desde el servicio y actualiza correctamente las propiedades *cantidad\_personajes* y personajes.
* Resultado esperado: *cantidad\_personajes* tiene el valor de *info.count*. y personajes contiene los objetos en el arreglo *results*.



## *P7: Creación exitosa del componente*

* Propósito: Verificar que el componente se crea correctamente.
* Prueba: expect(component).toBeTruthy();
* Resultado esperado: El componente debe instanciarse sin errores.



## *P8: Valor inicial de qrData*

* Propósito: Asegurarse de que el valor inicial de qrData es "Texto de base".
* Prueba:expect(component.qrData).toBe('Texto de base');
* Resultado esperado: qrData debe tener el valor "Texto de base" al inicio.



## *P9: Generación del código QR al pulsar un botón*

* Propósito: Comprobar que el método generateQRCode() asigna correctamente qrDataa createdCode.
* Prueba:component.qrData = 'Prueba QR'; component.generateQRCode(); expect(component.createdCode).toBe('Prueba QR');
* Resultado esperado: createdCode debe igualarse a qrData tras ejecutar el método.



## *P10: Verificar que el código QR no se muestra inicialmente*

* Propósito: Comprobar que el código QR no se renderiza antes de que el usuario lo genere.
* Prueba: const compiled = fixture.nativeElement; expect(compiled.querySelector('qrcode')).toBeNull();
* Resultado esperado: El elemento no debe estar presente en el DOM al inicio.



## *P11: Actualización de createdCode al llamar a generateQRCode()*

* Propósito: Verificar que generateQRCode() actualiza createdCode correctamente con nuevos datos.
* Prueba: component.qrData = 'Nuevo texto QR'; component.generateQRCode(); expect(component.createdCode).toBe('Nuevo texto QR');
* Resultado esperado: El valor de createdCode debe coincidir con el nuevo valor



# Conclusión

El desarrollo de RegistrApp fue un proceso con altibajos, caracterizado por un enfoque iterativo de aprendizaje y mejora constante. Cada versión de la aplicación fue refinada con base en los requerimientos del caso de estudio y las recomendaciones de nuestro docente. Este enfoque permitió al equipo identificar áreas de oportunidad, ajustar funcionalidades y optimizar el diseño, logrando así una herramienta que, si bien no alcanzó todas nuestras expectativas iniciales, se alinea plenamente con las necesidades estipuladas por el cliente.

A lo largo del proceso, cada iteración se centró en resolver problemáticas específicas planteadas en el caso. RegistrApp fue diseñada para abordar el principal "dolor" del cliente: la dificultad para llevar un control eficiente y preciso de la asistencia en entornos educativos. Cada nueva versión implementó mejoras significativas en términos de usabilidad, rendimiento y funcionalidad, evidenciando nuestra capacidad para adaptarnos a las demandas emergentes y a las restricciones del proyecto.

En definitiva, el resultado es una aplicación que cumple con los requisitos establecidos y que está diseñada para adaptarse fácilmente a nuevos contextos o integrarse con otros sistemas en el futuro. Esto garantiza que la herramienta no solo sea útil en el presente, sino que también sea sostenible y flexible a medida que evolucionen las necesidades del cliente.

# Conclusión sobre las Pruebas Unitarias Realizadas

Las pruebas implementadas durante este proyecto de pruebas se enfocaron en evaluar la correcta funcionalidad y robustez de distintos módulos, abordando tanto aspectos estructurales como dinámicos. Aunque estas pruebas se aplicaron a un proyecto distinto con fines de aprendizaje, permitieron consolidar conocimientos clave sobre la creación, implementación y validación de pruebas unitarias en aplicaciones desarrolladas con Ionic y Angular.

*Alcances de las Pruebas.*

***Componente ApiHtmlPage (P5 y P6):***

**P5:** Se verificó la creación exitosa del componente, asegurando que se instancie sin errores. Este tipo de prueba garantiza la estabilidad estructural inicial del módulo.

**P6:** Se comprobó que el componente es capaz de cargar datos desde una API externa (Rick and Morty) en su método ngOnInit. Esto aseguró que las propiedades se actualizan correctamente, validando la comunicación con el servicio y el manejo de la información.

***Componente GeneraQrPage (P7-P11):***

Estas pruebas abarcaron la funcionalidad de generación de códigos QR, desde la verificación de inicialización correcta hasta la actualización de datos y la representación en la interfaz de usuario. Además, incluyeron validaciones para garantizar que los datos procesados fueran reflejados correctamente en el DOM.

*Aprendizajes y Logros*

El aprendizaje obtenido se centró en dominar herramientas y metodologías esenciales para las pruebas unitarias, como la utilización de mocks para servicios, la simulación de flujos de datos, y la validación de interacciones con elementos de la interfaz. Estos conocimientos son transferibles y aplicables a proyectos futuros, reforzando la capacidad para garantizar la calidad y confiabilidad del software.

En conclusión, las pruebas realizadas no solo cumplieron con sus objetivos específicos en este contexto de aprendizaje, sino que también establecieron una base para aplicar principios de testing en proyectos reales, mejorando significativamente la capacidad para desarrollar aplicaciones confiables y funcionales.

# Bibliografía

* **Material visual proporcionado por el docente.** Plataforma AVA de Duoc UC. Disponible únicamente para estudiantes matriculados.
* **Material entregado clase a clase por el docente.** Documentación interna proporcionada en sesiones presenciales y virtuales, Duoc UC.
* **Herramientas de apoyo basadas en Inteligencia Artificial.** En algunos casos, se utilizó ChatGPT como soporte para aclaraciones conceptuales y técnicas, desarrollado por OpenAI.