Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Cámara ClarApp

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Eduardo Adam Climent

Tutor: David de Andrés Martínez, Juan de Ribera Serra Lluch

Curso 2018-2019

Resumen

Este trabajo se centra en el desarrollo de una aplicación móvil híbrida destinada a asistir a los artistas y otros profesionales en el momento del dibujo. La aplicación hace uso de la realidad aumentada para superponer en la cámara de un dispositivo móvil la imagen que se pretende reproducir sobre la superficie de dibujo. Se construye con el *framework* Ionic, implementando las funcionalidades de realidad aumentada a partir del *plugin* de Wikitude para *frameworks* basados en Cordova. Es una colaboración con el Departamento de Expresión Grafica Arquitectónica de la Universitat Politècnica de València.

Palabras clave: realidad aumentada, ionic, híbrida, arquitectura, dibujo.

Abstract

This thesis focuses on the development of a hybrid mobile application aimed at assisting artists at the moment of drawing. The application makes use of augmented reality to overlap on the camera of a mobile device the image that is intended to be reproduced on the drawing surface. It is built with the Ionic framework, implementing the augmented reality functionalities from the Wikitude plugin for frameworks based on Cordova. It is a collaboration with the Architectural Graphic Expression Department of the Universitat Politècnica de València.

Keywords : augmented reality, ionic, hybrid, architecture, drawing.

Tabla de contenidos

1. Introducción

En este primer apartado de la memoria se introduce el presente trabajo, describiendo su motivación, contexto, objetivos y finalmente la estructura de este documento.

1.1 Motivación

Una situación común a la que se enfrentan artistas y otros profesionales dedicados al dibujo es la reproducción exacta de la perspectiva del objeto que se pretende reproducir sobre una superficie. Para facilitar este proceso se patenta en el siglo XIX un dispositivo óptico analógico llamado cámara clara o cámara lúcida[1,2].

La cámara lúcida consta de un par de espejos que reflejan la imagen que se pretende dibujar de forma que queda superpuesta al papel. Gracias a este fenómeno el artista puede recrear perspectivas, transportar puntos y ayudarse con las tonalidades si las condiciones lo permiten.

Ya entrados en siglo XXI es incuestionable el auge de los dispositivos móviles, los *smartphones* y las *tablets*, que suelen acompañar a las personas en su día a día. Estos dispositivos cuentan con procesadores y cámaras cada vez más potentes, que combinados permiten desarrollar herramientas basadas en la realidad aumentada.

Tomando la necesidad de reproducir perspectivas de manera exacta y eficiente, la idea de la cámara clara analógica y las tecnologías disponibles en el momento de desarrollar este trabajo surge ClarApp, una cámara clara digital basada en la realidad aumentada.

1.2 Contexto

En la primera mitad del año 2018 podemos encontrar unas 3.600.000 aplicaciones disponibles en la Play Store de Google[3]y otras 2.450.000 en la Apple Store[4].

Cada día son más usuarios los que buscan solventar sus problemas diarios u optimizar situaciones rutinarias mediante aplicaciones móviles. En consecuencia se abre un amplio abanico de posibilidades para los desarrolladores, que además de proponer nuevas ideas nativas de estos dispositivos, pueden adaptar o reinventar soluciones analógicas o tediosas al mundo de los *smartphones*. Así podemos tener siempre a mano una calculadora de bolsillo, una cámara de fotos, un bloc de notas, un navegador *GPS* para nuestro vehículo e infinidad de herramientas específicas para ciertos trabajos.

Como comentamos en el apartado anterior, en el siglo XIX se patenta la Cámara clara o Cámara lúcida. Esta requiere de un espejo semitransparente y una lente negativa. Además es preferible realizar el dibujo sobre una superficie negra con trazo blanco. Toda esta parafernalia hace del invento algo poco accesible, poco portátil, poco versátil y por tanto de poco uso.

¿Por qué no recuperar y adaptar esta buena idea a las tecnologías disponibles en la segunda década del siglo XXI? El autor considera que puede ser interesante dar la posibilidad de llevar en el bolsillo esta herramienta a profesionales del dibujo, artistas y dado que este proyecto se realiza en un contexto académico, al alumnado del Grado en

Arquitectura y del Grado en Fundamentos de la Arquitectura de la Universitat Politècnica de València.

El desarrollo de aplicaciones móviles

Una vez planteado el interés que podría tener el desarrollo de una aplicación móvil nos encontramos con un abanico de opciones de implementación, cada una con sus ventajas y desventajas.

Principalmente disponemos de tres grandes tipos de aplicaciones: nativas, web e híbridas. Se describen a continuación.

Aplicaciones nativas

Las aplicaciones nativas son específicas de cada sistema operativo, por lo que requieren de expertos en la plataforma (Android, iOS, Windows...). Esta concreción hace de su desarrollo un procedimiento en muchos casos caro y costoso, pero el resultado puede proporcionar una muy buena experiencia de usuario además de dar acceso a los desarrolladores a todas las funcionalidades del dispositivo. Su rendimiento es muy bueno. Se pueden distribuir en las *stores*.

Aplicaciones web

Las aplicaciones web se ejecutan en el navegador del usuario, generalmente cargadas desde un servidor externo. Se desarrollan con las mismas tecnologías que un sitio web al uso, es decir, HTML, CSS y Javascript. Estas tecnologías combinadas con el navegador permiten construir aplicaciones *responsive*, es decir, que se adapten a la resolución de prácticamente cualquier dispositivo sin demasiado esfuerzo por parte del desarrollador. Se consigue una aplicación en principio muy portable entre plataformas.Al ejecutarse sobre el navegador web no se tiene acceso a todas las funcionalidades del dispositivo y hay un menor control sobre la seguridad. Estas aplicaciones precisan de menos tiempo y coste de desarrollo que las aplicaciones nativas. El rendimiento no es tan bueno.

Aplicaciones híbridas

Las aplicaciones híbridas combinan lo mejor de las dos anteriores. Se desarrollan también con tecnologías web (HTML, CSS y Javascript), pero esta vez se encapsulan dentro de un contenedor para ser ejecutadas como aplicaciones nativas, además pueden ser distribuidas en las *stores*. Tienen acceso a todas las funcionalidades del dispositivo mediante Javascript y su código es el mismo en todas las plataformas. Como en las aplicaciones web, su tiempo y coste de desarrollo es más razonable que en las aplicaciones nativas y se puede conseguir su mismo rendimiento si es necesario.

1.3 Objetivos

El principal objetivo de este trabajo es implementar una aplicación móvil destinada a ayudar a los artistas en el momento de reproducir una imagen mediante el dibujo a mano alzada. La aplicación permitirá visualizar el objeto a dibujar superpuesto a la superficie de dibujo a través de la cámara de su *smartphone* o *tablet*.

El usuario será capaz de añadir imágenes tanto de la galería como de la cámara del dispositivo y ajustar su opacidad para facilitar el proceso de dibujo. La imagen superpuesta se adaptará a la perspectiva de la superficie de dibujo, haciendo uso de tecnologías de realidad aumentada.

La implementación se llevará a cabo utilizando el *framework* Ionic, que proporciona la base para construir una aplicación híbrida, combinado con el *plugin* para *frameworks* basados en Cordova de Wikitude, que facilita el trabajo con tecnología de realidad aumentada. Más adelante se comentarán estas tecnologías en más profundidad.

En un plano más personal, es un objetivo del autor mejorar sus habilidades en el uso de tecnologías web y el desarrollo de aplicaciones móviles. Ionic es un framework muy adecuado, ya que combina HTML5, Typescript, CSS y Sass con el empaquetado en forma de aplicación móvil y el acceso a *APIs* nativas de los dispositivos.

1.4 Estructura de la memoria

Este documento está estructurado en lossiete bloques descritos a continuación:

1. **Introducción**: se plantea la motivación del proyecto, se pone en contexto y se explican sus objetivos.

2. **Estado del arte**: se revisan aplicaciones que cubren necesidades similares a las que se pretende dar respuesta.

3. **Especificación**: se presenta la arquitectura global de la solución propuesta, incluyendo las tecnologías y herramientas que se utilizarán.

4. **Implementación**: se presentan los aspectos relevantes del proceso de implementación, incluyendo codificación, herramientas, *frameworks*, problemas y alternativas.

5. **Resultados**: se muestra el estado final de la aplicación a través de capturas de pantalla y resultados de test.

6. **Conclusiones y trabajo futuro**: se analiza el grado de cumplimiento de los objetivos a partir de los resultados y se plantean posibles vías de crecimiento.

7. **Referencias**: se incluyen las referencias citadas y utilizadas a lo largo del documento.

2. Estado del arte

Podemos encontrar una amplia variedad de aplicaciones móviles basadas en la realidad aumentada o que hacen uso de esta, sin embargo, no hay un gran catálogo en lo que se refiere a asistentes de dibujo. A continuación se analiza la única herramienta reseñablemente similar a lo que se pretende conseguir con ClarApp.

2.1 Herramientas similares: SketchAR

2.1.1 Descripción

La única herramienta para asistir en el dibujo a mano alzada disponible en las tiendas de aplicaciones de Google y Apple en el momento de redactar este documento es SketchAR[5].

En el sitio web del producto (sketchar.tech) se describe como una herramienta completa para enseñar a dibujar usando realidad aumentada, aprendizaje automático y redes neuronales.

2.1.2 Principales funcionalidades

Esta herramienta permite seleccionar una imagen de la galería del dispositivo, de la cámara o de una colección propia de la aplicación como fuente para realizar el dibujo. Una vez seleccionada le aplica un filtro de detección de bordes y la convierte en un conjunto de trazos de color azul, que serán la imagen de referencia que aparecerá sobre la cámara del usuario.

Para comenzar a dibujar se deben trazar cuatro cruces de aproximadamente un centímetro sobre un folio de tamaño A4 o A5. Dos cruces se ubicarán en la parte superior del folio y las otras dos en la parte izquierda o derecha, dependiendo de si el usuario es diestro o zurdo respectivamente. Estas cruces hacen la función de marcadores para estabilizar la imagen virtual y ajustar la perspectiva.

Ya establecidos correctamente los marcadores se superpone la imagen de referencia sobre la vista de la cámara, permitiendo al usuario alterar sus dimensiones y su posición. Entonces el usuario podrá comenzar a dibujar.

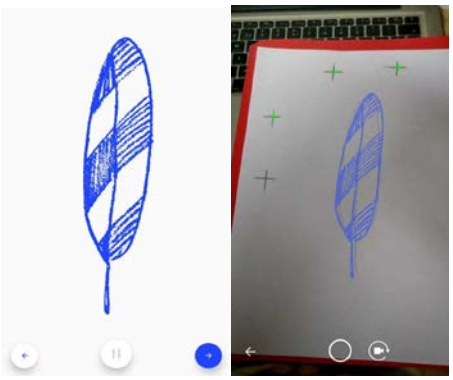

Ilustración 1: Imagen de referencia y vista de Realidad Aumentada en SketchAR

Si lo desea el usuario puede realizar capturas de pantalla sin las marcas virtuales durante la sesión de dibujo.

Además, la aplicación ofrece un conjunto de lecciones de dibujo paso a paso partiendo de la base descrita en este apartado.

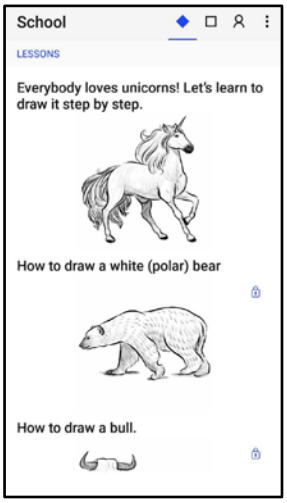

Ilustración 2: Ejemplos de lecciones de dibujo ofertadas por SketchAR

Si el usuario dispone de un dispositivo con ARCore de Google o ARKit de Apple puede hacer uso de una funcionalidad especialmente adaptada para dibujar murales sobre pared.

También está disponible para uso con gafas de realidad aumentada, concretamente para las HoloLens de Microsoft.

2.2 Análisis

En este apartado se comentarán las impresiones causadas por la aplicación SketchAR tras su instalación y test por parte del autor y la lectura de las opiniones de los usuarios en la tienda de Google.

2.2.1 Experiencia del autor

Una vez instalada la aplicación el usuario debe visualizar un vídeo introductorio que no se puede saltar, lo que puede resultar molesto cuando ya conoces su funcionamiento por haberla utilizado en otras ocasiones o dispositivos.

En el momento de tratar de dibujar resulta bastante complicado encuadrar la imagen en el folio para comenzar, aun habiendo situado las cuatro cruces marcadoras en su posición. Si bien es cierto que una vez comienzas a dibujar, la aplicación interpreta los nuevos trazos como marcadores y la imagen se vuelve algo más estable, aunque parece aumentar la carga a procesar, ya que se retarda la adaptación de la imagen a los cambios de ángulo del dispositivo respecto al folio.

El autor ha realizado tres pruebas y en solo una se pudo comenzar a dibujar, ya que en las dos restantes el sistema no consiguió fijar la imagen al folio.

2.2.2 Experiencias de los usuarios de Google Play

En el momento de redactar este documento la valoración de la aplicación en la tienda de Google era de 3'5 sobre 5.

Es una constante en las reseñas la dificultad para encuadrar la imagen a pesar de que los usuarios consideran que lo intentan en buenas condiciones, tanto lumínicas, como técnicas de sus dispositivos. En este punto se coincide con la experiencia del autor.

No aparecen menciones de los usuarios a las lecciones de dibujo que se proporcionan.

Los usuarios que dejan comentarios positivos la encuentran adecuada para realizar un primer esbozo de lo que se pretende dibujar.

2.3 Conclusión

SketchAR presenta características muy interesantes. En base al análisis realizado y los objetivos de este proyecto se han priorizado sus características como requisitos a implementar en ClarApp utilizando el método MoSCoW.

Tabla 1: Priorización de requisitos con método MoSCoW

Mo – Must have	• Superponer imagen de referencia sobre la cámara • Galería del dispositivo como fuente de imágenes • Ajuste de opacidad de la imagen de referencia • Ajuste de tamaño de la imagen de referencia • Ajuste de perspectiva
S – Should have	• Marcadores para mejorar encuadre y perspectiva • Situar marcadores en función de la mano de dibujo (diestro – zurdo) • Ajuste automático de perspectiva • Cámara del dispositivo como fuente de imágenes • Panel con los proyectos comenzados
Co – Could have	• Filtro de detección de bordes para facilitar el trazado • Filtro monocromo para facilitar el trazado
W – Won't have	• Colección de imágenes prediseñadas • Lecciones de dibujo • Desarrollo específico orientado a murales • Desarrollo específico orientado a gafas de realidad aumentada

Las filas de la tabla presentan las características de más a menos prioritarias.

SketchAR está enfocada al aprendizaje del dibujo, por lo que aunque hay requisitos coincidentes, no todos son relevantes para ClarApp.

El objetivo de este proyecto es presentar una herramienta para asistir el dibujo lo más sencilla posible, por lo que se descarta incluir lecciones de dibujo e imágenes prediseñadas. También se descartan los desarrollos especialmente orientados a murales o a gafas de realidad aumentada, dejándolos tal vez para el futuro.

3. Especificación

En este apartado se describe la arquitectura global de la aplicación desde el punto de vista funcional.

Será una aplicación a tres capas. A continuación se presenta cada una de las capas partiendo de la de persistencia, siguiendo con la capa de negocio y terminando con la capa de presentación.

3.1 Capa de persistencia

La persistencia de la aplicación tendrá lugar en el almacenamiento en el dispositivo del usuario utilizando la interfaz *localStorage* de la API de almacenamiento web.

Los datos se almacenan en *localStorage* como objetos en formato *JSON*.

Los objetos a almacenar representarán el histórico de dibujos realizados por el usuario, la mano de dibujo y el dibujo que está realizando en ese momento el usuario.

Se detalla el modelo de datos a utilizar:

3.1.1 Mano de dibujo

Almacena únicamente un valor binario indicando izquierda o derecha, según la preferencia del usuario

Tabla 2: Formato de la mano de dibujo

Mano
mano: Integer

3.1.2. Dibujos

Almacena un array de objetos con la información de cada dibujo del usuario, de forma que el usuario pueda continuar el dibujo con los parámetros que dejó.

Tabla 3: Formato de dibujo

Dibujo
ruta: String
tamano: Double
rotacion: Double
posicionX: Double
posicionY: Double
fecha: Integer

3.1.3 Dibujo actual

Almacena la información del dibujo que se está manipulando en un momento dado, el índice en la colección de dibujos y la ruta de la unidad de almacenamiento en el dispositivo.

Tabla 4: Formato de dibujo actual

Dibujo actual
dibujo: Dibujo
indice: Integer
almacenamiento: String

3.2 Capa de negocio

En este subapartado se presentan los diferentes casos de uso que se abordarán en la lógica de la aplicación.

3.2.1 Selección inicial de mano de dibujo (CU-01)

Tabla 5: CU-01

CU-01	Selección inicial de mano de dibujo
Objetivos	Especificar con qué mano dibujará el usuario al usar la aplicación
Descripción	El usuario podrá especificar que dibuja con la mano derecha o izquierda
Precondición	Es la primera vez que el usuario inicia la aplicación, es decir, no hay datos almacenados del usuario
Secuencia normal	1. El usuario inicia la aplicación por primera vez 2. Se pregunta al usuario con qué mano dibujará y se muestra un selector con las opciones 3. El usuario selecciona su preferencia 4. El usuario confirma sus preferencias
Postcondición	Se registra la selección del usuario. El usuario es dirigido a la vista principal. Se informa al usuario de dónde se encuentra la ayuda de la aplicación.
Excepciones	Ninguna
Comentarios	Ninguno

3.2.2 Comenzar dibujo (CU-02)

Tabla 6: CU-02

CU-02	Comenzar dibujo
Objetivos	Comenzar un nuevo dibujo
Descripción	El usuario podrá cargar la imagen que pretende dibujar y se realizarán los ajustes iniciales
Precondición	El usuario ha especificado su mano de dibujo
Secuencia normal	1. El usuario selecciona la imagen que desea desde la galería o la cámara del dispositivo, haciendo clic en el icono correspondiente desde la vista principal 2. La imagen se muestra superpuesta a la vista de la cámara del dispositivo junto a los controles de opacidad, tamaño y rotación en una nueva vista 3. El usuario realiza los ajustes iniciales de opacidad, tamaño, rotación y posición
Postcondición	Se guarda la imagen del usuario y sus ajustes iniciales.
Excepciones	Ninguna
Comentarios	Ninguno

3.1.3 Editar ajustes de imagen (CU-03)

Tabla 7: CU-03

CU-03	Editar ajustes de imagen	
Objetivos	Editar los ajustes de la imagen con la que se está trabajando	
Descripción	El usuario podrá editar los ajustes actuales de posición, tamaño y rotación de la imagen con la que trabaja	
Precondición	El usuario ha comenzado un nuevo dibujo	
Secuencia normal	1. El usuario pulsa el botón de ajustes mientras trabaja con una imagen 2. Aparecen los controles de ajustes 3. El usuario ajusta los controles según sus preferencias	
Postcondición	Se guarda la imagen del usuario y sus nuevos ajustes.	
Excepciones	**Paso**	**Acción**
	3	Si el usuario pulsa el botón de cancelar desaparecen los controles de ajustes, manteniendo la configuración actual
	4	Si el usuario pulsa el botón de cancelar desaparecen los controles de ajustes, manteniendo la configuración actual
Comentarios	Ninguno	

3.1.4 Cambiar mano de dibujo (CU-04)

Tabla 8: CU-04

CU-04	Cambiar mano de dibujo	
Objetivos	Especificar con qué mano dibujará el usuario al usar la aplicación	
Descripción	El usuario podrá editar su especificación actual de mano de dibujo	
Precondición	El usuario ha especificado su mano de dibujo	
Secuencia normal	1. El usuario despliega el menú lateral y pulsa el botón de preferencias 2. Se abre una nueva vista y se pregunta al usuario con qué mano dibujará y se muestra un selector con las opciones 3. El usuario selecciona su preferencia	
Postcondición	Se guardan las preferencias del usuario	
Excepciones	**Paso**	**Acción**
	4	Si el usuario abandona la vista sin modificar sus preferencias se mantienen las actuales
Comentarios	Ninguno	

3.1.5 Mostrar ayuda de aplicación (CU-05)

Tabla 9: CU-05

CU-05	Mostrar ayuda de aplicación
Objetivos	Mostrar la información de ayuda de la aplicación
Descripción	El usuario podrá visualizar una página con la información necesaria para conocer el funcionamiento completo de la aplicación en caso de duda.
Precondición	El usuario ha especificado su mano de dibujo
Secuencia normal	1. El usuario despliega el menú lateral y pulsa el botón de ayuda 2. El usuario accede a una vista con unalista de enlaces que contiene una entrada por cada tema de ayuda y más abajo el texto correspondiente a todos los temas 3. El usuario pulsa en un tema
Postcondición	Se despliega el tema seleccionado
Excepciones	Ninguna
Comentarios	Ninguno

3.1.6 Continuar dibujo (CU-06)

Tabla 10: CU-06

CU-06	Continuar dibujo
Objetivos	Continuar un dibujo ya comenzado
Descripción	El usuario podrá reabrir un dibujo ya comenzado con sus ajustes de posición, tamaño y rotación almacenados
Precondición	El usuario ha comenzado al menos un dibujo
Secuencia normal	1. El usuario hace clic en el icono correspondiente a continuar dibujo de la vista principal o despliega el menú lateral y pulsa el botón de continuar dibujo 2. El usuario accede a una vista con su colección de dibujos comenzados 3. El usuario pulsa en uno de los dibujos 4. El usuario accede a la vista de dibujo con el dibujo seleccionado y se cargan sus ajustes guardados
Postcondición	Ninguna
Excepciones	Ninguna
Comentarios	Ninguno

3.3 Capa de presentación

A continuación se presentan los mockups correspondientes a cada caso de uso visto en el subapartado anterior, describiendo así cómo percibirá el usuario la presentación de la aplicación

3.2.1 CU-01 - Selección inicial de mano de dibujo

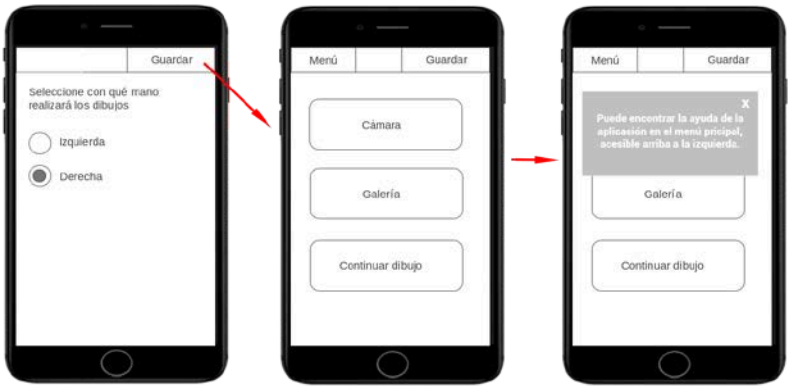

Ilustración 3: Mockup CU-01

3.2.2 CU-02 – Comenzar dibujo

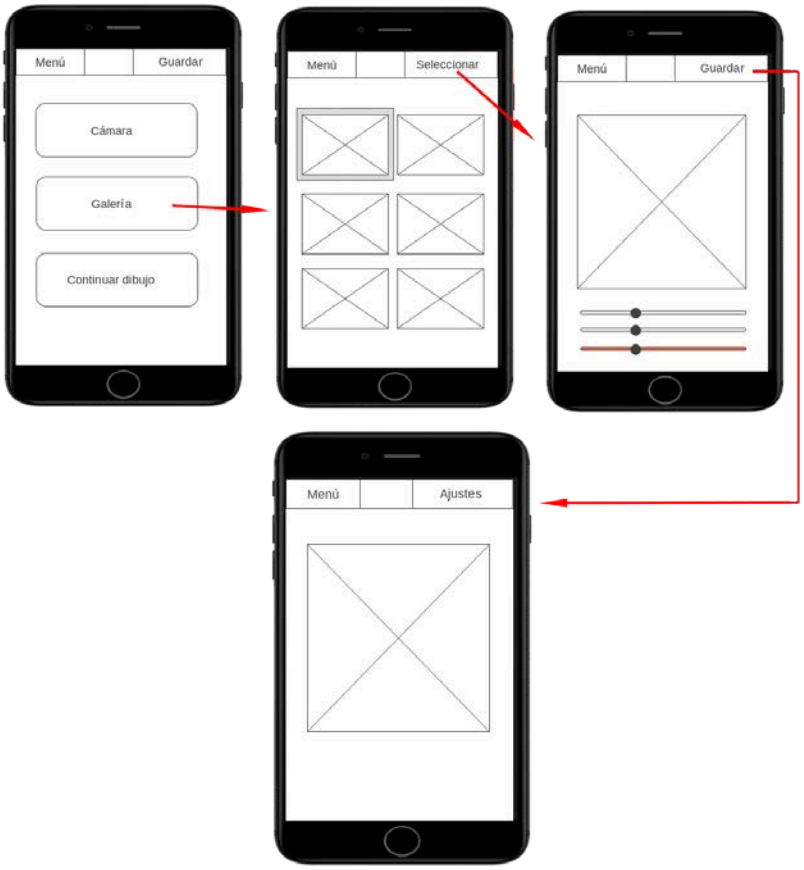

Ilustración 4: Mockup CU-02

3.2.3 CU-03 – Editar ajustes de imagen

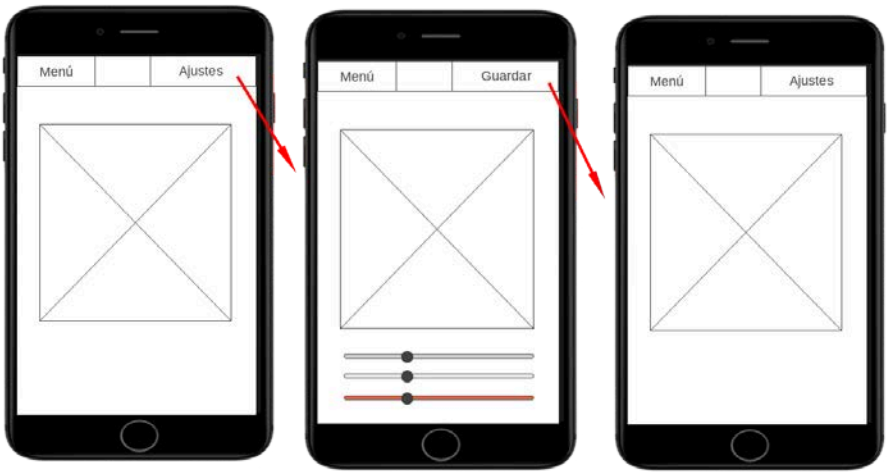

Ilustración 5: Mockup CU-03

3.2.4 CU-04 – Cambiar mano de dibujo

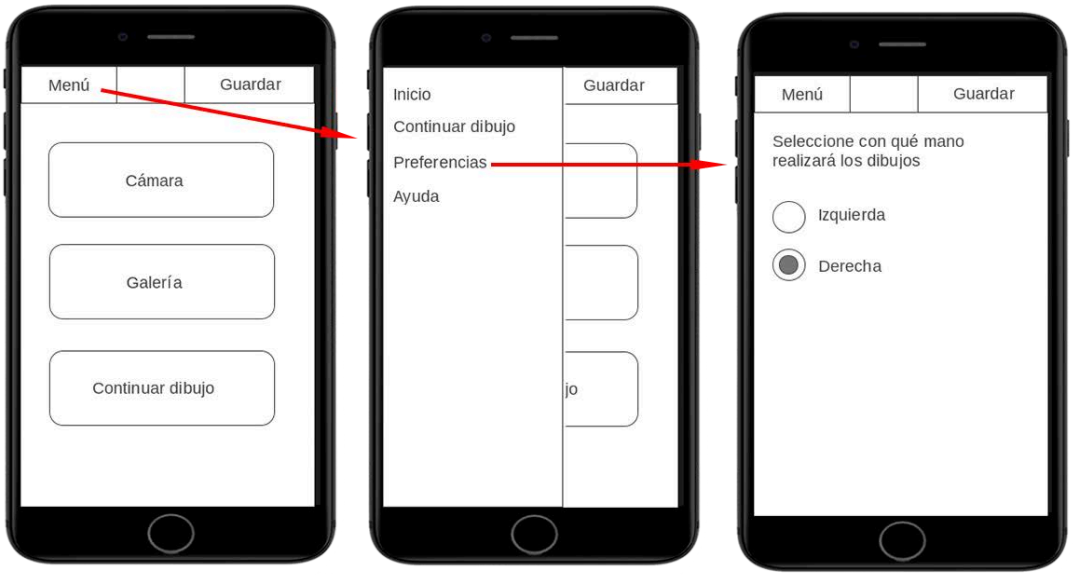

Ilustración 6: Mockup CU-04

3.2.5 CU-05 – Mostrar ayuda de aplicación

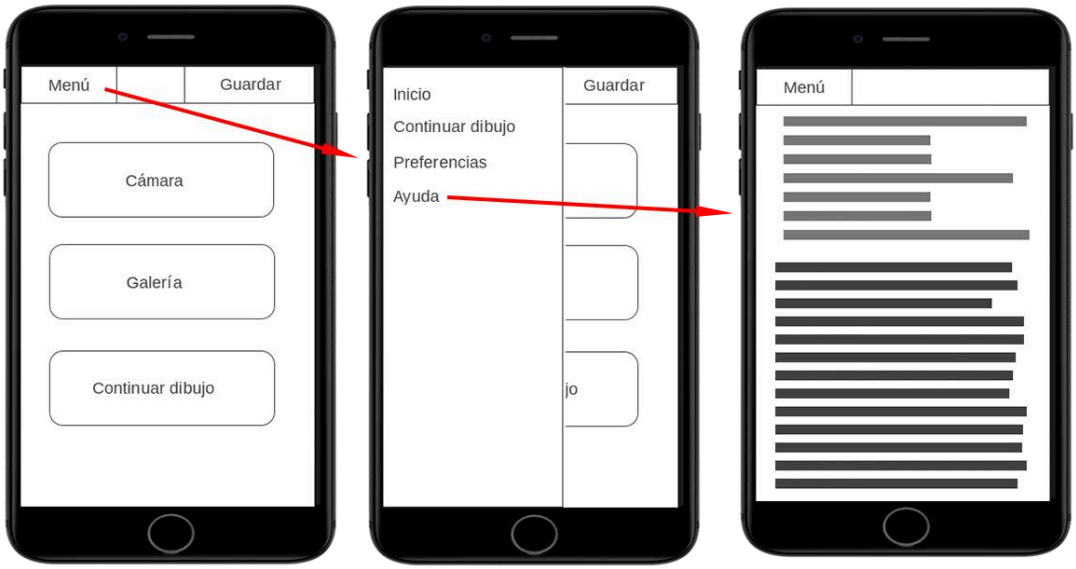

Ilustración 7: Mockup CU-05

3.2.6 CU-06 – Continuar dibujo

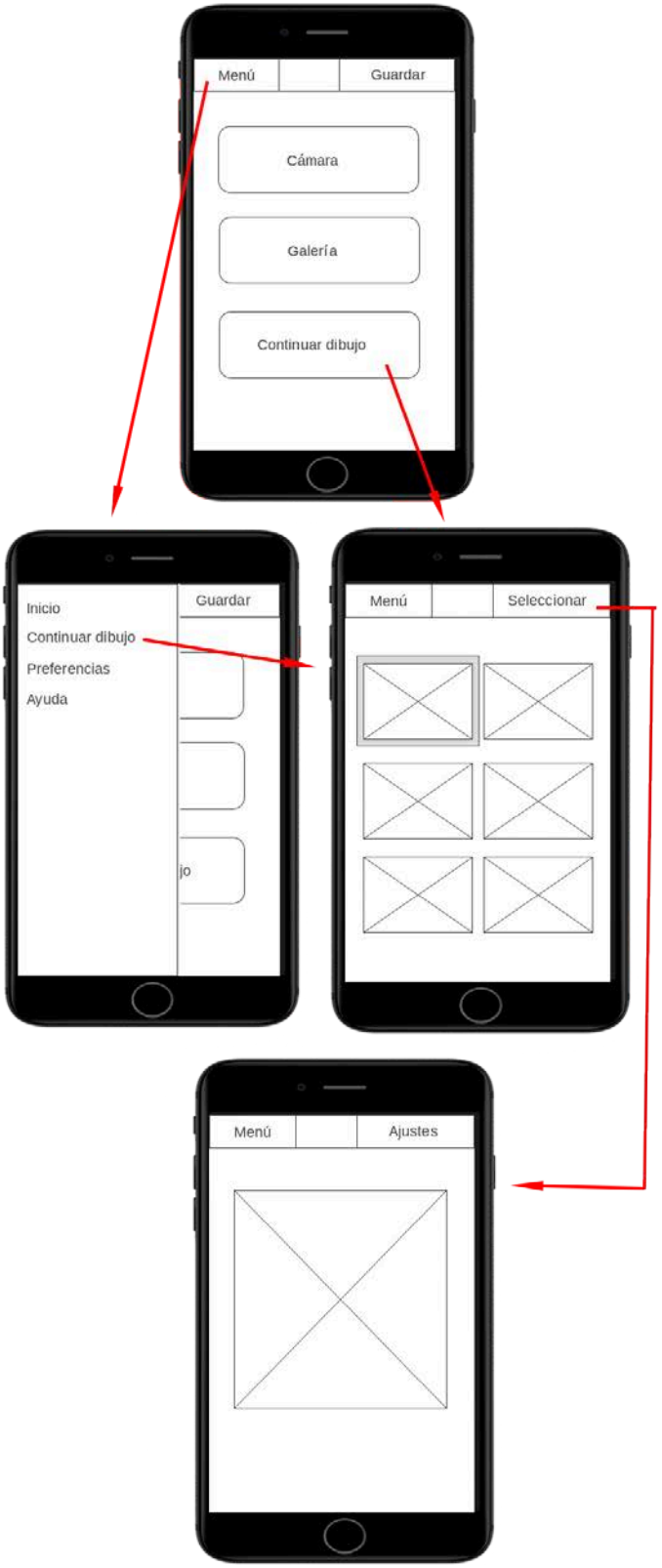

Ilustración 8: Mockup CU-06

4. Implementación

En este apartado se describen las tecnologías y herramientas utilizadas en el proyecto, así como el proceso de implementación de la aplicación.

4.1. Tecnologías

Como se ha expuesto anteriormente la aplicación se desarrolla utilizando el *framework* Ionic, combinado con el plugin Wikitude de realidad aumentada para aplicaciones basadas en Cordova. A continuación se presentan estas tecnologías.

4.1.1 Tecnologías web

En primer lugar se describen las tres tecnologías web elementales si se habla del *Frontend*, la parte de la aplicación con la que estará en contacto el usuario final, ya que se hace referencia a ellas en los siguientes puntos.

Ilustración 9: Logotipo HTML

HTML

Hypertext Markup Languaje(Lenguaje de Marcas de Hipertexto). Es el lenguaje utilizado para maquetar los documentos web a partir de etiquetas. Es decir, permite ordenar los diferentes elementos de una página web. Se encuentra en su versión 5.

Ilustración 10: Logotipo CSS

CSS

Cascading Style Sheets (Hojas de Estilo en Cascada). Es el lenguaje empleado para dar estilo a los elementos de un documento web que ha sido maquetado con HTML. Es el encargado del diseño visual (colores, tipografías, alineamientos, posiciones, tamaños...). Se encuentra en su versión 3.

Ilustración 11: Logotipo JvaScript

JavaScript

Es un lenguaje de programación no compilado, orientado a objetos y débilmente tipado, lo que lo hace muy versátil y dinámico.

4.2.1 Ionic

Ilustración 12: Logotipo Ionic

Ionic[6] es un *framework* que recopila una serie de librerías y componentes de interfaz de usuario construidos con tecnologías web, es decir HTML, CSS y JavaScript. Se puede emplear en conjunto con cualquier *framework* JavaScript, como es Angular en el caso de este trabajo. Con este completo *framework* es posible diseñar aplicaciones híbridas. Se encuentra en su versión 3.

4.2.2 Angular

Ilustración 13: Logotipo Angular

Angular[7] es un *framework* de JavaScript destinado a facilitar el desarrollo de aplicaciones SPA multiplataforma.

Una aplicación SPA[8] (*Simple-Page Application*) es una aplicación web de una sola página que va cargando el contenido de las diferentes secciones de forma dinámica, de esta forma al navegar no se cambia de página, se cambia el contenido de la misma. Esto proporciona una muy buena experiencia de usuario, ya que no hay recargas en el navegador, pero desarrollarlo puede ser complejo. Por esto resulta tan útil Angular.

Para la codificación emplea TypeScript. TypeScript es un lenguaje que añade a JavaScript un fuerte tipado y objetos basados en clases. Es mantenido por Microsoft. Se encuentra en su versión 6.

4.2.3 Apache Cordova

Ilustración 14: Isotipo Cordova

Si bien Ionic proporciona los componentes necesarios para que una aplicación desarrollada con tecnologías web tenga la apariencia de una aplicación móvil nativa,y Angular nos facilita la programación web, aun se requiere de un agente que realice el empaquetado y posibilite la comunicación de los componentes web con las apis nativas de los dispositivos. Esto es Apache Cordova[9]. Un *framework* de desarrollo de aplicaciones móviles encapsulando tecnologías web.

4.2.4 Wikitude SDK Cordova

Ilustración 15: Logotipo Wikitude

Wikitude SDK[10]provee de las herramientas necesarias para desarrollar aplicaciones basadas en Realidad Aumentada. Permite reconocer objetos, escenas e imágenes, además de seguir marcadores en tiempo real con la cámara del dispositivo, dando la posibilidad de superponer imágenes sobre cualquier superficie. Además se complementa con servicios basados en la ubicación.

Wikitude dispone de un *plugin* especialmente adaptado a aplicaciones construidas sobre Apache Cordova.

4.2 Herramientas

A continuación se presentan las herramientas utilizadas en el proceso de implementación.

4.2.1 Atom

Ilustración 16: Isotipo Atom

Atom es el editor escogido para la codificación de ClarApp.

Es un potente editor de código multiplataforma con soporte para una gran cantidad de lenguajes de programación. Es de código abierto y ha sido desarrollado por GitHub, por lo que se puede integrar perfectamente con Git para la gestión de versiones, aunque no se ha utilizado esta funcionalidad en el presente proyecto.

Cuenta además con un amplio catálogo de paquetes que ofrecen la posibilidad de agregarle funcionalidades adicionales. En el desarrollo de ClarApp se han utilizado esencialmente dos:

- *language-typescript*: proporciona soporte para trabajar con TypeScript en Atom.
- *atom-beautify*: ordena correctamente el indentado y las líneas de código para facilitar su lectura.

Otro de sus aspectos positivos es que utiliza una gama de colores para el marcado de los elementos en el código que no resultan cansados para la vista y ayudan a la lectura, incluyendo el color oscuro de fondo.

4.2.2 Node

Node es un entorno de ejecución para JavaScript. Por lo general se utiliza para construir servicios web del lado del servidor y herramientas para desarrolladores.

4.2.3 npm

Ilustración 17: Logotipo npm

npm es un gestor de paquetes de JavaScript.

Permite instalar y gestionar desde la línea de comandos del sistema los paquetes necesarios para el desarrollo del proyecto.

En el caso del presente proyecto se ha utilizado para instalar desde el propio Ionic hasta los diferentes plugins utilizados en la aplicación. En la explicación de la implementación se darán más detalles acerca de estos plugins y cómo se han instalado utilizando npm.

4.2.4 Ionic CLI

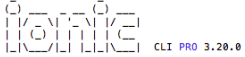

Ilustración 18: Cabecera Ionic CLI

Ionic CLI (por las siglas en ingles de Interfaz de Líneas de Comandos) es una herramienta de Ionic que trabaja sobre la consola de comandos del sistema. Su función es facilitar la realización de tareas comunes en un entorno de desarrollo de Ionic mediante ordenes escritas.

Algunas de las tareas más destacables[11] en las que asiste y que se han utilizado en este proyecto son:

- Creación de nuevos proyectos
- Generación de componentes, páginas, servicios...
- Despliegue de un servidor local de desarrollo de Ionic
- Instalación y uso del CLI de Cordova
- Preparación la aplicación para las plataformas objetivo

```
MacBook-Edu:ARTest Eduardo$ ionic serve
Starting app-scripts server: --address 0.0.0.0 --port 8100 --livereload-port
35729 --dev-logger-port 53703 --nobrowser - Ctrl+C to cancel
[22:41:35]  watch started ...
[22:41:35]  build dev started ...
[22:41:35]  clean started ...
[22:41:35]  clean finished in 36 ms
[22:41:35]  copy started ...
[22:41:36]  deeplinks started ...
[22:41:37]  deeplinks finished in 96 ms
[22:41:37]  transpile started ...
[22:41:51]  transpile finished in 14.92 s
[22:41:51]  preprocess started ...
[22:41:51]  preprocess finished in 1 ms
[22:41:51]  webpack started ...
[22:42:06]  copy finished in 30.51 s
[22:42:06]  webpack finished in 14.55 s
[22:42:06]  sass started ...
```

Ilustración 19: Primeras líneas de salida de una ejecución de "ionic serve"

Se puede encontrar una lista completa de los comandos disonibles en el sitio web de Ionic: https://ionicframework.com/docs/cli/commands.html

4.2.5 Google Chrome

El potente navegador de Google no solo es útil para visitar sitios web, proporciona una gran cantidad de herramientas para desarrollo web. Chrome, más que una herramienta, es un kit de herramientas.

Al ser ClarApp una app híbrida y por tanto construida fundamentalmente con tecnologías web, las herramientas de Google Chrome son de una enorme ayuda para su desarrollo. Se puede acceder a las herramientas desde el menú de Chrome, en el apartado de "Más herramientas" > "Herramientas para desarrolladores". Hay que destacar que estas herramientas son de utilidad para el desarrollo de aplicaciones Android, pero con dispositivos iOS resultan limitadas, aunque en Safari encontramos algunas similares.

Las principales utilidades empleadas en este trabajo son:

Inspector de página

Permite seleccionar con el ratón elementos de una página web e inspeccionar y modificar su código HTML.

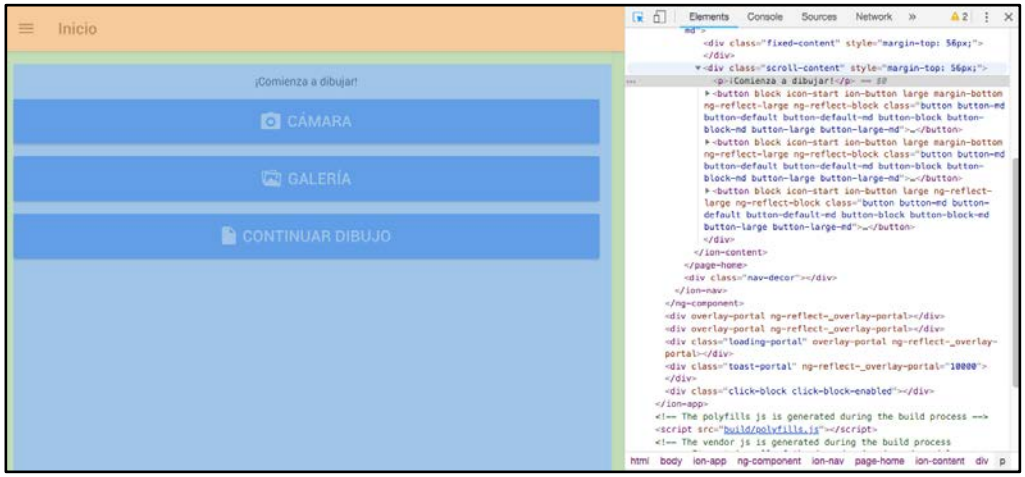

Ilustración 20: Selección de elemento HTML con el Inspector de Chrome

Consola

Permite ver los logs lanzados desde el código JavaScript de una aplicación web y ejecutar código JavaScript. Es fundamental para llevar la traza de la de la aplicación, ya que JavaScript es un lenguaje no compilado, por lo que puede ser difícil detectar errores de ejecución.

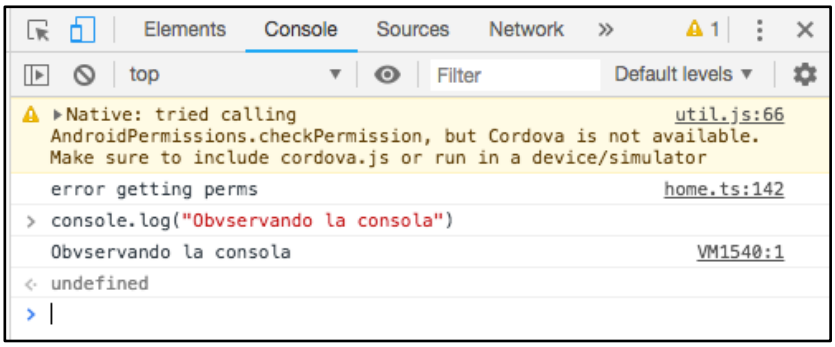

Ilustración 21: Colsola de Chrome

Barra de herramientas de dispositivo (Device Toolbar)

Ofrece la posibilidad de redimensionar la ventana del navegador adaptándola a la resolución de dispositivos predeterminados o personalizados. Resulta especialmente útil para visualizar la interfaz de la aplicación a la resolución de un smartphone o una tablet en tiempo de desarrollo.

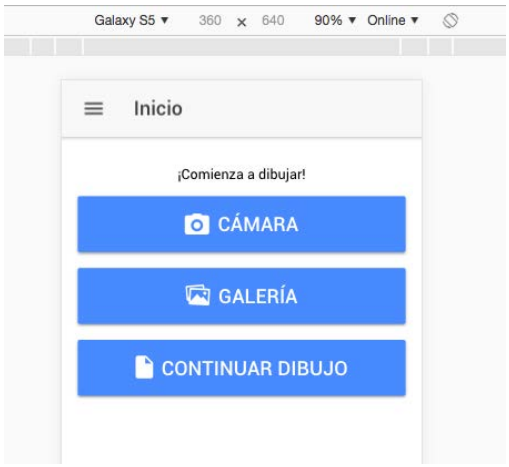

Ilustración 22: Emulación de la resolución de un Samsung Galaxy S5 con Device Toolbar

Inspector de dispositivos

Permite ejecutar las herramientas de desarrollador de Chrome sobre dispositivos conectados al ordenador por USB. Las herramientas mencionadas anteriormente son de gran ayuda antes de exportar la aplicación, pero una vez está instalada en el dispositivo es necesario utilizar este inspector para ver los resultados de las pruebas en tiempo real o interactuar con el código para observar comportamientos que no se han podido detectar en el entorno de desarrollo.

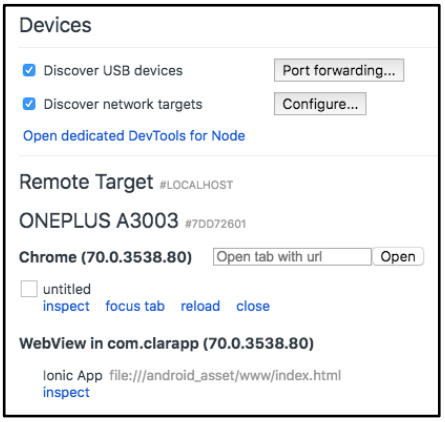

Ilustración 24: Inspector

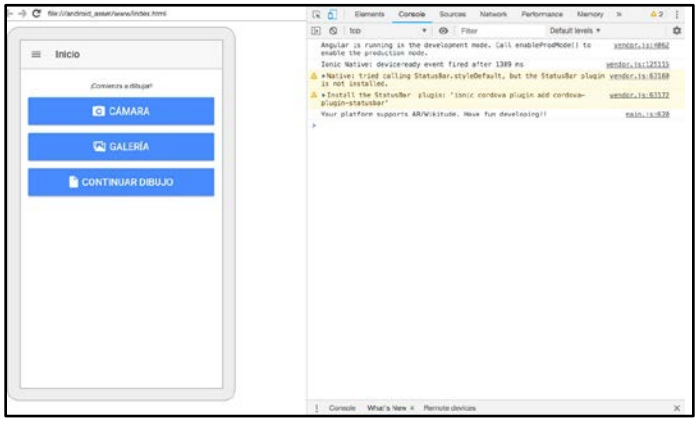

Ilustración 23: Consola desde Inspector de dispositivos

4.2.6 Google Drive

La conocida herramienta de almacenamiento en la nube de Google ha cumplido la función de preservar copias de seguridad periódicas del código fuente del proyecto, así como de la memoria y documentos asociados al proceso de desarrollo.

4.3 Preparación del entorno de desarrollo

4.3.1 Instalación de Node y npm

En primer lugar, para configurar el entorno de desarrollo de una aplicación Ionic, es necesario instalar Node y npm.

Es un proceso sencillo. Simplemente hay que dirigirse al sitio web de Node.js y descargar e instalar el entorno con el gestor que proporciona. En la misma instalación se instala npm.

4.3.2 Instalación de Ionic

El siguiente paso es instalar Ionic, para lo que se emplea npm. El proceso descrito a continuación es explicado en el sitio web de Ionic[12].

Se debe abrir una ventana de la consola del sistema, en el caso del autor del presente trabajo la Terminal de MacOSX. El comando para la instalación del CLI de Ionic es el siguiente:

$ npm install –g ionic@latest

La "*-g*" indica que queremos que la instalación sea global y el "*@latest*" indica que se quiere instalar la última versión disponible. Es posible verificar la versión instalada tecleando

$ ionic --version

En el caso de este proyecto se ha instalado la versión 3 de Ionic.

4.3.3 Comenzar un nuevo proyecto Ionic

Una vez instalado el CLI de Ionic el entorno está preparado para comenzar un nuevo proyecto.

Se ha llamado al proyecto "*ARTest*", pero esto no tiene ninguna influencia sobre el resultado final de la aplicación, es un simple identificador que nombrará el directorio donde se almacena el proyecto, contracción de *Augmented Reality Test*. El comando es el siguiente:

$ ionic start ARTest

Esto genera la estructura básica de ficheros de un proyecto Ionic. Para comenzar a trabajar sobre el proyecto es necesario trasladarse al directorio del mismo.

$ cd ARTest

4.3.4 Instalar Cordova

Creado el proyecto y dado que se va a utilizar, como hemos visto anteriormente, plugins para aplicaciones basadas en Cordova, llega el momento de instalarlo. Se utiliza una vez más npm:

$ npm install –g cordova

4.3.5 Configurar plataformas

El último paso antes de comenzar con la codificación es configurar las plataformas para las que se desarrollará la aplicación. Por el momento solo se añadirá *Android*, pero en cualquier momento se puede añadir también *iOS*.

Basta con ejecutar desde el CLI el siguiente comando:

$ionic cordova platform add android

Antes de continuar se prueba a compilar el proyecto para la plataforma recién añadida para detectar posibles errores.

$ionic cordova build android

4.4 Implementación paso a paso

Una vez preparado el entorno es momento de comenzar la codificación de la aplicación.

4.4.1 Instalar de plugins y módulos para Ionic

Hay que instalar algunos paquetes[13] para dar funcionalidades a la app que no proporciona la instalación básica de Ionic o Cordova. Algunos de estos paquetes forman parte de la colección de plugins de *Ionic Native*, que proporcionan funcionalidades de app nativa y están mantenidos por la comunidad.

A continuación se listan los módulos a instalar, junto a una breve descripción de las funcionalidades que aportan.

Image Picker

Plugin de Cordova para la sección de imágenes desde la galería del dispositivo.

Instalación:

En primer lugar lo añadimos al proyecto Ionic.

$ ionic cordova plugin add cordova-plugin-telerik-imagepicker

Una vez añadido instalamos el plugin con npm.

$ npm install --save @ionic-native/image-picker

Estos dos pasos los realizaremos con todos los plugins.

Camera

Plugin de Cordova para hacer fotos y grabar vídeo con la cámara del dispositivo.

Instalación:

$ ionic cordova plugin add cordova-plugin-camera

$ npm install --save @ionic-native/camera

File

Plugin que permite leer y escribir ficheros en el dispositivo.

Instalación:

$ ionic cordova plugin add cordova-plugin-file

$ npm install --save @ionic-native/file

Android Permissions

Plugin que permite comprobar y gestionar los permisos de la app en Android.

Instalación:

$ ionic cordova plugin add cordova-plugin-android-permissions

$ npm install --save @ionic-native/android-permissions

Wikitude Cordova Plugin

Como se ha descrito anteriormente en este mismo bloque, el plugin de Wikitude aporta funcionales de realidad aumentada esenciales en el proyecto que se va a desarrollar.

En este caso la integración es algo más complicada, ya que no se encuentra mucha documentación oficial acerca del uso de este plugin en las últimas versiones de Ionic. Estas utilizan TypeScript y Angular, no el JavaScript "crudo" que aparece en la mayoría de ejemplos de Wikitude.

El mejor ejemplo de integración que el autor ha encontrado y en el que se ha apoyado es del usuario de GitHub Philipp Breuss-Schneweis (pbreus), que según indica en su biografía es fundador de Wikitude. En el ejemplo[14], el usuario, configura una aplicación básica de Ionic 3 con Wikitude.

En primer lugar se ha instalado el plugin de forma similar a los vistos anteriormente, solo que en este caso se ha añadido desde GitHub:

$ ionic cordova plugin add https://github.com/Wikitude/wikitude-cordova-plugin.git

Una vez añadido al proyecto, hay que dirigirse al sitio web de Wikitudey obtener una licencia de desarrollador que permite utilizar las funcionalidades de Wikitude de forma gratuita con la condición de que aparezca en la vista de la cámara una marca de agua.

Ya teniendo la licencia necesaria hay que añadir un script proporcionado en GitHub por el usuario arriba citado para adaptar el plugin de Wikitude a Ionic 3. El fichero se llama *WikitudePlugin.d.ts* y declara una interfaz para utilizar el plugin e inicializa una variable de la misma que se usará en el *app.component.ts*. Se ha llamado a la variable "*WikitudePlugin*", igual que a la interfaz. *Tanto el app.component.ts*, como la interfaz de Wikitude se encuentran en */src/app/* del proyecto.

```
/**
 * Wrapper for the Wikitude SDK Phonegap Plugin - to use with IONIC2
 * (c) 2016 Schneeweis.Technology
 */
interface WikitudePlugin {

  isDeviceSupported(
    successCallback: (success: string) => void,
    errorCallback: (message: string) => void,
    requiredFeatures: [string]): void;

  loadARchitectWorld(
    successCallback: (success: string) => void,
    errorCallback: (message: string) => void,
    architectWorldPath: string,
    requiredFeatures: [string],
    startupConfiguration: JSON): void;

  close(): void;
```

Ilustración 25: Cabecera y algunos métodos de la interfaz de Wikitude para Ionic 2/3

```
declare var WikitudePlugin: WikitudePlugin;
```

Ilustración 26: Declaración del objeto WikitudePlugin

Declaración de la variable que se utilizará para llamar a las funcionalidades de Wikitude

Teniendo la interfaz definida hay que dirigirse a *app.component.ts* y codificar algunas funciones, también indicadas en el ejemplo visto.

En el fichero hay que dirigirse al método constructor. En el constructor se han añadido una serie de líneas que se ejecutarán cuando se detecte que la plataforma está lista para funcionar.

En primer lugar se ha añadido la clave correspondiente a la licencia de desarrollador que se ha obtenido antes.

```
WikitudePlugin._sdkKey =
"KX0ZQ0dHAiT020p7lMMq9jikg3BwCtNFU+jUSnwMB28fyF+urbVweDrc
DlWSIsnBnoBuTxHEhXJSPBMh2zn/ITeqlaHKQ8Zbq8rtbZWrBNYD8i2Qj
```

Ilustración 27: Fragmento de la clave para desarrollador de Wikitude

A continuación se añade una comprobación para verificar si el dispositivo soporta Wikitude.

```
WikitudePlugin.isDeviceSupported(
  function(success) {
    console.log("Tu plataforma soporta Wikitude");
  },
  function(fail) {
    console.log("Fallo al ejecutar Wikitude: " + fail);
  },
  [WikitudePlugin.FeatureGeo]
);
```

Ilustración 28: Verificación de soporte de Wikitude

La vista de realidad aumentada de Wikitude crea su propio contexto de ejecución a parte del resto de la app, por lo que se dificulta la comunicación entre Wikitude y el resto de componentes. Por ello es necesario codificar un sistema de comunicación de Wikitude a Ionic y de Ionic a Wikitude.

Para recibir datos en Ionic desde el contexto de Wikitude se utilizará la función descrita en la interfaz del plugin "*setJSONObjectReceivedCallback*", que permite recibir objetos JSON como callback. Una vez recibido el objeto se buscará un atributo llamado "*action*" en el mismo y utilizando una estructura condicional de tipo *switch* se determinará qué acción realizar en consecuencia.

```
WikitudePlugin.setJSONObjectReceivedCallback(obj => {
```

Ilustración 29: WikitudePlugin a la espera de recibir un JSON del contexto de Wikitude

```
if (obj["action"]) {
console.log("Action: "+obj["action"]);
  switch (obj["action"]) {
    case "closeWikitudePlugin":
      // close wikitude plugin
      WikitudePlugin.close();
      break;
    case "captureScreen":
```

Ilustración 30: Fragmento del switch de comprobación del JSON recibido

Más adelante se comentarán los diferentes casos implementados en el *switch*, incluyendo la comunicación a la inversa, de Ionic a Wikitude.

Ya preparado Wikitude en *app.components* se puede considerar que el plugin se ha instalado y se ha realizado una configuración básica.

4.4.2 Importar los módulos instalados

Para poder utilizar los módulos instalados es necesario importarlos en el fichero *app.module.ts* y declararlos como "*providers*". Para ello se codifican las líneas de importación en la parte superior del fichero y se añaden los nombres de los módulos al array de *providers*.

```
import { ImagePicker } from '@ionic-native/image-picker';
import { File } from '@ionic-native/file';
import { Camera } from '@ionic-native/camera';
import { AndroidPermissions } from '@ionic-native/android-permissions';
```

Ilustración 31: Importación de los plugins instalados

```
providers: [
  StatusBar,
  SplashScreen,
  ImagePicker,
  File,
  Camera,
  AndroidPermissions,
  Dialogs,
  {provide: ErrorHandler, useClass: IonicErrorHandler},
  DrawStorageProvider
]
```

Ilustración 32: Inserción en el array "providers" de los plugins instalados

También se deben realizar las importaciones en todos los componentes en los que se vaya a utilizar cada uno de los plugins. Esto se ejemplificará más adelante.

4.4.3 Generar las páginas de la aplicación

Ya instalados los módulos necesarios es momento de generar las páginas que conforman la aplicación a partir de los casos de uso planteados en el apartado de Especificación.

La páginas se pueden generar fácilmente utilizando una vez más el CLI de Ionic. El comando es el siguiente:

$ ionic cordova generate page [<nombre de la página>]

Se ejecuta este comando para cada una de las páginas. Se ha preferido indicar el nombre de la página en inglés. Las páginas son las siguientes:

- ***ARView***: página desde donde se lanza la vista de Realidad Aumentada de Wikitude.
- ***Continue***: página para continuar dibujo.
- ***Hand***: página de preferencias de mano de dibujo.
- ***Help***: página de ayuda.

- ***Home***: página de inicio.

Teniendo generadas las páginas todavía no se pueden utilizar, deben añadirse a *app.module.ts*. Para esto hay que importarlas de la misma forma que se importaron los módulos instalados y acontinuación añadirlas al array "*declarations*", después de *app.component*, que también se ha añadido con el nombre *MyApp*.

Ilustración 33: Inserción de las páginas creadas en el array "declarations"

4.4.4 Incorporar el menú lateral

Para navegar entre las diferentes páginas se va a utilizar un menú lateral. Antes de comenzar a dar contenido a cada página se creará el menú para facilitar el acceso durante el desarrollo.

El primer paso es dirigirse de nuevo a app.component, el componente principal de la aplicación y añadir un atributo "*pages*" que contendrá un array con las páginas y su nombre a mostrar en la interfaz de la aplicación.

Fuera del constructor se declara el atributo como un array de objetos formados por un título y un componente.

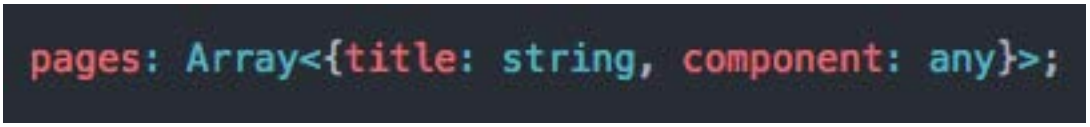

Ilustración 34: Declaración del array atributo que contendrá las páginas y su título

Antes de inicializar el atributo se deben importar las páginas del mismo modo que en los casos anteriores.

Ya importados podemos inicializar el atributo dentro del constructor.

```
this.pages = [
  { title: 'Inicio', component: HomePage },
  { title: 'Continuar dibujo', component: ContinuePage },
  { title: 'Preferencias', component: HandPage},
  { title: 'Ayuda', component: HelpPage },
];
```

Ilustración 35: Inicialización del atributo "pages"

En Angular los atributos declarados en los controladores de los componentes están ligados a las vistas de los mismos, por lo que para construir la vista del menú hay que dirigirse a *app.html* (la vista de *app.component*) e insertar una etiqueta de componente *ion-menu* que se rellenará con una lista de botones, uno por página, recorriendo el array *pages* definido en el controlador con la estructura **ngFor*.

```
<ion-menu [content]="content">
  <ion-header>
    <ion-toolbar>
      <ion-title>Menu</ion-title>
    </ion-toolbar>
  </ion-header>

  <ion-content>
    <ion-list>
      <button menuClose ion-item *ngFor="let p of pages" (click)="openPage(p)">
        {{p.title}}
      </button>
    </ion-list>
  </ion-content>
</ion-menu>
```

Ilustración 36: Estructura HTML del menú con iterador de páginas

Al hacer click se ejecuta el metodo *openPage()* pasando como parámetro la página que se quiere abrir. La función *openPage()* también se ha definido en el controlador.

```
openPage(page) {
  this.nav.setRoot(page.component);
}
```

Ilustración 37: Función para abrir una página estableciendola como raíz (sin boton "atrás")

Como se puede apreciar, las funciones también son directamente accesibles desde la vista.

Para ver el resultado del menú se puede ejecutar en el CLI el comando

$ ionic serve

Que lanza un servidor local con la app accesible desde el navegador. Además se refresca en tiempo real con cada guardado, por lo que será útil en el resto del desarrollo de las

vistas y controladores que no requieran explícitamente de un dispositivo móvil, es decir, las que no utilicen funciones nativas.

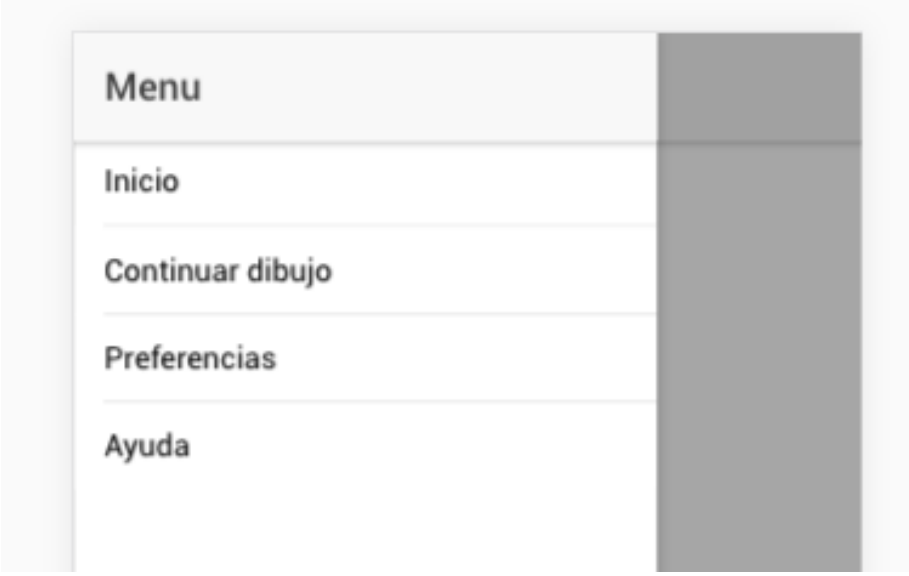

Ilustración 38: Vista del menú en el navegador durante la ejecución de "ionic serve"

4.4.5 Generar el servicio de persistencia

Para la persistencia de datos se va a utilizar el almacenamiento local *localStorage*. *localStorage* permite acceder a la interfaz *Storage* local. La interfaz *Storage* pertenece a la *API* de almacenamiento web, se puede utilizar en cualquier sitio web y dado que ClarApp está construida con tecnologías web es posible usarla sin problemas en este proyecto.

Los datos se almacenan en *localStorage* con formato *JSON*. Se guardan con el método *setItem(identificador, valor)* y se recuperan con *getItem(identificador)*.

Tanto el histórico de dibujos como la mano de dibujo se almacenan en un objeto que se ha llamado *collection* en *localStorage*.

Dado que el acceso y la modificación de los datos va a ser frecuente en las diferentes páginas es conveniente definir un servicio que abstraiga y simplifique estas acciones. Por ello se define un servicio (o *provider*, según la nomenclatura de Ionic 3) *DrawStorage*.

Para generar el servicio se utiliza una vez más el CLI con el siguiente comando:

$ionic generate provider DrawStorage

Dando como resultado un directorio *providers/draw-storage/* con el script correspondiente.

```
@Injectable()
export class DrawStorageProvider {

  constructor(private file: File) {

  }

  getCollection(){
    var collection: any;
    collection = localStorage.getItem("collection");
    if (!collection) {
      collection = {};
      collection.draws = [];
      collection.hand = -1;
    } else {
      collection = JSON.parse(collection);
    }
    return collection;
  }
```

Ilustración 39: Cabecera y primer método del servicio DrawStorgae

En el script del servicio se codifican los métodos que siguen:

- ***getCollection()****:* recupera la colección de dibujos realizados y la mano de dibujo o los inicializa si es la primera llamada.
- ***saveCollection(collection)****:* guarda la colección pasado como parámetro sobrescribiendo la existente si la hay.
- ***addDraw(draw)****:* añade un dibujo al final de la colección y guarda la colección.
- ***setCurrentDraw(index)****:*guarda como dibujo actual el situado en la posición *index* de la colección.
- ***getCurrentDraw()****:* recupera el dibujo actual.
- ***setHand(hand):*** establece la mano con que dibujará el usuario, siendo 0 para izquierda y 1 para derecha.
- ***getHand():*** recupera la mano de dibujo.

En apartados siguientes se apreciará el uso que se da a estos métodos en cada página.

Ya generado y codificado, para que sea posible utilizarlo, es necesario añadir el servicio a *app.module* como *provider*, de la misma forma que se hizo con los módulos instalados en el punto 4.4.2.

4.4.6 Programar las funcionalidades de Realidad Aumentada

Las funcionalidades de Realidad Aumentada son las que aportan el grueso de la utilidad de esta aplicación, ya que permitirán a los usuarios visualizar la imagen que quieren dibujar directamente sobre la superficie de dibujo. En este paso se describe cómo se desarrolla el script que gestionará la vista de Realidad Aumentada con Wikitude.

En primer lugar se construye un directorio en */assets/* al que se llama "*ar*". En él se situará el script, así como las imágenes, HTML y CSS que puede requerir.

Se añade un *index.html*, que será la vista en que se cargue el script en el contexto de Wikitude y un directorio *"js"* para el script, un *"css"*para los estilos y otro *"assets"* para las imágenes y los modelos que deberá reconocer Wikitude.

El siguiente paso será generar los modelos a reconocer.

El usuario utilizará marcadores impresos en cartulina o papel, por lo que se ha considerado que un cuadrado de 3x3cm será adecuado. Se ha hecho un diseño sencillo con el software de edición de imagen Gimp y se ha exportado en 4 colores distintos en formato PNG. La variedad de color permitirá al usuario utilizar el que más convenga según la superficie sobre la que dibujará, ya que es importante que la cámara lo distinga sin problemas.

Los modelos se deben generar en Wikitude Studio, una aplicación web de Wikitude a la que se accede con la cuenta de desarrollador. Simplemente hay que crear un nuevo proyecto y subir las imágenes escogidas. Ya generados se pueden descargar como fichero.*wtc*, el formato de modelo de Wikitude.

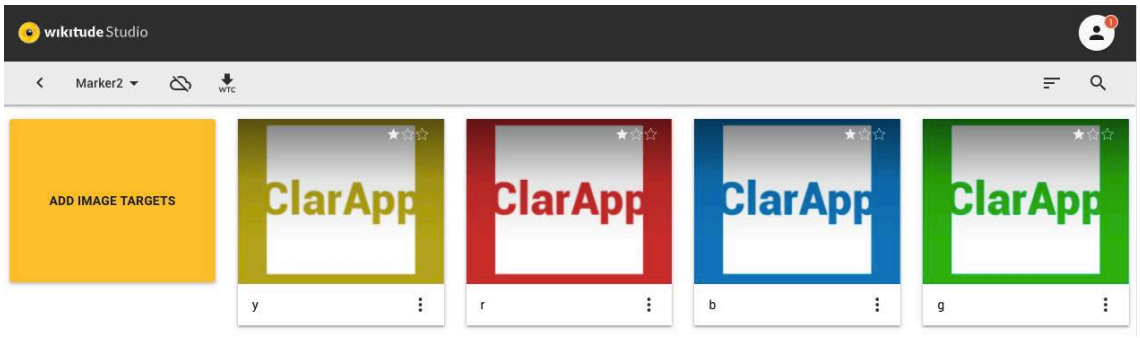

Ilustración 40: Proyecto con modelos generados en Wikitude Studio

Se añade al nuevo directorio *assets* el fichero *wtc*, al que se llamado *corners.wtc*.

Para programar el script se han estudiado diferentes ejemplos proporcionados por Wikitude[15].

Ya que se ha tomado como referencia el ejemplo 3-1 *MultipleTargets* se ha llamado al script *multipletargets.js*.

En las primeras líneas se han declarado variables globales, que almacenarán valores tales como la posición previa de la imagen en pantalla, el objeto que representa la imagen (*overlay*) o el condicional que indica si ya se ha reconocido un modelo.

```
//alamcenar valores previos a gestos
var previousDragValueX = 0.0;
var previousDragValueY = 0.0;
var previousRotationValue = 0.0;
var previousScaleValue = 1;
var previousOpacity = 0.5;
//ya hay imagen reconocida
var activePicture = false;
var picturePath = "";

var theTarget = null;
var theTracker = null;
var overlay = null;
var currentDraw = {};
var draws = [];
var draw = {};
```

Ilustración 41: Declaración de variables globales en el script de Realidad Aumentada

A continuación se declara una variable objeto *World* que representa todo el contexto de Wikitude y contiene las funciones que se ejecutarán en él. Estas funciones son:

init()

Ejecuta los métodos que se desea lanzar al comenzar, en este caso *createOverlays()*.

createOverlays()

Crea un *ImageTracker* a partir de los modelos generados y se lo pasa a un ImageTrackeable, que se encargará de asignar como objetivo la imagen reconocida y en caso de que sea la primera vez que se reconoce lanzar la función "*sendJSONObject*" pasando como parámetro un objeto JSON con un atributo"*action*" que contiene la cadena "*getCurrent*".

Hay que recordar que en el punto 4.4.1, al instalar Wikitude, se construyó una estructura *switch* que recibía el atributo "*action*" de un objeto recibido. Allí es donde se envía este JSON.

En caso de recibir la cadena *getCurrent* se ejecutará un fragmento de código que recupera el dibujo actual almacenado en *localStorage* y se lo pasa a la función *chargeValues()* del objeto *World*. Se explica a continuación.

chargeValues(current)

Recibe un JSON con los datos del dibujo actual y los asigna a las variables globales vistas previamente, de manera que en la primera carga de la imagen se recuperan los cambios de posición, rotación y escala que el usuario haya realizado en sesiones previas. A continuación ejecuta la función *loadPicture()*, que también se ha definido en el objeto *World*, pasando como parámetro la ruta de la imagen a cargar.

loadPicture(path)

Recibe una cadena con la ruta de la imagen a cargar en la vista de la cámara. A continuación genera un *ImageResource* a partir de la ruta. Luego se definen dos métodos:

- ***setStorage():***envía el dibujo actual modificado para que se actualice en localStorage, utilizando un JSON con el atributo "*action*" conteniendo "*updateCollection*".
- ***updateCurrent():*** actualiza las variables globales con los ajustes actuales de la imagen, los modifica en el objeto que contiene el dibujo actual e invoca al método anterior, *setStorage()* para guardar los cambios.

Una vez declarados estos métodos se inicializa el objeto *overlay*, que representa la imagen mostrada con realidad aumentada en la vista de cámara, a partir del *ImageResource* antes generado y se le asignan los valores que se almacenaron en las variables globales. Además se define el comportamiento a seguir cuando se detectan los gestos de arrastrar, rotar y escalar la imagen, que además de cambiar cómo se visualiza invocan al metodo *updateCurrent(),* para que la colección de dibujos siempre esté actualizada con los últimos ajustes.

La última función que se defineees*changeOpacity(),* que modificará la opacidad de la imagen y se programa un *listener* que llamará a esta función cuando detecte que se utiliza un control de tipo *slider* que se ha situado en la parte inferior de la vista en el fichero HTML. Finalmente se llama a *removeLoadingBar(),* también del objeto *World*.

removeLoadingBar()

Cierra una barra que se ha situado en la parte superior mostrando instrucciones de cómo comenzar a reconocer el modelo. La barra permanece abierta hasta que se reconoce por primera vez.

Finalmente cabe destacar que dado que el contexto de ejecución de Wikitude es distinto al de Ionic se ha diseñado una función para poder visualizar *logs* lanzados desde este script en la consola del navegador. La función se ejecuta enviando un objeto con el atributo "*action*" conteniendo la cadena "*notify*" y un atributo "*log*" conteniendo el texto a mostrar por consola. Este objeto se procesará en el *switch* de *app.component*, como el resto de los objetos enviados desde el script que se han indicado en este apartado.

Para que sea posible cargar el script desde Ionic hay que añadirlo al index.html creado antes e incorporar algunas líneas al controlador de la página ARView.

Las líneas se añaden en el método ionViewDidEnter(), que es lanzado cuando la página del controlador es la página activa. En este método se ejecutará la función loadArchitectWorld() de la variable WikitudePlugin declarada en el punto 4.4.1, pasando como parámetro la ruta del *index* de la vista que hemos creado, junto con el array *["ir"],* indicativo de que es queremos un "World" de tipo *Image Recognition* (Reconocimiento de Imagen).

4.4.7 La página de inicio

Teniendo ya la parte fundamental de la aplicación se puede pasar a dar funcionalidad a las páginas que harán uso de ella. La primera será la página de Inicio.

Ilustración 42: Vista de Inicio desde el navegador

Para comenzar se creará la vista, abriendo el fichero *home.html* y añadiendo tres botones a pantalla completa que darán acceso a las tres acciones principales deseadas en esta vista:

- Comenzar un dibujo a partir de una foto de la cámara
- Comenzar un dibujo a partir de una imagen de la galería
- Continuar un dibujo

```
<ion-content padding text-center>
  <p>¡Comienza a dibujar!</p>
  <button (click)="takePicture()" ion-button block large margin-bottom icon-start>
    <ion-icon name="camera"></ion-icon>
    Cámara
  </button>
  <button (click)="getPicture()" ion-button block large margin-bottom icon-start>
    <ion-icon name="images"></ion-icon>
    Galería
  </button>
  <button (click)="continue()" ion-button block large icon-start>
    <ion-icon name="document"></ion-icon>
    Continuar dibujo
  </button>
</ion-content>
```

Ilustración 43: Estructura HTML de la vista de Inicio

Como se aprecia en la imagen cada botón llama a una función que se definirá en el controlador a continuación.

En el controlador *home.ts* se definen las tres funciones que utiliza la vista:

takePicture()

Abre la cámara del dispositivo, toma una fotografía utilizando el plugin *Camera*y se almacena usando el plugin File, se guardan los datos en la colección de dibujos y se guarda como dibujo actual con la función *logDraw()* y se abre la vista de Wikitude accediendo a la página ARView.

getPicture()

Abre la galería, selecciona una imagen utilizando el plugin *ImagePicker* y se almacena usando el plugin File, se guardan los datos en la colección de dibujos y se guarda como dibujo actual con la función *logDraw()*y se abre la vista de Wikitude accediendo a la página ARView.

continue()

Se accede a la página Continue con las opciones para continuar un dibujo. Se explicará más adelante.

La función *logDraw()* que utilizan las dos primeras hace uso del servicio definido en el punto 4.4.5, *DrawStorage*, para lo que se ha importado en el controlador y pasado como parámetro al método constructor, al igual que *Camera*, *ImagePicker*, *File* y *AndroidPermissions*.

```
constructor(public navCtrl: NavController,
  private imagePicker: ImagePicker,
  private file: File,
  private camera: Camera,
  private platform: Platform,
  public permissions: AndroidPermissions,
  public dialogs: Dialogs,
  private drawStorage: DrawStorageProvider) {}
```

Ilustración 44: Método constructor del controlador de Inicio

```
//log draw in localstorage
logDraw(path) {
  var draw: any = {};
  var collection: any;
  draw.fecha = (new Date()).getTime();
  draw.ruta = path;
  draw.opacidad = 0.5;
  draw.rotacion = 0;
  draw.tamano = 1;
  draw.posicionX = 0;
  draw.posicionY = 0;
  collection = this.drawStorage.getCollection();
  console.log(collection.draws)
  this.drawStorage.addDraw(draw);
  this.drawStorage.setCurrentDraw(draw, collection.draws.length-1);
}
```

Ilustración 45: Método logDraw() del controlador de Inicio

AndroidPermissions se ha utilizado para asegurar que el usuario ha aceptado los permisos necesarios antes de lanzar Wikitude, evitando así errores de ejecución.

Al cargar la página se comprueba si el usuario ha especificado la mano de dibujo utilizando el servicio generado en el paso 4.4.5, si aún no se ha establecido se redirige al usuario a la página de preferencias para que lo haga.

```
var hand = drawStorage.getHand();
if(hand==-1) navCtrl.push(HandPage);
```

Ilustración 46: Comprobación de la configuración de mano de dibujo

4.4.8 La página Continuar Dibujo

La página Continuar Dibujo permite a los usuarios ver su histórico de dibujos y continuarlos a partir de los ajustes que realizaron en su momento.

Primero hay que recuperar los dibujos que, como se ha explicado antes, se encuentran almacenados en localStorage, por lo que una vez más se importará el servicio definido en el punto 4.4.5, *DrawStorage*, en el controlador Continue.ts.

Los dibujos se guardarán en un atributo *collection* del controlador.

A continuación se describen los métodos declarados en este controlador.

updateCollection()

Realiza la asignación del atributo *collection*, además es llamado desde el método *continueDraw()* para asegurar que se continua el dibujo con los últimos ajustes realizados.

```
updateCollection(){
  this.collection = this.drawStorage.getCollection();
}
```

Ilustración 47: Método updateCollection() del controlador de Continuar Dibujo

continueDraw(index)

Invoca a *updateCollection()*, selecciona el dibujo correspondiente a *index* como dibujo actual en *localStorage* y a continuación carga la página ARView, es decir, la vista de Realidad Aumentada.

```
continueDraw(i){
  this.updateCollection();

  var aux = this.collection.draws[i].ruta.split(".");
  this.drawStorage.setCurrentDraw(this.collection.draws[i], i);
  this.navCtrl.push(ARView);
}
```

Ilustración 48: Método continueDraw() del controlador de Continuar Dibujo

removeDraw(index)

Elimina el dibujo correspondiente a *index* del histórico tras pedir confirmación al usuario.

```
removeDraw(i){
  if(confirm("¿Borrar dibujo?")){
    this.collection.draws.splice(i,1);
    this.drawStorage.saveCollection(this.collection);
  }
}
```

Ilustración 49: Método removeDraw() del controlador de Continuar Dibujo

Teniendo preparado el controlador se puede pasar a maquetar la vista, que consistirá en una lista de tarjetas con una miniatura de la imagen de referencia de cada dibujo, la fecha en que se realizó y un botón de eliminar.

Cada tarjeta escuchará el evento *click*, que lanzará la función *continueDraw()*.

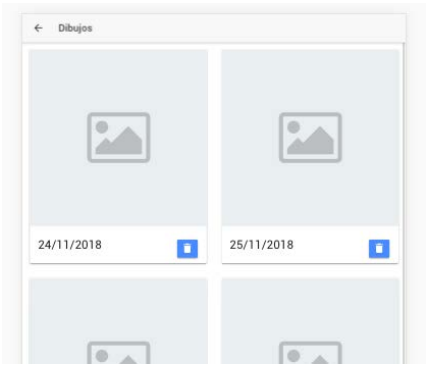

Ilustración 50: Vista de Continuar Dibujo desde el navegador con imágenes de ejemplo

```
<ion-content class="card-background-page">
  <ion-row *ngIf="collection.draws.length>0">
    <ion-col *ngFor="let pic of collection.draws; index as i" col-12 col-sm-6 col-md-6 col-lg-4 col-xl-4>
      <ion-card>
        <img (click)="continueDraw(i)" src="{{collectionPath+pic.ruta}}" />
        <ion-card-content>
          <ion-card-title>
            <span>{{pic.fecha | date: 'dd/MM/yyyy'}}</span>
            <button ion-button icon-only float-right (click)="removeDraw(i)">
              <ion-icon name="trash"></ion-icon>
            </button>
          </ion-card-title>
          <p>{{pic.path}}</p>
        </ion-card-content>
      </ion-card>
    </ion-col>
  </ion-row>
  <ion-row *ngIf="collection.draws.length==0">
    <ion-col text-center>
      <p>¡Todavía no has empezado ningún dibujo!</p>
    </ion-col>
  </ion-row>
</ion-content>
```

Ilustración 51: Estructura HTML de la vista Continuar Dibujo

4.4.9 La página Ayuda

La implementación de la página de ayuda es relativamente sencilla, ya que únicamente está destinada a recopilar algunos puntos que pueden ser de ayuda al usuario.

Para comenzar hay que dirigirse al fichero correspondiente a la vista de ayuda, es decir, help.html. Se añade una lista de botones, cada uno correspondiente a un punto de la página de ayuda. A continuación se construye un contenedor por cada punto con un encabezado y se completa con el texto.

Finalmente, para que pulsando en cada botón se muestre el contenedor correspondiente y se oculten los demás se añade un atributo booleano por cada uno. Utilizando la estructura **ngIf* de Angular se asegura que solo se muestren cuando el atributo sea *true*. Al hacer clic en un botón se ocultarán todos los contenedores y se pondrá a *true* el relacionado con el botón.

```
<ion-list>
  <button (click)="closeAll();what=true" ion-item>¿Qué es ClarApp?</button>
  <button (click)="closeAll();start=true" ion-item>¿Cómo empiezo a dibujar?</button>
  <button (click)="closeAll();settings=true" ion-item>¿Puedo hacer ajustes sobre la imagen?</button>
  <button (click)="closeAll();disappear=true" ion-item>¿Qué hago si la imagen desaparece mientras dibujo?</button>
  <button (click)="closeAll();continueD=true" ion-item>¿Puedo recuperar un dibujo ya comenzado donde lo dejé?</button>
</ion-list>

<div *ngIf="what" class="help-block">
  <h2 ion-text color="primary">¿Qué es ClarApp?</h2>
  <p>ClarApp una aplicación móvil destinada a ayudar a los artistas en el momento de reproducir una imagen mediante el d
  <p>El usuario es capaz de añadir imágenes tanto de la galería como de la cámara del dispositivo y ajustar sus caracter
</div>

<div *ngIf="start" class="help-block">
  <h2 ion-text color="primary">¿Cómo empiezo a dibujar?</h2>
  <ol>
  <li>Para comenzar a dibujar tienes que <b>escoger la imágen que usarás como referencia</b>. Puedes tomar la imágen de
```

Ilustración 52: Fragmento de la estructura HTML de la vista de Ayuda

Los botones se ocultan con la función *closeAll()* declarada en el controlador *help.ts*.

```
closeAll(){
  this.what = false;
  this.start = false;
  this.settings = false;
  this.disappear = false;
  this.continueD = false;
}
```

Ilustración 53: Método closeAll() del controlador de Ayuda para cerrar todos los apartados

4.4.10 La página de Preferencias

En la página de preferencias el usuario establecerá con qué mano prefiere realizar los dibujos, por lo que se ha optado por un simple formulario con dos *radio buttons*, uno por mano. Al modificar el valor, este es almacenado utilizando una vez más el servicio *DrawStorage* del punto 4.4.5. Según la mano escogida se mostrará un consejo acerca de dónde ubicar el marcador de ClarApp a la hora de dibujar.

```
<ion-list radio-group [(ngModel)]="hand">
  <ion-item>
    <ion-label>Derecha</ion-label>
    <ion-radio value="1" (click)="updateHand()"></ion-radio>
  </ion-item>
  <ion-item>
    <ion-label>Izquierda</ion-label>
    <ion-radio value="0" (click)="updateHand()"></ion-radio>
  </ion-item>
</ion-list>

<p *ngIf="hand==0">Si dibujas con la mano izquierda, el mejor sitio para situar el marcador de ClarApp es la parte superior derecha de la superficie de dubujo.</p>
<p *ngIf="hand==1">Si dibujas con la mano derecha, el mejor sitio para situar el marcador de ClarApp es la parte superior izquierda de la superficie de dubujo.</p>
```

Ilustración 54: Estructura HTML de la vista Preferencias

Para detectar el cambio de valor se escucha el evento *click* en los dos *radio buttons*. El atributo correspondiente a la mano es *hand*, declarado en el controlador y enlazado al formulario con la estructura *ngModel* de Angular.

```
hand: number;

constructor(public drawStorage: DrawStorageProvider) {
  var collection = drawStorage.getCollection();
  this.hand = collection.hand;
}

updateHand(){
  this.drawStorage.setHand(this.hand);
  console.log(this.hand);
}
```

Ilustración 55: Fragmento del controlador de Preferencias

4.5 Compilación

En este subapartado se compila la aplicación para la plataforma Android.

La compilación se realiza con el comando *build* seguido de la plataforma objetivo, por tanto, la orden queda de la siguiente forma:

$ionic cordova build android

Se comprueba que todo va bien con la salida resultado:

```
BUILD SUCCESSFUL in 22s
46 actionable tasks: 3 executed, 43 up-to-date
Built the following apk(s):
        /Users/Eduardo/ionic_proj/ARTest/platforms/android/app/build/outpu
ts/apk/debug/app-debug.apk
```

Ilustración 56: Conclusión del comando "ionic cordova build android"

5. Resultados

En este apartado se presentará una serie de capturas de pantalla realizadas desde un dispositivo real, un smartphone *OnePlus 3T*, utilizando ClarApp y se mostrarán los resultados de test de funcionalidad realizados por el autor y test de usabilidad realizados por usuarios reales.

5.1 Capturas de pantalla

Se exponen las capturas de pantalla realizadas desde smartphone ordenadas por la página en la que se tomaron.

5.1.1 Preferencias

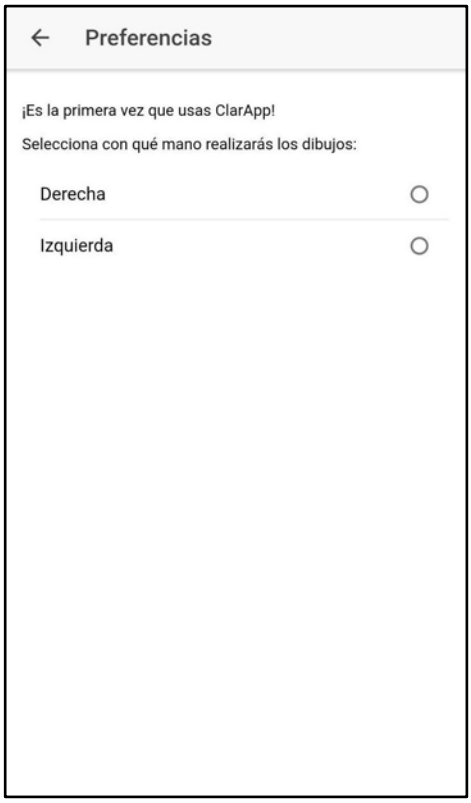

Ilustración 57: Vista de Preferencias en el primer arranque

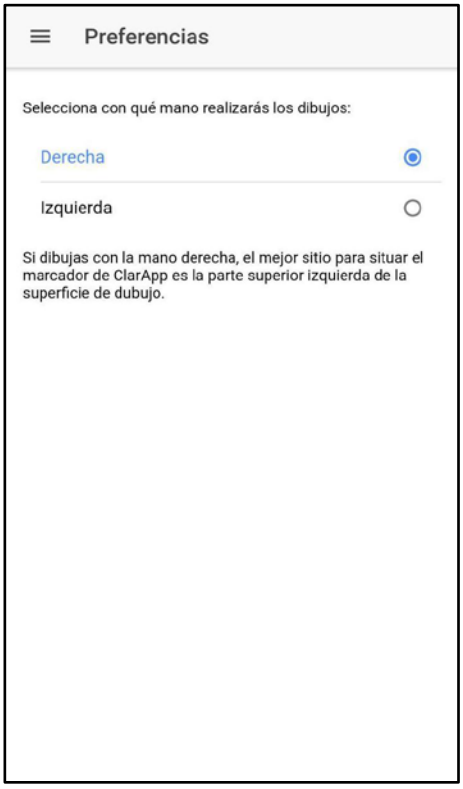

Ilustración 58: Vista de Preferencias una vez seleccionada la mano de dibujo

5.1.2 Inicio

Ilustración 59: Vista de Inicio

5.1.3 Ayuda

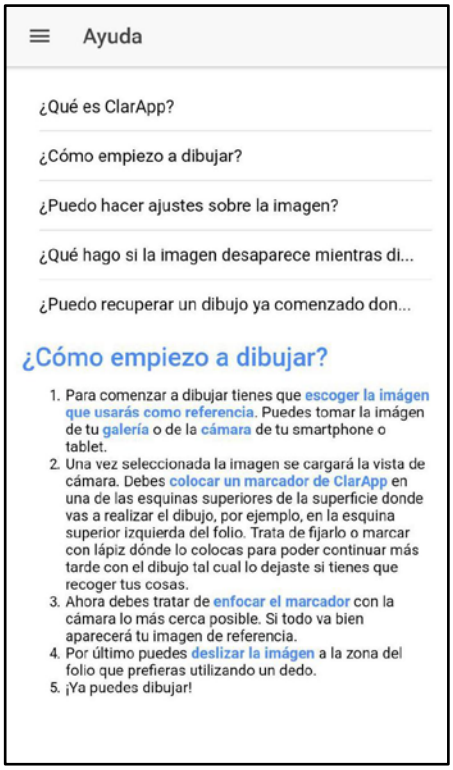

Ilustración 60: Vista de Ayuda

5.1.4 Continuar dibujo

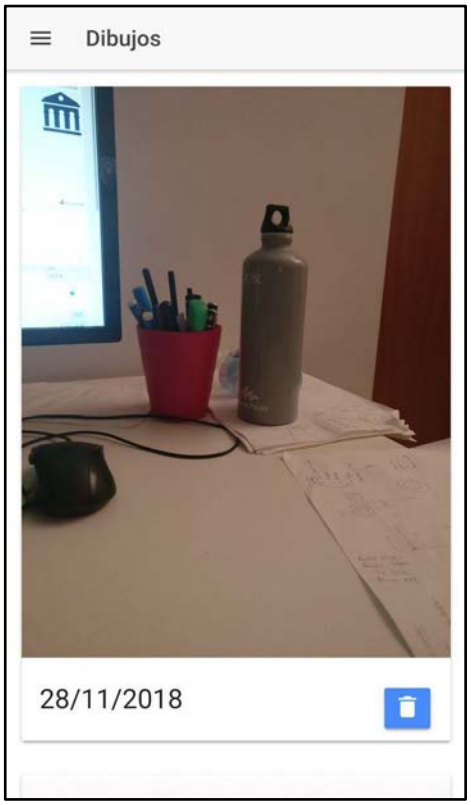

Ilustración 61: Vista de Continuar dibujo

5.1.6 Realidad Aumentada (Wikitude)

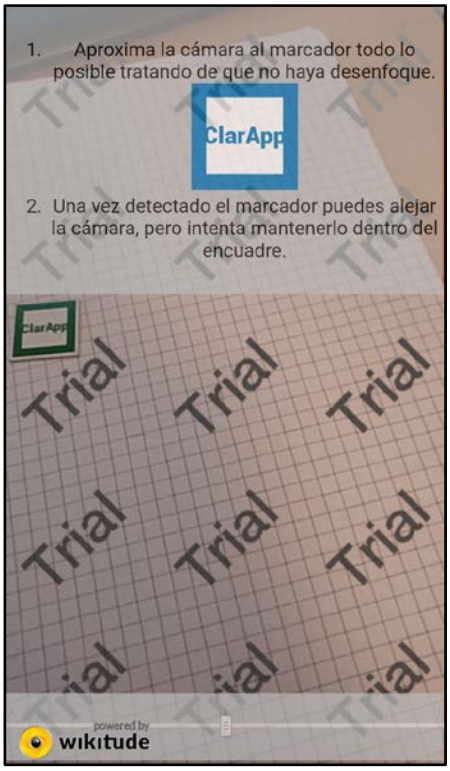

Ilustración 62: Vista de Realidad Aumentada con instrucciones previas

Ilustración 63: Vista de Realidad Aumentada una vez detectado el marcador

Ilustración 64: Vista de Realidad Aumentada con la imagen de referencia centrada

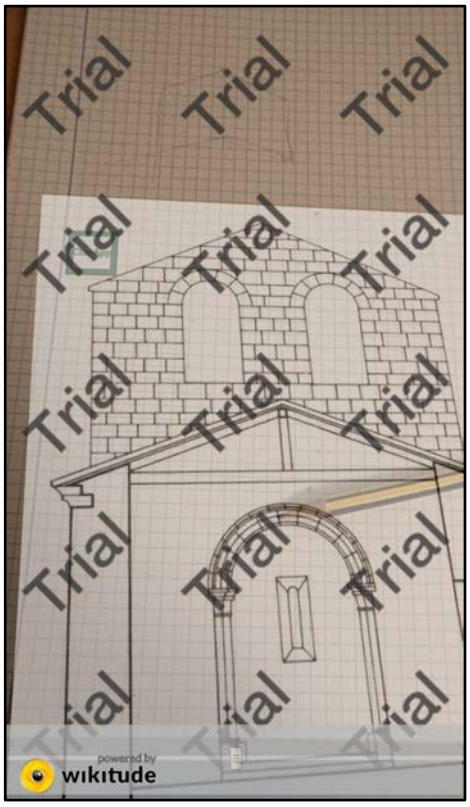

Ilustración 65: Vista de Realidad Aumentada con la imagen de referencia apliada y lista para el dibujo

Como se puede observar en las capturas de la vista de Realidad Aumentada aparece la marca de agua de Wikitude, que en parte dificulta la experiencia, además el logotipo aparece sobre el control de opacidad. Esto en una versión de producción con la licencia en orden no ocurriría.

5.2 Test de funcionalidad

Se han realizado test funcionales de los diferentes casos de uso reproduciendo los mismos en un dispositivo real para detectar posibles errores. A continuación se presentan los resultados.

Tabla 11: Resultados de los test de funcionalidad realizados por el autor

Test	Caso de uso	Resultado
1	CU-01	Se abre correctamente la página de preferencias de mano de dibujo y al elegirla se guarda correctamente.
2	CU-02	Se abre sin problemas la vista de realidad aumentada y se reconoce correctamente el marcador. Los ajustes iniciales se guardan sin errores.
3	CU-03	A lo largo de la sesión de dibujo se puden realizar ajustes y se guardan sin problemas.
4	CU-04	Al acceder a preferencias se puede cambiar la mano de dibujo y se guarda sin errores.
5	CU-05	Se accede a la vista de ayuda y se despliega cada elemento de ayuda sin problemas.
6	CU-06	Se abre la vista de continuar dibujo y se muestra el histórico de dibujo correctamente. Al seleccionar un dibujo se abre la vista de realidad aumentada sin errores y se cargan los ajustes realizados previamente.

Todos los casos de uso se reproducen sin errores.

5.3 Test de usabilidad

Se plantean tres test de usabilidad que cubren todos los casos de uso en todas sus variantes. Son realizados por cuatro potenciales usuarios reales de entre 20 y 24 años y se toma nota del tiempo y las notas dadas por los usuarios y el autor como observador.

5.3.1 Test 1

El primer test consiste en configurar la mano de dibujo por primera vez, comenzar un nuevo dibujo desde la cámara y modificar la opacidad.

Usuario 1

Tabla 12: Resultados del Test 1 para el Usuario 1

Tarea	Caso de uso	Observaciones
Selección inicial de mano de dibujo	CU-01	El usuario no sabía que el guardado era automático
Comienzo de dibujo desde cámara	CU-02	Sin incidencias
Cambio de opacidad	CU-03	Sin incidencias

Tiempo total: 48 segundos

Usuario 2

Tabla 13:Resultados del Test 1 para el Usuario 2

Tarea	Caso de uso	Observaciones
Selección inicial de mano de dibujo	CU-01	El usuario no sabía que el guardado era automático, sugiere que tenga lugar al pulsar un botón
Comienzo de dibujo desde cámara	CU-02	Sin incidencias
Cambio de opacidad	CU-03	El usuario no encontraba el control de opacidad

Tiempo total: 1 minuto 9 segundos

Usuario 3

Tabla 14: Resultados del Test 1 para el Usuario 3

Tarea	Caso de uso	Observaciones
Selección inicial de mano de dibujo	CU-01	El usuario no sabía que el guardado era automático, sugiere que una vez guardado el usuario sea devuelto a la vista de Inicio automáticamente
Comienzo de dibujo desde cámara	CU-02	Sin incidencias
Cambio de opacidad	CU-03	Sin incidencias

Tiempo total: 1 minuto 20 segundos

Usuario 4

Tabla 15: Resultados del Test 1 para el Usuario 4

Tarea	Caso de uso	Observaciones
Selección inicial de mano de dibujo	CU-01	El usuario no sabía que el guardado era automático
Comienzo de dibujo desde cámara	CU-02	Sin incidencias
Cambio de opacidad	CU-03	Sin incidencias, aunque el usuario sugiere añadir una etiqueta indicando que el control de opacidad corresponde a ese atributo

Tiempo total: 29 segundos

5.3.2 Test 2

Para el segundo test hay que continuar un dibujo y realizar ajustes de tamaño, posición y rotación.

Usuario 1

Tabla 16:Resultados del Test 2 para el Usuario 1

Tarea	Caso de uso	Observaciones
Continuar dibujo	CU-06	Sin incidencias
Ajuste de tamaño	CU-03	Sin incidencias
Ajuste de posición	CU-03	Sin incidencias
Ajuste de rotación	CU-03	Sin incidencias

Tiempo total: 24 segundos

Usuario 2

Tabla 17:Resultados del Test 2 para el Usuario 2

Tarea	Caso de uso	Observaciones
Continuar dibujo	CU-06	Sin incidencias
Ajuste de tamaño	CU-03	Sin incidencias
Ajuste de posición	CU-03	Sin incidencias
Ajuste de rotación	CU-03	Sin incidencias

Tiempo total: 24 segundos

Usuario 3

Tabla 18:Resultados del Test 2 para el Usuario 3

Tarea	Caso de uso	Observaciones
Continuar dibujo	CU-06	Sin incidencias
Ajuste de tamaño	CU-03	Sin incidencias
Ajuste de posición	CU-03	Sin incidencias
Ajuste de rotación	CU-03	Sin incidencias

Tiempo total: 23 segundos

Usuario 4

Tabla 19:Resultados del Test 2 para el Usuario 4

Tarea	Caso de uso	Observaciones
Continuar dibujo	CU-06	Sin incidencias
Ajuste de tamaño	CU-03	Sin incidencias
Ajuste de posición	CU-03	Sin incidencias
Ajuste de rotación	CU-03	Sin incidencias

Tiempo total: 39 segundos

5.3.3 Test 3

En el último test el usuario tendrá que abrir el primer elemento de la vista de Ayuda, cambiar sus preferencias de mano de dibujo y para finalizar comenzar un dibujo desde la galería.

Usuario 1

Tabla 20:Resultados del Test 3 para el Usuario 1

Tarea	Caso de uso	Observaciones
Abrir el primer elemento de la vista de Ayuda	CU-05	Sin incidencias
Cambiar preferencias de mano de dibujo	CU-04	Sin incidencias
Comenzar dibujo desde galería	CU-02	Sin incidencias

Tiempo total: 25 segundos

Usuario 2

Tabla 21:Resultados del Test 3 para el Usuario 2

Tarea	Caso de uso	Observaciones
Abrir el primer elemento de la vista de Ayuda	CU-05	Sin incidencias
Cambiar preferencias de mano de dibujo	CU-04	Sin incidencias
Comenzar dibujo desde galería	CU-02	Sin incidencias

Tiempo total: 33 segundos

Usuario 3

Tabla 22:Resultados del Test 3 para el Usuario 3

Tarea	Caso de uso	Observaciones
Abrir el primer elemento de la vista de Ayuda	CU-05	Sin incidencias
Cambiar preferencias de mano de dibujo	CU-04	Sin incidencias
Comenzar dibujo desde galería	CU-02	Sin incidencias

Tiempo total: 22 segundos

Usuario 4

Tabla 23:Resultados del Test 3 para el Usuario 4

Tarea	Caso de uso	Observaciones
Abrir el primer elemento de la vista de Ayuda	CU-05	Sin incidencias
Cambiar preferencias de mano de dibujo	CU-04	Sin incidencias
Comenzar dibujo desde galería	CU-02	Sin incidencias

Tiempo total: 21 segundos

5.3.4 Conclusiones

Como se puede apreciar el principal problema y lo que ha supuesto la mayor pérdida de tiempo para los usuarios es que no se informa de que el guardado de la mano de dibujo es automático ni se notifica que el ajuste ha sido guardado.

El segundo problema planteado es que el control de opacidad es poco visible o no indica que corresponde a ese ajuste.

Será necesario para una próxima versión de la aplicación revisar ambos puntos teniendo en cuenta los comentarios realizados por los usuarios.

Cabe comentar que tres de los cuatro usuarios han apuntado que la propuesta de la aplicación resulta muy interesante y resulta de utilidad para trazar las líneas generales del dibujo. Coinciden en que mejorando la usabilidad de la aplicación y solucionando el tener que sostener el dispositivo todo el tiempo podría resultar de gran utilidad.

6. Conclusiones y trabajo futuro

En este apartado se presentan las conclusiones técnicas y personales tras el trabajo realizado y una vez compilada y testeada la aplicación, tanto por el autor como por potenciales usuarios reales. También la relación con los estudios y el trabajo futuro.

6.1 Conclusiones técnicas

Recordando los objetivos técnicos expuestos inicialmente, se pretendía desarrollar una aplicación móvil basada en tecnologías web con funcionalidades de realidad aumentada para asistir a artistas, profesionales y estudiantes con sus dibujos a mano alzada. Se tomaría una imagen de referencia y se mostraría mediante la realidad aumentada sobre la superficie de dibujo.

Una vez generada una primera versión funcional de la aplicación las conclusiones resultan bastante positivas desde el punto de vista del autor.

Se ha conseguido una aplicación que cumple las expectativas iniciales y que potencialmente puede resultar muy útil realizando las mejoras oportunas de usabilidad.

Para tener una visión más concreta del nivel de completitud de la aplicación se van a revisar los requisitos planteados en la Tabla 1 situada en el punto 2.3. No se tienen en cuenta los requisitos concernientes al apartado *Won't have*.

Tabla 24: Cumplimiento de los requisitos planteados inicialmente

Prioridad	Requisitos	Se cumple
Debe tener	Superponer imagen de referencia sobre la cámara	Sí
	Galería del dispositivo como fuente de imágenes	Sí
	Ajuste de opacidad de la imagen de referencia	Sí
	Ajuste de tamaño de la imagen de referencia	Sí
	Ajuste de perspectiva	Sí
Debería tener	Marcadores para mejorar encuadre y perspectiva	Sí
	Situar marcadores en función de la mano de dibujo (diestro – zurdo)	Parcialmente
	Ajuste automático de perspectiva	Sí
	Cámara del dispositivo como fuente de imágenes	Sí
	Panel con los proyectos comenzados	Sí
Podría tener	Filtro de detección de bordes para facilitar el trazado	No
	Filtro monocromo para facilitar el trazado	No

Como se puede observar se cumple con todos los requisitos que, según se planteó, debe tener la aplicación. Además se cumple con cuatro de los cinco requisitos que debería tener. La situación de los marcadores según la mano de dibujo se cumple parcialmente, ya que si bien se recoge en las Preferencias la mano de dibujo del usuario y se le sugiere situar el marcador en una u otra posición en función de ese dato, esto no tiene ninguna consecuencia en la vista de Realidad Aumentada de la versión que concluye este trabajo. En el apartado de "Trabajo futuro" se profundizará más en cuestiones interesantes que se podrían desarrollar sobre este requisito.

En cuanto a los dos requisitos que la aplicación podría tener, no se ha implementado ninguno, pero pueden ser interesantes para versiones futuras, también se comentará más adelante.

6.2 Conclusiones personales

Desde el punto de vista personal el objetivo marcado por el autor era reforzar sus conocimientos en el desarrollo de aplicaciones móviles y uso de tecnologías web.

Se ha trabajado el desarrollo de aplicaciones móviles híbridas para Android, aunque las tecnologías utilizadas permitirían la compilación para iOS con pocos ajustes.

Al estar la aplicación basada en tecnologías web se ha adquirido experiencia en el uso de estas aplicadas a la programación y maquetación para dispositivos móviles.

Tras el periodo de estudio y desarrollo que ha conformado la realización del presente trabajo el autor considera cumplidos los objetivos.

6.3 Relación con los estudios cursados

A lo largo del Grado en Ingeniería Informática el autor ha cursado una serie de asignaturas que han sentado la base para realizar este trabajo final.

El temario de asignaturas como *Programación*, *Estructuras de Datos*, *Tecnologías de Sistemas de Información en la Red*, *Bases de Datos* o Desarrollo *Centrado en el Usuario* o *Desarrollo Web* tiene una relación directa en mayor o menor medida con el desarrollo de ClarApp.

6.4 Trabajo futuro

En cuanto al trabajo futuro, el desarrollo de esta primera versión de ClarApp abre las puertas a una serie de mejoras y ampliaciones que pueden resultar de gran utilidad para los usuarios.

Como se ha visto en los resultados de los test de usabilidad es necesario mejorar la comunicación de los eventos y los resultados de las acciones al usuario. El ejemplo más claro es que ninguno de los usuarios que realizaron los test sabía si las preferencias se habían guardado o si podían continuar navegando tras la configuración inicial, lo que ha supuesto un cuello de botella importante.

Otra mejora a implementar en relación a la usabilidad es indicar con más claridad cómo realizar los ajustes sobre la imagen de referencia en la vista de Realidad Aumentada. Si bien las interacciones más directas con la imagen (rotar, escalar, desplazar) resultan bastante intuitivas, porque se corresponden con el funcionamiento

de la mayoría de aplicaciones, el control de opacidad ha resultado más difícil de ubicar y utilizar.

Si se pone el foco en la funcionalidad, queda pendiente profundizar en el uso de las preferencias de mano de dibujo. Una posibilidad sería mostrar instrucciones distintas para situar el marcador en la vista de realidad aumentada según la mano seleccionada.

Otra alternativa podría ser que la imagen de referencia no apareciera directamente sobre el marcador, sino desplazada a la derecha o la izquierda del mismo según la mano preferida.

Algo a realizar antes de avanzar mucho más en el desarrollo debería ser la adaptación a dispositivos basados en iOS, que gracias a Ionic será relativamente sencillo.

En una futura versión sería interesante incluir la posibilidad de utilizar filtros de detección de bordes y monocromo sobre la referencia para facilitar el dibujo, ya que proporcionarían una imagen más homogénea y de trazos más sólidos. Esto ya aparecía recogido como posibles requisitos a implementar.

En relación al manejo físico del dispositivo podría ser de utilidad buscar alternativas para su sujeción durante la sesión de dibujo, como un trípode portátil o gafas de realidad aumentada, siempre como algo opcional para el usuario.

En cuanto a los marcadores se podría realizar un diseño más significativo que facilitara su reconocimiento por la cámara desde una mayor distancia, añadiendo texturas o mayor variedad de colores en conjunto. Además, con miras a su distribución en tiendas de aplicaciones, se debería incluir en la app un documento imprimible con los marcadores.

Una vez resueltos los casos anteriores sería conveniente trabajar para que la aplicación fuera capaz de identificar los márgenes de la superficie de dibujo, por ejemplo un folio. De esta forma se podría llegar a prescindir de los marcadores físicos.

7. Referencias

[1] Muro, J. (2010). *La cámara lúcida y la cámara oscura*. Recuperado de http://eldibujante.com/la-camara-lucida-y-la-camara-oscura/

[2] Cámara lúcida. Recuperado de https://es.wikipedia.org/wiki/C%C3%A1mara_l%C3%BAcida

[3] Number of available applications in the Google Play Store from December 2009 to September 2018. Recuperado de https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/

[4] Number of available apps in the Apple App Store from 2008 to 2018 (in 1,000s). Recuperado de https://www.statista.com/statistics/268251/number-of-apps-in-the-itunes-app-store-since-2008/

[5] *SketchAR: How to draw with augmented reality*. Recuperado de https://play.google.com/store/apps/details?id=ktech.sketchar&hl=es

[6] *What is Ionic*. Recuperado de https://ionicframework.com/what-is-ionic

[7] *What is Angular*. Recuperado de https://angular.io/docs

[8] *Que es SPA (Single-page Application)*. Recuperado de https://medium.com/@programacionjje/que-es-spa-single-page-application-4dbd3694fac9

[9] *Apache Cordova: Introducción*. Recuperado de https://cordova.apache.org/docs/es/latest/guide/overview/

[10] *Getting started Cordova Plugin*. Recuperado de https://www.wikitude.com/external/doc/documentation/latest/phonegap/

[11] *Ionic CLI*. Recuperado de https://ionicframework.com/docs/cli/

[12] *Ionic: Getting Started*. Recuperado de https://ionicframework.com/docs/cli/#getting-started

[13] *ionic cordova plugin*. Recuperado de https://ionicframework.com/docs/cli/cordova/plugin/

[14] *Wikitude Ionic 3 starter app*. Recuperado de https://github.com/pbreuss/wikitude-ionic-3-starter-app

[15] *Wikitude SDK samples*. Recuperado de https://github.com/Wikitude/wikitude-sdk-samples