CAPÍTULO 1

Implementación

1.1. Introducción

En este capítulo se muestra la integración de los conocimientos adquiridos para poder llevar a cabo la realidad aumentada en una aplicación real. Si bien era de gran interés del proyecto la exploración de distintos métodos y algoritmos parecía importante poder poner en práctica todo lo desarrollado en un producto final que pudiera parecerse a un prototipo de aplicación comercial. En particular se desarrolló una aplicación pensando en los cuadros de la planta baja del Museo Nacional de Artes Visuales (MNAV). Entre otros autores, tiene cuadros de Pedro Figari, Juan Manuel Blanes y de Joaquín Torres García que se eligieron para hacer el prototipo.
La aplicación consta de distintas funcionalidades que se describen en este capítulo, tales como:

(1) Detección QR

(2) Navegación por listas de cuadros

(3) Comunicación con un Servidor con la base de datos.

(4) Detección SIFT para identificar el cuadro.

(5) Diferentes realidades aumentadas según la obra.

En las próximas secciones se describe más en detalle cada uno de estos puntos y su integración a la aplicación final. También se describe el flujo de la aplicación y algunas clases implementadas.

1.2. Diagrama global de la aplicación

Para que sea más sencilla la comprensión de los bloques que componen la aplicación a continuación se muestra el *Storyboard* de la aplicación que es la interfaz de usuario y que sirve para visualizar bastante cada una de las clases que intervienen y cómo es el flujo de la aplicación.
A continuación se pasará a explicar algunas de las clases implementadas en la aplicación y que tienen cierta relevancia. Se mostrarán sus principales características y su rol dentro de la aplicación.

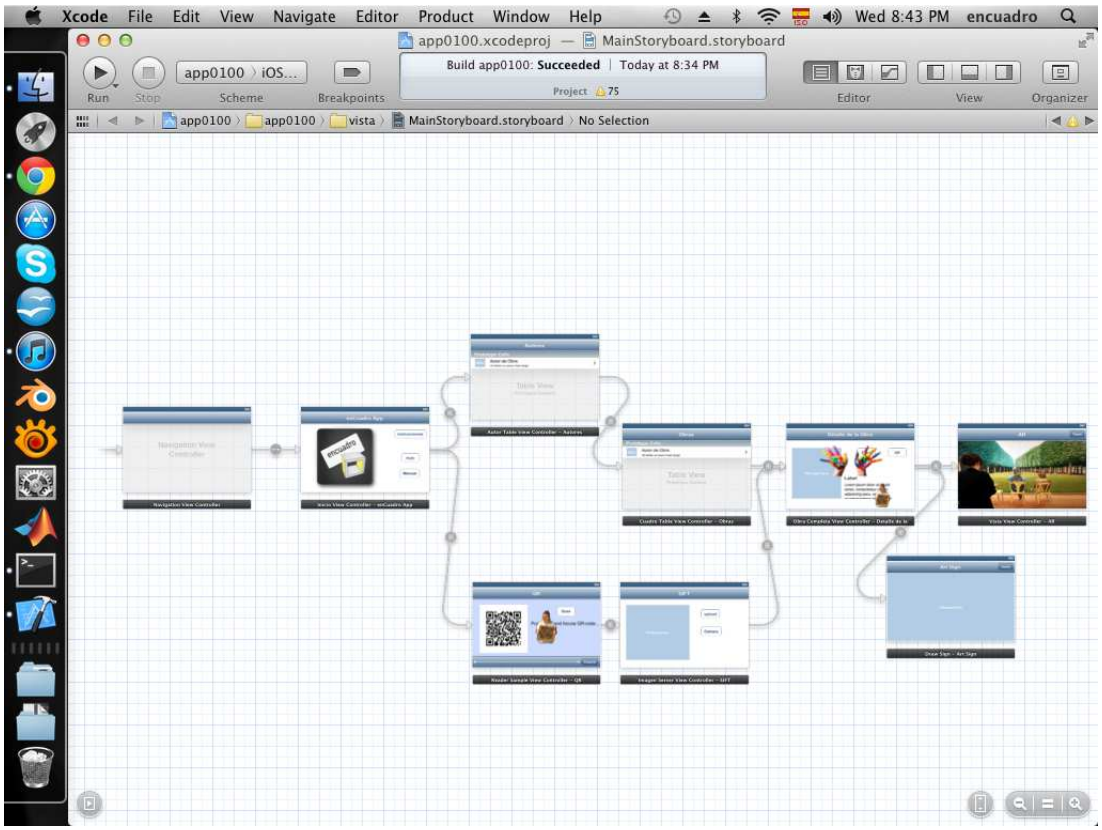

Figura 1.1: Diagrama global de la aplicación

1.2.1. NavigationViewController

La aplicación está embebida dentro de un *UINavigationController*. Esto implica que cada uno de los *ViewControllers* que tiene la aplicación es gestionado por esta clase. Es quien se encarga de la presentación y del pasaje de un *ViewController* a otro, creando y destruyendo instancias de cada uno. Está en esta clase la responsabilidad de manejar las jerarquías de los distintos *ViewControllers* así como de mantener cierta integridad visual utilizando las Toolbars ya sea arriba como encabezado o abajo al pie.
Para el caso particular de esta aplicación se optó por reimplementar esta clase ya que se buscaba tener cierto control sobre las rotaciones. En particular se reimplementaron los métodos *supportedInterfaceOrientations* y *preferredInterfaceOrientationForPresentation* de la siguiente manera

```
-(NSUInteger)supportedInterfaceOrientations
{
    NSLog(@"supportedInterfaceOrientations NAVIGATION");
    return UIInterfaceOrientationMaskLandscapeRight;
}

- (UIInterfaceOrientation)preferredInterfaceOrientationForPresentation
{
    NSLog(@"preferredInterfaceOrientationForPresentation NAVIGATION");
    return UIInterfaceOrientationLandscapeRight;
}
```

Esto lo que hace es fijar la orientación de la interfaz de usuario a modo *LandscapeRight*. También es posible lograr esto editando el archivo *Info.plist* que toda aplicación de xcode tiene y agregando el

item *SupportedInterfaceOrientations* y completado las opciones que se desean. Las autorotaciones es algo que tiene bastante relevancia en las aplicaciones. En particular se optó por esta forma, es decir, bloquear las autorotaciones y dejar solamente la vista en un sentido, para facilitar la reproyección de la realidad aumentada. De no haberlo hecho de esta manera, con cada rotación de la interfaz se tendrían que intercambiar los ejes de coordenadas en función del sentido de la rotación. Esto es posible de hacer ya que con cada rotación se ejecuta una serie de métodos en forma automática entre los cuales se encuentra el siguiente

```
- (void) willRotateToInterfaceOrientation:(UIInterfaceOrientation)toInterfaceOrientatio
```

Dentro de dicho método sería posible hacer el ajuste de coordenadas ya que el mismo se ejecuta previo a cualquier rotación. El tema de los métodos que son ejecutados al haber un evento del tipo rotación es algo que ha sufrido cambios recientes con la actualización de software a iOS 6. De todas formas, como ya se dijo, se optó por hacer esto más sencillo, simplemente evitando que la interfaz de usuario rote junto con el dispositivo, dejándola fija.

1.2.2. InicioViewController

Este *ViewController* es la pantalla de inicio de la aplicación. En la misma hay un botón que al ser presionado comienza un audio con instrucciones y una presentación sobre cómo es el recorrido y las funcionalidades con las que cuenta la aplicación. También hay dos botones más que dan la opción de elegir la forma de recorrer el museo. El botón de recorrido automático es un *segue* al *ReaderSampleViewController* y el de recorrido manual un *segue* al *AutorTableViewController*.

1.2.3. ReaderSampleViewController

Este *ViewController* es el encargado de hacer la lectura de los código QR y de invocar a los métodos necesarios para realizar la búsqueda de la zona del museo en la que se encuentra el usuario. Esto es, existe un código QR asociado a cada autor (Blanes, Figari y Torres García) y en base al código QR leído se despliega un texto e imagen asociadas al mismo. El funcionamiento de la decodificación se explica un poco más en detalle en 1.4.

1.2.4. ImagenServerViewController

Este *ViewController* es el encargado de la comunicación con el servidor. En el método *viewDidLoad* se instancia un *UIImagePickerController* encargado de implementar una captura de imagen. Una vez que se saca la foto la misma se muestra en una *UIImageView* y existen dos botones. Uno de ellos simplemente dispara una nueva instancia del *UIImagePickerController* dando la opción de sacar otra foto y el otro botón inicia la comunicación con el servidor.

PONER FOTO CON LA CAPTURA

El botón de *upload* es un *segue* hacia el *ObraCompletaViewController*. Dentro del método *prepareForSegue*, encargado de preparar todo previo a la invocación de *ObraCompletaViewController* se invoca el método *uploadImage*. Este método genera un mensaje HTTP del tipo POST y se lo envía a la IP del servidor. En el cuerpo del mensaje se agrega la foto sacada previamente y se le agrega un string *room*. Este string es completado previamente en el *ReaderSampleViewController* en base al QR detectado, dando la información sobre en qué sala/región del museo se encuentra (sala Figari, sala Blanes o sala Torres García). Este string lo que permite es tener un identificador

para poder hacer la búsqueda de la imagen sacada (luego de procesar la imagen con el algoritmo SIFT), en una base de datos más peque?a, que contenga solamente los cuadros de esa región del museo y no todos los cuadros del museo. En caso que el usuario no haya detectado ningún QR y haya seleccionado directamente la opción de sacar una foto para comenzar la comunicación con el servidor, entonces este string estará vacío y la búsqueda de la imagen más parecida se hace en toda la base de datos. En un primer caso, claramente los tiempos son mejores (del orden de 3s en una LAN) que en el segundo (del orden de 6s en una LAN).
Luego de establecida la conexión y enviada la consulta POST, el servidor responde con un string *returnString*. Este string contiene el resultado del procesamiento de la imagen enviada. Esto se logra mediante un archivo *upload.php* en el servidor que toma imágenes, ejecuta comandos en terminal (en este caso el ejecutable del algoritmo SIFT) y el resultado de esa ejecución se retorna como string en el php y el mismo se envía a la aplicación. Ese string resultado es recibido por la aplicación con una nomenclatura elegida que sigue la lógica Autor-Número, por ejemplo *Figari3* que se corresponde con la obra número 3 de la base de datos del autor Figari. Ese pasa a ser el identificador de la obra para ir a buscarla a la base de datos del servidor. El servidor cuenta con varias carpetas según la información:

(1) autor

(2) obra

(3) texto

(4) imagen

(5) audio

Para una obra dada, cuyo identificador sea *Figari3* por ejemplo, los archivos correspondientes, con la información son:

(1) http://IPservidor/autores/Figari3.txt

(2) http://IPservidor/obras/Figari3.txt

(3) http://IPservidor/textos/Figari3.txt

(4) http://IPservidor/imagenes/Figari3.jpg

(5) http://IPservidor/audios/Figari3.mp3

De esta manera una vez que se obtiene el identificador se hace una consulta al servidor que pide los distintos archivos de texto, audio e imágenes. Esta información solicitada es devuelta por el servidor y alojada en variables para ser mostradas (imagenes, texto) y reproducidas (audio) en el siguiente *ViewController*, en el *ObraCompletaViewController*.

1.2.5. TableViewControllers

o una opción manual en la que el usuario es quien elige la zona del museo (autor) y el cuadro en el que está interesado.

1.2.5.1. AutorTableViewController

1.2.5.2. CuadroTableViewController

1.2.5.3. CuadroTableViewCell

1.2.6. ObraCompletaViewController

1.2.7. VistaViewController

1.2.8. DrawSign

1.2.9. TouchVista

1.2.10. Isgl3dViewController y app0100AppDelegate

Una vez que se está frente al cuadro existe una serie de interacciones habilitadas para el usuario, como ser: audioguía sobre el cuadro en particular, dibujo con lienzo libre para realizar tweets con la imagen dibujada, o la realidad aumentada vinculada a la obra que seleccionó (recorrido maual) o la que fue derivada luego del reconocimiento de imagenes (recorrido automático)

SEGUIR REDACTANDO ESTA SECCION

1.3. TableViewController

jjjjj

1.4. QR

1.4.1. QR. Una realidad

El uso de los identificadores QR (Quick Response), es cada vez más generalizado. Últimamente debido al incremento significativo del uso de *smart devices* el hecho de poder contar con cámara y poder de procesamiento hace que sea frecuente encontrar aplicaciones con el poder de reconocimiento de QRs. Comenzaron a utilizarse en la industria automovolística japonesa como una solución para el trazado en la línea de producción pero su campo de aplicación se ha diversificado y hoy en día se pueden encontrar también como identificatorios de entradas deportivas, tickets de avión, localización geográfica, vínculos a páginas web o en algunos casos también como tarjetas personales.

1.4.2. Qué son realmente los QRs?

Se puede decir que los QRs tienen muchos puntos en común con los códigos de barras pero con la ventaja de poder almacenar mucho más información debido a su bidimensionalidad. Existen distintos tipos de QRs, con distintas capacidades de almacenamiento que dependen de la versión, el tipo de datos almacenados y del tipo de corrección de errores. En su versión 40 con detección de errores de nivel L, se pueden almacenar alrededor de 4300 caracteres alfanuméricos o 7000 dígitos

(frente a los 20-30 dígitos del código de barras) lo cual lo hace muy flexible para cualquier tipo de aplicación de identificación.
En la figura 1.1 se pueden ver las distintas partes que componen un QR como ser el bloque de control compuesto por las tres esquinas que dan información de la posición, alineamiento y sincronismo, así como también información de versión, formato, corrección de errores y datos. Fuera de toda esa información que podríamos denominar encabezado haciendo analogía con los paquetes de las redes de datos se encuentra la información a almacenar propiamente dicha que conforma el cuerpo del QR.

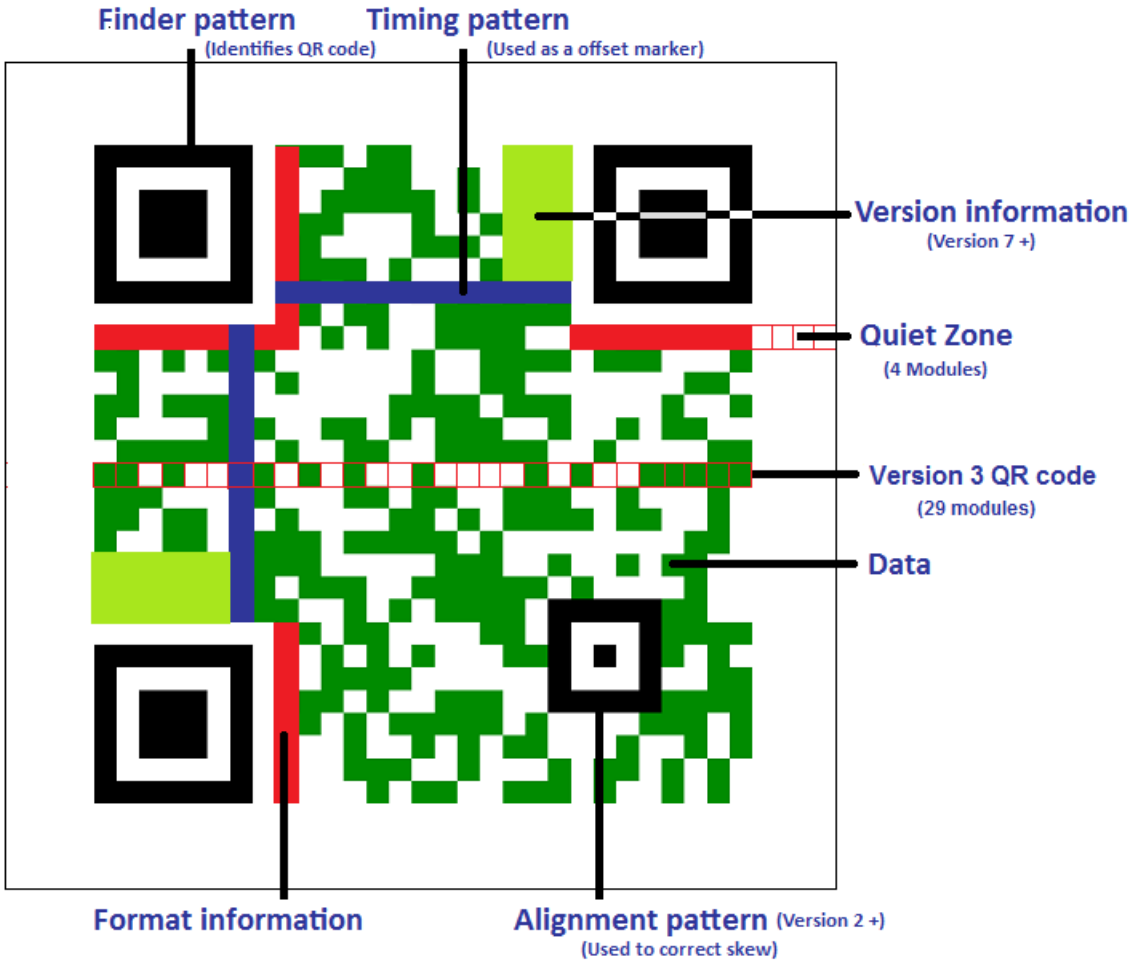

Figura 1.2: Las distintas componentes de un QR. Fuente [**?**].

1.4.3. Codificación y decodificación de QRs

Es fácil darse cuenta que la codificación resulta mucho más sencilla que la decodificación. Para la codificación es necesario comprender el protocolo, las distintas variantes y el tipo de información que se pretende almacenar. Sin embargo para la decodificación, además de tener que cumplir con lo anterior, es necesario contar con buenos sensores y ciertas condiciones de luminosidad y distancia que favorezcan a la cámara y se traduzcan en buenos resultados luego de la detección de errores. Si bien la plataforma es importante para lograr buenos resultados, dada una plataforma, existen variadas aplicaciones tanto para iOS como para Android que cuentan con performances bastante diferentes en función del algoritmo de procesamiento utilizado.
Debido a que el centro del proyecto de fin de carrera no fue la codificación y decodificación de QRs y que además ya existen distintas librerías que resuelven este problema se optó por investigar las distintas variantes e incorporar la más adecuada para la aplicación.
Dentro de todas las librerías que resuelven la decodificación se encuentran ZXing y ZBar como las más destacadas por su popularidad, simplicidad y buena documentación para la fácil implementación. ZXing, denominada así por "Zebra Crossing", es una librería open-source desarrollada en java y que tiene implementaciones que están adaptadas para otros lenguajes como C++, Objetive C o JRuby entre otros.
Por su parte ZBar también tiene soporte sobre varios lenguajes y cuenta con un SDK interesante para desarrollar fácilmente aplicaciones que integren el lector de QR. Se trabajó sobre el código de ejemplo que contiene la implementación de las clases principales para obtener un lector de QRs. Básicamente consta de una clase *ReaderSampleViewController* que hereda de *UIViewController* y que implementa un protocolo llamado *ZBarReaderDelegate* Al presionarse el botón de detección se crea una instancia de la clase *ReaderSampleViewController* y se presenta la vista de cámara. Luego el protocolo se encarga de la captura y procesamiento del QR teniendo como resultado la información que tiene el QR en la variable denominada *ZBarReaderControllerResults*. Esta variable luego se mapea en una hash table con el contenido en formato *NSDictionary*. De esta manera se accede fácilmente al contenido en formato legible y es fácil de hacer una lógica de comparación y búsqueda en una base de datos.

1.4.4. El QR en la aplicación

Para el caso particular de la aplicación se optó por tener un identificador QR para tres artistas elegidos del Museo Nacional de Artes Visuales (MNAV). Los mismos fueron Pedro Figari, Joaquín Torres García, Juan Manuel Blanes. De esta manera para el caso del recorrido del museo a través de la utilización con QRs es posible determinar la posición del usuario debido a imágenes QR debidamente ubicadas en cada zona. Esto sirve como localización y también sirve para lograr que el paso siguiente, que es la identificación de la obra que el usuario tiene enfrente, sea mediante una búsqueda en una base de datos discriminada por autor. Es decir, si el usuario no escanea el QR la búsqueda de la obra a identificar se hará en una base de datos global del museo, pero para el caso que el usuario sí decida escanear el QR entonces se cuenta con la posibilidad de realizar la búsqueda en una base de datos más reducida.

1.4.5. Arte con QRs

La opción de usar los QRs de una manera distinta ha comenzado a ser notoria en los últimos tiempos. Hay quienes desafían a la información *cruda de 1s y 0s* incorporando imágenes y modif-

icando colores y contornos en los QRs tradicionales para lograr un valor estético además del funcional. A continuación se muestran algunos ejemplos de tales casos en los que claramente se ve cómo puede lograrse el mismo resultado de información con el valor agregado de originalidad.

Figura 1.3: Ejemplo de un QR artista. Fuente [**?**].

1.5. Servidor

Si bien el desarrollo de la aplicación es un prototipo de una aplicación comercial y para tal caso no se manejan muchas imágenes y otros datos y registros, para lograr escalabilidad se hace imprescindible contar con un servior. Se pensó con el fin de almacenar toda aquella información relevante en cuanto a registro de obras (imagen, título y autor), descripciones de obras, audioguías, videos, modelos y animaciones para las realidades aumentadas asociadas y cualquier tipo de información que el museo quiera agregar y que por un tema de practicidad no se quiera almacenar dentro de la aplicación. En definitiva, almacenar toda esa información dentro de la aplicación quizá sea rentable para pocas obras, pero lo puede hacer inmanejable para un buen número de obras. Se pensó entonces en la instalación de un servidor que esté ubicado dentro del museo con el cual se tenga una conexión a través de una LAN de (54Mbps). Se aclara este punto pues, en caso de querer hacer un servidor remoto que tenga que ser accedido a través de internet, entonces baja notoriamente su performance, aunque funciona perfectamente.

1.5.1. Creando el servidor

Para la creación del servidor se buscó primeramente la alternativa de hacerlo sobre una máquina con sistema operativo con núcleo Linux, distribución Ubuntu. Luego también se buscó la posibilidad de tener el servidor corriendo sobre una plataforma Unix iOS. Para el segundo caso resultó incluso más sencilla que la primera dado que ya viene pensado por el sistema operativo el hecho de que funcione como servidor. A continuación se explica los pasos que se siguieron para implementar servidores en uno y otro sistema operativo.

1.5.1.1. Servidor iOS

redactar pasos...

1.5.1.2. Servidor LAMP

Se denota servidor LAMP por las siglas de Linux (Sistema Operativo), Apache (Servidor Web), MySQL (Gestor de base de datos), PHP/Perl/Python (lenguaje de programación).
Se instaló entonces el servidor Web Apache, que tiene dentro de sus principales ventajas el hecho de ser multiplataforma, gratis y de código abierto. Para eso desde terminal se debe hacer lo siguiente:

```
sudo apt-get install apache2
```

Con este comando se descarga el paquete *apache2* y se instala. Una vez finalizada la instalación de este paquete ya se cuenta con un servidor y se puede verificar ingresando desde la máquina donde se instaló el servidor abriendo el navegador e ingresando a *http://localhost* o equivalentemente a *http://127.0.0.1* y de esta manera aparece la página por defecto cuyo contenido está dado por el archivo */var/www/index.html*.
Luego de tener instalado el Apache se procede a instalar el php de la siguiente manera

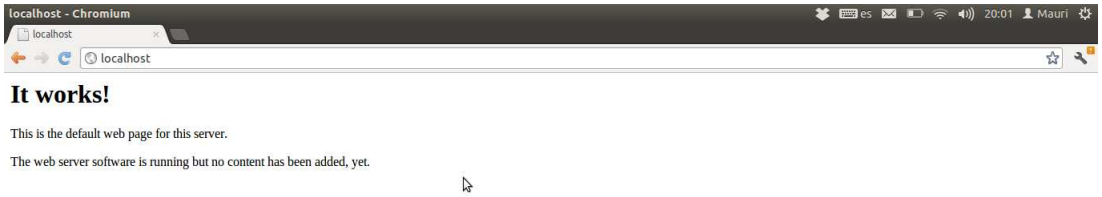

Figura 1.4: Impresión de pantalla al ingresar a http://localhost.

```
sudo apt-get install php5
```

Para el caso particular de los intereses del servidor creado no fue necesario instalar MySQL. Entonces con esto, luego de reiniciar el servidor apache ya se tiene un servidor con intérpetre php instalado. A partir de este momento todo se reduce a comprender bien el lenguaje php y poder realizar peque?os módulos de programación que puedan tomar entradas, procesarlas y arrojar una salida.

1.5.2. Lenguaje php y principales scripts

Como fue dicho en la sección anterior para los intereses el servidor creado, lo fundamental es el hecho de poder recibir archivos o identificadores de los mismos, poder realizar un procesamiento

en el servidor y devolver a la máquina cliente un archivo, mensaje o similar. Para esto se pasa a explicar algunos conceptos básicos de php.
El análogo a un Hello World en php es así:

```
<?php
echo"Hola Mundo";
?>
```

Esto se guarda en un archivo que con extensión .php en la ruta por defecto donde se alojan los php /var/www/holamundo.php. De esta manera al ingresar a la dirección *http://localhost/holamundo.php* se ve el print del texto.
como leer un input y hacer algo (ejemplo suma.php)
como hacer un acction.php ....
MOU SIGUE ESTO...

1.6. SIFT

1.7. Incorporación de la realidad aumentada a la aplicación

asdfasdfasdf

[**?**].