EXPLICACIONES MEDIANTE RA DE SISTEMAS DE RECOMENDACIÓN SOCIAL EN EL DOMINIO DEL OCIO

Diego Acuña Berger, Daniel Calle Sánchez, Carlos Gómez Cereceda, Zihao Hong

GRADO EN INGENIERÍA DEL SOFTWARE. FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

Trabajo Fin de Grado en Ingeniería del Software

Fecha

Director/es y/o colaborador:

Juan Antonio Recio García
Guillermo Jiménez Díaz

Autorización de difusión

Diego Acuña Berger, Daniel Calle Sánchez, Carlos Gómez Cereceda, Zihao Hong

Fecha

Los abajo firmantes, matriculados en el Grado en Ingeniería del Software de la Facultad de Informática, autoriza a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores el presente Trabajo Fin de Grado: "EXPLICACIONES MEDIANTE RA DE SISTEMAS DE RECOMENDACIÓN SOCIAL EN EL DOMINIO DEL OCIO", realizado durante el curso académico 2018-2019 bajo la dirección de Juan Antonio Recio García y Guillermo Jiménez Díaz en el Departamento de Ingeniería del Software e Inteligencia Artificial, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet y garantizar su preservación y acceso a largo plazo.

Resumen

Hemos desarrollado una aplicación móvil con la que poder identificar carteles de distintas películas para obtener información en tiempo real de éstas mediante el uso de Realidad Aumentada, tales como su trailer o su valoración. Además, se incluye la posibilidad de guardar esta información o incluir la película en un plan, para su posterior visualización según la recomendación que nos dé la aplicación en base a los gustos de los usuarios que están dentro del plan y las películas que se han añadido. También hemos realizado un pequeño ejemplo de reconocimiento facial de otros usuarios para observar, mediante Realidad Aumentada, información sobre sus gustos o películas vistas anteriormente.

Palabras clave

Lista de palabras clave

- Realidad Aumentada
- Sistema de Recomendación
- Unity
- Android
- Spring
- OpenGL
- GLSL
- Vuforia
- MySQL
- PostgreSQL

Abstract

We have developed a mobile phone application with which we can identify different movie posters to obtain information in real time about them using Augmented Reality, such as the movie trailer or its rating. Also, we include the posibility to save this information or to include the film in a plan, for its later visualization regarding the recommendation that the application gives us based on the tastes of the users that are inside of a plan and the movies that have been added. We have also made a little example of facial recognition of other users to observar, through Augmented Reality, information of their tastes or about films they have seen previously.

Keywords

List of keywords

Índice general

Agradecimientos

Nos gustaría agradecer a nuestra familia y compañerxs la confianza depositada en nosotros así como la realimentación recibida por parte de toda persona que se ha involucrado en mayor o menor medida en la realización de nuestro proyecto, ya fuera mediante comentarios constructivos o la redirección a la documentación de tecnologías que usaríamos posteriormente.

Dedicatoria

Texto...

Capítulo 1

Introducción

1.1. Antecedentes

Esta idea surge de la propuesta de mejora de un Trabajo de Fin de Grado anterior al nuestro, cuyo enfoque y finalidad era el mismo. Nuestro objetivo principal es ampliar su funcionalidad, diferenciándonos, sobre todo, en las tecnologías usadas y en los casos de uso representados. En nuestra aplicación hemos cambiado el enfoque desde la perspectiva del usuario, no nos centramos únicamente en el acto de ir al cine y reconocer películas que estuvieran en la cartelera para que nuestra aplicación nos recomiende la que más se ajuste a nuestros gustos. Si no que, hemos abogado por la creación de planes, en los cuales podemos incluir películas que hayamos visto o que queramos ver y pueden unirse otros usuarios de la aplicación, haciendo así que su funcionalidad sea mucho más interactiva y más abierta al uso en el día a día. En cuanto a la diferencia desde el punto de vista tecnológico, hemos decidido usar Unity y Vuforia para el reconocimiento de imágenes y el uso de la Realidad Aumentada para aportar valor a la experiencia de usuario. En cuanto a la parte backend, hemos usado Spring como framework para nuestra API REST, ya que está siendo usado ampliamente a nivel empresarial actualmente.

1.2. Objetivos

- Estudiar las tecnologías de realidad aumentada actuales e implementarlas.
- Analizar sistemas de recomendación, así como su uso dentro del proyecto.
- Desarrollar una aplicación para dispositivos móviles que aplique las anteriores tecnologías en un proyecto real.
- Definir casos de uso reales que aporten valor a la aplicación.
- Administrar y configurar servicios web que provisionen de información a la aplicación.
- Desplegar un servidor y codificarlo de forma que su modificación sea lo más sencilla posible.
- Coordinar el trabajo en grupo y la gestión de los cambios mediante el uso de prácticas de las metodologías ágiles.
- Investigar la mejor forma de mejorar la experiencia del usuario de nuestra aplicación.
- Aplicar los conocimientos y disciplinas aprendidos de la Ingeniería del Software.

1.3. Plan de trabajo

Para comenzar analizaremos el estado del arte en el que aprenderemos los conceptos básicos de la **RA** y buscaremos las tecnologías actuales a las fechas del proyecto, que funcionan en dispositivos móviles.

Después realizaremos las tareas de diseño en las que desarrollaremos prototipos de tecnologías de **RA** en distintas plataformas y crearemos otros para la **API REST**. Realizaremos una lista de las funcionalidades más importantes de nuestra aplicación y diseñaremos bocetos para la interfaz de usuario que nos servirán como guía en la implementación.

El siguiente paso será la implementación del software para el cual aplicaremos algunas metodologías **SCRUM**, estará dividido en tres repositorios, uno para la **API REST**, y dos más uno de ellos para la parte de **RA** que se integra como una escena en el segundo un proyecto de **Android Studio**. El desarrollo se realizará con iteraciones cercanas a las dos semanas, comenzará con la selección de objetivos más prioritarios y terminará con las reuniones de revisión junto con los directores del **TFG**, posteriormente otra reunión de retrospectiva en la que participa únicamente el equipo servirá para realizar mejoras en la siguiente interacción.

Finalizaremos con una evaluación de la aplicación con distintos tipos de usuarios obteniendo **feedback**, que servirá para evaluar nuestro trabajo y realizar propuestas de futuro que aporten valor a nuestro proyecto.

Capítulo 2

Estado del arte

2.1. Realidad Aumentada

2.1.1. Tecnologías actuales

ARCore: Plataforma creada por Google para desarrollar aplicaciones de realidad aumentada con soporte para Android, Android NDK, iOS, Unity y Unreal Engine. Aunque las funcionalidades que se ofrecen para iOS y Unity for iOS se limitan a Cloud Anchors, los anchors sirven para hacer que objetos virtuales aparezcan en un lugar captado por la cámara de nuestro dispositivo estos son compartidos en la nube para que multitud de dispositivos disfruten de la misma experiencia, los dispositivos con iOS podrán usarlos utilizando ARKit.

Tiene una curva de aprendizaje media y con su versión 1.5 demuestra una estabilidad interesante respecto a su reciente creación. Cabe destacar que no todos los dispositivos son compatibles, esto depende de que las empresas que desarrollan estos dispositivos cumplan unos requisitos para asegurar que la experiencia con ARCore es la adecuada y de la versión del sistema operativo. se puede ver con más detalle en esta dirección https://developers.google.com/ar/discover/supported-devices.

ARCore usa tres características a través de la cámara del dispositivo:

- **Motion tracking** permite al dispositivo entender y rastrear la posición relativa del mundo.
- **Environmental understanding** permite al dispositivo detectar el tamaño y locali-

zación de todos los tipos de superficies.

- **Light estimation** permite al dispositivo estimar las condiciones de luz del entorno actual.

ARKit: Podemos utilizar experiencias de realidad aumentada persistente, compartirlas entre distintos dispositivos iOS, detecta imágenes 2D incluso en movimiento y objetos 3D.

Wikitude: Kit de desarrollo para realidad aumentada con soporte para Android, iOS, Unity, Cordova, Xamarin (mala documentación y versiones obsoletas), Titanium, React Native. Su licencia es de pago aunque hay versiones limitadas gratuitas. Utiliza ARCore o ARKit cuando los dispositivos lo soportan y en caso que no utiliza tecnología de Wikitude para que el número de dispositivos compatibles sea mayor.

Vuforia: Kit de desarrollo para realidad aumentada con soporte para Android, iOS, UWP y Unity. Software de pago solo permite usarse gratis para pruebas.

ViroReact: Plataforma para construir aplicaciones con realidad aumentada usando React Native. Utiliza ARKit y ARCore para dotar a las aplicaciones de una experiencia de RA sin utilizar código distinto y con una curva de aprendizaje fácil. Al basarse en React Native que no tiene versión estable provoca problemas con las versiones de dependencias, configuraciones tediosas y largas compilaciones. Es un software privativo gratuito.

Expo AR: API que permite crear aplicaciones en React Native utilizando ARKit únicamente. Está en una fase muy inicial.

2.2. Análisis de las necesidades del usuario

2.3. Entrevistas

2.4. Análisis de la competencia

2.5. Fuentes de información

2.6. Técnicas de recomendación

Capítulo 3

Diseño de la aplicación

3.1. Stakeholders

3.2. Escenarios

3.3. Requisitos funcionales

3.4. Interfaz de usuario

3.5. Sistema de recomendación

3.6. Prototipos

3.6.1. ARCore

3.6.2. Viro Media

El objetivo del prototipo realizado con **Viro Media** es reconocer imágenes almacenadas en el dispositivo para mostrar texto y objetos virtuales. Además de probar tecnologías de desarrollo móvil web como en este caso **React Native** para plataformas **iOS** y **Android**.

Comenzamos construyendo una interfaz sencilla con botones que nos redirigen a la escena de Realidad Aumentada. Para esta interfaz utilizamos **NativeBase** que es una librería que nos permite realizar una aplicación con apariencias de tipo **iOS** o **Android** según el dispositivo.

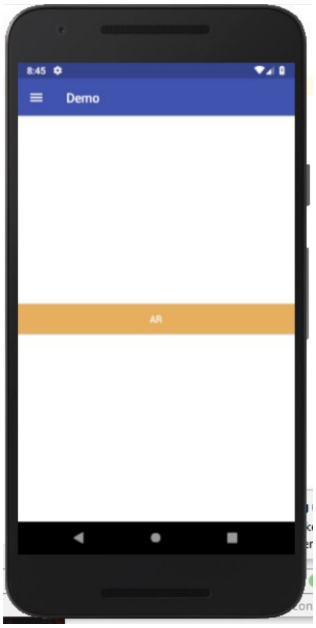


Figura 3.1: *Visualización con NativeBase en Android*

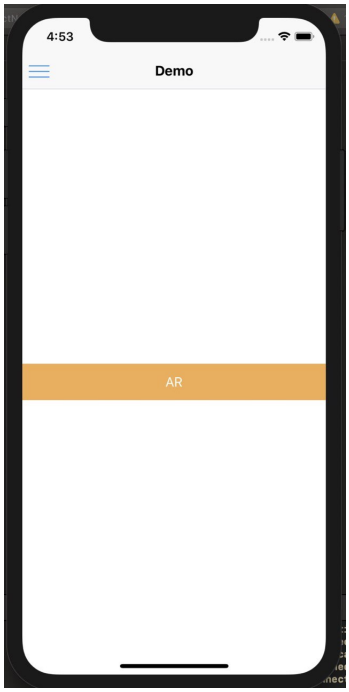


Figura 3.2: *Visualización con NativeBase en iOS*

Para la escena de Realidad Aumentada mostramos texto y al detectar el póster de Pantera Negra, reacciona mostrando una animación de dicho súper héroe saliendo del póster.

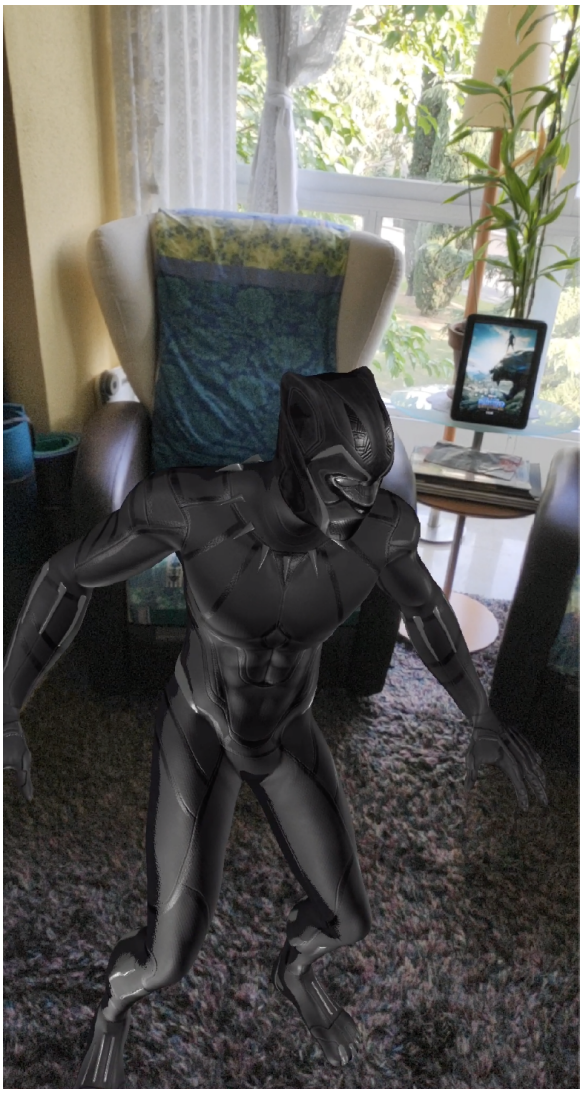


Figura 3.3: *Visualización de RA*

Una de las ventajas que apreciamos fue la facilidad del lenguaje, en este caso **Javascript**, utilizando la popular librería **ReactJS** y la buena documentación de **Viro Media** que hacian que el proceso de codificación fuera agradable.

Uno de los problemas que presentaba este prototipo era que las dependencias de **Viro Media** entraban en conflicto con las de **NativeBase** imposibilitándonos la forma de encontrar versiones compatibles. Utilizamos las últimas, que a pesar de lanzar advertencias, funcionaba en el ejemplo realizado. Otro problema fue la compilación de la

aplicación, **Viro Media** tiene una aplicación para probar lo que desarrollamos conectándose a nuestro ordenador a través de la red. El problema es que algunos recursos, como los iconos que utilizaba **NativeBase**, no eran descargados, por lo que la mejor forma era probar la versión compilada de **iOS** y **Android**. La forma de compilar la aplicación era un proceso costoso para los ordenadores, lento y con multitud de problemas según se ampliaban las librerías que se utilizan.

La conlusión que obtuvimos de este prototipo fue que **Viro Media** y **React Native** son tecnologías muy prometedoras, pero debido a los problemas surgidos y a que todas sus versiones no eran estables vimos un claro riesgo para nuestro proyecto.

3.6.3. Vuforia + Android

En este prototipo utilizamos la librería nativa de **Vuforia** para **Android** para realizar las pruebas de tecnología de reconocimiento de imágenes tanto en local como usando la nube que nos ofrecía **Vuforia**, para la posterior renderización de objetos y textos.
Las características tecnológicas de este prototipo son las siguientes:

1. La librería de **Vuforia** para **Android** está diseñada a muy bajo nivel.
2. **Vuforia** para dibujar en 3D usa la librería **OpenGL**.
3. **OpenGL** utiliza una serie de espacios donde se van colocando los elementos:
   a) **Local space**: Es el espacio local de cada objeto.
   b) **World space**: Es el mundo donde se encuentran los objetos.
   c) **View space**: El mundo visto desde la perspectiva de la cámara.
   d) **Clip space**: Se integra con la pantalla del móvil y, definiendo los límites visibles, se establecen unas coordenadas de rango (-1,-1) - (1,1).

   Las transformaciones de estos espacios se realizan mediante matrices 4x4, en las que la primera fila hace referencia a la coordenada x, la segunda a la coordenada y y la tercera a la coordenada z, mientras que la última columna hace referencia a los desplazamientos de los objetos en esos 3 ejes.

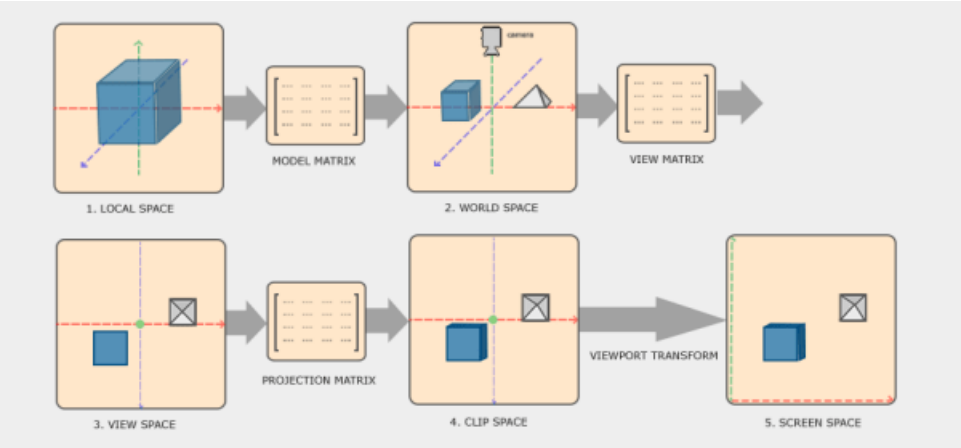


Figura 3.4: *Esquema de los distintos espacios que usa OpenGL*

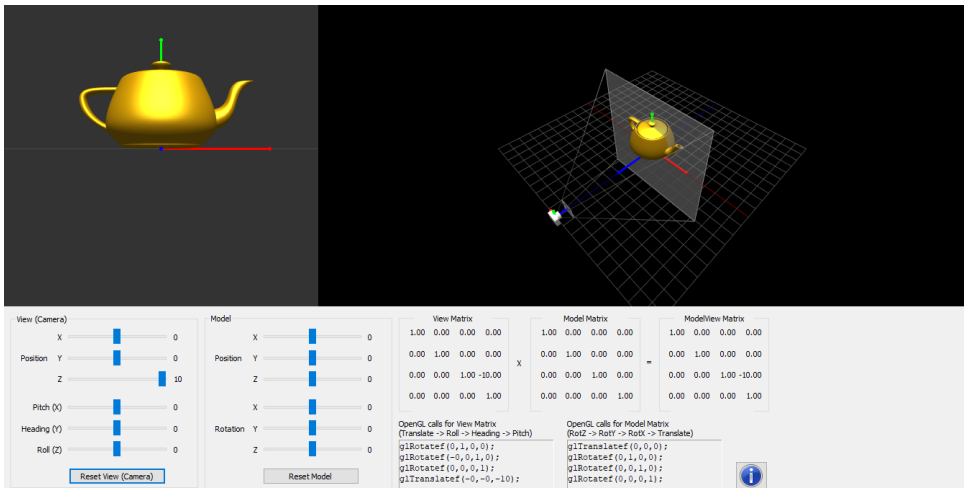


Figura 3.5: *Ejemplo de un modelo en 3D*

4. En **OpenGL** es necesario escribir código para que las tarjetas gráficas rendericen el modelo 3D, el lenguaje que se usa es **GLSL**. Este código de **GLSL** se escribe en forma de **String** y se llama a un método que proporciona **OpenGL**.

```
private String vertexShaderCode =
        // This matrix member variable provides a hook to manipulate
        // the coordinates of the objects that use this vertex shader
        "uniform mat4 uMVPMatrix;" +
        "attribute vec4 vPosition;" +
        "void main() {" +
        // The matrix must be included as a modifier of gl_Position.
        // Note that the uMVPMatrix factor *must be first* in order
        // for the matrix multiplication product to be correct.
        "  gl_Position = uMVPMatrix * vPosition;" +
        "}";
private String fragmentShaderCode =
        "precision mediump float;" +
        //"uniform vec4 vColor;" +
        "void main() {" +
        "  gl_FragColor = vec4(0.2f, 0.7f, 0.9f, 1.0f);" +
        "}";
```

Figura 3.6: *Código en GLSL*

5. Otro aspecto a tener en cuenta es que **OpenGL** solo nos ofrece lo básico, no nos ofrece métodos para dibujar directamente objetos sino que hay que seguir un pipeline de procesos para conseguir dibujar algo.

   Esto consiste en pasar un **array** de números (cada tres para definir un punto) a las tarjetas gráficas, establecer triángulos entre los puntos (más arrays de números) definir colores a partir de los puntos (más arrays)..., y con el código del shader, ejecutar estos datos.

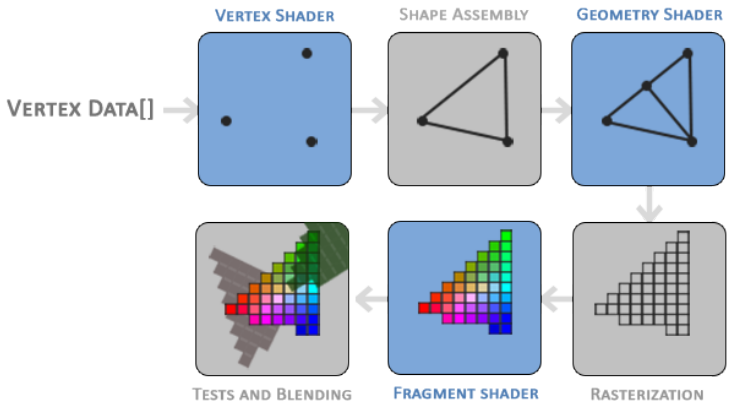


Figura 3.7: *Pipeline de la construcción de un modelo*

Figura 3.8: *Mapa de bits de caracteres usado*

6. Por último como sólo ofrece métodos básicos, no hay métodos de escritura de texto, y la forma que encontramos y que funcione fue usar un bitmap con los caracteres. Rechazamos esto por un principal motivo, para hacer que funciones hay que codificar a muy bajo nivel y nos costaría mucho tiempo y esfuerzo.

3.6.4. Vuforia + Unity

Para realizar este prototipo utilizamos **Unity** como herramienta básica para realizar la aplicación y **Vuforia** para dar soporte a la Realidad Aumentada. El prototipo a desarrollar consistió en un modelo 3D de un dragón que aparecía al detectar una imagen que previamente habíamos establecido como "imagen objetivo".

Figura 3.9: *Modelo en 3D que aparecía al detectar la imagen*

Pese a que nadie del equipo había utilizado **Unity** previamente el resultado fue bastante positivo ya que:

1. **Unity** resultó ser intuitivo y relativamente fácil en cuanto al aprendizaje de las funcionalidades básicas.
2. **Vuforia** parecía estar muy probada e incluía de serie muchas funcionalidades.
3. **Vuforia** tenía la opción de utilizar un **Cloud** para almacenar las imágenes objetivo.

3.6.5. Vuforia + Unity + Android

Una vez realizado el prototipo en **Vuforia** con **Unity** comenzamos a investigar como realizar el resto de la aplicación que no requería de Realidad Aumentada.
Hasta este momento teníamos claro que **Vuforia** con **Unity** era la mejor convinación para realizar la parte de Realidad Aumentada, sin embargo, **Unity** no era igual de intuitivo ni eficaz a la hora de realizar tareas propias de una aplicación "normal", como el desarrollo de interfaces o la lógica.
Por este motivo intentamos buscar la opción de realizar una aplicación en la que la Realidad Aumentada estuviese diseñada en **Unity** con **Vuforia** y el resto de la aplicación en Android.

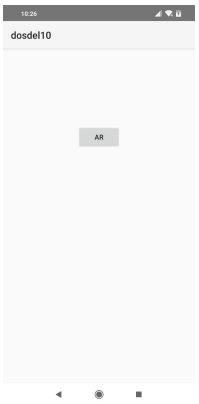


Figura 3.10: *Botón que comunicaba Android con Unity*

Finalmente conseguimos tener ambos proyectos independientes, la Realidad Aumentada se desarrollaba en **Unity** con **Vuforia** y se exportaba a un proyecto Android donde se encontraba el resto de la aplicación. Esto nos permitía realizar la Realidad Aumentada con la herramienta que tras las primeras tomas de contacto habíamos comprobado que era la mejor (**Unity** con **Vuforia**) y del mismo modo realizar el resto de la aplicación con la mejor herramienta para esta parte (AndroidStudio).

Tras realizar este prototipo, consideramos que estas herramientas podrían ser las que usásemos en la aplicación final puesto que:

1. Como ya habíamos descubierto en el prototipo anterior, **Unity** era una herramienta muy completa y junto con **Vuforia** nos proporcionaban todas las herramientas necesarias para cumplir con los casos de uso de Realidad Aumentada que teníamos en mente.

2. Al haber encontrado la forma de combinar **Unity** + **Android** no teníamos que renunciar a ninguna de las dos herramientas. Lo que nos permitía explotar las cosas buenas de ambas herramientas.

3. La comunicación entre **Unity** y Android era relativamente sencilla pese a ser dos proyectos distintos.

3.6.6. Server en Spring

Para la realización de la parte backend de la aplicación decidimos incorporar la tecnología de **Spring** para codificar un **servicio web REST** en **Java** y el acceso a datos mediante **MySQL**.

Para el prototipo seguimos el siguiente tutorial:

Cómo crear un microservicio o servicio web REST con Spring Boot

que consistía en 3 partes bien definidas para la creación de dicho **servicio web REST** para la gestión de una entidad de contactos muy simple, se puede observar su estructura en la figura 3.6.

```java
// (2)
@Entity
public class Contact implements Serializable {

    // (3)
    private static final long serialVersionUID = 4894729030347835498L;

    // (4)
    @Id
    @GeneratedValue
    private Long id;
    private String firstname;
    private String lastname;
    private String phonenumber;
    private String email;


    public Contact() {}

    public Contact(long id, String firstName, String lastName, String phoneNumber, String email) {
        this.id = id;
        this.firstname = firstName;
        this.lastname = lastName;
        this.phonenumber = phoneNumber;
        this.email = email;
    }
```

Figura 3.11: *Entidad de Contactos*

También se investigaron distintas formas de realizar el acceso a la base de datos desde el servidor pero finalmente nos decantamos por usar la clase **JPARepository** o **CRUDRepository**, las cuales nos ofrecen ya implementados los métodos típicos de las operaciones **CRUD**, además de la opción de poder crear nuestros propios métodos.

Para poder probar las distintas peticiones de tipo **POST** y **GET**, que realizamos en local, usamos la herramienta **Postman**, ver figura 3.7.

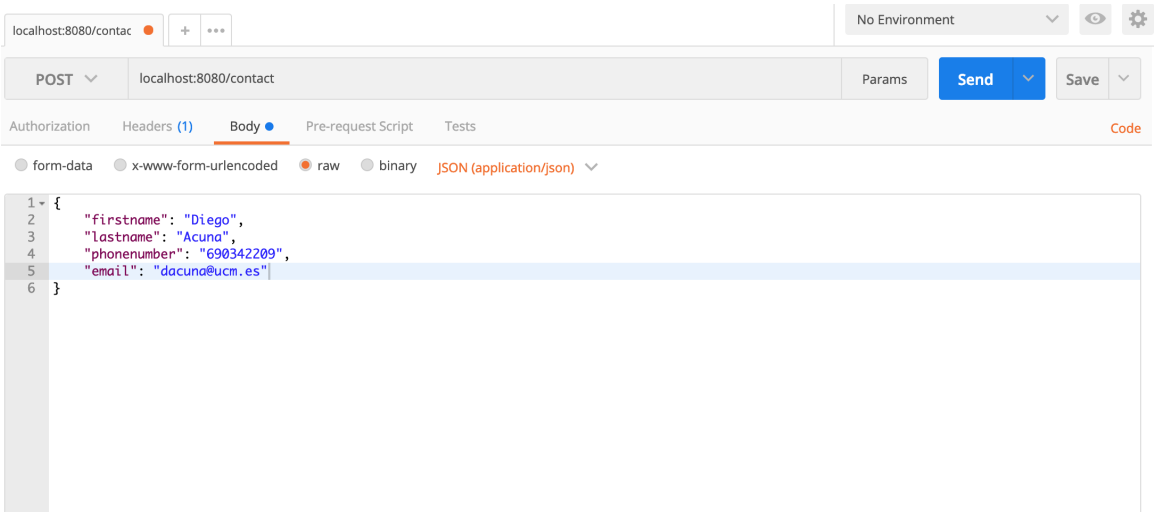


Figura 3.12: *Postman*

Para este prototipo decidimos usar **MySQL** como sistema de gestión de bases de datos relacional, aunque a la hora de probar a levantar nuestro servidor de prueba tuvimos que rehacer este prototipo, además de adaptarlo para que gestionara entidades de películas, y que usara **PostgreSQL**, además de cambiar una serie de anotaciones y limpiar el archivo **pom.xml** de líneas de código innecesarias, ver figura 3.8, que es el que contiene las dependencias de nuestro proyecto.

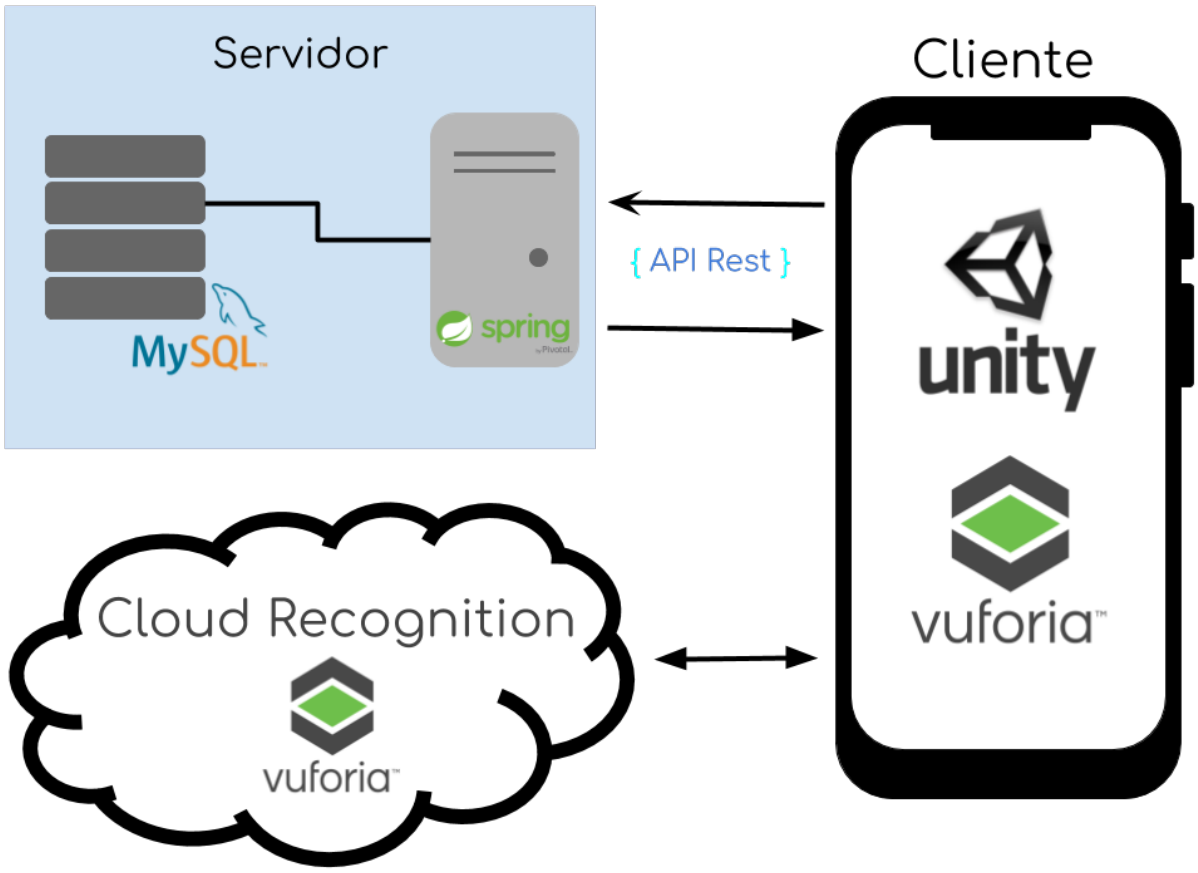
```xml
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0
    <modelVersion>4.0.0</modelVersion>

    <groupId>es.ucm.fdi.tfg.app</groupId>
    <artifactId>RESTApi</artifactId>
    <packaging>jar</packaging>
    <version>1.0-SNAPSHOT</version>
    <name>RESTApi</name>
    <url>http://maven.apache.org</url>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.0.5.RELEASE</version>
    </parent>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-tomcat</artifactId>
            <scope>provided</scope>
        </dependency>
```

Figura 3.13: *Archivo pom.xml*

Capítulo 4

Implementación

4.1. Arquitectura



4.2. Servidor

4.3. Aplicación

4.4. Dificultades encontradas

4.5. Herramientas de trabajo

Capítulo 5

Evaluación con usuarios

5.1. Método de evaluación

5.2. Resultado de la evaluación

Capítulo 6

Conclusiones

6.1. Conclusiones

6.2. Conclusions

Capítulo 7

Trabajo futuro

Capítulo 8

Plan de negocio

Capítulo 9

Contribución al proyecto

9.1. Diego Acuña Berger

9.2. Daniel Calle Sánchez

- Busqueda y configuración de herramientas que mejoren el rendimiento del equipo.
- Creación de prototipo con ViroReact y ARCore.
- Identificación de requisitos funcionales de la aplicación.
- Estudio e implantacion de servicios de autenticación.
- Configuración de Heroku.
- Diseño de la arquitectura.

9.3. Carlos Gómez Cereceda

9.4. Zihao Hong

Bibliografía

Apéndice A

Title for This Appendix

Apéndice B

Title for This Appendix