Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Grado en Ingeniería Informática

SHACU

Share Your Culture

Carlos Troyano Carmona

La Laguna, 12 de marzo de 2021

D. **Casiano Rodríguez León**, con DNI 42020072S profesor Titular de Universidad adscrito al Departamento de informática de la Universidad de La Laguna, como tutor

C E R T I F I C A (N)

Que la presente memoria titulada: *"Shacu"* ha sido realizada bajo su dirección por D. **Carlos Troyano Carmona**, con N.I.F. 78.851.937-W

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 12 de marzo de 2021

Agradecimientos

Hay mucha gente a la que tengo que agradecer estar aquí, además de que este proyecto sea posible. La primera persona a la que tengo que agradecer es a mi mejor amiga Siboney Luis Martín ya que ha estado a mi lado en todo momento en el desarrollo de este TFG y además ha participado activamente con los diseños tanto del logo como de los prototipos de implantación. También tengo que agradecer a mi familia por apoyarme en este camino que ha sido el grado de ingeniería informática. En especial a mi tío José Carmona Alonso, a mis abuelos, a mis padres y con especial devoción a mi hermana que me ha enseñado que rendirse no tiene porqué ser la opción.

También tengo que agradecer a las personas que me han apoyado a lo largo del camino cómo mi amigo Sergio Díaz Coello. Que siempre ha sido un referente para mí en el mundo universitario. Mi actual jefe en mi trabajo Jorge Elena. Que me ha permitido ajustar mis horarios para poder titularme además de ser mi mecenas en el mundo laboral ya que confió en mí y me dio la oportunidad de crecer.

Por último, agradecer a todos los profesores que me han enseñado lo duro y reconfortante que es recorrer este camino y en especial a mi tutor que desde el aula me enseñó el mundo de la programación web y ha seguido apoyándome a través de los años a pesar de no avanzar demasiado hasta el presente.

Licencia

© Esta obra está bajo una licencia de Creative Commons Reconocimiento 4.0 Internacional.

Resumen

El presente proyecto consiste en desarrollar e implementar una red social llamada Shacu. La principal funcionalidad proveída es la de compartir contenidos a través de códigos QR, localizando estos en un mapa y estableciendo una lectura por proximidad. Se da la posibilidad a sus usuarios de difundir materiales propios o información cultural acerca de un lugar asociado al QR.

El presente proyecto de ingeniería en sistemas sigue el Modelo Vista Controlador e implica dos partes. La primera es el cliente con el que interactúa el usuario (front end) codificado en AngularJS, conteniendo la vista de nuestra aplicación. La segunda contiene el servidor (back end) desarrollado en NodeJS conteniendo el controlador de nuestro proyecto y gestionando la base de datos, las sesiones de usuario y la seguridad. La base de datos se apoya en MySQL y se accede a ella a través del back end con un modelo de API. El diseño se ha realizado usando el lenguaje de modelado UML que nos ha facilitado la generación de los diagramas que han servido como base para la programación.

Las principales tecnologías y plataformas utilizadas en el proyecto son NodeJs, Angular, MySQL, Google Maps, OAuth, Git y Heroku.

Palabras clave: Shacu, QR, Googlemaps, Red Social, Angular, NodeJs, MySQL, Auth0, UML.

Abstract

The present project consists of developing and implementing a social network called Shacu. The main functionality provided is to share content through QR codes, locating these on a map and establishing a proximity reading. Users are given the possibility to disseminate their own materials or cultural information about a place associated with the QR.

This system engineering project follows the Controller View Model and involves two parts. The first is the client with which the user interacts (front end) encoded in AngularJS, containing the view of our application. The second contains the server (back end) developed in NodeJS containing the controller of our project and managing the database, user sessions and security. The database is supported by MySQL and accessed through the back end via an API. The design has been carried out using the UML modeling language that has facilitated the generation of the diagrams that have served as the basis for programming.

The main technologies and platforms used in the project are NodeJs, Angular, MySQL, Google Maps, OAuth, Git and Heroku.

Keywords: Shacu, QR, Googlemaps, Red Social, Angular, NodeJs, MySQL, Auth0, UML.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1
Introducción

En esta primera sección se pretende introducir breve y concisamente el Proyecto/Trabajo de Fin de Grado desarrollado por Carlos Troyano Carmona para el Grado en Ingeniería Informática de la Universidad de La Laguna. El proyecto presentado consiste en la implementación de una sencilla red social que permite intercambiar contenidos culturales entre sus usuarios a través de Códigos QR localizados por GPS.

1.1 Motivaciones principales del proyecto

Las redes sociales son un elemento que permiten comunicarse de diversas formas a las personas. El hecho de compartir cultura a través de estas suele ser una acción transversal a las propias redes sociales. Podemos pensar por ejemplo en los artistas que ponen su portfolio en Instagram o escritores que publican sus textos en Facebook.

Pero existen redes cómo Flickr que están diseñadas con el fin específico de compartir cultura, aunque no existe ninguna red social que sea simplemente un medio de compartir cultura, en sus formas digitales únicamente, excluyendo todos los demás contenidos. Por ello **Shacu** persigue dicho objetivo. Compartir contenidos culturales propios o legales de manera directa con el resto de los usuarios.

Por otro lado, compartir cultura desde la silla de tu salón o sentado en un ordenador, bajo un punto de vista personal, puede a llegar a ser algo contraproducente ya que se pierde la esencia de visitar una galería de arte, salir a comprar un libro, asistir a una lectura de poesía o un concierto de música.

Por ello **Shacu** añade un poco más de complejidad para compartir cultura, ya sea consumiéndola u ofreciéndola se debe desplazar físicamente a un punto de interés que estará marcado por **un código QR**.

Estos puntos de interés pueden ser temáticos o simples. Por lo que pueden estar colocados por fuera de una biblioteca y solo aceptar contenidos escritos. O estar en una parada de autobús y simplemente ser un punto de intercambio donde el transeúnte puede recibir o enviar un estímulo cultura.

1.2 Funcionamiento de la APP

Cómo ya se ha comentado en secciones anteriores Shacu pretende ser una red social para compartir contenidos artísticos en formatos digitales.

Primero los usuarios deberán iniciar sesión en la web app para poder utilizarla mediante su cuenta de Google.

Para acceder a un contenido compartido por otro usuario los usuarios deberán desplazarse físicamente a un punto de interés y acceder al contenido de ese punto a través de la lectura de un código QR situado en dicho punto.

Para desplazarse utilizaran el mapa desplegado en la aplicación que señalará los puntos y permitirá fijar una ruta a estos. Además de mostrar cierta información acerca de los puntos de interés y los contenidos desplegados en ellos.

El código QR tendrá asociado un contenido que pasado un tiempo definido, por ejemplo 48 horas, podrá ser reemplazado por otro usuario.

Los usuarios solo podrán tener un contenido publicado a la vez.

Cuando un usuario publica un contenido en un punto de interés señalado por un QR, deberá esperar a que sea validado. Una vez este proceso sea completado por un administrador, le llegará un mensaje a su perfil.

Cuando el usuario haya accedido a un contenido leyendo un QR con la aplicación. Podrá acceder a él y descargarlo siempre que quiera, desde la galería de contenido de la aplicación.

1.3 Antecedentes y estado del arte

Hay varias redes sociales que tienen como objetivo compartir cultura.

Instagram nació como una red social para compartir fotografías aún que haya ido transformándose con la utilización de esta por parte de los usuarios. Es una tendencia natural de adaptación al usuario de las redes sociales ya que el contenido de estas esta únicamente creado ellos.

Hay muchos más ejemplos de redes sociales para compartir cultura creadas con ese único fin. Centrémonos en estas a la hora de hacer el análisis de los precedentes.

Por otro lado, analizaremos aplicaciones que utilizan los códigos QR para referenciar elementos o elementos culturales.

1.3.1 Flickr

Es una red social para gestionar, etiquetar y compartir fotografías de los propios usuarios. Ya sea para compartir con otras personas o mantener organizadas tus fotografías es una de las opciones más punteras.

1.3.2 Printrest

Pinterest es una plataforma que permite a los usuarios crear y administrar, en tableros personales temáticos, colecciones de imágenes como eventos, intereses, aficiones y mucho más. Los usuarios pueden buscar otros pinboards, "repinchar" imágenes para sus propias colecciones. La misión de Pinterest es «conectar a todos en el mundo, a través de cosas que encuentran interesantes».

1.3.3 TikTok

TikTok es una plataforma de redes sociales se utiliza para hacer una variedad de videos de formato

corto, desde géneros como danza, comedia y educación, que tienen una duración de tres segundos a un minuto. Hoy en día es utilizado por muchos artistas para dar a conocer sus obras.

1.3.4 SoundCloud

SoundCloud es una red social para músicos, en la cual se les proporcionan canales para la distribución de su música. La idea es mostrar las canciones listas para ser escuchada. SoundCloud analiza la canción y su onda sonora, con el objetivo de que cualquiera que la esté escuchando pueda dejar su comentario en un momento determinado del audio.

1.3.5 QR [1] Codes for Cities & Cultures

Es una aplicación ofrecida por la empresa **QR Code generator**. Esta aplicación ofrece la posibilidad de ligar a la cultura de tu ciudad códigos QR que expliquen algo acerca de este lugar. Se utilizan para transmitir información cultural a modo de guía turística que permiten a través de códigos QR conocer la información cultural en más profundidad de un lugar o un elemento dentro de un espacio cultural.

1.3.6 Tales of things

Es una aplicación que permite crear un código QR para un objeto físico y añadirle recuerdos a este. Por ejemplo, añadir a mi coche a través de este código QR las aventuras que vivimos en las últimas vacaciones.

1.4 Herramientas utilizadas y control de versiones

A continuación, quedan enumeradas las herramientas utilizadas para la planificación y desarrollo de la aplicación:

1.4.1 Planificación

La planificación previa al proyecto fue el desarrollo de varios diagramas bajo el paradigma **4+1** todos fueron desarrollados bajo el mismo software. Además, se utilizó una herramienta de texto enriquecido de Google.

- **Visual Paradigm** [2]: v16.1
- **Google Docs**: versión (ofrecida en la nube)

1.4.2 Desarrollo

El desarrollo consta de tres partes separadas concretamente ya que fue desarrollado según el paradigma **MVC** (modelo vista controlador)

Para poder desplegar el entorno de la aplicación cliente servidor se ha creado una máquina virtual en Linux que da servicios web. Se han instalado los siguientes servicios en la máquina virtual para trabajar cómodamente:

- **Open SSH:** v8.2p1 Ubuntu-4ubuntu0.1
- **OpenSSL**: v1.1.1f 31 Mar 2020
- **Samba**: v4.11.6-Ubuntu

El entorno de desarrollo utilizado ha sido Visual code studio y Virtual Box para la máquina virtual.

- **Visual code studio**: v1.52.1
- **Virtual Box**: v6.1.12 r139181 (Qt5.6.2)

1.4.2.1 Modelo de datos

Desarrollado en **MySql** para Linux con SQL.

- **MySql**: v8.0.21-0ubuntu0.20.04.3 for Linux on x86_64 ((Ubuntu))

1.4.2.2 Back-end

Enfocado como conexión entre el front-end y el modelo de datos está desarrollado en **NodeJs** sobre el framework de **ExpressJs**. Además de una serie de módulos que facilitan la conexión a la base de datos y la gestión de usuarios, contraseñas, etc.

Además, se conectó a un sistema externo para la autenticación de los usuarios de tal forma que el inicio de sesión se delega con **Auth0** a **Google**. Los paquetes más importantes son:

- **NodeJS**: v10.19.0
- **Express**: v4.17.1
- **Passport**: v0.4.1
- **Passport-auth0**: v1.1.3
- **Express-mysql-session**: v2.1.4
- **Database-js-mysql**: v1.1.3

1.4.2.3 Front-end

El front end es el punto final de la aplicación al que los usuarios acceden y que contiene las vistas donde se muestran los datos del modelo obtenidos a través del controlador. Está programado en angular y utiliza algunos paquetes para formatear objetos de tiempo y para hacer las conexiones en segundo plano.

Contiene muchas líneas de código ya que engloba las vistas web de la aplicación que tienen **HTML** y **CSS**. Cabe destacar que angular es un framework que compila el código final en JS simple **HTML** y **CSS**, aunque la programación dentro de este es en **Typescrypt**.

Los paquetes más importantes utilizados en el front end son:

- **@Angular/core**: v10.0.3
- **@Angular/materials**: v10.2.4 (Para botones, formularios y otros elementos)
- **@agm/core**: v1.1.0 (Para la gestión de GoogleMaps)
- **@typescript**: v3.9.5
- **@zxing/ngx-scanne**: v3.0.1 (Para la lectura de los códigos QR)

1.4.3 Despliegue

El despliegue se ha hecho mediante la plataforma de **Heroku**. Aunque lamentablemente debido algunas limitaciones se ha tenido que desplegar en dos aplicaciones diferentes una para el **front-end** y otra para el **back-end.**

PONER AQUÍ LOS ENLACES DE LAS REFRECIAS.

1.4.4 Control de versiones

El control de versiones se ha hecho a través de **Git** utilizándolo como repositorio local y apuntando a repositorios remotos de **Github** y **Heroku**.

1.5 Enlaces de interés

A continuación, se exponen los enlaces que de interés de la aplicación.

Memoria [3]: La memoria subida en GitHub.

Frontend [4]: El código del frontend subido a GitHub.

Backend [5]: El código del backend desplegado en GitHub.

Despliegue de la App [6]: App desplegada en Heroku.

Capítulo 2
Planificación

Se quiere dar un enfoque bastante técnico a la aplicación, por lo tanto. Antes de desarrollarse se tiene que escoger algún tipo de metodología para ello. Se ha elegido para ello la metodología: Arquitectura 4+1.

2.1 Arquitectura 4+1 [3]

La arquitectura 4+1 consta de cuatro vistas que referencian a unos escenarios. Las vistas en cuestión son: La vista lógica, la vista de desarrollo, la vista de proceso y la vista física.

Para desarrollar dichas vistas primero se necesita una lista de requerimientos funcionales de la aplicación.

2.1.1 Requerimientos funcionales

Para elegir los requerimientos funcionales se eligen las acciones que va a realizar y se abstraen a una frase sencilla que describa dicha acción.

A continuación, se exponen los requisitos funcionales que se eligieron para el primer desarrollo y los que se pueden implementar en versiones futuras.

Para llevar a cabo las acciones necesitamos definir también los tipos de actores que intervendrán en ellos. Los actores en cuestión son: Administradores, usuarios, visitantes.

2.2.1.1 Requerimientos funcionales del desarrollo

Los requerimientos funcionales a desarrollar para este proyecto dentro del alcance del TFG son:

- **RF1 Entrar al sistema**: El sistema debe permitir a dos tipos de actores la entrada: usuarios y administradores.
- **RF2 Salir del sistema**: El sistema debe permitir a dos tipos de actores la salida: usuarios y administradores.
- **RF3 Gestionar contenido**: El sistema debe permitir a los actores del tipo administrador hacer alta, baja y modificación de los contenidos de la aplicación.
- **RF4 Localizar contenido en mapa**: El sistema debe permitir a los actores de tipo usuario poder localizar en un mapa los contenidos desplegados en la "url" de un código QR situado en un punto físico.
- **RF5 Verificar QR**: El sistema debe permitir a los actores de tipo usuario poder verificar a través de su dispositivo que el código QR leído pertenece al conjunto de los de la aplicación y además comprobar que el usuario está físicamente cerca del código QR.
- **RF6 Aprobar contenido**: El sistema debe permitir a los agentes de tipo Administrador aprobar los contenidos subidos por los usuarios a la plataforma.
- **RF7 Subir contenido**: El sistema debe permitir a los agentes de tipo usuario subir contenidos refrenándolos al QR que estén escaneando, si el contenido previamente

desplegado en el QR está caducado.

- **RF8 Descargar contenido**: El sistema debe permitir a los agentes de tipo usuario descargar el contenido asociado a un QR cuando se desplacen físicamente hasta él y hagan la lectura de este.
- **RF9 Registrarse**: El sistema debe permitir a los actores de tipo visitantes registrarse en el sistema.

2.2.1.2 Requerimientos funcionales futuros

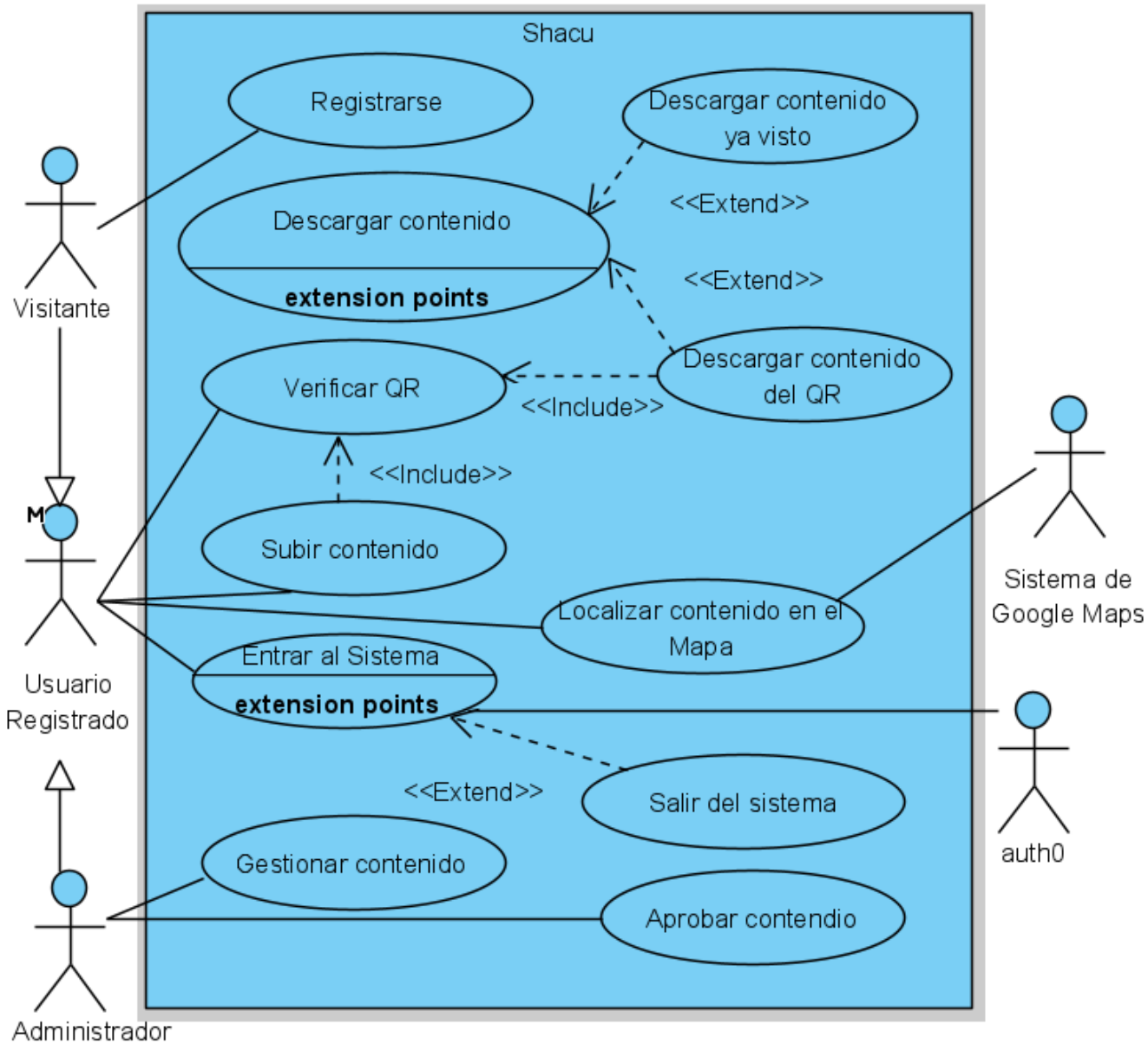
Vamos a definir unos requerimientos funcionales que podrían ponerse en un futuro para mejorar la experiencia de usuario en la plataforma. Los requerimientos en cuestión son:

- **RF10 Comentario contenido**: El sistema debe permitir a los actores de tipo usuarios comentar en los contenidos que ya hayan desbloqueado al llegar hasta los QR asociados a ellos físicamente.
- **RF11 Gestionar comentarios**: El sistema debe permitir a los actores de tipo administrador, poder hacer la baja o modificación de los comentarios en caso de que sea necesario.
- **RF12 Buscar contenidos**: El sistema debe permitir a los actores de tipo usuario filtrar los contenidos que aparecen en el mapa por su tipo.
- **RF13 Buscar usuarios**: El sistema debe permitir a los actores de tipo Usuario buscar usuarios en la plataforma por su nombre para tenerlos en una lista de "amigos".
- **RF14 Agregar amigos**: El sistema debe permitir a los actores de tipo Usuario agregar a otros usuarios a su lista de amistad.
- **RF15 Enviar mensajes**: El sistema debe permitir a los actores de tipo Usuario y Administrador enviar mensajes a otros usuarios.
- **RF16 Borrar mensajes**: El sistema debe permitir a los actores de tipo Usuario y administrador borrar mensajes.

2.1.2 Escenarios o casos de uso

Los escenarios describen secuencias de interacciones entre objetos, y entre procesos. Se utilizan para identificar y validar el diseño de arquitectura. Esta vista es también conocida como vista de casos de uso. Hemos extendido los requerimientos funcionales como casos de uso.

1 Escenarios de casos de uso



2.1.3 Vista lógica

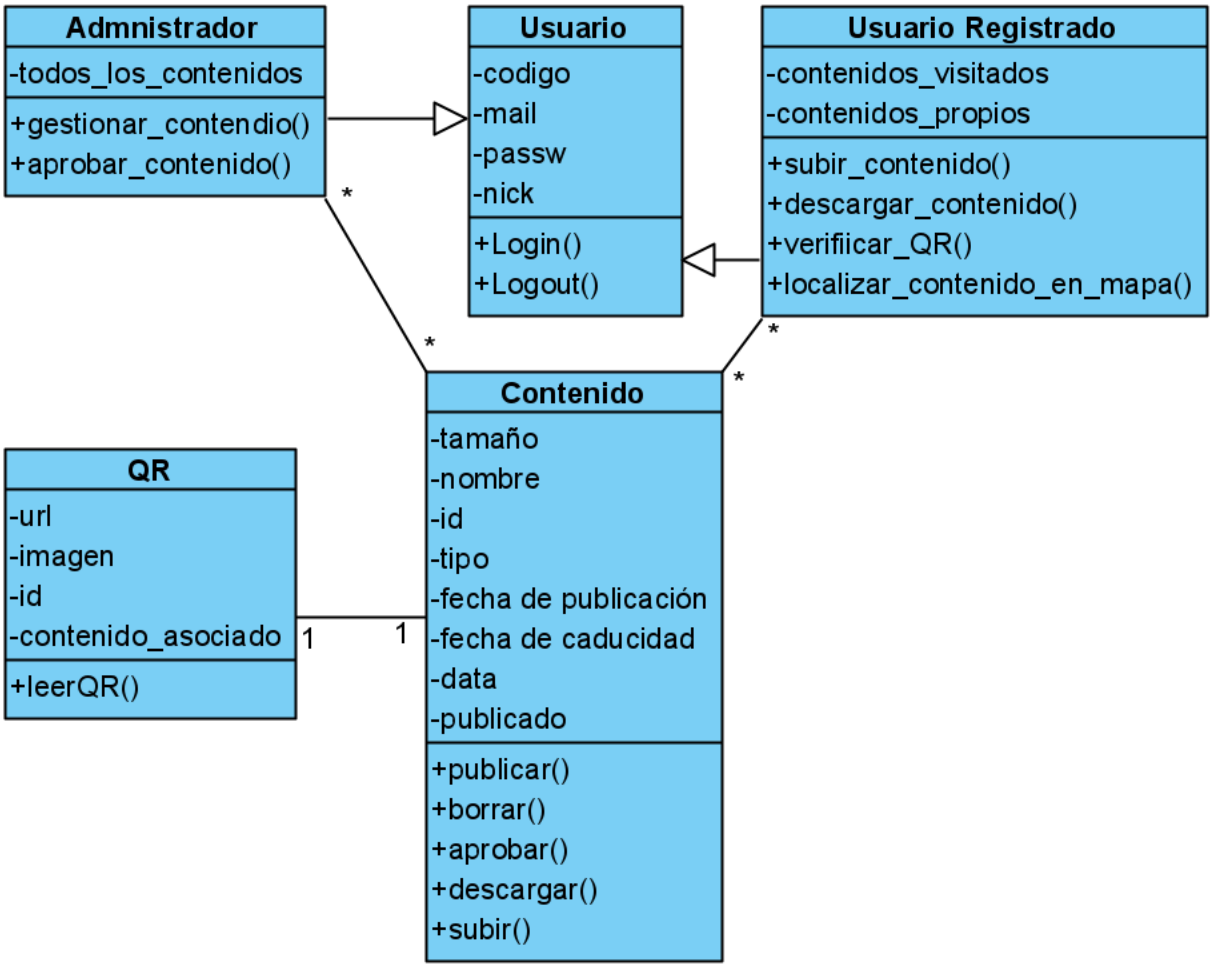
La vista lógica está enfocada en describir la estructura y funcionalidad del sistema. Los diagramas UML [3] que se utilizan para representar esta vista son los diagramas de clase, diagramas de comunicación y diagrama de secuencia.

2.1.3.1 Diagrama de clases

El diagrama de clases es un tipo de diagrama de estructura estática que describe la estructura de un sistema mostrando las clases del sistema, sus atributos, operaciones (o métodos), y las relaciones entre los objetos.

En nuestro caso es simplemente referencial ya que tal y cómo es estructura el sistema no se pueden definir las clases tal cual están descritas en el diagrama.

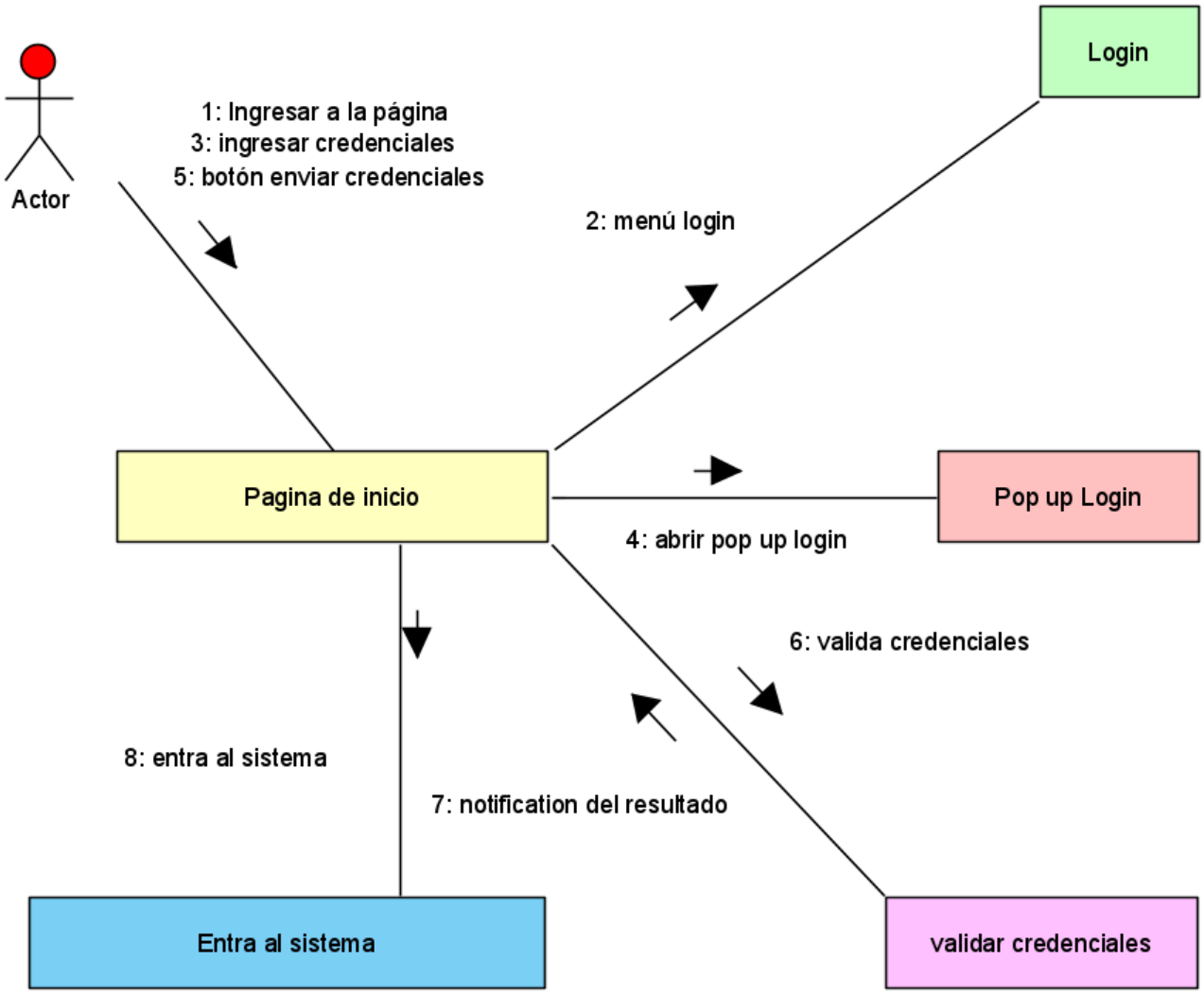
2.2.1 Diagrama de clases



2.1.3.2 Diagrama de Comunicación

El diagrama de comunicación modela las interacciones entre objetos o partes en términos de mensajes en secuencia. En nuestro caso debido al tiempo limitado solo hemos creado el diagrama de la interacción de entrar al sistema.

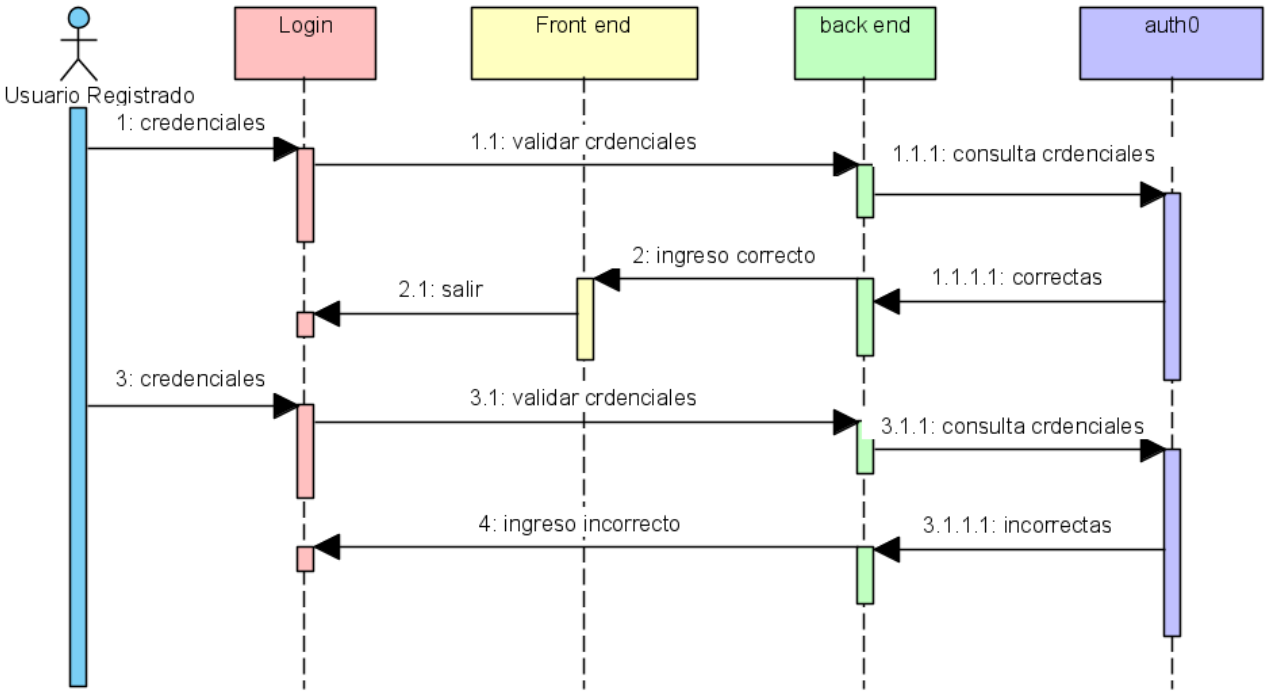
2.2.2 Diagrama de comunicación



2.1.3.3 Diagrama de secuencia

El diagrama de secuencia es usado para modelar la interacción entre objetos en un sistema. En nuestro caso vamos a implementar el de la interacción de entrar al sistema.

2.2.3 Diagrama de secuencia



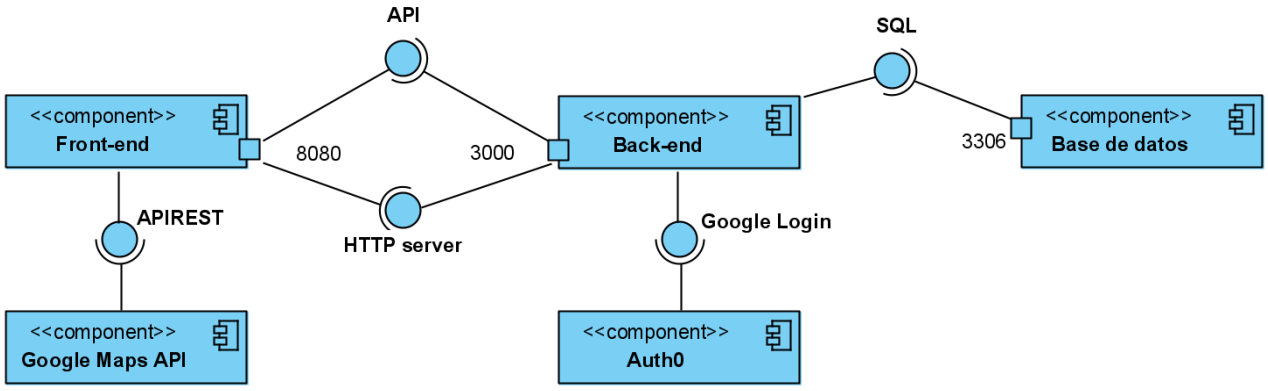
2.1.4 Vista de desarrollo

La vista de desarrollo ilustra el sistema de la perspectiva del programador y está enfocado en la administración de los artefactos de software. Esta vista también se conoce como vista de implementación. Utiliza el Diagrama de Componentes para describir los componentes del sistema y Diagrama de Paquetes.

2.1.4.1 Diagrama de componentes

El diagrama de componentes representa cómo un sistema de software es dividido en componentes y muestra las dependencias entre estos componentes. En nuestro caso vamos a simplificarlo a los componentes en una abstracción bastante alta.

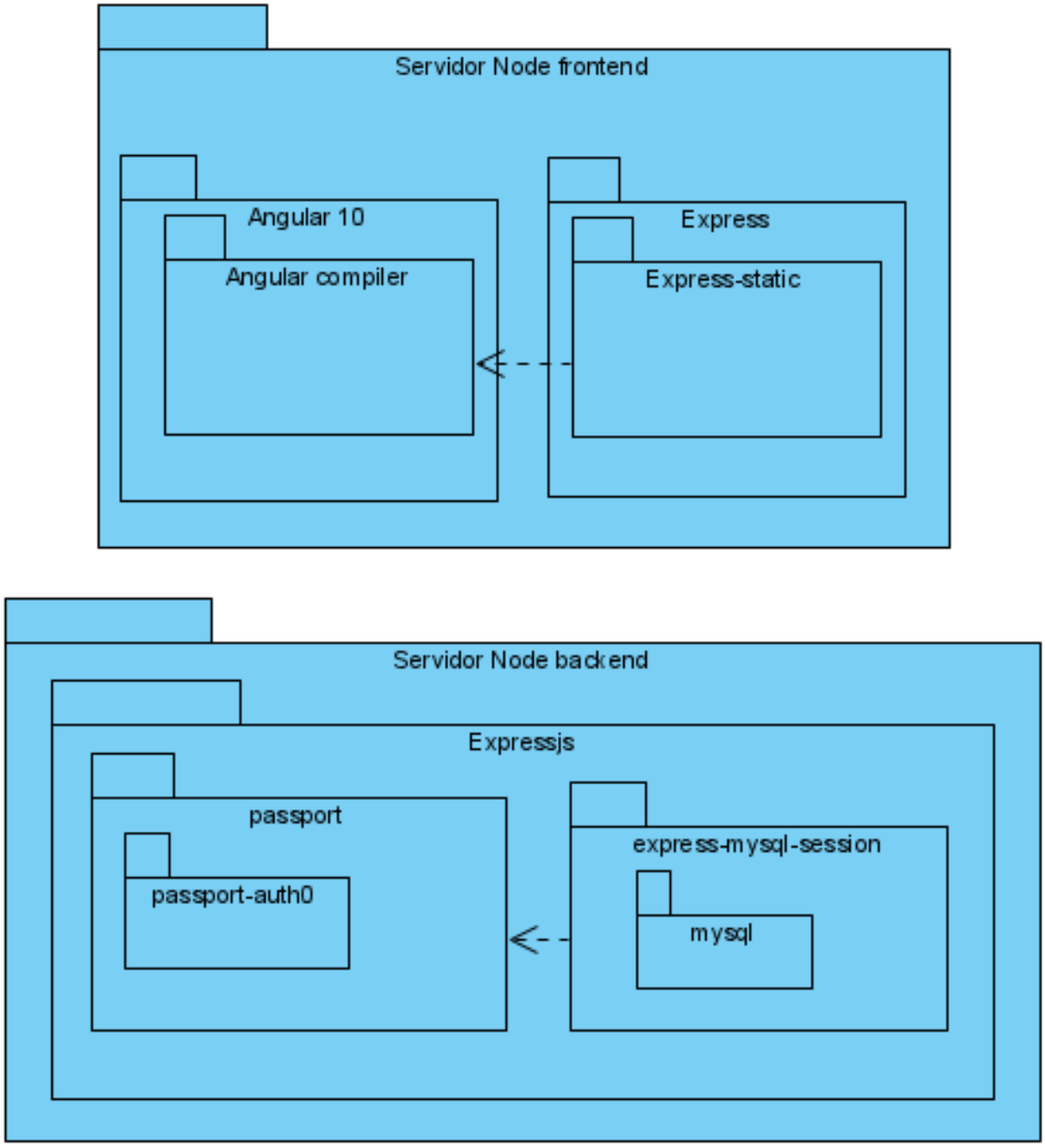
2.2.4 Diagrama de componentes



2.1.4.2 Diagrama de paquetes

El diagrama de paquetes representa las dependencias entre los paquetes que componen un modelo. Es decir, muestra cómo un sistema está dividido en agrupaciones lógicas y las dependencias entre esas agrupaciones. En nuestro caso vamos hacerlo de los principales paquetes de la aplicación.

2.2.5 Diagrama de paquetes



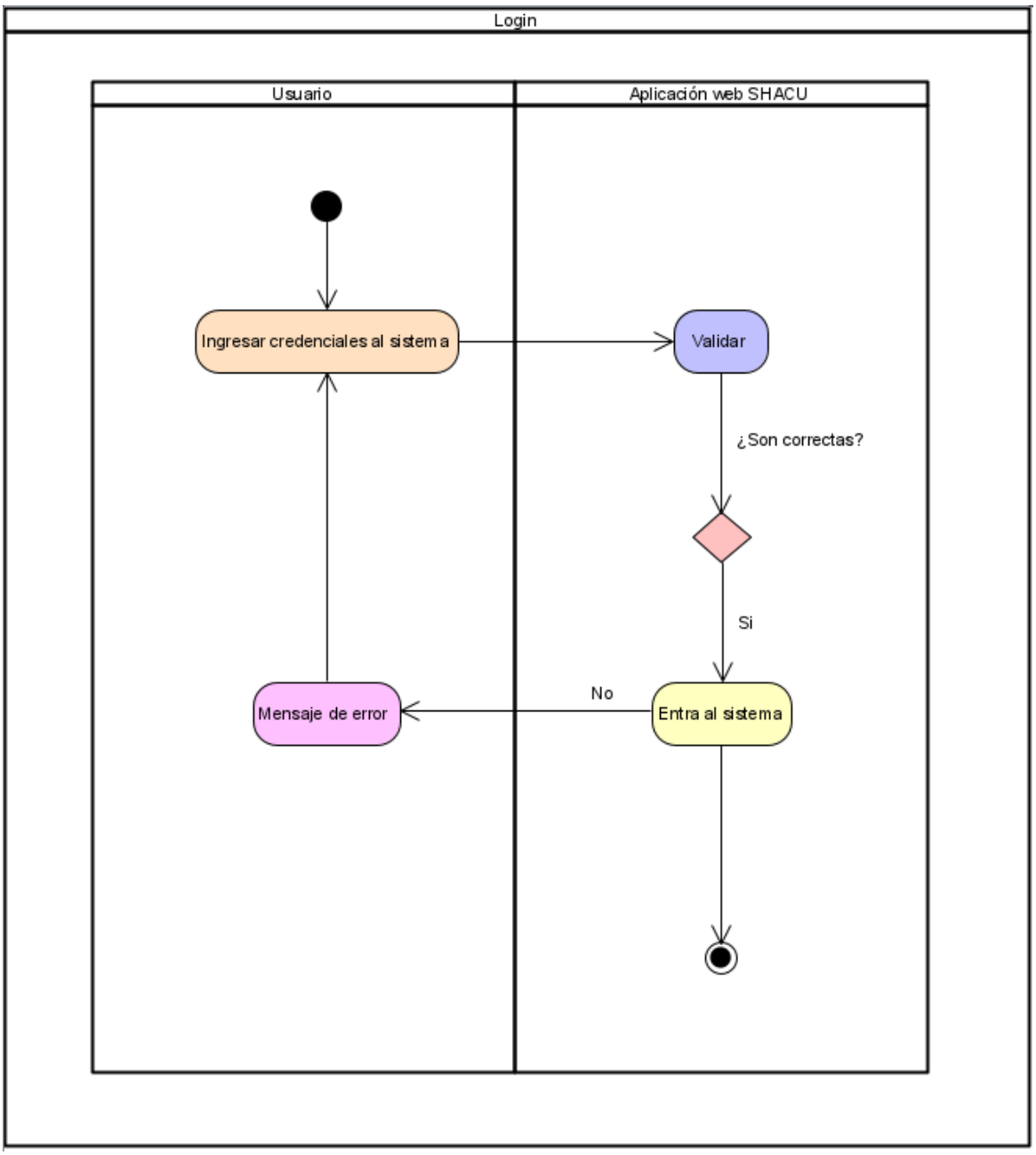
2.1.5 Vista de proceso

La vista de proceso trata los aspectos dinámicos del sistema, explica los procesos del sistema y cómo se comunican.

2.1.5.1 Diagrama de actividad

El diagrama de proceso es la representación gráfica de un algoritmo o proceso. Se ha desarrollado el diagrama del “login”.

2.2.6 Diagrama de actividad: "login"



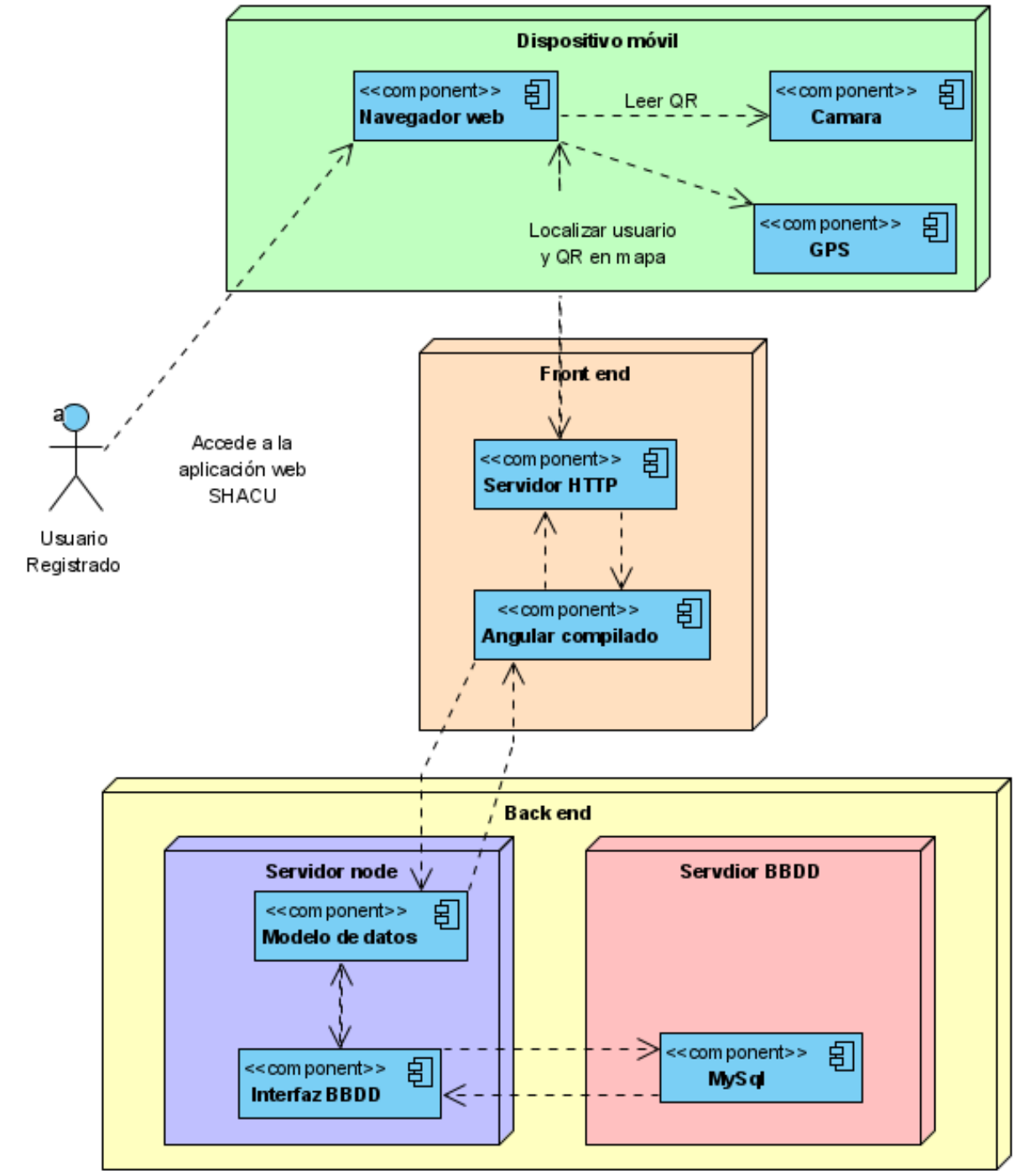
2.1.6 Vista física

La vista física describe el sistema desde el punto de vista de un ingeniero de sistemas. Está relacionada con la topología de componentes de software en la capa física, así como las conexiones físicas entre estos componentes.

2.1.6.1 Diagrama de despliegue

El Diagrama de Despliegue se utiliza para modelar la disposición física de los artefactos software en nodos (usualmente plataforma de hardware).

2.7 Diagrama de despliegue



Capítulo 3
Desarrollo

En el capítulo uno se definió el comportamiento que tiene la aplicación, así como la arquitectura de este, Ahora pasaremos a ver en detalle el desarrollo de los servicios creados dentro de la arquitectura para dar solución a los requisitos funcionales de la aplicación.

3.1 Introducción a la arquitectura MVC

Para el desarrollo del proyecto se ha elegido la arquitectura modelo-vista-controlador. Dicha arquitectura cuenta de tres partes que detallaremos a continuación y posteriormente iremos definiéndolas mediante el desarrollo que se realice de estas.

3.1.1 El modelo de datos

El modelo de datos contiene los datos en bruto de la aplicación. Generalmente es una base de datos que contiene los datos organizados y accesibles mediante la interfaz adecuada para dicha base de datos.

Por ejemplo, una tabla de usuarios en una base de datos MySql accesible mediante el Data Manipulation Languaje (DML) dentro del SQL.

3.1.2 La vista

La vista es punto final para el usuario en la arquitectura MVC. La vista será la interfaz que el usuario utilice para comunicarse con la aplicación. La vista será la entrada y salida de datos que se tomará desde y hacia el usuario.

Por ejemplo, en una web app, una página web en el navegador que permite al usuario visualizar y enviar datos. podría ser la vista en un MVC.

3.1.3 El controlador

El controlador son dos interfaces que permiten comunicarse a la vista y el modelo de datos de manera segura y ordenada. Una sirve para enviar datos desde la visita al modelo y la otra para enviar datos desde el modelo a la vista.

Por ejemplo, en una web app el back-end que permitiría hacer “login” de un usuario en concreto y ver los contenidos de dicho usuario en concreto podría ser controlador en un MVC.

3.2 Distribución de los servicios

Para cubrir las necesidades de la aplicación se tienen que distribuir las funcionalidades en diferentes servicios que están instaurados en las partes de la arquitectura MVC.

3.2.1 Modelo de datos

En este desarrollo vamos a guardar la mayor parte de los datos en una base de datos SQL. Aunque debido al problema detallado en la sección correspondiente se van a utilizar datos de servicios externos.

3.2.1.1 Base de datos

El modelo de datos principal va ser una base de datos MySql que se instaurará a priori en el mismo servidor que el servicio de back-end tal y cómo se ha detallado en el diagrama de despliegue.

En dicha base de datos se generarán las tablas necesarias mediante un análisis entidad relación y aplicando el modelo relacional [5] generamos las tablas necesarias para almacenar todos los datos de la aplicación excepto los usuarios.

3.2.1.2 Auth0

Auth0 es un servicio externo a la aplicación que utiliza una autenticación basada en riesgo [6]

que permite la autorización de forma segura y simple de una API. En nuestro caso utilizaremos el login de google para iniciar sesión en nuestra aplicación mediante una cookie de autorización en las cabeceras HTTP.

En el modelo de datos esta parte permitirá extraer algunos de los datos del perfil del usuario.

3.2.1.3 API Googlemaps

Esta API permite descargar los mapas y utilizarlos en un recurso externo a Google permitiendo a los desarrolladores tener su propio mapa dentro de la aplicación y manipular los elementos de este a través de la librería y la API.

En esta parte utilizaremos la API de mapas y la de rutas para poder manipular el mapa dentro de la aplicación añadiendo nuestros puntos de interés y crear rutas entre la posición del usuario y dicho punto

3.2.2 Controlador de datos

El controlador de datos será un servidor web que hará de back-end de forma que sirva y reciba datos a la vista que se detalla posteriormente. Para implementarlo utilizaremos diferentes servicios dentro de este.

3.2.2.1 Auth

Se encargará de hacer el “login” y “logout” con Google mediante Auth0. Se implementará como una serie de rutas HTTP que enviarán y recibirán información hacia el controlador, cuando se visite la url adecuada en la vista.

3.2.2.2 Download

Se encargará de enviar datos hacia el front-end a través de rutas HTTP cuando se le envíen los parámetros adecuados.

3.2.2.3 Upload

Se encargará de recibir datos desde el front-end y guardarlos en consonancia a los parámetros adecuados en el modelo de datos.

3.2.2.4 Middlewares

Se encargarán de interceptar las peticiones HTTP y ponerles algunos parámetros específicos para controlar ciertos problemas descritos en la sección correspondiente. En concreto añadirán parámetros para asegurar las rutas al back-end mediante el login y permitir las llamadas de diferentes dominios CORS.

3.2.2.5 Interfaces

Se utilizan interfaces para modificar el comportamiento en la librería que utilizamos para el servidor web dentro de NodeJs, ExpressJs. Dichas interfaces modifican el comportamiento de los usuarios creándose mediante Auth0 y sus sesiones, guardándolas en la base de datos del modelo para poder evitar el problema de la volatilidad de estas.

3.2.3 Vistas de la aplicación

Las vistas van a elaborarse con Angular que es un framework en TypeScript que nos permitirá organizar nuestro código de una forma mucho más ordenada. Ya que JavaScript suele tender a crear estructuras muy dispersas y que llevan a la confusión cuando el código crece un poco.

Una aplicación en Angular estará estructurada de tal forma que tendremos varias herramientas que cumplirán diferentes funciones y que todas juntas harán funcionar la aplicación. Estas herramientas se detallan a continuación.

3.2.3.1 Módulos

Los módulos sirven para estructurar nuestra aplicación en diferentes partes que tendrán diferentes funciones dentro de esta.

Los módulos tienen asociada una vista que es una plantilla dinámica que cambiará según lo que se especifique en su código.

Un módulo puede estar compuesto por diferentes componentes o por componentes y sub-módulos, pero la parte de visualización en la pantalla correspondiente, aunque sea única debe estar definida en un componente.

Se va a desarrollar un módulo por cada vista que necesite el usuario:

- **App-module:** Módulo principal padre de los demás.
- **App-material**: Aagrega todas las dependencias de Angular-material a cualquier otro módulo.

- **Login**: Gestiona el inicio de sesión del usuario.
- **Content**: Gestiona el contenido de los usuarios.
- **Home-web**: Página de información de la aplicación.
- **Maps**: Gestiona la búsqueda y de puntos de interés en el mapa su información y generará rutas a estos.
- **Messagges**: Gestiona los mensajes de los usuarios.
- **Profile**: Permite diferenciar usuarios de administradores para poder acceder a los módulos de contenido y QR.
- **QRCode**: Gestiona los QR ya sea por un administrador para su alta, baja o modificación o un usuario para su lectura.
- **App-routng-module**: Permite cambiar el un submódulo mostrado en modulo padre mediante rutas en la barra de navegación. También se pueden crear submódulos de enrutamiento en los módulos para hacer rutas con otro nivel de profundidad.

3.3.2.2 Animaciones

Las animaciones se pueden definir como funciones externas a los módulos de tal forma que podrán llamarse en diferentes módulos con diferentes parámetros cuando se necesite. Las animaciones hacen que elementos HTML pasen de ser estáticos a dinámicos con un comportamiento que se defina dentro de esta.

3.3.2.3 Componentes

Los componentes ocupan una parte de la vista en la pantalla, en este desarrollo solo utilizaremos un componente externo a los propios módulos que se utilizará en el principal. De tal forma que este estará formado por dicho componente, hace la función de cabecera, y contiene un submódulo que irá cambiando entre los definidos anteriormente a través de las rutas declaradas en las rutas.

Por otro lado, cada módulo enrutado como submódulo desde el principal tendrá un componente propio que representará todo el espacio en la vista que estará asociada a dicho componente.

3.2.2.4 Directivas

Las directivas permiten extender las funcionalidades del HTML usando para ello una nueva sintaxis dentro de este.

Cada Directiva que usamos tiene un nombre, y determina donde puede ser usada, sea en un elemento, atributo, componente o clase.

Se desarrollará una directiva para aplicar una animación a ciertos elementos HTML.

3.2.2.5 “Helpers”

Los “helpers” son unos interceptores que pueden llamarse a la hora de enrutar hacia otro módulo comprobando que se cumplen ciertas características antes de enrutar hacia él.

Se desarrollarán varios “helpers” para el funcionamiento de la web app.

- **Admin**: Se utilizará para proteger el enrutado entre módulos aquellas partes a las que solo

pueda acceder un administrador de la app, cómo la gestión de los QR, o la aprobación de contenidos.

- **Auth**: Se utilizará para proteger el enrutado de los módulos que puedan ser accedidos cuando se inicie sesión en la web app.
- **Qr-proximity**: Se utilizará para proteger las rutas codificadas en los QR de la web app. De tal forma que solo se podrá leer este si se está lo suficientemente cerca de él. Evitando que puedan replicarse y leerse desde otro punto.

3.2.2.6 Modelos

Los modelos sirven para crear una clase, de programación orientada a objetos, que defina un tipo de dato dentro de TypeScript para utilizarlo a lo largo de la aplicación.

Se desarrollará un modelo para la clase usuario que guardará la información que se obtenga de este desde el login de Google aparte de alguna información más.

3.2.2.7 Servicios

Los servicios son clases de tipo “singelton” que se utilizan para mantener el flujo de datos dentro de la aplicación.

Los servicios se inyectan en los diferentes módulos y son una forma eficaz de compartir funcionalidades e información a través de los diferentes módulos de la aplicación.

Se desarrollarán diferentes servicios para la web app:

- **Download**: Este servicio contendrá todas las llamadas HTTP al back-end que estén en el servicio definido con el mismo nombre dentro de este. Darán servicio para descargar datos del modelo de una forma adecuada a través de diferentes parámetros que se insertarán en las llamadas HTTP de tipo GET.
- **Upload**: Este servicio contendrá todas las llamadas HTTP al back-end que estén en el servicio definido con el mismo nombre dentro de este. Darán servicio para subir datos del modelo de una forma adecuada a través de diferentes parámetros que se insertarán en las llamadas HTTP de tipo POST.
- **Login**: Este servicio controlara el “login” y “logout” del usuario además de guardar ciertos parámetros como una copia del usuario conectado definido con la clase Usuario.
- **Workflow**: Este servicio será el encargado de gestionar el flujo de datos de la aplicación cuando se esté leyendo un QR.

  En este servicio estará la conexión al servicio externo de Googlemaps. iniciando la librería, y posicionando el dispositivo a través del GPS. Además de calcular los parámetros necesarios para saber la proximidad del usuario a un QR.

3.3 Desarrollo

En esta sección se detallarán las funcionalidades desarrolladas en cada uno de los servicios descritos con anterioridad de tal forma que dispondremos qué funciones se han delegado a ellos exactamente y cómo se han codificado de forma abstracta dichas funciones.

Se dividirán en dos grupos. Los servicios del front-end y los del back-end.

3.3.1 Base de datos

La base de datos ha sido generada mediante un modelo entidad relación se han generado las siguientes tablas:

3.3.1.1 Users

Es la tabla que guarda la información de los usuarios, tiene una relación con la tabla contenido del tipo muchos a muchos de tal manera que un usuario puede tener varios contenidos disponibles y un contenido puede estar disponible para varios usuarios.

Sus campos son:

- **Id**: identificador único generado incrementalmente de forma automática.
- **Fisrtname**: Guarda ambos apellidos el nombre viene del registro obtenido de Google.
- **Email**: Se utiliza para emparejar el usuario con el usuario de Google es único
- **Nick**: campo sacado del usuario de Google
- **Is_admin**: Booleano que marca los usuarios administradores.
- **Reg_date**: Se autogenera con la fecha y hora de creación / actualización del registro.

3.3.1.2 QR

Esta tabla representa la entidad del QR. Tiene una relación de uno a uno con la tabla contenido ya que un contenido está asociado a un QR y un QR está asociado a un solo contenido.

Sus campos son:

- **Id**: identificador único generado incrementalmente de forma automática.
- **Lat**: Latitud en el mapa de la posición del QR.
- **Lon**: Longitud en el mapa de la posición del QR.
- **Qr_name**: Nombre representativo el punto de interés.
- **Qr_image**: imagen del QR en base64.
- **Published**: Se utiliza para publicar o despublicar el QR de tal forma que sea o no accesible desde la aplicación.
- **Reg_date**: Se autogenera con la fecha y hora de creación / actualización del registro.

3.3.1.3 Content

Esta tabla representa los contenidos en la base de datos. Tiene una relación de muchos a muchos con la tabla usuarios y otra relación de uno a muchos con la misma tabla que representa el creador del contenido. Un usuario crea varios contenidos un contenido es creado por un solo usuario. Un contenido es accedido por varios usuarios un usuario accede a varios contenidos. Por otro lado, dos relaciones con la tabla QR. Las relaciones son de tipo de uno a uno. Por un lado, un contenido puede estar encolado en un QR y un QR solo puede tener un contenido en la cola. Por otro lado, un QR tiene contenido publicado y un contenido solo está publicado en un QR.

Cómo vemos las colas son unitarias si se intenta encolar contenido en un QR que ya tiene algo encolado la aplicación nos avisará de ello.

Sus campos son:

- **Id**: identificador único generado incrementalmente de forma automática.
- **Qr_queue:** QR en el que está encolado dicho contenido, False si no está en cola.
- **Queue_date:** Fecha en que el contenido se ha puesto en cola.
- **Qr_id**: Id del QR en el que está publicado. False si no está publicado en ninguno.
- **User_id**: Creador de dicho contenido en la plataforma.
- **Size**: Tamaño en bytes del contenido.
- **File_path**: Ruta en el servidor al contenido.
- **Content_name**: Nombre del contenido dado por el creador.
- **Content_type**: Tipo de contenido subido. Audio, video, texto…
- **Authorized**: Booleano que indica si el contenido está o no autorizado por el administrador.
- **Reg_date**: Se autogenera con la fecha y hora de creación / actualización del registro.

3.3.1.4 Content_user

Esta tabla resuelve la relación de muchos a muchos entre contenidos y usuarios en el modelo relacional. Contiene dos claves foráneas que son el id del QR y el id del usuario. Ambas claves juntas forman la clave primaria de la tabla.

Se utiliza para ligar los contenidos a los usuarios y hacerlos accesibles en su biblioteca de contenidos leídos en la web app.

- **User_id**: Id del usuario.
- **Content_id**: id del contenido.

3.3.1.5 Messages

Messages es la tabla que contiene los mensajes de los usuarios. Tiene una relación de uno a muchos con la tabla usuarios. Un mensaje pertenece a un usuario, un usuario puede tener varios mensajes. Además, los mensajes pueden estar ligados a un contenido cuando se utilizan al aprobar o rechazar el contenido e informar al usuario con un mensaje. Con una relación de uno a muchos. Un mensaje referencia a un contenido y un contenido puede ser referenciado por varios mensajes.

Sus campos son los siguientes:

- **Id**: Identificador único generado incrementalmente de forma automática.
- **Content_id**: Id del contenido referenciado. False si no referencia ningún contenido.
- **User_id**: Usuario al que pertenece el mensaje.
- **Message_user**: Cuerpo del mensaje.
- **Readed**: Booleano que indica si el mensaje está marcado cómo leído.
- **Message_type**: Cabecera del mensaje. Por ejemplo, Error.
- **Reg_date**: Se autogenera con la fecha y hora de creación / actualización del registro.

3.3.2 Back-end

Es la parte del sistema que el usuario no ve y que se comunica con la base de datos y el front-end. De tal forma que los usuarios no tendrán percepción de él.

El back-end se programará sobre un servidor web NodeJs y todo se desarrollará además sobre la librería exprés que nos permitirá tener una estructura dinámica ya accesible a la hora de programarlo.

Ahora se pasará a detallar los servicios de este expuestos en la anterior sección.

3.3.2.1 Rutas

Las rutas nos sirven para definir un punto de acceso HTTP a nuestro servidor web las dividimos en tres partes:

Auth

Consta de cuatro rutas y una funcionalidad principal:

- **La ruta de "login"**: Permite al usuario acceder al "login" de la aplicación. Se le añade el middleware de Passport que permite acceder a auth0 para hacer la autenticación.
- **La ruta de "logout"**: Destruye la sesión tanto en Auth0 como en nuestro back-end y nos redirige a una ruta del front-end. Dicha ruta permitirá destruir los datos también en este.
- **La ruta de callback**: es la ruta a la que se redirige auth0 para enviarte la información del usuario que ha iniciado sesión en su cuenta de Google.
- **La ruta itsAdmin**: esta ruta comprueba si el identificador de un usuario dado pertenece a un usuario que esté marcado como administrador en la aplicación.
- **La funcionalidad createUserAndMessage**: Esta funcionalidad comprueba si es la primera vez que el usuario hace login en la aplicación. Si es así lo registra en la base de datos y le envía un mensaje de bienvenida a su bandeja de entrada.

Download

Consta de cinco rutas principales todas bajo el middleware de headers. Cómo ya detallaremos este middleware añadirá una cabecera que identificará al usuario con un espacio de sesiones en el sistema en el que tendrá únicamente sus datos de usuario accesibles.

- **Ruta getModelData**: Permite obtener contenido de cualquier tabla con unas condiciones limitadas, siempre comprobando los requisitos para llamar a dicha tabla, que el usuario esté logueado, o que el usuario sea administrador.

- **Ruta havePublishedContent**: Permite comprobar si un usuario ya tiene un contenido publicado en este momento.
- **Ruta downloadContentQR**: Permite descargar el contenido para un QR en particular. Comprobando si el usuario tiene permisos para ello.
- **Ruta downloadUserMessages**: Descarga los mensajes para el usuario que mantiene la sesión que lo llama.
- **Ruta downloadContentUser**: Descarga los contenidos para el usuario que tiene la sesión activa. O todos para el administrador.

Upload

Consta de siete rutas principales todas bajo el middleware de headers. Además, se controla en ellas el usuario que está accediendo para comprobar si tiene permisos para realizar dicha acción en la base de datos.

- **Ruta insertMessage**: En esta ruta se envían los mensajes para insertarlos en la bandeja de entrada de un usuario desde el front-end.
- **Ruta removeOldQR**: Esta ruta elimina el QR asociado a un contenido cambiando el id en la base de datos del contenido asociado a este.
- **Ruta uploadQR**: Esta ruta permite a los administradores subir desde el front-end un nuevo punto de interés QR y asociar un contenido de la tabla de contenidos.
- **Ruta insertContentUser**: Esta ruta permite subir un contenido asociándolo a un punto QR de tal forma que se tenga que aprobar por un administrador.
- **Ruta update**: Esta ruta permite actualizar registros de la base de datos a los administradores.
- **Ruta delete**: Esta ruta permite a los administradores eliminar ciertos registros y a los usuarios eliminar mensajes.
- **Ruta exist**: Esta ruta permite comprobar si ciertos registros ya existen en la base de datos como un usuario.

3.1.1.2 Middlewares

Los middlewares se pasan como parámetros en la definición de las rutas y sirven para manipular dichos parámetros antes de la llegada a su punto final. Los utilizamos en nuestro caso para las cabeceras de las respuestas enviadas al back-end. Y para comprobar que está logueado para acceder a ciertas rutas.

Headers

Resuelve el problema de CORS y añade la necesidad de unas credenciales en la cabecera HTTP para acceder a una ruta que lo implemente.

Secured

Comprueba si las credenciales de la cabecera de la petición pertenecen a un usuario.

3.3.2.2 Interfaces

Ya detallamos el uso de las interfaces dentro de ExpressJs. Ahora se expondrán cómo se han codificado en detalle.

Passport

Passport es la Liberia que utilizamos para hacer inicio de sesión de un usuario en ExpressJs.

Contiene una sola funcionalidad que añade los parámetros de Auth0: dominio, decreto id del cliente y la url de callback. Que se utilizan para hacer el "login" con Auth0. Después añadimos unas funcionalidades que guardarán y obtendrán el usuario en ExpressJs en la sesión actual.

Session

Session es la librería de Express que se encarga de gestionar las sesiones de usuario en él. Para solucionar el problema de las sesiones volátiles creamos esta interfaz.

Contiene una sola funcionalidad que modifica cómo se guardan los usuarios en el sistema cambiando el hecho de que se guarden en la memoria del proceso para hacer que pasen a guardarse en la base de datos MySQL que ya utilizamos cómo modelo de datos.

3.3.2.3 Otros

Hay otras dos herramientas utilizadas en el back-end a tener en cuenta. Ya que son análogas al funcionamiento correcto del sistema.

La interfaz de la base de datos

Se ha generado un módulo dentro de NodeJs que hace de interfaz con la base de datos abre y cierra las conexiones a esta y tiene las funcionalidades necesarias para interactuar con ella. En su mayoría ejecutan sentencias SQL con diferentes parámetros.

El fichero .env

El fichero .env un fichero de entorno que carga unas variables globales accesibles desde cualquier punto de la aplicación del back-end. De tal forma que guardaremos ahí los parámetros de la conexión a la base de datos, a uth0 o la cuenta de administrador de la aplicación. Además de parámetros que indican si la aplicación está en desarrollo o producción para resolver el problema del despliegue en Heroku y en local.

3.3.3 Front-end

Es la parte del sistema que se ejecuta en el dispositivo del usuario. Está codificado en Angular cómo ya se ha nombrado. Angular nos permite crear una estructura diferente a HTML, CSS y JavaScript puro mediante su framework. Utilizaremos para ello TypeScript y SCSS y plantillas de HTML modificadas mediante directivas.

Angular compila estos elementos en HTML, CSS y JavaScript para llevarlos a producción. Así que a la hora de desplegarlo se hará mediante un servidor NodeJs que servirá estáticamente dichos archivos de la carpeta donde se compilan.

A continuación, se detallarán las partes más importantes del front end y sus funcionalidades principales así cómo el objetivo de estas dentro del mismo. Hay funcionalidades que quedaron suficientemente bien detalladas por su sencillez en el apartado de distribución de servicios y por lo tanto no las tomaremos en cuenta en este apartado.

3.3.3.1 Módulos

Los principales módulos de la web-app son:

App-module

Cómo modulo principal de la aplicación en su componente se muestra en esta en todo momento contiene el componente de la cabecera de la página que contiene el menú de usuario para navegar por la web app. Además, contiene un router-layout que permite ir insertando módulos mediante las rutas del navegador, de manera que al navegar entre rutas se cargarán todos los módulos de la aplicación en este. Aparte de esto no tiene funcionalidades destacables.

Login

Este módulo contiene un componente de login. La funcionalidad principal es comprobar si un usuario ha iniciado la sesión y cargar su información si así fuera. Contiene algunos métodos relevantes.

- **ngOnInit**: Este método se encarga de comprobar si el usuario ha iniciado sesión. Si lo ha hecho carga su información y sus mensajes para mostrarlos en el menú principal mandándoselo al servicio de login. Además, cabe destacar que este método se ejecuta al cargar el componente del módulo login.
- **Login**: redirige al back-end en la ruta de login.
- **Logout**: redirige al back-end a la ruta de logout.

Content

Este módulo gestiona todas las vistas relacionadas con el contenido por lo tanto contiene varios componentes para ello.

- **Contenet-router-modules**: Este no es un componente si no un submódulo que permitirá enrutar a un nivel de profundidad mayor en la ruta dentro de la ruta content. Se encargará de enrutar a alguno de los componentes que nombraremos a continuación.
- **Manage-content**: Este componente contiene una lista de contenidos que a través de unos botones en cada elemento de la lista permite acceder a la gestión del contenido para los administradores, y la visualización para los usuarios. Permite borrar, editar, autorizar o acceder a la información en detalles de un contenido. Sus métodos principales son:
    - **Delete_content**: Borra el contenido seleccionado.
    - **Open_content**: Abre el detalle del contenido.
    - **Authorize_content**: Permite autorizar el contenido en cola en un QR.
    - **Manage_content**: Permite editar el contenido.
- **New-contenet**: Es un formulario que permite crear un contenido nuevo y enviarlo a la base de datos. Este formulario se abre para los usuarios al leer un QR con el contenido ya caducado. Además, se utiliza para editar el contenido trayendo los datos del contenido que se quiera editar y rellenando con estos los campos del formulario. Sus principales funcionalidades son:
    - **getEditingData**: Esta funcionalidad trae los datos de la base de datos del contenido que quiere editarse y rellena los campos del formulario con los datos de este.
    - **uploadFile**: Envía todos los contenidos del formulario a través del servicio upload

a la base de datos. En estos campos irá el archivo binario del contenido en cuestión.

- **See-content**: Este componente permite visualizar en detalle los contenidos. Además, permite también realizar algunas acciones con este. Publicar contenido, editar contenido descargar contenido. Sus principales funcionalidades son:
    - **Publish**: Publica el contenido en el QR en el que está en cola. Solo administradores.
    - **Edit**: Enruta al componente New-content en modo edición.
    - **Download**: Descarga el archivo binario de contenido del back-end.

Home-web

Este módulo contiene un componente que es estático en su comportamiento sin funcionalidades significativas más allá de mostrar diferentes contenidos a través de pestañas.

Sirve para mostrar la información de la Aplicación así cómo la guía de usuario.

Maps

Este módulo contiene un componente que muestra el mapa en la pantalla con sus puntos de interés, además de una lista tipo galería que muestra los puntos de interés también. Esta lista permite ver en detalle un punto de interés y crear una ruta a este desde la ubicación actual. También si se está lo suficientemente cerca de un contenido se muestra un botón de acceso a la cámara que permite abrir una vista de la cámara para leer el código QR del punto de interés. También mostrará el icono de Subir nuevo contenido si estamos lo suficientemente cerca y el contenido de ese punto e interés está caducado, ha pasado un tiempo x desde que se publicó el contenido.

En el mapa se muestran los puntos de interés, la localización del usuario, con un círculo alrededor de esta que representa lo acertada que es esa ubicación. Además, se muestra también la ruta si se accede a esta desde la listad de puntos de interés.

En los detalles veremos cuánto tiempo lleva caducado el contenido o cuánto le falta para caducar.

Sus funcionalidades principales son:

- **CameraClick**: Enruta al componente de lectura de QR.
- **ClickedMarker**: Al pulsar en una marca del mapa vamos al elemento de la lista que representa este.
- **OpenDialog**: Abre el pop up de detalles del punto de interés.
- **SetPanel**: Cambia la lista de puntos de interés por las instrucciones al seleccionar un punto de interés como ruta.
- **ReturnList**: Oculta la ruta y muestra de nuevo la lista de puntos de interés.

Messages

Este módulo gestiona los mensajes mostrándonos dos bandejas de mensajes la de mensajes sin leer y la de los mensajes leídos. Permite borrar un mensaje, marcarlo cómo no leído o marcarlo como leído. También nos permitirá pulsar en un mensaje para visualizarlo en una ventana emergente.

Sus principales métodos son:

- **NewMessages**: Carga los mensajes no leídos.
- **OldMessages**: Carga los mensajes ya leídos.

- **OpenDialog**: Abre la ventana emergente del mensaje.
- **DeleteMessage**: Borra el mensaje en cuestión.
- **ReadMessage**: Marca el mensaje como leído o no leído dependiendo de su estado inicial.

Profile

Este módulo contiene un solo componente que simplemente Enruta con los parámetros debidamente adecuados, para distinguir un administrador de un usuario normal a la gestión de contenidos y puntos de interés. A parte de estos enrutamientos no contiene ninguna funcionalidad reseñable.

QRcode

Este módulo contiene varios componentes relacionados con los QR o puntos de interés. Estos componentes permiten la creación, gestión y lectura por cámara.

A continuación, se exponen en profundidad dichos componentes:

- **Contenet-router-modules**: Este no es un componente sino un submódulo que permitirá enrutar a un nivel de profundidad mayor en la ruta dentro de la ruta QR. Se encargará de enrutar a alguno de los componentes que nombraremos a continuación.
- **ManageQr**: Este componente es solo para administradores y permite gestionar los QR desde una lista de estos mismos. Permite borrarlos y modificarlos. Para modificarlos abrirá una ventana emergente de los datos de este. Permitirá modificar los datos en esta venta y una vez modificados permite guardar los cambios. Sus principales métodos son:
    - **Delete_**: Borra el QR en cuestión.
    - **OpenDialog**: Permite abrir la ventana emergente con los datos del QR.
    - **Save**: Guarda los cambios realizados en la ventana emergente
    - **ChangeState**: Permite publicar o despublicar un QR.
- **NewQr**: Este componente permite a los administradores crear un QR o punto de interés con un contenido asociado por defecto. Básicamente es un formulario en el que se meten los datos del QR y del contenido asociado por primera vez. Su principal funcionalidad es:
    - **UploadFile**: Envía a través del servicio upload la información al back—end para generar un nuevo punto de interés en la base de datos con todos los datos del formulario.

- **ReadQr**. Este componente permite acceder a la cámara y leer un QR identificando si este está en la base de datos asociado a un punto de interés. Solo podrá accederse a este componente si está lo suficientemente cerca de un punto de interés físico, marcado por sus coordenadas en el mapa. Para acceder a la cámara y hacer la lectura del QR se utiliza la librería @zxing/ngx-scanner. Sus principales funcionalidades son:
    - **OnCamerasFound**: Evento que mostrará un error si no se encuentran cámaras en el dispositivo.
    - **OnCodeResult**: Evento que se lanzará al leer un código QR con la cámara.
    - **Exist**: Comprueba si EL código QR leído existe en los puntos de interés de la base de datos.
    - **OnDeviceSelectChange**: Evento que se lanzará al cambiar la cámara del dispositivo. Por ejemplo, frontal y trasera.

- **onTorchCompatible**: Evento que se lanzará si el dispositivo es compatible con activar la luz del flash para darle la opción al usuario de hacerlo.

3.3.3.2 Servicios

Los servicios permiten compartir datos y funcionalidades a través de los diferentes componentes y / o módulos de la aplicación.

Se han utilizado para controlar el flujo de los datos a través de la web app, para todas las cargas de información en diferido, para él envió de datos al back-end. Así como la consulta de estos.

A continuación, se detallan los servicios utilizados y sus principales funcionalidades.

Download

Este servicio se encarga de todas las llamadas HTTP del tipo GET al servidor de back-end.

Sus principales funcionalidades son:

- **Download**: Descarga datos de cualquier modelo teniendo en cuenta en el back-end los permisos necesarios para ello.
- **DownlaodUserMEssage**: Descarga los mensajes del usuario conectado
- **DownloadContentFromQR**: Descargara el contenido asociado de un QR en particular.
- **DownloadContentUser**: Descargara el contenido disponible para un usuario.
- **DownloadNotAuthoerizedContent**: Descarga el contenido que aún no ha sido autorizado (Sólo administrador).
- **HavePublishedContent**: Comprueba si un usuario tiene contenido ya publicado.

Upload

Este servicio se encarga de todas las llamadas HTTP del tipo POST al servidor de back-end.

Sus principales funcionalidades son:

- **Upload**: Se utiliza a la hora de crear un punto de interés. Envía la información de este a él back-end para que se cree en la tabla de la base de datos correspondiente (Solo administradores).
- **InserMessage**: Inserta un mensaje para un usuario en la BBDD enviándole los parámetros necesarios al back-end
- **removeOldContent**: Actualiza la tabla QR para quitar el contenido caducado a la hora de asociarlo un contenido nuevo.
- **Update**: Permite actualizar registros en la base de datos teniendo en cuenta los permisos para diferentes acciones según el usuario y la tabla en el back-end.
- **Delete**: Permite borrar registros en la base de datos teniendo en cuenta los permisos para diferentes acciones según el usuario y la tabla en el back-end.
- **Content_userRelation**: Crea una entrada en la base de datos en la tabla que relaciona un contenido con un usuario.
- **Exist**: Comprueba si un id existe ya en la base de datos. Teniendo en cuenta los permisos necesarios para ello en el back-end.
- **setValToNull**: Permite poner un valor de un registro en la base de datos teniendo en cuenta

los permisos para diferentes acciones según el usuario y la tabla en el back-end.

Login

Este servicio se encarga de guardar y gestionar el login del usuario en el front-end. Guarda una copia de un objeto de usuario con sus datos. Sus principales funcionalidades son:

- **ItsLLogged**: Comprueba si el usuario está logueado en el front-end.
- **ItsLoggedIn**: Comprueba si un usuario está logueado en el back-end.
- **ItsAdmin**: Comprueba si un usuario es administrador en el objeto de este.
- **SetUser**: Guarda una copia del usuario en el LocalStorage [7]. del navegador.
- **GetUser**: Devuelve la copia del usuario guardada en el LocalStorage.

WorkFlowService

Este Servicio es el encargado de controlar el flujo de la web app así que en él se guardan los datos necesarios para ello además de tener las funcionalidades que esto necesita. A continuación, se exponen las funcionalidades más importantes.

- **InitializePoints**: Inicializa los parámetros del mapa al cargar la librería de google.
- **QrProximity**: Calcula la proximidad a los diferentes puntos de interés desde la posición del dispositivo en el GPS.
- **ItsRedable**: Consulta la última información almacenada por GPS si el punto de interés en él, está al alcance de la ubicación del usuario.

Capítulo 4
Problemas y soluciones

A pesar de tomar una planificación previa al desarrollo de la web app, se han encontrado diversos problemas a la hora de desarrollarla. Así que en este capítulo se detallan dichos problemas y cómo se han solucionado.

4.1 Problema de inicio de sesión propio

El inicio de sesión propio guardando las contraseñas de los usuarios, codificándolas y decodificándolas, era la primera opción que se había contemplado para el proyecto, pero surge una serie de las problemáticas a través de este.

4.1.1 Seguridad

La seguridad de un inicio de sesión es una web app es complejo ya que existen miles de exploits que tratan de forzar el sistema por la red. Controlarlo es una tarea que se nos escapa de tiempo de programación propuesto.

4.1.2 Comodidad y confianza

El registro de un usuario en la plataforma supone pedirle datos a un usuario de manera que puede pensar que nuestra aplicación quiere recopilarlos para algún fin diferente.

Por otra parte, también es un paso que puede incomodar a muchos usuarios y que puede hacer que debido a ello no usen la aplicación.

4.2 Solución Inicio de sesión con Google

Para solucionar las problemáticas descritas en la sección anterior Se ha decidido externalizar el inicio de sesión. Para ello utilizaremos una autenticación adaptable.

La autenticación adaptable, también conocida comúnmente como autenticación basada en riesgos, es un paradigma de autenticación que intenta hacer coincidir las credenciales de autenticación requeridas con el riesgo percibido de la conexión o las autorizaciones solicitadas. El objetivo es tratar de reducir la carga de autenticación de los usuarios y proporcionar una mejor experiencia, por un lado, al tiempo que aplica una autenticación sólida.

Para ello se ha usado el software de terceros auth0 y su plataforma. Para ello se tuvo que realizar un registro de la app sobre la misma y la configuración en el back-end. Tal y como se explica en la sección correspondiente del back-end.

4.3 Problema de las sesiones volátiles

Cuando se desarrolló el back-end se crearon sesiones con la librería Session de ExpressJs. Dichas sesiones manejadas por la librería por defecto se guardan en la memoria del proceso. El problema viene a la hora de desplegar la aplicación en desarrollo. Por ejemplo, en nuestro caso lo hemos desplegado en Heroku.

El despliegue en Heroku nos daba un problema con estas sesiones ya que esos servicios no permiten guardar datos de esta manera en la memoria del proceso. Y por lo tanto la aplicación no funcionaba al no poder guardar la información de la sesión de usuario que se utiliza para controlar el acceso posterior a los recursos.

4.4 Solución De las sesiones volátiles

Para solucionar dicho problema investigamos acerca de esto y vemos que se tienen varias opciones a la hora de guardar las sesiones. Implementado algunas librerías sobre la librería de sesiones podemos hacer que el almacenamiento de las sesiones se haga de otras formas.

En este caso se utilizó mysql-session una librería que nos permite guardar la información de las sesiones en una base de datos SQL de tal forma que aprovechamos la base de datos que ya tenemos para guardar en esta la sesiones.

4.5 Problema de HTTPS

A la hora de probar la app se vio que para utilizar la cámara y el GPS necesitábamos tener la aplicación sobre HTTPS. Ya que estas funcionalidades no están disponibles por seguridad sobre HTTP.

4.6 Solución HTTPS

Para solucionar este problema lo que se hizo fue crear unos certificados con OpenSSl en local y se añadieron a los servidores de back-end y front-end de NodeJs. Esto soluciona el problema, pero nos hizo encontrarnos con otro que se verá a continuación.

4.7 El problema de CORS

A la hora de desplegar la aplicación en local no podíamos acceder desde el front end al back-end ya que nos daba un error de CORS (Cross Origin Resouce Sharing)

El problema es que para acceder desde un dominio X un dominio Y. EL dominio X desde tener permitido el acceso del dominio Y es sus cabeceras HTTP. Por lo tanto, debe tener permitidos como orígenes el dominio X para estas llamadas HTTP.

4.8 Solución del problema CORS

Para solucionarlo se creó un middleware en el back-end que añadiera el dominio del front-end como posibles orígenes de dichas llamadas. Una vez se hizo esto se pueden recibir peticiones desde el front-end en el back-end.

4.9 Problema del despliegue múltiple y CORS

A la hora de desplegar la aplicación en Heroku se vio que se tenía que cambiar los orígenes permitidos a la aplicación y cambiar también en el front-end las líneas que contenían el dominio del back-end. Esto haría muy tedioso estar manipulado algo en desarrollo y desplegarlo en producción teniendo que cambiar una a una esas líneas.

4.10 Solución del despliegue múltiple y CORS

Para solucionar este problema se hizo primero una rama por cada uno de los despliegues que se tenían: Heroku en local, Heroku en remoto y el despliegue de desarrollo con la máquina virtual.

Aunque fue una buena solución pronto se vieron problemas en esta ya que al hacer un cambio en

la rama de desarrollo había que actualizar continuamente la rama de producción con los cambios.

Así que finalmente se adoptó otra solución. Se crearon archivos de entorno tanto en local como en remoto y se hizo que a través de variables de entorno el despliegue sepa qué valores ha de coger a la hora de ejecutar los servidores de NodeJs.

Para ello únicamente quedó un solo parámetro que cambiar en el back-end para que éste supiese que está en modo desarrollo o despliegue. Y que a través de condiciones se auto elijan los dominios correspondientes.

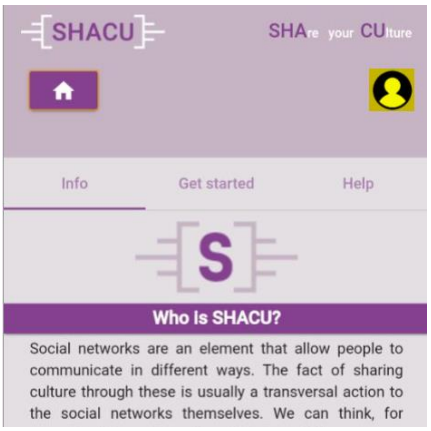
Capítulo 5
Guía de la aplicación

Esta sección explica cómo utilizar las diferentes funciones de Shacu. Se detallarán por secciones de tal forma que puedas deslizar, si lo desea, al apartado correspondiente. Esperamos que disfrutes la experiencia.

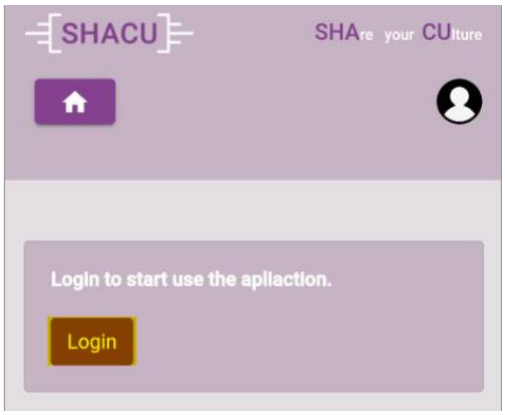
5.1 “Login”

1. Para empezar a usar Shacu debes primero pulsar sobre el botón de perfil que aparece en la parte superior derecha de la pantalla. Esto te llevará a la sección de “login”.
2. A continuación, en esta sección solo debe pulsar el botón e iniciar sesión que se encuentra en el centro de la pantalla. tal y como puede verse en la segunda imagen.

5.1 Login

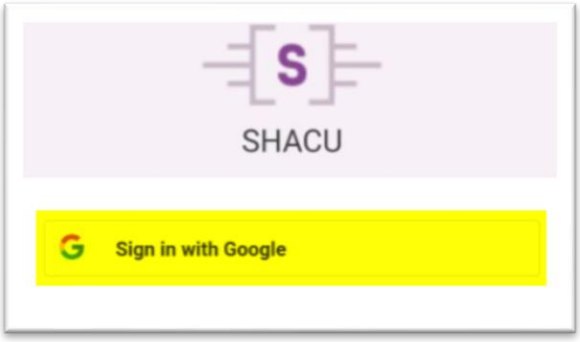


5.2 Login

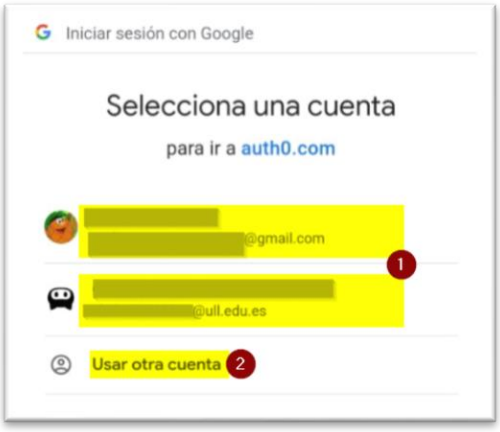


3. Después se le redireccionará al servicio externo Auth0 en el que deberá iniciar sesión con su cuenta de Google. Para ello pulse el botón e iniciar sesión con Google, tal y como se aprecia en la imagen número tres.
4. Con esto llegará a una pantalla donde podrá seleccionar su cuenta de google asociada a este dispositivo (1) o seleccionar otra cuenta (2).

5.3 Login



5.4 Login

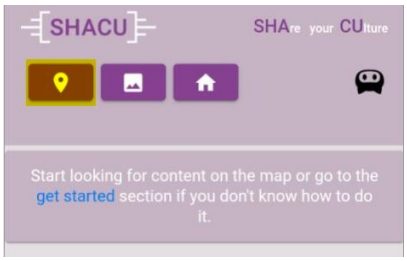


Una vez inicie sesión será redireccionado a la pantalla de contenido, aunque en este caso estará vacía hasta que usted empiece a leer contenidos de los códigos QR.

5.2 Mapas

1. Para empezar a escanear contenido debe primero dirigirse a la sección de mapas. Una vez inicie sesión podrá acceder a esta con el botón que aparece en el menú superior con el icono de posición, cómo puede observar en la primera imagen. Debe asegurarse de activar el GPS de su dispositivo, y dar permisos correspondientes, para un correcto funcionamiento de la aplicación.

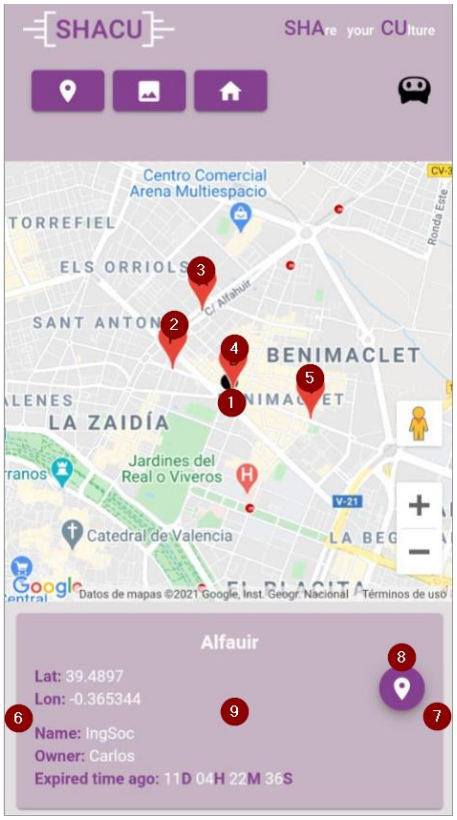
5.5 Mapas



2. Una vez esté en la sección de mapa, si todo ha ido bien podrá observar el mapa y en él su posición (1) y los puntos de interés (2,3,4,5). Puede pulsar sobre los puntos de interés para centrar el mapa en este y ver el nombre del contenido que contiene en una pequeña ventana

emergente. Si no visualiza ningún punto en el mapa aleje el zoom hasta encontrarlos.

5.6 Mapas



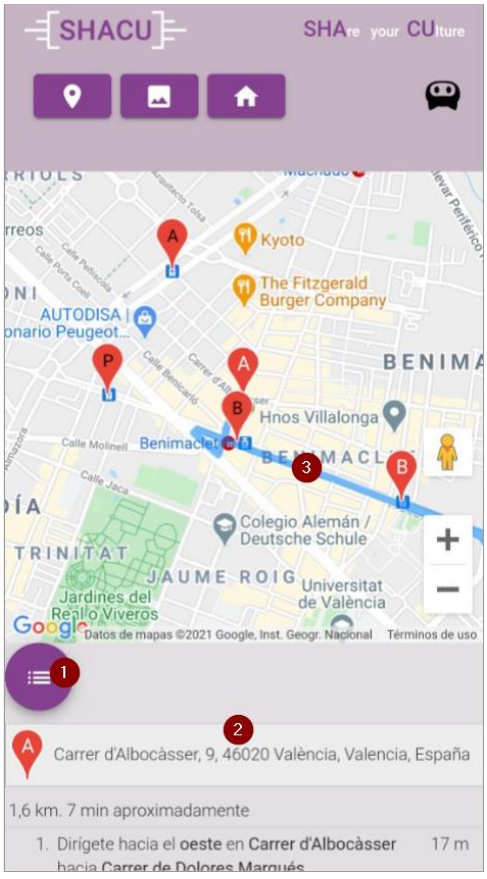
En la parte interior de la pantalla podemos observar un pequeño slider que podrá desplazar a derecha e izquierda (6, 7) Además de la información del punto de interés (9): Su longitud, su latitud, el nombre del punto de interés, el nombre del contenido asociado a este, el propietario del contenido y el tiempo que queda o hace que se ha caducado el contenido. También veremos un botón con el icono de la ubicación que nos permitirá ir a una ventana con más detalles sobre el contenido y la posibilidad de trazar una ruta hasta este.

3. Si pulsamos sobre el botón de detalles accederemos a la ventana emergente nombrada anteriormente. Ahí tendremos la información del punto de interés y el contenido relacionado a este (1). Además, también veremos tiempo de caducidad del contenido (2) cómo veíamos en la anterior sección. Por otro lado, tenemos el botón marcado con el icono de dirección (3) que nos creará una ruta desde nuestra ubicación hasta el punto de interés que tengamos seleccionado en la slider.
4. Si pulsamos sobre el botón de ruta se nos cerrará la ventana emergente y se abrirá en lugar del slider una lista de instrucciones de la ruta. Pudiendo pulsar sobre cada una de ellas para verlas en más detalle en el mapa. Por otro lado, tendremos encima de esta un botón que volverá a mostrar el slider y nos permitirá navegar de nuevo en él por los puntos de interés.

   Ahora lo que debe de hacer es desplazarse siguiendo las indicaciones hasta el punto de interés.

5.7 Mapas

5.8 Mapas



5. Una vez llegue al punto de interés podrá identificarlo en un cartel tal y cómo el que se muestra en la quinta imagen. Ahora solo tiene que empezar a leer el contenido.

5.9 Mapas

5.3 Lectura de QR

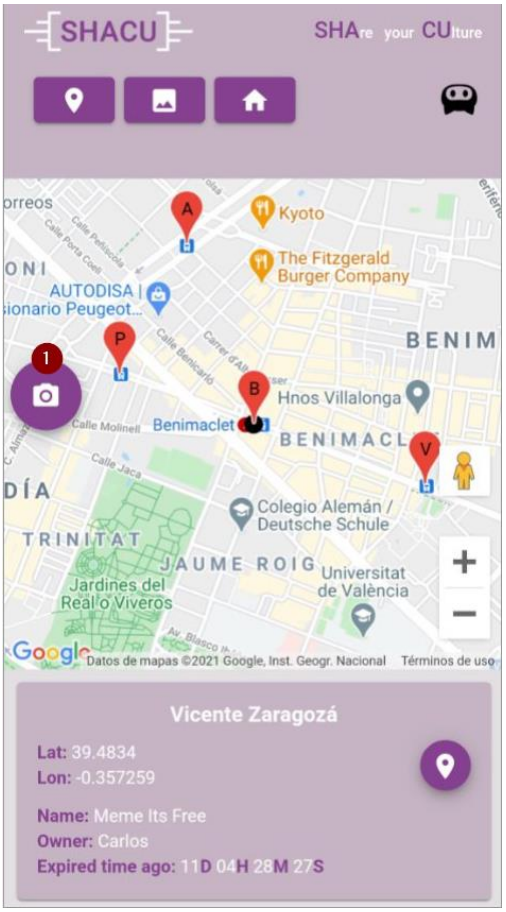
1. Cuando se encuentre lo suficientemente cerca de un punto de interés. En la pantalla del mapa aparecerá un icono de cámara (1). Pulse éste para acceder a la lectura del código QR.

   Alguna de las cámaras de su dispositivo debe funcionar correctamente. Además, debe haberse asegurado de dar los permisos correspondientes a la aplicación para acceder a la cámara.

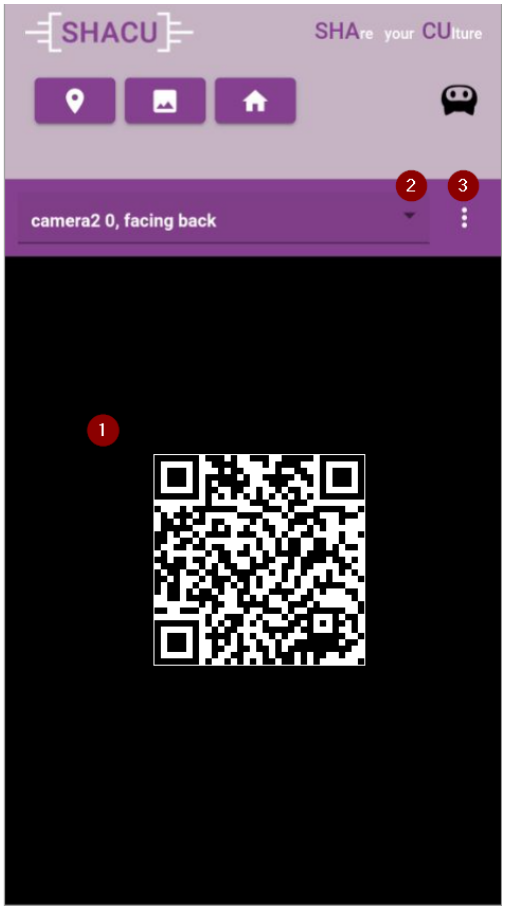
2. Una vez se abra la cámara, tiene opciones (3) para encender el flash de la cámara, y activar el modo “try-harder” que será útil en dispositivos con una menor resolución. Pruébelo si no logra leer el contenido correctamente. También podrá cambiar la cámara del dispositivo que quiere utilizar (2).

   Para la lectura del código enfoque el código QR mintiéndolo en el centro de la pantalla (1) a ser posible. Cuando se lea de manera correcta se le auto re-dirigirá al detalle del contenido que acaba de leer, podrá descargarlo en ese momento o siempre que lo quiera desde la galería de contenidos.

5.10 Lectura QR



5.11 Lectura QR



3. En la sección “detalles de contenido” podrá desplegar la descripción (1) que el autor ha dado a dicho contenido, ver los detalles del contenido, el nombre del fichero a descargar, su peso, etc.

   Por otro lado, tendrá un botón de descarga que descargará el contenido a su teléfono (3).

   Para ver la lista de contenidos leídos podemos pulsar en el botón del menú superior con el icono de imagen (4).

   Y por último la opción de subir un contenido nuevo (2) que nos llevará a un formulario donde deberá rellenar los campos y subir un contenido al pulsar el botón de subida.

   El contenido nuevo no se actualizará inmediatamente. Un administrador debe autorizarlo.

   Una vez se decida si se autoriza o no le llegará un mensaje a la aplicación con el resultado de la autorización. Si es positivo se publicará en ese momento. De tal forma que los demás usuarios tendrán acceso a tu contenido.

4. Al entrar en la lista de contenidos disponibles veremos una lista de estos con un icono, correspondiente a su tipo de archivo, y un botón (1) que nos permitirá abrir la sección “detalles de contenido” donde se nos redirige al leer el contenido. No podrá subir un contenido para reemplazar a éste a no ser que este a una distancia corta del punto de interés que lo contenga.

5.12 Lectura QR

5.13 Lectura QR

5.4 Perfil

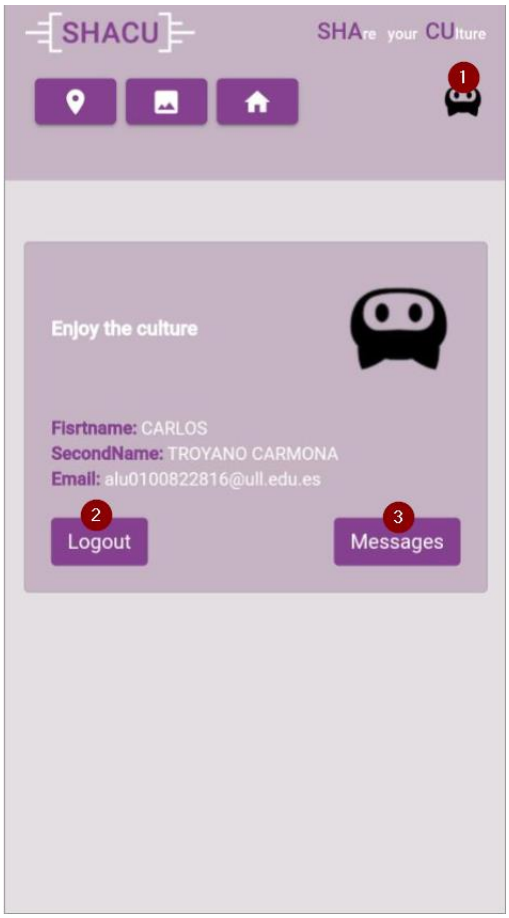
1 Para acceder a las opciones de perfil debe pulsar sobre el icono de su imagen e perfil de google en la parte superior derecha de la pantalla (1).

Una vez allí podrá ver sus datos personales traídos desde Google. Además, podrá cerrar sesión (2) y acceder su bandeja de entrada (3), de mensajes de la aplicación.

2 En la bandeja de entrada de mensajes tendrá por un lado los mensajes nuevos (1) y por otro lado los mensajes ya leídos (2). Podrá desplegar y replegar las listas de mensaje para su comodidad. Además, también tendrá la opción de pulsar sobre los mensajes listados para leerlos (4). Y por último podrá marcar un mensaje como leído o no leído (5) o simplemente borrarlos (6).

Con esto termina en la guía de usuario si tiene más preguntas póngase en contacto con nosotros en el correo que podrá encontrar en la sección de contacto.

5.14 Perfil de usuario



5.15 Perfil de usuario

Capítulo 6
Conclusiones y líneas futuras

6.1 Conclusiones

El presente proyecto hemos desarrollado e implementado una red social llamada Shacu que permite compartir contenidos a través de códigos QR, localizando estos en un mapa y estableciendo una lectura por proximidad, dando a sus usuarios la posibilidad de difundir materiales propios, así como la de compartir información sobre localización y posicionamiento, mapas y la cámara del dispositivo. Se ha elaborado un software desde su planteamiento hasta el producto final.

El desarrollo de un concepto conlleva mucho más que sentarse e implementar líneas de código una detrás de otra. Cuando programamos un proyecto sencillo, es fácil llevarlo a cabo con una serie de pasos ordenados que nos permitan desarrollarlo en alguna plataforma. La complejidad aumenta al añadir más factores que afecten a las funcionalidades.

Esto da lugar a que se haga más y más difícil la implementación con factores cómo: el soporte de varias plataformas, circunstancias no previstas en los modelos propuestos inicialmente.
La consecuencia es un aumento en el número de horas que conlleva la ejecución del proyecto. Por otra parte, la falta de tiempo y recursos para el desarrollo de un proyecto de envergadura más o menos grande hacen que su estructura y su programación entre en una entropía que hace que los códigos sean cada vez menos manipulables y entendibles.

Por ello cuanto mayor el número de funcionalidades mejor planificada debe de estar la aplicación. Y no solamente mejor planificada sino mejor ejecutada. No todo es planificación, es de gran importancia lo riguroso que se debe de ser a la hora de seguir protocolos y pautas. Mucho más si hay más de una persona implicada o se pretende que en algún punto otras personas puedan manipular ese código.

Finalmente, faltaría hacer una puesta en marcha de la aplicación para poder recoger datos y analizar la viabilidad de las herramientas que ofrece.

6.2 Líneas futuras

Hay tres grandes líneas futuras planteadas para este proyecto.

La implantación a nivel local dentro de una serie de paradas de tranvía o dentro de las facultades de una universidad. De tal forma que pudiéramos analizar la aplicación en producción y tomar datos de su uso y de lo relevantes que pueden ser o no para los usuarios ciertas funcionalidades o aspectos de esta.

Crear una funcionalidad que permita a los usuarios agregarse como amigos y enviarse mensajes entre ellos. De tal manera que le dé un punto de red social más interactiva entre los propios usuarios de forma directa, y no tanto basándose únicamente en compartir contenidos.

Desarrollar un recomendador de contenido basándose en los que haya previamente leído el usuario y sus amigos. De tal forma que con algún algoritmo de recomendaciones se pueda predecir los gustos de los usuarios, a la hora de mostrarles los puntos de acceso a los materiales en los que puedan estar interesados.

Capítulo 7
Summary and Conclusions

7.1 Conclusion

In this project we have developed and implemented a social network called Shacu that allows content to be shared through QR codes, locating these on a map and establishing a proximity reading, giving its users the possibility of disseminating their own materials, as well as sharing information about location and positioning, maps and the device's camera. A software has been developed from its approach to the final product.

Developing a concept involves much more than sitting down and implementing lines of code one after the other. When we program a simple project, it is easy to carry it out with a series of ordered steps that allow us to develop it on a platform. Complexity increases by adding more factors that affect functionality.

This results in the implementation becoming more and more difficult with factors such as: the support of several platforms, circumstances not foreseen in the initially proposed models.

The consequence is an increase in the number of hours involved in executing the project. On the other hand, the lack of time and resources for the development of a project of more or less large scale means that its structure and its programming enter an entropy that makes the codes less and less manipulable and understandable.

Therefore, the greater the number of functionalities, the better planned the application must be. And not only better planned but better executed. Not everything is planning, it is of great importance how rigorous you must be when following protocols and guidelines. Much more if there is more than one person involved or it is intended that at some point other people can manipulate that code.

Finally, it would be necessary to start up the application to be able to collect data and analyze the viability of the tools it offers.

7.2 Summary

There are three great future lines planned for this project.

Implementation at the local level within a series of tram stops or within the faculties of a university. In such a way that we could analyze the application in production and take data on its use and how relevant certain functionalities or aspects of it may or may not be for users.

Create a functionality that allows users to add themselves as friends and send messages to each other. In such a way that it gives a more interactive social network point between the users themselves directly, and not so much based solely on sharing content.

Develop a content recommender based on those previously read by the user and her friends. In such a way that with a recommendation algorithm, users' tastes can be predicted, when showing them the access points to the materials in which they may be interested.

Capítulo 8
Presupuesto

En el presupuesto se tienen en cuenta las horas de programación además de las invertidas en los diseños, los servicios externos, etc.

Tabla 8.1

Concepto	Unidad	Coste/unidad	Nº unidades	Subtotal
Externos				
Fotografías y montaje	horas	35	4	140€
Diseño logotipo y cartel + maquetación	horas	60	3	240€
Servicios				
Dyno Heroku StandartX1	mensual	20,59	1	20,59€
Google Map Dynamics	mensual	5,76	1	5,76€
Google Directions	mensual	4,12	1	4,12€
Google Distance Matrix	mensual	8,14	1	8,14€
Desarrollo				
Análisis y generación del modelo 4+1	horas	70	15	1090€
Análisis y generación de la BBDD	horas	70	15	1090€
Desarrollo de back-end	horas	50	120	6000€
Desarrollo front-end	horas	45	200	9000€
Total				17,600

Capítulo 9
Referencias

[1] Denso Wave, «www.qrcode.com,» 1994. [En línea]. Available: https://www.qrcode.com/en/.

[2] V. P. I. Ltd, «https://www.visual-paradigm.com/aboutus/,» [En línea]. Available: https://www.visual-paradigm.com/aboutus/.

[3] C. T. Carmona, «Repositorio de la memoria,» 2020. [En línea]. Available: https://github.com/ULL-ESIT-GRADOII-TFG/tfg-carlos-troyano-memoria.

[4] C. T. Carmona, «Repositorio Frontend,» 2020. [En línea]. Available: https://github.com/ULL-ESIT-GRADOII-TFG/tfg-carlos-troyano-software-frontend.

[5] C. T. Carmona, «Repositorio de código Backend,» 2020. [En línea]. Available: https://github.com/ULL-ESIT-GRADOII-TFG/tfg-carlos-troyano-software-backend.

[6] C. T. Carmona, «Despliegue de la apliación en Heroku,» 2021. [En línea]. Available: https://shacu-front-end.herokuapp.com/.

[7] P. Kruchten, «Architectural Blueprints—The “4+1” View Model of Software Architecture,» *IEEE Software,* vol. 12, pp. 42 -50, 1995.

[8] F. Asteasuain, UML GOTA A GOTA, S.A. ALHAMBRA MEXICANA, 1999, p. 224.

[9] O. D. C. Moreno, «Modelo Entidad - Relación,» 13 October 2007. [En línea]. Available: https://www.slideshare.net/oswchavez/clase-3-modelo-entidad-relacion. [Último acceso: 2020].

[10] OneSpan, «Autenticación basada en riesgos,» [En línea]. Available: https://www.onespan.com/es/topics/autenticacion-basada-en-riesgo-rba. [Último acceso: 2020].

[11] MDN Web Docs, «Window.localStorage,» MDN, 2020. [En línea]. Available: https://developer.mozilla.org/es/docs/Web/API/Window/localStorage. [Último acceso: 2020].

[12] Bezkoder, «Angular 10 + Node.js Express + MySQL example: CRUD Application,» bezkoder.com, 2020. [En línea]. Available:

https://bezkoder.com/angular-10-node-js-express-mysql/.

[13] Google, «Maps JavaScript API,» 2020. [En línea]. Available: https://developers.google.com/maps/documentation/javascript/overview.

[14] Heroku, «SQLite on Heroku,» 15 November 2017. [En línea]. Available: https://devcenter.heroku.com/articles/sqlite3.

[15] Heroku, «Deploying Node.js Apps on Heroku,» 12 February 2021. [En línea]. Available: https://devcenter.heroku.com/articles/deploying-nodejs. [Último acceso: 2020].

Capítulo 10
Apéndices

10.1 Base de datos

```
/*****************************************************************************
 *
 * init.sql
 *
 *****************************************************************************
 *
 * Carlos Troyano Carmona
 *
 * 15/12/2020
 *
 * Este fichero contiene la creación de la base de datos correspondiente al
 * análisis entidad relación
 *
 *****************************************************************************/

DROP DATABASE IF EXISTS shacu;

CREATE DATABASE shacu;

USE shacu;

CREATE TABLE users (
    id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    firstname VARCHAR(30) NOT NULL,
    email VARCHAR(50) NOT NULL,
    nick VARCHAR(10) NOT NULL,
    is_admin BOOLEAN DEFAULT false,
    reg_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    UNIQUE KEY unique_email (email)
);

CREATE TABLE qr (
    id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    lat float(20),
    lon float(20),
    qr_name VARCHAR(200) NOT NULL,
    qr_image VARCHAR(15000),
    published BOOLEAN DEFAULT false,
    reg_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);
```

```sql
/* content is published if have qr_id not qr_queque;  */
CREATE TABLE content (
    id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    qr_queue INT(6) UNSIGNED,
    queue_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    qr_id INT(6) UNSIGNED,
    user_id INT(6) UNSIGNED,
    size INT(10) UNSIGNED,
    file_path VARCHAR(500) NOT NULL,
    content_name VARCHAR(200) NOT NULL,
    content_description VARCHAR(1000) NOT NULL,
    content_type VARCHAR(50) NOT NULL,
    authorized BOOLEAN DEFAULT false,
    reg_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    FOREIGN KEY (qr_id)
        REFERENCES qr(id)
        ON DELETE CASCADE,
    FOREIGN KEY (user_id)
        REFERENCES users(id)
        ON DELETE CASCADE
);

CREATE TABLE content_user (
    user_id INT(6) UNSIGNED NOT NULL,
    content_id INT(6) UNSIGNED NOT NULL,
    PRIMARY KEY  (user_id, content_id),
    FOREIGN KEY (user_id)
        REFERENCES users(id)
        ON DELETE CASCADE,
    FOREIGN KEY (content_id)
        REFERENCES content(id)
        ON DELETE CASCADE
);

CREATE TABLE messages (
    id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    content_id INT(6) UNSIGNED,
    user_id INT(6) UNSIGNED NOT NULL,
    message_user VARCHAR(15000),
    readed BOOLEAN DEFAULT false,
    reg_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    message_type VARCHAR(200),
    FOREIGN KEY (user_id)
        REFERENCES users(id)
        ON DELETE CASCADE
);
```

10.2 Fichero de entorno

```
/*********************************************************************************
 *
 * .env
 *
 ********************************************************************************
 *
 * Carlos Troyano Carmona
 *
 * 15/12/2020
 *
 * Este fichero contiene las variables de entorno para que la aplicación funcione
 * en todos los entornos.
 ********************************************************************************/

################ DEVELOPMENT ##############
# true only in local
DEV=false
#DEV=true

################ AUTH0 local ##############
AUTH0_CLIENT_ID=9xJPNqFszIwJsLZsV79v4xQg6ClJoHM3
AUTH0_DOMAIN=dev-28g2czvu.eu.auth0.com
AUTH0_CLIENT_SECRET=*******
# callback dev
AUTH0_CALLBACK_URL_DEV=https://192.168.1.58:8080/callback
# callback heroku testing
AUTH0_CALLBACK_URL_LOCAL=https://192.168.1.58:5000/callback
# callbackheroku prod
AUTH0_CALLBACK_URL=https://shacu.herokuapp.com/callback


################ Freont end ##############
# DEV
FRONTEND_URL_LOCAL_DEV=https://192.168.1.58:4200
# HEROKU LCOAL
FRONTEND_URL_LOCAL=https://192.168.1.58:5000
# HEROKU PRODUCTION
FRONTEND_URL=https://shacu-front-end.herokuapp.com


################ Database ##############
# DEV
```

```
DB_CONNECTION_DEV=mysql://root:Cloud87?@localhost:3306/shacu
# HEROKU
DB_CONNECTION=mysql://ba737b535787a4:bd39c273@us-cdbr-east-
02.cleardb.com/heroku_afed953543e48b3?reconnect=true

# HEROKU DB SETTINGS
DB_HOST=us-cdbr-east-02.cleardb.com
DB_USER=ba737b535787a4
DB_PASSWORD=*******
DB_DATA_BASE=heroku_afed953543e48b3

# LOCAL DB SETTINGS
DB_HOST_LOCAL=localhost
DB_USER_LOCAL=root
DB_PASSWORD_LOCAL=*******
DB_DATA_BASE_LOCAL=shacu


################ APP config ##############
EXPIRATION_CONTENT=600000
```