Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

CanaryXperience

CanaryXperience

Sergio del Pino Hernández

La Laguna, 6 de julio de 2020

D. **Vicente José Blanco Pérez**, con N.I.F. 42.171.808-C profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor

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

Que la presente memoria titulada:

”*CanaryXperience*”

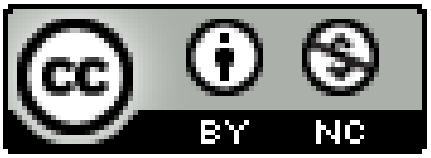
ha sido realizada bajo su dirección por D. **Sergio del Pino Hernández**, con N.I.F. 78.64.07.44-H

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 6 de julio de 2020

Agradecimientos

Agradecer a todos mis familiares, profesores, amigos y todas las personas implicadas directa e indirectamente en el desarrollo de mi carrera en la universidad y que han hecho posible este proyecto.

Licencia



Resumen

La gestión de una aplicación web para las empresas no IT es uno de los principales desafíos a los que deben enfrentarse en tiempos de comercio electrónico.

CanaryXperience es una plataforma de intercambio de contenido, por un lado, para compañías cuyo modelo de negocio sea la organización y venta, tanto de excursiones como de experiencias; y, por otro lado, para usuarios interesados en realizar dichas experiencias en las Islas Canarias

El objetivo de este trabajo es que las empresas posean un lugar común, accesible y económico para poder publicitarse de manera online.

La metodología llevada a cabo ha sido kanban, que ha ayudado a organizar las tareas de este TFG.

Una de las principales conclusiones que podemos extraer de este proyecto es que las empresas deberían beneficiarse de estos servicios web para que puedan abstraerse de ese tipo de preocupaciones y centrar todo su esfuerzo en lo importante, su modelo de negocio. Para todo ello se ha creado CanaryXperience.

Palabras clave: Aplicación web, diseño mobile first, Patrones de diseño, Typescript, VueJs, NodeJs, Mongodb, Docker, Kubernetes, Microservicios

Abstract

A website management is one of the main challenges for non-IT companies they must face in times of e-commerce.

CanaryXperience is a content exchange platform, on the one hand, for companies whose business model is the organization and sale of both excursions and experiences; and, on the other hand, for users interested in carrying out such experiences in the Canary Islands.

The main objective of this project is offering a common, accessible and economic place to companies in order to be online advertised.

A Kanban methodology has been carried out, which has helped in the organization of the tasks for this Final Year Project.

One of the main conclusions that we extract from this project is that companies should take advantage of this kind of web services. Thanks to these service, the companies can focus the effort in their bussines model. CanaryXperience has been created with this purposes in mind.

Keywords: web application, mobile first design, design patterns , Typescript, VueJs, NodeJs, Mongodb, Docker, Kubernetes, Microservices

Índice general

Índice de Figuras

Capítulo 1

Introducción

CanaryXperience es una plataforma de intercambio de información en la que las empresas del sector turístico de Canarias puedan publicar sus propios servicios. Esta aplicación web quiere ser un servicio al estilo de Airbnb, donde existan usuarios que reserven actividades y otro tipo de usuario (empresas) que ofrezcan ese tipo de servicio. El objetivo es que las compañías puedan aumentar el número de clientes potenciales a través de este medio. Asimismo, el usuario tendrá una sección especializada en la búsqueda de ofertas de excursiones y experiencias que serán ofrecidas por las empresas en nuestra plataforma, de modo que el acceso a ellas sea fácil e intuitivo para el consumidor final. Pretende ser un punto de encuentro donde muchos usuarios puedan también interactuar entre ellos, es decir, usuarios que ya hayan disfrutado de la experiencia puedan ser capaces de dar su opinión para orientar a otros usuarios que deseen disfrutar de la misma experiencia. Gracias a esta plataforma, muchas compañías que no podían y querían dar el paso a la vida digital, podrán hacerlo y les resultará mucho más cómodo y factible. Esto se debe a que, además de sus características, tendrán que asumir un coste más bajo, pues pagar pagar a una empresa externa para que les haga una web personalizada o contratar a un desarrollador para que haga una aplicación web a medida supondría más gastos.

1.1. Antecedentes

Tradicionalmente, todas las empresas turísticas que ofrecen servicios de excursiones en Canarias se apoyaban en las agencias de viajes para ofrecer sus servicios ganando la agencia una pequeña comisión por publicitar sus excursiones. Hoy en día, muchas de estas agencias han tenido que moverse al mundo digital, al igual que supermercados, tiendas de ropa, tiendas de electrónica, etc. Es innegable que día tras día hay más gente comprando y contratando servicios a través de internet. Por tanto, muchas de las empresas que ofrecían servicios de excursiones han seguido esta tendencia y se han intentado colocar, también, en el mercado digital, pero no muchas lo han conseguido. Algunas no pueden contratar los servicios de

forma online, otras directamente no están presentes en la red, y las que sí han apostado por esta modalidad, lo han hecho con páginas que no están pensadas para que reciban un gran pico de usuarios, o se han hecho con un presupuesto limitado. Creadaa con tecnologías antiguas o incluso han tenido que invertir una gran cantidad de dinero para externalizar este proceso a otras empresas de transformación digital para poder llevar a cabo esta transición.

1.2. Probemática

Tener una página en internet es un trabajo que no es muy complicado, pero debe conocerse el proceso o los problemas que esto conlleva. El desconocimiento limitan a empresas no IT, es decir, empresas que su principal actividad están alejadas de la informática, haciendo que haya empresas que aún no han dado el salto a internet y estén perdiendo clientes potenciales.

Por tanto la problemática que se quiere solucionar es la de externalizar esa responsabilidad y que las empresas de turismo, sin un gran capital para la creación y mantenimiento de una página web propia, puedan tener su lugar en internet y ser accesible a una gran cantidad de usuarios. Por otra parte, estos usuarios pueden buscar cientos de excursiones de manera centralizada porque dispondrán de numerosas empresas a su disposición. Todo ello, sin tener que realizar infinitas búsquedas en Google sobre las excursiones. Así pues, los propios usuarios podrían ir descubriendo empresas y experiencias que, de otra manera, les hubiese sido imposible, ya que podrían no conocer siquiera la empresa que organiza la experiencia.

1.3. Objetivo

El objetivo de la realización del trabajo es poder afianzar los conocimientos obtenidos durante la carrera y los distintos cursos de formación que he recibido a través de la universidad. De este modo, lograr hacer un TFG que realmente me gustara ya que es la rama de la informática a la que me dedico de manera profesional. Además, me ha servido como aprendizaje para fortalecer muchos conocimientos, así como usar y poner en práctica muchas tecnologías que no había utilizado antes, por lo que ha sido un buen entrenamiento para el mundo laboral.

Uno de los aspectos en los que también se ha querido mejorar es en la organización de las tareas y la priorización de estas, con el fin de poder realizar el trabajo de la manera más ordenada posible. Aprender el uso de metodologías ágiles y adoptarlas en el trabajo día a día ha sido clave para no entrar en una rueda de estrés en la que piensas más en lo que tienes que hacer que en lo que realmente estás haciendo. Marcar bien las tareas, dividiéndolas en pequeñas subtareas que se podrían estimar en horas para hacer en una semana de trabajo ha sido fundamental para poder llevar a cabo este trabajo.

1.4. Metodología

La metodología utilizada para organizar las tareas de este proyecto ha sido Kanban. [4] Kanban es una metodología ágil basada en la visualización de las tareas que se tienen que realizar. Dichas tareas pasan por un proceso de estimación del impacto que va a suponer esa funcionalidad en nuestro proyecto. Cuando se prevé la importancia de esta, se coloca una carta con la tarea a realizar en un tablero. Finalmente, se organiza de mayor a menor importancia o prioridad las tareas a realizar.

Para llevar el control de las tarjetas, se ha utilizado Google Keep que, aunque no es una herramienta pensada para esta motodología ya que existen aplicaciones web como Trello, Jira o Monday, ha servido de gran ayuda y fácil de integrar en el día a día. Utilizando esta metodología tenemos el resultado de todo esta `http://canaryxperience.site/` y el código visible en el repositorio de la aplcación frontend [13] y el repositorio de nuestro código backend [12]

Capítulo 2

Frontend

El frontend [18] de una aplicación es, en su traducción más literal, algo así como frente o fachada final. Es una parte fundamental del desarrollo que trabaja sobre la interfaz de usuario y hace que el cliente pueda interactuar con nuestra web. Además, se trata de aquello que el usuario ve. Por tanto, en caso de que exista algún error, podría suponer que un usuario deje de usar nuestra página web por el simple hecho de no ser agradable visualmente, funcione mal o no sea usable. Así pues, miles de empresas invierten parte del capital en mejorar sus sitios web, obteniendo un mejor diseño, más claro, usable y accesible.

El frontend orientado a lenguaje de marcas como HTML y CSS [5] y al lenguaje de scripting web de ejecución en equipos clientes, es decir, Javascript, sin necesidad de uso de servidores externos. Casi todo lo que se ve en la pantalla cuando se accede a una web es desarrollo frontend: la estructuración de los apartados, tamaños, márgenes entre estructuras, tipos de letra, colores, adaptación para distintas pantallas, los efectos de ratón, teclado, movimientos, desplazamientos, efectos visuales y un largo etcétera. Esto sería la base origen en la que se centra la especialidad frontend: dar formato a contenidos, dar estilo a la web y manipular resultados de datos obtenidos mediante peticiones al servidor.

Actualmente los términos UI/UX están a la orden del día [11]. Estas disciplinas, que mucha gente suele confundir o piensan que son lo mismo son las que marcan el diseño de cualquier página web. UX o User Experience es una rama de diseño que se centra en cómo el usuario interactúa con un elemento de nuestra interfaz y está directamente relacionado con el conocimiento de los usuarios. Cuanto mayor sea el público destino de las aplicaciones o sitios web, mejor tiene que estar trabajado el UX para que una mayor cantidad de personas puedan entender cómo usar cada uno de los componentes de la interfaz.

UI o User Interface es la rama del diseño que se centra en cómo debe ser la interfaz del usuario en términos de diseño. Es decir, UX se centra en cómo el usuario va a interactuar sobre un elemento y UI se centra en cómo debe ser el aspecto de ese componente. Por tanto, UX y UI deben ir de la mano para lograr

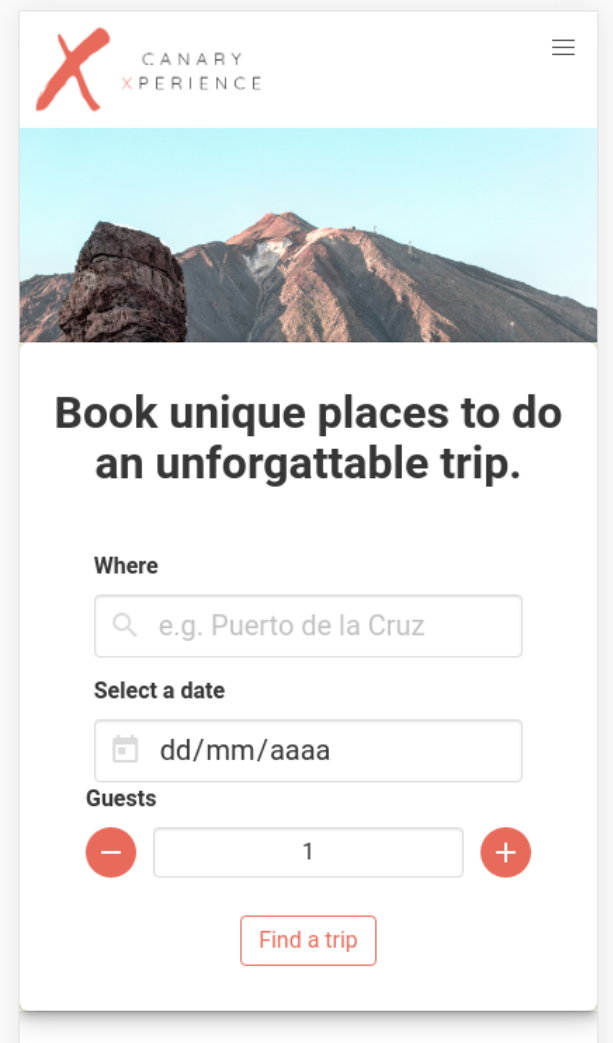


Figura 2.1: Vista diseño mobile

una interfaz de usuario lo más completa, atractiva y usable posible. Son elementos claves para poder lograr un disfrute total de uso de la aplicación por parte de los clientes.

Otros de los aspectos que se tuvieron en cuenta a la hora de realizar el diseño de la web fue el concepto de “mobile first”.[19] Mobile first es una filosofía de desarrollo que se centra en pensar primero en el diseño para móvil y dispositivos de pequeña pantalla, y después de tener este diseño completo, trasladarlo mediante un reescalado a escritorio. Esto nos ayuda a pensar en pequeño, en cómo podemos hacer los layouts de la manera más eficiente posible, mostrando en una pantalla pequeña todos los componentes más importantes y los mínimos necesarios para que nuestra aplicación sea usable.

2.1. Tecnologías (SPA VUE, BABEL Y WEBPACK)

Para la llevar a cabo del desarrollo de parte de cliente hay muchas maneras de hacerlo. Se puede realizar el HTML, CSS y JS como hasta ahora se había hecho de manera nativa. Se puede utilizar un servicio del lado del servidor que cree ese HTML con la información necesaria y enviarlo al navegador del cliente y, por último, que es la opción escogida para llevar a cabo esta parte, es usar una librería o framework que nos ayude en la parte de desarrollo y nos haga la vida más fácil: los frameworks SPA.

Los frameworks **SPA**, (single-page application) o aplicación de una sóla página se usan para crear sitios web en el que sólo existe una única página que va cambiando mediante javascript las páginas con el fin de dar una experiencia lo más fluida e inmediata posible a los usuarios. Todos los códigos de HTLM, CSS y javascript se cargan una única vez y mediante este último se va renderizando la vista de **componentes** que se debe mostrar.

Un **componente** es un pedazo de código que representa una parte de la interfaz con un diseño y funcionalidad definido, que puede ser reutilizado en varias partes de la web. De esta manera se aumenta la modularidad de las vistas de la interfaz gracias a el encapsulado del funcionamiento en cada uno de estos componentes siguiendo un estilo **atómico** de desarrollo.

Una de las limitaciones de este tipo de tecnologías es que si algún usuario ha decidido bloquear el javascript de su navegador no será capaz de ver absolutamente nada de la página.

A la hora de tomar una decisión hay que determinar qué factores nos influyen más y cuáles menos. Así que veamos a los frameworks candidatos. Existen varios frameworks a valorar tales como **Angular**, un framework hecho desde cero por la empresa Google. **Svelte**, uno de los framework que se han puesto más de moda entre los desarrolladores últimamente. **ReactJS**, es una librería creada por la empresa Facebook, que se ha ganado su fama por su gran su rendimiento. Librerías como React Hooks facilitan enormemente el desarrollo frontend y hacen que los desarrolladores estén cómodos con la hemmanienta. Además tiene una alta rapidez de reenderización del DOM gracias al algoritmo llamado react fiber. Estos y muchos otros desarrollos dentro del ecosistema de React han hecho que sea el más utilizado por las empresas y el favorito de muchos. Sin embargo, se ha decidido por usar otro framework, que no es tan conocido, pero que ha ido ganando poco a poco popularidad entre la comunidad de desarrollo. **VueJS**.

VueJS, es uno de los framework progresivo reactivo de una sola página más populares en la actualidad para el desarrollo de páginas webs. Tiene una gran comunidad de desarrollo detrás y es uno de los frameworks más queridos por la gente que lo ha probado. No es el primer proyecto que el alumno hace con este framework, sin embargo se ha decidido utilizar este por varios motivos.

Se ha decidido utilizar VueJS para aprenderlo en profundidad. La sencillez de su sintaxis, la facilidad con la que se definen los componentes, definición de propiedades y lo bien definido que están sus ciclos de vida hace que sea muy fácil de aprender, que no de dominar. Aunque la reactividad no es exclusiva de VueJS, hace que sea de una gran comodidad actualizar variables del html sin nosotros tener que hacer nada.

La velocidad de renderizado de VueJS, sin llegar a los extremos de React JS, es muy alta y se pueden realizar cambios de vistas y carga de componentes en muy poco tiempo. Estos consigue dar una sensación al usuario de fluidez y dinamismo difícilmente obtenible por otros frameworks. Además, el routing en VueJS es muy completo, sencillo de utilizar y muy configurable, por lo que enrutar entre las distintas vistas de la web es muy simple. [15] Hacer este TFG ha sido una experiencia bastante enriquecedora con la cual he afianzado aún más los conocimientos sobre este framework.

Para poder conseguir una retrocompatibilidad con navegadores anteriores, hemos decidido utilizar **Babel**. Babel es una librería para JavaScript que permite transpilar nuestro código desde ECMAS6 a ECMAS5, utilizado por Intenet Explorer 11. Transpilar es una operación entre el compilado y la traducción, mediante la cual el código creado con versiones superiores de Javascript (o supersets de Javascript como TypeScript) es transformado en código compatible con todos los navegadores del mercado.[3]

Para empaquetar todo nuestro código JS y CSS hemos utilizado Webpack, que es uno de los empaquetadores más empleados en el desarrollo de aplicaciones frontend y es uno de los más usados por la comunidad de desarrollo.

2.2. VUEX y ROUTER

Uno de los problemas que se nos plantea al utilizar componentes en estos frameworks es la dificultad de saber en qué momento de la aplicación estamos. Es decir, cada uno de los componentes tiene sus variables; sin embargo, unos componentes dependen del estado de otro para mostrar diferentes datos, necesitan comunicarse entre ellos. La comunicación padre a hijo de los componentes es bastante sencilla, ya que desde el componente padre se pueden pasar una serie de parámetros llamados props que podemos utilizar en el componente hijo y son pasado en tiempo de renderización.

El problema se encuentra cuando queremos comunicar componentes hermanos, o comunicar un cambio en el componente hijo que afecte al componente padre y este necesita renderizarse de nuevo, o dependiendo de un valor de una variable del componente que cambie otra vista completamente diferente.

Para solucionar este problema se ha decidido utilizar VUEX. **VUEX** es una librería de control de estado que implementa el patrón flux, más utilizada en el

framework, ya que dispone de una buena documentación y su uso es bastante sencillo. Flux[8] es un patrón de diseño para el manejo y el flujo de los datos en una aplicación web, particularmente en el FrontEnd. Se caracteriza por tener varias partes fundamentales dentro de la arquitectura como Vista, Store, Actions y Dispachers, que son las encargadas de los cambio de los datos dentro de nuestra vista. 2.2

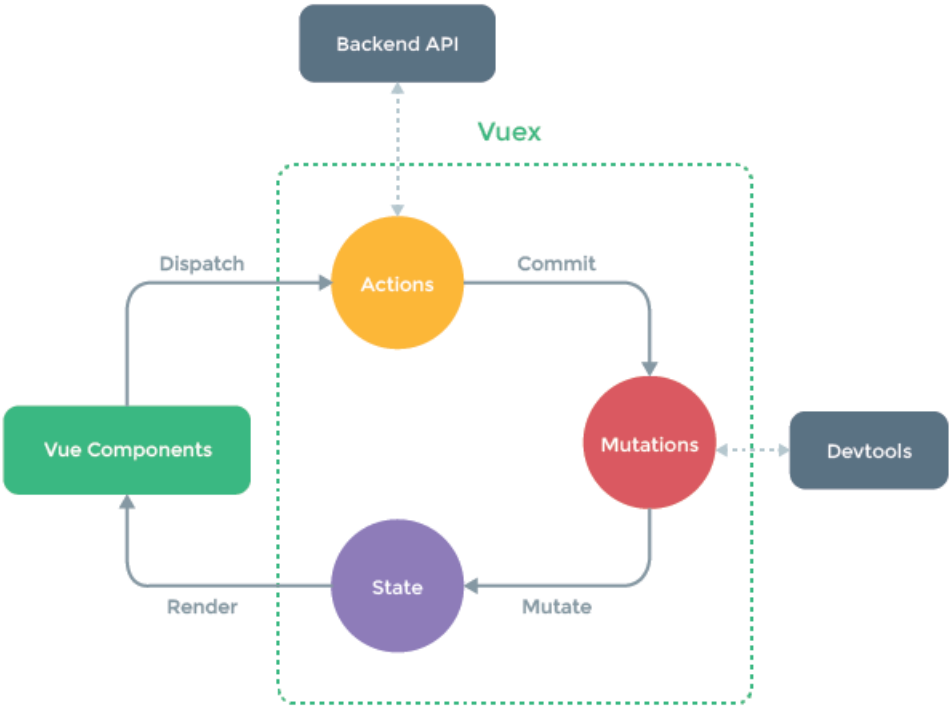


Figura 2.2: Vuex API

Una librería de control de estado es una librería que se usa para conseguir tener el estado de la aplicación global, usable por cualquier componente que pueda acceder al objeto de la librería también denominado **Store** donde se definen los métodos de actualización y lectura de este objeto también llamados **mutations**.

Además, no son variables globales sin más, son capaces de persistir a las recargas de páginas, por lo que es una utilidad bastante potente y que hay que tener muy en cuenta a la hora de hacer nuestras aplicaciones ya que es mucho más eficiente guardar información en este tipo de variables, antes que tener una serie de ids y en cada uno de los componentes hacer peticiones AJAX para recibir los datos de los componentes, ya que no es la manera más rápida y eficiente de conseguir los datos. 2.3

Otra de las librerías que le han dado sentido a los frameworks SPA es la librería de routing o enrutamiento, que en el caso de VUEJS es **Vue Router**.

Las librerías de routing son utilizadas en los frameworks SPA para, como su nombre indica, **enrutar entre las diferentes vistas de la aplicación**. Enrutar sig-

Figura 2.3: Ejemplo de uso de Vuex

nifica renderizar por la pantalla del usuario la vista que tenga que ser renderizada en ese momento. El Router de VUEJS es extremadamente sencillo de utilizar y configurable por lo que es realmente fácil hacer el salto entre páginas. 2.5

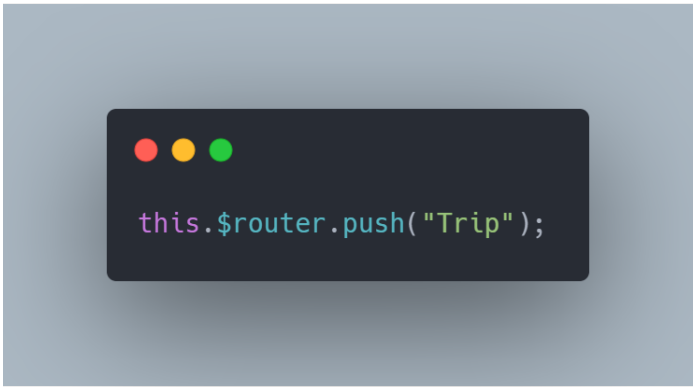


Figura 2.4: Ejemplo de uso de VueRouter

2.3. Ciclos de vida

Los **ciclos de vida** o **hooks** en los componentes son una serie de funciones que se ejecutan en un momento del tiempo determinado. Estas funciones son muy útiles y nos permiten realizar ciertas acciones cómo hacer una petición externa, realizar algunos cálculos, disparar un evento cada vez que ocurra algo, etc. Es una de la ventajas que tenemos respecto a javascript nativo. No necesitamos controlar de ninguna forma cuando esta cargando para inicializar ciertas variables, para realizar otras acciones, o lo que quiera hacerse, sin necesidad de tener en cuenta nada. 2.5

2.4. BULMA y BUEFY

Un framework CSS es una estructura específica y un conjunto de herramientas para poder trabajar con el CSS de una manera más cómoda, fácil y dinámica y así

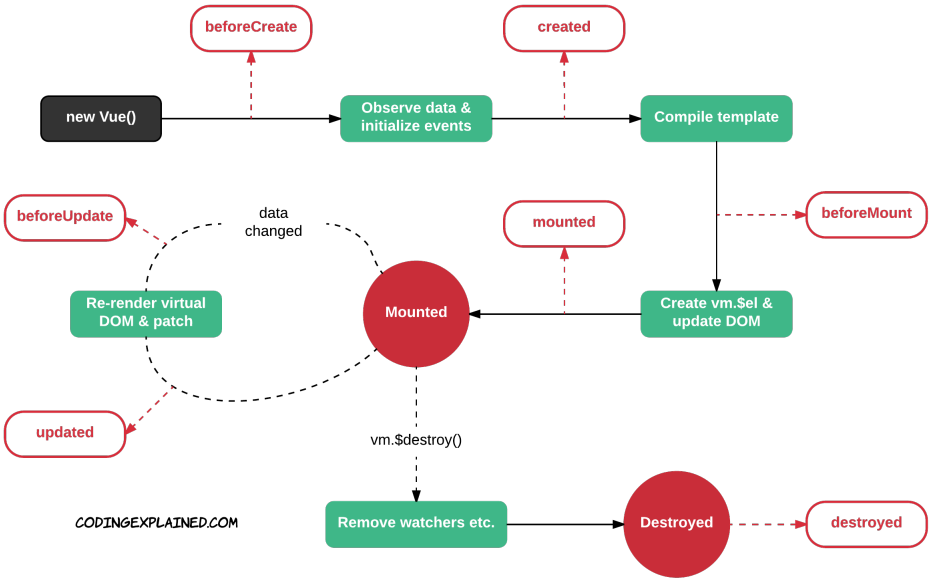


Figura 2.5: Ciclo de vida de los componentes de Vue

implementar diseños para la web. Se centran en la estandarización, en la compatibilidad de navegadores y en el responsive design y código mucho más entendible además de poder así agilizar el desarrollo de la interfaz de las aplicaciones webs. En su estructura vienen hojas de estilo, archivos de configuración, plugins necesarios para su funcionamiento, manejo de iconos e imágenes, etc.

Entre muchas opciones disponemos de el archiconocido **Bootstrap**, que se lleva utilizando en el desarrollo web desde hace casi diez años. Últimamente, se están ganando la confianza de la comunidad de desarrolladores muchos otros frameworks como **Tailwind, Bulma, Semantic UI o Styled Component** entre otros. Sin embargo, se ha decidido por el uso de Bulma.

Bulma [6] es uno de los framework más utilizado de CSS para llevar a cabo el diseño de nuestras aplicaciones webs. Se ha sido el candidato elegido por su sencillez de uso, su estandarización y por su muy buen e intuitivo manejo de los contenedores flex. Sin embargo, lo que realmente se está utilizando es **Buefy**. Este framework una implementación de Bulma orientado a ser usado en VUEJS. Dispone de una muy buena documentación, componentes reutilizable, una gran variedad de elementos fáciles de usar, y un gran manejo de las **renditions** de las fotografías que hace que las imágenes sean capaces de adaptarse a distintos tamaños.

En cuanto a iconos se ha utilizado la librería de **material design icons**, que tiene una gran cantidad de iconos, es de uso libre y su integración con el framework BUEFY es muy buena.

2.5. Testing E2E y Cypress

El **testing** de software de aplicación de tipo funcional consiste en escribir una serie de pruebas o test que ejecuten nuestro código con unos datos de entrada con el fin de comprobar que nuestro código devuelve el resultado esperado. Existen muchos tipos de test pero los principales que se usan a día de hoy son los test unitarios o **unit tests**, test de integración o **integration tests** y test de regresión o **test end to end (E2E)**. 2.6

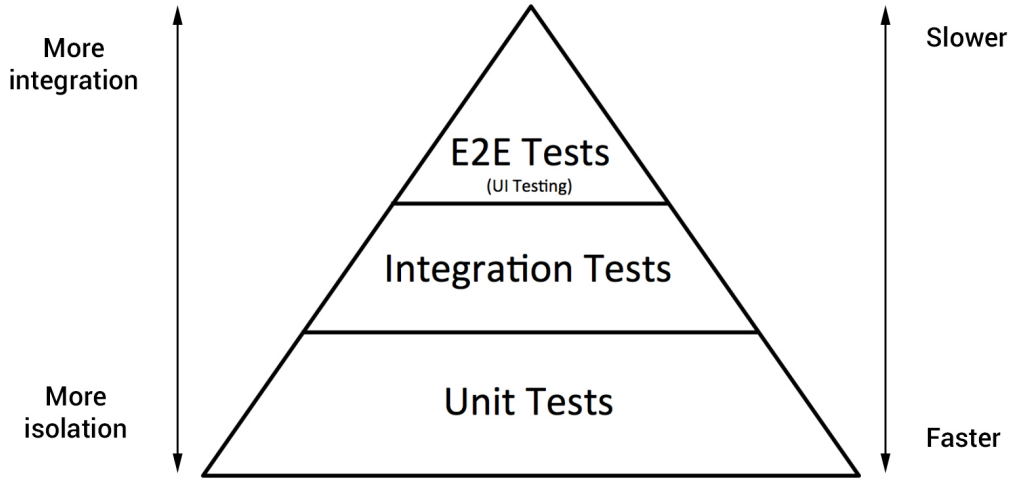


Figura 2.6: Pirámides de tests

Los **unit tests** son los que en mayor cantidad se realizan en el software, ya que son fáciles de programar, muy rápidos de ejecutar, fáciles de mantener y son muy útiles a la hora de probar funcionalidades concretas. [16] Los **integration tests**, son los test que se realizan para comprobar que dos módulos de software diferentes interactúan entre sí de la manera esperada. Son más difíciles de programar, y de ejecutar porque se necesita en ocasiones de conexiones externas o llegar incluso a ser necesario el uso de alguna herramienta adicional a las del software únicamente, como configuración de variables de entorno o logueos antes de realizar los test. Son más lentos en ejecución y por lo tanto caros de producir. Pero en este trabajo queremos enfocarnos sobre los tests end to end a los cuales a partir de ahora se referirá como tests E2E.

Los **tests E2E**, son test que dependen de aplicaciones externas para ser ejecutados, por lo que su realización es mucho más compleja, lleva configuración adicional, instalación de software, configuración de variables de entornos, etc. Su realización debe hacerse de forma más limitada por los motivos anteriormente nombrados. Son test que tiene una gran integración con el producto final. Para llevar a cabo estos tipos de test existen muchos softwares capaces de hacer esto, como **Selenium, Puppeteer, Katalon o Cypress**. En esta ocasión el elegido ha sido cypress ya que su integración con frameworks de js es muy buena, aunque su compatibilidad no sea perfecta es una de las herramientas más sencillas para aprender a usar test E2E. Una de las características de este framework de testing es que los

test pueden ser escritos, lo que es una gran ventaja respecto a otros sistemas que necesitan de ser usados por una persona para que luego el software replique esas acciones, por lo que es mucho más rápido de implementar los tests. Por lo tanto, lo que ejecuta **Cypress** es una instancia de un navegador, donde carga toda la aplicación web emula las acciones que hemos descrito en los test anteriormente comentados. 2.7

```
describe('Test users', () => {
  it('Check counter guest working properly', () => {
    cy.visit('http://localhost:8080')

    cy.get('.selector-guest').find('input').invoke('val')
      .then(value => expect(value).to.be.equal("1"));

    cy.get('.selector-guest').find('.button').first().click();

    cy.get('.selector-guest').find('input').invoke('val')
      .then(value => expect(value).to.be.equal("0"));
  })
})
```

Figura 2.7: Ejemplo de test E2E con Cypress

Lo primero que se debe hacer para llevar a cabo estos test es utilizar la función de **cy.visit()** para que Cypress sepa qué aplicación web debe emular. Una vez cargado el entorno de testeo podemos ejecutar acciones dentro de nuestro **DOM**. Lo primero que hacemos es buscar nuestro inputs donde tenemos seteado la cantidad de usuarios que se quieren apuntar a la excursión y comprobar que el valor por inicial es de uno. Después de esto, buscamos el primer botón de nuestro selector de clientes, es decir el botón que elimina un cliente de nuestra variable, por lo que encontramos ese botón y ejecutamos un **click()** para pulsarlo y ejecutar la lógica asociada al listener que tiene este elemento. Tras esa ejecución volvemos a comprobar el valor de nuestro input de clientes en la cual esperamos que el valor de este sea cero. Si es así podemos concluir que nuestro test pasa y que el comportamiento de nuestro botón es el correcto.

Esto es sólo un pequeño y sencillo ejemplo pero podemos hacer test tan complejos como los requerimientos de nuestra aplicación lo precise. Sin embargo, debemos pensar también que en estos test son muy lentos de ejecutar y de implementar, así que debemos tener en cuenta la cantidad de estos que desarrollemos, ya que si por cada pequeña implementación necesitamos comprobar que todos los test funcionan, estaríamos más tiempo esperando por la ejecución de los tests que escribiendo código, lo que repercutirá en un malgasto de tiempo y dinero. 2.8

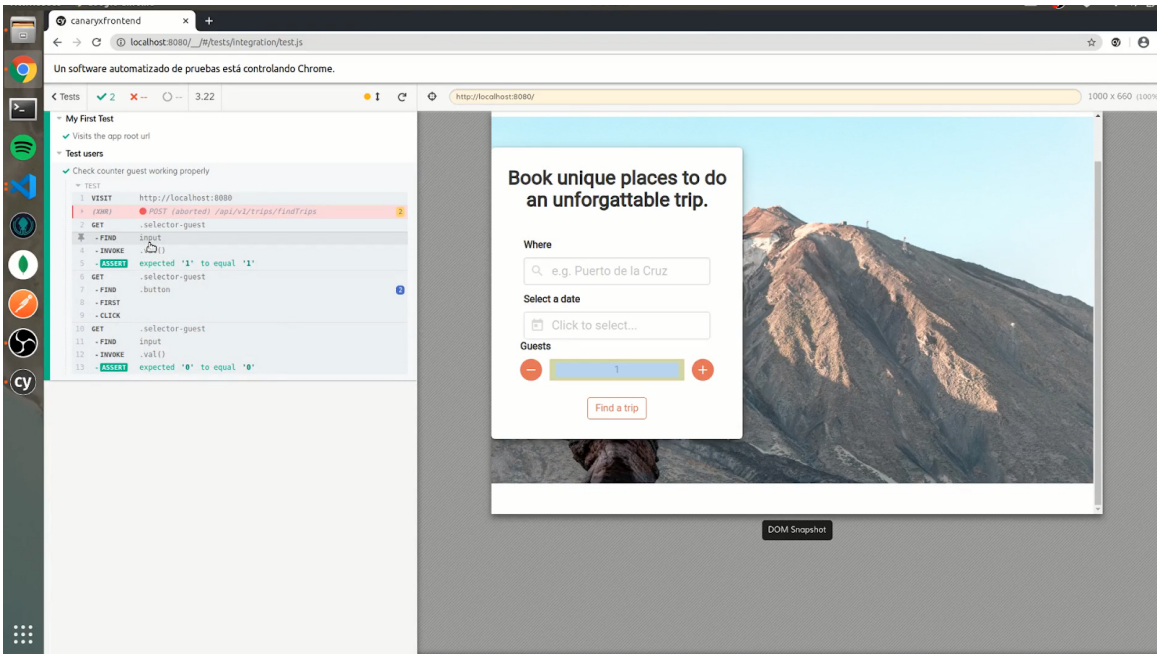


Figura 2.8: Test E2E funcionando

Capítulo 3

Backend

El **backend** [20] de una aplicación web es entendido por la lógica necesaria que es ejecutado en otro lugar que no sea en el ordenador del cliente, normalmente un servidor. Es decir, toda la lógica de la aplicación o también llamada lógica de negocio, la cual no es accesible directamente por el usuario final es la que se ejecuta en dicho lugar. Además, es la encargada de realizar las conexiones a bases de datos, y de entender cómo un ordenador cliente hace una petición a este servicio. El diseño backend es un elemento crucial dentro de una aplicación web, ya que un fallo en esta parte podría significar en una mala experiencia de usuario, problemas de velocidad de carga por parte del usuario e incluso un fallo que haga perder toda la información de los clientes. Por lo que, aunque sea transparente al usuario final, el cliente es directamente afectado por esta lógica, y es uno de los apartados más importantes a tener en cuenta dentro del desarrollo de aplicaciones webs.

Para llevar a cabo el desarrollo de este apartado se ha tenido en cuenta muchos factores para poder aplicar una buena arquitectura de la aplicación.

La arquitectura de una aplicación o servicio es la forma que tenemos de distinguir una serie de capas o partes de una aplicación para poder separar el código dependiendo de su funcionalidad, para facilitarnos el desarrollo y evitar tener malas prácticas. A día de hoy se han popularizado el empleo de varios tipos de arquitecturas que se pueden utilizar, como por ejemplo, la **arquitectura MVVM**[7], la arquitectura hexagonal[21], que se ha popularizado entre la comunidad y la arquitectura por capas entre otras muchas. La arquitectura utilizada para el desarrollo de nuestro backend ha sido la MVC (Modelo-Vista-Controlador) ya que es una arquitectura que nos ayuda a tener una estructura simple y bien definida, pudiendo acotar las funcionalidades de cada uno de nuestros métodos. Además es simple de entender y de utilizar. Esta arquitectura consiste en, como bien dice su nombre, diferenciar tres grandes partes de nuestra lógica, la **vista**, **el controlador** y **el modelo**. La vista consiste en la parte que es vista por el usuario, esta parte se ha llamado así ya que tradicionalmente lo que se renderizaba era una vista html en nuestro backend para enviarlo al cliente. Sin embargo hoy en día esta parte está

separada de la lógica backend, aunque podemos seguir entendiendo esta parte de la vista como la parte que es consultada por el cliente, es decir los endpoint accesibles de nuestra aplicación. Un **endpoint** es un punto de entrada a una api y así actualizar nuestro modelo de datos de nuestra aplicación frontend siguiendo un modelo MVVM. **MVVM** es un patrón de diseño que se utiliza en los frameworks de JS para que la vista pueda cambiar nuestro modelos de datos y viceversa. Por lo tanto, toda la lógica de nuestro endpoint, es la encargada generar una respuesta al cliente para que esta sea pintada en la vista del cliente. En nuestro caso el formato de la respuesta es en formato **JSON**. El controlador es la parte de nuestro código en donde se encuentra toda nuestra lógica de negocio. Él sabe quién tiene permiso para acceder, que se debe hacer con esos datos, donde tiene que ir a buscar los que faltan y que modelo utilizar para recuperarlos. Con todo ello genera una respuesta para devolverla a la vista para poder ser enviada al cliente.

Por último tenemos al modelo, que es el encargado de gestionar la parte de los datos. En casi todas las aplicaciones webs se necesitan guardar datos de los usuarios, y lo más común es almacenar estos datos en una base de datos. Por lo tanto, nuestro modelo es el encargado de entender ese tipo de dato almacenado en la base de datos, conectarse a ella y hacer las consultas necesarias para poder obtener los datos que le han sido pedidos por el controlador. El modelo es la lógica que conoce cómo es el modelo de datos de la aplicación, haciendo invisible esta implementación al controlador. Gracias a esta capa de abstracción, el tipo de base de datos, la manera de guardar y obtener esos datos y manipularlos estaría totalmente desacoplada de nuestra lógica de negocios.

3.1. Tecnologías

Para codificar nuestro código backend se ha utilizado desde hace años lenguajes tales como PHP, JAVA y ASP.NET entre otros, aunque recientemente se han incorporado a estos tres lenguajes otros muy válidos como python o ruby. Estos últimos son lenguajes de scripting interpretados y de alto nivel no tipados que se están utilizando para programar la lógica de negocio de muchas aplicaciones. Sin embargo, lo novedoso es que hay una pequeña parte de la comunidad que están creando este código fuente en un lenguaje de programación llamado javascript. Sí, has leído bien, el mismo que se usa en el lado del cliente o sea en frontend. Muchas personas de la comunidad de desarrollo rehúsan de utilizar este lenguaje para escribir un código tan complejo e importante como este, argumentando que no es un lenguaje tan estable como otros anteriormente mencionados o que tiene algunas incongruencias. Además es un lenguaje que no fue diseñado pensando en conexiones a bases de datos ni que tenga un tipado fuerte, realmente Javascript es un lenguaje no tipado. Sin embargo, han salido muchas adaptaciones de este y librerías que han hecho posible el uso de este lenguaje en backend. **NodeJS** es el culpable. [9]

NodeJS es un intérprete de javascript para el lado del servidor que utiliza el motor de interpretación V8 de google, el mismo motor que se usa en el navegador Google Chrome.

Las peculiaridades del lenguaje javascript es que es un lenguaje orientado a evento y no bloqueante por lo que si el servicio necesita hacer una lectura a disco, este hilo quedaría libre para poder atender a otro usuario. Mientras que el lenguajes como JAVA o PHP las conexiones con lo servidores crearán un nuevo hilo de ejecución y reservan memoria para ese hilo, en NodeJS se gestiona de manera completamente diferente. NodeJS, es capaz recibir esas peticiones y tratarlas como eventos que ocurren, ya que están ejecutando código javascript. Como consecuencia de todo lo anterior, un servidor que ejecuta NodeJS en teoría debería poder atender muchas más peticiones por segundo de lo que cualquier otro podría.

Para poder solucionar uno de los problemas anteriormente mencionados de javascript, el del no tipado, podemos hacer uso de librerías externas que nos ayudaría en el desarrollo. Uno de los superset más comunes para esto es typeScript.

Typescript,[10] es un lenguaje de programación desarrollado por Microsoft, aunque realmente es un superset de Javascript. Un **superset** es un lenguaje escrito sobre otro lenguaje para mejorar ciertas características del primero. Por lo tanto, todo el código escrito en typescript realmente debe pasar por un proceso denominado **trasnpilación** (que viene a ser una traducción fuente a fuente), para convertirlo en código javascript.

Usamos Typescript porque nos da la capacidad de asignar un tipo de dato a variables en javascript, y en proyectos de gran envergadura esto nos puede ayudar a la hora de identificar cada uno de los componentes de nuestra aplicación. Además, es una herramienta muy útil a la hora de autocompletar nuestro código, ya que sabiendo el tipo de objeto que es, nuestro editor o IDE puede sugerirnos qué propiedad queremos escribir haciendo que trabaje por nosotros. Hay que recalcar que es un tipado a nivel de código, es decir, es un tipado orientativo para que el programador sepa de qué tipo es una variable. Realmente este tipado no se mantiene en tiempo de ejecución, ya que se está ejecutando código javascript, y este no es fuertemente tipado. Sin embargo, supone una gran ventaja y recomendaría a cualquier persona su uso frente a javascript nativo o también conocido como javascript Vanilla.

3.2. Express, un framework de Node para backend

Dentro de los frameworks de desarrollo de backend para javascript se ha elegido Express por diversos motivos. Es el framework más maduro que existe entre sus competidores, además de tener una amplia comunidad de desarrolladores encargada de su actualización y mantenimiento y es uno de los más utilizados en

el mercado laboral.

Express ha sido clave para este proyecto para el manejo de las rutas de nuestro endpoint, ya que ha ayudado a mantener la lógica de nuestra aplicación lo más separada y sencilla posible. Se ha utilizado también para crear nuestro servidor web el cual se está ejecutando constantemente a la espera de recibir peticiones.

3.3. Arquitectura. Modelo basado en Microservicios

Para realizar la arquitectura de nuestra aplicación hay muchas opciones en el que podemos optar, sin embargo las que han triunfado dentro de la industria han sido tres a muy grosso modo. Realmente existen mucha más opciones, pero son variantes de las que vamos a comentar o no han entrado con tanta fuerza dentro de la comunidad de desarrollo. Tradicionalmente, todas las aplicaciones web seguían un diseño monolítico, es decir toda la lógica se ejecutaba en un mismo proceso, en la misma máquina, ya que ciertamente era una ventaja para tener todo centralizado en el mismo lugar, menos complejo de programar y mantener. Además, como tradicionalmente se generaban los HTML de la aplicación en la parte del backend, esta arquitectura era la idónea para este tipo de aplicaciones. Sin embargo, en los últimos años y con la finalidad de hacer sus aplicaciones más resilientes a fallos, se ha optado por una arquitectura de microservicios. La arquitectura de microservicios, es uno de los modelos de aplicaciones que más han triunfado en la actualidad. Como bien dice su nombre, consiste en dividir el backend de nuestra aplicación en servicios más pequeños que se ejecutan de manera independiente, pero que pueden estar coordinados entre ellos si fuera necesario. Gracias a esto presentan una alta resiliencia a fallos, ya que aunque un microservicio no este disponible en ese momento, la aplicación podría seguir funcionando.

Por último, el modelo serverless es una arquitectura que no necesita de tener un servidor con nuestra aplicación corriendo en todo momento, sino que se utiliza un servicio como AWS lambda que ejecuta nuestro código cuando es necesario y que se elimina una vez terminada su acción. Realmente esta es la forma más moderna, sencilla y económica de administrar nuestro backend de la aplicación ya que no nos tenemos que preocupar de crear máquinas virtuales, de la configuración, de la escalabilidad, etc. Todos estos problemas están superados por el tipo de metodología que se utiliza y por el concepto que esto supone.

Sin embargo, y aunque la arquitectura serverless es la que utilizaría en cualquier otro tipo de aplicación, para este TFG hemos usado una arquitectura de microservicios. El enfoque que se le ha querido dar a este proyecto es otro. Se trata de profundizar, y mejorar también en la parte de servidores y de cómo es el despliegue de una aplicación y así aprender más conceptos que nos podrán ser útiles dentro del mercado laboral.

3.3.1. API

Una de las parte fundamental de nuestra aplicación es el API donde se encuentra toda la lógica de negocios de la parte de la usuarios y excursiones. En este API se encuentran los endpoints principales de donde nuestra aplicación frontend se va a nutrir de los datos necesarios.

3.3.2. Servicio de fotografías

El otro servicio del que dispone la aplicación es un servicio que se utiliza para el almacenamiento de fotografías. De esta manera quitamos la carga extra CPU de nuestra API de usuarios y excursiones para una funcionalidad que no es principal. Por lo que toda la lógica de almacenar las fotografías, crear la carpeta del usuario necesaria para alojar sus fotografías, y la de cada excursión, ya que esta funcionalidad la hemos gestionado nosotros. Más adelante en este documento se explicará como se ha realizado.

3.4. Bases de datos

La base de es otro de los puntos críticos de toda aplicación web, ya que es el lugar donde se almacenan la gran cantidad de datos que utiliza nuestra aplicación. Además estos datos pueden ser muy valiosos, no solo para el propio funcionamiento de la aplicación, sino para hacer estudios de mercados, campañas de marketing, utilizarlos para estudiar un comportamiento social, etc. Existen varias alternativas a la hora de elegir una base de datos, aunque en el mundo corporativo han triunfado dos opciones por encima de otras. Estas opciones son las bases de datos relacional o las SQL y las bases de datos no relacional o noSQL. Tradicionalmente se han utilizado las bases de datos SQL para el almacenamiento de los datos, y realmente tiene muchas ventajas, ya que son eficientes porque no necesitan de la duplicidad de datos, tienen un esquema claro que no deja que pueda haber fallo de inconsistencias. Los datos están relacionados entre sí, por lo que su manejo a la hora de borrarlo o actualizar datos que dependen unos de otros son más sencillos, pero tienen una limitación, y es su velocidad de lectura. Cuando las tablas de una base de datos van aumentando su tamaño y tenemos que cruzar datos, es decir, hacer consultas donde el resultado sea la unión de dos o más tablas, la consulta se puede demorar mucho tiempo. El sistema de gestión de base de datos necesita hacer una gran cantidad de cálculos, por lo que para almacenar millones de datos puede que no sea la mejor opción. Por esta misma razón aparecieron las bases de datos noSQL. Las bases de datos no SQL no tienen ningún tipo de correlación entre los datos, no pueden actualizarse datos en cascada, no pueden relacionarse varias colecciones (son las tablas de las bases de datos no relacionales), su estructura no está definida del todo ya que realmente son estructuras JSON, objetos o de nodos. Sin embargo, son extremadamente buenas para una cosa en concreto,

almacenar grandes volúmenes de datos y recuperarlos de manera muy rápida. Por ello, para aplicaciones que manejan un gran volumen de datos que no tienen que estar relacionados entre sí son un alternativa perfecta. Por lo que se ha elegido usar una base de datos de este tipo. Existen varias alternativas de bases de datos noSQL, como por ejemplo cassandra, o redis, pero se ha optado por mongoDB.

3.4.1. MongoDb

MongoDB es una las bases de datos noSQL más fáciles de utilizar y muy recomendada para aplicaciones con alta cantidad de datos, sobre todo por su velocidad de lectura, ya que sus operaciones están escritas en el lenguaje de programación C++, son bastante rápidas y eficientes.

Otra de las opciones tomada por algunas empresas del sector es la de externalizar el servicio de la bases de datos a otra empresa que se dedique exclusivamente a eso. Los datos es una de las partes más importantes y se deben conservar pase lo que pase, por lo que se deben de dar unas condiciones de seguridad que no todas las empresas pueden hacer frente económicamente hablando. Por tanto se ha tomado la decisión de externalizar la bases de datos en el servicio de mongo Atlas.

Mongo Atlas es un servicio que permite gestionar una base de datos de manera remota, con muchas facilidades gracias a su UI muy completa e intuitiva que hace que se puedan modificar, crear y borrar colecciones y usuarios de una manera muy sencilla. Además, dispone de un servicio de replicación por lo que se puede crear en el mismo cluster todas las réplicas que se deseen. En nuestro caso hemos creado 3 réplicas, una que actúa como master y dos que actúan como esclavos o secondary, que empiezan a recibir peticiones cuando la réplica master empieza a aumentar su uso. De este modo se asegura una alta disponibilidad. Se ha optado por una estructura que se basa en dos colecciones principales, una de usuarios y otra de excursiones. Cuando se necesitan relacionar los datos entre sí, como por ejemplo, saber cuando un usuario compra una excursión se hace una copia de su email dentro de los usuarios asistentes a la excursión por lo que existe una duplicidad de los datos pero es necesaria justamente para relacionar ambas tablas entre sí aunque no es lo habitual.3.1

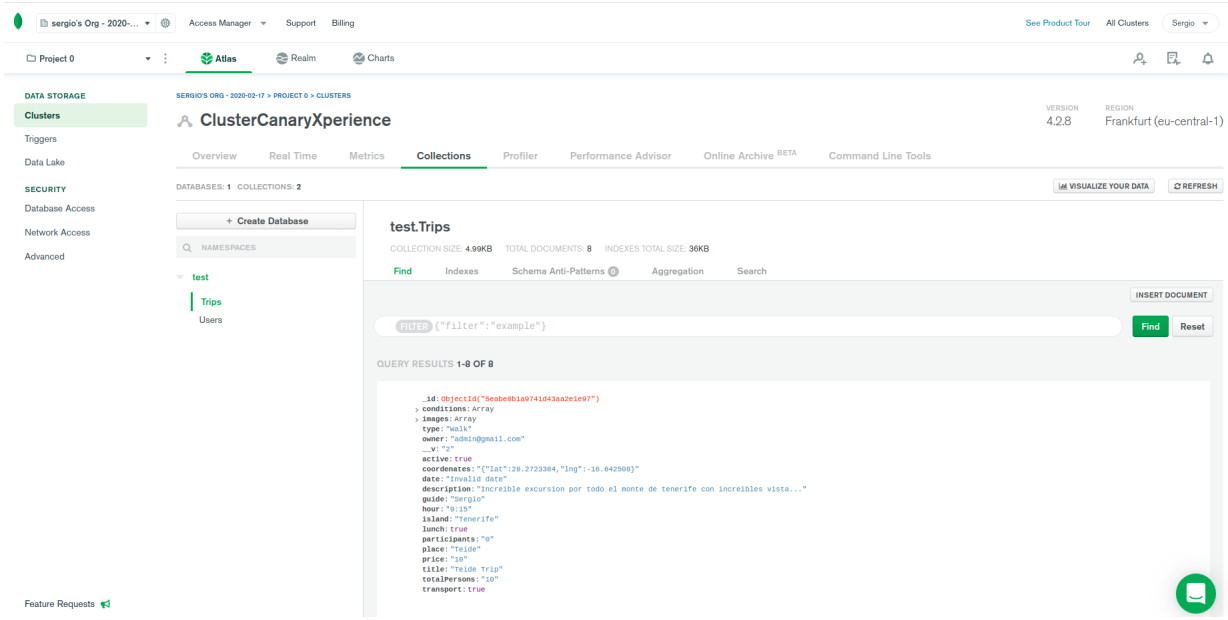


Figura 3.1: Cluster de Mongo Atlas

Capítulo 4

Despliegue

El despliegue de una aplicación es el conjunto de acciones y configuración que debemos de hacer para que nuestra **aplicación** sea accesible a todos los usuarios **a través de internet**.

Tradicionalmente esta parte era mucho más costosa en cuanto a dinero y esfuerzo, ya que necesitabas de un ordenador bastante potente, configurar tu servicio de internet, para que tu ordenador sea accesible desde internet, tener una conexión a internet muy alta, tener alojado tu servidores en un lugar seguro para que sea resiliente a catástrofes naturales, etc. Todos estos problemas, hacian que el coste de tener una aplicación en internet fuera bastante alto. Sin embargo, en estos últimos años ha cambiado bastante el panorama. Han nacido plataformas como AWS, Google Cloud o DigitalOcean, que nos facilita enormemente la vida.[17]

El modelo de negocio de estas empresas consiste utilizar una granja de servidores virtualizados propios que alquilan por una cierta cantidad de precio. Su modelo de cobro puede ser por tiempo de utilización es decir, por el tiempo que se tiene utilizar una ip pública para tu servicio, o que siga el modelo de consumo, a mayor consumo de recursos mayor es el corte. Este tipo de infraestructuras nos eliminan mucho de los problemas que teníamos anteriormente, y nos hacen que tener una aplicación en internet sea más sencillo y económico que nunca.

Estos proveedores de infraestructura, integran en ella también múltiples servicios de software como **kubernetes** en el caso de google y digitalocean o **lambda**[14] en el caso de AWS. Además, ofrecen una plataforma con muchas funcionalidades para facilitar la creación de máquinas virtuales con diferente cantidad de recursos, crear snapshots y backups de forma muy sencilla, gestionar una base de datos, etc. Por lo que realmente son infraestructura, software y plataforma como servicios, IASS, SASS y PASS. 4.1

Para llevar a cabo esta parte de nuestro TFG se ha elegido el servicio de DigitalOcean, ya que aunque no es el más conocido ni el más utilizado del momento, es uno de lo más completos en cuanto a funcionalidades se refiere, tiene una in-

Figura 4.1: Esquema de Servicios

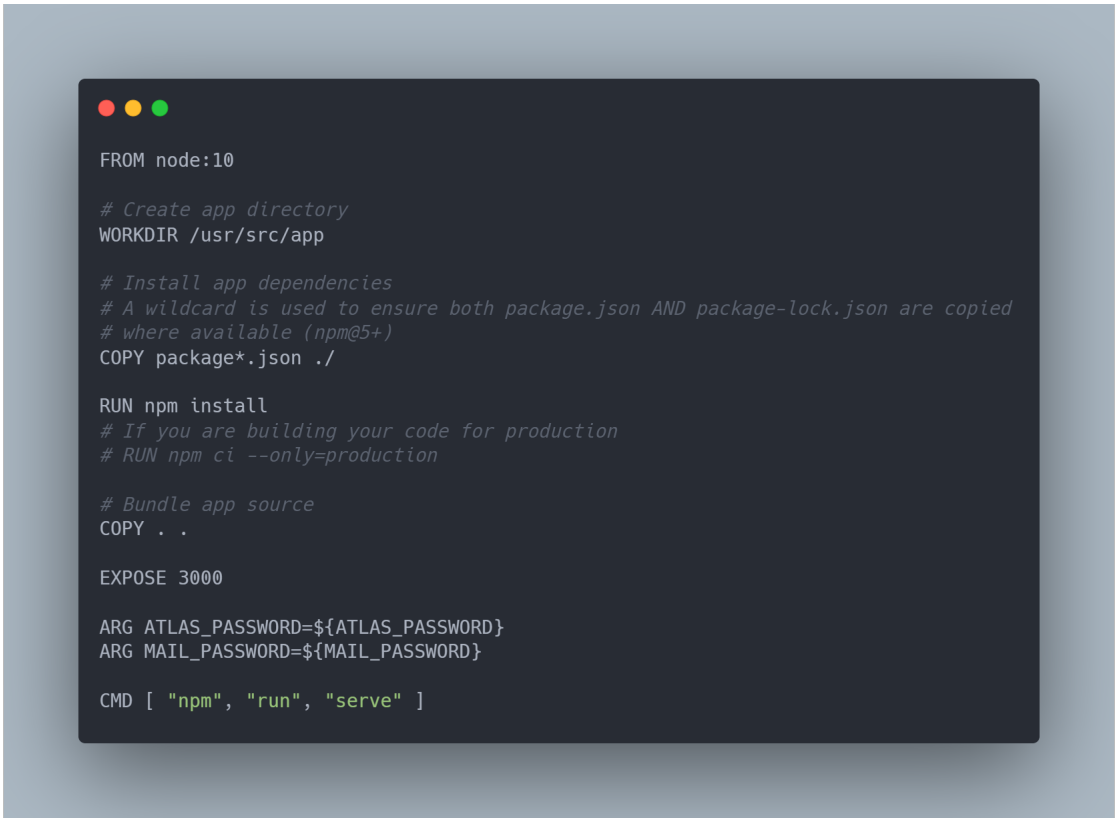
terfaz de uso bastante sencilla y, sobre todo, es bastante más económico que su competencia.

4.1. Docker

Muchas veces en el desarrollo de software nos hemos preguntado por qué funciona una parte de una aplicación en nuestro ordenador local cuando la estamos desarrollando y cuando vamos a utilizarlo en otra máquina funciona de manera errática o diferente. No tiene realmente mucho sentido. ¿Es culpa del procesador de la otra máquina? ¿culpa del sistema operativo?. El verdadero problema de esto son las versiones de las librerías o de los compiladores que usamos en esa máquina. Puede que en el entorno de desarrollo local estemos utilizando, por ejemplo, Node en su versión 12, pero cuando pasamos ese código a otra máquina que va a ejecutar con una versión de Node 10, veremos que empiezan los problemas. No compila, errores de sintaxis, funcionamiento errático, entre muchos otros problemas. Esto ha sido siempre un dolor de cabeza durante mucho tiempo para los desarrolladores de software. No había ninguna solución definitiva sino la de tener mucho cuidado y mantener nuestro entorno de producción lo más en consonancia posible con nuestro entorno desarrollo. En prácticamente cualquier aplicación que falla cuando se traslada de un entorno a otro en el que no podemos controlar cuál es su estado, es un problema de versiones.

Docker[2] es un software usado para crear máquinas virtuales llamadas contenedores con la premisa que sean pequeñas, de pocos recursos y portables entre distintos sistemas operativos y entornos para ejecutar software dentro de ellas. Gracias a estos contenedores, se facilita el control de las versiones de las librerías de las que depende el software facilitando así los despliegues.

Docker nos abstrae de este tipo de errores, ya que realmente todo nuestro código se ejecuta dentro de esa pequeña máquina virtual, que es la misma que se traslada a producción por lo que en ambos entornos funcionará de la misma manera. Realmente se está ejecutando justamente lo mismo. Mismo código, con mismas versiones de las librerías etc. 4.2



```
FROM node:10

# Create app directory
WORKDIR /usr/src/app

# Install app dependencies
# A wildcard is used to ensure both package.json AND package-lock.json are copied
# where available (npm@5+)
COPY package*.json ./

RUN npm install
# If you are building your code for production
# RUN npm ci --only=production

# Bundle app source
COPY . .

EXPOSE 3000

ARG ATLAS_PASSWORD=${ATLAS_PASSWORD}
ARG MAIL_PASSWORD=${MAIL_PASSWORD}

CMD [ "npm", "run", "serve" ]
```

Figura 4.2: DockerFile

Además, docker se puede utilizar como nodos de kubernetes, y esto es clave. Es muy eficiente, ya que, Docker es capaz de compartir los recurso entre las diferentes maquinas virtuales, haciendo que las imágenes de Docker compartan estas dependencias para no tenerlas duplicadas, por lo que su espacio es mucho menor a una máquina virtual convencional. 4.3

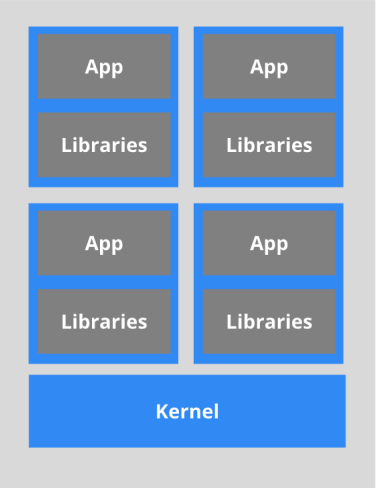


Figura 4.3: Arquitectura Docker

4.2. Kubernetes

Desde que se crearon las aplicaciones web los desarrolladores siempre se han preguntado lo mismo, ¿Cómo podemos hacer que esta aplicación no se caiga cuando se conecte una gran cantidad de usuarios a la vez? Escalar la aplicación es la clave. Tradicionalmente se han utilizado varias técnicas para escalar una aplicación. Se han utilizado métodos manuales, como aumentar la potencia de las máquinas que están ejecutando esa aplicación, por lo que serán más potentes y podrán atender las peticiones de una manera mucha más rápida, encender más máquinas donde poner la aplicación a funcionar y repartir la carga con un servidor web proxy y demás soluciones que se han ido utilizando a lo largo de los años. Sin embargo, en los últimos 5 años, la forma de trabajar ha cambiado ya que ha aparecido Kubernetes.

Kubernetes,[1] como describe en su documentación oficial, es un orquestador de contenedores de Docker. Es decir, es el encargado de que los contenedores puedan interactuar entres sí y que puedan existir varios a la vez. Algunos miembros de la comunidad, afirman que kubernetes es todo lo que le falta a docker para poder correr en producción. Además kubernetes sigue el enfoque **Infraestructura como código**. Todo lo que se pueda desplegar puede ser escrito en un fichero de configuración en de tipo .yaml mejorando así el control del estado de la aplicación y ayudándonos a usar un control de versiones también para nuestra parte de deploy.

Kubernetes está jerarquizado en varias capas de abstracción. También, es el encargado de automatizar todas las acciones que se van a realizar en nuestra aplicación referido siempre a la parte de infraestructura.

La capa o unidad más pequeña se llama **pod**. Un pod es realmente una imagen o más imágenes de docker ejecutándose. Es la unidad mínima que considera kubernetes. En la siguiente escala encontraríamos lo que kubernetes llama un **de-**

ployment que es el que contiene las réplicas de nuestros pods y es el encargado de decidir cuántas réplicas se van a ejecutar de nuestro pod. Y en la capa más externa estarían nuestros **services**, que son los encargados de conectar nuestra aplicación con el exterior y enrutar tráfico hasta los diferenets deployments.

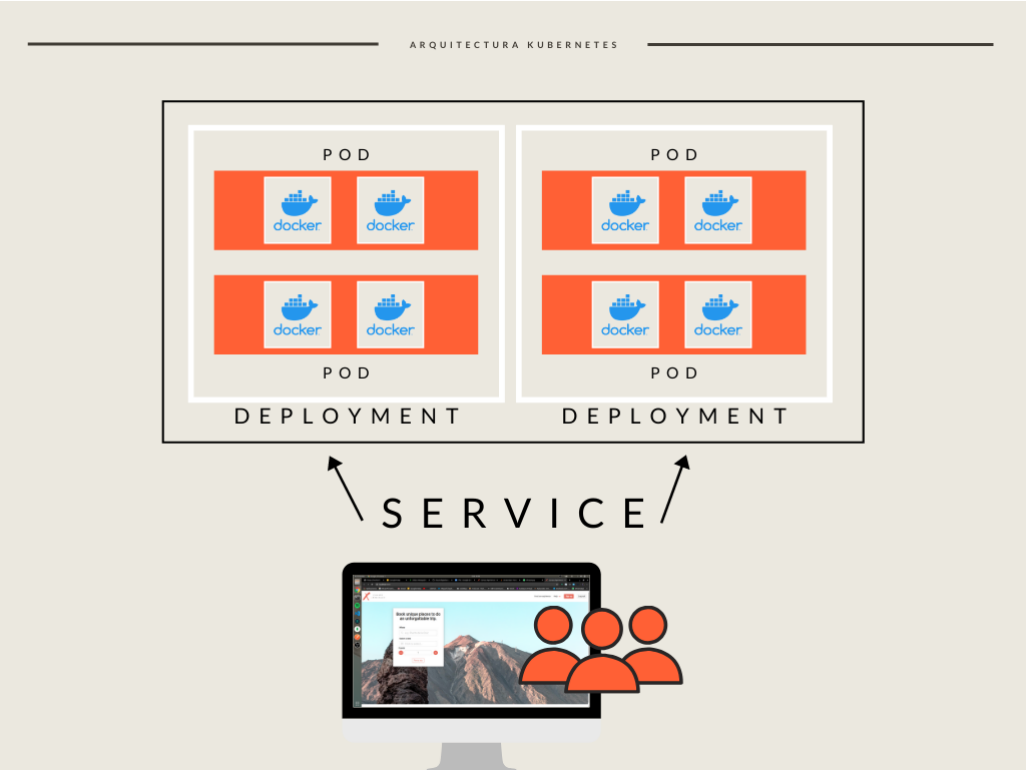


Figura 4.4: Esquema Kubernetes

En nuestro caso particular hemos distribuido en nuestra aplicación en un único pod base, con la imagen de docker de nuestra aplicación de backend. Además, se dispone de un deployment encargado de replicar este contenedores dependiendo de unos casos de métricas que se den. Y por último,el servicio que es el encargado en enrutar el tráfico desde internet al deployment.4.5

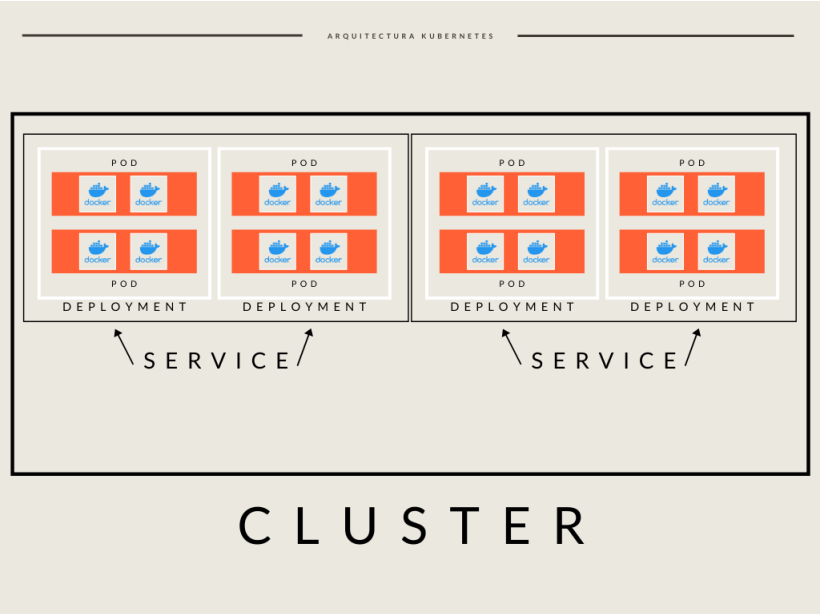


Figura 4.5: Arquitectura de un nodo de Kubernetes

4.3. HPA

HPA o Horizontal Pod Autoescaler es un servicio de kubernetes que hemos utilizado para escalar nuestras réplicas dependiendo de un unas ciertas métricas. Estas métricas son obtenidas gracias a que se ha instalado un servicio de métricas de kubernetes el cual se puede descargar desde su documentación oficial.[1] Gracias a este servicio podemos obtener varios datos como, el uso de CPU, el consumo de memoria, el número de peticiones por segundo etc. 4.6

Figura 4.6: Manifiesto HPA

Por lo que hemos usado este HPA para escalar nuestra aplicación de forma automática cuando el uso de la CPU media de los pods de la aplicación superan el 50 por ciento. Además de escalar, también es capaz de desescalar, es decir, es capaz eliminar réplicas cuando ya la utilizaciñon de nuestros pods baja y ya no es necesaria esta cantidad de réplicas, disminuyendo así los recursos utilizados de la aplicación. 4.7

Tras dejar de aplicar carga y esperar unos minutos vemos como nuestro HPA redujo de nuevo las réplicas a uno como era inicialmente, ya que al no tener carga no necesita de esa réplicas extras. El número de réplicas dependerá de la potencia que tengamos contratada de nuestro cluster de kubernetes. En este caso particular se ha comprobado que el número máximo de réplicas que soporta nuestro cluster de kubernetes sin empezar a tener problemas de rendimiento es de 10 réplicas. Aunque hay que mencionar que, se trata del modelo de cluster de kubernetes más económico que dispone digital ocean, por lo que en una aplicación con más recursos podría escalar todo lo que fuera necesario.

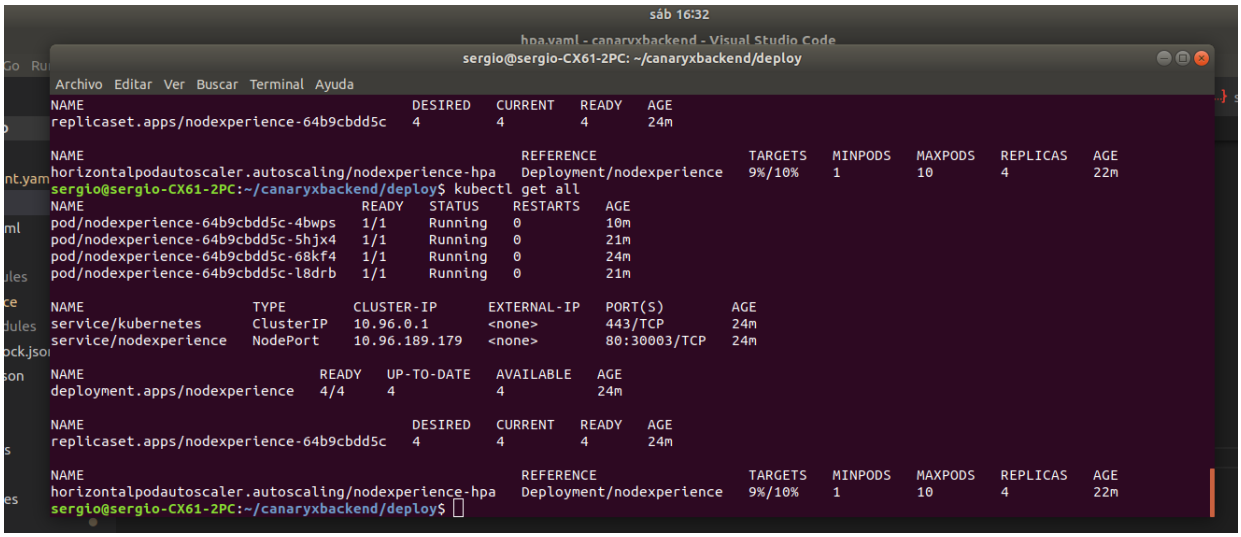


Figura 4.7: HPA con carga

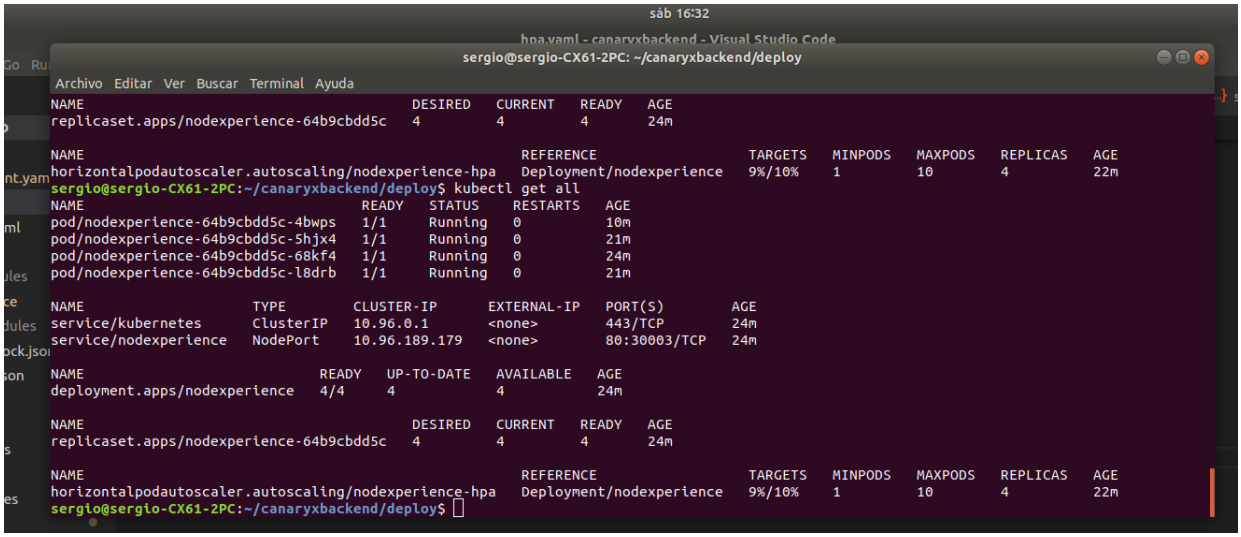


Figura 4.8: HPA sin carga

Postman es que es un cliente para hacer peticiones http y puede configurarse para que haga un cierto número de peticiones.

En el siguiente gráfico vemos como ha sido la carga de nuestro cluster de kubernetes. Comentar que las pruebas han sido realizadas con un sólo ordenador, ya que no se disponían de más equipos en el momento de realizarlas. Hemos configurado tres usuarios de postman realizando peticiones a la vez, con un retardo entre petición de 60 ms, por lo que en total hemos realizado un máximo de 3000 request por segundo (eje x de nuestro gráfico)4.9 y vemos como ha sido el aumento del número de réplicas (eje y) consiguiendo el máximo de 4 réplicas de las 10 disponibles.

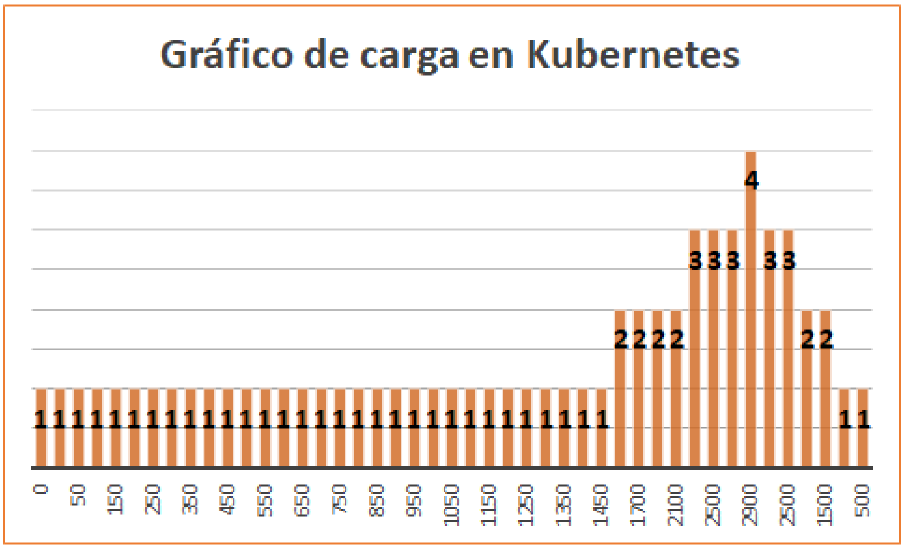


Figura 4.9: Gráfico de uso de Kubernetes

4.4. TRAVIS Y CI/CD

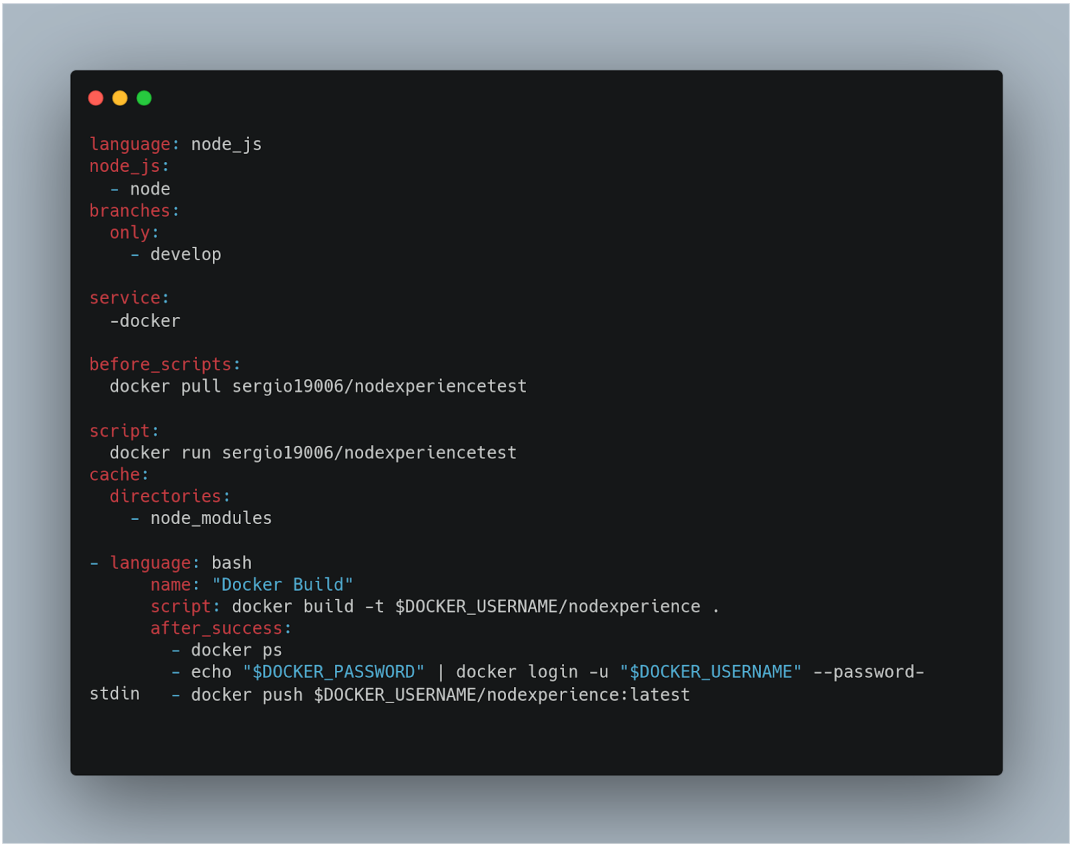
Para asegurarnos de que de que las implementaciones de los requerimientos funcionan de manera correcta en nuestro entorno de pruebas, se ha utilizado Travis. Travis [22] es un sistema de integración continua que se funciona junto a github y es capaz de crear nuestras pipelines cuando se hace un push o se mezcla un pull request en una ramadeterminada.

La configuración de Travis puede realizarse de manera muy sencilla a través de un archivo de extensión YAML. Además, se ha integrado docker en nuestro entorno de CI, ya que es la manera en la que se ejecuta nuestra aplicación en producción junto con kubernetes.

El workflow utilizado es el siguiente: El desarrollo se centraba sobre la rama de develop, donde utilizabamos nuestra instancia de docker en local para programar todos nuestras funcionalidades y crear nuestros test. Una vez impleentada la nueva funcionalidad y con el test debidamente probado, hacemos push a la rama de origin/develop, es decir en el repositorio de github. Justo cuando se hace este paso, se arranca en Travis lo que se conoce como una pipeline. Una **pipeline** es un conjunto de pasos que se realizan para conseguir un objetivo en concreto. En este proceso se realiza la configuración, tests e instalación del nuevo código en los entornos de prueba.

En el caso que nos ocupa, el proceso pasaba por ejecutar en nuestra imagen de docker todo los test de nuestra aplicación. Si los test pasan de manera satisfactoria se procede a hacer el build de esa imagen de docker desde el servicio de CI de Travis y hacer el push a nuestro servicio de Docker Hub donde tenemos almcenadas las imágenes de docker de nuestra aplicación. Además, el código que se ha

sido subido a la rama de develop pasa automáticamente a la rama master donde Travis es el encargado de hacer el mergeo de dichas ramas. 4.10



```
language: node_js
node_js:
  - node
branches:
  only:
    - develop

service:
  -docker

before_scripts:
  docker pull sergio19006/nodexperiencetest

script:
  docker run sergio19006/nodexperiencetest
cache:
  directories:
    - node_modules

- language: bash
      name: "Docker Build"
      script: docker build -t $DOCKER_USERNAME/nodexperience .
      after_success:
        - docker ps
        - echo "$DOCKER_PASSWORD" | docker login -u "$DOCKER_USERNAME" --password-
stdin   - docker push $DOCKER_USERNAME/nodexperience:latest
```

Figura 4.10: Travis

Sin embargo, una de las limitaciones de este TFG fue la puesta en funcionamieto del servicio de CD es decir, despliegue continuo. Los servicios elegidos no han sido los adecuados, ya que no se ha conseguido conectar travis CI a nuestro nodo de kubernetes de digital ocean de una manera sencilla. Cuando hemos creado nuestro Service Account de kubernetes se ha rechazado la petición por no confiar en digital ocean. Aún se está investigando que tipo de configuración hace falta para que dichos servicios funcionen de manera correcta, pero a día de hoy no se ha podido encontrar una solución a este problema. Se hubiera logrado realizar todo de manera más sencilla utilizando el servicio de kubernetes de Google, el GKE o utilizando otro servicio de integración continua como CircleCI.

Realmente lo que se debe hacer manual, es conectarnos a nuestro cluster de kubernetes de DO y aplicar de nuevo el manifiesto de deployment. Con esto conseguimos que descargue y se ponga en funcionamiento la última versión de nuestra imagen de docker nuestro código actualizado creado por Travis CI. Es sólo un paso muy sencillo pero que por inexperiencia y por falta de tiempo ha sido un problema que no se ha podido abordar para tener un CI/CD completo de principio a fin. Sin embargo, se ha de tomar más como un aprendizaje para el futuro y poder elegir los servicios a utilizar en base a este tipo de limitaciones.4.11

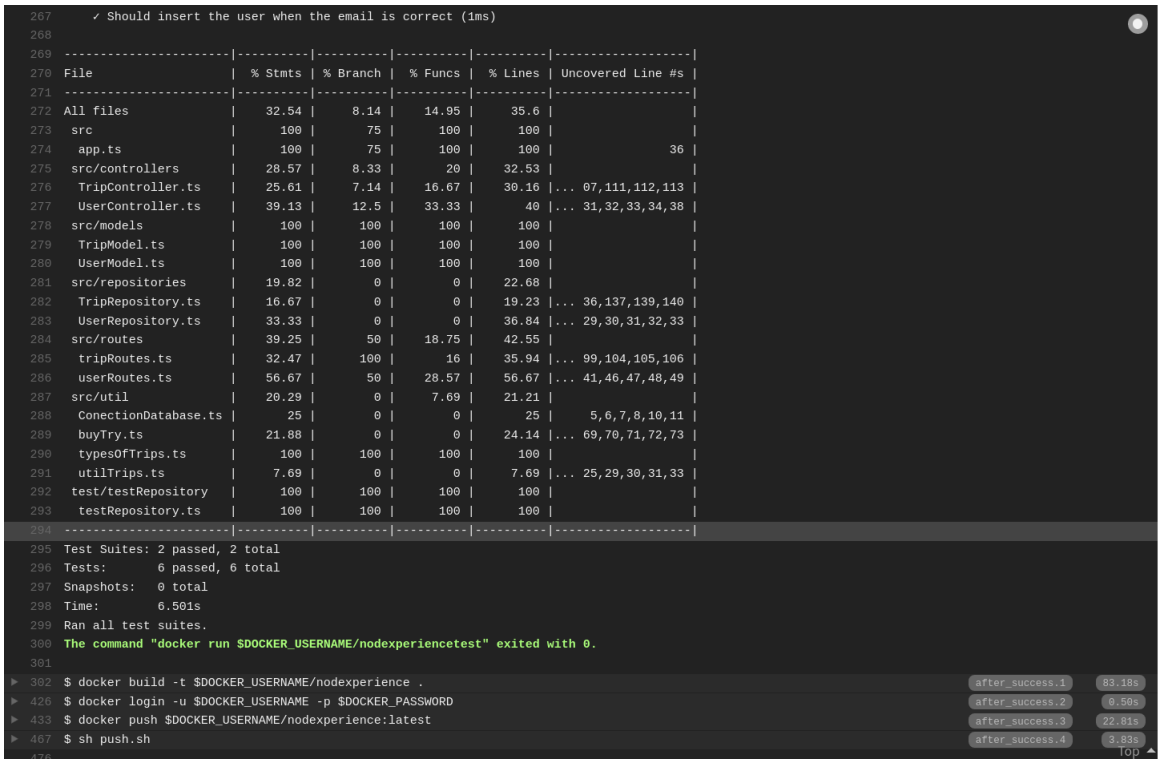


Figura 4.11: Travis en funcionamiento

4.5. NGINX

Nginx es un servidor web que se usa como punto de entrada de cualquier aplicación web. Hemos utilizado este servicio para devolver nuestros estáticos, ya que webpack al empaquetarlo y exportarlo pueden ser servidos directamente.

Por lo que es una buena alternativa a el tradicionalmente usado apache, ya que es mucho más ligero y eficiente. Gracias a Nginx, hemos conseguido crear nuestro propio CDN por para las imágenes de las aplicaciones, y así no dependemos de un servicio externo. Además, esto nos ha ayudado a poder aprender y profundizar en la configuración de este tipo de servicios.4.12

```
server{
    server_name _;

    location / {
        root    /data/www/dist;
        index   index.html;
    }
    location /users/ {
        root    /data;
    }
    location /trips/ {
                root    /data;
        }
    location /pdfs/ {
                root    /data;
        }
    location /qr/ {
                root    /data;
        }
}
```

Figura 4.12: Configuración Nginx

4.6. Dominio

Para llevar a cabo la compra de nuestro dominio hemos utilizado el gestor de dominios llamado **NameCheap**, que es el gestor de dominios más económico. Los NS que hemos utilizado son los propios que nos dispone digital ocean dentro de el servicio de ip públicas que dispone, por lo que ha sido muy sencillo de configurar.

Además para crear nuestra caché de los estáticos y aumentar así su velocidad de acceso y eliminar tráfico de nuestro droplet en donde tenemos desplegada nuestra aplicación hemos utilizado un **CDN** llamado Cloudflare.4.13

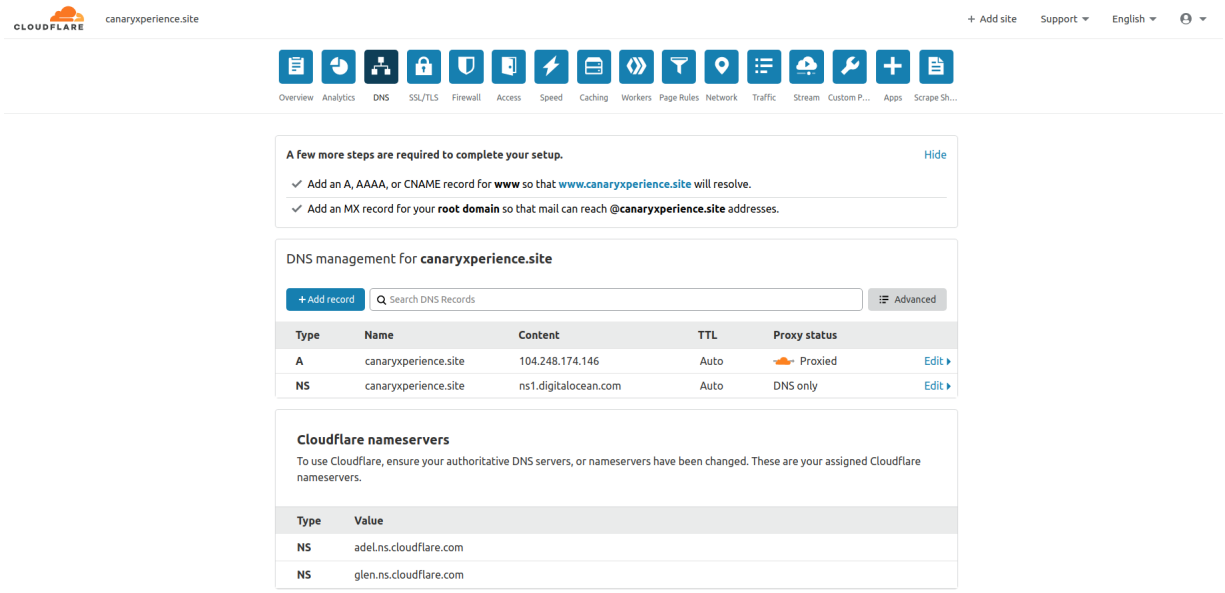


Figura 4.13: Configuración de CloudFare

Capítulo 5

Conclusiones y líneas futuras

En conclusión, el desarrollo de este TFG ha sido una buena práctica para poder seguir avanzando en mi formación. Además se ha conseguido afianzar nuevos conocimientos y profundizar en tecnologías que están usándose actualmente en el mercado laboral. He logrado aprender a ser más autodidacta y autosuficiente y a centrar todos los conocimientos que he ido aprendiendo en las asignaturas de las carrera y ponerlo en conjunto en un proyecto final.

Dentro de la aplicación se pueden diferenciar dos usuarios, los usuarios de tipo cliente y los usuarios que son de tipo empresa, que son a los que se les cobrará por crear una cuenta de este tipo. Una vez creado los usuarios de tipo empresa serán capaces de crear excursiones en la sección llamada “Host an Experiencie”, donde tienen un modo de creación de excursiones con su configuración de varios aspectos como las fotografías, el lugar con una marca configurable desde la página en google maps, precio guía fecha y hora, etc.

Por otra parte, cuando un usuario entra puede acceder a un buscador personalizado por lugar y fecha, que estos son auto rellenados dependiendo de los días y lugares en los que existan excursiones, es decir, si existe una excursión al Teide se rellenará automáticamente. Una vez seleccionado la cantidad de usuarios y la excursión que desea realizar, pasará a la sección de realizar el pago donde se le mostrará un formulario para rellenar los datos de su tarjeta bancaria. Después de ser enviado este formulario satisfactoriamente, le llegará un email personalizado al correo con el que se compró la excursión detallando día, fecha y lugar de la excursión con su número de reserva y un código QR válido.

La aplicación resultante es realmente usable, ya que dispone de muchas funcionalidades diferentes, desde su login y registro, pasando por la compra y creación de excursiones por parte de los usuarios de tipo empresa y hasta el envío de un correo electrónico personalizado al cliente cuando finaliza el proceso de compra.

Como líneas futuras, se podrían añadir muchas más funcionalidades como externalizar el cobro de las excursiones a una pasarela de pagos externas, ya que ahora mismo está únicamente mockeado; permitir crear grupos de amigos para

ponerse en común a la hora de comprar una excursión; incluir un chat privado para cada excursión donde cada uno de los participantes tengan comunicación directa con el guía.

Capítulo 6

Summary and Conclusions

The objective of carrying out this project is to be able to provide a complete, inexpensive and easy-to-use tool for tourism companies. Therefore, the fact of being on the Internet would not be a problem or a limitation. In addition, the development of this web application has served to me as a way of learning, to strengthen my knowledge, and to put in practice many technologies that I have never been used before.

To achive the main objectives that we have set ourselves, we have used a set of leading-edge technologies, and we have tuned them towards a real application. The application implemented could be a good starting point and I think that its development should be continued due to its importance in the tourism sector. What is more, it could be a viable and functional application with an extra knowledge in other areas such as online marketing.

Moreover, this web application would have an impact on improving communication between tourism companies and target users. It could improve sales and the amount of potential customers thanks to the fact of being on the same web. This community could find a company even if they have not searched for it.

Clients are able to look for all the options of excursion offers and experiences on a single page, thus comparing in a simpler way the different services offered by tourism companies without visiting dozens of websites of the different companies.

The resulting project has served to learn much more and enter even more in web development and continue training as a developer.

It has been a very interesting practice in which I have challenged all my knowledge in web development. Furthermore, I have been able to learn technologies that I use to employ every day.

Capítulo 7

Presupuesto

En cuanto al presupuesto se ha creado la siguiente tabla con un desglose aproximado de las actividades que pueden observarse en la siguiente tabla: 7.1

CANARY XPERIENCE

PRESUPUESTO

DESCRIPCIÓN	HORAS	PRECIO/HORA	IMPORTE
Login	4	25,00 €	100,00 €
Registro	4	25,00 €	100,00 €
Creación de excursiones	10	25,00 €	250,00 €
Envío de correos	6	25,00 €	150,00 €
Configuración de entornos	6	25,00 €	150,00 €
Creación de servicio de fotos	5	25,00 €	125,00 €
Diseño	18	25,00 €	450,00 €
Configuración de Kubernetes	10	25,00 €	250,00 €
Entorno Kubernetes	-	N/A	10,00 €
Nombre de dominio	-	N/A	10,00 €
API Google Maps	-	N/A	200,00 €
Realización de API	40	25,00 €	1.000,00 €
Configuración de servicio de fotos	12	25,00 €	300,00 €
Configuración de droplets	10	25,00 €	250,00 €
		Importe total:	3345,00 €
		Importe total + IGIC (7%):	**3579,15 €**

Figura 7.1: Presupuesto

Apéndice A

Ejemplos de código

A.1. Uso del Store para el control de estado

```
1
2  /*****************************************************************************
3  * store/mutations.js
4  *****************************************************************************/
5  export const mutations = {
6    setEmail: (state, email) => {
7      email ? (state.email = email) : (state.email = "");
8    },
9    setToken: (state, token) => {
10     token ? (state.token = token) : (state.token = "");
11   },
12   setTrip: (state, trip) => {
13     trip ? (state.trip = trip) : (state.trip = {});
14   },
15   setQuery: (state, query) => {
16     query ? (state.query = query) : (state.query = {});
17   },
18   setClientTrip: (state, trip) => {
19     trip ? (state.clientTrip = trip) : (state.clientTrip = {});
20   },
21   setCoordenates: (state, coordenates) => {
22     coordenates ?
23         (state.trip.coordenates = coordenates) : (state.trip.coordenates = {});
24   },
25   setNumberOfPersons: (state, numberOfPersons) => {
26     numberOfPersons ?
27         (state.numberOfPersons = numberOfPersons) : (state.numberOfPersons = 0);
28   }
29 };
30
31
```

Listing A.1: Store

A.2. Uso de router vue para enrutar entre las vistas

```
1
2  /*****************************************************************************
3   *
4   * router/index.js
5   *
6   *****************************************************************************/
7  import Vue from 'vue';
8  import VueRouter from 'vue-router';
9  import Explorer from '../views/Explorer';
10 import Login from '../views/Login';
11 import Signup from '../views/Signup';
12 import Trip from '../views/Trip';
13 import Experience from '../views/Experience';
14 import NewTrip from '../views/NewTrip';
15 import Admin from '../views/Admin';
16 import Buy from '../views/Buy';
17
18 Vue.use(VueRouter);
19
20 const routes = [
21   { path: '/', component: Explorer },
22   { path: '/login', component: Login },
23   { path: '/signup', component: Signup },
24   { path: '/trip', component: Trip },
25   { path: '/experience', component: Experience },
26   { path: '/newtrip', component: NewTrip },
27   { path: '/admin', component: Admin },
28   { path: '/buy', component: Buy }
29 ]
30 export default new VueRouter({
31   routes,
32   mode: 'history'
33 });
34
35
```

Listing A.2: Rutas de la aplicación

A.3. Test E2E para comprobar funcionamiento del tripPicker

```
/**********************************************************************************
 *
 * test/E2E.js
 *
 **********************************************************************************/

describe('Test users', () => {
  it('Check counter guest working properly', () => {
    cy.visit('http://localhost:8080')

    cy.get('.selector-guest').find('input').invoke('val')
      .then(value => expect(value).to.be.equal("1"));

    cy.get('.selector-guest').find('.button').first().click();

    cy.get('.selector-guest').find('input').invoke('val')
      .then(value => expect(value).to.be.equal("0"));
  })
})


```

Listing A.3: Test E2E

Bibliografía

[1] Documentación docker. https://kubernetes.io/es/docs/home/.

[2] Documentación kubernetes. https://docs.docker.com/.

[3] Transpilado de javascript con webpack y babel. https://desarrolloweb.com/articulos/transpilado-javascript-webpack.html.

[4] What is kanban? https://www.atlassian.com/agile/kanban.

[5] ¿qué es css? https://es.ryte.com/wiki/CSS.

[6] Bulma framework. https://bulma.io/, 2017.

[7] Patrón mvvm con vue. una joya para javascript. http://buhoprogramador.com/entrada/9/patr%C3%B3n-mvvm-con-vue-una-joya-para-javascript, 2017.

[8] ¿qué es flux? entendiendo su arquitectura. https://carlosazaustre.es/como-funciona-flux, 2018.

[9] M. Abernethy. ¿simplemente, qué es nodejs? https://www.ibm.com/developerworks/ssa/opensource/library/os-nodejs/index.html, 2011.

[10] M. Caceres. ¿qué es typescript? https://devcode.la/blog/que-es-typescript/, 2018.

[11] R. Castaño. ¿qué es diseño ux? https://www.neoland.es/blog/que-es-el-ux-ui-design, 2018.

[12] S. del Pino. Repositorio backend. https://github.com/Sergio19006/canaryxbackend, 2020.

[13] S. del Pino. Repositorio frontend. https://github.com/Sergio19006/canaryxfrontend, 2020.

[14] L. Garcia. ¿qué es una arquitectura serverless? https://.unpocodejava.com/2016/06/22/que-es-una-arquitectura-serverless-y-aws-lambda/, 2016.

[15] B. Huete. Vida de vue. https://benjaminhuete.gitbooks.io/vuejs/chapter1/ciclo-de-vida-de-vue.html, 2017.

[16] J. Barrio Jiménez. Tipos de pruebas de software. lhttps://openwebinarsopenwebinars.net/blog/tipos-de-pruebas-de-software/, 2019.

[17] Microsoft. ¿qué es paas? https://azure.microsoft.com/es-es/overview/what-is-paas/.

[18] A. Pedraza. ¿qué es desarrollo frontend? https://desarrollofrontend.com/que-es-desarrollo-frontend/, 2014.

[19] A. Pedraza. ¿qué es mobile first y cómo puede mejorar tu posicionamiento? https://www.initcoms.com/que-es-mobile-first-posicionamiento/, 2017.

[20] E. Rodríguez. ¿qué es backend? https://www.seoestudios.es/blog/que-es-backend-web/, 2020.

[21] E. Salguedo. Arquitectura hexagonal. https://medium.com/@edusalguero/arquitectura-hexagonal-59834bb44b7f, 2018.

[22] F. Toledo. Travis-ci para integración continua. https://www.federico-toledo.com/travis-ci-para-integracion-continua/, 2016.