UNIVERSITAT DE BARCELONA

Trabajo de Fin de Grado

GRADO DE INGENIERÍA INFORMÁTICA

Facultad de Matemáticas e Informática
Universidad de Barcelona

EASY-RACETRACK: DESARROLLO DE UNA WEB PARA UN NEGOCIO

David López Ruedas

Director: Jordi José Bazán
Realizado en: Departamento de Matemáticas e Informática

Barcelona, 13 de junio de 2022

Resumen

El proyecto se centra en la creación desde cero de una aplicación web que será la parte visible de un negocio de alquiler de vehículos y material para llevar motocicletas y coches a eventos competitivos.

El objetivo principal es crear el proyecto con una buena base estructural y de desarrollo, con el fin de ser muy escalable y mantenible. El segundo objetivo, a nivel personal, era afrontar el desarrollo con tecnologías que no había utilizado nunca y tenía muchas ganas de aprender.

El proyecto ,principalmente, ha sido desarrollado en Python, en un framework llamado Flask, y en React, un framework que, en mi caso, he utilizado con TypeScript. He utilizado tecnologías como MongoDB, como base de datos o Google Cloud Storage, para el almacenamiento de imágenes en la nube. Para el testing he utilizado Moche, Pytest y react-test-render. Y para acabar, para desplegar la aplicación he utilizado Heroku.

Link de la web: https://easy-racetrack.herokuapp.com/

Resum

El projecte se centra en la creació des de zero d'una aplicació web que serà la part visible d'un negoci de lloguer de vehicles i material per portar motocicletes i cotxes a esdeveniments competitius.

L'objectiu principal és crear el projecte amb una bona base estructural i de desenvolupament, per tal de ser molt escalable i mantenible. El segon objectiu, en l'àmbit personal, era afrontar el desenvolupament amb tecnologies que mai no havia utilitzat i tenia moltes ganes d'aprendre.

El projecte, principalment, ha estat desenvolupat a Python, en un framework anomenat Flask, i a React, un framework que, en el meu cas, he fet servir amb TypeScript. He fet servir tecnologies com MongoDB, com a base de dades o Google Cloud Storage, per a l'emmagatzematge d'imatges al núvol. Per al testing he fet servir Mocha, Pytest i react-test-render. I per acabar, per a desplegar l'aplicació he fet servir Heroku.

Link de la web: https://easy-racetrack.herokuapp.com/

Abstract

The project is focused on the creation from zero of a web application that will be the visible part of a vehicle and material rental business to take motorcycles and cars to competitive events.

The main objective is to create the project with a good structural and development base, in order to be very scalable and maintainable. The second objective, in a personal way, was to develop with technologies that I had never used and I want to learn a lot.

The project has mainly been developed in Python, in a framework called Flask, and in React, a framework that, in my case, I have used with TypeScript. I have used technologies such as MongoDB, as a database or Google Cloud Storage, to store images in the cloud. For testing I have used Moche, Pytest and react-test-render. And finally, to deploy the application I have used Heroku.

Web's link: https://easy-racetrack.herokuapp.com/

Introducción

Tecnologías

Introducción

El proyecto

Mi proyecto consiste en la creación de una página web que sería el medio para poder lanzar un negocio de alquiler de productos y servicios. Estos serían remolques para coches o motos, todo el material necesario para transportarlos e ir a realizar tandas en un circuito, e incluso gestionar el transporte de diversos vehículos de diferentes clientes para transportarlos a la vez.

A pesar de que el objetivo en sí es diseñar y desarrollar una web, detrás hay muchas tareas que son muy importantes. Cómo sería obtener los conocimientos necesarios para poder desarrollar en lenguajes que no conocía o utilizar tecnologías que nunca había utilizado. Como también, aprender a realizar un business plan que respalde la creación de esta web, mostrando el potencial del negocio que trae consigo esta.

Motivación

He escogido el desarrollo en web ya que es algo que nunca había tocado demasiado y tenía ganas de probarme a ver hasta dónde era capaz de llegar. Podría haber hecho una aplicación en android o ios, que es algo que tengo más por la mano, pero, al fin y al cabo, creo que este trabajo es para demostrar lo que has aprendido no solo a nivel de conocimientos sino también a nivel de adaptabilidad a nuevas tecnologías.

También, decidí llevar a cabo este desarrollo ya que llevaba bastante tiempo en mente con el proyecto completo del negocio de alquiler de material y servicios relacionados con el mundo del motor y circuito. Esta idea que me rondaba por la cabeza más la influencia por el gran nivel de emprendimiento que existe entre los jóvenes actualmente, me llevó a querer tener una parte del negocio montada. Porque como dijo Albert Einsteint :

“Una persona que nunca ha cometido un error nunca ha intentado nada nuevo”

Lo cual se alinea mucho con mi mentalidad, ya que he hecho muchas cosas mal, pero las he hecho mal porque las he hecho, sino nunca lo

hubiese hecho mal. Creo que gracias a esta mentalidad he tenido las ganas y fuerzas para tirar este proyecto adelante teniendo en mente el proyecto final que es montar la empresa.

Por otro lado, en cuanto al código y su diseño, también tenía ganas de aplicar muchos conocimientos que he ido adquiriendo durante estos años, tanto en la universidad como fuera. Esto venía motivado por todo lo que he visto a lo largo de la carrera en proyectos que he hecho tanto a nivel individual como en grupo. En todo este tiempo, he visto como hacíamos mis compañeros de grupo y yo, muchas malas prácticas porque no sabíamos cómo hacer las cosa o porque era lo más rápido.

Y como dijo **Alan Kay,** miembro de la Academia Nacional de Ingeniería, entre otras, y conocido por su trabajo pionero en la programación orientada a objetos,

"La mayoría del software actual es muy parecido a una pirámide egipcia, con millones de ladrillos puestos unos encima de otros sin una estructura integral, simplemente realizada a base de fuerza bruta y miles de esclavos"

Por tanto, para hacer que mi proyecto no fuese una “pirámide” decidí aplicar todos mis nuevos conocimientos y realizar correctamente un buen diseño de software con todos los nuevos conocimientos que he ido adquiriendo.

En cuanto a cómo me surgió la necesidad de este proyecto, fue porque yo soy aficionado al mundo del motor y tengo una moto de gran cilindrada, la cual me gusta disfrutar en circuito de vez en cuando. Y ¿cuál fué mi sorpresa cuando me dispuse a ir a un circuito? No hay nadie que se centre ni facilite en este tipo de transporte de vehículos. Hay sitios donde alquilan remolques, otros donde alquilan furgonetas, ambos válidos para el transporte de la motocicleta, pero ninguno de ellos especializado en ello.

¿Qué implica esto? Básicamente, que no te ofrecen los accesorios necesarios para poder tener un servicio integral. Por ejemplo, en el caso de las furgonetas, es necesaria una rampa para subir la motocicleta a la furgoneta y cinchas para sujetar y poder transportarla de forma segura. Estos accesorios, alguien que sea muy asiduo a rodar en un circuito si que dispondrá de ello, pero alguien amateur como yo y otras muchas personas, no disponemos de esto, lo cual

nos obliga a comprarlo, cuando no lo usamos demasiado, o bien pedirlo a alguien. Y en este punto, mi idea de negocio ofrecería una gran ayuda, ya que pondría a la disposición de los usuarios todo este material necesario para que lo pudiesen alquilar durante el tiempo que lo necesitasen.

Por otro lado, muchas veces para alquilar uno de estos servicios hay que desplazarse al lugar donde esté el remolque o la furgoneta y, hoy en día todos queremos todo ya y en casa. Este es otro de los puntos que mi idea de negocio mejoraría, ya que por un módico precio abriría la disponibilidad de tener el servicio contratado en la puerta de casa.

Objetivos

Los objetivos de este proyecto son diversos y como puntos básicos los podríamos resumir en:

- Crear un código de FrontEnd y BackEnd completamente desacoplado, para poder reutilizar la API y poder implementar una app desde 0 solo por parte del FrontEnd
- Crear un código bien diseñado por capas
- Crear una web que permita obtener lo meteriales para ir a rodar
- Crear una web que permita obtener los servicios necesarios para transportar tu vehículo

Planificación

Planteamiento inicial

En la primera reunión con mi tutor le expliqué todo lo que tenía pensado hacer y cómo lo llevaría a cabo. Expuse el producto que quería desarrollar y para qué serviría este. También, planteé la no utilización de Sprints, ya que debido a que estoy trabajando me iba a ir muy justo y me iba a agobiar el hecho de tener un deadline de dos semanas para ir realizando las tareas. Por tanto, decidí que lo mejor sería crear un backlog en trello con diferentes boards.

El primero, sería con las tareas iniciales de decisiones técnicas y de producto, es decir, desde qué tecnologías utilizar, hasta cómo enfocaría el producto que quería ofrecer.

El segundo, con las tareas técnicas definidas de forma más extensa y de forma más clara.

El tercero, donde organicé todas las tareas de desarrollo que tenía que llevar a cabo durante el proyecto. Estas serían la User Stories, con su estructura bien definida y sus acceptance criteria. También, las ordené por prioridad por si no me daba tiempo a llevarlas todas a cabo.

Definición de tecnologías y tareas

Una vez tenía los boards listos, comencé por el primero de todos, donde decidí utilizar las diferentes tecnologías para cada una de las partes de mi web. También, configuré la base de datos y creé las aplicaciones en PasS, para poder ir haciendo CD/CI, ya que era lo que mejor se adapta a mis situación y mis necesidades.

A continuación, hice un desglose de todas las User Stories en subtareas, como era el desarrollo de la parte de la API que necesitaría esa funcionalidad, el diseño y desarrollo de la vista de la aplicación y, no menos importante, la integración entre ambas partes. Esta última, a veces, puede resultar tediosa si no se implementa la API de una forma bien estandarizada y teniendo en cuenta cómo se va a devolver la información.

Fase final

Aproximadamente, un mes y medio antes de hacer la entrega final del código y la memoria me reuní con mi tutor y comentamos que era el momento de comenzar a destinar bastante tiempo a la redacción de la memoria, ya que es una parte importante del proyecto.

En este punto, decidí dejar un poco apartada la web, que ya estaba muy avanzada, pero no pulida del todo, y enfocarme casi al 100% en ordenar toda la información que tenía y comenzar a redactar una memoria bien estructurada y más formal.

Por otro lado, revisé si estaba cumpliendo mis objetivos y si finalmente podría añadir todas las funcionalidades que en un principio había pensado para mi aplicación web. Vi que estaba todo muy avanzado,

pero la parte social, donde los usuarios podrían interactuar entre ellos para compartir gastos no iba a ser posible ponerla ya que entonces tendría que dejar de lado otras cosas que les faltaban pequeños detalles por pulir.

En ese momento, me sentí algo disgustado ya que no podría desarrollar el total del producto que quería presentar. Pero por otra parte asumí que todo lo demás si sería entregado y posiblemente con una calidad mejor que si hubiese intentado abarcar esta funcionalidad también.

Tecnologías

Back-End

Esta parte de mi web es una API REST, la cual se encarga de proveer al Front-End toda la información necesaria de la base de datos. Esto es una explicación simple, pero ahora entraremos en profundidad a ver que es todo esto que he comentado previamente.

¿Qué es una API REST ?

Es una interfaz de programación de aplicaciones (API), basada en los principios de diseño de la arquitectura REST. Por este motivo a veces se las conoce como API RESTful.

El concepto REST apareció por primera vez de manos del ingeniero informático el Dr. Roy Fielding, en su tesis doctoral. En ella define como ésta proporciona un nivel relativamente alto de flexibilidad y libertad para los desarrolladores.

Características de diseño de REST

1. Interfaz uniforme

   Para la transferencia de datos, se aplican acciones concretas (POST, GET, PUT y DELETE) sobre los recursos que gestionan. Para que esto suceda es necesario que estos estén identificados por una URI. Lo cual facilita la existencia de una interfaz uniforme, la cual sistematiza el proceso con la información.

   En otras palabras, con estas acciones, mencionadas previamente, y la URI que señala a los datos, conseguimos poder acceder de una forma casi *inercial*.

2. Desacoplamiento del cliente-servidor

   En el diseño de la API REST, ambas partes deben ser completamente independientes la una de la otra. Solamente debe de haber una información que el cliente debe conocer, la URI del recurso solicitado. De la misma forma, el servidor no

debe de modificar la aplicación del cliente, sino limitarse a pasarle los datos solicitados mediante la petición HTTP.

3. Protocolo sin estado

Todas las peticiones HTTP contienen toda la información necesaria para llevarla a cabo, por tanto ni el cliente ni el servidor necesitan ningún estado previo para poder ejecutarla con éxito.

4. Arquitectura del sistema en capas

Esta se basa en un arquitectura jerárquica entre los componentes que la forman. Cada una de estas capas es la encargada de llevar a cabo una funcionalidad. Esto nos permite una separación entre capas para poder aislar correctamente las responsabilidades de cada una de ellas.

¿Por qué usar REST API?

Como ya he mencionado anteriormente, una de las mayores ventajas que nos ofrece este tipo de implementación es la **separación entre cliente y servidor.**
Al ser independientes, ya que solamente se comunican mediante las peticiones HTTP, pueden ser desarrollados independientemente por diversos equipos sin complicaciones. También permite hacer actualizaciones en independientes de FrontEnd y BackEnd, así como la implementación de diversos FrontEnds con diferentes tecnologías y poder seguir utilizando el mismo BackEnd.

Por otra parte, y un poco relacionado con lo anterior, nos permite una i**ndependencia de tecnologías** magnífica.
Esto permite utilizar cualquier tecnología para implementar la API, con la cual nos sintamos más cómodos o nos sea más rápido trabajar. Y, a su vez, facilita el cambio de tecnología, siempre que respetemos el “acuerdo” establecido en cuanto a las URIs de las acciones. Es decir, siempre que mantengamos la API con las mismas llamadas y parámetros, el cliente podrá seguir haciendo uso de esta sin importarle más allá de la información que le es retornada.

También, nos ofrece una gran **fiabilidad, flexibilidad y escalabilidad.**
Básicamente porque solamente tenemos que preocuparnos de que la conexión cliente-servidor sea correcta. Ya que, como ya he mencionado, si al cliente le llega la información que necesita, de la forma que la necesita todo funcionará correctamente.
Además, como no tenemos ningún tipo de estado, desde el FrontEnd, podemos estar haciendo peticiones a la API, sin importarnos en que servidor esté cada parte de esta alojada, ya que siempre obtendremos respuesta.

En otro ámbito, por lo general, nos aporta una buena **experiencia de usuario**.
Esto es debido a que los datos que se transfieren del servidor hacia el cliente son datos planos, los cuales son menos pesados y, por tanto, rápidos de transferir. Por ende, conseguimos que el usuario no tenga que esperar para ver el contenido o la información que solicita en la parte del cliente.

Y para acabar, en cuanto a performance, **REST requiere menos recursos del servidor.** Esto no siempre es cierto, pero en la mayoría de casos si. Y esto es debido a que al no mantener ningún estado, no requiere memoria y, por tanto se puede utilizar para atender más peticiones. Tampoco requiere escribir el HTML y, por esto, hay menos procesamiento en el servidor.

Desventajas de usar REST API

Para una aplicación ya establecida de forma muy centralizada el concepto de REST es un problema, ya que mediante esta arquitectura no tiene porque estar todo en el mismo servidor, ni siquiera un servidor tiene porque saber de la existencia de la parte de la API que se encuentra en otro. Por tanto, a la hora de migrar hacia esta tecnología podría ser algo tedioso, pero muy provechoso.

Por otra parte, como no mantenemos estados, es necesario montar una infraestructura propia con el fin de poder identificarte en el servidor. Esto, normalmente, se hace mediante un token, con el cual le dices al servidor quién eres.

En cuanto a *timings* se podría dar la situación de que hubiese un desacompasamiento entre FrontEnd y BackEnd debido a la

independencia de estas. Esto puede tomarse como algo malo desde el punto de vista de la entrega de producto acabado, pero a la vez es algo bueno porque no hay dependencias entre equipos.

¿Porqué Python?

Últimamente, la elección de Python como lenguaje para desarrollar aplicaciones se está volviendo la más común. Ya sea por su fácil entendimiento, sintaxis clara y poco obstructiva, por la gran cantidad de bibliotecas *OpenSource* que hay disponibles o por toda la documentación al alcance de cualquiera. Por ejemplo, lo podemos ver en el número de preguntas realizadas sobre este lenguaje en *StackOverflow.*

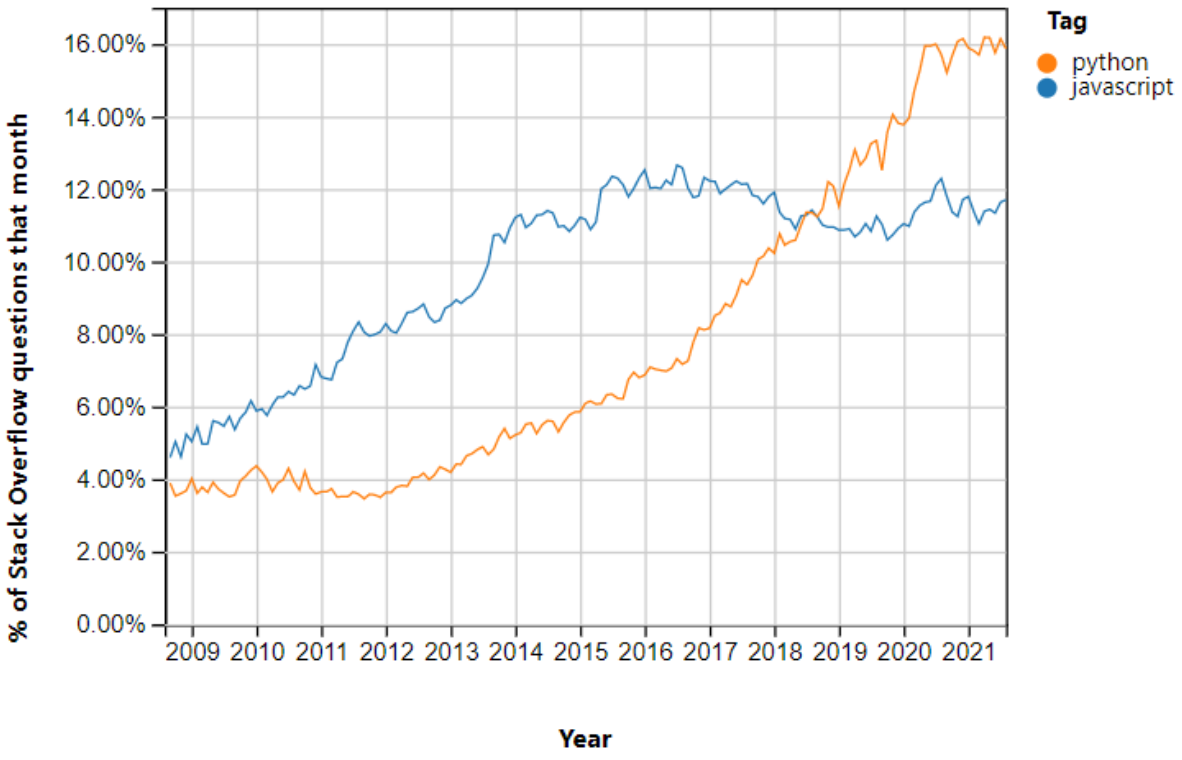


Imagen 1. Gráfico del número de consultas en StackOverflow de Python vs JavaScript.

También, es apreciable en el número de empleos ofertados con este lenguaje como base del stack tecnológico a manejar. Como se puede observar, en Estados Unidos es el más solicitado con diferencia a su competidor más cercano, Java. En Europa, está algo más igualada la demanda, aún así, el mayor referente para analizar estos datos sigue siendo Estados Unidos.

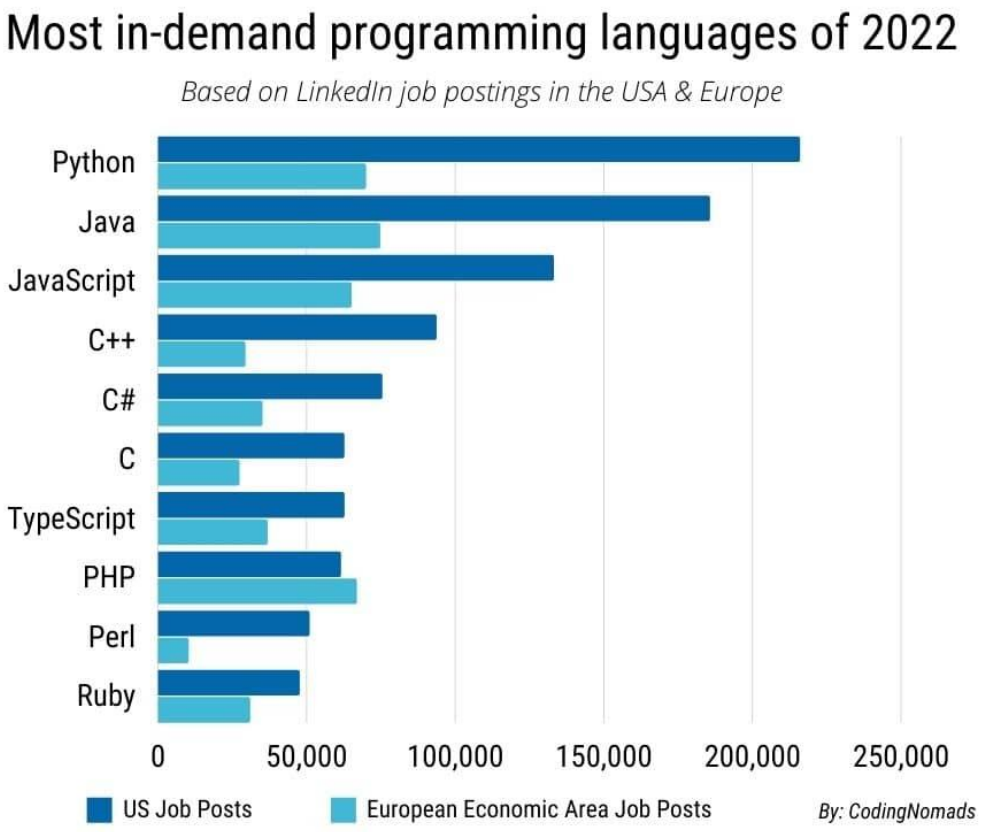


Imagen 2. Gráfico de los lenguajes de programación más demandados en 2022 en Europa vs USA.

¿Flask o Django como framework?

Podríamos decir que cuando desarrollamos en Python existen dos frameworks que son los más utilizados Django y Flask. El primero, es más antiguo y, en consecuencia, más popular. En Github podemos encontrar datos como que hay casi 350 mil repositorios donde se hace referencia a este. Flask, es algo menos popular en Github, pero tampoco se queda atrás con caso 290 mil repositorios haciendo referencia a este.

Aunque Django es más antiguo y está más establecido y tiene algo más de alcance, Flask tiene bastantes puntos a su favor. Este, desde un principio se pensó para ser más escalable y simple. Gracias a esto, permite que las aplicaciones sean más ligeras en comparación con las creadas utilizando Django. Flask es conocido como un 'MicroFramework', que básicamente quiere decir que está formado por un núcleo simple y robusto, pero a su vez es muy extensible. Gracias a esto, podremos elegir qué base de datos utilizar o que tipo de errores queremos utilizar. Y esto es posible gracias a toda la

documentación y bibliotecas que existen alrededor de este magnífico Framework.

Base de datos

En este punto tuve que decidir si utilizar una base de datos relacional, como mySQL, o bien no relacional, como MongoDB. Utilizando ambas tengo algo de experiencia, el primer tipo lo había utilizado en algún proyecto de la universidad y, el segundo, es con el que trabajo a diario en mi actual empleo. Habiendo utilizado las dos, me decanté por el uso de una base de datos no relacional, MongoDB.

¿Por qué MongoDB?

Hay muchos beneficios que aporta la utilización de esta base de datos, los cuales explicaré a continuación.

Primeramente, nos ofrece un **modelado de datos** que facilita mucho la obtención desde el BackEnd debido a su simpleza de formato. Esto es porque MongoDB guarda todos los datos en formato similar a JSON. Además, todas las librerías que trabajan con mongo vienen preparadas para serializar los datos desde el lenguaje que se esté utilizando en documentos que MongoDB entienda.

En cuanto a performance, MongoDB nos permite una gran **escalabilidad** para poder recibir gran cantidad de flujo de usuarios. Al ser una base de datos distribuida nos permite escalar tanto horizontalmente, creando más nodos, como verticalmente, aumentando la CPU y la RAM.

También, ofrece una gran **flexibilidad**, debido a su esquema Schema Less,lo cual permite hacer crecer la base de datos sin la necesidad de crear manualmente o mediante scripts nuevos campos con valores por defecto al agregar nuevos campos en nuestros registros.

Por otro lado, nos ofrece la posibilidad de tener **clusters distribuidos** por diversas partes del uno. Esto nos permitirá ofrecer una mayor velocidad de lectura de datos si nuestra aplicación estuviese presente en diferentes países del mundo, por ejemplo.

Para acabar y no menos importante, MongoDB nos permite hacer consultas mediante una sintaxis bastante sencilla pero a la vez muy

potente. Esta potencia unida con las Agregaciones, nos permite realizar operaciones entre diversas colecciones, lo cual es muy útil.

Conexión a base de datos

En este punto tuve algunos problemas, ya que al inicio del proyecto me había decantado por la librería PyMongo. Ya que según había estado leyendo e informándome parecía ser una buena opción. Después de hacer algunas pruebas de conexión con la base de datos y de envío y recepción de información parecía ir todo correctamente. Poco tiempo después, al empezar a desarrollar la API esta librería comenzó a darme problemas con la lectura y escritura en base de datos.
Después de encontrarme en esta situación volví a buscar posibles librerías para conectar mi API con MongoDB y encontré Mongoengine. Con esta no tuve ningún problema de conexión o envío y recepción de datos.

Además, encontré una gran cantidad de documentación sobre la utilización de esta librería, donde se explica en detalle todos los métodos que ofrece para las *Queries*. (https://docs.mongoengine.org/).

FrontEnd

Esta parte de mi web es la cara visible de mi trabajo, lo que el usuario verá y le interesa de todo mi producto.

Al pensar en este aspecto del proyecto tuve bastantes dudas ya que nunca había hecho web más allá de algún proyecto de un par de asignaturas de la carrera. En esta habiamos usado Vue.js, pero no me acabó de gustar este lenguaje. Sin embargo, no había utilizado desarrollado en ningún otro lenguaje de web, pero si había tocado react-native, el cual me gustaba bastante más como funcionaba.

En este punto de indecisión, decidí arriesgar y aprender react desde prácticamente 0, pero me parecía un buen reto para un Trabajo de Fin de Grado. Con esto podría demostrar la facilidad o no que tengo a la hora de adaptarme a un nuevo lenguaje de programación que no conocía.

¿Por qué React?

React es una librería de Javascript, la cual está enfocada en el desarrollo de interfaces de usuario. Esta, es un gran aliado a la hora de llevar a cabo el desarrollo de aplicaciones web e incluso aplicaciones para smartphones. Para poder llevar a cabo todas estas tareas existe un gran conjunto de módulos, componentes y herramientas capaces de facilitar el cumplimiento de los objetivos a los desarrolladores.

Es decir, React es una muy buena base sobre la que construir cualquier aplicación con JavaScript. Ya que nos ayuda a desarrollar de forma más eficiente y ordenada que si usaramos librerías como jQuery, centradas en la manipulación del DOM, o bien utilizando JavaScript puro. Un ejemplo de algo muy útil que ofrece es la vinculación entre datos y vista, lo que nos permite actualizar una vista en función del cambio de los datos.

Como he mencionado previamente, React nos aporta bastantes ventajas con respecto a librerías como jQuery. Primeramente, nos ofrece la posibilidad de utilizar una arquitectura de desarrollo mucho más avanzada. Algunas de estas podrían ser: la conversación entre componentes, con intercambios de información entre ellos o la encapsulación del código de en componentes.

Puntos fuertes de React

La encapsulación en componentes es uno de los motivos que más me atrae de este lenguaje, debido a las posibilidades de escalabilidad, reusabilidad y mantenimiento que esto nos otorga. Al tener pequeñas piezas de código como componentes, podemos moverlos de una pantalla a otra sin afectar a nada de esa misma, al igual que podemos utilizar ese mismo código en diversas pantallas a la vez sin la necesidad de tener que repetir código absurdamente.

Otro motivo, es la posibilidad de trabajar con TypeScript en lugar de JavaScript, ya que es bastante mejor y menos laborioso el trabajo con este lenguaje. Por tanto, la posibilidad de todas las ventajas que ofrece React junto con todo lo bueno que conlleva el uso de TypeScript. Este nos aporta cosas geniales como el tipado de funciones y atributos, para poder prevenir errores de asignación de

valores incorrectos o devolución de datos incompatibles por parte de una función.

También, el hecho de que React sea isomórfico ayuda mucho al desarrollo en el mundo web. Esto quiere decir que es capaz de, con un mismo código, renderizar HTML en la parte del servidor y en el cliente. Esta arquitectura nos permite desacoplar el desarrollo del lado del servidor el del lado del cliente.

En definitiva, React permite el isomorfismo, un concepto que, por ejemplo, angular numba había tenido hasta su última versión 2.x. Además de muchas otras librerías que siguen sin ser capaces de soportar el isomorfismo.

¿Cómo funciona React?

La base es el componente, que es una pequeña parte de la interfaz de usuario. Como norma general, para construir una aplicación en react se construyen muchos de estos componentes, piezas pequeñas y reutilizables, para crear interfaces de usuario más complejas.

Cada uno de estos componentes se agrupan en un componente padre, el cual se suele nombrar *Contenedor*. De esta forma, podemos tener diferentes componentes creados con diversos más pequeños. A su vez, todos estos están siempre agrupados en el componente “Raíz” que es el encargado de gestionar toda la aplicación.

En cuanto a lenguaje, los componentes se crean en JSX,si usamos JavaScript, o TSX,si utilizamos TypeScript, en lugar de como se hacía previamente con HTML.

¿Cómo funcionan los componentes?

Los componentes pueden ser, por lo general de dos tipos: *Class Components*, o *Functional Components.*

Los primeros, como su nombre sugiere, se crean utilizando la sintaxis de clase ES6 y difieren de los functional components en muchos aspectos.
Primero de todo, estos necesitan crearse extendiendo de la clase de *Component* nativa del lenguaje, gracias a lo cual tiene acceso a todas la funciones del componente padre. Segundo, necesitan un

método *render()* que devuelva un elemento de react (código TSX o JSX).

Además, podemos utilizar lo que se conoce como *Lifecycle methods*. Para acabar, decir que este tipo de componentes utiliza una función **constructor()**, la cual puede aceptar propiedades y mantener sus datos con un estado. A estos datos, podemos acceder más tarde, haciendo uso de las palabras reservadas "this.props", porque los *Class Components* tienen instancias.

```
import React, { Component } from 'react';
import RecipeList from './RecipeList';

class RecipeContainer extends Component {
    constructor() {
      super();
      this.state = {
          recipes: []
        };
    }

    componentDidMount() {
        fetch("http://localhost:3000/recipes")
          .then(res => res.json())
          .then(data => this.setState({ recipes: data }))
      }

    render() {
      return (
          <div>
              <RecipeList recipes={this.state.recipes} />
          </div>
      )
    }
}

export default RecipeContainer;
```

Imagen 3. Ejemplo de un Class Component de React.

En cuanto a los *Functional Components,* básicamente son funciones que aceptan propiedades como argumentos y devuelven un código JSX en lugar de utilizar un método *render()* como los anteriores. Estos pueden ser declarados como una función de TypeScript o JavaScript o bien como una *Arrow function*.

A diferencia de los anteriores, no hace falta extender la clase nativa “Component”, que conlleva no tener acceso a los métodos que proporciona esta.

```
import React from 'react';

const RecipeList = recipes => {
    return (
        <ul>
            {recipes.map(recipe => <li>{recipe.title}</li>)}
        </ul>
    )
}

export default RecipeList;
```

Imagen 4. Ejemplo de un Functional Component de React.

En resumen, los *Functional components* son más concisos y simples de entender y mantener. Para la implementación de componentes simples son ideales, pero como he comentado antes carecen de ciertas características. Por otro lado, los *Class components* implementan más lógica y son más complejos ya que extienden de la clase “Component”. Lo más recomendable es empezar con *Functional components* mientras no se requiera de métodos de estado y ciclos de vida.

Deployments a servidor

Para esta tarea he elegido Heroku, ya que ofrece un servicio fácil de utilidad y de calidad. Lo he utilizado tanto para tener la API como la parte del FrontEnd de mi web.
Heroku utiliza contenedores de Linux, los cuales llaman “dynos”. En ellos se aloja la web, webservices, api o aplicaciones que se ejecutan del lado del servidor.

¿Cómo funciona Heroku?

Se puede utilizar de diferentes formas, mediante la CLI de heroku, conectando tu cuenta de github o mediante el panel de control. Esta segunda opción, en mi caso y en el caso de otros compañeros no funcionaba, por tanto no he podido probarla ni saber como funciona en su plenitud. La tercera es bastante intuitiva, ya que puedes configurar que tipo de proyecto utilizaras, el lenguaje que

utilizas y el nombre que le darás a tu aplicación, pero no acaba de gustarme para casi nada. Por eso mismo la primera opción, que es la que he utilizado, me parece la mejor. Consiste en instalar heroku en tu ordenador y mediante los comandos que este ofrece, como hacer login, crear la app en heroku, deployar o ver los logs de tu aplicación de heroku.

Por otro lado, mediante automatizaciones como las Github Actions, permite hacer deploys automáticos cuando tu lo configures, es decir, en cada commit a main, en cada Pull Request cerrada a main, o cuando lo desees.

Configuración de las Github Actions

Como he comentado previamente, para automatizar los deploys a Heroku he utilizado la funcionalidad de Github que permite automatizaciones, las Actions.

Primeramente hay que crear en el proyecto una carpeta llamada “.github” que contenga otra carpeta llamada “workflows”. Una vez tenemos ambas, hay que crear un fichero de tipo *.yaml*, donde estará el código de configuración de la Acción automática.

En este fichero se indica cuándo se ejecutará esta acción mediante el comando “on:”. Esta se ejecuta teniendo en cuenta diversos eventos: pull_request, push, fork. También hay que indicar sobre que rama se ejecutará cuando se reciba uno de los eventos anteriores. Y, para acabar, se configuran los diferentes *jobs* que queremos llevar a cabo.

En mi caso, he configurado que la action se ejecute cuando se cierre una pull request sobre main. Y se ejecuta la acción que lleva a cabo el deploy a Heroku.

```yaml
name: Deploy

on:
  pull_request:
    types:
      - closed
    branches:
      - main

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - uses: akhileshns/heroku-deploy@v3.12.12 # This is the action
        with:
          heroku_api_key: ${{secrets.HEROKU_API_KEY}}
          heroku_app_name: "easy-racetrack" #Must be unique in Heroku
          heroku_email: "dlopez.ruedas@gmail.com"
```

Imagen 5. Código de configuración de la Github Action de deploy a Heroku.

¿Por qué Heroku?

Hay muchos motivos por los cuales me he decantado por el uso de este PasS, pero principalmente ha sido su rapidez a la hora de deployar cualquier aplicación en sus servidores y su estabilidad, ya que su infraestructura está basada en Amazon Web Services. Esto último nos garantiza un excelente rendimiento, estabilidad y velocidad. Por otro lado, es muy escalable ya que nos permite escalar de forma horizontal (asignar más recursos al dyno) o de forma vertical (añadir más dynos).
Por otra parte, es un servicio multiplataforma, es decir, se puede interactuar con ellos desde Windows, Mac o Linux sin importar que realmente están basados únicamente en Linux. Lo cual es muy útil ya que no tienes que preocuparte por más que desarrollar tu código y desplegarlo en los “dynos”.

Para acabar, y para mi uno de los puntos más importantes, es la facilidad que ofrece a la hora de hacer los despliegues. En poco más de 10 minutos, entre configuración y despliegue puedes tener tu aplicación lista para ser usada desde una dirección pública y usable desde cualquier lugar.

Desventajas de heroku

Una de sus mayores desventajas es la falta de documentación sobre este, sobre todo cuando te encuentras con errores, no

aportan gran documentación ni información que ayude a resolverlos.

Por otro lado, tiene un límite de almacenamiento, debido a su arquitectura y el ciclo de vida de los 'dynos'. Y por ende, guardar logs en el servidor es una tarea bastante costosa y complicada, ya que requiere hacerse de una forma concreta y a veces requiere el uso de un add-on especial.

Almacenamiento en el cloud

Para esta funcionalidad he utilizado Google Cloud Storage, con el fin de almacenar las imágenes de los productos y no tenerlas almacenadas dentro del proyecto.

Primeramente, tenía en mente utilizar Firebase Storage, pero investigando con los diferentes proveedores de esos servicios vi que Google Cloud Storage se adecuaba más a mis necesidades.

¿Cuál es su función?

Google Cloud ofrece la posibilidad de crear buckets donde almacenar archivos y poder acceder a ellos mediante una URL pública. Para ello hay que configurar los permisos de acceso a los datos y hacer estas imágenes públicas para todos los usuarios.

En este caso yo almaceno las imágenes de todos los productos y servicios disponibles disponibles en la página web, con el fin de hacer más ligera la web.

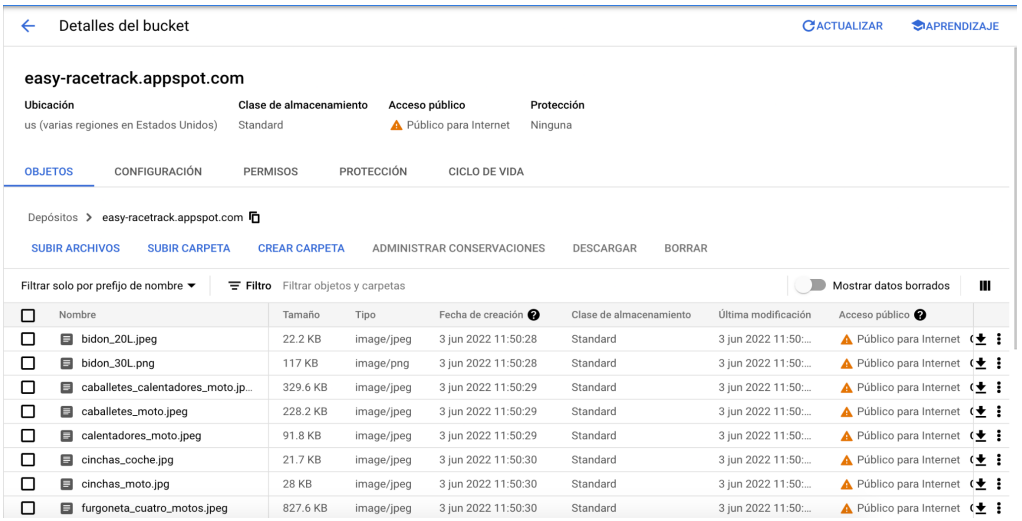


Imagen 6. Bucket de Google Cloud Storage que contiene las imágenes de los productos.

Diseño del software

Arquitectura por capas

El diseño de mi aplicación, tanto en el FrontEnd como en el BackEnd, está basado en 3 capas bien aisladas y separadas entre sí, para facilitar el desarrollo, la iteración y la mantenibilidad. Gracias a esto, he conseguido un código claro, sencillo y entendible por cualquiera que se una ahora al proyecto.

BackEnd

Como ya he mencionado, esta parte de mi aplicación es una API Rest, por tanto, cuanto más modular y más aisladas sus capas más reusable podrá ser. Con esto quiero decir que con el diseño que he hecho he conseguido que la API no esté acoplada a ninguna tecnología de FrontEnd. Por otro lado, en mi diseño a lo único que estoy acoplado es a la tecnología de persistencia de datos, ya que si en algún momento se decidiese cambiar este habria que invertir más tiempo que si esto no estuviese diseñado de esta forma. También cabe decir, que un cambio en la tecnología de persistencia no es algo que suela suceder en un proyecto ya desarrollado, ya que desde un principio se han tenido en cuenta las ventajas y desventajas de este.

Una vez introducido, explicaré el patrón de diseño que he escogido para mi aplicación. Este es MVC (Modelo-Vista-Controlador).

¿Por qué MVC?

Este patrón nos ofrece muchas ventajas en el diseño de la api debido a diversos puntos como la separación entre capas, que he mencionado anteriormente, la simplicidad de mantenimiento y escalabilidad del código, la facilidad a la hora de desarrollar Test Unitarios de cada una de sus partes, etc.

Separación por capas

Esto nos ofrece la posibilidad de aislar las capas entre sí, es decir, la vista solo conoce al controlador y este solo conoce al modelo. De esta forma conseguimos encapsular responsabilidades en cada una de las capas y así no modificar desde la vista nada del modelo, por ejemplo.

La capa de la vista es la encargada de comunicar la API con el exterior, es decir es la parte visible de esta. En ella he definido las rutas, junto a los parámetros que esperan, que sirven para que el FrontEnd se comunique y obtenga los datos que necesite para llevar a cabo sus tareas.

El controlador, se encarga de filtrar la información que le llega de la vista y se asegura que los datos que le envía al modelo son los que este espera. En caso de que algún dato no tenga un formato adecuado o falte alguno, lanza un error para evitar la propagación del error hacia adentro. Esta capa también se encarga de la comunicación con otros controladores, en el caso de necesitar información de otro modelo. Por ejemplo, de cada cesta guardo una lista con el ID de los productos y cuando necesito recuperar el detalle de los productos el controlador del modelo “Cesta” se comunica con el controlador de “Producto” y este le proporciona la información necesaria.

El modelo es el encargado de modelar la parte de la realidad que representa dentro de la aplicación. En él se definen los “atributos” que tendrá nuestro modelo y su tipo, por tal de tener una acotación de los valores que se esperan en cada uno de los campos. En mi caso, este también es el encargado de comunicarse con la persistencia de datos, que en esta caso es MongoDB. Como ya he comentado anteriormente, esto podría mejorarse creando una capa más que fuese la de persistencia y de este modo desacopla riamos el modelo de esta, lo que nos permitirá en algún momento hacer una migración a otro tipo de sistema de persistencia de datos.

Gracias a este diseño de software, he conseguido que mi API sea totalmente funcional con cualquier tecnología de FrontEnd. El día de mañana, podría decidir pasar a aplicación nativa en dispositivos iOs o android y solo habría que hacer la parte de la aplicación del FrontEnd, ya que la API sería perfectamente válida y útil.

Mantenibilidad y escalabilidad

Estos son otros de los motivos por los cuales he decidido utilizar este diseño, ya que son dos factores, para mi, imprescindibles en el desarrollo de una API.

En cuanto a escalabilidad, al tener todo bien estructurado, podemos hacer evolucionar y crecer la API a velocidades grandes y sin un gran coste. Esto no seria mas dificil que añadir una vista, un controlador y un modelo del nuevo Objeto que necesitemos controlar dentro de la aplicación.

En la vista tendríamos todas las nuevas rutas, que no tendrían ningún conflicto con ninguna otra, ya que esta todo bien aislado en sus correspondiente ficheros.
En cuanto al modelo, definimos los campos que necesitamos que tenga este y las acciones que necesitemos realizar sobre la base de datos.
Para acabar, el controlador sería el único que podría llevarnos a modificar otros controladores en el caso que necesitemos alguno de los datos de otro modelo. Pero como he mencionado antes, gracias al aislamiento de cada fichero y capa sería algo sencillo y rápido de solventar.

En cuanto a mantenibilidad, podríamos decir que lo único que debemos encargarnos es de ir actualizando las librerías que utilizamos y adaptar, si fuese necesario, el uso que se les da.

Facilidad de testing

Gracias a este diseño resulta muy fácil realizar tests de cada una de las funciones de nuestra API de forma individual. Esto es posible ya que tienen únicamente una función y por tanto, no dependen de otras para poder comprobar que están funcionando correctamente. Esto en cuanto a los Test Unitarios, cuando queremos hacer tests de integración también facilita mucho el testing debido a que tenemos bien localizado y aislado un flujo completo que lleva a cabo cada una de las llamadas de la API.

FrontEnd

En esta parte de mi aplicación también utilizo un diseño similar al del BackEnd, ya que es bastante similar al MVC, ya que tiene una vista, que es la implementación de React de la UI. El controlador, en este caso Service, sería el encargado de hablar con la API y a su vez con el modelo, que sería la Store. Es decir, tendríamos un diseño Vista-Service-Store, donde la store sería la encargada de almacenar todos los datos y estados que el la Vista necesita y obtiene mediante el Service.

Mi idea era utilizar este diseño, pero finalmente lo he utilizado recortando un poco su estructura. He prescindido de las Stores, ya que por ahora no tengo gran cantidad de datos que almacenar en el FrontEnd.

Ahora almaceno estos datos en la vista mediante React hooks, en su gran mayoría State Hooks. Estos datos los obtengo mediante las Stores, que son las encargadas de hablar con la API y obtener la información necesaria para mostrar en la vista.

Esto es algo que no me convence mucho y seria mejor tener separado, ya que a la hora de escalar la aplicación facilita las cosas, pero por tiempo es algo que he decidido sacrificar y, en un futuro, implementar para mejorar el diseño del software.

Arquitectura de componentes

Esta es mi parte preferida de react y uno de los motivos que me llevó a elegir esta librería para desarrollar mi proyecto. Este diseño de componentes nos ofrece diversas ventajas, desde reusabilidad hasta facilidad de testeo. Estos componentes se agrupan en componentes mayores, llamados Containers, que como su propio nombre indica, tienen la función de contener de 1 a N pequeños componentes.

En cuanto a la reusabilidad, al poder crear pequeños componentes con funcionalidades propias, podemos utilizarlos múltiples veces de un mismo Contenedor de forma muy sencilla y sin tener que repetir código. También, nos permite utilizar este en diversos Contenedores sin que tener que reimplementarlos una y otra vez, simplemente los importamos y ya lo tenemos disponible.

A su vez, esto nos facilita el testing a nivel individual de cada uno de los componentes, ya que tienen una única responsabilidad y forma de renderizarse. Por tanto, podemos tener una fiabilidad muy alta ya que tendremos casi todo el código testeado, lo cual aporta seguridad y tranquilidad.

Desarrollo

Antes de empezar todo el desarrollo de código planteé el diseño de software de forma gráfica mediante diagramas UML, para tener una idea mucho más visual del código que quería conseguir.

Bases

¿Por qué usar UML?

UML aporta una buena base a la hora de desarrollar ya que conseguimos un dibujo de aquello que tendremos que desarrollar más adelante. De esta forma todo será más claro y no habrá dudas en el momento de la programación. Es decir, si realizamos un buen conjunto de diagramas UML, podríamos darle esta documentación a alguien externo al proyecto y sería capaz de desarrollar todo el código sin ningún tipo de duda.

Esto es posible gracias a los diagramas principales que se utilizan en los proyectos: Diagrama de Casos de uso, Diagrama de Clases, Diagrama de Estados o de Contexto y Diagrama de Paquetes.

Diagrama de Casos de uso

Para empezar, tendremos que definir que es un caso de uso. Un caso de uso es '*Una conversación entre el sistema y el actor externo al sistema para obtener un resultado observable de interés'*.
Por tanto, este diagrama sirve para detallar lo máximo esta conversación entre el sistema y el actor, con tal de tener en cuenta todos los resultados de esta conversación.

En mi caso, he creado un diagrama por cada uno de los casos de usos que me han surgido al diseñar mi Diagrama de Estados o Contexto. Además, he creado un diagrama donde se muestran los casos de uso que realiza cada actor (en mi caso solo es uno) y la relación entre algunos de ellos. Por ejemplo, todos los casos de usos, excepto registrarse,

incluyen el caso de uso *Login,* ya que para poder realizar cualquier acción es necesario estar logueado.

Diagrama de Clases

En este diagrama se trata de llevar a cabo la división del sistema en diferentes partes, las cuales tendrán relación entre sí, las clases. Estas pueden ser ideas muy generales y hacer referencia solamente a conceptos que pertenecen al dominio. O bien, pueden ser muy concretas y describir en profundidad la clase, tanto sus atributos como su métodos. Este diagrama puede ir evolucionando y ser utilizado durante todas las fases del desarrollo.

En mi caso, he optado por el modo general y básicamente he diagramado el tipo de relación que existe entre cada una de las clases de mi dominio y el tipo de relación que existe entre ellas.

Diagrama de Estados o Contexto

En este diagrama se muestran todos los estamos en los que el usuario puede estar dentro de la app y como transiciona entre ellos. Para cambiar de estados se hace mediante Casos de uso, es decir, para ir de un estado a otro tiene que haber una conversación entre el usuario y el sistema.

Lo he usado para esto, básicamente, gracias a este diagrama he dejado claro cómo se pasará de un estado a otro y mediante que caso de uso.

Diagrama de paquetes

Este diagrama sirve para ordenar los ficheros que crearemos para nuestro proyecto según su función. Lo ideal, es crear una vista por caso de uso y un controlador y un modelo por modelos de nuestra aplicación.

CI/CD (Continuous Integration - Continuous Deployment)

Esta metodología es una práctica muy común hoy en día en los equipos de desarrollo, sobre todo en los que utilizan metodologías ágiles. Esta permite la continua entrega de trabajo según se finaliza el desarrollo ya que está todo automatizado. Es necesario tener tests, la cantidad que te haga sentir seguro, los cuales se ejecuten previamente al deploy del software finalizado.

El testing automatizado es algo básico para poder llevar a cabo este tipo de desarrollo, ya que no es viable deployar algo que no es seguro al 100% que funcionará correctamente.

Esta metodología la he utilizado para llevar a cabo mi proyecto, ya que por mi situación no me sentía cómodo con los sprints, ya que me causan más estrés y frustración de lo que podrían ayudarme.

Testing

BackEnd

En esta parte de mi proyecto, he llevado a cabo tests unitarios y tests de integración. Los primeros se encargan de probar que cada una de las funciones de la API hace su función, por ejemplo, que el método encargado de guardar en la base de datos un usuario cuando se registra lo hace correctamente y devuelve lo esperado.

Los segundos, son los encargados de una función más extensa, se encargan de verificar que cuando desde el exterior se hace una llamada a una ruta de la API con los parámetros requeridos, esta responda lo que se espera.

Como he comentado previamente, la cantidad de tests que se desarrollan va en función de lo inseguro que te sientas en cuanto a tu código. En este caso, mi código no me genera excesiva inseguridad, por lo que he realizado los tests en las funciones de los controladores de mi API. Lo he hecho aquí, ya que los modelos simplemente se encargan de gestionar la comunicación con la base de datos, y en caso de que algo falle en esa comunicación, será un error no proveniente de mi código. En este caso podría ser que el servidor de la Base de datos estuviera caído.

FrontEnd

Por esta parte, he realizado test por componentes, ya que como ya he explicado, son la base de mi Web. Estos se basan en hacer un render del componente y 'machtearlo' con un snapshot, que es una captura de él mismo. Esto consolida la base del proyecto ya que todos los elementos que utilizo estarán probados de forma individual y no habrá lugar a que fallen.

Frameworks que he utilizado

A día de hoy existen muchísimas librerías de testing ya sea para Python, React o simplemente llamadas http. En mi caso he utilizado una diferente para cada tipo de test que he mencionado anteriormente.

- **Test unitarios de la API**: Para esta tarea he utilizado PyTest, que es un framework que ofrece una forma muy sencilla de test. Simplemente tenemos que crear una función donde definimos con que debe 'assertar' la respuesta que nos devuelve la función que estamos testeando.

- **Test de integración de la API:** En este caso, he utilizado Mocha test, que es un framework basado en JS y que corre sobre Node.js, que ofrece la posibilidad de hacer tests asíncronos de forma sencilla. En este caso es perfecto, ya que lo he utilizado para los tests de las llamadas http.

- **Test de componentes de la Web:** Para testear los componentes de mi aplicación he utilizado **testing-library/react** que se encarga de permitir testear los componentes mediante la comparación con snapshot que tenemos guardados de este mismo.

Buenas prácticas con GitHub

Flujo de trabajo

Ya sea trabajando solo o en equipo, es muy importante elegir un buen flujo de trabajo. Para esto es importante decidir el procedimiento a seguir cuando se detecte un bug o cómo se van a separar las diferentes tareas que se van a llevar a cabo, con el fin de identificar los cambios del proyecto.

Existen diversos flujos de trabajo, en función del objetivo que se tenga a la hora de desarrollar. Tres de ellos son:

- **Forking Workflow:** consiste en que cada desarrollador tiene un repositorio propio donde trabaja. Este, no es escalable ni mantenible a no ser que los diferentes desarrolladores no trabajen en nada que interfiera entre sí, cosa bastante improbable.

- **Gitflow Workflow:** en este todas las ramas están diseñadas alrededor de las entregas de producto que se van a realizar. Esta es una mejor opción que la anterior, ya que aunque los desarrolladores trabajan sobre código muy cercano, siempre se irán actualizando entre ellos para no tener luego grandes conflictos.

- **Git Feature Branch Workflow:** este se basa en la creación de una rama independiente para cada una de las features que se estén desarrollando. Para mi, este es el más acertado y más óptimo, ya que aporta limpieza a nuestro repositorio, además de evitar conflictos ya que cada feature no afectará a otra, en el caso ideal, claro está.

Una vez analizadas las tres y mirado otras opciones, me decidí por la que creo que es la más idónea para mi proyecto, y realmente para cualquier otro, que es el tercero de los anteriores flujos de trabajo.

Conventional commits

Gracias a esto, se puede identificar el trabajo realizado en cada uno de los commits y además tener un listado de todo lo que se ha hecho en esa rama. Y evitamos estar desinformados con lo realizado en cada uno de estos.

Imagen 7. Ejemplo de commits no convencionales, por tanto poco entendibles.

Para utilizar esta convención se suelen utilizar 5 prefijos:

- **feat:** indica que se ha trabajado en una nueva feature
- **fix:** indica que se ha solucionado un bug
- **docs:** indica que se ha añadido o actualizado documentación
- **test:** indica que se ha añadido un test
- **refactor:** indica que se ha ejecutado un refactor en el código

Merges correctamente

Github permite 3 formas de mergear ramas mediante Pull Requests, pero idealmente sólo habría que utilizar una. Las opciones que ofrece Github son:

- Create a **merge commit**, a través del cual se añaden todos los commits uno a uno a la rama base mediante un merge.

- **Rebase and merge**, con el que se añaden directamente a la rama base los commits de la rama a comparar.

- **Squash and merge**, a través del cual se hace un único commit a la rama base con la información de todos los commits de la rama a comparar.

Según mi experiencia, la última opción es la mejor y más segura, ya que no ensucia la rama principal y facilita todo en caso de que sea necesario quitar lo que se haya desarrollado en una rama en concreto.

Por tanto, he decidido utilizar esta última opción ya que es la más segura y eficiente. En mi proyecto quizá es un poco indiferente ya que solamente trabajo yo, pero es muy importante tener buenos hábitos para poder aplicarlos en el trabajo o en proyectos personales más grandes.

Evolución

Detallado del BackEnd

Siguiendo las metodologías que había escogido empecé diagramando con UML todo lo que fuese necesario para empezar el desarrollo correctamente. Empecé por el diagrama de casos de uso, seguidamente el de clases, donde relacioné todos los modelos que tendría mi aplicación. Una vez acabado esto, comencé con el diagrama de estados o contexto, para ver cómo se iban a conectar mis casos de estado, es decir, a través de qué casos de uso lo iban a hacer. Para acabar, hice el diagrama de paquetes, el cual me permitiría comenzar a picar código con la mayor brevedad posible.

Una vez tenía claros los casos de uso, y la estructura de paquetes que iba a tener, comencé por la API.
Lo primero que hice fue crear todos los controladores, modelos y vistas, vacíos y organizados en carpetas. Seguidamente, me dispuse a crear las configuraciones del proyecto basado en Python, mediante un fichero *runn.py* que es el encargado de ejecutar toda mi aplicación. Una vez hecho esto, pasé a las pruebas e integración con la base de datos, ya que nunca había utilizado python con MongoDB, no tenía muy claro cómo iba. Primeramente, escogí MongoPy e

hice algunas pruebas. Todo parecía ir bien, hasta que al cabo de pocas pruebas empecé a tener problemas de conexión con colecciones concretas y conexiones muy lentas.

En este punto, me frustré un poco porque parecía ir todo bien hasta que dejó de funcionar sin motivo aparente. Entonces, volví a buscar y encontré Mongoengine, otra librería para el mismo fin. Esta tenía muchísima documentación y ,de hecho, parecía incluso más simple de integrar y utilizar. Cambie toda la implementación que tenía de configuración de la otra librería y una vez tuve todo cambiado comencé a hacer las mismas pruebas. Tras un buen rato probando vi que todo iba bien y decidí quedarme con esta librería.

Una vez tenía este tema solucionado, comencé a desarrollar las vistas de mi API, donde tengo las rutas, mediante las cuales se obtiene la información mediante peticiones http. Con todas definidas añadí una herramienta de Flash, Blueprint, que permite agrupar todas las rutas de cada una de las vistas para, luego, poder gestionarlas todas desde el fichero raíz, el *run.py*.

Definidas todas las rutas, comencé con los modelos de mi dominio. Decidí qué métodos y atributos tendría, cuáles de estos serían obligatorios o no, y de qué tipo serían. Para ayudarme en esta tarea, creé la clase de cada uno de mis modelos, con la ayuda de la librería Mongoengine, que ofrece una serie de tipos para los atributos del modelo, mediante su función ".Document". Seguidamente, continué desarrollando los modelos, ya que únicamente tenían los métodos vacíos y los atributos tipados.

La mayoría de estos métodos son los encargados de la comunicación con la base de datos, es decir, se encargan de guardar, actualizar, borrar y obtener los objetos del modelo de la base de datos. Como ya he comentado previamente, aquí he acoplado mi modelo a la tecnología de persistencia de datos, lo cual no es lo más correcto, pero tomé esta decisión ya que no tenía intención de cambiar de persistencia. Lo ideal hubiese sido crear una capa más, la cual si estuviera acoplada a persistencia y no fuese el modelo, pero era demasiado complejo para las necesidades de este proyecto.

Avanzados los modelos, proseguí con los controladores, que se encargan de pasar la información necesaria que llega de las vistas, mediante las peticiones http, hacia los modelos, para poder interactuar como sea necesario con la base de datos. Los controladores también se encargan de evitar que llegue cualquier tipo de información incorrecta o información que pueda llegar a provocar un error en el momento de la comunicación con la base de datos. Por ejemplo, antes de intentar actualizar una cesta, comprueba que esta existe, ya que de esta forma están controlados todos los errores que no sean provocados por un error de la base de datos.

Cuando ya tenía toda la API finalizada, seguí haciendo pruebas y acabando de retocar algunos tests, ya que durante la implementación, lo esperado inicialmente y lo que finalmente estaban devolviendo los Endpoints había discernido ligeramente.

Detallado del FrontEnd

Finalizado, por el momento, el desarrollo de la API, me dispuse a comenzar la implementación del FrontEnd. Para esto había escogido el framework React y como nunca lo había utilizado, dediqué tiempo a aprender las bases y hacer algunos proyectos de prueba para tener una primera toma de contacto con este. Una vez ya tenía algo de soltura, inicialice el proyecto mediante un pequeño template basado en TypeScript que ofrece el propio framework en Github.

Pero antes de empezar a desarrollar código necesitaba una idea de qué aspecto quería para mi web, entonces hice algunos diseños. Estos los hice en la visión móvil de la aplicación, pero tanto en desktop como en móvil será muy similar. Para llevar a cabo este mockup, utilicé la plataforma ‘proto.io’, que ofrece diferentes elementos prediseñados para poder montar el diseño general de tu aplicación mediante el sistema drag-and-drop, lo que lo hace bastante sencillo. Acabados estos diseños, tenía una idea general de cómo sería mi web y comencé a diseñar las ventanas de mi app.

Primeramente, creé los ficheros de todos los Contenedores, los componentes grandes que contienen los componentes

que forman cada una de las ventanas. Seguidamente, pasé a la implementación de los componentes que necesitaban cada uno de estos Contenedores. Los componentes los hice lo más genéricos y reusables posible, con el fin de poder reutilizarlos al máximo posible, ya que este es uno de los puntos fuertes de trabajar con componentes.

Durante el desarrollo de la parte del FrontEnd me encontré con varias dificultades, ya que nunca lo había utilizado y había muchas cosas que no sabía. Lo primero que me surgió, fue cómo gestionar que un usuario no pudiese acceder a ninguna de las funcionalidades de la web, que no fuese login o registrarse, sin estar logueado, ya que es un requisito para mi aplicación.

Empecé a buscar por internet cómo hacerlo y encontré una posible solución, que consistía en crear un componente “router” custom. Dentro del cual se hacía la comprobación de si el usuario estaba logueado o no y se devolvía el contenedor correspondiente o, en caso que no, se devolvía el container “Login Screen” para que se loguearse el usuario. Después de invertirle tiempo, lo implementé pero me daba errores por el tipo de componente que estaba retornando, luego al recuperar el valor mediante el cual compruebo si estaba logueado.
En este punto, decidí que no me merecía la pena seguir invirtiendo más tiempo en intentar solucionar esto y puse el código de comprobación dentro de cada uno de los containers. Antes de renderizar el contenido de cada uno de los containers hago la comprobación y en caso de que el usuario no esté logueado lo redirecciono a la pantalla de LogIn.

```
const logeado: boolean = localStorage.getItem("userId") != null;

return (
  <div>
    {!logeado ? (
      <Navigate to="/login" />
    ) : (
      <div className="backgroundProductsScreen">
        <NavBar />
        <div className="itemsList pb-5">
          {!isLoading ? (
            listItems
          ) : (
            <h1 className="text-center h-screen">Loading...</h1>
          )}
        </div>
        <Footer />
      </div>
    )}
    ;
  </div>
```

Imagen 8. Código de control sobre el estado logueado o no de un usuario.

Para saber si un usuario ha hecho login utilizo el LocalStorage del propio navegador, donde posteriormente a que el usuario se logee con unas credenciales correctas, guardo su userId. En este sentido, una posible iteración sería cambiar esto por un token generado cada vez que se hace login, con una duración determinada de 1 hora, por ejemplo. Pero por ahora, como primera versión es una buena opción, ya que este local storage si cierras la pestaña se borra.

Una vez solucionado este tema, me surgieron otros problemas en este mismo container y en el de Registro. Ya que quería utilizar componentes para cada uno de los inputs de los formularios de registro y login, pero me encontré con el problema de no saber como pasar los valores desde el componente hacia el container, ya que enviar información “hacia dentro” es muy sencillo, ya que los componentes aceptan parámetros como input, pero el retorno de valores no es tan sencillo. Estuve buscando y no fui capaz de encontrar nada que me fuese útil y me solucionase el problema. Así que decidí no utilizar estos pequeños componentes en estos contenedores.

Más adelante del proyecto, en el momento de crear los formularios del checkout de la reserva me encontré en la misma situación, ya que tenía un componente que contenía todos los campos de la información personal y otro que contenía la información del método de pago. Viendo que era el mismo problema que ya había tenido decidí volver a hacer investigación ya que si era algo que pasaba tanto, a mí en un proyecto pequeño ya dos veces, tenía que existir una solución en algún foro o blog de internet.

Tras una intensa búsqueda, acabé encontrado una forma de hacerlo, había que crear cada uno de los componentes asignándoles explícitamente el tipo. Esto se hace asignando el tipo 'React.FC<*InterfazConLoNecesario*>'. Explico este tipado, mediante esta asignación de tipo, le estamos diciendo al componente que será del tipo React Functional Component y esperará las propiedades que necesitemos, que es lo que se contiene entre '<>'.

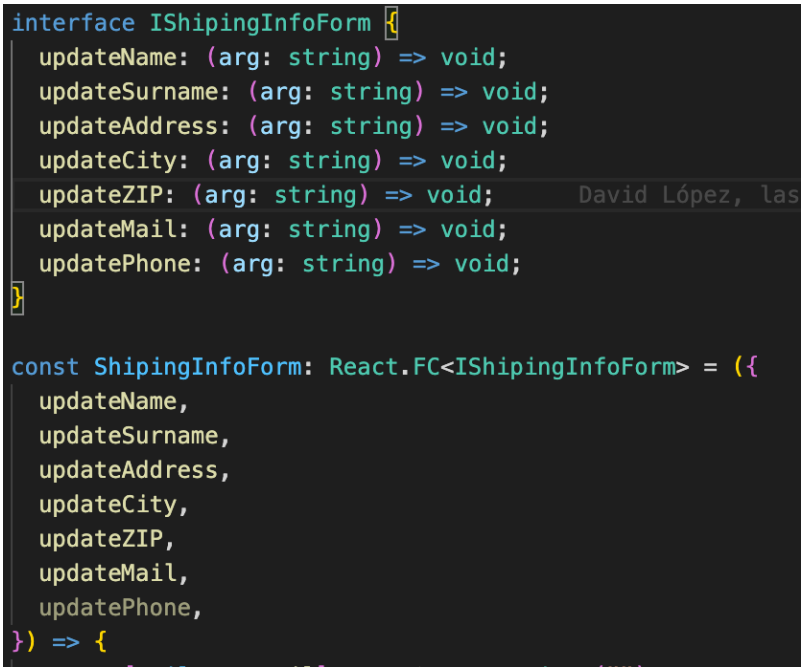


Imagen 9. Código de los formularios para poder pasar información entre el componente padre e hijo.

Volviendo a la evolución del desarrollo, seguí con la pantalla principal donde se ven todos los productos disponibles. Aquí he utilizado la reusabilidad que permiten los componentes, he implementado solamente una vez el componente que contiene los datos del producto y el botón de añadir a la cesta. Primero obtengo la lista de productos de la API

mediante el 'ProductService', y seguidamente aplico la función map al array de productos con el componente 'ProductCard'.

```
const listItems = products.map((product:IProduct) => (
  <ProductCard product={product}/>
));
```

Imagen 10. Código de reutilización de un componente, usándolo para crear tantos componentes como elementos haya en la lista.

En el método 'useEffect()' de react llevo a cabo la recepción de datos de la API y seteo el estado 'isLoading' a true, para mostrar un mensaje de carga, con el fin de conseguir una mejor UX, y una vez he obtenido todos los datos cambio el valor de este estado para mostrar todos los productos obtenidos.

```
useEffect(() => {
  getProducts();
  setIsLoading(true);
}, []);

async function getProducts() {
  const response = await ProductService.getProducts();
  setProducts(response as IProduct[]);
  setIsLoading(false);
}
```

Imagen 11. Código de como obtener información de la API cada vez que se recarga la página.

Esta misma estructura la utilizo también en el contenedor 'Cart', donde necesito conseguir de la API los productos que el usuario ha añadido a su cesta. También en el contenedor 'Bookings' donde el usuario puede visualizar todas las reservas que ha hecho, las cuales obtengo de la API.

Siguiendo con la lista de todos los productos, el componente de la card del producto, me dio algún dolor de cabeza ya que es 'touchable' para poder acceder al detalle del producto si haces click sobre él y a la vez tiene el botón para añadirlo a la cesta. Estas dos zonas 'touchables' tenían conflictos, ya que al hacer click en el botón de añadir a la cesta se detectaba el

click sobre el botón pero también sobre la card, lo que hacía navegar al detalle del producto sin yo querer esto.

Para solucionar esto, tuve que utilizar los 'Mousse events' de React, para poder capturar los eventos que se lanzan sobre el botón y poder parar la propagación del evento. De esta forma, solo se ejecuta una acción 'onClick', la del botón y no la de la zona 'touchable' de le prouctCard.

```
async function handleClick(event: React.MouseEvent<Button>) {
  event.stopPropagation();
```

Imagen 12. Código que hace posible hacer click en un Componente Touchable sin que se lance el evento del que está por encima.

Por otro lado, en el tema de los estilos, he utilizado la librería Tailwind, que ofrece configuraciones de estilos css ya preconfiguradas y se aplican utilizando palabras clave en la tag "className" de los componentes. Esta librería facilita mucho el trabajo de darle una bonita estética a la web ya que tiene muchísima documentación y en su web te ofrece previsualizaciones de cómo actúa cada una de las palabras clave sobre los componentes.

Esta librería también ofrece la posibilidad de hacer los componentes responsive, mediante el uso de las palabras clave: 'sm','md','lg','xl' o '2xl'. Que un componente sea responsive significa que en función de la resolución de la pantalla en la que lo estés visualizando cambie de tamaño o incluso cambie la información que este muestra.

Para utilizar esta funcionalidad, simplemente hay que utilizar los prefijos que he mencionado previamente en la tag 'className', seguido de el estilo que queremos que se aplique en esa resolución de la pantalla. En mi proyecto lo he utilizado en todos los containers y componentes, con el fin de que la web se vea bien tanto en un dispositivo móvil como en un ordenador.

Por ejemplo, lo he utilizado en el contendor de la lista de productos, para cambiar el tamaño y, por ende, el número de productos que se ven simultáneamente en función del

dispositivo de visualización. En desktop se ven 5 productos de un tamaño y en móvil se ve 1 más grande. Esto es una decisión tomada, pero es fácil de cambiar en futuras iteraciones si es necesario.

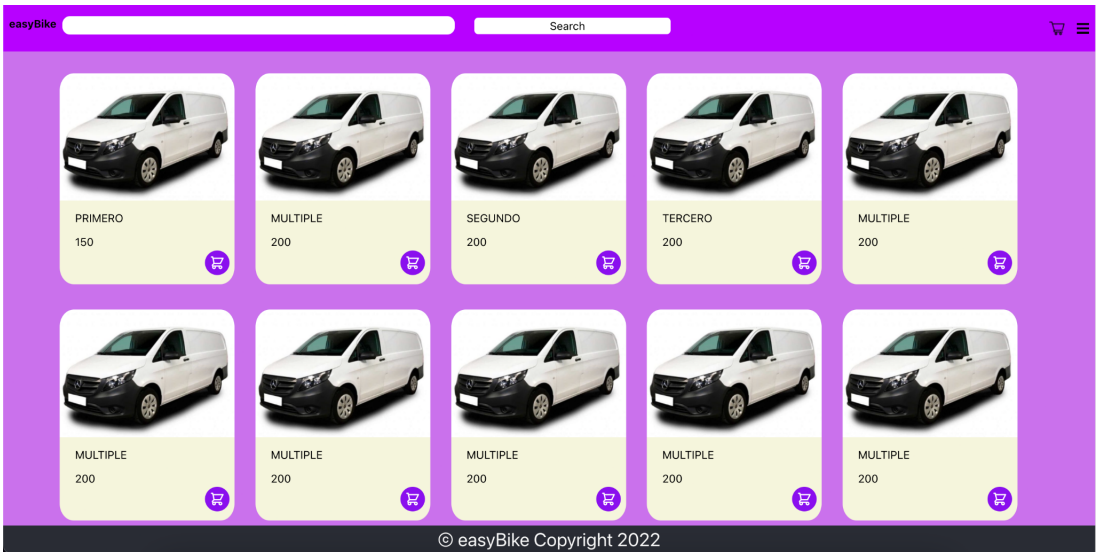


Imagen 13. Vista de la web en desktop.

Imagen 14. Vista de la web en mobile.

Otra librería que he utilizado es ‘react-hot-toast’ que sirve para mostrar toast personalizados para dar feedback al usuario de la acción que ha realizado. Por ejemplo lo he utilizado para el LogIn, tango sea exitoso como no, muestro

un toast informando al usuario si ha ido correctamente o que tipo de error ha ocurrido.

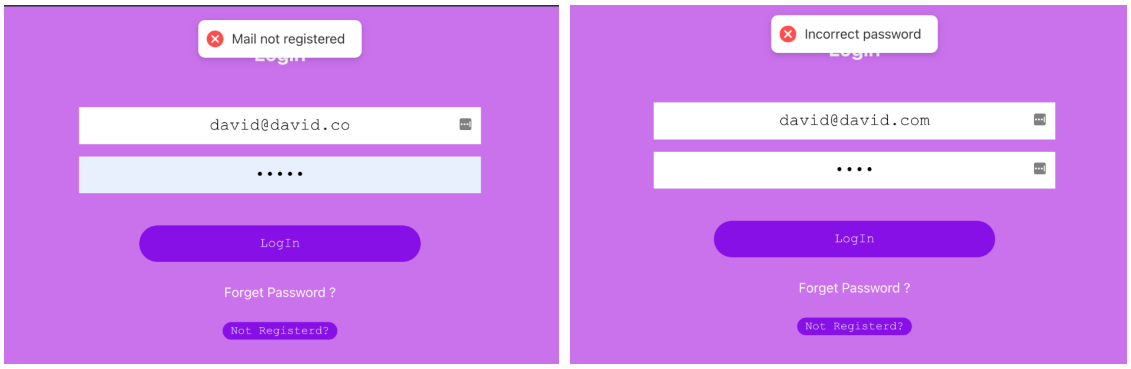


Imagen 15 y 16. Toast de error al hacer login con credenciales incorrectas

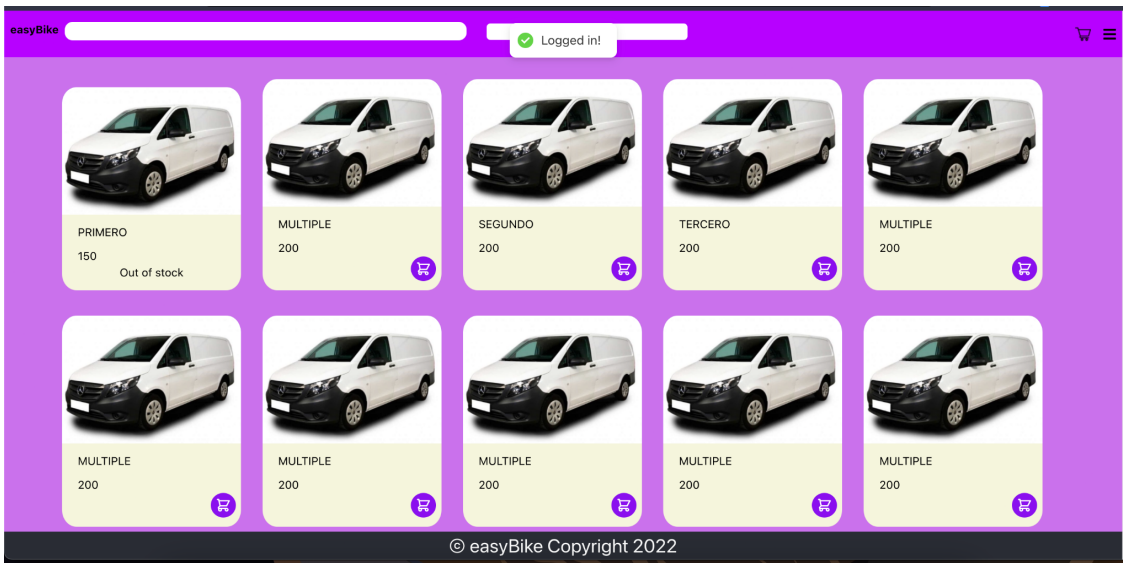


Imagen 17. Toast de todo correcto al hacer login.

Para acabar, añadí una capa de seguridad a la hora de hacer el Login y el Registro. La contraseña que el usuario introduce, siempre se hashea mediante sha256, un tipo de hash que no se puede desencriptar. Esto proporciona un buen nivel de seguridad ya que aunque alguien obtuviese la contraseña de la base de datos mediante cualquier acción, no podría hacer nada con esas credenciales.

Objetivos

Al empezar este proyecto me marqué bastantes objetivos ya que muchas de las tecnologías que iba a utilizar no las había utilizado antes. Mi mayor objetivo era aprender react web, ya que era algo que me llamaba la atención pero nunca lo había utilizado para nada. Por otro lado, también el aplicar buenas metodologías y patrones a la hora de hacer el código era algo primordial para mi.

Aprender react y todo lo que ofrece

Como ya he comentado, este era uno de mis principales objetivos, ya que el FrontEnd no lo había trabajado en exceso ni en la carrera, ni a nivel personal ni profesional. Por esto mismo tenía muchas ganas de aprender a desarrollar FrontEnd y react me pareció un muy buen framework por el cual empezar.

Empecé con algunos tutoriales de youtube y algunos artículos sobre esto, para coger un poco de base. Cómo funcionan los componentes, los diferentes tipos que hay, como desarrollarlos de la mejor manera y más eficiente posible, etc.

Una vez establecidas las bases comencé con el desarrollo de la web y durante este proceso me encontré con problemas que no sabía solucionar con lo que había aprendido hasta ese momento. Por tanto, tuve que volver al mundo del aprendizaje a leer artículos y foros con el fin de aprender y aplicar estos conocimientos a la solución de mis problemas.

Creación y manejo de una API rest

Por esta parte, sí había participado en la creación de una API en un trabajo de la universidad pero no en mucha profundidad, por eso mismo tenía ganas de hacer un proyecto de estas características desde 0.

El proyecto que hicimos en la universidad no me gustó demasiado la organización y estructura que le dimos y quería hacerlo realmente bien, aplicando patrones de diseño y buen diseño de software.

Este objetivo que tenía marcado creo que lo he cumplido con creces, he aplicado el patrón MVC, para separar las capas y

responsabilidades correctamente. También he aplicado diferentes principios SOLID, como *Single Responsability Principle* u *Open-Closed Principle.*

El primero lo he aplicado en los modelos y en los controladores. Ya que únicamente tienen una única tarea y responsabilidad. Gracias a esto he podido realizar un testing unitario sencillo y eficaz.

El segundo lo he aplicado en toda la API, ya que en cualquiera de sus capas es sencillo iterar y añadir funcionalidades sin necesidad de cambio del código actual.

Aplicar correctamente CI/CD

En este punto tenía algo de experiencia en proyectos de la universidad y del trabajo, pero no había iniciado y gestionado toda la infraestructura que esto conlleva. Ya que para poder llevarlo a cabo son necesarias varias cosas.

Primeramente, he creado la aplicación en Heroku y la he conectado con mi proyecto mediante la configuración que ofrece su propio CLI.

Seguidamente, hay que realizar unos buenos tests, para ejecutar antes del *Deployment* del código finalizado, que se ejecuten de forma automática.

Para acabar con este proceso he trabajado con Github Actions, que me ha permitido crear una Action que ejecute los tests y, en caso de que todos pasen, se haga el *Deployment* automático a Heroku.

Aplicar un correcto diseño de software

Este punto era muy importante para mí, ya que a lo largo de la carrera se nos han explicado diferentes patrones de diseño que son muy útiles.

Probablemente en proyectos en grupo que hemos realizado a lo largo del grado no los hemos utilizado a penas y por eso mismo era importante para mi hacer uso de patrones SOLID, arquitectura MVC o CI/CD.

Features de la web

En este aspecto, tenía como objetivo que la web final tuviese implementados todos los casos de uso que definí en un principio.

Estoy muy contento ya que los he podido implementar todos excepto el último que era realizar reservas grupales, ya que no me ha dado tiempo. He preferido sacrificar esta parte e invertir tiempo en tener el resto de partes bien implementadas y con una funcionalidad total.

Testing con usuarios

Tras tener una versión bastante final de la aplicación contacte con algunos compañeros y familiares para pedirles que hiciesen una serie de tareas y me dieran algo de feedback sobre la web. Los probadores serán:

- **Nuria**: 50 años, acostumbrada a realizar algunas compras online en páginas como Amazon. Le cuesta realizar según qué tareas en internet.

- **Jose**: 23 años, familiarizado con las compras online en todo tipo de webs. Se desenvuelve con soltura por internet.

- **Carlos**: 27 años, suele comprar online solo por aplicaciones móvil de android, no suele hacer compras mediante una web.

- **Andrea**: 45 años, muy asidua a las compras online tanto en aplicaciones móvil como mediante web.

Una vez he tenido los usuarios he definido una serie de tareas que debían llevar a cabo para comprobar si el funcionamiento de la web es intuitivo y sencillo. Primeramente he definido las tareas como registrarse o realizar una reserva. Una vez realizadas estas, les he planteado una serie de preguntas para saber su opinión y sensaciones en relación a la web.

Tareas a realizar y preguntas

Tareas:

1. Crearse un usuario y acceder a la web. Una vez dentro ir a ver tu perfil.
2. Buscar el producto “Rampa para moto” verlo en detalle y añadirlo a la cesta.
3. Añadir 5 productos a la cesta y eliminar los 3 más baratos.
4. Añadir algún producto más a la cesta y realizar una reserva (los datos de la tarjeta pueden ser ficticios).
5. Después de realizar la reserva, ir a “Reservas” y comprobar que está todo correcto.

Preguntas:

1. Nivel de satisfacción con la web
2. ¿Con qué probabilidad recomendarías a un amigo utilizar los servicios?
3. ¿Qué te ha parecido el diseño de la web?
4. ¿Te ha resultado fácil moverte por la web?

Valoración de dificultad de las tareas y respuestas a las preguntas

Tareas:

Tarea / Persona	Nuria	Jose	Carlos	Andrea
1	5	5	5	5
2	3	4	4	5
3	4	5	5	4
4	3	5	3	4
5	3	4	4	4

Imagen 17. Tabla de valoración de dificultad de las tareas a realizar.

Preguntas:

Pregunta / Persona	Nuria	Jose	Carlos	Andrea
1	3	5	4	5
2	4	4	5	4
3	3	3	4	3
4	4	5	5	4

Imagen 18. Tabla de valoración de las preguntas sobre la web.

Conclusiones del testing con usuarios

Tras analizar los resultados de las encuestas y recibir feedback directo de los usuarios he decidido hacer algunos cambios para mejorar la navegación y el flujo dentro de la aplicación.

Primeramente, me han comentado, en general, que el menú lateral no era demasiado intuitivo y sería más cómodo tener todas las opciones en la barra superior para tener un acceso directo, con menos clicks.

Otra cosa que me han comentado, es que estaría bien recibir un mail de confirmación tras acabar la reserva.

Estos son los dos puntos negativos que me han destacado.
El primero lo cambiaré rápidamente y con pocas dificultades. Ya que es algo sencillo de hacer y si mejora el entendimiento y la facilidad de llevar a cabo las reservas es un gran punto.

El segundo, tengo que estudiarlo y ver si es una implementación sencilla y si me dará tiempo. En caso de ser así lo implementaré y estará disponible en la versión final de la web. Sino, quedará como una futura iteración que llevaré a cabo para acabar de cerrar el funcionamiento completo de la aplicación.

Conclusiones

A lo largo del desarrollo del proyecto he ido consiguiendo la mayoría de objetivos planteados, pero por falta de tiempo algunas ideas que tenía en mente no he podido llevarlas a cabo. Sin embargo, los objetivos principales que eran crear la web desde cero, aprender a manejar lenguajes que no había utilizado nunca y conseguir una buena arquitectura de software los he conseguido.

A continuación hablaré sobre problemas que me he encontrado durante el desarrollo del proyecto y cómo los he solventado, o si no he conseguido solventarlos. También mencionaré una serie de futuras mejores posibles para la web.

Problemas durante el desarrollo

Configuración del deploy

Al comenzar con la configuración del deploy tuve algunos problemas. Primeramente, Heroku tenía algún problema con la conexión con Github, lo cual hacía imposible conectarse con el repositorio y hacer deploy desde su web. Esto lo solventé haciendo toda la configuración desde el terminal de mi ordenador.

Una vez hecha la configuración y conexión con Heroku me surgió otro problema, la plataforma no detectaba automáticamente el lenguaje de mi proyecto y por ende tuve que especificarlo explícitamente mediante un fichero de configuración llamado 'Procfile'.

Paso de información entre componentes de React

Este problema me lo encontré en el momento de implementar los formularios de mi web, ya que como he mencionado, he implementado todo lo más componentizado posible con el fin de aislar y poder reutilizar el máximo código posible.

Entonces, al querer utilizar la información del componente formulario desde el contenedor donde este se encontraba, no era posible con mi primera implementación.

Tras buscar documentación y consultar en foros encontré una forma de conseguir lo que necesitaba. Había que crear componentes que tipándolos de la siguiente forma '`React.FC<IBookingSummaryAmount>`', donde `IBookingSummaryAmount` es una Interfaz donde se especifican los parámetros que este componente recibirá. A su vez esto permite mantener la información en el contenedor de este componente.

Configuración de los tests

La configuración de los tests me dio algunos problemas, sobre todo la configuración de los test unitarios de la API. Ya que nunca había utilizado la herramienta PyTest y lo que había mirado parecía sencillo. Al introducirlo en mi proyecto vi que necesitaba realizar mocks y esto me complicó todo. Mientras lo intentaba configurar estaba gastando tiempo que podía invertir en seguir desarrollando la web y así lo hice. Estos tests los dejé de lado por este motivo.

Futuras mejoras

Nuevas funcionalidades para el usuario

En cuanto a funcionalidades no implementadas y no contempladas al inicio tengo algunas en mente que me gustaría seguir desarrollando para acabar de dejar el funcionamiento de la web completamente cerrado.

Primeramente, implementar una pasarela de pago para poder cerrar las reservas mediante un pago real. No lo he implementado, ya que no lo tuve en mente en ningún momento, pero he buscado información y es algo medianamente laborioso pero asequible en un plazo corto.

Seguidamente, llevar a cabo la implementación de la posibilidad de permitir a los usuarios ponerse en contacto entre ellos mediante una parte social de la web. Sería algo similar a un foro donde los usuarios podrían publicar que van a alquilar y los espacios disponibles que tienen y así compartir gastos con otro usuario.

También, me gustaría añadir algún tipo de filtro por categoría o buscador de productos.

Por otro lado, algo que si quería implementar pero por tiempo no he podido llevar a cabo es la recuperación de contraseña. Lo dejé para el final y cuando me quise poner a ello tenía cosas más prioritarias que finalizar.

No me ha dado tiempo ya que había que tener muchas cosas en cuenta. Primero, hay que generar un link o una contraseña temporal, lo cual tiene su dificultad. Segundo, hay que enviar un email con este link o contraseña al mail que el usuario introduzca. Para acabar, en caso del link, ver cómo se mantienen los datos para saber qué es ese usuario y segundo, redirigirlo a cambiar la contraseña. Por el

lado de la contraseña temporal, hay que tener en cuenta si esta ha caducado o no y comprobar si coincide con la que hemos generado previamente.

Por la complejidad de todo esto no me ha sido posible implementarlo.

Mejoras o cambios

En cuanto a la UI, dedicaría un buen tiempo en mejoras algún detalle ya que no está todo lo perfecta que me gustaría debido a mi nula experiencia en este campo.

En la pantalla de *Cart* me gustaría cambiar el hecho de agrupar el mismo elemento en una sola visualización con un contador y poder subir o bajar la cantidad del producto seleccionado.
En el lado de la reserva de productos y servicios, sería interesante añadir un campo donde el usuario añada la duración de la reserva del producto y/o servicios. Podría implementarse mediante un widget de selección de fechas.

Funcionalidades de administración

A día de hoy la parte del administrador por parte del FrontEnd no está implementada, pero sería una funcionalidad muy útil que llevar a cabo.

Esta incluiría un panel de control para gestionar las reservas, ya se para contactar con el usuario para concretar alguna recogida por falta de información, modificar una reserva a petición del usuario o cancelarla si fuese necesario.

Desarrollo de la parte de administración

Para poder llevar a cabo esta nueva funcionalidad en mi aplicación sería necesario el desarrollo de toda la parte de FrontEnd, ya que no se ha empezado con ellos. Por la parte de BackEnd posiblemente sería necesaria la implementación de algunos nuevos métodos y rutas en la API, pero no conllevaria un desarrollo excesivo.

En la parte del FrontEnd, habría dos opciones: crear una nueva web con un dominio diferente o bien diferenciar tipos de usuario. Existirían los usuarios “compradores” y los administradores. Seguramente me decantaría por esta segunda, ya que estar todo

bien estructurado en capas y componentes sería mucho más sencillo que hacer una nueva aplicación.

Como administrador se tendrían diferentes paneles de controles para gestionar todo lo que la web incluye. Habría un apartado de gestión de usuarios, donde cambiar algún dato cuando el usuario lo solicite, un apartado para modificar o anular las reservas, en caso de que un usuario haya cometido algún error a la hora de realizarla o por algún motivo no pueda finalmente en las fechas seleccionadas.

Por otro lado, estaría toda la gestión de productos. Control de stock, añadir productos nuevos, eliminar productos descatalogados o actualizar los actuales.

Esta funcionalidad también necesitaría solo de FrontEnd, ya que a día de hoy ya es posible hacerlo mediante llamadas HTTP a la API.

Objetivos cumplidos

Aprendizaje de nuevos lenguajes y frameworks

En este ámbito, personalmente creo que he cumplido con creces mis objetivos.

Por la parte de FrontEnd he pasado de no saber nada de react a haber desarrollado una web desde 0. Por esto mismo estoy muy satisfecho del nivel de conocimiento que he adquirido en este ámbito ya que partía de 0 y ahora tengo una compresión y desarrollo de este bastante buenos.

Por el lado del BackEnd, ya tenía conocimientos del desarrollo de APIs en Python, pero la vez que lo llevé a cabo en un proyecto de la universidad no hicimos una buena implementación ya que estaba desordenado y mal implementado realmente.

Aplicación de patrones de diseño y desarrollo

Como he mencionado en el apartado anterior, en otros proyectos que había hecho en la universidad con compañeros no habíamos hecho arquitectura ni diseño de software.

Por esto mismo personalmente creo que he dado un gran salto en cuanto a calidad de software, llevando a cabo, en el BackEnd, una arquitectura por capas. Con esta he conseguido un gran aislamiento entre ellas y un bajo acoplamiento, lo cual lo hace muy escalable y reutilizable.

Por el lado del FrontEnd, también he aplicado una buena arquitectura por capas y, además, he componentizado al máximo todos los elementos de mi aplicación. Con esto he conseguido dos cosas: reusabilidad de estos componentes, ya que puedo utilizarlos en cualquier container sin tener que repetir código, y facilidad en el momento de hacer tests, ya que aplicarlos a componentes pequeños es mucho más sencillo y útil que aplicarlos a toda una pantalla.

Este objetivo para mi era muy importante, ya que creo que un software de calidad es muy importante en todos los proyectos y, muchas veces, en los proyectos de la universidad esto no se tiene en cuenta ni se busca, por tanto es algo que he decidido aplicar en este proyecto ya que es algo que he aprendido durante la carrera y he seguido aprendiendo sobre esto fuera de ella también.

Para acabar, decir que he sufrido y disfrutado a partes iguales a lo largo de la realización de este proyecto, lo cual no es algo malo desde mi punto de vista. Ha sido un gran reto unir todo lo aprendido desde que empecé el grado universitario hasta el día de hoy.

Ha sido un gran reto para mí y estoy orgulloso de lo que he conseguido, ya que he tenido que compaginar el desarrollo de este proyecto con el trabajo a tiempo completo. Lo cual posiblemente me ha impedido llevar a cabo algunas funcionalidades que me hubiese gustado desarrollar pero no ha sido posible.

Bibliografía

[1] Introduction to React - Build your first React App video
https://www.youtube.com/watch?v=NzpbupWoIV4

[2] Template web to create a react app - Create React App documentation
https://create-react-app.dev/

[3] Reactjs - Library to build user interfaces
https://es.reactjs.org/

[4] Mongo connection with Python - Documentation to connect mongoDB with Python
https://www.mongodb.com/languages/python#:~:text=The%20first%20step%20to%20connect,text%20editor%20like%20Textpad%2FNotepad.&text=Use%20the%20connection_string%20to%20create,get%20the%20MongoDB%20database%20connection

[5] Npm Rc-Touchable - Npm library to have touchable components
https://www.npmjs.com/package/rc-touchable

[6] Documentation about flask-mongo-python - How to connect python with mongodb through flask
https://www.mongodb.com/developer/how-to/flask-python-mongodb/

[7] Rest API python - Video where explain the bases of Python Rest API
https://www.youtube.com/watch?v=GsCCyN3fRoI

[8] W3School mongo find - Documentation about find in mongodb
https://www.w3schools.com/python/python_mongodb_find.asp

[9] W3School mongo update - Documentation about update in mongodb
https://www.w3schools.com/python/python_mongodb_update.asp

[10] Github repo about react app - Code about a reac base app
https://github.com/QuentinWatt/React-for-beginners-tutorial-series/blob/master/src/App.js

Annexos

Business plan

Resumen ejecutivo

Easy-racetrack comercializa productos y servicios necesarios para el transporte de vehículos deportivos a recintos de competiciones.

El cliente ideal de Easy-racetrack es un adulto aficionado al mundo del motor con un presupuesto medio para el mundo del motor, de entre 20 y 50 años. El transporte de vehículos mundo del motor amateur y aficionado es una pequeña parte de este sector, pero que no está para nada explotada, por tanto es una gran oportunidad de mercado.

Dado que el servicio inicial será a nivel de comunidad autónoma, Easy-racetrack utilizará tres estrategias de marketing principales: publicidad mediante redes con influencers del mundo del motor de la propia comunidad, asistencia a eventos del mundo del motor y publicidad en eventos competitivos de este sector.

Easy-racetrack está actualmente en fase de desarrollo, pero cuando se lance oficialmente sus proyecciones de ingresos són de 1.000€ en los primeros 3 meses posteriores a su lanzamiento.

El negocio está fundado por David López Ruedas, con amplia experiencia en el sector del motor, y un socio, con amplios conocimientos en la gestión de empresas y el marketing.

Compañía

Estructura del negocio

Easy-racetrack es una Sociedad Limitada (SL) gestionada por David López Ruedas y su socio.

Naturaleza del negocio

Easy-racetrack ofrece un servicio completo para los aficionados amateur del mundo del motor para trasladar de manera cómoda y fácil sus vehículos a los recintos de competición. Easy-racetrack inicialmente será un intermediario en

el alquiler de vehículos y remolques y ofrecerá el servicio propio de alquiler de material para los eventos.

Industria

Easy-racetrack opera en el sector del alquiler de vehículos, y vende servicios que se podrían categorizar en el alquiler de vehículos ligeros y no autónomos.

Información de fondo

David tiene gran conocimiento en desarrollo informático y acerca del mundo del motor y el laborioso trabajo que es desplazarse con un vehículo a un evento competitivo y su socio tiene gran conocimiento sobre el manejo de empresas y experiencia en el mundo del marketing. Uniendo el conocimiento de ambos quieren atacar un nicho de mercado que nadie tiene: alquiler de vehículos y servicios para el transporte de vehículos a eventos competitivos.

Objetivos del negocio

Tener la web y la logística lista para inicios del siguiente año, con el comienzo de la nueva temporada de circuitos para los aficionados a las motocicletas. Para los aficionados a los coches no existe temporada, ya que durante todo el año hay eventos. Por tanto, empezar a inicios de año es la mejor idea ya que junta ambos territorios.

Al cabo de dos años, aspiran a obtener 200.000€ en ingresos anuales por la venta de sus servicios y pasar de ser intermediario al 100% y comenzar a adquirir vehículos propios para ser más independientes del mercado y poder ofrecer precios más competitivos.

Equipo

David Lopez y su socio serán los propietarios y únicos empleados a tiempo completo. Sin embargo, contratarán a abogados externos para temas legales ya que carecen de estos conocimientos.

Análisis de mercado

Tamaño del mercado

Cuando se me ocurrió el proyecto no tenía conocimiento si había alguien que se dedicase a este nicho de mercado. Por tanto tuve que realizar un poco de investigación buscando información y hablando con gente del mundo del motor.

En la búsqueda por internet, encontré diversas empresas especializadas en el transporte de motocicletas y coches no de forma individual, que es una de las cosas que yo tenía en mente ofrecer como servicio. También encontré, como esperaba, empresas que se dedicaban a alquilar solamente furgonetas o solamente remolques. Todo esto sin ofrecer ninguno de los servicios extras de material necesario que yo tengo en mente ofrecer.

En cuanto a cifras, encontré que en Cataluña hay alrededor de 600 empresas de alquiler de vehículos, pero ninguna especializada en el nicho de mercado que yo estoy enfocado.

Trabajo de investigación focalizada

Después de obtener esta información y realizar estos planteamientos, comencé a hablar con amigos y otra gente del mundo del motor, tanto apasionados de los coches como de las motocicletas. La gente que tiene coches de circuito, pero no tiene un presupuesto muy elevado, me comentaron diversos puntos que me ayudaron a reafirmar lo que yo pensaba.

Primeramente, que muchas veces tienen que desplazarse hasta los circuitos con sus coches rodando. Esto no es muy recomendable, ya que ese tipo de vehículo suele estar únicamente pensado para rodar en circuito e incluso a veces llegan a carecer de ITV u homologaciones de algunas de sus modificaciones.

Como segundo punto, me comentaron que muchas veces intentan buscar camiones de transporte de coches, lo cual es fácil, pero no suele salir económico, ya que tiene que contratarlo individualmente. Por esta parte, yo podría abaratar mucho los costes, ya que yo reservaría el camión entero y en el momento de ofrecer mi servicio, podría hacerlo a un coste mucho más económico que lo que suelen conseguir ellos.

Análisis competitivo

Empresas de alquiler de furgonetas

Existen muchas empresas de alquiler de vehículos, las cuales alquilan furgonetas, como es lógico, pero solamente ofrecen eso. Estas empresas son un competidor pero a la vez les saco del terreno ofreciendo al cliente todo lo necesario para poder cargar la motocicleta en la furgoneta, como son cinchas y rampa. También, ofreceré el servicio de entrega en una dirección del vehículo alquilado, algo que las empresas tradicionales no ofrecen.

Empresas de alquiler de remolques

Existen diversas empresas de alquiler de remolques, que son competidoras directas, pero al igual que con el anterior competidor, no ofrecen un servicio completo. Yo ofreceré las cinchas y otro material necesario para las competiciones en circuito, como calentadores de ruedas, caballetes para las motocicletas o bidones de gasolina.

Productos y servicios

Como servicios, ofreceré remolques y furgonetas de alquiler, siendo un intermediario y buscando los mejores precios y acuerdos con las diferentes compañías de alquiler de vehículos y remolques.

En cuanto a productos, ofreceré todo el material necesario para el transporte de los vehículos y la estancia en el circuito. Rampas, cinchas, bidones de gasolina, caballetes y calentadores.

Plan de marketing

Productos

Ofrecemos diversos productos de alquiler, los cuales no podrán ser alquilados si no se adquiere un servicio de furgoneta o remolque. Esto es debido a que el negocio se basa en el alquiler de estos y no de los productos.

- Juego de 2 calentadores para motocicleta
- Juego de 2 caballetes para motocicleta
- Pack de 4 cinchas para motocicleta
- Pack de 4 cinchas para coche
- Rampa para subir la motocicleta a la furgoneta
- Bidones de gasolina de 15L y 30L

Servicios

Easy-racetrack ofrece servicios de alquiler de remolques para motocicletas y coches, furgonetas para motocicletas y transportes en grupo para estos también.

- Remolque para una motocicleta
- Remolque para dos motocicletas
- Remolque para 3 motocicletas

- Remolque para 1 quad
- Remolque para 1 coche
- Transporte en camión para coche
- Transporte en camión / furgoneta para motocicleta
- Furgoneta para 1 o 2 motocicletas de carretera o hasta 3 de montaña
- Furgoneta para 3 o 4 motocicletas de carretera o hasta 6 de montaña

Precios

Dado que la idea es que los clientes adquieran productos cuando compren los servicios, se harán packs para ser más competitivos.

- Furgoneta+cinchas+bidón+rampa: precio furgoneta + 25€
- Furgoneta+calentadores+caballetes: precio furgoneta + 50€
- Furgoneta+cinchas+bidón+rampa+calentadores+caballetes: precio furgoneta + 60€
- Furgoneta+rampa+calentadores: precio furgoneta + 35€
- Remolque+cinchas+bidón+rampa: precio remolque + 25€
- Remolque+calentadores+caballetes: precio remolque + 50€
- Remolque+cinchas+bidón+rampa+calentadores+caballetes: precio remolque + 60€
- Remolque+rampa+calentadores: precio remolque + 35€
- Transporte de coche en camión :1,5 € el kilómetro
- Transporte de moto en camión: 1,25€ el kilómetro

Distribución

Easy-racetrack será una tienda online, por lo que los clientes harán las reservas online. Dispondrán de la posibilidad de la recogida en nuestras instalaciones o bien la entrega en el domicilio por un cargo extra.

Catálogo de llamadas a la API

Cart

GetOne

https://easy-racetrack.herokuapp.com/cart/<string:id>, methods = [GET]

PostOne

https://easy-racetrack.herokuapp.com/cart, methods = [POST], requested data : {userId:String, totalPrice:Int, productsList:List}

GetCartProducts

https://easy-racetrack.herokuapp.com/cartProducts/<string:id>, methods = [GET]

AddProductToCart

https://easy-racetrack.herokuapp.com/addProduct/<string:userId>/<string:productId>, methods = [PATCH]

DeleteProductFromCart

https://easy-racetrack.herokuapp.com/deleteProduct/<string:userId>/<string:productId>, methods = [PATCH]

User

GetOne

https://easy-racetrack.herokuapp.com/user/<string:id>, methods = [GET]

GetAll

https://easy-racetrack.herokuapp.com/users, methods = [GET]

PostOne

https://easy-racetrack.herokuapp.com/user, methods = [POST], requested data : {name:String, surname:String, mail:Email,password:String}

UpdateOne

https://easy-racetrack.herokuapp.com/user/<string:id>, methods = [PUT], requested data: (name, surname, password)

Product

GetOne

https://easy-racetrack.herokuapp.com/product/<string:id>, methods = [GET]

GetAll

https://easy-racetrack.herokuapp.com/products, methods = [GET]

PostOne

https://easy-racetrack.herokuapp.com/user, methods = [POST], requested data : {productId:Int, name:String, description:String, price:Int, amountAvailable:Int, canRentedAlone:Boolean, type:String}

UpdateOne

https://easy-racetrack.herokuapp.com/user/<string:id>, methods = [PUT], requested data: {name:String, description:String, price:Int, amountAvailable:Int, canRentedAlone:Boolean, type:String}

Booking

GetOne

https://easy-racetrack.herokuapp.com/booking/<string:id>, methods = [GET]

GetAll

https://easy-racetrack.herokuapp.com/bookings, methods = [GET]

GetAllByUser

https://easy-racetrack.herokuapp.com/bookings/<string:userId>, methods = [GET]

PostOne

https://easy-racetrack.herokuapp.com/user, methods = [POST], requested data : {userId:String, bookingId:String, totalAmount:Int, productsList:List, shipmentInfo:Dict, paymentInfo:Dict}

UpdateOne

https://easy-racetrack.herokuapp.com/user/<string:id>, methods = [PUT], requested data: {totalAmount:Int, productsList:List, shipmentInfo:Dict, paymentInfo:Dict}

GetBookingProducts

https://easy-racetrack.herokuapp.com/bookingProducts/<string:id>, methods = [GET]

Diagramas UML

Diagrama de contexto

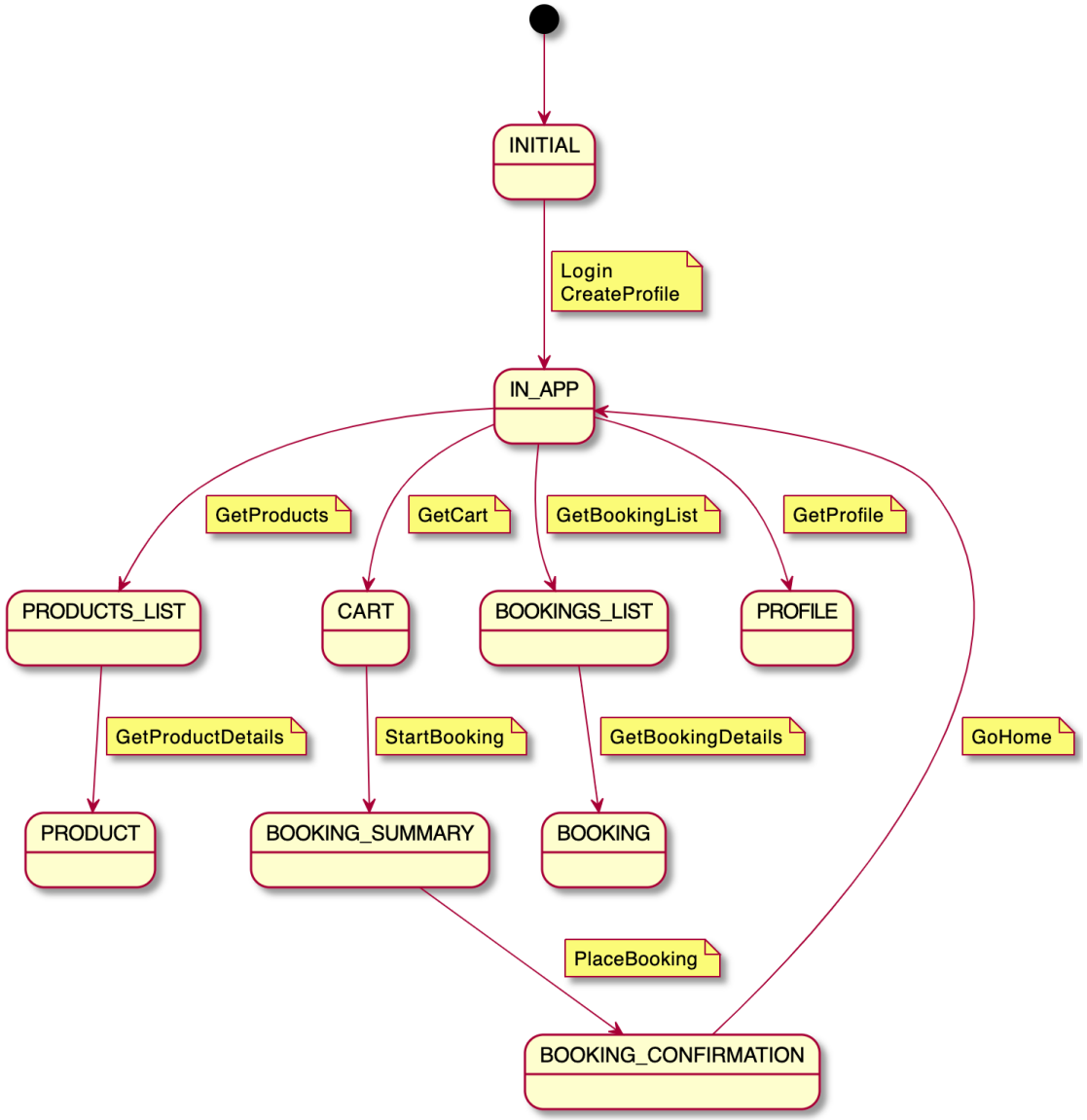


Diagrama de paquetes

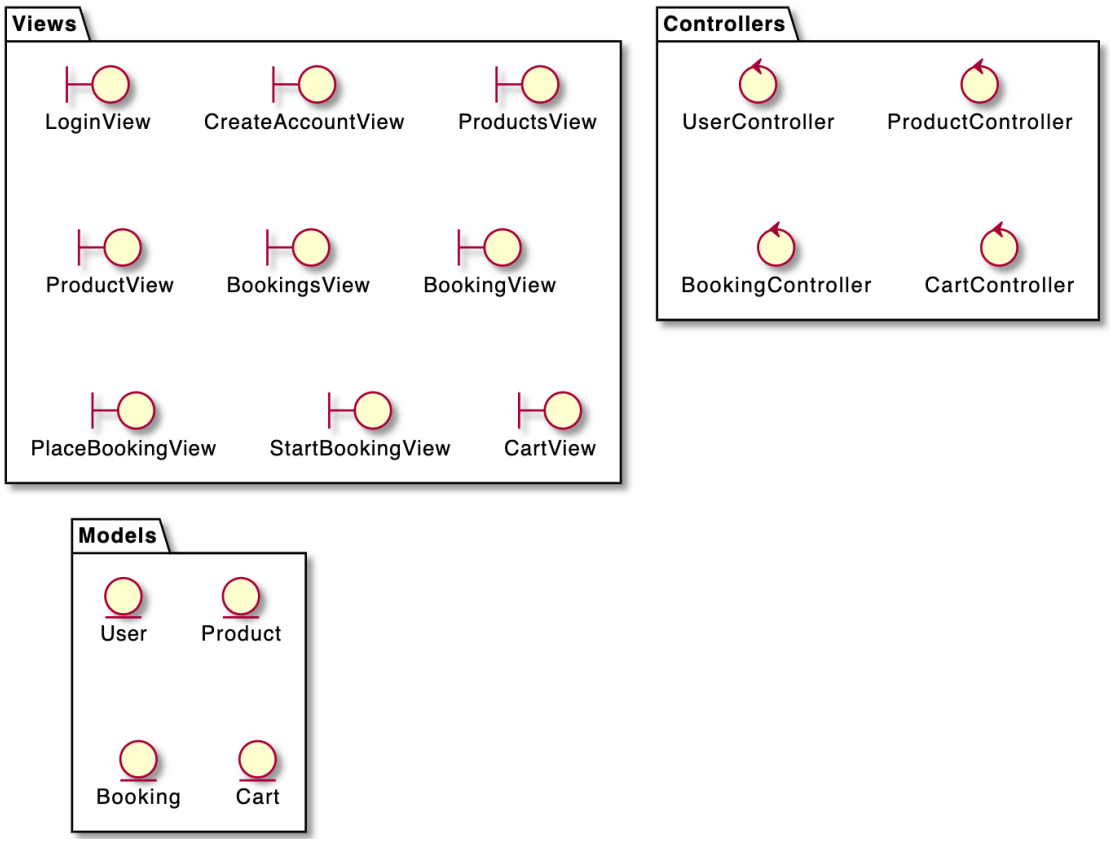


Diagrama del modelo del dominio

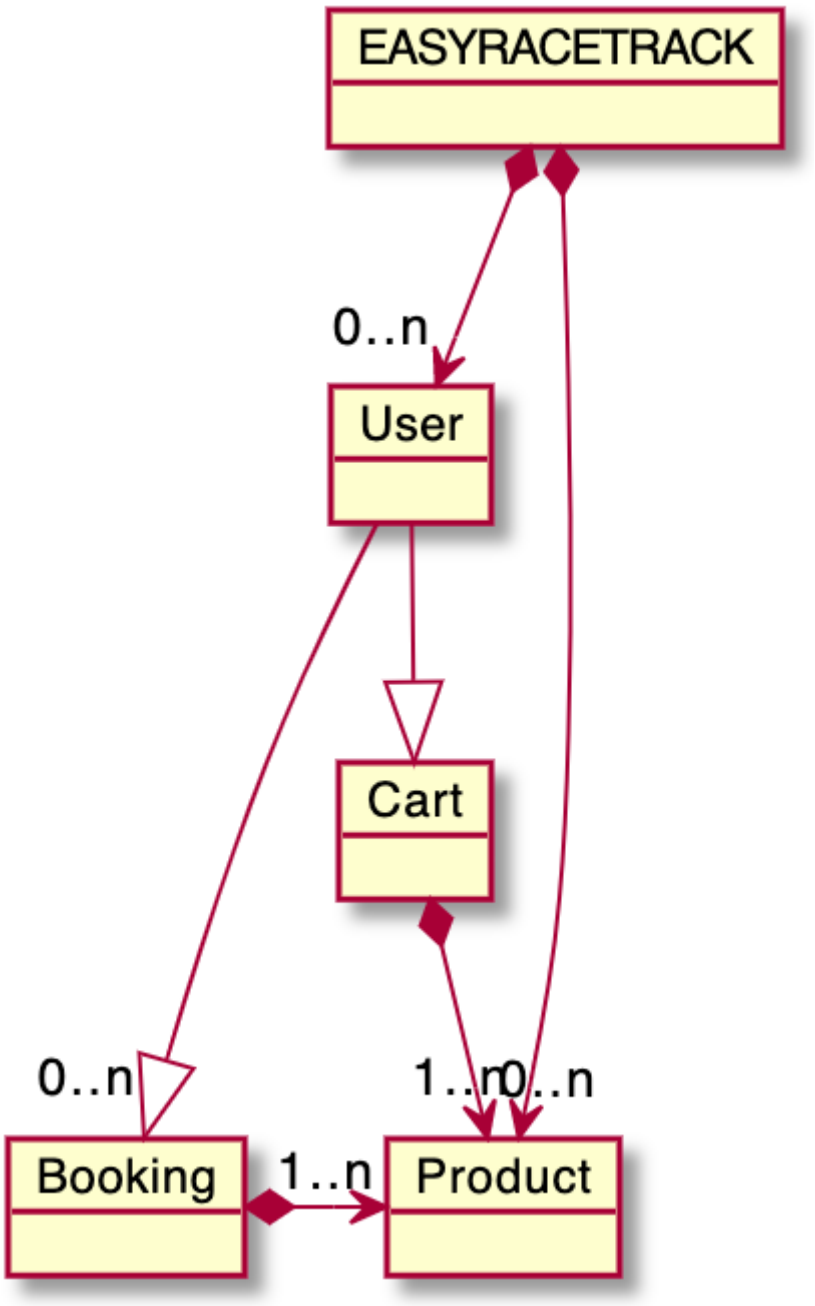
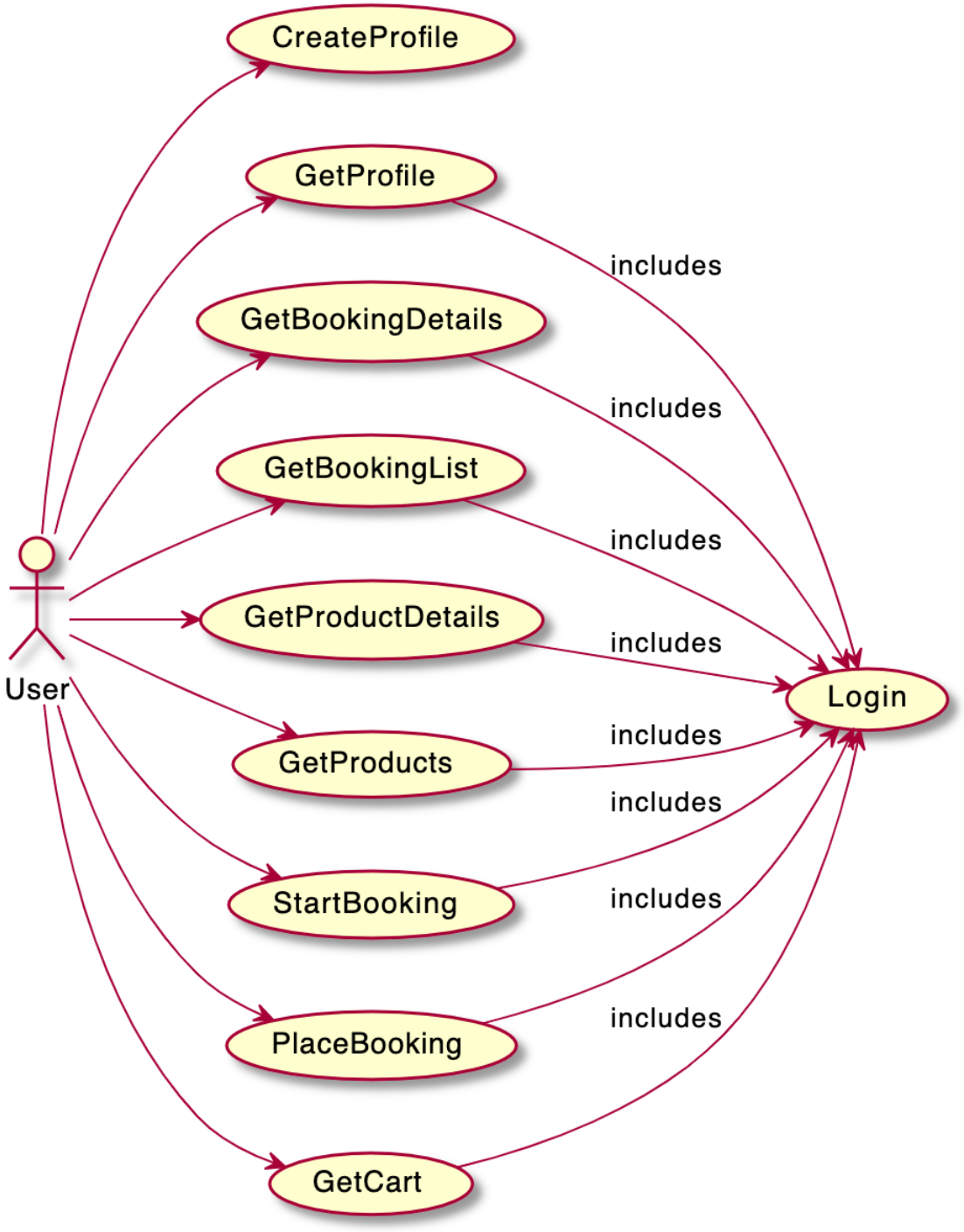
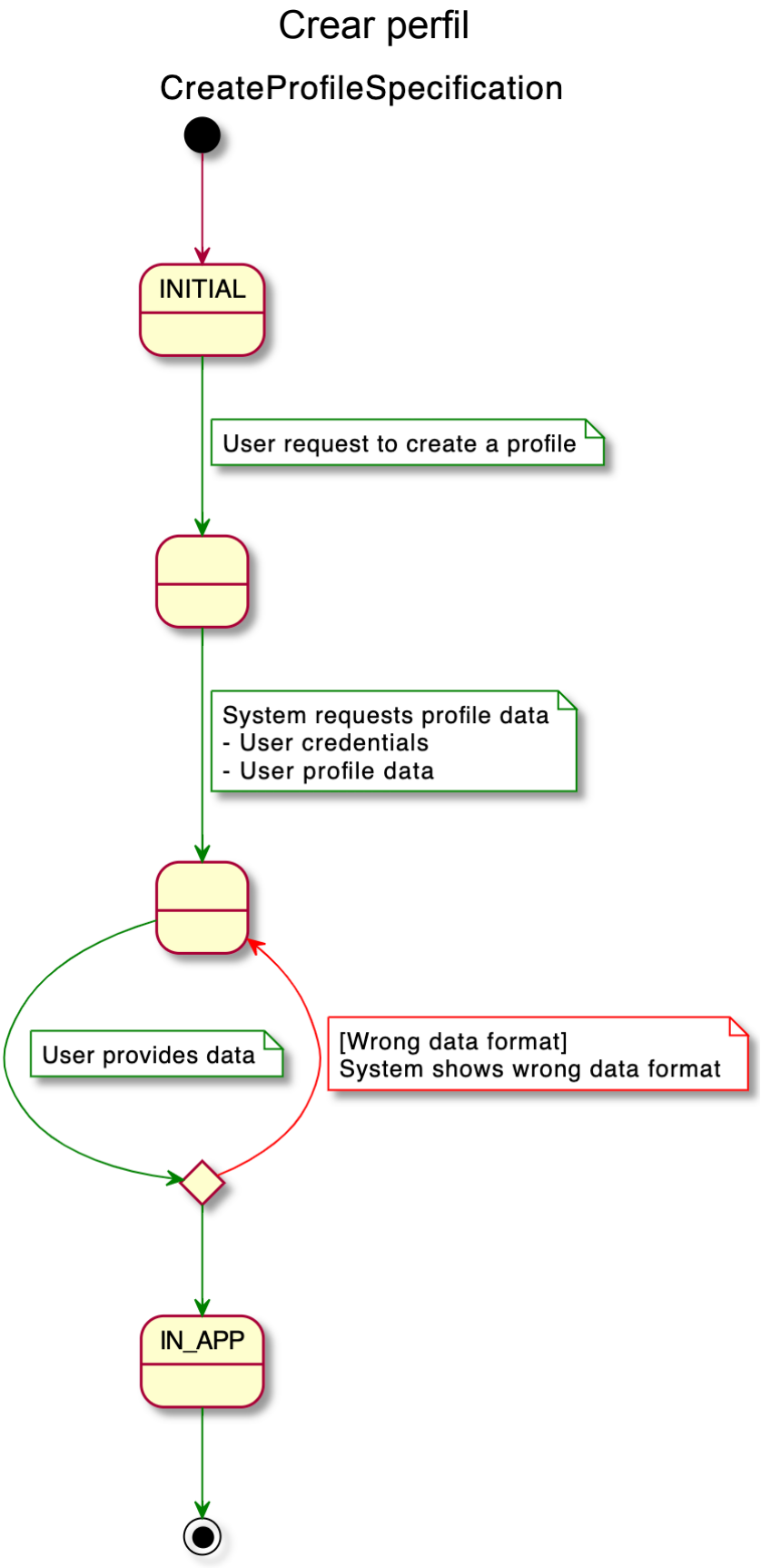


Diagrama de relación entre actor y casos de uso

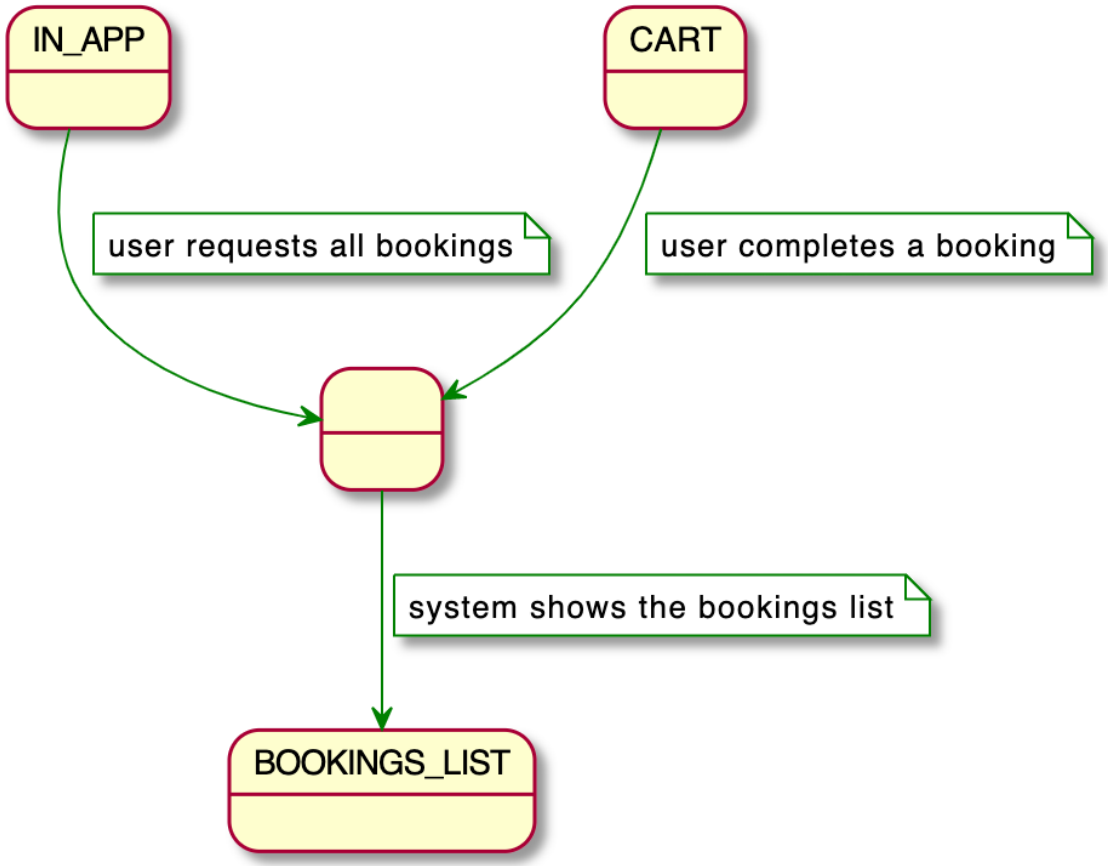


Diagramas de casos de uso

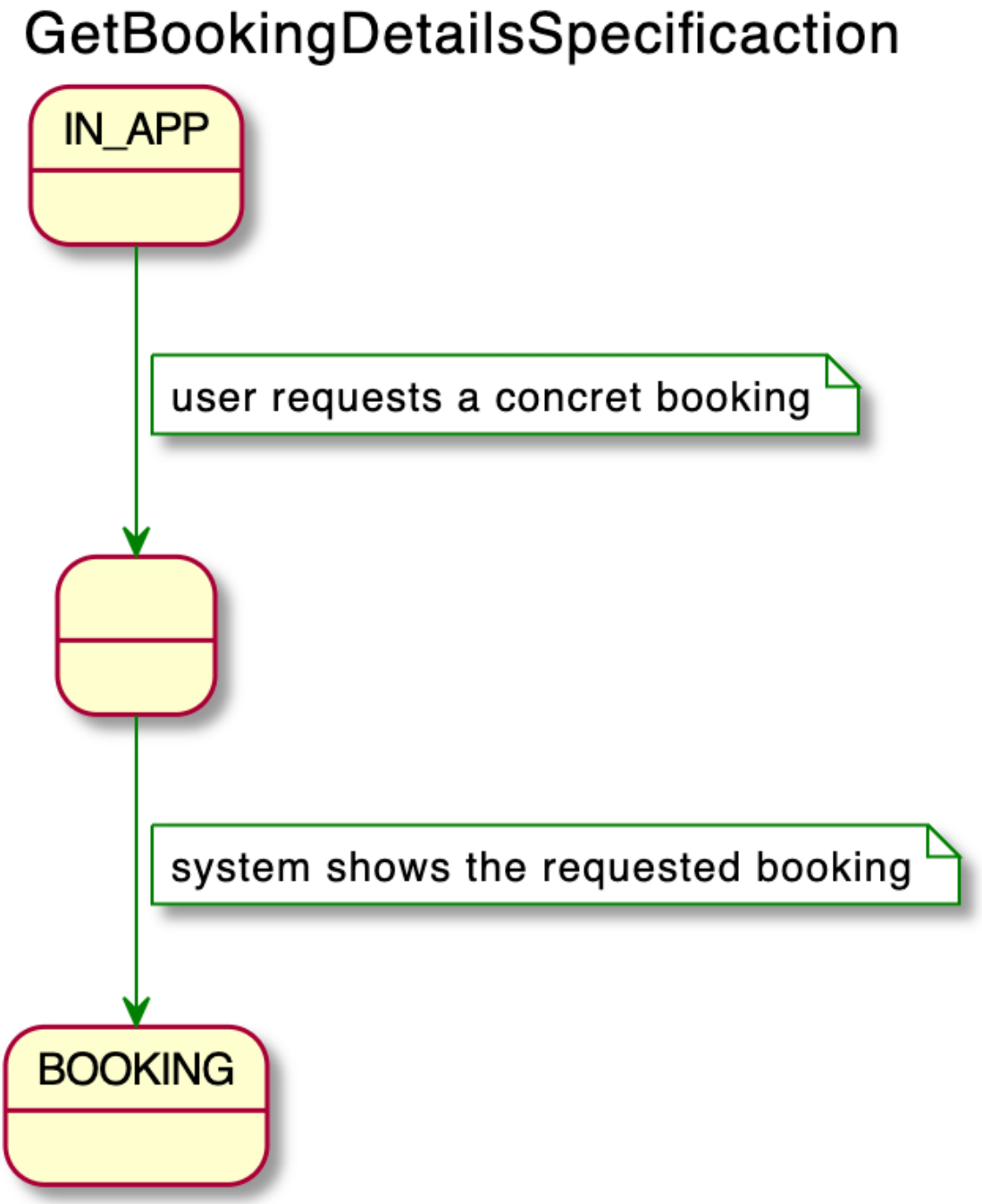


Ver mis reservas

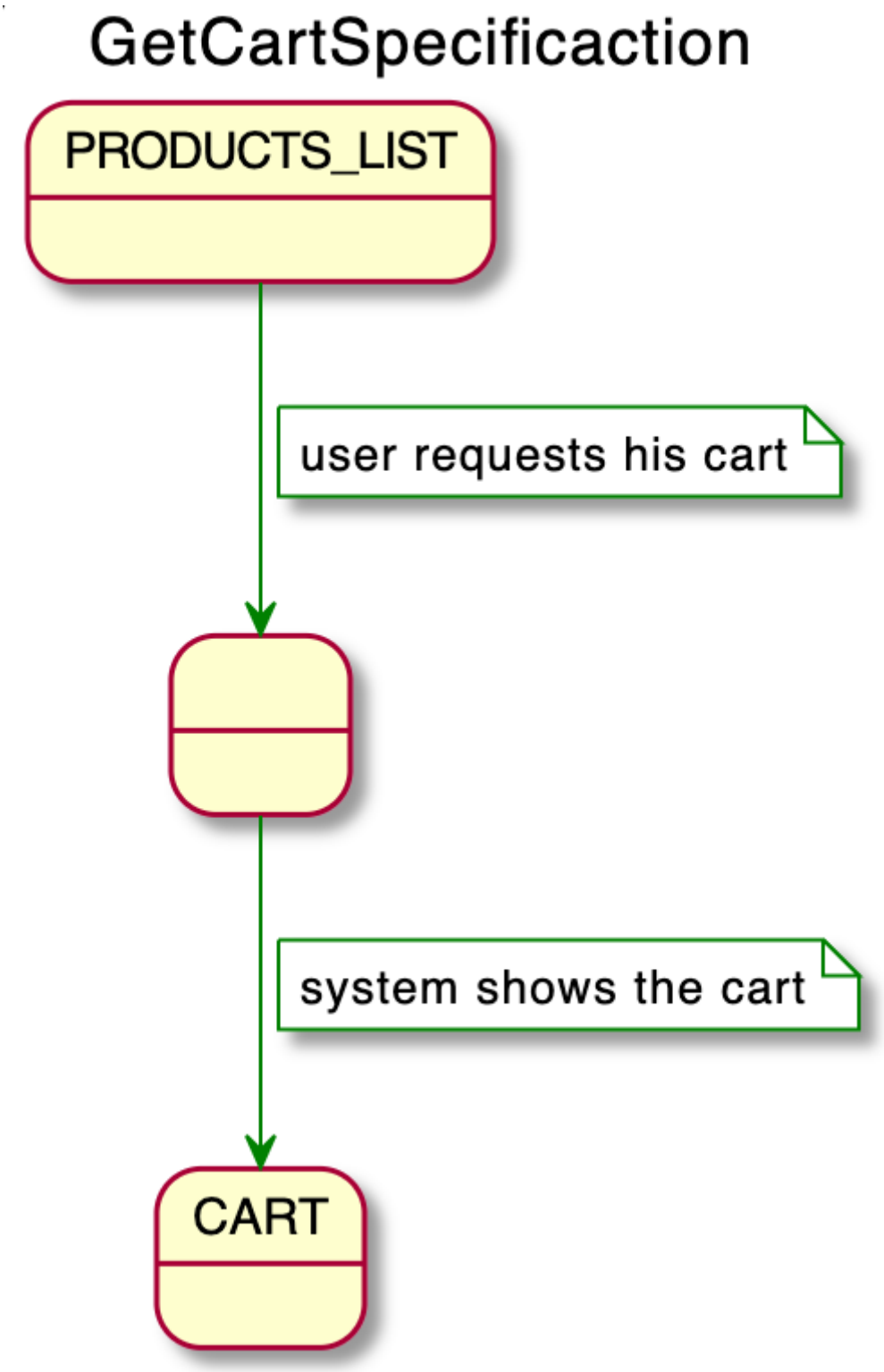
GetBookingsSpecificaction



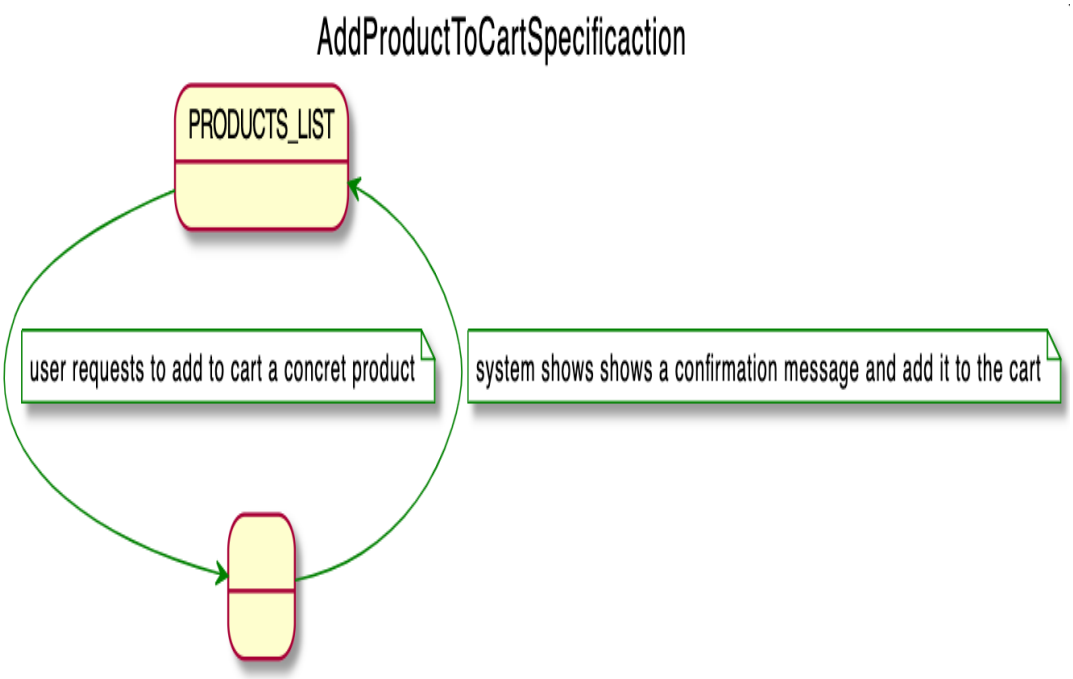
Ver mi reserva en detalle



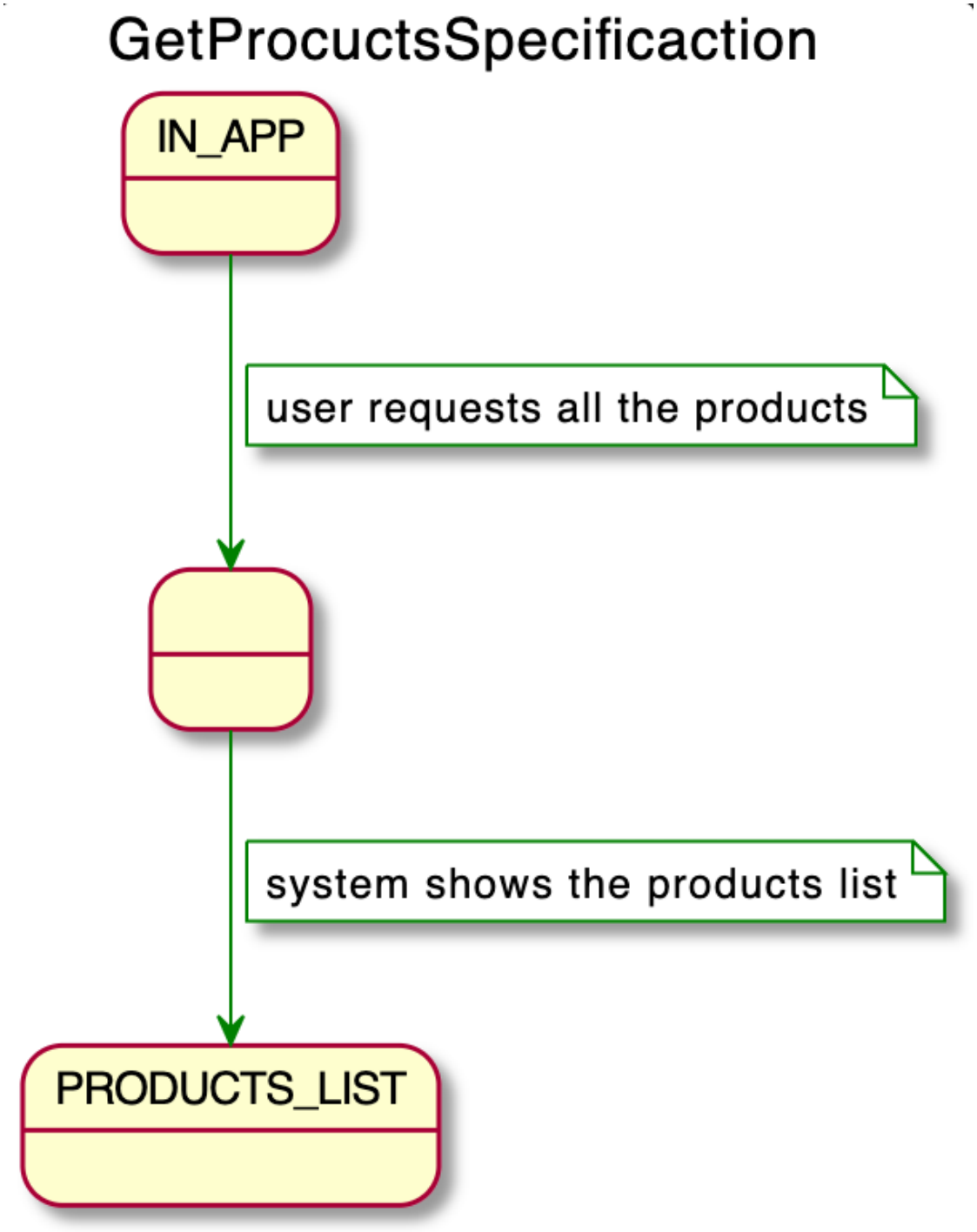
Ver mi cesta


GetCartSpecificaction
PRODUCTS_LIST
user requests his cart
system shows the cart
CART

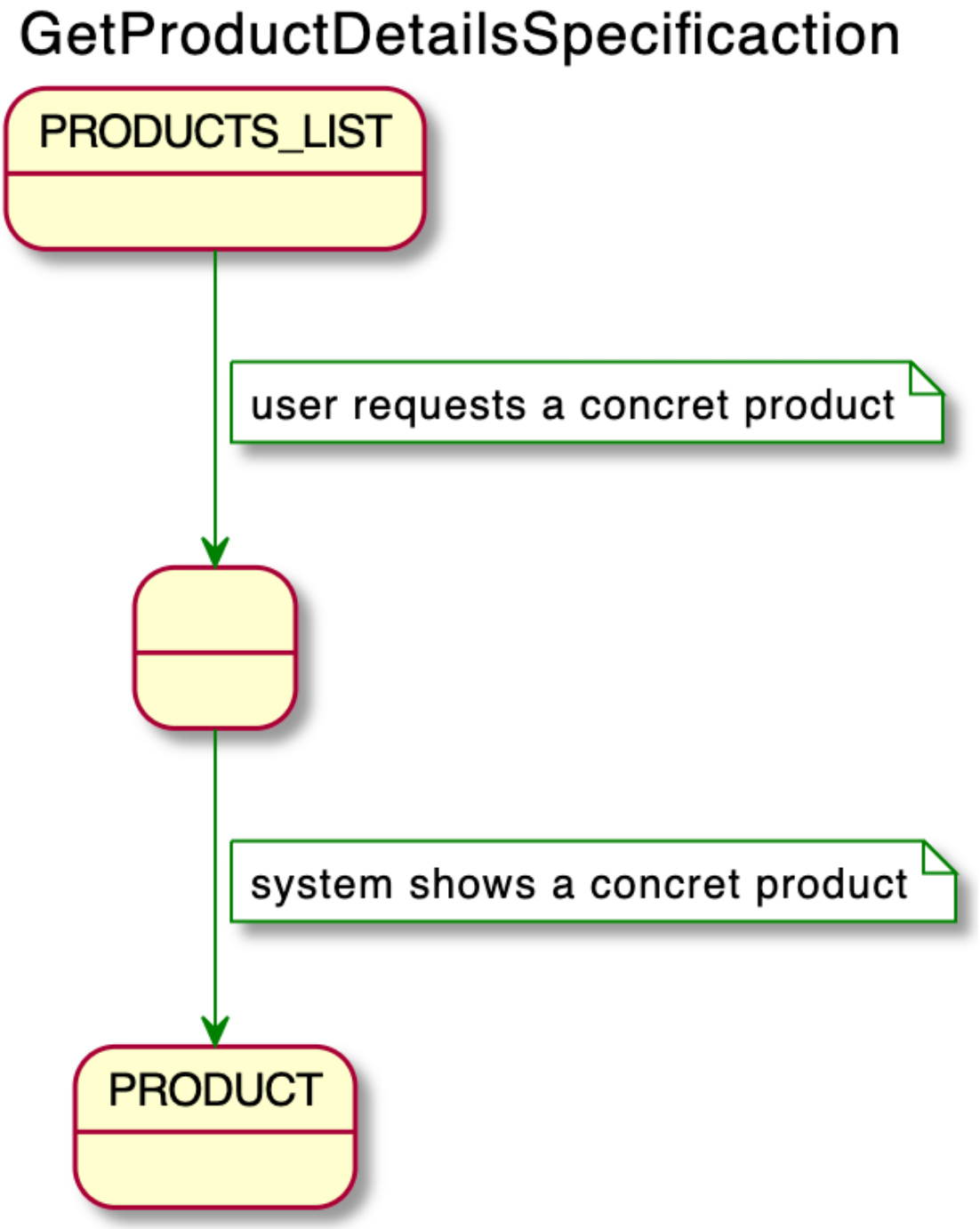
Añadir producto a mi cesta



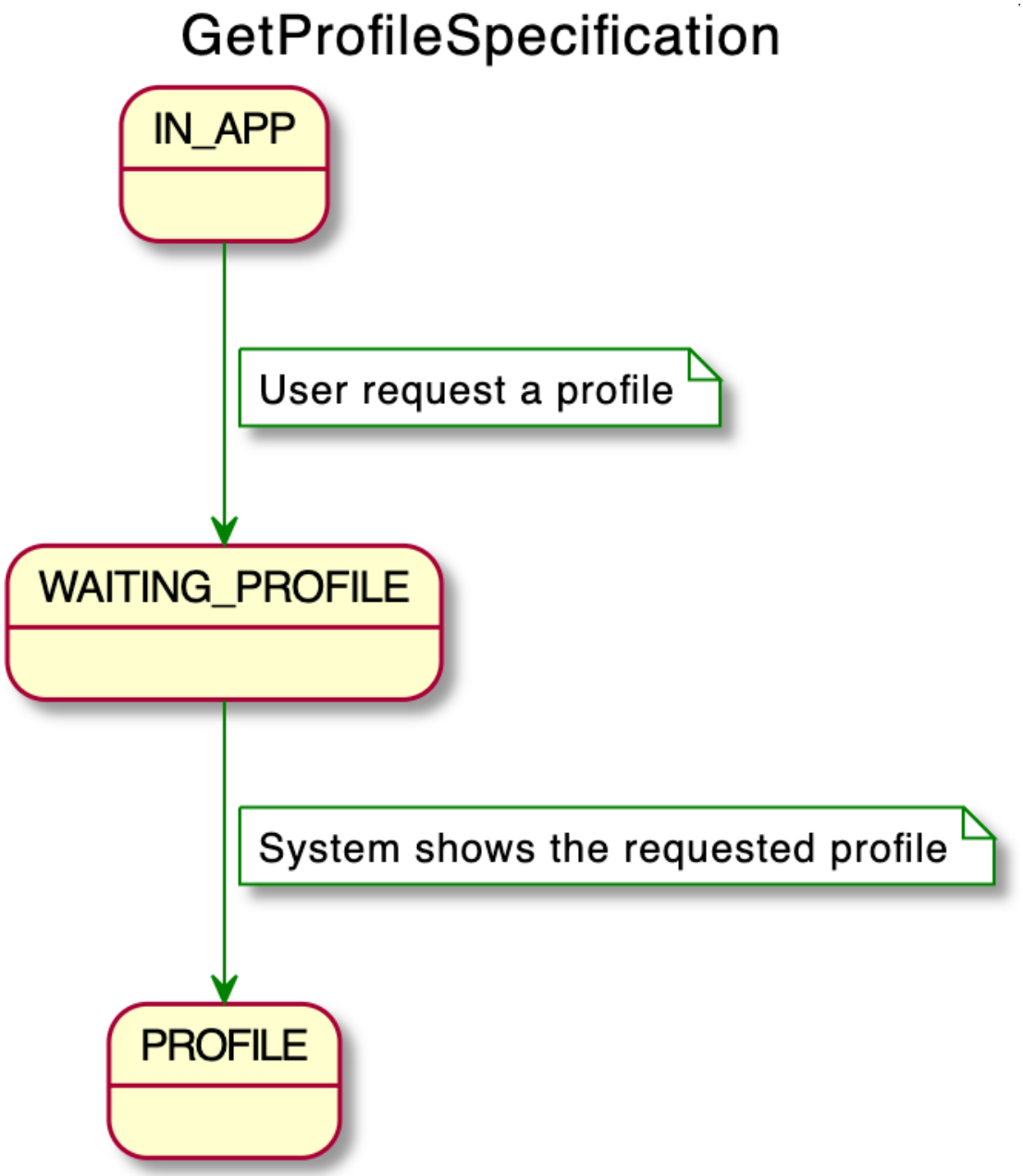
Ver productos

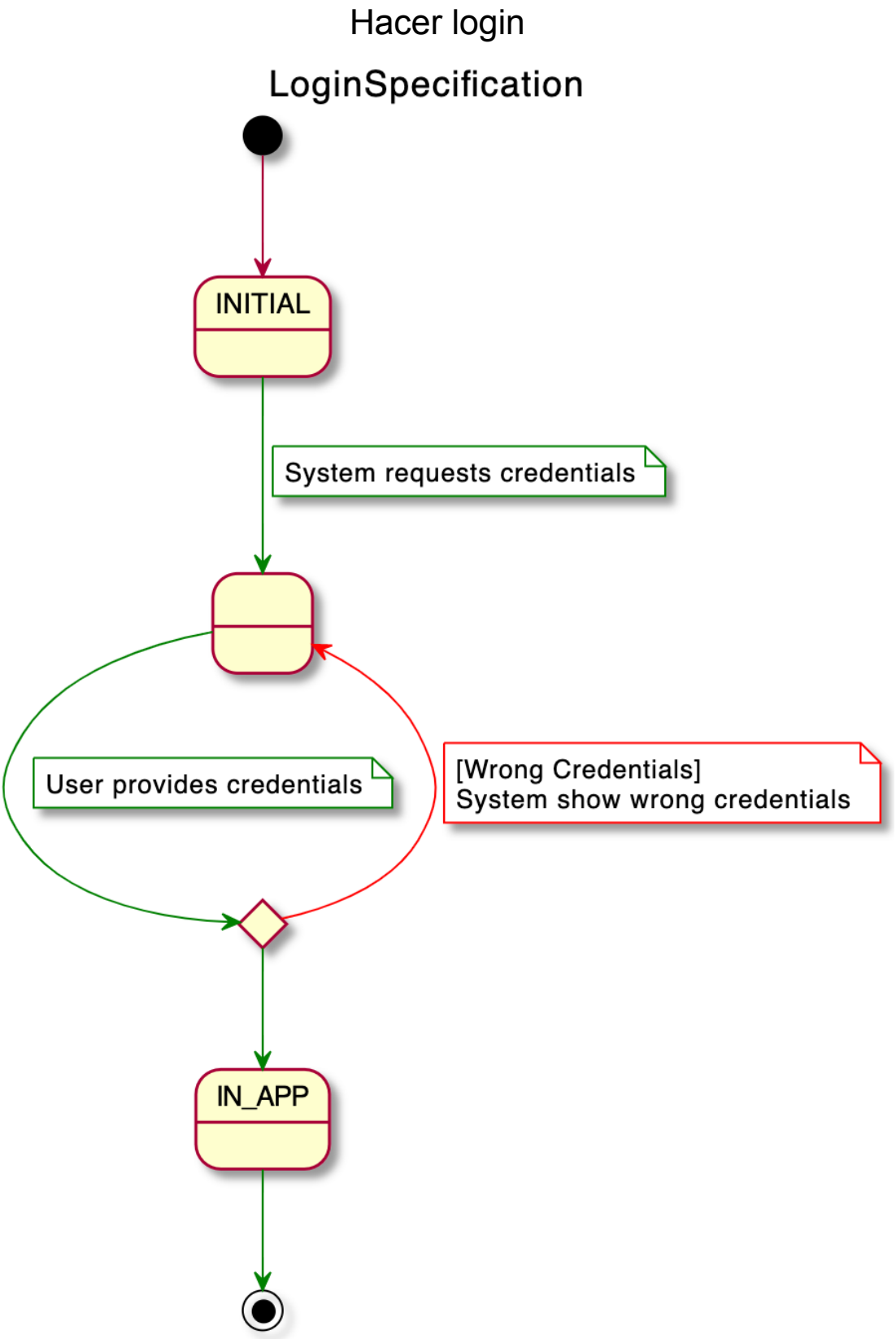


Ver producto en detalles



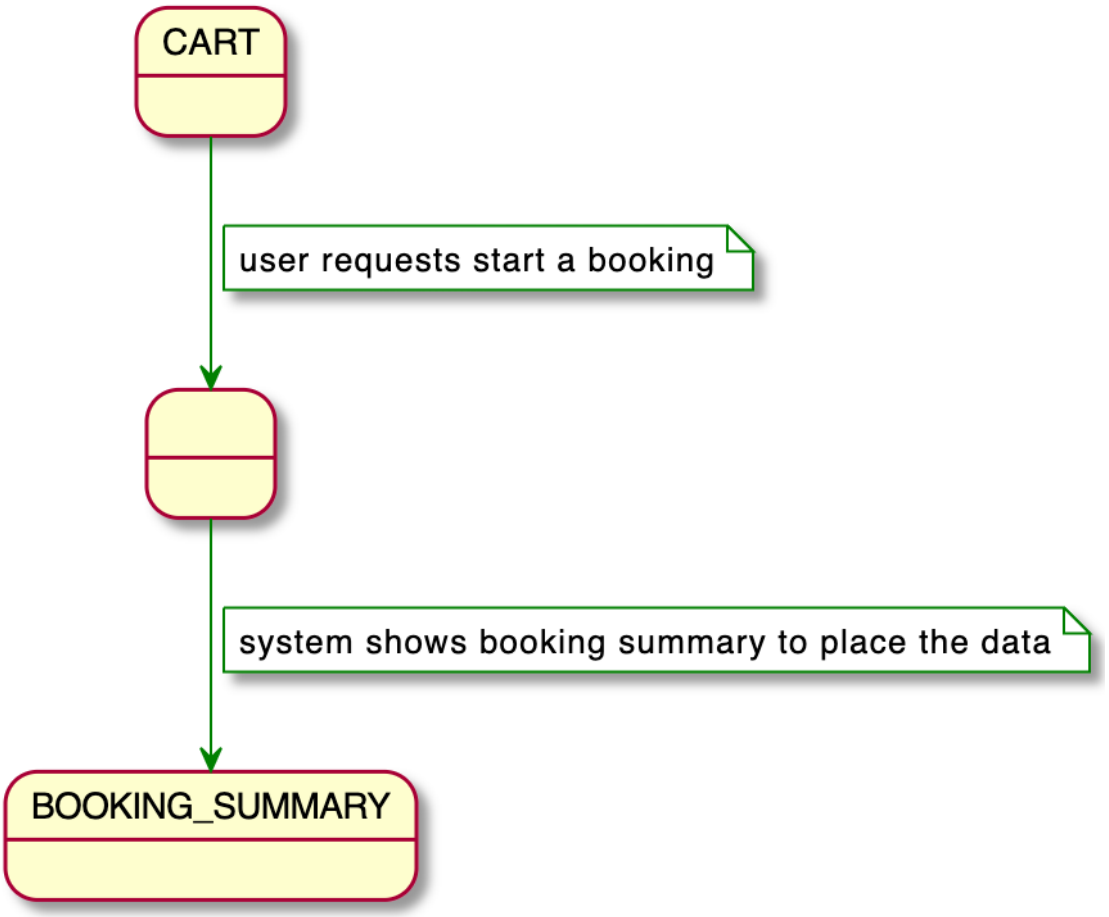
Ver mi perfil



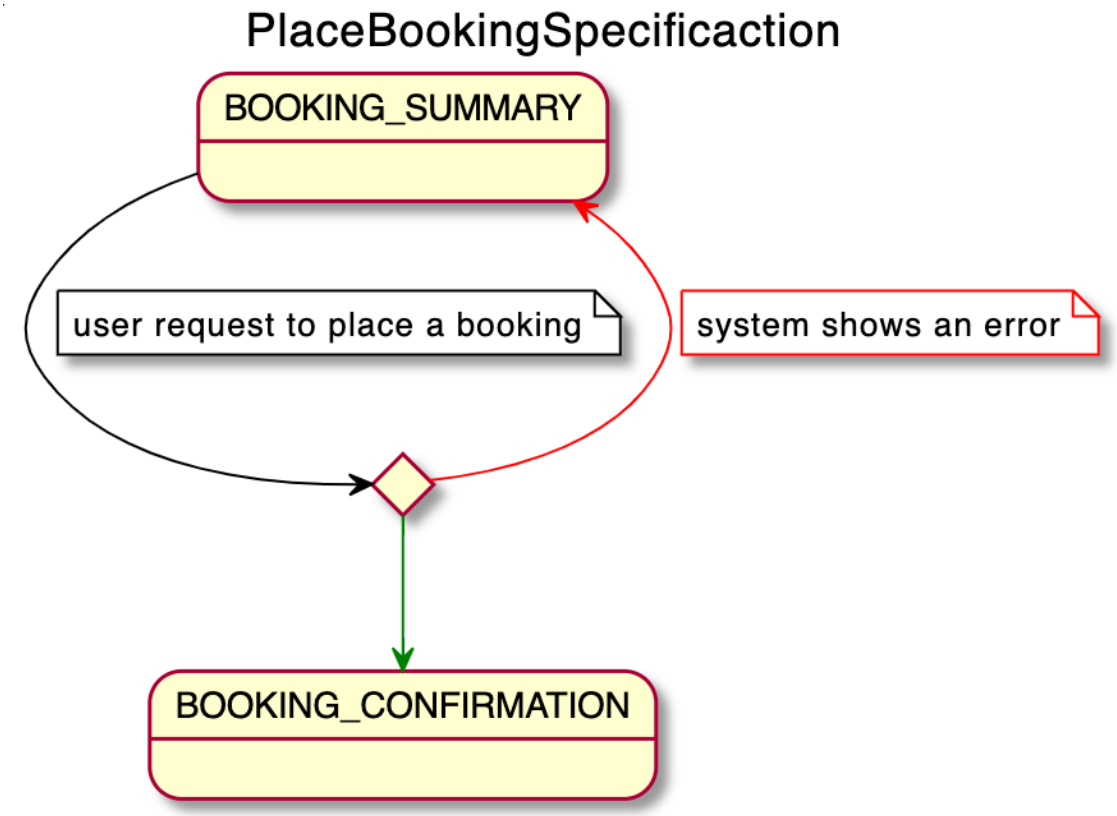

Hacer login
LoginSpecification
INITIAL
System requests credentials
User provides credentials
[Wrong Credentials]
System show wrong credentials
IN_APP

Empezar una reserva

StartBookingSpecificaction



Confirmar una reserva



User stories

1. Crear una cuenta

CÓMO nuevo usuario QUIERO acceder a la web PARA poder hacer reservas.

Acceptance criteria:

- Creación correcta: DADOS unos datos no registrados en la web CUANDO intento crear una cuenta ENTONCES obtengo un Toast de éxito y acceso a la web.
- Creación incorrecta: DADOS unos datos ya registrados en la web CUANDO intento crear una cuenta ENTONCES obtengo un Toast de error no puedo acceder a la web.

2. Hacer Log In

CÓMO usuario ya registrado QUIERO ver productos de mininteres PARA realizar reservas.

Acceptance criteria:

- Acceso correcto: DADOS unos datos registrados en la web CUANDO intento acceder a la web ENTONCES obtengo un Toast de éxito y acceso a la web.

- Acceso incorrecto: DADOS unos datos no registrados en la web CUANDO intento acceder a la web ENTONCES obtengo un Toast de error no puedo acceder a la web.

- Acceso incorrecto: DADA una contraseña diferente a la asociada a mi usuario CUANDO intento acceder a la web ENTONCES obtengo un Toast de error no puedo acceder a la web.

3. Ver productos en detalle

CÓMO usuario ya registrado QUIERO ver un producto en detalle PARA saber si se adecua a mis necesidades

Acceptance criteria:

- Visualización correcta: DADO el producto que quiero ver en detalle CUANDO hago click sobre él en el catálogo principal ENTONCES veo toda la información disponible sobre este.

4. Añadir productos a mi cesta

CÓMO usuario ya registrado QUIERO ver el catálogo de productos PARA añadirlos a mi cesta.

Acceptance criteria:

- Adición correcta: DADOS los productos que se adaptan a mis necesidades CUANDO estoy mirando el catálogo de productos ENTONCES añado el producto a mi cesta, ya que está disponible.

- Adición incorrecta: DADOS los productos que se adaptan a mis necesidades CUANDO estoy mirando el catálogo de productos ENTONCES no puedo añadir el producto que quiero ya que no está disponible

5. Realizar una reserva

CÓMO usuario ya registrado QUIERO ver mi cesta PARA tramitar mi reserva.

Acceptance criteria:

- Reserva correcta: DADOS los productos he añadido previamente a mi cesta CUANDO esto dentro de la cesta ENTONCES procedo a realizar la reserva añadiendo mis datos personales y de pago.

- Reserva incorrecta: DADOS los productos que se adaptan a mis necesidades CUANDO estoy mirando el catálogo de productos ENTONCES no puedo añadir el producto que quiero ya que no está disponible

6. Ver mis reservas

CÓMO usuario ya registrado QUIERO ver un registro de mis reservas PARA saber los datos de esta.

Acceptance criteria:

- Visualización correcta: DADAS las reservas realizadas previamente CUANDO entro en el resumen de reservas ENTONCES puedo ver toda la información de mis reservas.

7. Revisar mis datos de usuario

CÓMO usuario ya registrado QUIERO mis datos personales con los cuales me he registrado PARA cambiar alguno en caso de que me sea necesario.

Acceptance criteria:

- Revisión correcta: DADOS mis datos de registro CUANDO accedo a mi perfil ENTONCES puedo revisar mi información y solicitar a soporte algún cambió si fuese necesario.

8. Recibir soporte

CÓMO usuario ya registrado QUIERO poder solicitar ayuda con cualquier problema que tenga o haya tenido PARA llevar a cabo o modificar una reserva.

Acceptance criteria:

- Caso correcto: DADOS un número de teléfono y un mail CUANDO necesito ayuda con alguna de mis reservas ENTONCES obtengo un medio para contactar y recibir ayuda.

9. Realizar reservas grupales

CÓMO usuario ya registrado QUIERO acceder a una sección donde pueda ver otros usuarios con las mismas necesidades que yo PARA poder realizar una reserva conjunta.

Acceptance criteria:

- Caso correcto: DADA una sección donde la gente publique sus futuras reservas de algún servicio CUANDO necesito algo similar a esa persona que ha publicado ENTONCES le contacto para poder compartir el servicio.