

E-COMMERCE CON MODELO 3D INTERACTIVO

JOSE ÁNGEL DOMÍNGUEZ ESPINACO

Trabajo fin de Grado

Supervisado por Dr. Pablo Trinidad Martín-Arroyo



Universidad de Sevilla

mayo 2021

Publicado en mayo 2021 por
Jose Ángel Domínguez Espinaco
Copyright © MMXXI
sepinaco@gmail.com

Esta obra está licenciada bajo la Licencia Creative Commons
Atribución-CompartirIgual 4.0 Internacional. Para ver una copia de esta licencia,
visita <http://creativecommons.org/licenses/by-sa/4.0>

Yo, D. Jose Ángel Domínguez Espinaco con NIF número 29519653G,

DECLARO

mi autoría del trabajo que se presenta en la memoria de este trabajo fin de grado que tiene por título:

E-commerce con modelo 3D interactivo

Lo cual firmo,

Fdo. D. Jose Ángel Domínguez Espinaco
en la Universidad de Sevilla
27/05/2021

Dedicado a todos los profesores, canales de Youtube y webs de desarrollo que han contribuido en mi aprendizaje.



AGRADECIMIENTOS

Agradecimiento especial a la Universidad de Sevilla y sus profesores por haberme dado los conocimientos bases necesarios para poder continuar con mi aprendizaje en el mundo del desarrollo. A mi tutor, Pablo, por haberme encaminado en el desarrollo de aplicaciones 3D con WebGL y por todos sus consejos. También a todas las personas que hacen que Internet se haya convertido en un lugar de aprendizaje gratuito y de calidad. Por último, agradezco a mi familia por permitirme enfocarme exclusivamente en el estudio y en especial a mi madre, por haber contribuido en ellos durante mi niñez y adolescencia.

RESUMEN

Este proyecto pretende realizar una introducción al desarrollo de aplicaciones web e-commerce, utilizando la tecnología WebGL, permitiéndonos mostrar gráficos en 3D en el propio navegador. Esto permite tener un modelo 3D interactivo del producto a vender en la propia web, para acercar aún más al cliente con el producto que se está comercializando.

Las tecnologías empleadas para realizar este proyecto han sido: React (biblioteca Javascript para la creación de interfaces de usuario), Three.js (librería de gráficos que utiliza por debajo WebGL), react-three-frame (unión de React con Three.js), Gatsby (framework basado en React para crear páginas webs estáticas), Stripe (plataforma de pago que nos ofrece la posibilidad de crear productos dentro de su plataforma, además de proporcionarnos la infraestructura técnica de prevención de fraude en los pagos con tarjeta bancaria), GraphQL (lenguaje de consultas por el cual se comunica nuestra aplicación Gatsby con Stripe), Docker (herramienta que facilita la configuración para el despliegue del proyecto) y DigitalOcean (plataforma de hosting utilizada).

ÍNDICE GENERAL

I	Introducción	1
1.	Contexto	3
1.1.	El mundo del e-commerce	4
1.2.	Estado del arte	5
2.	Contexto	7
2.1.	Motivación	8
2.2.	Listado de objetivos	8
II	Organización del proyecto	9
3.	Metodología	11
3.1.	Estructura organizacional del proyecto	12
3.2.	Metodología de desarrollo	12
4.	Planificación	13
4.1.	Resumen temporal del proyecto	14
4.2.	Planificación inicial	14
4.3.	Informe de tiempos del proyecto	15
5.	Costes	17

ÍNDICE GENERAL

5.1.	Resumen de costes del proyecto	18
5.2.	Costes de personal	18
5.3.	Costes materiales	18
5.4.	Costes indirectos	19
III	Desarrollo del proyecto	21
6.	Análisis	23
6.1.	Análisis de las tecnologías empleadas	24
7.	Desarrollo	27
7.1.	Antes de comenzar	28
7.2.	Inicio de un proyecto con Gatsby	28
7.3.	Conexión a la API de Stripe	29
7.4.	Obtención de productos por medio de GraphQL	32
7.5.	Creación de la escena 3D con react-three-fiber	36
7.6.	Creación del fichero docker-compose.yaml	54
7.7.	Registro de un nombre de dominio	56
7.8.	Despliegue del proyecto en DigitalOcean	58
IV	Cierre del proyecto	67
8.	Conclusiones	69
8.1.	Informe post-mortem	70
8.1.1.	Lo que ha ido bien	70
8.1.2.	Lo que ha ido mal	71

8.1.3. Discusión	71
8.2. Trabajos futuros	71
V Bibliografía	73
9. bibliografia	75

ÍNDICE DE CUADROS

4.1.	Tabla resumen de tiempos y planificación	14
4.2.	Planificación temporal de iteraciones	14
4.3.	Planificación temporal de iteraciones	15
5.1.	Tabla resumen de costes	18

PARTE I

INTRODUCCIÓN

CONTEXTO

El objetivo de este capítulo es realizar una introducción al proyecto que hemos realizado.

1.1 EL MUNDO DEL E-COMMERCE

El e-commerce o comercio electrónico consiste en la compra y venta de productos o de servicios a través de internet.

Hay quien sitúa los orígenes del e-commerce con la llegada de la venta por catálogo (Estados Unidos, 1920), pero el e-commerce electrónico comienza en 1960, con la aparición de un sistema llamado Electronic Data Interchange (EDI). Dicho sistema permitía a las empresas realizar transacciones electrónicas e intercambios de información comercial.

En 1989 apareció la World Wide Web, esto es el Internet que conocemos hoy en día. Fue en los años 90, cuando se impulsó el comercio electrónico en la web y en 1995 aparecieron Ebay y Amazon (anteriormente llamados Auction Web y Cadabra) los cuales se convirtieron en un canal de venta masivo con clientes que llegaban desde cualquier parte del mundo gracias a Internet.

En España aparece en 1995 la web barrabes.com, que consistía (y consiste) en un e-commerce dedicado exclusivamente a la montaña y el alpinismo. A finales de los 90 el Corte Inglés también se suma al comercio electrónico online y con la llegada del abaratamiento del acceso a la banda ancha fija de Telefónica y la popularidad de los ordenadores domésticos, el comercio electrónico comienza a ser más popular y una opción rentable para muchos negocios. Posteriormente se sumaron empresas como Inditex y aparecieron nuevos portales de comercio electrónico como PCComponentes.

Con la llegada de los smartphones y la banda ancha móvil, las webs de comercio electrónico se dispararon. Comenzaron a aparecer startups enfocadas en los e-commerce. Ejemplos de esto fueron Privalia (outlet de ropa online), Groupalia (ofertas de ocio local, viajes, servicios y productos), Idealista (sector inmobiliario), El Tenedor (reservas de restaurantes), Wallapop (compra y venta de segunda mano) y muchas más.

Hoy en día el comercio electrónico forma parte de nuestras vidas. Las personas ya se han acostumbrado a este tipo de negocios y participan activamente en ellos introduciendo sus datos bancarios y realizando las compras. Ha demostrado ser un negocio mucho más rentable que tener la propia tienda física, y la mayoría de negocios de venta actuales se enfocan en tener también una tienda online disponible para los clientes. Además, con la llegada del coronavirus, la demanda de tiendas online se ha disparado todavía más, puesto que es una gran opción ante la situación de evitar que se formen

aglomeraciones de personas en un mismo lugar. Esta época que estamos viviendo va a fomentar que las personas se acostumbren todavía más a las compras online, así que pienso que es una buena decisión enfocarse en el desarrollo web de este tipo de aplicaciones.

1.2 ESTADO DEL ARTE

Hoy en día cualquier persona puede tener una tienda online. Existe una gran cantidad de herramientas que te facilitan la creación de una tienda. El punto negativo de estas herramientas es que a mayor facilidad de uso, mayor costo mensual o comisión de venta.

Actualmente plataformas como Instagram shopping permiten vender productos asociados a una cuenta de Instagram, y herramientas como Wix nos permiten crear páginas web y tiendas online fácilmente sin la necesidad de tener que aprender a programar.

Como opinión personal, La mayoría de tiendas que se ve por Internet son muy parecidas entre ellas, se utilizan plantillas parecidas y en muchas ocasiones presentan un rendimiento no muy bueno por haberse desarrollado por personas sin conocimientos de programación con herramientas fáciles de usar pero con la desventaja de ser poco optimizadas para el rendimiento.

El siguiente paso que se está dando en la web y en las tiendas online es la aparición de la tecnología WebGL complementando a la tecnología CSS. Un ejemplo de esto es reemplazar las imágenes de los productos por sus correspondientes modelos 3D y que el cliente tenga la posibilidad de interactuar con dichos productos modificando sus propiedades como el color, el tamaño o cualquier otra propiedad que se encuentre disponible. Un ejemplo real de esto es en la compra de un coche, la empresa nos ofrecerá un enlace web en donde se nos mostrará el coche y podremos cambiar el color del coche, de los asientos, añadirle características, etc. Y veremos en tiempo real como se verían esos cambios aplicado en el modelo 3D que se nos muestra en pantalla para facilitar al cliente su decisión de compra.

Este proyecto pretende ser una introducción a este nuevo enfoque, en donde mostraremos el modelo 3D de un bolso y daremos la opción de poder comprar dicho modelo dentro de nuestra web.

CONTEXTO

El objetivo de este capítulo es explicar los motivos por los que he realizado el proyecto y los objetivos que se tienen que cumplir para considerar al proyecto como finalizado.

2.1 MOTIVACIÓN

La motivación principal por la que realice este proyecto es para aprender a construir aplicaciones web interactivas con gráficos en 3D. Esto me permite realizar páginas webs visualmente muy superior a las realizadas utilizando tan solo css, además de abrirme las puertas a otros campos como el de la realidad virtual.

Los escenarios reales en donde se aplica esta tecnología son:

- E-commerce en donde se muestra un modelo 3D del producto a vender.
- Tours virtuales con imágenes en 360.
- Páginas webs con efectos visuales renderizados con WebGL.
- Proyectos artísticos que utilizan software de visualización.
- Proyectos de experiencias con realidad virtual para el navegador.
- Proyectos de videojuegos para el navegador.

Este proyecto también me sirve para aprender tanto el desarrollo frontend de la web como el desarrollo backend, de tal manera, que llegue a ser capaz de construir una aplicación web desde cero y alojarla en internet con un nombre de dominio real y cumpliendo con los certificados de seguridad como el https. Esto me permite tener la libertad de poder crear proyectos personales y alojarlos en internet, además de mejorar en mi futuro profesional acercándome a la figura de fullstack.

2.2 LISTADO DE OBJETIVOS

Objetivo 1. Analizar el estado actual del e-commerce.

Objetivo 2. Analizar las tecnologías actuales existentes para el desarrollo de este tipo de aplicaciones.

Objetivo 3. Desarrollo de una aplicación web e-commerce básica que muestre un modelo 3D del producto a vender.

Objetivo 4. Acercar al lector a la tecnología 3D con WebGL y a la creación de una aplicación web e-commerce, explicando todo el proceso de desarrollo de la aplicación.

PARTE II

ORGANIZACIÓN

DEL PROYECTO

METODOLOGÍA

El objetivo de este capítulo es mostrar la metodología y las herramientas de gestión de proyecto que se han utilizado durante el desarrollo de éste.

3.1 ESTRUCTURA ORGANIZACIONAL DEL PROYECTO

Este proyecto ha sido desarrollado por una sola persona, adquiriendo la propia persona todos los roles de responsabilidades necesarios para cumplir con los objetivos del proyecto.

3.2 METODOLOGÍA DE DESARROLLO

Se ha usado una metodología de desarrollo basada en Scrum. Esta metodología se caracteriza por ser ligera, incremental e iterativa, adaptándose al proyecto en el que se está ejecutando. La diferencia de nuestra metodología con Scrum es que no todas las iteraciones que se realizan van a durar el mismo tiempo. Se ha planificado previamente la duración de tiempo para cada iteración.

Se realizará una reunión por cada iteración, para comprobar que se están cumpliendo con los objetivos del proyecto en el tiempo planificado.

Para la gestión de tareas se ha utilizado la herramienta Trello, la cual nos permite organizar nuestras tareas utilizando un tablero virtual. Para controlar el tiempo de cada tarea se ha utilizado la herramienta Clockify, la cual se integra con Trello mediante un plugin.

Para la gestión del código se ha utilizado la herramienta Git, que consiste en un software de control de versiones muy utilizado actualmente y que también nos permite alojar nuestro código en plataformas como Github, el mayor repositorio de código actual.

PLANIFICACIÓN

El objetivo de este capítulo es realizar la planificación temporal del proyecto junto con la planificación real que se ha dado durante el desarrollo del proyecto.

4.1 RESUMEN TEMPORAL DEL PROYECTO

El presente proyecto tendrá una duración de 7 semanas más 2 días. Cada día se trabajará 8 horas sin tener en cuenta los días sábados y domingos. A continuación se muestra su resumen temporal:

Resumen del proyecto	
Fecha de inicio	10/07/2020
Fecha de fin	30/08/2020
Periodicidad de las revisiones	1 semanas
Carga de trabajo semanal	40 horas
Horas totales previstas	296 horas
Horas finales	308 horas

Cuadro 4.1: Tabla resumen de tiempos y planificación

4.2 PLANIFICACIÓN INICIAL

A continuación se mostrará un desglose de las iteraciones, comienzo y fin de cada una:

Resumen de iteraciones	
Iteración 1	10/07/20 a 15/07/20
Iteración 2	15/07/20 a 01/08/20
Iteración 3	01/08/20 a 11/08/20
Iteración 4	11/08/20 a 23/08/20
Iteración 5	23/08/20 a 30/08/20

Cuadro 4.2: Planificación temporal de iteraciones

Las fechas han sido decididas en función de la carga de trabajo de cada iteración. A continuación se mostrará las tareas a realizar en cada iteración:

Iteración 1. Análisis de las tecnologías a usar.

Iteración 2. Desarrollo de la parte frontend del proyecto.

Iteración 3. Desarrollo de la parte backend del proyecto.

Iteración 4. Despliegue del proyecto.

Iteración 5. Elaboración de la documentación del proyecto.

4.3 INFORME DE TIEMPOS DEL PROYECTO

A continuación se mostrará un desglose de las iteraciones con los datos reales. Ver Tabla §4.3.

Resumen de iteraciones	
Iteración 1	10/07/20 a 14/07/20
Iteración 2	14/07/20 a 05/08/20
Iteración 3	05/08/20 a 11/08/20
Iteración 4	11/08/20 a 23/08/20
Iteración 5	23/08/20 a 30/08/20

Cuadro 4.3: Planificación temporal de iteraciones

Iteración 1: Se han analizado las tecnologías actuales disponibles para poder realizar un proyecto e-commerce. Además, se han analizado los servicios de hosting más económicos. En un principio se optó por utilizar Woocommerce, pero al ver que la plataforma de pago Stripe nos permitía prescindir de Woocommerce para vender productos, se acabó utilizando Stripe para también almacenar los productos. Hemos conseguido finalizar con esta tarea antes del tiempo planificado, así que hemos adelantado el comienzo de la siguiente iteración, de tal modo, que si sufrimos un retraso en el proyecto por mala planificación, nos perjudique lo menos posible.

Iteración 2: Se han realizado los estudios de las siguientes tecnologías: React, Three.js, react-three-frame, Gatsby, Stripe, GraphQL. Posteriormente se ha realizado un proyecto e-commerce básico en el que mostramos el modelo 3D de un bolso y la opción de compra de dicho modelo. En esta iteración nos hemos encontrado con una serie de problemas que no se han tenido en cuenta a la hora de su planificación. Dichos problemas han sido: Bloqueo CORS en la carga de imágenes, problemas al realizar la build por ejecutarse el código en un contexto fuera del

navegador, problema con el evento onclick que no funciona en los smartphones, problemas con el checkout de la tienda que solo funciona con el proyecto en producción, problemas al cargar el modelo 3D debido a que se necesitaba crear la carpeta static en la raíz del proyecto y alojar ahí los archivos. Todos esos problemas han sido comentados y solucionados en el apartado de desarrollo de este proyecto.

Iteración 3: Se ha elaborado el fichero docker-compose con las tecnologías nginx con Let's Encrypt para el https y apache para servir nuestra web estática.

Iteración 4: Se ha realizado la compra y configuración de un nombre de dominio en la web namecheap.com. Se ha aprendido a utilizar la plataforma de hosting DigitalOcean y a enlazarla con el dominio adquirido. Se ha mejorado en el manejo de Linux por terminal. Algunos de los conocimientos que hemos tenido que adquirir han sido: los comandos ssh y scp para enviar archivos mediante ssh, configuración de puertos que por defecto se encuentran bloqueados, mejora en el manejo de Docker, gestión del espacio libre para evitar adquirir una máquina de mayor costo.

Iteración 5: Se ha elaborado la documentación del proyecto utilizando la herramienta Latex.

COSTES

En este capítulo se realizará un desglose de los costes del desarrollo del proyecto.

5.1 RESUMEN DE COSTES DEL PROYECTO

Resumen del proyecto	
Costes de personal	1.904,14 €
Sueldo bruto	1.451,88 €
Costes sociales	452,26 €
Costes materiales	1.075,85 €
Costes indirectos	297,99 €
TOTAL	3.277,98 €

Cuadro 5.1: Tabla resumen de costes

5.2 COSTES DE PERSONAL

En este apartado se calcularán los costes de personal en base a la planificación realizada en el apartado de planificación de este proyecto. El coste de personal es el resultado de la suma del sueldo bruto y los costes sociales. El sueldo mínimo bruto para un programador/analista en aplicaciones informáticas es de 39,24 € por día trabajado. Esta referencia ha sido publicada en el BOE con fecha de 13 de mayo de 2020 (en el apartado de bibliografía se encuentra un enlace a este documento). Los costes sociales varían en función del tipo de contrato: 29,90 % para los trabajos de tiempo completo y 31,15 % para los trabajos de tiempo parcial. En este caso se calcularán los costes correspondiente a un trabajo de tiempo parcial. Por lo tanto, los costes totales de personal nos quedaría de la siguiente manera:

Sueldo bruto Programador-Analista de Informática: $39,24 \text{ €} / \text{día} \times 37 \text{ días} = 1.451,88 \text{ €}$

Costes sociales: $1.451,88 \text{ €} * 0,3115 = 452,26 \text{ €}$

Coste total: $1.451,88 \text{ €} + 452,266 \text{ €} = 1.904,14 \text{ €}$

5.3 COSTES MATERIALES

Los equipos informáticos deberán ser amortizados. Según la Agencia Tributaria, el coeficiente lineal máximo aplicable a los equipos para procesos de información es del 25 % anual, con un máximo de 8 años. En este caso, consideramos el período mínimo de

4 años, ya que la empresa necesitará una actualización de equipos frecuente al dedicar los equipos a una tarea intensiva como es el desarrollo.

Para este proyecto se ha utilizado un ordenador portátil msi valorado en 1.000€, que se amortizará a un ritmo de 250€ anuales, 20,83€ al mes, 0,69€ al día. Como nuestro proyecto va a tener una duración de 37 días, se nos quedaría un total de: 0,69€/día * 37 días = 25,69€

Con respecto al Software, la plataforma de hosting DigitalOcean tiene un costo de 5\$/mes (4,18€/mes). Suponiendo que tendremos desplegada la aplicación durante un año, haría un costo total de: 4,18€ * 12 meses = 50,16€ al año.

Con todo lo anterior, los costes materiales nos harían un total de: 1000€ + 25,69€ + 50,16€ = 1.075,85€

5.4 COSTES INDIRECTOS

Conforme a los conocimientos adquiridos a lo largo de la carrera de Ingeniería del Software, sabemos que hay que establecer un plan de contingencias para obtener una reserva en caso de que puedan surgir problemas a lo largo del desarrollo del proyecto. Este se establece como el 10% de los costes directos. Por lo tanto nos quedaría de la siguiente forma: $(1.904,14€ + 1.075,85€) * 0.1 = 297,99€$

PARTE III

DESARROLLO DEL PROYECTO

ANÁLISIS

En este capítulo se realizará un análisis de las tecnologías que han sido empleadas para el desarrollo del proyecto.

6.1 ANÁLISIS DE LAS TECNOLOGÍAS EMPLEADAS

Las tecnologías que más se utilizan para realizar un e-commerce hoy en día son: Shopify, Prestashop, Magento y Woocommerce.

De entre todas ellas se eligió en un principio Woocommerce por ser una opción gratuita y por ser un plugin para Wordpress. Wordpress es un sistema de gestión de contenido, escrito en php, muy popular hoy en día. Se dice que el 25% de las páginas webs que existen actualmente en internet están construidas con Wordpress. Entre las ventajas de usar Wordpress se encuentran:

- Sistema de código abierto de gran popularidad.
- Estabilidad, tiene bastantes años de desarrollo por delante.
- Tiene una gran comunidad de usuarios que participan en ayudar a los demás con la herramienta.
- Permite a los desarrolladores crear páginas webs sin la necesidad de escribir código, proporcionando funcionalidades mediante la utilización de plugins tanto gratuitos como de pago, y ofreciendo la opción de poder crear nuestros propios plugins mediante código php.

Las desventajas de Wordpress son:

- Al ser tan popular y de código abierto se ha convertido en el punto de mira para los Hackers, por lo tanto, hay que tener mucho más en cuenta la seguridad y las actualizaciones a la hora de desarrollar una página web con Wordpress.
- Es necesario un servidor más potente de lo normal para correr un Wordpress ya que al ser un sistema tan completo requiere de más memoria RAM que otros sistemas CMS más ligeros como Strapi.
- Hay que aprender cómo funciona Wordpress para desarrollar en Wordpress si se quiere evitar tener que pagar por los plugins.
- Como opinión personal, es difícil encontrar buenas explicaciones de desarrollo en Wordpress de forma gratuita. La mayoría de información que encuentras más accesible consiste en la utilización de plugins de pago para solucionar un problema o en explicaciones básicas en donde te ofrecen posteriormente un pago para

6.1. ANÁLISIS DE LAS TECNOLOGÍAS EMPLEADAS

profundizar más sobre el tema. Al ser tan popular se ha generado ese movimiento que dificulta el acceso a la información gratuita y de calidad de la herramienta. Pero eso no significa que no exista dicha información disponible, simplemente hay que profundizar más en las búsquedas para encontrarla.

Como plataforma de pago existen las opciones de Paypal y Stripe. Se ha elegido Stripe por poseer una comisión de venta inferior a la de Paypal y porque nos da la opción de gestionar los productos de venta desde la propia plataforma, pudiendo así prescindir de utilizar Woocommerce para realizar dicha gestión. Por lo tanto, al poderse utilizar Stripe como Backend para almacenar los productos, nos ahorraremos el dinero de tener que mantener un Wordpress alojado en un servidor.

Desde el lado Frontend utilizaremos el framework Gatsby. Este framework basado en React nos da la opción de poder crear páginas webs estáticas. Esto significa que podemos alojar dichas webs en plataformas como Github pages o Vercel que nos da la posibilidad de alojamiento web sin ningún coste mensual. Aunque nosotros utilizaremos la plataforma de hosting Digitalocean para aprender a desplegar cualquier tipo de aplicación web, sea estático o no, en un servidor Linux. Gatsby también nos optimiza automáticamente el código al realizar la build para producción, quedando un código resultante con unas estadísticas en rendimiento web cercanas al 100%. Nuestro objetivo será conectar Gatsby con la plataforma de pago Stripe.

Con la utilización de React, nos permitimos tener un código Javascript mucho más organizado, limpio y eficiente que si utilizasemos Javascript puro. Para el visualizado de gráficos en 3D utilizamos la herramienta react-three-fiber que nos permite escribir código para la librería Three.js utilizando los principios de React. Three.js es una librería que nos facilita construir escenas 3D sin tener que utilizar directamente WebGL, la cual es una API de bajo nivel para los gráficos 3D en el navegador.

Como herramienta de despliegue se ha utilizado Docker, que es un proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software. Se ha elegido esta herramienta porque una vez que se configure el proyecto con Docker no hace falta configurar el servidor linux manualmente para que todo funcione. Docker se encarga de hacerlo por nosotros. También porque cuando ocurre algún fallo no afecta a nuestro sistema operativo principal y se puede solucionar más fácilmente reiniciando los contenedores o volviéndolos a instalar. En definitiva, Docker nos ahorra tiempo de configuración de un servidor y quebraderos de cabeza ante los problemas.

Como plataforma de hosting, se ha elegido Digitalocean. Esta plataforma nos permite tener un control completo del servidor Linux en el que estamos desplegando nuestras aplicaciones. Siendo nosotros los responsables de tener que configurar la máquina para que todo funcione, pero con la ventaja de tener el control absoluto de dicha máquina sin costos adicionales. Además tiene unos precios que junto con Heroku son las dos mejores opciones económicas de hosting que hay actualmente en el mercado.

DESARROLLO

En este capítulo se realizará una explicación detallada del proceso de desarrollo de un e-commerce básico con un modelo interactivo en 3D.

7.1 ANTES DE COMENZAR

Antes de comenzar, hay que proceder a la instalación de las siguientes herramientas: nodejs, npm, yarn, git. Como editor de texto utilizaré Sublime Text y VSCode pero se puede utilizar el que le resulte más cómodo al lector.

Las imágenes del navegador web que se muestran a lo largo del desarrollo aparecen con el modo nocturno activado debido a que utilizo una extensión para Chrome llamada Dark Reader.

Los proyectos que vamos a realizar a continuación se encuentran adjuntos en la carpeta de este trabajo. Dichos proyectos están ya finalizados y listos para producción.

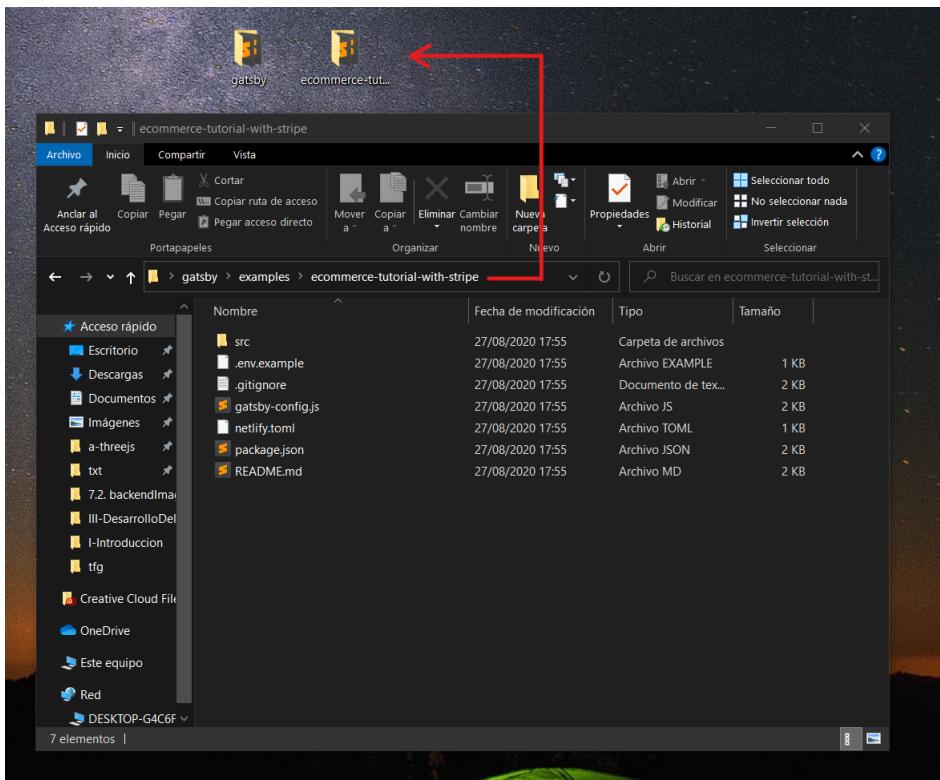
7.2 INICIO DE UN PROYECTO CON GATSBY

Para comenzar nuestro proyecto y facilitarnos el trabajo, vamos a utilizar un template que nos proporciona Gatsby en su web oficial. Además nos dispone de un tutorial explicandonos cómo se ha creado ese template:

- <https://www.gatsbyjs.com/tutorial/ecommerce-tutorial/>

- <https://github.com/gatsbyjs/gatsby/tree/master/examples/ecommerce-tutorial-with-stripe>

Lo primero que haremos será clonar dicho repositorio en la carpeta que más nos guste, yo utilizaré el Escritorio. Una vez clonado en el Escritorio, abrimos la terminal de git y ejecutamos el comando `git clone https://github.com/gatsbyjs/gatsby.git` para posteriormente acceder a la carpeta `examples/ecommerce-tutorial-with-stripe`. Copiamos dicha carpeta en el Escritorio y ya podemos borrar la carpeta de Gatsby que hemos clonado, obteniendo así nuestro proyecto inicial.



También se puede descargar el proyecto desde Github, obtener la carpeta que nos interesa y ponerla en el Escritorio manualmente.

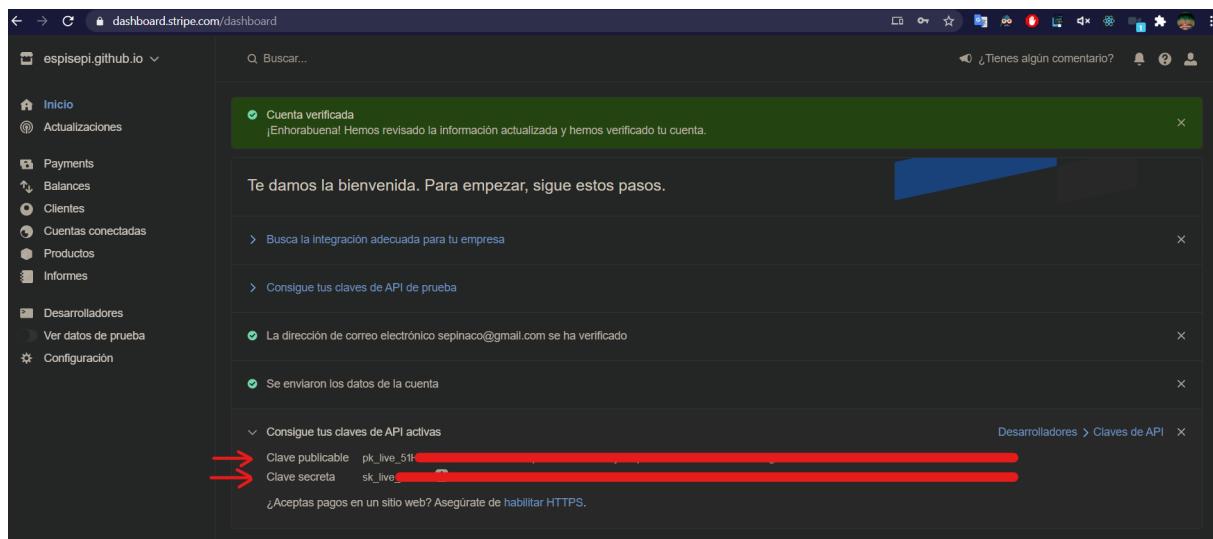
Una vez tenemos la carpeta en nuestro Escritorio, abrimos una terminal dentro de nuestra carpeta y ejecutamos el comando **yarn install** para instalar las dependencias de la aplicación. También se puede utilizar npm, pero utilizamos yarn por ser más estable y veloz que npm.

Una vez instaladas las dependencias ejecutaremos el comando **yarn start** y recibiremos un error debido a que no tenemos indicada las claves de la API de Stripe, lo cual lo indicaremos en el capítulo siguiente.

7.3 CONEXIÓN A LA API DE STRIPE

Para conectarnos a la API de Stripe lo primero que tenemos que hacer es acceder a <https://stripe.com/es> y crearnos una cuenta. Una vez tengamos creada la cuenta, en la sección de **Inicio** veremos que Stripe nos ofrece las claves de API de prueba y activas. Nosotros utilizaremos directamente las activas. Mi recomendación es copiar en un fichero txt aparte la clave pública y secreta, ya que la secreta una vez que la miremos no la podemos volver a ver en la plataforma Stripe por razones de seguridad.

CAPÍTULO 7. DESARROLLO



Una vez localizadas las claves pública y secreta, volvemos a nuestro proyecto Gatsby, y crearemos dos ficheros: **.env.development** y **.env.production**. Estos ficheros estarán basados en el fichero que nos viene por defecto de ejemplo llamado **.env.example**. Estos ficheros son los que utilizará gatsby para leer las claves de Stripe tanto en development como en production, según el modo en el que se esté ejecutando. Una vez rellenado los dos ficheros con nuestra clave nos quedaría de la siguiente manera:

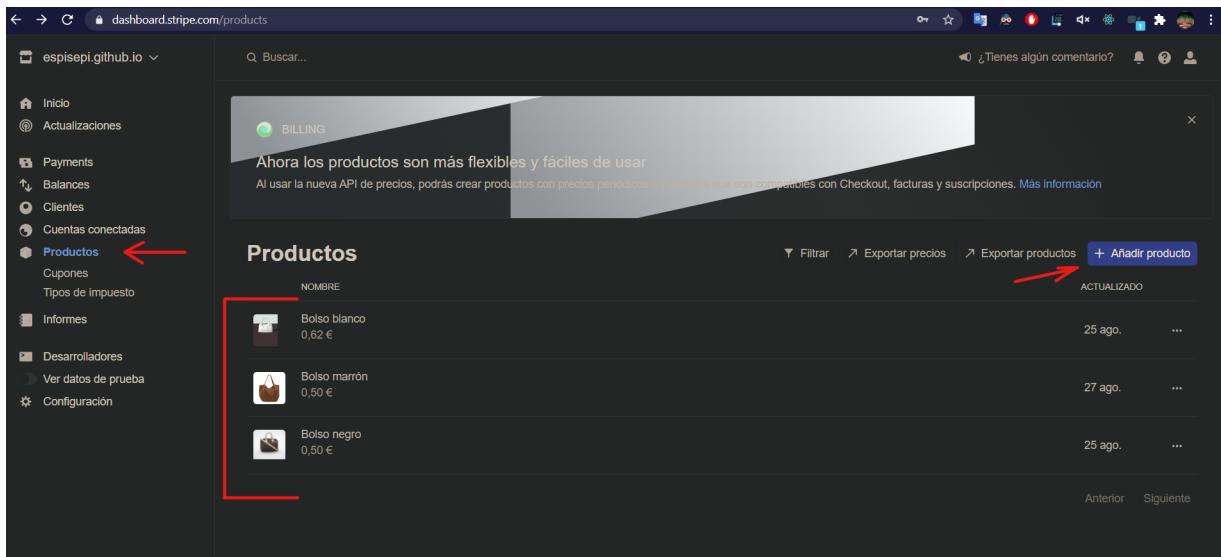
Two screenshots of Sublime Text showing environment variable files. The top window is titled '.env.production' and the bottom one is '.env.development'. Both files contain the following code:

```
1 # Details see: https://www.gatsbyjs.org/docs/environment-variables/
2 # Find your API keys in the Dashboard: https://dashboard.stripe.com/test/apikeys
3 → GATSBY_STRIPE_PUBLISHABLE_KEY=pk_live_51...
4 → GATSBY_BUTTON_PRICE_ID=
5 → STRIPE_SECRET_KEY=sk_live_51...
```

The API keys 'pk_live_51...', 'sk_live_51...', and 'GATSBY_BUTTON_PRICE_ID=' are highlighted with red boxes and arrows pointing to them from the explanatory text above.

Antes de ejecutar nuestro proyecto, vamos a añadir tres productos en nuestra cuenta de Stripe, para ello nos vamos a la sección de **Productos** y en **Añadir producto** creamos nuestros productos de ejemplo.

7.3. CONEXIÓN A LA API DE STRIPE



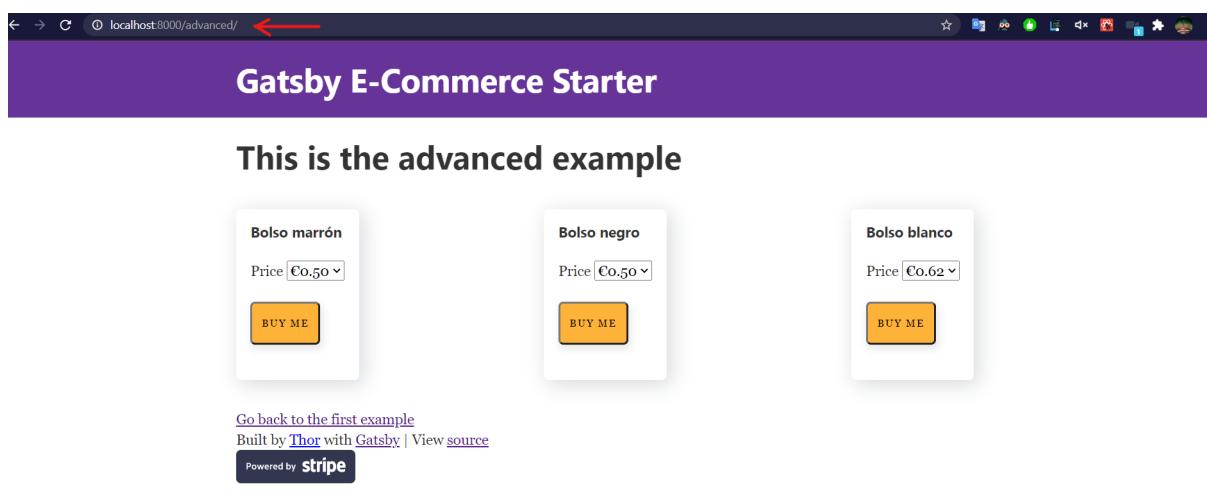
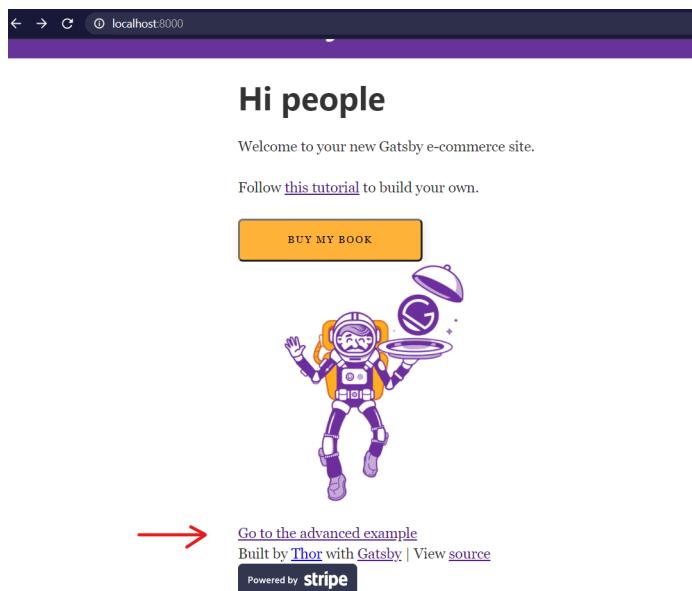
Una vez creados nuestros productos, procedemos a ejecutar el comando **yarn start** en la raíz de nuestro proyecto y ya nos debería de funcionar sin darnos ningún error tal y como se muestra en la imagen.

```
C:\Users\Jose Angel Dominguez\Desktop\ecommerce-tutorial-with-stripe>yarn start ↗
> ecommerce-gatsby-tutorial@0.2.0 develop C:\Users\Jose Angel Dominguez\Desktop\ecommerce-tutorial-with-stripe
> gatsby develop

Debugger listening on ws://127.0.0.1:9229/fdfbd6b1-32e1-4990-8f0b-be78d31dc9ef
For help, see: https://nodejs.org/en/docs/inspector
success open and validate gatsby-configs - 0.03s
success load plugins - 1.079s
success onPreInit - 0.044s
success initialize cache - 0.010s
success copy gatsby files - 0.128s
success onPreBootstrap - 0.015s
success createSchemaCustomization - 0.004s
success Checking for changed pages - 0.004s
success source and transform nodes - 0.705s
success building schema - 0.384s
success createPages - 0.004s
success Checking for changed pages - 0.003s
success createPagesStatefully - 0.108s
success update schema - 0.025s
success write out redirect data - 0.005s
success Build manifest and related icons - 0.126s
success onPostBootstrap - 0.131s
info bootstrap finished - 5.926s
success onPreExtractQueries - 0.005s
success extract queries from components - 0.359s
success write out requires - 0.052s
success run static queries - 0.053s - 1/1 18.08/s
success run page queries - 0.039s - 6/6 171.41/s
You can now view ecommerce-gatsby-tutorial in the browser.
http://localhost:8000/ ↗
View GraphiQL, an in-browser IDE, to explore your site's data and schema
http://localhost:8000/__graphiql
Note that the development build is not optimized.
To create a production build, use gatsby build
success Building development bundle - 3.939s
```

Para comprobar que todo funciona, nos dirigimos al navegador e introducimos la url <http://localhost:8000/>. Seguidamente, en la imagen que se nos muestra, pulsamos la opción **Go to the advanced example** o también podemos escribir en el navegador el enlace <http://localhost:8000/advanced/>. Una vez dentro, veremos los productos que hemos subido a Stripe.

CAPÍTULO 7. DESARROLLO



7.4 OBTENCIÓN DE PRODUCTOS POR MEDIO DE GRAPHQL

Una vez tenemos conectada nuestra aplicación a Stripe, podemos ver los productos introduciendo la url <http://localhost:8000/advanced/> en el navegador, como se muestra en la imagen anterior. Vamos a profundizar en el código que nos permite mostrar dichos productos.

La tecnología de comunicación que se utiliza se denomina GraphQL. GraphQL consiste en un lenguaje de consultas que pretende complementar los problemas que tiene API REST. Estos son que cuando realizamos una petición a un recurso por medio del protocolo GET, obtenemos todos los atributos de dicho recurso aunque solo necesite-

7.4. OBTENCIÓN DE PRODUCTOS POR MEDIO DE GRAPHQL

mos algunos de ellos. Si queremos que se nos devuelva solo los atributos que vayamos a utilizar, tenemos que crear un nuevo endpoint devolviendo el recurso con los atributos específicos que le indiquemos. Para aclararlo aún más, imaginémosnos que tenemos una API REST con el enlace **GET /productos** que nos devuelve una lista de los productos que hemos creado. Pero nosotros solo queremos las imágenes de dichos productos, no queremos ni el título, ni la descripción, ni el precio. Sin embargo el servidor nos devuelve esos atributos aunque nosotros no lo necesitamos. Para ello existe la solución de crear un nuevo enlace (o endpoint) **GET /productosImages** en el que solo devolvemos las imágenes de todos los productos. Esto al final provocaría que tuviésemos que crear muchos endpoints personalizados para cada situación y Facebook decidió solucionar este problema creando GraphQL. GraphQL se conecta a un único endpoint, y es en la propia petición que realizamos al servidor donde le indicamos cuáles son los recursos y los atributos de dichos recursos que queremos que el servidor nos envíe. Para verlo mejor, mostraremos un ejemplo que podemos realizar con nuestra aplicación ya que Gatsby nos proporciona una herramienta para realizar peticiones GraphQL.

The screenshot shows the GraphiQL interface running at `localhost:8000/_graphiql`. On the left, the 'Explorer' sidebar lists various GraphQL fields and types. A red box highlights the 'product' field under the 'nodes' section. Another red box highlights the 'images' field under the 'product' field. In the center, the 'GraphiQL' tab displays a query and its corresponding JSON response. The query is:

```
query MyQuery {
  allStripePrice {
    nodes {
      product {
        name
        description
        images
      }
    }
  }
}
```

The response JSON is:

```
{
  "data": {
    "allStripePrice": [
      {
        "nodes": [
          {
            "product": {
              "name": "Bolso blanco",
              "description": "Este producto es ficticio, si introduce sus datos la compra será realizada.",
              "images": [
                "https://files.stripe.com(links/f1_live_GFCN9YC8x5eE9glhzZ5oYuYI"
              ]
            }
          },
          {
            "product": {
              "name": "Bolso marrón",
              "description": "Este producto es ficticio, si introduce sus datos la compra será realizada.",
              "images": [
                "https://files.stripe.com(links/f1_live_RffUKaRGBV4utsxORB2E7RPf"
              ]
            }
          },
          {
            "product": {
              "name": "Bolso marrón",
              "description": "Este producto es ficticio, si introduce sus datos la compra será realizada.",
              "images": [
                "https://files.stripe.com(links/f1_live_RffUKaRGBV4utsxORB2E7RPf"
              ]
            }
          },
          {
            "product": {
              "name": "Bolso negro",
              "description": "Este producto es ficticio, si introduce sus datos la compra será realizada.",
              "images": [
                "https://files.stripe.com(links/f1_live_1bGis19Alubd1E5iIiH546TL"
              ]
            }
          }
        ]
      }
    ]
  }
}
```

Como podemos ver en la imagen, estamos obteniendo todos los productos, y de todos los productos los atributos correspondientes al nombre, descripción e imagen de cada producto. Pero también podemos obtener fácilmente solo las imágenes de todos los productos indicando que se nos devuelva solamente ese atributo. A continuación se muestra un ejemplo.

The screenshot shows the GraphiQL interface running at localhost:8000/_graphiql. On the left, the Explorer sidebar shows a tree structure of available queries and types. Two specific fields are highlighted with red arrows: 'product' and 'images'. In the main area, a query named 'MyQuery' is defined:

```

query MyQuery {
  allStripePrice {
    nodes {
      product {
        images
      }
    }
  }
}

```

The results pane displays the JSON response from the GraphQL endpoint. It shows a list of products, each with a URL for its image:

```

{
  "data": {
    "allStripePrice": {
      "nodes": [
        {
          "product": {
            "images": [
              "https://files.stripe.com/files/f1_live_GFCN9YC8x5eE9glhzZ5oYuYI"
            ]
          }
        },
        {
          "product": {
            "images": [
              "https://files.stripe.com/files/f1_live_RfFUKaRGB4utsxORB2E7RPf"
            ]
          }
        },
        {
          "product": {
            "images": [
              "https://files.stripe.com/files/f1_live_RfFUKaRGB4utsxORB2E7RPf"
            ]
          }
        },
        {
          "product": {
            "images": [
              "https://files.stripe.com/files/f1_live_1bG1S19AlubdIE5iiHS46TL"
            ]
          }
        }
      ],
      "extensions": {}
    }
  }
}

```

Si se quiere profundizar sobre GraphQL, en Youtube existe una buena cantidad de recursos de calidad. Entre los que he utilizado para mi aprendizaje se encuentra el vídeo creado por el canal **EDteam** denominado **¿Qué es GraphQL?**. Y si se quiere ver más ejemplos sobre GraphQL también existe el video **Queries y GraphiQL con la API de Rick Morty [Curso express GraphQL]** del canal de Youtube **midudev**. Estos recursos y muchos más aparecen también en el apartado de Bibliografía de este proyecto.

Creo que merece la pena aclarar que nuestra aplicación no se conecta al enlace <http://localhost:8000/graphql> para realizar las peticiones de los productos. Se conectará a un enlace parecido a <https://stripe.com/graphql> que estará a la escucha para devolver los productos o cualquier otro recurso que se le pida. <http://localhost:8000/graphql> no es más que una herramienta que nos proporciona Gatsby para facilitarnos la vida a la hora de probar a realizar peticiones GraphQL para luego incluirlas en nuestro código sabiendo que funcionan correctamente.

Una vez entendido qué es GraphQL vamos a ver dónde aparece dicha consulta dentro de nuestro proyecto. Para ello abriremos el proyecto con VSCode, y nos dirigiremos al siguiente archivo **src/components/Products/Products.js**. Aquí encontramos la petición GraphQL que se le envía al servidor de Stripe. En este fichero será donde insertaremos nuestra escena 3D que veremos en el siguiente capítulo.

7.4. OBTENCIÓN DE PRODUCTOS POR MEDIO DE GRAPHQL

```
JS Products.js src\components\Products\Products.js...
const Products = () => {
  return (
    <StaticQuery
      query={graphql`...
        query ProductPrices {
          prices: allStripePrice(
            filter: { active: { eq: true } }
            sort: { fields: [unit_amount] }
          ) {
            edges {
              node {
                id
                active
                currency
                unit_amount
                product {
                  id
                  name
                }
              }
            }
          }
        }
      render={({ prices }) => {
        // Group prices by product
        const products = {}
        for (const { node: price } of prices.edges) {
          const product = price.product
          if (!products[product.id]) {
            products[product.id] = product
            products[product.id].prices = []
          }
          products[product.id].prices.push(price)
        }

        return (
          <div style={containerStyles}>
            {Object.keys(products).map(key => (
              <ProductCard key={products[key].id} product={products[key]} />
            ))}
          </div>
        )
      }
    }
  )
}
```

Vamos a cambiar la Query por la que se muestra a continuación y vamos a realizar un **console.log(products)** para ver qué es lo que nos llega desde el servidor.

```

const Products = () => {
  return (
    <StaticQuery
      query={graphql`query ProductPrices {
        prices: allstripePrice(
          filter: { active: { eq: true } }
          sort: { fields: [unit_amount] }
        ) {
          edges {
            node {
              id
              active
              currency
              unit_amount
              product {
                id
                name
                images
              }
            }
          }
        }
      render={({ prices }) =>
        // Group prices by product
        const products = {}
        for (const { node: price } of prices.edges) {
          const product = price.product
          if (!products[product.id]) {
            products[product.id] = product
            products[product.id].prices = []
          }
          products[product.id].prices.push(price)
        }
        console.log(products) ←
      }
    
```

▼ {prod_HrdBV4v7j8yU8p: {...}, prod_HqsTh1eiHdw09Y: {...}, prod_HrdDt2NVuRTTzT: {...}} ⓘ

- ▼ prod_HqsTh1eiHdw09Y:
 - id: "prod_HqsTh1eiHdw09Y"
 - images: ["https://files.stripe.com/files/fl_live_1bG1S19Alubd1E5iiHS46TL"]
 - name: "Bolso negro"
 - prices: [...]
 - __proto__: Object
- ▶ prod_HrdBV4v7j8yU8p: {id: "prod_HrdBV4v7j8yU8p", name: "Bolso marrón", images: Array(1), prices: Array(1)}
- ▶ prod_HrdDt2NVuRTTzT: {id: "prod_HrdDt2NVuRTTzT", name: "Bolso blanco", images: Array(1), prices: Array(1)}
- ▶ __proto__: Object

Si aparecen los productos como se muestran en la imagen, estaríamos preparado para pasar al siguiente capítulo.

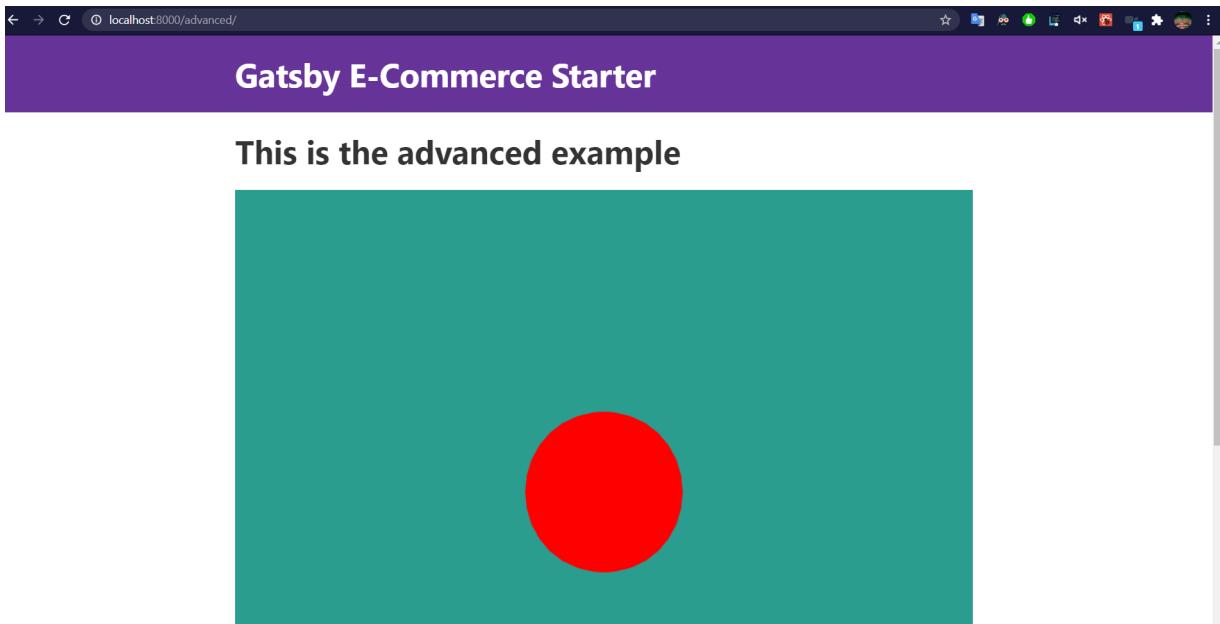
7.5 CREACIÓN DE LA ESCENA 3D CON REACT-THREE-FIBER

Antes de comenzar, vamos a instalar las librerías three.js y react-three-fiber en nuestro proyecto. Para ello abrimos una terminal en VSCode y ejecutamos el comando **yarn add three react-three-fiber**. Una vez finalice, comprobaremos si todo funciona correctamente mostrando un ejemplo en pantalla. Para ello, nos vamos al fichero **Product.js** e importamos el componente **Canvas** de la librería **react-three-fiber** para poder comenzar a dibujar en pantalla. Después añadiremos una esfera de color rojo para comprobar que todo funciona. El código quedaría de la siguiente manera.

```
import React from "react"
import { graphql, StaticQuery } from "gatsby"
import ProductCard from "./ProductCard"
import {Canvas} from 'react-three-fiber' ←
```

```
return (
  <Canvas style={{width:'100%', height:'100vh', background:'#2a9d8f', display:'block'}}>
    <ambientLight />
    <mesh visible position={[0, 0, 0]} rotation={[0, 0, 0]}>
      <sphereGeometry attach="geometry" args={[1, 16, 16]} />
      <meshStandardMaterial
        attach="material"
        color="red"
      />
    </mesh>
  </Canvas>
)
```

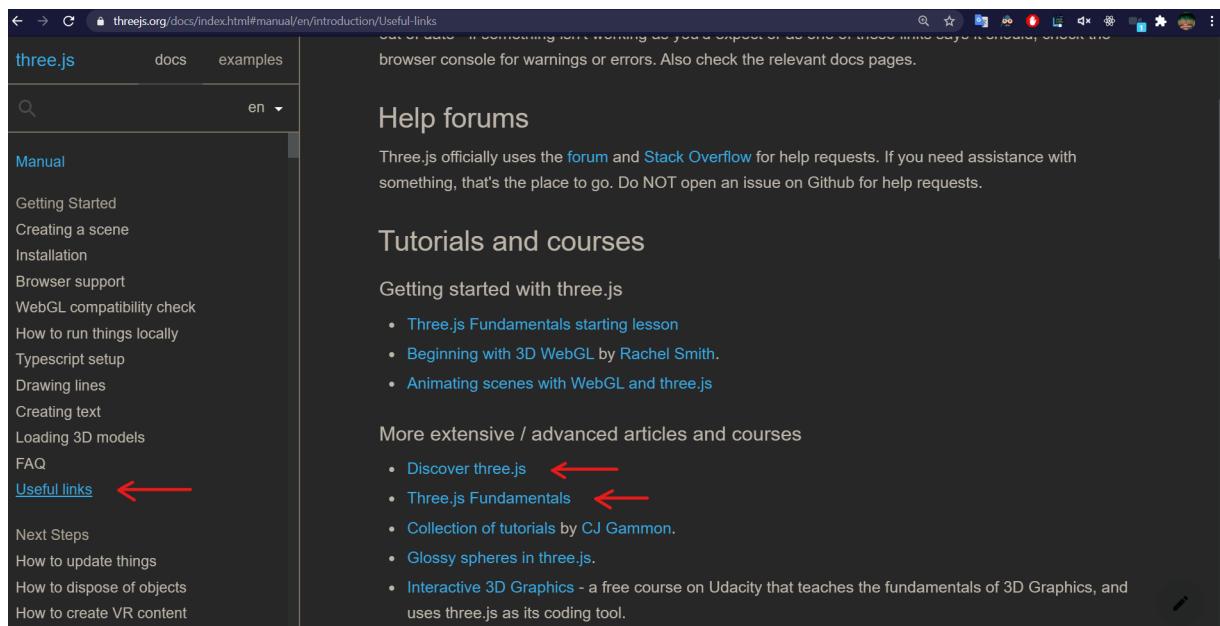
El código **return** que se muestra en la imagen corresponde al return del componente Product del archivo **Product.js**. Es decir, eliminamos el código que teníamos por defecto en la plantilla y añadimos el código mostrado en la imagen. Esto nos dará como resultado la siguiente escena.



Explicar la teoría de Three.js para entender qué es lo que está ocurriendo por debajo es algo que se nos escapa de nuestro objetivo por falta de tiempo. A continuación voy a realizar un breve resumen y ofreceré enlaces para poder profundizar sobre el tema.

Lo primero que observamos en nuestro código es el componente **<Canvas>**, la escena 3D que vemos se renderiza dentro de un elemento canvas. Lo que observamos dentro de nuestro canvas es posible gracias a la API WebGL, que consiste en una API implementada en Javascript para la renderización de gráficos en 3D en un navegador.

web. Algo así como OpenGL o DirectX pero enfocado para que se ejecute en el navegador. WebGL es una API difícil de dominar, puesto que es una API de bajo nivel para poder ofrecer el mayor rendimiento posible. Es por ello que el autor Ricardo Cabello, alias Mr.doob, desarrolló la librería Three.js, con la que se nos permite trabajar con la API WebGL sin la complejidad que supone trabajar directamente con dicha API. Three.js posee una documentación de gran calidad y es con la que tenemos que trabajar a la hora de utilizar la librería. Para aprender más sobre esta librería tenemos el apartado **Useful links** dentro de la documentación. En la que imagen siguiente se señala dos enlaces recomendados que yo utilicé para mi aprendizaje con esta herramienta.

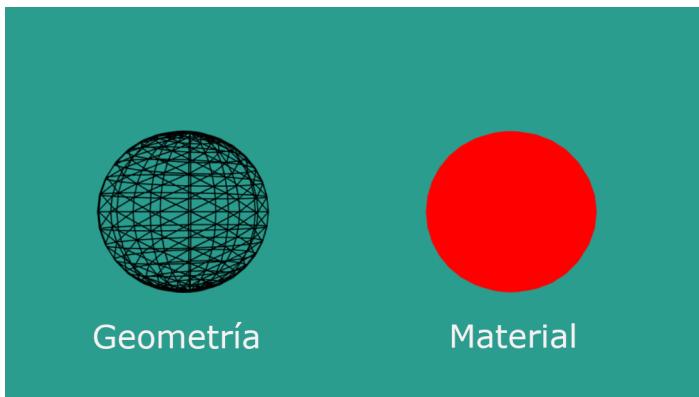


Una vez explicado en qué consiste Three.js, nos centraremos en react-three-fiber. react-three-fiber es una librería que nos permite utilizar Three.js basándonos en componentes React. Esto hace que tengamos un código mucho mejor organizado, basado en componentes, pero con la inconveniencia de que hay que aprender React para dominar esta librería. No entra dentro de nuestros objetivos explicar qué es React pero en la Bibliografía aparecen enlaces a recursos que me ayudaron a comprenderlo. En este proyecto se va a hacer uso de los hooks. Característica disponible a partir de React 16.8 y que consiste en sustituir las clases React por componentes funcionales. Utilizaremos dicha característica debido a que react-three-fiber hace uso de los hooks.

react-three-fiber nos ofrece una **api.md** como documentación en su repositorio GitHub pero también tenemos que hacer uso de la documentación oficial de Three.js, ya que react-three-fiber hace uso de dicha librería ofreciéndonos una manera más cómoda de poder trabajar con ella.

Una vez introducidas las dos librerías, procederemos a realizar una breve explicación del código que hemos escrito. El primer componente que observamos es `<ambientLight />`, este componente sirve para crear una luz ambiental en nuestra escena 3D. Si no tuviésemos ninguna luz en nuestra escena, todos los objetos que añadimos no tendrían color, es decir, aparecerían todos en negro. Por lo tanto es importante añadir una luz a nuestra escena. Existen más tipos de luz, todas ellas vienen explicadas en la documentación oficial de Three.js.

El segundo componente a observar es el componente `<mesh>`. Un mesh es un objeto 3D que se añade a la escena y la manera en la que se dibuja este objeto en la escena viene definido por sus atributos `geometry` y `material`. Para que quede más claro he realizado la siguiente composición:



Es decir, la geometria indica el número de vértices y la forma en la que van a estar distribuidos en el espacio dichos vértices. Ejemplos de geometría son: cubo, esfera, cono, círculo...

El material consiste en decir de qué manera vamos a dibujar dichos vértices. Ejemplos de material son el color con el que lo dibujemos o la rugosidad visual que le podemos dar al objeto. Con la rugosidad podemos hacer que una esfera se vea como una bola de cristal o como la bola de un Kendama. Existen muchos más parámetros para definir un material, incluso nos podemos inventar nosotros nuestro propio material utilizando el componente `ShaderMaterial` y utilizando el lenguaje GLSL. Si se está interesado en aprender sobre GLSL existe la web <https://thebookofshaders.com/> que explica con claridad los conceptos y la web <https://www.shadertoy.com/> que nos muestra ejemplos reales de código GLSL.

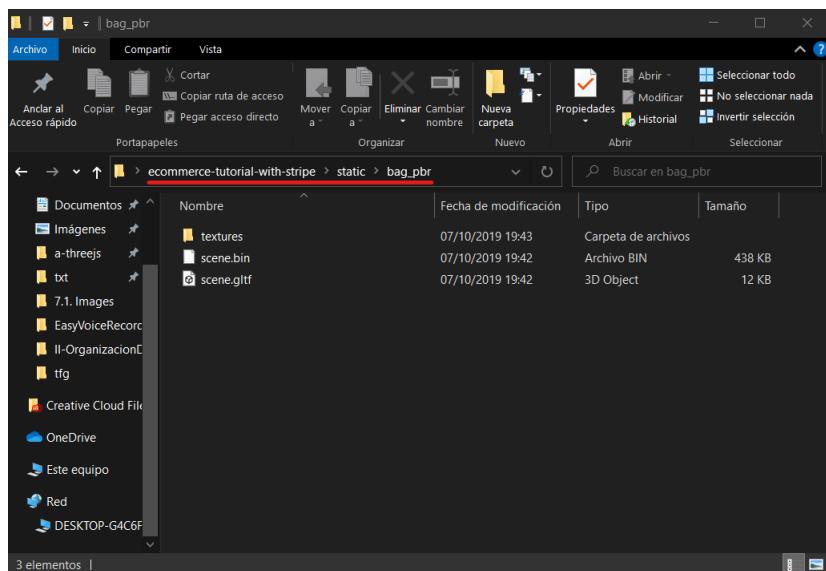
Una vez entendido el concepto de mesh, vamos a proceder a introducir el modelo 3D de un bolso en nuestra escena. Previamente vimos que una geometría puede ser un cubo, una esfera, un cono... pero también puede ser puntos puestos en el espacio arbi-

trariamente o formando una figura como la de un bolso como veremos a continuación.

Para poner nuestro modelo 3D en la escena lo primero que haremos será descargar dicho modelo utilizando la web sketchfab.com. Dicha web nos proporciona un repositorio de modelos 3D tanto gratuitos como de pago. El autor del modelo que vamos a utilizar es **Alvaro Sanint** y el enlace para descargarlo es el siguiente: <https://sketchfab.com/3d-models/bag-pbr-da3d686b38c141e1bf54f9df6ca39313>

El modelo lo descargaremos en formato **glTF** ya que es el formato que ofrece mayor rapidez en la carga del modelo en la escena.

Para añadir el modelo en nuestro proyecto, crearemos una carpeta **static** en la raíz de nuestro proyecto y dentro de ésta descomprimiremos el zip quedándose de la siguiente manera:



Gatsby automáticamente reconoce la carpeta **static** como la carpeta en donde se almacenan los recursos, y la tiene en cuenta a la hora de buscar los recursos y a la hora de realizar la build para tener el proyecto preparado para producción.

Para cargar nuestro modelo en la escena, vamos a crear un nuevo componente llamado **Bag** dentro de una nueva carpeta llamada **3d** que se encontrará dentro de la carpeta **components**. De esta manera tenemos el código organizado poniendo todos los componentes que tengan que ver con **react-three-fiber** dentro de la carpeta **3d**.

7.5. CREACIÓN DE LA ESCENA 3D CON REACT-THREE-FIBER

The screenshot shows the VS Code interface with the following details:

- EXPLORER:** Shows the project structure under "ECOMMERCE-TUTORIAL-WITH-STRIPE".
- JS Bag.js:** The code defines a component named "Bag" which uses a GLTF loader to load a "bag_pbr.scn.gltf" model and rotates it by -4 degrees.
- JS Products.js:** This file imports the "Bag" component from "3d/Bag". It also includes a "Loading" component which is a simple sphere.
- JS Products.js (continued):** The "Loading" component is defined as a mesh with specific geometry, material, and properties.

```

    JS Bag.js
    ...
    import React from "react";
    import { useLoader } from 'react-three-fiber';
    import { GLTFLoader } from 'three/examples/jsm/loaders/GLTFLoader';
    ...
    const Bag = () => {
      const gltf = useLoader(GLTFLoader, '/bag_pbr/scn.gltf')
      return <primitive object={gltf.scene} position={[0,0,-4]} scale={[0.1,0.1,0.1]} rotation={[0, Math.PI/2,0]} dispose={null} />
    };
    export default Bag;

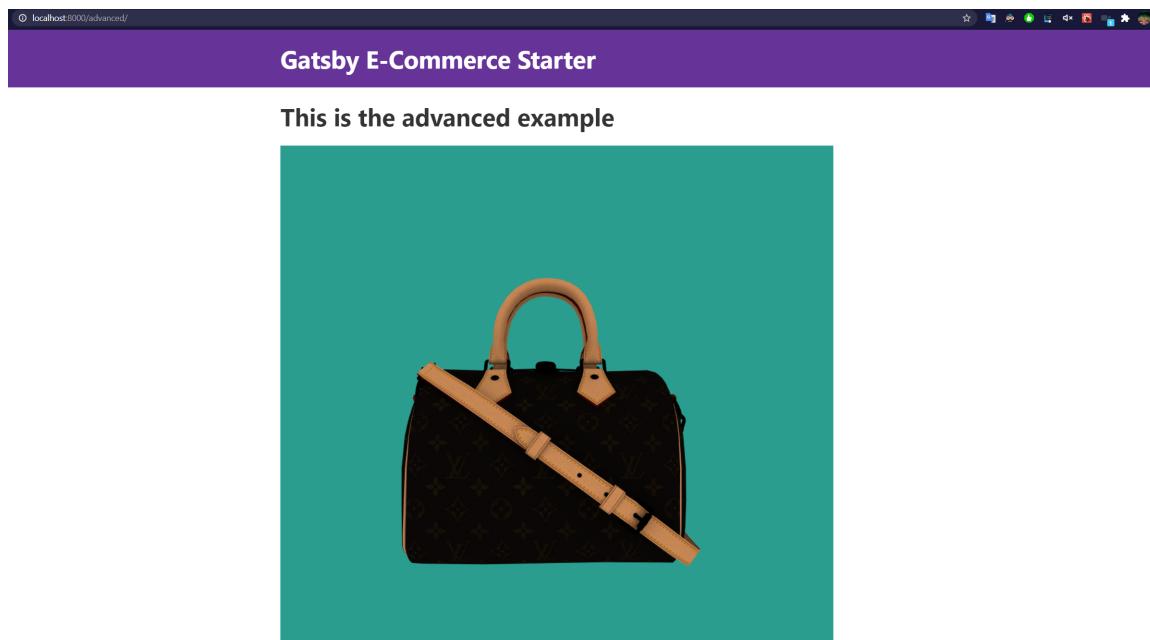
    JS Products.js
    ...
    import React, {Suspense} from "react"
    import {graphql, StaticQuery} from "gatsby"
    import {Canvas} from 'react-three-fiber'
    import Bag from '../3d/Bag' ←

    return (
      <Canvas style={{width:'100%', height:'100vh', background:'#2a9d8f', display:'block'}}>
        <ambientLight />
        <Suspense fallback=<Loading />> ←
        | <Bag /> ←
        </Suspense>
      </Canvas>
    )
    JS Products.js
    ...
    import React, {Suspense} from "react"
    import {graphql, StaticQuery} from "gatsby"
    import {Canvas} from 'react-three-fiber'
    import Bag from '../3d/Bag'

    const Loading = () => {
      return (
        <mesh visible position={[0, 0, 0]} rotation={[0, 0, 0]}>
          <sphereGeometry attach="geometry" args={[1, 16, 16]} />
          <meshStandardMaterial
            attach="material"
            color="white"
            transparent
            opacity={0.6}
            roughness={1}
            metalness={0}
          />
        </mesh>
      );
    }

    const Products = () => {
      return (
        <StaticQuery
          query={graphql` ...
    
```

Para que el modelo 3D se cargue correctamente, tenemos que utilizar el componente React **<Suspense>** que nos obliga a crear otro componente que llamaremos **<Loading>** el cual es el que se mostrará mientras se carga el modelo 3D en memoria. En este caso mostraremos una simple esfera como se ve en el código de la imagen anterior. El resultado final es el siguiente.



A continuación, le vamos a añadir la funcionalidad de poder rotar la cámara alrededor de la escena con el ratón. Esta funcionalidad se denomina OrbitControls y se encuentra especificada en la documentación oficial de Three.js. Para añadir esa funcionalidad a nuestro proyecto, vamos a crear un componente denominado OrbitControlsCustom. Va a ser en ese componente donde vamos a importar la clase OrbitControls que nos ofrece Three.js. Para crear este nuevo componente crearemos una carpeta llamada **controls** dentro de nuestra carpeta **3d**.

```

EXPLORER      ...
JS OrbitControlsCustom.js X
src > components > 3d > controls > JS OrbitControlsCustom.js > (1) default
1  import React, { useRef, useEffect } from 'react';
2  import { useThree, extend, useFrame } from 'react-three-fiber';
3  import { orbitControls } from 'three/examples/jsm/controls/OrbitControls';
4
5  extend({OrbitControls});
6  const OrbitControlsCustom = () => {
7    const {
8      camera,
9      gl: { domElement },
10     } = useThree();
11    const controls = useRef();
12    useFrame((state) => controls.current.update());
13    useEffect(() => {
14      camera.position.set(0,0,10)
15      controls.current.enableDamping = true;
16      controls.current.enablePan = false;
17      controls.current.dampingFactor = 0.1;
18      controls.current.minDistance = 5;
19      controls.current.maxDistance = 10;
20    })
21    return <orbitControls ref={controls} args={[camera, domElement]} />;
22  };
23
24 export default OrbitControlsCustom;

```

React-three-fiber nos permite instanciar objetos utilizando la nomenclatura JSX gra-

cias a la función **extend**, función que se encuentra señalada en la imagen junto con su instancia utilizando JSX. Para crear un objeto OrbitControls necesitamos pasarle por parámetro la cámara de la escena y el elemento canvas del html, de tal manera, que lo que realiza OrbitControls por debajo es: añadirle al canvas la funcionalidad de poder rotar la cámara pasada por parámetro alrededor de la escena, siempre que se pulse con el ratón dentro del canvas pasado por parámetro. Para obtener la cámara y el domElement (elemento html canvas) utilizamos la función **useThree()** de react-three-fiber.

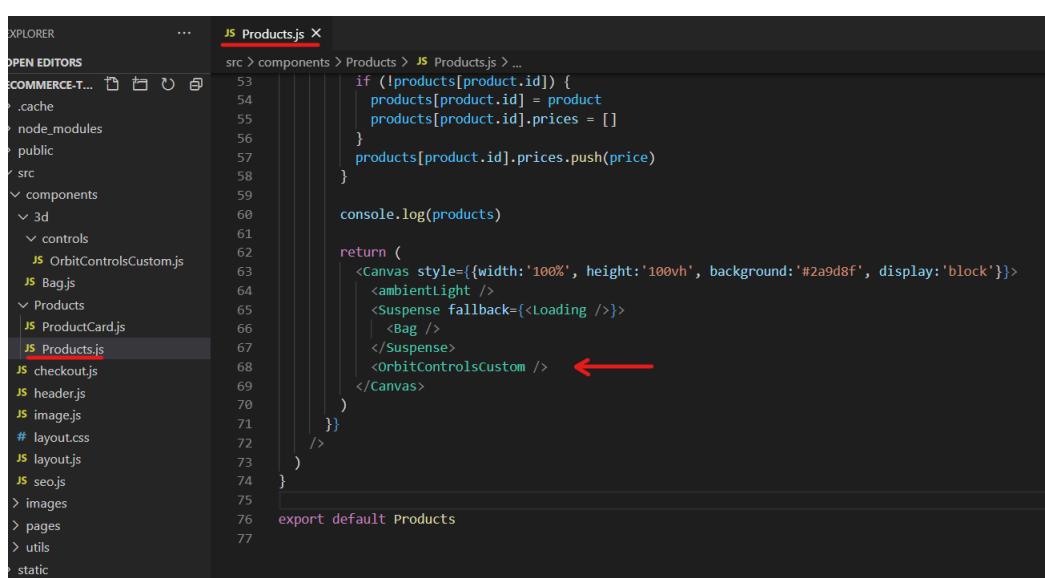
Posteriormente con **useRef()** obtenemos el objeto orbitControls que hemos creado. Al crearlo con JSX necesitamos usar esa función para tener el objeto javascript puro y poderlo manipular tranquilamente con código javascript.

useFrame() es un método que nos proporciona react-three-fiber, sirve para ejecutar el código que se encuentre dentro de él 60 veces por segundo en el caso de que nuestra escena se esté renderizando a 60fps (frames por segundo). Si la máquina es mucho más lenta y se ejecuta a 23 fps, por ejemplo, pues significará que ese código escrito se está ejecutando 23 veces por segundo. Igual que los videojuegos, que cuando bajan de fps se realentizan. Eso es porque el código que se repite en bucle se está ejecutando muchas menos veces por segundo y por lo tanto vemos el videojuego que va más lento. La solución ante este problema es utilizar el ingenio y las técnicas de optimización para lograr que el código se ejecute fluidamente en la mayor cantidad de dispositivos posibles. En nuestra clase OrbitControlsCustom lo que estamos haciendo con este método es actualizar los valores del orbitControls para que la animación de rotación quede más fluida.

useEffect() es un método que se ejecuta una vez que se han instanciado todos los elementos JSX que hayamos escritos, es decir, que dentro de este método todos los objetos con los que hayamos utilizado la función **useRef()** se encuentran instanciados y podemos empezar a utilizarlos. Si no utilizaramos **useEffect()** podría ocurrir que al utilizar el objeto orbitControls todavía no estuviese cargado en memoria y su valor fuese de **undefined**. En este método lo que hacemos es mover la cámara para que el bolso se vea en la escena de forma completa (línea 14), configuramos un efecto de inercia al rotar que le da un poco de másrealismo a la escena (línea 15, 17), desactivamos el efecto fotográfico de barrido o panning puesto que no tenemos un fondo para apreciar dicho efecto (línea 16) y configuramos la distancia mínima y máxima de zoom que podemos tener con el ratón en la escena (línea 18, 19) de la imagen anterior.

Una vez que tenemos el componente OrbitControls creado, lo añadimos a la escena en el componente donde hemos definido nuestro Canvas (**Products.js**).

CAPÍTULO 7. DESARROLLO



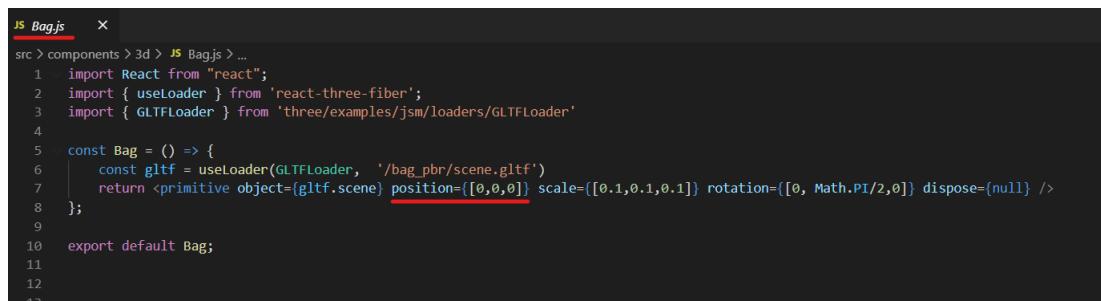
```
JS Products.js X
src > components > Products > JS Products.js > [e] Products > ⚡ <function>
  1 import React, {Suspense} from "react"
  2 import { graphql, staticQuery } from "gatsby"
  3 import {Canvas} from 'react-three-fiber'
  4 import Bag from '../3d/Bag'
  5 import OrbitControlsCustom from '../3d/controls/OrbitControlsCustom' ←
  6
```

```
EXPLORER ... JS Products.js X
src > components > Products > JS Products.js > ...
  53 if (!products[product.id]) {
  54   products[product.id] = product
  55   products[product.id].prices = []
  56 }
  57 products[product.id].prices.push(price)
  58 }

  59 console.log(products)

  60 return (
  61   <Canvas style={{width:'100%', height:'100vh', background:'#2a9d8f', display:'block'}}>
  62     <ambientLight />
  63     <Suspense fallback={<Loading />}>
  64       <Bag />
  65     </Suspense>
  66     <OrbitControlsCustom /> ←
  67   </Canvas>
  68 )
  69 )
  70 )
  71 )
  72 )
  73 )
  74 }
  75
  76 export default Products
  77
```

Si el bolso no rota correctamente sobre su eje, nos aseguramos de que la posición del bolso está en las coordenadas [0,0,0], ya que OrbitControls está configurado por defecto para rotar alrededor de ese punto como referencia.



```
JS Bag.js X
src > components > 3d > JS Bag.js > ...
  1 import React from "react";
  2 import { useLoader } from 'react-three-fiber';
  3 import { GLTFLoader } from 'three/examples/jsm/loaders/GLTFLoader';
  4
  5 const Bag = () => {
  6   const gltf = useLoader(GLTFLoader, '/bag_pbr/scene.gltf')
  7   return <primitive object={gltf.scene} position={[0,0,0]} scale={[0.1,0.1,0.1]} rotation={[0, Math.PI/2,0]} dispose=null />
  8 }
  9
 10 export default Bag;
 11
 12
 13
```

Vamos a añadir una luz de tipo **pointLight** a la escena para crear un efecto más realista de foco iluminando el bolso.



```
return (
  <Canvas style={{width:'100%', height:'100vh', background:'#2a9d8f', display:'block'}}>
    <ambientLight />
    <pointLight position={[10, 10, 10]} /> ←
    <Suspense fallback={<Loading />}>
      <Bag />
    </Suspense>
    <OrbitControlsCustom />
  </Canvas>
)
```

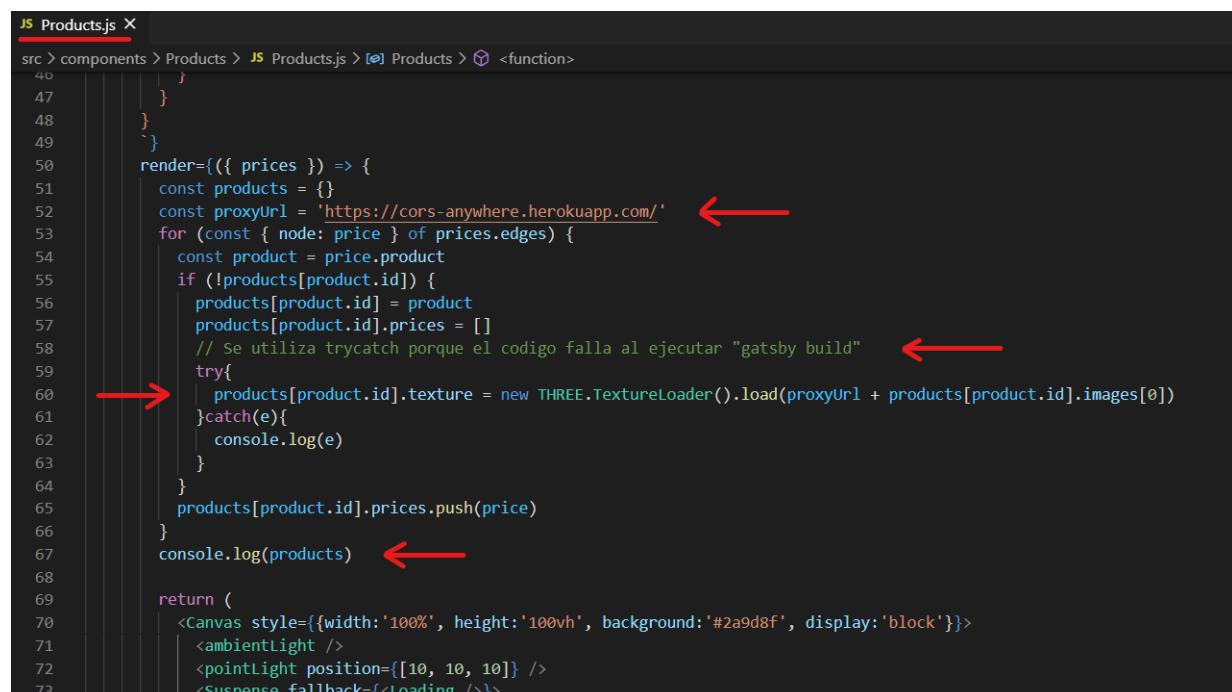


Una vez que tenemos nuestro bolso rotando en pantalla, vamos a proceder a mostrar la imagen de los productos que tenemos en Stripe y a ofrecer la opción de compra de dichos productos. Para ello, vamos a crear un nuevo componente denominado **UI** (User interface), que recibirá por parámetro los productos de **Stripe**, los cuales ya hemos obtenido en el componente **Products**. Antes de crear el componente, vamos a modificar nuestra variable **products** para convertir los enlaces de las imágenes en objetos de tipo textura para posteriormente mostrarlos en un mesh con geometría de tipo círculo y con un material que nos muestra la textura en la geometría que le indiquemos. En nuestro caso la geometría que utilizaremos para mostrar los productos será un círculo con la imagen del producto.

El código que se muestra a continuación soluciona los siguientes dos problemas con los que me he encontrado a la hora del desarrollo:

- Al mostrar las imágenes de Stripe como textura, obteniamos un bloqueo CORS. Lo hemos solucionado utilizando una aplicación alojada en heroku como proxy para que sea la aplicación quien nos devuelva las imágenes sin el bloqueo CORS.

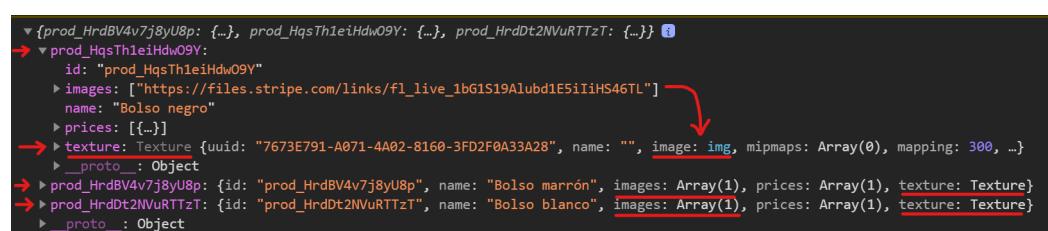
- Cuando ejecutamos el comando `gatsby build` ocurre un error de compilación al ejecutarse la línea de código `THREE.TextureLoader()`. Esto ocurre porque gatsby ejecuta el código con `nodejs` para realizar el build y el método `THREE.TextureLoader()` utiliza por debajo la variable global `document` que tenemos al ejecutar código javascript en un navegador. Cómo el código no se está ejecutando en un navegador sino que se está ejecutando con node, esa variable global no está definida y obtenemos el siguiente error: **ReferenceError: document is not defined**. Para solucionar este problema se ha capturado el error dentro de un bloque trycatch produciendo un fallo controlado a la hora de realizar la build que nos permite finalizar la build correctamente y posteriormente ejecutarse en un navegador sin producir ningún fallo.



```

JS Products.js
src > components > Products > JS Products.js > Products > <function>
46
47
48
49
50 render={({ prices }) => {
51   const products = {}
52   const proxyUrl = 'https://cors-anywhere.herokuapp.com/'
53   for (const { node: price } of prices.edges) {
54     const product = price.product
55     if (!products[product.id]) {
56       products[product.id] = product
57       products[product.id].prices = []
58     // Se utiliza trycatch porque el codigo falla al ejecutar "gatsby build"
59     try{
60       products[product.id].texture = new THREE.TextureLoader().load(proxyUrl + products[product.id].images[0])
61     }catch(e){
62       console.log(e)
63     }
64   }
65   products[product.id].prices.push(price)
66 }
67 console.log(products) ←
68
69 return (
70   <Canvas style={{width:'100%', height:'100vh', background:'#2a9d8f', display:'block'}}>
71     <ambientLight />
72     <pointLight position={[10, 10, 10]} />
73     <Suspense fallback=<Loading />>
74     <Bag />

```



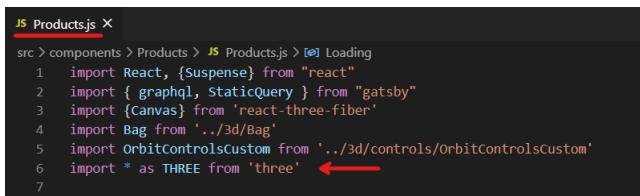
```

▼ {prod_HrdBV4v7j8yU8p: {...}, prod_HqsTh1eiHdw09Y: {...}, prod_HrdDt2NVuRTTzT: {...}} ⓘ
  → ▶ prod_HqsTh1eiHdw09Y:
    id: "prod_HqsTh1eiHdw09Y"
    images: ["https://files.stripe.com/links/fl_live_1bG1S19Alubd1E5iiHS46TL"]
    name: "Bolso negro"
    prices: [...]
  → ▶ texture: Texture {uuid: "7673E791-A071-4A02-8160-3FD2F0A33A28", name: "", image: img, mipmaps: Array(0), mapping: 300, ...}
  → ▶ _proto__: Object
  → ▶ prod_HrdBV4v7j8yU8p: {id: "prod_HrdBV4v7j8yU8p", name: "Bolso marrón", images: Array(1), prices: Array(1), texture: Texture}
  → ▶ prod_HrdDt2NVuRTTzT: {id: "prod_HrdDt2NVuRTTzT", name: "Bolso blanco", images: Array(1), prices: Array(1), texture: Texture}
  → ▶ _proto__: Object

```

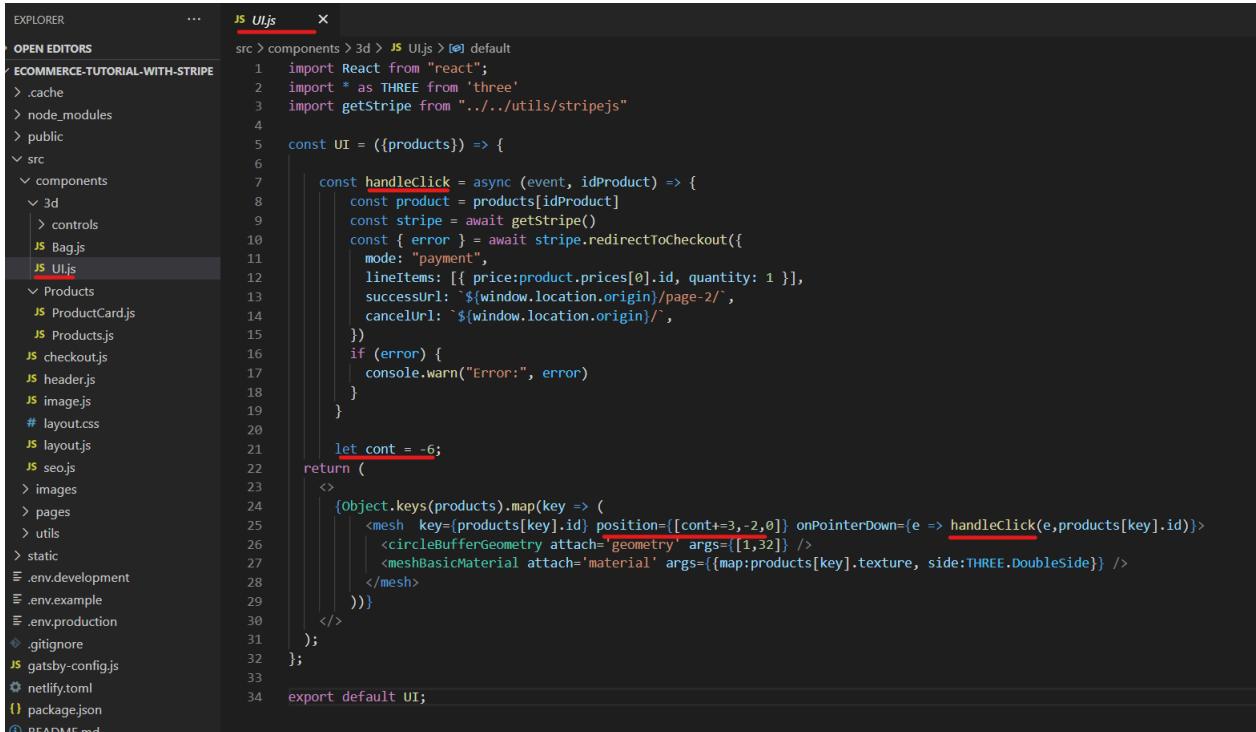
Para ejecutar la línea de código `THREE.TextureLoader()` tenemos que importar la librería `THREE`.

7.5. CREACIÓN DE LA ESCENA 3D CON REACT-THREE-FIBER



```
JS Products.js
src > components > Products > JS Products.js > [↻] Loading
1 import React, {Suspense} from "react"
2 import { graphql, StaticQuery } from "gatsby"
3 import {Canvas} from 'react-three-fiber'
4 import Bag from './3d/Bag'
5 import OrbitControlsCustom from '../3d/controls/OrbitControlsCustom'
6 import * as THREE from 'three' ←
7
```

Una vez que obtenemos la salida por consola mostrada anteriormente, procederemos a la creación el componente UI.



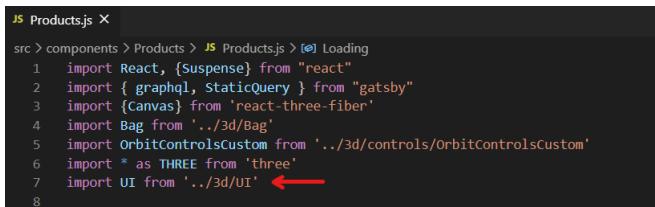
EXPLORER

- OPEN EDITORS
- ECOMMERCE-TUTORIAL-WITH-STRIPE
 - >.cache
 - > node_modules
 - > public
 - ✓ src
 - ✓ components
 - ✓ 3d
 - > controls
 - JS Bag.js
 - JS UI.js**
 - Products
 - JS ProductCard.js
 - JS Products.js
 - JS checkout.js
 - JS header.js
 - JS image.js
 - # layout.css
 - JS layout.js
 - JS seo.js
 - images
 - pages
 - > utils
 - > static
 - ✗ .env.development
 - ✗ .env.example
 - ✗ .env.production
 - ✗ .gitignore
 - JS gatsby-config.js
 - netlify.toml
 - { package.json
 - README.md

JS UI.js

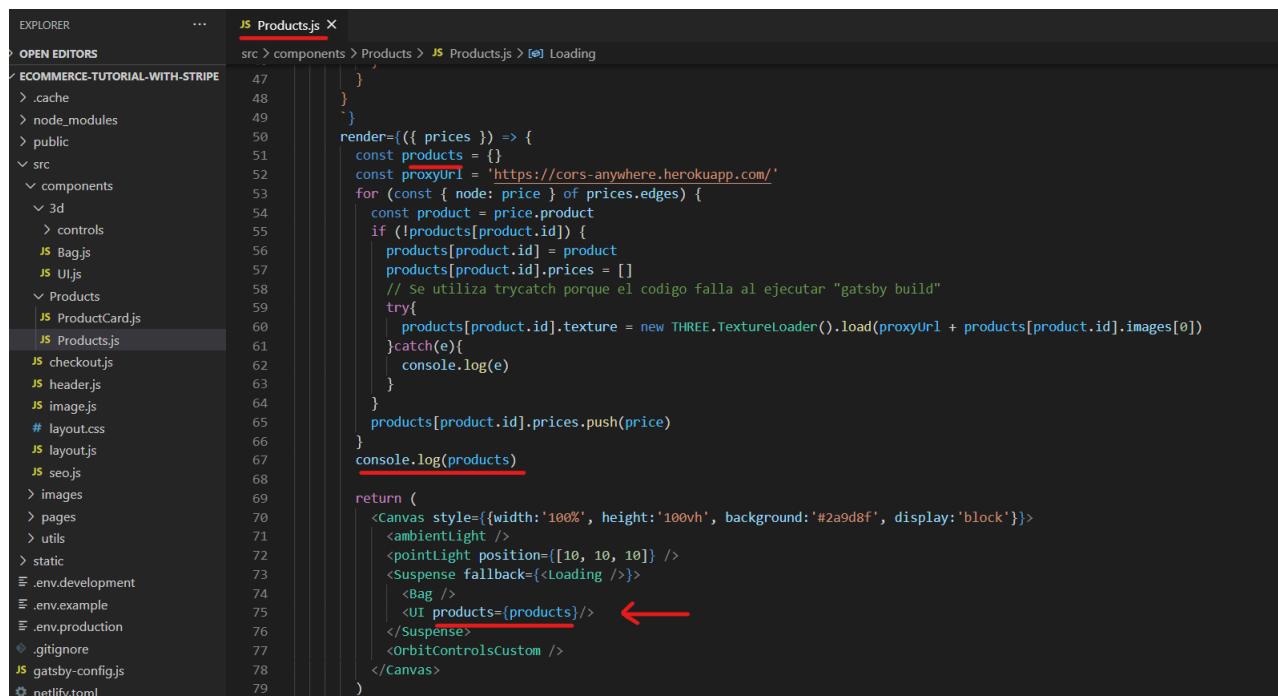
```
src > components > 3d > JS UI.js > [↻] default
1 import React from "react";
2 import * as THREE from 'three';
3 import getStripe from "../utils/stripejs";
4
5 const UI = ({products}) => {
6   const handleClick = async (event, idProduct) => {
7     const product = products[idProduct];
8     const stripe = await getStripe();
9     const { error } = await stripe.redirectToCheckout({
10       mode: "payment",
11       lineItems: [{ price:product.prices[0].id, quantity: 1 }],
12       successUrl: `${window.location.origin}/page-2/`,
13       cancelUrl: `${window.location.origin}/`,
14     });
15     if (error) {
16       console.warn("Error:", error);
17     }
18   }
19
20   let cont = -6;
21   return (
22     <>
23       {Object.keys(products).map(key => (
24         <mesh key={products[key].id} position={[cont+3,-2,0]} onPointerDown={e => handleClick(e,products[key].id)}>
25           <circleBufferGeometry attach='geometry' args={[1,32]} />
26           <meshBasicMaterial attach='material' args={[{map:products[key].texture, side:THREE.DoubleSide}]}/>
27         </mesh>
28       ))}
29     </>
30   );
31 };
32
33
34 export default UI;
```

Y añadiremos dicho componente dentro de nuestro Canvas para que aparezca en la escena.



```
JS Products.js
src > components > Products > JS Products.js > [↻] Loading
1 import React, {Suspense} from "react"
2 import { graphql, StaticQuery } from "gatsby"
3 import {Canvas} from 'react-three-fiber'
4 import Bag from './3d/Bag'
5 import OrbitControlsCustom from '../3d/controls/OrbitControlsCustom'
6 import * as THREE from 'three'
7 import UI from '../3d/UI' ←
```

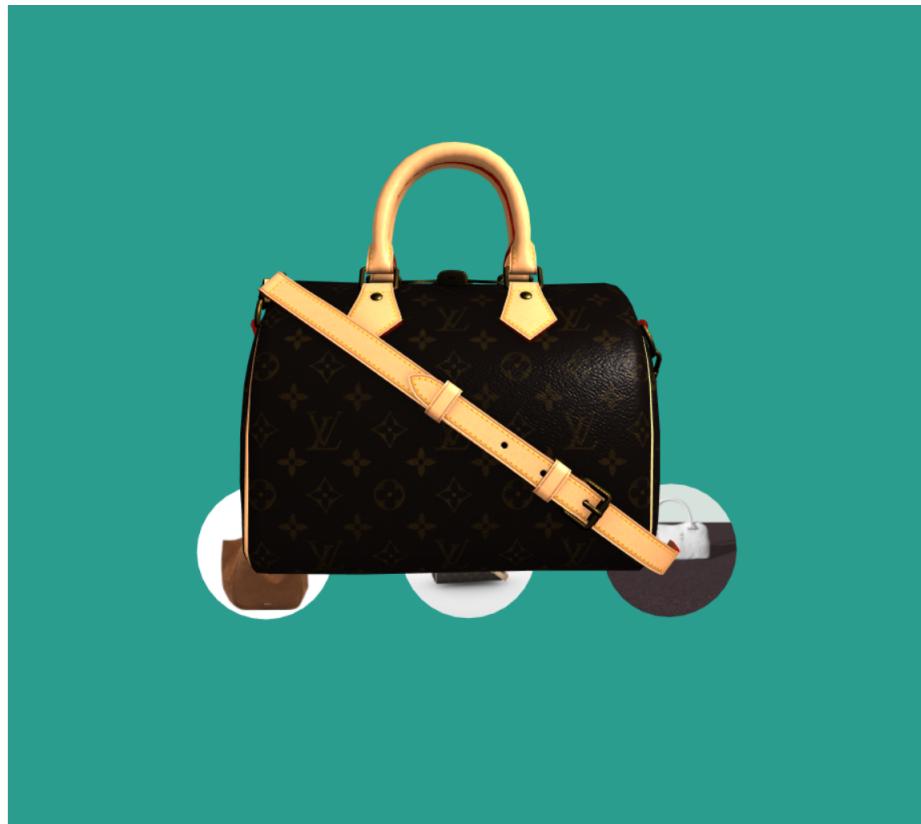
CAPÍTULO 7. DESARROLLO



```
EXPLORER      ...
OPEN EDITORS
ECOMMERCE-TUTORIAL-WITH-STRIPE
> .cache
> node_modules
> public
< src
  < components
    < 3d
      > controls
        JS Bag.js
        JS UI.js
    < Products
      JS ProductCard.js
      JS Products.js
      JS checkout.js
      JS header.js
      JS image.js
      # layout.css
      JS layout.js
      JS seo.js
    > images
    > pages
    > utils
    > static
  < .env.development
  < .env.example
  < .env.production
  < .gitignore
  JS gatsby-config.js
  netlify.toml

JS Products.js
src > components > Products > JS Products.js > [?] Loading
47   }
48   }
49   }
50   }
51   render={({ prices }) => {
52     const proxyUrl = 'https://cors-anywhere.herokuapp.com/'
53     for (const { node: price } of prices.edges) {
54       const product = price.product
55       if (!products[product.id]) {
56         products[product.id] = product
57         products[product.id].prices = []
58       }
59       // Se utiliza trycatch porque el código falla al ejecutar "gatsby build"
60       try{
61         products[product.id].texture = new THREE.TextureLoader().load(proxyUrl + products[product.id].images[0])
62       }catch(e){
63         console.log(e)
64       }
65     }
66     products[product.id].prices.push(price)
67   }
68   console.log(products)
69
70   return (
71     <Canvas style={{width:'100%', height:'100vh', background:'#2a9d8f', display:'block'}}>
72       <ambientLight />
73       <pointLight position={[10, 10, 10]} />
74       <Suspense fallback={<Loading />}>
75         <Bag />
76         <UI products={products}/> ←
77       </Suspense>
78       <OrbitControlsCustom />
79     </Canvas>
)
```

Obteniendo el siguiente resultado.

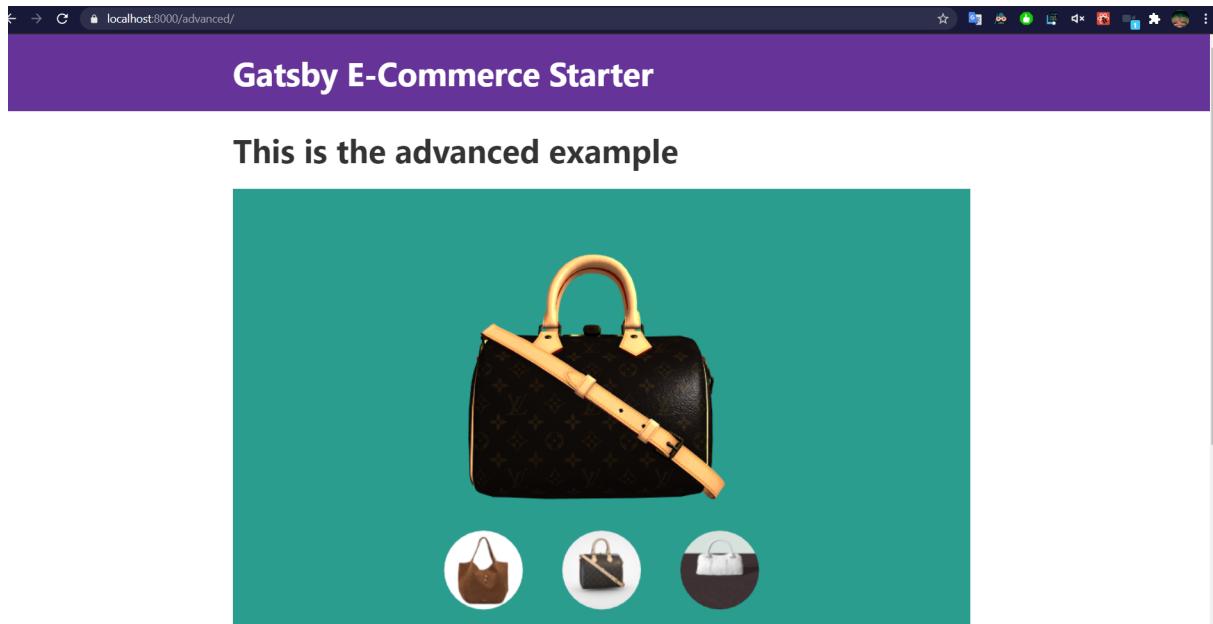


Encuadramos la escena subiendo el bolso 2 unidades en el eje Y, quedándonos la

7.5. CREACIÓN DE LA ESCENA 3D CON REACT-THREE-FIBER

escena finalmente como se muestra a continuación.

```
JS Bag.js
src > components > 3d > JS Bag.js > ...
1 import React from "react";
2 import { useLoader } from 'react-three-fiber';
3 import { GLTFLoader } from 'three/examples/jsm/loaders/GLTFLoader';
4
5 const Bag = () => {
6   const gltf = useLoader(GLTFLoader, '/bag_pbr/scene.gltf')
7   return <primitive object={gltf.scene} position={[0,2,0]} scale={[0.1,0.1,0.1]} rotation={[0, Math.PI/2,0]} dispose={null} />
8 };
9
10 export default Bag;
11
```

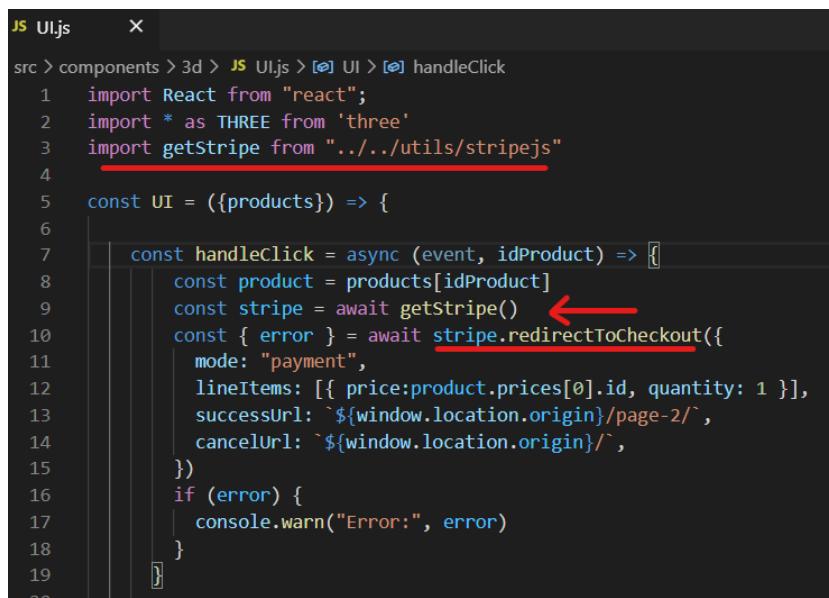


Vamos a proceder a explicar el código del componente UI. Lo primero en lo que nos vamos a fijar es en el siguiente código.

```
JS UI.js
src > components > 3d > JS UI.js > UI > handleClick
21 let cont = -6;
22 return (
23   <>
24     <object.keys(products).map(key => (
25       <mesh key={products[key].id} position={[cont+3,-2,0]} onPointerDown={e => handleClick(e,products[key].id)}>
26         <circleBufferGeometry attach='geometry' args={[1,32]} />
27         <meshBasicMaterial attach='material' args={{map:products[key].texture, side:THREE.DoubleSide}} />
28       </mesh>
29     )));
30   </>
31 );
32 );
33
34 export default UI;
```

Vamos a crear un mesh por cada producto. Dicho mesh tendrá una geometría de tipo círculo (para ver qué significan los parámetro introducidos en circleBufferGeometry dirijirse a la documentación oficial de Three.js) y un material con la textura de la imagen que tenemos asociada al producto en Stripe. Le indicaremos al material que se

muestre dicha imagen en ambas caras del círculo para que al rotar el círculo siempre se vea la imagen. Los 3 meshes se mostrarán a lo largo del eje x, teniendo el primero una posición de $x = -3$, el segundo $x = 0$ y el tercero $x = 3$. De tal manera que veremos los círculos puesto uno al lado del otro. Por último, le añadimos a cada mesh un evento cada vez que el usuario pulse en el mesh. Este evento se captura con la función **onPointerDown** (**onClick** solo funciona al pulsar con el ratón, no con el dedo en un smartphone). Dicha función llama a una nueva función que hemos creado en la cual se le pasa por parámetro el evento de click correspondiente (el cual no vamos a utilizar en este caso) y el id del producto asociado al mesh que hemos pulsado. En esta nueva función llamada **handleClick** lo que hacemos es usar la instancia singleton de stripe que nos proporciona la plantilla que estamos utilizando y llamar al método responsable de dirigirnos a la página checkout de Stripe para proceder al pago del producto seleccionado. En dicho método le indicamos la url a la que nos tiene que llevar si la compra se realiza exitosamente o a la url que nos tiene que llevar si se cancela la compra.

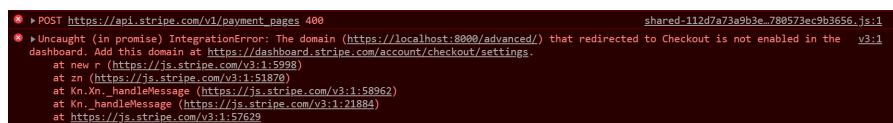


```

JS UI.js      X
src > components > 3d > JS UI.js > [o] UI > [o] handleClick
1 import React from "react";
2 import * as THREE from 'three'
3 import stripe from "../../utils/stripejs"
4
5 const UI = ({products}) => {
6
7   const handleClick = async (event, idProduct) => [
8     const product = products[idProduct]
9     const stripe = await stripe() ←
10    const { error } = await stripe.redirectToCheckout({
11      mode: "payment",
12      lineItems: [{ price:product.prices[0].id, quantity: 1 }],
13      successUrl: `${window.location.origin}/page-2/`,
14      cancelUrl: `${window.location.origin}/`,
15    })
16    if (error) {
17      console.warn("Error:", error)
18    }
19  ]
20}

```

Con esto ya tendríamos finalizada la parte frontend del proyecto. Si probamos a realizar una compra, veremos que nos aparecerá un error en consola al intentar dirigirse al checkout de Stripe. A continuación una imagen del error.



```

shared-112d7a73a9b3e..780573ec9b3656.js:1
✖ POST https://api.stripe.com/v1/payment_pages 400
✖ Uncaught (in promise) IntegrationError: The domain (https://localhost:8000/advanced/) that redirected to Checkout is not enabled in the v3:1
  dashboard. Add this domain at https://dashboard.stripe.com/account/checkout/settings.
  at new r (https://js.stripe.com/v3:1:5998)
  at zn (https://js.stripe.com/v3:1:51870)
  at Kn.Xn.handleMessage (https://js.stripe.com/v3:1:58962)
  at Kn._handleMessage (https://js.stripe.com/v3:1:21884)
  at https://js.stripe.com/v3:1:57629

```

Esto ocurre porque Stripe no permite que una aplicación acceda a su checkout a menos que tenga un nombre de dominio real y esté alojada en un servidor con https.

7.5. CREACIÓN DE LA ESCENA 3D CON REACT-THREE-FIBER

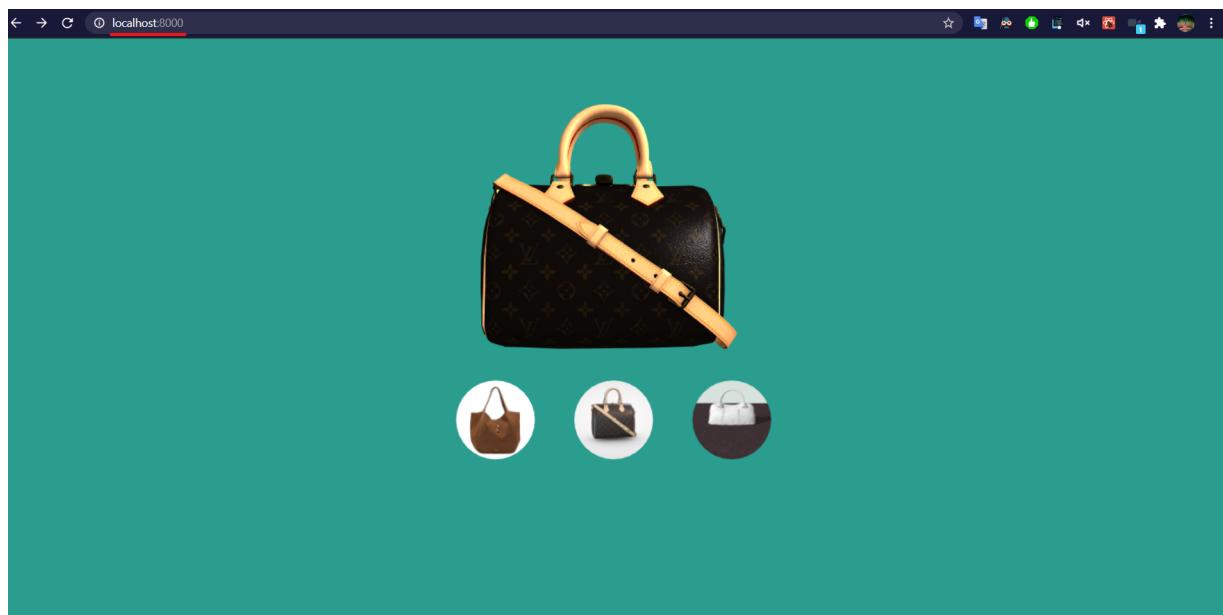
Eso será lo que haremos en los siguientes capítulos.

Antes de pasar al siguiente capítulo, vamos a poner nuestra escena 3D en el enlace / en vez del enlace /advanced, y vamos a eliminar todo lo que nos viene por defecto con Gatsby dejando la escena a pantalla completa. Para ello realizamos los siguientes cambios.

```
JS index.js x
src > pages > JS index.js > ...
1 import React from "react"
2 import Products from "../components/Products/Products"
3 import "../components/layout.css" ←
4
5 const IndexPage = () => (
6   <Products /> ←
7 )
8
9 export default IndexPage
10
```

```
# layout.css x
src > components > # layout.css > html
1 html {
2   font-family: sans-serif;
3   -ms-text-size-adjust: 100%;
4   -webkit-text-size-adjust: 100%;
5 }
6 html, body { ←
7   margin: 0;
8   padding: 0;
9   overflow: hidden; ←
10}
```

```
JS Products.js x
src > components > Products > JS Products.js > ...
68
69   return (
70     <Canvas style={{width:'100%', height:'100vh', background:'#2a9d8f', display:'block'}}>
71       <ambientLight />
72       <pointLight position={[10, 10, 10]} />
73       <Suspense fallback={<Loading />}>
74         <Bag />
75         <UI products={products}/>
76       </Suspense>
77       <OrbitControlsCustom />
78     </Canvas>
79   )
80 }
81
82
83
84
85 export default Products
86
```



En **index.js** llamamos únicamente a nuestro componente **Products**, que es el que contiene el Canvas con la escena 3D. Seguidamente importamos el fichero **layout.css**, en el cual indicamos que tanto el margin como el padding sea de 0px. Con **overflow:hidden** eliminamos del navegador la barra de scroll lateral. Por último, indicamos en el componente **Products** que el canvas ocupe el máximo, tanto de alto como de ancho, de la pantalla y le añadimos la característica de **display:block**.

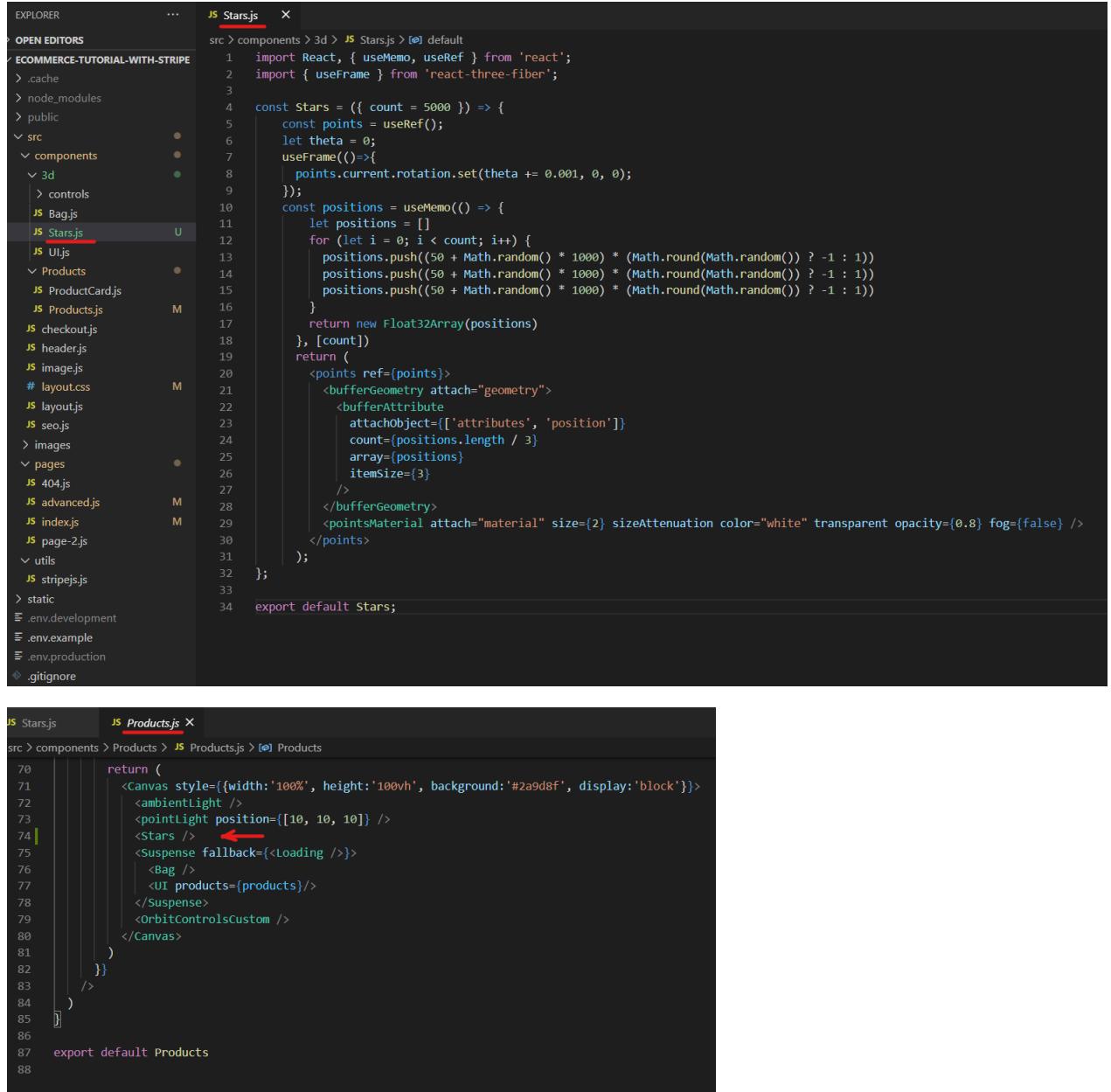
Seguidamente modificamos la página **advanced.js** para dejarla de la siguiente forma.

```
JS advanced.js X
src > pages > JS advancedjs > ...
1  import React from "react"
2  import { Link } from "gatsby"
3
4  import Layout from "../components/layout"
5
6  const AdvancedExamplePage = () => (
7    <Layout>
8      <Link to="/">Volver al inicio</Link>
9    </Layout>
10 )
11
12 export default AdvancedExamplePage
13
```

Por último, vamos a añadir el detalle adicional de poner muchos puntos pequeños blancos moviéndose en el fondo, quedando un bonito efecto visual. Para realizar dicho efecto vamos a crear un nuevo componente 3d llamado **Stars** (estrellas) y lo añadire-

7.5. CREACIÓN DE LA ESCENA 3D CON REACT-THREE-FIBER

mos dentro del Canvas que se encuentra en **Products.js**. Como este componente no es obligatorio en el proyecto y no entra dentro de nuestros objetivos, no procederemos a la explicación del código por falta de tiempo.



```

EXPLORER ... JS Stars.js X
OPEN EDITORS
> ECOMMERCE-TUTORIAL-WITH-STRIPE
> .cache
> node_modules
> public
> src
> components
> 3d
> controls
JS Bag.js
JS Stars.js U
JS Ul.js
> Products
JS ProductCard.js
JS Products.js M
JS checkout.js
JS header.js
JS image.js
# layout.css M
JS layout.js
JS seo.js
> images
> pages
JS 404.js
JS advanced.js M
JS index.js M
JS page-2.js
> utils
JS stripe.js
> static
.env.development
.env.example
.env.production
.gitignore

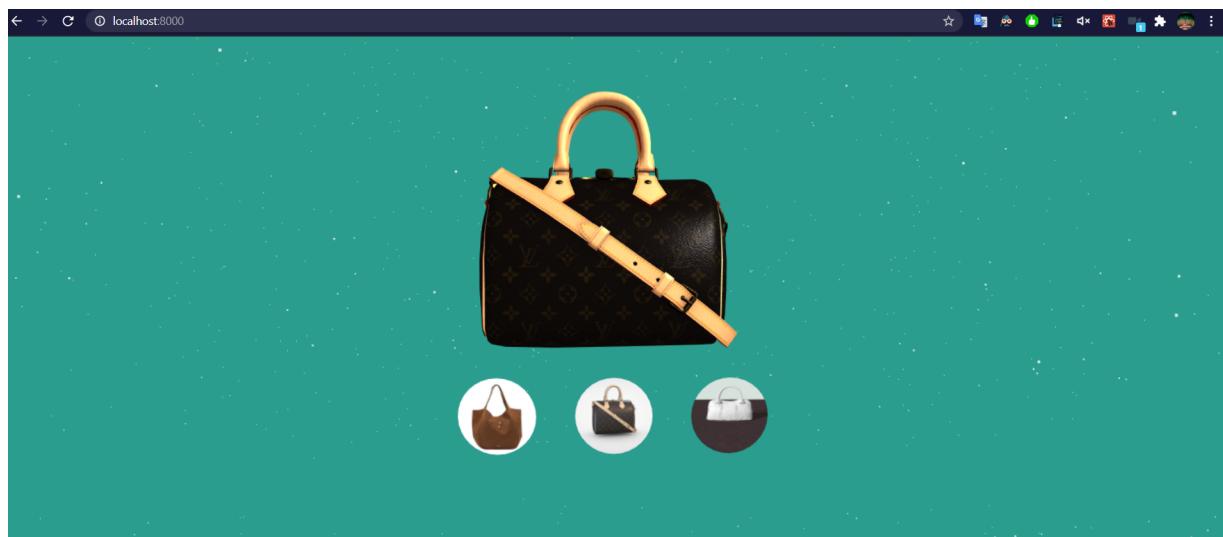
src > components > 3d > JS Stars.js > [e] default
1 import React, { useMemo, useRef } from 'react';
2 import { useFrame } from 'react-three-fiber';
3
4 const Stars = ({ count = 5000 }) => {
5   const points = useRef();
6   let theta = 0;
7   useFrame(()=>{
8     points.current.rotation.set(theta += 0.001, 0, 0);
9   });
10  const positions = useMemo(() => {
11    let positions = [];
12    for (let i = 0; i < count; i++) {
13      positions.push((50 + Math.random() * 1000) * (Math.round(Math.random()) ? -1 : 1));
14      positions.push((50 + Math.random() * 1000) * (Math.round(Math.random()) ? -1 : 1));
15      positions.push((50 + Math.random() * 1000) * (Math.round(Math.random()) ? -1 : 1));
16    }
17    return new Float32Array(positions);
18  }, [count]);
19  return (
20    <points ref={points}>
21      <bufferGeometry attach="geometry">
22        <bufferAttribute
23          attachObject={['attributes', 'position']}
24          count={positions.length / 3}
25          array={positions}
26          itemSize={3}
27        />
28      </bufferGeometry>
29      <pointsMaterial attach="material" size={2} sizeAttenuation color="white" transparent opacity={0.8} fog={false} />
30    </points>
31  );
32}
33
34 export default Stars;

JS Stars.js JS Products.js X
src > components > Products > JS Products.js > [e] Products
return (
  <Canvas style={{width:'100%', height:'100vh', background:'#2a9d8f', display:'block'}}>
    <ambientLight />
    <pointlight position={[10, 10, 10]} />
    <Stars /> ←
    <Suspense fallback={<Loading />}>
      <Bag />
      <UI products={products}/>
    </Suspense>
    <OrbitControlsCustom />
  </Canvas>
)
}
}

export default Products

```

Finalmente nuestro proyecto con las estrellas de fondo, quedaría de la siguiente forma.



7.6 CREACIÓN DEL FICHERO DOCKER-COMPOSE.YAML

En este capítulo crearemos el fichero responsable para poder desplegar nuestro proyecto en cualquier servidor Linux.

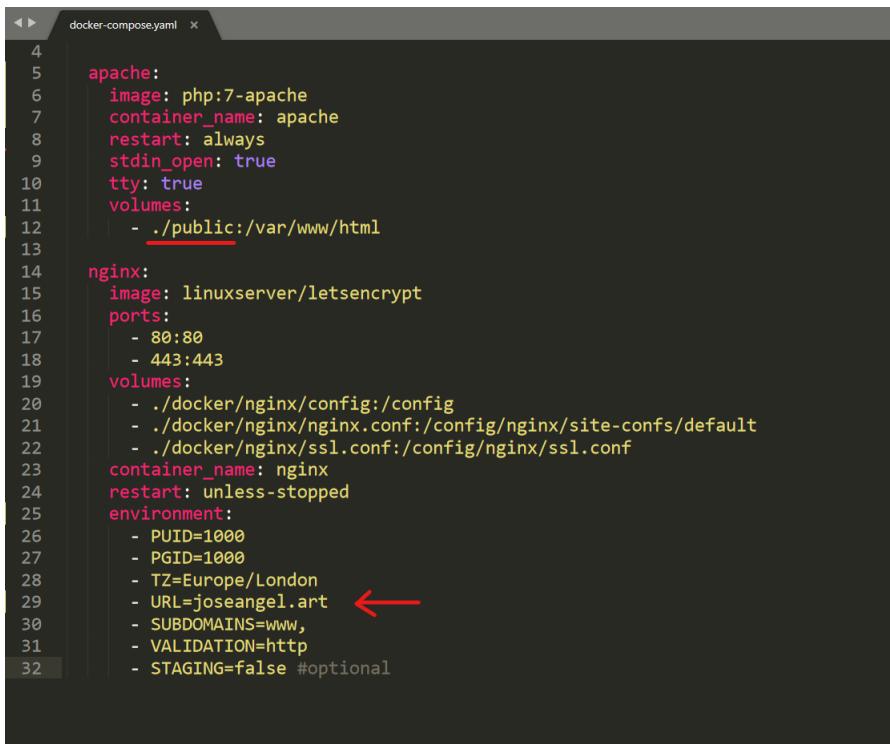
Para crear un fichero docker-compose que utilice nginx con Let's Encrypt para el https, nos hemos basado en el siguiente tutorial del canal de Youtube AyyazTech:

- Install Let's Encrypt SSL Certificate using Docker, Wordpress and DigitalOcean | Web Development. URL <https://www.youtube.com/watch?v=1bXc6mKh7U0>)

- Repositorio del proyecto. URL <https://github.com/ayyazzafar/Tutorial-1-Install-SSL-0>

Procederemos a clonar dicho repositorio en el escritorio de nuestro ordenador y modificaremos el fichero docker-compose.yaml para que quede de la siguiente forma.

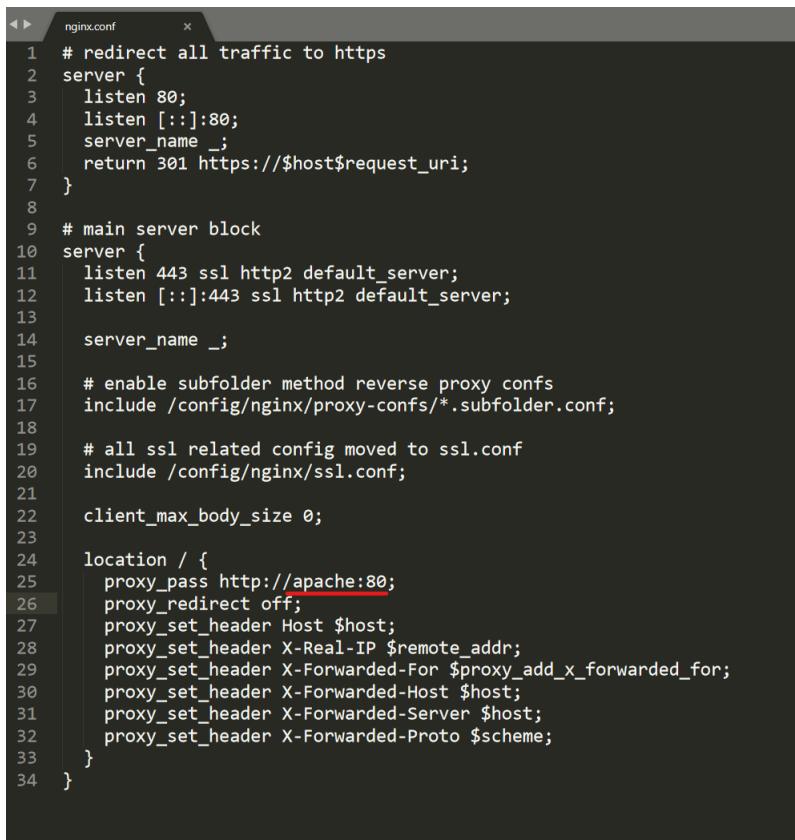
7.6. CREACIÓN DEL FICHERO DOCKER-COMPOSE.YAML



```
4
5 apache:
6   image: php:7-apache
7   container_name: apache
8   restart: always
9   stdin_open: true
10  tty: true
11  volumes:
12    - ./public:/var/www/html
13
14 nginx:
15   image: linuxserver/letsencrypt
16   ports:
17     - 80:80
18     - 443:443
19   volumes:
20     - ./docker/nginx/config:/config
21     - ./docker/nginx/nginx.conf:/config/nginx/site-confs/default
22     - ./docker/nginx/ssl.conf:/config/nginx/ssl.conf
23   container_name: nginx
24   restart: unless-stopped
25   environment:
26     - PUID=1000
27     - PGID=1000
28     - TZ=Europe/London
29     - URL=joseangel.art ←
30     - SUBDOMAINS=www,
31     - VALIDATION=https
32     - STAGING=false #optional
```

En este fichero le estamos indicando a Docker que instale una imagen de Apache, la cual será la que se encargará de servir nuestro proyecto Gatsby. Para que funcione correctamente, debemos de crear una carpeta **public**. Dicha carpeta será la carpeta que Gatsby nos creará automáticamente al ejecutar el comando **gatsby build** en nuestro proyecto. Lo que tendremos que hacer es copiar esa carpeta y pegarla dentro de la carpeta en la que tenemos nuestro fichero docker-compose.yaml. También le tenemos que indicar el nombre del dominio que compraremos para alojar nuestra web en Internet, en nuestro caso el nombre de dominio será **joseangel.art** el cual compraremos en el siguiente capítulo.

Posteriormente modificaremos el archivo **docker/nginx/nginx.conf** para indicar que el contenedor al que le aplicamos el https ha pasado de llamarse wordpress (nombre que venía por defecto en el tutorial que utilizamos) a llamarse apache.



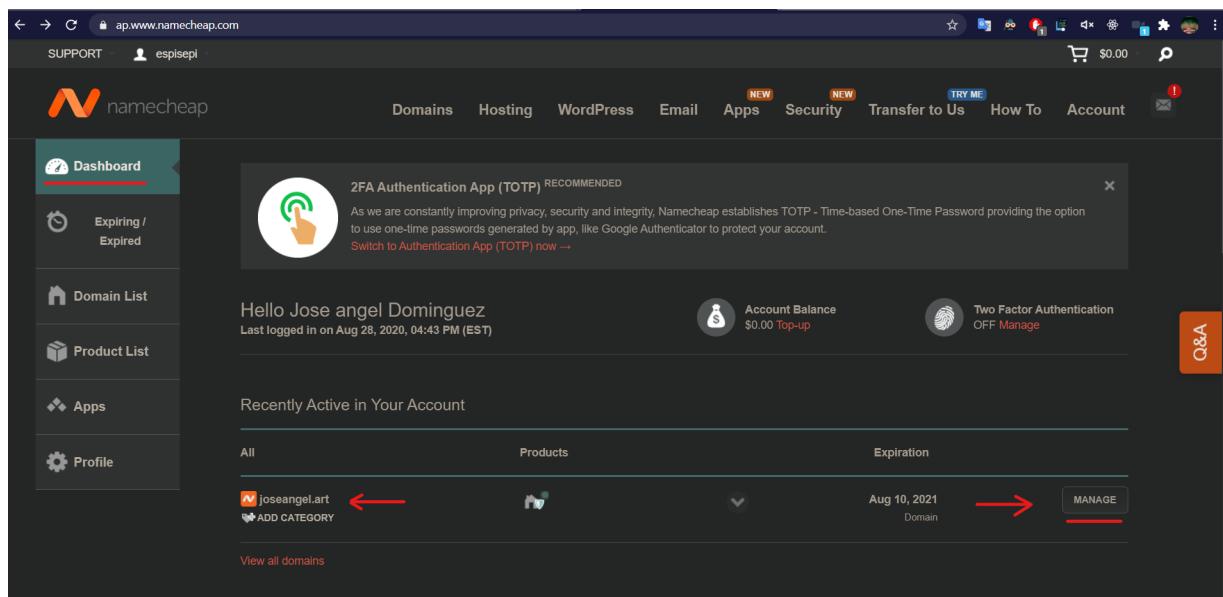
```
nginx.conf
1 # redirect all traffic to https
2 server {
3     listen 80;
4     listen [::]:80;
5     server_name _;
6     return 301 https://$host$request_uri;
7 }
8
9 # main server block
10 server {
11     listen 443 ssl http2 default_server;
12     listen [::]:443 ssl http2 default_server;
13
14     server_name _;
15
16     # enable subfolder method reverse proxy confs
17     include /config/nginx/proxy-confs/* .subfolder.conf;
18
19     # all ssl related config moved to ssl.conf
20     include /config/nginx/ssl.conf;
21
22     client_max_body_size 0;
23
24     location / {
25         proxy_pass http://apache:80;
26         proxy_redirect off;
27         proxy_set_header Host $host;
28         proxy_set_header X-Real-IP $remote_addr;
29         proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
30         proxy_set_header X-Forwarded-Host $host;
31         proxy_set_header X-Forwarded-Server $host;
32         proxy_set_header X-Forwarded-Proto $scheme;
33     }
34 }
```

Posteriormente, eliminaremos el archivo **.env** y la carpeta **docker/mysql** puesto que no nos va a hacer falta.

7.7 REGISTRO DE UN NOMBRE DE DOMINIO

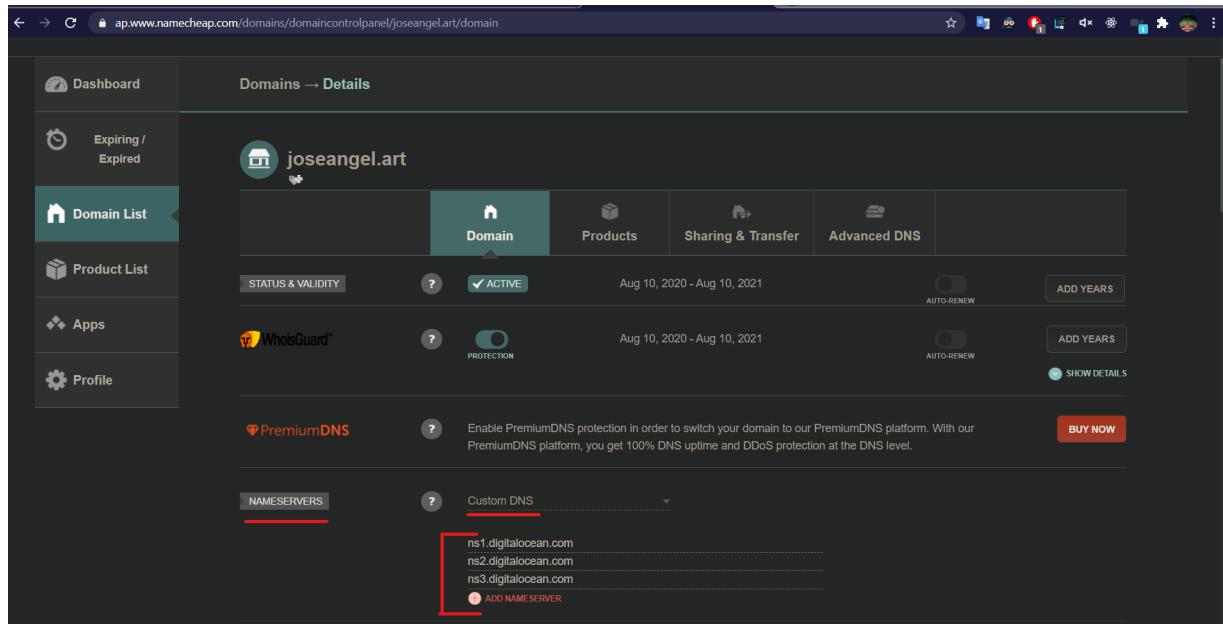
Para comprar un nombre de dominio, utilizaremos la web www.namecheap.com. Nos registraremos en dicha web, buscaremos un nombre de dominio que nos guste, lo añadiremos al carrito y lo compraremos. Una vez comprado, nos dirigimos a nuestro **Dashboard** y seleccionamos la opción de **MANAGE** que aparece al lado del nombre de dominio que acabamos de comprar.

7.7. REGISTRO DE UN NOMBRE DE DOMINIO



The screenshot shows the Namecheap dashboard. On the left sidebar, there are links for Dashboard, Expiring / Expired, Domain List, Product List, Apps, and Profile. The main area displays a message about 2FA Authentication App (TOTP). Below it, it says "Hello Jose angel Dominguez" and shows the last login date as "Aug 28, 2020, 04:43 PM (EST)". It also shows "Account Balance \$0.00 Top-up" and "Two Factor Authentication OFF Manage". Under "Recently Active in Your Account", there is a table with columns "All", "Products", and "Expiration". A row for the domain "joseangel.art" is shown, with a red arrow pointing to its name. The expiration date is listed as "Aug 10, 2021". There is a "MANAGE" button at the end of the row.

Al pulsar sobre esta opción, nos aparecerá la siguiente ventana, en la cual le indicaremos a los servidores dns de namecheap que cuando una máquina ingrese a nuestro dominio se redirija a los servidores dns de DigitalOcean, que será la plataforma de hosting en la que alojaremos nuestro proyecto.

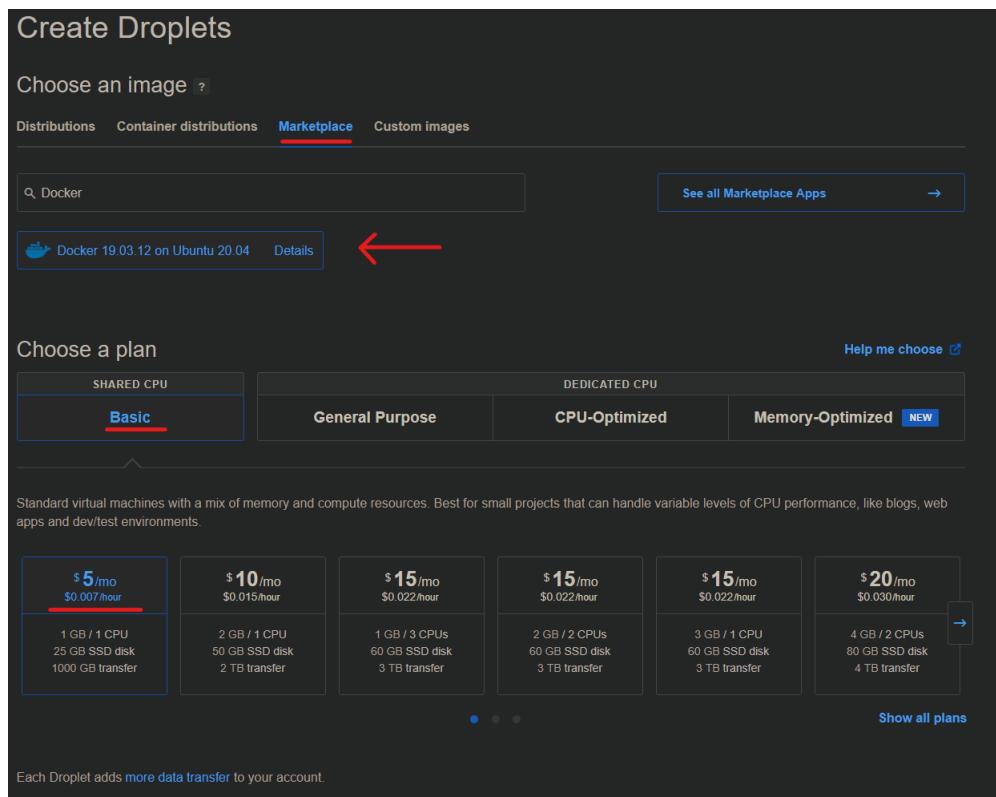


The screenshot shows the Namecheap domain control panel for the domain "joseangel.art". The left sidebar includes links for Dashboard, Expiring / Expired, Domain List, Product List, Apps, and Profile. The main page has a header "Domains → Details" and shows the domain "joseangel.art". Below the domain name, there are tabs for Domain, Products, Sharing & Transfer, and Advanced DNS. Under "STATUS & VALIDITY", the domain is marked as ACTIVE and valid from Aug 10, 2020, to Aug 10, 2021. Under "WhoisGuard", it is turned off. Under "PremiumDNS", there is a "BUY NOW" button. The "NAMESERV" tab is selected, showing a "Custom DNS" section with three entries: "ns1.digitalocean.com", "ns2.digitalocean.com", and "ns3.digitalocean.com". A red box highlights this section.

Una vez realizado estos pasos, procederemos a crearnos una cuenta en DigitalOcean para alojar nuestro proyecto.

7.8 DESPLIEGUE DEL PROYECTO EN DIGITALOCEAN

Lo primero que haremos será crearnos una cuenta en DigitalOcean. Una vez creada, crearemos un nuevo proyecto (yo lo llamaré joseangel) y dentro de ese nuevo proyecto crearemos un Droplet. Un Droplet es una máquina virtual en la que nosotros elegiremos la distribución Linux que queremos instalar y los componentes internos de la máquina. Cuanto mejor sean los componentes, más dinero nos costará al mes mantener dicha máquina. Nosotros llamaremos a nuestra máquina **joseangelmachine** y la configuraremos instalándole por defecto Ubuntu con Docker. Elegiremos el plan más barato que hay (5 dólares/mes) y configuraremos una autenticación ssh para poder conectarnos a la máquina desde nuestra terminal de Windows.



7.8. DESPLIEGUE DEL PROYECTO EN DIGITALOCEAN

Choose a datacenter region

New York	Amsterdam	San Francisco	Singapore	London	Frankfurt
1	1	1	1	1	1

Toronto	Bangalore
1	1

VPC Network

No VPC
This Droplet will have no Private IP

This resource can communicate over Private IP address only with other resources in the same VPC network. [Learn more](#)

Enable Private Networking
To enable Private Networking on this Droplet, select a VPC Network.

OK [Learn more](#)

Select additional options ?

IPv6 User data Monitoring

Authentication ?

SSH keys A more secure authentication method

Password Create a root password to access Droplet (less secure)

Choose your SSH keys
 joseangelssh
[New SSH Key](#)

Finalize and create

How many Droplets?
Deploy multiple Droplets with the same configuration.

Choose a hostname
Give your Droplets an identifying name you will remember them by. Your Droplet name can only contain alphanumeric characters, dashes, and periods.

1 Droplet joeangelmachine

Add tags
Use tags to organize and relate resources. Tags may contain letters, numbers, colons, dashes, and underscores.

Type tags here

Select Project
Assign Droplets to a project

joseangel

Add backups

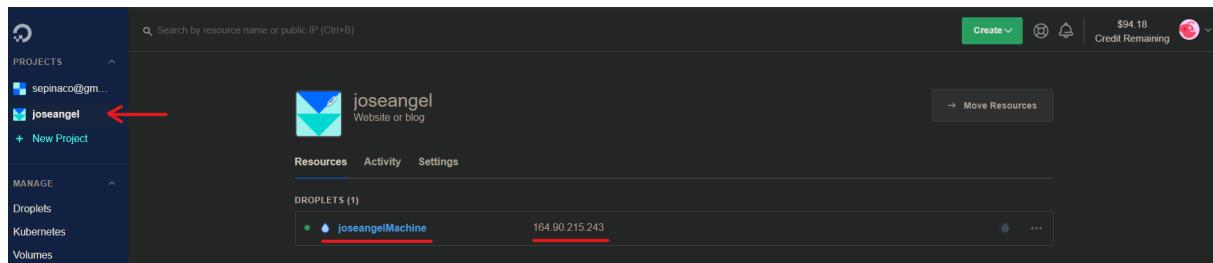
Enable backups RECOMMENDED
A system-level backup is taken once a week, and each backup is retained for 4 weeks.

\$1.00/mo (per Droplet)
20% of the Droplet price

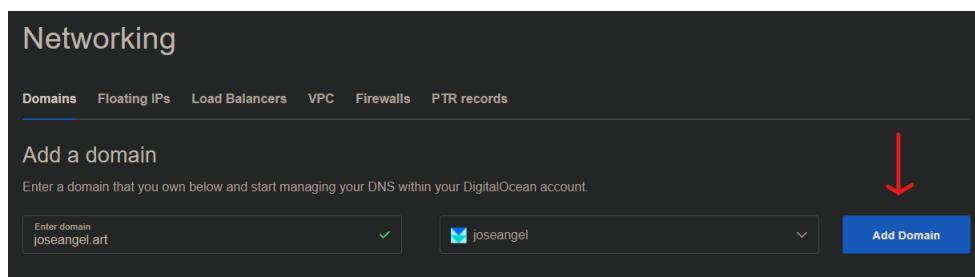
Create Droplet

Una vez creado el Droplet, nuestra pantalla nos quedaría de la siguiente manera.

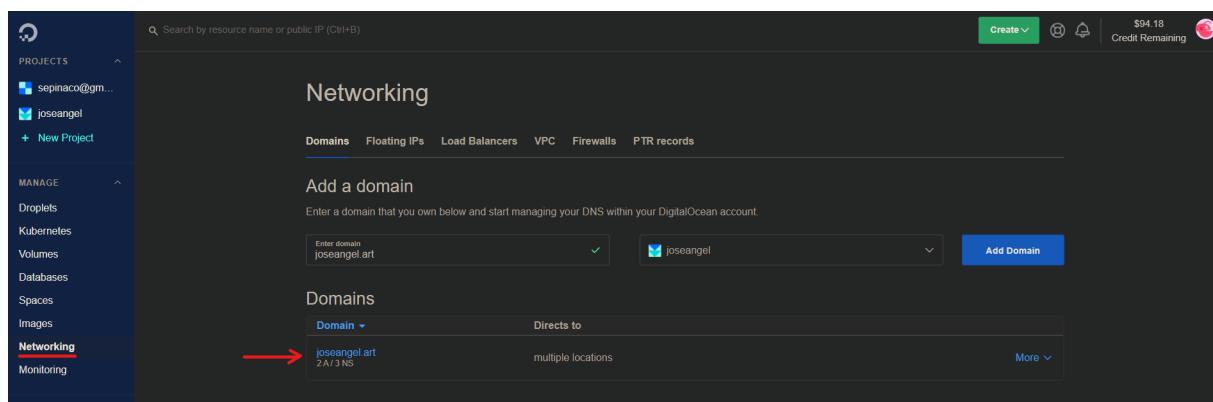
CAPÍTULO 7. DESARROLLO



El siguiente paso es asociar nuestro nombre de dominio a la nueva máquina virtual que hemos creado. Para ello nos vamos al apartado **Networking** y añadimos el dominio **joseangel.art** al proyecto **joseangel**. Seguidamente pulsamos en **Add Domain**.



Para terminar con la configuración, pulsaremos en el dominio que hemos creado y enlazaremos nuestro dominio con la IP del Droplet que creamos anteriormente, como se muestra en la imagen siguiente.



7.8. DESPLIEGUE DEL PROYECTO EN DIGITALOCEAN

Type	Hostname	Value	TTL (seconds)	More
A	www.joseangel.art	directs to 164.90.215.243	3600	More
A	joseangel.art	directs to 164.90.215.243	3600	More
NS	joseangel.art	directs to ns3.digitalocean.com	1800	More
NS	joseangel.art	directs to ns1.digitalocean.com	1800	More
NS	joseangel.art	directs to ns2.digitalocean.com	1800	More

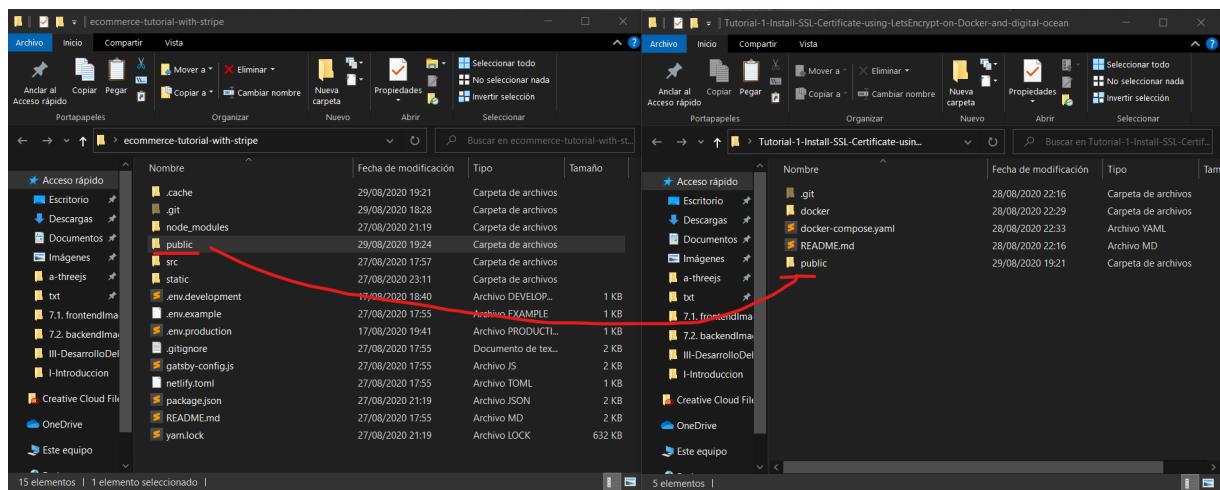
En este punto, ya tendríamos nuestro nombre de dominio asociado a la IP de la máquina virtual (o Droplet) que hemos creado. El siguiente paso sería pasar nuestra carpeta con los ficheros de Docker y el proyecto preparado para producción a la máquina virtual para terminar de desplegar nuestro proyecto utilizando Docker. Para ello vamos a realizar la build de nuestro proyecto Gatsby y a copiar y pegar la carpeta public que se nos va a crear, en la carpeta donde se encuentran nuestros ficheros de Docker.

Lo primero que haremos será ejecutar el comando **gatsby build** en la raíz de nuestro proyecto Gatsby.

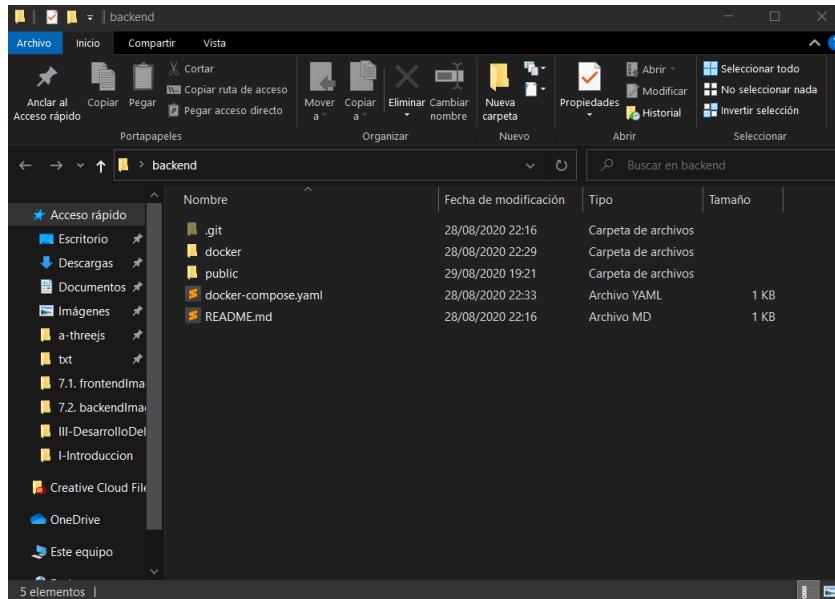
```
PS C:\Users\Jose Angel Dominguez\Desktop\ecommerce-tutorial-with-stripe> gatsby build ←
success open and validate gatsby-configs - 0.035s
success load plugins - 1.591s
success onPreInit - 0.051s
success delete html and css files from previous builds - 0.010s
success initialize cache - 0.010s
```

Cuando finalice, copiaremos la carpeta public generada por Gatsby en la raíz de nuestra carpeta de Docker.

CAPÍTULO 7. DESARROLLO



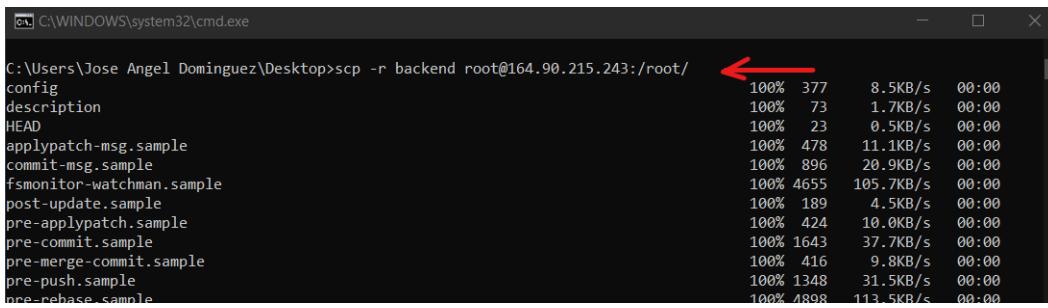
Vamos a pasar de llamar a la carpeta **Tutorial-1-Install-SSL-Certificate-using-LetsEncrypt-on-Docker-and-digital-ocean**, que es el nombre que tenía el repositorio del proyecto del que nos basamos, por el nombre **backend**. Quedándonos finalmente de la siguiente manera.



Ya tendríamos nuestra carpeta preparada para pasarl a producción. Lo primero que haremos será pasar la carpeta **backend** a nuestra máquina virtual (o Droplet) mediante ssh. Para ello, abriremos una terminal en el Escritorio (lugar en el que se encuentra nuestra carpeta backend) y ejecutaremos el siguiente comando.

```
scp -r backend root@<IP droplet>:/root/
```

7.8. DESPLIEGUE DEL PROYECTO EN DIGITALOCEAN

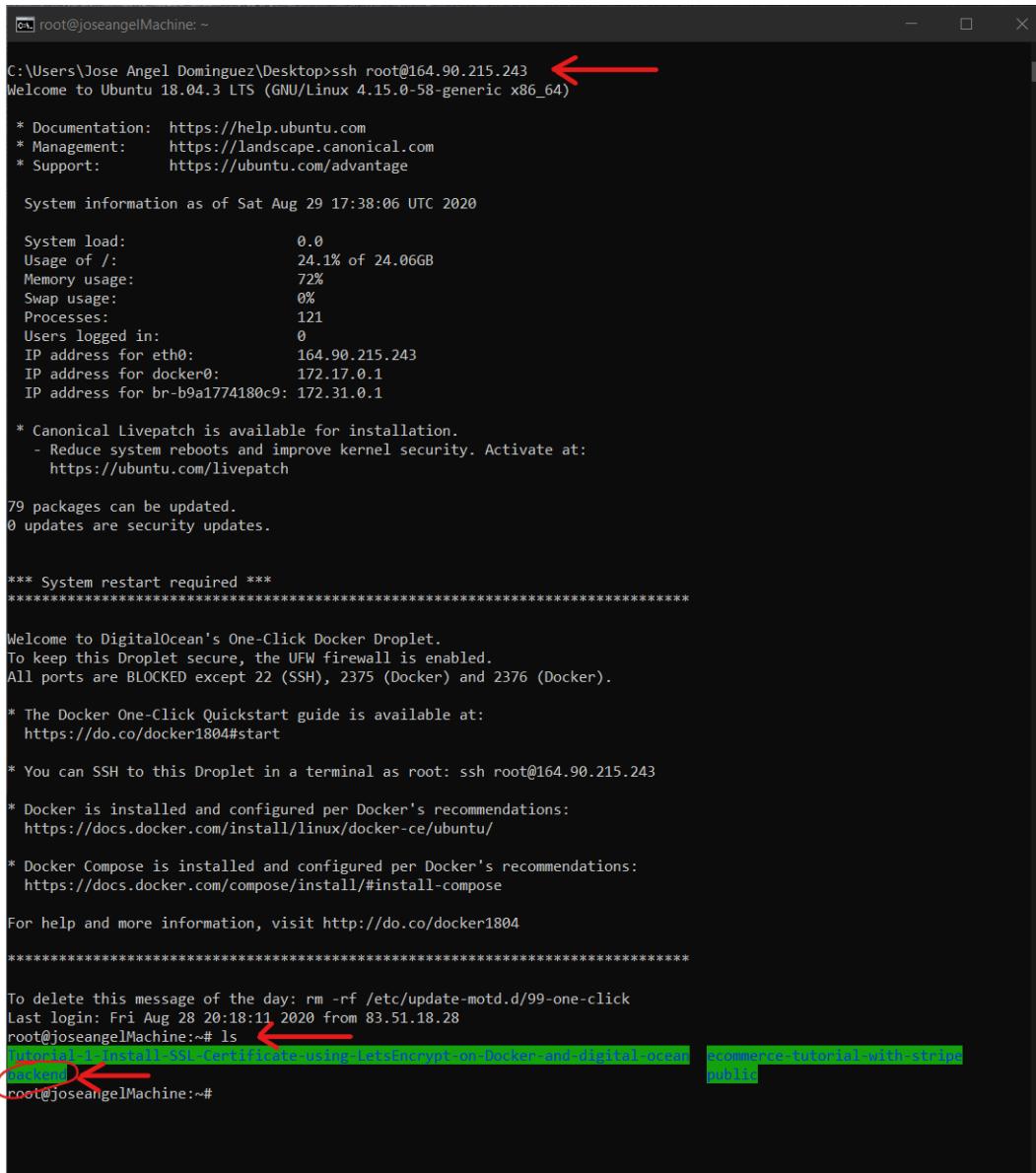


```
C:\Users\Jose Angel Dominguez\Desktop>scp -r backend root@164.90.215.243:/root/
config
description
HEAD
applypatch-msg.sample
commit-msg.sample
fsmonitor-watchman.sample
post-update.sample
pre-applypatch.sample
pre-commit.sample
pre-merge-commit.sample
pre-push.sample
pre-rebase.sample
```

Progress (%)	File	Transfer Speed	Time Left
100%	377	8.5KB/s	00:00
100%	73	1.7KB/s	00:00
100%	23	0.5KB/s	00:00
100%	478	11.1KB/s	00:00
100%	896	20.9KB/s	00:00
100%	4655	105.7KB/s	00:00
100%	189	4.5KB/s	00:00
100%	424	10.0KB/s	00:00
100%	1643	37.7KB/s	00:00
100%	416	9.8KB/s	00:00
100%	1348	31.5KB/s	00:00
100%	4898	113.5KB/s	00:00

Una vez que finalice el proceso, ejecutaremos el siguiente comando para controlar la máquina virtual por medio de ssh.

ssh root@<IP droplet>



```
root@joseangelMachine: ~
C:\Users\Jose Angel Dominguez\Desktop>ssh root@164.90.215.243
Welcome to Ubuntu 18.04.3 LTS (GNU/Linux 4.15.0-58-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

System information as of Sat Aug 29 17:38:06 UTC 2020

System load:          0.0
Usage of /:           24.1% of 24.06GB
Memory usage:         72%
Swap usage:           0%
Processes:            121
Users logged in:     0
IP address for eth0: 164.90.215.243
IP address for docker0: 172.17.0.1
IP address for br-b9a1774180c9: 172.31.0.1

* Canonical Livepatch is available for installation.
- Reduce system reboots and improve kernel security. Activate at:
  https://ubuntu.com/livepatch

79 packages can be updated.
0 updates are security updates.

*** System restart required ***
***** Welcome to DigitalOcean's One-Click Docker Droplet.
To keep this Droplet secure, the UFW firewall is enabled.
All ports are BLOCKED except 22 (SSH), 2375 (Docker) and 2376 (Docker).

* The Docker One-Click Quickstart guide is available at:
  https://do.co/docker1804#start

* You can SSH to this Droplet in a terminal as root: ssh root@164.90.215.243

* Docker is installed and configured per Docker's recommendations:
  https://docs.docker.com/install/linux/docker-ce/ubuntu/

* Docker Compose is installed and configured per Docker's recommendations:
  https://docs.docker.com/compose/install/#install-compose

For help and more information, visit http://do.co/docker1804
***** To delete this message of the day: rm -rf /etc/update-motd.d/99-one-click
Last login: Fri Aug 28 20:18:11 2020 from 83.51.18.28
root@joseangelMachine:~# ls
Tutorial 1 Install SSL Certificate using Let's Encrypt on Docker and digital ocean  ecommerce-tutorial-with-stripe
public
backend
root@joseangelMachine:~#
```

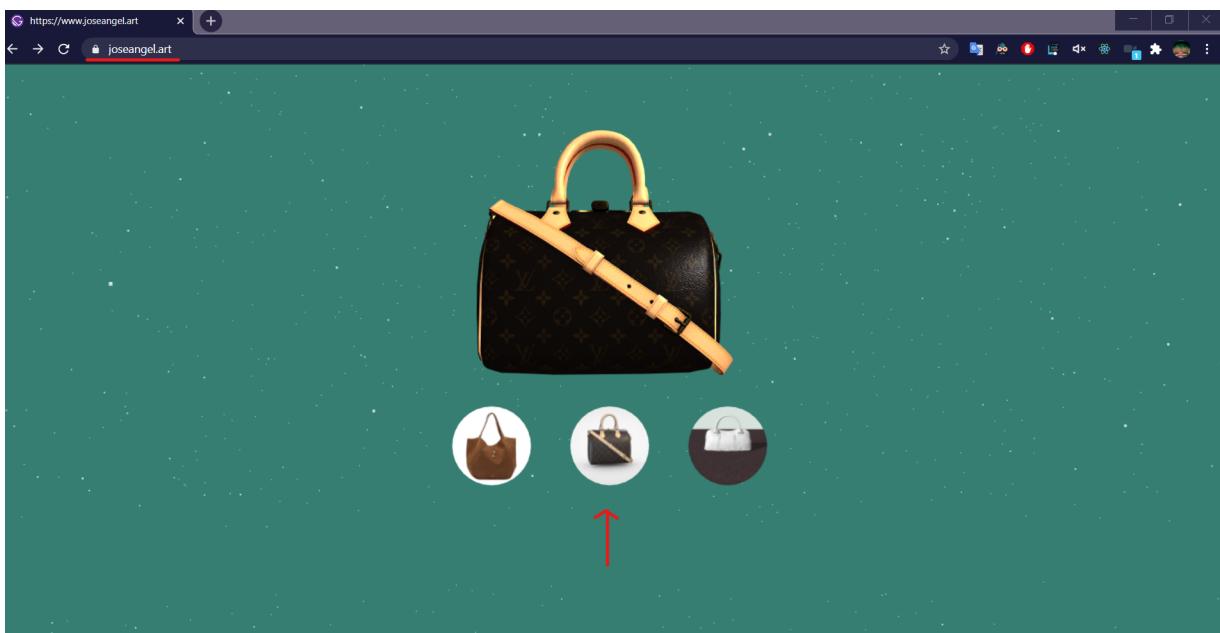
Como comprobamos en la imagen anterior, al ejecutar el comando `ls` debería aparecernos la carpeta **backend** que hemos mandado a nuestra máquina mediante ssh con el comando **scp**. Ignorar las demás carpetas que aparecen en la imagen puesto que son para hacer pruebas, no pertenecen a este tutorial. Una vez que nos aseguremos que tengamos esa carpeta, nos metemos dentro de la carpeta con el comando `cd backend` y ejecutamos el comando **docker-compose up -d**.

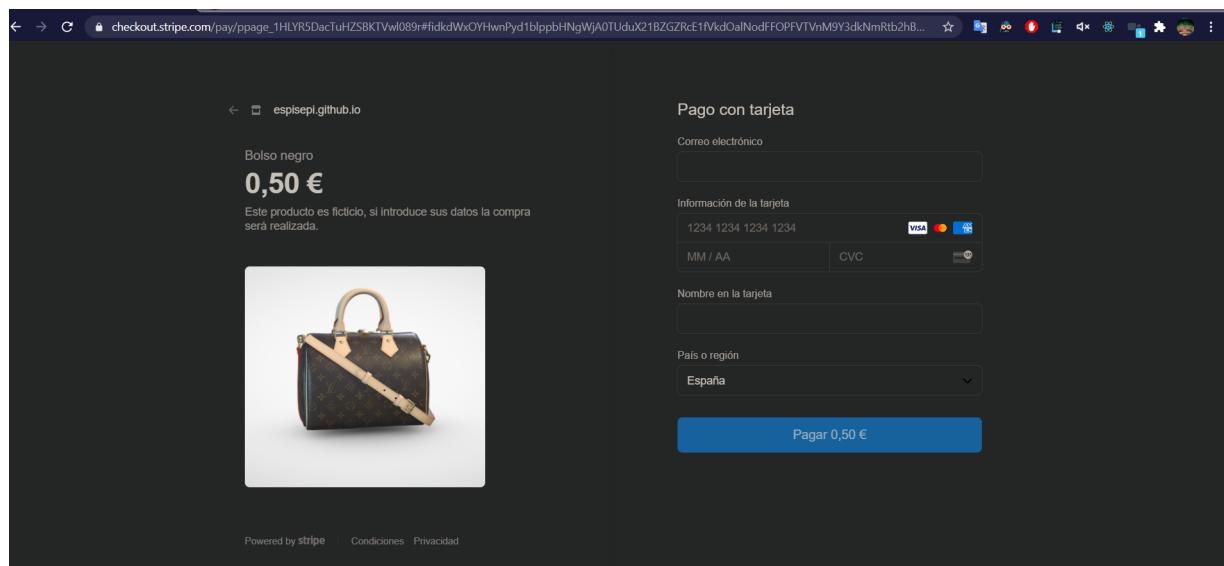
```
root@joseangelMachine:~# cd backend
root@joseangelMachine:~/backend# ls
README.md docker docker-compose.yaml public
root@joseangelMachine:~/backend# docker-compose up -d
Creating network "backend_default" with the default driver
Pulling apache (php:7-apache)...
7-apache: Pulling from library/php
bf5952930446: Pull complete
a409b57eb464: Pull complete
3192e6c84ad0: Pull complete
43553740162b: Pull complete
d8b8bba42dea: Pull complete
eb10907c0110: Pull complete
10568906f34e: Pull complete
03fe17709781: Pull complete
98171b7166c8: Pull complete
3978c2fb05b8: Pull complete
71bf21524fa8: Pull complete
24fe81782f1c: Pull complete
7a2dfd067aa5: Pull complete
Digest: sha256:797978f6894ba27ff07d85acefc8d4bbc5f6377ac65edddc0d5190963d1c12c2
Status: Downloaded newer image for php:7-apache
Pulling nginx (linuxserver/letsencrypt)...
Latest: Pulling from linuxserver/letsencrypt
aaa4765532d8: Pull complete
8814f5f5b65f: Pull complete
22f837c7f4ee: Pull complete
4b679be8fefc: Pull complete
5d80aeb62c09: Pull complete
dd54282078a1: Pull complete
d1d4c2474bc8: Pull complete
Digest: sha256:4a8f942aac0707877f79cea137d71e945e7e8655ce4b5f33f8ff96ce9c9e859a
Status: Downloaded newer image for linuxserver/letsencrypt:latest
Creating apache ... done
Creating nginx ... done
```

Una vez que finalice todo el proceso, podemos ver el logs generado por docker-compose con el comando **docker-compose logs -f** para comprobar que todo está funcionando correctamente.

7.8. DESPLIEGUE DEL PROYECTO EN DIGITALOCEAN

Si todo ha ido correcto, deberíamos poder acceder al enlace <https://joseangel.art> y ver nuestra escena 3D funcionando correctamente. Si pulsamos en cualquiera de los productos, debería llevarnos al checkout de Stripe para comprar dicho producto.





Ya tendríamos nuestro proyecto finalizado.

PARTE IV

CIERRE DEL PROYECTO

CONCLUSIONES

El objetivo de este capítulo es analizar el proceso de desarrollo que ha tenido este proyecto para aprender de los errores y mejorar evitando volver a cometerlos.

8.1 INFORME POST-MORTEM

Debido a que hemos realizado un proyecto utilizando tecnologías desconocidas hasta ahora, han surgido problemas que no han sido planificados con el suficiente margen de tiempo, sobrepasándose dicho margen y obligándonos a trabajar un número de horas extras que se han considerado como horas de trabajo no remuneradas.

A continuación se detallarán algunos de los problemas importantes que nos hemos ido encontrando durante el desarrollo del proyecto:

- Al mostrar las imágenes de Stripe en nuestra web, obteníamos un error por bloqueo CORS en la petición de imágenes al servidor Stripe. Este error ha sido solucionado utilizando una aplicación alojada en heroku como proxy para que sea la aplicación quien nos devuelva las imágenes sin el bloqueo CORS. En el apartado de desarrollo se muestra el código resultante.
- Al realizar la build para preparar el proyecto a producción, obteníamos un error por ejecutar nuestro código fuera del contexto de un navegador. Esto se producía porque utilizamos variables globales solo disponibles en los navegadores. Se ha solucionado utilizando bloques trycatch. En el apartado de desarrollo se muestra el código.
- El evento onclick solo funciona al pulsar en la pantalla con el ratón. Si queremos pulsar con los dedos en un smartphone necesitamos usar el evento onPointerDown. Dicho evento funciona tanto para ordenadores con el ratón como para smartphones y tablets.
- Necesitábamos tener el proyecto en producción con https y con un nombre de dominio para que el checkout de Stripe funcionase correctamente. Este error nos ha hecho perder mucho tiempo intentando que funcionase en local.

8.1.1 Lo que ha ido bien

- Se han cumplido con los objetivos del proyecto y se ha finalizado y entregado en la fecha acordada.
- Se han solucionado todos los problemas con los que nos hemos ido encontrando durante el desarrollo del proyecto.

8.1.2 Lo que ha ido mal

- No se ha cumplido con la planificación inicial, debido a los problemas que nos hemos ido encontrando durante el desarrollo y que no estaban planificados. Esto ha sido debido a la inexperiencia con las herramientas que estamos utilizando.
- Se ha trabajado más horas de las planificadas para llegar a cumplir con los objetivos en la fecha estipulada.
- La solución propuesta para evitar el bloqueo CORS, es una solución temporal. Puesto que no deberíamos depender de un proyecto externo, sin conocer su mantenimiento, para el correcto funcionamiento de nuestra web.

8.1.3 Discusión

Si comenzase de nuevo con el proyecto, dejaría un mayor margen de tiempo en la iteración correspondiente al desarrollo frontend. También haría una planificación que no quedase tan justa con la fecha de entrega del proyecto, para así evitar trabajar horas extras de las estipuladas durante el día, para conseguir así llegar a cumplir con los objetivos en la fecha de entrega del proyecto.

8.2 TRABAJOS FUTUROS

- Desplegar en nuestros servidores nuestra propia aplicación web proxy para evitar el bloqueo CORS, para así asegurarnos de que siempre estará funcionando o que tenemos el control de dicha aplicación por si surgiese algún problema. Para realizar esto, se utilizaría el proyecto de GitHub <https://github.com/Rob--W/cors-anywhere>.
- Mejorar la organización del código escrito y aplicar patrones de Refactoring en donde sea necesario.
- Añadir la funcionalidad de carrito de compras al proyecto.

PARTE V

BIBLIOGRAFÍA

BIBLIOGRAFIA

- Historia del comercio electrónico. URL https://es.wikipedia.org/wiki/Comercio_electrónico
- Historia del comercio electrónico en España. URL <https://marketing4ecommerce.net/historia-del-ecommerce-en-espana/>
- Metodología de desarrollo SCRUM. URL [https://es.wikipedia.org/wiki/Scrum_\(desarrollo_de_software\)](https://es.wikipedia.org/wiki/Scrum_(desarrollo_de_software))
- BOE. Ministerio de trabajo y economía social. URL <https://www.boe.es/boe/dias/2020/05/13/pdfs/BOE-A-2020-5006.pdf>
- Tutorial para realizar un e-commerce con el framework Gatsby. URL <https://www.gatsbyjs.com/tutorial/ecommerce-tutorial/>. Repositorio del tutorial <https://github.com/gatsbyjs/gatsby/tree/master/examples/ecommerce-tutorial-with-stripe>
- Repositorio del proyecto que utilizamos como proxy para evitar el bloqueo CORS. URL <https://github.com/Rob--W/cors-anywhere>
- Canal de Youtube EDteam, que nos ha ayudado a entender los conceptos de React. URL [¿Qué es React.js y cómo funciona?](#). También los conceptos de GraphQL. URL [¿Qué es GraphQL?](#)
- Canal de Youtube Bluuweb, que nos ha ayudado a entender los conceptos de React hooks. URL [React Hooks - Curso para Principiantes \[Fundamentos Reactjs \]](#)
- Canal de Youtube midudev, que nos ha ayudado a afianzar conceptos en React y ver ejemplos de uso de GraphQL. URL [Queries y GraphiQL con la API de Rick Morty \[Curso express GraphQL\]](#)
- Canal de Youtube AyyazTech que nos ha ayudado en la elaboración de los ficheros para Docker. URL [Install Let's Encrypt SSL Certificate using Docker, Wordpress and DigitalOcean | Web Development](#)

- Documentación oficial de la librería Three.js. URL <https://threejs.org/docs/>
- Conjunto de artículos para aprender los conceptos de Three.js. URL <https://threejsfundamentals.org/>
- Repositorio oficial de la librería react-three-fiber. URL <https://github.com/react-spring/react-three-fiber>
- Artículos para aprender react-three-fiber. URL <https://codeworkshop.dev/>
- Manual para aprender el lenguaje GLSL. URL <https://thebookofshaders.com/>
- Modelo 3D utilizado en el proyecto. Autor: Alvaro Sanint. URL <https://sketchfab.com/3d-models/bag-pbr-da3d686b38c141e1bf54f9df6ca39313>