

Proyecto DAW

2º Año – 3º Trimestre

(Junio 2020)

Ana Alexandra Morais Vingada
Patricia Martínez Espert

Grado Superior en Desarrollo de Aplicaciones Web (DAW)

2018 – 2020

INDICE

1. Componentes del Equipo y aportación	3
2. Coordinación del Proyecto y Equipo	3
3. Etapas del Proyecto	4
4. Tecnologías y lenguajes utilizados	4
5. Herramientas y programas utilizados	5
6. Presentación del Proyecto	5
6.1. Descripción del Proyecto	5
6.2. Objetivos de la página web	7
6.3. Objetivos del proyecto	7
6.4. Público objetivo / Target Audience	7
6.5. Ámbito geográfico del website	11
7. Diseño - Conceptos, Estructuras y Sitemap	11
7.1. Identificación de conceptos	11
7.2. Asignación de conceptos	13
7.3. Identificación de secciones/estructuras	15
7.4. Elaboración del SiteMap	17
7.5. Mapas conceptuales	18
7.6. Wireframes	18
7.7. Prototipos	19
7.8. Guía de estilo	20
7.9. Logotipo	20
8. Front-End	21
9. Back-End	24
9.1. Estructura	24
9.2. Rutas algo especiales I (CRUD)	25
9.3. Rutas algo especiales II (el reto)	27
9.4. Más rutas	39
9.5. Base de datos	39
9.6. MODELO E-R	41
9.7. Otros puntos	42
10. Conclusiones / Consideraciones Finales	43
11. Bibliografía y Recursos	43

1. Componentes del Equipo y aportación

Ana Vingada

- Diseño, UX, UI y Usabilidad
- Selección y gestión de contenidos
- Desarrollo front-end
- Apoyo en desarrollo back-end

Patricia Martínez

- Diseño
- Desarrollo back-end
- Apoyo en desarrollo front-end
- Organización de la aplicación y despliegue

Hemos distribuido tareas de acuerdo con nuestros gustos/preferencias, formación y experiencia.

2. Coordinación del Proyecto y Equipo

En primero lugar, hemos decidido hacer el proyecto en grupo porque, una vez trabajando como desarrolladoras, tendremos mayoritariamente que trabajar en equipo, en que aptitudes como la comunicación y coordinación son indispensables.

Para que todo funcionase de la mejor forma, hemos discutido en detalle el proyecto, además de distribuir tareas. Cada una se enfocó en lo que más le gusta: back y front. En proyectos reales normalmente los perfiles están diferenciados, así que consideramos nuestra distribución un punto importante.

Hemos creado una carpeta en Google Drive para compartir información del proyecto, de que es el caso de toda la información en el presente documento.

Para el proyecto de desarrollo, inicialmente hemos subido al github nuestro trabajado de forma separada, o sea, en dos proyectos diferentes. Lo decidimos porque al inicio Patri estaba probando las diferentes acciones de la web con back-end en una aplicación muy básica, mientras que Ana estaba desarrollando la aplicación a nivel visual y funcional (Javascript).

A la medida que fuimos desarrollando nuestros proyectos, hemos cambiado algunos de los detalles del diseño, por varios motivos: desde funcionales y organizativos a usabilidad y apariencia.

Terminadas las pruebas de back-end y front end, hemos emergido los proyectos, acertando detalles y haciendo las tareas de conexión con front y back end. Podemos utilizar como ejemplo la definición de una acción del formulario y establecimiento de conexión con la base de datos.

A lo largo del planteamiento, diseño y desarrollo, hemos comunicado y coordinado nuestro trabajo, apoyándonos también en documentos explicativos detallados y vídeos.

3. Etapas del Proyecto

- Planteamiento
 - Creación de carpeta de proyecto en Google drive
 - Creación de repositorio en GitHub
 - Definición del Proyecto
 - Discusión de ideas, herramientas, lenguajes de programación
 - Definición de tiempos, organización de equipo y tareas
- Diseño
 - conceptos y estructuras
 - mapas conceptuales
 - Layout / wireframes
 - Guía de estilo
- Desarrollo front-end y back-end
- Pruebas y revisión

4. Tecnologías y lenguajes utilizados

- HTML
- CSS
- Bootstrap
- Json
- Javascript
- JQuery

- SQL
- Python / Framework Flask
- Docker

5. Herramientas y programas utilizados

- Adobe XD
- Gloomaps
- Flaticon
- Visual Studio Code
- Pycharm
- Docker
- GoogleDrive
- Nginx
- GitHub
- SourceTree
- Fontawesome

6. Presentación del Proyecto

Para el proyecto práctico de la FP de Grado Superior de Desarrollo de Aplicaciones Web, hemos desarrollado una página web para el negocio “El Horno Dulce - Panadería y Repostería”.

Estando disponible online, el website tendría como dirección www.elhornodulce.es y sería una tienda online de encomienda de productos de panadería y repostería. La entrega sería gestionada en persona y de acuerdo con el pedido efectuado por el cliente a través de la web.

6.1. Descripción del Proyecto

Desde el inicio del proyecto que definimos que íbamos a intentar hacer una página web próxima a un caso real. Es un detalle que consideramos indispensable: nos sirve como experiencia y ejemplo para nuestra profesión como desarrolladores; y porque así podemos intentar colocarnos en la posición del cliente y ser autocríticas.

En una fase inicial, hemos hablado con algunas personas, de diferentes edades y experiencias y, a través de sus necesidades, hemos destacado 3 opciones de proyecto reales, sencillas y prácticas:

- Una página web para presentación de una abogada, para presencia online y medio de contacto.
- Una página de arquitectos como portfolio online, presentación profesional y medio de contacto.
- Una página web como tienda de repostería online para negocio local.

Discutiendo y analizando cada idea, hemos optado por desarrollar la página web para tienda de repostería online. Las razones son las siguientes:

- A nivel de back-end es muy interesante gestionar desde una base de datos los clientes y sus pedidos; además, gestionar toda la información de producto para presentar al cliente.
- A nivel de front-end es una oportunidad para gestionar y presentar de una forma atractiva la información para captación de clientes, y también de trabajar el SEO y usabilidad.
- Es un caso basado en una compañera de Ana que perdió a su trabajo y empezó a hacer repostería por encomienda y a entregar en mano sus productos. Para facilitar el trabajo, es una idea interesante, desarrollar una página para que el cliente pueda hacer su encomienda de una forma más fácil.
- Es una web versátil, en el sentido que, no solo presentará a sus productos, como también presentará a la persona responsable por la tienda, “el autor”. O sea, será una página web de promoción personal y de producto.

A pesar de esta página web estar basada en un caso real, el nombre del negocio, los productos y todos los contenidos son de nuestra propia creación y autoría.

Así, podemos decir que el negocio se llama “El Horno Dulce - Panadería y Repostería” y que es gestionada por una señora llamada Rosa Silva. Esta señora vive en Madrid y hace todos sus productos de panadería y repostería en su casa, con su equipo.

A través de su página web, se presenta a su tienda, además de sus productos (por ejemplo, pastel de chocolate, palmeras de chocolate blanco, negro o de azúcar, etc.). Dispone además de información sobre alergénicos y la posibilidad de encomiendas especiales por parte de los clientes (por ejemplo, encomienda de un bizcocho sin lactosa, o un pastel de cumpleaños con forma de pelota, etc.).

6.2. Objetivos de la página web

- Presentar la marca online
- Vender productos de panadería y repostería online
- Captar nuevos clientes
- Fidelizar clientes
- Ganar visibilidad online como tienda de repostería
- Conseguir reconocimiento como marca de confianza y calidad
- Poner a la disposición un medio de contacto con la marca y el usuario

6.3. Objetivos del proyecto

- Aprender a desarrollar un proyecto web completo
- Poner en prácticas los conocimientos adquiridos durante la formación profesional en todas las disciplinas
- Aprender nuevas tecnologías tanto en front como en back end
- Entender y aplicar todas las fases de desarrollo
- Conseguir la información necesaria y gestionarla de una forma organizada y práctica
- Registrar todos los pasos de desarrollo de proyecto de forma organizada, visual y completa
- Conseguir desarrollar una página web user-friendly y accesible
- Conseguir desarrollar un buen proyecto, funcional y bien documentado

6.4. Público objetivo / Target Audience

El proyecto apunta para la comunidad online en general como target audience, porque los contenidos estarán accesibles para todos los que tengan acceso a internet.

Sin embargo, el usuario tipo general del website será hombre o mujer, entre los 15 y los 70 años de edad, que vive en la comunidad de Madrid, España.

Analizamos el usuario tipo en más detalle:

- A) Sexo
 - a. Hombre
 - b. Mujer (mayoritario)

B) Edad por rangos

- a. 15 – 17
- b. 18 – 29
- c. 30 - 45 (mayoritario)
- d. 46 – 59
- e. 60 – 70

C) Entorno socio-económico

- a. Educación
 - i. Escolaridad obligatoria
 - ii. FP
 - iii. Grado universitario (mayoritario)
- b. Ocupación
 - i. empleado (mayoritario)
 - ii. desempleado

D) Entorno geográfico

- a. Regional
 - i. Madrid
 - ii. Aranjuez
 - iii. Alcorcón
 - iv. Arganda del Rey
 - v. Arroyomolinos
 - vi. Boadilla del Monte
 - vii. Collado Villalba
 - viii. Coslada
 - ix. Fuenlabrada
 - x. Getafe
 - xi. La Rozas
 - xii. Leganés
 - xiii. Majadahonda
 - xiv. Móstoles
 - xv. Pinto
 - xvi. Pozuelo de Alarcón
 - xvii. Rivas-Vaciamadrid
 - xviii. Alcalá de Henares
 - xix. San Sebastián de los Reyes
 - xx. Alcobendas
 - xxi. Torrejón de Ardoz

- xxii. Tres Cantos
- xxiii. Valdemoro
- xxiv. Otras ciudades

- b. Tipo
 - i. Urbano (mayoritario)
 - ii. Rural
- c. Tamaño
 - i. Grandes ciudades (más probable)
 - ii. Medias ciudades (más probable)
 - iii. Pequeñas ciudades
 - iv. Otros entornos más pequeños

E) Lifestyle

- a. Usuarios de internet
- b. Foodies
- c. Personas con poco tiempo para cocinar
- d. Personas malas en la cocina
- e. Personas a que les gusta celebrar
- f. Personas a que les gusta la comida
- g. otros

F) Producto / servicio que el usuario utiliza o busca

- a. Dulce
- b. postres
- c. pastelería
- d. repostería
- e. panadería
- f. información

G) Momento de uso

- a. Planeado
- b. Espontáneo
- c. De emergencia

H) Tipo de visita a la web

- a. Primera vez
- b. Repite la visita
- c. Subscriptor

I) Donde visita la web

- a. En casa
- b. Local público

- c. otros
- J) Beneficio que el usuario busca
 - a. buen producto
 - b. originalidad creatividad
 - c. calidad
 - d. sabor
 - e. probar cosas nuevas
 - f. información útil
 - g. buen servicio
- K) Perspectivas de la visita a la web
 - a. Probar / conocer algo nuevo
 - b. conseguir una información
 - c. formar una opinión
 - d. conocer la marca
 - e. saber cómo funciona el servicio
 - f. conocer los productos
- L) Preferencias del usuario
 - a. marca respetable y de confianza
 - b. servicio rápido y eficaz
 - c. calidad del producto
 - d. precios accesibles
 - e. buena relación calidad-precio

Teniendo todas estas características en consideración, podemos priorizar el siguiente target audience como el usuario tipo del website:

- Mujeres
- Edades entre los 30 y 55 años
- Ciudad de Madrid y ciudades más próximas
- Usuarios diarios de internet
- Usuarios acostumbrados en hacer compras online
- Usuarios a que les gusta la panadería y repostería

6.5. Ámbito geográfico del website

Como es una tienda de repostería, y la entrega es en mano, el ámbito geográfico de la página web y del negocio es local, o sea, en la comunidad de Madrid, para que el servicio de entrega sea rápido y eficaz.

7. Diseño - Conceptos, Estructuras y Sitemap

7.1. Identificación de conceptos

En este punto identificamos conceptos que vamos a insertar en la página web y su propósito.

Conceptos	Propósito
homepage	Página principal, de presentación y primer contacto
Sobre Nosotros	Página de presentación del negocio, que hacen, la misión
Productos	Página de presentación de los productos, organización por categorías
Contacto	Página de contactos, formulario, email, teléfono, FAQs, redes sociales
FAQS	página con preguntas útiles sobre la venta, compra, entregas, etc.
Búsqueda de productos	Búsqueda de productos por palabras clave
Política de Cookies	Información sobre la política de cookies E implementación de las mismas.
Política de Privacidad	Página de información sobre privacidad y derechos del usuario y del propietario
Carrito para clientes	Visualizar el carrito y hacer determinadas operaciones como añadir, borrar, cambiar
Login para cliente	Login del usuario para gestionar sus pedidos
Registro para clientes	Página de registro de un nuevo usuario
Gestión de pedidos y entregas	Página de gestión de pedidos y entregas, que se puede actualizar para cada usuario, si está activo, entregue, etc
gestión de mensajes	Página de gestión de mensajes, para contestar, marcar como leído, etc.
Quien somos	información sobre la marca y negocio

Misión	misión del negocio, objetivos
Como surgió	cómo surgió la idea de negocio y como llegamos hasta aquí
Donde estamos	donde están ubicados
Que hacemos	que hace la marca/negocio
Como lo hacemos	Como funciona el negocio, procedimientos generales
categorías de productos	separar los productos por categorías; ejemplo: bizcochos, galletas, pasteles, tartas, etc.
nombre de producto	identificación del producto
ingredientes del producto	descripción de que lleva el producto
precio del producto	definir el precio de cada producto
fotografías del producto	fotografías del producto para presentación
formulario de mensaje	formulario con validación para contacto por mensaje
email	email de contacto
teléfono	teléfono de contacto
FAQs	Preguntas frecuentes de apoyo al usuario sobre el negocio o otro procedimiento
Redes sociales	Punto de contacto con el negocio
aceptación de cookies	ventana para aceptación de cookies por parte del usuario
copyright	detalles de copyright de la web - su registro
menú	Sección conexión principal con otras páginas del website
footer	quick links, copyright, redes sociales
información de pago	métodos de paga aceptes
información de pedidos	información con fecha de pedido, persona, dirección, fecha de entrega, productos, etc
información de entregas	información de día de entrega, método de entrega, detalles, etc.
añadir producto	añadir producto
borrar producto	borrar producto
cambiar producto	cambiar producto
pedidos especiales	posibilidad de hacer pedidos especiales de productos que no están en el listado, o con ingredientes especiales
información de alérgenos e intolerancias	información tipo infografía sobre alérgenos e intolerancias.
perfil usuario	información del usuario

quick links	links rápidos de acceso a información pertinente
logo	logotipo de identificación de la mar
nombre de la marca	nombre oficial de la marca.
Sitemap	página estructurada del website para ayudar el usuario a navegar

7.2. Asignación de conceptos

En este punto vamos a asignar a cada concepto un local/parte/ubicación en el website.

Conceptos	Ubicación/ubicaciones
homepage	página; dominio principal
Sobre Nosotros	página; link en menú
Productos	página; link en menú
Contacto	página; link en menú
FAQS	página; link en contacto; link en sobre nosotros; link en quick links
Búsqueda de productos	sección en página de productos
Política de Cookies	página; link en quick links; link en sobre nosotros
Política de Privacidad	página; link en quick links; link en sobre nosotros
Carrito para clientes	página; sección en el menú; link en el perfil de cliente
Login para cliente	página; link en menú;
Registro para clientes	página; link en menú;
Gestión de pedidos y entregas	página; acceso restringido a admin
gestión de mensajes	página; acceso restringido a admin
Quien somos	sección en página de sobre nosotros
Misión	sección en página de sobre nosotros
Como surgió	sección en página de sobre nosotros
Donde estamos	sección en página de sobre nosotros; FAQS
Que hacemos	sección en página de sobre nosotros; FAQS
Como lo hacemos	sección en página de sobre nosotros; FAQS
categorías de productos	en la página de productos

nombre de producto	en cada sección de producto
ingredientes del producto	en cada sección de producto
precio del producto	en cada sección de producto
fotografías del producto	en cada sección de producto
formulario de mensaje	sección en página de contacto
email	sección en página de contacto
teléfono	sección en página de contacto
FAQ	sección en página de FAQS
Redes sociales	sección en footer y página de contacto
aceptación de cookies	popup en la homepage o landing page (página de entrada del usuario)
copyright	footer
menú	en todas las páginas
footer	en todas las páginas
información de pago	footer; FAQS, página sobre nosotros
información de pedidos	FAQS, Página sobre nosotros
información de entregas	FAQS, Página sobre nosotros
añadir producto	en página de productos; carrito
borrar producto	en página de productos; carrito
cambiar producto	en página de productos; carrito
pedidos especiales	FAQS, página sobre nosotros; página de información
información de alérgenos e intolerancias	link en footer; página
perfil usuario	página de perfil del usuario; link en icono en header
quick links	footer
logo	header; pagicon;
nombre de la marca	header; página sobre nosotros; copyright
Sitemap	link in footer; FAQS;

7.3. Identificación de secciones/estructuras

En este punto identificamos con base en lo anterior, las estructuras y/o secciones del website (páginas, secciones de cada página, ...).

Páginas	Secciones de cada página
Homepage	menú / header
	banner
	presentación
	cajas con secciones de conexión a otras partes del website
	footer
Sobre nosotros	menú / header
	imagen
	Quien somos
	Como surgió
	Donde estamos
	Que hacemos
	Como lo hacemos
	información de pago
	footer
Productos	menú / header
	búsqueda
	cada producto con nombre, ingredientes, precio y fotografías añadir o cambiar cantidad del producto
	pedidos especiales
	categorías de productos
Contacto	menú / header
	imagen
	Introducción
	formulario de contacto
	email
	teléfono

	FAQs
	Redes sociales
	footer
Política de Cookies	menú / header
	Texto
	footer
Política de privacidad	menú / header
	Texto
	footer
información alérgenos e intolerancias	menú / header
	iconos + texto
	pedidos especiales
	footer
Sitemap	menú / header
	texto + links con estructura del website
	footer
FAQS	menú / header
	preguntas y respuestas separadas por temas y links pertinentes
	footer
Login	menú / header
	campos para insertar datos de sesión
	footer
Registro	menú / header
	campos para insertar datos de registro de usuario y sesión
	footer
Carrito	menú / header
	productos en carrito
	footer
perfil usuario	menú / header
	información del usuario

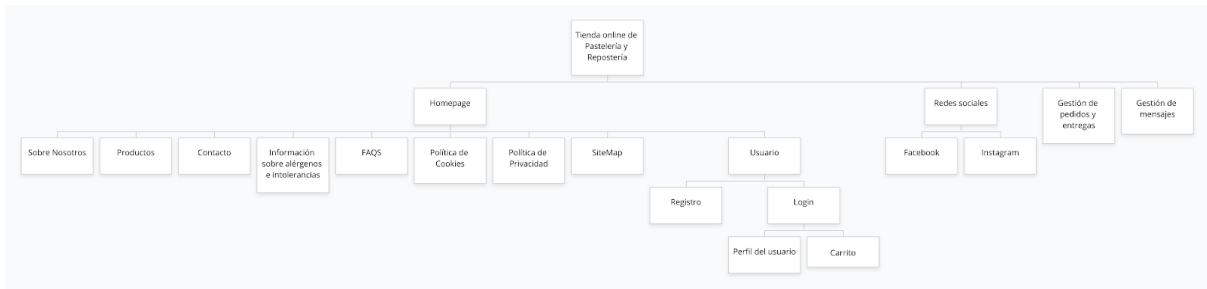
	footer
Gestión de pedidos	tabla de pedidos y toda su información en base de datos
Gestión de mensajes	Tabla de mensajes con su información; nombre, email, contacto, mensaje.
Página de error	menú / header
	texto con información de error y link de vuelta a la homepage
	footer
Menú / header	logo
	links para páginas principales: productos, sobre nosotros y contacto
	links para perfil de usuario , login o registro
Footer	quick links: política de cookies, política de privacidad, sitemap, faqs, información alérgenos y intolerancias, productos
	redes sociales: instagram y facebook
	contacto email and teléfono
	copyright
	métodos de pago

7.4. Elaboración del SiteMap

En este punto identificamos conceptos que vamos a insertar en la página web y su propósito.

- Homepage
 - Sobre nosotros
 - Productos
 - Contacto
 - Información sobre alérgenos e intolerancias
 - FAQs
 - Política de Cookies
 - Política de Privacidad
 - SiteMap
 - Usuario
 - Registro
 - Login

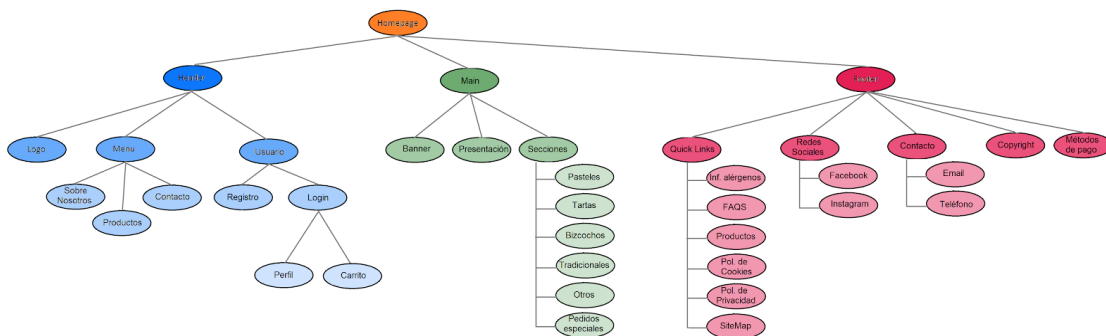
- Perfil usuario
- Carrito
- Redes Sociales
 - Facebook
 - Instagram
- Gestión de pedidos y entregas
- Gestión de mensajes



7.5. Mapas conceptuales

Utilizamos como ejemplo el mapa conceptual de la homepage.

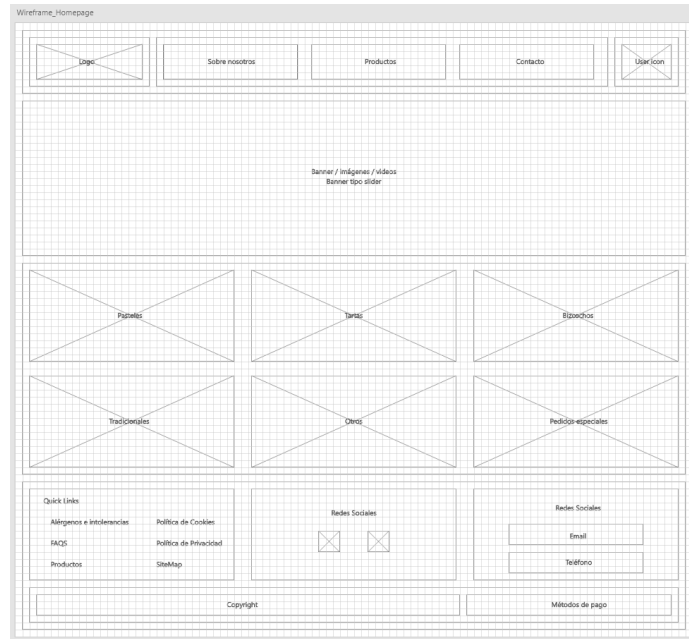
Los mapas conceptuales de cada página se encuentran en la carpeta “mapas_conceptuales”.



7.6. Wireframes

Utilizamos como ejemplo el wireframe de la homepage.

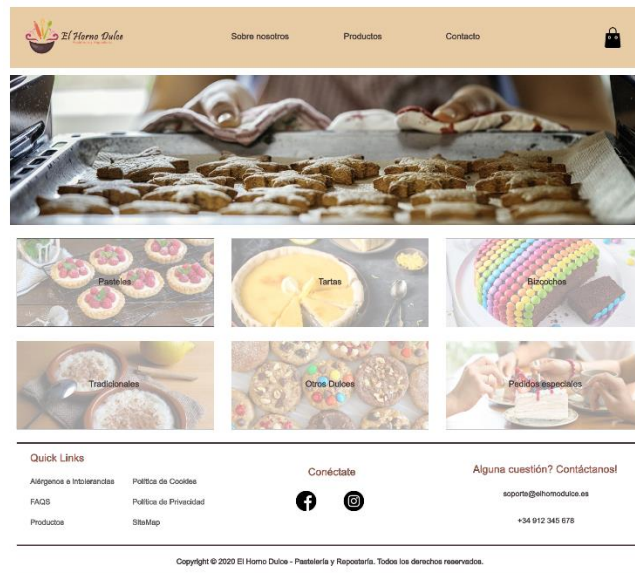
Los wireframes se encuentran en la carpeta “wireframes”.



7.7. Prototipos

Utilizamos como ejemplo el prototipo de la homepage.

Los prototipos se encuentran en la carpeta “prototipos”.



7.8. Guía de estilo

En tipología, la página utiliza Arial, sans serif. El Logotipo está diseñado con Pristina. Para el diseño de icono, se utilizó Flaticon. Para aplicación de icono en la página web, se utilizó Fontawesome.

Las imágenes están recopiladas de Google.

La paleta de colores utilizada es la siguiente:

- Para textos

 HEX #432E2F RGB 67, 46, 47 HSL 357, 31%, 22%

 HEX #644A4F RGB 100, 74, 79 HSL 348, 26%, 34%

- Para títulos, enlaces, textos destacados

 HEX #D15A43 RGB 209, 90, 67 HSL 10, 68%, 54%

- Para fondos y otros contenidos

 HEX #E7CCA6 RGB 231, 204, 166 HSL 35, 28%, 78%

- Colores básicos

 HEX #000000 RGB 0, 0, 0 HSL 0, 0%, 0%

HEX #FFFFFF RGB 255, 255, 255 HSL 0, 0%, 100%

7.9. Logotipo

El logotipo está disponible en .psd, .png y .jpg. Estos formatos, permiten hacer cambios de tamaño, diseño, color, etc.

Además, el logotipo está creado de 3 formas para que se pueda adaptar a los contenidos: solo el icono, en horizontal y en vertical.



8. Front-End

El front-end estuvo a cargo de Ana. Decidí trabajar con Visual Code e investigar cómo podía organizar las carpetas.

Una vez completada esta primera fase he creado un HTML base, dando especial importancia al contenido de la etiqueta <head> dónde están los enlaces del Favicon, Fontawesome, JQuery, Bootstrap, CSS, y otros enlaces.

Todas las páginas están montadas con HTML, CSS, Javascript (cuando necesario), JQuery y Bootstrap. Además, las páginas tienen un Favicon y trabajan con Fontawesome para mostrar iconos. Todas las páginas son responsive.

Con el HTML base, me dediqué a desarrollar el header y el footer. Estos son muy importantes en la medida en que están incorporados en todas las otras páginas. Como en el inicio no teníamos front conectado con back, he conseguido incorporar el header y el footer a través de Ajax y con la extensión de Chrome "[Web Server for Chrome](#)", que simula un servidor. Más tarde, Patri consiguió incorporar el header y el footer con Ajax y con Flask.

El footer es muy sencillo, pero el header fue montado con un navbar de Bootstrap y con una condición de back end que, en función del usuario estar log-in o log-out, se muestra un menú distinto (cuando se pincha en el icono del usuario).

En seguida he pasado a desarrollar las páginas más sencillas de la web. Me refiero a páginas que no iban a necesitar mucho de back, javascript, validación o ficheros externos. Esas son las páginas de Sitemap, Política de Cookies, Política de Privacidad, Sobre Nosotros, Alérgenos e intolerancias, y la homepage.

El Sitemap contiene enlaces para las otras páginas del website. Las páginas Sobre Nosotros y Alérgenos e intolerancias contienen apenas imágenes, texto y enlaces de interés. Ya la Política de Cookies y la Política de Privacidad son esencialmente texto retirado de ejemplo de un website externo (mencionado en la bibliografía). La homepage es la página de presentación de la web con un carrusel montado en Bootstrap y enlaces para otras páginas del website.

En seguida, pasé a desarrollar las páginas de Login, Registro, Pedido especial y Registro. La particularidad de estas páginas es su formulario. Primero fue creado el HTML y diseño responsive y después fue trabajada la validación de los formularios.

Los formularios de Login y Registro, además de la validación de campos (por ejemplo, nombre, email, password), tienen una función que permite al usuario ver su password.

Los formularios de Pedido especial y Contacto pedían al inicio datos como el nombre, email o teléfono, pero, cuando conectamos front con back, estos campos fueron eliminados. Para enviar pedidos especiales o contactar, el usuario tiene que estar log-in. El back se encarga de enviar a la base de datos la información de usuario.

Para evitar el envío de formulario con datos incorrectos, he montado en Javascript la validación de los formularios en front. Si el formulario tiene campos inválidos, entonces no se puede enviar. Solo si son todos los campos válidos, se puede enviar el formulario y enviar a back.

Ida he pasado a desarrollar la página del perfil de usuario que imprime en pantalla los datos del usuario en sesión. Además, la página permite actualizar el perfil, cerrar sesión o borrar la cuenta. Todo este fue posible con back.

En seguida, y como preparación para páginas más complejas, empecé a trabajar en la página de FAQs. Buscaba una manera de reproducir el mismo código para cada Pregunta Frecuente, sin tener que hacerlo manualmente.

Investigando, identifiqué que JSON era el método recomendado. Así, empecé por crear el HTML y CSS con apenas una pregunta frecuente para test.

Sabía que las FAQs tenían que ser enviadas a front desde back, pero, como todavía no estaban conectados he creado un fichero con todas las preguntas y respuestas en formato JSON simulando tal como front las recibiría desde back.

En Javascript, utilicé un fetch para ir a buscar la información del fichero de FAQs y, a través de un for, he recogido cada pregunta y respuesta, mostrándolas en pantalla. Más tarde,

insertó los datos en la base de datos y trató de enviar desde back la información a back tal como habíamos testado en el fichero JSON.

Comprobando que todo funcionaba, he pasado a la página de productos. Esta tiene un menú que muestra sus resultados en función de la categoría que el usuario elije (back). Para mostrar los productos en pantalla, he creado un fichero con todos los productos y sus características en formato JSON, una vez más simulando la manera como recibiría la información desde la base de datos. A través de un for, fui mostrando los productos en pantalla. Además, he añadido paginación y habilitado el añadir un producto al carrito (esta último se relaciona con la página y con el javascript de carrito).

En seguida pasé a la página de carrito, que se hace casi por completo en Javascript. Guardando la información en un array de sesión, el carrito muestra en pantalla todos los productos en que el usuario pinchó en la página de productos (en el icono de carrito de cada producto). Ese array se guarda por sesión, o sea, hasta que el usuario cierre la pestaña de la página web.

Además, se habilita la posibilidad de vaciar por completo el carrito o de continuar comprando, redirigiendo el usuario a la página de productos. El botón de finalizar encomienda requiere conexión con back end, guardando la información del array como pedido en la base de datos.

En seguida pasé a la página de historial del usuario, donde he utilizado el mismo proceso que en la página de FAQs. He creado un fichero con datos de encomiendas ficticias y con un for, recojo todos los pedidos realizados por el usuario mostrándolos en pantalla. Más tarde, Patri hizo la conexión con la base de datos.

Para terminar, he creado las cookies que son unicamente un fichero Javascript que está conectado con todas las páginas del website. Si el usuario visita por primera vez el website, aparece un banner en la primera página que visita (configurado en HTML en el fichero de Javascript) con el mensaje de cookies. Si navegamos por otras páginas o si incluso salimos y volvemos a la página ya no nos aparece nada porque tenemos la cookie guardada en nuestro navegador.

Después de conectar back con front y todo estar funcionando, hemos pasado a revisar todo el proyecto, haciendo cambios donde necesario. Una de las cosas que fue implementada en esta última fase fue el banner de mensajes de “error” o “success” originadas por back. Por ejemplo: cuando se hace login, aparece un mensaje de back en pantalla (pero personalizado por front) diciendo que el login se realizó correctamente; si un usuario se intenta registrar con el mismo email, aparece un mensaje de error de back con personalización de front diciendo que no se puede efectuar el registro porque el email ya existe.

9. Back-End

Yo soy Patricia y me he encargado del Backend, lo cual ha supuesto un reto dado que apenas había tocado flask (el micro-framework backend que he utilizado) y python (el lenguaje de programación en el que este ha sido estructurado).



Me encantan los retos y creo que ser programador es un reto en sí, así que sin más dilación me metí “manos al código”.

Comencé por desarrollar una estructura simple a modo de esqueleto (y la idea ha sido ir generando todo ese cuerpo del back a partir del esqueleto); este apenas contaba (en la carpeta templates) con 4 vistas (base, index, login y signup) y un solo fichero de estilos (style.css). Una vez que tenía (para ser sinceros, más o menos) la conjunción de funcionalidades que podía realizar con este framework (y una vez que mi compañera me adjuntó sus vistas) me aventuré a la ardua tarea de enlazar backend y frontend (finalmente con éxito).

Ahora sí que sí comienza esta guía:

9.1. Estructura

El ‘alma mater’ de la aplicación es el archivo **routes** en el que se encuentra todo el esquema de enrutado. Podemos encontrar rutas como:

- /index o / → nos direcciona al homepage
- /about → nos direcciona a sobre-nosotros
- /products → nos direcciona a productos
- /contact → nos direcciona a contacto
- /carrito → nos direcciona a carrito
- /politica → nos direcciona a politica

- /site → nos direcciona a sitemap
- /historial → nos direcciona a historial-usuario
- /faqs → nos direcciona a faqs
- /alergenosenos → nos direcciona a alergenosenos
- /pedido-especial nos direcciona a pedido-especial
- /footer → nos muestra el footer (implementado en todas las páginas)

9.2. Rutas algo especiales I (CRUD)

- /signup

Acepta tanto los métodos GET como POST, la función login (def signup():) se ejecuta de la siguiente manera:

Si el método HTTP que empleamos es POST (el más adecuado para envío de información a través de formularios (nuestro caso)) entonces recogemos el objeto usuario mediante una 'query' lanzada con el módulo de flask flask_sqlalchemy. Para la contraseña he empleado la variable hashed_pw que equivale al resultado de emplear la función generate_password_hash sobre la contraseña capturada en el formulario de registro y el método de encriptación sha256. Pueden ocurrir 3 cosas:

- Que no exista usuario (--> nos registramos) generamos el usuario, commit a la tabla users y redirección a login
- Que exista el usuario pero que su cuenta esté inactiva (--> se reactiva la cuenta), rellenan todos los campos capturados del formulario y se pone la cuenta en activo (en la tabla users aparecerá con un 1 (true))
- Se usa un mail que ya ha sido registrado (--> se lanza un error de que ese usuario ya ha sido registrado) y redirigimos de nuevo a registro

- /delete

Acepta tanto los métodos GET como POST, la función login (def delete():) se ejecuta de la siguiente manera:

Recogemos el usuario y ponemos el parámetro active en false (en la tabla users aparecerá con un 0), después hacemos un commit y redirigimos a registro.

- /login

Acepta tanto los métodos GET como POST, la función login (def login():) se ejecuta de la siguiente manera:

Si el método HTTP que empleamos es POST (el más adecuado para envío de información a través de formularios (nuestro caso)) entonces recogemos el objeto usuario mediante una 'query' lanzada con el módulo de flask flask_sqlalchemy, lo gracioso es que en esa misma query estamos preguntando o mejor dicho 'filtrando' que el email del objeto usuario sea igual al email que hemos capturado en ese formulario, y con el método first pedimos el primer email que encuentre con esas características, así que solo vamos a retornar un objeto y solo uno. Una vez recogido el objeto usuario generamos la cookie email a la que le asignamos el valor del email capturado por formulario.

Si la propiedad del objeto user → 'active' está a true entonces quiere decir que esa cuenta (valga la redundancia) está activa, además añadí otra condición, que la password del objeto usuario sea la misma que la capturada en el formulario (para lo cual me valí de la función `check_password_hash(contraseña1, contraseña2)` siendo contraseña1 la propia del usuario y contraseña2 la capturada por formulario. Después generaba un 'diccionario' python clave-valor llamado `user_details` y completaba el objeto sesión con esos datos. Si entrábamos a la sesión entonces enviaba ese diccionario a perfil-usuario y sino, en caso de que la cuenta estuviese 'inactiva' o la contraseña fallase, o ambas, entonces eliminaba la cookie email con `session.pop("email", None)` de esta forma no se producirían errores al desactivar la cuenta y redireccionaba al login de nuevo.

- /logout

Recuperamos el objeto user (de nuevo) y preguntamos que, si existe usuario y si la cuenta asociada a este está activa, entonces de nuevo eliminamos la cookie email, y redirigimos a login, si no existe user o su cuenta no está activa o ambas, entonces redirigimos a index (homepage).

- /update

Acepta tanto los métodos GET como POST, la función login (`def update():`) se ejecuta de la siguiente manera:

Si el método HTTP que empleamos es POST (el más adecuado para envío de información a través de formularios (nuestro caso)) entonces recogemos el objeto usuario mediante una 'query' lanzada con el módulo de flask flask_sqlalchemy. Una vez recuperado el objeto user, lanzamos un condicional if, y preguntamos si la contraseña se está actualizando, si esto es así, entonces la nueva contraseña (`hashed_pw`) va a equivaler a esa nueva contraseña introducida por formulario 'hasheada' con el método sha256

Redirigimos a perfil-usuario y enviamos el diccionario `user-details` (de nuevo) por si tenemos que echar mano de esos datos en la vista.

9.3. Rutas algo especiales II (el reto)

- /getProducts

Acepta el método post (aunque se llame getProducts), dentro de la función get_products(): ocurre lo siguiente:

Declaramos la variable i como global porque si no nos va a dar un error de referencia (referenced before assignment), después generamos un array llamado productList que inicializamos a vacío (productList = []), después rescata los objetos producto y los guardamos en la variable products, max es otra variable que equivale al length de products - 1 porque si no nos saltaría un index bound out of exception.

Después con un bucle for que recorre esos productos nos movemos desde el 0 hasta la longitud de productos (la total), vamos modelando un array de objetos 'json' y cuando llegamos a la longitud de products-1 entonces salimos del bucle, pues ya tenemos toda la productList llena y mediante la función jsonify enviamos el array de objetos json a que sea interpretado por el front.

```
#endpoint products to ana (from mysql to json)

@main.route('/get_products', methods=['POST'])
def get_products():
    global i
    productList = []
    products = Products.query.all()
    max = len(products)-1
    print('patri',len(products))

    for i in range(0,len(products)):
        productList.append(
            {
                'id':products[i].id,
                'image':products[i].image,
                'nameProd': products[i].nameProd,
                'categoriaProd0':products[i].categoriaProd0,
                'categoriaProd1':products[i].categoriaProd1,
                'alergenosProd': [
                    { "nameAlerg":products[i].alergenosProd0nameAlerg, "imageAlerg":products[i].alergenosProd0imageAlerg},
                    { "nameAlerg":products[i].alergenosProd1nameAlerg, "imageAlerg":products[i].alergenosProd1imageAlerg},
                    { "nameAlerg":products[i].alergenosProd2nameAlerg, "imageAlerg":products[i].alergenosProd2imageAlerg}
                ],
                'price': products[i].price,
                'unidades': products[i].unidades
            }
        )

    if i == max:
        return jsonify(productList)
```

Para ello, elaboramos una función asíncrona. Cuando carga la página se ejecuta una función async que espera (await) a otra función llamada getProductos().

```
window.onload = async () => {
    console.log("ready!", window.location)
    allobjJson = await getProductos();
    objJson = allobjJson;
    // console.log(window.location);
    // console.log("onload");
    changePage(1);
};
```

Se trata de una petición ajax que actúa como un get (aunque ponga post), obtenemos resultados de esa url y lo metemos dentro de esa variable data, si todo está ok (.done) entonces mostramos el resultado que hemos recogido del endpoint (pero por consola) y retornamos el valor guardado por el endpoint (esta vez a la página).

```
async function getProductos() {
  var jqxhr;
  await $.post('http://127.0.0.1:5000/get_products', function (data, status) {
    console.log(data);
    jqxhr = data;
  })
  .done(function (result) {
    console.log('Datos regreso')
    console.log(result);
  });
  return jqxhr;
}
/* const promise = fetch('../productosLista.txt').then(
  function (u) { return u.json(); }
).then(
  function (json) {
    objJson = json;
    console.log(objJson);
  }
) */
```

Por último, por cada página se muestra la info de los productos.

```
for (var i = (page - 1) * records_per_page; i < (page * records_per_page); i++) {
  y = "";
  y +=
  `<div class="producto col-xl-3 col-lg-4 col-md-5 col-sm-6 col-11"><div class="precioCarrito justify-content-around"><span class="my-auto"><b>${obj
  listing_table.innerHTML += y;
};
```

- /get_faqs

Más de lo mismo que en productsList, lo único que varía es que el array que inicializamos a vacío (array = []) se llama faqsList. Recogemos las faqs con la query de sqlalchemy y lo pasamos a una variable faqs.

Después de nuevo con el método.append de python vamos introduciendo en la posición i+1 cada elemento y modelando así un array de objetos json (otra vez), al llegar al tope (lista llena) tomamos mano de la función jsonify y retornamos (ahora sí) el array de objetos json.

```

@main.route('/get_faqs', methods=['POST'])
def get_faqs():
    faqsList = []
    faqs = Faqs.query.all()
    max = len(faqs)-1
    print(max)
    for i in range(0, len(faqs)):
        faqsList.append(
            {
                'id': faqs[i].id,
                'question': faqs[i].question,
                'answer': faqs[i].answer
            }
        )

    if i == max:
        return jsonify(faqsList)

```

De nuevo con el método .onload dejamos claro que cuando cargue la ventana vamos a llamar a una función asíncrona que va a estar esperando a otra función llamada getFaqs() y cuándo esta esté preparada le pasará su valor a una variable llamada data;

```

window.onload = async () => {
    data = await getFaqs();
    actFAQs();
};

```

Esta función getFaqs() a su vez es otra asíncrona que await(espera) que la petición ajax surta efecto y retorne la información (recordemos que esa información se la pasamos a data!)

```

async function getFaqs() {
    var jqxhr;
    await $.post('http://127.0.0.1:5000/get_faqs', function (data) {
        console.log(data);
        jqxhr = data;
    })
    .done(function (result) {
        console.log('Datos regreso')
        console.log(result);
    });
    return jqxhr;
}

```

Por último se muestra la información con un bucle for en la función actFAQs().

```
function actFAQs() {  
  for (i in data) {  
    console.log('hash-->' + hash);  
    y += `<div class="card faq"><div class="card-header headFaq"><div  
      hash++;  
    }  
    document.getElementById("accordion").innerHTML = y;  
    console.log(data)
```

```
target="#id${hash}" aria-expanded="false" aria-controls="">${data[i].question}</button></div></div><div
```

- /getHistorial

Dentro de la función getHistorial(): recogemos el objeto usuario filtrando el correo por el mismo que se encuentre asociado a la cookie de sesión llamada email. Recogemos todas las encomiendas que ha realizado una persona dado que en la tabla orders hay una clave foránea llamada user_id que referencia a la clave primaria id de la tabla users.

```
@main.route("/getHistorial", methods=['GET'])  
def gethistorial():  
    user = Users.query.filter_by(email=session.get("email")).first()  
    order = Orders.query.filter_by(user_id=user.id).all() #todos los order por un id determinado  
    print(user)
```

Una vez hecho esto, generamos dos arrays en vacío:

```
result = []  
product = []
```

Ahora comenzamos con los bucles anidados:

Desde i = 0 hasta el mayor número de orders (len(order)):

relproducts = hacer una query en la tabla relacional (se encarga de generar una relación entre products y orders, es decir, esta tabla (relProductsOrders) sería algo así como la línea de pedidos. Lanzamos una query en esa tabla relacional y recogemos todas las líneas de pedido que existan cuya order_id = al id de la orden que estamos recorriendo.

Desde $j=0$ hasta el mayor número de líneas de pedido por orden, en ese momento ejecutamos la misma query, pero esta vez rescatamos todos los productos cuyo `id = product_id` de la línea de pedido que estemos ejecutando en ese momento

Desde $a=0$ hasta el mayor numero de productos por línea de pedido. Vamos añadiendo al array `product` con el método `.append` el nombre de cada producto

```
for a in range(0, len(products)):
    product.append({
        'name': products[a].nameProd
    })
```

Cuando el bucle más inferior termina (bucle de `a`). Continuamos con segundo bucle, y vamos añadiendo toda la información que necesitamos con el método `.append` al array vacío `result`, una curiosidad es que uno de los elementos de ese array de objetos json, `'productos'` referencia al array `products` que acabamos de llenar al terminar el bucle `for` de `a`.

```
result.append(
    {
        'id': order[i].id,
        'date': order[i].date,
        'productos': product,
        'precioTotal': order[i].total,
        'message': order[i].message
    }
)
```

Al salir del bucle `for` inicializamos el array `Product` a vacío otra vez, dado que comenzaremos con la siguiente orden que tendrá sus propias líneas de pedido nuevas.

Al final una vez modelado el array de objetos json `result`, lo enviamos mediante la función `jsonify` al front

Todo el código del que hemos hablado:

```

@main.route("/getHistorial", methods=['GET'])
def gethistorial():
    user = Users.query.filter_by(email=session.get("email")).first()
    order = Orders.query.filter_by(user_id=user.id).all() #todos los orden por un id determinado
    print(user)
    result = []
    product = []
    for i in range(0,len(order)):
        relproducts = RelProductsOrders.query.filter_by(order_id=order[i].id).all()
        for j in range(0,len(relproducts)):
            products = Products.query.filter_by(id=relproducts[j].product_id).all()
            print('productos',products)
            for a in range(0,len(products)):
                product.append({
                    'name': products[a].nameProd
                })
        result.append(
            {
                'id':order[i].id,
                'date':order[i].date,
                'productos': product,
                'precioTotal': order[i].total,
                'message': order[i].message
            }
        )
    product = []
    print(order[i].id)

    return jsonify(result)

```

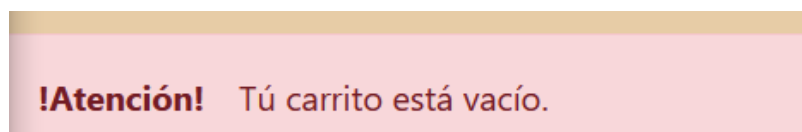
La recogida de información por parte del front sería la misma que en getHistorial y getFaqs.

- /insert_order

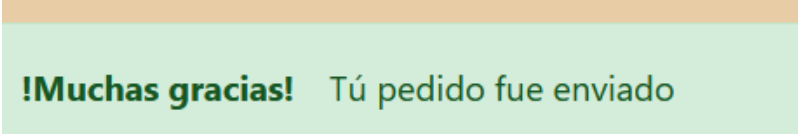
Esta ruta es a la que nos dirigimos cada vez que pinchamos en el botón 'finalizar encomienda'



Si pinchamos en finalizar encomienda y el carrito está vacío, saltará el mensaje (alert-danger, en rojo) !Atención! tu carrito está vacío

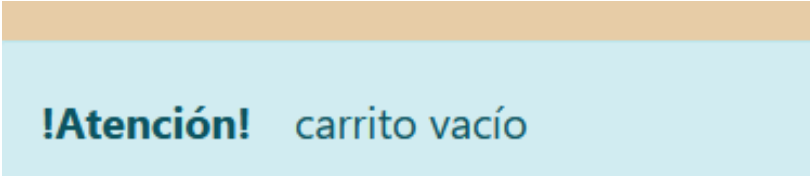


Si pinchamos en finalizar encomienda y el carrito está lleno:



!Muchas gracias! Tú pedido fue enviado

Si pinchamos en vaciar carrito, el carrito se vaciará y nos saltará el mensaje



!Atención! carrito vacío

Una vez pinchado en finalizar encomienda, saltará la función `saveCart()` (implementada en `carrito.js`)

```
// Save cart
function saveCart() {
  console.log('guardando carrito',cart);
  sessionStorage.setItem('shoppingCart', JSON.stringify(cart));
}
```

Después saltará la función `sendShoppingCart()`.

En esta función principalmente le damos formato al dato para enviarlo al backend. Añadimos los datos formateados al array `format`, tanto los del carrito como `.total-cart` que extraemos de la función `displayCart()`. Después generamos la petición AJAX y de esta forma enviamos el json al backend para que este lo interprete.

```

async function sendShoppingCart() {
  let format = [];
  let resp;
  let totalPrice = $('#total-cart').html();
  format.push(cart);
  format.push({ total: totalPrice });
  // console.log('Pedido: ',cart)
  if(cart[0]){ //if cart[0] != ''
    await $.ajax({
      type: 'POST',
      contentType: 'application/json',
      data: JSON.stringify(format),
      dataType: 'json',
      url: 'http://127.0.0.1:5000/insert_order',
      success: async function (data) {
        console.log('respuesta: ',data);
        await createRel(data);
        showAlert('exito');
      },
      error: function (error) {
        if (error.status === 401) {
          console.log('No autorizado');
        }
        //redirectBack();
        console.log('redirigir...')
        window.location.replace('http://127.0.0.1:5000/login');
      }
    });
  } else { //false, entonces carrito vacio
    showAlert('vacio');
    // window.location.replace('http://127.0.0.1:5000/login');
  }
}

```

Se trata de una función asíncrona, primero con un falso boolean preguntamos que si cart[0] (es decir, el contenido de cart) es igual a true, si es igual a false entonces el carrito está vacío pero sino entonces esperamos (await) a una petición ajax de tipo POST cuyo contenido va a ser json (contentType: 'application/json' y dataType: 'json'), el data va a ser lo mismo a coger el array format y convertirlo en json con data: JSON.stringify(format), la url a la que nos dirigimos con esta petición ajax es http://127.0.0.1:5000/insert_order

Si todo sale bien

Entonces success va a equivaler a una función asíncrona que recibe data (recordemos que data = array format lleno) entonces esperamos (await) a la función createRel(data).

```

async function createRel(data){
  let format = [];
  let element = [];
  cart.forEach(e => {
    element.push({
      product_id:e.id,
      quantity:e.count
    })
  });
  format.push({order_id: data});
  format.push(element);
  console.log("datos formateados:",format);
  await $.ajax({
    type: 'POST',
    contentType: 'application/json',
    data: JSON.stringify(format),
    dataType: 'json',
    url: 'http://127.0.0.1:5000/rel',
    success: function (e) {
      console.log('Guardado...', e);
      cart = [];
      sessionStorage.removeItem('shoppingCart');
      // redirectBack();
      //window.location.replace('http://127.0.0.1:5000/carrito');
    },
    error: function (error) {
      console.log(error);
      cart = [];
      sessionStorage.removeItem('shoppingCart');
      // redirectBack();
    }
  });
}

// redirectBack();
}).done(
  function (data) {
    console.log('Guardado...');
    cart = [];
    sessionStorage.removeItem('shoppingCart');
    // redirectBack();
    //window.location.replace('http://127.0.0.1:5000/carrito');
  }
);
}

```

Esta función es muy similar a sendShoppingCart, se trata de una función asíncrona, generamos dos array format y element y después con un bucle for each recorremos el carrito que ana generó en la función shoppingCart, en cada vuelta añadimos al array element el id y el precio del producto, después al salir del bucle añadimos al array format el order_id y le insertamos el array de productos element.

```

async function createRel(data){
  let format = [];
  let element = [];
  cart.forEach(e => {
    element.push({
      product_id:e.id,
      quantity:e.count
    })
  });
  format.push({order_id: data});
  format.push(element);
  console.log("datos formateados:",format);
}

```

Ahora esperamos (await) otra petición ajax de tipo post que envía json al backend, concretamente a la url: <http://127.0.0.1:5000/rel>, desde esa ruta el backend envía una respuesta, y un error.

Si envía la respuesta entonces la capturamos y la mostramos por consola (un OK, 200), entonces ponemos el carrito a vacío y removemos shoppingCart (ya que el carrito está guardado por sesión)

Si envía un error entonces lo mostramos pero igualmente procedemos a vaciar el carro y remover el objeto de sesión

Una vez que ha finalizado esta petición AJAX, entonces con .done indicamos que cuando todo termine volvemos a vaciar el carrito y a quitar el objeto de sesión shoppingCart.

```

await $.ajax({
  type: 'POST',
  contentType: 'application/json',
  data: JSON.stringify(format),
  dataType: 'json',
  url: 'http://127.0.0.1:5000/rel',
  success: function (e) {
    console.log('Guardado...', e);
    cart = [];
    sessionStorage.removeItem('shoppingCart');
    // redirectBack();
    //window.location.replace('http://127.0.0.1:5000/carrito');
  },
  error: function (error) {
    console.log(error);
    cart = [];
    sessionStorage.removeItem('shoppingCart');
    // redirectBack();
  }
}).done(
  function (data) {
    console.log('Guardado...');
    cart = [];
    sessionStorage.removeItem('shoppingCart');
    // redirectBack();
    //window.location.replace('http://127.0.0.1:5000/carrito');
  }
);

```

Si todo sale mal

Entonces error va a equivaler a una función que recibe un error (que le envía el backend). Si ese error es equivalente a 401 entonces mostramos con `console.log('No autorizado')`

Por el lado del back

En `/insert_order` aceptamos el método POST y dentro de la función `def insert_order()` ocurre lo siguiente:

`req_data = request.get_json()` esta es una forma de capturar el json que nos llega del front, recordemos que habíamos usado la función `JSON.stringify()` para convertir el array de objetos format en array de objetos json y pasarlo a back, pues así lo capturamos!.

Después como siempre recogemos con `sql_alchemy` el usuario de la sesión y si este existe y está activo entonces insertamos la order

```
order =  
Orders(date=datetime.now(),user_id=my_id,  
special_id=null(),  
message=null(),total=req_data[1]['total']  
)  
  
db.session.add(order)  
db.session.commit()
```

El total de la order = `req_data[1]['total']`, recordemos que a mi se me envía la información desde el front , primero todo el carrito que sería el puntero [0] y luego total: `totalPrice` siendo `totalPrice = $('#total-cart').html();`

```
let totalPrice = $('#total-cart').html();  
format.push(cart);  
format.push({ total: totalPrice });  
// console.log('Pedido: ',cart)
```

Por tanto si accedo al puntero 1, luego tengo que acceder al puntero total para obtener el dato.

Hacemos el commit de order

Generamos el diccionario `order_details` y en la variable `res`, de la mano de `make_response` generamos una respuesta para el front, en la que yo envío un 200 y el id de la order, despues devuelvo `res`. Si no existe usuario o no está activo entonces el objeto respuesta va a ser el texto 'Not Authorized' y un 401.

```
order_details = ({
    'id': order.id,
    'date': order.date,
    'user_id': order.user_id,
    'special_id': order.special_id,
    'message': order.message,
    'total': order.total
})

res = make_response(jsonify(order_details['id']), 200)
return res
else:
    res = make_response(jsonify('Not authorized'), 401)
    return res
```

Luego empleamos (internamente) la ruta `/rel` para insertar los datos en la tabla relacional

```
@main.route('/rel', methods=['POST'])
def rel():
    global i
    req_data = request.get_json()
    print('paso 2: ', req_data)
    max = len(req_data[1])-1

    #select order_id from orders where user_id =(select id f
    #select order_id from orders o join users u on o.user_id

    #sqlalchemy
    #query = db.session.query(Orders).\
    #join(Users).\
    #filter(Users.id == Orders.user_id).first()

    for i in range(len(req_data[1])):
        rel = RelProductsOrders(
            quantity=req_data[1][i]['quantity'],
            product_id= req_data[1][i]['product_id'],
            order_id = req_data[0]['order_id']
        )
        db.session.add(rel)

    db.session.commit()
    res = make_response(jsonify('OK', 200))
    return res
```

9.4. Más rutas

- /form → empleado en el update
- /insert_special_order → para los pedidos especiales
- /user_messages → para los mensajes que nos mandan los usuarios

9.5. Base de datos

En el archivo models se encuentra todo el esquema de nuestra base de datos, esta está formada por 6 tablas

Tabla Users

```
class Users(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(50), nullable=False)
    lastname = db.Column(db.String(50))
    telephone = db.Column(db.String(50), unique=False, nullable=False)
    email = db.Column(db.String(50), unique=False, nullable=False)
    password = db.Column(db.String(80), nullable=False)
    active = db.Column(db.Boolean, nullable=False)
```

Tabla Products

```
class Products(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    image = db.Column(db.String(50), nullable=False)
    nameProd = db.Column(db.String(50), nullable=False)
    categoriaProd0 = db.Column(db.String(50), nullable=False)
    categoriaProd1 = db.Column(db.String(50), nullable=False)
    alergenProd0nameAlerg = db.Column(db.String(50), nullable=True)
    alergenProd0imageAlerg = db.Column(db.String(50), nullable=True)
    alergenProd1nameAlerg = db.Column(db.String(50), nullable=True)
    alergenProd1imageAlerg = db.Column(db.String(50), nullable=True)
    alergenProd2nameAlerg = db.Column(db.String(50), nullable=True)
    alergenProd2imageAlerg = db.Column(db.String(50), nullable=True)
    price = db.Column(db.Integer)
    unidades = db.Column(db.Integer)
```

Tabla Faqs

```
class Faqs(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    question = db.Column(db.String(500), nullable=False)
    answer = db.Column(db.String(500), nullable=False)
```

Tabla Messages

```
class Messages(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    message = db.Column(db.String(500), nullable=False)
    user_id = db.Column(db.Integer, db.ForeignKey('users.id'))
```

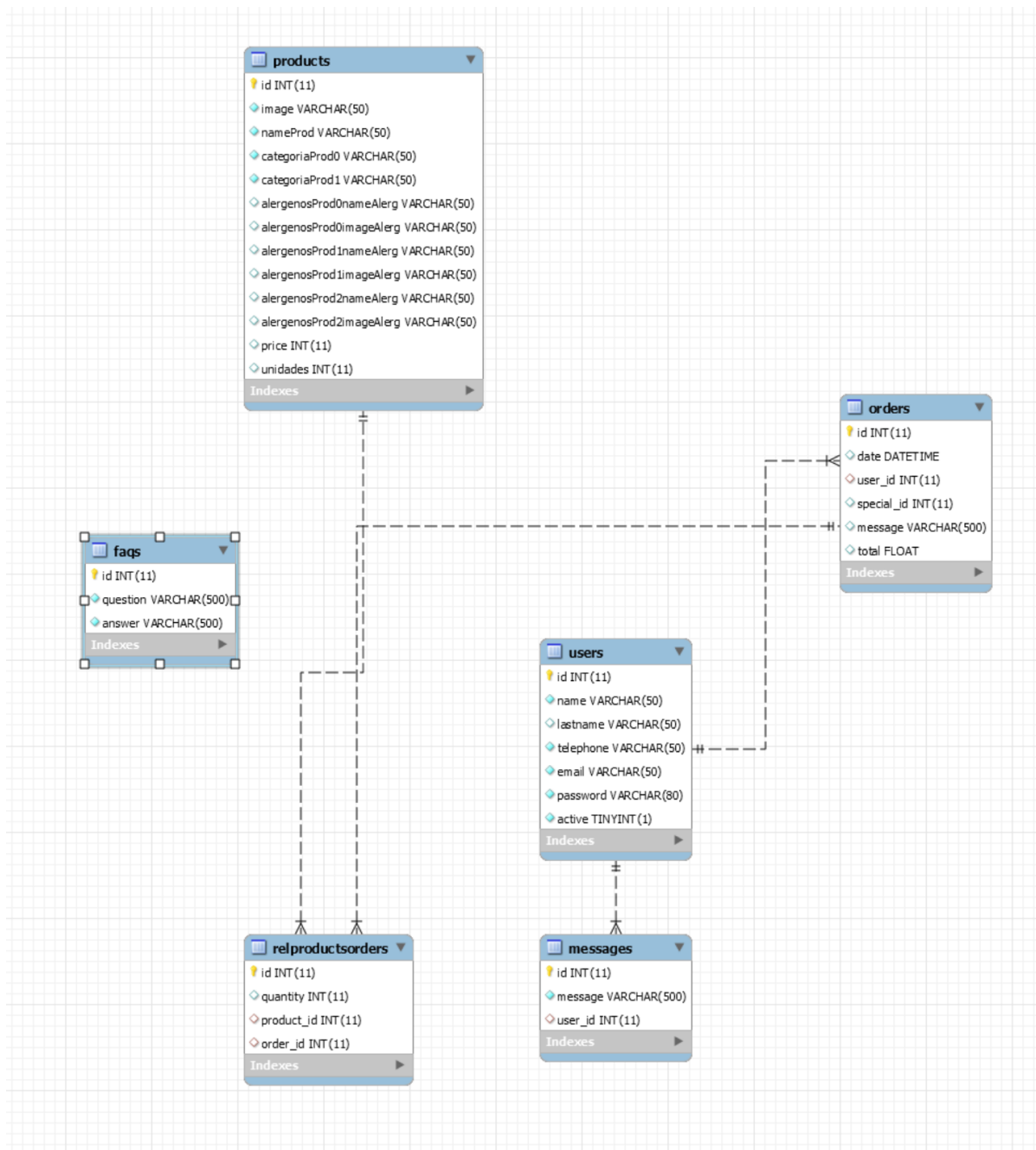
Tabla Orders

```
class Orders(db.Model):
    id = db.Column(db.Integer, primary_key=True) #order id
    date = db.Column(db.DateTime, index=True)
    user_id = db.Column(db.Integer, db.ForeignKey('users.id'))
    special_id = db.Column(db.Integer, nullable=True)
    message = db.Column(db.String(500), nullable=True)
    total = db.Column(db.Float, nullable=True)
```

Tabla RelProductsOrders

```
class RelProductsOrders(db.Model): #tabla relacional entre orders y products
    __tablename__ = 'relProductsOrders'
    id = db.Column(db.Integer, primary_key=True)
    quantity = db.Column(db.Integer, nullable=True) #en esa orden tenemos 'x' cantidad
    product_id = db.Column(db.Integer, db.ForeignKey('products.id', onupdate='cascade', ondelete="cascade"))
    order_id = db.Column(db.Integer, db.ForeignKey('orders.id', onupdate='cascade', ondelete="cascade"))
    products = relationship('Products', backref='relProductsOrders')
    orders = relationship('Orders', backref='relProductsOrders')
```


9.6. MODELO E-R



9.7. Otros puntos

En el fichero init se encuentra la ‘inicialización’ de nuestra aplicación y de la respectiva base de datos con su cadena de conexión.

```
db = SQLAlchemy()

def create_app(config_obj=None):

    app = Flask(__name__, instance_relative_config=False)
    Bootstrap(app)
    app.config['SQLALCHEMY_DATABASE_URI'] = 'mysql+pymysql://root:password@localhost:3306/mydb'
    app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
    app.secret_key = "12345"
    db.init_app(app)

    from core.routes import main
    from core.models import Users, Products

    # register the blueprints
    app.register_blueprint(main)

    with app.app_context():
        from . import routes
        db.create_all()

    return app
```

Finalmente el fichero run es el que inicializamos con Python.

```
from core import create_app

class Config:
    DEBUG = True

app = create_app(Config)

if __name__ == '__main__':
    app.run(debug=True, host='127.0.0.1')
```

10. Conclusiones / Consideraciones Finales

Creemos que la página está completa y que funciona correctamente.

En el inicio del proyecto habíamos pensado en desarrollar una pequeña plataforma para que la empresa pudiera gestionar todos sus clientes, pedidos y mensajes, pero optamos por no seguir con la idea por falta de tiempo.

También teníamos la idea de desplegar la aplicación. Sin embargo, por falta de tiempo, hemos decidido presentarla sin desplegar y dedicarnos a perfeccionar otros puntos de la aplicación.

11. Bibliografía y Recursos

- Apuntes de Instituto Tecnológico Telefónica, disciplinas de DAW
- Gordiyenko, Svetlana. "Website Development Process: Full Guide in 7 Steps". XB Software, 2015 - <https://xbsoftware.com/blog/website-development-process-full-guide/>
- Web Lion Technology. "Website Development Process: Full Guide in 7 Steps". Web Lion Technology - <https://webliontechnology.com/website-development-process-blog/>
- Kiss, Eva. "Step-by-Step Guide to the Website Development Process". Neglia Design - <https://negliadesign.com/general/website-development-process/>
- Wodehouse, Carey. "A Beginner's Guide to Back-End Development". Up Work, 2018 - <https://www.upwork.com/hiring/development/a-beginners-guide-to-back-end-development/>
- Reimer, Luke. "Following A Web Design Process". Smashing Magazine, 2011 - <https://www.smashingmagazine.com/2011/06/following-a-web-design-process/>
- Meazey, Matt. "7 simple steps to the web design process". WebFlow, 2020 - <https://webflow.com/blog/the-web-design-process-in-7-simple-steps>
- Arnolj, Barbara. "How to design a style guide for websites". UX Collective, 2019 - <https://uxdesign.cc/all-you-need-to-know-about-style-guide-9513ebf50b46>
- Laurinavicius, Tomas. "How To Create a Web Design Style Guide". Design Modo, 2017 - <https://designmodo.com/create-style-guides/>
- Gloomaps - <https://www.gloomaps.com/>
- Adobe XD - <https://www.adobe.com/products/xd.html>
- Flaticon - <https://www.flaticon.com>
- Stack Overflow - <https://stackoverflow.com/>

- Pypi - <https://pypi.org/project/Flask/>
- Full Stack Python - <https://www.fullstackpython.com/flask.html>
- Flask - <https://flask.palletsprojects.com/en/1.1.x/>
- Json - <https://www.json.org/json-en.html>
- W3Schools - <https://www.w3schools.com/>
- Developer Mozilla - <https://developer.mozilla.org/en-US/>
- Javascript Info - <https://javascript.info/>
- Bootstrap - <https://getbootstrap.com/>
- JQuery - <https://jquery.com/>
- JQuery User Interface - <https://jqueryui.com/>
- GitHub - <https://github.com/>
- Source Tree - <https://www.sourcetreeapp.com/>
- Visual Studio Code - <https://code.visualstudio.com/>
- Visual Studio Code. "Flask Tutorial in Visual Studio Code". <https://code.visualstudio.com/docs/python/tutorial-flask>
- Jet Brains Pycharm - <https://www.jetbrains.com/pycharm/>
- Docker - <https://www.docker.com/>
- Pérez, Mabel. "Plantilla de ejemplo de Política de privacidad [2019]". Super Admin, 2019 - <https://superadmin.es/blog/hosting/plantilla-politica-de-privacidad/>
- Pérez, Mabel. "Plantilla de ejemplo de Política de Cookies". Super Admin, 2018 - <https://superadmin.es/blog/privacidad/plantilla-politica-de-cookies/>
- Screen Sizes - <http://screensiz.es/phone>
- Tutorials Point. "Flask - Quick Guide". Tutorials Point - https://www.tutorialspoint.com/flask/flask_quick_guide.htm
- Code No School. "Curso Flask". Code No School Youtube, 2018 - <https://www.youtube.com/playlist?list=PLBO4apWPK7b7K6c-jpEI0zfIZYDjVp7cd>