LAUREATE INTERNATIONAL UNIVERSITIES®

UVM Hispano

Tecnologías de Construcción de Servicios Web

CARRERA:
INGENIERÍA EN SISTEMAS COMPUTACIONALES

Propuesta de proyecto: Tienda en Línea

AUTORA:
Mónica Gómez

CICLO ESCOLAR 2025-2026

Índice

Introducción

En la materia **Tecnologías de Construcción de Servicios Web** del séptimo semestre de la Ingeniería en Sistemas Computacionales de UVM Hispano se planteó el desarrollo de un proyecto que demuestre lo aprendido a lo largo del ciclo escolar.

Nuestro equipo acordó hacer un prototipo de tienda de ropa que *emule* las dinámicas de los negocios por *e-commerce*, y que presente las características típicas de una tienda en línea. La intención no es hacer una tienda completa ni *un nuevo competidor de Amazon*, pero sí se busca crear, como mínimo, un prototipo funcional que ayude a aplicar los conceptos aprendidos durante el semestre.

Requisitos funcionales

La tienda debe...

- permitir el registro y acceso de usuarios.
- implementar la verificación en 2 pasos.
- incorporar un *carrito de compras* que permita comprar múltiples productos en una sola exhibición.
- implementar una API para el tratamiento de la información.
- permitir distintos métodos de pago.
- mostrar el historial de compras por usuario.
- permitir que los usuarios administradores agreguen, eliminen y modifiquen productos.

Requisitos no funcionales

La tienda debe...

- implementar un tema claro y uno oscuro.
- mostrar su interfaz de usuario lo más rápido posible. Habiendo mostrado la interfaz, se permite un retraso corto para la carga y el despliegue de la información.
- adaptarse a las preferencias de cada usuario.
- ser intuitiva y fácil de usar.
- usar elementos gráficos y colores que ayuden a distinguir cada sección.
- implementar medidas de seguridad que no pongan en riesgo la privacidad de la información personal de los usuarios.

Roles de equipo

Con base en lo acordado por el equipo, los roles quedarán asignados de la siguiente manera:

- Archer Cervantes: Backend — implementación de *endpoints* y respuestas de la API.
- Axel Renovato: Backend — implementación de consultas en la BD y procesado de la información.
- Hoshi Zárate: Llenado y edición de productos.
- Juan Flores: Llenado y edición de productos.
- Leonardo Rivera: Llenado y edición de productos.
- Mónica Gómez: Frontend — diseño de interfaz y experiencia de usuario. Backend — manejo de usuarios y autenticación, implementación de la API de Stripe para el procesado de pagos. Supervisión del proyecto en general.

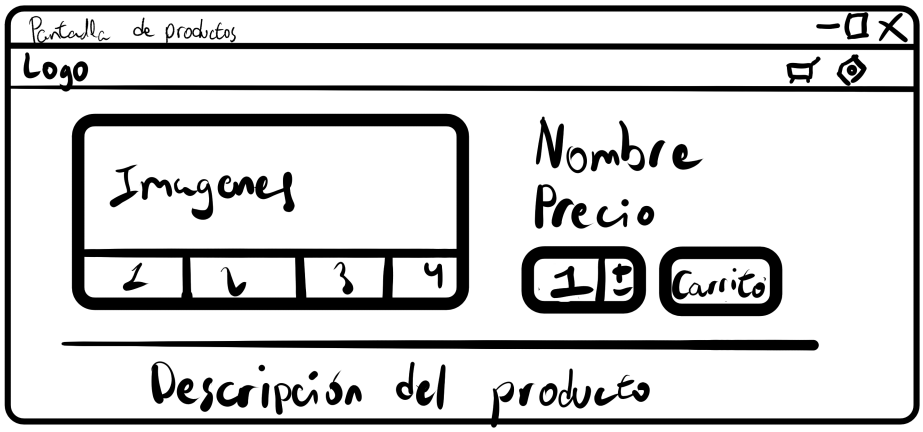
Propuesta de Wireframes

Los Wireframes propuestos fueron hechos a mano mediante Krita.

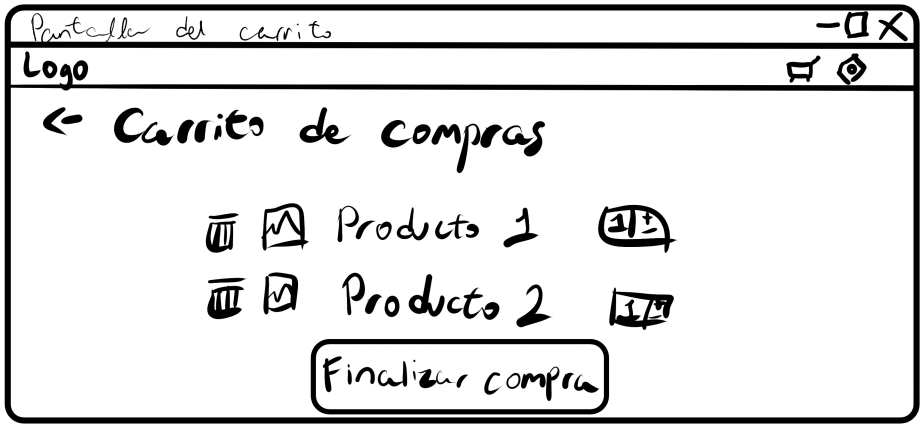
Pantalla principal

Pantalla de categorías

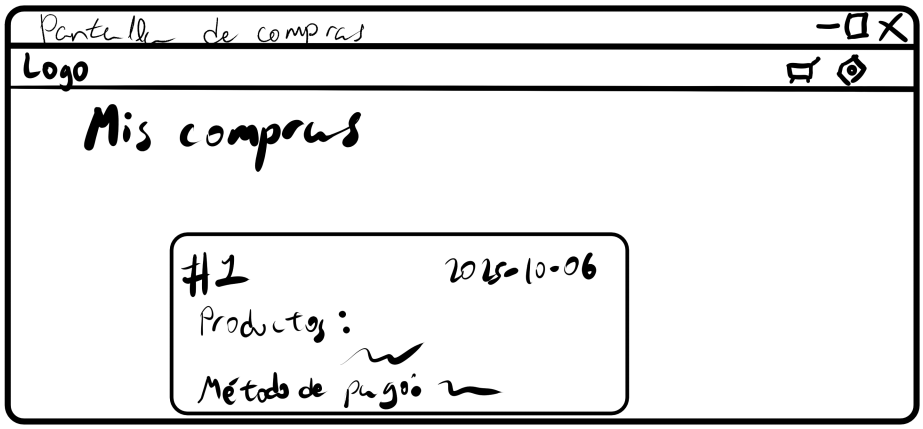
Vista de producto



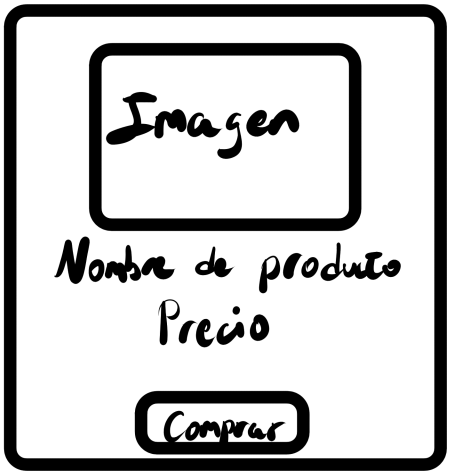
Pantalla del carrito



Pantalla “Mis compras”



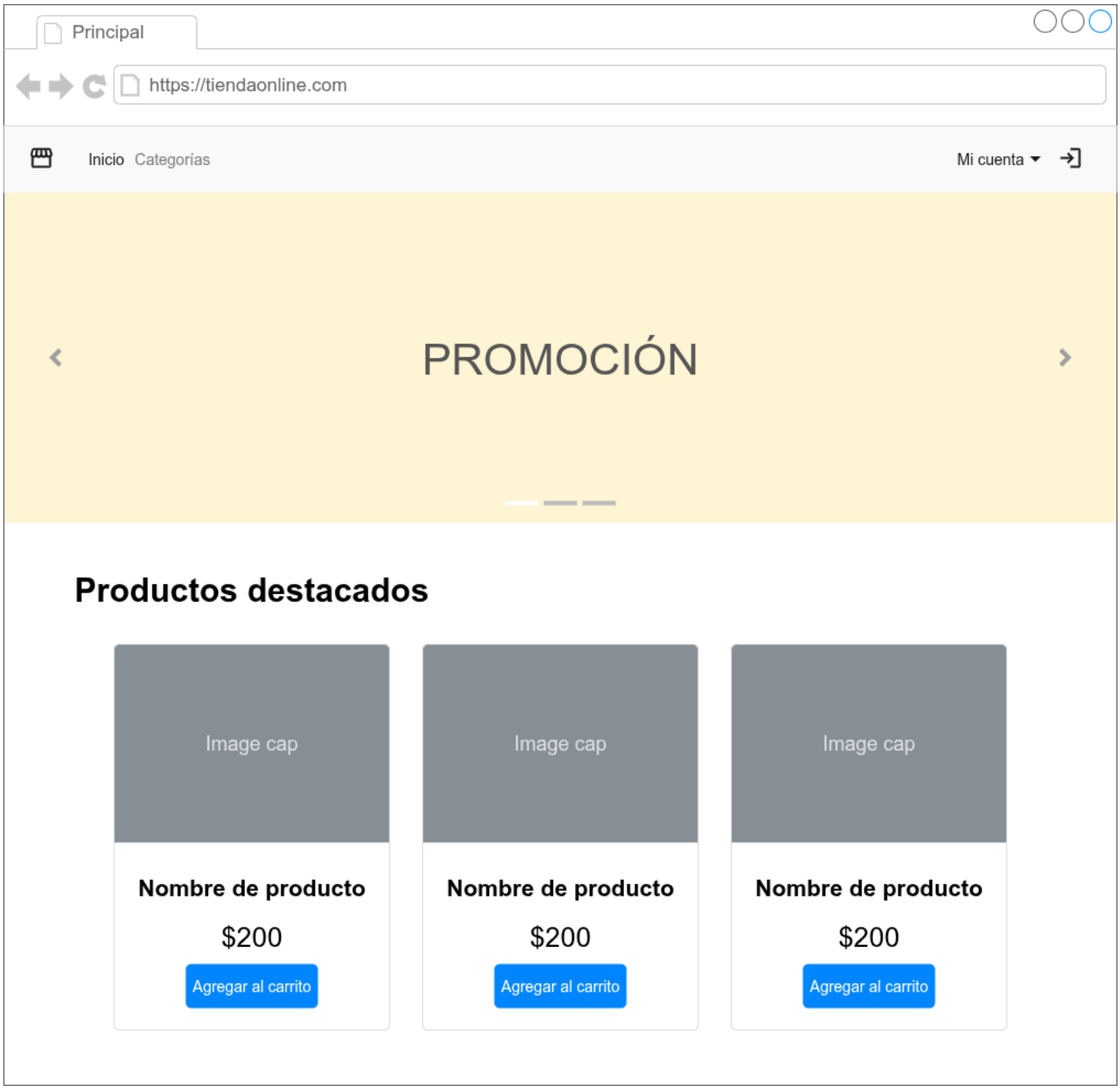
Tarjeta de productos



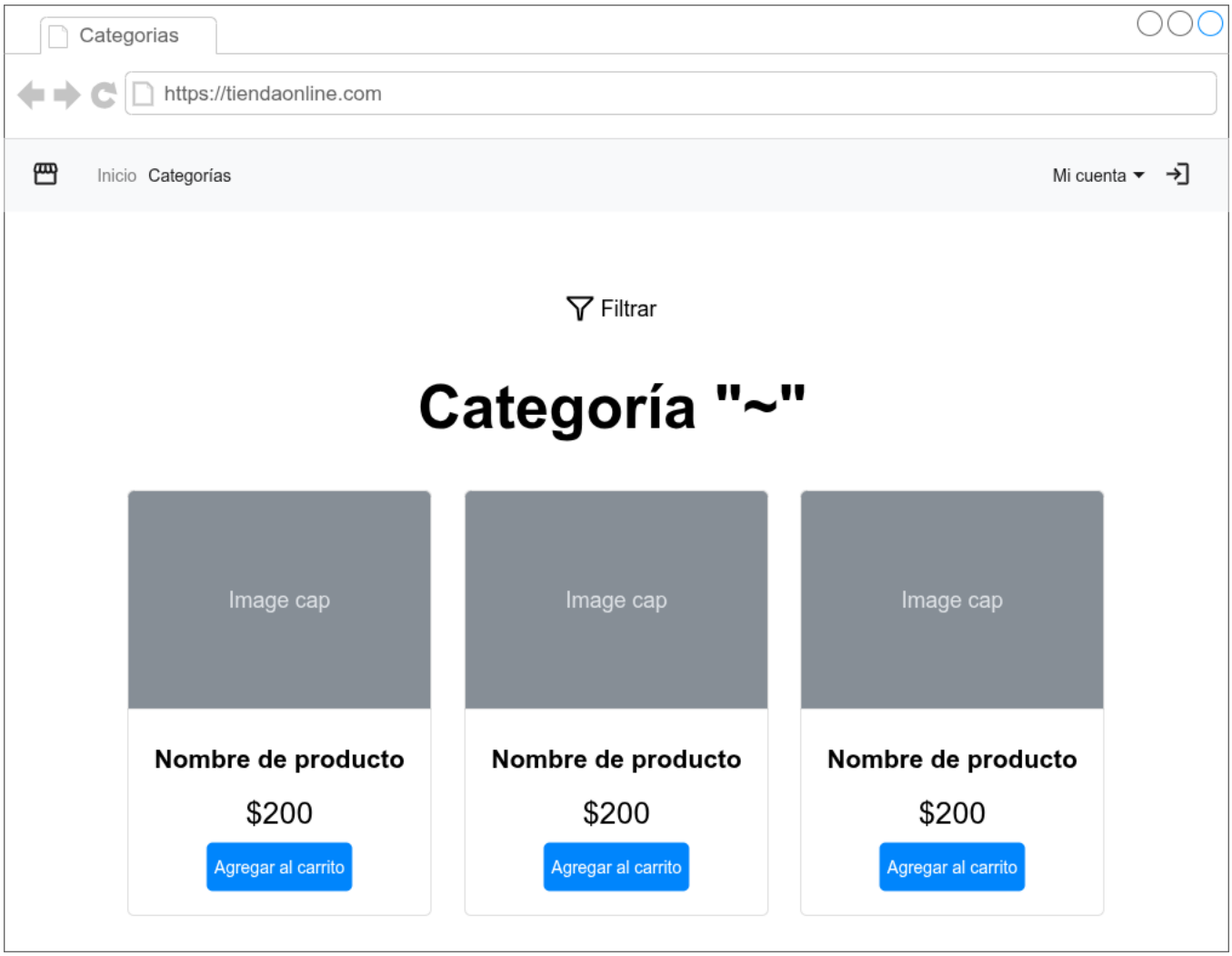
Propuesta de Mockups

Los Mockups propuestos fueron hechos mediante **draw.io**.

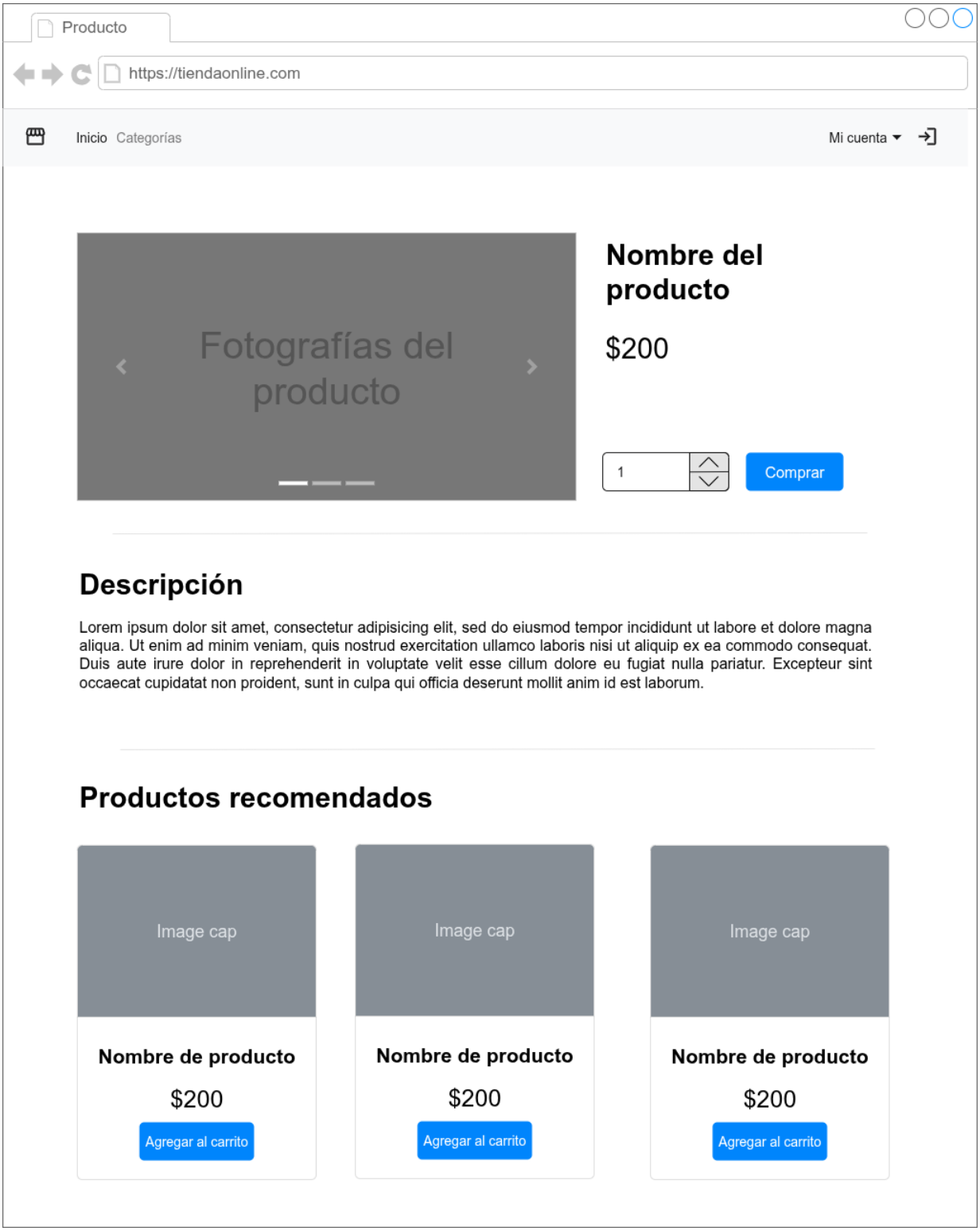
Pantalla principal



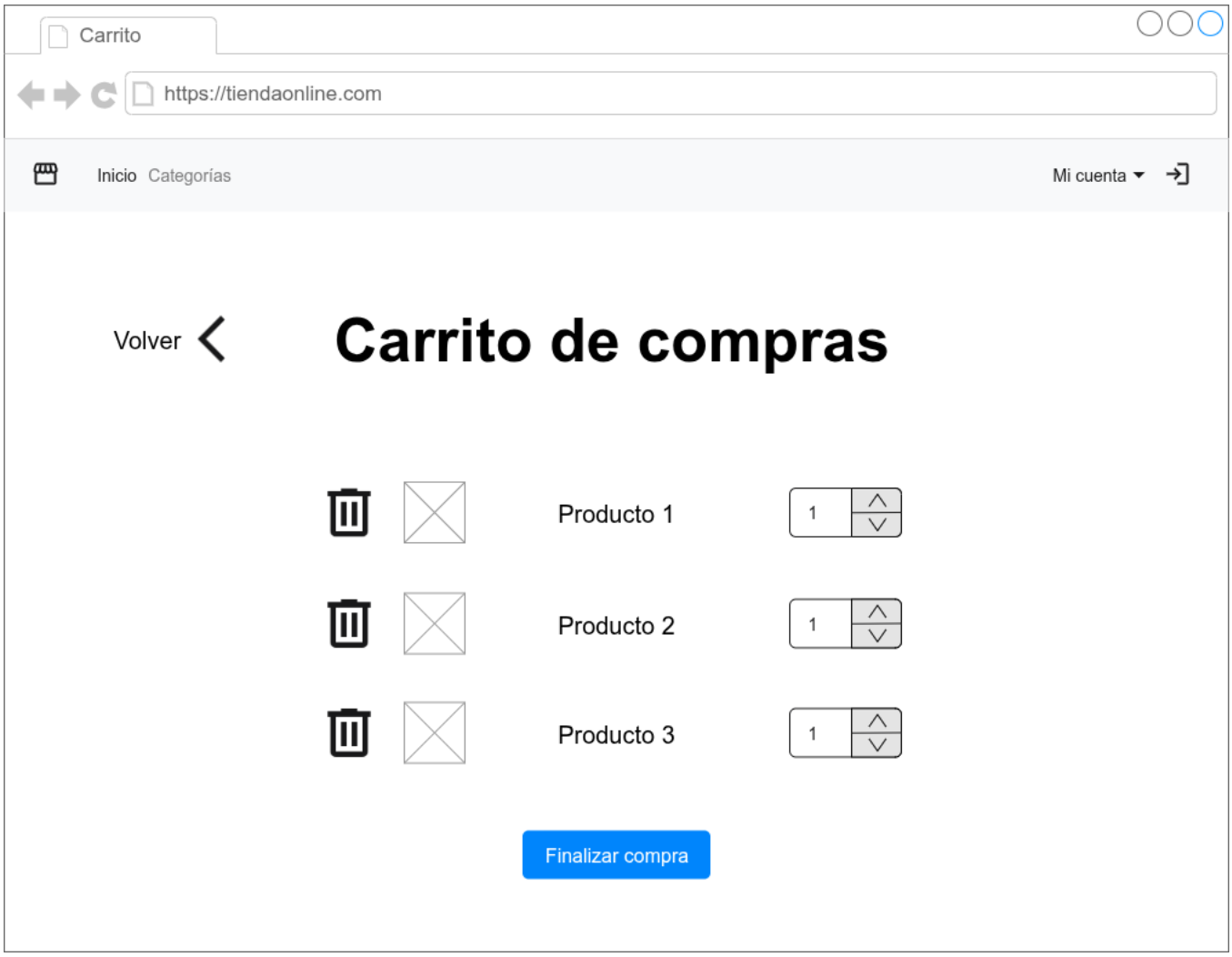
Pantalla de categorías



Vista de producto

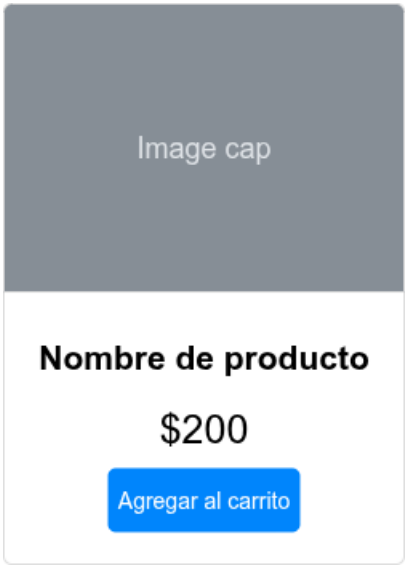


Pantalla del carrito

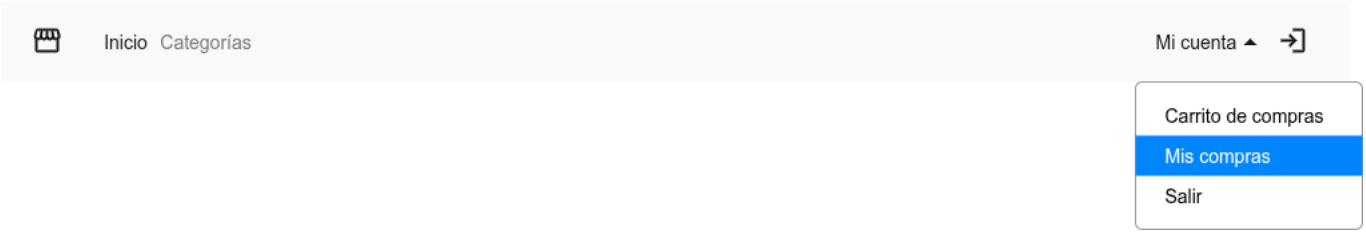


Pantalla “Mis compras”

Tarjeta de productos



Barra de navegación



Especificaciones

Dado que hacer una tienda completa y destinada a producción no forma parte de los objetivos del proyecto, se buscará utilizar tecnologías que no tengan una complejidad demasiado alta, pero que cuenten con las capacidades necesarias para cumplir con los requisitos planteados. Esto, sumado al contexto del semestre y de las tecnologías que hemos estudiado en clase, será lo que determinen las tecnologías a utilizar para el desarrollo del proyecto.

Frontend

Tecnologías a utilizar

Con base en lo mencionado anteriormente y aplicando lo aprendido en clase, las tecnologías que se utilizarán para el *frontend* son las siguientes:

- Interfaz de usuario: Bootstrap.
- Manejo de datos dinámicos e interactividad con la página: jQuery.

Especificaciones de frontend

Se utilizarán plantillas, que se generan con la etiqueta HTML `<template>`, para crear elementos reutilizables que puedan colocarse y quitarse de la página con facilidad. Por ejemplo, las tarjetas para mostrar los productos de la página principal podrían utilizar la siguiente plantilla:

```
<template id="product-card-template">
<div class="card product-card" style="width: 18rem;">
  <img class="card-img-top">
  <div class="card-body">
    <h4 class="card-title"></h4>
    <h4 class="card-text"></h4>
    <a class="btn btn-primary">Agregar al carrito</a>
  </div>
</div>
</template>
```

Con esta plantilla se puede generar una cantidad ilimitada de tarjetas de producto sin necesidad de repetir código. Un ejemplo de esto aplicándolo mediante jQuery con una función podría ser el siguiente:

```
function createProductCard($parent, img, name, price, link){
  let $card = $($('#product-card-template').html()); // Se clona la tarjeta
 para poder generarla.

  $card.find("img").attr("src", img);
  $card.find(".card-title").text(name);
  $card.find(".card-text").text(price);
  $card.find(".btn").attr("href", link);

  $parent.append($card);
}
```

Si después deseamos generar tarjetas de productos en la pantalla principal a partir del resultado obtenido desde la API, basta con aplicar un bucle de la siguiente manera:

```
for (let element of response.data){
  createProductCard($("#container"), element.imagenes[0], element.nombre,
 element.precio, 'https://tienda.com/products/' + element.id);
}
```

Y con ello habremos generado tarjetas de productos utilizando código simple y reutilizable.

Backend

Tecnologías a utilizar

Dada la naturaleza del proyecto, las tecnologías para el *backend* que se utilizarán conforme a lo acordado son las siguientes:

- Servidor Backend/API: Flask.
- Base de datos: MySQL.
- Proxy inverso: Nginx.
- Herramienta de procesamiento de pagos: Stripe.

Especificaciones de API

La API en su totalidad dará respuestas en formato JSON. Una respuesta JSON bien formada puede contar con cuatro campos, los cuales son `code`, `status`, `data` y `message`.

Un ejemplo de respuesta válida para una petición correcta es el siguiente:

```
{
  "code": 200,
  "status": "success",
  "data": {
    "id": 2,
    "username": "autumn64"
  }
}
```

Un ejemplo de respuesta válida para una petición errónea puede ser el siguiente:

```
{
  "code": 404,
  "status": "fail",
  "message": "El usuario no existe"
}
```

A continuación se explica el significado y los posibles valores para cada campo:

Campo	Valores permitidos	Significado
`code`	2xx, 4xx, 5xx	Código de estado HTTP que indica el resultado de la petición realizada.
`status`	"`success`", "`fail`", "`error`"	Indica si la operación fue exitosa.
`data`	objeto, array, `null`	Información de respuesta que debe ser procesada por el cliente.
`message`	`string`	Mensaje para mostrar al usuario.

Si bien es posible utilizar `data` y `message` en la misma respuesta, lo ideal es utilizar sólo uno a la vez, ya que el primero envía información que después el cliente (como el navegador web) procesará, mientras que el segundo retorna un mensaje que se mostrará directamente al usuario como un mensaje o un modal.

Los códigos de estado HTTP que se tiene planeado utilizar son los siguientes:

Código	Nombre	Descripción
200	OK	La petición se realizó correctamente.
201	Created	La petición se completó y se creó un recurso nuevo.
204	No Content	La petición se completó y no retornó ningún contenido.
400	Bad Request	La petición está malformada o es incorrecta, y no se procesará.
401	Authorization Required	La petición es correcta, pero necesita que el cliente esté autenticado.
404	Not Found	El recurso pedido no existe o no fue encontrado.
405	Method Not Allowed	La petición se realizó utilizando un método diferente al esperado.
500	Internal Server Error	La petición fue correcta, pero el servidor no es capaz de procesarla debido a un error interno.

El uso de estos códigos de estado necesariamente implica que se utilizarán los siguientes métodos HTTP para realizar las peticiones del cliente al servidor:

Método	Descripción
GET	Acceder al recurso especificado.
POST	Agregar un recurso nuevo.
PUT	Actualiza un recurso ya existente.
DELETE	Elimina el recurso especificado.

La API se expondrá mediante una serie de "*endpoints*", que corresponden a un recurso y que aceptarán distintos métodos. Estos *endpoints* deberán llevar el prefijo "`/api/`", y deben ser nombrados mediante sustantivos. Por ejemplo:

- Un *endpoint* llamado `/api/createUser/` es incorrecto aunque, en teoría, cree usuarios nuevos.
- Un *endpoint* llamado `/api/deleteUser/` también es incorrecto aunque, en teoría, elimine usuarios.

Para ambos casos es preferible crear un único *endpoint*, llamado `/api/user/`, que sea capaz de realizar ambas operaciones (crear y eliminar usuarios) dependiendo del método con el que se hizo la petición (en este caso, POST o DELETE).

Adicionalmente, para que las peticiones sean aceptadas deben proporcionar un *token* de autenticación, que se genera y se guarda al momento de iniciar sesión.

Entonces, una petición válida para obtener los productos de la categoría “Pantalones” podría verse de la siguiente manera:

```
GET /products?category=pantalones HTTP/1.1
Host: api.tienda.com
Authorization: Bearer TOKEN
Accept: application/json
```

La respuesta recibida por parte del servidor podría ser algo similar a lo siguiente:

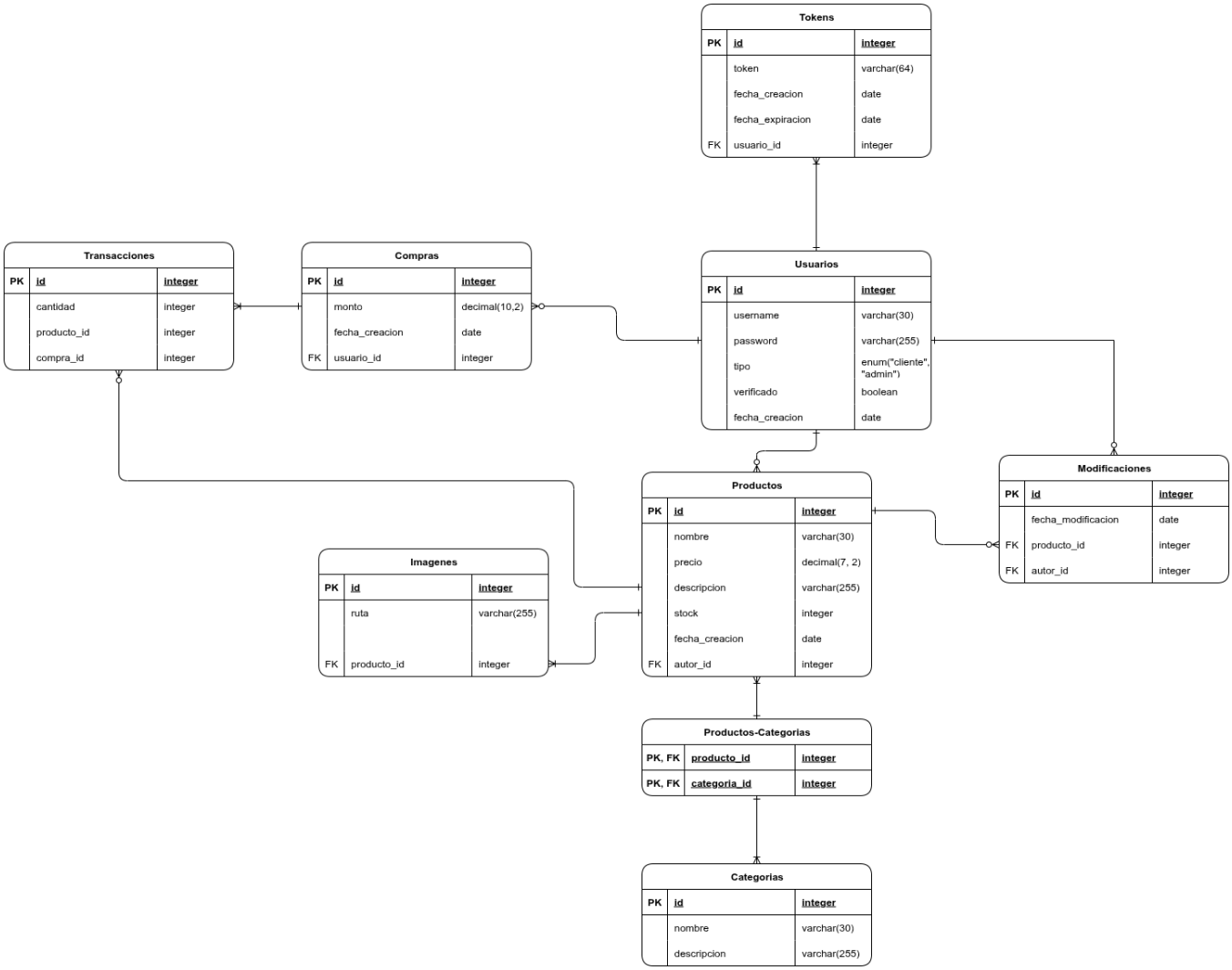
```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "code": 200,
  "status": "success",
  "data": [
    {
      "id": 15,
      "nombre": "Pantalones Jeans Hombre",
      "descripcion": "Pantalones jeans para hombre talla mediana.",
      "categoria": "pantalones",
      "imagenes": [
        "static/img/pj1321423.png",
        "static/img/pj1234328.png"
      ],
      "precio": 50,
      "stock": 5
    },
    {
      "id": 17,
      "nombre": "Pantalones de vestir Mujer",
      "descripcion": "Pantalones de vestir para mujer talla chica.",
      "categoria": "pantalones",
      "imagenes": [
        "static/img/pv578934.png",
        "static/img/pv905438.png"
      ],
      "precio": 70,
      "stock": 2
```

```
      },
      {
        "id": 26,
        "nombre": "Pantalones Jeans" Mujer,
        "descripcion": "Pantalones jeans para mujer talla grande.",
        "categoria": "pantalones",
        "imagenes": [
          "static/img/pj9094321.png",
          "static/img/pj5462397.png"
        ],
        "precio": 65,
        "stock": 10
      }
    ]
  }
```

De este modo, se pueden obtener respuestas que siempre tengan la misma estructura, que proporcionen toda la información necesaria para la operación, que no requieran manejo de estados y que separen por completo al cliente del servidor en la arquitectura.

Base de Datos

El diagrama de base de datos propuesto es el siguiente:



Consideraciones de diseño

Las siguientes propuestas son recomendaciones que se deberían tomar en cuenta para el diseño de la aplicación, si bien su implementación no es estrictamente obligatoria y depende de los acuerdos a los que se lleguen en el equipo.

Carrito de compras

Se sugiere que el carrito de compras se implemente del lado del cliente, mediante el uso de localStorage o de variables locales. La implementación se podría llevar a cabo mediante un *array* que contenga la información de cada artículo, que después se enviaría al servidor en formato JSON para su procesado en las operaciones de compra.

Esta implementación tiene algunas ventajas y desventajas, las cuales se plantean a continuación:

- **Ventajas**:
  - Es muy fácil y rápido de implementar.
  - No depende de consultas SQL para agregar o quitar carritos (no así para mostrarlos).
  - Se puede procesar rápidamente en el servidor.
- **Desventajas**:
  - No es escalable.
  - Sólo se guarda en el dispositivo en el que se creó el carrito, lo que impide que el carrito sea persistente y se comparta entre dispositivos.
  - Su implementación depende del cliente que se está utilizando, lo que lo vuelve menos estándar.

Operaciones de compra

No se tiene planeado implementar un sistema de reembolsos ni de devoluciones, razón por la cual se omitió el campo `estado` en la tabla `Compras` de la base de datos.

Pantalla de compra

La idea es utilizar la pantalla de compra proporcionada por la API de Stripe, con el propósito de evitar aumentar considerablemente la complejidad del proyecto.

Autenticación y manejo de tokens

Los tokens de autenticación deberían generarse cada vez que el usuario inicia sesión, esto con el propósito de permitir el inicio de sesión desde múltiples dispositivos a la vez. Asimismo, el token generado para un determinado dispositivo debería eliminarse cuando el usuario cierre sesión en ese dispositivo, o cuando éste expire.

Estos tokens deben guardarse en el localStorage y no en una *cookie* ni en una sesión de estado.

Control de usuarios

La aplicación debería permitir enviar un *link* de reestablecimiento de contraseña para los usuarios que lo necesiten. Dicho *link* debería incluir una opción que permita que el usuario escoja si desea cerrar la sesión en todos los dispositivos.

Las contraseñas deberían convertirse en un *hash* que después se guarde en la base de datos; el almacenamiento de contraseñas en texto plano está desaconsejado.

Manejo de estado en respuestas API

La API está diseñada para retornar tres posibles valores de estado, los cuales son `success`, `fail` y `error`. Con respecto a estos dos últimos, el primero debería utilizarse con errores que fueron provocados por el cliente, mientras que el segundo debería reservarse para errores producidos desde el servidor. Entonces, cada respuesta de estado debería corresponder a los siguientes códigos HTTP:

- **success**: 200, 201, 204
- **fail**: 400, 401, 404, 405
- **error**: 500

Licenciamiento del proyecto

Dado que esto es un prototipo que no está pensado para utilizarse en producción, se sugiere que el código fuente en HTML, CSS, JavaScript, Flask y SQL se haga disponible al dominio público de acuerdo con la licencia CC0, lo que permite que el proyecto sea incluido en cualquier momento y de forma libre en las aplicaciones, los proyectos o los portafolios de todos los integrantes del equipo, además de que permite que el código sea utilizado como referencia o guía sin restricciones para personas que estén interesadas en crear proyectos similares.

Todos los miembros del equipo tienen derecho a aceptar o rechazar esta recomendación, en cuyo caso se sugiere discutir una licencia más pertinente que proteja los intereses de todos los involucrados.

La licencia de dominio público “CC0” se puede acceder desde la URL: https://creativecommons.org/public-domain/cc0/

Disclaimer

NO HAY GARANTÍA PARA EL PROGRAMA, PARA LA EXTENSIÓN PERMITIDA POR LA LEY APLICABLE. EXCEPTO CUANDO SE INDIQUE LO CONTRARIO POR ESCRITO, LOS TITULARES DE LOS DERECHOS DE AUTOR (“COPYRIGHT”) Y/O TERCEROS PROPORCIONAN EL PROGRAMA “TAL CUAL” SIN GARANTÍAS DE NINGÚN TIPO, BIEN SEAN EXPLÍCITAS O IMPLÍCITAS, INCLUYENDO, PERO NO LIMITADO A, LAS GARANTÍAS IMPLÍCITAS DE COMERCIALIZACIÓN Y APTITUD PARA UN PROPÓSITO PARTICULAR. EL RIESGO TOTAL EN CUANTO A CALIDAD Y RENDIMIENTO DEL PROGRAMA ES CON USTED. SI EL PROGRAMA PRESENTA ALGÚN DEFECTO, USTED ASUME EL COSTO DE TODAS LAS REVISIONES NECESARIAS, REPARACIONES O CORRECCIONES.