Contenido

Introducción

En este documento se encontrará una explicación detallada de cómo se desarrolló el examen para la Competencia 1 de la materia Tópicos Especializados de Desarrollo Web.

Para el desarrollo de este examen se utilizaron dos tecnologías con las que se estuvieron trabajando a lo largo de la competencia: Laravel y NodeJS.

Con NodeJS se solicita la creación de dos APIs, una para la creación y consulta de usuarios y otra para la consulta de órdenes de compra. Con Laravel se crean las vistas y sripts necesarios para consumir esas APIs y mostrar los resultados.

En el documento se detallarán las metodologías y pasos seguidos para la resolución del examen, así como evidencia de la ejecución de todo lo solicitado.

Desarrollo

1. Modificaciones en la base de datos

Para comenzar con la solución de este examen, primero fue necesario realizar algunas modificaciones a la base de datos “e-commerce” que se tiene creada en MongoDB.

Uno de los requisitos del examen es crear una API que permita registrar y consultar usuarios, por lo que fue necesario crear una colección donde se guarden los documentos con la información de los usuarios; el nombre de esta colección es “users”. También es necesario crear una API que permita consultar órdenes de compra y, al igual que con los usuarios, se creó una colección con el nombre de “orders”.

Como la API de las órdenes de compra es sólo para consultarlas, directamente se puede empezar a ingresar la información. En el examen se solicita que en la colección se ingresen por lo menos 1000 documentos, por lo que hacer esta tarea a mano sería muy pesado; es por eso por lo que se proporciona una herramienta llamada Mockaroo, la cual permite generar información de manera rápida.

Los campos que tienen los documentos de las órdenes de compra son los siguientes:

- order_no
- product_name
- created_at
- price
- status, con los valores “Approved” y “Not Approved”.

Esos mismos campos se ingresan en Mockaroo, y a cada uno se le dan los siguientes tipos, respectivamente:

- Row Number, para que lleve un conteo.
- Custom List, para ingresar una serie de opciones que se puedan seleccionar aleatoriamente. Aquí se pudo usar la opción “Product (Grocery)”, pero esa generaba productos de abarrotería, así que decidí usar una Custom List para ingresar productos más acordes al tipo de tienda que se está desarrollando.
- Datetime, para ingresar fechas con el formato dd/mm/yyyy.
- Number, para ingresar un número con decimales.
- Custom List, para ingresar los valores adecuados para este campo.

Una vez terminada la configuración de los campos, se cambia el formato a JSON para poder ingresarlos a la base de datos y se da clic en el botón “Generate Data”. Con esto se descarga un archivo .json, el cual se usará para importar los datos a la base de datos.

2. Proyecto NodeJS

Para esta sección del examen se utilizó como base la API que ya había sido desarrollada anteriormente en clase. Se realizaron algunos cambios en el archivo “app.js” para poder implementar correctamente el servicio web.

Se crearon 2 constantes llamadas “usersRoutes” y “ordersRoutes” en las cuales se almacena la ruta de los archivos que contienen las rutas para realizar las peticiones. Estas constantes se usan posteriormente dentro de la función .use() para poder realizar las peticiones.

Otro cambio importante fue la implementación de CORS. Aunque este no fue implementado durante la realización de este examen (se encargó a modo de práctica anteriormente), era necesario ya que podría haber algunos errores cuando se realicen las peticiones.

2.1. Creación de API REST para usuarios

Para esta API se creó un archivo llamado “usersRoutes.js” dentro de la carpeta “routes” del proyecto. Aquí se encuentran todas las rutas necesarias para realizar las peticiones.

Se creó una ruta para la petición GET, en la cual se realizará una consulta en la base de datos para poder obtener a todos los usuarios que se hayan registrado.

La otra ruta sirve para realizar la petición POST, mediante la cual se insertará un documento en la colección de la base de datos con los campos que se envíen mediante un formulario.

2.2. Creación de API REST para órdenes de compra

Al igual que la API anterior, para esta se creó un archivo llamado “ordersRoutes.js”.

En este archivo solamente se crea una ruta para realizar una petición GET, ya que solamente se solicita realizar una consulta de las órdenes de compra. Nuevamente, se realiza una consulta en la base de datos que permite obtener la información de las órdenes de compra.

3. Proyecto Laravel

Para esta sección del examen se realizaron algunas modificaciones como creación de vistas y configuración de rutas. Se sabe que las vistas son creadas en su respectiva carpeta y las rutas y funciones necesarias son configuradas en los archivos “web.php” y “SiteController.php”, pero esto se explicará más detalladamente en secciones posteriores.

3.1. Creación de formulario para registro de usuarios

Para la creación del formulario se creó una vista llamada “user-form.blade.php”. Se utilizó como base la plantilla “login.html” que contiene la carpeta que se proporcionó

al inicio del curso con toda una estructura para una tienda online. Se usó esa ya que tenía un formulario de registro ya realizado.

Los únicos cambios que se realizaron a ese formulario fueron:

- Añadir un id para su posterior uso en JavaScript.
- Añadir el atributo “method” para indicar como se va a realizar el envío de datos (POST).
- Cambiar algunos campos a los que se solicitan.
- Añadir el atributo “required” para hacer que el llenado de todos los campos sea obligatorio.

Una vez terminado de realizar la vista, se añade la función que retorna la vista en el archivo “SiteController.php”, y también se añade la ruta en el archivo “web.php”.

Con esto se termina la parte de la vista.

3.1.1. Script para consumo de servicio web de usuarios

Primero se creó un archivo llamado “api-users.js”. Dentro de este archivo se creó la función para hacer que todo el código se ejecute cuando el documento HTML esté totalmente cargado. Dentro de esa función se configura otra función que funciona como un manejador de eventos; con esta se indica que se realizarán distintas acciones una vez que ocurra el evento “submit”, es decir, el envío de formulario. Dentro de esa función se realizan las siguientes acciones:

- Se evita que el formulario se envíe de manera predeterminada, para así también evitar que la página sea recargada automáticamente.
- Se serializa el formulario, es decir, los datos que se ingresaron en los campos son ingresados en un arreglo.
- Los datos que se obtuvieron anteriormente son convertidos a un objeto JSON.
- Se realiza la petición AJAX, dentro de la cual se indica la URL de la API para poder insertar los datos. Se indica el tipo de solicitud HTTP que se va a realizar, que en este caso es POST para poder enviar los datos. Se indica también el tipo de contenido de los datos que se van a enviar, en este caso será un objeto JSON. Después, se indican los datos que se enviarán; aquí se utiliza una función que convierte el objeto JSON en una cadena JSON. Si hubo éxito con el registro del usuario, se envía un mensaje y se reinician los campos del formulario; si no, se manda un mensaje que muestra que hubo un error.

3.2. Creación de vista para la consulta de órdenes de compra

Para esta parte se creó una vista llamada “api-orders.blade.php”. Esta vista realmente es una copia de otros archivos que tienen la misma estructura, ya que se estuvieron realizando prácticas para la obtención de datos y manejo de DataTables. Los únicos cambios que se realizaron fueron a la tabla, se cambiaron los títulos de

los encabezados para que estuvieran más adecuados para los datos que se mostraron ahí. También se añadió un footer a la tabla para mostrar la suma de los precios. Los IDs de la tabla y el botón también fueron cambiados.

3.2.1. Script para consumo de servicio web de órdenes de compra

Primero se creó un archivo llamado “api-orders.js”. En este script se realiza toda la configuración necesaria para DataTables y la petición AJAX para la obtención de datos.

El archivo comienza con la creación de la función que hace que todo el código comience a ejecutarse una vez que el documento HTML haya sido completamente cargado. Dentro de esa función se crea otra que sirve como manejador de eventos, ahí se indica que cuando se dé clic al botón, se realizará la petición y se mostrarán los datos en la tabla. Ahora, dentro de esa función se realiza la configuración de DataTables:

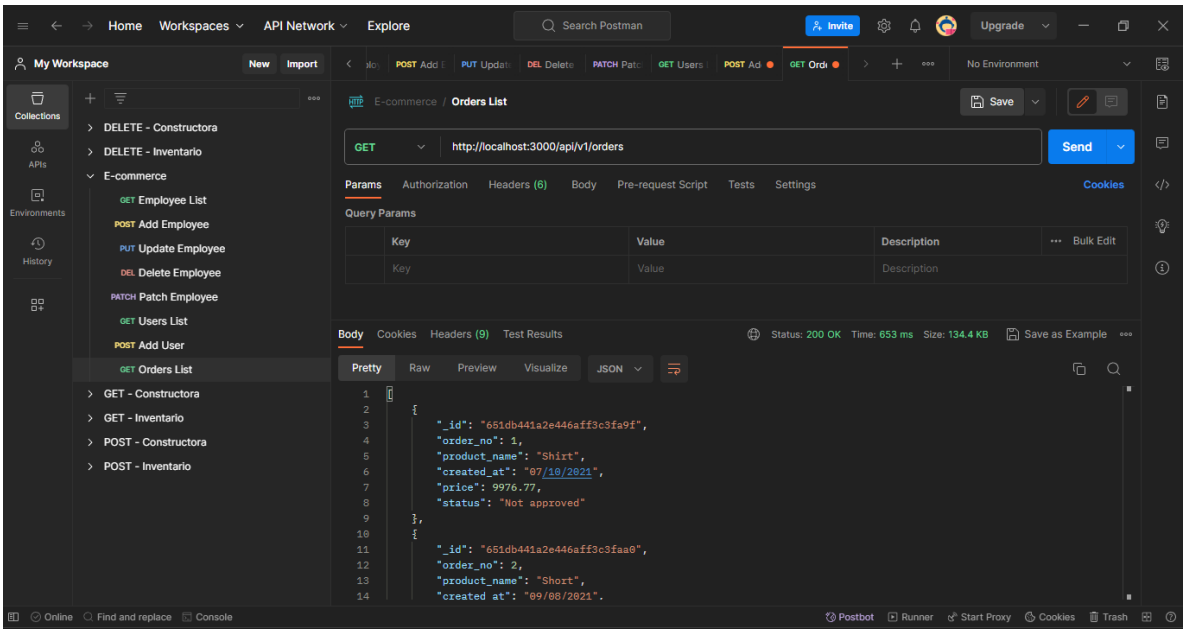
- Primero se realiza una limpieza de cualquier tabla existente que pudiera haber. Se borran todos los datos y la tabla.
- Después se configura la tabla para que adapte el formato de DataTable.
- Se realiza la petición AJAX, en la cual se indica la URL de la API y el tipo de solicitud HTTP que se está haciendo (GET).
- Una vez que DataTable termina de cargar y procesar los datos, se llama a la función footerCallback, la cual permite realizar cálculos con los datos de la tabla.
- La función footerCallback funciona de la siguiente manera:
    - Primero obtiene una referencia a la API de DataTable para trabajar con los datos de la tabla.
    - Se realiza una conversión del dato con el que se realizará la operación, que en este caso es el precio. Si el dato está almacenado como un String, se convierte a un número entero o decimal.
    - Se realiza la suma de los precios de todas las órdenes aprobadas (con el campo “state” con valor “Approved”). Esta suma se realiza tanto en todas las páginas como en la página actual.
    - Los totales se formatean con dos decimales.
    - Se muestra ese valor en el footer de la tabla.
- Terminada la función footerCallback, se especifica cómo deben de mapearse los datos que se reciben.
- Finalmente se crea una instancia de DataTable para que se cree la tabla.

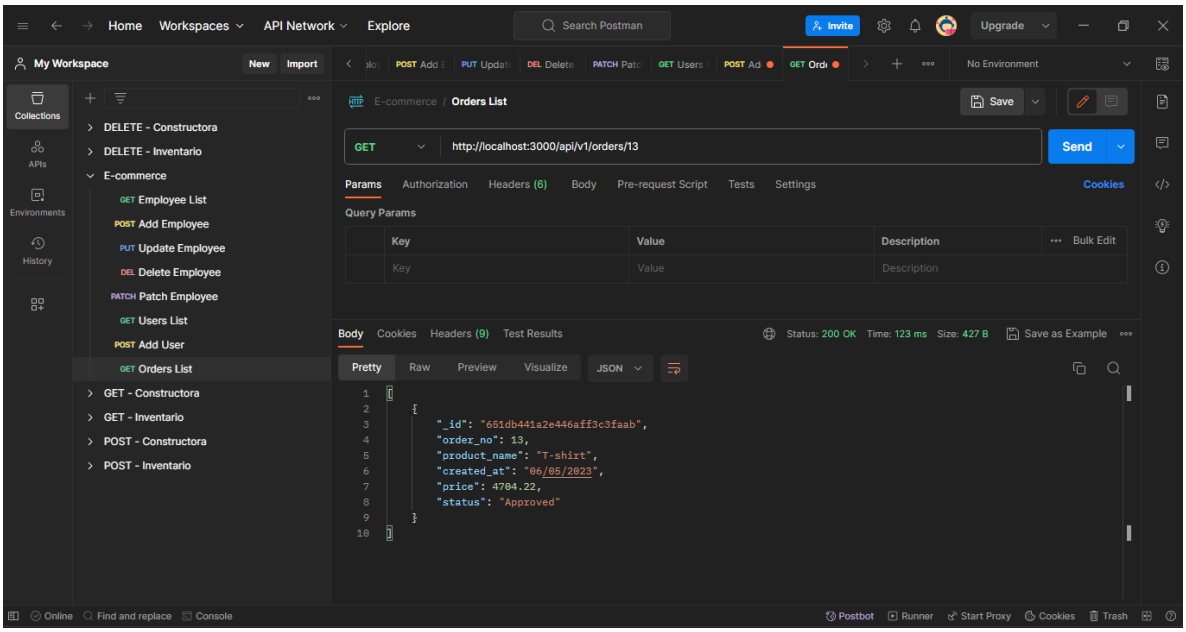
Resultados

Proyecto NodeJS

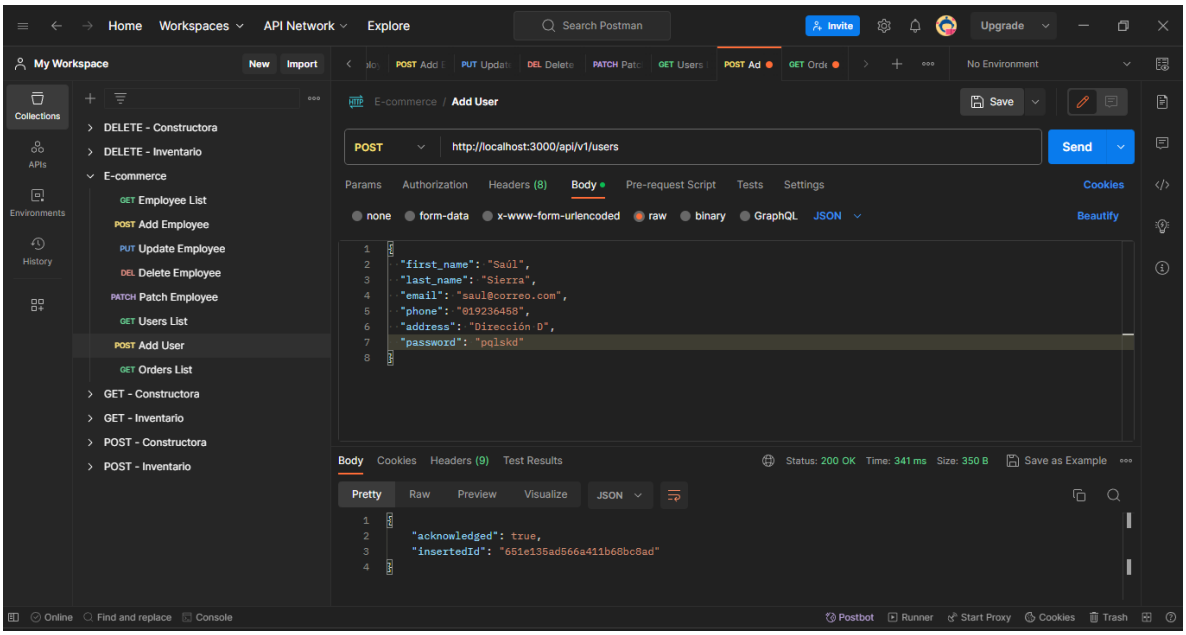
Para comprobar el correcto funcionamiento de la API se utilizará Postman.

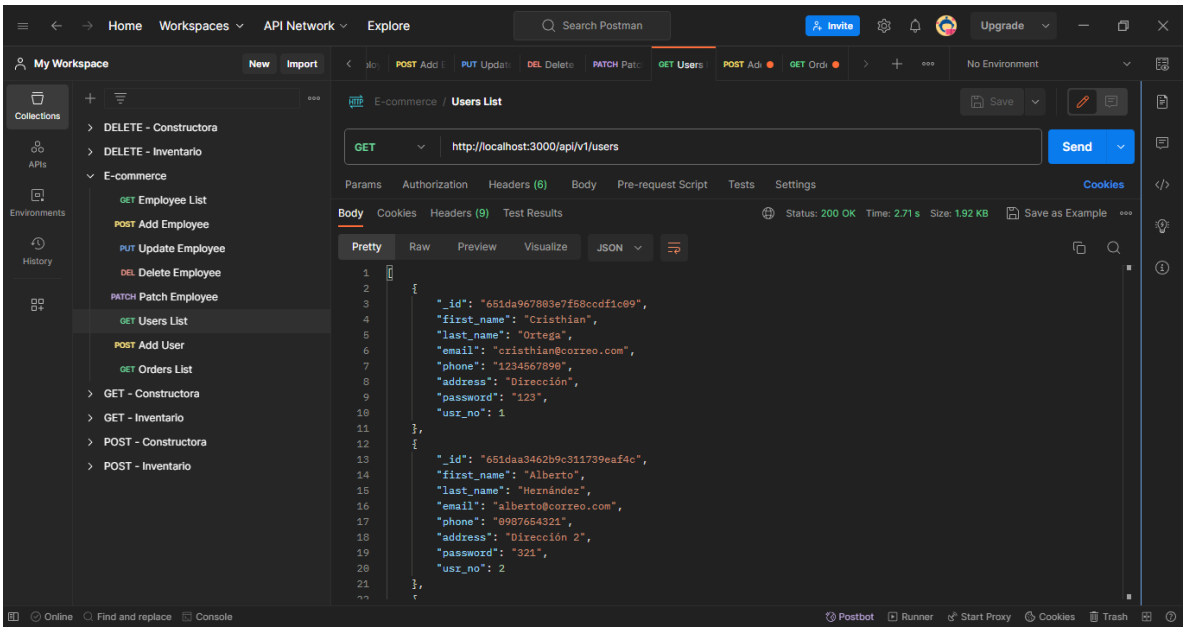
Comenzamos con la consulta de las órdenes de compra.

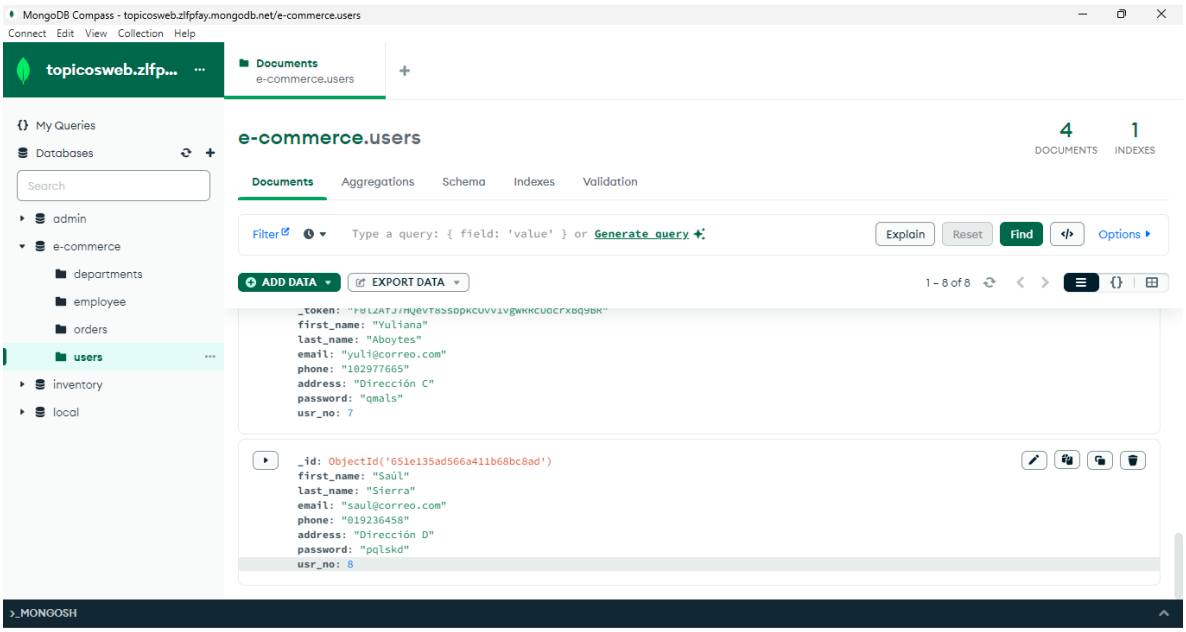




Ahora se muestra el funcionamiento de registros y consultas de usuarios.


Home
Workspaces
API Network
Explore
Search Postman
Invite
Upgrade
My Workspace
New
Import
No Environment
Collections
APIs
Environments
History
DELETE - Constructora
DELETE - Inventario
E-commerce
GET Employee List
POST Add Employee
PUT Update Employee
DEL Delete Employee
PATCH Patch Employee
GET Users List
POST Add User
GET Orders List
GET - Constructora
GET - Inventario
POST - Constructora
POST - Inventario
E-commerce / Add User
Save
POST
http://localhost:3000/api/v1/users
Send
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"first_name": "Saúl",
"last_name": "Sierra",
"email": "saul@correo.com",
"phone": "019236458",
"address": "Dirección D",
"password": "pqlskd"
}
Body
Cookies
Headers (9)
Test Results
Status: 200 OK
Time: 341 ms
Size: 350 B
Save as Example
Pretty
Raw
Preview
Visualize
JSON
{
"acknowledged": true,
"insertedId": "651e135ad566a411b68bc8ad"
}
Online
Find and replace
Console
Postbot
Runner
Start Proxy
Cookies
Trash

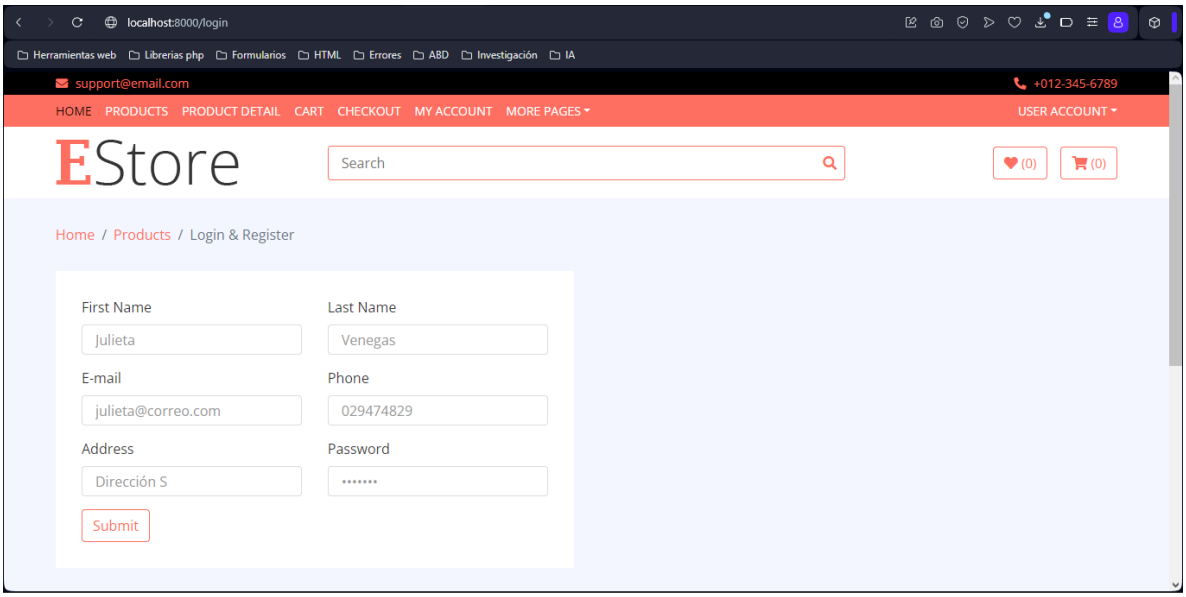

Home
Workspaces
API Network
Explore
Search Postman
Invite
Upgrade
My Workspace
New
Import
No Environment
Collections
APIs
Environments
History
DELETE - Constructora
DELETE - Inventario
E-commerce
GET Employee List
POST Add Employee
PUT Update Employee
DEL Delete Employee
PATCH Patch Employee
GET Users List
POST Add User
GET Orders List
GET - Constructora
GET - Inventario
POST - Constructora
POST - Inventario
E-commerce / Users List
Save
GET
http://localhost:3000/api/v1/users
Send
Params
Authorization
Headers (6)
Body
Pre-request Script
Tests
Settings
Cookies
Body
Cookies
Headers (9)
Test Results
Status: 200 OK
Time: 2.71 s
Size: 1.92 KB
Save as Example
Pretty
Raw
Preview
Visualize
JSON
[
{
"_id": "651da967883e7f58ccdf1c09",
"first_name": "Cristhian",
"last_name": "Ortega",
"email": "cristhian@correo.com",
"phone": "1234567890",
"address": "Dirección",
"password": "123",
"usr_no": 1
},
{
"_id": "651daa3462b9c311739eaf4c",
"first_name": "Alberto",
"last_name": "Hernández",
"email": "alberto@correo.com",
"phone": "0987654321",
"address": "Dirección 2",
"password": "321",
"usr_no": 2
},
Online
Find and replace
Console
Postbot
Runner
Start Proxy
Cookies
Trash



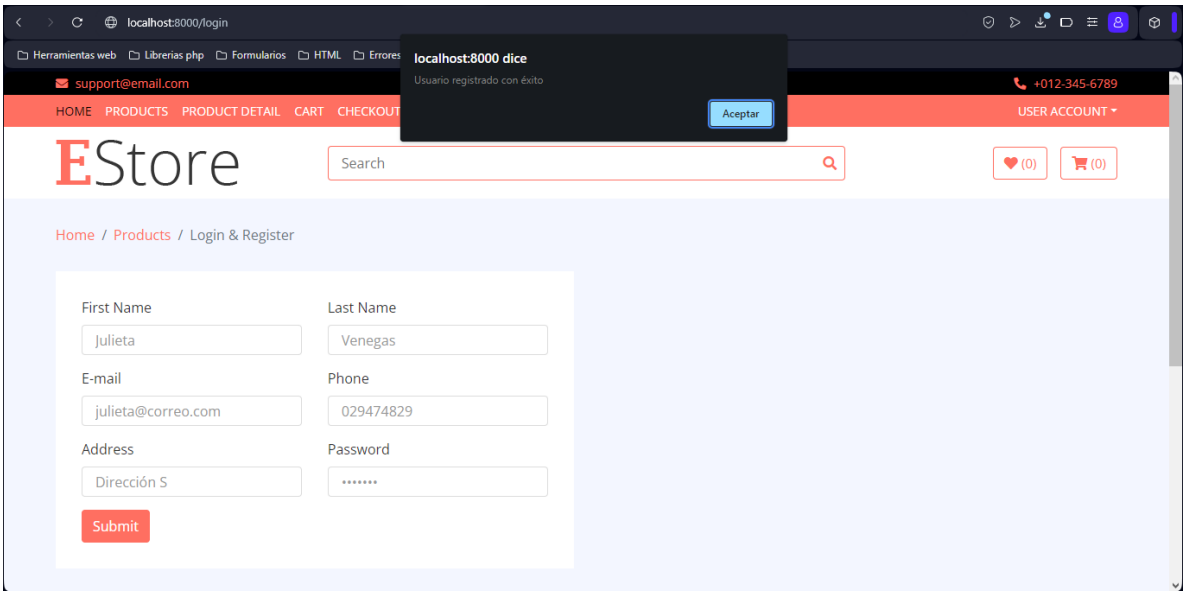
Proyecto Laravel

Registro de usuarios

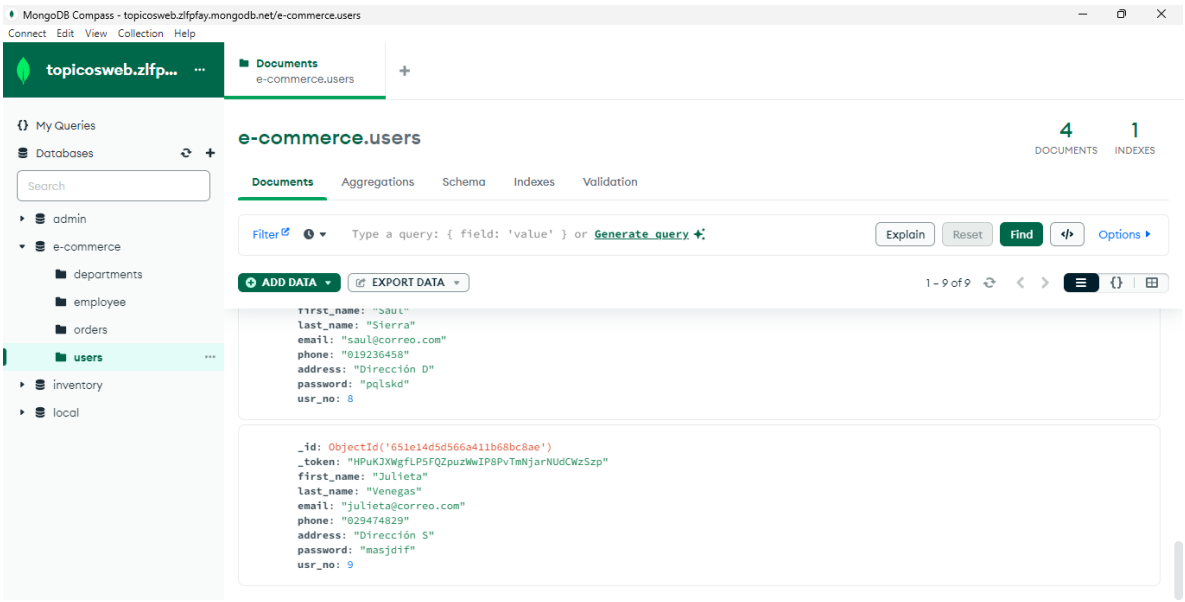
Se llenan los datos del formulario.



Al presionar el botón “Submit” se enviará un mensaje diciendo que el usuario se registró con éxito.

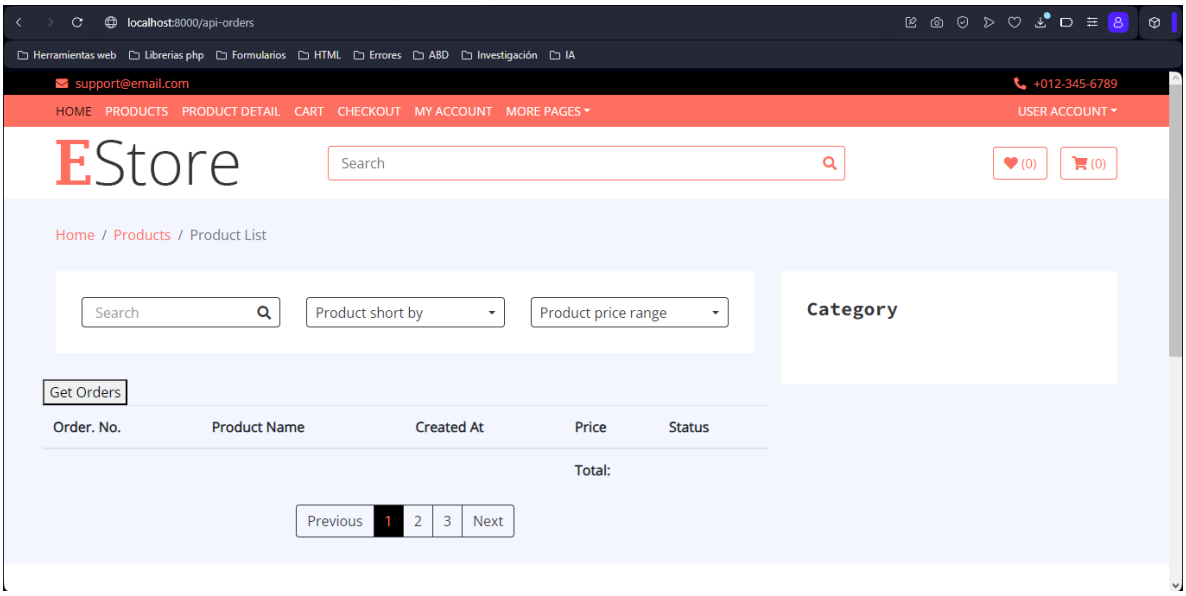


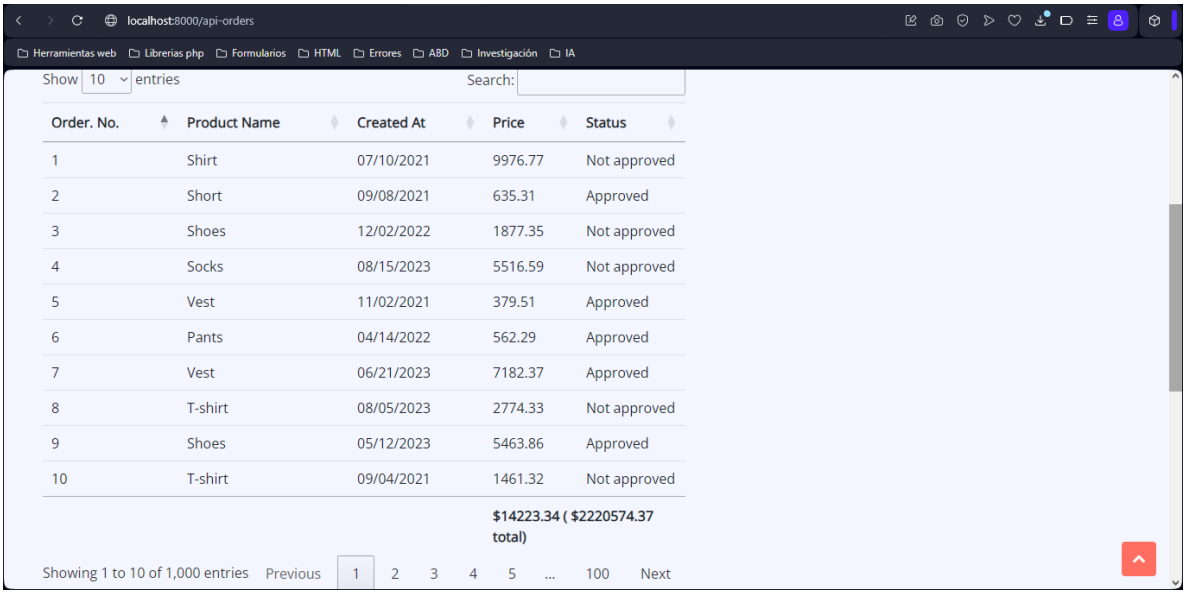
En la base de datos se puede comprobar que fue insertado correctamente el usuario.



Consulta de órdenes de compra

Para realizar la consulta de los datos, simplemente se presiona el botón.





Se puede observar que en el pie de la tabla se muestra la suma de los precios de las órdenes aprobadas, tanto de la página actual como de todas.

Conclusiones

Durante el desarrollo de este parcial estuvimos trabajando con diferentes tecnologías que son muy utilizadas en el desarrollo web. Utilizamos frameworks como Laravel para desarrollar una aplicación web y también usamos NodeJS para el desarrollo de APIs.

Estas tecnologías fueron nuevas para mí, pero con el transcurso de las clases fui comprendiéndolas mejor y su uso se fue facilitando. Para el desarrollo de este examen considero que tenía los conocimientos necesarios para saber por donde comenzar y como poder resolver los problemas que se plantearon.

Definitivamente fue un examen que puso aprueba mis habilidades, y me siento satisfecho con los resultados que se obtuvieron.

Referencias bibliográficas

DataTables example - Footer callback. (s. f.).

https://datatables.net/examples/advanced_init/footer_callback.html

Mockaroo - Random Data Generator and API Mocking Tool | JSON / CSV / SQL / Excel.

(s. f.). https://www.mockaroo.com/