

Manual Técnico

ExchangeHub



Juan Pablo Ruiz Marin, Geronimo Bustamante,
Julian Estiven Posso

Contenido

1. Introducción	2
2. Arquitectura del Sistema	2
2.1. Modelo Cliente-Servidor	2
2.2. Tecnologías Utilizadas	2
3. Instalación y Configuración	2
3.1. Requisitos Previos	2
3.2. Clonar el Repositorio	2
3.3. Configurar el Backend	3
3.4. Configurar el Frontend	3
4. Base de Datos	4
4.1. Modelo de Datos	4
5. API REST - Endpoints	9
6. Seguridad y Autenticación	11
7. Mantenimiento y Actualización	11
8. Despliegue en Producción	12
9. Soporte y Contacto	13

1. Introducción

Este manual proporciona información técnica sobre la arquitectura, tecnologías utilizadas, instalación, configuración y mantenimiento de **ExchangeHub**, una plataforma de intercambio de objetos sin uso de dinero.

2. Arquitectura del Sistema

2.1. Modelo Cliente-Servidor

- **Frontend:** React
- **Backend:** Node.js con Express
- **Base de Datos:** MySQL

2.2. Tecnologías Utilizadas

Componente	Tecnología
Frontend	React, Tailwindcss, React Router
Backend	Node.js, Express, Axios
Base de Datos	MySQL
Pagos	PayPal API (para membresía 'Elite Exchange')
Notificaciones	WebSockets con Socket.io
Mensajes	WebSockets con Socket.io
Despliegue	Vercel (Frontend), Render o DigitalOcean (Backend), Railway (Base de datos)

3. Instalación y Configuración

3.1. Requisitos Previos

- Node.js >= 18.x
- MySQL >= 8.0
- Git >= 2.30
- Un entorno de desarrollo (VS Code recomendado)

3.2. Clonar el Repositorio

3:

<https://github.com/Gero-Trujillo/exchangehub.git>

cd exchangehub

3.3. Configurar el Backend

1. Instalar dependencias

cd backend

npm install

2. Configurar variables de entorno (.env)

PORt=3000

DATABASE_URL=mysql://user:password@host:3306/exchangehubdb

PAYPAL_CLIENT_ID=

3. Ejecutar el servidor

npm run dev

3.4. Configurar el Frontend

1. Instalar dependencias

cd frontend

npm install

2. Ejecutar la aplicación

npm run dev

4. Base de Datos

4.1. Modelo de Datos

❖ Usuarios (`users`)

```
CREATE TABLE IF NOT EXISTS `users` (
    `idUser` int NOT NULL AUTO_INCREMENT,
    `name` varchar(50) NOT NULL,
    `lastname` varchar(50) NOT NULL,
    `email` varchar(100) NOT NULL,
    `address` varchar(255) DEFAULT NULL,
    `cellphone` varchar(15) DEFAULT NULL,
    `password` varchar(255) NOT NULL,
    `isPremium` tinyint(1) DEFAULT '0',
    `profileImageUrl` varchar(255) DEFAULT NULL,
    `created_at` datetime DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (`idUser`),
    UNIQUE KEY `email` (`email`)
)
```

❖ Mensajes (`messages`)

```
CREATE TABLE IF NOT EXISTS `messages` (
    `idMessage` int NOT NULL AUTO_INCREMENT,
    `idSender` int NOT NULL,
    `idReceiver` int NOT NULL,
    `text` text,
```

```

`image` varchar(255) DEFAULT NULL,
`sentAt` datetime DEFAULT CURRENT_TIMESTAMP,
`isSpecial` tinyint(1) DEFAULT '0',
`offerDetails` json DEFAULT NULL,
`read` tinyint(1) DEFAULT '0',
PRIMARY KEY (`idMessage`)
)

```

◆ Notificaciones (notifications)

```

CREATE TABLE notifications (
    idNotification INT AUTO_INCREMENT PRIMARY KEY,
   idUser INT NOT NULL,
    idSender INT NOT NULL,
    message VARCHAR(255) NOT NULL,
    isRead BOOLEAN DEFAULT FALSE,
    createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (idUser) REFERENCES users(idUser),
    FOREIGN KEY (idSender) REFERENCES users(idUser)
);

```

◆ Articulos (articles)

```

) CREATE TABLE IF NOT EXISTS `articles` (
    `idArticle` int NOT NULL AUTO_INCREMENT,
    `name` varchar(100) NOT NULL,
    `description` text,

```

6:

```
`category` varchar(50) DEFAULT NULL,  
 `idOwner` int DEFAULT NULL,  
 `imageUrl` text CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci,  
 `created_at` datetime DEFAULT CURRENT_TIMESTAMP,  
 PRIMARY KEY (`idArticle`),  
 KEY `idOwner` (`idOwner`),  
 CONSTRAINT `articles_ibfk_1` FOREIGN KEY (`idOwner`) REFERENCES  
 `users` (`idUser`) ON DELETE CASCADE
```

)

❖ Imágenes del artículo (articlesimages)

```
CREATE TABLE IF NOT EXISTS `articlesimages` (
```

```
 `idImage` int NOT NULL AUTO_INCREMENT,  
 `idArticle` int DEFAULT NULL,  
 `url` text NOT NULL,
```

```
 `is_main` tinyint(1) DEFAULT NULL,
```

```
 PRIMARY KEY (`idImage`),
```

```
 KEY `idArticle` (`idArticle`),
```

```
 CONSTRAINT `articlesimages_ibfk_1` FOREIGN KEY (`idArticle`)  
 REFERENCES `articles` (`idArticle`) ON DELETE CASCADE
```

❖ Cambios (exchanges)

```
CREATE TABLE IF NOT EXISTS `exchanges` (
```

```
 `idExchange` int NOT NULL AUTO_INCREMENT,
```

```
 `idArticleOne` int DEFAULT NULL,
```

```

`idArticleTwo` int DEFAULT NULL,
`idUserOne` int DEFAULT NULL,
`idUserTwo` int DEFAULT NULL,
`exchangeDate` datetime DEFAULT CURRENT_TIMESTAMP,
`status` enum('pending','accepted','rejected') DEFAULT 'pending',
PRIMARY KEY (`idExchange`),
KEY `idArticleOne` (`idArticleOne`),
KEY `idArticleTwo` (`idArticleTwo`),
KEY `idUserOne` (`idUserOne`),
KEY `idUserTwo` (`idUserTwo`),
CONSTRAINT `exchanges_ibfk_1` FOREIGN KEY (`idArticleOne`)
REFERENCES `articles` (`idArticle`) ON DELETE CASCADE,
CONSTRAINT `exchanges_ibfk_2` FOREIGN KEY (`idArticleTwo`)
REFERENCES `articles` (`idArticle`) ON DELETE CASCADE,
CONSTRAINT `exchanges_ibfk_3` FOREIGN KEY (`idUserOne`)
REFERENCES `users` (`idUser`) ON DELETE CASCADE,
CONSTRAINT `exchanges_ibfk_4` FOREIGN KEY (`idUserTwo`)
REFERENCES `users` (`idUser`) ON DELETE CASCADE
)

```

➡ Cambios (exchanges)

```

CREATE TABLE IF NOT EXISTS `exchanges` (
`idExchange` int NOT NULL AUTO_INCREMENT,
`idArticleOne` int DEFAULT NULL,
`idArticleTwo` int DEFAULT NULL,
`idUserOne` int DEFAULT NULL,

```

```

`idUserTwo` int DEFAULT NULL,
`exchangeDate` datetime DEFAULT CURRENT_TIMESTAMP,
`status` enum('pending','accepted','rejected') DEFAULT 'pending',
PRIMARY KEY (`idExchange`),
KEY `idUserOne` (`idUserOne`),
KEY `idUserTwo` (`idUserTwo`),
CONSTRAINT `exchanges_ibfk_1` FOREIGN KEY (`idUserOne`)
REFERENCES `articles` (`idArticle`) ON DELETE CASCADE,
CONSTRAINT `exchanges_ibfk_2` FOREIGN KEY (`idUserTwo`)
REFERENCES `articles` (`idArticle`) ON DELETE CASCADE,
CONSTRAINT `exchanges_ibfk_3` FOREIGN KEY (`idUserOne`) REFERENCES
`users` (`idUser`) ON DELETE CASCADE,
CONSTRAINT `exchanges_ibfk_4` FOREIGN KEY (`idUserTwo`) REFERENCES
`users` (`idUser`) ON DELETE CASCADE
)

```

◆ Calificación (ratings)

```

CREATE TABLE IF NOT EXISTS `ratings` (
`idRating` int NOT NULL AUTO_INCREMENT,
`idUser` int DEFAULT NULL,
`rating` int DEFAULT NULL,
`created_at` datetime DEFAULT CURRENT_TIMESTAMP,
PRIMARY KEY (`idRating`),
KEY `idUser` (`idUser`),

```

```

CONSTRAINT `ratings_ibfk_1` FOREIGN KEY (`idUser`) REFERENCES `users`(`idUser`)
ON DELETE CASCADE,
CONSTRAINT `ratings_chk_1` CHECK ((`rating` between 1 and 5))
)

```

❖ Subscripciones (subscriptions)

```
CREATE TABLE IF NOT EXISTS `subscriptions` (
```

```
 `idSubscription` int NOT NULL AUTO_INCREMENT,
```

```
 `idUser` int DEFAULT NULL,
```

```
 `startDate` datetime NOT NULL,
```

```
 `endDate` datetime NOT NULL,
```

```
 `amount` decimal(10,2) NOT NULL,
```

```
 `status` enum('active','expired') DEFAULT 'active',
```

```
 PRIMARY KEY (`idSubscription`),
```

```
 KEY `idUser` (`idUser`),
```

```

CONSTRAINT `subscriptions_ibfk_1` FOREIGN KEY (`idUser`) REFERENCES `users`(`idUser`)
ON DELETE CASCADE
)
```

5. API REST - Endpoints

❖ Autenticación

- POST /api/auth/register → Registrar usuario
- POST /api/auth/login → Iniciar sesión
- GET /api/verify → Verificar Token
- POST /api/logout → Cerrar Sesión
- PATCH /api/confirm/:idUser → Confirmar cuenta

❖ **Productos**

- GET /api/articles → Obtener todos los productos
- POST /api/articles → Crear un nuevo producto
- PUT /api/articles/:id → Editar un producto
- DELETE /api/articles/:id → Eliminar producto

❖ **Intercambios**

- POST /api/exchanges → Crear solicitud de intercambio
- POST /api/exchanges/sendNotification → Enviar notificación vía correo cuando envían una oferta a un Usuario
- GET /api/trades/:ser/:Id → Ver intercambios por usuario pendientes
- GET /api/exchanges/status/:status → Ver estado del intercambio
- PATCH /api/exchanges/:id → Actualizar intercambio
- PATCH /api/exchanges/cancel/:id → Cancelar intercambio
- GET /api/exchanges/articles/:idProductoOne/:idProductoTwo → Ver intercambio por productos

❖ **Mensajes**

- GET /api/messages/users/:id → Ver los usuarios en la barra lateral del frontend
- GET /api/messages/:idUser/:myId → Ver mensajes enviados
- POST /api/messages/send/:receiverId/:senderId → Enviar mensaje
- PATCH /api/messages/:idMessage → Cambiar estado de mensaje de oferta

❖ **Notificaciones**

- GET /api/notifications/:id → Ver las notificaciones por usuario
- POST /api/notifications → Crear notificaciones
- PATCH /api/notifications/:id → Actualizar estado de lectura del mensaje

❖ **Pagos con PayPal**

- POST /api/payments/verify → Procesar pago
- GET /api/payments → Ver los pagos

❖ **Calificaciones**

- POST /api/ratings → Agregar calificación
- GET /api/ratings/:idUser → Ver las calificaciones del usuario

(Se recomienda usar Postman para probar estos endpoints)

6. Seguridad y Autenticación

❖ Autenticación con JSON Web Tokens (JWT)

- Se usa JWT para manejar sesiones en el backend.
- Tokens firmados con una clave secreta almacenada en variables de entorno.

❖ Protección de Rutas en el Backend

```
// Importacion de SECRET_KEY y jwt
import { SECRET_KEY } from "../libs/jwt.js";
import jwt from "jsonwebtoken";

// Middleware para validar el token de acceso
export const authRequired = async (req, res, next) => {
    // Obtenemos el token de acceso de las cookies
    const { accessToken } = req.cookies;
    // Si no existe el token de acceso, retornamos un error
    if (!accessToken) {
        return res.status(401).json({ message: "Unauthorized" });
    }
    try {
        // Verificamos el token de acceso
        jwt.verify(accessToken, SECRET_KEY, (err, user) => {
            if (err) {
                return res.status(401).json({ message: "Unauthorized" });
            }
            // Continuamos con la ejecucion del codigo
            next();
        });
    } catch (err) {
        // Si ocurre un error, retornamos un error
        return res.status(401).json({ message: "Unauthorized" });
    }
};
```

7. Mantenimiento y Actualización

❖ Respaldo de Base de Datos

- Se recomienda usar `mysqldump` para exportar los datos semanalmente:

```
mysqldump -u root -p exchangehub > backup.sql
```

📌 Actualización del Sistema

1. Pull de cambios:

```
git pull origin main
```

2. Reinstalar dependencias (si es necesario):

```
npm install
```

3. Reiniciar el servidor:

```
pm2 restart all
```

📌 Manejo de Errores y Logs

- Los errores del backend se almacenan en un archivo `logs/error.log`.
- Se recomienda usar herramientas como **Sentry** para monitoreo.

8. Despliegue en Producción

📌 Frontend en Vercel

```
vercel deploy
```

📌 Backend en DigitalOcean / Render

```
git push origin main
```

📌 Configuración de Variables de Entorno

En el servidor de producción, configurar `.env` con credenciales de **MySQL** y **PayPal**.

📌 Dominio y Certificado SSL

- Se recomienda usar **Cloudflare** para seguridad y optimización.

9. Soporte y Contacto

- ❖ Email: exchagehub00@gmail.com
- ❖ Documentación técnica: <https://github.com/Gero-Trujillo/exchagehub>

❖ Con este manual, cualquier desarrollador podrá instalar, configurar, mantener y escalar ExchangeHub sin problemas.

