Estoy desarrollando un sistema completo de hardware y software que cumpla con los siguientes requisitos. Quiero que me generes todo el código completo, paso a paso y sin omitir ningún archivo, para que funcione correctamente en producción. Trabajo en **Windows 11**, y usaré **terminales tipo Git Bash o CMD**.

**✅ Requisitos del proyecto**

**1. Hardware**

* Microcontrolador: **ESP8266** o **ESP32**
* Funcionalidad: controlar LEDs (encender/apagar) y enviar su estado en tiempo real
* Comunicación: usar **WebSocket** para enviar y recibir actualizaciones constantemente con la app web
* Código completo del sketch de Arduino (con WebSocket)

**2. Backend**

* Lenguaje: **PHP 8**
* Base de datos: **SQLite**
* Funcionalidades:
  + CRUD de usuarios
  + Login de usuario
  + Control de acceso con **rol de administrador y rol de usuario común**
  + API para recibir el estado de los LEDs y devolver respuestas vía WebSocket
  + Endpoints para encender/apagar LEDs (simulados desde el servidor por ahora)

**3. Frontend**

* HTML5 + **Bootstrap 5**
* Mostrar botones para **Encender/Apagar** LEDs según su estado (botón activo o deshabilitado)
* Interfaz responsive y visualmente clara
* Conexión en tiempo real con WebSocket para mostrar el estado actualizado de los LEDs

**4. Despliegue**

* Subir todo a **GitHub**
* Hacer deploy del backend PHP + frontend HTML + SQLite en **Render.com** (con instrucciones paso a paso)
* Incluir configuraciones necesarias para SQLite en producción
* Adaptar el proyecto para **Render Free Tier**

**🎯 Objetivo**

Quiero que me generes:

* Sketch de Arduino completo (ESP8266 o ESP32) con WebSocket funcional
* Backend PHP con SQLite (incluye archivo index.php, rutas, controladores, conexión a DB, scripts de creación de tablas)
* Interfaz web HTML + Bootstrap conectada por WebSocket al backend
* Archivos de configuración para despliegue en Render (como render.yaml, .htaccess, etc.)
* Instrucciones paso a paso para:
  + Configurar el proyecto en local en **Windows 11**
  + Probar WebSocket con ESP
  + Subir a GitHub
  + Hacer deploy en Render
* Todo debe ser funcional desde el inicio y preparado para subir a **producción**.
* **📁 Estructura general del proyecto**
* perl
* CopiarEditar
* led-control-system/
* │
* ├── arduino/
* │ └── esp\_led\_websocket.ino # Sketch para ESP8266 o ESP32
* │
* ├── backend/
* │ ├── public/
* │ │ ├── index.php # Punto de entrada principal
* │ │ ├── ws-server.php # Servidor WebSocket en PHP
* │ │ └── .htaccess # Redirección para producción en Render
* │ │
* │ ├── src/
* │ │ ├── config/
* │ │ │ └── database.php # Conexión a SQLite
* │ │ ├── controllers/
* │ │ │ ├── AuthController.php # Login/registro
* │ │ │ └── LedController.php # Encender/apagar LEDs
* │ │ ├── models/
* │ │ │ └── User.php # Modelo de usuario
* │ │ └── routes/
* │ │ └── api.php # Definición de rutas
* │ │
* │ └── database/
* │ ├── db.sqlite # Base de datos SQLite
* │ └── seed.sql # Script opcional para poblar
* │
* ├── frontend/
* │ ├── index.html # Interfaz principal
* │ ├── js/
* │ │ └── app.js # Lógica JS con WebSocket
* │ └── css/
* │ └── style.css # Estilos opcionales
* │
* ├── .gitignore
* ├── render.yaml # Configuración para Render
* └── README.md # Documentación del proyecto

**🔍 Explicación por secciones**

**🔌 arduino/**

Aquí irá el sketch completo de Arduino (ESP8266 o ESP32), con WebSocket funcionando para enviar el estado de los LEDs y recibir comandos (encender/apagar).

**🔙 backend/**

Aplicación PHP con SQLite:

* Gestión de usuarios con login
* Roles: administrador / usuario común
* Encendido/apagado de LEDs
* Servidor WebSocket en PHP para mantener la comunicación en tiempo real

**🎨 frontend/**

HTML5 + Bootstrap:

* Página visual que muestra el estado actual de los LEDs
* Botones dinámicos que cambian según el estado del LED
* Comunicación con WebSocket

**⚙️ Archivos clave**

* .gitignore: excluir db.sqlite, carpetas de cache, etc.
* render.yaml: para configurar el build y start command en Render.
* README.md: documentación para clonar, instalar y desplegar.

**✅ 1. Sketch WebSocket — ESP8266**

cpp

CopiarEditar

#include <ESP8266WiFi.h>

#include <WebSocketsClient.h>

const char\* ssid = "TU\_SSID";

const char\* password = "TU\_PASSWORD";

// Dirección IP o dominio de tu backend (Render, etc.)

const char\* websocket\_server = "example.onrender.com";

const uint16\_t websocket\_port = 80;

const char\* websocket\_path = "/ws-server.php"; // ruta en backend PHP

WebSocketsClient webSocket;

const int ledPin = D5;

void webSocketEvent(WStype\_t type, uint8\_t\* payload, size\_t length) {

switch (type) {

case WStype\_TEXT:

Serial.printf("[WebSocket] Mensaje recibido: %s\n", payload);

if (strcmp((char\*)payload, "ON") == 0) {

digitalWrite(ledPin, HIGH);

} else if (strcmp((char\*)payload, "OFF") == 0) {

digitalWrite(ledPin, LOW);

}

break;

}

}

void setup() {

Serial.begin(115200);

pinMode(ledPin, OUTPUT);

digitalWrite(ledPin, LOW);

WiFi.begin(ssid, password);

Serial.println("Conectando a WiFi...");

while (WiFi.status() != WL\_CONNECTED) {

delay(500);

Serial.print(".");

}

Serial.println("\nConectado a WiFi");

webSocket.begin(websocket\_server, websocket\_port, websocket\_path);

webSocket.onEvent(webSocketEvent);

webSocket.setReconnectInterval(5000); // reconexión automática

}

void loop() {

webSocket.loop();

}

**✅ 2. Sketch WebSocket — ESP32**

cpp

CopiarEditar

#include <WiFi.h>

#include <WebSocketsClient.h>

const char\* ssid = "TU\_SSID";

const char\* password = "TU\_PASSWORD";

const char\* websocket\_server = "example.onrender.com";

const uint16\_t websocket\_port = 80;

const char\* websocket\_path = "/ws-server.php";

WebSocketsClient webSocket;

const int ledPin = 18;

void webSocketEvent(WStype\_t type, uint8\_t\* payload, size\_t length) {

switch (type) {

case WStype\_TEXT:

Serial.printf("[WebSocket] Mensaje recibido: %s\n", payload);

if (strcmp((char\*)payload, "ON") == 0) {

digitalWrite(ledPin, HIGH);

} else if (strcmp((char\*)payload, "OFF") == 0) {

digitalWrite(ledPin, LOW);

}

break;

}

}

void setup() {

Serial.begin(115200);

pinMode(ledPin, OUTPUT);

digitalWrite(ledPin, LOW);

WiFi.begin(ssid, password);

Serial.println("Conectando a WiFi...");

while (WiFi.status() != WL\_CONNECTED) {

delay(500);

Serial.print(".");

}

Serial.println("\nConectado a WiFi");

webSocket.begin(websocket\_server, websocket\_port, websocket\_path);

webSocket.onEvent(webSocketEvent);

webSocket.setReconnectInterval(5000);

}

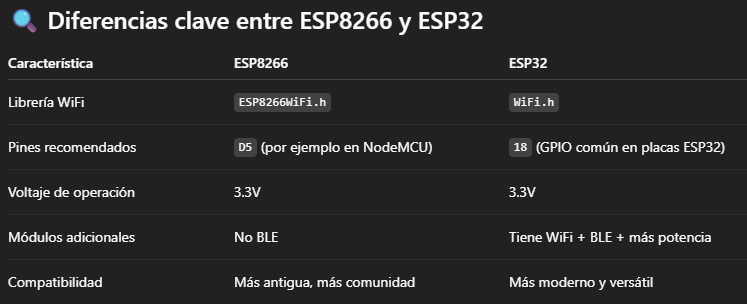
void loop() {

webSocket.loop();

}

**🔍 Diferencias clave entre ESP8266 y ESP32**

| **Característica** | **ESP8266** | **ESP32** |
| --- | --- | --- |
| Librería WiFi | ESP8266WiFi.h | WiFi.h |
| Pines recomendados | D5 (por ejemplo en NodeMCU) | 18 (GPIO común en placas ESP32) |
| Voltaje de operación | 3.3V | 3.3V |
| Módulos adicionales | No BLE | Tiene WiFi + BLE + más potencia |
| Compatibilidad | Más antigua, más comunidad | Más moderno y versátil |



**🔒 PASO 1: Backend (PHP + SQLite) con login y roles**

**✅ Funcionalidades que cubriremos:**

* Registro y login de usuarios
* Control de acceso por rol: admin y user
* Conexión a SQLite
* Estructura MVC básica en PHP puro
* API RESTful simple para login y gestión de usuarios

**📁 Estructura del backend**

pgsql

CopiarEditar

backend/

├── public/

│ ├── index.php

│ └── .htaccess

│

├── src/

│ ├── config/

│ │ └── database.php

│ ├── controllers/

│ │ ├── AuthController.php

│ │ └── UserController.php

│ ├── models/

│ │ └── User.php

│ └── routes/

│ └── api.php

│

├── database/

│ ├── db.sqlite

│ └── seed.sql

**🧠 1. Base de datos SQLite**

**📄 database/seed.sql**

sql

CopiarEditar

CREATE TABLE IF NOT EXISTS users (

id INTEGER PRIMARY KEY AUTOINCREMENT,

username TEXT NOT NULL UNIQUE,

password TEXT NOT NULL,

role TEXT NOT NULL CHECK(role IN ('admin', 'user'))

);

INSERT INTO users (username, password, role) VALUES

('admin', '$2y$10$e0MYzXyjpJS7Pd0RVvHwHeFQ0W8Hkjg0cfGqXwGJw5Duz8Y8Y8FZC', 'admin');

-- password: admin123

**🛠 Crear base de datos**

Ejecuta desde terminal:

bash

CopiarEditar

sqlite3 database/db.sqlite < database/seed.sql

**🛠 2. Conexión a SQLite**

**📄 src/config/database.php**

php

CopiarEditar

<?php

function connectDB() {

$dbPath = \_\_DIR\_\_ . '/../../database/db.sqlite';

try {

$pdo = new PDO("sqlite:" . $dbPath);

$pdo->setAttribute(PDO::ATTR\_ERRMODE, PDO::ERRMODE\_EXCEPTION);

return $pdo;

} catch (PDOException $e) {

die("Error de conexión: " . $e->getMessage());

}

}

**👤 3. Modelo de Usuario**

**📄 src/models/User.php**

php

CopiarEditar

<?php

require\_once \_\_DIR\_\_ . '/../config/database.php';

class User {

private $db;

public function \_\_construct() {

$this->db = connectDB();

}

public function findByUsername($username) {

$stmt = $this->db->prepare("SELECT \* FROM users WHERE username = ?");

$stmt->execute([$username]);

return $stmt->fetch(PDO::FETCH\_ASSOC);

}

public function create($username, $password, $role = 'user') {

$hashed = password\_hash($password, PASSWORD\_DEFAULT);

$stmt = $this->db->prepare("INSERT INTO users (username, password, role) VALUES (?, ?, ?)");

return $stmt->execute([$username, $hashed, $role]);

}

}

**🔐 4. Controlador de Autenticación**

**📄 src/controllers/AuthController.php**

php

CopiarEditar

<?php

require\_once \_\_DIR\_\_ . '/../models/User.php';

class AuthController {

public function login($data) {

$userModel = new User();

$user = $userModel->findByUsername($data['username']);

if ($user && password\_verify($data['password'], $user['password'])) {

return json\_encode([

"status" => "success",

"user" => [

"id" => $user['id'],

"username" => $user['username'],

"role" => $user['role']

]

]);

} else {

http\_response\_code(401);

return json\_encode(["status" => "error", "message" => "Credenciales incorrectas"]);

}

}

public function register($data) {

$userModel = new User();

$success = $userModel->create($data['username'], $data['password'], $data['role'] ?? 'user');

return json\_encode(["status" => $success ? "success" : "error"]);

}

}

**🌐 5. Rutas**

**📄 src/routes/api.php**

php

CopiarEditar

<?php

require\_once \_\_DIR\_\_ . '/../controllers/AuthController.php';

$uri = $\_SERVER['REQUEST\_URI'];

$method = $\_SERVER['REQUEST\_METHOD'];

$input = json\_decode(file\_get\_contents("php://input"), true);

$auth = new AuthController();

if ($uri === '/login' && $method === 'POST') {

echo $auth->login($input);

} elseif ($uri === '/register' && $method === 'POST') {

echo $auth->register($input);

} else {

http\_response\_code(404);

echo json\_encode(["status" => "error", "message" => "Ruta no encontrada"]);

}

**🚪 6. Punto de entrada**

**📄 public/index.php**

php

CopiarEditar

<?php

header("Access-Control-Allow-Origin: \*");

header("Access-Control-Allow-Headers: Content-Type");

header("Content-Type: application/json");

require\_once \_\_DIR\_\_ . '/../src/routes/api.php';

**📄 .htaccess (para Render)**

apache

CopiarEditar

RewriteEngine On

RewriteCond %{REQUEST\_FILENAME} !-f

RewriteRule ^ index.php [QSA,L]

**✅ Prueba desde terminal (opcional)**

bash

CopiarEditar

curl -X POST http://localhost/login -H "Content-Type: application/json" -d '{"username":"admin","password":"admin123"}'

**🔄 Paso 2: Servidor WebSocket en PHP (ws-server.php)**

**📋 ¿Qué haremos?**

* Crear un servidor WebSocket en PHP que maneje conexiones simultáneas
* Permitir recibir mensajes para encender/apagar LEDs
* Notificar a todos los clientes conectados el estado actual de los LEDs
* Control básico para que el backend controle y comparta el estado en tiempo real

**🛠 Herramienta recomendada**

Para PHP no existe WebSocket nativo, pero usaremos la biblioteca **Ratchet** que es estándar para WebSocket en PHP.

**1. Instalación de Ratchet**

Asumiendo que tienes **Composer** instalado en Windows 11:

Abre la terminal en la carpeta backend y ejecuta:

bash

CopiarEditar

composer require cboden/ratchet

Esto creará la carpeta vendor/ y descargará Ratchet y dependencias.

**2. Código del servidor WebSocket**

**Crear archivo backend/public/ws-server.php**

php

CopiarEditar

<?php

require dirname(\_\_DIR\_\_) . '/vendor/autoload.php';

use Ratchet\MessageComponentInterface;

use Ratchet\ConnectionInterface;

class LedWebSocketServer implements MessageComponentInterface {

protected $clients;

protected $ledState; // true = encendido, false = apagado

public function \_\_construct() {

$this->clients = new \SplObjectStorage;

$this->ledState = false;

echo "Servidor WebSocket iniciado\n";

}

public function onOpen(ConnectionInterface $conn) {

$this->clients->attach($conn);

echo "Nueva conexión: ({$conn->resourceId})\n";

// Enviar estado actual del LED al nuevo cliente

$conn->send(json\_encode([

'type' => 'status',

'led' => $this->ledState ? 'ON' : 'OFF'

]));

}

public function onMessage(ConnectionInterface $from, $msg) {

echo "Mensaje recibido de {$from->resourceId}: $msg\n";

$data = json\_decode($msg, true);

if (!$data) return;

if (isset($data['command'])) {

if ($data['command'] === 'toggle') {

$this->ledState = !$this->ledState;

// Enviar a todos clientes el nuevo estado

foreach ($this->clients as $client) {

$client->send(json\_encode([

'type' => 'status',

'led' => $this->ledState ? 'ON' : 'OFF'

]));

}

}

}

}

public function onClose(ConnectionInterface $conn) {

$this->clients->detach($conn);

echo "Conexión cerrada: ({$conn->resourceId})\n";

}

public function onError(ConnectionInterface $conn, \Exception $e) {

echo "Error: {$e->getMessage()}\n";

$conn->close();

}

}

// Configuración del servidor WebSocket

use Ratchet\Server\IoServer;

use Ratchet\Http\HttpServer;

use Ratchet\WebSocket\WsServer;

$port = 8080; // puerto WebSocket

$server = IoServer::factory(

new HttpServer(

new WsServer(

new LedWebSocketServer()

)

),

$port

);

echo "Servidor WebSocket corriendo en ws://localhost:$port\n";

$server->run();

**3. Ejecutar servidor WebSocket**

En tu terminal, en la carpeta backend:

bash

CopiarEditar

php public/ws-server.php

Verás mensajes en consola indicando conexiones y mensajes.

**4. Configuración para producción**

* En **Render**, deberás desplegar este proceso como un servicio background o worker para que el servidor WebSocket esté corriendo.
* El puerto puede ser diferente, Render asigna uno con la variable de entorno $PORT. Más adelante te doy ejemplo para render.yaml.

**5. Resumen del protocolo**

* Clientes se conectan y reciben el estado actual (ON/OFF).
* Cliente envía mensaje JSON: {"command": "toggle"} para cambiar estado.
* Servidor actualiza estado y lo notifica a todos.

Claro, aquí tienes un script JavaScript sencillo para probar la conexión con tu servidor WebSocket PHP (ws-server.php). Este script se conecta, recibe el estado actual del LED y permite enviar el comando para alternar (toggle) el LED.

**Script JS de prueba WebSocket**

html

CopiarEditar

<!DOCTYPE html>

<html lang="es">

<head>

<meta charset="UTF-8" />

<title>Prueba WebSocket LED</title>

</head>

<body>

<h2>Estado LED: <span id="led-status">Desconocido</span></h2>

<button id="toggle-btn" disabled>Conectar WebSocket</button>

<script>

let ws;

const ledStatus = document.getElementById('led-status');

const toggleBtn = document.getElementById('toggle-btn');

function initWebSocket() {

// Cambia localhost y puerto según donde tengas corriendo el servidor

ws = new WebSocket('ws://localhost:8080');

ws.onopen = () => {

console.log('Conectado al WebSocket');

toggleBtn.disabled = false;

toggleBtn.textContent = 'Alternar LED';

};

ws.onmessage = (event) => {

console.log('Mensaje recibido:', event.data);

try {

const data = JSON.parse(event.data);

if (data.type === 'status') {

ledStatus.textContent = data.led;

toggleBtn.disabled = false;

}

} catch (err) {

console.error('Error parseando mensaje:', err);

}

};

ws.onclose = () => {

console.log('Desconectado del WebSocket');

ledStatus.textContent = 'Desconectado';

toggleBtn.disabled = true;

toggleBtn.textContent = 'Reconectar';

};

ws.onerror = (err) => {

console.error('Error WebSocket:', err);

};

}

toggleBtn.addEventListener('click', () => {

if (!ws || ws.readyState !== WebSocket.OPEN) {

initWebSocket();

toggleBtn.disabled = true;

toggleBtn.textContent = 'Conectando...';

return;

}

// Enviar comando para alternar LED

ws.send(JSON.stringify({ command: 'toggle' }));

toggleBtn.disabled = true; // evitar clicks múltiples

});

// Inicializar la conexión al cargar la página

initWebSocket();

</script>

</body>

</html>

**Cómo usarlo**

1. Guarda este código en un archivo test-websocket.html.
2. Abre con cualquier navegador moderno (Chrome, Firefox).
3. Asegúrate que tu servidor WebSocket PHP (ws-server.php) esté corriendo en ws://localhost:8080.
4. Al cargar, se conectará y mostrará el estado actual del LED.
5. El botón “Alternar LED” enviará el comando para cambiar el estado.

Si tu servidor está en otra IP o dominio (como en Render), cambia la URL en:

js

CopiarEditar

ws = new WebSocket('ws://localhost:8080');

a algo como

js

CopiarEditar

ws = new WebSocket('ws://tu-dominio.onrender.com:PUERTO');

**🖼️ Paso 3: Frontend con HTML5 + Bootstrap y botones dinámicos**

**1. Frontend HTML5 + Bootstrap con botones para encender/apagar LEDs dinámicos según estado recibido por WebSocket**

html

CopiarEditar

<!DOCTYPE html>

<html lang="es">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1" />

<title>Control LED - Sistema Arduino + Web</title>

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css" rel="stylesheet" />

</head>

<body class="bg-light">

<div class="container py-5">

<h1 class="mb-4">Control de LED - Sistema Arduino ESP + Web</h1>

<div class="card p-4 shadow-sm bg-white">

<h4>Estado del LED: <span id="led-status" class="fw-bold">Desconocido</span></h4>

<div class="mt-3">

<button id="btn-encender" class="btn btn-success me-2" disabled>Encender</button>

<button id="btn-apagar" class="btn btn-danger" disabled>Apagar</button>

</div>

</div>

</div>

<script>

const wsUrl = 'ws://localhost:8080'; // Cambia según tu entorno

let ws;

const ledStatusEl = document.getElementById('led-status');

const btnEncender = document.getElementById('btn-encender');

const btnApagar = document.getElementById('btn-apagar');

function conectarWebSocket() {

ws = new WebSocket(wsUrl);

ws.onopen = () => {

console.log('Conectado al WebSocket');

// botones se habilitan sólo cuando se recibe estado

};

ws.onmessage = (event) => {

const data = JSON.parse(event.data);

if (data.type === 'status') {

actualizarEstadoLed(data.led);

}

};

ws.onclose = () => {

console.log('Conexión WebSocket cerrada');

ledStatusEl.textContent = 'Desconectado';

deshabilitarBotones();

// Intentar reconectar después de 5 segundos

setTimeout(conectarWebSocket, 5000);

};

ws.onerror = (error) => {

console.error('Error WebSocket:', error);

};

}

function actualizarEstadoLed(estado) {

ledStatusEl.textContent = estado;

if (estado === 'ON') {

btnEncender.disabled = true;

btnApagar.disabled = false;

} else if (estado === 'OFF') {

btnEncender.disabled = false;

btnApagar.disabled = true;

} else {

deshabilitarBotones();

}

}

function deshabilitarBotones() {

btnEncender.disabled = true;

btnApagar.disabled = true;

}

btnEncender.addEventListener('click', () => {

if (ws && ws.readyState === WebSocket.OPEN) {

ws.send(JSON.stringify({ command: 'toggle' }));

}

});

btnApagar.addEventListener('click', () => {

if (ws && ws.readyState === WebSocket.OPEN) {

ws.send(JSON.stringify({ command: 'toggle' }));

}

});

// Iniciar conexión al cargar página

conectarWebSocket();

</script>

<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"></script>

</body>

</html>

**🚀 Paso 4: Configuración para desplegar en Render**

**Render solo acepta 1 proceso por servicio, así que debes crear dos servicios separados:**

* Uno para el **backend PHP (login, API REST)**, que use Apache o PHP-FPM
* Otro para el **WebSocket server PHP** (ws-server.php) como servicio de tipo background worker

**Archivos principales para Render:**

**1. render.yaml para backend web (PHP + SQLite + API REST)**

yaml

CopiarEditar

services:

- type: web

name: sistema-backend-php

env: php

plan: free

buildCommand: ''

startCommand: 'php -S 0.0.0.0:$PORT -t public'

envVars:

- key: PORT

value: '10000'

**2. render-worker.yaml para WebSocket server (worker)**

yaml

CopiarEditar

services:

- type: worker

name: sistema-websocket-worker

env: php

plan: free

buildCommand: ''

startCommand: 'php public/ws-server.php'

**Nota:**  
En Render, el puerto $PORT debe usarse para el web.  
El servidor WebSocket puede escuchar un puerto fijo (como 8080), pero en producción deberás usar $PORT o configurar un proxy inverso para conectarte.

**📦 Paso 5: Subida a GitHub + instrucciones para Windows 11**

**1. Inicializar repositorio Git (si no lo tienes)**

Abre PowerShell o terminal en la carpeta raíz del proyecto y ejecuta:

bash

CopiarEditar

git init

git add .

git commit -m "Primer commit: backend, websocket y frontend"

**2. Crear repositorio en GitHub (web)**

* Ve a [github.com](https://github.com)
* Crea un nuevo repositorio público o privado
* No marques opción de README ni .gitignore ni licencia (para evitar conflictos)

**3. Conectar repositorio local con remoto**

bash

CopiarEditar

git remote add origin https://github.com/tu\_usuario/tu\_repositorio.git

git branch -M main

git push -u origin main

**4. Configurar Render para deploy desde GitHub**

* Ve a Render dashboard
* Crea nuevo servicio **Web** para backend, conectando a tu repositorio GitHub y selecciona la rama main
* Crea nuevo servicio **Worker** para WebSocket server con la misma conexión

Render detectará y desplegará automáticamente.