GUÍA TÉCNICA DE HOGARFIX - PARTE 2

Manual Completo para Entender y Explicar tu Proyecto (Continuación)

CÓDIGO EXPLICADO LÍNEA POR LÍNEA

1. BACKEND: server.js (Archivo Principal)

```
// ===== IMPORTAR BIBLIOTECAS =====
const express = require('express');
// Express: framework para crear el servidor web

const cors = require('cors');
// CORS: permite que el frontend (en otro dominio) haga peticiones al backend

const path = require('path');
// Path: utilidad para trabajar con rutas de archivos

const jwt = require('jsonwebtoken');
// JWT: crear y verificar tokens de autenticación

// ===== IMPORTAR MÓDULOS PROPIOS =====
const { initDB } = require('./src/config/database');
// Función que crea las tablas de la base de datos

const authRoutes = require('./src/controllers/authController');
// Rutas de autenticación (login, registro)

const pedidoRoutes = require('./src/controllers/pedidoController');
// Rutas de pedidos (crear, listar, actualizar)

// ... más imports ...

// ===== CREAR LA APLICACIÓN =====
const app = express();
// Crea la instancia del servidor Express

const PORT = process.env.PORT || 3000;
// Puerto donde escucha el servidor (3000 local, variable en Render)

// ===== MIDDLEWARE =====
app.use(cors());
// Permite peticiones desde cualquier origen

app.use(express.json());
// Permite recibir datos en formato JSON en el body de las peticiones

app.use(express.static(path.join(__dirname, '../frontend')));
// Sirve archivos estáticos del frontend (HTML, CSS, JS, imágenes)
```

```
// ===== RUTAS DEL API =====
app.use('/api/auth', authRoutes);
// Todas las rutas que empiecen con /api/auth van a authController
// Ejemplo: POST /api/auth/login → authController.login()

app.use('/api/pedidos', pedidoRoutes);
// Ejemplo: GET /api/pedidos → pedidoController.getPedidos()

// ===== RUTA RAÍZ =====
app.get('/', (req, res) => {
    res.redirect('/frontend/index.html');
});
// Cuando alguien entra a https://hogarfix.onrender.com/
// lo redirige automáticamente a /frontend/index.html

// ===== INICIAR SERVIDOR =====
initDB().then(() => {
    // Primero inicializa la base de datos (crea tablas)

    app.listen(PORT, () => {
        console.log(`Servidor corriendo en http://localhost:${PORT}`);
    });
    // Luego inicia el servidor
}).catch(err => {
    console.error('Error al inicializar la base de datos:', err);
});
```

---

2. BACKEND: authController.js (Autenticación)

```
const bcrypt = require('bcryptjs');
// Biblioteca para encriptar contraseñas

const jwt = require('jsonwebtoken');
// Crear tokens de autenticación

const userRepository = require('../repositories/userRepository');
// Acceso a la base de datos de usuarios

const JWT_SECRET = 'hogarfix_secret_2024';
// Llave secreta para firmar tokens (debería estar en variable de entorno)

// ===== REGISTRAR NUEVO USUARIO =====
async function register(req, res) {
    // req.body contiene los datos enviados desde el frontend
    const { nombre, email, password, rol, categoria } = req.body;

    try {
        // 1. Verificar si el email ya existe
        const existingUser = await userRepository.findByEmail(email);
```

```
        if (existingUser) {
            return res.status(400).json({ error: 'El email ya está registrado' });
        }

        // 2. Encriptar la contraseña
        const hashedPassword = await bcrypt.hash(password, 10);
        // bcrypt.hash(password, 10) genera un hash seguro
        // El "10" es el número de rondas de encriptación (más = más seguro pero
más lento)

        // 3. Crear el usuario en la base de datos
        const newUser = await userRepository.create({
            nombre,
            email,
            password: hashedPassword,  // Guardamos el hash, NO la contraseña
original
            rol,
            categoria: rol === 'profesional' ? categoria : null,
            aprobado: rol === 'cliente' || rol === 'admin' ? 1 : 0
            // Clientes y admins se aprueban automáticamente
            // Profesionales necesitan aprobación manual
        });

        // 4. Generar token de autenticación
        const token = jwt.sign(
            {
                id: newUser.id,
                rol: newUser.rol
            },
            JWT_SECRET,
            { expiresIn: '24h' }  // El token expira en 24 horas
        );

        // 5. Responder con el token y datos del usuario
        res.status(201).json({
            success: true,
            token,
            user: {
                id: newUser.id,
                nombre: newUser.nombre,
                email: newUser.email,
                rol: newUser.rol
            }
        });

    } catch (error) {
        console.error('Error en register:', error);
        res.status(500).json({ error: 'Error al registrar usuario' });
    }
}

// ===== INICIAR SESIÓN =====
async function login(req, res) {
    const { email, password } = req.body;
```

```
    try {
        // 1. Buscar usuario por email
        const user = await userRepository.findByEmail(email);
        if (!user) {
            return res.status(404).json({ error: 'Usuario no encontrado' });
        }

        // 2. Verificar la contraseña
        const isPasswordValid = await bcrypt.compare(password, user.password);
        // bcrypt.compare compara la contraseña ingresada con el hash guardado

        if (!isPasswordValid) {
            return res.status(401).json({ error: 'Contraseña incorrecta' });
        }

        // 3. Verificar si el profesional está aprobado
        if (user.rol === 'profesional' && user.aprobado !== 1) {
            return res.status(403).json({
                error: 'Tu cuenta está pendiente de aprobación por un
administrador'
            });
        }

        // 4. Generar token
        const token = jwt.sign(
            { id: user.id, rol: user.rol },
            JWT_SECRET,
            { expiresIn: '24h' }
        );

        // 5. Responder con el token
        res.json({
            success: true,
            token,
            user: {
                id: user.id,
                nombre: user.nombre,
                email: user.email,
                rol: user.rol,
                categoria: user.categoria,
                saldo: user.saldo
            }
        });

    } catch (error) {
        console.error('Error en login:', error);
        res.status(500).json({ error: 'Error al iniciar sesión' });
    }
}

module.exports = { register, login };
```

3. MIDDLEWARE: authMiddleware.js (Seguridad)

```
const jwt = require('jsonwebtoken');
const JWT_SECRET = 'hogarfix_secret_2024';

// ===== VERIFICAR QUE EL USUARIO ESTÁ AUTENTICADO =====
function authMiddleware(req, res, next) {
    // next es una función que se llama para continuar al siguiente
middleware/controlador

    try {
        // 1. Obtener el token del header Authorization
        const authHeader = req.headers.authorization;
        // Formato esperado: "Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."

        if (!authHeader) {
            return res.status(401).json({ error: 'Token no proporcionado' });
        }

        // 2. Extraer el token (quitar "Bearer ")
        const token = authHeader.split(' ')[1];
        // "Bearer TOKEN" → ["Bearer", "TOKEN"] → TOKEN

        // 3. Verificar y decodificar el token
        const decoded = jwt.verify(token, JWT_SECRET);
        // Si el token es inválido o expiró, jwt.verify lanza un error

        // 4. Guardar los datos del usuario en req.user
        req.user = decoded;
        // decoded contiene { id: 5, rol: 'cliente', iat: 1234567890, exp:
1234654290 }

        // 5. Continuar al siguiente middleware/controlador
        next();

    } catch (error) {
        if (error.name === 'TokenExpiredError') {
            return res.status(401).json({ error: 'Token expirado' });
        }
        return res.status(403).json({ error: 'Token inválido' });
    }
}

module.exports = authMiddleware;
```

¿Cómo se usa?

```
// En server.js o en los controllers:
app.get('/api/pedidos', authMiddleware, pedidoController.getPedidos);
```

```
//                      ↑ Primero verifica el token
//                                          ↑ Luego ejecuta esto
```

---

4. MIDDLEWARE: adminMiddleware.js (Restringir por Rol)

```
// ===== VERIFICAR QUE EL USUARIO ES ADMINISTRADOR =====
function adminMiddleware(req, res, next) {
    // Este middleware se ejecuta DESPUÉS de authMiddleware
    // por lo tanto, req.user ya existe

    if (!req.user || req.user.rol !== 'admin') {
        return res.status(403).json({
            error: 'Acceso denegado. Solo administradores.'
        });
    }

    next();  // Si es admin, continúa
}

module.exports = adminMiddleware;
```

Uso combinado:

```
app.get('/api/admin/estadisticas',
    authMiddleware,     // 1. Verifica que esté autenticado
    adminMiddleware,    // 2. Verifica que sea admin
    adminController.getEstadisticas  // 3. Ejecuta la función
);
```

---

5. FRONTEND: auth.js (Login y Registro)

```
// ===== FUNCIÓN DE REGISTRO =====
async function register(event) {
    event.preventDefault();
    // Evita que el formulario se envíe de forma tradicional (recargar página)

    // 1. Obtener datos del formulario
    const nombre = document.getElementById('nombre').value;
    const email = document.getElementById('email').value;
    const password = document.getElementById('password').value;
    const rol = document.getElementById('rol').value;
    const categoria = document.getElementById('categoria')?.value;

    try {
        // 2. Enviar petición al backend
```

```
        const response = await fetch('http://localhost:3000/api/auth/register', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify({
                nombre, email, password, rol, categoria
            })
        });

        // 3. Obtener la respuesta
        const data = await response.json();

        if (response.ok) {
            // 4. Guardar el token en sessionStorage
            sessionStorage.setItem('token', data.token);
            sessionStorage.setItem('user', JSON.stringify(data.user));

            // 5. Redirigir según el rol
            if (data.user.rol === 'cliente') {
                window.location.href = 'panel-cliente.html';
            } else if (data.user.rol === 'profesional') {
                window.location.href = 'panel-profesional.html';
            } else {
                window.location.href = 'panel-admin.html';
            }
        } else {
            alert(data.error);
        }

    } catch (error) {
        console.error('Error:', error);
        alert('Error al registrarse');
    }
}

// ===== FUNCIÓN DE LOGIN =====
async function login(event) {
    event.preventDefault();

    const email = document.getElementById('email').value;
    const password = document.getElementById('password').value;

    try {
        const response = await fetch('http://localhost:3000/api/auth/login', {
            method: 'POST',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify({ email, password })
        });

        const data = await response.json();

        if (response.ok) {
            sessionStorage.setItem('token', data.token);
```

```
            sessionStorage.setItem('user', JSON.stringify(data.user));

            // Redirigir según rol
            switch(data.user.rol) {
                case 'admin':
                    window.location.href = 'panel-admin.html';
                    break;
                case 'profesional':
                    window.location.href = 'panel-profesional.html';
                    break;
                default:
                    window.location.href = 'panel-cliente.html';
            }
        } else {
            alert(data.error);
        }

    } catch (error) {
        console.error('Error:', error);
        alert('Error al iniciar sesión');
    }
}

// ===== FUNCIÓN DE LOGOUT =====
function logout() {
    sessionStorage.removeItem('token');
    sessionStorage.removeItem('user');
    window.location.href = 'login.html';
}

// ===== VERIFICAR SI EL USUARIO ESTÁ LOGUEADO =====
function checkAuth() {
    const token = sessionStorage.getItem('token');
    if (!token) {
        window.location.href = 'login.html';
    }
    return token;
}
```

---

6. FRONTEND: panelCliente.js (Panel del Cliente)

```
// ===== CARGAR PEDIDOS DEL CLIENTE =====
async function cargarPedidos() {
    const token = sessionStorage.getItem('token');

    try {
        const response = await fetch('http://localhost:3000/api/pedidos', {
            headers: {
                'Authorization': `Bearer ${token}`
            }
```

```
        });

        const pedidos = await response.json();

        // Mostrar pedidos en la tabla
        const tbody = document.getElementById('pedidosTableBody');
        tbody.innerHTML = '';

        pedidos.forEach(pedido => {
            const row = document.createElement('tr');
            row.innerHTML = `
                <td>${pedido.id}</td>
                <td>${pedido.profesional_nombre}</td>
                <td>${pedido.descripcion}</td>
                <td>$${pedido.precio || 'Por confirmar'}</td>
                <td>
                    <span class="badge bg-${getEstadoBadge(pedido.estado)}">
                        ${pedido.estado}
                    </span>
                </td>
                <td>
                    ${pedido.estado === 'completado' ?
                        `<button
onclick="abrirModalCalificar(${pedido.id})">Calificar</button>` :
                        ''}
                </td>
            `;
            tbody.appendChild(row);
        });

    } catch (error) {
        console.error('Error al cargar pedidos:', error);
    }
}

// ===== CREAR NUEVO PEDIDO =====
async function crearPedido(event) {
    event.preventDefault();

    const token = sessionStorage.getItem('token');
    const profesionalId = document.getElementById('profesionalSelect').value;
    const descripcion = document.getElementById('descripcion').value;

    try {
        const response = await fetch('http://localhost:3000/api/pedidos', {
            method: 'POST',
            headers: {
                'Authorization': `Bearer ${token}`,
                'Content-Type': 'application/json'
            },
            body: JSON.stringify({
                profesionalId,
                descripcion
            })
```

```
        });

        const data = await response.json();

        if (response.ok) {
            alert('Pedido enviado exitosamente');
            cargarPedidos();  // Recargar lista
            document.getElementById('formPedido').reset();
        } else {
            alert(data.error);
        }

    } catch (error) {
        console.error('Error:', error);
        alert('Error al crear pedido');
    }
}

// ===== FUNCIÓN AUXILIAR: COLOR DEL BADGE SEGÚN ESTADO =====
function getEstadoBadge(estado) {
    switch(estado) {
        case 'pendiente': return 'warning';
        case 'confirmado': return 'info';
        case 'completado': return 'success';
        default: return 'secondary';
    }
}

// ===== CARGAR AL INICIAR LA PÁGINA =====
document.addEventListener('DOMContentLoaded', () => {
    checkAuth();  // Verificar que esté logueado
    cargarPedidos();
    cargarProfesionales();
});
```

REPOSITORY PATTERN - EXPLICADO

¿Por qué usar Repositories?

Sin Repository (malo ✗):

```
// pedidoController.js
app.get('/api/pedidos', (req, res) => {
    db.all('SELECT * FROM pedidos WHERE cliente_id = ?', [req.user.id], (err,
rows) => {
        if (err) return res.status(500).json({ error: err.message });
        res.json(rows);
    });
});
```

```
// userController.js
app.get('/api/usuarios', (req, res) => {
    db.all('SELECT * FROM usuarios', (err, rows) => {
        if (err) return res.status(500).json({ error: err.message });
        res.json(rows);
    });
});
```

Problemas:

1. SQL mezclado con lógica de negocio
2. Si cambias de base de datos, modificas TODO
3. Difícil de testear

Con Repository (bueno ☑):

```
// pedidoRepository.js
class PedidoRepository {
    getPedidosByCliente(clienteId) {
        return new Promise((resolve, reject) => {
            db.all(
                'SELECT * FROM pedidos WHERE cliente_id = ?',
                [clienteId],
                (err, rows) => {
                    if (err) reject(err);
                    else resolve(rows);
                }
            );
        });
    }

    create(pedido) {
        return new Promise((resolve, reject) => {
            db.run(
                'INSERT INTO pedidos (cliente_id, profesional_id, descripcion)
VALUES (?, ?, ?)',
                [pedido.clienteId, pedido.profesionalId, pedido.descripcion],
                function(err) {
                    if (err) reject(err);
                    else resolve({ id: this.lastID });
                }
            );
        });
    }
}

module.exports = new PedidoRepository();
```

```
// pedidoController.js (limpio)
const pedidoRepository = require('../repositories/pedidoRepository');

async function getPedidos(req, res) {
    try {
        const pedidos = await pedidoRepository.getPedidosByCliente(req.user.id);
        res.json(pedidos);
    } catch (error) {
        res.status(500).json({ error: error.message });
    }
}
```

Beneficios:

- ☑ Lógica de BD separada
- ☑ Fácil de cambiar a otra BD
- ☑ Fácil de testear
- ☑ Código más limpio

SEGURIDAD - BUENAS PRÁCTICAS

1. **Encriptación de Contraseñas**

NUNCA guardes contraseñas en texto plano:

```
// ❌ MAL
const user = { email: 'juan@test.com', password: '123456' };
db.run('INSERT INTO usuarios (email, password) VALUES (?, ?)', [user.email,
user.password]);

// ☑ BIEN
const bcrypt = require('bcryptjs');
const hashedPassword = await bcrypt.hash(user.password, 10);
db.run('INSERT INTO usuarios (email, password) VALUES (?, ?)', [user.email,
hashedPassword]);
```

¿Cómo funciona bcrypt?

```
Contraseña original: "123456"
Hash generado: "$2a$10$N9qo8uLOickgx2ZMRZoMyeIjZAgcfl7p92ldGxad68LJZdL17lhWy"
               ↑    ↑  ↑                                          ↑
               Alg  Rounds  Salt (aleatorio)                      Hash
```

- Mismo input genera SIEMPRE el mismo hash
- Hash NO se puede revertir a la contraseña original
- Cada hash tiene un "salt" aleatorio (evita ataques de rainbow tables)

2. **Prepared Statements (Prevenir SQL Injection)**

❌ **Vulnerable a SQL Injection:**

```
const email = req.body.email;
db.all(`SELECT * FROM usuarios WHERE email = '${email}'`, (err, rows) => {
    // Si email = "' OR '1'='1" → SELECT * FROM usuarios WHERE email = '' OR
'1'='1'
    // Esto devuelve TODOS los usuarios
});
```

☑ **Seguro con Prepared Statements:**

```
db.all('SELECT * FROM usuarios WHERE email = ?', [email], (err, rows) => {
    // El "?" se reemplaza de forma segura, escapando caracteres especiales
});
```

3. **Validación de Datos**

```
// ===== VALIDAR EMAIL =====
function isValidEmail(email) {
    const regex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
    return regex.test(email);
}

// ===== VALIDAR CONTRASEÑA =====
function isValidPassword(password) {
    return password.length >= 6;
}

// ===== USAR EN EL CONTROLLER =====
async function register(req, res) {
    const { email, password } = req.body;

    if (!isValidEmail(email)) {
        return res.status(400).json({ error: 'Email inválido' });
    }

    if (!isValidPassword(password)) {
        return res.status(400).json({ error: 'La contraseña debe tener al menos 6
caracteres' });
    }

    // ... continuar con el registro
}
```

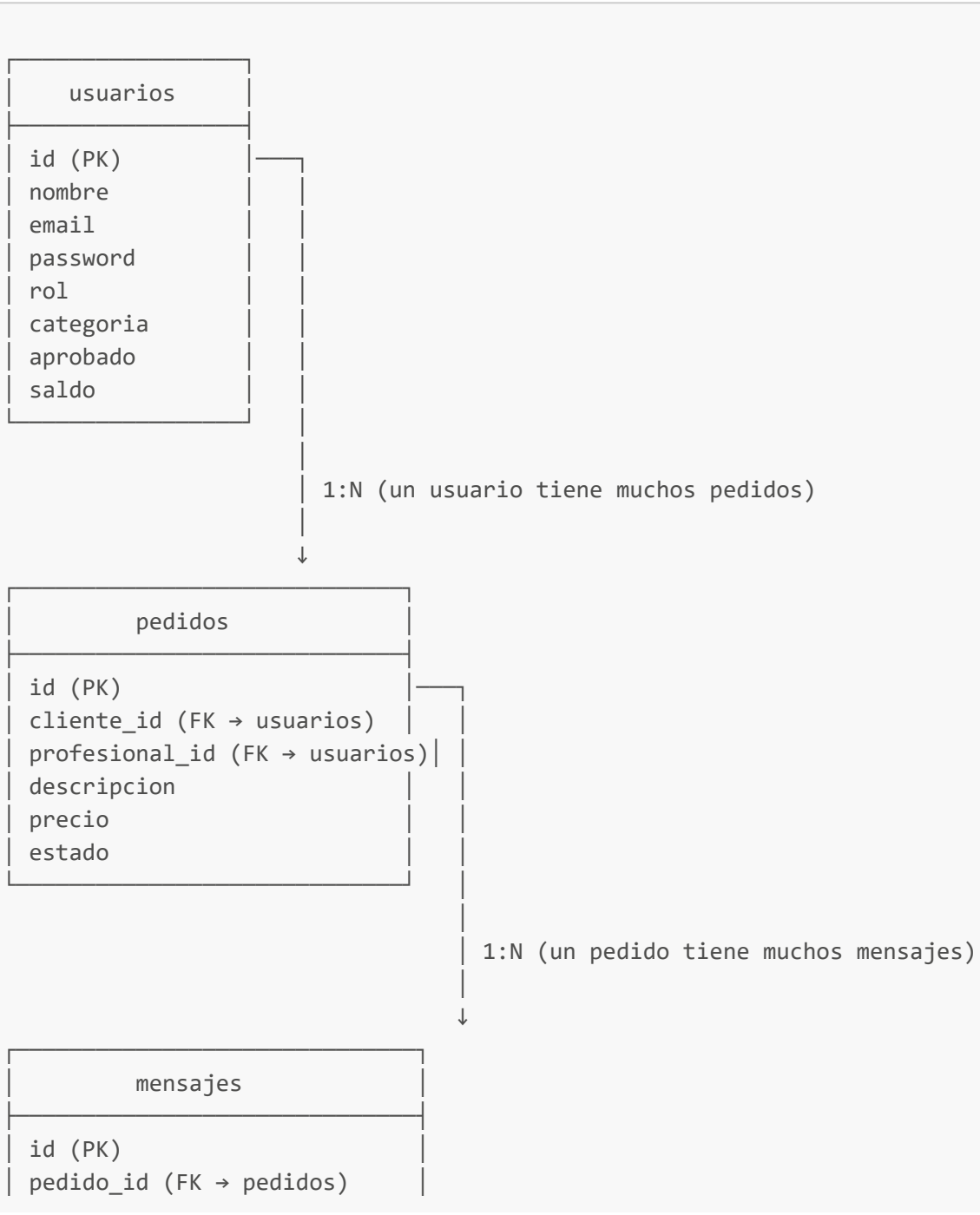
4. **CORS (Cross-Origin Resource Sharing)**

```
const cors = require('cors');
app.use(cors());
// Permite peticiones desde cualquier origen

// Para producción, es mejor especificar orígenes permitidos:
app.use(cors({
    origin: 'https://hogarfix.onrender.com',
    credentials: true
}));
```

MODELO DE DATOS COMPLETO

Diagrama de Relaciones



```
| remitente_id (FK → usuarios) |
| destinatario_id (FK → usuarios)|
| mensaje                      |
| imagen                       |
└──────────────────────────────┘

┌──────────────────────────┐
│      categorias          │
├──────────────────────────┤
| id (PK)                  |
| nombre                   |
| descripcion              |
| icono                    |
└──────────────────────────┘

┌──────────────────────────┐
│       ratings            │
├──────────────────────────┤
| id (PK)                  |
| pedido_id (FK → pedidos) |
| estrellas                |
| comentario               |
└──────────────────────────┘
```

🚀 GUÍA DE ACTUALIZACIÓN Y DESPLIEGUE

Flujo Completo de Desarrollo a Producción

```
1. DESARROLLO LOCAL
   ├─ Escribes código
   ├─ Pruebas en http://localhost:3000
   └─ Verificas que funciona

2. CONTROL DE VERSIONES (Git)
   ├─ git add .
   ├─ git commit -m "Descripción de cambios"
   └─ git push origin main

3. DESPLIEGUE AUTOMÁTICO (Render)
   ├─ Render detecta el push
   ├─ Ejecuta npm install (instala dependencias)
   ├─ Ejecuta npm start (inicia servidor)
   └─ Publica en https://hogarfix.onrender.com
```

Comandos Git Esenciales

```
# Ver estado de archivos
git status
```

```
# Agregar todos los cambios
git add .

# Agregar archivo específico
git add backend/server.js

# Confirmar cambios
git commit -m "Mensaje descriptivo"

# Subir a GitHub
git push origin main

# Ver historial
git log --oneline

# Crear nueva rama
git checkout -b feature/nueva-funcionalidad

# Cambiar a rama main
git checkout main
```

💡 PREGUNTAS FRECUENTES EN DEFENSAS

1. "¿Por qué elegiste Node.js y no Python/PHP/Java?"

Respuesta: "Elegí Node.js porque permite usar JavaScript tanto en frontend como en backend, lo que facilita el desarrollo y reduce la curva de aprendizaje. Además, Node.js tiene un ecosistema muy rico (npm) y es ideal para aplicaciones que requieren manejo de múltiples conexiones simultáneas gracias a su modelo asíncrono."

2. "¿Por qué SQLite y no MySQL/PostgreSQL?"

Respuesta: "Para esta fase del proyecto, SQLite es ideal porque:

- No requiere servidor separado
- Es ligero y rápido para volúmenes moderados de datos
- Facilita el despliegue (solo un archivo)
- Para escalar en el futuro, podría migrar a PostgreSQL sin cambiar mucho código gracias al patrón Repository"

3. "¿Cómo manejan la seguridad?"

Respuesta: "Implementamos varias capas de seguridad:

- Autenticación con JWT
- Encriptación de contraseñas con bcrypt
- Prepared statements para prevenir SQL injection
- Middleware de autorización basado en roles
- HTTPS en producción
- Validación de datos en frontend y backend"

4. "¿Qué harían diferente con más tiempo?"

Respuesta: "Con más tiempo implementaría:

- Chat en tiempo real con WebSockets
- Pasarela de pago integrada (Stripe/PayPal)
- Sistema de geolocalización para encontrar profesionales cercanos
- App móvil nativa con React Native
- Tests unitarios y de integración
- CI/CD automatizado"

5. "¿Cómo escalaría el sistema?"

Respuesta: "Para escalar implementaría:

- Migración a PostgreSQL para mejor rendimiento con muchos usuarios
- Caché con Redis para consultas frecuentes
- Load balancer para distribuir tráfico
- Microservicios para separar funcionalidades (chat, pagos, notificaciones)
- CDN para servir archivos estáticos
- Queue system para tareas asíncronas (envío de emails, procesamiento de imágenes)"

📚 TÉRMINOS TÉCNICOS CLAVE

API (Application Programming Interface)

"Conjunto de rutas y métodos que permite que el frontend se comunique con el backend. Es como un menú de restaurante: lista qué puedes pedir y cómo hacerlo."

REST (Representational State Transfer)

"Estilo de arquitectura para APIs que usa HTTP y URLs descriptivas. GET /api/usuarios = obtener usuarios, POST /api/usuarios = crear usuario."

JSON (JavaScript Object Notation)

"Formato de texto para intercambiar datos. Ejemplo: `{ 'nombre': 'Juan', 'edad': 25 }`"

Token JWT

"Credencial digital que identifica al usuario. Es como tu credencial de estudiante: te la dan al inscribirte (login) y la muestras para acceder (peticiones)."

Middleware

"Función que se ejecuta antes del destino final. Como un guardia de seguridad que verifica tu ID antes de dejarte pasar."

Hash

"Resultado de una función de encriptación de un solo sentido. Puedes convertir 'contraseña123' a un hash, pero no puedes revertir el hash a la contraseña original."

Asíncrono (async/await)

"Código que no bloquea la ejecución. Mientras esperas una respuesta de la BD, el servidor puede atender otras peticiones."

Promise

"Objeto que representa un valor que estará disponible ahora, en el futuro, o nunca. Como un ticket de una orden en un restaurante."

Repository Pattern

"Patrón de diseño que separa la lógica de acceso a datos. Es como tener un bibliotecario: tú pides un libro, él sabe dónde buscarlo."

MVC (Model-View-Controller)

"Patrón que separa datos (Model), presentación (View) y lógica (Controller). Hace el código más organizado y mantenible."

🎤 SCRIPT DE PRESENTACIÓN (10 MINUTOS)

Minuto 1-2: Introducción

"Buenos días/tardes. Soy [tu nombre] y les presento HogarFix, una plataforma web que conecta a personas que necesitan servicios para el hogar con profesionales verificados. Resolver el problema de confianza y accesibilidad en la contratación de servicios domésticos."

Minuto 3-4: Tecnologías

"El proyecto usa una arquitectura de tres capas:

- Frontend con HTML5, CSS3, JavaScript y Bootstrap
- Backend con Node.js y Express.js
- Base de datos SQLite Implementamos autenticación JWT, encriptación con bcrypt, y un sistema de roles completo."

Minuto 5-6: Funcionalidades

"HogarFix ofrece:

- Registro y aprobación de profesionales
- Búsqueda por categorías
- Sistema de pedidos con confirmación de precio
- Chat integrado
- Calificaciones y reseñas
- Panel de administración

- Respaldos automáticos cada 24 horas"

Minuto 7-8: Demo en Vivo

[Mostrar flujo completo] "Voy a demostrar el flujo completo:

1. Cliente solicita servicio
2. Profesional confirma precio
3. Chat entre ambos
4. Profesional completa servicio
5. Cliente califica
6. Admin ve estadísticas"

Minuto 9: Innovación

"Lo innovador de HogarFix es el sistema integral que combina verificación de profesionales, comunicación directa, sistema de pagos con QR, y gestión automatizada con respaldos periódicos."

Minuto 10: Cierre

"HogarFix está desplegado en producción en hogarfix.onrender.com, con integración continua desde GitHub. Futuras mejoras incluyen WebSockets, pasarela de pago, y app móvil. Gracias por su atención, quedo atento a sus preguntas."

☑ ÚLTIMO CHECKLIST

Pre-presentación (Hoy):

- ☐ Leer esta guía completa 2 veces
- ☐ Practicar explicación de arquitectura
- ☐ Probar https://hogarfix.onrender.com
- ☐ Crear datos de prueba (3 profesionales, 5 pedidos, mensajes)
- ☐ Preparar cuentas de demo
- ☐ Verificar que el chat funciona
- ☐ Tomar screenshots de respaldo
- ☐ Dormir bien (importante!)

Día de la presentación:

- ☐ Llegar 30 minutos antes
- ☐ Probar conexión a internet
- ☐ Probar proyector/pantalla
- ☐ Tener laptop con batería completa
- ☐ Tener cable HDMI
- ☐ Cerrar pestañas innecesarias
- ☐ Abrir pestañas necesarias:
  - hogarfix.onrender.com
  - GitHub repository
  - Esta guía (por si olvidas algo)

Durante la presentación:

- ☐ Respirar profundo
- ☐ Hablar despacio y claro
- ☐ Mantener contacto visual
- ☐ Usar el mouse/puntero para señalar
- ☐ Si algo falla, usar screenshots
- ☐ No decir "no sé", decir "es una mejora futura"

🎯 PUNTOS CLAVE PARA MAXIMIZAR PUNTOS

Calidad Técnica (25 puntos):

☑ "Usamos arquitectura en capas para separación de responsabilidades" ☑ "Implementamos patrones de diseño: Repository, MVC, Middleware" ☑ "Código modular y reutilizable"

Funcionalidad (20 puntos):

☑ "Sistema completo y funcional en producción" ☑ "Maneja múltiples roles con permisos diferenciados" ☑ "Incluye funcionalidades avanzadas: chat, ratings, backups"

Innovación (15 puntos):

☑ "Sistema integral que otras soluciones no ofrecen" ☑ "Automatización de backups" ☑ "Integración de pagos con QR"

Claridad (15 puntos):

☑ Explicar con analogías simples ☑ Usar diagramas visuales ☑ Demo bien preparada y fluida

Demo (15 puntos):

☑ Mostrar flujo completo ☑ Datos de prueba realistas ☑ Destacar características únicas

Motivación (10 puntos):

☑ "Resolver un problema real de confianza" ☑ "Aprendizaje de tecnologías modernas" ☑ "Visión de crecimiento y escalabilidad"

💥 MENSAJE FINAL

¡Tú puedes hacerlo!

Recuerda:

- Has trabajado duro en este proyecto
- Conoces tu código mejor que nadie
- Los jueces quieren ver tu pasión, no perfección
- Es normal estar nervioso, úsalo como energía positiva

- Si olvidas algo, respira y continúa
- Disfruta el momento de mostrar tu trabajo

¡Mucha suerte en tu presentación! 🚀

FIN DE LA PARTE 2

Archivos completos:

- `GUIA_TECNICA_PARTE1.md`
- `GUIA_TECNICA_PARTE2.md`