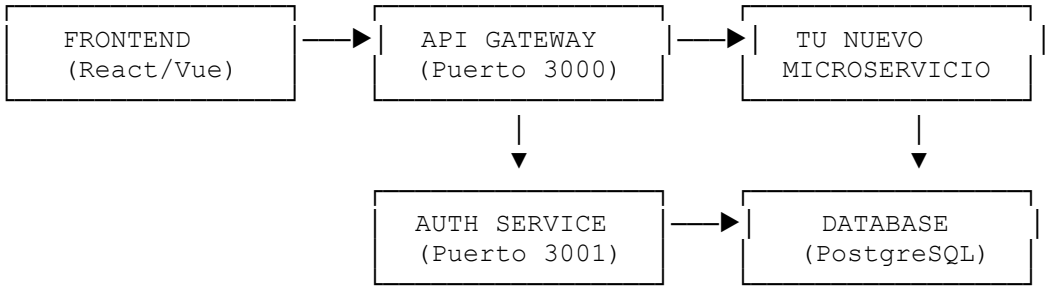
📚 Guía del Desarrollador - Creación de Microservicios

🎯 Introducción

Esta guía te ayudará a crear **nuevos microservicios** que se integren automáticamente con nuestro **sistema de autenticación centralizada**. Tu microservicio heredará toda la funcionalidad de login, permisos y seguridad sin necesidad de implementar nada desde cero.

📐 Arquitectura del Sistema



Flujo de Autenticación:

1. **Frontend** → Envía token JWT en headers
2. **API Gateway** → Redirige request a tu microservicio
3. **Tu Microservicio** → Verifica token con Auth Service
4. **Auth Service** → Valida token y permisos
5. **Tu Microservicio** → Procesa la lógica de negocio

🚀 Creando tu Primer Microservicio

📁 Estructura Recomendada

```
mi-microservicio/
├── src/
│   ├── index.js              # Servidor principal
│   ├── routes/
│   │   └── miRoutes.js       # Rutas específicas
```

```
│   ├── middleware/
│   │   └── auth.js           # Middleware de autenticación
│   ├── models/
│   │   └── miModelo.js       # Modelos de datos
│   └── config/
│       └── database.js       # Configuración de DB
├── package.json              # Dependencias
├── Dockerfile               # Contenedor Docker
└── .env                     # Variables de entorno
```

🔧 PASO 1: Configuración Inicial

1.1 Crear package.json

```
{
  "name": "mi-microservicio",
  "version": "1.0.0",
  "description": "Mi microservicio con auth centralizada",
  "main": "src/index.js",
  "scripts": {
    "start": "node src/index.js",
    "dev": "nodemon src/index.js"
  },
  "dependencies": {
    "express": "^4.18.2",
    "axios": "^1.5.0",
    "cors": "^2.8.5",
    "helmet": "^7.0.0",
    "dotenv": "^16.3.1",
    "joi": "^17.10.0",
    "pg": "^8.11.3"
  },
  "engines": {
    "node": ">=18.0.0"
  }
}
```

1.2 Crear archivo .env

```
# Configuración del microservicio
PORT=3020
SERVICE_NAME=mi-microservicio

# Servicios externos
AUTH_SERVICE_URL=http://auth-service:3001
DATABASE_URL=postgresql://postgres:password@postgres:5432/mi_db

# Entorno
NODE_ENV=development
```

🔐 PASO 2: Middleware de Autenticación

2.1 Crear middleware/auth.js

```
// src/middleware/auth.js
const axios = require('axios');

class MicroserviceAuth {
  constructor(authServiceUrl = process.env.AUTH_SERVICE_URL) {
    this.authServiceUrl = authServiceUrl;
    console.log(`🔐 Auth middleware inicializado: ${authServiceUrl}`);
  }

  // Middleware de autenticación básica
  authenticate = async (req, res, next) => {
    try {
      const authHeader = req.headers.authorization;

      if (!authHeader || !authHeader.startsWith('Bearer ')) {
        return res.status(401).json({
          success: false,
          message: 'Token de autorización requerido'
        });
      }

      const token = authHeader.split(' ')[1];

      console.log(`🔍 Verificando token para: ${req.method}
${req.path}`);

      // Verificar token con Auth Service
      const response = await
axios.get(`${this.authServiceUrl}/auth/verify`, {
        headers: { Authorization: `Bearer ${token}` },
        timeout: 5000
      });

      if (response.data.success) {
        req.user = response.data.data.user;
        console.log(`✅ Usuario autenticado: ${req.user.email}
(${req.user.role})`);
        next();
      } else {
        return res.status(401).json({
          success: false,
          message: 'Token inválido'
        });
      }

    } catch (error) {
      console.error('❌ Error en autenticación:', error.message);

      if (error.response?.status === 401) {
        return res.status(401).json({
```

```
          success: false,
          message: 'Token inválido o expirado'
        });
      }

      return res.status(503).json({
        success: false,
        message: 'Servicio de autenticación no disponible'
      });
    }
  };

  // Middleware de permisos específicos
  requirePermission = (permission) => {
    return async (req, res, next) => {
      try {
        if (!req.user) {
          return res.status(401).json({
            success: false,
            message: 'Usuario no autenticado'
          });
        }

        console.log(`🔐 Verificando permiso: ${permission} para
${req.user.email}`);

        // Verificar permiso con Auth Service
        const response = await axios.post(
          `${this.authServiceUrl}/auth/check-permission`,
          {
            userId: req.user.id,
            permission
          },
          {
            headers: { Authorization: req.headers.authorization },
            timeout: 5000
          }
        );

        if (response.data.success && response.data.hasPermission) {
          console.log(`✅ Permiso ${permission} concedido a
${req.user.email}`);
          next();
        } else {
          console.log(`❌ Permiso ${permission} denegado a
${req.user.email}`);
          return res.status(403).json({
            success: false,
            message: `Sin permisos para: ${permission}`,
            requiredPermission: permission,
            userRole: req.user.role
          });
        }

      } catch (error) {
        console.error('❌ Error verificando permisos:', error.message);
```

```
        return res.status(403).json({
          success: false,
          message: 'Error verificando permisos'
        });
      }
    };
  };

  // Middleware de roles (más simple)
  requireRole = (allowedRoles) => {
    return (req, res, next) => {
      if (!req.user) {
        return res.status(401).json({
          success: false,
          message: 'Usuario no autenticado'
        });
      }

      if (allowedRoles.includes(req.user.role)) {
        console.log(`✅ Rol ${req.user.role} permitido para
${req.user.email}`);
        next();
      } else {
        console.log(`❌ Rol ${req.user.role} no permitido para
${req.user.email}`);
        return res.status(403).json({
          success: false,
          message: `Rol requerido: ${allowedRoles.join(', ')}`,
          currentRole: req.user.role
        });
      }
    };
  };
}

module.exports = MicroserviceAuth;
```

🎯 PASO 3: Servidor Principal

3.1 Crear src/index.js

```
// src/index.js
const express = require('express');
const cors = require('cors');
const helmet = require('helmet');
require('dotenv').config();

// Importar nuestro middleware de auth
const MicroserviceAuth = require('./middleware/auth');

const app = express();
const PORT = process.env.PORT || 3020;
const SERVICE_NAME = process.env.SERVICE_NAME || 'mi-microservicio';
```

```
// Configuración de seguridad
app.use(helmet());
app.use(cors({
  origin: '*',
  methods: ['GET', 'POST', 'PUT', 'DELETE', 'OPTIONS'],
  allowedHeaders: ['Content-Type', 'Authorization']
}));
app.use(express.json());

// Inicializar middleware de autenticación
const auth = new MicroserviceAuth();

console.log(`🚀 ${SERVICE_NAME} iniciando...`);

// ================== RUTAS PÚBLICAS ==================

// Health check (sin autenticación)
app.get('/health', (req, res) => {
  res.json({
    success: true,
    service: SERVICE_NAME,
    status: 'OK',
    port: PORT,
    timestamp: new Date().toISOString(),
    version: '1.0.0'
  });
});

// Ruta pública de ejemplo
app.get('/public', (req, res) => {
  res.json({
    success: true,
    message: `¡Hola desde ${SERVICE_NAME}! (público)`,
    service: SERVICE_NAME,
    timestamp: new Date().toISOString()
  });
});

// ================== RUTAS PRIVADAS ==================

// Ruta que requiere autenticación
app.get('/private', auth.authenticate, (req, res) => {
  res.json({
    success: true,
    message: `¡Hola ${req.user.firstName}! Acceso privado a
${SERVICE_NAME}`,
    user: {
      email: req.user.email,
      role: req.user.role,
      name: `${req.user.firstName} ${req.user.lastName}`
    },
    service: SERVICE_NAME,
    timestamp: new Date().toISOString()
  });
});
```

```
// Ruta que requiere permiso específico
app.post('/crear-recurso',
  auth.authenticate,
  auth.requirePermission('mi-servicio.create'),
  (req, res) => {
    res.json({
      success: true,
      message: `Recurso creado exitosamente en ${SERVICE_NAME}`,
      createdBy: req.user.email,
      data: req.body,
      timestamp: new Date().toISOString()
    });
  }
);

// Ruta solo para admins
app.get('/admin-panel',
  auth.authenticate,
  auth.requireRole(['admin']),
  (req, res) => {
    res.json({
      success: true,
      message: `Panel de administración de ${SERVICE_NAME}`,
      adminUser: req.user.email,
      adminData: {
        service: SERVICE_NAME,
        port: PORT,
        uptime: process.uptime(),
        memoryUsage: process.memoryUsage()
      },
      timestamp: new Date().toISOString()
    });
  }
);

// ================== RUTAS DE NEGOCIO ==================

// Aquí van tus rutas específicas del microservicio
app.get('/mi-funcionalidad', auth.authenticate, (req, res) => {
  // Tu lógica de negocio aquí
  res.json({
    success: true,
    message: 'Mi funcionalidad específica',
    user: req.user.email,
    // ... tu lógica
  });
});

// ================== MANEJO DE ERRORES ==================

app.use((err, req, res, next) => {
  console.error('❌ Error no manejado:', err);
  res.status(500).json({
    success: false,
    message: 'Error interno del servidor',
    service: SERVICE_NAME
  });
```

```
});

// Ruta catch-all
app.use('*', (req, res) => {
  res.status(404).json({
    success: false,
    message: `Ruta no encontrada en ${SERVICE_NAME}: ${req.method}
${req.originalUrl}`,
    service: SERVICE_NAME
  });
});

// ================== INICIAR SERVIDOR ==================

app.listen(PORT, () => {
  console.log(`🎉 ${SERVICE_NAME} ejecutándose en puerto ${PORT}`);
  console.log('📍 Rutas disponibles:');
  console.log('   GET  /health              - Health check (público)');
  console.log('   GET  /public              - Ruta pública');
  console.log('   GET  /private             - Ruta privada (requiere
login)');
  console.log('   POST /crear-recurso       - Crear recurso (requiere
permiso)');
  console.log('   GET  /admin-panel         - Solo admins');
  console.log('   GET  /mi-funcionalidad    - Tu lógica específica');
  console.log('');
  console.log(`🔗 Integrado con Auth Service:
${process.env.AUTH_SERVICE_URL}`);
});
```

🐳 PASO 4: Dockerización

4.1 Crear Dockerfile

```
FROM node:18-alpine

WORKDIR /app

# Copiar archivos de configuración
COPY package*.json ./

# Instalar dependencias
RUN npm config set strict-ssl false && npm install --only=production

# Copiar código fuente
COPY . .

# Crear usuario no-root
RUN addgroup -g 1001 -S nodejs
RUN adduser -S nodejs -u 1001
RUN chown -R nodejs:nodejs /app
USER nodejs
```

```
# Exponer puerto
EXPOSE 3020

# Comando de inicio
CMD ["npm", "start"]
```

🔗 PASO 5: Integración con API Gateway

5.1 Agregar al docker-compose.yml

```
  # Tu nuevo microservicio
  mi-microservicio:
    build: ./mi-microservicio
    ports:
      - "3020:3020"
    environment:
      - NODE_ENV=development
      - PORT=3020
      - SERVICE_NAME=mi-microservicio
      - AUTH_SERVICE_URL=http://auth-service:3001
      - DATABASE_URL=postgresql://postgres:password@postgres:5432/mi_db
    depends_on:
      - auth-service
      - postgres
    volumes:
      - ./mi-microservicio:/app
      - /app/node_modules
    networks:
      - microservices-network
```

5.2 Agregar rutas al API Gateway

Editar `api-gateway/src/index.js` y agregar:

```
// ================== MI MICROSERVICIO ROUTES ==================

// Rutas de tu microservicio
app.get('/mi-servicio/health', async (req, res) => {
  await proxyRequest(req, res, 'http://mi-microservicio:3020/health');
});

app.get('/mi-servicio/public', async (req, res) => {
  await proxyRequest(req, res, 'http://mi-microservicio:3020/public');
});

app.get('/mi-servicio/private', async (req, res) => {
  await proxyRequest(req, res, 'http://mi-microservicio:3020/private');
});

app.post('/mi-servicio/crear-recurso', async (req, res) => {
  await proxyRequest(req, res, 'http://mi-microservicio:3020/crear-
recurso');
```

```
});

app.get('/mi-servicio/admin-panel', async (req, res) => {
  await proxyRequest(req, res, 'http://mi-microservicio:3020/admin-
panel');
});

console.log('🔗 Mi Microservicio routes agregadas al API Gateway');
```

🛡️ PASO 6: Configurar Permisos

6.1 Agregar permisos al Auth Service

Conectarse a la base de datos y ejecutar:

```
-- Agregar permisos para tu microservicio
INSERT INTO permissions (name, display_name, description, service,
resource, action) VALUES
('mi-servicio.create', 'Crear en Mi Servicio', 'Crear recursos en mi
microservicio', 'mi-servicio', 'recursos', 'create'),
('mi-servicio.read', 'Leer de Mi Servicio', 'Ver recursos de mi
microservicio', 'mi-servicio', 'recursos', 'read'),
('mi-servicio.update', 'Actualizar Mi Servicio', 'Modificar recursos de
mi microservicio', 'mi-servicio', 'recursos', 'update'),
('mi-servicio.delete', 'Eliminar de Mi Servicio', 'Eliminar recursos de
mi microservicio', 'mi-servicio', 'recursos', 'delete');

-- Asignar permisos a roles
INSERT INTO role_permissions (role_id, permission_id)
SELECT r.id, p.id
FROM roles r, permissions p
WHERE r.name = 'admin'
AND p.name IN ('mi-servicio.create', 'mi-servicio.read', 'mi-
servicio.update', 'mi-servicio.delete');

INSERT INTO role_permissions (role_id, permission_id)
SELECT r.id, p.id
FROM roles r, permissions p
WHERE r.name = 'moderator'
AND p.name IN ('mi-servicio.read', 'mi-servicio.update');

INSERT INTO role_permissions (role_id, permission_id)
SELECT r.id, p.id
FROM roles r, permissions p
WHERE r.name = 'user'
AND p.name IN ('mi-servicio.read');
```

🧪 PASO 7: Pruebas

7.1 Levantar servicios

```
docker-compose up --build
```

7.2 Probar endpoints

```
# Health check
Invoke-RestMethod -Uri "http://localhost:3000/mi-servicio/health"

# Ruta pública
Invoke-RestMethod -Uri "http://localhost:3000/mi-servicio/public"

# Login y rutas privadas
$login = Invoke-RestMethod -Uri "http://localhost:3000/auth/login" -
Method Post -ContentType "application/json" -Body '{"email":
"admin@admin.com", "password": "admin123"}'
$headers = @{ Authorization = "Bearer $($login.data.accessToken)" }

# Ruta privada
Invoke-RestMethod -Uri "http://localhost:3000/mi-servicio/private" -
Headers $headers

# Panel de admin
Invoke-RestMethod -Uri "http://localhost:3000/mi-servicio/admin-panel" -
Headers $headers

# Crear recurso (con permiso)
Invoke-RestMethod -Uri "http://localhost:3000/mi-servicio/crear-recurso"
-Method Post -Headers $headers -ContentType "application/json" -Body
'{"nombre": "Mi Recurso"}'
```

🎨 PASO 8: Integración con Frontend

8.1 Llamar desde frontend

```
// En tu frontend
async function callMiMicroservicio() {
  const token = localStorage.getItem('authToken');

  try {
    // Ruta pública
    const publicResponse = await fetch('http://localhost:3000/mi-
servicio/public');

    // Ruta privada
    const privateResponse = await fetch('http://localhost:3000/mi-
servicio/private', {
      headers: { Authorization: `Bearer ${token}` }
    });

    // Crear recurso
    const createResponse = await fetch('http://localhost:3000/mi-
servicio/crear-recurso', {
      method: 'POST',
      headers: {
```

```
        'Authorization': `Bearer ${token}`,
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({ nombre: 'Nuevo recurso' })
    });

    console.log('Microservicio responses:', {
      public: await publicResponse.json(),
      private: await privateResponse.json(),
      create: await createResponse.json()
    });

  } catch (error) {
    console.error('Error llamando microservicio:', error);
  }
}
```

📋 Checklist de Desarrollo

✅ Antes de empezar:

- [ ] Asegúrate que el sistema base esté funcionando
- [ ] Define qué permisos necesita tu microservicio
- [ ] Elige un puerto único (3020, 3021, etc.)

✅ Durante el desarrollo:

- [ ] Usa el middleware de autenticación proporcionado
- [ ] Implementa health check en `/health`
- [ ] Agrega logging para debug
- [ ] Maneja errores apropiadamente

✅ Antes de producción:

- [ ] Agrega tu servicio al docker-compose.yml
- [ ] Configura rutas en API Gateway
- [ ] Registra permisos en la base de datos
- [ ] Prueba todos los endpoints
- [ ] Documenta tu API

🔥 Ejemplos de Microservicios

📊 Project Service

```
// Ejemplo: Gestión de proyectos
app.post('/projects',
  auth.authenticate,
  auth.requirePermission('projects.create'),
  async (req, res) => {
    // Crear proyecto en DB
    const project = await createProject(req.body, req.user.id);
    res.json({ success: true, project });
  }
);
```

💰 Finance Service

```
// Ejemplo: Gestión financiera
app.get('/budget/:projectId',
  auth.authenticate,
  auth.requirePermission('finance.read'),
  async (req, res) => {
    // Obtener presupuesto del proyecto
    const budget = await getBudget(req.params.projectId);
    res.json({ success: true, budget });
  }
);
```

📝 Document Service

```
// Ejemplo: Gestión de documentos
app.upload('/documents',
  auth.authenticate,
  auth.requireRole(['admin', 'moderator']),
  async (req, res) => {
    // Subir documento
    const document = await uploadDocument(req.file, req.user.id);
    res.json({ success: true, document });
  }
);
```

🆘 Solución de Problemas

❌ Error: "Servicio de autenticación no disponible"

- Verifica que el Auth Service esté corriendo
- Revisa la URL en AUTH_SERVICE_URL
- Asegúrate que el contenedor puede conectar a auth-service

❌ Error: "Token inválido o expirado"

- El token JWT expiró, haz login de nuevo
- Verifica que el token se esté enviando correctamente

- Revisa los headers en la request

❌ Error: "Sin permisos para..."

- El usuario no tiene el permiso requerido
- Verifica que el permiso esté registrado en la DB
- Asegúrate que el rol del usuario tenga ese permiso

❌ Error: "Ruta no encontrada"

- Verifica que agregaste las rutas al API Gateway
- Asegúrate que el nombre del servicio coincida
- Revisa que el puerto esté correcto

🎯 Siguientes Pasos

1. **Crea tu primer microservicio** siguiendo esta guía
2. **Extiende la funcionalidad** según tus necesidades
3. **Agrega más permisos** granulares si es necesario
4. **Implementa base de datos** específica para tu servicio
5. **Crea tests** automatizados para tu API
6. **Documenta** tu microservicio para otros desarrolladores

💡 Tips Adicionales

🚀 Buenas Prácticas:

- **Usa nombres descriptivos** para permisos (`projects.create`, no `create`)
- **Implementa logging** detallado para debugging
- **Maneja errores** de forma consistente
- **Valida input** usando Joi o similar
- **Usa variables de entorno** para configuración

🔒 Seguridad:

- **Nunca omitas** la autenticación en rutas sensibles
- **Usa permisos granulares** en lugar de solo roles
- **Valida** todos los inputs del usuario
- **Registra** accesos y acciones importantes

📊 Monitoreo:

- **Implementa health checks** completos
- **Agrega métricas** de rendimiento
- **Usa logs estructurados** (JSON)
- **Monitorea** el uso de memoria y CPU

¡Con esta guía, cualquier desarrollador puede crear microservicios que se integren perfectamente con tu ecosistema de autenticación! 🎉