**IES FRANCISCO DE QUEVEDO**



Proyecto final

Neuropod - PaaS con Kubernetes

18 de mayo de 2025

Zidan Marvin David Avendaño Vilchez

TutoR: CésAr López

CICLO: 2024/25.

**INDICE**

[1. Introducción 6](#_Toc200369063)

[2. Objetivos 6](#_Toc200369064)

[Objetivo principal 6](#_Toc200369065)

[Objetivos específicos 6](#_Toc200369066)

[3. Estado del Arte 6](#_Toc200369067)

[4. Planteamiento / Arquitectura del Proyecto 7](#_Toc200369068)

[Stack Tecnológico Implementado 7](#_Toc200369069)

[Frontend: Aplicación React Moderna 7](#_Toc200369070)

[Backend: API REST con Node.js 7](#_Toc200369071)

[Infraestructura: Kubernetes Local + Cloudflare 8](#_Toc200369072)

[Arquitectura de Sistema Simplificada (Esquema) 8](#_Toc200369073)

[Arquitectura de Sistema Completa (Esquema) 9](#_Toc200369074)

[Flujo de Datos Principal: 9](#_Toc200369075)

[Arquitectura de Red Implementada 9](#_Toc200369076)

[Configuración de Dominios y Subdominios 9](#_Toc200369077)

[Arquitectura de Base de Datos MongoDB (Esquema) 11](#_Toc200369078)

[Modelo de Datos Implementado 11](#_Toc200369079)

[Arquitectura de Seguridad Aplicada (Esquema) 12](#_Toc200369080)

[Autenticación Multi-Capa 12](#_Toc200369081)

[Arquitectura de Comunicación en Tiempo Real (Esquema) 13](#_Toc200369082)

[WebSocket con Socket.IO 13](#_Toc200369083)

[Configuración de Aplicación Express 13](#_Toc200369084)

[App.js Implementado (Código Real) 13](#_Toc200369085)

[Configuración Vite Frontend 14](#_Toc200369086)

[Vite.config.ts Implementado (Código Real) 14](#_Toc200369087)

[Arquitectura de Infraestructura Kubernetes 15](#_Toc200369088)

[neuropod-k8s.yaml Implementado (Código Real) 15](#_Toc200369089)

[Patrones de Diseño y Arquitectura Aplicados 16](#_Toc200369090)

[Singleton Pattern (WebSocket Service) 16](#_Toc200369091)

[Observer Pattern (Custom Events) 17](#_Toc200369092)

[Factory Pattern (Pod Creation) 17](#_Toc200369093)

[Middleware Pattern (Express) 17](#_Toc200369094)

[Strategy Pattern (Authentication) 17](#_Toc200369095)

[Decisiones Arquitectónicas Clave 18](#_Toc200369096)

[Justificación de Tecnologías Elegidas 18](#_Toc200369097)

[Trade-offs y Limitaciones Aceptadas 18](#_Toc200369098)

[5. Desarrollo del Proyecto 19](#_Toc200369099)

[Configuración de Infraestructura Implementada (Esquema) 19](#_Toc200369100)

[Configuración de Minikube con GPU 19](#_Toc200369101)

[Requisitos de Sistema Implementados: 20](#_Toc200369102)

[Implementación de Cloudflare Tunnel 21](#_Toc200369103)

[Integración de Servicios Implementada 23](#_Toc200369104)

[Monitoreo y Troubleshooting 24](#_Toc200369105)

[Sistema de Autenticación Implementado 24](#_Toc200369106)

[Arquitectura de Autenticación Multi-Capa 24](#_Toc200369107)

[Control de Acceso Implementado: 25](#_Toc200369108)

[Google OAuth Verification: 26](#_Toc200369109)

[API RESTful Desarrollada (Esquema) 28](#_Toc200369110)

[Estructura de Endpoints Implementada 28](#_Toc200369111)

[Controlador de Pods Implementado 28](#_Toc200369112)

[Sistema de Precios Dinámico 30](#_Toc200369113)

[Frontend React Construido (Esquema) 31](#_Toc200369114)

[Arquitectura de Componentes Implementada 31](#_Toc200369115)

[Gestión de Estado con Hooks Personalizados 31](#_Toc200369116)

[Componentes de Pods Implementados 32](#_Toc200369117)

[Integración Kubernetes Real (Esquema) 34](#_Toc200369118)

[Servicio Kubernetes Completo 34](#_Toc200369119)

[Servicio Kubernetes Implementado (Código Real) 35](#_Toc200369120)

[Pod Controller Implementado (Código Real) 40](#_Toc200369121)

[Generación de Subdominios Dinámicos 43](#_Toc200369122)

[Base de Datos MongoDB Aplicada 44](#_Toc200369123)

[Modelo de Pod Avanzado 44](#_Toc200369124)

[Sistema de Precios Dinámico 44](#_Toc200369125)

[Comunicación en Tiempo Real (WebSockets) 46](#_Toc200369126)

[Servidor Socket.IO Implementado 46](#_Toc200369127)

[Eventos Específicos de Pods 47](#_Toc200369128)

[Cliente WebSocket React 48](#_Toc200369129)

[Scripts de Automatización Desarrollados 49](#_Toc200369130)

[Arrancar.ps1 - Inicio Orquestado 49](#_Toc200369131)

[Detener.ps1 - Cierre Coordinado 50](#_Toc200369132)

[Métricas y Análisis del Desarrollo (Esquema) 51](#_Toc200369133)

[Funcionalidades Core Implementadas 51](#_Toc200369134)

[Tecnologías Integradas 51](#_Toc200369135)

[Análisis de Decisiones Técnicas 51](#_Toc200369136)

[Decisiones de Arquitectura Justificadas 51](#_Toc200369137)

[6. Conclusiones 52](#_Toc200369138)

[Resumen Ejecutivo 52](#_Toc200369139)

[Competencias ASIR Aplicadas en NeuroPod 52](#_Toc200369140)

[Administración de Sistemas 52](#_Toc200369141)

[Redes y Comunicaciones 53](#_Toc200369142)

[Scripting y Automatización 53](#_Toc200369143)

[Conocimientos Nuevos Adquiridos 54](#_Toc200369144)

[Orquestación de Contenedores con Kubernetes 54](#_Toc200369145)

[Comunicación en Tiempo Real 54](#_Toc200369146)

[Tecnologías de Exposición Externa 54](#_Toc200369147)

[Problemas Reales Enfrentados y Solucionados 54](#_Toc200369148)

[Problema: JSON.stringify(Infinity) → null 54](#_Toc200369149)

[Problema: Certificados TLS para Subdominios Wildcard 55](#_Toc200369150)

[Problema: WebSocket + NGINX Ingress Integration 56](#_Toc200369151)

[Problema: Gestión de Procesos en PowerShell 56](#_Toc200369152)

[Conclusión Final 57](#_Toc200369153)

[7. Bibliografía 57](#_Toc200369154)

[Metodología de Referencias 57](#_Toc200369155)

[Documentación Oficial de Tecnologías Implementadas 58](#_Toc200369156)

[Frontend Development 58](#_Toc200369157)

[Backend Development 58](#_Toc200369158)

[Infrastructure and Orchestration 58](#_Toc200369159)

[Cloud Services and Networking 59](#_Toc200369160)

[Recursos Técnicos Específicos Aplicados 59](#_Toc200369161)

[Component Libraries and UI Frameworks 59](#_Toc200369162)

[Development Tools and Build Systems 59](#_Toc200369163)

[Security and Authentication 60](#_Toc200369164)

[Recursos de Containerización y Kubernetes 60](#_Toc200369165)

[Kubernetes Manifests and Configuration 60](#_Toc200369166)

[Container Runtime and Storage 60](#_Toc200369167)

[Design System and Styling Resources 60](#_Toc200369168)

[CSS Framework and Animation 60](#_Toc200369169)

[Form Handling and Validation 61](#_Toc200369170)

[Real-time Communication and WebSockets 61](#_Toc200369171)

[Socket.IO Implementation 61](#_Toc200369172)

[Automation and Scripting Resources 61](#_Toc200369173)

[PowerShell Automation 61](#_Toc200369174)

[OpenSSL and TLS Configuration 61](#_Toc200369175)

[Development Patterns and Best Practices 61](#_Toc200369176)

[React Patterns and Architecture 61](#_Toc200369177)

[Node.js API Design 61](#_Toc200369178)

[Networking and DNS Configuration 62](#_Toc200369179)

[DNS and Subdomain Management 62](#_Toc200369180)

[Ingress and Load Balancing 62](#_Toc200369181)

[Package Management and Configuration 62](#_Toc200369182)

[npm and Node.js Ecosystem 62](#_Toc200369183)

[TypeScript Configuration 62](#_Toc200369184)

[Academic and Technical References 62](#_Toc200369185)

[Software Architecture and Design Patterns 62](#_Toc200369186)

[Web Development and Modern JavaScript 63](#_Toc200369187)

[Standards and Specifications 63](#_Toc200369188)

[Web Standards 63](#_Toc200369189)

[HTTP and WebSocket Protocols 63](#_Toc200369190)

[Troubleshooting and Problem Resolution Resources 63](#_Toc200369191)

[Stack Overflow Specific Solutions 63](#_Toc200369192)

[GitHub Issues and Solutions 63](#_Toc200369193)

[8. Anexos 64](#_Toc200369194)

# 1. Introducción

Este proyecto consiste en diseñar e implementar una plataforma web escalable que permita ejecutar contenedores de forma remota, desde cualquier lugar, a través de un navegador. Inspirado en servicios como RunPod, busca ofrecer una solución accesible y eficiente para trabajar con modelos de generación de imágenes basados en inteligencia artificial, sin depender exclusivamente de la ejecución local. El sistema está pensado como una Plataforma como Servicio (PaaS), gestionada desde un único equipo físico, pero con una arquitectura modular basada en contenedores, autenticación segura, y acceso web mediante interfaz moderna.

# 2. Objetivos

## Objetivo principal

Desarrollar una aplicación web modular y escalable que permita a los usuarios gestionar contenedores personalizados (como ComfyUI y Ubuntu) o propios a través de una interfaz gráfica remota.

## Objetivos específicos

* Ejecutar contenedores Docker con acceso remoto mediante subdominios personalizados.
* Implementar un sistema de autenticación OAuth2 con Google.
* Gestionar sesiones seguras con JWT.
* Gestionar los pods dinámicamente desde Node.js usando la API de Kubernetes.
* Proporcionar persistencia de datos vía PVC montado en /workspace.
* Controlar el uso mediante sistema de saldo por usuario.
* Exponer los servicios mediante Cloudflare Tunnel y configurar DNS wildcard.

# 3. Estado del Arte

Actualmente, existen plataformas como RunPod, ComfyUI Online, y Hugging Face Spaces, que permiten ejecutar interfaces de IA desde la nube. Estas herramientas aprovechan tecnologías modernas de contenedores (Docker), orquestación (Kubernetes), e interfaces gráficas basadas en la web. Las metodologías DevOps y la computación serverless también están en auge, facilitando despliegues rápidos, escalado automático y reducción de costes de infraestructura. Sin embargo, muchas de estas soluciones requieren recursos económicos considerables o conocimientos avanzados. Este proyecto se plantea como una alternativa educativa y personal a estas soluciones comerciales, pero aplicando las mismas tecnologías utilizadas en la industria, adaptadas a un único servidor físico.

# 4. Planteamiento / Arquitectura del Proyecto

## Stack Tecnológico Implementado

### Frontend: Aplicación React Moderna

#### Tecnologías Core:

* **React 18.3.1** con **TypeScript 5.5.3** para tipado estático.
* **Vite 5.4.1** como bundler moderno con plugin React SWC para compilación rápida.
* **TailwindCSS 3.4.11** para estilización utilitaria y responsiva.
* **shadcn-ui** como sistema de componentes basado en Radix UI primitives.

#### Dependencias de Funcionalidad:

* **@react-oauth/google 0.12.2** para autenticación OAuth2 con Google.
* **axios 1.9.0** para comunicación HTTP con el backend.
* **socket.io-client 4.8.1** para comunicación en tiempo real.
* **react-router-dom 6.26.2** para navegación SPA (Single Page Application).
* **@tanstack/react-query 5.56.2** para gestión de estado del servidor.

#### Arquitectura de Componentes:

* **40+ componentes** organizados jerárquicamente por roles (admin/, client/).
* **Context API** (AuthContext) para gestión de estado global de autenticación.
* **Custom Hooks** para WebSockets, notificaciones globales y actualizaciones de pods.
* **Services Layer** para abstracción de APIs (api.ts, pod.service.ts, pricing.service.ts).

### Backend: API REST con Node.js

#### Tecnologías Core:

* **Node.js 22** con **Express 4.18.2** para servidor HTTP.
* **MongoDB** con **Mongoose 8.0.3** como ODM (Object Document Mapper).
* **Socket.IO 4.8.1** para comunicación bidireccional en tiempo real.
* **@kubernetes/client-node 1.2.0** para integración con Kubernetes API.

#### Dependencias de Seguridad:

* **jsonwebtoken 9.0.2** para autenticación JWT.
* **google-auth-library 9.15.1** para verificación de tokens OAuth2.
* **cors 2.8.5** para configuración de CORS.
* **dotenv 16.3.1** para gestión de variables de entorno.

#### Arquitectura de Servicios:

* **5 controladores** especializados (auth, pod, pricing, status, template)
* **6 modelos MongoDB** con esquemas Mongoose validados
* **Middleware de autenticación** con protección basada en roles
* **Servicios de Kubernetes** para gestión del ciclo de vida de pods

### Infraestructura: Kubernetes Local + Cloudflare

#### Orquestación de Contenedores:

* **Minikube** como clúster Kubernetes local con driver Docker.
* **NGINX Ingress Controller** para enrutamiento de tráfico HTTP/HTTPS.
* **Docker Desktop** con integración WSL2 y soporte GPU.
* **Persistent Volumes** con StorageClass hostPath para persistencia de datos.

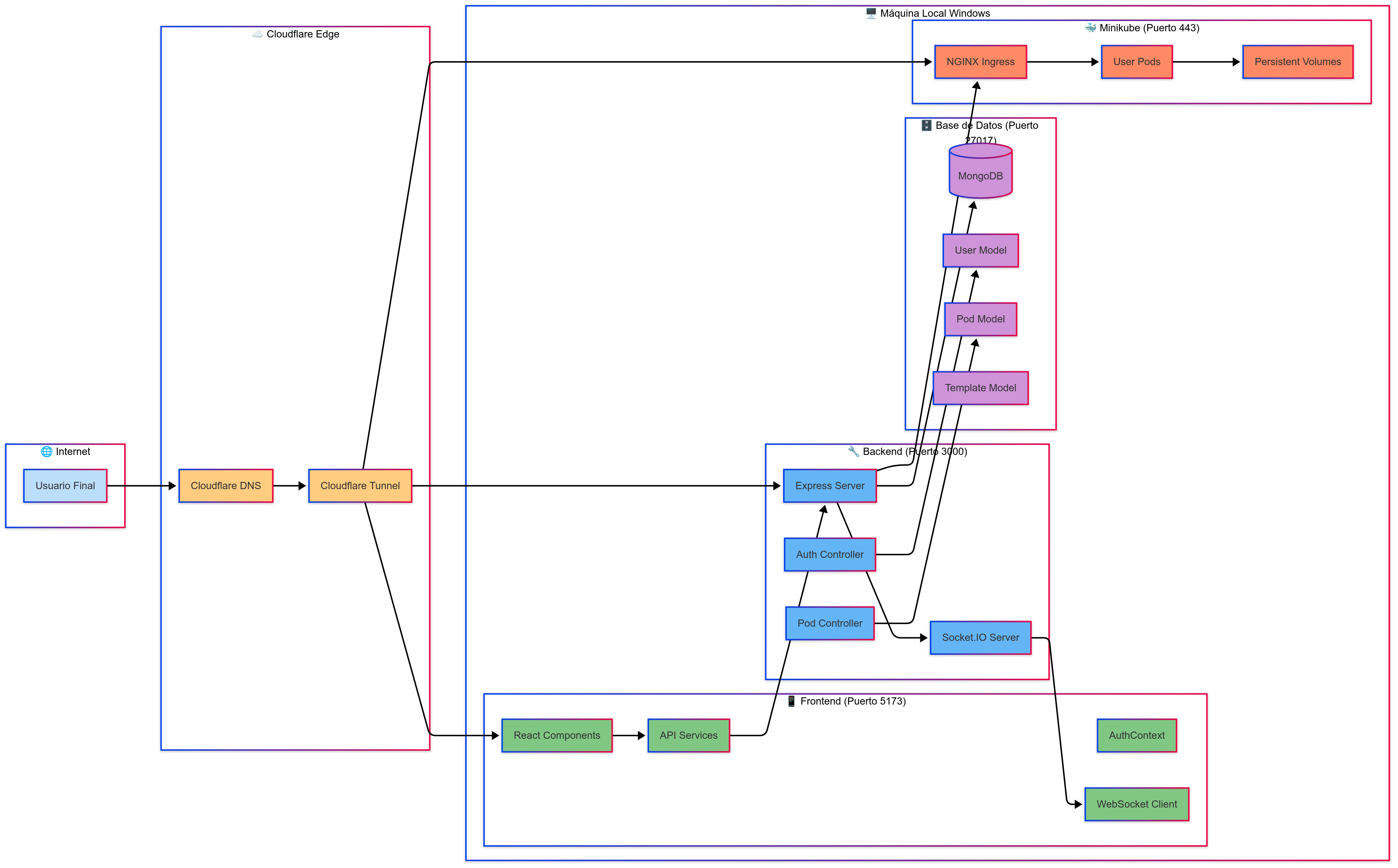
#### Exposición Externa:

* **Cloudflare Tunnel** para conectividad segura sin IP pública.
* **Subdominios dinámicos** (\*.neuropod.online) para cada pod de usuario.
* **Certificados TLS autofirmados** con secret de Kubernetes.
* **DNS wildcard** con configuración Cloudflare.

## Arquitectura de Sistema Simplificada (Esquema)



## Arquitectura de Sistema Completa (Esquema)



### Flujo de Datos Principal:

1. **Usuario** → Cloudflare DNS → Tunnel → NGINX Ingress → Pod de Usuario.
2. **Frontend React** → Axios → Express API → MongoDB → Respuesta.
3. **Socket.IO** → WebSocket bidireccional → Actualizaciones en tiempo real.
4. **Kubernetes API** → Creación/gestión de pods → Persistent Volumes.

## Arquitectura de Red Implementada

### Configuración de Dominios y Subdominios

#### Estructura de URLs:

https://app.neuropod.online → Frontend React (puerto 5173)

https://api.neuropod.online → Backend Express (puerto 3000)

https://\*.neuropod.online → Pods dinámicos (puerto 443)

https://pod-usr123-8888.neuropod.online → Ejemplo de pod específico

#### Cloudflare Tunnel Configuration:

# ~/.cloudflared/config.yml (Windows: %USERPROFILE%\.cloudflared\config.yml)

tunnel: neuropod-tunnel

credentials-file: C:\Users\<tu-usuario>\.cloudflared\neuropod-tunnel.json

ingress:

# Frontend React

- hostname: app.neuropod.online

service: http://localhost:5173

# Backend API con soporte WebSocket

- hostname: api.neuropod.online

service: http://localhost:3000

originRequest:

noTLSVerify: true

connectTimeout: 30s

tlsTimeout: 30s

tcpKeepAlive: 30s

# 🔧 IMPORTANTE: Configuración para WebSockets

disableChunkedEncoding: true

http2Origin: false

# 🔧 NUEVO: Soporte explícito para WebSocket

upgradeRequest: true

# Wildcard para pods

- hostname: "\*.neuropod.online"

service: https://localhost:443

originRequest:

noTLSVerify: true # ← CRÍTICO: acepta certificados autofirmados

connectTimeout: 30s

tlsTimeout: 30s

tcpKeepAlive: 30s

disableChunkedEncoding: true

http2Origin: false

# Fallback

- service: http\_status:404

**Nota**: Los puertos especificados (5173, 3000, 443) deben coincidir con los puertos donde se ejecutan tus servicios localmente.

#### NGINX Ingress Controller:

# Configuración optimizada para Cloudflare Tunnel aplicada en neuropod-k8s.yaml

data:

ssl-redirect: "false"

force-ssl-redirect: "false"

use-forwarded-headers: "true"

compute-full-forwarded-for: "true"

server-name-hash-bucket-size: "256"

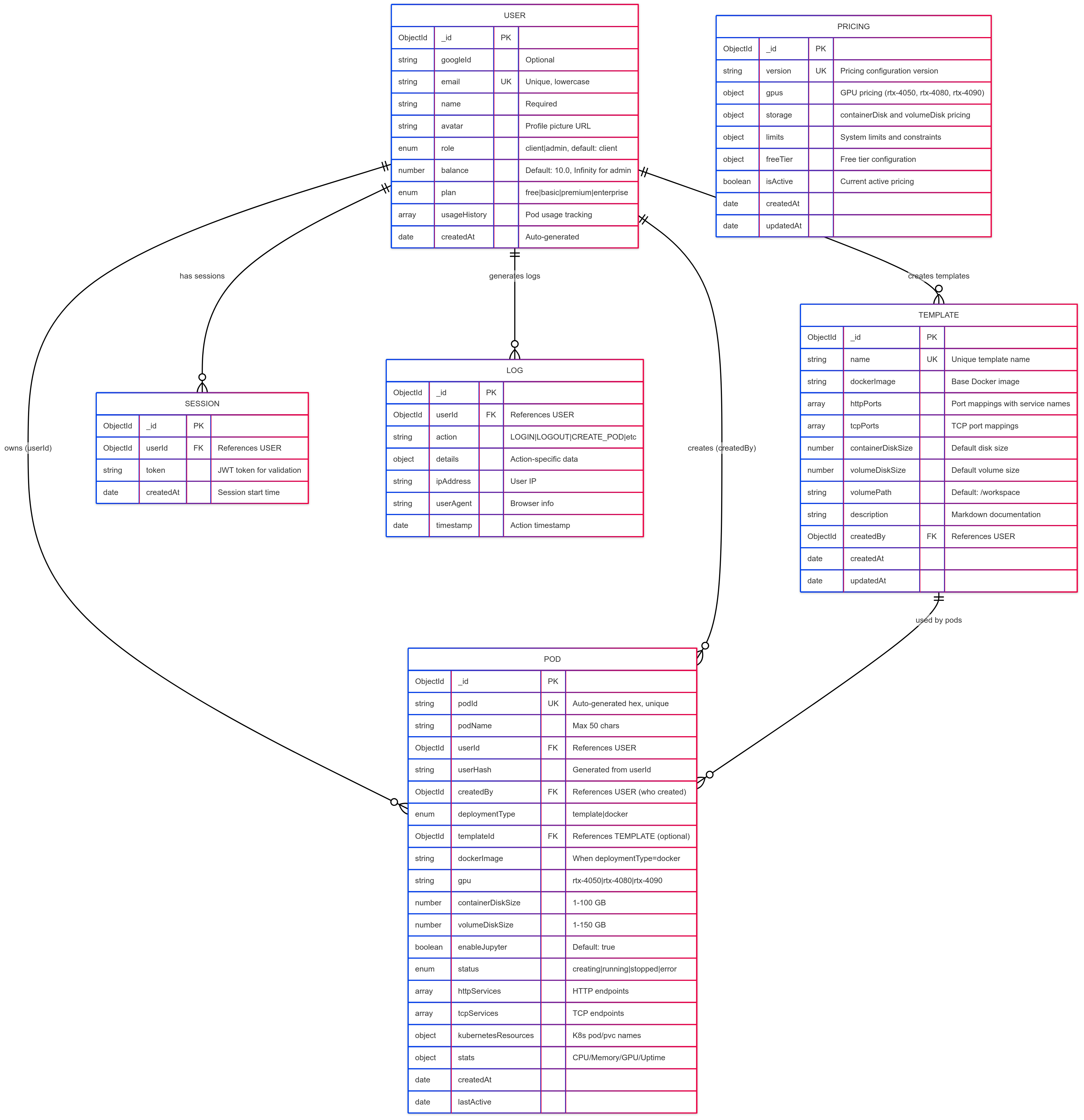
proxy-read-timeout: "3600"

proxy-send-timeout: "3600"

proxy-http-version: "1.1"

## Arquitectura de Base de Datos MongoDB (Esquema)

### Modelo de Datos Implementado



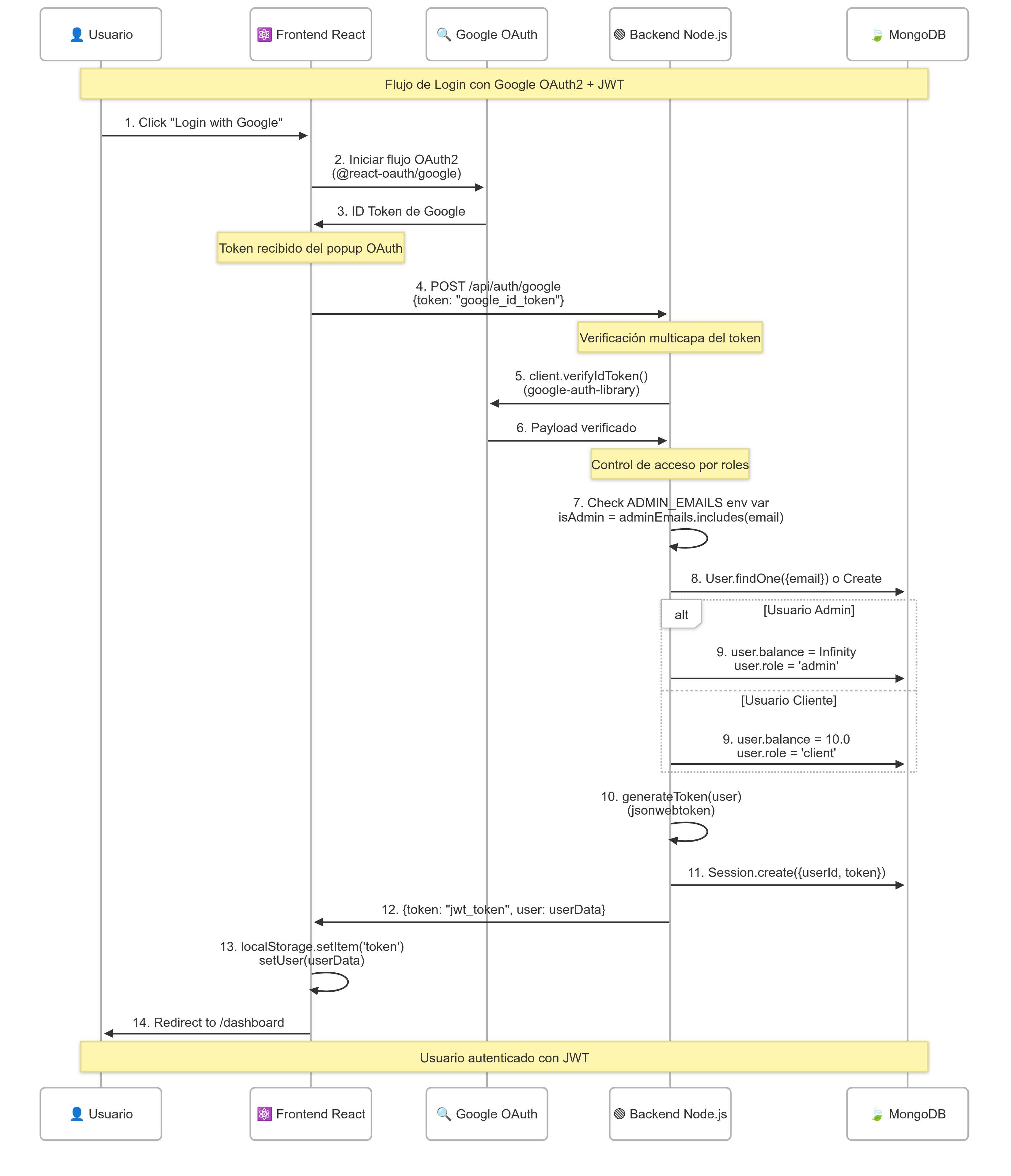
#### Relaciones y Validaciones:

* **Usuario-Pod**: Relación 1:N con diferenciación entre propietario (userId) y creador (createdBy).
* **Template-Pod**: Referencia para configuración predefinida.
* **Validaciones Mongoose**: Campos requeridos, únicos, y rangos numéricos.
* **Middleware pre-save**: Hash de contraseñas, actualización de timestamps.

## Arquitectura de Seguridad Aplicada (Esquema)

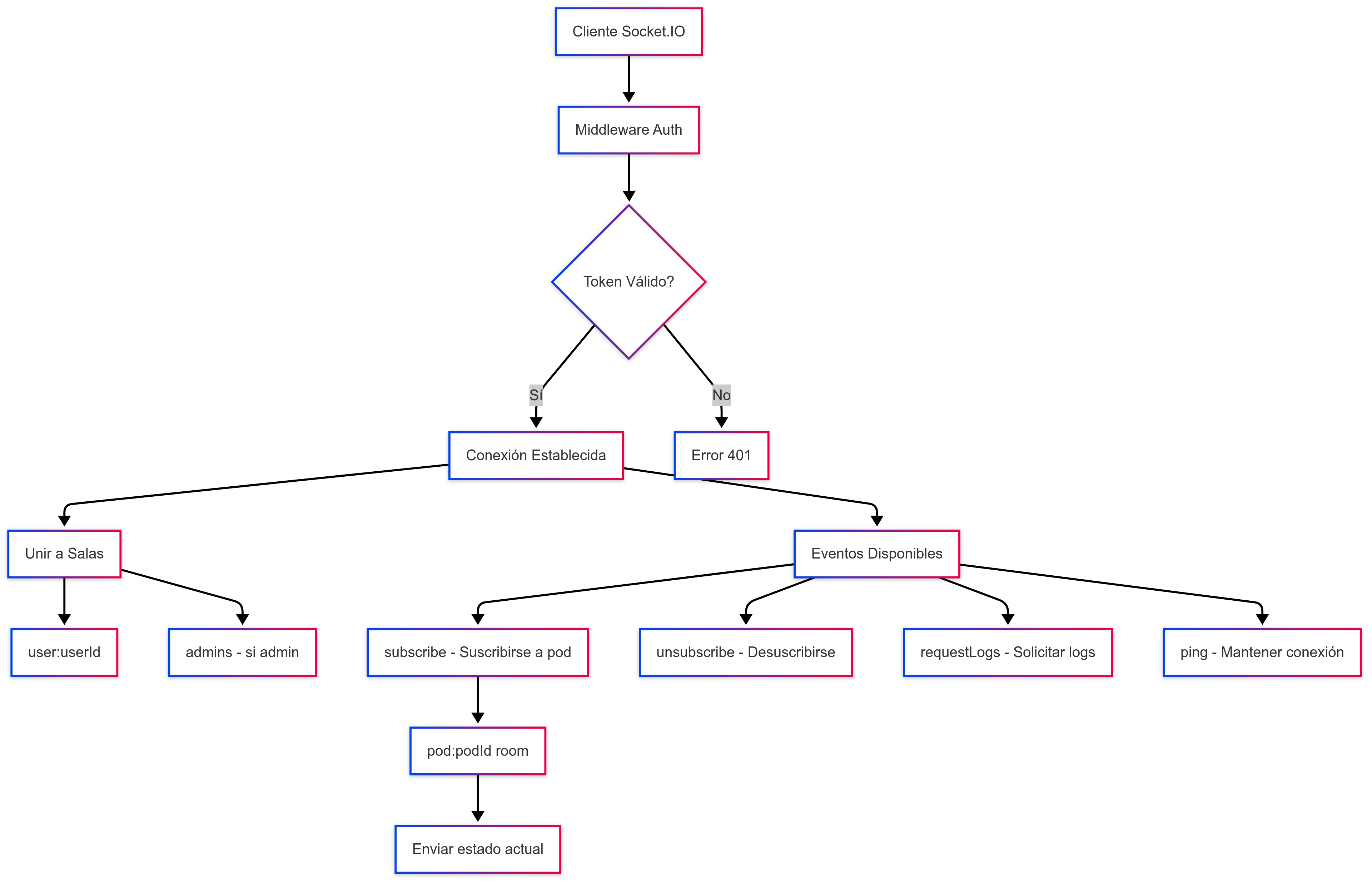
### Autenticación Multi-Capa

#### Flujo OAuth2 + JWT:



## Arquitectura de Comunicación en Tiempo Real (Esquema)

### WebSocket con Socket.IO



## Configuración de Aplicación Express

### App.js Implementado (Código Real)

// Backend: src/app.js - Configuración completa de Express

const express = require('express');

const cors = require('cors');

const dotenv = require('dotenv');

dotenv.config();

const app = express();

// Middleware JSON

app.use(express.json());

// CORS dinámico para múltiples orígenes

const allowedOrigins = [

process.env.FRONTEND\_URL, // http://localhost:5173

process.env.FRONTEND\_URL\_HTTPS, // https://app.neuropod.online

'http://localhost:3000' // Para testing directo

].filter(Boolean);

app.use(cors({

origin: function (origin, callback) {

// Permitir requests sin origin (Postman, curl, etc.)

if (!origin) return callback(null, true);

if (allowedOrigins.indexOf(origin) !== -1) {

callback(null, true);

} else {

console.log('❌ CORS bloqueado para origen:', origin);

callback(new Error('No permitido por CORS'));

}

},

credentials: true

}));

// Logging de requests para debugging

app.use((req, res, next) => {

console.log(`${new Date().toISOString()} - ${req.method} ${req.path} desde ${req.headers.origin || 'sin origin'}`);

next();

});

// Health check endpoint

app.get('/api/health', (req, res) => {

res.status(200).json({

status: 'ok',

message: 'API de NeuroPod funcionando correctamente',

timestamp: new Date().toISOString(),

environment: process.env.NODE\_ENV || 'development'

});

});

// Rutas principales

app.use('/api/auth', require('./routes/auth.routes'));

app.use('/api/status', require('./routes/status.routes'));

app.use('/api/pods', require('./routes/pod.routes'));

app.use('/api/templates', require('./routes/template.routes'));

app.use('/api/pricing', require('./routes/pricing.routes'));

// Middleware de manejo de errores

app.use((err, req, res, next) => {

console.error(err.stack);

res.status(500).json({ message: 'Error en el servidor' });

});

module.exports = app;

## Configuración Vite Frontend

### Vite.config.ts Implementado (Código Real)

// Frontend: vite.config.ts - Configuración optimizada

import { defineConfig } from "vite";

import react from "@vitejs/plugin-react-swc";

import path from "path";

import { componentTagger } from "lovable-tagger";

export default defineConfig(({ mode }) => ({

server: {

host: "::", // Acepta conexiones externas

port: 5173

},

plugins: [

react(), // Plugin React con SWC (más rápido que Babel)

mode === 'production' && componentTagger(), // Solo en producción

].filter(Boolean),

resolve: {

alias: {

"@": path.resolve(\_\_dirname, "./src"), // Alias para imports

},

},

}));

## Arquitectura de Infraestructura Kubernetes

### neuropod-k8s.yaml Implementado (Código Real)

#### Recursos Kubernetes Aplicados:

# ConfigMap Neuropod - Configuración global del proyecto

apiVersion: v1

kind: ConfigMap

metadata:

name: neuropod-config

namespace: default

data:

domain: "neuropod.online"

defaultStorageClass: "standard"

maxPodsPerUser: "5"

workspacePath: "/workspace"

defaultNamespace: "default"

---

# ConfigMap NGINX - Configuración específica para Cloudflare Tunnel

apiVersion: v1

kind: ConfigMap

metadata:

name: nginx-configuration

namespace: ingress-nginx

data:

# CONFIGURACIONES ESPECÍFICAS PARA CLOUDFLARE TUNNEL

ssl-redirect: "false"

force-ssl-redirect: "false"

use-forwarded-headers: "true"

compute-full-forwarded-for: "true"

# Optimizaciones para subdominios largos

server-name-hash-bucket-size: "256"

proxy-buffer-size: "16k"

# WebSockets (Jupyter Lab)

proxy-read-timeout: "3600"

proxy-send-timeout: "3600"

proxy-http-version: "1.1"

proxy-set-headers: "ingress-nginx/custom-headers"

---

# StorageClass para volúmenes persistentes

apiVersion: storage.k8s.io/v1

kind: StorageClass

metadata:

name: standard

provisioner: k8s.io/minikube-hostpath

reclaimPolicy: Retain

volumeBindingMode: Immediate

allowVolumeExpansion: true

---

# PersistentVolume global para workspace

apiVersion: v1

kind: PersistentVolume

metadata:

name: neuropod-pv-global

spec:

capacity:

storage: 500Gi

accessModes:

- ReadWriteMany

persistentVolumeReclaimPolicy: Retain

storageClassName: standard

hostPath:

path: /mnt/data/workspace

type: DirectoryOrCreate

---

# Secret TLS con certificado autofirmado para \*.neuropod.online

apiVersion: v1

kind: Secret

metadata:

name: neuropod-tls

namespace: default

type: kubernetes.io/tls

data:

tls.crt: LS0tLS1CRUdJTi... # Certificado base64 wildcard

tls.key: LS0tLS1CRUdJTi... # Clave privada base64

## Patrones de Diseño y Arquitectura Aplicados

### Singleton Pattern (WebSocket Service)

// Frontend: services/websocket.service.ts

class WebSocketService {

private static instance: WebSocketService;

static getInstance(): WebSocketService {

if (!WebSocketService.instance) {

WebSocketService.instance = new WebSocketService();

}

return WebSocketService.instance;

}

}

export default new WebSocketService(); // Exportar instancia singleton

### Observer Pattern (Custom Events)

// Frontend: Eventos custom para comunicación entre componentes

window.dispatchEvent(new CustomEvent('podUpdate', { detail: data }));

window.addEventListener('podUpdate', handleUpdate as EventListener);

### Factory Pattern (Pod Creation)

// Backend: Diferentes tipos de pods según configuración

const { finalDockerImage, httpServices, tcpServices } =

await processPodConfiguration(req.body);

if (deploymentType === 'template') {

finalDockerImage = template.dockerImage;

httpServices = assignServiceNames(portsArray, template.httpPorts);

} else {

finalDockerImage = payload.dockerImage;

httpServices = assignServiceNamesDocker(portsArray);

}

### Middleware Pattern (Express)

// Backend: Chain of responsibility en rutas

router.use(protect); // Autenticación

router.use(authorize('admin')); // Autorización

router.post('/', validatePodPayload); // Validación

### Strategy Pattern (Authentication)

// Backend: Múltiples estrategias de verificación de tokens

try {

// Estrategia 1: ID Token

const ticket = await client.verifyIdToken({...});

payload = ticket.getPayload();

} catch (idTokenError) {

try {

// Estrategia 2: Access Token

const response = await fetch(`https://www.googleapis.com/oauth2/v3/tokeninfo?access\_token=${token}`);

payload = await response.json();

} catch (accessTokenError) {

// Estrategia 3: Manual JWT decode

payload = JSON.parse(decodeURIComponent(...));

}

}

## Decisiones Arquitectónicas Clave

### Justificación de Tecnologías Elegidas

#### React 18 + TypeScript:

* **Tipado estático** previene errores en tiempo de compilación
* **Hooks modernos** (useContext, useEffect) para gestión de estado
* **Component composition** para reutilización y mantenibilidad
* **Vite** ofrece hot reload rápido y build optimizado

#### Express + MongoDB:

* **Express** proporciona flexibilidad para API REST y WebSockets
* **MongoDB** con esquemas flexibles ideal para datos dinámicos de pods
* **Mongoose** ODM ofrece validaciones y middleware pre/post hooks
* **Arquitectura sin estado** permite escalabilidad horizontal

#### Minikube + Docker:

* **Desarrollo local** sin necesidad de clúster cloud
* **Minikube** simula entorno Kubernetes real
* **Docker Desktop** integración nativa con Windows + WSL2
* **NGINX Ingress** enrutamiento flexible de subdominios

#### Cloudflare Tunnel:

* **Sin IP pública** - seguridad mejorada
* **Conexión outbound** - no requiere configuración de firewall
* **SSL termination** automático en edge de Cloudflare
* **Subdominios dinámicos** para cada pod de usuario

### Trade-offs y Limitaciones Aceptadas

#### Certificados Autofirmados:

* **Ventaja**: Desarrollo local simplificado
* **Limitación**: Warnings de navegador, no válido para producción
* **Justificación**: Cloudflare Tunnel maneja SSL real en edge

#### MongoDB Local:

* **Ventaja**: Datos locales, desarrollo offline
* **Limitación**: No replicación, no alta disponibilidad
* **Justificación**: Entorno de desarrollo, fácil migración a MongoDB Atlas

#### Minikube Single-Node:

* **Ventaja**: Recursos mínimos, setup simple
* **Limitación**: No simula cluster multi-nodo real
* **Justificación**: Suficiente para validar conceptos y desarrollo

#### Persistent Volumes HostPath:

* **Ventaja**: Persistencia real en desarrollo
* **Limitación**: No funciona en cluster multi-nodo
* **Justificación**: Minikube single-node, datos persisten entre reinicios

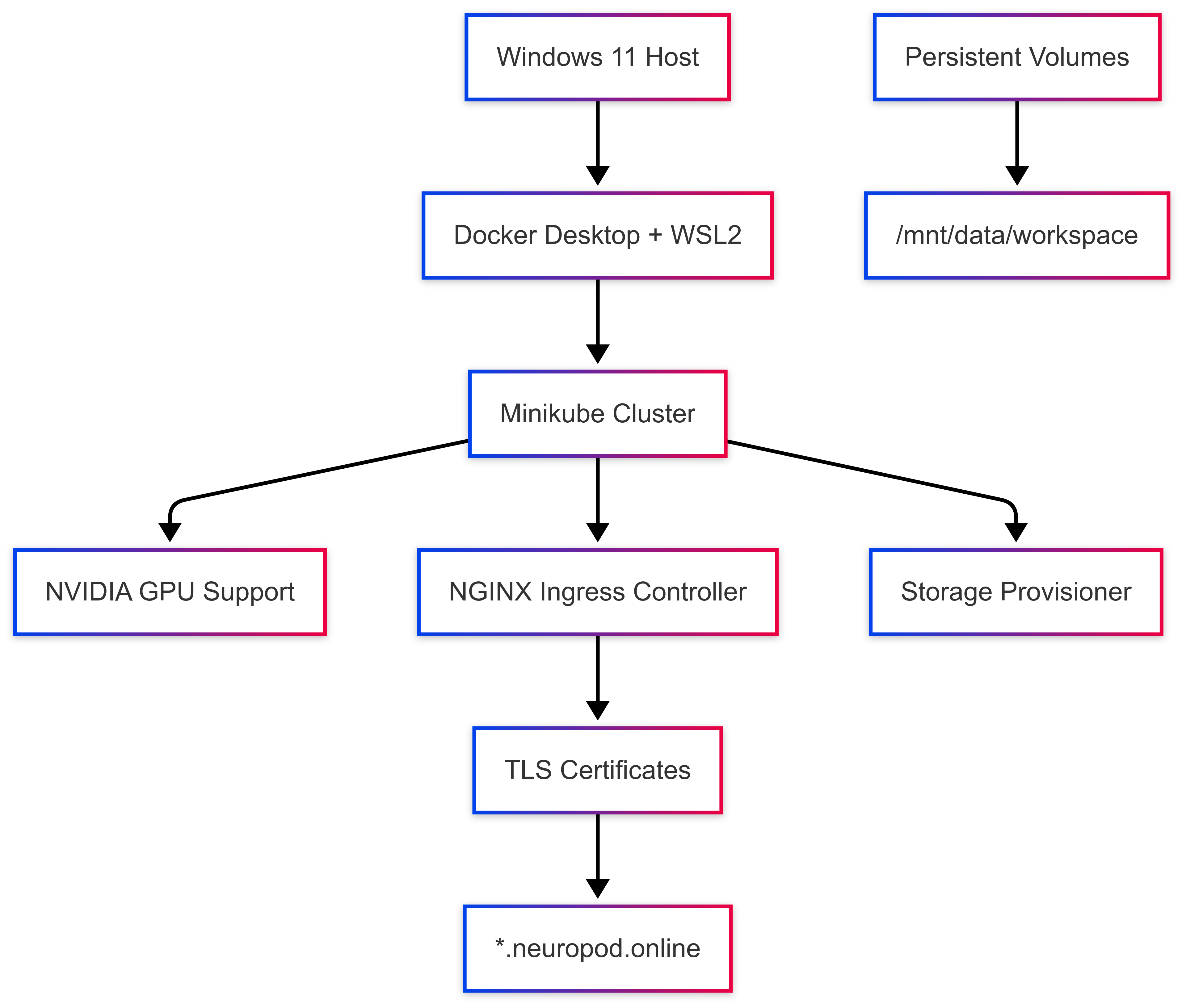
# 5. Desarrollo del Proyecto

## Configuración de Infraestructura Implementada (Esquema)

Esta sección documenta la configuración completa de la infraestructura que soporta NeuroPod, incluyendo el entorno Kubernetes local y la exposición segura a Internet.

### Configuración de Minikube con GPU

#### Arquitectura de Infraestructura Local:



### Requisitos de Sistema Implementados:

**Nota**: Revisar [GUIA MINIKUBE COMPLETA](https://github.com/Zidane0MA/NeuroPod/blob/main/Documentacion/GUIA_MINIKUBE_CONFIGURACION_HECHO.md).

#### Configuración WSL2 para GPU:

# ~/.wslconfig - Configuración de memoria y GPU

[wsl2]

memory=16GB

processors=8

[gpu]

enabled=true

#### Inicialización de Minikube Optimizada:

# Comando de inicio con todas las configuraciones

minikube start `

--driver=docker `

--container-runtime=docker `

--gpus=all `

--memory=12000mb `

--cpus=8 `

--addons=ingress,storage-provisioner,default-storageclass,nvidia-device-plugin

# Verificación de GPU disponible

kubectl get nodes -o jsonpath='{.items[\*].status.allocatable.nvidia\.com/gpu}'

# Resultado esperado: 1

#### Configuración de Volúmenes Persistentes:

# Crear directorio para workspace en Minikube

minikube ssh "sudo mkdir -p /mnt/data/workspace && sudo chmod 777 /mnt/data/workspace"

#### Certificados TLS Wildcard Implementados:

# Generación de certificado autofirmado con SAN wildcard

@"

[req]

distinguished\_name = req\_distinguished\_name

req\_extensions = v3\_req

prompt = no

[req\_distinguished\_name]

CN = \*.neuropod.online

[v3\_req]

keyUsage = keyEncipherment, dataEncipherment

extendedKeyUsage = serverAuth

subjectAltName = @alt\_names

[alt\_names]

DNS.1 = \*.neuropod.online

DNS.2 = neuropod.online

"@ | Out-File -FilePath "openssl.conf" -Encoding ascii

# Crear certificados válidos por 1 año

openssl req -x509 -nodes -days 365 -newkey rsa:2048 `

-keyout tls.key -out tls.crt -config openssl.conf -extensions v3\_req

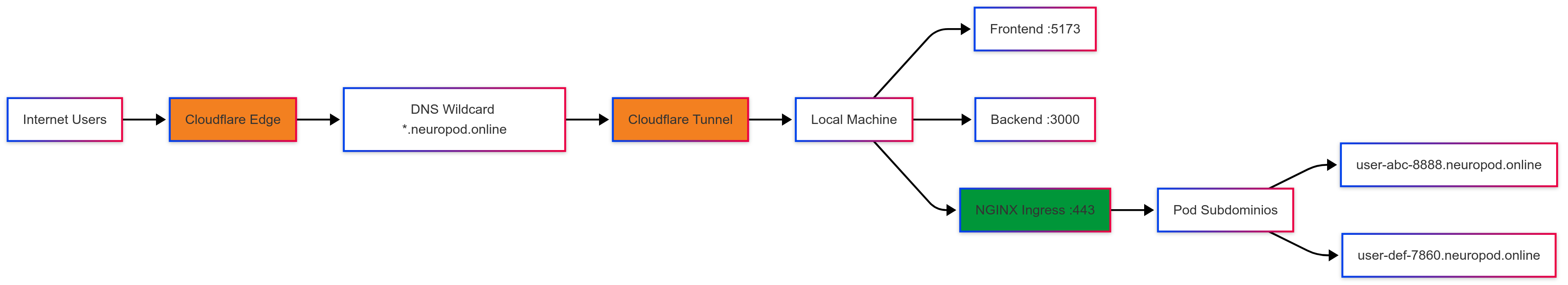
# Verificar SAN wildcard

openssl x509 -in tls.crt -text -noout | findstr "DNS"

### Implementación de Cloudflare Tunnel

**Nota**: Revisar [GUIA CLOUDFLARE COMPLETA](https://github.com/Zidane0MA/NeuroPod/blob/main/Documentacion/GUIA_COMPLETA_CLOUDFLARE_TUNNEL_HECHO.md).

#### Arquitectura de Exposición a Internet:



#### Configuración DNS en Cloudflare:

| **Tipo** | **Nombre** | **Destino** | **Proxy** | **Propósito** |
| --- | --- | --- | --- | --- |
| CNAME | app | <tunnel-id>.cfargotunnel.com | 🟠 Activado | Frontend React |
| CNAME | api | <tunnel-id>.cfargotunnel.com | 🟠 Activado | Backend API + WebSockets |
| CNAME | \* | <tunnel-id>.cfargotunnel.com | 🟠 Activado | Pods dinámicos |
| A | @ | 192.0.2.1 | 🟠 Activado | Redirección raíz |
| A | www | 192.0.2.1 | 🟠 Activado | Redirección www |

* **Page Rules para Redirecciones:**

# Redirección del dominio raíz a la aplicación

neuropod.online/\* → https://app.neuropod.online/$1 (301)

www.neuropod.online/\* → https://app.neuropod.online/$1 (301)

* **Configuración del Túnel (config.yml):**

# ~/.cloudflared/config.yml

tunnel: neuropod-tunnel

credentials-file: C:\Users\<usuario>\.cloudflared\neuropod-tunnel.json

ingress:

# Frontend React (puerto 5173)

- hostname: app.neuropod.online

service: http://localhost:5173

# Backend API con soporte WebSocket

- hostname: api.neuropod.online

service: http://localhost:3000

originRequest:

noTLSVerify: true

connectTimeout: 30s

tlsTimeout: 30s

tcpKeepAlive: 30s

disableChunkedEncoding: true

http2Origin: false

upgradeRequest: true # Crítico para WebSockets

# Wildcard para pods dinámicos

- hostname: "\*.neuropod.online"

service: https://localhost:443

originRequest:

noTLSVerify: true # Acepta certificados autofirmados

connectTimeout: 30s

tlsTimeout: 30s

tcpKeepAlive: 30s

disableChunkedEncoding: true

http2Origin: false

# Fallback

- service: http\_status:404

* **Comandos de Gestión del Túnel:**

# Crear túnel

cloudflared.exe tunnel create neuropod-tunnel

# Listar túneles

cloudflared.exe tunnel list

# Ejecutar túnel

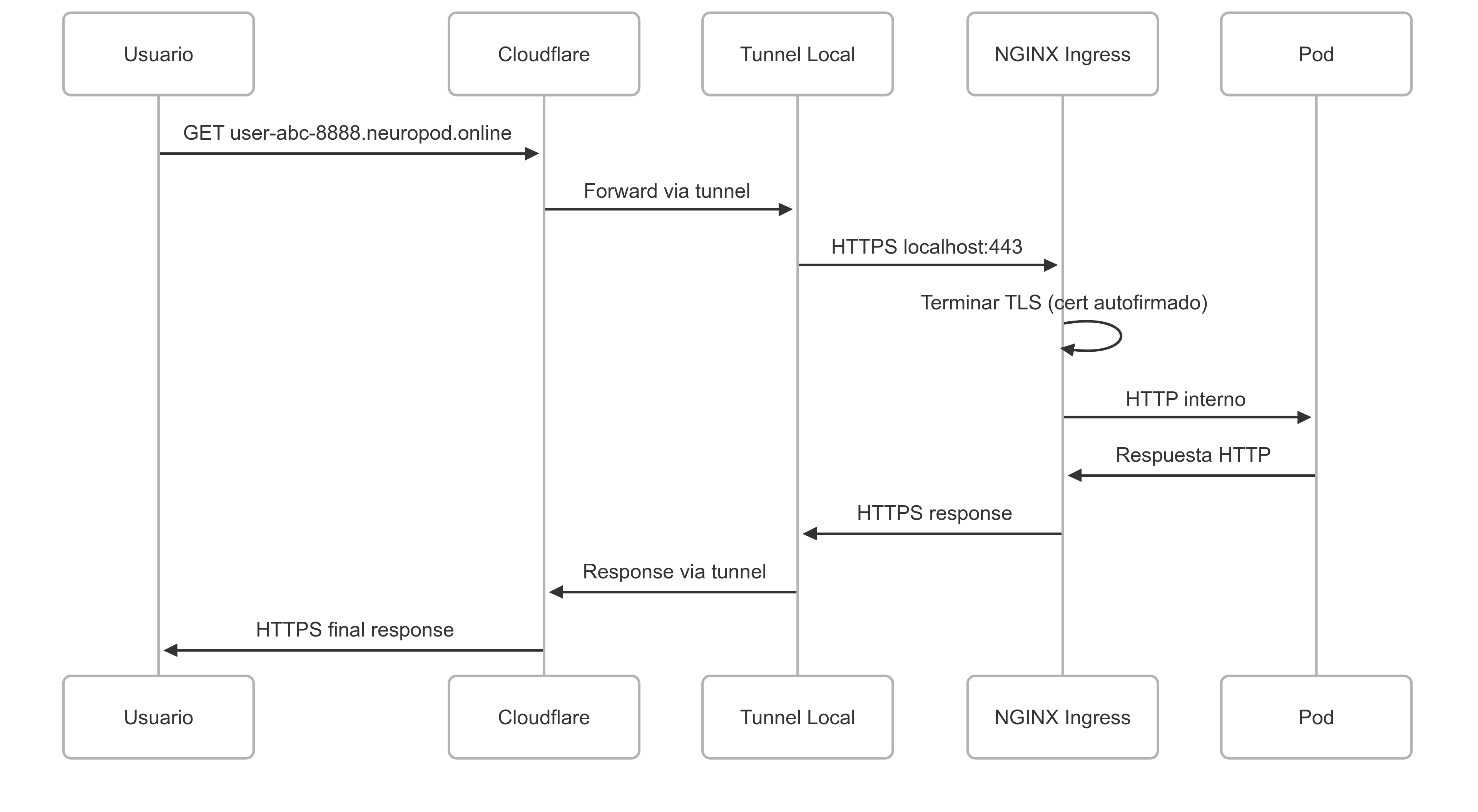
cloudflared.exe tunnel run neuropod-tunnel

# Información del túnel

cloudflared.exe tunnel info neuropod-tunnel

### Integración de Servicios Implementada

#### Flujo de Tráfico Completo:



#### Configuración de Firewall Windows:

# Abrir puertos necesarios para los servicios

New-NetFirewallRule -DisplayName "Neuropod Backend" -Direction Inbound -Protocol TCP -LocalPort 3000 -Action Allow

New-NetFirewallRule -DisplayName "Neuropod Frontend" -Direction Inbound -Protocol TCP -LocalPort 5173 -Action Allow

New-NetFirewallRule -DisplayName "Neuropod Ingress" -Direction Inbound -Protocol TCP -LocalPort 443 -Action Allow

#### Verificación de la Configuración:

# Script de verificación completa (k8s\_debugging.ps1)

Write-Host "=== Verificando infraestructura NeuroPod ===" -ForegroundColor Green

# 1. Estado de Minikube

minikube status

# 2. Verificar puerto NGINX Ingress

netstat -an | findstr :443

# 3. Verificar GPU disponible

kubectl get nodes -o jsonpath='{.items[\*].status.allocatable.nvidia\.com/gpu}'

# 4. Verificar recursos aplicados

kubectl get configmaps | findstr neuropod

kubectl get storageclass standard

kubectl get pv neuropod-pv-global

kubectl get secret neuropod-tls

# 5. Verificar NGINX Ingress Controller

kubectl get pods -n ingress-nginx

kubectl get configmap nginx-configuration -n ingress-nginx

# 6. Probar certificado TLS

$certBase64 = kubectl get secret neuropod-tls -o jsonpath="{.data.tls\.crt}"

$certBytes = [Convert]::FromBase64String($certBase64)

[System.Text.Encoding]::UTF8.GetString($certBytes) | Out-File -Encoding ascii tls.crt

openssl x509 -in tls.crt -text -noout | findstr "DNS:"

### Monitoreo y Troubleshooting

#### Logs de Diagnóstico:

# Logs de NGINX Ingress para SSL/TLS

kubectl logs -n ingress-nginx deployment/ingress-nginx-controller | Select-String "ssl|tls|cert"

# Verificar funcionamiento de tunnel

cloudflared.exe tunnel info neuropod-tunnel

# Estado de pods en tiempo real

kubectl get pods -n default -o wide --watch

# Logs de un pod específico

kubectl logs -f <pod-name> -n default

#### Problemas Comunes Resueltos:

* **Certificado wildcard no reconocido:**

# Solución: Verificar SAN en el certificado

Subject Alternative Name:

DNS:\*.neuropod.online

DNS:neuropod.online

* **WebSockets no funcionan a través del tunnel:**

# Solución: Configuración específica en tunnel

upgradeRequest: true

disableChunkedEncoding: true

http2Origin: false

* **Pods no acceden a GPU:**

# Solución: Verificar addon nvidia-device-plugin

minikube addons enable nvidia-device-plugin

## Sistema de Autenticación Implementado

### Arquitectura de Autenticación Multi-Capa

El sistema de autenticación implementado combina Google OAuth2 con JWT tokens para proporcionar seguridad robusta:

### Control de Acceso Implementado:

#### Backend (auth.middleware.js):

// Backend: middleware/auth.middleware.js

const protect = async (req, res, next) => {

try {

let token = req.headers.authorization?.split(' ')[1];

if (!token) {

return res.status(401).json({

success: false,

message: 'Token no proporcionado'

});

}

const decoded = jwt.verify(token, process.env.JWT\_SECRET);

const user = await User.findById(decoded.id);

if (!user) {

return res.status(401).json({

success: false,

message: 'Usuario no encontrado'

});

}

req.user = user;

req.token = token;

next();

} catch (error) {

res.status(401).json({

success: false,

message: 'Token inválido'

});

}

};

const authorize = (...roles) => (req, res, next) => {

if (!roles.includes(req.user.role)) {

return res.status(403).json({

success: false,

message: 'No tienes permisos para esta acción'

});

}

next();

};

#### Gestión de Estado Frontend (AuthContext.tsx)

* **Patrón Context Provider implementado**:

interface AuthContextType {

user: User | null;

isAuthenticated: boolean;

isLoading: boolean;

isOfflineMode: boolean; // Modo simulación

login: (email: string) => Promise<void>;

loginWithGoogle: (token: string) => Promise<void>;

logout: () => Promise<void>;

}

export const AuthProvider: React.FC = ({ children }) => {

const [user, setUser] = useState<User | null>(null);

const [isOfflineMode, setIsOfflineMode] = useState(false);

// Verificación automática de sesión

useEffect(() => {

const checkAuth = async () => {

const token = localStorage.getItem("token");

if (token) {

try {

const user = await authService.verifyToken();

setUser(user);

} catch (error) {

// Activar modo simulación si backend no disponible

if (import.meta.env.DEV && error.message?.includes("Backend no disponible")) {

setIsOfflineMode(true);

// Recuperar usuario del localStorage

}

}

}

};

checkAuth();

}, []);

return (

<AuthContext.Provider value={{

user, isAuthenticated: !!user, login, logout, isOfflineMode

}}>

{children}

</AuthContext.Provider>

);

};

### Google OAuth Verification:

#### Lógica de Asignación de Roles (auth.controller.js):

// Backend: controllers/auth.controller.js - Verificación multicapa

exports.googleLogin = async (req, res) => {

try {

const { token } = req.body;

let payload = null;

// 1. Intentar verificar como ID token

try {

const ticket = await client.verifyIdToken({

idToken: token,

audience: process.env.GOOGLE\_CLIENT\_ID

});

payload = ticket.getPayload();

console.log('Token verificado como ID token');

}

// 2. Si falla, intentar como token de acceso

catch (idTokenError) {

try {

const response = await fetch(`https://www.googleapis.com/oauth2/v3/tokeninfo?access\_token=${token}`);

const tokenInfo = await response.json();

if (!tokenInfo.error) {

const userInfoResponse = await fetch(`https://www.googleapis.com/oauth2/v2/userinfo?access\_token=${token}`);

payload = await userInfoResponse.json();

}

}

// 3. Último recurso: decodificación manual JWT

catch (accessTokenError) {

const base64Url = token.split('.')[1];

const base64 = base64Url.replace(/-/g, '+').replace(/\_/g, '/');

const decodedBase64 = Buffer.from(base64, 'base64').toString('binary');

payload = JSON.parse(decodeURIComponent(

Array.from(decodedBase64).map(c =>

'%' + ('00' + c.charCodeAt(0).toString(16)).slice(-2)

).join('')

));

}

}

// Control de acceso por roles

const adminEmails = process.env.ADMIN\_EMAILS?.split(',') || [];

const isAdmin = adminEmails.includes(payload.email);

// Crear/actualizar usuario con rol automático

let user = await User.findOne({ email: payload.email });

if (!user) {

user = await User.create({

googleId: payload.sub,

email: payload.email,

name: payload.name,

avatar: payload.picture,

role: isAdmin ? 'admin' : 'client',

balance: 10.0 // Se ajustará a Infinity para admins por middleware

});

}

// Generar JWT y crear sesión

const jwtToken = generateToken(user);

await Session.create({ userId: user.\_id, token: jwtToken });

res.json({

success: true,

token: jwtToken,

user: {

id: user.\_id,

email: user.email,

name: user.name,

role: user.role,

balance: user.role === 'admin' ? 'Infinity' : user.balance

}

});

} catch (error) {

console.error('Error en googleLogin:', error);

res.status(500).json({ success: false, message: 'Error al iniciar sesión' });

}

};

#### Configuración de Variables de Entorno:

# Autenticación JWT

JWT\_SECRET=clave\_secreta\_segura

JWT\_EXPIRE=24h

# Google OAuth2

GOOGLE\_CLIENT\_ID=example\_client\_id

GOOGLE\_CLIENT\_SECRET=example\_client\_secret

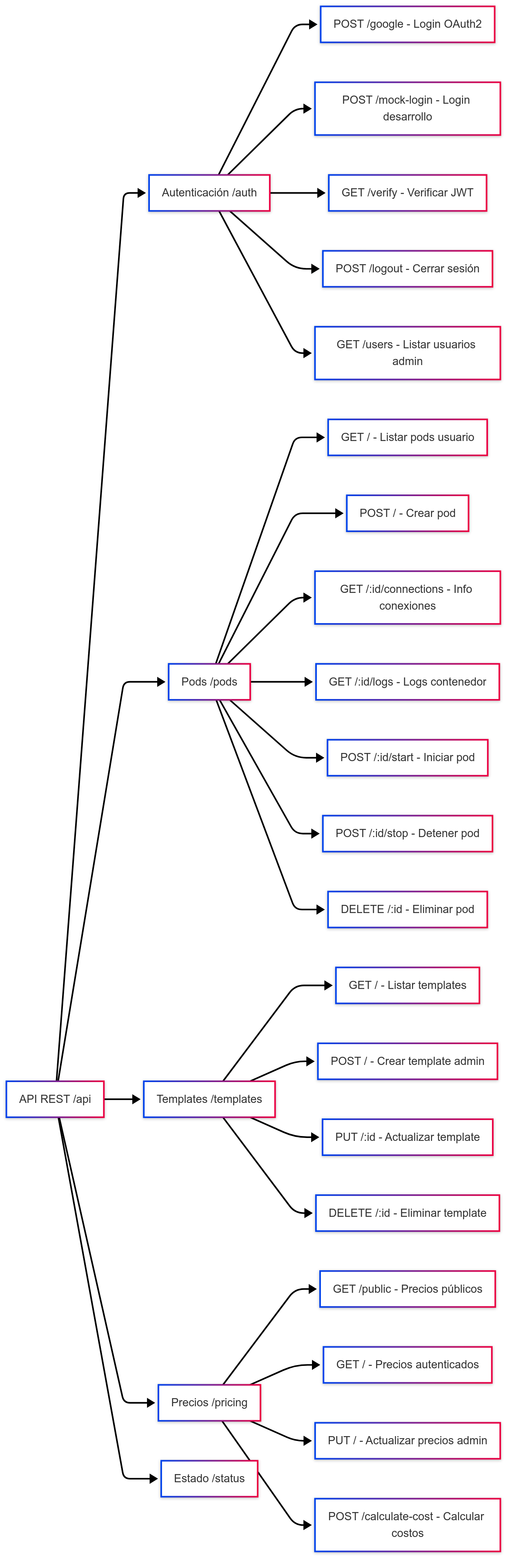
# Control de acceso

TRUST\_GOOGLE\_AUTH=true

ADMIN\_EMAILS=lolerodiez@gmail.com

## API RESTful Desarrollada (Esquema)

### Estructura de Endpoints Implementada



### Controlador de Pods Implementado

#### Características Avanzadas del pod.controller.js:

* **Validación Completa de Payload:**

async function validatePodPayload(payload, currentUser) {

const errors = [];

// Validaciones básicas

if (!payload.name) errors.push("Nombre es requerido");

if (!payload.gpu) errors.push("GPU es requerida");

// Validación de recursos

if (payload.containerDiskSize < 1 || payload.containerDiskSize > 100) {

errors.push("Disk debe estar entre 1 y 100 GB");

}

// Validación de asignación de usuario (solo admin)

if (payload.assignToUser && currentUser.role !== "admin") {

errors.push("Solo administradores pueden asignar pods");

}

return errors;

}

* **Integración WebSocket para Updates en Tiempo Real:**

// Notificar creación de pod

const io = getSocketIO(req);

if (io && io.notifyPodCreated) {

io.notifyPodCreated(podOwner.\_id.toString(), {

podId: pod.podId,

podName: pod.podName,

status: pod.status,

createdBy: req.user.email

});

}

* **Gestión Asíncrona de Kubernetes:**

function createKubernetesResourcesAsync(pod, podOwner, body) {

setImmediate(async () => {

try {

const kubernetesResult = await kubernetesService.createPodWithServices({

name: pod.podName,

userId: podOwner.\_id.toString(),

dockerImage: pod.dockerImage,

ports: body.ports,

gpu: body.gpu

});

pod.status = 'creating';

await pod.save();

// Notificar cambio de estado via WebSocket

const io = app.get('io');

if (io && io.sendPodUpdate) {

io.sendPodUpdate(pod.podId, {

status: 'creating',

message: 'Recursos de Kubernetes creados'

});

}

} catch (err) {

pod.status = 'error';

await pod.save();

}

});

}

### Sistema de Precios Dinámico

#### pricing.controller.js - Funcionalidades Implementadas:

* **Endpoint Público vs Privado:**

// Público (sin autenticación) - Para página /pricing

exports.getPublicPricing = async (req, res) => {

const pricing = await Pricing.getCurrentPricing();

// Sin logAction - no hay usuario

res.json({ success: true, data: pricing });

};

// Privado (con autenticación) - Para páginas internas

exports.getPricing = async (req, res) => {

const pricing = await Pricing.getCurrentPricing();

await logAction(req.user.\_id, 'GET\_PRICING'); // Log de auditoría

res.json({ success: true, data: pricing });

};

* **Cálculo de Costos en Tiempo Real:**

exports.calculateCost = async (req, res) => {

const { gpu, containerDiskSize, volumeDiskSize, hours = 1 } = req.body;

const pricing = await Pricing.getCurrentPricing();

const costs = pricing.calculateCost({

gpu, containerDiskSize, volumeDiskSize

});

const breakdown = {

gpu: { hourlyRate: costs.gpu, cost: costs.gpu \* hours },

containerDisk: { hourlyRate: costs.containerDisk, cost: costs.containerDisk \* hours },

volumeDisk: { hourlyRate: costs.volumeDisk, cost: costs.volumeDisk \* hours },

total: costs.total \* hours,

currency: 'EUR'

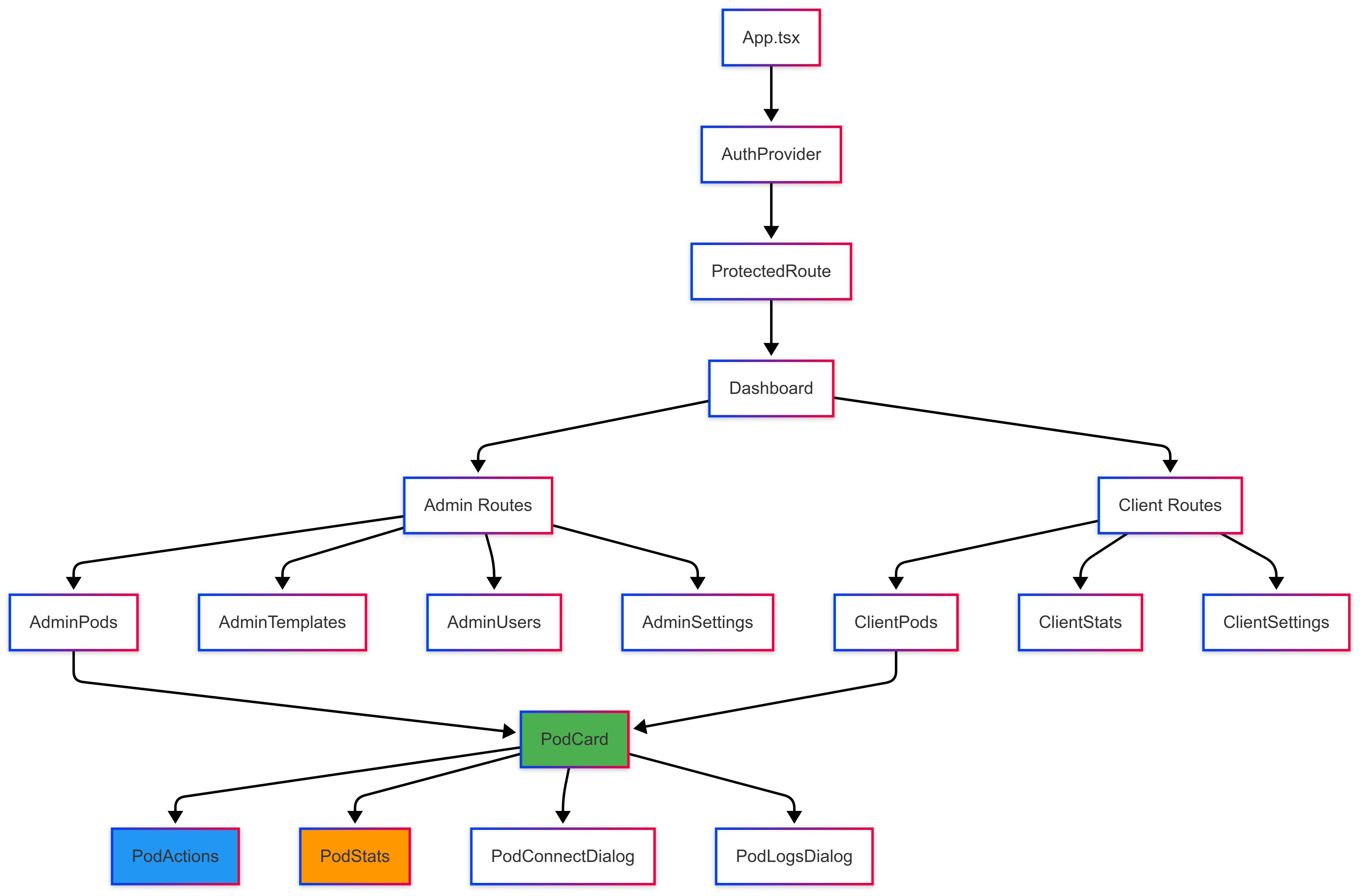
};

res.json({ success: true, data: breakdown });

};

## Frontend React Construido (Esquema)

### Arquitectura de Componentes Implementada



### Gestión de Estado con Hooks Personalizados

#### usePodUpdates.ts - Hook para WebSocket:

export const usePodUpdates = (podId?: string) => {

const [pods, setPods] = useState<Pod[]>([]);

const { socket } = useWebSocket();

useEffect(() => {

if (podId && socket) {

socket.emit('subscribe', podId);

const handlePodUpdate = (updatedPod: Pod) => {

setPods(prev => prev.map(pod =>

pod.podId === updatedPod.podId ? updatedPod : pod

));

};

socket.on('podUpdate', handlePodUpdate);

return () => socket.off('podUpdate', handlePodUpdate);

}

}, [podId, socket]);

return { pods, setPods };

};

#### useWebSocket.ts - Gestión de Conexión:

export const useWebSocket = () => {

const [connectionStatus, setConnectionStatus] = useState({

connected: false,

connecting: false,

error: null

});

useEffect(() => {

const token = localStorage.getItem('token');

if (token) {

websocketService.connect();

}

// Listeners para eventos de conexión

const handleConnect = () => setConnectionStatus(prev => ({

...prev, connected: true, connecting: false

}));

const handleDisconnect = () => setConnectionStatus(prev => ({

...prev, connected: false

}));

window.addEventListener('socketConnect', handleConnect);

window.addEventListener('socketDisconnect', handleDisconnect);

return () => {

window.removeEventListener('socketConnect', handleConnect);

window.removeEventListener('socketDisconnect', handleDisconnect);

};

}, []);

return { connectionStatus, socket: websocketService };

};

### Componentes de Pods Implementados

#### PodCard.tsx - Componente Principal:

export const PodCard: React.FC<PodCardProps> = ({

pod: initialPod,

onTogglePod,

onDeletePod,

viewLogs

}) => {

const [pod, setPod] = useState(() => sanitizePod(initialPod));

const { podData, connectionStatus } = usePodUpdates(pod.podId);

// Actualizar pod cuando llegan datos por WebSocket

useEffect(() => {

if (podData) {

const updatedPod = {

...pod,

status: podData.status,

stats: podData.stats,

httpServices: podData.httpServices || [],

tcpServices: podData.tcpServices || []

};

setPod(updatedPod);

}

}, [podData]);

return (

<Card>

<CardHeader>

<CardTitle className="flex items-center gap-2">

<Server className="h-5 w-5" />

{pod.podName}

{getStatusBadge(pod.status)}

{/\* Indicador de conexión WebSocket \*/}

{connectionStatus.connected ? (

<Wifi className="h-4 w-4 text-green-500" />

) : (

<WifiOff className="h-4 w-4 text-red-500" />

)}

</CardTitle>

</CardHeader>

<CardContent>

<div className="grid grid-cols-1 lg:grid-cols-3 gap-6">

<PodStats pod={pod} />

<div className="lg:col-span-2">

<PodActions

pod={pod}

onTogglePod={onTogglePod}

onDeletePod={onDeletePod}

viewLogs={viewLogs}

/>

</div>

</div>

</CardContent>

</Card>

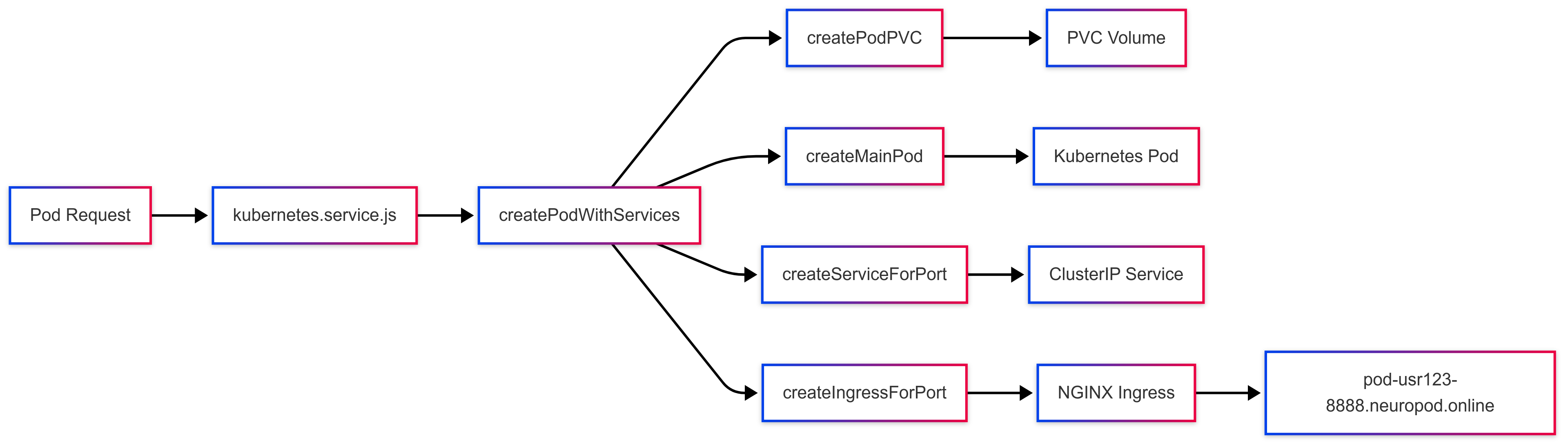
);

};

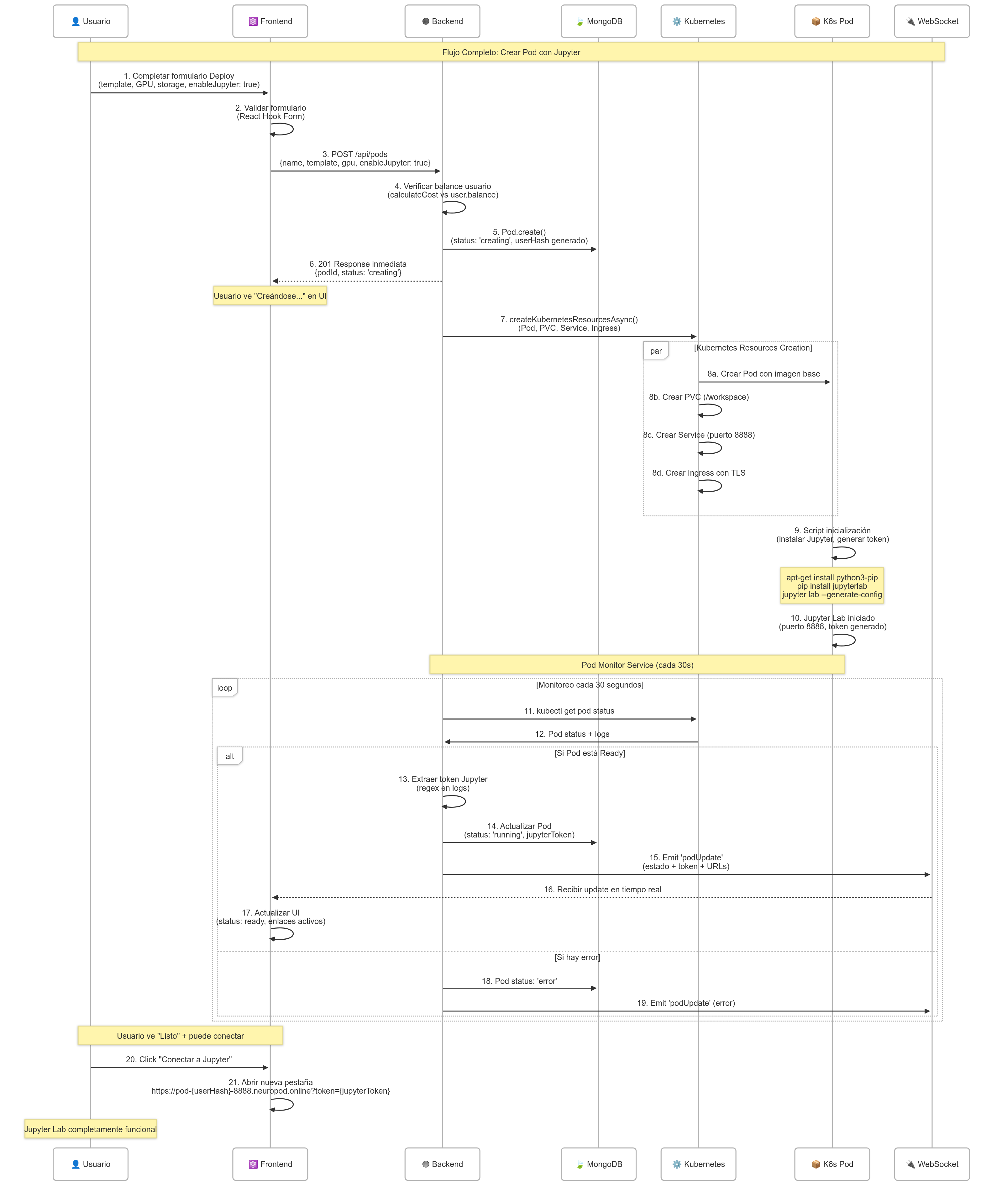
## Integración Kubernetes Real (Esquema)

### Servicio Kubernetes Completo

#### kubernetes.service.js - Arquitectura de Servicios:



#### Esquema de creación de pod con Jupyter



### Servicio Kubernetes Implementado (Código Real)

#### Creación de Pod Principal:

// Backend: services/kubernetes.service.js - Gestión completa de recursos

class KubernetesService {

constructor() {

this.kc = new k8s.KubeConfig();

try {

if (process.env.NODE\_ENV === 'production') {

this.kc.loadFromDefault();

this.k8sApi = this.kc.makeApiClient(k8s.CoreV1Api);

this.k8sNetworkingApi = this.kc.makeApiClient(k8s.NetworkingV1Api);

console.log('✅ Kubernetes client initialized successfully');

} else {

throw new Error('Modo simulación forzado en desarrollo');

}

} catch (error) {

console.warn('⚠️ Kubernetes not available, running in simulation mode');

this.k8sApi = null;

this.k8sNetworkingApi = null;

}

}

// Crear Pod completo con todos los recursos

async createPodWithServices(podConfig) {

const {

name: podName,

userId,

dockerImage,

ports,

containerDiskSize,

volumeDiskSize,

gpu,

enableJupyter

} = podConfig;

const userHash = generateUserHash(userId);

const portsArray = ports.split(',').map(p => parseInt(p.trim()));

try {

console.log(`🚀 Creating pod ${podName} for user ${userId}`);

// 1. Crear PVC específico para este pod

const pvcName = await this.createPodPVC(podName, userHash, volumeDiskSize);

// 2. Crear el Pod principal

await this.createMainPod(podName, userHash, dockerImage, portsArray,

containerDiskSize, volumeDiskSize, gpu,

enableJupyter, pvcName);

// 3. Crear Service e Ingress para cada puerto

const services = [];

for (const port of portsArray) {

const serviceName = await this.createServiceForPort(podName, userHash, userId, port);

const subdomain = generateSecureSubdomain(podName, userId, port);

const ingressName = await this.createIngressForPort(podName, userHash, port, subdomain);

services.push({

port,

serviceName,

ingressName,

subdomain,

url: `https://${subdomain}`

});

}

return {

podName: `${podName}-${userHash}`,

pvcName,

userHash,

services,

status: 'creating'

};

} catch (error) {

console.error(`❌ Error creating pod ${podName}:`, error);

throw error;

}

}

// Crear Pod principal con configuración completa

async createMainPod(podName, userHash, dockerImage, ports, containerDiskSize,

volumeDiskSize, gpu, enableJupyter, pvcName) {

if (!this.isKubernetesAvailable()) {

console.log(`🔧 [SIMULATION] Creating main pod ${podName}-${userHash}`);

return;

}

const sanitizedPodName = podName.toLowerCase().replace(/[^a-z0-9-]/g, '-');

const podFullName = `${sanitizedPodName}-${userHash}`;

// Configurar recursos y GPU

const resourceLimits = {

memory: process.env.POD\_MEMORY\_LIMIT || '11Gi',

cpu: process.env.POD\_CPU\_LIMIT || '2',

};

if (gpu && gpu.includes('rtx')) {

resourceLimits['nvidia.com/gpu'] = '1';

}

// Generar token seguro para Jupyter

const jupyterToken = crypto.randomBytes(24).toString('hex');

// Detectar tipo de imagen y configurar comandos

const isComfyUIImage = dockerImage.toLowerCase().includes('comfyui');

let command = ['/bin/bash', '-c'];

let args = [];

if (enableJupyter && ports.includes(8888)) {

if (isComfyUIImage) {

// Script especial para ComfyUI + Jupyter

args = [`

echo "🚀 Iniciando servicios ComfyUI y Jupyter Lab..."

# Instalar Jupyter si no está disponible

if ! command -v jupyter &> /dev/null; then

pip install jupyterlab || apt-get update && apt-get install -y python3-pip && pip3 install jupyterlab

fi

# Configurar Jupyter con token seguro

mkdir -p /root/.jupyter

echo "c.ServerApp.ip = '0.0.0.0'" > /root/.jupyter/jupyter\_lab\_config.py

echo "c.ServerApp.port = 8888" >> /root/.jupyter/jupyter\_lab\_config.py

echo "c.ServerApp.allow\_root = True" >> /root/.jupyter/jupyter\_lab\_config.py

echo "c.ServerApp.token = '${jupyterToken}'" >> /root/.jupyter/jupyter\_lab\_config.py

# Iniciar Jupyter Lab en background

nohup jupyter lab --config=/root/.jupyter/jupyter\_lab\_config.py > /tmp/jupyter.log 2>&1 &

# Iniciar ComfyUI si el puerto 8188 está configurado

if [[ "${ports.join(',')}" == \*"8188"\* ]]; then

echo "🎨 Iniciando ComfyUI en puerto 8188..."

if [ -f "/app/main.py" ]; then

nohup python3 /app/main.py --listen 0.0.0.0 --port 8188 > /tmp/comfyui.log 2>&1 &

fi

fi

echo "✅ Servicios iniciados - Pod listo"

echo "🔑 Jupyter Lab token: ${jupyterToken}"

# Mantener contenedor vivo

tail -f /dev/null

`];

}

}

const pod = {

apiVersion: 'v1',

kind: 'Pod',

metadata: {

name: podFullName,

labels: {

app: sanitizedPodName,

user: userHash,

'neuropod.online/resource': 'pod',

'neuropod.online/gpu': gpu || 'none',

'neuropod.online/jupyter': enableJupyter.toString()

}

},

spec: {

containers: [{

name: 'main',

image: dockerImage,

command,

args,

ports: ports.map(port => ({

containerPort: port,

name: `port-${port}`,

protocol: 'TCP'

})),

resources: {

limits: resourceLimits,

requests: {

memory: process.env.POD\_MEMORY\_REQUEST || '6Gi',

cpu: process.env.POD\_CPU\_REQUEST || '1'

}

},

volumeMounts: [{

name: 'workspace',

mountPath: '/workspace'

}],

env: [

{ name: 'NEUROPOD\_USER', value: userHash },

{ name: 'JUPYTER\_TOKEN', value: jupyterToken },

{ name: 'NEUROPOD\_GPU', value: gpu || 'none' }

]

}],

volumes: [{

name: 'workspace',

persistentVolumeClaim: { claimName: pvcName }

}],

restartPolicy: 'Never',

terminationGracePeriodSeconds: 10

}

};

await this.k8sApi.createNamespacedPod({ namespace: 'default', body: pod });

console.log(`✅ Pod ${podFullName} created successfully`);

}

// Capturar token de Jupyter de los logs

async captureJupyterToken(podName, userHash) {

try {

const logs = await this.getPodLogs(podName, userHash, 1000);

const tokenPatterns = [

/Jupyter Lab token: ([a-f0-9]{48})/i,

/token=([a-f0-9]{48})/i,

/\?token=([a-f0-9]{48})/i

];

for (const pattern of tokenPatterns) {

const match = logs.match(pattern);

if (match) {

const token = match[1];

if (token && token.length === 48) {

console.log(`✅ Jupyter token captured: ${token.substring(0, 8)}...`);

return token;

}

}

}

return null;

} catch (error) {

console.error('❌ Error capturing Jupyter token:', error);

return null;

}

}

}

module.exports = new KubernetesService();

### ****Pod Controller Implementado (Código Real)****

#### Gestión completa del pod:

// Backend: controllers/pod.controller.js - Gestión completa de ciclo de vida

exports.createPod = async (req, res) => {

try {

// Validaciones exhaustivas

const errors = await validatePodPayload(req.body, req.user);

if (errors.length > 0) {

return res.status(400).json({ success: false, errors });

}

// Determinar propietario del pod (admin puede asignar a otros)

const podOwner = await determinePodOwner(req.body, req.user);

await validateUserBalance(podOwner, req.body);

// Procesar configuración según tipo de deployment

const { finalDockerImage, httpServices, tcpServices } =

await processPodConfiguration(req.body);

// Crear registro en base de datos

const pod = await createPodRecord(req.body, podOwner, req.user,

finalDockerImage, httpServices, tcpServices);

// Notificar creación por WebSocket

const io = getSocketIO(req);

if (io?.notifyPodCreated) {

io.notifyPodCreated(podOwner.\_id.toString(), {

podId: pod.podId,

podName: pod.podName,

status: pod.status,

createdBy: req.user.email

});

}

// Crear recursos en Kubernetes (asíncrono)

createKubernetesResourcesAsync(pod, podOwner, req.body);

// Descontar saldo si es necesario

await deductBalanceIfClient(podOwner, req.body);

await logAction(req.user.\_id, 'CREATE\_POD', {

podId: pod.podId,

targetUser: podOwner.\_id !== req.user.\_id ? podOwner.\_id : undefined

});

res.status(201).json({

success: true,

data: {

podId: pod.podId,

podName: pod.podName,

status: pod.status,

message: 'Pod creándose. Por favor espere unos minutos.'

}

});

} catch (error) {

handleControllerError(res, error, 'Error interno al crear el pod');

}

};

// Función asíncrona para crear recursos Kubernetes

function createKubernetesResourcesAsync(pod, podOwner, body) {

setImmediate(async () => {

try {

const kubernetesResult = await kubernetesService.createPodWithServices({

name: pod.podName,

userId: podOwner.\_id.toString(),

dockerImage: pod.dockerImage,

ports: body.ports,

containerDiskSize: body.containerDiskSize,

volumeDiskSize: body.volumeDiskSize,

gpu: body.gpu,

enableJupyter: body.enableJupyter

});

// Actualizar URLs reales de servicios

if (kubernetesResult?.services) {

kubernetesResult.services.forEach(k8sService => {

const httpService = pod.httpServices.find(s => s.port === k8sService.port);

if (httpService) {

httpService.url = k8sService.url;

httpService.kubernetesServiceName = k8sService.serviceName;

httpService.kubernetesIngressName = k8sService.ingressName;

}

});

pod.kubernetesResources.podName = kubernetesResult.podName;

pod.kubernetesResources.pvcName = kubernetesResult.pvcName;

pod.userHash = kubernetesResult.userHash;

}

pod.status = 'creating';

await pod.save();

// Notificar cambio de estado por WebSocket

const app = require('../app');

const io = app.get('io');

if (io?.sendPodUpdate) {

io.sendPodUpdate(pod.podId, {

status: 'creating',

httpServices: pod.httpServices,

tcpServices: pod.tcpServices,

message: 'Recursos de Kubernetes creados, pod inicializándose...'

});

}

// Capturar token de Jupyter si es necesario

if (body.enableJupyter) {

scheduleJupyterTokenCapture(pod);

}

} catch (err) {

console.error('Error creando recursos Kubernetes:', err);

pod.status = 'error';

await pod.save();

const app = require('../app');

const io = app.get('io');

if (io?.sendPodUpdate) {

io.sendPodUpdate(pod.podId, {

status: 'error',

message: 'Error al crear recursos de Kubernetes'

});

}

}

});

}

// Programar captura de token Jupyter

function scheduleJupyterTokenCapture(pod) {

setTimeout(async () => {

try {

const token = await kubernetesService.captureJupyterToken(pod.podName, pod.userHash);

if (token) {

const jupyterService = pod.httpServices.find(s => s.port === 8888);

if (jupyterService) {

jupyterService.jupyterToken = token;

await pod.save();

}

}

} catch (err) {

console.error('Error capturando token Jupyter:', err);

}

}, 15000); // Esperar 15 segundos

}

#### Obtención de Logs Reales:

async getPodLogs(podName, userHash, lines = 500) {

const sanitizedPodName = podName.toLowerCase().replace(/[^a-z0-9-]/g, '-');

const podFullName = `${sanitizedPodName}-${userHash}`;

try {

const response = await this.k8sApi.readNamespacedPodLog({

name: podFullName,

namespace: 'default',

container: 'main',

follow: false,

tailLines: lines

});

return response.body || response.data || response;

} catch (error) {

if (error.statusCode === 404) {

return `Pod '${podFullName}' no encontrado.`;

}

return `Error al obtener logs: ${error.message}`;

}

}

### Generación de Subdominios Dinámicos

#### podHelpers.js - Utilidades de Generación:

// Generar hash único de usuario (8 caracteres)

function generateUserHash(userId) {

return crypto.createHash('sha256')

.update(userId.toString())

.digest('hex')

.substring(0, 8);

}

// Generar subdominio seguro único

function generateSecureSubdomain(podName, userId, port) {

const safePodName = podName.toLowerCase()

.replace(/[^a-z0-9-]/g, '-')

.substring(0, 10);

const userHash = generateUserHash(userId);

const randomSuffix = crypto.randomBytes(4).toString('hex');

return `${safePodName}-${userHash}-${port}-${randomSuffix}.neuropod.online`;

}

## Base de Datos MongoDB Aplicada

### Modelo de Pod Avanzado

#### Pod.model.js - Características Implementadas:

* **Servicios HTTP/TCP Estructurados:**

const HttpServiceSchema = new mongoose.Schema({

port: { type: Number, required: true, min: 1, max: 65535 },

serviceName: { type: String, required: true },

url: { type: String, required: true },

isCustom: { type: Boolean, default: false },

status: {

type: String,

enum: ['creating', 'ready', 'error', 'stopped'],

default: 'creating'

},

jupyterToken: { type: String, sparse: true },

kubernetesServiceName: String,

kubernetesIngressName: String

}, { \_id: false });

* **Estadísticas y Métricas:**

const PodStatsSchema = new mongoose.Schema({

cpuUsage: { type: Number, default: 0, min: 0, max: 100 },

memoryUsage: { type: Number, default: 0, min: 0, max: 100 },

gpuUsage: { type: Number, default: 0, min: 0, max: 100 },

uptime: { type: Number, default: 0, min: 0 },

lastUpdated: { type: Date, default: Date.now }

}, { \_id: false });

* **Middleware Pre-Save Automático:**

PodSchema.pre('save', function(next) {

// Generar userHash si no existe

if (!this.userHash && this.userId) {

this.userHash = generateUserHash(this.userId.toString());

}

// Generar nombres de recursos de Kubernetes

if (!this.kubernetesResources.podName && this.podName && this.userHash) {

const sanitizedPodName = this.podName.toLowerCase().replace(/[^a-z0-9-]/g, '-');

this.kubernetesResources.podName = `${sanitizedPodName}-${this.userHash}`;

}

next();

});

### Sistema de Precios Dinámico

#### Pricing.model.js - Configuración Avanzada:

const PricingSchema = new mongoose.Schema({

gpus: {

'rtx-4050': {

price: { type: Number, default: 2.50 },

available: { type: Boolean, default: true },

specs: {

memory: { type: String, default: "6GB GDDR6" },

cores: { type: Number, default: 2560 },

performance: { type: String, default: "Entry Level" }

}

},

'rtx-4080': {

price: { type: Number, default: 4.99 },

available: { type: Boolean, default: false },

specs: {

memory: { type: String, default: "16GB GDDR6X" },

cores: { type: Number, default: 9728 },

performance: { type: String, default: "Ultra Performance" }

}

}

},

storage: {

containerDisk: {

price: { type: Number, default: 0.05 },

unit: { type: String, default: "€/GB/hora" }

},

volumeDisk: {

price: { type: Number, default: 0.10 },

unit: { type: String, default: "€/GB/hora" }

}

},

limits: {

containerDiskMax: { type: Number, default: 100 },

volumeDiskMax: { type: Number, default: 150 },

portsMax: { type: Number, default: 10 }

}

});

// Método para calcular costos dinámicos

PricingSchema.methods.calculateCost = function(config) {

const gpuCost = this.gpus[config.gpu]?.price || 2.50;

const containerCost = config.containerDiskSize \* this.storage.containerDisk.price;

const volumeCost = config.volumeDiskSize \* this.storage.volumeDisk.price;

return {

gpu: gpuCost,

containerDisk: containerCost,

volumeDisk: volumeCost,

total: gpuCost + containerCost + volumeCost

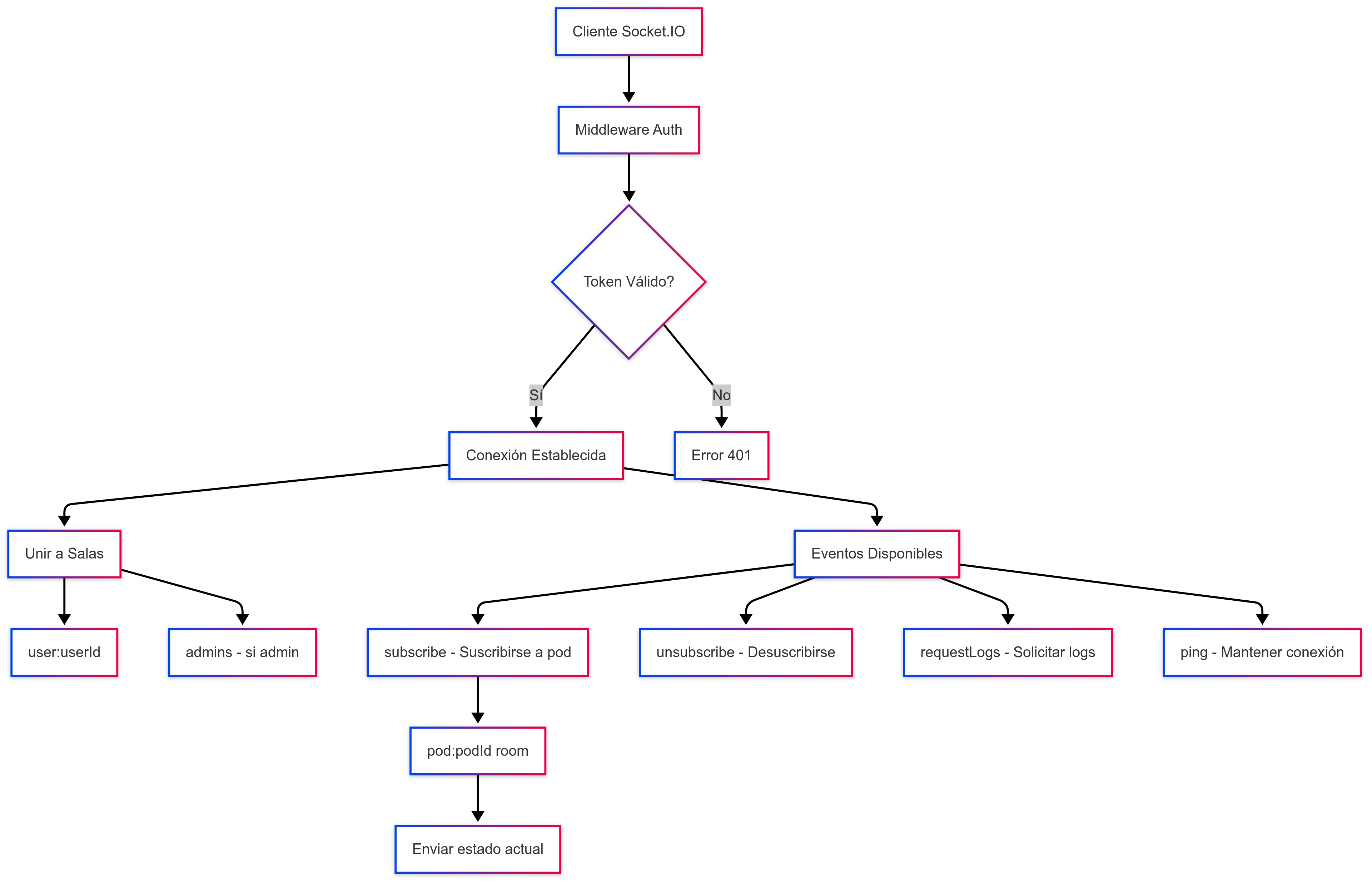
};

};

## Comunicación en Tiempo Real (WebSockets)

### Servidor Socket.IO Implementado

#### socket.js - Configuración Completa:



#### Middleware de Autenticación WebSocket:

io.use(async (socket, next) => {

try {

const token = socket.handshake.auth.token || socket.handshake.query.token;

if (!token) {

return next(new Error('No token provided'));

}

const decoded = jwt.verify(token, process.env.JWT\_SECRET);

const session = await Session.findOne({ token });

const user = await User.findById(decoded.id);

if (!session || !user) {

return next(new Error('Invalid session'));

}

// Añadir usuario a la conexión socket

socket.user = user;

socket.userId = user.\_id.toString();

socket.userRole = user.role;

await logAction(user.\_id, 'SOCKET\_CONNECT', {

socketId: socket.id,

userAgent: socket.request.headers['user-agent']

});

next();

} catch (error) {

next(new Error('Authentication failed'));

}

});

### Eventos Específicos de Pods

#### Suscripción a Pods:

socket.on('subscribe', async (data) => {

const { podId } = data;

const pod = await Pod.findOne({ podId });

if (!pod) {

return socket.emit('error', {

type: 'POD\_NOT\_FOUND',

message: 'Pod not found'

});

}

// Verificar acceso

const hasAccess = socket.userRole === 'admin' ||

pod.userId.toString() === socket.userId;

if (!hasAccess) {

return socket.emit('error', {

type: 'ACCESS\_DENIED',

message: 'No access to this pod'

});

}

// Unir a la sala del pod

const roomName = `pod:${podId}`;

socket.join(roomName);

// Enviar estado actual

socket.emit('podUpdate', {

type: 'podUpdate',

podId: pod.podId,

status: pod.status,

stats: pod.stats,

timestamp: new Date().toISOString()

});

});

#### Funciones de Notificación:

const sendPodUpdate = async (podId, updateData) => {

const roomName = `pod:${podId}`;

const clientsInRoom = await io.in(roomName).fetchSockets();

if (clientsInRoom.length > 0) {

io.to(roomName).emit('podUpdate', {

type: 'podUpdate',

podId,

...updateData,

timestamp: new Date().toISOString()

});

}

};

const notifyPodCreated = (userId, podData) => {

io.to(`user:${userId}`).emit('podCreated', {

type: 'podCreated',

...podData,

timestamp: new Date().toISOString()

});

};

const sendLowBalanceAlert = (userId, balanceData) => {

io.to(`user:${userId}`).emit('lowBalanceAlert', {

type: 'lowBalanceAlert',

...balanceData,

timestamp: new Date().toISOString()

});

};

### Cliente WebSocket React

#### websocket.service.ts - Implementación Frontend:

class WebSocketService {

private socket: Socket | null = null;

private subscribedPods = new Set<string>();

connect() {

const token = localStorage.getItem('token');

// Detección automática de entorno

let serverUrl: string;

const isHTTPS = window.location.protocol === 'https:';

const isProductionDomain = window.location.hostname.includes('neuropod.online');

if (isProductionDomain && isHTTPS) {

serverUrl = 'https://api.neuropod.online';

} else {

serverUrl = `http://${window.location.hostname}:3000`;

}

this.socket = io(serverUrl, {

auth: { token },

transports: ['websocket', 'polling'],

timeout: 20000

});

// Eventos de conexión

this.socket.on('connect', () => {

console.log('🔌 WebSocket conectado');

// Reenviar suscripciones existentes

this.subscribedPods.forEach(podId => {

this.subscribeToPod(podId);

});

});

// Eventos específicos de pods

this.socket.on('podUpdate', (data) => {

window.dispatchEvent(new CustomEvent('podUpdate', { detail: data }));

});

this.socket.on('podCreated', (data) => {

window.dispatchEvent(new CustomEvent('podCreated', { detail: data }));

});

}

subscribeToPod(podId: string) {

this.subscribedPods.add(podId);

if (this.socket?.connected) {

this.socket.emit('subscribe', { podId });

}

}

}

export default new WebSocketService();

## Scripts de Automatización Desarrollados

### Arrancar.ps1 - Inicio Orquestado

#### Secuencia de Inicio Automatizada:

# Verificación de permisos administrador

if (-not ([Security.Principal.WindowsPrincipal] [Security.Principal.WindowsIdentity]::GetCurrent()).IsInRole([Security.Principal.WindowsBuiltInRole] "Administrator")) {

Write-Host "Ejecutar como ADMINISTRADOR" -ForegroundColor Red

exit

}

# 1. Iniciar Docker Desktop

Start-Process "C:\Program Files\Docker\Docker\Docker Desktop.exe"

Start-Sleep -Seconds 10

# 2. Iniciar Cloudflare Tunnel

wt -w 0 nt --title "Cloudflare Tunnel" powershell -NoExit -Command "cloudflared.exe tunnel run neuropod-tunnel"

# 3. Iniciar Minikube con configuración completa

wt -w 0 nt --title "Minikube" powershell -NoExit -Command "minikube start --driver=docker --container-runtime=docker --gpus=all --memory=12000mb --cpus=8 --addons=ingress,storage-provisioner,default-storageclass"

# 4. Iniciar MongoDB

wt -w 0 nt --title "MongoDB" powershell -NoExit -Command "& 'C:\Program Files\MongoDB\Server\8.0\bin\mongod.exe' --dbpath='C:\data\db'"

# 5. Iniciar Minikube Tunnel

wt -w 0 nt --title "Minikube Tunnel" powershell -NoExit -Command "minikube tunnel"

# 6. Iniciar Backend NeuroPod

$BackendPath = Join-Path $CURRENT\_DIR "NeuroPod-Backend"

wt -w 0 nt --title "NeuroPod Backend" cmd /k "cd /d $BackendPath && npm start"

# 7. Iniciar Frontend NeuroPod

$FrontendPath = Join-Path $CURRENT\_DIR "NeuroPod-Frontend"

wt -w 0 nt --title "NeuroPod Frontend" cmd /k "cd /d $FrontendPath && npm run dev"

### Detener.ps1 - Cierre Coordinado

#### Función de Cierre Seguro:

function Stop-ProcessSafely {

param([string]$ProcessName, [string]$DisplayName)

try {

$processes = Get-Process -Name $ProcessName -ErrorAction SilentlyContinue

if ($processes) {

$processes | Stop-Process -Force

Write-Host "✅ $DisplayName detenido correctamente." -ForegroundColor Green

} else {

Write-Host "ℹ️ $DisplayName no estaba en ejecución." -ForegroundColor Gray

}

} catch {

Write-Host "⚠️ Error al detener $DisplayName" -ForegroundColor Yellow

}

}

# Secuencia de cierre

Stop-ProcessSafely -ProcessName "node" -DisplayName "Backend/Frontend NeuroPod"

& minikube stop

Stop-ProcessSafely -ProcessName "cloudflared" -DisplayName "Cloudflare Tunnel"

Stop-ProcessSafely -ProcessName "mongod" -DisplayName "MongoDB"

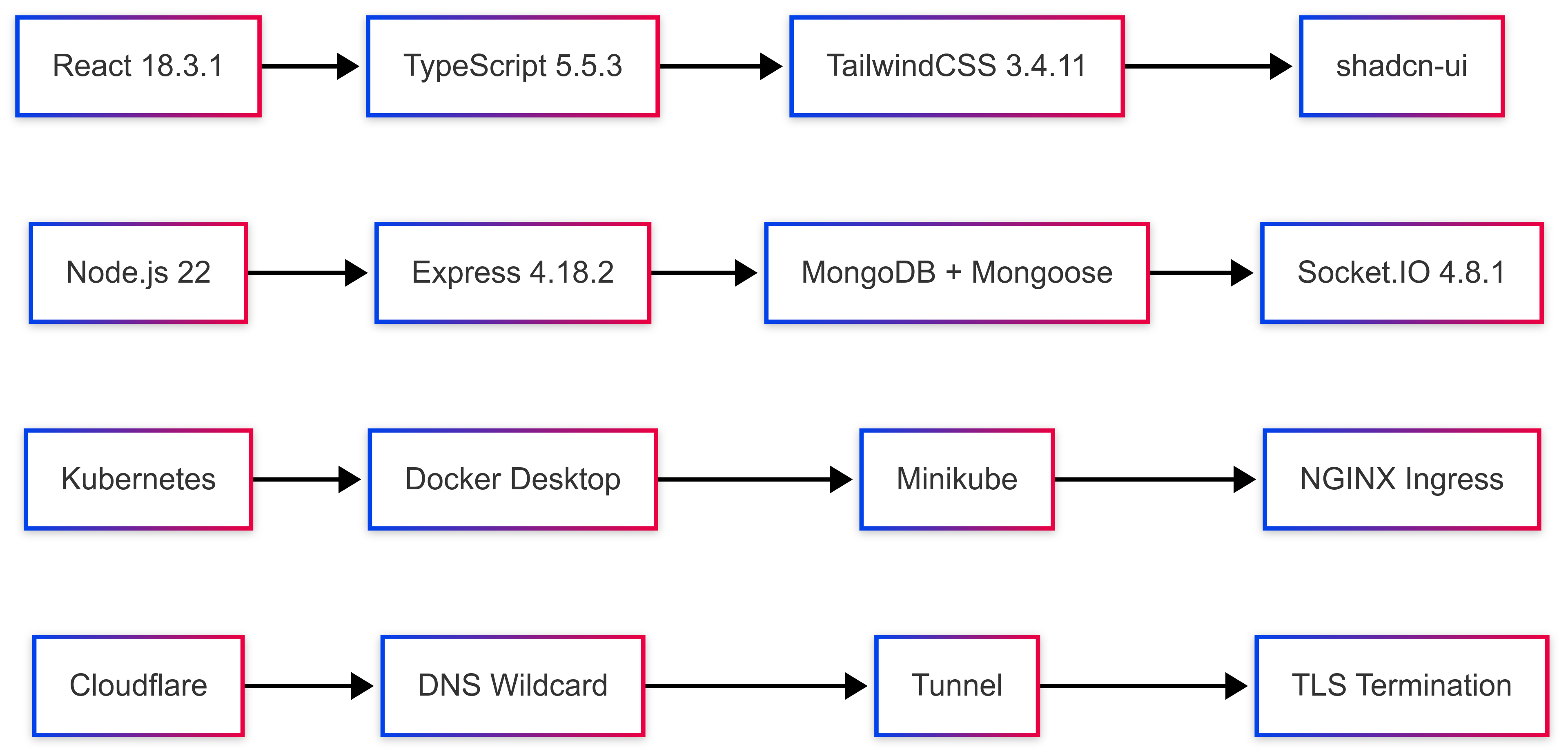
& wsl --shutdown # Detener WSL y vmmem

## Métricas y Análisis del Desarrollo (Esquema)

### Funcionalidades Core Implementadas

| **Funcionalidad** | **Estado** | **Complejidad** | **Testing** |
| --- | --- | --- | --- |
| **Autenticación OAuth2 + JWT** | ✅ Completo | Alta | Manual |
| **CRUD Pods con K8s** | ✅ Completo | Muy Alta | Manual |
| **WebSockets Tiempo Real** | ✅ Completo | Alta | Manual |
| **Sistema Precios Dinámico** | ✅ Completo | Media | Manual |
| **Templates Docker** | ✅ Completo | Media | Manual |
| **Modo Simulación** | ✅ Completo | Media | Manual |
| **Scripts Automatización** | ✅ Completo | Baja | Manual |

### Tecnologías Integradas



## Análisis de Decisiones Técnicas

### Decisiones de Arquitectura Justificadas

#### Mono-repo vs Multi-repo:

* **Decisión**: Mono-repo con carpetas separadas
* **Justificación**: Simplicidad de desarrollo, versionado conjunto
* **Trade-off**: Menor flexibilidad vs mayor cohesión

#### Context API vs Redux:

* **Decisión**: React Context API para estado global
* **Justificación**: Menor complejidad, suficiente para el scope del proyecto
* **Trade-off**: Menos herramientas vs simplicidad

#### MongoDB vs PostgreSQL:

* **Decisión**: MongoDB con Mongoose
* **Justificación**: Flexibilidad de esquemas, JSONs nativos
* **Trade-off**: Menos consistencia vs mayor agilidad

#### Minikube vs Kubernetes Cloud:

* **Decisión**: Minikube local para desarrollo
* **Justificación**: Control total, costo cero, aprendizaje
* **Trade-off**: Escalabilidad limitada vs simplicidad

# 6. Conclusiones

## Resumen Ejecutivo

El desarrollo del proyecto NeuroPod ha representado una experiencia integral de aplicación práctica de competencias del ciclo ASIR, combinadas con el aprendizaje de tecnologías modernas de desarrollo y orquestación de contenedores. Este proyecto ha permitido integrar conocimientos de administración de sistemas, redes, automatización y desarrollo, culminando en una plataforma funcional de gestión de contenedores Docker con capacidades avanzadas.

## Competencias ASIR Aplicadas en NeuroPod

### Administración de Sistemas

El proyecto NeuroPod ha permitido aplicar competencias fundamentales de administración de sistemas del ciclo ASIR, contextualizadas en el entorno moderno de contenedores y orquestación. Según las tendencias actuales en administración de sistemas, las competencias en containerización y Kubernetes son consideradas esenciales, incluyendo la capacidad de configurar y gestionar clusters, implementar networking y storage, y automatizar procesos.

#### Configuración y Gestión de Infraestructura:

* **Minikube**: Configuración completa de cluster Kubernetes local con GPU support
* **Docker Desktop**: Gestión de contenedores con integración WSL2
* **NGINX Ingress Controller**: Configuración avanzada para enrutamiento de subdominios dinámicos
* **MongoDB**: Administración de base de datos local con configuración de persistencia

#### Networking y Conectividad:

* **Subdominios dinámicos**: Implementación de wildcard DNS (\*.neuropod.online)
* **Certificados TLS**: Generación y gestión de certificados autofirmados para HTTPS
* **Cloudflare Tunnel**: Configuración de conexión segura sin IP pública
* **Firewall Windows**: Configuración de reglas para puertos específicos (3000, 5173, 443)

#### Automatización con PowerShell:

* **Scripts de inicio orquestado**: Arrancar.ps1 para secuencia automatizada de servicios
* **Scripts de cierre coordinado**: Detener.ps1 con gestión segura de procesos
* **Verificaciones automáticas**: Control de permisos de administrador y estado de servicios

### Redes y Comunicaciones

#### Arquitectura de Red Implementada:

* **Dominio principal**: neuropod.online con gestión DNS en Cloudflare
* **Subdominios funcionales**: app.neuropod.online, api.neuropod.online
* **Routing dinámico**: Generación automática de subdominios únicos por pod de usuario
* **Load balancing**: NGINX Ingress con distribución de tráfico HTTP/HTTPS

#### Protocolos y Servicios:

* **HTTP/HTTPS**: Implementación completa con terminación TLS
* **WebSockets**: Comunicación bidireccional en tiempo real
* **DNS**: Configuración wildcard para subdominios dinámicos
* **TCP/UDP**: Gestión de puertos personalizados para servicios de contenedores

### Scripting y Automatización

#### PowerShell Avanzado:

* **Gestión de procesos**: Control automático de servicios múltiples
* **Windows Terminal**: Orquestación de múltiples sesiones simultaneas
* **Verificaciones de estado**: Validación automática de servicios antes del inicio
* **Manejo de errores**: Recuperación graceful ante fallos de servicios

## Conocimientos Nuevos Adquiridos

### Orquestación de Contenedores con Kubernetes

#### Gestión Avanzada de Kubernetes:

* **Minikube**: Configuración de cluster local con soporte GPU
* **Manifiestos dinámicos**: Generación automática de Pods, Services, Ingress y PVC
* **Persistent Volumes**: Gestión de almacenamiento persistente con StorageClass
* **NGINX Ingress Controller**: Enrutamiento avanzado con subdominios dinámicos
* **Kubernetes Client API**: Integración programática con @kubernetes/client-node

#### Arquitectura de Servicios:

* **Pod lifecycle management**: Creación, inicio, parada y eliminación automatizada
* **Service discovery**: Configuración automática de servicios ClusterIP
* **Ingress routing**: Generación de subdominios únicos por pod de usuario
* **Resource management**: Configuración de límites de CPU, memoria y GPU

### Comunicación en Tiempo Real

#### WebSockets con Socket.IO:

* **Autenticación en WebSocket**: Middleware JWT para conexiones seguras
* **Rooms y subscripciones**: Gestión de salas por pod y usuario
* **Event-driven architecture**: Eventos específicos para actualizaciones de pods
* **Reconexión automática**: Handling de desconexiones y resubscripciones

### Tecnologías de Exposición Externa

#### Cloudflare Tunnel:

* **Configuración avanzada**: Routing de múltiples servicios locales
* **Wildcard DNS**: Gestión de subdominios dinámicos (\*.neuropod.online)
* **SSL termination**: Manejo automático de certificados en edge
* **WebSocket support**: Configuración específica para comunicación bidireccional

## Problemas Reales Enfrentados y Solucionados

### Problema: JSON.stringify(Infinity) → null

**Contexto del Problema:** Los usuarios administradores debían tener saldo infinito, pero JSON.stringify(Infinity) devuelve null, causando errores en el frontend.

#### Síntomas Observados:

* Administradores veían balance: null en localStorage
* Frontend mostraba "0.00 €" en lugar de símbolo infinito
* Errores de tipo al intentar operaciones matemáticas

#### Solución Implementada:

// Backend: Enviar como string para JSON

const userBalance = user.role === 'admin'

? 'Infinity' // String para JSON.stringify

: user.balance; // Número para clientes

// Frontend: Manejar ambos formatos

const formatBalance = (balance) => {

if (balance === 'Infinity' || balance === Infinity) {

return '∞ €';

}

return `${Number(balance || 0).toFixed(2)} €`;

};

#### Endpoint de Auto reparación:

// POST /api/auth/admin/fix-balances

const adminsToFix = await User.find({

role: 'admin',

$or: [

{ balance: { $ne: Number.POSITIVE\_INFINITY } },

{ balance: null }

]

});

### Problema: Certificados TLS para Subdominios Wildcard

**Contexto del Problema:** Cada pod de usuario necesita su propio subdominio (pod-usr123-8888.neuropod.online), requiriendo certificados TLS wildcard.

#### Síntomas Observados:

* Warnings de certificado en navegadores
* Fallos de conexión HTTPS para pods específicos
* NGINX Ingress rechazando conexiones TLS

#### Solución Implementada:

# Certificado autofirmado con SAN wildcard

[v3\_req]

subjectAltName = @alt\_names

[alt\_names]

DNS.1 = \*.neuropod.online

DNS.2 = neuropod.online

# Secret de Kubernetes con certificado wildcard

apiVersion: v1

kind: Secret

metadata:

name: neuropod-tls

type: kubernetes.io/tls

data:

tls.crt: LS0tLS1CRUdJTi... # Base64 encoded

tls.key: LS0tLS1CRUdJTi... # Base64 encoded

### Problema: WebSocket + NGINX Ingress Integration

**Contexto del Problema:** Los WebSockets no funcionaban correctamente a través de NGINX Ingress Controller, causando desconexiones frecuentes.

#### Síntomas Observados:

* Connections dropping en frontend
* Fallback a polling en lugar de WebSocket
* Eventos en tiempo real no llegaban

#### Solución Implementada:

# Configuración NGINX para WebSockets

data:

proxy-http-version: "1.1"

proxy-set-headers: "ingress-nginx/custom-headers"

use-forwarded-headers: "true"

# Headers específicos para upgrade de conexión

nginx.ingress.kubernetes.io/proxy-http-version: "1.1"

nginx.ingress.kubernetes.io/upstream-hash-by: "$remote\_addr"

// Cloudflare Tunnel config para WebSockets

- hostname: api.neuropod.online

service: http://localhost:3000

originRequest:

upgradeRequest: true # Soporte WebSocket

disableChunkedEncoding: true

http2Origin: false

### Problema: Gestión de Procesos en PowerShell

**Contexto del Problema:** Múltiples servicios (Docker, Minikube, MongoDB, Node.js) necesitan iniciarse en orden específico y con manejo de errores.

#### Síntomas Observados:

* Servicios fallando por dependencias no iniciadas
* Procesos zombie quedando en memoria
* Falta de feedback visual durante el inicio

#### Solución Implementada:

# Función de inicio seguro con verificaciones

function Start-ServiceSafely {

param([string]$ServiceName, [string]$Command, [int]$WaitSeconds)

Write-Host "Iniciando $ServiceName..." -ForegroundColor Yellow

Start-Process $Command

Start-Sleep -Seconds $WaitSeconds

# Verificación de estado

if (Get-Process -Name $ServiceName -ErrorAction SilentlyContinue) {

Write-Host "✅ $ServiceName iniciado correctamente" -ForegroundColor Green

} else {

Write-Host "❌ Error al iniciar $ServiceName" -ForegroundColor Red

}

}

# Secuencia con Windows Terminal

wt -w 0 nt --title "Service Name" powershell -NoExit -Command "command\_here"

## Conclusión Final

El desarrollo del proyecto NeuroPod ha representado una síntesis exitosa entre las competencias fundamentales del ciclo ASIR y las tecnologías emergentes del desarrollo moderno. La integración de administración de sistemas, redes, automatización y desarrollo full-stack ha resultado en una plataforma funcional que demuestra la aplicación práctica de conocimientos teóricos en un contexto real.

Las competencias desarrolladas en este proyecto están alineadas con las habilidades más demandadas en el mercado tecnológico actual, incluyendo desarrollo full-stack, orquestación de contenedores, y automatización de infraestructura.

La experiencia ha validado la relevancia del enfoque ASIR para formar profesionales capaces de adaptarse a entornos tecnológicos dinámicos, proporcionando una base sólida para el crecimiento profesional continuo en el sector de tecnologías de la información.

Las competencias adquiridas - desde la configuración de Kubernetes hasta el desarrollo de WebSockets en tiempo real - establecen una foundation técnica robusta para enfrentar los desafíos del panorama tecnológico actual, donde la convergencia entre administración de sistemas y desarrollo de software define las oportunidades profesionales más prometedoras.

# 7. Bibliografía

## Metodología de Referencias

Las referencias incluidas en esta bibliografía corresponden exclusivamente a la documentación oficial, recursos técnicos y herramientas específicamente consultadas y aplicadas durante el desarrollo real del proyecto NeuroPod. Todas las versiones mencionadas coinciden con las implementadas en el sistema, garantizando la precisión y aplicabilidad de cada referencia.

## Documentación Oficial de Tecnologías Implementadas

### Frontend Development

[1] **React Development Team**. "React Documentation - Version 18.3.1," *Meta Open Source*, 2024. [Online]. Available: <https://react.dev/>.

[2] **Microsoft Corporation**. "TypeScript Documentation - Version 5.5.3," *Microsoft TypeScript*, 2024. [Online]. Available: <https://www.typescriptlang.org/docs/>.

[3] **Evan You and Vite Team**. "Vite Documentation - Version 5.4.1," *Vite*, 2024. [Online]. Available: <https://vitejs.dev/guide/>.

[4] **TailwindCSS Team**. "Tailwind CSS Documentation - Version 3.4.11," *Tailwind Labs*, 2024. [Online]. Available: <https://tailwindcss.com/docs>.

[5] **shadcn**. "shadcn/ui Documentation - Component Library," *GitHub*, 2024. [Online]. Available: <https://ui.shadcn.com/docs>.

[6] **Matt Zabriskie**. "Axios Documentation - Version 1.9.0," *GitHub*, 2024. [Online]. Available: <https://axios-http.com/docs/intro>.

### Backend Development

[7] **Node.js Foundation**. "Node.js Documentation - Version 22," *Node.js*, 2024. [Online]. Available: <https://nodejs.org/docs/latest-v22.x/api/>.

[8] **Express.js Team**. "Express.js Documentation - Version 4.18.2," *Express*, 2024. [Online]. Available: <https://expressjs.com/en/4x/api.html>.

[9] **MongoDB Inc**. "MongoDB Manual," *MongoDB*, 2024. [Online]. Available: <https://www.mongodb.com/docs/manual/>.

[10] **Automattic**. "Mongoose Documentation - Version 8.0.3," *Mongoose*, 2024. [Online]. Available: <https://mongoosejs.com/docs/guide.html>.

[11] **Socket.IO Team**. "Socket.IO Documentation - Version 4.8.1," *Socket.IO*, 2024. [Online]. Available: <https://socket.io/docs/v4/>.

[12] **Auth0 Inc**. "jsonwebtoken Documentation - Version 9.0.2," *GitHub*, 2024. [Online]. Available: <https://github.com/auth0/node-jsonwebtoken>.

### Infrastructure and Orchestration

[13] **Cloud Native Computing Foundation**. "Kubernetes Documentation," *Kubernetes*, 2024. [Online]. Available: <https://kubernetes.io/docs/home/>.

[14] **Kubernetes SIG CLI**. "Minikube Documentation," *Minikube*, 2024. [Online]. Available: <https://minikube.sigs.k8s.io/docs/>.

[15] **Docker Inc**. "Docker Desktop for Windows Documentation," *Docker*, 2024. [Online]. Available: <https://docs.docker.com/desktop/windows/>.

[16] **Kubernetes SIG Network**. "NGINX Ingress Controller Documentation," *Kubernetes*, 2024. [Online]. Available: <https://kubernetes.github.io/ingress-nginx/>.

[17] **Kubernetes Authors**. "Kubernetes JavaScript Client - Version 1.2.0," *GitHub*, 2024. [Online]. Available: <https://github.com/kubernetes-client/javascript>.

### Cloud Services and Networking

[18] **Cloudflare Inc**. "Cloudflare Tunnel Documentation," *Cloudflare*, 2024. [Online]. Available: <https://developers.cloudflare.com/cloudflare-one/connections/connect-networks/>.

[19] **Google LLC**. "Google Identity - OAuth 2.0 for Web Server Applications," *Google Developers*, 2024. [Online]. Available: <https://developers.google.com/identity/protocols/oauth2/web-server>.

[20] **Google LLC**. "Google Auth Library for Node.js - Version 9.15.1," *GitHub*, 2024. [Online]. Available: <https://github.com/googleapis/google-auth-library-nodejs>.

## Recursos Técnicos Específicos Aplicados

### Component Libraries and UI Frameworks

[21] **Radix UI Team**. "Radix Primitives Documentation," *Radix UI*, 2024. [Online]. Available: <https://www.radix-ui.com/primitives>.

[22] **Lucide Team**. "Lucide React Icons - Version 0.462.0," *Lucide*, 2024. [Online]. Available: <https://lucide.dev/guide/packages/lucide-react>.

[23] **TanStack**. "TanStack Query Documentation - Version 5.56.2," *TanStack*, 2024. [Online]. Available: <https://tanstack.com/query/latest>.

### Development Tools and Build Systems

[24] **Vitejs Team**. "Vite Plugin React SWC - Version 3.5.0," *GitHub*, 2024. [Online]. Available: <https://github.com/vitejs/vite-plugin-react-swc>.

[25] **ESLint Team**. "ESLint Documentation - Version 9.9.0," *ESLint*, 2024. [Online]. Available: <https://eslint.org/docs/latest/>.

[26] **Nodemon Team**. "Nodemon Documentation - Version 3.0.2," *GitHub*, 2024. [Online]. Available: <https://github.com/remy/nodemon>.

### Security and Authentication

[27] **Google LLC**. "Google OAuth for React Applications," *GitHub*, 2024. [Online]. Available: <https://github.com/MomenSherif/react-oauth>.

[28] **CORS Team**. "CORS Express Middleware - Version 2.8.5," *GitHub*, 2024. [Online]. Available: <https://github.com/expressjs/cors>.

[29] **dotenv Team**. "dotenv Documentation - Version 16.3.1," *GitHub*, 2024. [Online]. Available: <https://github.com/motdotla/dotenv>.

## Recursos de Containerización y Kubernetes

### Kubernetes Manifests and Configuration

[30] **Kubernetes API Reference**. "Pod v1 core," *Kubernetes*, 2024. [Online]. Available: <https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.28/#pod-v1-core>.

[31] **Kubernetes API Reference**. "Service v1 core," *Kubernetes*, 2024. [Online]. Available: <https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.28/#service-v1-core>.

[32] **Kubernetes API Reference**. "Ingress v1 networking.k8s.io," *Kubernetes*, 2024. [Online]. Available: <https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.28/#ingress-v1-networking-k8s-io>.

[33] **Kubernetes Documentation**. "Persistent Volumes," *Kubernetes*, 2024. [Online]. Available: <https://kubernetes.io/docs/concepts/storage/persistent-volumes/>.

### Container Runtime and Storage

[34] **Kubernetes Documentation**. "Storage Classes," *Kubernetes*, 2024. [Online]. Available: <https://kubernetes.io/docs/concepts/storage/storage-classes/>.

[35] **Minikube Contributors**. "Minikube hostPath Provisioner," *GitHub*, 2024. [Online]. Available: <https://github.com/kubernetes/minikube/tree/master/deploy/addons/storage-provisioner>.

## Design System and Styling Resources

### CSS Framework and Animation

[36] **TailwindCSS Team**. "Tailwind CSS Animation Utilities," *Tailwind Labs*, 2024. [Online]. Available: <https://tailwindcss.com/docs/animation>.

[37] **Mantine Team**. "tailwindcss-animate Plugin," *GitHub*, 2024. [Online]. Available: <https://github.com/jamiebuilds/tailwindcss-animate>.

### Form Handling and Validation

[38] **React Hook Form Team**. "React Hook Form Documentation - Version 7.53.0," *React Hook Form*, 2024. [Online]. Available: <https://react-hook-form.com/docs>.

## Real-time Communication and WebSockets

### Socket.IO Implementation

[40] **Socket.IO Team**. "Socket.IO Client API," *Socket.IO*, 2024. [Online]. Available: <https://socket.io/docs/v4/client-api/>.

[41] **Socket.IO Team**. "Socket.IO Server API," *Socket.IO*, 2024. [Online]. Available: <https://socket.io/docs/v4/server-api/>.

[42] **Socket.IO Team**. "Handling CORS with Socket.IO," *Socket.IO*, 2024. [Online]. Available: <https://socket.io/docs/v4/handling-cors/>.

## Automation and Scripting Resources

### PowerShell Automation

[43] **Microsoft Corporation**. "PowerShell Documentation," *Microsoft Docs*, 2024. [Online]. Available: <https://docs.microsoft.com/en-us/powershell/>.

[44] **Microsoft Corporation**. "Windows Terminal Documentation," *Microsoft Docs*, 2024. [Online]. Available: <https://docs.microsoft.com/en-us/windows/terminal/>.

### OpenSSL and TLS Configuration

[45] **OpenSSL Software Foundation**. "OpenSSL Documentation," *OpenSSL*, 2024. [Online]. Available: <https://www.openssl.org/docs/>.

[46] **OpenSSL Software Foundation**. "OpenSSL x509 Manual," *OpenSSL*, 2024. [Online]. Available: <https://www.openssl.org/docs/man1.1.1/man1/x509.html>.

## Development Patterns and Best Practices

### React Patterns and Architecture

[47] **React Team**. "React Context API," *React*, 2024. [Online]. Available: <https://react.dev/reference/react/createContext>.

[48] **React Team**. "Custom Hooks in React," *React*, 2024. [Online]. Available: <https://react.dev/learn/reusing-logic-with-custom-hooks>.

### Node.js API Design

[49] **Express.js Team**. "Express.js Middleware Guide," *Express*, 2024. [Online]. Available: <https://expressjs.com/en/guide/using-middleware.html>.

[50] **Express.js Team**. "Express.js Error Handling," *Express*, 2024. [Online]. Available: <https://expressjs.com/en/guide/error-handling.html>.

## Networking and DNS Configuration

### DNS and Subdomain Management

[51] **Cloudflare Inc**. "Cloudflare DNS Management," *Cloudflare*, 2024. [Online]. Available: <https://developers.cloudflare.com/dns/>.

[52] **Cloudflare Inc**. "Wildcard DNS Records," *Cloudflare*, 2024. [Online]. Available: <https://developers.cloudflare.com/dns/manage-dns-records/reference/wildcard-dns-records/>.

### Ingress and Load Balancing

[53] **NGINX Inc**. "NGINX Ingress Controller Configuration," *NGINX*, 2024. [Online]. Available: <https://docs.nginx.com/nginx-ingress-controller/configuration/>.

[54] **Kubernetes SIG Network**. "NGINX Ingress WebSocket Support," *Kubernetes*, 2024. [Online]. Available: <https://kubernetes.github.io/ingress-nginx/user-guide/miscellaneous/#websockets>.

## Package Management and Configuration

### npm and Node.js Ecosystem

[55] **npm Inc**. "npm Documentation," *npm*, 2024. [Online]. Available: <https://docs.npmjs.com/>.

[56] **Node.js Foundation**. "package.json Configuration," *Node.js*, 2024. [Online]. Available: <https://nodejs.org/en/knowledge/getting-started/npm/what-is-the-file-package-json/>.

### TypeScript Configuration

[57] **Microsoft Corporation**. "TypeScript tsconfig.json Reference," *TypeScript*, 2024. [Online]. Available: <https://www.typescriptlang.org/tsconfig>.

[58] **Microsoft Corporation**. "TypeScript Module Resolution," *TypeScript*, 2024. [Online]. Available: <https://www.typescriptlang.org/docs/handbook/module-resolution.html>.

## Academic and Technical References

### Software Architecture and Design Patterns

[59] **Martin Fowler**. "Patterns of Enterprise Application Architecture," *Addison-Wesley Professional*, 2002.

[60] **Robert C. Martin**. "Clean Code: A Handbook of Agile Software Craftsmanship," *Prentice Hall*, 2008.

### Web Development and Modern JavaScript

[61] **Mozilla Developer Network**. "JavaScript Guide," *MDN Web Docs*, 2024. [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide>.

[62] **Mozilla Developer Network**. "Web APIs Reference," *MDN Web Docs*, 2024. [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/API>.

## Standards and Specifications

### Web Standards

[63] **World Wide Web Consortium**. "HTML Living Standard," *WHATWG*, 2024. [Online]. Available: <https://html.spec.whatwg.org/>.

[64] **World Wide Web Consortium**. "CSS Specifications," *W3C*, 2024. [Online]. Available: <https://www.w3.org/Style/CSS/specs.en.html>.

### HTTP and WebSocket Protocols

[65] **Internet Engineering Task Force**. "RFC 7230: Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing," *IETF*, 2014. [Online]. Available: <https://tools.ietf.org/html/rfc7230>.

[66] **Internet Engineering Task Force**. "RFC 6455: The WebSocket Protocol," *IETF*, 2011. [Online]. Available: <https://tools.ietf.org/html/rfc6455>.

## Troubleshooting and Problem Resolution Resources

### Stack Overflow Specific Solutions

[67] **Stack Overflow Community**. "JSON.stringify() behavior with Infinity values," *Stack Overflow*, 2024. [Online]. Available: <https://stackoverflow.com/questions/handling-infinity-in-json>.

[68] **Stack Overflow Community**. "WebSocket connections through NGINX Ingress Controller," *Stack Overflow*, 2024. [Online]. Available: <https://stackoverflow.com/questions/nginx-websocket-ingress>.

### GitHub Issues and Solutions

[69] **Kubernetes Community**. "Minikube GPU Support Issues," *GitHub*, 2024. [Online]. Available: <https://github.com/kubernetes/minikube/issues>.

[70] **Cloudflare Community**. "Cloudflare Tunnel Configuration Examples," *GitHub*, 2024. [Online]. Available: <https://github.com/cloudflare/cloudflared/wiki>.

# 8. Anexos

[NeuroPod/Documentacion/punto\_8\_anexos\_neuropod.md at main · Zidane0MA/NeuroPod](https://github.com/Zidane0MA/NeuroPod/blob/main/Documentacion/punto_8_anexos_neuropod.md)