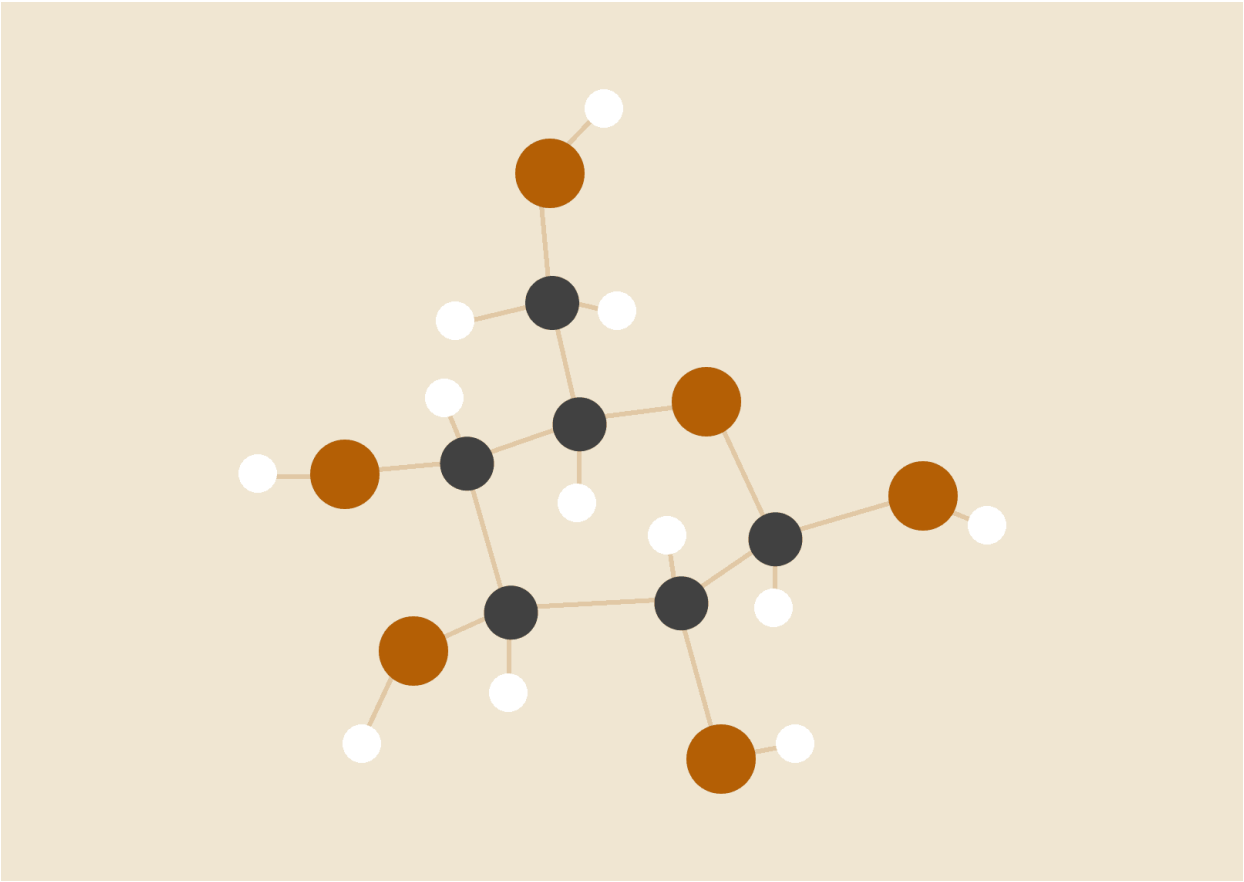
MANUAL TÉCNICO

Proyecto 1



Carlos Pac - 201931012

Fernando Rodriguez - 201931012

Luis Monterroso - 201931012

01/04/2025

9.º Semestre Ingeniería en Ciencias y Sistemas

Introducción

Este documento técnico presenta la arquitectura general, las tecnologías empleadas, la configuración del entorno y el despliegue de una aplicación web desarrollada bajo una estructura modular, utilizando herramientas de código abierto y desplegada en una infraestructura basada en Google Cloud Platform (GCP).

El sistema implementa una arquitectura MVC de tres capas, organizada mediante el enfoque de vertical slicing, que permite una estructura clara y mantenible por módulos. La aplicación se compone de un frontend desarrollado con Nuxt 3 y un backend basado en Spring Boot 3, con persistencia de datos en MariaDB y documentación API generada mediante Swagger.

El entorno de despliegue utiliza una máquina virtual en GCP, donde los servicios se exponen mediante un túnel estático con Ngrok y un servidor Nginx configurado como reverse proxy. Este esquema permite el acceso externo a módulos clave de la aplicación como el frontend, el backend y Jenkins, responsable de gestionar el flujo de integración y entrega continua (CI/CD).

El presente manual incluye también el diagrama de despliegue, la configuración de la instancia, la definición del servicio de Ngrok y la estructura técnica que compone el sistema.

Tecnologías Utilizadas

Tecnologías utilizadas en el backend:

- Java 17
- Spring 3.3.9
- Mysql 8.0 / MariaDB 11.11.10

Tecnologías utilizadas en el frontend:

- Nuxt 3.16
- Typescript
- Tailwind 4
- Vitest

Tecnologías para la documentación:

- Swagger v3

Arquitectura de la aplicación

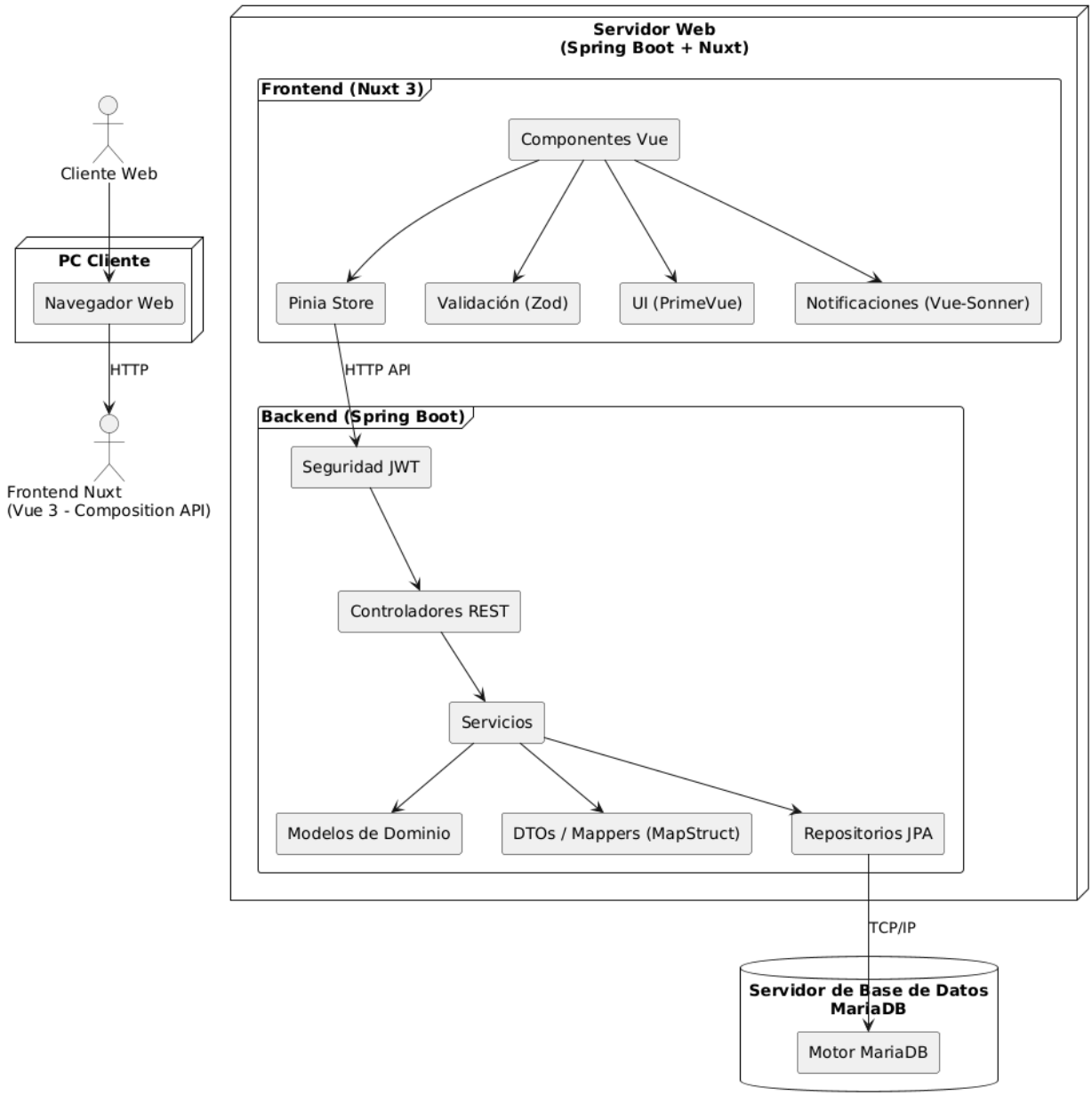


Diagrama de despliegue

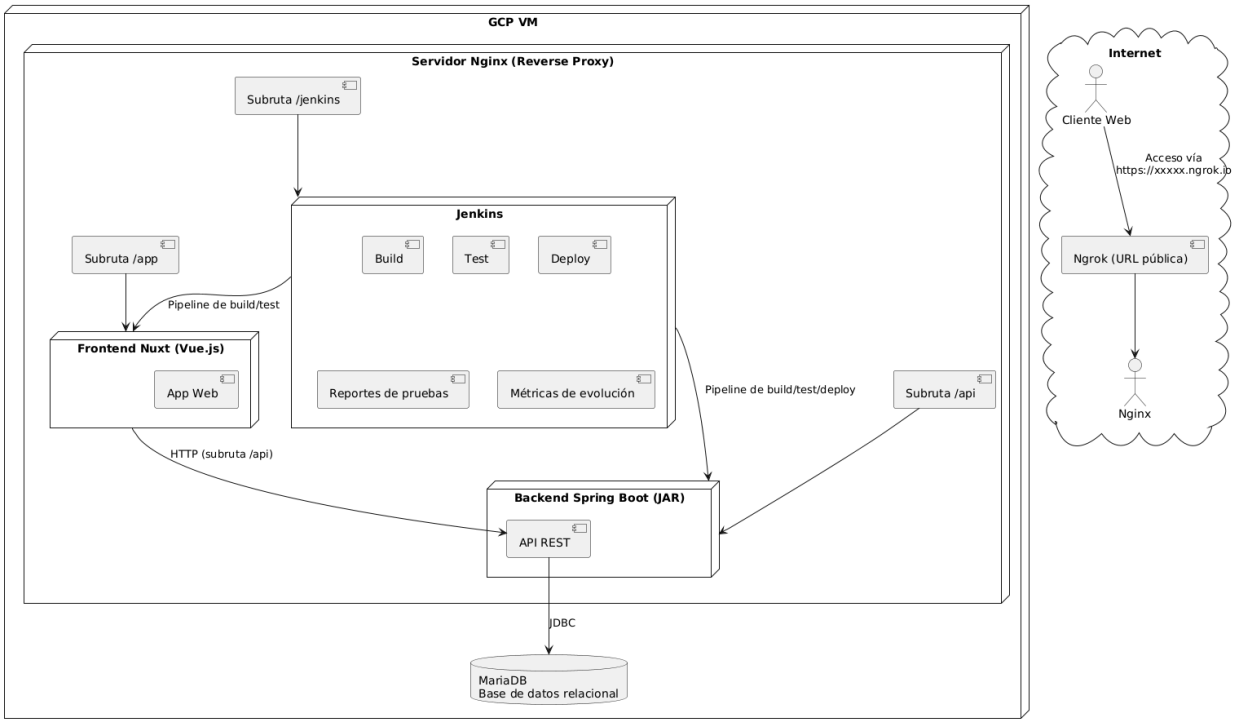
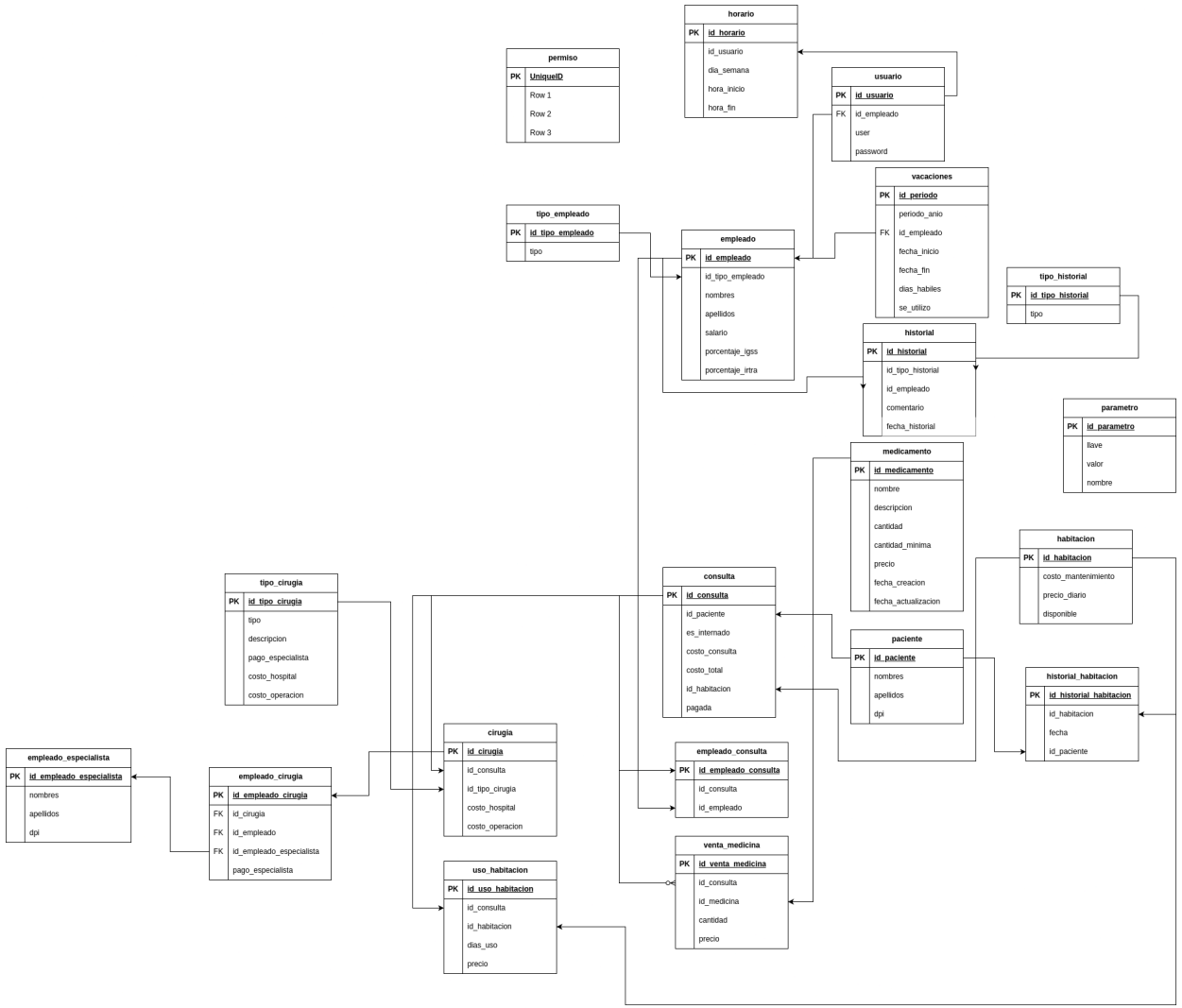
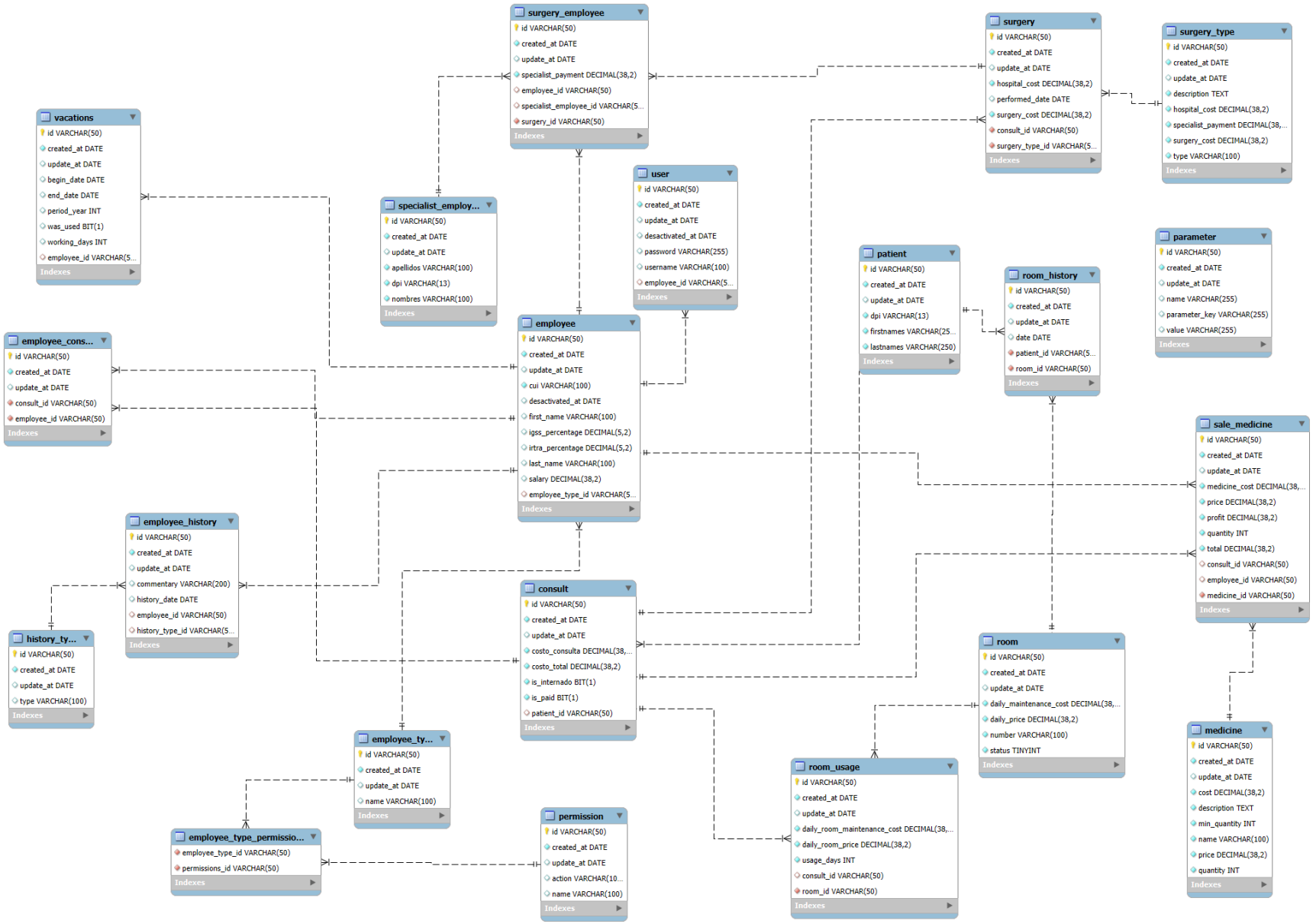


Diagrama DB


horario
PK id_horario
id_usuario
dia_semana
hora_inicio
hora_fin
permiso
PK UniquelD
Row 1
Row 2
Row 3
usuario
PK id_usuario
FK id_empleado
user
password
vacaciones
PK id_periodo
periodo_anio
FK id_empleado
fecha_inicio
fecha_fin
dias_habiles
se_utilizo
tipo_empleado
PK id_tipo_empleado
tipo
empleado
PK id_empleado
id_tipo_empleado
nombres
apellidos
salario
porcentaje_igss
porcentaje_irtra
tipo_historial
PK id_tipo_historial
tipo
historial
PK id_historial
id_tipo_historial
id_empleado
comentario
fecha_historial
parametro
PK id_parametro
llave
valor
nombre
medicamento
PK id_medicamento
nombre
descripcion
cantidad
cantidad_minima
precio
fecha_creacion
fecha_actualizacion
habitacion
PK id_habitacion
costo_mantenimiento
precio_diario
disponible
tipo_cirugia
PK id_tipo_cirugia
tipo
descripcion
pago_especialista
costo_hospital
costo_operacion
consulta
PK id_consulta
id_paciente
es_internado
costo_consulta
costo_total
id_habitacion
pagada
paciente
PK id_paciente
nombres
apellidos
dpi
historial_habitacion
PK id_historial_habitacion
id_habitacion
fecha
id_paciente
empleado_especialista
PK id_empleado_especialista
nombres
apellidos
dpi
empleado_cirugia
PK id_empleado_cirugia
FK id_cirugia
FK id_empleado
FK id_empleado_especialista
pago_especialista
cirugia
PK id_cirugia
id_consulta
id_tipo_cirugia
costo_hospital
costo_operacion
empleado_consulta
PK id_empleado_consulta
id_consulta
id_empleado
venta_medicina
PK id_venta_medicina
id_consulta
id_medicina
cantidad
precio
uso_habitacion
PK id_uso_habitacion
id_consulta
id_habitacion
dias_uso
precio


surgery_employee
id VARCHAR(50)
created_at DATE
update_at DATE
specialist_payment DECIMAL(38,2)
employee_id VARCHAR(50)
specialist_employee_id VARCHAR(5...
surgery_id VARCHAR(50)
Indexes
surgery
id VARCHAR(50)
created_at DATE
update_at DATE
hospital_cost DECIMAL(38,2)
performed_date DATE
surgery_cost DECIMAL(38,2)
consult_id VARCHAR(50)
surgery_type_id VARCHAR(5...
Indexes
surgery_type
id VARCHAR(50)
created_at DATE
update_at DATE
description TEXT
hospital_cost DECIMAL(38,2)
specialist_payment DECIMAL(38,...
surgery_cost DECIMAL(38,2)
type VARCHAR(100)
Indexes
vacations
id VARCHAR(50)
created_at DATE
update_at DATE
begin_date DATE
end_date DATE
period_year INT
was_used BIT(1)
working_days INT
employee_id VARCHAR(5...
Indexes
specialist_employ...
id VARCHAR(50)
created_at DATE
update_at DATE
apellidos VARCHAR(100)
dpi VARCHAR(13)
nombres VARCHAR(100)
Indexes
user
id VARCHAR(50)
created_at DATE
update_at DATE
desactivated_at DATE
password VARCHAR(255)
username VARCHAR(100)
employee_id VARCHAR(5...
Indexes
patient
id VARCHAR(50)
created_at DATE
update_at DATE
dpi VARCHAR(13)
firstnames VARCHAR(25...
lastnames VARCHAR(250)
Indexes
room_history
id VARCHAR(50)
created_at DATE
update_at DATE
date DATE
patient_id VARCHAR(5...
room_id VARCHAR(50)
Indexes
parameter
id VARCHAR(50)
created_at DATE
update_at DATE
name VARCHAR(255)
parameter_key VARCHAR(255)
value VARCHAR(255)
Indexes
employee_cons...
id VARCHAR(50)
created_at DATE
update_at DATE
consult_id VARCHAR(50)
employee_id VARCHAR(50)
Indexes
employee
id VARCHAR(50)
created_at DATE
update_at DATE
cui VARCHAR(100)
desactivated_at DATE
first_name VARCHAR(100)
igss_percentage DECIMAL(5,2)
irtra_percentage DECIMAL(5,2)
last_name VARCHAR(100)
salary DECIMAL(38,2)
employee_type_id VARCHAR(5...
Indexes
sale_medicine
id VARCHAR(50)
created_at DATE
update_at DATE
medicine_cost DECIMAL(38,...
price DECIMAL(38,2)
profit DECIMAL(38,2)
quantity INT
total DECIMAL(38,2)
consult_id VARCHAR(50)
employee_id VARCHAR(50)
medicine_id VARCHAR(50)
Indexes
employee_history
id VARCHAR(50)
created_at DATE
update_at DATE
commentary VARCHAR(200)
history_date DATE
employee_id VARCHAR(50)
history_type_id VARCHAR(5...
Indexes
consult
id VARCHAR(50)
created_at DATE
update_at DATE
costo_consulta DECIMAL(38,...
costo_total DECIMAL(38,2)
is_internado BIT(1)
is_paid BIT(1)
patient_id VARCHAR(50)
Indexes
history_ty...
id VARCHAR(50)
created_at DATE
update_at DATE
type VARCHAR(100)
Indexes
room
id VARCHAR(50)
created_at DATE
update_at DATE
daily_maintenance_cost DECIMAL(38,...
daily_price DECIMAL(38,2)
number VARCHAR(100)
status TINYINT
Indexes
medicine
id VARCHAR(50)
created_at DATE
update_at DATE
cost DECIMAL(38,2)
description TEXT
min_quantity INT
name VARCHAR(100)
price DECIMAL(38,2)
quantity INT
Indexes
employee_ty...
id VARCHAR(50)
created_at DATE
update_at DATE
name VARCHAR(100)
Indexes
room_usage
id VARCHAR(50)
created_at DATE
update_at DATE
daily_room_maintenance_cost DECIMAL(38,...
daily_room_price DECIMAL(38,2)
usage_days INT
consult_id VARCHAR(50)
room_id VARCHAR(50)
Indexes
employee_type_permissio...
employee_type_id VARCHAR(50)
permissions_id VARCHAR(50)
Indexes
permission
id VARCHAR(50)
created_at DATE
update_at DATE
action VARCHAR(10...
name VARCHAR(100)
Indexes

Configuraciones de la instancia

1. Configuración de dominio por Ngrok

Debido a que la aplicación necesita de tener un nombre de acceso el cual no cambie para establecer comunicación con los servicios de **CI/CD** y acceso a otros módulos del sistema se tomó la decisión de usar **Ngrok** como servicio de redirección y en conjunto con **Nginx** el reverse proxy de acceso a las demás aplicaciones.

Para mantener la funcionalidad de **Ngrok** se crea un servicio de systemD en la VM de GCP con los siguientes comandos:

```
sudo nano /etc/systemd/system/ngrok-static.service
```

Agregando el siguiente contenido en la configuración del servicio.

```
[Unit]
Description=Ngrok HTTP tunnel
After=network.target

[Service]
ExecStart=/usr/local/bin/ngrok http --url=STATIC_URL_NGROK 80
Restart=always
User=carlosbpac
WorkingDirectory=/home/carlosbpac
StandardOutput=file:/var/log/ngrok.log
StandardError=file:/var/log/ngrok.log

[Install]
WantedBy=multi-user.target
```

Usando los siguientes comandos podemos controlar el **inicio**, **status**, **apagado** e **inicio automático** en la instancia de **GCP**.

```
#Status del servicio
```

```
sudo systemctl status ngrok-static.service

#Inicio del servicio
sudo systemctl start ngrok-static.service

#Habilitar el inicio automatico del servicio
sudo systemctl enable ngrok-static.service
```

Para visualizar los logs del servicio

```
journalctl -u ngrok-static.service -f
```

2. Configuración de Nginx

sudo cat /etc/nginx/sites-available/default

```
##
# You should look at the following URL's in order to grasp a solid
understanding
# of Nginx configuration files in order to fully unleash the power of
Nginx.
# https://www.nginx.com/resources/wiki/start/
#
https://www.nginx.com/resources/wiki/start/topics/tutorials/config_pitfalls
/
# https://wiki.debian.org/Nginx/DirectoryStructure
#
# In most cases, administrators will remove this file from sites-enabled/
and
# leave it as reference inside of sites-available where it will continue to
be
# updated by the nginx packaging team.
#
# This file will automatically load configuration files provided by other
# applications, such as Drupal or Wordpress. These applications will be
made
# available underneath a path with that package name, such as /drupal8.
#
# Please see /usr/share/doc/nginx-doc/examples/ for more detailed examples.
##

server {
    listen 80 default_server;
    listen [::]:80 default_server;

    server_name _; # O tu dominio o subdominio si tienes uno

    # ----------- Proxy para el BACKEND (API en 8080) -----------
    location /api/ {
        proxy_pass http://localhost:8080/;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
```

```
    # ----------- Proxy para el FRONTEND (APP en 3000) -----------
    location /app/ {
        proxy_pass http://localhost:3000/app/;

        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }

    # ----------- Proxy para JENKINS (en 8081) -----------
    location /jenkins/ {
        proxy_pass http://localhost:8081/jenkins/;

        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;

        proxy_redirect     off;
        proxy_http_version 1.1;
        proxy_request_buffering off;
    }
}
```

1. Definición del servidor

```
listen 80 default_server;
listen [::]:80 default_server;
```

Estas directivas indican que el servidor Nginx escuchará en el puerto 80 (protocolo HTTP), tanto para conexiones IPv4 como IPv6. El modificador default_server establece este bloque como el servidor predeterminado para todas las solicitudes que no coincidan con otro server_name.

```
server_name _;
```

El carácter _ se utiliza como comodín para aceptar cualquier nombre de host. En caso de contar con un dominio o subdominio específico, este valor puede ser reemplazado por el nombre correspondiente (por ejemplo: server_name miapp.midominio.com;).

2. Redirección de solicitudes a la API Backend (/api/)

```
location /api/ {
    proxy_pass http://localhost:8080/;
    ...
}
```

Todas las solicitudes que inicien con /**api**/ serán redirigidas al servicio de backend que se ejecuta localmente en el puerto 8080. Las cabeceras adicionales garantizan que el backend reciba la información original de la petición, como el nombre del host, la IP del cliente y el esquema del protocolo.

3. Redirección de solicitudes a la aplicación frontend (/app/)

```
location /app/ {
    proxy_pass http://localhost:3000/app/;
    ...
}
```

Este bloque redirige las solicitudes con prefijo /app/ al servicio frontend desplegado en el puerto 3000, suponiendo que la aplicación está configurada para servir desde esa subruta. Es fundamental que el enrutamiento interno del frontend (en este caso Nuxt) esté alineado con esta ruta base.

4. Redirección de solicitudes a Jenkins (/jenkins/)

```
location /jenkins/ {
    proxy_pass http://localhost:8081/jenkins/;
    ...
}
```

Las solicitudes con el prefijo /jenkins/ se redirigen al servicio de Jenkins en el puerto 8081. Además de las cabeceras estándar, se configuran parámetros específicos que aseguran el correcto funcionamiento de Jenkins tras el proxy:

- **proxy_redirect off:** desactiva redirecciones automáticas que podrían romper el contexto /jenkins/.
- **proxy_http_version 1.1:** Jenkins requiere HTTP/1.1 para ciertas operaciones.
- **proxy_request_buffering off:** mejora el manejo de peticiones grandes o prolongadas, como despliegues o cargas de archivos.

BACKEND

El backend del sistema está desarrollado en Java 17 utilizando el framework Spring Boot 3.3.9 y sigue una arquitectura MVC de tres capas, organizada bajo el principio de vertical slicing. Esta organización permite que cada módulo del sistema contenga de forma encapsulada sus propios controladores, servicios, modelos, DTOs, puertos, mappers y repositorios, facilitando la escalabilidad, mantenibilidad y reutilización del código.

La comunicación entre el frontend y el backend se realiza a través de una API RESTful, compuesta por múltiples controladores organizados por dominio funcional, como empleados, pacientes, consultas, medicamentos, habitaciones, permisos, usuarios, reportes, entre otros. Cada controlador expone endpoints específicos para la gestión de las respectivas entidades.

El sistema implementa una capa de seguridad basada en JWT (JSON Web Tokens), mediante filtros personalizados y utilidades de generación y validación de tokens. Esta capa permite autenticar y autorizar las solicitudes entrantes, asegurando el acceso restringido a los recursos de la API.

Para la persistencia de datos, el backend utiliza JPA y Hibernate, integrados con una base de datos relacional MariaDB 11.11.10, en la que se almacenan todas las entidades del sistema. Cada módulo cuenta con su propia interfaz de repositorio, alineada con el patrón Repository, permitiendo consultas personalizadas mediante Specifications.

Asimismo, el sistema hace uso de herramientas de mapeo automático como MapStruct para convertir entidades en objetos de transferencia (DTOs) y viceversa, facilitando la separación entre la lógica de negocio y las estructuras de presentación de datos.

La lógica de negocio se encuentra centralizada en los servicios de cada módulo. Estos servicios implementan interfaces o puertos definidos previamente, siguiendo el patrón Ports & Adapters, lo que permite mantener una baja dependencia entre capas.

El backend también incluye la configuración de Swagger v3, lo que permite la generación automática de la documentación de la API, facilitando la exploración, prueba y consumo de los servicios REST desde una interfaz web.

Módulos Funcionales del Sistema

El backend está organizado en múltiples módulos, cada uno de los cuales encapsula su lógica, controladores, servicios, modelos y demás componentes que son los siguientes:

1. **Autenticación y Seguridad:**

   Gestión de inicio de sesión y generación/validación de tokens JWT.

2. **Usuarios:**

   Registro y gestión de usuarios del sistema.

3. **Empleados:**

   Gestión de empleados, roles, historial, tipos, especialistas y períodos laborales.

4. **Pacientes:**

   Registro y mantenimiento de la información de pacientes.

5. **Consultas médicas:**

   Gestión de consultas, asignación de empleados y estados clínicos.

6. **Cirugías:**

   Gestión de cirugías, tipos de cirugía y personal asignado.

7. **Medicamentos y Farmacia:**

   Gestión de inventario de medicamentos y registro de ventas por consulta o mostrador.

8. **Habitaciones:**

   Administración del uso, historial y disponibilidad de habitaciones.

9. **Permisos del sistema:**

   Control de permisos internos por tipo de usuario.

10. **Vacaciones:**

    Registro y control de periodos vacacionales de empleados.

11. **Reportes:**

    Generación de reportes financieros, históricos y de rendimiento para empleados y medicamentos.

12. **Parámetros del sistema**

    Definición y consulta de parámetros internos para configuración general.

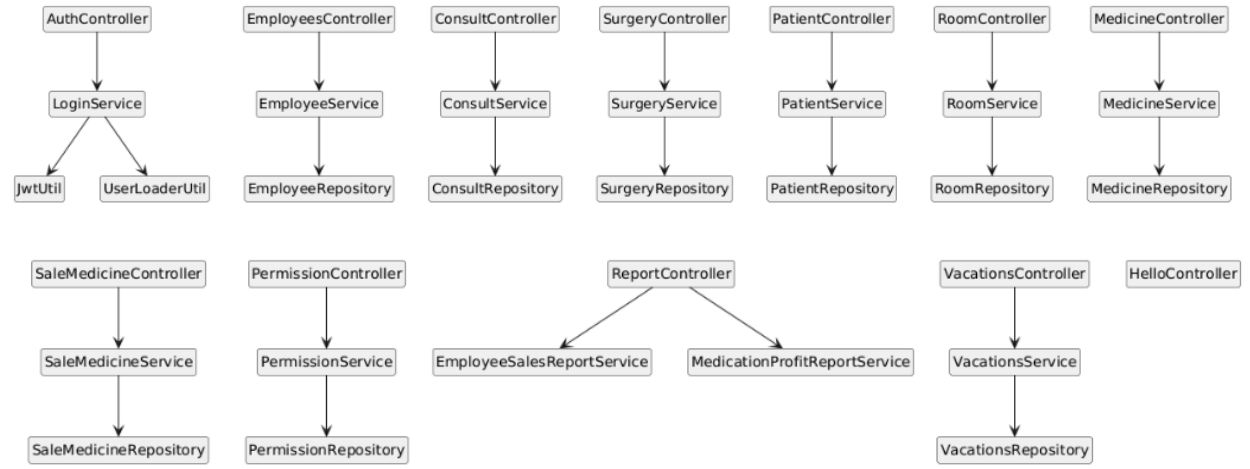
Estructura general de carpetas:

```
hospitalApi/
├── auth/
│   ├── jwt/
│   │   ├── filters/
│   │   ├── ports/
│   │   └── utils/
│   └── login/
│       ├── controllers/
│       ├── dtos/
│       ├── ports/
│       ├── service/
│       └── utils/
├── consults/
│   ├── controllers/
│   ├── dtos/
│   ├── mappers/
│   ├── models/
│   ├── port/
│   ├── repositories/
│   ├── services/
│   └── specifications/
├── employees/
│   ├── controllers/
│   ├── dtos/
│   ├── enums/
│   ├── mappers/
│   ├── models/
│   ├── ports/
```

```
│   ├── repositories/
│   ├── services/
│   └── specifications/
├── hello/
│   ├── controllers/
│   └── dtos/
├── medicines/
│   ├── controllers/
│   ├── dtos/
│   ├── mappers/
│   ├── models/
│   ├── ports/
│   ├── repositories/
│   ├── services/
│   └── utils/
├── parameters/
│   ├── enums/
│   ├── models/
│   ├── ports/
│   ├── repositories/
│   └── services/
├── patients/
│   ├── controllers/
│   ├── dtos/
│   ├── mappers/
│   ├── models/
│   ├── ports/
│   ├── repositories/
│   └── services/
├── permissions/
│   ├── controllers/
│   ├── dtos/
│   ├── enums/
│   ├── mappers/
│   ├── models/
│   ├── ports/
│   ├── repositories/
│   └── services/
├── reports/
│   ├── controllers/
│   ├── dtos/
│   │   ├── request/
```

```
│   │   └── response/
│   ├── ports/
│   └── services/
├── rooms/
│   ├── controllers/
│   ├── dtos/
│   ├── enums/
│   ├── mappers/
│   ├── models/
│   ├── ports/
│   ├── repositories/
│   └── services/
├── shared/
│   ├── config/
│   ├── dtos/
│   ├── enums/
│   ├── exceptions/
│   ├── models/
│   └── utils/
├── surgery/
│   ├── controllers/
│   ├── dtos/
│   ├── mappers/
│   ├── models/
│   ├── ports/
│   ├── repositories/
│   └── services/
├── users/
│   ├── dtos/
│   ├── mappers/
│   ├── models/
│   ├── ports/
│   ├── repositories/
│   ├── services/
│   └── utils/
└── vacations/
    ├── controllers/
    ├── dtos/
    ├── mappers/
    ├── models/
    ├── ports/
    ├── repositories/
```

```
└── services/
```



FRONTEND

El diseño del frontend de la aplicación sigue un enfoque modular y desacoplado, basado en las capacidades del framework Nuxt 3 y la filosofía de separación de responsabilidades. La estructura del proyecto está organizada por función y dominio, con cada sección alojada bajo carpetas específicas como pages/, lib/api/, components/ y stores/.

Páginas y enrutamiento

El directorio pages/ representa el núcleo de la navegación de la aplicación. Cada carpeta dentro de pages/ corresponde a una sección del sistema (por ejemplo, admin, pacientes, consultas), y Nuxt genera automáticamente las rutas a partir de esta estructura. Esto permite un mapeo directo entre las funcionalidades del sistema y su representación en la interfaz de usuario.

Consumo de servicios y modularidad

Los servicios que consumen la API REST se encuentran en lib/api/, organizados por entidad (admin, medicines, consults, etc.). Esta separación favorece la reutilización de lógica y el desacoplamiento entre vistas y lógica de negocio. El consumo de los servicios se hace a través de **Pinia-Colada** que permite un manejo granular de las peticiones y respuestas.

Componentes reutilizables

Los componentes visuales compartidos, como tarjetas o botones, se ubican en components/, con subcarpetas específicas como cards/ que contienen unidades visuales reutilizables, lo cual mejora la mantenibilidad del diseño.

Estado global y lógica compartida

El estado global de la aplicación se gestiona con **Pinia**, y se encuentra en la carpeta stores/, mientras que la lógica reutilizable (composables personalizados) se encuentra en composables/. Estas herramientas permiten compartir datos y comportamientos entre distintas partes de la aplicación de forma reactiva y controlada.

Diseño y validación

Para el diseño visual se utiliza **Tailwind CSS v4**, y para la validación de formularios se emplea **Zod**, que garantiza el control tipado de datos y la integridad al interactuar con el backend. El conjunto de herramientas UI se apoya en **PrimeVue**.

Seguridad y middleware

La carpeta middleware/ contiene funciones globales como la verificación de autenticación para proteger rutas. En particular, se ha definido un middleware llamado 01.auth.global.ts, que controla el acceso a las vistas protegidas.

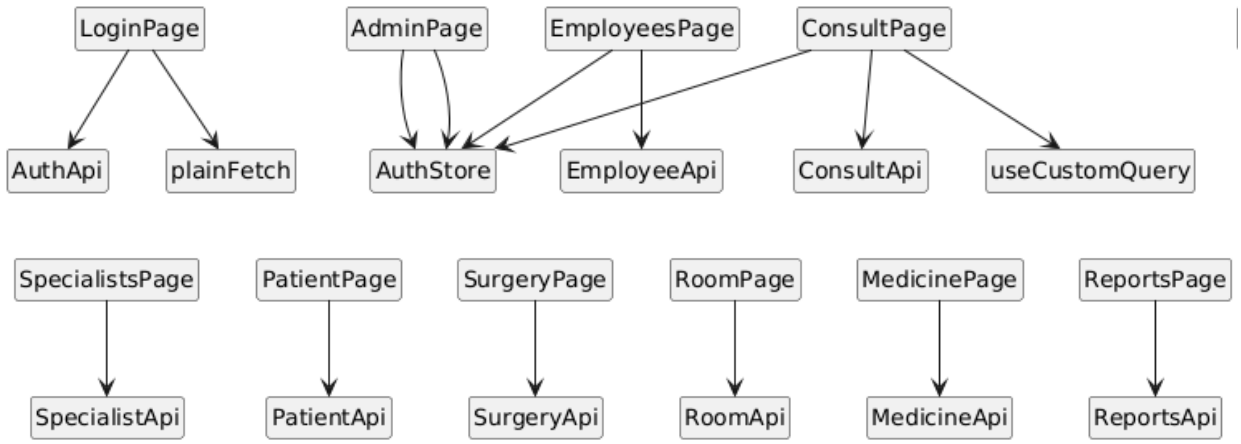
Testing y CI/CD

La aplicación cuenta con pruebas automatizadas realizadas con **Vitest**, cuyos archivos .spec.ts están presentes tanto en las páginas como en los servicios. Estas pruebas se integran en el pipeline de Jenkins para validar la calidad del código antes del despliegue.

Estructura general de carpetas:

```
app
|-- components
|   +-- cards
|-- composables
|-- layouts
|-- lib
|   |-- api.admin
|   |-- api.consults
```

```
|   |-- api.habitaciones
|   |-- api.medicines
|   |-- api.patients
|   |-- api.reportes
|   +-- api.surgeries
|-- middleware
|-- pages
|   |-- admin
|   |-- consultas
|   |-- cirugias
|   |-- farmacia
|   |-- habitaciones
|   |-- pacientes
|   |-- reportes
|   +-- login
|-- stores
|-- utils
+-- server
    +-- api
```



JENKINS

Para adaptar Jenkins a las necesidades específicas del sistema de integración y entrega continua (CI/CD) desarrollado, se han instalado los siguientes **plugins adicionales**, los cuales no forman parte del conjunto predeterminado que Jenkins incluye tras una instalación estándar:

1. Automatización de pipelines

- **Pipeline (Declarative, Stage View, SCM Step, Input Step, etc.)**
  Habilita la definición de flujos de trabajo como código mediante Jenkinsfile, con etapas visibles, control de versiones, entradas manuales y lógica condicional.

- **Pipeline: GitHub Groovy Libraries**
  Permite importar bibliotecas Groovy directamente desde repositorios GitHub, facilitando la reutilización de lógica común entre jobs.

- **Pipeline Graph View Plugin**
  Añade una visualización gráfica de las etapas del pipeline para una interpretación más clara del flujo de ejecución.

2. Integración con control de versiones

- **Git Plugin y GitHub Plugin**
  Proveen soporte completo para clonar repositorios, ejecutar builds basados en ramas, usar webhooks y visualizar cambios desde GitHub.

- **GitHub Branch Source**
  Facilita la detección automática de ramas y pull requests en repositorios GitHub para pipelines multibranch.

3. Construcción y pruebas

- **Gradle Plugin**
  Integra el sistema de construcción Gradle con Jenkins, permitiendo compilar y testear aplicaciones Java como el backend del sistema.

- **JaCoCo Plugin**

Publica reportes de cobertura de código generados por JaCoCo, útil para evaluar la calidad de pruebas del backend.

- **JUnit Plugin**
  Presenta resultados de pruebas unitarias en formato JUnit, integrando métricas visuales al entorno Jenkins.

- **HTML Publisher Plugin**
  Permite mostrar reportes HTML generados por tareas de compilación, como métricas o cobertura del frontend.

4. Gestión de seguridad y credenciales

- **Credentials Binding Plugin**
  Permite inyectar credenciales como variables de entorno durante la ejecución del pipeline.

- **SSH Credentials Plugin**
  Facilita el almacenamiento y uso de claves SSH para conexiones seguras con agentes o repositorios remotos.

- **Matrix Authorization Strategy Plugin**
  Habilita un esquema de control de acceso basado en roles y permisos granulares por usuario y proyecto.

- **Script Security Plugin**
  Controla qué scripts pueden ejecutarse en Jenkins, protegiendo contra ejecuciones no autorizadas.

- **Plain Credentials Plugin**
  Permite manejar claves simples o archivos secretos dentro de Jenkins de manera segura.

5. Optimización del entorno

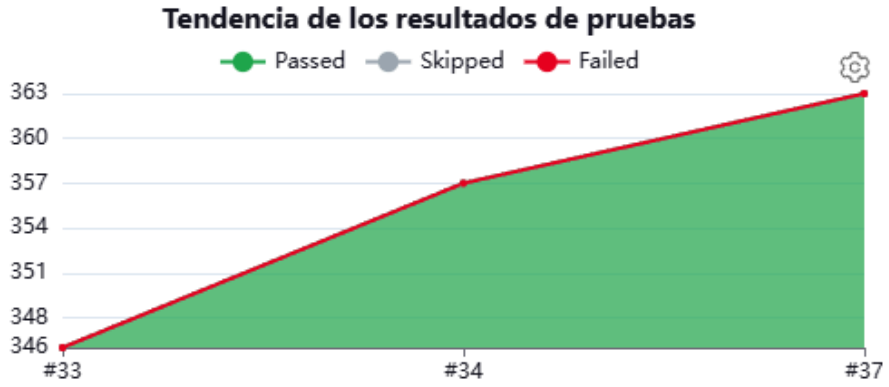
- **Workspace Cleanup Plugin**
  Limpia automáticamente los archivos del workspace al finalizar los jobs, manteniendo un entorno ordenado.

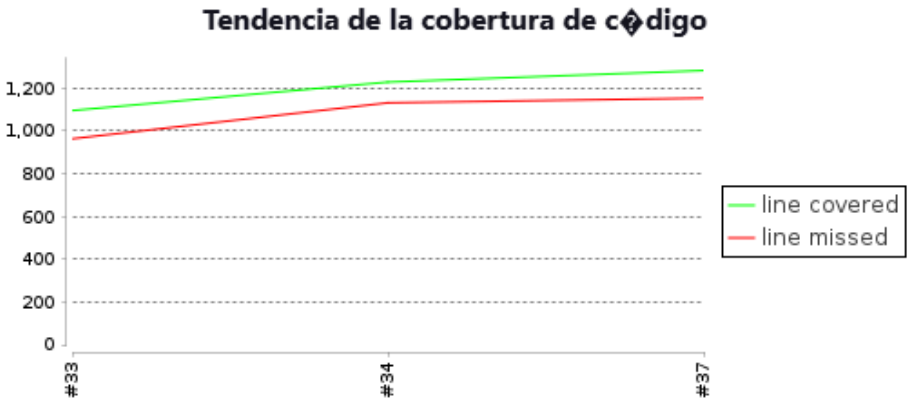
- **Build Timeout Plugin**
  Finaliza builds que superen un tiempo máximo establecido, previniendo bloqueos de recursos.

6. Visualización avanzada

- **ECharts API Plugin**
  Habilita la integración de gráficos dinámicos interactivos, ideal para dashboards personalizados de métricas.

Stage View





enlarge

- Status
- Changes
- Construir ahora
- Configurar
- Borrar Pipeline
- Full Stage View
- Jacoco Code Coverage
- Frontend Coverage
- Stages
- Rename
- Tendencia de cobertura
- Pipeline Syntax
- GitHub Hook Log

Pipeline de Jenkins

El siguiente pipeline de Jenkins define un proceso automatizado de integración y despliegue continuo (CI/CD), abarcando la compilación, pruebas, análisis de cobertura y despliegue tanto del backend como del frontend. Se ejecuta en cualquier agente disponible (agent any) y utiliza variables de entorno para rutas específicas y versiones.

```
pipeline {
    agent any

    environment {
        NVM_DIR = "/var/lib/jenkins/.nvm"
        DEPLOY_BACKEND_DIR = "/var/api/hospitalApi"
        DEPLOY_BACKEND_JAR = "hospitalApi.jar"
        DEPLOY_FRONTEND_DIR = "/var/www/app_proyecto1_ayd2"
        NODE_VERSION = "22.14.0"
    }

    stages {

        stage('Checkout') {
            steps {
                git branch: 'dev', url:
'https://github.com/FernandoJRR/Proyecto1AyD2.git'
            }
        }

        stage('Build Backend') {
            steps {
                script {
                    echo "🚀 Compilando backend y ejecutando pruebas..."
                    dir('hospitalApi') {
                        sh 'mvn clean verify' // Compila, ejecuta tests y genera
Jacoco

                        // Publicar resultados de pruebas
                        junit 'target/surefire-reports/*.xml'

                        // Publicar cobertura Jacoco (plugin Jacoco instalado)
                        jacoco execPattern: 'target/jacoco.exec'

                        // Publicar reporte HTML Jacoco (plugin HTML Publisher
instalado)
                        publishHTML (target : [
                            reportDir: 'target/site/jacoco',
```

```
                        reportFiles: 'index.html',
                        reportName: 'Jacoco Code Coverage'
                    ])
                }
            }
        }
    }

    stage('Deploy Backend') {
        steps {
            script {
                dir('hospitalApi') {
                    echo "🧹 Eliminando JAR anterior..."
                    sh "rm -f $DEPLOY_BACKEND_DIR/$DEPLOY_BACKEND_JAR"

                    echo "📦 Copiando nuevo JAR a $DEPLOY_BACKEND_DIR ..."
                    sh """
                        rsync -av target/hospitalApi-0.0.1-SNAPSHOT.jar
$DEPLOY_BACKEND_DIR/$DEPLOY_BACKEND_JAR
                    """

                    echo "🔄 Reiniciando servicio hospital-api.service..."
                    sh 'sudo systemctl restart hospital-api.service'
                }
            }
        }
    }

    stage('Test Frontend') {
        steps {
            script {
                catchError(buildResult: 'SUCCESS', stageResult: 'FAILURE') {
                    echo "Test de Frontend..."
                    dir('app_proyecto1_ayd2') {

                        echo "🔧 Cargando NVM y usando Node.js
$NODE_VERSION..."
                        sh """
                            export NVM_DIR="$NVM_DIR"
                            [ -s "$NVM_DIR/nvm.sh" ] && . "$NVM_DIR/nvm.sh"
                            nvm use $NODE_VERSION

                            echo "🏗️ Ejecutando test frontend..."
                            npm run coverage
                        """
                    }
                }
```

```
                dir('app_proyecto1_ayd2') {
                    publishHTML (target : [
                        reportDir: 'app/coverage',
                        reportFiles: 'index.html',
                        reportName: 'Frontend Coverage'
                    ])
                }
            }
        }
    }

    stage('Build Frontend') {
        steps {
            script {
                echo "🚀 Compilando frontend..."
                dir('app_proyecto1_ayd2') {

                    echo "🔧 Cargando NVM y usando Node.js $NODE_VERSION..."
                    sh """
                        export NVM_DIR="$NVM_DIR"
                        [ -s "$NVM_DIR/nvm.sh" ] && . "$NVM_DIR/nvm.sh"
                        nvm use $NODE_VERSION

                        echo "📦 Instalando dependencias frontend..."
                        #npm install --legacy-peer-deps
                        #rm -rf node_modules package-lock.json
                        #npm install

                        echo "🏗️ Ejecutando build frontend..."
                        npm run build
                    """
                }
            }
        }
    }

    stage('Deploy Frontend') {
        steps {
            script {
                dir('app_proyecto1_ayd2') {

                    sh """
                        export NVM_DIR="$NVM_DIR"
                        [ -s "$NVM_DIR/nvm.sh" ] && . "$NVM_DIR/nvm.sh"
                        nvm use $NODE_VERSION

                        echo "🛑 Deteniendo PM2 frontend si existe..."
```

```
                            pm2 stop frontend || true

                            echo "🧹 Limpiando el deploy anterior de frontend en
$DEPLOY_FRONTEND_DIR ..."
                            rm -rf $DEPLOY_FRONTEND_DIR/*

                            echo "🚚 Copiando build a $DEPLOY_FRONTEND_DIR ..."
                            rsync -av --delete .output/ $DEPLOY_FRONTEND_DIR/

                            echo "✅ Reiniciando PM2 frontend..."
                            pm2 start frontend || pm2 restart frontend

                            echo "📝 Logs de PM2 frontend..."
                            pm2 logs frontend --lines 20 --nostream
                        """
                    }
                }
            }
        }
    }

    post {
        success {
            echo "🎉 Pipeline completado con éxito."
        }
        failure {
            echo "❌ Algo falló en el pipeline."
        }
    }
}
```

Variables de Entorno

NVM_DIR: Ruta de instalación de NVM usada para gestionar versiones de Node.js.

DEPLOY_BACKEND_DIR: Ruta destino donde se despliega el archivo .jar del backend.

DEPLOY_BACKEND_JAR: Nombre del artefacto generado del backend.

DEPLOY_FRONTEND_DIR: Ruta destino del build de frontend generado.

NODE_VERSION: Versión de Node.js usada en las etapas del frontend.

Etapas del Pipeline

1. **Checkout**

   Clona el repositorio desde GitHub, rama dev. Esta etapa obtiene el código fuente más reciente para trabajar en las siguientes fases.

2. **Build Backend**

   Dentro del directorio hospitalApi, se realiza:

   - Compilación del backend usando Maven (mvn clean verify).
   - Ejecución de pruebas unitarias.
   - Publicación de resultados de pruebas en formato JUnit.
   - Análisis y publicación de cobertura de código mediante JaCoCo.
   - Generación de un reporte HTML para la cobertura, visible desde Jenkins.

3. **Deploy Backend**

   - Elimina versiones anteriores del artefacto .jar.
   - Copia el nuevo .jar generado hacia el directorio de despliegue.
   - Reinicia el servicio hospital-api.service con systemctl, que ejecuta el backend en la máquina destino.

4. **Test Frontend**

   - Cambia al directorio del frontend app_proyecto1_ayd2.
   - Carga el entorno de Node.js mediante NVM.
   - Ejecuta pruebas del frontend utilizando el comando npm run coverage.
   - Publica el reporte de cobertura HTML generado por Vitest (o Jest) mediante el plugin HTML Publisher.

   Nota: Esta etapa utiliza catchError para que el pipeline continúe incluso si las pruebas fallan, marcando la etapa como fallida pero no deteniendo el pipeline completo.

5. **Build Frontend**
    - Carga Node.js con NVM.
    - Instala dependencias del frontend si es necesario (comentado).
    - Ejecuta el build del frontend (npm run build), generando la carpeta .output.
6. **Deploy Frontend**
    - Detiene cualquier instancia previa del frontend gestionada con PM2.
    - Limpia el contenido anterior del directorio de despliegue.
    - Copia el nuevo build al destino con rsync.
    - Inicia o reinicia el proceso frontend con PM2 y muestra los últimos logs.
7. **Sección post**

    success: Muestra un mensaje indicando que el pipeline fue ejecutado exitosamente.

    failure: Informa que ocurrió un error durante la ejecución del pipeline.

CONCLUSIÓN

El desarrollo e implementación de este sistema de gestión hospitalaria representa una solución integral, escalable y mantenible, construida sobre una arquitectura moderna de tres capas con separación clara entre el frontend, backend y base de datos.

En el backend, la utilización de Spring Boot 3.3.9 con Java 17, junto con una estructura modular basada en vertical slicing, permitió una organización eficiente del código y una alta cohesión por dominio funcional. La implementación de controladores REST, servicios desacoplados, especificaciones y puertos facilita la extensión futura del sistema, mientras que la persistencia de datos se resuelve de forma robusta mediante MariaDB 11.11.10.

El frontend, desarrollado con Nuxt 3, TypeScript y tecnologías como Tailwind CSS, Pinia, Zod y Vitest, ofrece una interfaz moderna, responsiva y validada.

La integración de un proceso de CI/CD mediante Jenkins, con pruebas automatizadas para frontend y backend, garantiza la calidad continua del software. El despliegue en una máquina virtual de GCP utilizando Nginx como reverse proxy y Ngrok como túnel persistente permite exponer los servicios de forma segura y controlada, sin necesidad de configuraciones complejas de red.