Sistema de monitoreo NMS-Udenar

MANUAL DE DESARROLLADOR

ÍNDICE

INTRODUCCIÓN

NMS-Udenar es un sistema de monitoreo desarrollado para brindar soporte a las labores de gestión de la red de datos presente en la Universidad de Nariño que realiza el personal administrativo de la oficina de redes y telecomunicaciones del Aula de Informática de la universidad.

Este manual de desarrollador está enfocado a aquellos usuarios que requieran el conocimiento de cómo se realizan las consultas y entregas de información desde los dispositivos de red por medio del protocolo SNMP hasta el servidor del aplicativo, para este motivo la población que tenga la intención de revisar el código fuente debe tener un conocimiento básico en los siguientes temas:

- Uso de sistemas operativos para servidores basados en Linux Ubuntu.
- Modelo Cliente/Servidor de aplicaciones web.
- Arquitectura API de aplicativos webs.
- Arquitectura MVC (Modelo-Vista-Controlador).
- Desarrollo web básico (HTML, CSS, JavaScript de lado cliente).
- JavaScript de lado servidor.
- Bases de datos no relacionales.
- Motores de plantillas HTML.
- Bootstrap framework.
- Arquitectura y funcionamiento del protocolo SNMP v2c.
- Control de versiones.

Este sistema de monitoreo está desarrollado haciendo uso de las siguientes tecnologías:

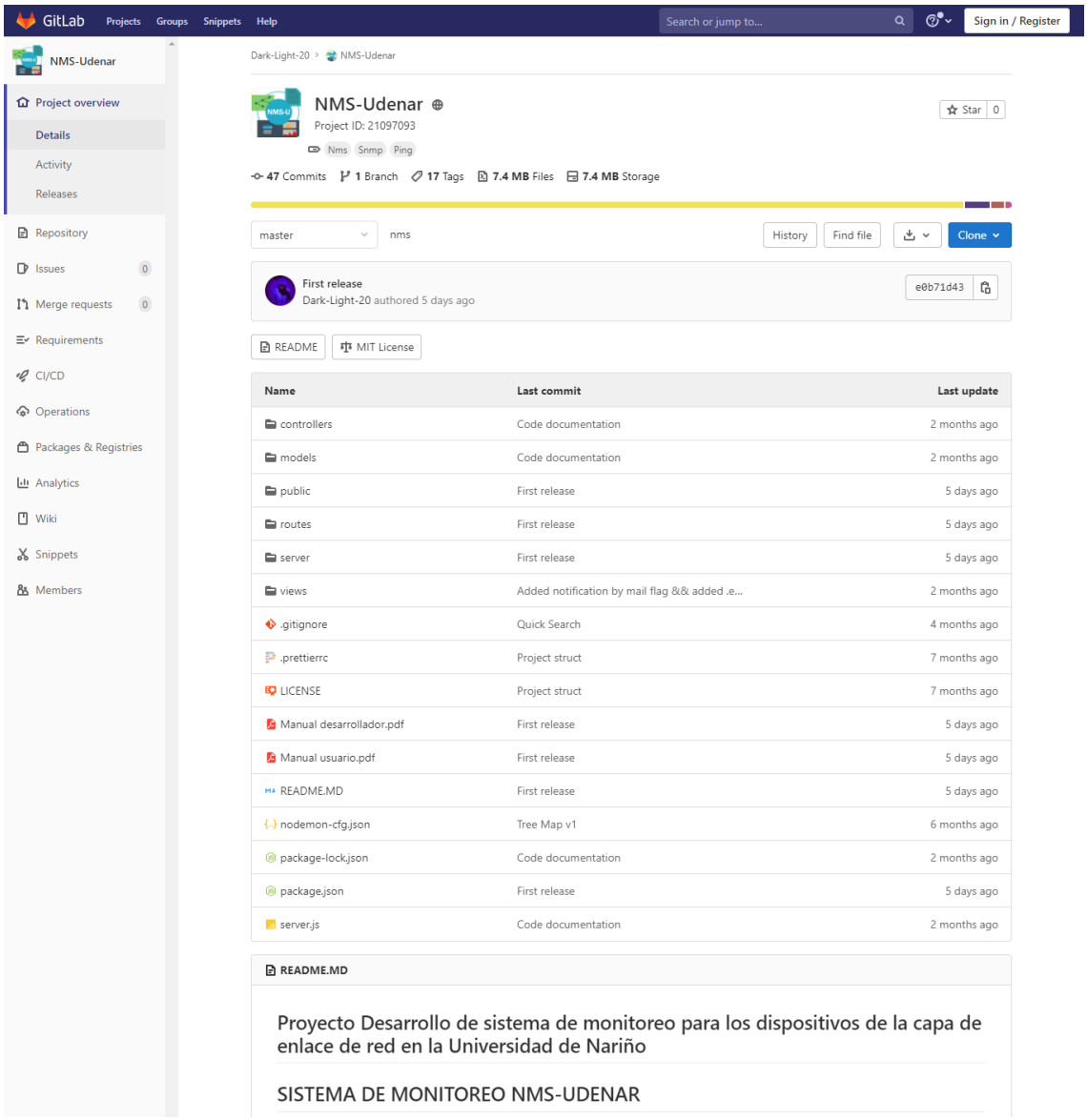
- NodeJs como motor de ejecución del lenguaje JavaScript de lado servidor.
- ExpressJs como componente framework de desarrollo de aplicaciones web.
- MongoDB como motor de base de datos no relacionales.
- Pug como motor de plantillas HTML para renderizar desde lado servidor.
- net-snmp como modulo que permite la implementación del protocolo SNMP en aplicaciones node.
- Git como software para control de versiones.
- GitLab como repositorio remoto del proyecto.
- PM2 como gestor de despliegue de aplicativos node.

También se hace uso de varios módulos node para simplificar procesos y ofrecer las funcionalidades específicas del sistema de monitoreo, estos módulos están especificados en el archivo package.json del proyecto y su respectiva documentación puede ser consultada en la página https://www.npmjs.com/.

REPOSITORIO DEL PROYECTO

El proyecto del sistema de monitoreo NMS-Udenar se encuentra alojado en GitLab, plataforma de colaboración y control de versiones en la web. Dentro de este repositorio se puede revisar la evolución del proyecto, evidenciar comentarios, reportar errores dentro del funcionamiento o código del sistema y poder ramificar más repositorios para así poder complementar y evolucionar este proyecto.

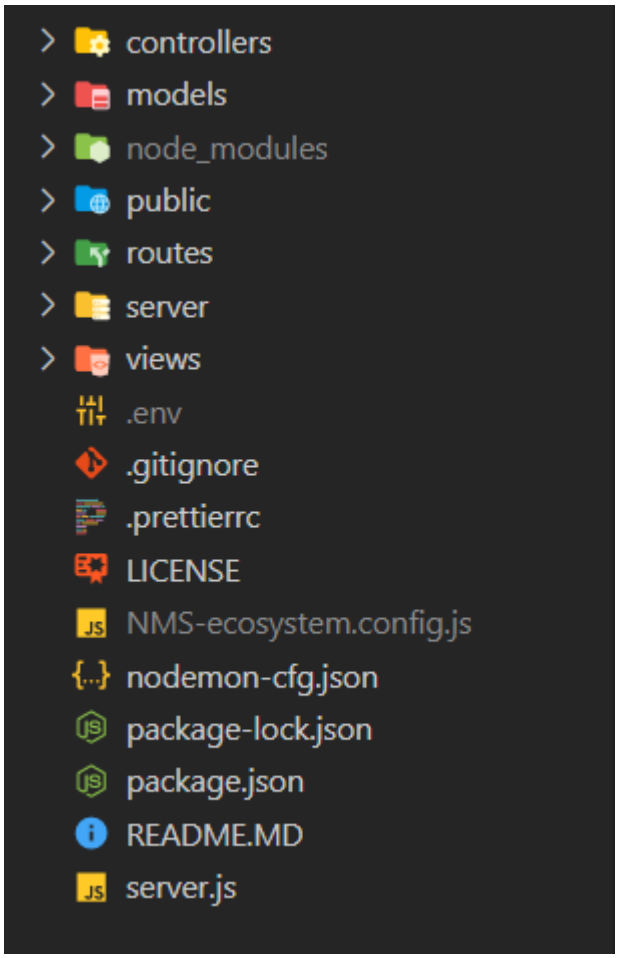
Enlace del repositorio GitLab: https://gitlab.com/Dark-Light-20/nms



ESTRUCTURA DEL PROYECTO

El proyecto del sistema de monitoreo SNMP está construido bajo una adaptación de la arquitectura MVC. Esto debido a que los proyectos de NodeJs no son restrictivos a seguir una estructura como tal o restricción a nivel de directorios o archivos que deben ser manejados.

La estructura del proyecto es la siguiente:



De la anterior imagen se destaca lo siguiente:

- Directorio “controllers”: Dentro de este directorio se encuentran los archivos que contienen el procesamiento de información y lógica de negocios del aplicativo web. Estos controladores son los encargados de recibir las

solicitudes del cliente web y renderizar la vista o respuesta que se le debe retornar a su consulta.

- Directorio “models”: Dentro de este directorio se encuentran los archivos que contienen la estructura de los objetos (clases) que son manejados por le proyecto, la representación a nivel de sistema de las entidades que son tratadas dentro del aplicativo, por ejemplo, los dispositivos de red o los usuarios.

- Directorio “public”: Dentro de este directorio se encuentran todos los archivos estáticos que son enviados al cliente web para el correcto funcionamiento del aplicativo y las vistas de las páginas que el usuario final ve, por ejemplo, estilos CSS o códigos JavaScript de lado cliente.

- Directorio “routes”: Dentro de este directorio se encuentran los archivos que contienen los redireccionamientos de las consultas que realiza el cliente web, son los encargados de vincular las solicitudes con los métodos que procesarán las solicitudes.

- Directorio “server”: Dentro de este directorio se encuentran los archivos que contienen funciones propias del sistema de monitoreo que sirven para brindar las funcionalidades especificas de los usuarios, por ejemplo, el servicio (Daemon) encargado de realizar copias de seguridad históricas o el método que realiza la conexión con la base de datos.

- Directorio “views”: Dentro de este directorio se encuentran todas las plantillas que renderizan las vistas entregadas al cliente web de acuerdo a cada solicitud que realiza, esto con el fin de reutilizar y simplificar los archivos entregados al cliente.

- Archivo “.env”: Este archivo es donde están declaradas las variables internas del sistema (variables de entorno) que son usadas en los procesos más importantes del proyecto, por ejemplo, las credenciales de acceso a la base de datos o la cadena de comunidad que usará el sistema para las consultas del protocolo SNMP.

- Archivo “package.json”: Este archivo contiene información acerca del proyecto, contacto, scripts y los módulos utilizados tanto en etapa de desarrollo como en etapa de producción.

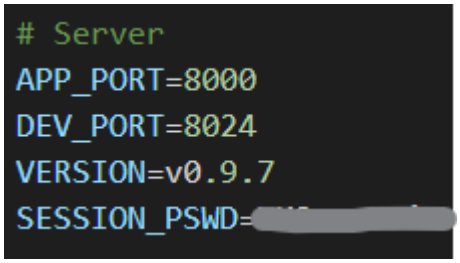
- Archivo “server.js”: Este archivo es el punto de partida del aplicativo web, dentro de él se encuentran las declaraciones e importaciones de todos los demás archivos del proyecto, es el que se encarga de servir la aplicación web con la configuración requerida.

VARIABLES DE ENTORNO

Las variables de entorno son necesarias para la configuración y funcionamiento del aplicativo web, al momento de descargar o clonar el proyecto desde el repositorio GitLab el archivo ".env" no será creado, esto se hace con el fin de proteger datos sensibles de quienes hagan uso del sistema, por lo cual se debe crear un archivo en la raíz del proyecto con el nombre exacto ".env".

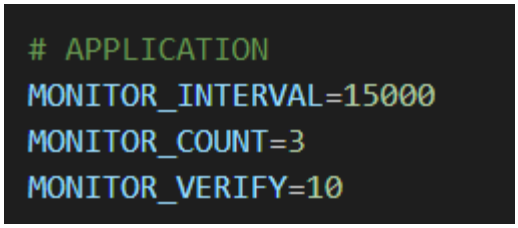
Las variables que deben ser declaras obligatoriamente son las siguientes:

- Variables del aplicativo:



  Estas variables son usadas al momento de configurar el aplicativo para servirlo.
  - APP_PORT: Puerto por el cual el aplicativo web será servido en producción.
  - DEV_PORT: Puerto por el cual el aplicativo será servido en desarrollo.
  - VERSION: Versión del aplicativo.
  - SESSION_PSWD: Cadena usada para configurar el modulo de manejo de variables de sesión en el aplicativo web.

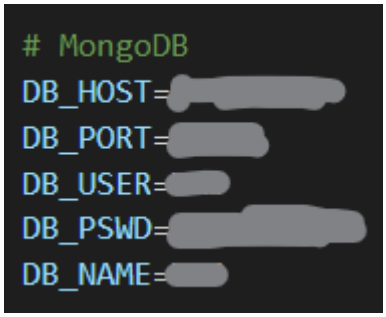
- Tiempos de monitoreo:



  Estas variables representan los tiempos que se usarán para monitorear el sistema.
  - MONITOR_INTERVAL: Esta variable representa los milisegundos en los que el sistema realizará las verificaciones de los enlaces a los dispositivos registrados. En este caso cada 15 segundos se realizará la consulta de todos los dispositivos.

- MONITOR_COUNT: Esta variable representa la cantidad de paquetes ICMP que son enviados por medio del módulo “ping” implementado en el aplicativo para verificar el enlace a los dispositivos registrados en el sistema.
- MONITOR_VERIFY: Esta variable representa la cantidad de paquetes ICMP que son enviados para verificar un enlace en caso de que el primer intento sea fallido. Con esto se puede tener mayor confianza en que un enlace realmente está desconectado y así reportarlo.

- Credenciales de base de datos:

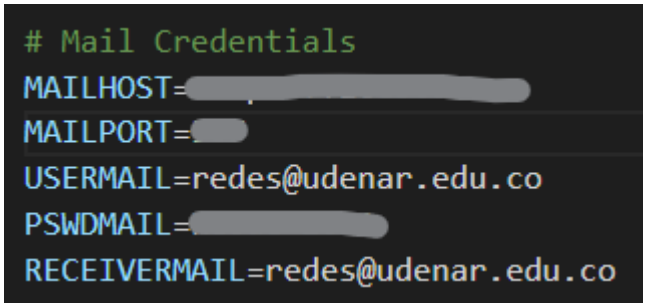


Estas variables son usadas para configurar la conexión a la base de datos desde el aplicativo web.

- Cadena de comunidad SNMP:

Esta variable contiene la cadena de comunidad que usará el sistema para hacer todas las consultas por medio del protocolo SNMP, se debe asegurar que este valor sea el que se vaya a configurar en cada dispositivo de red y la versión del protocolo sea la **v2c**.

- Credenciales de correo electrónico:



Estas variables representan la configuración que debe tener el módulo de “nodemailer” para realizar la funcionalidad de notificación por medio de correo electrónico.

- MAILHOST – MAILPORT: Son variables usadas para configurar el servidor de correo al cual se hará la conexión.
- USERMAIL – PSWDMAIL: Son variables que representan las credenciales del correo desde el cuál se enviará las notificaciones del sistema.
- RECEIVERMAIL: Es la dirección de correo electrónico a donde llegaran las notificaciones.

ARCHIVO PRINCIPAL DEL SISTEMA

Este archivo (“server.js”) contiene toda la configuración inicial que realiza el motor de NodeJs al momento de servir el aplicativo, sirve como punto inicial para servir el aplicativo. A continuación, se dará un detalle de cada sección de este archivo:

- Importación de módulos:

```
// Imports
const express = require('express');
const path = require('path');
const multer = require('multer');
const favicon = require('serve-favicon');
const session = require('express-session');
require('dotenv').config({ path: path.join(__dirname, '.env') });
```

  En este apartado se declaran las variables que contienen referencia a los módulos principales para servir el aplicativo además de configurar las variables de entorno que se recuperan desde el archivo “.env”.

  En todos los archivos JavaScript que pertenezcan al servidor se encontrará con esta primera sección.

- Declaración de aplicativo y configuración inicial:

```
// App
const app = express();

// Multer
const upload = multer();

// Open DB connection      You, 4 months ago • Aut
const connection = require('./server/connectDB');
connection();
```

  Con estas líneas se instancia lo siguiente:
  - una variable que contendrá al aplicativo web que se va a servir,
  - Una variable que permite el uso de formularios web y la recepción de datos a través de estos.
  - La conexión a la base de datos (*el proceso se detallará más adelante*).

- Configuración del aplicativo web y utilidades:

```
// Load utilities
app.use(express.json());
app.use(express.urlencoded({ extended: true }));
app.use(express.static(path.join(__dirname, 'public')));
app.use(favicon(path.join(__dirname, 'public', 'favicon.ico')));
app.use(
  session({
    secret: process.env.SESSION_PSWD,
    resave: false,
    saveUninitialized: false
  })
);

// For non-file multipart forms
app.use(upload.none());

// Views engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'pug');
```

En este apartado se configuran los middlewares que hará uso el aplicativo web como manejo de formato JSON, servir archivos estáticos, uso de variables de sesión, uso de motor de plantillas HTML, etc.

- Carga inicial de variables de dispositivos:

```
// Get Server IPs
const getServerIPs = require('./server/serverIPs');
app.set('serverIPs', getServerIPs());

// Load Switches
const getDB_SWs = require('./server/getDB_SWs');
const monitorD = require('./server/monitor');
const backupsDaemon = require('./server/backups');
async function loadSWs() {
  try {
    // Retrieve Switches from database
    const SWs = await getDB_SWs();
    app.set('SWs', SWs);
    // Switches monitor daemon
    setInterval(() => monitorD(app.get('SWs')), process.env.MONITOR_INTERVAL);
    // Switches backups daemon
    backupsDaemon(app);
  } catch (err) {
    console.error(err);
  }
}
loadSWs();
```

En este apartado primero se consultan las direcciones IPv4 que tenga el servidor en sus interfaces, esto para excluirlas en algunas funcionalidades finales. Después se obtienen todos los dispositivos registrados en la base de datos del sistema y se instancian en variables que manejará el aplicativo web, además de inicializar los procesos en segundo plano como el servicio de monitoreo de enlaces y el servicio de copias de seguridad.

- Rutas del aplicativo:

```
// Import Routes
const statusRouter = require('./routes/status');
const appRouter = require('./routes/app');
const userRouter = require('./routes/user');
const reportsRouter = require('./routes/reports');
//const tryRouter = require('./routes/try'); // For dev

// Use Routes
app.use('/status', statusRouter);
app.use('/', appRouter);
app.use('/', userRouter);
app.use('/reports', reportsRouter);
//app.use('/try', tryRouter); // For dev
```

En este apartado se importan las rutas y se las incorpora al aplicativo web para su uso.

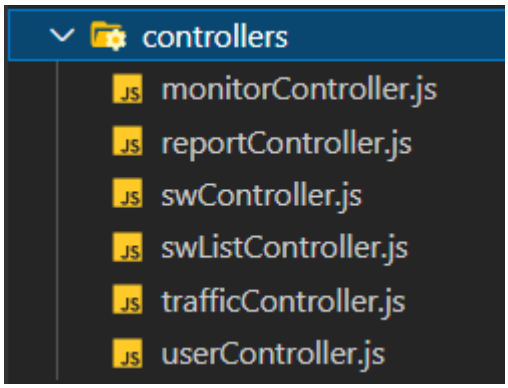
- Servir el aplicativo:

```
// Start application
app.listen(process.env.APP_PORT, () => {
  console.log('Running NMS on port:', process.env.APP_PORT);
});
```

Finalmente se sirve el aplicativo web por el puerto especificado en las variables de entorno.

CONTROLADORES

El proyecto del sistema de monitoreo posee los siguientes controladores que son los encargados de recibir una solicitud de un cliente web a través de alguna ruta del aplicativo y se encargar de retornar una respuesta:



- monitorController: Con este controlador se retorna el listado de dispositivos del sistema, dentro de los cuales se encuentran los datos de sus respectivos enlaces para monitoreo, esta respuesta es usada por ejemplo para el mapa de red y actualizar los nodos.

- reportController: En este controlador está desarrollada la lógica de retornar los diferentes reportes que están registrados en la base de datos del sistema para que el usuario pueda visualizarlos.

- swController: En este controlador está desarrollada toda la lógica de gestión de los dispositivos del sistema además de tener implementada la lógica de uso de las consultas por medio del protocolo SNMP. Este es de los archivos más importantes del sistema ya que es en él donde reside la mayor funcionalidad y por lo tanto el más complejo de todos.

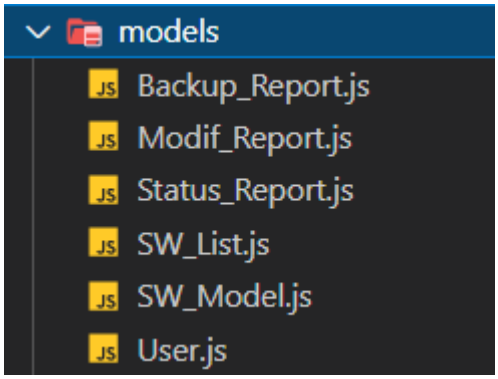
- swListController: En este controlador está desarrollada la lógica de retorno del listado de los dispositivos registrados en el sistema para la tabla principal del módulo “Listado de Switches”.

- trafficController: En este controlador está desarrollada la lógica de consulta por medio del protocolo SNMP de los consumos de cada interfaz de los dispositivos.

- userController: En este controlador está desarrollada la gestión de usuarios incluyendo la autenticación de los mismos en el uso del aplicativo web.

MODELOS

El proyecto del sistema de monitoreo posee los siguientes modelos que son los encargados de representar a las diferentes entidades que participan en el aplicativo web. Estas entidades son objetos JavaScript que son creados a partir del módulo “mongoose” que es el encargado de la gestión de los registros y manipulación de colecciones de la base de datos MongoDB del sistema:



- Backup_Report: Contiene los datos que posee una copia de seguridad de un dispositivo, tanto la configuración del mismo como la fecha del reporte.

- Modif_Report: Contiene los datos que posee un reporte de una modificación realizada a un dispositivo registrado en el sistema, el usuario que la realizó, la fecha del reporte y el tipo de modificación.

- Status_Report: Contiene los datos de un registro en el cambio del enlace de un dispositivo de red detectado por el sistema de monitoreo.

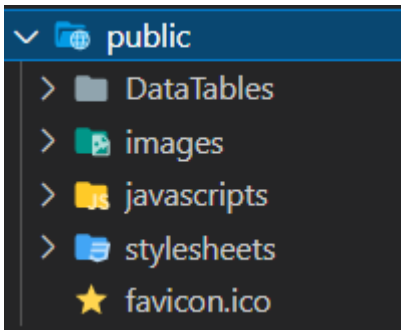
- SW_List: Contiene los datos de un dispositivo del sistema de monitoreo para la población del listado en el módulo de “Listado de Switches”, son recuperados de la colección de todos los dispositivos registrados en el sistema.

- SW_Model: Contiene los datos de los diferentes modelos que son soportados por el sistema de monitoreo, estos datos son visibles en el apartado de información básica al realizar una consulta de un dispositivo en el sistema.

- User: Contiene los datos de cada usuario del sistema.

ARCHIVOS ESTÁTICOS PÚBLICOS

El aplicativo web tiene especificada la carpeta “public” en donde residen todos los archivos públicos que son usados para la generación de páginas web y la funcionalidad que ofrecen.



- DataTables: En este directorio se encuentran todos los archivos necesarios para el funcionamiento de las tablas dinámicas que se encuentran en el aplicativo web. Para más información sobre DataTables se puede consultar su documentación oficial en: https://datatables.net/.

- images: En este directorio se encuentran todas las imágenes que usa el aplicativo web.

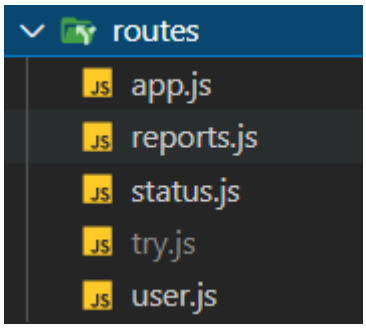
- javascripts: En este directorio se encuentran todos los archivos JavaScript que son usados en las páginas web del aplicativo para brindar la funcionalidad y dinamismo.

- stylesheets: En este directorio se encuentran todos los archivos de estilos de las páginas web del aplicativo.

- favicon.ico: Icono del aplicativo web.

RUTAS

Las direcciones web URL disponibles en el aplicativo web están separadas por varios archivos JavaScript para brindar un mejor orden y separación de responsabilidades:



La estructura de estos archivos es la siguiente:

```
// Express Router
const router = require('express').Router();
// Controllers
const swListController = require('../controllers/swListController');
const swController = require('../controllers/swController');
// Authentication middleware
const isAuthenticated = require('../controllers/userController')
  .isAuthenticated;
```

Primero se encuentra la importación de controladores y middlewares que serán empleados en las rutas.

```
router.get('/map', isAuthenticated, (req, res) =>
  res.render('map', {
    title: 'Network-Map',
    navbarText: 'Mapa de Red',
    user: { name: req.session.user, type: req.session.type }
  })
);
router.get('/switchList', isAuthenticated, swListController.getList);
router
  .route('/switch/:IP')
  .all(isAuthenticated)
  .get(swController.getSwitch)
  .post(swController.addSwitch)
  .patch(swController.editSwitch)
  .delete(swController.deleteSwitch);
```

Luego se encuentra la declaración de las rutas y su comportamiento dependiendo del verbo usado y la funcionalidad que debe proveer, además de qué vista debe renderizar de acuerdo a las variables que se le proveen.

- app.js: En este archivo están declaradas las rutas de todas las páginas principales del aplicativo web.

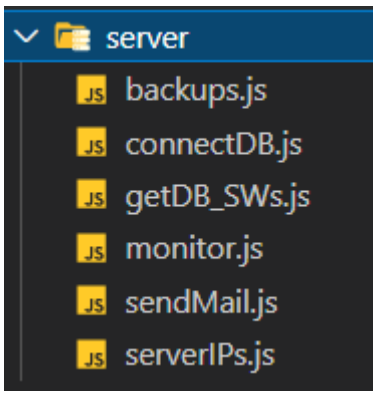
- reports.js: En este archivo están declaradas las rutas de las diferentes páginas de reportes del aplicativo web.

- status.js: En este archivo están declaradas las rutas del componente API del aplicativo web, el usuario común no ingresará directamente a estas rutas, son usadas por las mismas páginas web para consultar o actualizar información, además de brindar la posibilidad de servir de punto de datos hacia otras plataformas o aplicativos webs si se desea.

- user.js: En este archivo están declaradas las rutas para la gestión de usuarios del aplicativo web.

FUNCIONES DEL SERVIDOR

Dentro de esta sección se encuentran las diversas funcionalidades de monitoreo y notificaciones que hace uso el aplicativo web:



- backups.js: En este archivo está desarrollada la lógica del servicio (Daemon) de copias de seguridad históricas, la creación de cada registro y su almacenamiento a la base de datos.

- connectDB.js: En este archivo está desarrollada la conexión a la base de datos de MongoDB del aplicativo web.

- getDB_SWs.js: En este archivo está desarrollada la lógica de recuperación de todos los dispositivos registrados en la base de datos del aplicativo web y la creación y retorno de estos mismos en forma de variables de sistema para que el aplicativo pueda hacer uso de estos datos y modificarlos según convenga el caso.

- monitor.js: En este archivo está desarrollada la lógica del servicio de monitoreo continuo a cada uno de los dispositivos registrados en el sistema y en caso de detectar un enlace desconectado se encarga de generar el reporte y realizar la notificación por correo.

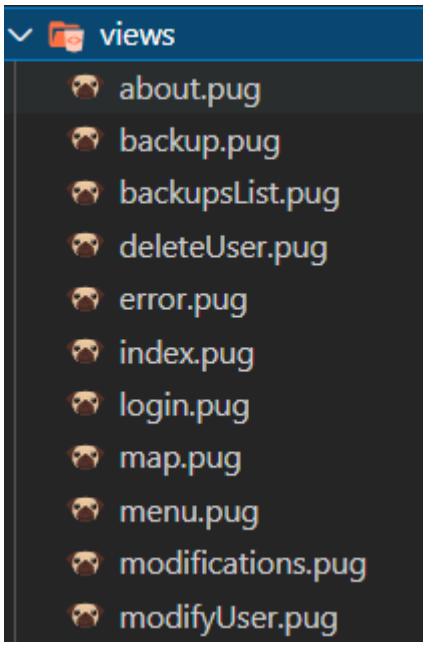
- sendMail.js: En este archivo está desarrollada la lógica del envío de correos electrónicos para notificar del cambio de enlace de un dispositivo.

- serverIPs.js: En este archivo está desarrollada la funcionalidad de recuperar las direcciones IPv4 que posee el servidor que sirve el aplicativo.

VISTAS

El aplicativo web hace uso del motor de plantillas Pug para la generación de las vistas hacia el cliente web, para mayor información de cómo funciona este motor de plantillas se puede consultar su documentación oficial en: https://pugjs.org.

El uso de un motor de plantillas permite la reutilización de secciones, uso de variables, mejor claridad en la definición de componentes web y eficiencia a la hora de escribir código HTML




```
extends template

append links
    script(src="https://kit.fontawesome.com/1742c5ad30.js", crossorigin="anon

block container
    .container.text-center
        .row.mt-3
            .col
                img(src="/favicon.ico", alt="nms-logo", widtg="250")
        .row.mt-1
            .col
                h3 NMS-Udenar
                h4 "Sistema de monitoreo para dispositivos de la capa de enla
                h5
                    i= version
```

IMPLEMENTACIÓN DEL PROTOCOLO SNMP

El uso del protocolo SNMP para este sistema de monitoreo se limita a la funcionalidad de consulta de información de los dispositivos de red, por lo cual todo se remite al uso equivalente en comandos de terminal de snmpget o snmpwalk.

Con el uso de un aplicativo web y del módulo de “net-snmp” se simplifica mucho y se mejora la experiencia de usuario a la hora de obtener la información de una forma más amigable y entendible. Normalmente hacer esto desde un servidor y con líneas de comandos sería de la siguiente manera:

```
darklight@NMS-Server:~$ snmpwalk -v2c -c [redacted] 10.16.16.24 1.3.6.1.2.1.1
iso.3.6.1.2.1.1.1.0 = STRING: "SG350X-48P 48-Port Gigabit PoE Stackable Managed Switch"
iso.3.6.1.2.1.1.2.0 = OID: iso.3.6.1.4.1.9.6.1.94.48.5
iso.3.6.1.2.1.1.3.0 = Timeticks: (1401537800) 162 days, 5:09:38.00
iso.3.6.1.2.1.1.4.0 = STRING: "redes"
iso.3.6.1.2.1.1.5.0 = STRING: "BIBLIO-P1-CORE"
iso.3.6.1.2.1.1.6.0 = STRING: "udenar"
iso.3.6.1.2.1.1.7.0 = INTEGER: 6
iso.3.6.1.2.1.1.8.0 = Timeticks: (0) 0:00:00.00
iso.3.6.1.2.1.1.9.1.2.1 = OID: iso.3.6.1.4.1.89.73
iso.3.6.1.2.1.1.9.1.3.1 = STRING: "RS capabilities"
iso.3.6.1.2.1.1.9.1.4.1 = Timeticks: (0) 0:00:00.00
```

Lo cual se hace con la siguiente sintaxis:

1. Comando snmp a usar (*Requiere del paquete snmp*): En este caso se hace un recorrido de todas las MIBs y OIDs existentes dentro del árbol jerárquico con **snmpwalk**.
2. Versión snmp a usar: Para este sistema de monitoreo se hace uso de la versión **-v2c**.
3. Cadena de comunidad: La cadena de comunidad configurada en el dispositivo **-c <cadena>**.
4. Dirección IP del dispositivo.
5. OID a consultar: En este caso se hace la consulta a la OID **1.3.6.1.2.1.1** que refiere a la MIB de system.

Este proceso automatizado en el sistema se visualiza de la siguiente forma:

```
function getSwInfo(session) {       You, 6 months ago • MVC model
  return new Promise((resolve, reject) => {
    // OIDs for Basic SW Info => ['ObjectID', 'UpTime', 'Contact', 'Name', 'Location']
    const OIDs = [
      '1.3.6.1.2.1.1.2.0',
      '1.3.6.1.2.1.1.3.0',
      '1.3.6.1.2.1.1.4.0',
      '1.3.6.1.2.1.1.5.0',
      '1.3.6.1.2.1.1.6.0'
    ];
```

Se declaran las OIDs a consultar, en este caso son del apartado de información básica.

```
const INFO = {};
session.get(OIDs, (error, results) => {
  if (error) {
    error.section = 'INFO';
    reject(error);
  } else {
    // ObjectID
    const OID = results[0];
    if (snmp.isVarbindError(OID)) console.error(snmp.varbindError(OID));
    else INFO.oid = OID.value.toString();
    // UpTime
    const UPTIME = results[1];
    if (snmp.isVarbindError(UPTIME))
      console.error(snmp.varbindError(UPTIME));
    else INFO.uptime = prettyms(UPTIME.value * 10, { verbose: true }); // i
    // Contact
    const CONTACT = results[2];
    if (snmp.isVarbindError(CONTACT))
      console.error(snmp.varbindError(CONTACT));
    else INFO.contact = CONTACT.value.toString();
    // Name
    const NAME = results[3];
    if (snmp.isVarbindError(NAME)) console.error(snmp.varbindError(NAME));
    else INFO.name = NAME.value.toString();
    // Location
    const LOCATION = results[4];
    if (snmp.isVarbindError(LOCATION))
      console.error(snmp.varbindError(LOCATION));
    else INFO.location = LOCATION.value.toString();
```

Seguido de eso se realiza una consulta directa a cada OID para retomar su valor y almacenarlo en una variable INFO la cual se asociará con la variable que representa a todo el dispositivo.

El proceso de consulta se empieza a complicar cuando la información requerida no se obtiene con una OID estándar para todos los modelos, por ejemplo, el listado de puertos existentes:

```
function getSwPorts(session, SW_OID) {
  return new Promise((resolve, reject) => {
    let maxRep = 2;
    let OID = '1.3.6.1.2.1.2.2.1.8';
    let ciscoSG = false; // For the break in SGXXX models
    let portTypeOID;
    const PORTS = {};
    switch (SW_OID) {
      // Catalyst Switches
      case '1.3.6.1.4.1.9.1.1745':
      case '1.3.6.1.4.1.9.1.1208':
        maxRep = 1;
        OID = '1.3.6.1.4.1.9.9.46.1.6.1.1.2';
        portTypeOID = '1.3.6.1.4.1.9.9.46.1.6.1.1.13';
        break;
      // Cisco Switches
      case '1.3.6.1.4.1.9.6.1.94.48.5':
      case '1.3.6.1.4.1.9.6.1.94.48.1':
      case '1.3.6.1.4.1.9.6.1.91.24.8':
      case '1.3.6.1.4.1.9.6.1.81.28.2':
      case '1.3.6.1.4.1.9.6.1.85.48.1':
        maxRep = 20;
        portTypeOID = '1.3.6.1.4.1.9.6.1.101.48.22.1.1';
        ciscoSG = true;
        break;
      // let maxRep = 2 // for HP v1910 ***
    }
```

En esta función se aprecia como se evalúa el valor de SW_OID que contiene la OID del modelo del dispositivo y de acuerdo a este valor se asignan ciertos valores a las variables usadas por la consulta SNMP.

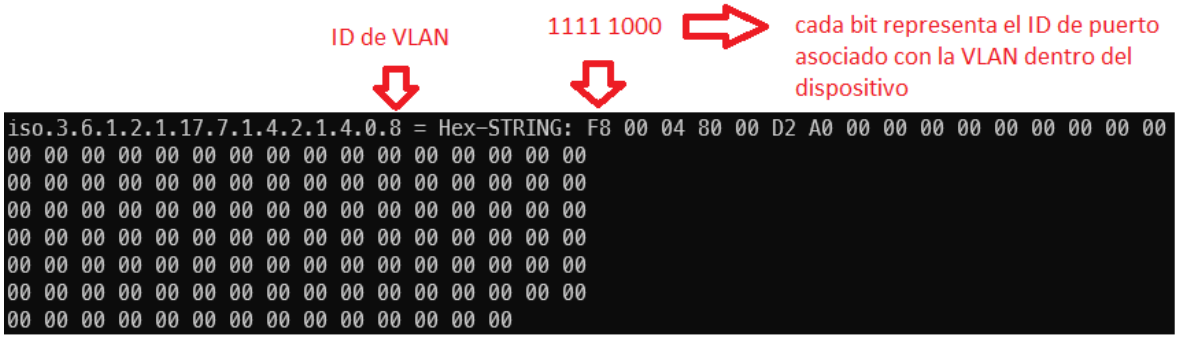
```
session.subtree(
  OID,
  maxRep,
  (results) => {
    for (const item of results) {
      if (snmp.isVarbindError(item))
        console.error('PORT Error: ', snmp.varbindError(item));
      else {
        const SPLIT = item.oid.split('.');
        const PORT = SPLIT[SPLIT.length - 1]; // Last number in O
        if (ciscoSG) {
          if (PORT.length > 3) break; // Port Index never past ov
          if (item.value === 6) continue; // Not present interfac
        }
        PORTS[PORT] = {};
        // OIDs for port info => [ Port Descp (name), Port Descp
        let OIDs = [
          `1.3.6.1.2.1.2.2.1.2.${PORT}`,
          `1.3.6.1.2.1.31.1.1.1.18.${PORT}`,
          `1.3.6.1.2.1.31.1.1.1.15.${PORT}`
        ];
        if (ciscoSG) PORTS[PORT].status = item.value;
        else OIDs.push(`1.3.6.1.2.1.2.2.1.8.${PORT}`);
        if (portTypeOID) OIDs.push(`${portTypeOID}.${PORT}`);
```

Dentro del código se encuentran comentarios a algunas líneas para explicar el porqué de ciertos procesos.

Principalmente las OIDs básicas son más simples de entender debido a que son cadenas de texto o números específicos, pero la cuestión se complica a medida que se requiere información más detallada, por ejemplo, las VLANs que están asociadas a los puertos de un dispositivo:

```
darklight@NMS-Server:~$ snmpwalk -v2c -c [redacted] 10.16.16.24 1.3.6.1.2.1.17.7.1.4.2.1.4.0
iso.3.6.1.2.1.17.7.1.4.2.1.4.0.1 = Hex-STRING: 03 C0 06 7F 03 FE FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 01 FE
iso.3.6.1.2.1.17.7.1.4.2.1.4.0.8 = Hex-STRING: F8 00 04 80 00 D2 A0 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

En este caso se está consultando las VLANs asociadas a los puertos de un dispositivo de modelo SG350X-48P. El resultado de la consulta son cadenas hexadecimales por cada puerto, la cadena hexadecimal representa en trasfondo qué ID de VLAN está asociada al puerto:



En este caso la VLAN de ID 8 está asignada a los puertos con ID 1, 2, 3, 4 y 5 en primer lugar. Estos ID de interfaz corresponden a los puertos 1, 2, 3, 4 y 5 respectivamente.

Este proceso para este modelo de Switch se debe realizar por cada tramo de la cadena hexadecimal. Pero esto no es común para todos los modelos soportados,

por ejemplo, para los modelos de la serie Catalyst de Cisco el tratamiento de la consulta de este dato es diferente y más complejo, donde se debe realizar una consulta para los puertos de tipo acceso y otra de tipo troncal y donde la lógica se cambia de sentido, primero se detecta el ID de la interfaz y la cadena hexadecimal representa los IDs de las VLANs asociadas a ese puerto de tipo troncal.

Este proceso ya está automatizado por los métodos del controlador swController:

```
function loadPortsVLANs(session, PORTS, SW_OID) {      You, 6 months
  return new Promise(async (resolve, reject) => {
    if (!PORTS) reject({ section: 'VLANS PORTS' });
    let OID;
    switch (SW_OID) {
      // Catalyst Switches
      case '1.3.6.1.4.1.9.1.1208':
      case '1.3.6.1.4.1.9.1.1745':
        try {
          for (let i = 0; i < 4; i++) {
            // Trunk ports - sending VLAN group
            await loadTrunkVlansCatalyst(session, PORTS, i);
            // Access ports
            resolve(await loadAccessVlansCatalyst(session, PORTS));
          }
        } catch (err) {
          reject(err);
        }
        return;
      // Cisco Switches
      case '1.3.6.1.4.1.9.6.1.94.48.5':
      case '1.3.6.1.4.1.9.6.1.94.48.1':
      case '1.3.6.1.4.1.9.6.1.91.24.8':
      case '1.3.6.1.4.1.9.6.1.81.28.2':
      case '1.3.6.1.4.1.9.6.1.85.48.1':
        // OID for Current VLAN EgressPorts
        OID = '1.3.6.1.2.1.17.7.1.4.2.1.4.0';
        break;
      default:
        // OID for Static VLAN EgressPorts
        OID = '1.3.6.1.2.1.17.7.1.4.3.1.2';
        break;
    }
```

Este método tiene como parámetros la sesión SNMP creada al momento de la consulta del dispositivo, los puertos del dispositivo y el OID del modelo del dispositivo para saber que proceso ejecutar. Dentro de la función primero se verifica

que la variable PORTS tenga asignado un valor, luego de esto se verifica de que modelo de dispositivo se está tratando para saber que OID consultar, esto se hace debido a que entre modelos las OIDs cambian y algunas no existen.

```
session.subtree(
  OID,
  (results) => {
    for (const item of results) {
      if (snmp.isVarbindError(item))
        console.error('VLANS PORT Error: ', snmp.varbindError(item));
      else {
        const split = item.oid.split('.');
        const vlan = split[split.length - 1]; // Last number in OID is VLAN ID
        const HEX = item.value.toString('hex');
        let binary = '';
        let vlanPorts = [];
        for (const i of HEX) binary += hex2bin(i);
        for (let i = 0; i < binary.length; i++)
          if (binary.charAt(i) === '1') vlanPorts.push(i + 1);
        for (const port of vlanPorts)
          if (PORTS[port] !== undefined) PORTS[port].VLANs.push(vlan);
      }
    }
  },
```

Luego haciendo uso de la sesión SNMP que es una variable instanciada del modulo “net-snmp” se hace un recorrido al árbol de la OID seleccionada, se trata tanto las OIDs existentes como sus valores en cadenas hexadecimales, se recorren los bits 1 y se transforma su posición a un valor de ID de interfaz.

Nota: Este proceso es para los dispositivos de modelos SG. Para los modelos Catalyst existen otras funciones que realizan el proceso.

Para los Catalyst primero se asignan las VLANs a los puertos de tipo troncal:

```
function loadTrunkVlansCatalyst(session, PORTS, loop) {
  // OID for vlanTrunkPortVlansEnabled (k,2k,3k,4k)
  const OIDs = [
    '1.3.6.1.4.1.9.9.46.1.6.1.1.4',
    '1.3.6.1.4.1.9.9.46.1.6.1.1.17',
    '1.3.6.1.4.1.9.9.46.1.6.1.1.18',
    '1.3.6.1.4.1.9.9.46.1.6.1.1.19'
  ];
  // Default SNMP VLANs values for NO VLAN assigned
  const badHEXs = [
    'ffffffffffffffffffffffffffffffffffffffffffffffffffffffff
    'ffffffffffffffffffffffffffffffffffffffffffffffffffffffff
    '7fffffffffffffffffffffffffffffffffffffffffffffffffffffff
    ''
  ];
```

Se establecen las 4 OIDs que representan las relaciones de una interfaz con las VLANs entre rangos de 1024, es decir, la primera OID contiene los resultados de las VLANs 1 a 1023, la segunda de las VLANs 1024 a 2047, y así sucesivamente.

```
session.subtree(
  OIDs[loop],
  (results) => {
    for (const item of results) {
      if (snmp.isVarbindError(item))
        console.error('TRUNK VLAN Error: ', snmp.varbindError(item));
      else {
        const split = item.oid.split('.');
        const port = split[split.length - 1]; // Last number in OID is port ID
        const HEX = item.value.toString('hex');
        if (!badHEXs.includes(HEX)) {
          let binary = '';
          let vlansPort = [];
          for (const i of HEX) binary += hex2bin(i);
          for (let i = 0; i < binary.length; i++)
            if (binary.charAt(i) === '1')
              vlansPort.push((i + 1024 * loop).toString());
          for (const vlan of vlansPort)
            if (PORTS[port] !== undefined) PORTS[port].VLANs.push(vlan);
        }
      }
    }
  },
```

Luego se realiza el proceso de análisis de cada cadena hexadecimal para detectar los bits que son 1 los cuales representan la posición de la VLAN que si está asignada

al puerto, también se validan aquellas cadenas hexadecimales que no aportan información (badHEXs).

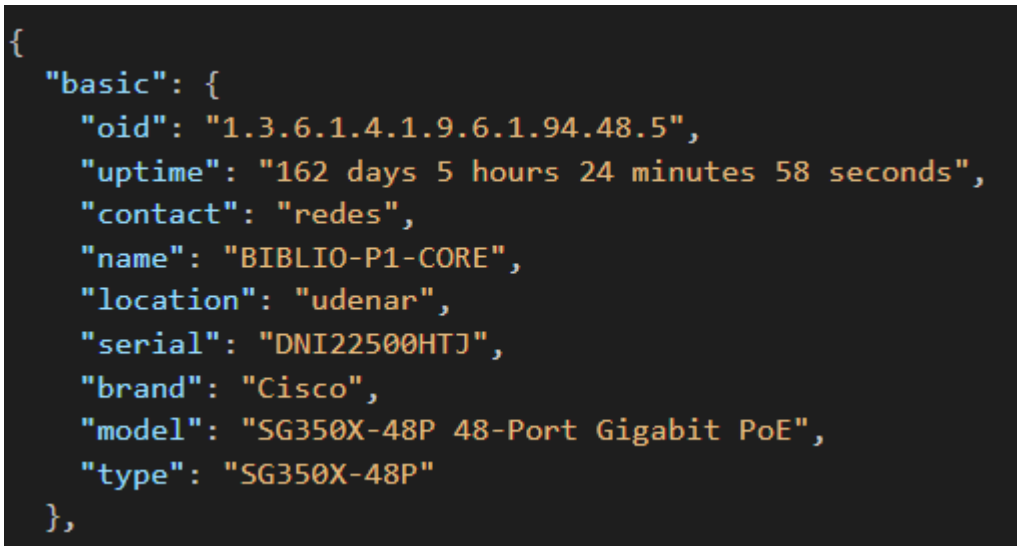
Una vez hecho este proceso se asignan las VLANs a los puertos de tipo acceso:

```
const OID = '1.3.6.1.4.1.9.9.68.1.2.2.1.2';
return new Promise((resolve, reject) => {
  session.subtree(
    OID,
    (results) => {
      for (const item of results) {
        if (snmp.isVarbindError(item))
          console.error('ACCESS VLAN Error: ', snmp.varbindError(item));
        else {
          const split = item.oid.split('.');
          const port = split[split.length - 1]; // Last number in OID is port ID
          const vlan = item.value.toString();
          if (!(item.value in PORTS[port].VLANs))
            PORTS[port].VLANs.push(vlan);
        }
      }
    },
```

Donde es más sencillo ya que la consulta de la OID retorna un valor numérico que representa la VLAN asociada.

Una vez comprendido el funcionamiento de una consulta SNMP y los resultados que retorna, los tipos de variable que retorna, lo que queda finalmente es comprender la lógica de cada método del controlador, cada método cumple una consulta especifica de cada apartado para el retorno completo de la variable del dispositivo. Los métodos están comentados y ordenados para que el usuario que revise el código pueda comprenderlos con poco tiempo de dedicación y análisis.

La variable que representa un dispositivo posee la siguiente estructura:



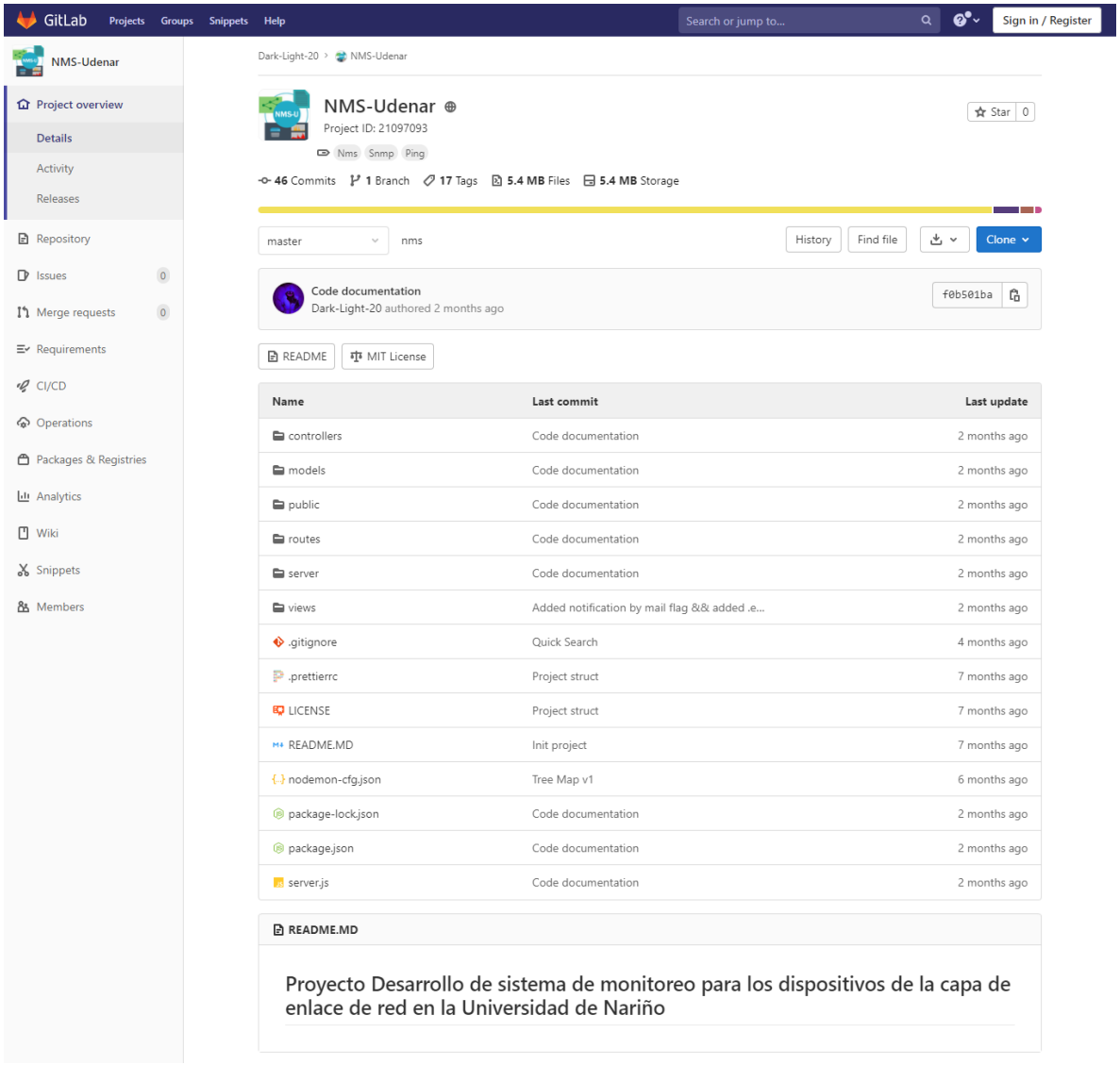
Un primer apartado que compone la información básica.

Un segundo apartado que posee el listado de VLANs registradas en el equipo.

```
{
  "status": 1,
  "name": "GigabitEthernet1/0/46",
  "alias": "TRONCAL-SW-COMITE-MATRICULAS",
  "speed": 1000,
  "type": 12,
  "VLANs": [
    "1",
    "16",
    "64",
    "1100"
  ],
  "id": "46"
},
{
  "status": 1,
  "name": "GigabitEthernet1/0/47",
  "alias": "TRONCAL-SW-BIBLIOTECA-P2",
  "speed": 1000,
  "type": 12,
  "VLANs": [
    "1",
    "8",
    "16",
    "64",
    "66"
  ],
  "id": "47"
},
```


Un tercer apartado que posee el listado de los puertos con su configuración.

Esta variable es renderizada en la vista de consulta para generar la página web final que es visible por el usuario.



[Enlace del repositorio NMS-Udenar]