

# Estructura del api rest Dory

## Objetivo:

Mostrar cómo es la estructura del api rest de la plataforma web Dory.

## Procedimiento:

Explicar cada uno de los componentes por los que se encuentra conformada el api rest de la plataforma web Dory.

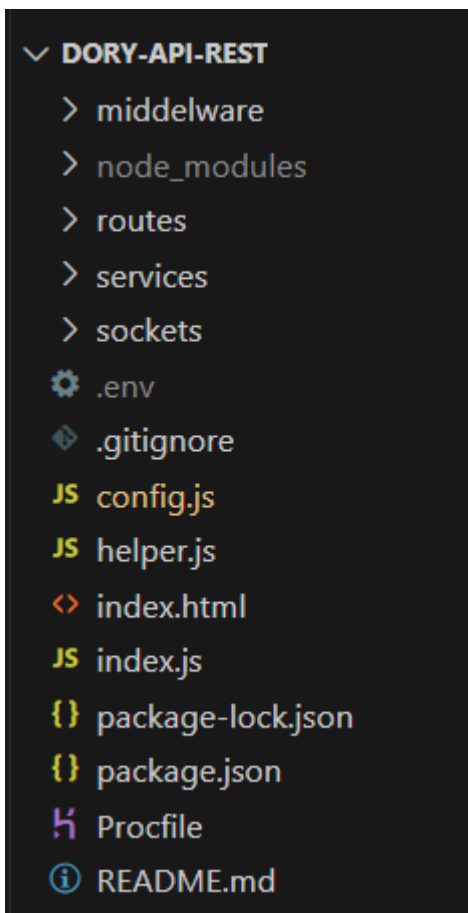


Imagen 1. Estructura visual del api rest Dory en visual studio code

## Recursos necesarios:

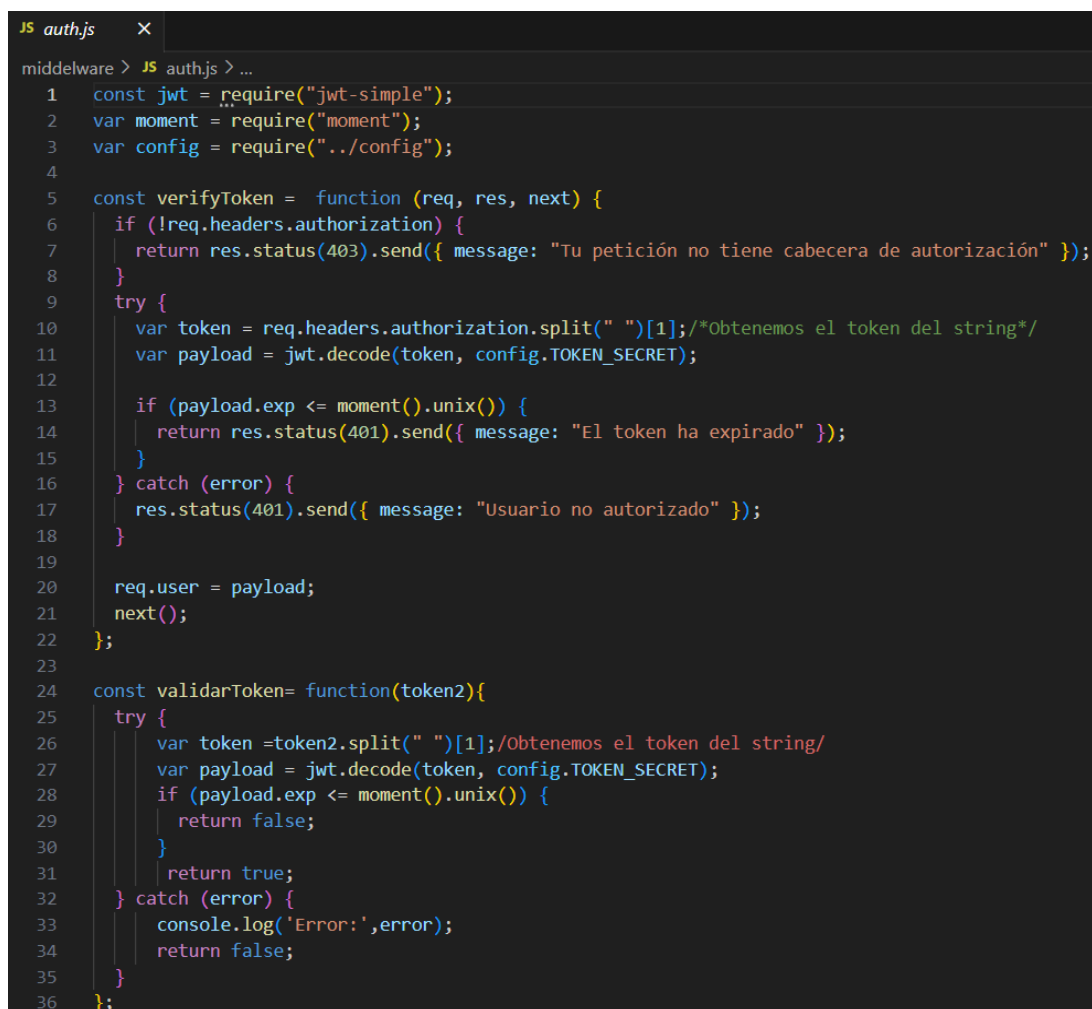
Es necesario una imagen visual del api rest de la plataforma web Dory.

## Pasos:

Tomando como referencia la imagen visual de la estructura del api rest de la plataforma web Dory, se realizará una descripción de cada uno de los componentes que la integran (ver imagen 1).

### 1. Descripción del componente middleware.

Middleware es el componente que se implementó para verificar la autenticación del usuario que realiza la solicitud en función de sus credenciales y está compuesto por el módulo **"auth.js"**, en el cual se valida y verifica el token del usuario (ver imagen 2).



```
JS auth.js x
middleware > JS auth.js > ...
1  const jwt = require("jwt-simple");
2  var moment = require("moment");
3  var config = require("../config");
4
5  const verifyToken = function (req, res, next) {
6    if (!req.headers.authorization) {
7      return res.status(403).send({ message: "Tu petición no tiene cabecera de autorización" });
8    }
9    try {
10     var token = req.headers.authorization.split(" ")[1]; /*Obtenemos el token del string*/
11     var payload = jwt.decode(token, config.TOKEN_SECRET);
12
13     if (payload.exp <= moment().unix()) {
14       return res.status(401).send({ message: "El token ha expirado" });
15     }
16   } catch (error) {
17     res.status(401).send({ message: "Usuario no autorizado" });
18   }
19
20   req.user = payload;
21   next();
22 };
23
24 const validarToken= function(token2){
25   try {
26     var token = token2.split(" ")[1]; /*Obtenemos el token del string*/
27     var payload = jwt.decode(token, config.TOKEN_SECRET);
28     if (payload.exp <= moment().unix()) {
29       return false;
30     }
31     return true;
32   } catch (error) {
33     console.log('Error:',error);
34     return false;
35   }
36 };
```

Imagen 2. Módulo auth.js

## 2. Descripción del componente routes.

Routes es el componente que se implementó para las rutas de los endpoints. Está compuesto por una clasificación de módulos JavaScript que referencia a los diferentes endpoints (ver imagen 3), cada uno de estos módulos tienen enrutamientos a los diferentes servicios del api rest Dory (ver imagen 4).

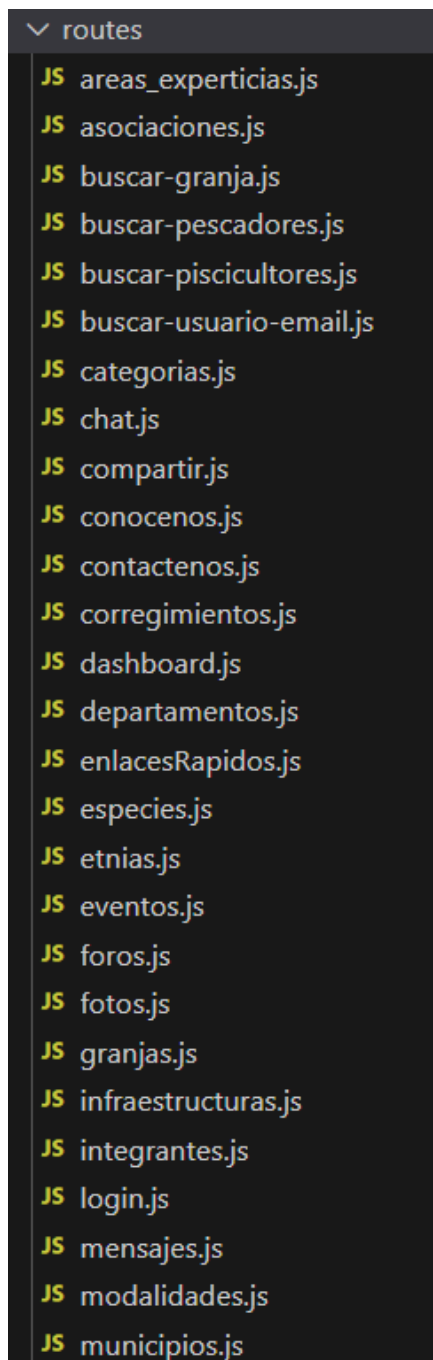


Imagen 3. Módulos del componente routes

```

JS areas_experticias.js X
routes > JS areas_experticias.js > router.post('/registrar') callback
1  const express = require('express');
2  const router = express.Router();
3  const areas_experticias = require('../services/areas_experticias');
4
5  router.get('/obtener', async function(req, res, next) {
6    try {
7      res.json(await areas_experticias.getAreasExperticia(req.query.page));
8    } catch (err) {
9      console.error(`Error al traer las areas_experticias`, err.message);
10     next(err);
11   }
12 });
13
14 router.post('/registrar', async function(req, res, next) {
15   try {
16     res.json(await areas_experticias.createAreaExperticia(req.body));
17   } catch (err) {
18     console.error(`Error creando area de experticia`, err.message);
19     next(err);
20   }
21 });

```

Imagen 4. Descripción del Módulo *areas\_experticias.js* del componente *routes*

### 3. Descripción del componente *services*.

Services es el componente que se implementó para los servicios del api rest Dory. Está compuesto por una clasificación de módulos JavaScript que referencia a diferentes servicios (ver imagen 5), cada uno de estos módulos referencia a uno o varios procesos de la plataforma web Dory (ver imagen 6).

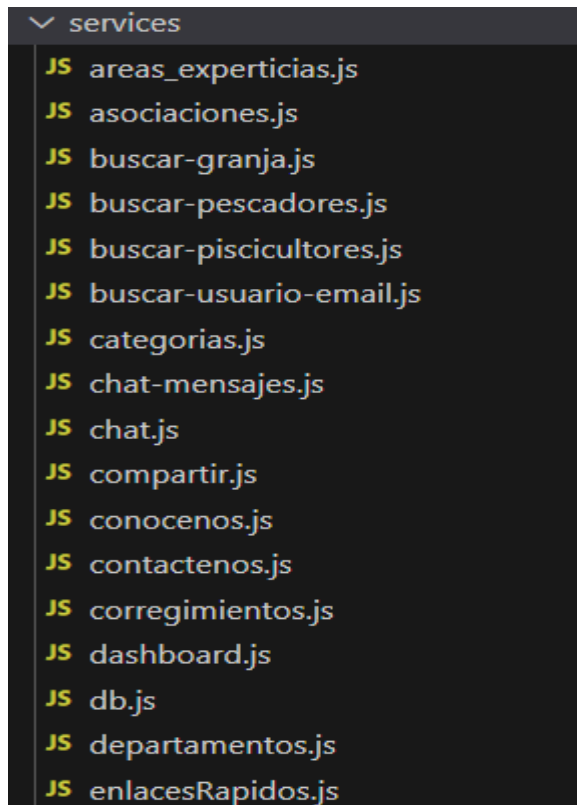


Imagen 5. Módulos *del componente services*

```
const db = require('./db');
const helper = require('../helper');
const config = require('../config');
var createError = require('http-errors');

/* _____ getAreaExperticia _____ */
async function getAreasExperticia(page = 1){
  const offset = helper.getOffset(page, config.listPerPage);
  const rows = await db.query(
    `SELECT * FROM areas_experticias LIMIT ?,?`,
    [offset, config.listPerPage]
  );
  const data = helper.emptyOrRows(rows);
  const meta = {page};
  return {
    data,
    meta
  }
}/*End getAreaExperticia*/
```

Imagen 6. Descripción del módulo *areas\_experticias.js del componente services*

#### 4. Descripción del componente sockets.

Sockets es el componente que se implementó para los servicios del chat. Está compuesto por el módulo "**controller.js**" que referencia a diferentes procesos del chat de la plataforma web Dory (ver imagen 7).



Imagen 7. Módulo controller.js del componente sockets

#### 5. Descripción del módulo "gitignore".

Gitignore es el módulo utilizado en el control de versiones Git para especificar que archivos y directorios se deben ignorar y no deben ser incluidos en el repositorio Git, ya que pueden ser irrelevantes, causar conflictos o violar la seguridad y privacidad (ver imagen 8).

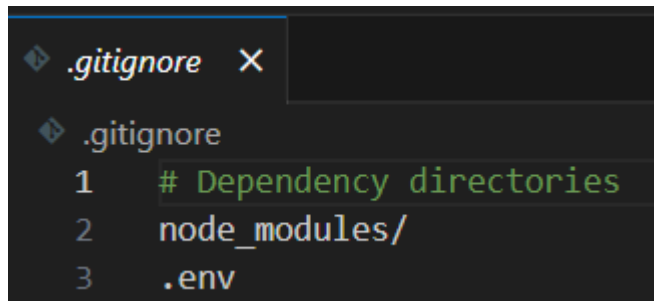


Imagen 8. Módulo gitignore

#### 6. Descripción del módulo "env".

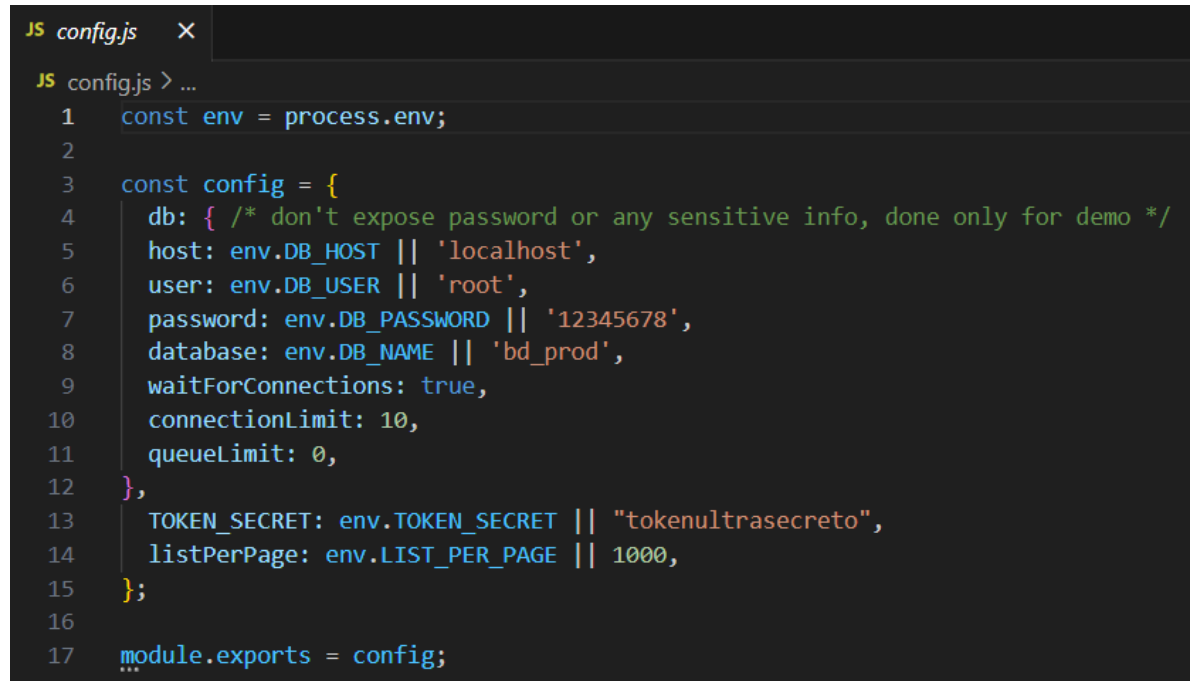
Env es el módulo de configuración utilizado para almacenar las variables de entorno. Su propósito es mantener configuraciones sensible o específica del entorno fuera del código fuente del proyecto (ver imagen 9).



Imagen 9. Archivo "env"

7. Descripción del módulo "config.js".

Config.js es el módulo de configuración utilizado para conectar el api rest con la base de datos (ver imagen 10).



```
JS config.js X
JS config.js > ...
1  const env = process.env;
2
3  const config = {
4    db: { /* don't expose password or any sensitive info, done only for demo */
5      host: env.DB_HOST || 'localhost',
6      user: env.DB_USER || 'root',
7      password: env.DB_PASSWORD || '12345678',
8      database: env.DB_NAME || 'bd_prod',
9      waitForConnections: true,
10     connectionLimit: 10,
11     queueLimit: 0,
12   },
13   TOKEN_SECRET: env.TOKEN_SECRET || "tokenultrasecreto",
14   listPerPage: env.LIST_PER_PAGE || 1000,
15 };
16
17 module.exports = config;
```

Imagen 10. Módulo "config"

8. Descripción del módulo "helper.js".

Helper.js es el módulo de utilidades auxiliares que brindan funcionalidades comunes y reutilizables para el api rest (ver imagen 11).

```

JS helper.js X
JS helper.js > emptyOrRows
1  const jwt = require("jwt-simple");
2  var moment = require("moment");
3  const config = require('./config');
4  const nodemailer = require('nodemailer');
5
6  function getOffset(currentPage = 1, listPerPage) {
7      return (currentPage - 1) * [listPerPage];
8  }
9
10 function emptyOrRows(rows) {
11     if (!rows) {
12         return [];
13     }
14     return rows;
15 }
16
17 function isProductionEnv(){
18     return process.env.NODE_ENV == 'production';
19 }
20
21 function createToken (user,minutes) {
22     var payload = {
23         email:user.email,
24         sub: user.id,
25         rol: user.nombre_tipo_usuario,
26         iat: moment().unix(),
27         exp: moment().add(minutes, "minutes").unix(),
28     };
29     return jwt.encode(payload, config.TOKEN_SECRET);
30 };
31
32 function parseJwt(token) {
33     var base64Payload = token.split('.')[1];
34     var payload = Buffer.from(base64Payload, 'base64');
35     return JSON.parse(payload.toString());
36 };

```

Imagen 11. Módulo "helper.js"



9. Descripción del archivo "index.html".

Index.html es la página predeterminada para mostrar información del api rest de la plataforma web Dory, éste archivo es opcional más no necesario (ver imagen 12).

## **Bienvenido a Dory Api Rest**

Con esta api puedes consultar información del sector psicología de Sucre

Test

Imagen 12. *Index.html* desplegado

10. Descripción del módulo "index.js".

Index.js es el módulo de entrada principal de la aplicación, se ejecuta cuando se inicia el servidor del api rest. A través de este módulo, se configuran y se inician los componentes esenciales del servidor, como la configuración del servidor web, la definición de rutas, la configuración de middleware y se maneja la lógica fundamental para procesar solicitudes y proporcionar respuestas adecuadas. Es una parte fundamental en la estructura y el funcionamiento del api rest (ver imagen 13).

```

JS index.js X
JS index.js > ...
1  const express = require('express');
2  const app = express();
3  const bodyParser = require('body-parser');
4  const port = process.env.PORT || 3000;
5  const {verifyToken, validarToken} = require ('./middleware/auth');
6  require('dotenv').config();
7
8  /*Socket.io con express*/
9  const http = require('http');
10 const server = http.createServer(app);
11 const { Server } = require("socket.io");
12 const io = new Server(server,{
13   cors: {
14     origin: "*"
15   }
16 });
17
18 const { socketController } = require('./sockets/controller')
19 const departamentosRouter = require('./routes/departamentos');
20 const tipos_usuariosRouter = require('./routes/tipos_usuarios');
21 const infraestructurasRouter = require('./routes/infraestructuras');
22 const areas_experticiasRouter = require('./routes/areas_experticias');
23 const especiesRouter = require('./routes/especies');
24 const tipos_eventosRouter = require('./routes/tipos_eventos');
25 const modalidadesRouter = require('./routes/modalidades');
26 const categoriasRouter = require('./routes/categorias');
27 const proveedoresRouter = require('./routes/proveedores');
28 const tipos_asociacionesRouter = require('./routes/tipos_asociaciones');
29 const municipiosRouter = require('./routes/municipios');
30 const corregimientosRouter = require('./routes/corregimientos');
31 const veredasRouter = require('./routes/veredas');
32 const asociacionesRouter = require('./routes/asociaciones');
33 const granjasRouter = require('./routes/granjas');
34 const subregionesRouter = require('./routes/subregiones');
35 const proyectosRouter = require('./routes/proyectos');
36 const sectoresRouter = require('./routes/sectores');
37 const tipos_normatividadesRouter = require('./routes/tipos_normatividades');

```

Imagen 13. Módulo *Indexs.js*

## 11. Descripción del archivo "package-lock.json".

Package-lock.json es el archivo de bloqueo de versiones generado automáticamente por npm, cuando instalas o actualizas paquetes en un proyecto de node.js. Su principal propósito es garantizar la consistencia y la reproducibilidad de las dependencias del proyecto, lo que ayuda que todos

los miembros del equipo tengan las mismas versiones de las bibliotecas instaladas y que los entornos de desarrollo y producción se comporten de manera coherente (ver imagen 14).

```
{ } package-lock.json > ...
1  {
2    "name": "api-rest-piscicola",
3    "version": "1.0.0",
4    "lockfileVersion": 2,
5    "requires": true,
6    "packages": {
7      "": {
8        "name": "api-rest-piscicola",
9        "version": "1.0.0",
10       "license": "ISC",
11       "dependencies": {
12         "bcrypt": "^5.0.1",
13         "cors": "^2.8.5",
14         "dayjs": "^1.11.3",
15         "dotenv": "^16.0.3",
16         "express": "^4.17.2",
17         "google-auth-library": "^7.14.1",
18         "http-errors": "^2.0.0",
19         "jwt-simple": "^0.5.6",
20         "moment": "^2.29.1",
21         "mysql2": "^2.2.5",
22         "nodemailer": "^6.7.2",
23         "socket.io": "^4.5.2"
24       },
25       "engines": {
26         "node": "14.17.3"
27       }
28     },
```

Imagen 14. Archivo *package-lock.json*

## 12. Descripción del archivo "package.json".

Package.json es un archivo de configuración fundamental en proyectos de node.js que se genera automáticamente al iniciar el proyecto con el comando `npm init`. Este archivo proporciona información crítica sobre dependencias, versiones, script personalizados e información del proyecto (ver imagen 15).

```
{ } package.json > ...
1  {
2    "name": "api-rest-piscicola",
3    "version": "1.0.0",
4    "description": "A REST API with Node.js Express.js and MySQL for Piscícola app",
5    "main": "index.js",
6    "scripts": {
7      "start": "node index.js",
8      "test": "echo \"Error: no test specified\" && exit 1"
9    },
10   "author": "Luis Alberto Pontón",
11   "license": "ISC",
12   "dependencies": {
13     "bcrypt": "^5.0.1",
14     "cors": "^2.8.5",
15     "dayjs": "^1.11.3",
16     "dotenv": "^16.0.3",
17     "express": "^4.17.2",
18     "google-auth-library": "^7.14.1",
19     "http-errors": "^2.0.0",
20     "jwt-simple": "^0.5.6",
21     "moment": "^2.29.1",
22     "mysql2": "^2.2.5",
23     "nodemailer": "^6.7.2",
24     "socket.io": "^4.5.2",
25     "swagger-ui-express": "^5.0.0"
26   },
27   "engines": {
28     "node": "14.17.3"
29   }
30 }
```

Imagen 15. Archivo *package.json*