Es un

LINAJE

SAAWIT: API que permite gestionar noticias colaborativas, donde los usuarios puedan registrarse y publicar una noticia en una serie de categorías temáticas fijas. Segundo proyecto realizado para Hackaboss usando tecnología javascript, Node.js, MySQL y Postman.

Proyecto realizado por: Juan Garrido Troche.

Contenido

[LINEAGE 2](#_Toc135077458)

[Título 2](#_Toc135077459)

[Descripción 2](#_Toc135077460)

[Usuarios anónimos 2](#_Toc135077461)

[Usuarios registrados 2](#_Toc135077462)

[Usuarios Admin registrados 3](#_Toc135077463)

[BASE DE DATOS 3](#_Toc135077464)

[ENTIDADES 4](#_Toc135077465)

[RELACIONES 5](#_Toc135077466)

[CREACIÓN DE TABLAS EN MYSQL WORKBENCH 5](#_Toc135077467)

[INTRODUCCIÓN 8](#_Toc135077468)

[DEPENDENCIAS NECESARIAS PARA SAAWIT 9](#_Toc135077469)

[RUTAS / ENDPOINTS 11](#_Toc135077470)

# LINEAGE

En este proyecto desarrollaremos una web utilizando de las siguientes tecnologías: JavaScript, Node.js, MySQL y REACT. Ya que es un proyecto de gestión de documentos por medio de perfiles o roles de usuario hemos investigado acerca de las tecnologías más adecuadas para este fin. Hemos denominado al proyecto “Lineage” ya que los usuarios que se logueen, y después de que el administrador conceda permisos de distintas familias de linaje, pueda acceder a diferente documentación.

## Título

Lineage: gestión de documentos según tu apellido.

## Descripción

En Lineage implementaremos una API que permita gestionar documentos, donde los usuarios puedan registrarse y descargar un pdf según los permisos que le otorgues a ese usuario.

## Usuarios anónimos

Acciones que pueden realizar los usuarios no registrados o anónimos:

* Pueden acceder a la página principal de la web.
* Visualizar noticias.
* Filtrado por:
  + apellido
* Login.
* Registro:
  + Nombre
  + apellidos
  + email
  + password

## Usuarios registrados

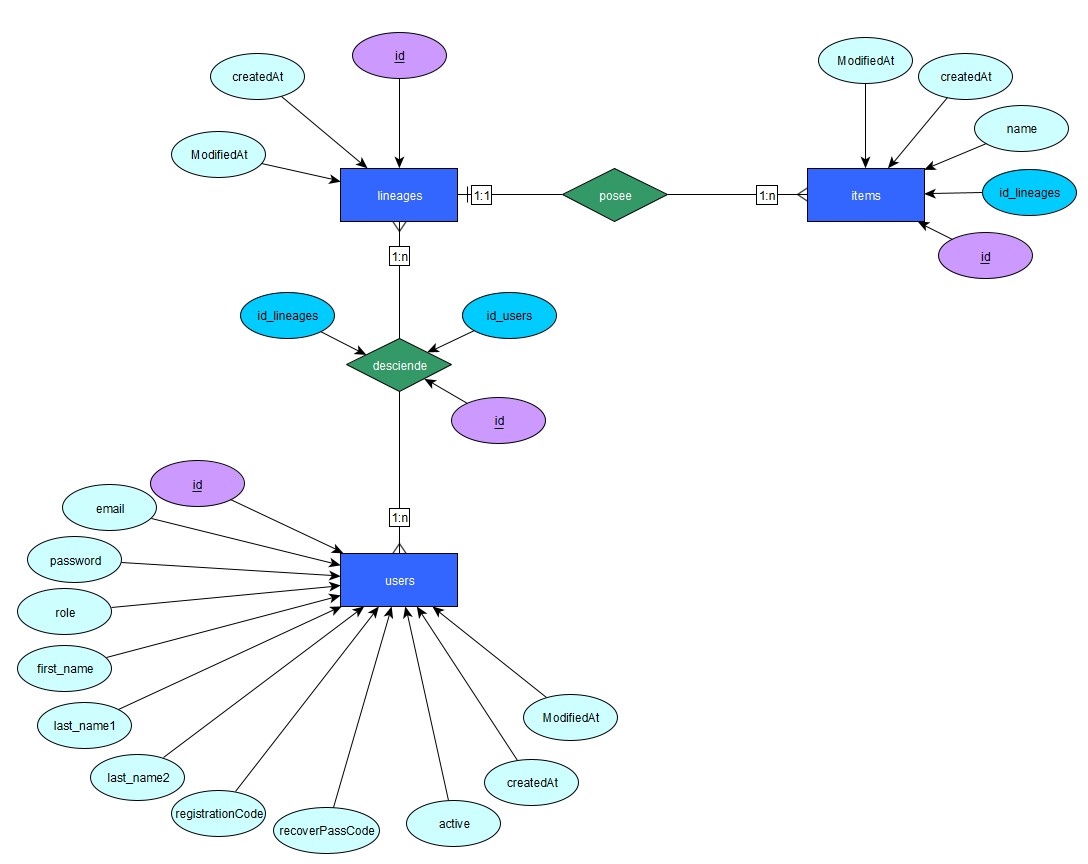
* Las mismas acciones que los usuarios anónimos
* Descargar documentación perteneciente a tu familia:
  + PDF

## Usuarios Admin registrados

* Las mismas acciones que los usuarios anónimos
* Las mismas acciones que los usuarios registrados
* Asignar a cada usuario registrado los permisos para aquellos apellidos o linaje a los que pertenece.
* Modificar roles de usuario (user y admin). No se puede eliminar un admin si éste es el único que existe en la Base de datos.
* Puede subir documentos.
* CRUD de linajes: nombre, escudo de armas, descripción

# BASE DE DATOS

Para realizar la base de datos hemos utilizado la herramienta yEd Graph Editor, donde hemos creado las entidades con los siguientes atributos:



## ENTIDADES

Users:

* Id: Clave primaria, Identificador único, autoincremental y entero de cada registro.
* email: Tipo texto, VARCHAR de un máximo de 100 caracteres, requerido y obligatorio.
* password: Tipo texto, VARCHAR de un máximo de 100 caracteres, requerido y obligatorio.
* role: ENUM('admin', 'mod', 'user') DEFAULT 'user', role: Solo las opciones de: admin, mod(puede modificar permisos de usuario pero no roles) y user (por defecto).
* first\_name: Tipo texto, VARCHAR de un máximo de 100 caracteres, requerido y obligatorio.
* last\_name1: Tipo texto, VARCHAR de un máximo de 100 caracteres, requerido y obligatorio.
* last\_name2: Tipo texto, VARCHAR de un máximo de 100 caracteres, requerido y obligatorio.
* recoverPassCode: Código de recuperación de contraseña: VARCHAR (20)
* registrationCode: Código de recuperación de contraseña: VARCHAR (20)
* active: El usuario puede darse de alta pero hasta que no introduce el código de registro no se activa su cuenta. Por defecto vendrá como false. BOOLEAN.
* createdAt: Fecha de creación del usuario. Timestamp y requerido.
* modifiedAt: Fecha de modificación de alguno de los atributos. Timestamp.

lineages:

* id: Clave primaria, Identificador único, autoincremental, solo puede ser un número entero y positivo.
* lineage: Apellido del linaje, obligatorio, VARCHAR(100).
* description: Descripción del linaje, VARCHAR(900).
* Shield: nombre de la imagen del escudo de armas. VARCHAR(100).
* createdAt: Fecha de creación del usuario. Timestamp y requerido.
* modifiedAt: Fecha de modificación de alguno de los atributos. Timestamp.

items:

* id: Clave primaria, Identificador único, autoincremental y entero de cada registro.
* Id\_lineages: Clave foránea de la tabla lineage y número entero.
* name: Nombre del documento subido de manera automática, obligatorio, VARCHAR(100)
* createdAt: Fecha de creación del usuario. Timestamp y requerido.
* modifiedAt: Fecha de modificación de alguno de los atributos. Timestamp.

## RELACIONES

descends: Un usuario desciende de 1 a n linajes (1:n) y 1 linaje puede tener descendencia entre 1 a n usuarios (1:n). Como existen n usuarios y linajes (n:m) esta relación se convierte en una tabla incluyendo los atributos de id de ambas entidades (users y news):

* Id: Clave primaria, Identificador único, autoincremental y entero de cada registro.
* Id\_user: Clave foránea de la tabla user, obligatorio y número entero.
* Id\_lineage: Clave foránea de la tabla news, obligatorio y número entero.
* createdAt: Fecha de creación del usuario, obligatorio y Timestamp.

have: Un linaje tiene de 1 a n items (1:n) y 1 item solo puede pertenecer a un linaje (1:1). Por lo tanto, el id de usuario debe añadirse a la tabla de items. No es necesario crear una tabla para esta relación.

## CREACIÓN DE TABLAS EN MYSQL WORKBENCH

Las tablas se crearán dentro de la base de datos llamada saawit que, a su vez, se incluirá dentro del proyecto dentro del archivo bbdd/initDB.js:

* + - CREATE DATABASE IF NOT EXISTS lineage;
    - USE lineage;

Se eliminan las tablas de forma inversa a como se han creado

* + - DROP TABLE IF EXISTS items;
    - DROP TABLE IF EXISTS lineages;
    - DROP TABLE IF EXISTS descends;
    - DROP TABLE IF EXISTS users;
    - CREATE TABLE IF NOT EXISTS users (

id INT UNSIGNED PRIMARY KEY AUTO\_INCREMENT,

email VARCHAR(100) UNIQUE NOT NULL,

password VARCHAR(100) NOT NULL,

role VARCHAR(500) NOT NULL,

photo VARCHAR(100),

role ENUM('admin', 'mod', 'user') DEFAULT 'user',

registrationCode VARCHAR(100),

recoverPassCode VARCHAR(20),

active BOOLEAN DEFAULT false,

createdAt TIMESTAMP NOT NULL,

modifiedAt TIMESTAMP);

* + - CREATE TABLE IF NOT EXISTS news (

id INT UNSIGNED PRIMARY KEY AUTO\_INCREMENT,

category ENUM('deportes', 'videojuegos', 'noticias', 'programación', 'viajes', 'tecnología', 'música', 'memes', 'general') DEFAULT 'general',

feedback INT DEFAULT 0,

idUser INT UNSIGNED NOT NULL,

FOREIGN KEY (idUser) REFERENCES users(id),

title VARCHAR(100) NOT NULL,

photo VARCHAR(100),

summary VARCHAR(250),

body MEDIUMTEXT NOT NULL,

createdAt TIMESTAMP NOT NULL,

modifiedAt TIMESTAMP);

* + - CREATE TABLE IF NOT EXISTS votes (

id INT UNSIGNED PRIMARY KEY AUTO\_INCREMENT,

value BOOLEAN NOT NULL DEFAULT false,

idUser INT UNSIGNED NOT NULL,

FOREIGN KEY (idUSer) REFERENCES users(id),

idNews INT UNSIGNED NOT NULL,

FOREIGN KEY (idNews) REFERENCES news(id),

createdAt TIMESTAMP NOT NULL);

* + - CREATE TABLE IF NOT EXISTS photoNews (

id INT UNSIGNED PRIMARY KEY AUTO\_INCREMENT,

name VARCHAR(100) NOT NULL,

idNews INT UNSIGNED NOT NULL,

FOREIGN KEY (idNews) REFERENCES news(id),

createdAt TIMESTAMP NOT NULL,

modifiedAt TIMESTAMP);

# INTRODUCCIÓN

Antes de instalar alguna dependencia crearemos nuestro fichero package.json con la siguiente línea de comandos, situada en la carpeta de nuestro proyecto:

* + - npm init
    - npm init -y (si queremos crear package.json con valores por defecto)

En nuestro proyecto queda del siguiente modo:

{

  "name": "lineage",

  "version": "1.0.0",

  "description": "API que permita gestionar documentación según el rol del usuario",

  "main": "server.js",

  "scripts": {

    "dev": "nodemon server.js

  },

  "repository": {

    "type": "git",

    "url": "git+https://github.com/JuanGarridoTroche/PROYECTOS/lineage.git"

  },

  "author": "Juan Garrido Troche",

  "license": "ISC",

  "bugs": {

    "url": "https://github.com/JuanGarridoTroche/PROYECTOS/lineage/issues"

  },

  "homepage": "https://github.com/JuanGarridoTroche/PROYECTOS/lineage#readme",

}

# DEPENDENCIAS NECESARIAS PARA LINEAGE

Las dependencias son aquellas aplicaciones o bibliotecas requeridas por otro programa para poder funcionar correctamente. Por ello, se dice que determinado programa “depende” de tal aplicación o biblioteca (library). Para instalarlos, emplearemos un sistema de gestión de paquetes de Node llamado “npm” (Node Package Management).

Existen 3 tipos de dependencias:

* Dependencias CORE: Aquellas que ya vienen instaladas y se pueden utilizar solo llamando al módulo, como, por ejemplo, path o fs.
* Dependencias: Aquellas necesarias que instalaremos para que nuestra API funcione de manera correcta. Ejemplo: mysql2, dotenv, bcrypt, etc.
* Dependencias para desarrolladores(-D): Aquellas dependencias que nos harán la programación más fácil como control de sintaxis (eslint), formateo del código para que se vea ordenado (prettier) o reinicio de nuestro servidor cada vez que guardamos un cambio (nodemon). Este tipo de dependencias se instalan añadiendo un -D en la línea de comandos. En cuanto finalice nuestro proyecto y lo pongamos en producción se podrían desinstalar (npm uninstall mi\_DevDependence).

A continuación, vamos a indicar que dependencias son las que vamos a utilizar en nuestro proyecto:

|  |  |  |  |
| --- | --- | --- | --- |
| NOMBRE | INSTALACIÓN | TIPO | USO |
| express | npm i express | Dependencia | Nos permite levantar un servidor. Además nos permite leer los datos enviados desde el body en formato raw -JSON. |
| dotenv | npm i dotenv | Dependencia | Nos permite leer los valores de las variables del archivo oculto .env |
| mysql2 | npm i mysql2 | Dependencia | Crea una conexión con una base de datos y una vez conectados, podemos hacer “crud” de tablas y/o registros pero nunca crear la base de datos. |
| bcrypt | npm i bcrypt | Dependencia | Permite encriptar y desencriptar datos. |
| nodemon | npm i nodemon -D | Dev | Monitoriza los cambios en el código fuente que se está desarrollando y automáticamente reinicia el servidor que está corriendo sin tener que reiniciarlo. |
| jsonwebtoken | npm i jsonwebtoken | Dependencia | Nos crea un token de seguridad estándar para transmitir de forma segura en internet, por medio del formato JSON. |
| fileUpload | npm i express-fileupload | Dependencia | Decodifica (o deserializa) los datos enviados desde el body con el formato “form-data”. |
| fs | No hay que instalar | Core | Gestiona los ficheros. |
| path | No hay que instalar | Core | Gestiona las rutas de nuestros ficheros. |
| sharp | npm i sharp | Dependencia | Sirve para manipular nuestras imágenes como redimensionar o cambiar a otro tipo de extensión. |
| uuid | Npm i uuid | Dependencia | Renombra los ficheros que subamos a nuestra bbdd. |
| joi | Npm i @hapi/joi | Dependencia | Sirve para validar todo tipo de datos introducidos por el usuario. |
| randomstring | npm i randomstring | Dependencia | Genera cadenas de caracteres alfanuméricos. En nuestro caso lo utilizaremos para generar el código de recuperación. |
| nodemailer | npm i nodemailer | Dependencia | Nos permite enviar correos a través de una configuración sencilla. |

# RUTAS / ENDPOINTS

Un endpoint es una pasarela que conecta los procesos del servidor de la aplicación con una interfaz (API, Application Programming Interface), es decir, la ruta que hay que añadir a nuestra URL, a la que se envían peticiones.

Antes de empezar a programar nuestra API debemos establecer los endpoints necesarios para que nuestra aplicación sea funcional:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| MÉTODO | RUTA | FUNCIÓN | USO | PERMISOS |
| ENDPOINTS USUARIOS | | | | |
| POST | /users/login | loginUser.js | Login de usuario. | Todos los usuarios |
| POST | /users/register | registerUser.js | Registrar un nuevo usuario. | Todos los usuarios |
| PUT | /users/profile | editProfile.js | Puedes modificar el usuario, el email, la bio y desactivar tu cuenta. | isAuth |
| PUT | /users/password | editPassword.js | Editar password del usuario. | isAuth |
| PUT | /users/password/solicitude | sendRecoverPassword.js | Solicitud de nueva contraseña por email. | Todos los usuarios |
| PUT | /users/ password/recover | recoverPassword.js | Recuperación de contraseña. | isAuth, isImg |
| PUT | /users/photo | editPhoto.js | Editar nuestra foto de perfil. | isAuth, isImg |
| ENDPOINTS NOTICIAS | | | | |
| GET | /news/top | topRankedNews.js | Lista de las últimas noticias del día ordenadas por valoración. | Todos los usuarios |
| GET | /news/date=? | newsByDate.js | Noticias de días anteriores. | Todos los usuarios |
| GET | /news?category=”general” | newsByCategory.js | Noticias filtradas por categoría. | Todos los usuarios |
| POST | /news | createNews.js | Crear una noticia. | isAuth |
| PUT | /news/:idNews | editNews.js | Editar una noticia ya creada. | isAuth |
| DELETE | /news/:idNews | deleteNews.js | Borrar una noticia publicada. | isAuth |
| POST | /news/:idNews | voteNews.js | Vota una noticia publicada (de otro usuario). | isAuth |

FRONTEND

Habrá que comenzar haciendo la parte de usuario. Intentaremos dar las horas de trabajo a cada trabajo indivisible de nuestra web. Una jornada de trabajo equivale a 8h de trabajo.

Crear el entorno de usuario es algo que se hace habitualmente pero como estamos en el comienzo, cuesta más ir generándolo puesto que vamos creando las bases de la web. A medida que vamos implementando más funcionalidades, tendremos componentes ya creados que podremos reutilizar y por lo tanto iremos más rápidos:

Primero de todo el registro (2 jornadas de trabajo):

- Registro: RegisterPage.jsx

- Modal de registro: Modal.jsx

- Mensaje para el error y modal de registro (

<Modal setShowModal={setShowModal}>

<Message message=”mensaje enviado como children de Modal” type=”data”/>

</Modal>

Type=”data” o “error” para modificar el css del mensaje (color: black o red)

- Enviar correo de activación de cuenta (configurando sendinblue)

- Activar la cuenta con el código de registro que te llega al mail (configuración de un html para que le llegue al usuario con mayor presencia).

- css de las páginas RegisterPage.jsx, Modal, Message, validateRegistrationCode siguiendo la metodología BEM (Bloque\_\_elemento—Modificador)

- css mobile first y responsive para mobile (hasta 549px), Tablet (550px a 879px) y PC( más de 880px).

A continuación haremos el login (1 jornada de trabajo):

* Creamos el formulario con los campos usuario y contraseña.
* Creamos el servicio de login y guardamos el token que nos devuelve.
* Creamos el contexto AuthContext.jsx
  + Creamos el objeto global de contexto (Authcontext = createContext(); que servirá para leer las variables del contexto.
  + Crear un componente proveedor del contexto que envolverá a nuestra aplicación dentro de nuestro fichero main.jsx o index.jsx (incluso, dentro de App, siempre y cuando envuelva los componentes que necesiten las variables que queramos proveer):

<AuthProvidercomponent>

<App />

</AuthProvidercomponent>

Esto, lo que hace es crear un componente padre de todos los componentes (arriba de la pirámide) con x variables, las cuales tendremos acceso desde cualquier componente sin tener que llevar ir heredando esos valores desde uno o más componentes “padre”.

* css de LoginUserPage siguiendo la metodología BEM (Bloque\_\_elemento—Modificador).
* css mobile first y responsive para mobile (hasta 549px), Tablet (550px a 879px) y PC( más de 880px).

Editamos el perfil del usuario logueado:

* UserProfilePage.jsx
* css de UserProfilePage siguiendo la metodología BEM (Bloque\_\_elemento—Modificador).
* css mobile first y responsive para mobile (hasta 549px), Tablet (550px a 879px) y PC( más de 880px).