PROYECTO FINAL FULL-STACK DEL CURSO:
FULL-STACK DEVELOPER FEBRERO 2023

FREE ART ACADEMY

Autor: Aitor Iriarte Cervera
Tutor: Antonio Otero

Convocatoria Septiembre de 2024

INDICE

1. Introducción

1.1. Motivación y objetivos del proyecto:

Como artista autodidacta, acceder a recursos libres de aprendizaje siempre ha supuesto un reto. El objetivo del ¡proyecto es crear una plataforma que aglutine recursos y herramientas libres de calidad puestos al servicio de quién los necesite.

1.2. Descripción general de la plataforma

Free Art Academy es una plataforma donde los usuarios que se registren gratis tendrán acceso libre a una serie de recursos, principalmente cursos completos de dificultad variable.

1.3. Estructura del TFM

El TFM consta de; una memoria, el código completo almacenado en Github y un video que muestra el funcionamiento.

2. Análisis y Diseño del Proyecto

2.1. Requerimientos funcionales y no funcionales

Los **requerimientos funcionales** describen las funcionalidades esenciales que debe proporcionar el sistema para cumplir con los objetivos del proyecto:

1. **Autenticación de usuarios y gestión de datos**: El sistema debe permitir que los usuarios se registren y autentiquen para acceder a funciones protegidas, así como ver sus datos.
2. **Interfaz de usuario dinámica**: El front-end debe ser interactivo y ofrecer una experiencia de usuario fluida, basada en interacciones dinámicas con la API del backend.
3. **Comunicación cliente-servidor**: El sistema debe facilitar la comunicación entre el cliente (front-end) y el servidor (back-end) utilizando API REST.
4. **Visualización de datos**: Los datos almacenados deben poder visualizarse de manera clara a través de tablas, gráficos o listas. Ejem: Los cursos, o los datos del usuario.
5. Escalabilidad y mantenimineto: El diseño, las herramientas, lenguajes y frameworks han sido escogidos pensando en el futuro de la aplicación.

2.2. Arquitectura de la aplicación

La arquitectura de la aplicación utiliza un patrón **Cliente-Servidor** y **M.V.C.**, que consta de dos componentes principales; el frontend que corre en **Vue/Vite** y la lógica de negocio que la gestiona el backend implementado en **NestJS**. El backend se sirve de una API RESTful a través de la que se gestiona la operación de los datos.

2.3. Diseño del front-end con Vue y Vite

El front-end ha sido desarrollado en Vue 3, un framewrok JavaScript progresivo. Se han aprovechado su funcionalidades modulares, atomizando el diseño en componentes y partes reutilizables así como sus directivas que aumentan la flexibilidad en el desarrollo.

Así pues, las **vistas** serán fácilmente ampliables en funcionalidad y requisitos gracias a este enfoque.

Para la navegación entre rutas se ha optado por hacer uso del Router-View que incluye Vue3 el cual facilita el desarrollo tanto de las vistas y la navegación como la seguridad al proteger las rutas.

También se ha usado Vite en la parte del front-end, una herramienta que se usa como compilador y servidor de desarrollo que permite una recarga en caliente. Optimizando la aplicación el despliegue al ser un bundle eficiente.

Ambas herramientas en conjunto permitirán un escalado robusto de la aplicación conforme vaya aumentando la funcionalidad.

2.4. Diseño del back-end con NestJS

En el back-end se ha implementado **NestJs**, un robusto framework de Node.Js con **arquitectura modular** que permitirá que la aplicación sea **escalable** y **mantenible**.

El back-end se ha diseñado pensando en el principio de responsabilidad única. Programando los módulos para que se encarguen de una lógica de negocio en específico (autenticación, login, registros…). Tanto los controladores como los servicios, están diseñados para que su funcionalidad pueda ser ampliada, o reducida, y mantenida con facilidad.

2.5. Persistencia (HeidiSQL y TypeORM) y JSON

La información se almacena en una base de datos **MySQL** gestionada en este caso con el cliente **HeidiSQL**. La interacción con la base de datos se realiza en el back-end mediante el uso de la librería **TypeORM**, un ORM (Object-Relational Mapping), las **entidades** son gestionadas junto a los **DTO (Data-Transfer-Objects)** y el patrón **Repository,** que facilita la manipulación de datos mediante objetos en lugar de consultas **SQL** directas. Esto aumenta enormemente la **seguridad**, **fiabilidad** y reduce el porcentaje de **error**.

A la hora de comunicar datos desde el front-end al back-end, y viceversa, se ha usado el formato **JSON** tanto en las peticiones como para almacenar los datos de los cursos que se muestran en la vista (***web_host_name/courses/***) ya que ahora mismo por la naturaleza y el objetivo didáctico del presente proyecto se ha uso de un servicio de hosting externo (Youtube) y en el front-end se apunta a ellos.

3. Seguridad

En este proyecto, se han implementado varias medidas de seguridad tanto en el front-end como en el backend para asegurar una comunicación segura entre el cliente y el servidor, así como para proteger los datos sensibles.

3.1. CORS (Cross-Origin Resource Sharing)

Cross-Origin Resource Sharing (CORS) es un mecanismo que permite a las aplicaciones web controlar cómo se realizan las solicitudes HTTP desde dominios distintos al del servidor. En este proyecto, se ha habilitado CORS para permitir que el front-end (en Vue.js) pueda comunicarse con el backend (NestJS), dado que ambos pueden estar en servidores diferentes durante la fase de desarrollo o en producción.

Se ha configurado para restringir los orígenes permitidos y los métodos HTTP que pueden utilizarse, lo que mitiga ciertos riesgos de seguridad como ataques **CSRF (Cross-Site Request Forgery)** y **XSS (Cross-site scripting).**

CORS en el back-end de este proyecto:

//main.ts

```
const corsOptions: CorsOptions = {

   origin: 'http://localhost:5173', // Dominio permitido
   methods: 'GET,POST,PUT,PATCH,DELETE', // Métodos CRUD permitidos
   allowedHeaders: 'Content-Type, Authorization', // Cabeceras
permitidas
   credentials: true, // Si necesitas incluir cookies o autenticación
basada en cabeceras
  };
  app.enableCors(corsOptions);
```

3.2. Manejo de rutas

Se ha implementado un sistema de autenticación y autorización que restringe el acceso a ciertas rutas según el rol del usuario.

Ejemplo de control de rutas en este proyecto:

//index.ts

```
router.beforeEach((to, from, next) => {
  const isAuthenticated = !!localStorage.getItem('token');  // O
comprobar cookie
  if (to.meta.requiresAuth && !isAuthenticated) {
    next('/login');  // Redirigir al login si no está autenticado
  } else {
    next();
  }
});
```

3.3. JWT y cookies

El sistema utiliza JSON Web Tokens (JWT), una forma estándar y segura de realizar autenticación sin necesidad de almacenar sesiones en el servidor. Además, se utilizan cookies para almacenar el token de manera segura en el navegador del usuario.

Implementación en este proyecto:

// auth.service.ts

```
export class AuthService {
 private readonly JWT_SECRET = 'jwt'; //Sustituir en producion por ENV
 constructor(private readonly userService: UserService) {}
 //Creamos y firmamos el token
 generateToken(payload: any): string {
  return jwt.sign(payload, this.JWT_SECRET, { expiresIn: '1h' });
 }
```

Ejemplo de JSON Web Token:

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

```
HMACSHA256(
  base64UrlEncode(header)
+ "." +

base64UrlEncode(payload),
'jwt'

) secret base64 encoded
```

Fuente https://jwt.io/

3.4. Encriptación con Bcrypt

Para garantizar que las contraseñas de los usuarios estén seguras en caso de que la base de datos se vea comprometida, se ha implementado Bcrypt para el hashing de contraseñas.

Ejemplo de implentación de hashing con Bcrypt (Fuente: documentación de BCrypt)

```
import * as bcrypt from 'bcrypt';


//Hashing password
async hashPassword(password: string): Promise<string> {
  const salt = await bcrypt.genSalt();
  return await bcrypt.hash(password, salt);
}

//Verify password

async comparePassword(enteredPassword: string, hashedPassword: string):
Promise<boolean> {
  return await bcrypt.compare(enteredPassword, hashedPassword);
}
```

3.5. Variable de entorno

Para gestionar de manera segura la configuración sensible del proyecto, como claves secretas o credenciales de base de datos, se utilizan **variables de entorno**. Estas variables permiten separar la configuración de la lógica del código, facilitando el despliegue en diferentes entornos (desarrollo, producción). En el código de este proyecto está señalado mediante comentarios los sitios donde debe usarse las variables de entorno. Por su naturaleza y objetivo didáctico y para facilitar el entendimiento del flujo del programa no se ha implementado en el código final.

Pueden instalarse librería como ***dotenv*** o importar archivos ***.env*** con @nestjs/config en el caso del back-end para usar variables de entorno de forma cómoda y sencilla:

Ejemplo de archivo .env

PORT=3000
DB_HOST=localhost
DB_PORT=5432
JWT_SECRET=ClaveSecreta

4. Pruebas y Validación

Las pruebas cubren desde la verificación de pequeñas unidades de código hasta la validación de la experiencia del usuario con el sistema completo.

4.1. Validación con usuarios reales

Se realizó una fase de validación con usuarios reales para obtener retroalimentación sobre la funcionalidad y la experiencia de usuario. Esta validación se hizo mediante pruebas controladas, donde se les pidió a los usuarios que interactuaran con la aplicación mientras se observaba su comportamiento y se registraban los comentarios.

4.2. Postman

Para realizar pruebas exhaustivas de la API REST, se utilizó Postman como herramienta de testing. Se creó una colección de pruebas para simular distintos flujos de usuario y comprobar la correcta respuesta de los endpoints.

```
POST /{{HOST}}/login
Content-Type: application/json
Body: {
  "userEmail":
"admin@gmail.com",
  "userPassword": "123456"
}
```

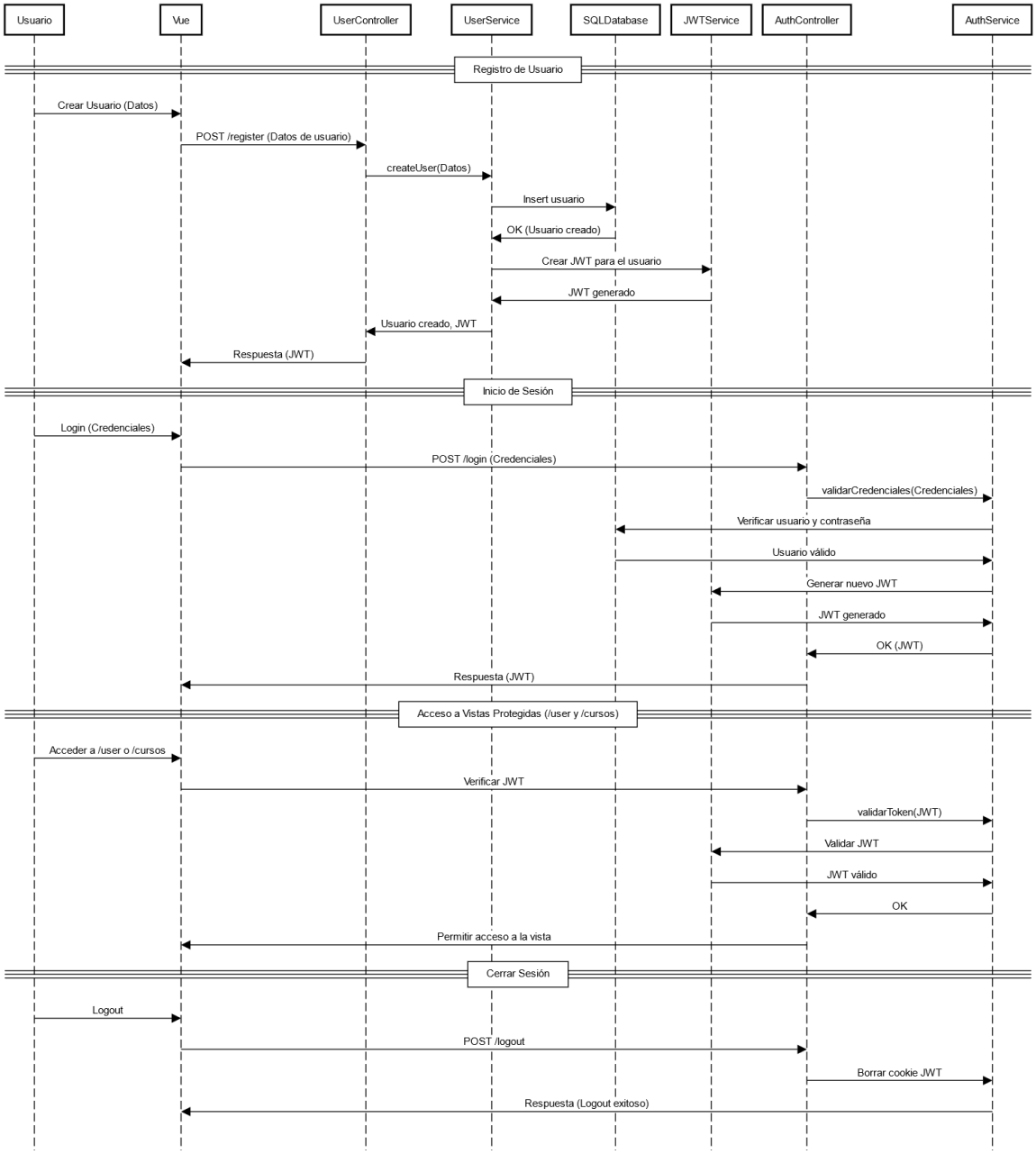
5. Bibliografía y Referencias

5.1. Documentación oficial de Vue, Vite, NestJS y TypeORM

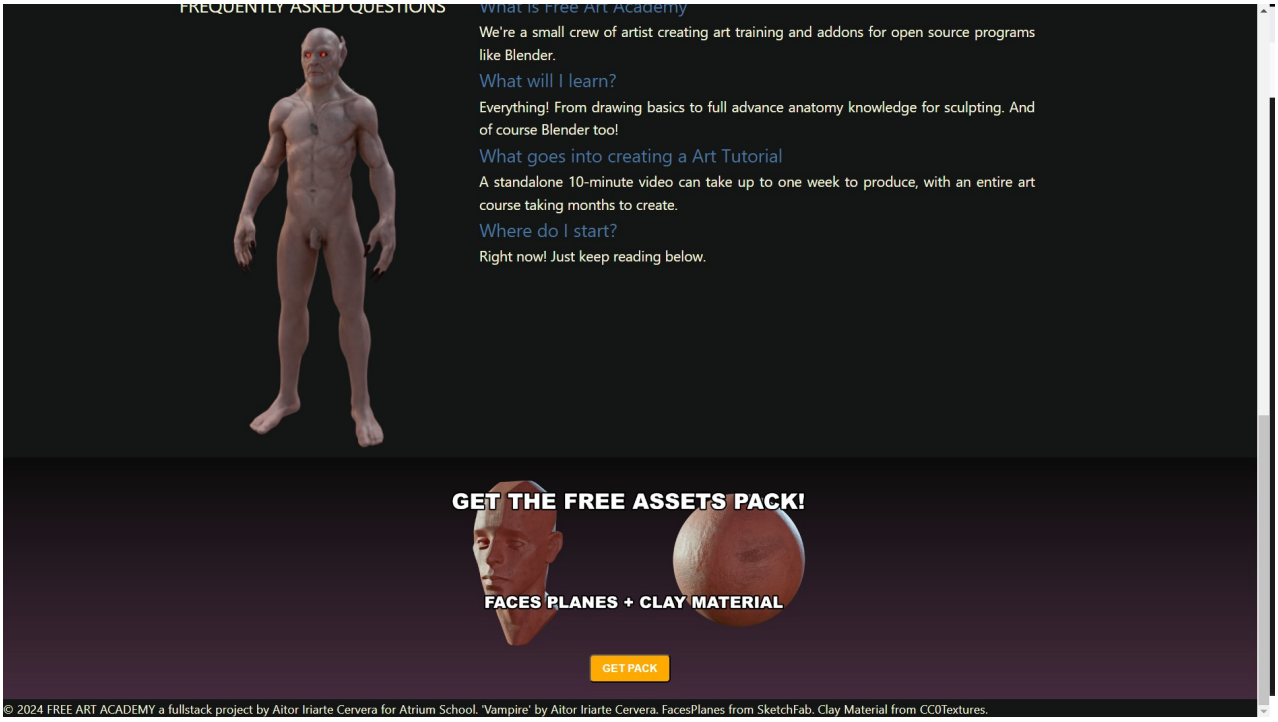
- **Vite Documentation**: https://vitejs.dev/guide/
- **NestJS Documentation**: https://docs.nestjs.com/
- **Postman Documentation**: https://learning.postman.com/docs/getting-started/introduction/
- **TypeORM Documentation**: https://typeorm.io/

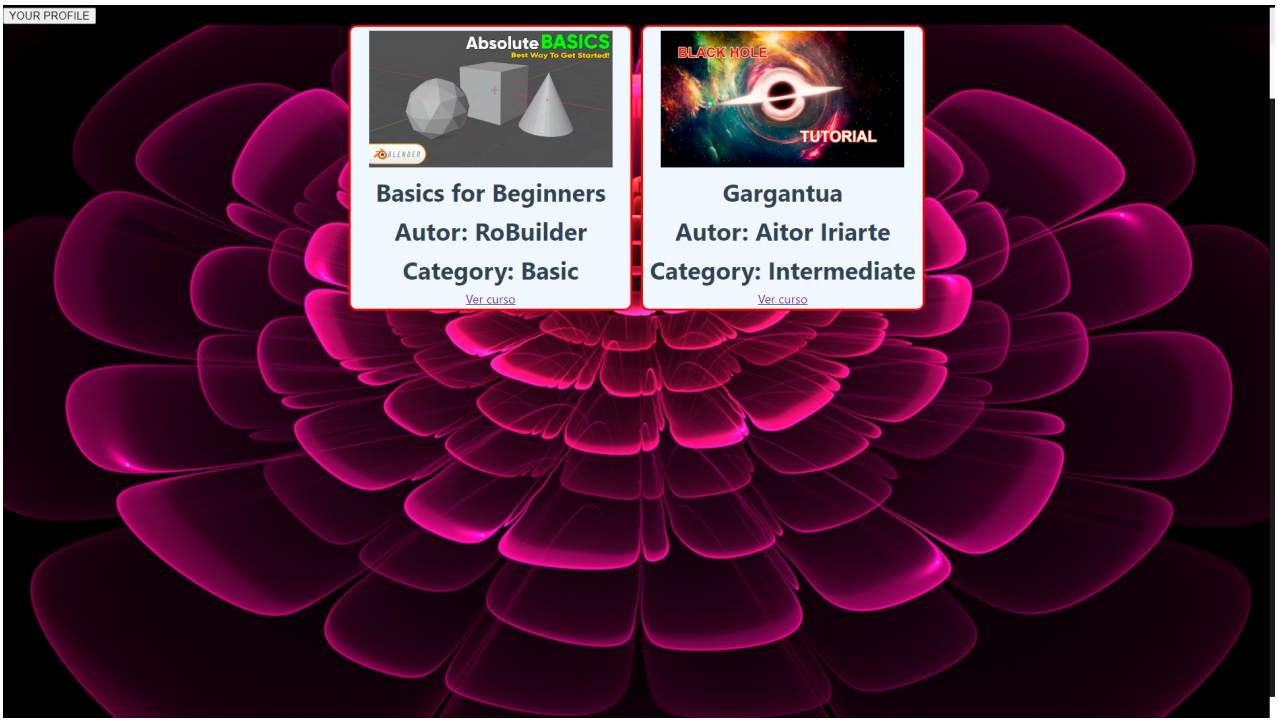
6. Anexos

6.1. Diagrama secuencial de la aplicación.



6.2. Capturas de pantalla del prototipo final





6.3. Manual de usuario o guía de instalación

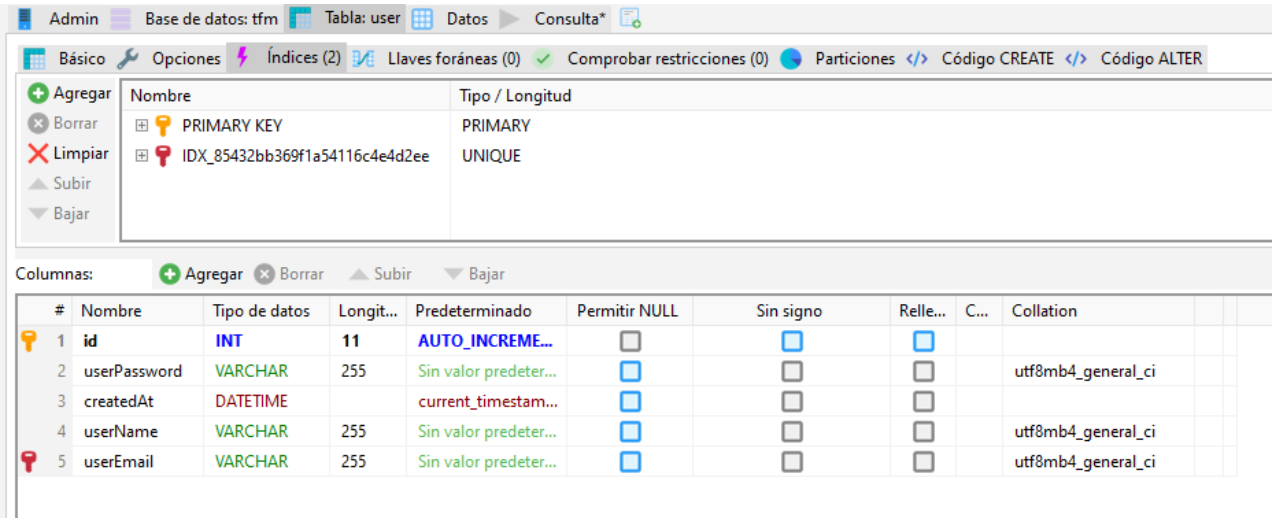
- **Clonar el repositorio**
  git clone https://github.com/Iriarte93/Free_Art_Academy
- **Una vez clonado, accede a la carpeta del repositorio:**

cd free_art_academy

- **Instalación de dependencias. Instala una herramienta de gestión, en esta guía se usa npm.** Ejecuta *'npm install'* en la carpeta del back-end y en la carpeta del front-end
- **Configura en el app.module.ts del back-end, tu base de datos SQL.**

Si el parámetro **'Synchronize:' está en 'true'**, sino existe la tabla en la base de datos debería de generarse automáticamente. Sino puedes hacerla manualmente o restaurar la copia que viene en el repositorio.

```
@Module({
  imports: [
    TypeOrmModule.forRoot({
      type: 'mysql',
      host: 'localhost',
      port: 3306,
      username: 'admin',
      password: 'admin',
      database: 'tfm',
      entities: [__dirname + '/**/*.entity{.ts,.js}'],
      synchronize: true,
    }),
    UserModule,
    AuthModule,
  ],
})
```



- **Iniciar el front-end:**

  npm run dev

- **Inicia el backend**

  npm run start:dev

- **Ya puedes navegar. Puedes crear un usuario nuevo o si restauraste la base de datos usar “usuario:** *admin@gmail.com*” **“password:** *1234*”

7. Créditos

FREE ART ACADEMY, proyecto full-stack por Aitor Iriarte Cervera para Grupo Atrium.

7.1. Arte:

- 'Vampire' creado originalmente por Aitor Iriarte Cervera para el cortometraje ‘Ex Anima’ © 20024 de Aitor Iriarte Cervera.
- Head Planes Reference by ZSoltsulyok from SketchFab.
- Clay002_4K material from CC0Textures.
- Login User Icon from Pixabay. CC
- Fractal Background from Pixabay. CC
- Video: Blender Basic Concepts by RobBuilder