

ChatIES

Adrián Marín Chorro.
José Manuel Mármol Alfocea.
2º Desarrollo de Aplicaciones Web.

Índice

1. Introducción:	3
2. Requisitos:	3
Requisitos Funcionales:	3
Requisitos No Funcionales:	3
3. Arquitectura de software:	4
4. Diseño:	4
5. Proceso y desarrollo:	5
1. Instalación del framework y contenedores de docker.	5
2. Instalación y configuración de LdapRecord.	7
3. Front-end de la aplicación: Vue.	18
4. Back-end de la aplicación: Laravel	28
5. Testing (PHPUnit)	30
6. Despliegue de la Aplicación	32

1. Introducción:

El proyecto Chatles tuvo como objetivo el desarrollo de una aplicación web que emula las funcionalidades del sistema ChatGPT mediante la API de OpenAI. Esta aplicación brinda a los profesores la capacidad de utilizarla a través de un sistema de registro y autenticación de usuarios autorizados, basado en el protocolo LDAP.

2. Requisitos:

Requisitos Funcionales:

1. Inicio de sesión: Los usuarios podrán iniciar sesión en la aplicación utilizando sus credenciales registradas en el servidor LDAP.
2. Interfaz de Chat: La aplicación proporciona una interfaz de chat donde los usuarios pueden interactuar con ChatGPT mediante mensajes de texto.
3. Envío de mensajes: Los usuarios podrán enviar mensajes a ChatGPT para hacer preguntas o solicitar información.
4. Respuestas de ChatGPT: La inteligencia artificial generará respuestas basadas en los mensajes recibidos de los usuarios, proporcionando información relevante.
5. Historial de chat: La aplicación almacenará un historial de chat para cada usuario, permitiéndoles ver las conversaciones pasadas con ChatGPT.
6. Eliminación del historial: Los usuarios podrán seleccionar y eliminar partes específicas del historial de chat o borrar el historial completo según su preferencia.

Requisitos No Funcionales:

1. Usabilidad: La aplicación debe ser fácil de usar y tener una interfaz intuitiva.
2. Seguridad: La aplicación debe implementar medidas de seguridad, como la autenticación de usuarios, para proteger la información confidencial.
3. Tiempo de respuesta: La aplicación debe responder rápidamente a las consultas de los usuarios.
4. Disponibilidad: La aplicación debe estar disponible y accesible para los usuarios.
5. Escalabilidad: La aplicación debe ser capaz de manejar un número creciente de usuarios y mantener un rendimiento óptimo.
6. Compatibilidad del navegador: La aplicación debe ser compatible con los navegadores más populares (Google Chrome, Mozilla Firefox).

3. Arquitectura de software:

En el proyecto, hemos optado por utilizar el patrón Modelo-Vista-Controlador (MVC) para la estructura y organización del código. El patrón MVC es ampliamente utilizado en el desarrollo de aplicaciones web y se basa en la separación de responsabilidades entre los componentes clave del sistema.

Modelo: Es responsable de manejar la lógica de negocio, interactuar con la base de datos y proporcionar los datos necesarios para la vista.

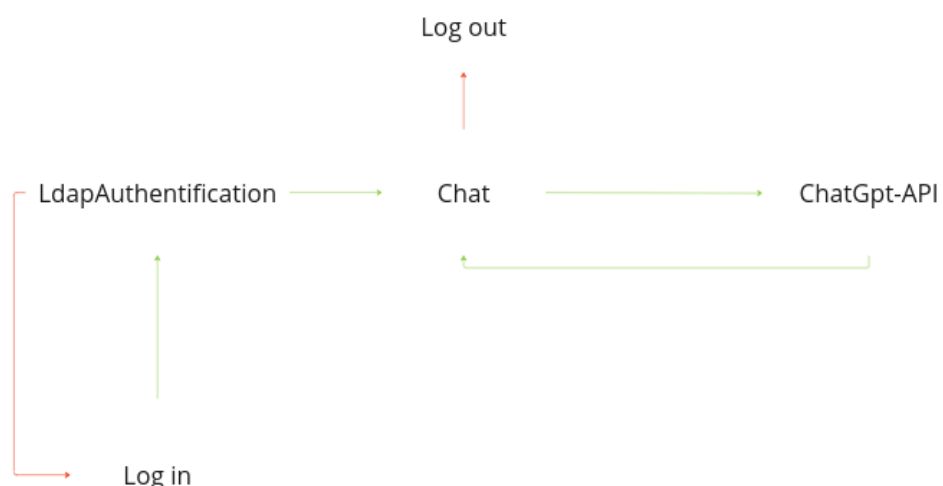
Vista: Es la interfaz de usuario de la aplicación. Es responsable de la presentación de los datos al usuario y de recibir las interacciones del usuario.

Controlador: Actúa como intermediario entre el modelo y la vista. Es responsable de manejar las solicitudes del usuario, coordinar las acciones requeridas y actualizar tanto el modelo como la vista según sea necesario.

4. Diseño:

Con el fin de diseñar la aplicación, hemos empleado el enfoque de desarrollar nuestra propia aplicación de ChatGPT, tomando a éste como referencia. Sin embargo, como previamente mencionamos, la autenticación de usuario será gestionada por el servicio LDAP.

En consecuencia, hemos elaborado un esquema inicial que presenta de manera general el flujo completo de la aplicación.



Para el diseño de la base de datos, hemos adoptado un esquema entidad-relación que se ajusta a los requisitos y estructura de nuestra aplicación. Este esquema ha

sido cuidadosamente elaborado para garantizar la integridad y eficiencia de la gestión de datos en el proyecto.



5. Proceso y desarrollo:

1. Instalación del framework y contenedores de docker.

Durante el desarrollo del proyecto, se ha utilizado el framework Laravel 10 para la construcción de la aplicación web. Dado que el proceso de instalación de una aplicación web desde cero utilizando Laravel ha sido cubierto en detalle durante las clases, se ha omitido incluir ese paso en la presente documentación.

No obstante, para aquellos que deseen obtener información detallada sobre el proceso de instalación de Laravel, se proporciona un enlace a la documentación oficial del framework. Esta documentación incluye instrucciones precisas sobre cómo instalar y configurar Laravel en un entorno de desarrollo:

<https://laravel.com/docs/10.x/installation>.

En lugar de utilizar Laradock para levantar los contenedores de Docker, se ha optado por utilizar Laravel Sail, una herramienta proporcionada por Laravel que simplifica el entorno de desarrollo local utilizando contenedores Docker.

Antes de utilizar Laravel Sail para levantar los contenedores de Docker, es necesario realizar la instalación de esta herramienta. A continuación, se detallan los comandos necesarios para instalar Laravel Sail:

En primer lugar, debemos agregar el paquete de Laravel Sail al proyecto ejecutando el siguiente comando en la terminal:

composer require laravel/sail --dev

Este comando instalará Laravel Sail como una dependencia de desarrollo del proyecto.

Una vez que se haya instalado Laravel Sail, se debe generar el archivo de configuración de Sail ejecutando el siguiente comando:

php artisan sail:install

Este comando generará el archivo docker-compose.yml y los archivos de configuración necesarios para Laravel Sail.

Una vez completada la instalación de Laravel Sail, podemos proceder a utilizar los comandos para crear y gestionar los contenedores Docker. A continuación, se detallan dichos comandos:

Para crear los contenedores y levantar el entorno de desarrollo, se ejecuta el siguiente comando en la terminal:

`./vendor/bin/sail up -d`

Este comando iniciará los contenedores de Docker definidos en el archivo docker-compose.yml y configurados en el archivo .env.

Para detener y eliminar los contenedores del entorno de desarrollo, se utiliza el siguiente comando:

`./vendor/bin/sail down`

En caso de que surjan dudas o se requiera más información sobre Laravel Sail, se recomienda consultar la documentación oficial de Laravel Sail. Esta documentación proporciona una guía completa y detallada sobre el uso de Laravel Sail y puede ser encontrada en el siguiente enlace:

<https://laravel.com/docs/10.x/sail#main-content>.

Para poder trabajar con OpenLDAP y phpLDAPAdmin en un entorno local, se requirió agregar las siguientes configuraciones al archivo de configuración **composer.yml**:

```
openldap:
  image: 'osixia/openldap:1.5.0'
  ports:
    - '389:389'
  volumes:
    - 'ldap_data:/var/lib/ldap'
    - 'ldap_config:/etc/ldap/slapd.d'
  networks:
    - sail
phpldapadmin:
  image: 'osixia/phpldapadmin:latest'
  environment:
    PHPLDAPADMIN_LDAP_HOSTS: openldap
```

```
    PHPLDAPADMIN_HTTPS: 'false'
  ports:
    - '8080:80'
  depends_on:
    - openldap
  networks:
    - sail
```

Estas configuraciones definen los servicios de OpenLDAP y phpLDAPadmin utilizando contenedores Docker.

2. Instalación y configuración de LdapRecord.

LdapRecord-Laravel es un paquete de software diseñado para facilitar la búsqueda en directorios LDAP, ejecutar operaciones y autenticar usuarios LDAP dentro de una aplicación de Laravel.

Nosotros la vamos a utilizar para esto último.

A continuación, procederemos a detallar el proceso de instalación de LdapRecord-Laravel. Sin embargo, es importante destacar que en caso de surgir alguna duda durante el proceso, se recomienda consultar detalladamente la documentación oficial del paquete.

<https://ldaprecord.com/docs/laravel/v2>.

Pasos de instalación:

composer require directorytree/ldaprecord-laravel

Este comando instala el paquete LdapRecord dentro de nuestro proyecto de Laravel.

En caso de que encuentres un error al intentar instalar ldaprecord mediante el comando anterior, es posible solucionarlo siguiendo la recomendación proporcionada por la consola. La recomendación consiste en restaurar el archivo composer.lock a su estado original. Al eliminar este archivo y volver a ejecutar el comando, se generará un nuevo composer.lock y se realizará la instalación sin problemas.

El archivo composer.lock es generado por Composer y contiene información detallada sobre las versiones y dependencias de los paquetes instalados en tu proyecto. Al eliminarlo, obligas a Composer a reconstruirlo desde cero, asegurando así que todas las dependencias estén correctamente resueltas.

Después de completar los pasos anteriores, se debe ejecutar el siguiente comando:

```
php artisan vendor:publish  
--provider="LdapRecord\Laravel\LdapServiceProvider"
```

Este comando generará en nuestro proyecto el archivo `Ldap.php`. Una vez que hayamos ejecutado este comando, procederemos a agregar los siguientes campos en nuestro archivo `.env`:

```
LDAP_LOGGING=true  
LDAP_CONNECTION=default  
LDAP_HOST="openldap"  
LDAP_USERNAME="cn=admin,dc=example,dc=org"  
LDAP_PASSWORD=admin  
LDAP_PORT=389  
LDAP_BASE_DN="dc=example,dc=org"  
LDAP_TIMEOUT=5  
LDAP_SSL=false  
LDAP_TLS=false  
LDAP_GROUP="Chatuser"
```

Una vez completados los pasos anteriores, procederemos a realizar una prueba de conexión entre el servidor LDAP y nuestra aplicación. Para ello, utilizaremos el comando `php artisan ldap:test`.

Este comando nos permitirá verificar que la configuración establecida en nuestra aplicación, incluyendo el archivo `Ldap.php`, sea correcta y que la conexión con el servidor LDAP se establezca de manera exitosa.

Durante la ejecución de este comando, se realizarán pruebas de conectividad y se evaluará la capacidad de comunicación entre nuestra aplicación y el servidor LDAP. Cualquier error o problema detectado durante esta prueba nos indicará la necesidad de revisar y ajustar la configuración correspondiente, o bien solucionar cualquier inconveniente de conectividad que pueda surgir.

Una vez finalizados los pasos anteriores y verificado que nuestra prueba de conexión haya sido exitosa, procederemos a agregar la migración del LDAP a nuestra aplicación. Para realizar esta tarea, utilizaremos el siguiente comando:

```
php artisan vendor:publish  
--provider="LdapRecord\Laravel\LdapAuthServiceProvider"
```

Este comando permitirá la publicación del proveedor de autenticación LDAP, específicamente la migración asociada al mismo. La migración del LDAP es esencial para configurar la estructura de la base de datos en relación a la autenticación y autorización de usuarios LDAP.

Al ejecutar este comando, se generará la migración necesaria para configurar los esquemas y tablas requeridos en nuestra base de datos para la integración con LDAP.

Es importante destacar que la ejecución de este comando debe llevarse a cabo una vez que estemos seguros de que la conexión con el servidor LDAP ha sido establecida correctamente y de que hemos realizado todas las configuraciones necesarias previas a la migración.

Para configurar el proveedor de autenticación de Ldap con una base de datos, tenemos que dirigirnos al fichero config/auth.php y sustituir el array de 'providers' por el siguiente:

```
'providers' => [  
    'users' => [  
        'driver' => 'ldap',  
        'model' => LdapRecord\Models\OpenLDAP\User::class,  
        'rules' => [  
            App\Ldap\Rules\OnlyAdministrators::class,  
        ],  
        'database' => [  
            'model' => App\Models\User::class,  
            'password_column' => false,  
            'sync_attributes' => [  
                'name' => 'cn',  
                'username' => 'uid',  
                'email' => 'mail',  
            ],  
        ],  
    ],  
],
```

El campo '**driver**' se establece como 'ldap' para indicar que se utilizará LDAP como método de autenticación.

El campo '**model**' se asigna a la clase LdapRecord\Models\OpenLDAP\User::class para especificar el modelo de usuario utilizado en el contexto de LDAP.

El campo '**rules**' define una regla de acceso específica para el proveedor LDAP. En este ejemplo, se utiliza la clase App\Ldap\Rules\OnlyAdministrators::class como regla de acceso.

El campo '**database**' establece la configuración relacionada con la base de datos sincronizada. El modelo de usuario de la base de datos se establece como App\Models\User::class. El campo '**password_column**' se desactiva (false) para indicar que la contraseña no se almacenará en la base de datos sincronizada. Los atributos a sincronizar entre LDAP y la base de datos se definen en el campo '**sync_attributes**', donde se especifican los correspondientes atributos de nombre, nombre de usuario y correo electrónico.

Una vez realizado este proceso de configuración, el proveedor de autenticación LDAP con base de datos sincronizada estará correctamente establecido en la aplicación.

A continuación, procederemos a instalar Laravel Breeze. Laravel Breeze es un kit de inicio liviano diseñado para brindar una configuración mínima y funcionalidad esencial para la autenticación en aplicaciones Laravel.

Para instalar Laravel Breeze, ejecutaremos el siguiente comando en la línea de comandos:

composer require laravel/breeze --dev

Este comando descarga e instala todas las dependencias necesarias para utilizar Laravel Breeze en nuestro proyecto.

Una vez completada la instalación, podremos generar las vistas y rutas básicas para la autenticación ejecutando el siguiente comando:

php artisan breeze:install

Este comando generará las vistas y rutas necesarias para el registro, inicio de sesión, restablecimiento de contraseña y verificación de correo electrónico.

Al ejecutar el comando, se mostrará un menú interactivo que nos permitirá seleccionar el paquete que deseamos instalar para el frontend de nuestra aplicación. Entre las opciones disponibles se encuentran Vue, React, Blade, y otros.

En nuestro caso, hemos elegido Vue como paquete para el frontend. Esta elección nos permitirá utilizar Vue.js, un framework de JavaScript popular y altamente flexible, para construir la interfaz de usuario de nuestra aplicación Laravel.

Al seleccionar Vue, el comando generará la estructura inicial necesaria, incluyendo las dependencias y configuraciones correspondientes, para trabajar con Vue en nuestra aplicación.

Es importante destacar que esta elección determinará la forma en que construiremos y desarrollaremos el frontend de nuestra aplicación, y deberá ajustarse a nuestras necesidades y conocimientos específicos.

Además, también nos permite elegir la opción de pruebas (testing) que deseamos utilizar en nuestra aplicación. En nuestro caso, hemos seleccionado PHPUnit como opción de pruebas, el cual ya se ha visto en clase.

Una vez realizado este proceso, procedemos a ejecutar los comandos **npm install** y **npm run dev**.

El comando npm install se utiliza para instalar todas las dependencias necesarias para nuestro proyecto Laravel. Esto incluye las bibliotecas y paquetes requeridos por el frontend, como Vue, React o cualquier otra biblioteca que hayamos seleccionado durante la configuración.

Posteriormente, el comando `npm run dev` se encarga de compilar y generar los archivos JavaScript y CSS optimizados para producción. Esto nos permite preparar el frontend de nuestra aplicación para su despliegue en un entorno de producción.

La ejecución de estos comandos asegura que todas las dependencias estén correctamente instaladas y que los activos (assets) del frontend estén generados de manera adecuada

Antes de proceder a ejecutar las migraciones, es necesario configurar la migración que establecerá la conexión entre nuestro servidor LDAP y la base de datos de la aplicación.

Para ello modificamos la migración de base del modelo Users:

```
public function up(): void
{
    Schema::create('users', function (Blueprint $table) {
        $table->id();
        $table->string('name');
        $table->string('username')->nullable();
        $table->string('email')->nullable();
        $table->timestamp('email_verified_at')->nullable();
        $table->string('password')->nullable();
        $table->rememberToken();
        $table->timestamps();
    });
}
```

También generamos la siguiente migración:

```
<?php

use Illuminate\Support\Facades\DB;
use Illuminate\Support\Facades\Schema;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class AddLdapColumnsToUsersTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        $driver = Schema::getConnection()->getDriverName();
```

```

        Schema::table('users', function (Blueprint $table) use
($driver) {
            $table->string('guid', 40)->nullable();
            $table->string('domain')->nullable();

            if ($driver !== 'sqlsrv') {
                $table->unique('guid');
            }
        });

        if ($driver === 'sqlsrv') {
            DB::statement(
$this->compileUniqueSqlServerIndexStatement('users', 'guid')
            );
        }
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::table('users', function (Blueprint $table) {
            $table->dropColumn(['guid', 'domain']);
        });
    }

    /**
     * Compile a compatible "unique" SQL Server index
    constraint.
     *
     * @param string $table
     * @param string $column
     *
     * @return string
     */
    protected function
compileUniqueSqlServerIndexStatement($table, $column)
    {
        return sprintf('create unique index %s on %s (%s)
where %s is not null',
            implode('_', [$table, $column, 'unique']),
            $table,
            $column,
            $column

```

```

    );
  }
}

```

Esta última migración agrega las columnas **guid** y **domain** a la tabla **users**, donde se verifica el controlador de la base de datos para añadir una restricción de tipo única a la columna **guid**, en caso de que no sea SQL Server, para así evitar duplicados.

Ahora vamos a proceder a irnos a la vista que se encarga del login, puesto que vamos a utilizar el nombre de usuario en vez del correo electrónico para el inicio de sesión.

```

<script setup>
import Checkbox from '@/Components/Checkbox.vue';
import GuestLayout from '@/Layouts/GuestLayout.vue';
import InputError from '@/Components/InputError.vue';
import InputLabel from '@/Components/InputLabel.vue';
import PrimaryButton from '@/Components/PrimaryButton.vue';
import TextInput from '@/Components/TextInput.vue';
import { Head, Link, useForm } from '@inertiajs/vue3';

defineProps({
  canResetPassword: {
    type: Boolean,
  },
  status: {
    type: String,
  },
});

const form = useForm({
  username: '',
  password: '',
  remember: false,
});

const submit = () => {
  form.post('/login', {
    onFinish: () => form.reset('password'),
  });
};
</script>

<template>
  <GuestLayout>
    <Head title="Iniciar sesión" />

```

```

        <div v-if="status" class="mb-4 font-medium text-sm
text-green-600">
            {{ status }}
        </div>

        <form @submit.prevent="submit">
            <div>
                <InputLabel for="username" value="Usuario" />

                <TextInput
                    id="username"
                    type="text"
                    class="mt-1 block w-full"
                    v-model="form.username"
                    required
                    autofocus
                    autocomplete="username"
                />

                <InputError class="mt-2"
:message="form.errors.username" />
            </div>

            <div class="mt-4">
                <InputLabel for="password" value="Contraseña"
/>

                <TextInput
                    id="password"
                    type="password"
                    class="mt-1 block w-full"
                    v-model="form.password"
                    required
                    autocomplete="current-password"
                />

                <InputError class="mt-2"
:message="form.errors.password" />
            </div>

            <div class="block mt-4">
                <label class="flex items-center">
                    <Checkbox name="remember"
v-model:checked="form.remember" />
                    <span class="ml-2 text-sm
text-gray-400">Remember me</span>
                </label>
            </div>

```

```

        <div class="flex items-center justify-end mt-4">
            <Link
                v-if="canResetPassword"
                :href="route('password.request')"
                class="underline text-sm text-gray-400
hover:text-gray-100 rounded-md focus:outline-none
focus:ring-2 focus:ring-offset-2 focus:ring-offset-gray-800"
            >
                Forgot your password?
            </Link>

            <PrimaryButton class="ml-4" :class="{
'opacity-25': form.processing }" :disabled="form.processing">
                Log in
            </PrimaryButton>
        </div>
    </form>
</GuestLayout>
</template>

```

Una vez hayamos hecho esto procedemos a modificar el archivo
app\Http\Requests\Auth\LoginRequest.php de la siguiente manera:

```

<?php

namespace App\Http\Requests\Auth;

use Illuminate\Auth\Events\Lockout;
use Illuminate\Foundation\Http\FormRequest;
use Illuminate\Support\Facades\Auth;
use Illuminate\Support\Facades\RateLimiter;
use Illuminate\Support\Str;
use Illuminate\Validation\ValidationException;

class LoginRequest extends FormRequest
{
    /**
     * Determine if the user is authorized to make this
    request.
     */
    public function authorize(): bool
    {
        return true;
    }

    /**

```

```

    * Get the validation rules that apply to the request.
    *
    * @return array<string,
\Illuminate\Contracts\Validation\Rule|array|string>
    */
    public function rules(): array
    {
        return [
            'username' => ['required', 'string'],
            'password' => ['required', 'string'],
        ];
    }

    /**
     * Attempt to authenticate the request's credentials.
     *
     * @throws \Illuminate\Validation\ValidationException
     */
    public function authenticate(): void
    {
        $this->ensureIsNotRateLimited();

        $credentials = [
            'uid' => $this->username,
            'password' => $this->password,
        ];

        if (! Auth::attempt($credentials,
$this->filled('remember')) {
            RateLimiter::hit($this->throttleKey());

            throw ValidationException::withMessages([
                'username' => 'Las credenciales introducidas
no son correctas',
            ]);
        }

        RateLimiter::clear($this->throttleKey());
    }

    /**
     * Ensure the login request is not rate limited.
     *
     * @throws \Illuminate\Validation\ValidationException
     */
    public function ensureIsNotRateLimited(): void
    {

```



```

        if (!
RateLimiter::tooManyAttempts($this->throttleKey(), 5)) {
            return;
        }

        event(new Lockout($this));

        $seconds =
RateLimiter::availableIn($this->throttleKey());

        throw ValidationException::withMessages([
            'username' => trans('auth.throttle', [
                'seconds' => $seconds,
                'minutes' => ceil($seconds / 60),
            ]),
        ]);
    }

    /**
     * Get the rate limiting throttle key for the request.
     */
    public function throttleKey(): string
    {
        return
Str::transliterate(Str::lower($this->input('username')).'|'.$
this->ip());
    }
}

```

Con esto utilizamos el campo **uid**, como nombre del usuario.

Ahora procedemos a realizar modificaciones en el modelo **User**, añadiendo los paquetes necesarios para autenticar al usuario mediante Ldap:

```

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Relations\HasOne;
use Illuminate\Foundation\Auth\User as Authenticatable;
use Illuminate\Notifications\Notifiable;
use LdapRecord\Laravel\Auth\HasLdapUser;
use LdapRecord\Laravel\Auth\LdapAuthenticatable;
use LdapRecord\Laravel\Auth\AuthenticatesWithLdap;

class User extends Authenticatable implements
LdapAuthenticatable
{

```

```

    use HasFactory, Notifiable, AuthenticatesWithLdap,
    HasLdapUser;

    /**
     * The attributes that are mass assignable.
     *
     * @var array<int, string>
     */
    protected $fillable = [
        'name',
        'email',
        'password',
    ];

    /**
     * The attributes that should be hidden for serialization.
     *
     * @var array<int, string>
     */
    protected $hidden = [
        'password',
        'remember_token',
    ];

    /**
     * The attributes that should be cast.
     *
     * @var array<string, string>
     */
    protected $casts = [
        'email_verified_at' => 'datetime',
    ];
}

```

Tras seguir los pasos descritos anteriormente, hemos finalizado exitosamente la instalación y configuración de la librería LdapRecord en nuestro proyecto.

3. Front-end de la aplicación: Vue.

Para el frontend de la aplicación, se ha optado por utilizar el framework Vue.js. Vue.js es una biblioteca de JavaScript de código abierto que se utiliza para construir interfaces de usuario interactivas.

Con Vue.js, se pueden crear vistas dinámicas y actualizaciones en tiempo real sin la necesidad de recargar la página.

En el frontend de la aplicación, se ha utilizado el layout base proporcionado por Vue.js, con modificaciones en el CSS y Html para lograr el diseño deseado y garantizar la capacidad de respuesta (responsive) de la aplicación. Este layout se ha establecido como la plantilla principal que será extendida por todas las vistas y componentes creados para la aplicación.

Una vez explicado esto, es importante comprender cómo se han generado las vistas y cómo se propagan las funciones de JavaScript generadas, así como la información proveniente del backend de la aplicación, para ello vamos a proceder a explicar aquellas vistas y/o componentes que se consideran más importantes:

Chat.vue:

```
<template>
  <Head title="Chat" />
  <ChatLayout>

    <div class="flex">

      <div class="w-1/6 bg-gray-800 border-4
border-gray-600">
        <SideBarChat :allChats="allChats"
:chat="chats"
:loadingResponse="loadingResponse"></SideBarChat>
      </div>

      <div class="w-5/6 min-h-full bg-gray-800">

        <div>
          <TextAreaChat
:chats="chats"></TextAreaChat>
        </div>

        <InputChat class="max-w-full pl-6"
:chats="chats" @loadingResponse="loadingResponse = true">

          </InputChat>

        </div>

      </div>
    </ChatLayout>
</template>

<script>
import ChatLayout from '@/Layouts/ChatLayout.vue';
import { Head, router } from '@inertiajs/vue3';
import InputChat from '@/Components/chat/InputChat.vue';
```

```

import TextAreaChat from
'@/Components/chat/TextAreaChat.vue';
import SideBarChat from "@/Components/chat/SideBarChat.vue";

export default {

  components: {
    SideBarChat,
    InputChat, Head, ChatLayout, TextAreaChat
  },

  props: [
    'chats', 'history' , 'allChats'
  ],
  data() {
    return {
      loadingResponse: false
    };
  },
  updated() {
    this.loadingResponse = false;
  }
}
</script>

```

Vamos a proceder a explicar este archivo en detalle:

1º Script:

```

<script>
import ChatLayout from '@/Layouts/ChatLayout.vue';
import { Head, router } from '@inertiajs/vue3';
import InputChat from '@/Components/chat/InputChat.vue';
import TextAreaChat from '@/Components/chat/TextAreaChat.vue';
import SideBarChat from "@/Components/chat/SideBarChat.vue";

```

```

export default {

```

```

  components: {
    SideBarChat,
    InputChat, Head, ChatLayout, TextAreaChat
  },

```

```

  props: [

```

```

      'chats', 'history' , 'allChats'
    ],
    data() {
      return {
        loadingResponse: false
      };
    },
    updated() {
      this.loadingResponse = false;
    }
  }
}
</script>

```

Import: Para poder extender de un archivo o contener otros componentes, es decir hacer uso de estos, es necesario su importación (el import Head se utiliza para cambiar el Html, y router se utiliza para llamar a las rutas definidas en el back-end).

Export default: Esta declaración se utiliza para exportar el componente, permitiendo así ser importado y utilizado por otros archivos.

Components: Aquí se definen los componentes hijos que se utilizan dentro del componente actual.

Prop: En esta sección se declaran las propiedades que el componente espera recibir. Pueden ser pasadas al componente desde su componente padre, o desde el back-end de la aplicación (en este archivo son todas pasadas desde el back-end).

Data(): Aquí se define las variables que puede utilizar el componente. En este caso, se ha definido una propiedad loadingResponse con el valor inicial de false. Esta propiedad se utiliza para controlar el estado de carga de respuestas de la API.

Updated(): Es un método de Vue que se ejecuta después de que el componente se actualice. En este caso, se ha utilizado para establecer loadingResponse en false después de una actualización del componente.

2º Template:

```

<template>
  <Head title="Chat" />
  <ChatLayout>

    <div class="flex">

      <div class="w-1/6 bg-gray-800 border-4 border-gray-600">
        <SideBarChat :allChats="allChats" :chat="chats"
        :loadingResponse="loadingResponse"></SideBarChat>
      </div>
    </div>
  </ChatLayout>
</template>

```

```
</div>
```

```
<div class="w-5/6 min-h-full bg-gray-800">
```

```
<div>
```

```
<TextAreaChat :chats="chats"></TextAreaChat>
```

```
</div>
```

```
<InputChat class="max-w-full pl-6" :chats="chats"  
@loadingResponse="loadingResponse = true">
```

```
</InputChat>
```

```
</div>
```

```
</div>
```

```
</ChatLayout>
```

```
</template>
```

```
<SideBarChat :allChats="allChats" :chat="chats"  
:loadingResponse="loadingResponse"></SideBarChat>
```

<SideBarChat>: Esta etiqueta representa la instancia del componente SideBarChat que se va a renderizar en la interfaz de usuario.

:allChats="allChats": Aquí se establece una vinculación de propiedades (props) entre el componente padre y el componente SideBarChat. Esto permite que el componente hijo acceda y utilice el valor de la propiedad allChats en su lógica interna.

:chat="chats": De manera similar al caso anterior, este atributo establece la vinculación de propiedades para la propiedad chat.

:loadingResponse="loadingResponse": Este atributo establece la vinculación de propiedades para la propiedad loadingResponse.

```
<TextAreaChat :chats="chats"></TextAreaChat>
```

<TextAreaChat>: Esta etiqueta indica la instancia del componente TextAreaChat que se renderizará en la interfaz de usuario.

:chats="chats": Aquí se establece una vinculación de propiedades entre el componente padre y el componente TextAreaChat.

<InputChat class="max-w-full pl-6" :chats="chats" @loadingResponse="loadingResponse = true">

<InputChat>: Esta etiqueta indica la instancia del componente InputChat que se renderizará en la interfaz de usuario. El nombre InputChat se refiere al componente registrado y disponible para su uso en el contexto actual.

:chats="chats": Se establece una vinculación de propiedades entre el componente padre y el componente InputChat.

@loadingResponse="loadingResponse = true": Aquí se establece un listener de eventos en el componente InputChat. El evento loadingResponse se dispara en el componente hijo y se asigna true a la propiedad loadingResponse del componente padre cuando se produce dicho evento. Esto permite actualizar el estado del componente padre en respuesta a la interacción del componente hijo.

InputChat.vue:

```
<template>
  <div class="bg-gray-800 min-w-full">

    <form @submit.prevent="submit">
      <div class="flex" v-if="!loadingResponse">
        <input type="text" placeholder="Introduce tu
pregunta" class="break-words h-auto rounded w-5/6"
v-model="message">
        <button type="submit" class="ml-2 btn
btn-active glass w-1/6 w-40">Enviar</button>
      </div>
      <div v-else class="flex justify-center">
        <button class="ml-2 btn loading
btn-active glass w-40">cargando</button>
      </div>
    </form>

  </div>
</template>

<script>
import {router} from "@inertiajs/vue3";

export default {
  props: ['chats'],
```

```

data() {
  return {
    message: '',
    loadingResponse: false,
  }
},

watch: {
  chats() {
    this.loadingResponse = false;
  }
},

methods: {
  submit() {

    if(this.message != '') {
      this.loadingResponse = true;
      this.$emit('loadingResponse')
      var chat_id = null;

      if(this.chats){
        chat_id = this.chats.id
      }

      var data = {message: this.message , chats_id :
chat_id};

      router.post('chat', data, {
        onCancel: () => this.loadingResponse =
false,
        onError: () => this.loadingResponse =
false,
      });

      this.message = '';
    }
  },
},
},

```

Como ya se ha explicado con anterioridad, un archivo similar, procederemos a explicar funcionalidades y fragmento de código no visto aún.

1º Script:

<script>


```

import {router} from "@inertiajs/vue3";

export default {
  props: ['chats'],
  data() {
    return {
      message: "",
      loadingResponse: false,
    }
  },

  watch: {
    chats() {
      this.loadingResponse = false;
    }
  },
  methods: {
    submit(){

      if(this.message !== "") {
        this.loadingResponse = true;
        this.$emit('loadingResponse')
        var chat_id = null;

        if(this.chats){
          chat_id = this.chats.id
        }

        var data = {message: this.message , chats_id : chat_id};

        router.post('chat', data, {
          onCancel: () => this.loadingResponse = false,
          onError: () => this.loadingResponse = false,
        });

        this.message = "";
      }
    }
  }
}

```

```

    },
  },
}
</script>

```

watch: La propiedad watch se utiliza en Vue.js para observar cambios en las propiedades o datos del componente y ejecutar código en respuesta a esos cambios.

chats(): Aquí se define la propiedad chats que se va a observar. Cuando se detecte un cambio en la propiedad **chats**(la cual procede del elemento padre, es decir de los **props**), se ejecutará el código asociado.

this.loadingResponse = false; Dentro de la función de watch, se establece this.loadingResponse en false.

methods: La propiedad methods se utiliza en Vue.js para definir métodos que se pueden llamar desde la plantilla o desde otros métodos del componente.

submit(): Aquí se define el método submit() que se ejecuta cuando se envía el formulario.

if (this.message != ""): Se realiza una comprobación para asegurarse de que el campo de mensaje no está vacío.

this.loadingResponse = true; Se establece this.loadingResponse en true para indicar que se está cargando una respuesta.

this.\$emit('loadingResponse'); Se emite un evento personalizado llamado loadingResponse para notificar a los componentes padre que se ha iniciado la carga de respuesta.

var chat_id = null; Se inicializa una variable chat_id como null.

if (this.chats) { chat_id = this.chats.id; }: Se verifica si la propiedad chats existe en el componente y, de ser así, se asigna el valor de chats.id a chat_id. Esto permite obtener el ID del chat al que se enviará el mensaje.

var data = { message: this.message, chats_id: chat_id }; Se crea un objeto data que contiene la información del mensaje y el ID del chat.

router.post('chat', data, {

onCancel: () => this.loadingResponse = false,

onError: () => this.loadingResponse = false,

}); Se realiza una solicitud HTTP POST utilizando el objeto data y enviándolo a la ruta 'chat' del enrutador. Esto implica enviar el mensaje y el ID del chat al backend para su procesamiento, y en caso de que ocurra un error o se cancele la petición se resetea a false el valor de loadingResponse.

this.message = ""; Después de enviar el mensaje, se restablece this.message a una cadena vacía para borrar el contenido del campo de entrada.

2º Template:

<template>

<div class="bg-gray-800 min-w-full">

<form @submit.prevent="submit">

<div class="flex" v-if="!loadingResponse">

<input type="text" placeholder="Introduce tu pregunta"

class="break-words h-auto rounded w-5/6" v-model="message">

<button type="submit" class="ml-2 btn btn-active glass w-1/6 w-40">Enviar</button>

</div>

<div v-else class="flex justify-center">

<button class="ml-2 btn loading btn-active glass w-40">loading</button>

</div>

</form>

</div>

</template>

<form @submit.prevent="submit"> Se establece un formulario que escucha el evento de envío (submit). Al utilizar @submit.prevent, se evita el comportamiento predeterminado de recargar la página al enviar el formulario y ejecuta el método submit().

<div class="flex" v-if="!loadingResponse"> Se muestra un contenedor div si loadingResponse es falso.

<input type="text" placeholder="Introduce tu pregunta" class="break-words h-auto rounded w-5/6" v-model="message"> La propiedad v-model establece la

vinculación bidireccional entre el valor del campo de entrada y la propiedad message del componente, es decir el bindeo.

<div v-else class="flex justify-center">: Se muestra otro contenedor div si loadingResponse es verdadero.

4. Back-end de la aplicación: Laravel

Debido a que en clase se ha abordado el framework Laravel, en esta sección de la documentación se hará mención y explicación de aquellos aspectos o componentes que no se han cubierto durante el curso.

Se proporcionará información adicional sobre funcionalidades, técnicas o herramientas específicas utilizadas en el proyecto, que van más allá del alcance de los contenidos del curso.

Primero que nada para la realización de las peticiones a la API de ChatGPT, se ha utilizado la siguiente librería: <https://github.com/orhanerday/open-ai>.

Con esta documentación hemos aprendido cómo funciona la petición a la API y cómo poder hacer nuestras propias peticiones.

Para realizar dichas peticiones hemos realizado un método dentro de nuestro **ChatController.php**:

```
public function apiRequestResponse($request) {
    $user = auth()->user();

    $open_ai_key = getenv('OPENAI_API_KEY');
    $open_ai = new OpenAi($open_ai_key);
    $messages = [];

    if($user->history) {

        $history = $user->history;
        $chat = $history->chats()->find($request->chats_id);
        $datas = json_decode($chat->data);

        foreach($datas as $data) {
            $messages[] = [
                "role" => "user",
                "content" => $data->request
            ];
        }

        $messages[] = [
```

```

        "role" => "assistant",
        "content" => $data->response
    ];
}

}

$messages[] = [
    "role" => "user",
    "content" => $request["message"]
];

$result = $open_ai->chat([
    'model' => getenv('OPENAI_API_MODEL'),
    'messages' => $messages,
    'temperature' => 1.0,
    'max_tokens' => 1000,
    'frequency_penalty' => 0,
    'presence_penalty' => 0,
]);

return json_decode($result);
}

```

Este método recibe una request como parámetro, mediante la cual vamos a recibir la consulta del usuario.

Obtiene del fichero de variables de entorno la clave privada de la API de ChatGPT, la cual es usada para crear una nueva instancia de la clase importada con el paquete anterior (OpenAi, cuyo constructor establece los header con el token de la API que se le ha pasado como parámetro y el tipo de contenido que va a tener la petición (json)), comprueba si el usuario posee un historial y en caso afirmativo, almacena los mensajes previos del chat en el que se está produciendo la petición, y así la inteligencia artificial sea capaz de acceder a las preguntas hechas con anterioridad para que sea cada vez más precisas sus respuestas con preguntas correlacionadas entre sí y una vez hecho esto tanto si posee historial como si no añade la consulta recibida en la request y se termina de construir la petición con las opciones que se le pasan como parámetro en el método **chat()** y se envía la petición.

Index.php:

```

public function index(Request $request)
{
    $user = auth()->user();
    $history = null;
    $chat = null;
    $allChats = null;
    $chatId = null;

```

```

$validated = $request->validate([
    'id' => 'exists:App\Models\Chat,id'
]);

if(isset($validated['id'])){
    $chatId = $validated['id'];
}

if($user->history){
    $history = $user->history;

    if($chatId) {
        $chat = Chat::find($chatId);
    } else {
        $chat = Chat::where('history_id',
$history->id)->latest()->first();
    }

    $chat->data = json_decode($chat->data);

    $allChats = $history->chats;

}

return Inertia::render('Chat', [
    "history" => $history,
    "chats" => $chat,
    "allChats" => $allChats
]);
}

```

Como ya se ha visto en clase durante el curso solo queríamos explicar que para utilizar las vistas de Vue, ha sido necesario la instalación del paquete InertiaJs, y mediante el método render, es capaz de renderizar la vista de Vue asignándole a ésta las propiedades que se le pasan como array asociativo.

5. Testing (PHPUnit)

Los tests son una parte fundamental en el desarrollo de software, ya que permiten verificar el funcionamiento correcto de las distintas funcionalidades y aseguran la calidad del código.

A través de estos tests, se podrá evaluar la robustez y confiabilidad del sistema, identificar posibles errores o fallos, y facilitar el mantenimiento y la evolución del proyecto en el futuro.

Tal como se mencionó con anterioridad, PHPUnit es el framework elegido para llevar a cabo las pruebas de nuestro código, como se ha visto también a lo largo de nuestro curso, en esta sección de la documentación proporcionaremos una explicación de algunos tests relevantes para el proyecto.

```
public function test_user_can_chat_with_the_ai(): void
{
    $this->signIn();

    $response = $this->post('/chat', [
        'message' => 'Hola que tal?'
    ]);

    $user = User::first();
    $history = $user->history;
    $chats = $history->chats;

    $response->assertInertia(fn (Assert $page) => $page
        ->where('chats.id', $chats[0]->id)
        ->where('chats.data', json_decode($chats[0]->data,
true))));
}
```

De este test cabe destacar que se emplea el método **assertInertia()**, para verificar la respuesta de una solicitud cuando el encargado del renderizado de la vista es el framework InertiaJs, la función anónima que recibe como parámetro recibe a su vez un objeto de tipo Assert que permite realizar aserciones más concretas en la vista.

```
public function test_user_can_change_the_chat_view(): void
{
    $this->signIn();

    $user = User::first();
    $history = History::factory()->create(['user_id' =>
$user->id]);

    $chat1 = Chat::factory()->create(['history_id' =>
$history->id]);
    $chat2 = Chat::factory()->create(['history_id' =>
$history->id]);

    $response = $this->actingAs($user)->get('/chat?id=' .
$chat1->id);
}
```

```

$response->assertInertia(fn (Assert $page) => $page
    ->where('chats.id', $chat1->id)
    ->whereNot('chats.id', $chat2->id)
);

$response = $this->actingAs($user)->get('/chat?id=' .
$chat2->id );

$response->assertInertia(fn (Assert $page) => $page
    ->where('chats.id', $chat2->id)
    ->whereNot('chats.id', $chat1->id)
);
}

```

De este test cabe destacar el uso del método **actingAs()**, para obligar a actuar al test como el usuario debido a que por un problema de PHPUnit cuando se realizan varias peticiones post en un mismo test nos creaba registros distintos a la hora de almacenar en la base de datos debido a que no era capaz de entender bien las relaciones entre los modelos.

6. Despliegue de la Aplicación

Para la parte de despliegue se ha empleado una herramienta denominada Laravel Forge.

Laravel Forge es una plataforma en la nube que proporciona herramientas y servicios para facilitar el aprovisionamiento, configuración y administración de servidores web para aplicaciones Laravel.

Para ello previamente hemos accedido y registrado en DigitalOcean para contratar un servidor en la nube.

Hemos ingresado al sitio web oficial de DigitalOcean, reconocido proveedor de servicios en la nube. Con el propósito de alojar y gestionar nuestra aplicación, hemos completado el proceso de registro para crear una cuenta en la plataforma.

Una vez hecho esto, hemos generado un token, que permite el acceso y manipulación de los servicios de DigitalOcean mediante la aplicación de Laravel Forge.

×

New personal access token

Token name

Enter token name

token-Chatles

✓

Expiration

Select token expiry

60 days

▼

Select scopes

☒ Read (default)
 ☒ Write (optional)

Read our [personal access token documentation](#) for more information on scopes.

Generate Token

Una vez hecho esto, ya procedemos a trabajar sobre Laravel Forge.

Una vez registrado en Laravel Forge, nos pide enlazar nuestra cuenta de Github e introducir el token generado anteriormente, posteriormente hemos seleccionado la opción de nuevo servidor y nos ha salido un menú con el cual podemos configurar las opciones que mejor nos convenga para montar nuestro propio servidor.

DigitalOcean

Akamai

AWS

Vultr

HETZNER

Custom VPS

Everything you need to deploy your PHP / Laravel application.

Deploy your app in minutes with this all-in-one provisioned server.

PHP

Ngix

Database

Redis

Memcached

Mellicsearch

Name

Chatles

Type

App Server

Region

Frankfurt

Server Size

4GB RAM (High CPU) - 2 CPU Cores - 25GB SSD

Private Network

laravel-forge-fra1

Server OS

Ubuntu 22.04 LTS (Jammy)

PHP Version

8.1

ⓘ Ubuntu 22.04 includes OpenSSL 3.0.0, which is only supported by PHP 8.1 and later. If you require an older version of PHP, you should [provision an Ubuntu 20.04 server](#).

Database

MySQL 8

Database Name

laravel

☒ Add Server's SSH Key To Source Control Providers ⓘ
 ☐ Enable DigitalOcean Weekly Backups

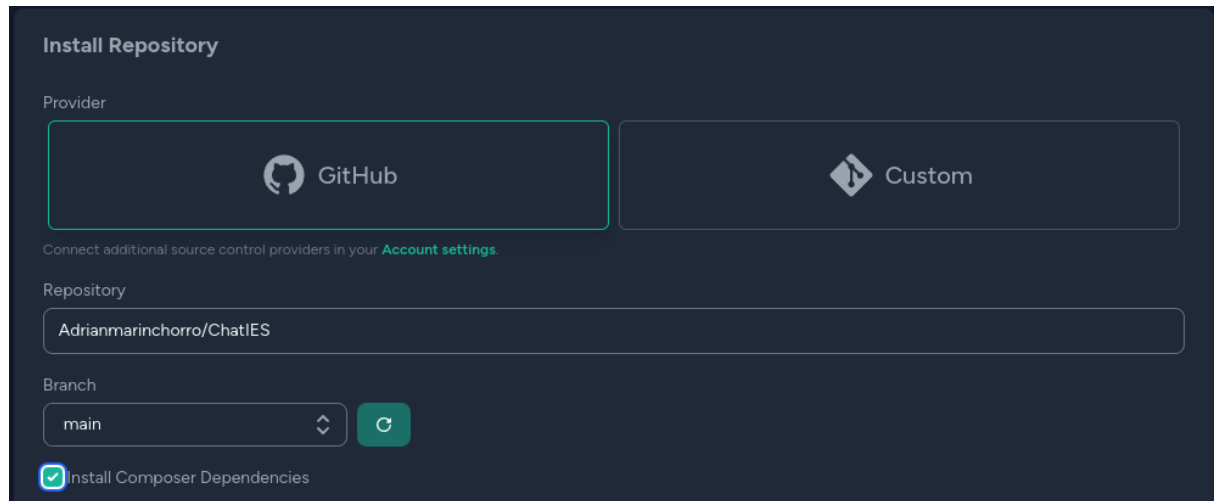
Cancel

Hide Advanced Settings

Create Server

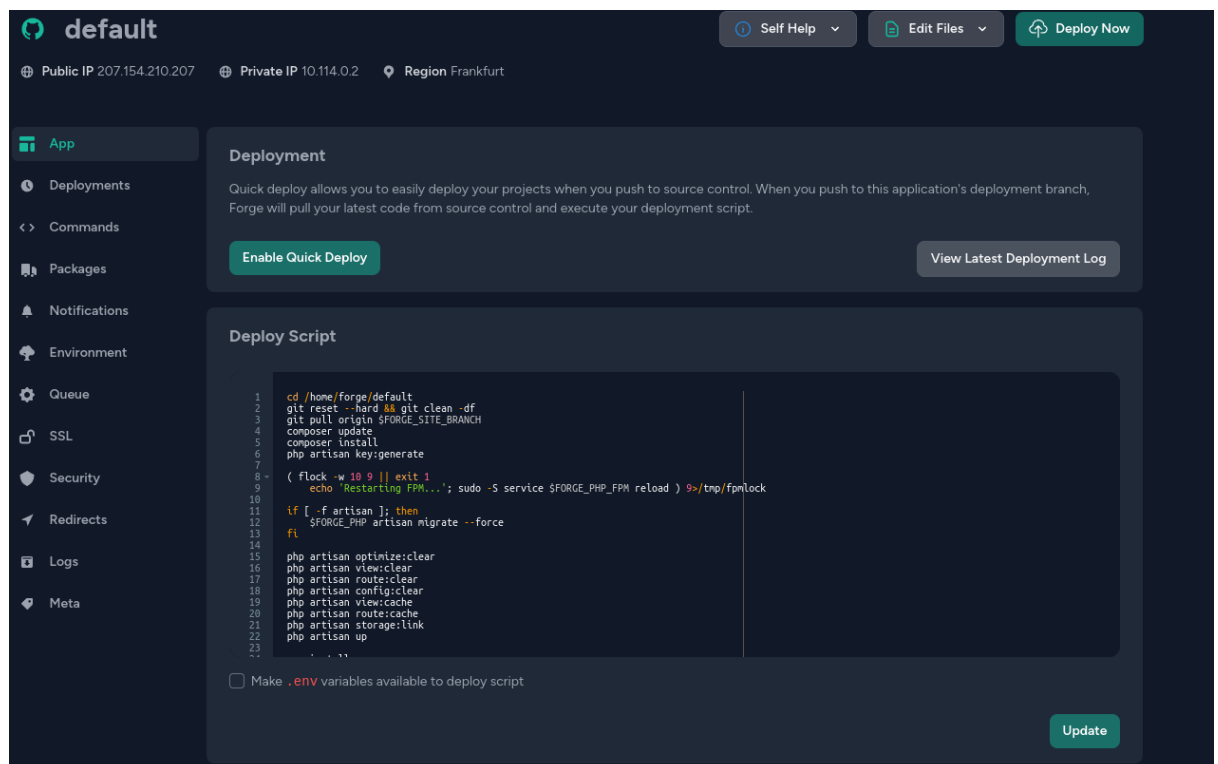
Una vez generado el servidor, el servicio de Laravel Forge, nos entrega las claves de acceso al servidor y a su respectiva base de datos.

Una vez finalizado el proceso de creación, tenemos que configurar nuestro sitio web, Laravel Forge nos crea uno por defecto, en nuestro caso le hemos añadido un dominio, y posteriormente hemos enlazado el proyecto con el sitio web, eligiendo la rama de producción.



The screenshot shows the 'Install Repository' form in Laravel Forge. It has a dark theme. At the top, there's a title 'Install Repository'. Below it, under the 'Provider' section, there are two buttons: 'GitHub' (highlighted with a green border) and 'Custom'. A link below says 'Connect additional source control providers in your Account settings.' The 'Repository' section has a text input field containing 'Adrianmarinchorro/ChatIES'. The 'Branch' section has a dropdown menu showing 'main' and a refresh button. At the bottom, there's a checkbox labeled 'Install Composer Dependencies' which is checked.

Como tuvimos dificultades a la hora de instalar las dependencias del composer, nosotros personalizamos el script del despliegue para poder instalar estas.



The screenshot shows the 'default' application configuration page in Laravel Forge. It has a dark theme. At the top, there's a header with 'default' and buttons for 'Self Help', 'Edit Files', and 'Deploy Now'. Below the header, there's a section for 'App' with a sidebar menu containing 'Deployments', 'Commands', 'Packages', 'Notifications', 'Environment', 'Queue', 'SSL', 'Security', 'Redirects', 'Logs', and 'Meta'. The main content area has a 'Deployment' section with a description and an 'Enable Quick Deploy' button. Below that is a 'Deploy Script' section with a code editor containing a shell script. At the bottom, there's a checkbox for 'Make .env variables available to deploy script' and an 'Update' button.

```
1 cd /home/forge/default
2 git reset --hard $GIT_SITE_BRANCH
3 git pull origin $GIT_SITE_BRANCH
4 composer update
5 composer install
6 php artisan key:generate
7
8 ( flock -w 10 0 || exit 1
9   echo "Restarting FPM..." ; sudo -S service $FORGE_PHP_FPM reload ) && /tmp/fpmlock
10
11 if [ -f artisan ]; then
12   $FORGE_PHP artisan migrate --force
13 fi
14
15 php artisan optimize:clear
16 php artisan view:clear
17 php artisan route:clear
18 php artisan config:clear
19 php artisan view:cache
20 php artisan route:cache
21 php artisan storage:link
22 php artisan up
23
```

```
Deploy Script

1 git reset --hard $(git rev-parse --short HEAD)
2 git pull origin $FORCE_SITE_BRANCH
3 composer update
4 composer install
5 php artisan key:generate
6
7
8 ( flock -w 10 9 || exit 1
9   echo 'Restarting FPM...'; sudo -S service $FORCE_PHP_FPM reload ) 9>/tmp/fpmlock
10
11 if [ -f artisan ]; then
12   $FORCE_PHP artisan migrate --force
13 fi
14
15 php artisan optimize:clear
16 php artisan view:clear
17 php artisan route:clear
18 php artisan config:clear
19 php artisan view:cache
20 php artisan route:cache
21 php artisan storage:link
22 php artisan up
23
24 npm install
25 npm run build

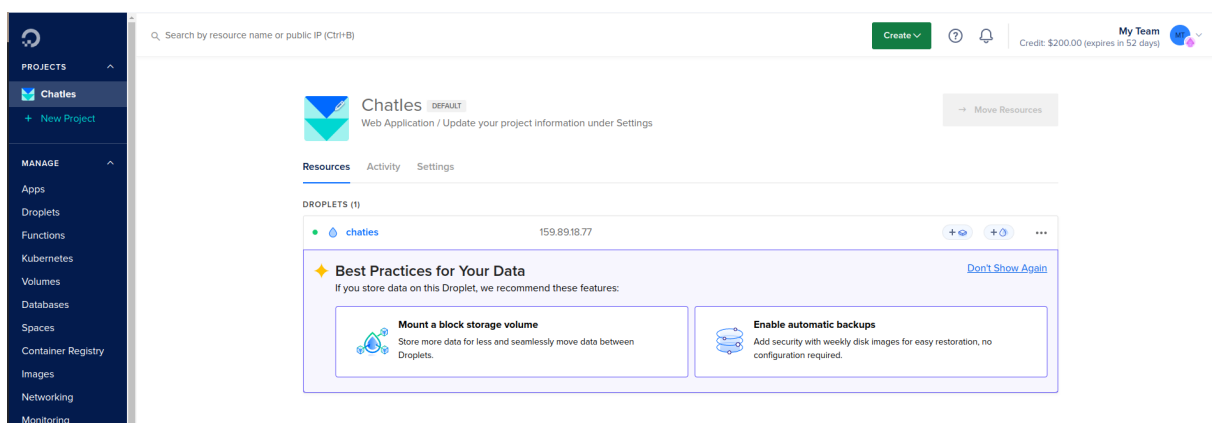
☐ Make .env variables available to deploy script

Update
```

Una vez hecho esto le damos a desplegar y como podemos observar somos capaces de acceder a nuestra aplicación por la url de nuestro dominio.

Pero no tiene protocolo seguro (HTTPS), para ello hemos tenido que acceder a las configuraciones del sitio web, y nos hemos introducido en la opción **ssl**, y ahí es donde nos permite escoger la encriptación y obtener así el protocolo seguro de HTTPS.

Ahora vamos a la página web de DigitalOcean y en el panel control podemos acceder a nuestro servidor que se ha encargado de crear y configurar Laravel Forge.



Seleccionamos nuestro servidor y en las opciones somos capaces de escoger **Access**, la cual nos permite desplegar una terminal y acceder a través de esta a nuestro servidor.

Una vez aquí, hemos procedido a instalar OpenLDAP en el servidor, pero debido a ciertas incompatibilidades de las nuevas versiones de OpenLDAP hemos tenido que instalar diferente paquete en el servidor con respecto a los contenedores locales.

Mediante este comando hemos procedido a instalar los paquetes de ldap así como sus utilidades para trabajar con servidores y clientes.

sudo apt install slapd ldap-utils -y

Una vez completada la instalación nos pide la contraseña de administrador que será la cuenta que gestione el servicio LDAP.

Luego hemos ejecutado el siguiente comando:

sudo dpkg-reconfigure slapd

El cual nos permite configurar las distintas opciones que nos aporta el servicio LDAP.

Y por último hemos instalado un servicio web, para poder gestionar el servicio LDAP.

sudo apt -y install ldap-account-manager

Y para finalizar, nos vamos a Laravel Forge y en el fichero de nginx añadimos la siguiente línea:

include /etc/ldap-account-manager/nginx.conf;

```
# FORGE CONFIG (DO NOT REMOVE!)
include forge-conf/chaties.marmol89.com/before/*;

server {
    listen 443 ssl http2;
    listen [::]:443 ssl http2;
    server_name chaties.marmol89.com;
    server_tokens off;
    root /home/forge/chaties.marmol89.com/public;

    # FORGE SSL (DO NOT REMOVE!)
    ssl_certificate
    /etc/nginx/ssl/chaties.marmol89.com/1808635/server.crt;
    ssl_certificate_key
    /etc/nginx/ssl/chaties.marmol89.com/1808635/server.key;

    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_ciphers
    ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384;
    ssl_prefer_server_ciphers off;
    ssl_dhparam /etc/nginx/dhparams.pem;
```

```

add_header X-Frame-Options "SAMEORIGIN";
add_header X-XSS-Protection "1; mode=block";
add_header X-Content-Type-Options "nosniff";

index index.html index.htm index.php;

charset utf-8;

# FORGE CONFIG (DO NOT REMOVE!)
include forge-conf/chaties.marmol89.com/server/*;

include /etc/ldap-account-manager/nginx.conf;

location / {
    try_files $uri $uri/ /index.php?$query_string;
}

location = /favicon.ico { access_log off; log_not_found off;
}
location = /robots.txt  { access_log off; log_not_found off;
}

access_log off;
error_log  /var/log/nginx/chaties.marmol89.com-error.log
error;

error_page 404 /index.php;

location ~ /\.php$ {
    fastcgi_split_path_info ^(.+\.(php))(/.+)$;
    fastcgi_pass unix:/var/run/php/php8.2-fpm.sock;
    fastcgi_index index.php;
    include fastcgi_params;
}

location ~ /\.(!well-known).* {
    deny all;
}

# FORGE CONFIG (DO NOT REMOVE!)
include forge-conf/chaties.marmol89.com/after/*;

```

Después de hacer esto reiniciamos el servicio de Nginx, desde Laravel Forge y procedemos a configurar el servicio web recién instalado.

Configurando lo necesario para que se conecte el nuevo LDAP a nuestra aplicación web emulando las configuraciones que tenía nuestro servicio LDAP de docker.

Para ello nos vamos a LAM configuración y escogemos las opciones del perfil del servidor y ahí modificamos lo siguiente dentro de general settings:

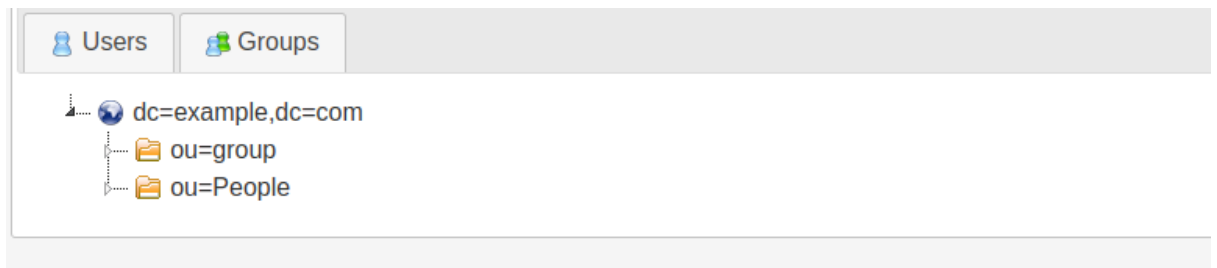
The screenshot shows two sections of a configuration interface. The top section, 'Tool settings', is titled with a wrench icon and contains a 'Hidden tools' area with a grid of checkboxes for 'Multi edit', 'Schema browser', 'File upload', 'Server information', 'PDF editor', 'WebAuthn devices', 'LDAP import/export', 'OU editor', 'Profile editor', 'Tests', and 'Tree view'. Below this is a 'Tree view' section with a 'Tree suffix' field containing 'dc=example,dc=com'. The bottom section, 'Security settings', is titled with a lock icon and contains a 'Login method' dropdown set to 'Fixed list' and a 'List of valid users' field containing 'cn=admin,dc=example,dc=com'.

Ahora nos vamos a la configuración de account types y ponemos lo siguiente:

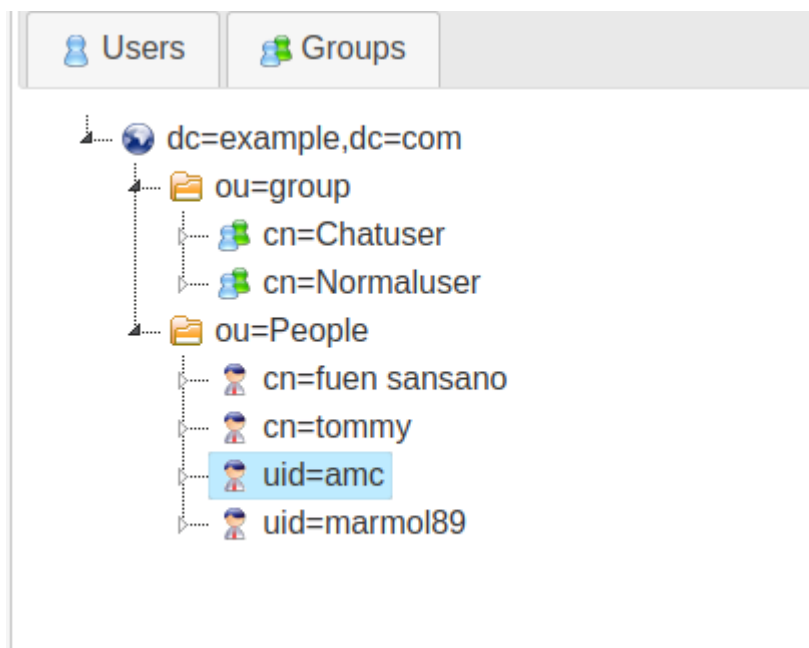
The screenshot shows the 'Active account types' section, which is divided into two main parts: 'Users' and 'Groups'. Each part has a list of configuration fields. For 'Users', the fields are: 'LDAP suffix' (ou=People,dc=example,dc=com), 'List attributes' (#uid;#givenName;#sn;#uidNumber;#gidNumber), 'Custom label' (empty), 'Additional LDAP filter' (empty), and 'Hidden' (unchecked). For 'Groups', the fields are: 'LDAP suffix' (ou=group,dc=example,dc=com), 'List attributes' (#cn;#gidNumber;#memberUID;#description), 'Custom label' (empty), 'Additional LDAP filter' (empty), and 'Hidden' (unchecked). Each field has a help icon to its right.

Guardamos los cambios realizados y hemos finalizado.

Una vez terminado con esto, nos logueamos con la contraseña que pusimos al configurar LDAP y nos pedirá crear unas organizaciones base:



Y a partir de ahí podemos crear nuestro grupo y con este nuestros usuarios:



Y con estos usuarios y grupo (Chatuser) creado podemos iniciar sesión en nuestra aplicación web.