Aplicación de Suscripciones Compartidas para Productos y Servicios.

REDES DE COMPUTADORAS III

-Michael Giovanny Miguel Padilla. -Gerardo Castañeda Martin. -Ana Paulina Martinez Luevano. -Ahylin Aketzali Castorena Rodriguez

Joinify es una plataforma diseñada con el propósito de administrar mejor el uso de suscripciones compartidas, desde crear grupos que permitan utilizar un mismo servicio y compartir los privilegios, así como los costos, de esta manera se puede tener un mejor control de aquellos servicios a los que esta suscrito y es una opción más barata ya que compartes los gastos con otros usuarios y es accesible ya que puedes acceder desde cualquier dsipositivo a la página web o a la aplicación movil y usando cualquier tipo de sistema operativo ya sea Ubuntu o Windows.

I. Introduction

En la actualidad, muchas personas buscan maneras de reducir los costos de servicios que operan bajo un modelo de suscripción. Esta propuesta presenta una plataforma de suscripciones compartidas que permite a los usuarios dividir el costo de productos y servicios recurrentes, como membresías de gimnasios y servicios de streaming. La implementación de este sistema no solo beneficiará a los usuarios en términos de ahorro, sino que también proporcionará una comprensión más profunda de los conceptos de sistemas distribuidos, así como diferentes herramientas y conceptos vistos a lo largo de este semestre como apis, solicitudes POST y GET así como la conexión de servidores con páginas web y aplicaciones móviles. Este proyecto integrará servicios web externos, almacenamiento distribuido y mecanismos de seguridad para crear una solución robusta y eficiente que nos proporcione un producto final que pueda cumplir con todo lo establecido en este proyecto.

II. Integración de Servicios Web

La aplicación integra servicios externos para facilitar los pagos y la gestión de suscripciones. Por ejemplo, utilizamos apis de plataformas de pago (como Stripe) para gestionar transacciones financieras, mientras que otra API podría obtener datos relevantes de un proveedor de servicios (como Spotify o Netflix) [1], verificando los permisos de usuario y gestionando los detalles de suscripciones

III. Estructura de Almacenamiento

El componente de almacenamiento de datos se divide en 6 tablas las cuales son: usuarios, suscripciones, usuario_grupo, notificación, historial_pagos y servicio_suscripción las cuales fueron implementadas como archivos de texto plano para gestionar todos los datos de cada tabla y simular de esta manera una base de datos que nos permita tener un control sobre las acciones y los datos del usuario en nuestro sistema. Estos archivos se almacenarán en dos servidores: un servidor NFS local que esta montado en una maquina virtual de Ubuntu server y desde esta administraremos los resultados obtenidos en cada interaccion del usuario con nuestro sistema desde cualquier plataforma ya sea atravez de la pagian web o desde la aplicación movil.

IV. Seguridad de los datos

La plataforma protegera los datos proporcionados por los usuarios por medio de la encriptacion de los archivos del servidor NFS, La aplicación implementará encriptación por medio del uso de librerias que nos permitan proteger de manera facil y bien estructurada toda la información sensible almacenada, como credenciales de usuario. Los archivos encriptados asegurarán que solo la aplicación pueda desencriptar y presentar datos a los usuarios, protegiendo la información sensible contra accesos no autorizados.

V. Acceso Multiplataforma

El proyecto soportará una aplicación web y una aplicación móvil, permitiendo a los usuarios gestionar y visualizar sus suscripciones desde distintos dispositivos. El backend estará alojado en un servidor Ubuntu que está montado en una máquina virtual que posee las características necesarias para permitir al servidor conectarse a nuestra página web y a la aplicación móvil, además de permitir y gestionar con eficacia las solicitudes POST y GET que se realizan al servidor, incluso desde diferentes sistemas operativos como Windows permitiendo así que sin importar la

PLATAFORMA O EL SISTEMA DISTRIBUIDO DEL USUARIO EL SERVIDOR SINCRONIZARÁ DATOS ENTRE CLIENTES Y PROPORCIONARÁ UNA INTERFAZ CENTRALIZADA PARA ACTUALIZACIONES Y NOTIFICACIONES

VI. Resultados

En el proyecto hemos logrado cumplir con los objetivos mencionados, proporcionando una plataforma eficiente que facilita la administración de suscripciones compartidas. A través de la implementación de sistemas distribuidos, servicios web externos y almacenamiento distribuido, la aplicación ofrece una solución para gestionar servicios de suscripciones de una manera colaborativa y accesible.

Optimización del uso de suscripciones

Joinify permite a los usuarios crear y administrar grupos para compartir los costos de servicios como plataformas de streaming, y otros servicios bajo modelos de suscripción. De esta manera, los usuarios pueden reducir significativamente sus gastos mensuales mientras mantienen acceso completo a los beneficios del servicio.

Accesibilidad multiplataforma

La plataforma está diseñada para ser compatible con múltiples dispositivos y sistemas operativos, como Ubuntu y Windows, y es accesible tanto desde una aplicación móvil como desde una página web. Esto garantiza una experiencia de usuario eficiente, sin importar la plataforma utilizada.

Integración de servicios web externos

Para facilitar la gestión de pagos y suscripciones, se han integrado APIs externas:

- APIs de pagos seguros (Stripe) esta API se implemento para gestionar las transacciones financieras de manera segura y eficiente, a travez de ellas los usuarios pueden realizar pagos sin riesgos, garantizando la proteccion de los datos personales y financieros mediante mecanismo de autenticacion.
- TMDB API. (The Movie Database) se utilizo para mostrar informacion relevantesobre series y peliculas, como titulos, imágenes, sinopsis y detalles populares. esta API nos permitio mantener a los usuarios informados sobre los contenidos disponibles.

Estas integraciones permiten automatizar procesos clave y garantizar una experiencia segura y transparente para los usuarios.

Almacenamiento distribuido y seguro

Los datos de la plataforma se almacenan utilizando un sistema de archivos distribuido (DFS) implementado en dos servidores:

- Un servidor NFS local, donde los datos se gestionan y replican.

Además, los archivos almacenados son cifrados y descifrados por la aplicación, lo que asegura la protección de la información y cumple con los estándares de seguridad.

Aplicación de conceptos de sistemas distribuidos

El desarrollo del proyecto ha permitido la aplicación práctica de conceptos clave vistos durante el semestre, como:

- Uso de apis restful para enviar solicitudes POST y GET.
- Integración de servidores con páginas web y aplicaciones móviles.

Aquí, para la conexión con el servidor fue desarrollado un frontend en Angular, donde se implementaron métodos en los archivos .ts que permitían la conexión con el servidor a través de solicitudes HTTP usando el módulo HTTPClient. Estos métodos realizaban peticiones de tipo GET, POST, PUT y DELETE hacia las API expuestas en el servidor, permitiendo una comunicación para gestionar diferentes operaciones. En una solicitud POST se enviaban datos correspondientes a través del cuerpo de la solicitud y se procesaban las respuestas recibidas para actualizar la interfaz o manejar posibles errores. De igual forma, se empleaban las peticiones GET para recuperar datos necesarios para la visualización de información, PUT para actualizar registros existentes, y DELETE para eliminar recursos.

Parte del codigo empleado para la conexion con el servidor:

Solicitud:

```
this.http.post('http://192.168.50.20:3001/usuario', userData)
.subscribe(
  (response: any) => {
    console.log('Registro exitoso', response);
    // Redirigir al usuario a una página de éxito o login
    this.router.navigate(['/login']); // Esto es solo un ejemplo

  },
  (error) => {
    console.error('Error al registrar el usuario', error);
    // Aquí puedes manejar errores si es necesario
  }
);
```

ENDPOINT EN EL SERVIDOR:

```
// Nuevo usuario con cifrado de contraseña
app.post('/usuario', async (req, res) => {
    const nuevoUsuario = req.body;

    // Encriptar nombre y correo
    const nombreEncriptado = encryptData(nuevoUsuario.fullname);
    const correoEncriptado = encryptData(nuevoUsuario.email);

    // Encriptar la contraseña con bcrypt
    const hashedPassword = await encryptPassword(nuevoUsuario.password);

    const usuarios = JSON.parse(fs.readFileSync(USERS_FILE));
    nuevoUsuario.id = usuarios.length + 1; // Asigna un nuevo ID
    nuevoUsuario.fullname = nombreEncriptado; // Guarda el nombre encriptado
    nuevoUsuario.email = correoEncriptado; // Guarda el correo encriptado
    nuevoUsuario.password = hashedPassword; // Guarda la contraseña encriptada

    usuarios.push(nuevoUsuario);
    fs.writeFileSync(USERS_FILE, JSON.stringify(usuarios)); // Guarda el nuevo usuario en el archivo

    res.status(201).json({ message: 'Usuario creado correctamente', usuario: nuevoUsuario });
});
```

SOLICITUD:

```
  // envio de los datos de login
  onLoginSubmit(): void {
    const url = 'http://192.168.50.20:3001/login';

    this.http.post<{ userId: number }>(url, this.loginData).subscribe(
      (response) => {
        if (response && response.userId) {
          localStorage.setItem('userId', response.userId.toString());
          localStorage.setItem('username', this.loginData.username);

          // Redirigir a la página de inicio después de loguearse
          this.router.navigate(['/home']).then(() => {
            window.location.reload();
          });
        } else {
          console.log('Login fallido: Respuesta inesperada del servidor');
          alert('Error en el login. Por favor, verifica tus credenciales.');
        }
      },
      (error) => {
        this.showError('Usuario o contraseña incorrectos. Intenta nuevamente.');
      }
    );
  }
showError(message: string) {
  this.errorMessage = message;
  setTimeout(() => {
    this.errorMessage = '';
  }, 2000);
}

}
```

ENDPOINT EN EL SERVIDOR:

```
// Endpoint de login
app.post('/login', async (req, res) => {
    const { username, password } = req.body;
    const usuarios = JSON.parse(fs.readFileSync(USERS_FILE, 'utf-8'));

    // usuario por nombre
    const usuario = usuarios.find(u => {

        console.log(u);

        if (u.fullname && u.fullname.encryptedData && u.fullname.iv && u.email && u.email.encryptedData && u.email.iv) {
            // Desencriptar el nombre de usuario y correo
            const decryptedUsername = decryptData(u.fullname.encryptedData, u.fullname.iv);
            const decryptedEmail = decryptData(u.email.encryptedData, u.email.iv);

            // Comparar los valores desencriptados con el username proporcionado
            return decryptedUsername === username;
        } else {

            return false;
        }
    });

    if (usuario) {
        // Comparar la contraseña proporcionada con el hash almacenado
        const match = await bcrypt.compare(password, usuario.password);

        if (match) {
            res.json({ userId: usuario.id, message: 'Login exitoso' });
        } else {
            res.status(401).json({ message: 'Credenciales incorrectas' });
        }
    } else {
        res.status(401).json({ message: 'Usuario no encontrado' });
    }
});
```

SOLICITUD:

```
  unirseAGrupo(grupoId: number): void {
    const usuarioId = localStorage.getItem('userId');
    if (!usuarioId) {
      alert('Debes iniciar sesión para unirte a un grupo');
      this.router.navigate(['/login']);
      return;
    }

    // Se hacee la solicitud POST para unirse al grupo
    this.http.post<any>('http://192.168.50.20:3001/api/grupos/unirse', {
      groupId: grupoId,
      userId: usuarioId
    })
    .subscribe(
      response => {
        console.log(response);
        alert('Te has unido al grupo exitosamente: ' + grupoId + ": " + usuarioId);
        this.obtenerGruposDisponibles();

        // Se actualizar disponibilidad
        this.actualizarDisponibilidad(grupoId);
      },
      error => {
        console.error('Error al unirse al grupo', error);
        alert('No se pudo unir al grupo: ' + error.error.message);
      }
    );
  }
```

ENDPOINT EN EL SERVIDOR:

```
// Unirse a un grupo
app.get('/gruposdisponibles/:usuarioId', (req, res) => {
    const usuarioId = req.params.usuarioId;

    if (!usuarioId) {
        return res.status(400).json({ error: 'ID de usuario no proporcionado' });
    }

    try {

        const usuarioGrupo = JSON.parse(fs.readFileSync(USER_GROUP_FILE, 'utf-8')); // Relaciones usuario-grupo
        const grupos = JSON.parse(fs.readFileSync(SUBSCRIPTIONS_FILE, 'utf-8')); // Todos los grupos

        console.log('Usuario ID recibido:', usuarioId);

        // IDs de grupos asociados al usuario
        const gruposAsociadosIds = usuarioGrupo
            .filter(ug => parseInt(ug.userId) === parseInt(usuarioId))
            .map(ug => ug.groupId);

        console.log('IDs de grupos asociados al usuario:', gruposAsociadosIds);


        const gruposDisponibles = grupos.filter(grupo =>
            !gruposAsociadosIds.includes(grupo.id) &&
            grupo.currentUsers < grupo.maxUsers
        );

        console.log('Grupos disponibles para el usuario:', gruposDisponibles);

        res.json(gruposDisponibles);
    } catch (err) {
        console.error('Error al obtener los grupos disponibles:', err);
        res.status(500).json({ message: 'Error al procesar la solicitud' });
    }
});
```

SOLICITUD:

```
    // Crear grupo
this.http.post<{ groupId: number }>('http://192.168.50.20:3001/api/grupos/crear', grupoConUsuario)
.subscribe(
  (response) => {
    console.log('Grupo creado:', response);

    // Guardar suscripción con el ID del grupo
    this.guardarSuscripcion(this.groupData.serviceType, this.groupData.costPerUser, response.groupId);
  },
  (error) => {
    console.error('Error al crear el grupo', error);
    alert('Hubo un error al crear el grupo.');
  }
);
}
```

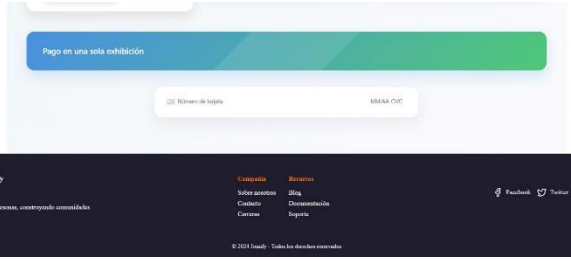
ENDPOINT EN EL SERVIDOR:

```
app.post('/api/grupos/crear', (req, res) => {

    if (!name || !serviceType || !maxUsers || !costPerUser || !paymentPolicy || !userId) {
        return res.status(400).json({ message: 'Todos los campos son obligatorios' });
    }

    const fechaCreacion = new Date();
    const fechaLimite = new Date(fechaCreacion);
    fechaLimite.setDate(fechaLimite.getDate() + 7);

    const nuevoGrupo = {
        id: generarIdCorto(),
        name,
        serviceType,
        maxUsers,
        currentUsers: 1,
        costPerUser,
        paymentPolicy,
        fechaCreacion: fechaCreacion.toISOString(),
        fechaLimite: fechaLimite.toISOString()
    };

    fs.readFile(SUBSCRIPTIONS_FILE, 'utf8', (err, data) => {
        if (err) return res.status(500).json({ message: 'Error al leer suscripciones' });

        const grupos = JSON.parse(data);
        grupos.push(nuevoGrupo);

        fs.writeFile(SUBSCRIPTIONS_FILE, JSON.stringify(grupos, null, 2), (err) => {
            if (err) return res.status(500).json({ message: 'Error al guardar el grupo' });

            fs.readFile(USER_GROUP_FILE, 'utf8', (err, data) => {
                if (err) return res.status(500).json({ message: 'Error al leer relaciones' });

                const usuarioGrupo = JSON.parse(data);
                usuarioGrupo.push({
                    userId: userId,
                    groupId: nuevoGrupo.id,
                    createdAt: nuevoGrupo.fechaCreacion
                });

                fs.writeFile(USER_GROUP_FILE, JSON.stringify(usuarioGrupo, null, 2), (err) => {
                    if (err) return res.status(500).json({ message: 'Error al guardar relación' });
                    res.status(201).json({ message: 'Grupo creado exitosamente', groupId: nuevoGrupo.id });
                });
            });
        });
    });
});
```

ESTO SOLO ES PARTE DEL CODIGO EMPLEADO PARA LA CONEXION CON EL SERVIDOR, PERO, TODO ES BASICAMENTE LO MISMO, DESDE EL FRONT SE REALIZA UNA SOLICITUD (GET, POST, PUT DELETE) Y EL SERVIDOR PROCESA DICHA SOLICITUD Y ENVIA UNA RESPUESTA EXITOSA O DE ERROR. EL SERVIDOR ES BASICAMENTE QUIEN LLEVA O MANEJA TODA LA LOGICA CON NUESTROS ARCHIVOS JSON EN EL SERVIDOR NFS.

APIS EXTERNAS EMPLEADAS EN NUESTRA APLICACIÓN

INICIALMENTE TENEMOS EL API DE STRIPE QUE IMPLEMENTAMOS PARA QUE LOS USUARIOS REALICEN LOS PAGOS DE LOS GRUPOS DE SUSCRIPCIONES A LOS QUE PERTENECES, EL CÓDIGO EMPLEADO PARA EL USO DE ESTA ES EL SIGUIENTE:



CODIGO EN EL SERVIDOR:

PRIMERAMENTE TUVIMOS QUE SOLICITAR NUESTRA LLAVE PUBLICA DE PRUEBA EN STRIPE:

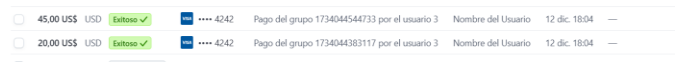
```
const stripe = new Stripe('sk_test_51QTYYiKKALSQZx7JkPBrLyHJRtyVXZoXGcWxcXI4tC1lW0vqw3upqThCRMCWBSO1cF9uQseweEzAj2twn58SvltPM08BUDAMcOX'); // Clave de STRIPE
```

ENDPOINT EN EL SERVIDOR:

```
app.post('/api/pagos/simular', async (req, res) => {
    const { userId, groupId, amount } = req.body;

    try {
        // Simulación de pago con Stripe
        const paymentIntent = await stripe.paymentIntents.create({
            amount: amount * 100, // En centavos
            currency: 'usd',
            description: `Pago del grupo ${groupId} por el usuario ${userId}`,
        });

        // Registrar pago en historial
        const pagos = JSON.parse(fs.readFileSync(PAYMENTS_FILE, 'utf-8'));
        pagos.push({
            id: pagos.length + 1,
            userId,
            groupId,
            amount,
            status: 'success',
            date: new Date().toISOString(),
            paymentIntentId: paymentIntent.id
        });
        fs.writeFileSync(PAYMENTS_FILE, JSON.stringify(pagos, null, 2));

        //
        // Aquí solo se el PaymentIntent sin procesar el pago.

        res.status(200).json({
            message: 'Transacción de pago exitosa',
            clientSecret: paymentIntent.client_secret, // Devolver el clientSecret al frontend
        });
    } catch (err) {
        console.error('Error al realizar el pago:', err);
        res.status(500).json({ message: 'Error al realizar el pago' });
    }
});
```

EN NUESTRO FRONT, TUVIMOS QUE HACER LO SIGUIENTE PARA COLOCAR EL METODO DE PAGO CON TARJETA DE STRIPE Y PROCESAR EL PAGO POR MEDIO DE NUESTRO ENDPOINT EN EL SERVIDOR:

```
    this.http.post(`http://192.168.50.20:3001/api/pagos/simular`, {
      userId: userId,
      groupId: grupoId,
      amount: monto
    }).subscribe(
      (res: any) => {
        const { clientSecret } = res;  // Obtener el clientSecret desde el servidor
        this.confirmPayment(clientSecret); // Confirmar el pago usando Stripe Elements
      },
      (error) => {
        console.error('Error en la simulación de pago:', error);
        alert('Hubo un problema al procesar la simulación. Inténtalo de nuevo.');
      }
    );
  }

  // Método para simular la confirmación del pago con Stripe
  confirmPayment(clientSecret: string) {
    this.stripe.confirmCardPayment(clientSecret, {
      payment_method: {
        card: this.card,
        billing_details: {
          name: 'Nombre del Usuario',
        },
      },
    }).then((result: any) => {
      if (result.error) {
        console.error('Error al procesar la simulación:', result.error);
        alert('Hubo un error en la simulación del pago. Inténtalo de nuevo.');
      } else {
        if (result.paymentIntent.status === 'succeeded') {
          alert('Pago realizado con éxito!');
```

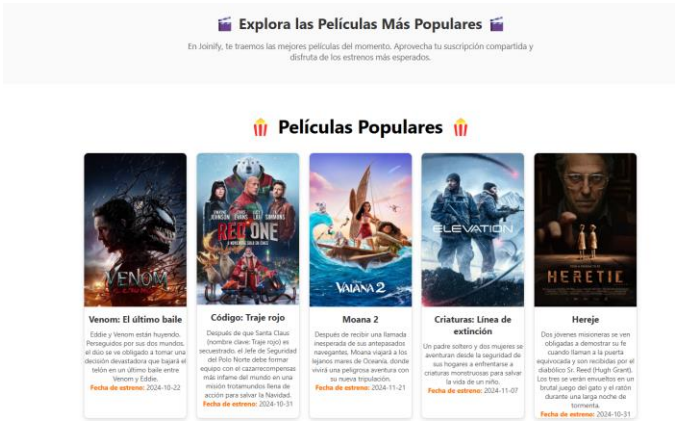


DESPUES DE ELLO TENEMOS NUESTRA OTRA API EMPLEADA LLAMADA TMB (The Movie Database), ESTA FUE EMPLEADA PARA MOSTRAR PELICULAS DE ESTRENO EN NUESTRA PAGINA DE INICIO TANTO EN NUESTRA APP MÓVIL COMO EN NUESTRO SERVIDOR.

EN ELLA, INICIALMENTE TAMBIEN TUVIMOS QUE OBTENER NUESTRA TMB API KEY, Y BASICAMENTE ESTE

FUÉ EL CODIGO EMPLEADO PARA EL USO E IMPLEMENTACIÓN DE LA MISMA:

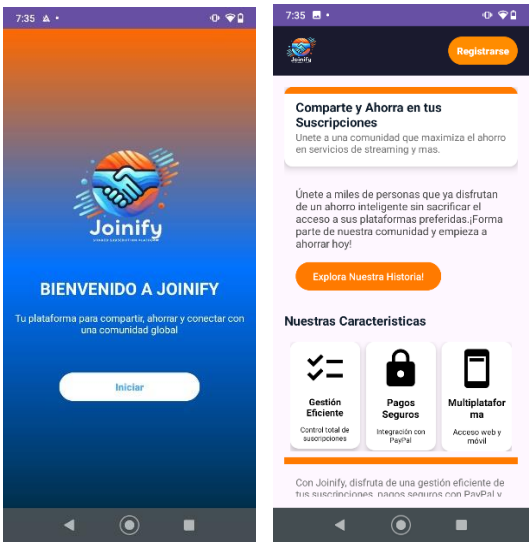
```typescript
export class HomeComponent implements OnInit {
  movies: any[] = []; // Array para almacenar las películas
  apiKey: string = '5c208ff4ecedc410685c70b86d4abcd9';
  apiUrl: string = 'https://api.themoviedb.org/3/movie/popular'; // Endpoint de TMDB

  constructor(private http: HttpClient) {}

  ngOnInit(): void {
    this.fetchMovies();
  }

  fetchMovies(): void {
    this.http
      .get<any>(`${this.apiUrl}?api_key=${this.apiKey}&language=es-MX&page=1`)
      .subscribe({
        next: (data) => {
          this.movies = data.results;
          console.log(this.movies);
        },
        error: (error) => {
          console.error('Error al obtener las películas:', error);
        },
      });
  }
}
```



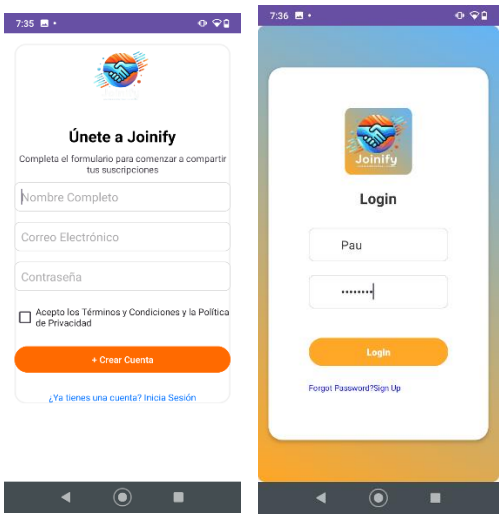
EN NUESTRA APP MÓVIL, FUE EMPLEADO EL USO DE ANDROID STUDIO PARA EL DESARROLLO DE LA MISMA, Y DE IGUAL FORMA PARA LOGRAR LA CONEXION DE ESTA CON NUESTRO SERVIDOR FUE USADA LA LIBRERÍA RETROFIT QUE NOS PERMITE REALIZAR LAS MISMAS OPERACIONES REST (GET, POST, PUT Y DELETE) EN SOLICITUDES HTTP:

```java
public interface ApiService {  15 usages
    @POST("/usuario")  1 usage
    Call<Void> registrarUsuario(@Body Usuario usuario);

    // Endpoint para iniciar sesión
    @POST("/login")  1 usage
    Call<Loginresponse> login(@Body LoginRequest loginRequest);

    // Endpoint para crear un grupo
    @POST("/api/grupos/crear")  1 usage
    Call<GroupResponse> crearGrupo(@Body GrupoRequest grupoRequest);

    //Endpoint para mostrar los grupos de un usuario
    @GET("/api/grupos/usuario")  1 usage
    Call<List<GrupoResponse>> obtenerGrupos(@Query("usuarioId") String usuarioId);

    // Endpoint para obtener los grupos disponibles basados en el usuarioId
    @GET("/gruposdisponibles/{usuarioId}")  1 usage
    Call<List<Grupo>> obtenerGruposDisponibles(@Path("usuarioId") String usuarioId);

    // Endpoint para unirse a un grupo
    @POST("/api/grupos/unirse")  1 usage
    Call<RespuestaUnirse> unirseAlGrupo(@Body UnirseGrupoRequest request);

    @POST("/api/servicio-suscripcion/guardar")  1 usage
    Call<Void> saveSubscription(@Body SuscripcionRequest subscriptionRequest);

    // Endpoint for exiting the group
    @DELETE("/api/grupos/salir/{groupId}/{userId}")  1 usage
    Call<Void> salirDelGrupo(@Path("groupId") String groupId, @Path("userId") int userId);

    @DELETE("/api/grupos/baja/{groupId}")  1 usage
    Call<ResponseBody> unsubscribeFromGroup(@Path("groupId") int groupId);

    @PUT("/api/servicio-suscripcion/actualizar/{groupId}")  2 usages
    Call<ResponseBody> updateGroupAvailability(@Path("groupId") int groupId);

    @POST("/api/pagos/simular")  1 usage
    Call<PaymentResponse> simularPago(@Body PaymentRequest paymentRequest);
```

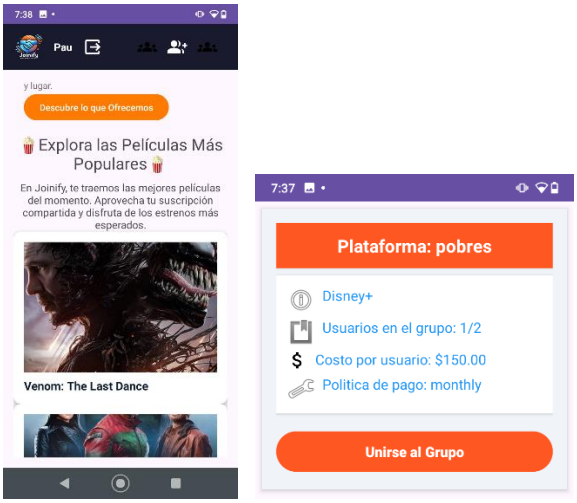
IMÁGENES DE NUESTRA APP MÓVIL:



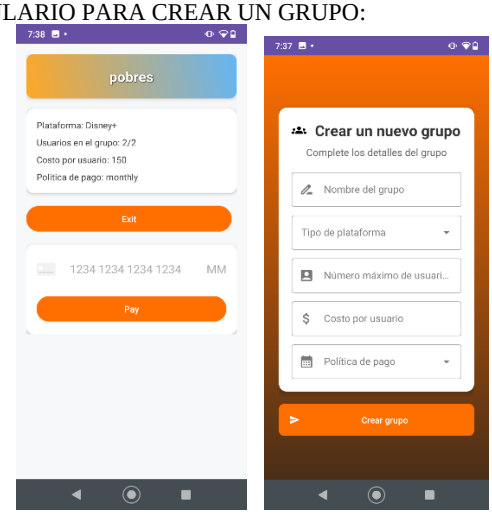
REGISTRO Y LOGIN DE UN USUARIO:



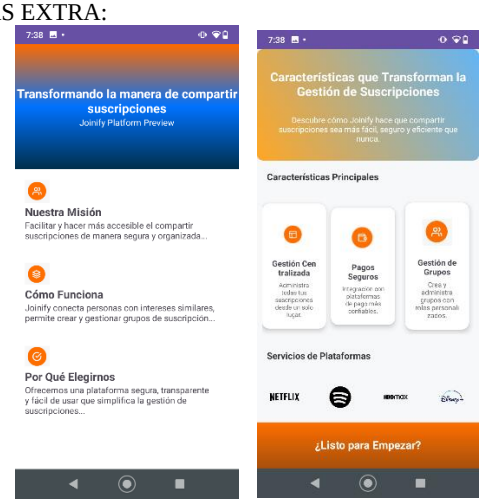
USUARIO LOGUEADO Y GRUPOS A LOS QUE PUEDE UNIRSE:



GRUPOS A LOS QUE PERTENECE UN USUARIO Y FORMULARIO PARA CREAR UN GRUPO:



PÁGINAS EXTRA:



BENEFICIOS LOGRADOS

JOINIFY PROPORCIONA A LOS USUARIOS UNA MANERA EFECTIVA Y ECONÓMICA DE ADMINISTRAR SUS SUSCRIPCIONES COMPARTIDAS. LA SOLUCIÓN IMPLEMENTADA NO SOLO PERMITE AHORROS SIGNIFICATIVOS AL DIVIDIR LOS COSTOS ENTRE USUARIOS, SINO QUE TAMBIÉN PROMUEVE UN USO ORGANIZADO Y TRANSPARENTE DE LOS SERVICIOS.

PROBLEMAS ENCONTRADOS DUARNTE EL DESARROLLO DEL PROYECTO.

- **PROBLEMA CON LA SINCRONIZACIÓN DEL SERVIDOR CON GOOGLE DRIVE**: SE TRATÓ DE UTILIZAR RCLONE PARA REALIZAR ESTA SINCRONIZACIÓN ENTRE EL SERVIDOR Y LOS ARCHIVOS NFS Y GOOGLE DRIVE, PERO NO SE SINCRONIZABAN BIEN LOS CAMBIOS YA QUE SE TENÍA QUE RECARGAR EL SERVIDOR PARA PODER VER LOS CAMBIOS QUE SE REALIZABAN EN LOS ARCHIVOS, POR LO QUE CREAMOS UN SCRIPT CON LA TAREA DE RECARGAR Y PERMITIR VER REFLEJADOS LOS CAMBIOS, PERO DESPUÉS SE PRESENTÓ UN PROBLEMA NUEVO, YA QUE NO SE PODÍA VER CON ÉXITO EL CONTENIDO DE LOS ARCHIVOS QUE SE CREABAN EN EL SERVIDOR, POR LO QUE OPTAMOS POR IR CON EL PROFESOR PARA QUE NOS ASESORARA AL RESPECTO, PERO COMO NOS MENCIONÓ QUE ESTA PARTE DEL PROYECTO CONTARÍA COMO UN ASPECTO EXTRA DEBIDO A SU COMPLEJIDAD Y AL POCO TIEMPO QUE TENÍAMOS, POR LO QUE DECIDIMOS DARLE PRIORIDAD A LOS DEMÁS ASPECTOS DEL PROYECTO.
- **PROBLEMAS CON LAS IPS:** DEBÍAMOS DE CONFIGURAR NUESTRO SERVIDOR DE MANERA QUE LOS CLIENTES PUDIERAN TENER ACCESO A LA PÁGINA WEB Y QUE ESTA REALIZARA CORRECTAMENTE LAS SOLICITUDES PERTINENTES PARA EL CORRECTO FUNCIONAMIENTO DEL FLUJO DE DATOS ENTRE LOS SERVIDORES Y LOS CLIENTES, POR LO QUE OPTAMOS POR HACER LAS CONFIGURACIONES NECESARIAS A LA MÁQUINA VIRTUAL DONDE SE ENCUENTRA EL

servidor y editar el archivo exports que es donde se especifican las IPs que se permiten montar y para aceptar un buen rango de estas se especificó un rango dependiendo de en qué lugar y a que red nos conectáramos.

- Problemas con los endpoint y la aplicación móvil: una vez que teníamos la aplicación web procedimos a configurar la conexión con el servidor de manera que el flujo de información funcionara correctamente, pero tuvimos un problema con la sección de los grupos ya que no aparecía los grupos a los cuales pertenecía el usuario que inicio sesión, esto debido a que los endpoint no estaban configurados correctamente, por lo que una vez detectado el problema procedimos a realizar los cambios necesarios para que la conexión y el flujo de informacion entre el servidor y la aplicación móvil fuera la adecuada.

Conclusiones

El desarrollo del proyecto Joinify ha permitido implementar una plataforma eficiente y funcional que aborda la problemática actual de reducir los costos de servicios por suscripción mediante un modelo colaborativo. A través de la creación de grupos de suscripciones compartidas, se logró una solución que facilita a los usuarios dividir los costos de servicios como plataformas de streaming, manteniendo una experiencia accesible y organizada

En cuanto al almacenamiento distribuido, se implementó un DFS (Distributed File System) que utilizó un servidor NFS local, logrando replicar los datos para garantizar redundancia y disponibilidad. La seguridad de la información se fortaleció con encriptación de archivos, asegurando que solo la aplicación pudiera desencriptar y mostrar la información protegida.

El soporte multiplataforma fue otro aspecto destacado del proyecto, permitiendo el acceso desde aplicaciones móviles y navegadores web. Esto garantiza la compatibilidad con distintos sistemas operativos como Ubuntu y Windows, logrando que la plataforma sea accesible para cualquier usuario.

En conclusión, Joinify no solo proporciona una herramienta práctica y económica para la gestión de suscripciones compartidas, sino que también sirvió como una aplicación integral de conceptos de sistemas distribuidos, tales como la integración de APIs RESTful, almacenamiento distribuido, replicación de datos y medidas de seguridad los cuales son temas y prácticas que aprendimos a lo largo de este semestre por lo que pasar de la teoría o de ambientes controlados a implementaciones en la vida real nos ayudó a adquirir experiencia con respecto a todas las herramientas que existen y nos ayudan, además de la comprensión de cómo funcionan distintos conceptos tecnológicos con lo que ahora ya estamos familiarizados. Este proyecto final representa un ejemplo claro de cómo las tecnologías distribuidas pueden aplicarse para desarrollar sistemas robustos, seguros y accesibles, respondiendo a las necesidades actuales de los usuarios de manera eficiente.

LINK DEL REPOSITORIO EN GITHUB CON EL CÓDIGO DE LA APP MÓVIL Y EL CODIGO DE LA PÁGINA WEB:

https://github.com/MichaelPad2356/ProyectoRedesWEB_APP

References

[1] https://www.bbvaapimarket.com/es/mundo-api/como-netflix-puede-verse-en-mas-de-1000-dispositivos-gracias-las-apis/

[2] https://developer.themoviedb.org/reference/intro/getting-started

[3] https://learn.microsoft.com/es-es/windows-server/storage/nfs/nfs-overview