

01 NEST MICROSERVICIOS - SUPERFLIGHT

- En la carpeta api-gateway creo el proyecyo con **nest new superflights**
- Instalo **@nestjs/config** con npm
- Configuro para usar variables de entono en imports de **app.module con ConfigModule**

```
ConfigModule.forRoot({
  envFilePath: ['.env'],
  isGlobal: true
})
```

Filtro Global

- Para configurar el **filtro global para las excepciones** creo en la carpeta **src/common/filters/http-exception-filter.ts**
- Creo la clase AllExceptionHandler que implementa ExceptionFilter de @nestjs/common
- Le añado el decorador @Catch, con ctrl+click implemento la interfaz
- Creo el logger con new Logger() y le paso el nombre con AllExceptionHandler.name
- Creo el contexto, extraigo la Response y la Request
- Creo la constante status y evalúo si es una instancia de HttpException para obtener el status, si no devuelvo un server error
- Creo el mensaje
- Al logger le paso el status y el error
- Creo la Response pasándole la fecha, el path de la url y el message

```
import { ArgumentsHost, Catch, ExceptionFilter, HttpException, Logger } from
'@nestjs/common'
@Catch()
export class AllExceptionHandler implements ExceptionFilter{
  private readonly logger = new Logger(AllExceptionHandler.name)

  catch(exception: any, host: ArgumentsHost){
    const ctx = host.switchHttp()
    const res = ctx.getResponse()
    const req = ctx.getRequest()

    const status = exception instanceof HttpException
      ? exception.getStatus()
      : HttpStatus.INTERNAL_SERVER_ERROR

    const msg = exception instanceof HttpException ? exception.getResponse():
exception

    this.logger.error(`Status ${status} Error: ${JSON.stringify(msg)}`)

    res.status(status).json({
```

```
        timestamps: new Date().toISOString(),
        path: req.url,
        error: msg
      })
    }
  }
}
```

- Configuro en el main el uso global de filtros

```
app.useGlobalFilters(new AllExceptionHandler())
```

Interceptor Global

- Creo en la carpeta **src/common/interceptors/timeOut.interceptor.ts**
- La clase implementa NestInterceptor
- Le coloco el decorador @Injectable
- La interfaz me obliga a implementar el método intercept

```
imports {CallHandler, ExecutionContext, Injectable, NestInterceptor} from
 '@nestjs/common'
import {Observable} from 'rxjs'
import {timeout} from 'rxjs/operator'

@Injectable()
export class TimeOutInterceptor implements NestInterceptor{
  intercept(context:ExecutionContext, next: CallHandler<any>): Observable<any> |
  Promise<Observable<any>>{
    return next.handle().pipe(timeout(120000)) //120 segundos
  }
}
```

- En el main seteo el uso de interceptores globales

```
app.useGlobalInterceptors(new TimeOutInterceptor())
```

Instalacion de dependencias

- api-gateway nos hará de proxy
- Para RabbitMQ es amqplib

```
npm i amqplib amqp-connection-manager class-validator class-transformer @nestjs/microservices
```

- En api/gateway creo el módulo y el controlador con **nest g mo user** y **nest g co user**
- Creo la carpeta dto
- @ApiProperty es de swagger

```
export class UserDTO{

  @ApiProperty()
  @IsNotEmpty()
  @IsString()
  readonly name: string

  @ApiProperty()
  @IsNotEmpty()
  @IsString()
  readonly userName: string

  @ApiProperty()
  @IsNotEmpty()
  @IsString()
  readonly email: string

  @ApiProperty()
  @IsNotEmpty()
  @IsString()
  readonly password: string

}
```

Configuración de RabbitMQ

- Creamos un módulo de proxy
- En el guardaremos la configuración de la conexión a RabbitMQ
- Creo en src/common/proxy/client-proxy.ts y proxy.module.ts
- En client-proxy

```
import {Injectable} from '@nestjs/common'
import {ConfigService} from '@nestjs/config'
import {ClientProxy, ClientProxyFactory, Transport} from '@nestjs/microservices'

@Injectable()
export class ClientProxySuperFilghts{
  constructor(private readonly config: ConfigService){}

  clientProxyUsers(): ClientProxy{
    return ClientProxy.create({
      transport: Transport.RMQ,
      option:{
        urls: this.config.get('AMQP_URL'), //hay que crear esta variable
      }
    })
  }
}
```

```

de entorno en .env
        queue: 'users' //el nombre de la cola con la que vamos a manejar
usuarios, usaré RabbitMQ.UserQueue
    }
  })
}
}

```

- Creo en src/common/constants.ts un enum donde guardaré las colas (queue)

```

export enum RabbitMQ{
  UserQueue = 'users'
}

```

- En proxy.module exporto el ClientProxy

```

@Module({
  providers: [ClientProxySuperFlights],
  exports: [ClientProxySuperFlights]
})
export class ProxyModule{}

```

- En users.module importo el ProxyModule en imports para tenerlo disponible
- Configuro la variable de entorno AMQP_URL que añadí en la conexión
- Creo una nueva instancia gratuita de cloudamqp.com (RabbitMQ as a Service)
- En Details, AMQP URL tengo la url que colocaré en mi variable de entorno

```
AMQP_URL=amqp://aljshalsjlaksjlaksj
```

- Si ingreso en RabbitMQManager (boton verde arriba a la izquierda) puedo ver las conexiones, las queues
- Debo configurar en el main el microservicio

Controlador Usuario

- En el constructor inyectaremos el ClientProxy
- En el método POST, cómo estoy trabajando con microservicios obtendré de la respuesta un Observable de tipo IUser
- IUser es una interfaz que he creado

```

@Controller('api/v2/user')
export class UsersController{
  constructor(private readonly clientProxy: ClientProxySuperFlights){}

```

```

    private clientProxyUser = this.clientProxy.ClientProxyUsers()

    @Post()
    create(@Body() userDto: UserDTO): Observable<IUser>{
        return this.clientProxyUser.send()
    }
}

```

- IUser

```

export interface IUser{
    name: string
    username: string
    email: string
    password: string
}

```

- También tengo las interfaces de IFlight, Ipassenger, ILocation e IWeather
- Creo en constants.ts el enum para el CRUD de mi aplicación

```

export enum UserMSG{
    CREATE = 'CREATE_USER',
    FIND_ALL = 'FIND_USERS',
    FIND_ONE= 'FIND_USER',
    UPDATE = 'UPDATE_USER',
    DELETE='DELETE_USER',
    VALID_USER = 'VALID_USER'
}

```

- Ahora en el .send le mando UserMSG.CREATE y el dto

```

@Post()
create(@Body() userDto: UserDTO): Observable<IUser>{
    return this.clientProxyUser.send(UserMSG.CREATE, userDto)
}

```

- Hago lo mismo con el resto de CRUD

```

@Controller('api/v2/user')
export class UsersController{
    constructor(private readonly clientProxy: ClientProxySuperFlights){}

    private clientProxyUser = this.clientProxy.ClientProxyUsers()

```

```

    @Post()
    create(@Body() userDto: UserDTO): Observable<IUser>{
        return this.clientProxyUser.send()
    }

    @Get()
    findAll(): Observable<IUser[]>{
9        return this.clientProxyUser.send(UserMSG.findAll, '')
    }

    @Get('/:id')
    findOne(@Param('id') id: String): Observable<IUser>{
        return this.clientProxy.send(UserMSG.findOne, id)
    }

    @Put('/:id')
    update(@Param('id') id: string, @Body() userDto: UserDTO) : Observable<IUser>{
        return this.clientProxy.send(UserMSG.UPDATE, {id, userDto})
    }

    @Delete('/:id')
    delete(@Param('id') id: string) : Observable <IUser>{
        return thisclientProxy.send(UserMSG.DELETE, id)
    }
}

```

Estructura de módulo

- Creo el módulo con nest g mo passenger y el controlador con nest g co passenger
- Creo en passenger/dto/passenger.dto
- Para configurar la conexión de RabbitMQ, en constants creo la cola de pasajeros

```

export enum RabbitMQ{
    UserQueue = 'users',
    PassengerQueue = 'passengers'
}

```

- En ClientProxySuperFlights creo la conexión de Passengers

```

import {Injectable} from '@nestjs/common'
import {ConfigService} from '@nestjs/config'
import {ClientProxy, ClientProxyFactory, Transport} from '@nestjs/microservices'

@Injectable()
export class ClientProxySuperFlights{
    constructor(private readonly config: ConfigService){}

```

```

    clientProxyUsers(): ClientProxy{
        return ClientProxy.create({
            transport: Transport.RMQ,
            option:{
                urls: this.config.get('AMQP_URL'), //hay que crear esta variable
de entorno en .env
                queue: 'users' //el nombre de la cola con la que vamos a manejar
usuarios, usaré RabbitMQ.UserQueue
            }
        })
    }

    clientProxyPasenngers(): ClientProxy{
        return ClientProxy.create({
            transport: Transport.RMQ,
            option:{
                urls: this.config.get('AMQP_URL'),
                queue: RabbitMQ.PassengerQueue
            }
        })
    }
}

```

- Debo importar el ProxyModule en imports de PassengerModule
- Inyecto el clientProxy en PassengerController y hago el CRUD completo
- Cada método me devuelve un Observable de tipo IPassenger
- En constants.ts creo el enum de PassengerMSG con los métodos del CRUD
- Hacemos lo mismo con el módulo de Weather
- El main de api-gateway queda así

```

import { NestFactory } from '@nestjs/core';
import { DocumentBuilder, SwaggerModule } from '@nestjs/swagger';
import { AppModule } from './app.module';

async function bootstrap() {
    const app = await NestFactory.create(AppModule);

    const options = new DocumentBuilder()
        .setTitle('SuperFlight API')
        .setDescription('Scheduled Flights App')
        .setVersion('2.0.0')
        .addBearerAuth()
        .build();

    const document = SwaggerModule.createDocument(app, options);

    SwaggerModule.setup('/api/docs', app, document, {
        swaggerOptions: {
            filter: true,
        },
    },

```

```
});

  await app.listen(process.env.PORT || 3000);
}
bootstrap();
```

- Y el app.module así

```
import { Module } from '@nestjs/common';
import { ConfigModule } from '@nestjs/config';
import { AppController } from './app.controller';
import { AppService } from './app.service';

@Module({
  imports: [
    ConfigModule.forRoot({
      envFilePath: ['.env.development'],
      isGlobal: true,
    }),
  ],
  controllers: [AppController],
  providers: [AppService],
})
export class AppModule {}
```

02 MS NEST - SUPERFLIGHT USERS

- Fuera de la carpeta de api-gateway creo otro proyecto que será mi microservicio
- Lo llamaré ms-users

```
nest new ms-users
```

- **instalación de dependencias**

```
npm i bcrypt mongoose @nestjs/mongoose amqpplib amqp-connection-manager @nestjs/config @nestjs/config
```

- **Estructura del microservicio**
- Copio lo que tengo en mi aplicación monolítica
- En users.module tengo la conexión a mongoose

```
import { Module } from '@nestjs/common'
import { MongooseModule } from '@nestjs/mongoose'
import { UserController } from './user.controller'
import { UserService } from './user.service'
import { USER } from 'src/common/models/models'
import { UserSchema } from 'schema/user.schema'
```



```
@Module({
  imports:[
    MongooseModule.forFeatureAsync([
      {
        name: USER.name,
        useFactory: () => UserSchema
      }
    ])
  ],
  controllers: [UserController],
  providers: UserService,
  exports: [UsersService]
})
export class UsersModule
```

- Importo el UserModule en AppModule
- Quito del UserDTO todos los decoradores de las propiedades (por eso no instalamos class-validator y class-transformer)
- Creo en src/common/models/models.ts importando el modelo de la aplicación monolítica

```
export const USER = {name:'users'}
export const PASSENGER = {name:'passengers'}
export const FLIGHT = {name:'flights'}
```

- Ahora en src/interfaces si dejo que IUser extienda de Document ya que usaremos Mongoose

```
export interface IUser extends Document{
  name: string
  username: string
  password: string
  email: string
}
```

- Ya no vamos a necesitar exportar el servicio de UsersService porque haremos la comunicación a través del microservicio

Configuración de microservicio

- Copio la conexión con RabbitMQ que tengo en la variable de entorno de api-gateway y la copio en .env de users
- Creo la variable de entorno para la conexión con la DB

```
AMQP_URL=amqps://lkjholihjoliij
URI_MONGODB=mongodb://localhost:27017/superflights
```

- En app.module configuro para usar las variables de entorno con ConfigModule.forRoot

```
@Module({
  imports:[
    ConfigModule.forRoot({
      envFilePath: ['.env.development'],
      isGlobal: true
    }),
    MongooseModule.forRoot(process.env.URI_MONGODB,{
      useCreateIndex: true,
      useFindAndModify: false
    })
    UsersModule
  ],
  controllers: [AppController],
  providers: [AppService]
})
export class AppModule {}
```

- En el main configuro la conexión con microservicios

```
async function bootstrap(){
  const app= await NestFactory.createMicroservice(AppModule, {
    Transport: Transport.RMQ,
    options:{
      urls:[process.env.AMQP_URL],
      queue: RabbitMQ.UserQueue
    }
  })
  await app.listen()
}
bootstrap()
```

- Copio el archivo constants.ts de api-gateway en users (dejo solo el enum de RabbitMQ y UserMSG)

Controlador del microservicio de User

- En lugar de usar los decoradores @Get, @Post, etc. usaré **@MessagePattern**
- Ahora ya no dispongo de los decoradores como **@Body**, **@Param**, solo del decorador **@Payload**

```
@Controller()
export class UserController{
  constructor(private readonly userService: UserService)

  @MessagePattern>(UserMSG.CREATE)
  create(@Payload() userDto: UserDTO){
```

```

        return this.userService.create(userDto)
    }

    @MessagePattern(UserMSG.FIND_ONE)
    findOne(@Payload() id: string){
        return this.userService.findOne(id)
    }

    @MessagePattern(UserMSG.UPDATE){
        update(@Payload() payload: any){
            return this.userService.update(payload.id, payload.userDto)
        }
    }
}

```

Pruebas de encolamiento con RabbitMQ

- Si levanto el server, RabbitMq ya reconoció la conexión y encontró una conexión, un canal y una queue (de user)
- Desde POSTMAN, si apunto a localhost:3000/api/v2/users obtengo lo deseado
- Yo puedo cancelar mi petición en tiempo de ejecución pero la cola se mantiene
- Si vuelvo a iniciar el server realiza la petición que había en la cola

Instalación de microservicio

- Creo el microservicio de passenger fuera de api-gateway

```
nest new ms-passengers
```

- Hacemos las mismas instalaciones que en el microservicio anterior
- Copiamos el módulo completo de Passenger de la aplicación monolítica que creamos dentro de src
- Ya no exportamos el servicio en passenger.module
- Importamos el PassengerModule en app.module^o
- En el controlador eliminamos el UseGuard y todo lo demás, sólo dejamos @Controller
- En **src/common/models/models.ts** dejo solo el de PASSENGER

```
export const PASSENGER = {name: 'passengers'}
```

- En src/interfaces/passenger.interface.ts copio la interfaz

```

export interface IPassenger extends Document{
    name: string
    email:string
}

```

- El PassengerSchema se queda igual

```
import * as mongoose from 'mongoose'

export const PassengerSchema = new mongoose.Schema({
  name: {type: String, required: true},
  email: {type: String, required: true}
})

PassengerSchema.index({email:1}, {unique: true})
```

- En el dto quitamos los decoradores del class-validator

```
export class PassengerDTO{
  readonly name: string
  readonly email: string
}
```

Configuración Microservicio

- Copio las variables de entorno de ms-user y las pego en ms-passenger
- Tenemos la conexión a la Db y a RabbitMQ
- En el main cambio a microservicios

```
async function bootstrap(){
  const app= await NestFactory.createMicroservice(AppModule, {
    Transport: Transport.RMQ,
    options:{
      urls:[process.env.AMQP_URL],
      queue: RabbitMQ.UserQueue
    }
  })
  await app.listen() //aquí ya no enviamos nada
}
bootstrap()
```

- En app.module importo el ConfigModule

```
import { MongooseModule } from '@nestjs/mongoose';
import { Module } from '@nestjs/common';
import { AppController } from './app.controller';
import { AppService } from './app.service';
import { ConfigModule } from '@nestjs/config';

@Module({
```

```
imports: [  
  ConfigModule.forRoot({  
    envFilePath: ['.env.development'],  
    isGlobal: true,  
  }),  
  mongooseModule.forRoot(process.env.URI_MONGODB),  
],  
controllers: [AppController],  
providers: [AppService],  
})  
export class AppModule {}
```

- Hago el mismo proceso de cambiar los decoradores Get, Post por MessagePattern y ya no dispongo de Body, ni Params, solo Payload

Protección de rutas

- Lo haremos desde api-gateway
- Instalamos passport-jwt @nestjs/jwt passport-local passport @nestjs/passport
- Copiamos el módulo de la aplicación monolítica y lo copiamos en src de api-gateway
- Importamos el módulo en app.module de api-gateway

```
import { Module } from '@nestjs/common';  
import { ConfigModule } from '@nestjs/config';  
import { AppController } from './app.controller';  
import { AppService } from './app.service';  
import { UserModule } from './user/user.module';  
import { PassengerModule } from './passenger/passenger.module';  
import { FlightModule } from './flight/flight.module';  
import { AuthModule } from './auth/auth.module';  
  
@Module({  
  imports: [  
    ConfigModule.forRoot({  
      envFilePath: ['.env.development'],  
      isGlobal: true,  
    }),  
    UserModule,  
    PassengerModule,  
    FlightModule,  
    AuthModule,  
  ],  
  controllers: [AppController],  
  providers: [AppService],  
})  
export class AppModule {}
```

- auth.module

```
import { Module } from '@nestjs/common';
import { ConfigModule, ConfigService } from '@nestjs/config';
import { JwtModule } from '@nestjs/jwt';
import { PassportModule } from '@nestjs/passport';
import { ProxyModule } from 'src/common/proxy/proxy.module';
import { UserModule } from 'src/user/user.module';
import { AuthController } from './auth.controller';
import { AuthService } from './auth.service';
import { JwtStrategy } from './strategies/jwt.strategy';
import { LocalStrategy } from './strategies/local.strategy';

@Module({
  imports: [
    UserModule,
    PassportModule,
    ProxyModule, //importo ProxyModule para la autenticación
    JwtModule.registerAsync({
      imports: [ConfigModule],
      inject: [ConfigService],
      useFactory: (config: ConfigService) => ({
        secret: config.get('JWT_SECRET'),
        signOptions: {
          expiresIn: config.get('EXPIRES_IN'),
          audience: config.get('APP_URL'),
        },
      }),
    }),
  ],
  controllers: [AuthController],
  providers: [AuthService, LocalStrategy, JwtStrategy],
})
export class AuthModule {}
```

- proxy.module es este

```
import { Module } from '@nestjs/common';
import { ClientProxySuperFlights } from './client-proxy';

@Module({
  providers: [ClientProxySuperFlights],
  exports: [ClientProxySuperFlights],
})
export class ProxyModule {}
```

- El **client-proxy** es este

```
import { Injectable } from '@nestjs/common';
import { ConfigService } from '@nestjs/config';
```

```

import {
  ClientProxy,
  ClientProxyFactory,
  Transport,
} from '@nestjs/microservices';
import { RabbitMQ } from '../constants';

@Injectable()
export class ClientProxySuperFlights {
  constructor(private readonly config: ConfigService) {}

  clientProxyUsers(): ClientProxy {
    return ClientProxyFactory.create({
      transport: Transport.RMQ,
      options: {
        urls: this.config.get('AMQP_URL'), //obtenemos la variable de entorno
        queue: RabbitMQ.UserQueue,
      },
    });
  }

  clientProxyPassengers(): ClientProxy {
    return ClientProxyFactory.create({
      transport: Transport.RMQ,
      options: {
        urls: this.config.get('AMQP_URL'),
        queue: RabbitMQ.PassengerQueue,
      },
    });
  }

  clientProxyFlights(): ClientProxy {
    return ClientProxyFactory.create({
      transport: Transport.RMQ,
      options: {
        urls: this.config.get('AMQP_URL'),
        queue: RabbitMQ.FlightQueue,
      },
    });
  }
}

```

- auth.service
- En auth.service no tenemos userService, está en el microservicio
- Usaremos el clientProxy, llamamos a .send y usamos el valor de la constante que pertenece al servicio de user

```

import { ClientProxySuperFlights } from '../../common/proxy/client-proxy';
import { Injectable } from '@nestjs/common';
import { JwtService } from '@nestjs/jwt';

```

```
import { UserDTO } from 'src/user/dto/user.dto';
import { UserMSG } from 'src/common/constants';

@Injectable()
export class AuthService {
  constructor(
    private readonly clientProxy: ClientProxySuperFlights,
    private readonly jwtService: JwtService,
  ) {}

  private _clientProxyUser = this.clientProxy.clientProxyUsers();

  async validateUser(username: string, password: string): Promise<any> {
    const user = await this._clientProxyUser
      .send(UserMSG.VALID_USER, {
        username,
        password,
      })
      .toPromise();

    if (user) return user;

    return null;
  }

  async signIn(user: any) {
    const payload = {
      username: user.username,
      sub: user._id,
    };

    return { access_token: this.jwtService.sign(payload) };
  }

  async signUp(userDTO: UserDTO) {
    return await this._clientProxyUser
      .send(UserMSG.CREATE, userDTO)
      .toPromise();
  }
}
```

- En guards tengo
- jwt-auth.guard

```
import { Injectable } from '@nestjs/common';
import { AuthGuard } from '@nestjs/passport';

@Injectable()
export class JwtAuthGuard extends AuthGuard('jwt') {}
```


- local.auth.guard

```
import { Injectable } from '@nestjs/common';
import { AuthGuard } from '@nestjs/passport';

@Injectable()
export class LocalAuthGuard extends AuthGuard('local') {}
```

- jwt.strategy

```
import { Injectable } from '@nestjs/common';
import { PassportStrategy } from '@nestjs/passport';
import { ExtractJwt, Strategy } from 'passport-jwt';

@Injectable()
export class JwtStrategy extends PassportStrategy(Strategy) {
  constructor() {
    super({
      jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),
      ignoreExpiration: false,
      secretOrKey: process.env.JWT_SECRET,
    });
  }

  async validate(payload: any) {
    return { userId: payload.sub, username: payload.username };
  }
}
```

- local.strategy

```
import { Injectable, UnauthorizedException } from '@nestjs/common';
import { PassportStrategy } from '@nestjs/passport';
import { Strategy } from 'passport-local';
import { AuthService } from '../auth.service';

@Injectable()
export class LocalStrategy extends PassportStrategy(Strategy) {
  constructor(private readonly authService: AuthService) {
    super();
  }

  async validate(username: string, password: string): Promise<any> {
    const user = await this.authService.validateUser(username, password);

    if (!user) throw new UnauthorizedException();

    return user;
  }
}
```

```
}
}
```

- Para la protección de rutas en los microservicios
- En el main de api-gateway añadido addBearerAuth para que swagger pueda usar jwt

```
import { NestFactory } from '@nestjs/core';
import { DocumentBuilder, SwaggerModule } from '@nestjs/swagger';
import { AppModule } from './app.module';
import { AllExceptionHandler } from './common/filters/http-exception.filter';
import { TimeoutInterceptor } from './common/interceptors/timeout.interceptor';

async function bootstrap() {
  const app = await NestFactory.create(AppModule);
  app.useGlobalFilters(new AllExceptionHandler());
  app.useGlobalInterceptors(new TimeoutInterceptor());

  const options = new DocumentBuilder()
    .setTitle('SuperFlight API')
    .setDescription('Scheduled Flights App')
    .setVersion('2.0.0')
    .addBearerAuth()
    .build();

  const document = SwaggerModule.createDocument(app, options);

  SwaggerModule.setup('/api/docs', app, document, {
    swaggerOptions: {
      filter: true,
    },
  });

  await app.listen(process.env.PORT || 3000);
}
bootstrap();
```

- auth.controller

```
import { Body, Controller, Post, Req, UseGuards } from '@nestjs/common';
import { ApiTags } from '@nestjs/swagger';
import { UserDTO } from 'src/user/dto/user.dto';
import { AuthService } from './auth.service';
import { LocalAuthGuard } from './guards/local-auth.guard';

@ApiTags('Authentication')
@Controller('api/v2/auth')
export class AuthController {
  constructor(private readonly authService: AuthService) {}
```

```

@UseGuards(LocalAuthGuard) //hago uso de la autenticación
@Post('signin')
async signIn(@Req() req) {
  return await this.authService.signIn(req.user);
}

@Post('signup')
async signUp(@Body() userDTO: UserDTO) {
  return await this.authService.signUp(userDTO);
}
}

```

- Hay que colocar en .env de api-gateway la clave secreta y el tiempo de expiración del token
- Paso el constants.ts de api-gateway y las interfaces

```

export enum RabbitMQ {
  UserQueue = 'users',
  PassengerQueue = 'passengers',
  FlightQueue = 'flights',
}

export enum UserMSG {
  CREATE = 'CREATE_USER',
  FIND_ALL = 'FIND_USERS',
  FIND_ONE = 'FIND_USER',
  UPDATE = 'UPDATE_USER',
  DELETE = 'DELETE_USER',
  VALID_USER = 'VALID_USER',
}

export enum PassengerMSG {
  CREATE = 'CREATE_PASSENGER',
  FIND_ALL = 'FIND_PASSENGERS',
  FIND_ONE = 'FIND_PASSENGER',
  UPDATE = 'UPDATE_PASSENGER',
  DELETE = 'DELETE_PASSENGER',
}

export enum FlightMSG {
  CREATE = 'CREATE_FLIGHT',
  FIND_ALL = 'FIND_FLIGHTS',
  FIND_ONE = 'FIND_FLIGHT',
  UPDATE = 'UPDATE_FLIGHT',
  DELETE = 'DELETE_FLIGHT',
  ADD_PASSENGER = 'ADD_PASSENGER',
}

```

- flight.interface

```
import { IPassenger } from 'src/common/interfaces/passenger.interface';
import { IWeather } from './weather.interface';
export interface IFlight {
  _id?: string;
  pilot: string;
  airplane: string;
  destinationCity: string;
  flightDate: Date;
  passengers: IPassenger[];
  weather: IWeather[];
}
```

- location.interfaces

```
export interface ILocation {
  title: string;
  location_type: string;
  woeid: number;
  latt_long: string;
}
```

- passenger.interface

```
export interface IPassenger {
  name: string;
  email: string;
}
```

- user.interface

```
export interface IUser {
  name: string;
  username: string;
  email: string;
  password: string;
}
```

- weather.interface

```
export interface IWeather {
  id: number;
  weather_state_name: WeatherStateName;
  weather_state_abbr: WeatherStateAbbr;
  wind_direction_compass: WindDirectionCompass;
```

```

    created: Date;
    applicable_date: Date;
    min_temp: number | null;
    max_temp: number | null;
    the_temp: number | null;
    wind_speed: number;
    wind_direction: number;
    air_pressure: number | null;
    humidity: number | null;
    visibility: number | null;
    predictability: number;
  }

  export enum WeatherStateAbbr {
    C = 'c',
    Hc = 'hc',
    Lc = 'lc',
    Lr = 'lr',
    S = 's',
  }

  export enum WeatherStateName {
    Clear = 'Clear',
    HeavyCloud = 'Heavy Cloud',
    LightCloud = 'Light Cloud',
    LightRain = 'Light Rain',
    Showers = 'Showers',
  }

  export enum WindDirectionCompass {
    False = 'False',
    N = 'N',
    Nne = 'NNE',
    Nw = 'NW',
    Wsw = 'WSW',
  }

```

- En api-gateway, la conexión con los microservicios se hace mediante el controller
- flight.dto

```

import { IsNotEmpty, IsString } from 'class-validator';
import { ApiProperty } from '@nestjs/swagger';
export class FlightDTO {
  @ApiProperty()
  @IsNotEmpty()
  @IsString()
  pilot: string;

  @ApiProperty()
  @IsNotEmpty()
  @IsString()

```

```

    airplane: string;

    @ApiProperty()
    @IsNotEmpty()
    @IsString()
    destinationCity: string;

    @ApiProperty()
    @IsNotEmpty()
    @IsString()
    flightDate: Date;
}

```

- flight.controller (api-gateway). Inyectamos el clientProxy que necesitamos de ./client-proxy

```

import { FlightMSG, PassengerMSG } from '../common/constants';
import { FlightDTO } from '../dto/flight.dto';
import { Observable } from 'rxjs';
import { ClientProxySuperFlights } from '../common/proxy/client-proxy';
import {
    Body,
    Controller,
    Delete,
    Get,
    HttpException,
    HttpStatus,
    Param,
    Post,
    Put,
    UseGuards,
} from '@nestjs/common';
import { IFlight } from 'src/common/interfaces/flight.interface';
import { ApiTags } from '@nestjs/swagger';
import { JwtAuthGuard } from 'src/auth/guards/jwt-auth.guard';

@ApiTags('flights')
@UseGuards(JwtAuthGuard)
@Controller('api/v2/flight')
export class FlightController {
    constructor(private readonly clientProxy: ClientProxySuperFlights) {}

    private _clientProxyFlight = this.clientProxy.clientProxyFlights();
    private _clientProxyPassenger = this.clientProxy.clientProxyPassengers();

    @Post()
    create(@Body() flightDTO: FlightDTO): Observable<IFlight> {
        return this._clientProxyFlight.send(FlightMSG.CREATE, flightDTO);
    }

    @Get()
    findAll(): Observable<IFlight[]> {
        return this._clientProxyFlight.send(FlightMSG.FIND_ALL, '');
    }
}

```

```

    }

    @Get('/:id')
    findOne(@Param('id') id: string): Observable<IFlight> {
        return this._clientProxyFlight.send(FlightMSG.FIND_ONE, id);
    }

    @Put('/:id')
    update(
        @Param('id') id: string,
        @Body() flightDTO: FlightDTO,
    ): Observable<IFlight> {
        return this._clientProxyFlight.send(FlightMSG.UPDATE, { id, flightDTO });
    }

    @Delete('/:id')
    delete(@Param('id') id: string): Observable<any> {
        return this._clientProxyFlight.send(FlightMSG.DELETE, id);
    }

    @Post('/:flightId/passenger/:passengerId')
    async addPassenger(
        @Param('flightId') flightId: string,
        @Param('passengerId') passengerId: string,
    ) {
        const passenger = await this._clientProxyPassenger
            .send(PassengerMSG.FIND_ONE, passengerId)
            .toPromise();

        if (!passenger)
            throw new HttpException('Passenger Not Found', HttpStatus.NOT_FOUND);

        return this._clientProxyFlight.send(FlightMSG.ADD_PASSENGER, {
            flightId,
            passengerId,
        });
    }
}

```

- En flight.module debemos importar el ProxyModule

```

import { ProxyModule } from '../common/proxy/proxy.module';
import { Module } from '@nestjs/common';
import { FlightController } from './flight.controller';

@Module({
    imports: [ProxyModule],
    controllers: [FlightController],
})
export class FlightModule {}

```

- En passenger.dto

```
import { ApiProperty } from '@nestjs/swagger';
import { IsNotEmpty, IsString, IsEmail } from 'class-validator';

export class PassengerDTO {
  @ApiProperty()
  @IsNotEmpty()
  @IsString()
  name: string;

  @ApiProperty()
  @IsNotEmpty()
  @IsEmail()
  email: string;
}
```

- En passenger.controller lo mismo, inyectamos a través de clientProxy el servicio que necesitamos en el controlador

```
import {
  Body,
  Controller,
  Delete,
  Get,
  Param,
  Post,
  Put,
} from '@nestjs/common';
import { PassengerMSG } from '../common/constants';
import { PassengerDTO } from '../dto/passenger.dto';
import { Observable } from 'rxjs';
import { ClientProxySuperFlights } from '../common/proxy/client-proxy';
import { IPassenger } from 'src/common/interfaces/passenger.interface';
import { ApiTags } from '@nestjs/swagger';
import { JwtAuthGuard } from 'src/auth/guards/jwt-auth.guard';
import { UseGuards } from '@nestjs/common';

@ApiTags('passengers')
@UseGuards(JwtAuthGuard)
@Controller('api/v2/passenger')
export class PassengerController {
  constructor(private readonly clientProxy: ClientProxySuperFlights) {}
  private _clientProxyPassenger = this.clientProxy.clientProxyPassengers();

  @Post()
  create(@Body() passengerDTO: PassengerDTO): Observable<IPassenger> {
    return this._clientProxyPassenger.send(PassengerMSG.CREATE, passengerDTO);
  }
}
```



```

@Get()
findAll(): Observable<IPassenger[]> {
  return this._clientProxyPassenger.send(PassengerMSG.FIND_ALL, '');
}

@Get('/:id')
findOne(@Param('id') id: string): Observable<IPassenger> {
  return this._clientProxyPassenger.send(PassengerMSG.FIND_ONE, id);
}

@Put('/:id')
update(
  @Param('id') id: string,
  @Body() passengerDTO: PassengerDTO,
): Observable<IPassenger> {
  return this._clientProxyPassenger.send(PassengerMSG.UPDATE, {
    id,
    passengerDTO,
  });
}

@Delete('/:id')
delete(@Param('id') id: string): Observable<any> {
  return this._clientProxyPassenger.send(PassengerMSG.DELETE, id);
}
}

```

- user.dto

```

import { ApiProperty } from '@nestjs/swagger';
import { IsEmail, IsNotEmpty, IsString } from 'class-validator';

export class UserDTO {
  @ApiProperty()
  @IsNotEmpty()
  @IsString()
  readonly name: string;
  @ApiProperty()
  @IsNotEmpty()
  @IsString()
  readonly username: string;
  @ApiProperty()
  @IsNotEmpty()
  @IsEmail()
  readonly email: string;
  @ApiProperty()
  @IsNotEmpty()
  @IsString()
  readonly password: string;
}

```

- user.controller

```
import { ApiTags } from '@nestjs/swagger';
import {
  Body,
  Controller,
  Delete,
  Get,
  Param,
  Post,
  Put,
  UseGuards,
} from '@nestjs/common';
import { UserMSG } from '../common/constants';
import { Observable } from 'rxjs';
import { UserDTO } from '../dto/user.dto';
import { ClientProxySuperFlights } from '../common/proxy/client-proxy';
import { IUser } from 'src/common/interfaces/user.interface';
import { JwtAuthGuard } from 'src/auth/guards/jwt-auth.guard';

@ApiTags('users')
@UseGuards(JwtAuthGuard)
@Controller('api/v2/user')
export class UserController {
  constructor(private readonly clientProxy: ClientProxySuperFlights) {}
  private _clientProxyUser = this.clientProxy.clientProxyUsers();

  @Post()
  create(@Body() userDTO: UserDTO): Observable<IUser> {
    return this._clientProxyUser.send(UserMSG.CREATE, userDTO);
  }

  @Get()
  findAll(): Observable<IUser[]> {
    return this._clientProxyUser.send(UserMSG.FIND_ALL, '');
  }

  @Get('/:id')
  findOne(@Param('id') id: string): Observable<IUser> {
    return this._clientProxyUser.send(UserMSG.FIND_ONE, id);
  }

  @Put('/:id')
  update(@Param('id') id: string, @Body() userDTO: UserDTO): Observable<IUser> {
    return this._clientProxyUser.send(UserMSG.UPDATE, { id, userDTO });
  }

  @Delete('/:id')
  delete(@Param('id') id: string): Observable<any> {
    return this._clientProxyUser.send(UserMSG.DELETE, id);
  }
}
```

