

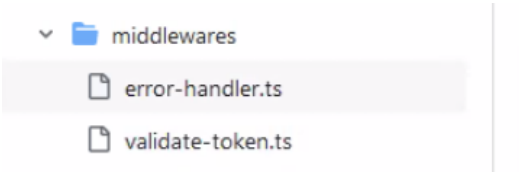
# Proyecto final

## Requisitos

- ☐ Identificar uso de patrones de diseño y describir cómo se implementan
- ☐ Identificar uso de pilares de programación orientada a objetos y describir cómo se implementan
- ☐ Identificar uso de principios SOLID y describir cómo se implementan
- ☐ Evidencias de clean architecture de los códigos implementados

## Patrones de Diseño

<pre>const container = new Container({ defaultScope: 'Singleton' });</pre>	<p>Este código forma parte del archivo <i>Inversify</i> donde se establecen dependencias para otros archivos (sign-in, sign-up, base-controller...). El patrón <b>singleton</b> lo podemos ver aplicado cuando se declara la instancia de una variable, siendo única, sin modificaciones ni duplicaciones.</p>
<pre>export interface CacheService {   getByKey&lt;T&gt;(key: string): Promise&lt;T   null&gt;;  export class RedisImpl implements CacheService {    async getKeyTLL(key: string): Promise&lt;number&gt; {     const client = getRedisClient();     return await client.ttl(key);   } }</pre>	<p>Este código forma parte del archivo <i>CacheService</i>, donde identificamos el patrón <b>adapater</b> ya que se definen métodos y atributos, los cuales después se asignan en el archivo <i>redis-impl</i>.</p>
<pre>export abstract class BaseController {   public abstract execute()  export class UserSignupController extends BaseController {  export class UserSigninController extends BaseController {</pre>	<p>Con los archivos <i>sign-in</i>, <i>sign-up</i> y <i>base-controller</i> también observamos el patrón <b>adapter</b> ya que el <i>base-controller</i> se divide para aplicarse en los <i>sign</i>.</p>
<pre>12 13   userRouter.post("/signin", (req, res) =&gt; signin.execute(req, res)); 14 15   userRouter.post("/signup", (req, res) =&gt; signup.execute(req, res)); 16</pre>	<p>Sabemos que un router es un administrador de rutas por lo que funciona como el patrón <b>proxy</b>, ya que se encarga de procesar la petición.</p>

	<p>Un <i>Middleware</i> como su nombre lo dice es un mensajero intermedio por lo tanto, también funciona como <b>proxy</b>. Tanto <i>error-handler</i> y <i>validate-token</i>.</p>
<pre> 17  export const errorHandler = ( 18    error: CustomError, 19    req: CustomRequest, 20    res: Response 21  ) =&gt; { 22    try { 23      console.error(error, req.requestId); 24      const translatedMessage = error.message    defaults.ERROR_MESSAGE; 25      const response = buildResponse(error, translatedMessage); 26      return res 27        .status(error.statusCode    HTTP_STATUS_INTERNAL_SERVER_ERROR) 28        .json(response); 29    } catch (err) { 30      return res.status(HTTP_STATUS_INTERNAL_SERVER_ERROR).json(err); 31    } 32  }; </pre>	<p>El código que se muestra proviene de <i>error-handler</i>, recibe una petición y una respuesta. En la línea 18 podemos ver que recibe un error, busca cuál es y regresa un mensaje que es entendible por el usuario. Este archivo contiene el patrón <b>chain of responsibility</b> en su manejo para encontrar la solución para que el servidor deje de funcionar.</p>
<pre> import type { Config } from './types'; const config: Config = {   environment: process.env.ENVIRONMENT    "local",   port: +process.env.PORT    8080, </pre>	<p>El patrón <b>memento</b> se puede observar dentro del archivo <i>index.ts</i> que se encuentra en <i>config</i>, el cual está en <i>platform</i> en <i>src</i>. Esto es debido a que al inicio del programa se genera una instancia de configuración default que no se cambia a menos de que se detenga la aplicación.</p>

# Pilares de la POO

<pre>✓ export interface IAuthService {   generateToken(data: any): any;   verifyToken(token: string): any;   matchPassword(password, hash);   hashPassword(plainPassword): string; }  export class AuthService implements IAuthService {   generateToken(data: any): any {</pre>	<p>Como ya se mencionó anteriormente, en <i>IAuthService</i> se define lo que contiene y en <i>AuthService</i> siguiendo esas mismas reglas se escribe el código, aquí es donde se aplica la <b>herencia</b> y el <b>polimorfismo</b>.</p>
<pre>3 4   let redisClient; 5 6   export function getRedisClient(): Redis.Redis { 7     return redisClient; 8   } 9 10  ✓ export async function connectRedisClient( 11    options?: Redis.RedisOptions, 12    requestId?: string   null 13  ): Promise&lt;void&gt; { 14 15    let opts = { 16      port: &lt;config&gt;.cache.port, 17      host: &lt;config&gt;.cache.host, 18      password: &lt;config&gt;.cache.password, 19    }; 20    redisClient = new Redis(opts); 21  }</pre>	<p>El <b>encapsulamiento</b> se puede observar dentro del archivo <i>redis</i> en la carpeta db, ya que existe el valor <i>redisClient</i>, mas sólo se puede acceder a través de su método.</p>
<pre>public async execute(requestDto: requestDto): Promise&lt;responseDto&gt; {   try {     const { userName, password } = requestDto;     const user = await this.userRepository.getUserByUserName(userName);     await this.authService.matchPassword(password, user.password);     const generateToken = this.authService.generateToken(user);      return Promise.resolve({       token: generateToken,     });   } catch (error) {     throw new Error(error?.message);   } }</pre>	<p>En <i>sign-in</i> y <i>sign-up</i> podemos ver la aplicación del <b>polimorfismo</b> debido a que se usa la misma función <u><b>execute</b></u> pero se ejecuta de manera distinta.</p>

# Principios SOLID

<pre> ✓ export interface IAuthService {   generateToken(data: any): any;   verifyToken(token: string): any;   matchPassword(password, hash);   hashPassword(plainPassword): string; }  export class AuthService implements IAuthService {   generateToken(data: any): any {  export class Signup implements SignupUseCase {   private readonly userRepository: UserRepository;   private readonly authService: AuthService; </pre>	<p><i>AuthService</i> exporta una interfaz que contiene cuatro funciones para la autenticación.</p> <ul style="list-style-type: none"> <li>• <i>auth-service</i> exigente la interfaz del primer <i>AuthService</i> y le da el desarrollo a sus funciones</li> <li>• Las mismas funciones se utilizan en <i>sign-up</i>, porque llaman a la misma implementación de <i>auth-service</i></li> <li>• Con esto podemos concluir que se usa el principio <b>open/closed</b> ya que se usa la interfaz <i>auth-service</i>, pero no se modifica</li> <li>• Además hay una implementación de <i>CacheService</i> dentro de <i>redis-impl</i></li> </ul>
<pre> export interface CacheService {   getByKey&lt;T&gt;(key: string): Promise&lt;T   null&gt;;  export class RedisImpl implements CacheService {    async getKeyTLL(key: string): Promise&lt;number&gt; {     const client = getRedisClient();     return await client.ttl(key);   } </pre>	<p>Identificamos <b>dependency inversion</b> dentro del archivo <i>CacheService</i> ya que esté proporciona métodos para manejar el cache.</p> <p>RedisImpl es la dependencia.</p>
<pre> -- 21 ✓ async getUserByUsername(userName: string): Promise&lt;any&gt; { 22   const results = await sql({ 23     query: ` 24       SELECT 25         usr_act_id as "userId", 26         usr_act_name as "userName", 27         usr_act_password as password 28       FROM user_account 29       where usr_act_name = \$1 30     `, 31     bind: [userName], 32   }) as any; 33 34   if (results.rowCount) { 35     return results.rows[0]; 36   } 37 38   return null; 39 } </pre>	<p>En <i>user-repository</i> identificamos <b>single responsibility</b> ya que contiene varias funciones que cumplen con responsabilidades específicas y únicas.</p>

## Clean architecture

Entities	Estructuras de datos u objetos como <i>User</i> .
Use cases	Inicio de sesión o registro
Interface adapters	Cómo se ejecutan los <i>use cases</i> .
Controllers	Dirige la petición del <i>use case</i> al código que la procesa. <i>base-controllers</i> , <i>router</i> , <i>sign-up</i> , <i>sign-in</i> .
Gateways	Código que interactúa con las bases de datos o interfaces externas para ejecutar los <i>use cases</i> , como los <i>sql</i> .
Presenters	Toma la info después de ejecutar el <i>use case</i> y lo empaqueta en algo que pueda utilizarse por medio de una interfaz. Un ejemplo es el <i>error-handler</i> .
Frameworks & drivers	La capa externa de la aplicación se puede modificar, sobre las que la aplicación opera. Como una interfaz o una base de datos.