PROYECTO

DISEÑO DE INTERFACES

HUGO IVAN ROMERO DUARTE

ENTIQUE AGUILAR

18/05/24

FACULTAD DE INFOMRATICA

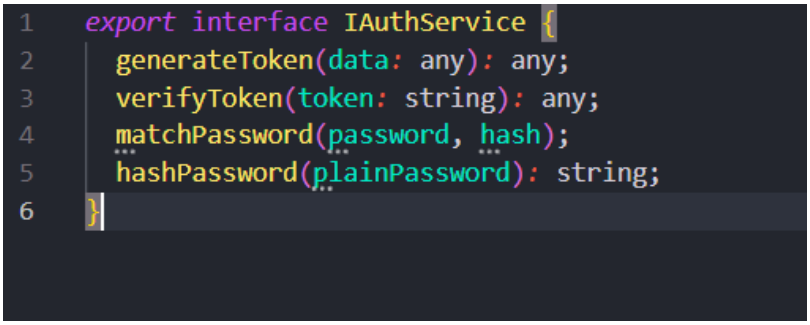
DISEÑO DE INTERFACES

SRC

Comon

-Patrones seguros

Stratergy: En la carpeta Config el archivo Auth-Service declara la interfaz IAuthService, la cual a su vez declara varios métodos



```
export interface IAuthService {
  generateToken(data: any): any;
  verifyToken(token: string): any;
  matchPassword(password, hash);
  hashPassword(plainPassword): string;
}
```

A su vez estos metodos se utilizan en en archivo auth-service.

```
import bcrypt from 'bcryptjs';

import { config } from "@example-api/platform/index";
import { IAuthService } from './AuthService';
import { injectable } from 'inversify';
@injectable()
export class AuthService implements IAuthService {
  generateToken(data: any): any {
    return jwt.sign({ data }, config.auth.secret, {
      expiresIn: "24h",
    });
  }

  verifyToken(token: string): any {
    try {
      return jwt.verify(token, config.auth.secret);
    } catch (err) {
      return false;
    }
  }

  async matchPassword(password, hash) {
    return await bcrypt.compare(password, hash);
  }

  hashPassword(plainPassword): string {
    const saltRounds = config.auth.salts;
    const salt = bcrypt.genSaltSync(saltRounds);
    const hash = bcrypt.hashSync(plainPassword, salt);

    return hash;
  }
```

-Patrones no tan seguros

Singelton: El patron singelton creo que se puede ver en las instancia de AuthService ya que permite que el código tenga acceso global a esta

```ts
export interface IAuthService {
  generateToken(data: any): any;
  verifyToken(token: string): any;
  matchPassword(password, hash);
  hashPassword(plainPassword): string;
}
```

```ts
import bcrypt from 'bcryptjs';

import { config } from "@example-api/platform/index";
import { IAuthService } from './AuthService';
import { injectable } from 'inversify';
@injectable()
export class AuthService implements IAuthService {
  generateToken(data: any): any {
    return jwt.sign({ data }, config.auth.secret, {
      expiresIn: "24h",
    });
  }

  verifyToken(token: string): any {
    try {
      return jwt.verify(token, config.auth.secret);
    } catch (err) {
      return false;
    }
  }

  async matchPassword(password, hash) {
    return await bcrypt.compare(password, hash);
  }

  hashPassword(plainPassword): string {
    const saltRounds = config.auth.salts;
    const salt = bcrypt.genSaltSync(saltRounds);
    const hash = bcrypt.hashSync(plainPassword, salt);

    return hash;
  }
```

Config

-Patrones seguros

Singleton: Se utiliza en un archivo llamado index en la carpeta inversify en el cual crea una instancia llamada container, además de que da acceso global a esta:

```
const container = new Container({ defaultScope: 'Singleton' });

container.bind<CacheService>(SYMBOLS.CacheService).to(RedisImpl);

container.bind<IAuthService>(SYMBOLS.AuthService).to(AuthService);

container.bind<BaseController>(BaseController).toSelf();

container
  .bind<UserSignupController>(UserSignupController)
  .toSelf();
container
  .bind<UserSigninController>(UserSigninController)
  .toSelf();

container.bind<SignupUseCase>(SYMBOLS.SignupUseCase).to(Signup);
container.bind<SigninUseCase>(SYMBOLS.SigninUseCase).to(Signin);

container.bind<UserRepository>(SYMBOLS.UserRepository).to(UserRepositoryImpl);


export { container };
```

Stratergy: Se utiliza también en el archivo index de la carpeta inversify, en esta se definen dos métodos que ya había visto en la carpeta de Common que son Cacheservice y AuthService

```
container.bind<CacheService>(SYMBOLS.CacheService).to(RedisImpl);

container.bind<IAuthService>(SYMBOLS.AuthService).to(AuthService);
```

Observer: En el archivo llamado redis de la carpeta db Se puede ver en el código ya que se notifica cuando se conecta y cuando se produce el error

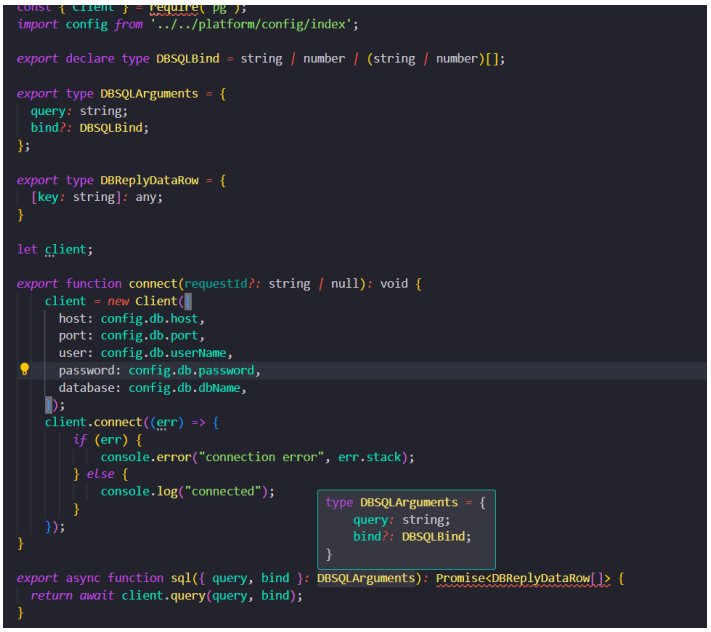
```
redisClient.on("connect", () => {
  console.info(
    "connect-redis-client",
    requestId || "not requestId was provided",
  );
});
redisClient.on("error", async (error) => {
  console.error(
    error,
    "connect-redis-client",
    requestId || "not requestId was provided",
    "cache-sdk"
  );
});
}
```

-Patrones no tan seguros

Singleton: En el archivo llamado redis se declara la variable RedisClient la cual se reutiliza a lo largo de el código

```
import Redis from "ioredis";
import config from "../../platform/config/index";

let redisClient;

export function getRedisClient(): Redis.Redis {
    return redisClient;
}

export async function connectRedisClient(
  options?: Redis.RedisOptions,
  requestId?: string | null
): Promise<void> {

    let opts = {
      port: +config.cache.port,
      host: config.cache.host,
      password: config.cache.password,
    };
    redisClient = new Redis(opts);

    redisClient.on("connect", () => {
      console.info(
        "connect-redis-client",
        requestId || "not requestId was provided",
      );
    });
    redisClient.on("error", async (error) => {
      console.error(
        error,
        "connect-redis-client",
        requestId || "not requestId was provided",
        "cache-sdk"
      );
    });
}
```

Así como también en el archivo llamado sql se declara la variable client la cual se utiliza nuevamente en todo el archivo


const { Client } = require('pg');
import config from '../../platform/config/index';
export declare type DBSQLBind = string | number | (string | number)[];
export type DBSQLArguments = {
query: string;
bind?: DBSQLBind;
};
export type DBReplyDataRow = {
[key: string]: any;
}
let client;
export function connect(requestId?: string | null): void {
client = new Client({
host: config.db.host,
port: config.db.port,
user: config.db.userName,
password: config.db.password,
database: config.db.dbName,
});
client.connect((err) => {
if (err) {
console.error("connection error", err.stack);
} else {
console.log("connected");
}
});
}
type DBSQLArguments = {
query: string;
bind?: DBSQLBind;
}
export async function sql({ query, bind }: DBSQLArguments): Promise<DBReplyDataRow[]> {
return await client.query(query, bind);
}

Constants

No encontre ningun patron de diseño :(

Controllers

-Patrones seguros

Stratergy: Este se encuentra en sing.in y en sing.up ya que implementan una clase llamada UserSingInController y UserSingInController las cuales necesitan de la instancia SingupUseCase y SinginUseCase

```
@injectable()
export class UserSigninController extends BaseController {
  private signin: SigninUseCase;

  public constructor(
    @inject(SYMBOLS.SigninUseCase)
    signin: SigninUseCase
  ) {
    super();
    this.signin = signin;
  }
```

```
@injectable()
export class UserSignupController extends BaseController {
  private signup: SignupUseCase;

    public constructor(
      @inject(SYMBOLS.SignupUseCase)
      signup: SignupUseCase
    ) {
      super();
      this.signup = signup;
    }
```

-Patrones no tan seguros

Singleton: La identifique en la parte de @injectable la cual hace una “inyección de dependencias” lo que lo hace muy parecido a singleton

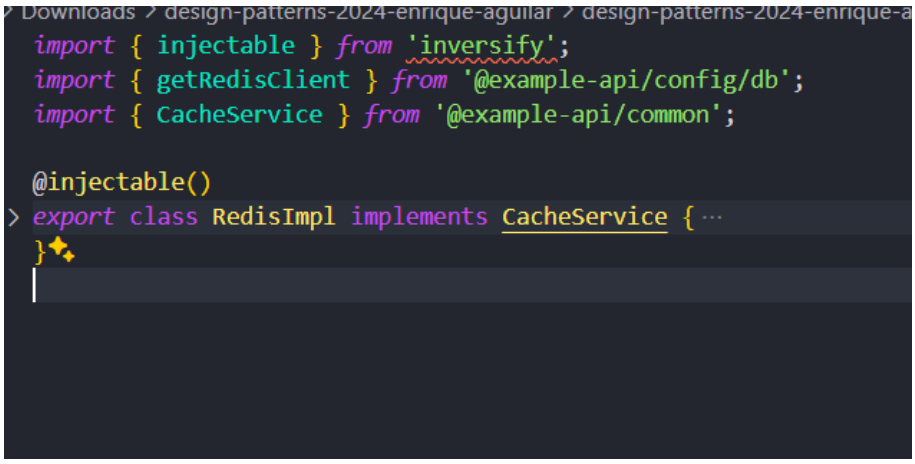
```
@injectable()
export class UserSigninController extends BaseController {
  private signin: SigninUseCase;

  public constructor(
    @inject(SYMBOLS.SigninUseCase)
    signin: SigninUseCase
  ) {
    super();
    this.signin = signin;
  }
```

```typescript
@injectable()
export class UserSigninController extends BaseController {
  private signin: SigninUseCase;

  public constructor(
    @inject(SYMBOLS.SigninUseCase)
    signin: SigninUseCase
  ) {
    super();
    this.signin = signin;
  }
```

Data-Acces

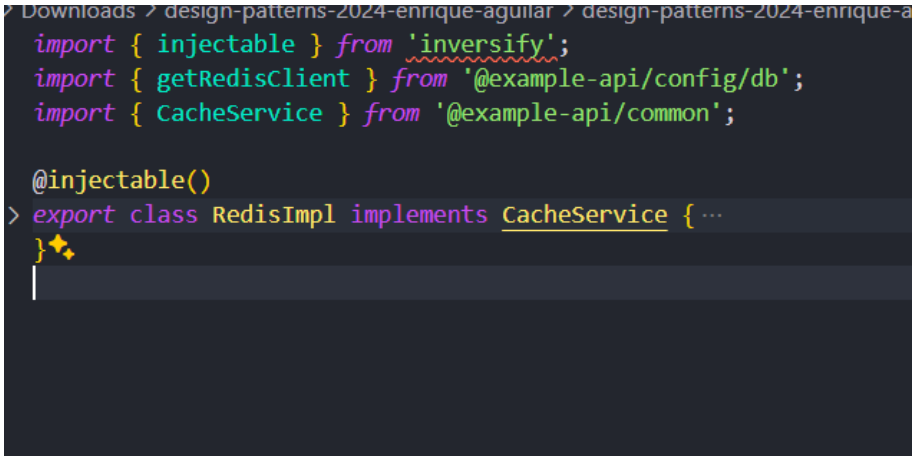
-Patrones seguros

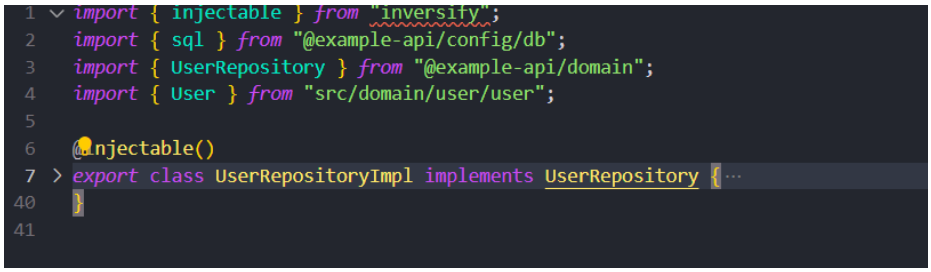
Stratergy: Se puede observar como en el archivo user-repository se implementa la interfaz UserRepository asi como también en el archivo redis-impl se implementa la interfaz CacheService lo que hace que tengan que implementar sus propios métodos:



-Patrones no tan seguros

Singleton: Nuevamente me encuentro la implementación @injectable lo que hace que se “Inyecten dependencias”

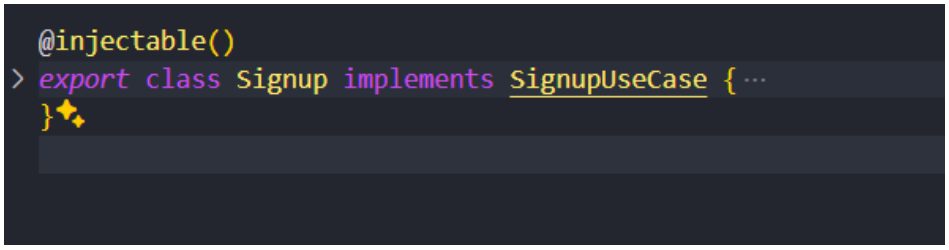


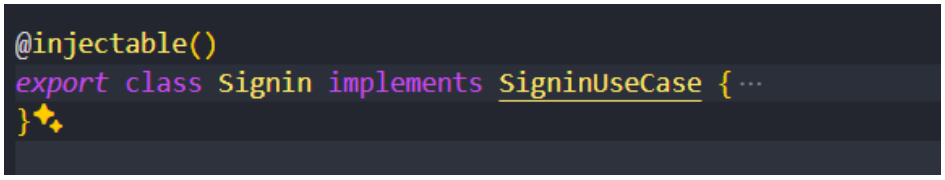

import { injectable } from "inversify";
import { sql } from "@example-api/config/db";
import { UserRepository } from "@example-api/domain";
import { User } from "src/domain/user/user";
@injectable()
export class UserRepositoryImpl implements UserRepository {
}

Domain

-Patrones seguros

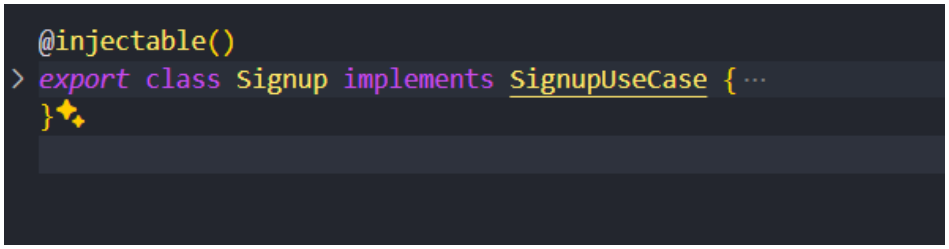
Stratergy: Ya que tanto la clase siginn como signup implementan una clase SigninUseCase y SignupUseCase lo que los hacer usar los métodos de las mismas

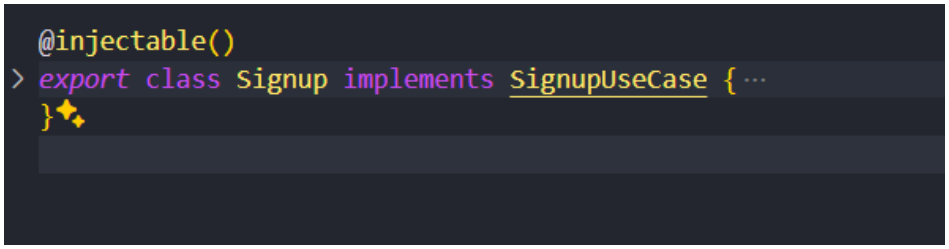




-Patrones no tan seguros

Singleton: Nuevamente encontramos la implementación @inyectable usada en las clases Singnin y singnup





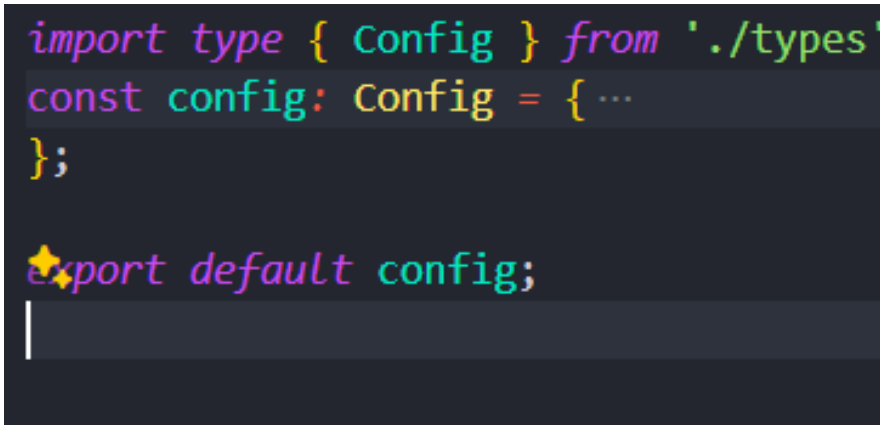
Interfaces

No encontre ningun patron de diseño ·͡⌒·

Platform

-Patrones seguros

Singleton:En el archivo index se crea una constante llamada config la cual se utiliza en todo el código



Routes

-Patrones seguros

Singleton: En este caso en el archivo user podemos ver como se implementan las clases signup y signin las cuales posteriormente se reutilizan en el código:

```
const signup = container.get(UserSignupController);
const signin = container.get(UserSigninController);

const userRouter = Router();

userRouter.post("/signin", (req, res) => signin.execute(req, res));

userRouter.post("/signup", (req, res) => signup.execute(req, res));
```

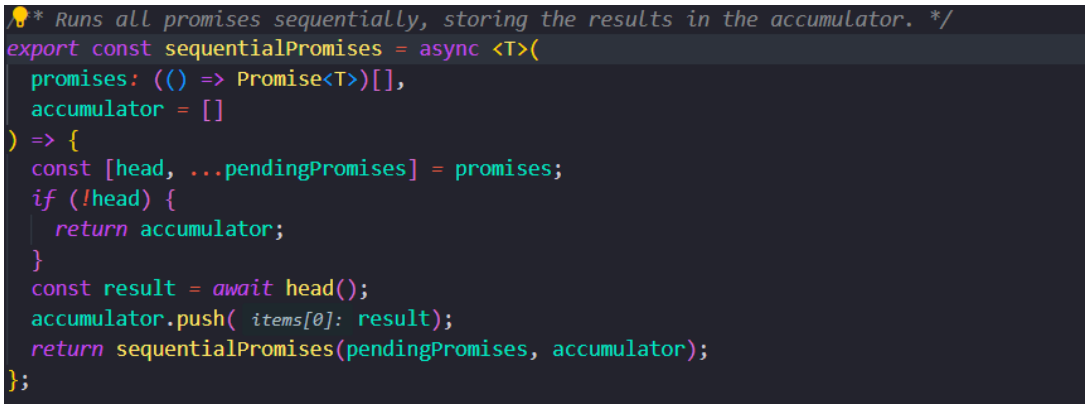
Stratergy: Notamos que tanto la función signin.execute como signup.execute se utilizan para manejar el inicio y registro de sesión

```
const signup = container.get(UserSignupController);
const signin = container.get(UserSigninController);

const userRouter = Router();

userRouter.post("/signin", (req, res) => signin.execute(req, res));

userRouter.post("/signup", (req, res) => signup.execute(req, res));
```

Util

-Patrones seguros

Chain Responsability: Lo podemos ver en la función sequentialPromises ya que ejecuta una serie de promesas secuencialmente



```
/** Runs all promises sequentially, storing the results in the accumulator. */
export const sequentialPromises = async <T>(
  promises: (() => Promise<T>)[],
  accumulator = []
) => {
  const [head, ...pendingPromises] = promises;
  if (!head) {
    return accumulator;
  }
  const result = await head();
  accumulator.push( items[0]: result);
  return sequentialPromises(pendingPromises, accumulator);
};
```