



Universidad Autónoma de Querétaro Facultad de Informática

Carrera: Ingeniería de Software

Patrones de Diseño

Parcial 2

Proyecto Final

Manuel Iván Gerardo Rosas

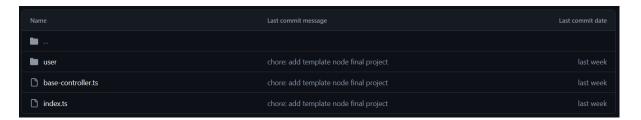
Docente: Enrique Aguilar Orozco

Expediente: 325748

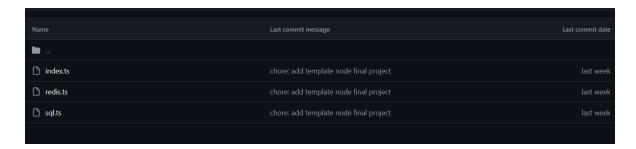
Fecha: 18/05/2024

Clean Architecture

En otra capa podemos encontrar una carpeta destinada a los controladores.



Se tiene en la parte externa de las capas una carpeta destinada a la base de Datos.

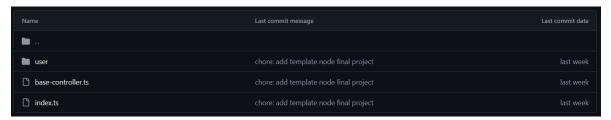


- Patrones de Diseño
 - Creacionales
 - Estructurales
 - Proxy:
 - Se encontró en el documento Base Controller.TS
 en la carpeta"Controllers", en "SRC", ya que se
 envía una solicitud, y dependiendo del acceso
 manda una respuesta.

```
import { injectable } from 'inversify';
import { errorHandler } from '@example-api/platform/middlewares/error-handler';
    ort { CustomError } from '@example-api/platform/lib/class/general-error';
import { CustomRequest } from '../platform';
@injectable()
export abstract class BaseController {
   request: Request,
   response: Response
  ): Promise<Response>;
 protected ok<T>(
   req: CustomRequest,
   httpCode: number,
  ): Response {
   console.info(
     `[END] - Path: ${req.originalUrl}`,
     rea.reauestId
   if (dto) {
     res.type('application/json');
      return res.status(httpCode).json(dto);
   return res.sendStatus(httpCode);
```

A su vez se encontró en la carpeta SRC – Controllers – User, tanto en los documentos Sign in como Sign up como sign in

- De comportamiento
- Logica de Negocio
- Controladores, presentadores
 - Existe una carpeta llamada "Controllers"



- POO
 - o Polimorfismo
 - Herencia

Se encontró en la carpeta SRC – Controllers – User, tanto en los documentos Sign in como Sign up como sign in

```
const {
   HTTP_STATUS_CREATED,
   HTTP_STATUS_BAD_REQUEST,
   HTTP_STATUS_INTERNAL_SERVER_ERROR,
} = constants;

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

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

```
async execute(request: CustomRequest, response: Response): Promise<Response> {
 const { body } = request;
 const inputDto = {
   ...body,
   requestId: request.requestId,
 const { userName, password} = body;
 if(!userName || !password){
   throw new Error("missing fields");
   const dataDto = await this.signup.execute(inputDto);
   return this.ok(request, response, HTTP_STATUS_CREATED, dataDto);
 } catch (error) {
   return this.fail(
    request,
    response,
    HTTP_STATUS_INTERNAL_SERVER_ERROR,
     error
```

- Abstracción
- Encapsulación
- Capas
- Principios Solid
 - Single Responsability

- En general en todo el proyecto la mayoría de los objetos y métodos tienen una sola responsabilidad
- Open/Close: Se encontró este principio en la carpeta "Base Controller" permitiendo a la clase"Base Controller" extenderse pero lo derivado de ella no puede ser modificado.

```
export abstract class BaseController {
 public abstract execute(
   request: Request,
   response: Response
 ): Promise<Response>;
 protected ok<T>(
   req: CustomRequest,
   res: Response,
   httpCode: number,
   dto?: T
 ): Response {
   console.info(
     `[END] - Path: ${req.originalUrl}`,
     BaseController.name,
     req.requestId
   if (dto) {
     res.type('application/json');
```

Liskov Substitution principle
 En el archivo Base Controller.ts, dependiendo del tipo de objeto
 que sea, cambiará el atributo Nombre.

- o Interface Segregation principle
- Dependency inversión principle