
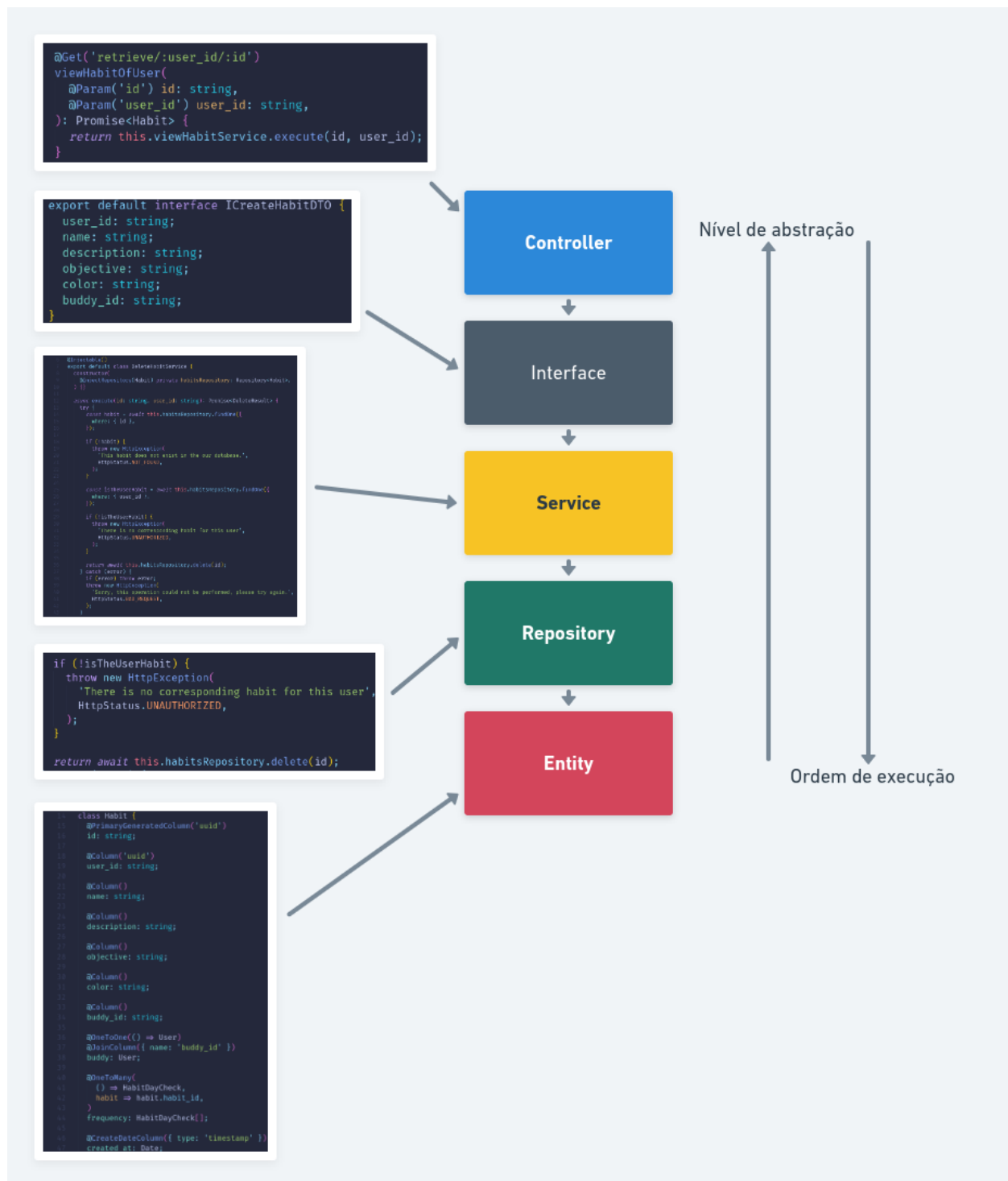


Padrão de Acesso aos Dados

 Property	
--	--



Controller: São responsáveis por lidar com as solicitações recebidas e retornar as respostas ao cliente. Funciona como um intermediário entre o cliente e o servidor.

```

@Controller('habit')
export default class HabitController {
  constructor(
    private createHabitService: CreateHabitService,
    private updateHabitService: UpdateHabitService,
    private deleteHabitService: DeleteHabitService,
    private listHabitsService: ListHabitsService,
    private viewHabitService: ViewHabitService,
    private getCurrentWeekFrequency: GetCurrentWeekFrequency,
  ) {}

  @Get('/habit')
  getHabit(@HabitDecorator() habit) {
    return habit;
  }

  @Post()
  createHabit(
    @Body()
    { user_id, name, description, objective, color, buddy_id }: ICreateHabitDTO,
  ): Promise<Habit> {
    return this.createHabitService.execute({
      user_id,
      name,
      description,
      objective,
      color,
      buddy_id,
    });
  }
}

```



Interface: São responsáveis por lidar com as solicitações recebidas e retornar as respostas ao cliente. Funciona como um intermediário entre o cliente e o servidor.

```

export default interface ICreateHabitDTO {
  user_id: string;
  name: string;
  description: string;
  objective: string;
  color: string;
  buddy_id: string;
}

```



Service: Será responsável pelo armazenamento e recuperação dos dados, e é projetado para ser usado pelo Controller.

```
@Injectable()
export default class CreateHabitService {
  constructor(
    @InjectRepository(Habit) private habitsRepository: Repository<Habit>,
    @InjectRepository(User) private usersRepository: Repository<User>,
  ) {}

  async execute({
    user_id,
    name,
    description,
    objective,
    color,
    buddy_id,
  }: ICreateHabitDTO): Promise<Habit> {
    try {
      const user = await this.usersRepository.findOne({
        where: { id: user_id },
      });

      if (!user)
        throw new HttpException(
          'It is not possible to perform the operation, as there is no corresponding registered user',
          HttpStatus.NOT_FOUND,
        );

      const habit = this.habitsRepository.create({
        user_id,
        name,
        description,
        objective,
        color,
        buddy_id,
      });

      await this.habitsRepository.save(habit);

      return habit;
    }
  }
}
```



Repository: No TypeORM, o repository é o responsável por acessar os dados, é uma abstração criada para que o service não acesse diretamente o banco de dados.

```
const habit = this.habitsRepository.create({
  user_id,
  name, | Italo Lima, 4 months ago • refac
  description,
  objective,
  color,
  buddy_id,
});
```

```
constructor(
  @InjectRepository(Habit) private habitsRepository: Repository<Habit>,
  @InjectRepository(User) private usersRepository: Repository<User>,
) {}
```



Entity: Entidade que representa a abstração da tabela do banco de dados a qual se refere, contém as propriedades e seus referentes tipos. Além disso, no TypeORM é usada para descrever o(s) relacionamento(s) com outra(s) entidade(s)

```

@Entity('habits')
class Habit {
    @PrimaryGeneratedColumn('uuid')
    id: string;

    @Column('uuid')
    user_id: string;

    @Column()
    name: string;

    @Column()
    description: string;

    @Column()
    objective: string;

    @Column()
    color: string;

    @Column()
    buddy_id: string;

    @OneToOne(() => User)
    @JoinColumn({ name: 'buddy_id' })
    buddy: User;

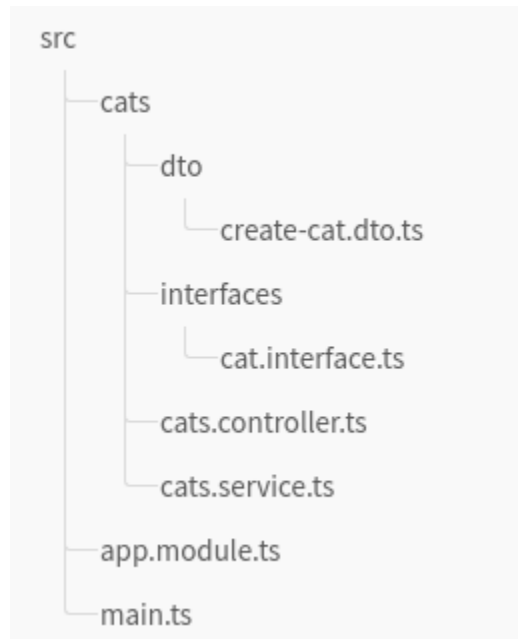
    @OneToMany(
        () => HabitDayCheck,
        habit => habit.habit_id,
    )
    frequency: HabitDayCheck[];

    @CreateDateColumn({ type: 'timestamp' })
    created_at: Date;

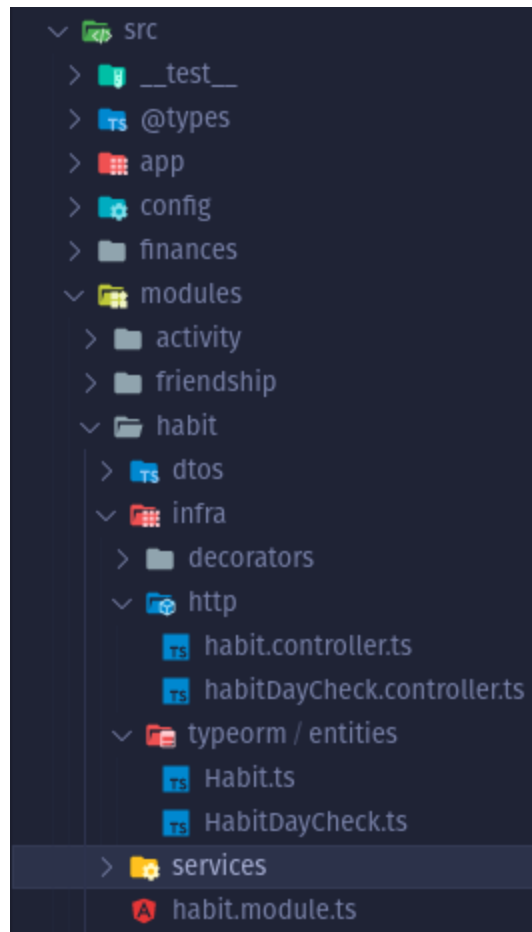
    @CreateDateColumn({ type: 'timestamp' })
    updated_at: Date;

```

Estrutura de pastas do projeto Nest:



Documentação do nest



Nosso projeto

Estrutura:

- src
 - modules
 - habit
 - dtos
 - interfaces
 - CreateHabit.ts
 - infra
 - http
 - habit.controller.ts

- typeorm
 - entities
 - Habit.ts
- services
 - habit.service.ts

Resumo + abstração:

O controller (vamos chamá-lo de ponte), faz o intermédio entre o cliente (usuário que faz a requisição) e o servidor (nosso back-end). A partir da requisição do cliente, o controller vai acessar o service responsável pela execução de tal funcionalidade, o service, por sua vez, não pode acessar os dados do cliente diretamente, pois esses estão protegidos por um guardião (repository), que os armazena de maneira lógica, assim, o service requisita ao repository o desejo de acessar tais dados (varia de acordo com o escopo de cada funcionalidade), o repository irá então coletar as informações desejadas e as retornas para o service, que por sua vez, as utilizará para executar a ação desejada, essa ação gera uma resposta, que é "transformada" pelo service e devolvida em um formato mais legível (interface) para o controller, que então, entrega ao cliente que a requisitou.