GRUPO 07

INTEGRANTES

BRUNA LUIZA DA SILVA GONÇALVES
DANIEL ALMEIDA ANDRADE
JULIANA RODRIGUES MATOS
KEVEN DOS SANTOS COSTA
LUCAS DE OLIVEIRA MAFRA
NAYANE RODRIGUES MATOS SANTOS OI

PROJETO INTEGRADOR
ESCOPO DO PROJETO

TEMA: CRM

São Paulo
Dezembro de 2025

GRUPO 07

INTEGRANTES

BRUNA LUIZA DA SILVA GONÇALVES
DANIEL ALMEIDA ANDRADE
JULIANA RODRIGUES MATOS
KEVEN DOS SANTOS COSTA
LUCAS DE OLIVEIRA MAFRA
NAYANE RODRIGUES MATOS SANTOS OI

PROJETO INTEGRADOR
ESCOPO DO PROJETO

TEMA: CRM

Relatório solicitado pela Generation Brasil para compor o projeto final. O relatório refere-se ao escopo do projeto integrador.

São Paulo
Dezembro de 2025

1 Título do Projeto e Modelo de Negócio escolhido

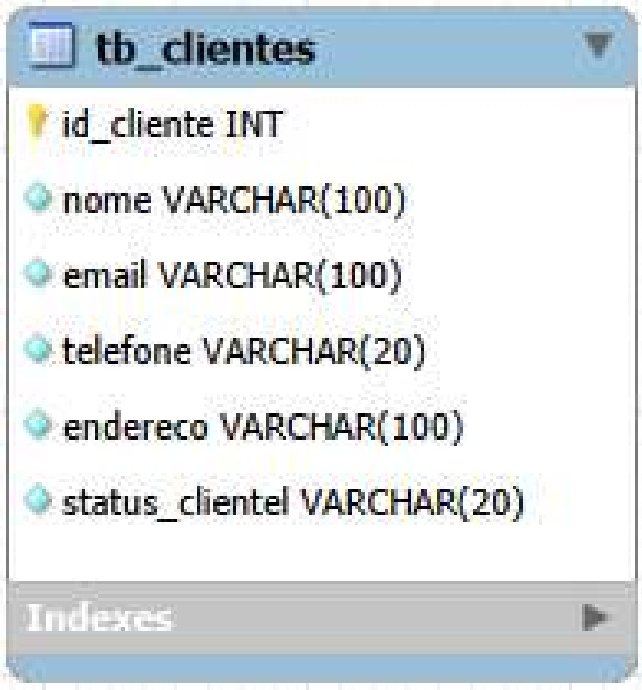
CRM Grupo 7 – Gestão de Clientes e Vendas de pequenas e médias empresas.

2 Descrição Geral do Projeto

O projeto tem como objetivo desenvolver um sistema CRM, focado em pequenas e médias empresas para gerenciamento de informações de clientes.

A aplicação será construída utilizando tecnologias modernas no backend, e incluirá operações CRUD completas (Criar, Ler, Atualizar e Excluir). O projeto contara com uma equipe de 6 pessoas, sendo um PO (Product Owner), um Scrum Master, três Desenvolvedores e um Tester.

3 Descrição da Entidade/Model foi criada e seus atributos



Este é um modelo inicial do projeto, no qual o modulo *Cliente.module* vai gerar uma tabela

Classe Cliente.entity (Model no modelo MVC), é a classe responsável pela abstração dos objeto, servirá como base para a criação da tabela tb_clientes no banco de dados db_CRM. A estrutura e os campos (atributos) dessa tabela seguirão o que foi definido no Diagrama de Classes.

4 Funcionalidades Principais (CRUD) implementadas

O ClienteService é responsável por toda a regra de negócio e as operações CRUD da entidade Cliente. Ele se comunica diretamente com o banco de dados por meio do Repository. Este service organiza tudo o que pode ser feito com o cliente no CRUD: buscar, cadastrar, atualizar e deletar.

O service utiliza o decorator @Injectable(), que permite que ele seja injetado em outras partes da aplicação. Dentro do construtor, é usado o @InjectRepository(Cliente), que injeta automaticamente o repositório do TypeORM correspondente à tabela tb_clientes.

Abaixo detalhamento das funcionalidades:

1. Listar todos os clientes – findAll()

O método findAll() retorna todos os registros da tabela de clientes. Ele utiliza this.clienteRepository.find() para fazer uma consulta completa no banco de dados.

2. Buscar cliente pelo ID – findById()

O método findById(id_cliente) busca um cliente específico usando seu identificador. Ele utiliza findOne() com uma condição where para localizar o registro. Caso o cliente não seja encontrado, o serviço lança uma exceção HttpException com status 404 – NOT FOUND, garantindo que o sistema trate corretamente erros de busca e evitando operações inválidas.

3. Buscar clientes pelo nome – findByName()

O método findByName(nome) permite pesquisar clientes utilizando um nome ou parte dele. Ele utiliza o operador ILike, que faz uma busca textual ignorando letras maiúsculas e minúsculas.

4. Criar um novo cliente – create()

O método create(cliente) recebe um objeto Cliente e o salva no banco de dados usando repository.save(). Esse método insere um novo registro na tabela e retorna os dados cadastrados, incluindo o ID gerado automaticamente.

5. Atualizar dados de um cliente – update()

O método update(cliente) atualiza as informações de um cliente já existente. Primeiro, ele verifica se o cliente existe chamando findById(). Caso o cliente não seja

encontrado, é lançada uma exceção de erro 404. Se o cliente existir, o método salva as novas informações usando novamente repository.save().

6. Excluir cliente – delete()

O método delete(id_cliente) remove um cliente da tabela. Antes de excluir, ele verifica se o cliente existe utilizando findById(). Se existir, a exclusão é feita com repository.delete(id_cliente), que retorna um objeto DeleteResult.

5 Tecnologias Utilizadas (banco de dados e backend)

NestJS – Framework para Node JS, foi utilizando as dependências e tratamento padronizado das requisições HTTP, para organização do projeto em módulos, controllers, services e entities.

TypeScript – Linguagem de programação.

TypeORM - ORM utilizado para mapear classes do TypeScript em tabelas do banco de dados.

Node.js - Ambiente de execução TypeScript utilizado para rodar o servidor backend.

MySQL - Banco de dados relacional armazenou a tabela tb_clientes.

Insomnia - Ferramenta utilizada para testar.