Introdução e Desafios

Para o projeto, foi escolhido utilizar o framework NestJS, que já havia sido apresentado em aula. Ele possui uma clara divisão e organização de módulos e classes que já ajudam a manter um código organizado e limpo.

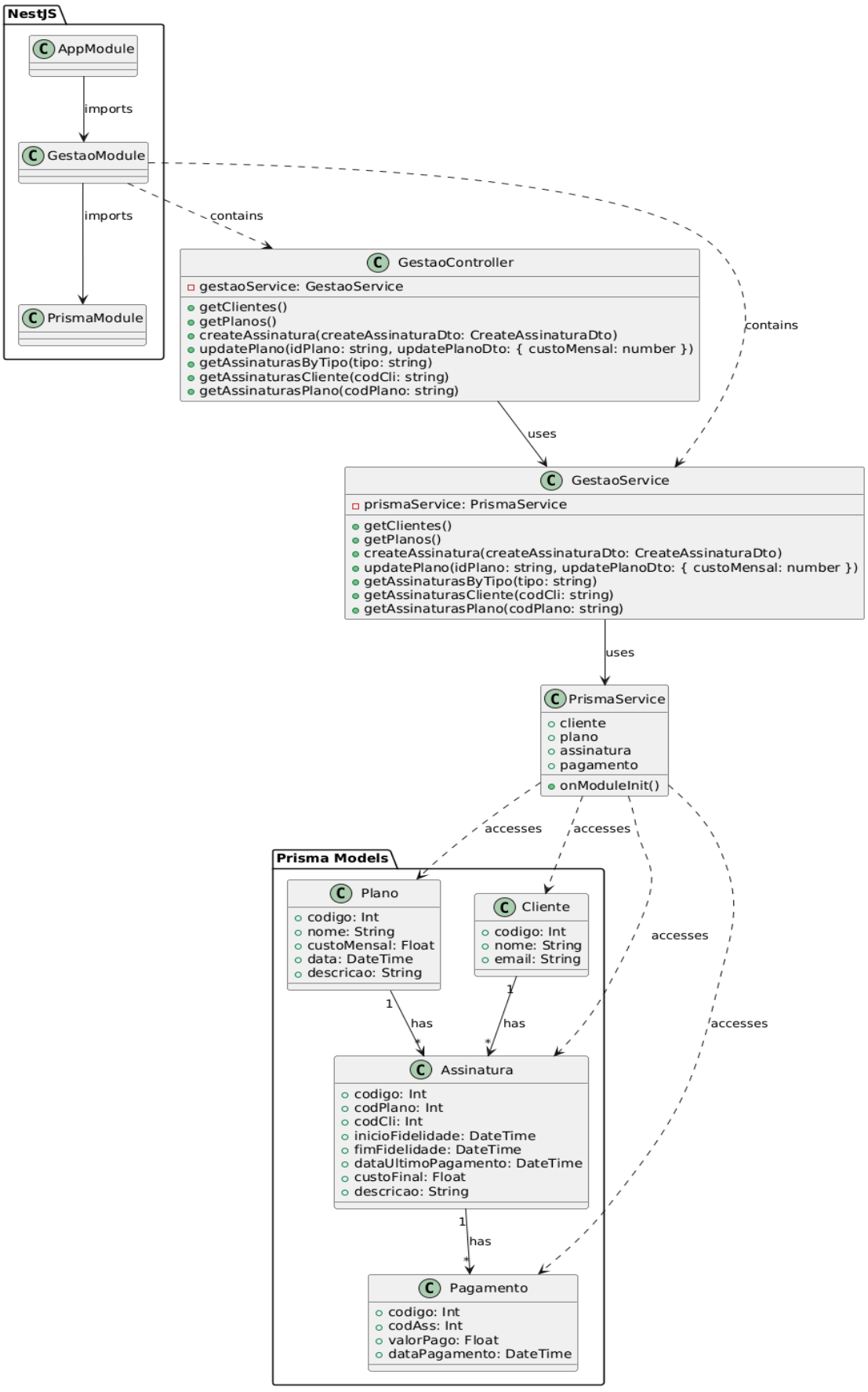
Para ferramentas relacionadas à persistência de dados, pesquisei e encontrei o Prisma ORM, que é bastante utilizado em conjunto com o NestJS. Por questões de simplicidade, escolhi usar SQLite como banco de dados. A configuração inicial do Prisma com SQLite foi simples, embora tive que pesquisar em sua própria documentação. Entretanto tive alguns problemas ao executar as migrações e o script de seeding. Isso se resolveu ao modificar o modo como as configurações do Prisma estavam definidos (aparentemente algo nas últimas atualizações não está funcionando muito bem, e, como há muito mais material na internet com as configurações de versões anteriores, resolvi adotar o que achei com mais conteúdo na internet).

O projeto NestJS contém 3 módulos, o módulo app, o módulo gestao e o módulo prisma.

O módulo app está servindo apenas para importar a definição de todo o resto do projeto. O módulo gestao contém as definições de controller e service de gestão. Dentro do controller de gestao está os endpoints definidos para a etapa 1, e dentro do service estão definidas as implementações desses endpoints, assim como as integrações com o Prisma. O módulo prisma apenas possui um service para que o prisma esteja disponível para o resto do projeto.

Links de referência:
https://www.prisma.io/docs/getting-started/quickstart-sqlite
https://www.prisma.io/docs/orm/prisma-migrate/workflows/seeding
https://docs.nestjs.com/

Diagrama UML das Classes do Projeto Gestao



Arquitetura e Princípios SOLID no Projeto

1. Single Responsibility Principle (SRP) - Responsabilidade Única

Cada classe possui uma única responsabilidade bem definida:

- GestaoController: Responsável apenas por receber requisições HTTP, validar parâmetros e delegar a lógica de negócio ao service. Não contém regras de negócio.

- GestaoService: Contém toda a lógica de negócio relacionada à gestão de assinaturas, clientes e planos. Não conhece detalhes de HTTP ou acesso direto ao banco.

- PrismaService: Responsável exclusivamente pelo acesso ao banco de dados através do Prisma ORM. Encapsula todas as operações de persistência.

2. Open/Closed Principle (OCP) - Aberto para Extensão, Fechado para Modificação

O projeto utiliza o sistema de módulos do NestJS, permitindo extensão sem modificar código existente:

- Mdulos independentes: GestaoModule e PrismaModule podem ser estendidos através de novos providers ou imports sem alterar suas implementações atuais.

- Injeção de dependências: Novos serviços podem ser adicionados e injetados sem modificar classes existentes.

3. Liskov Substitution Principle (LSP) - Substituição de Liskov

- PrismaService: Estende PrismaClient e implementa OnModuleInit, mantendo compatibilidade total com a interface base. Pode ser substituído por qualquer implementação que siga o mesmo contrato.

4. Interface Segregation Principle (ISP) - Segregação de Interface

Embora o projeto use classes concretas, a arquitetura permite que:

- Serviços especializados: Cada serviço expõe apenas os métodos necessários para seu domínio específico. PrismaService fornece acesso ao banco, enquanto GestaoService fornece operações de negócio.

5. Dependency Inversion Principle (DIP) - Inversão de Dependência

O projeto aplica inversão de dependência através da injeção de dependências do NestJS:

- GestaoService: Depende de PrismaService através de injeção no construtor, não de uma implementação concreta. Isso permite substituir facilmente o provedor de dados.
- GestaoController: Depende de GestaoService através de injeção, não de uma instância concreta.

Informações de como executar o projeto
(as mesmas também estão disponíveis no README.MD no projeto e no github https://github.com/giovannabmarinho/projeto-back-end):

```
1. Navegue até a raiz do projeto e depois entre na pasta do código
fonte do projeto:
cd servico-gestao

2. Instale as dependências:
npm install

3. Execute as migrações do Prisma:
npx prisma migrate deploy

4. (Opcional) Popule o banco com dados iniciais:
npx prisma db seed

5. Execute o projeto
npm run start:dev
O servidor estará disponível em `http://localhost:3000`
```