Objetivos:

I. ORM;
II. Migrações de Banco de Dados;
III. Prisma;

i. ORM

O ORM (Object-Relational Mapping), ou Mapeamento Objeto-Relacional, é uma técnica de programação que facilita a interação entre uma aplicação e um BD relacional. Ele cria uma camada de abstração que permite manipular dados no BD usando objetos, classes e métodos, em vez de escrever comandos SQL manualmente.

Em termos práticos, um ORM mapeia as tabelas do BD para classes, as colunas para propriedades, e os registros para instâncias dessas classes. Com isso, é possível realizar operações de CRUD (Create, Read, Update, Delete) diretamente em objetos da aplicação, sem a necessidade de construir queries SQL.

O principal objetivo de um ORM é reduzir a complexidade do desenvolvimento ao integrar aplicações orientadas a objetos (como as escritas em TS, JS, Java ou C#) com BD relacionais (como PostgreSQL, MySQL ou SQLite). Essa integração é feita de forma transparente, permitindo que o desenvolvedor foque mais na lógica de negócio e menos na manipulação do BD.

O uso de um ORM traz diversas vantagens para o desenvolvimento de aplicações:

- Abstração do BD: com um ORM, trabalhamos diretamente com objetos da nossa aplicação, abstraindo as tabelas e colunas do BD. Isso torna o código mais legível, intuitivo e alinhado ao paradigma orientado a objetos;
- Portabilidade de BD: muitos ORMs suportam diversos SGBDs como MySQL, PostgreSQL, Oracle e SQLite. Isso facilita a migração da aplicação entre diferentes bancos, bastando ajustar a configuração do ORM;
- Manutenção simplificada: o código se torna mais limpo e fácil de manter, pois as consultas SQL complexas ficam encapsuladas nas classes do ORM. Isso também reduz a duplicação de código e facilita refatorações futuras;
- Segurança: ORMs geralmente incluem mecanismos internos para prevenir vulnerabilidades, como injeção de SQL, protegendo a aplicação de ataques comuns;
- Desenvolvimento mais rápido: o ORM automatiza boa parte do trabalho relacionado a interações com o BD, como criação de queries e mapeamento de resultados. Isso acelera o desenvolvimento e reduz a probabilidade de erros humanos.

Exemplos populares de ORMs:

- Java: Hibernate;
- C#/.NET: Entity Framework;
- Python: Django ORM;
- JavaScript/Node.js: Sequelize, Prisma;

- TypeScript: TypeORM, Prisma.

Apesar das vantagens, o uso de um ORM nem sempre é ideal. Ele pode não ser a melhor escolha em casos onde:

- A aplicação exige consultas SQL extremamente complexas e otimizadas, que não são facilmente geradas pelo ORM;
- Há restrições severas de desempenho que exijam controle total sobre as queries;
- O BD é não relacional (NoSQL), como MongoDB ou Redis, onde o conceito de ORM não se aplica.

ii. Migrações de Banco de Dados

No desenvolvimento de aplicações, o BD está em constante evolução para atender às novas necessidades do sistema, como a criação de novas tabelas, a adição de colunas ou alterações em índices. As migrações de BD são uma maneira estruturada de gerenciar essas mudanças de forma rastreável e automatizada.

Uma migração pode ser vista como um "script de mudança" que descreve como modificar o esquema do BD de um estado para outro. Isso garante que as alterações sejam consistentes e possam ser aplicadas em diferentes ambientes, como desenvolvimento, teste e produção, sem necessidade de intervenção manual.

Motivos para usar migrações:

- Rastreamento de alterações: cada migração registra uma alteração no BD, permitindo rastrear o histórico completo das mudanças realizadas ao longo do tempo;
- Consistência entre ambientes: as migrações garantem que todos os desenvolvedores e servidores do projeto utilizem a mesma versão do esquema do BD, evitando problemas causados por inconsistências;
- Automatização e controle: em vez de criar scripts SQL manualmente, as ferramentas de migração geram e aplicam essas mudanças de forma automática e segura;
- Facilidade de reversão: muitas ferramentas de migração permitem desfazer mudanças aplicadas, tornando mais seguro experimentar alterações no BD.

Funcionamento das migrações:

O processo de migração geralmente segue os passos a seguir:

1. Definir a alteração: o desenvolvedor especifica o que deve ser alterado, como criar uma nova tabela, adicionar colunas ou modificar um índice;
2. Gerar o arquivo de migração: a ferramenta de migração cria um arquivo que descreve as mudanças em um formato legível e rastreável;
3. Aplicar a migração: a ferramenta executa o arquivo de migração no BD, aplicando as mudanças de forma controlada;
4. Versionamento e sincronização: cada migração possui um identificador único que permite saber quais mudanças já foram aplicadas, mantendo o BD sincronizado com o código.

iii. Prisma

O Prisma (https://www.prisma.io) é um ORM moderno para Node.js e TS que simplifica o desenvolvimento de aplicações que interagem com BD relacionais. Ele é projetado para oferecer uma experiência de desenvolvimento intuitiva, produtiva e alinhada com as práticas modernas de programação. O Prisma não é apenas um ORM tradicional; ele também inclui ferramentas adicionais que ampliam sua funcionalidade e flexibilidade, tornando-o uma escolha popular para projetos baseados em Node.js.

O Prisma é composto por três partes principais, que trabalham em conjunto para facilitar o desenvolvimento e o gerenciamento do BD:

1. Prisma Client

   É uma biblioteca gerada automaticamente com base no esquema do BD. Ele permite realizar operações CRUD usando métodos tipados e otimizados, o que reduz erros e melhora a produtividade.

2. Prisma Migrate

   Uma ferramenta para gerenciar migrações de esquema de BD. Ela permite criar, modificar e sincronizar a estrutura do BD de forma segura e eficiente.

3. Prisma Studio

   Uma interface gráfica para visualizar e gerenciar os dados do banco diretamente no navegador, sem necessidade de acessar o banco via SQL.

Vantagens do Prisma:

- Integração com TS: o Prisma fornece suporte completo ao TS, com tipagem automática gerada a partir do esquema do BD. Isso ajuda a evitar erros e aumenta a segurança do código;
- Gerenciamento simples de Migrações: com o Prisma Migrate, é possível aplicar mudanças no BD de maneira controlada e rastreável, garantindo a consistência entre o esquema e o código;
- Queries tipadas e intuitivas: o Prisma Client gera métodos que permitem criar queries de forma simples, com autocomplete e validação direta no editor de código;
- Suporte a múltiplos BD: ele suporta vários SGBDs, como PostgreSQL, MySQL, SQLite, SQL Server e MongoDB. Isso oferece flexibilidade para escolher a solução mais adequada para cada projeto;
- Documentação e comunidade: o Prisma conta com uma excelente documentação e uma comunidade ativa, o que facilita a aprendizagem e a resolução de problemas.