Arquitetura do Software GetDogs

A arquitetura escolhida para o projeto de software GetDogs é inspirada na arquitetura Clean DDD, que é uma junção da arquitetura Clean com princípios de Domain-Driven Design.

A arquitetura limpa ou Clean Architecture é um padrão criado por Robert Martin para proporcionar uma maior reusabilidade de código, coesão, independência de tecnologias de terceiros e testabilidade. Nesta arquitetura, há quatro camadas principais: entidades, casos de uso, adaptadores e frameworks externos.

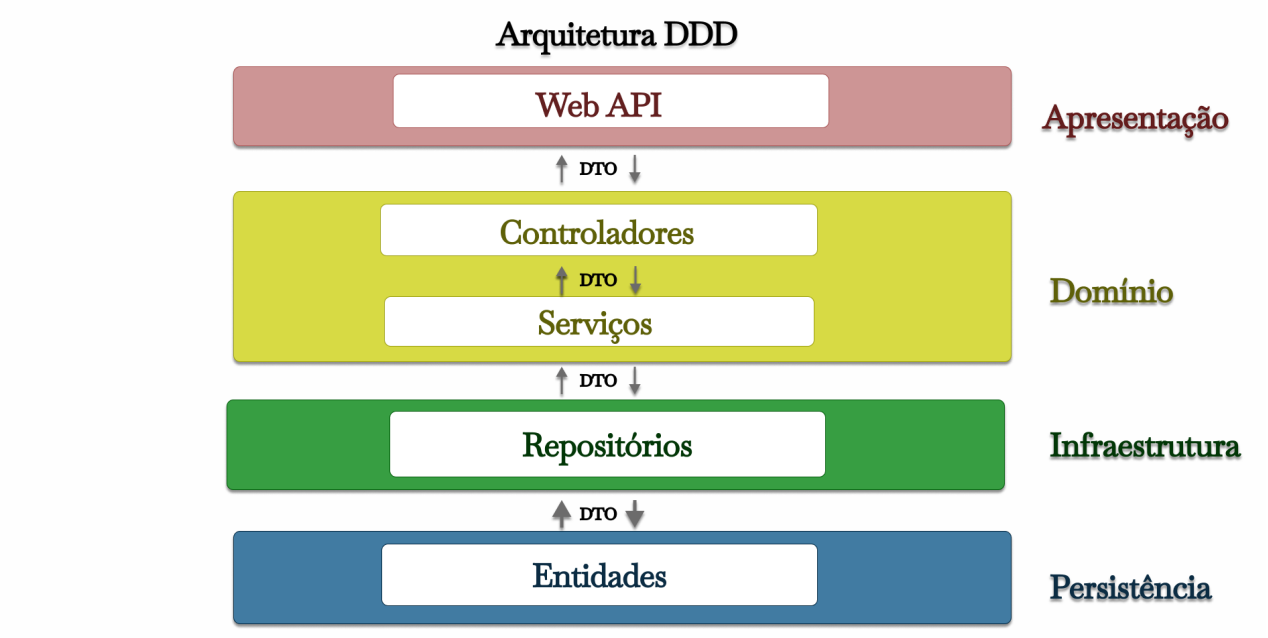
A camada de entidade representa todas as classes que deverão ter algum tipo de persistência em banco de dados. Já os casos de uso implementam as regras de negócio específicas do sistema. Logo em seguida aos casos de uso, tem-se os adaptadores que normalmente são classes e interfaces que fazem a conexão com a camada de frameworks externos e as camadas internas e próprias do sistema. Ou seja, elas se comportam como tradutores de mão dupla, adaptando os dados para a forma em que cada camada pode trabalhar. Por último, há os frameworks externos, estes são bibliotecas, frameworks ou quaisquer sistemas externos que têm alguma participação no software que se pretende criar. Por exemplo, sistemas responsáveis pela persistência dos dados, envio de e-mails etc.

Quanto ao Domain-Driven Design ou DDD, é uma abordagem de desenvolvimento de software feita por Eric Evans. Esta abordagem é focada em um melhor entendimento do domínio daquilo que se deseja construir. Para isso, vale-se de vários conceitos para alcançar esse objetivo como entidades, agregados, domínios, objetos de valor etc.

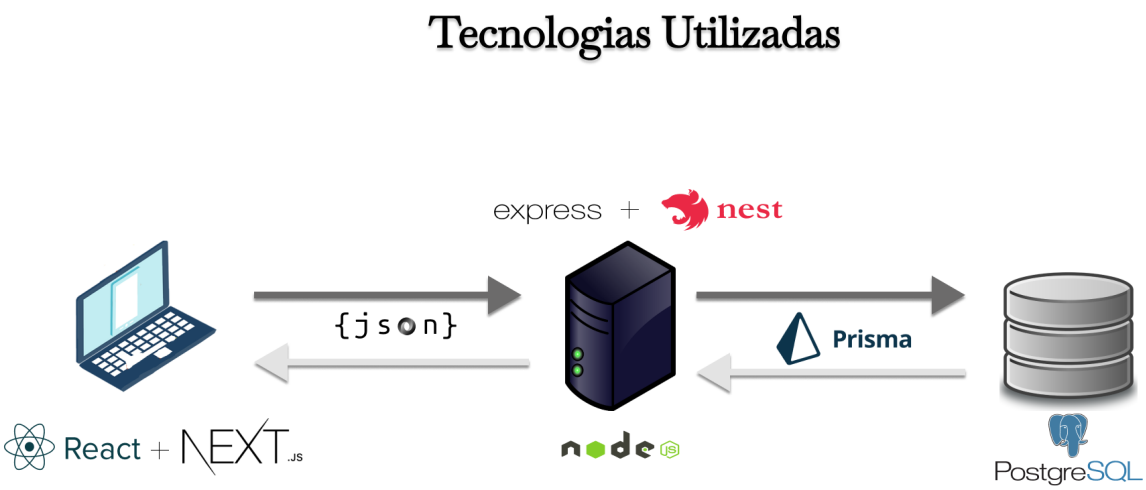
Em Domain-Driven Design, domínio é definido como a lógica do negócio que definirá como a solução será construída. Isso significa que ele quem delimita o que se pode ou não fazer. Já as entidades são objetos com identificadores exclusivos que as definem. Outro conceito importante são os objetos de valor que são objetos imutáveis que possuem atributos, mas não possuem uma identidade própria. Agregados são um agrupamento de entidades e objetos de valor com uma delimitação em si. Serviços são operações com o modelo e uma interface autônoma para satisfazer alguma funcionalidade. Por último, os repositórios são serviços com interfaces globais que garantem acesso a entidades e objetos de valor que são englobados por um agregado.

A arquitetura Clean DDD mescla essas duas abordagens para a criação de uma solução mais elegante focada no domínio da aplicação que determina como funcionará, além de permitir uma fácil manutenibilidade e aplicação dos princípios SOLID

No projeto de software GetDogs, a arquitetura é baseada nos princípios apresentados. Como é mostrado no diagrama abaixo:



Em conjunto com a arquitetura escolhida, é relevante mencionar as tecnologias que serão utilizadas para o desenvolvimento de cada parte do projeto:



Frontend:

- **React:** "React é uma biblioteca JavaScript que permite a criação de interfaces de usuário (UI) por meio de componentes reutilizáveis. Com React, é possível desenvolver interfaces com layouts complexos e comportamentos personalizados de forma eficiente. Ele é especialmente conhecido por sua abordagem de atualização eficiente do DOM, que melhora o desempenho das aplicações."
- **Next.js:** "Next.js é um framework para desenvolvimento web que se integra bem com o React. Ele oferece ferramentas poderosas para simplificar o desenvolvimento de aplicações web, incluindo renderização do lado do servidor

(SSR) e geração de páginas estáticas (SSG). Next.js permite criar aplicações web mais rápidas e eficientes, oferecendo uma experiência de desenvolvimento ágil."

Backend:

- **Nodejs**: Nodejs é um ambiente de desenvolvimento em tempo de execução de JavaScript assíncrono orientado a eventos. Com seu mecanismo de loop de eventos, há uma construção de tempo de execução JavaScript de forma eficiente e não bloqueante, adequado para aplicações que exigem alta concorrência e manipulação eficiente de entrada/saída. O Node.js é amplamente utilizado para criação de aplicativos de servidor e outras aplicações que depende E/S assíncrona.
- **Express.js:** Express.js, comumente conhecido como Express, é um framework minimalista para o desenvolvimento de aplicativos web em Node.js. Ele simplifica a criação de APIs e aplicações web ao fornecer uma série de utilitários e recursos que facilitam o tratamento de rotas, solicitações e respostas HTTP. O Express é amplamente utilizado pela comunidade Node.js devido à sua simplicidade e flexibilidade.
- **Nest.js:** Nest.js é um framework para desenvolvimento de aplicativos em Node.js que se concentra na construção de aplicações escaláveis e facilmente testáveis. Ele segue a arquitetura de módulos, fornecendo uma estrutura organizada para desenvolver aplicações em Node.js com TypeScript. O Nest.js é especialmente adequado para a criação de APIs RESTful e GraphQL e oferece um conjunto de ferramentas para a construção de aplicativos robustos e de fácil manutenção.

Banco de Dados:

- **Prisma:** "Prisma é uma ferramenta de mapeamento objeto-relacional (ORM) e camada de acesso a banco de dados para aplicações em JavaScript e TypeScript. Ele simplifica a interação com bancos de dados SQL, permitindo que os desenvolvedores escrevam consultas e manipulem dados usando código TypeScript ou JavaScript, em vez de SQL puro. Prisma oferece um conjunto de recursos que facilitam o desenvolvimento de aplicativos com bancos de dados, como geração de esquema automática, migrações e consultas seguras."
- **PostgreSQL:** "PostgreSQL, frequentemente referido como Postgres, é um sistema de gerenciamento de banco de dados relacional de código aberto. Ele é conhecido por sua robustez, recursos avançados e conformidade com padrões. O PostgreSQL permite armazenar, organizar e recuperar dados em formato tabular usando a linguagem SQL. Ele é amplamente utilizado em aplicações de alto desempenho e missão crítica devido à sua confiabilidade e suporte a recursos avançados, como indexação, transações ACID, funções armazenadas e suporte a tipos de dados personalizados."