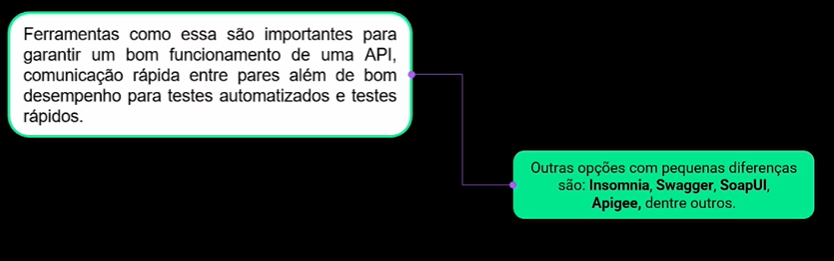
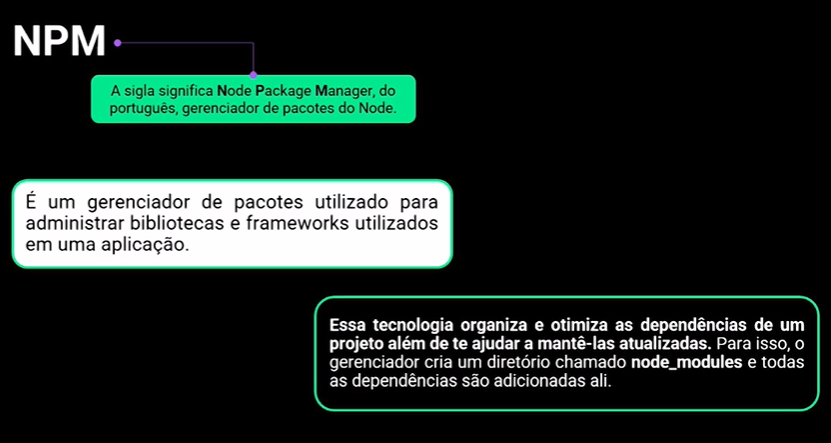
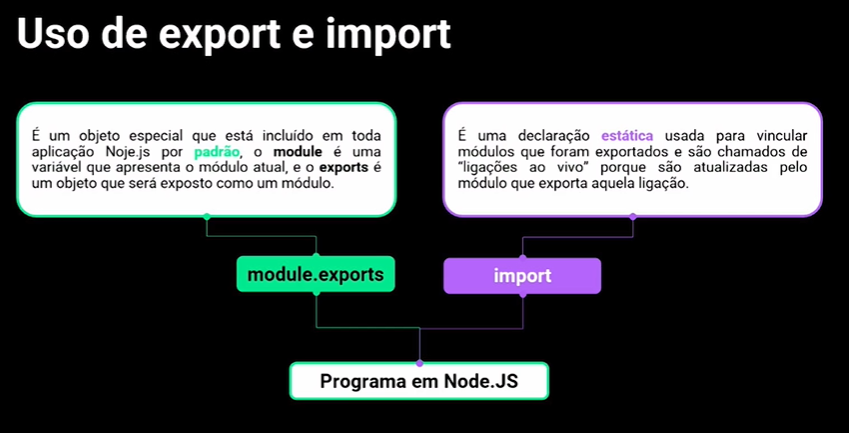


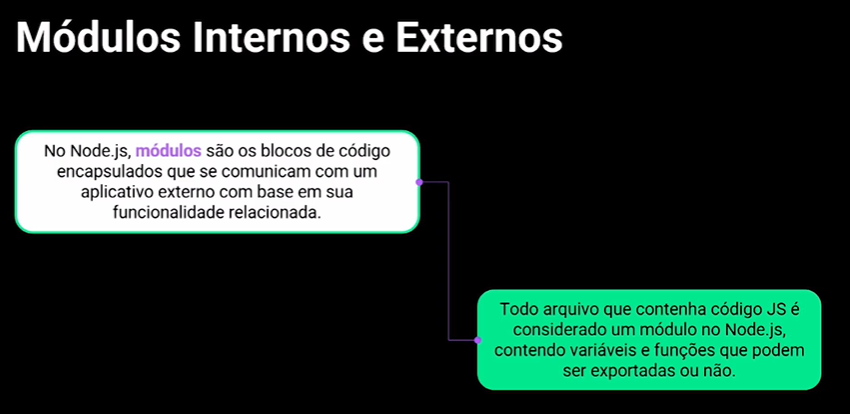
Garantir se a API está de acordo!



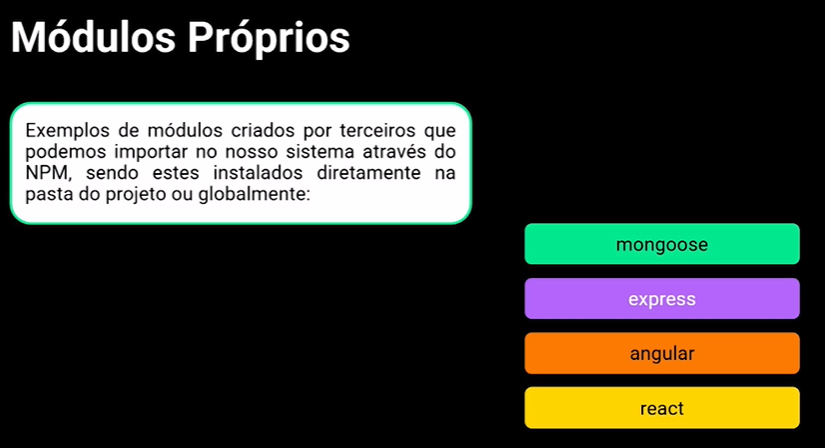
NPM:



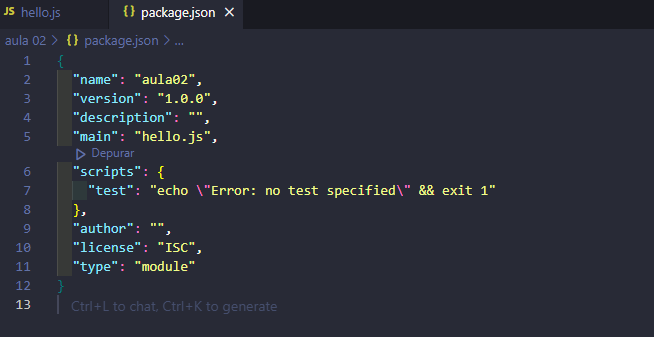








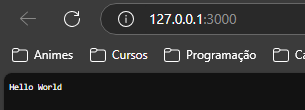
Dei um NPM init no terminal e criou:



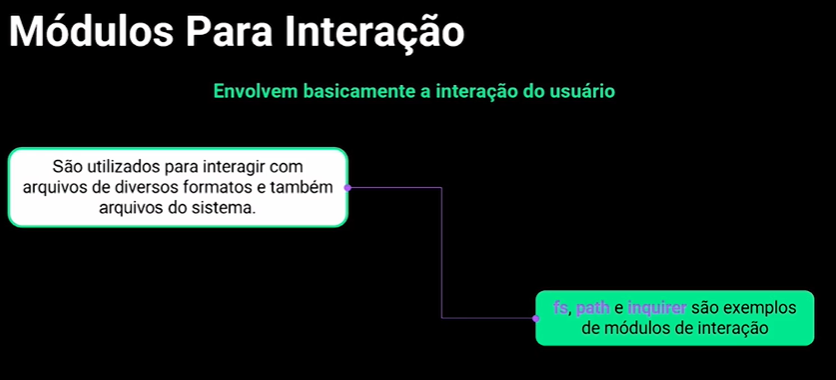
Adicionei apenas o type module para rodar.

Criei o hello.js assim:





Modulo para interação:

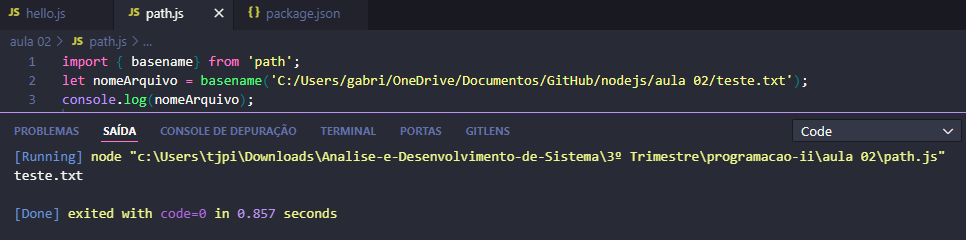


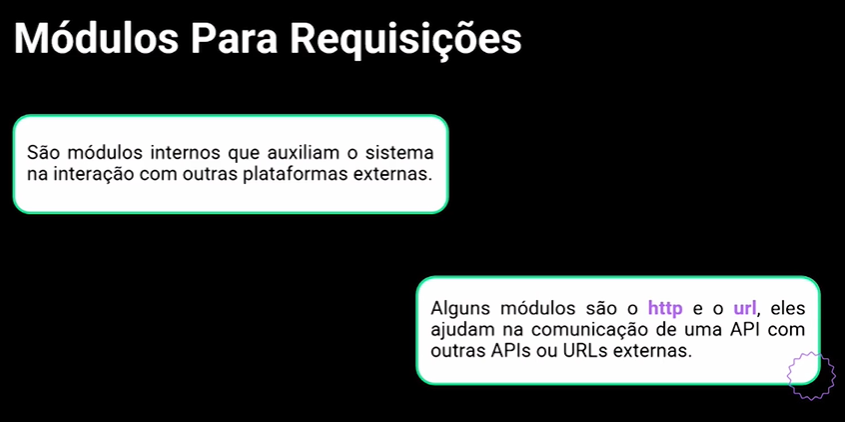
**1. Módulo fs (File System):** é o módulo nativo do Node.js usado para trabalhar com arquivos no sistema. Ele oferece uma ampla gama de funcionalidades, como leitura, escrita, modificação e exclusão de arquivos. Este módulo é fundamental para qualquer operação que envolva manipulação de arquivos, sendo uma das ferramentas mais utilizadas para tal fim.

**2. Módulo path:** facilita a manipulação de caminhos de arquivos e diretórios, ajudando a construir caminhos que sejam compatíveis com diferentes sistemas operacionais. Ele fornece métodos para extrair informações de caminhos de arquivo, como o diretório base, extensão do arquivo, e o nome do arquivo.

**3. Módulo inquirer:** utilizado para criar interfaces de linha de comando interativas. Ele permite que os desenvolvedores façam perguntas ao usuário, coletando inputs que podem ser usados para guiar a lógica de execução da aplicação.

Modulo path para saber o nome do arquivo que esta utilizando:

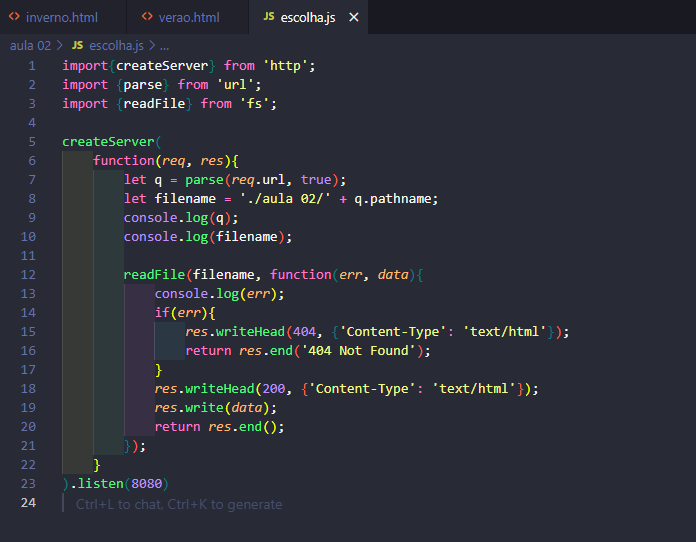


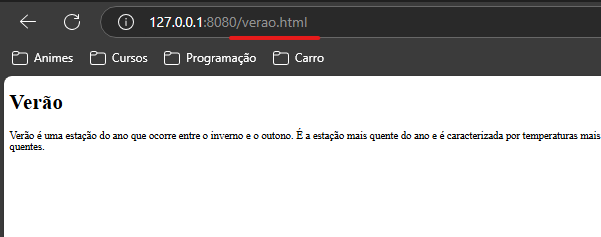


**1. Módulo http:** já introduzido em práticas anteriores, este módulo é essencial para criar servidores HTTP locais. Ele nos permite escutar requisições em portas específicas e responder a elas, essencial para qualquer aplicação web desenvolvida com Node.js.

**2. Módulo url:** este módulo fornece utilitários para a análise e manipulação de URLs. É particularmente útil para extrair partes específicas de uma URL ou construir URLs dinamicamente, baseadas em diferentes condições ou entradas.

Nesse caso fizemos um serve de escolha, no qual a pessoa pode escolher o inverno.html ou o verão html. Basta apenas mudar a URL





Projeto da disciplina:

O projeto proposto é um sistema de **gerenciamento para consultórios médicos**, uma aplicação web que permite administrar informações relacionadas a médicos, pacientes, consultas e prescrições médicas. Este sistema será desenvolvido seguindo boas práticas de mercado, com código em inglês, permitindo que você se familiarize com o ambiente de desenvolvimento profissional.

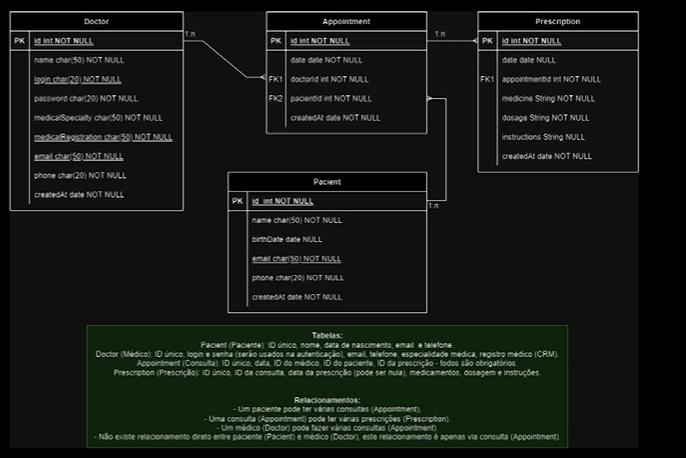
O sistema contará com quatro entidades principais:

* Doutores (Doctors): profissionais de saúde que poderão acessar o sistema para gerenciar suas consultas e prescrições;
* Pacientes (Patients): indivíduos que recebem atendimento médico e têm suas informações registradas no sistema;
* Consultas (Appointments): registros das visitas dos pacientes aos médicos, incluindo data, horário e detalhes da consulta;
* Prescrições (Prescriptions): detalhes das medicações e instruções prescritas pelos médicos aos pacientes.

Estas entidades estarão inter-relacionadas de maneira que um doutor possa ter múltiplas consultas agendadas, cada consulta pode gerar várias prescrições, e um paciente pode ter um histórico de múltiplas consultas.

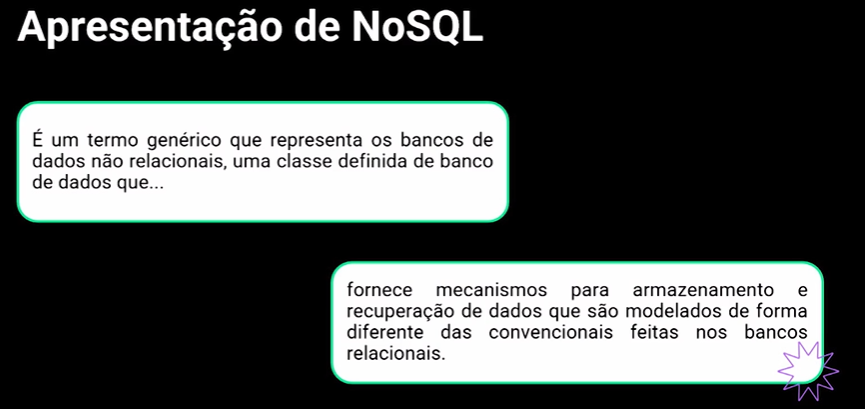
O design e a lógica do sistema deverão refletir e facilitar o gerenciamento eficiente dessas relações.



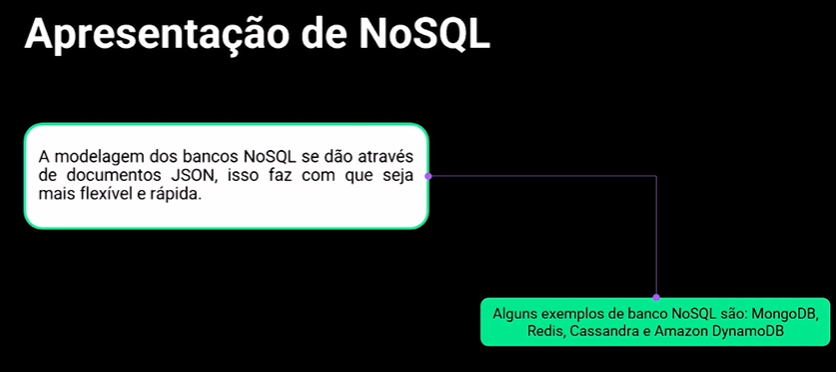


Banco de Dados:

NoSQL:

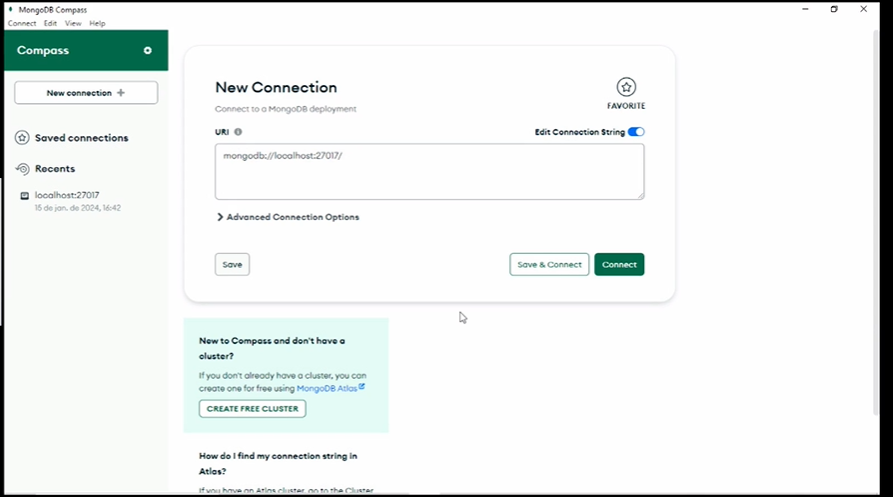


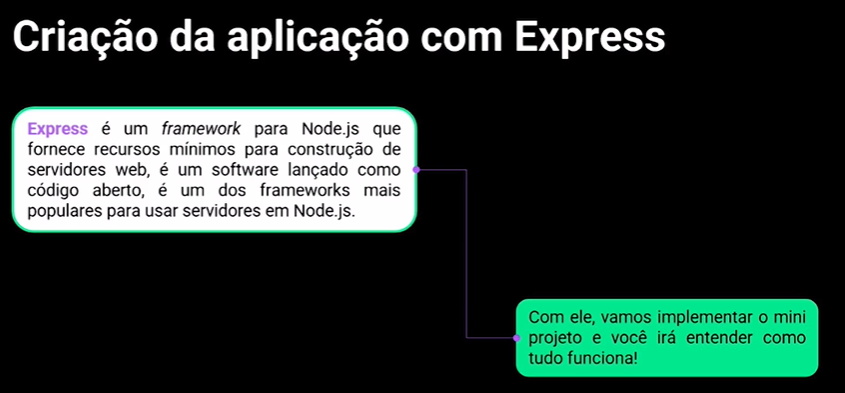
NoSQL representa os bancos de dados não relacionais e oferece uma estrutura de armazenamento flexível, ideal para dados não estruturados ou semiestruturados. Diferentemente dos bancos relacionais, os dados em NoSQL são organizados em coleções, e não tabelas, consistindo em documentos e campos, que formam uma estrutura de documentos JSON em chave-valor. Esta abordagem permite um modelo de dados mais dinâmico e escalável, adequado para as demandas de aplicações modernas que requerem alta velocidade e volumes variáveis de dados.



**Iremos utilizar o MongoDB**

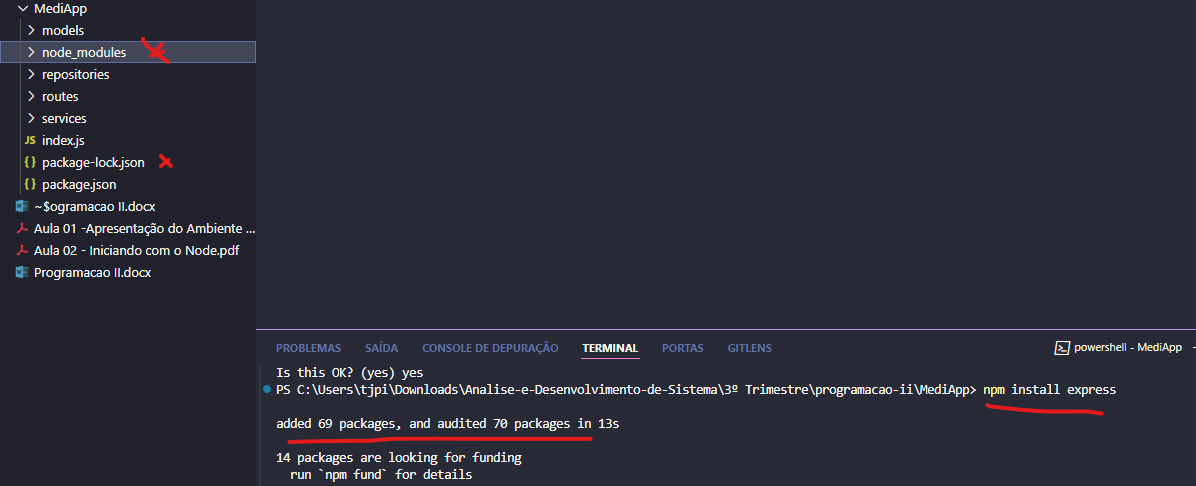


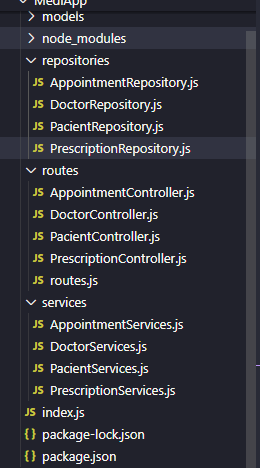




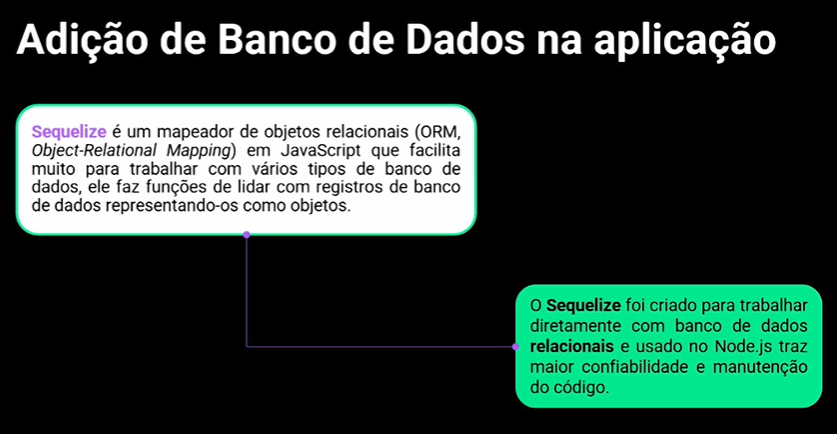
Express facilita a configuração e a gestão das rotas da aplicação, oferecendo um conjunto de recursos que tornam o desenvolvimento mais ágil e eficiente. Similar ao Spring Boot do Java em sua proposta de simplificar a criação de aplicações, o Express se destaca no ecossistema Node por sua popularidade e eficácia.

Só colocar npm install express na pasta do seu projeto!

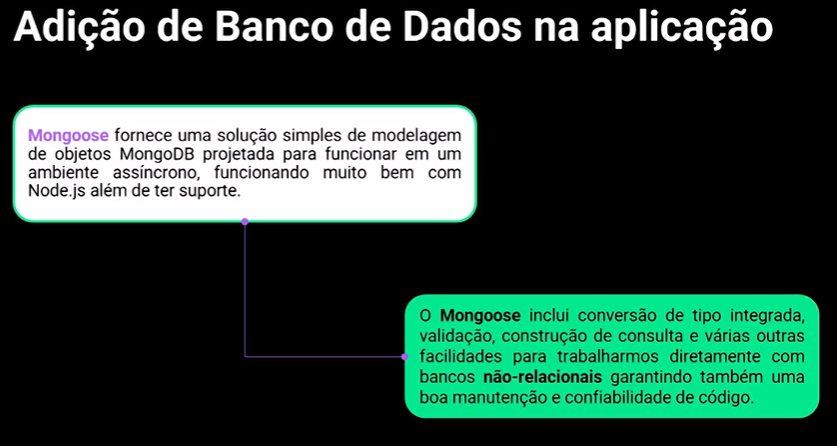




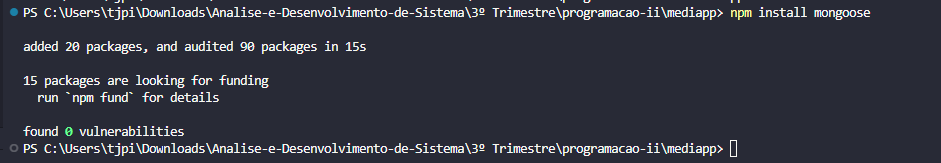
**Adição do banco de dados:**



**Não da para ser NÃO RELACIONAIS!**

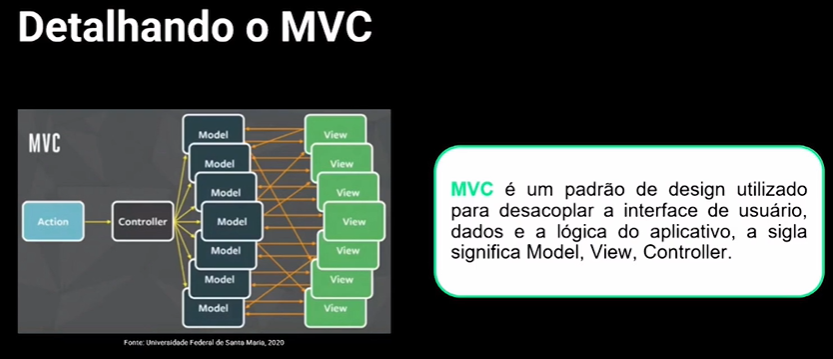


**Não da para ser RELACIONAIS!**



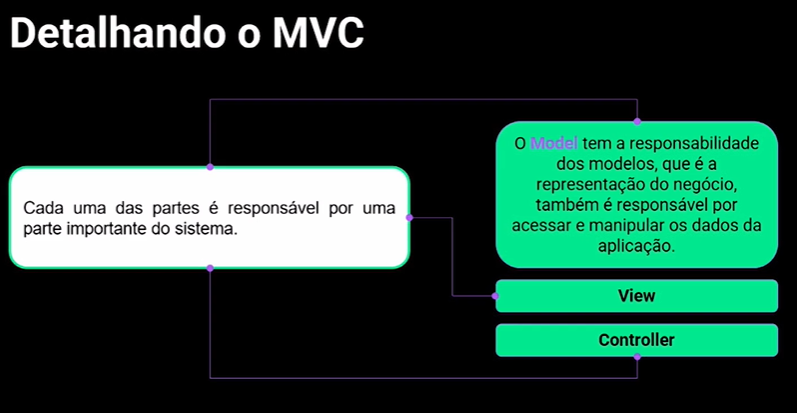
**Padrão MVC: (Model – View – Controller)**

Técnicas e padrões usados para projetar e desenvolver uma aplicação.



Separa a lógica de negócios, a interface do usuário e a interação do usuário em partes independentes. Essa separação ajuda a gerenciar a complexidade de aplicações, facilitando a manutenção, a escalabilidade e a testabilidade.

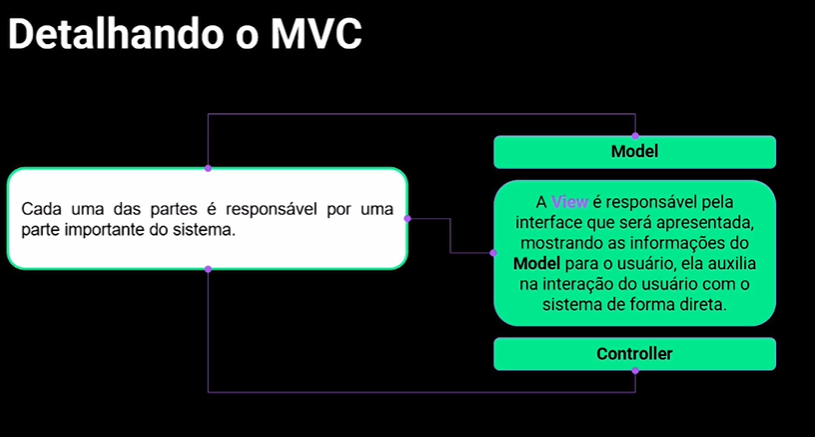




O componente **Model** representa a camada de dados e a lógica de negócios. Ele é responsável por acessar o banco de dados, executar consultas e retornar os dados necessários.

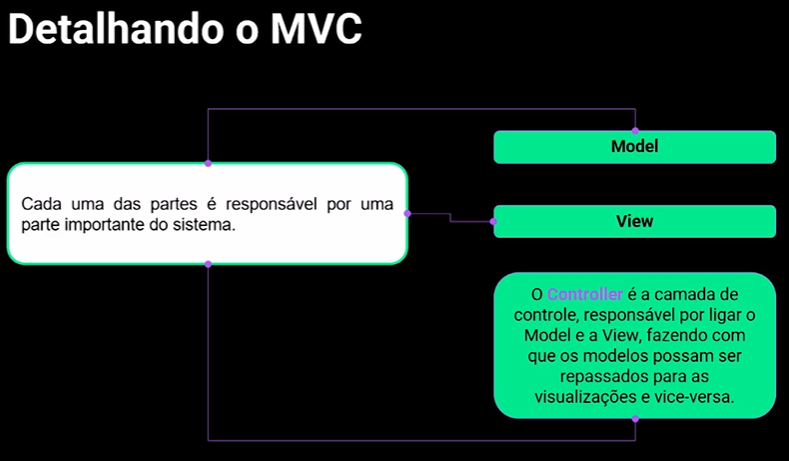
Em nosso projeto, por exemplo, ao definirmos entidades como consultas médicas, médicos e pacientes, estamos trabalhando na camada Model, que cuida de toda a gestão de informações sem se preocupar com a apresentação ou as ações do usuário.

Imagine uma funcionalidade em nosso projeto que gere relatórios médicos. O Model seria responsável por acessar o banco de dados e recuperar os dados dos pacientes, dos médicos e das consultas. Por exemplo, para gerar um relatório de consultas por médico, o Model consultaria o banco de dados para encontrar todas as consultas associadas a um determinado médico, tratando essas informações sem se preocupar com como serão apresentadas ou interagidas pelo usuário.



A camada **View** é o que o usuário vê: a interface gráfica. Ela exibe os dados recebidos do Model e envia as interações do usuário (como cliques e entradas de dados) ao Controller. Esta camada é crucial para a experiência do usuário por ser onde a interação acontece, mas ela não processa os dados recebidos, apenas os apresenta de forma legível e acessível.

Após o Model preparar os dados do relatório de consultas, a camada View se encarrega de apresentá-los. Isso pode ser feito por meio de uma tabela na interface do usuário que lista os detalhes das consultas, como data, horário, paciente e diagnóstico. A View é projetada para ser atraente e intuitiva, permitindo que o usuário visualize as informações de forma clara, mas sem executar lógicas de negócios ou acessar diretamente os dados.

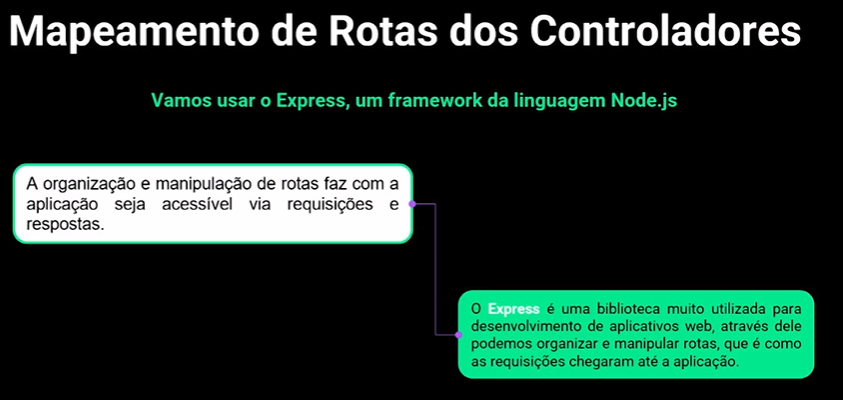
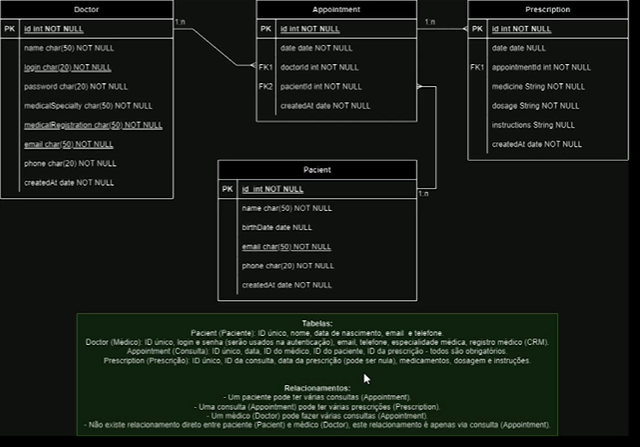


O **Controller** atua como intermediário entre a View e o Model. Ele recebe as ações do usuário transmitidas pela View, processa essas ações (podendo incluir validações) e solicita dados ao Model. Após receber os dados solicitados, o Controller os encaminha de volta à View para serem exibidos. Essa camada de controle é essencial para a orquestração do fluxo de informações e para a lógica de interação no sistema.

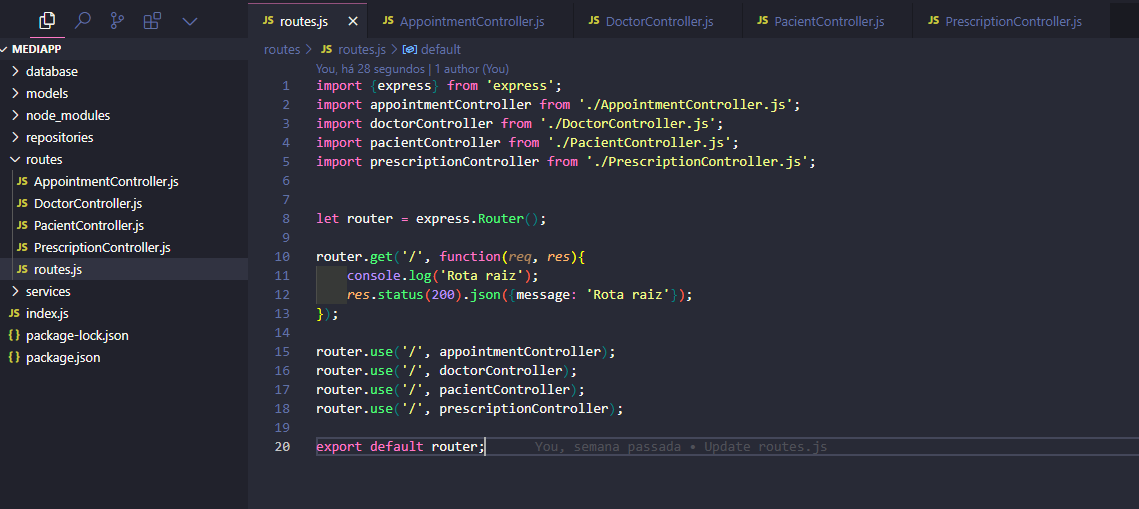
Funcionando como o maestro dessa orquestra, o Controller recebe a solicitação do usuário para gerar o relatório de consultas por médico. Ele então solicita ao Model os dados necessários e, uma vez recebidos, encaminha esses dados à View para apresentação. Se o usuário decidir filtrar as consultas por data, por exemplo, ele interage com a View, que passa essa solicitação ao Controller. O Controller, por sua vez, ajusta a solicitação de dados ao Model conforme necessário e atualiza a View com as novas informações.

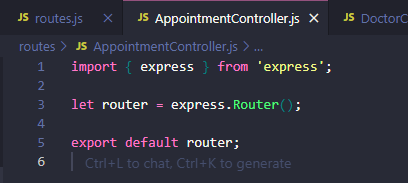
**Crud na aplicação:**



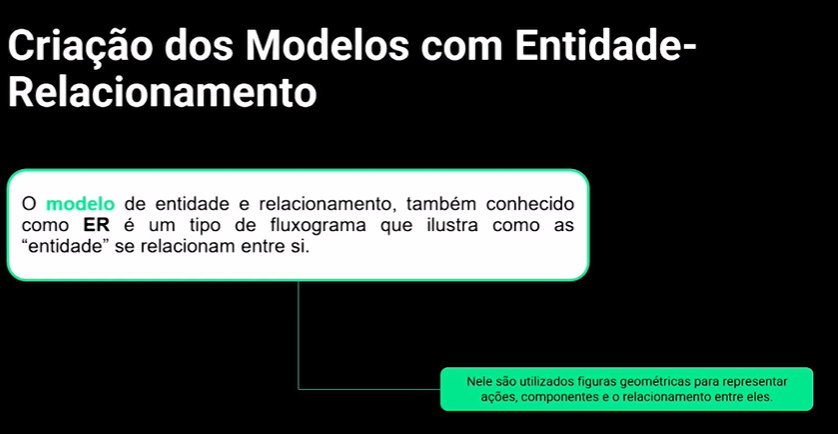


Em routes no nosso projeto fazemos:

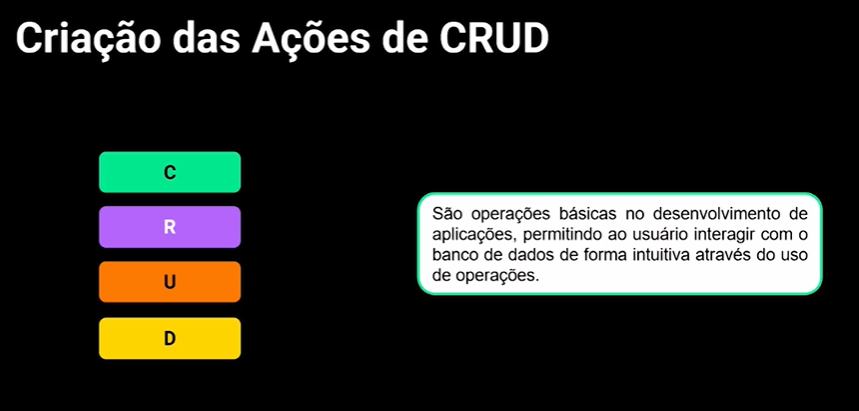


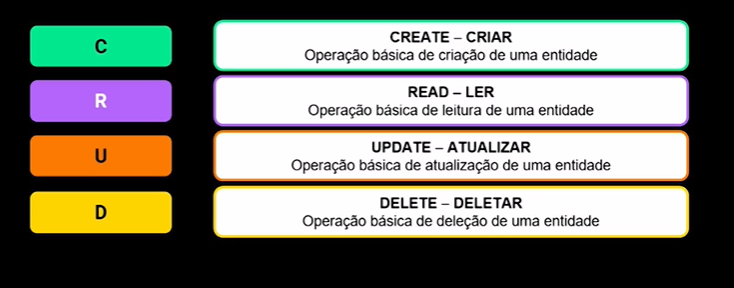


**Criando modelos com entidade-relacionamento:**

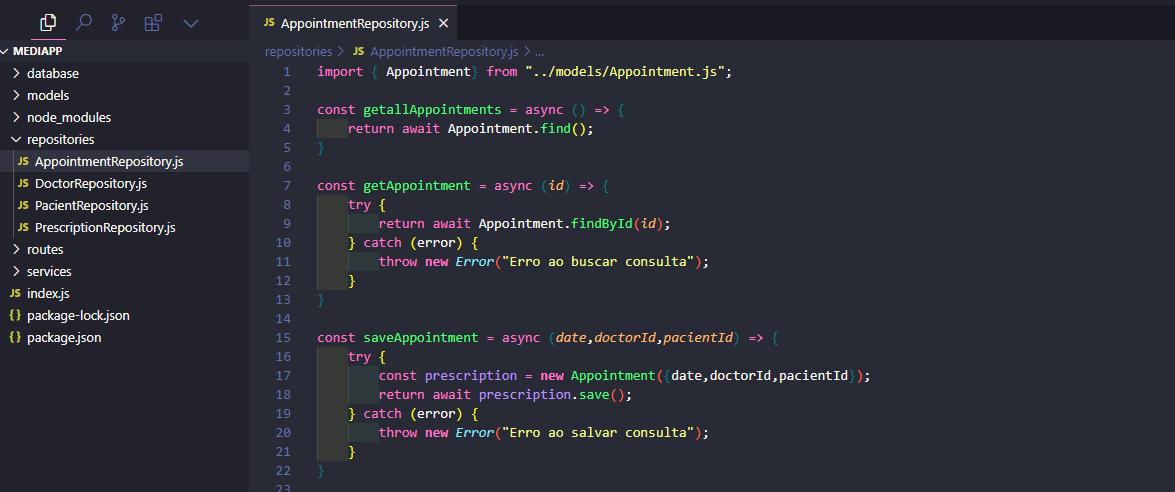


**Criando ações de CRUD:**



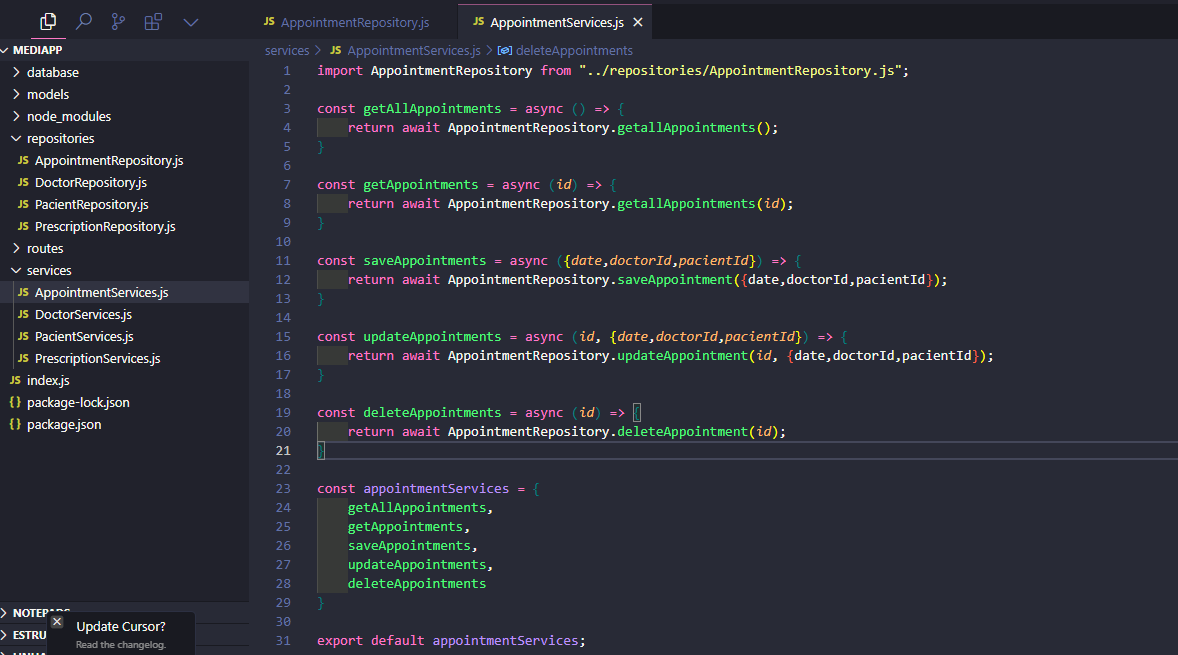


No repositorio vamos:





No service:



E para encerrar, encerramos no controller!



