

Parte 1: Questões Teóricas

1. Explique a diferença entre Eloquent ORM e Query Builder no Laravel. Quais são os prós e contras de cada abordagem?

Resposta: A diferença está no nível de abstração presente no Eloquent e no foco de uso.

Eloquent oferece uma abordagem orientada a objetos e abstrai muito do trabalho com o banco de dados, enquanto o **Query Builder** oferece mais controle e flexibilidade na criação de queries SQL, com melhor performance em operações mais diretas.

Eloquent ORM		Query Builder	
Prós	Contras	Prós	Contras
Orientado a objeto, pois permite trabalhar com modelos de dados.	Baixa performance em consultas complexas	Melhor performance em consultas simples e complexas	Não é orientado a objeto
Relacionamentos fáceis entre as tabelas e facilitam a navegação entre os modelos	Maior consumo de memória por converter cada linha da consulta em objeto	Menor consumo de memória por trabalhar com arrays associativos ao invés de objetos	Maior complexidade nos relacionamentos entre as tabelas, pois eles tem de ser definidos manualmente a cada consulta
Abstração alta na construção das queries e na construção dos CRUDs, pois os campos das tabelas geralmente estão definidos nas models	Mais complexo para uso em consultas personalizadas	Flexibilidade para consultas complexas	Menos concreto nas operações de CRUD, pois precisa especificar todos os detalhes da query

2. Como você garantiria a segurança de uma aplicação Laravel ao lidar com entradas de usuários e dados sensíveis? Liste pelo menos três práticas recomendadas e explique cada uma delas.

Resposta:

Primeiramente validar as requisições pois o Laravel já fornece algumas proteções na camada de Request, como a proteção à CSRF.

Utilizando a autenticação e liberando as consultas/inserções e updates somente quando validada a autenticação do usuário.

1. Autenticação e Autorização

- O uso de Middleware de autenticação para proteger as rotas sensíveis da aplicação, principalmente nas ações de CRUD.
- Utilizar Policies e Gates para controlar o acesso a recursos da aplicação.

2. Validação de dados

- Utilizando de ferramentas do próprio Laravel para validar os conteúdo enviado pelas requisições validando o tipo de dados passado e Sanitização de entrada de dados usando filtros para limpar os dados evitando assim injeção de scripts maliciosos

3. Configurações de segurança

- Manter variáveis sensíveis no .env e nunca no código e manter esta .env fora do repositório de versionamento da aplicação.
- Desativar a função de debug no ambiente de produção; APP_DEBUG=true; evitando assim a exibição de mensagens de erro detalhadas, que podem revelar informações sensíveis.
- Setar a variável APP_ENV como production, evitando que comandos de desenvolvedor sejam executados pelo artisan sem a confirmação do mesmo, como por exemplo o migrate

3. Qual é o papel dos Middlewares no Laravel e como eles se integram ao pipeline de requisição? Dê um exemplo prático de como você criaria e aplicaria um Middleware personalizado para verificar se o usuário está ativo antes de permitir o acesso a uma rota específica.

Resposta: Middlewares são uma camada intermediária que gerencia e inspeciona as requisições HTTP recebidas antes que elas alcancem o controle (Controller) da aplicação. Eles atuam como filtros que podem executar uma série de ações em uma requisição e, em seguida, passar essa requisição adiante ou bloqueá-la, dependendo da lógica implementada. Os middlewares permitem adicionar autenticação, autorização, logging, manipulação de dados e outras funcionalidades ao pipeline de requisição.

Exemplo:

- Usando o comando Artisan para gerar um novo middleware: `php artisan make:middleware CheckIfUserIsActive`
- No arquivo gerado implementaria a lógica para verificar se o usuário está autenticado utilizando a Facade Auth::check e Auth::user() → is_active
- Registrar essa Middleware no arquivo `app/Http/Kernel.php` na seção `routeMiddleware`

- Aplicar esta Middleware a rota
- Atualizar a Model/User e a tabela para que tenha o campo `is_active`

4. Descreva como o Laravel gerencia migrations e como isso é útil para o desenvolvimento de aplicações. Quais são as melhores práticas ao criar e aplicar migrations?

Resposta:

- As Migrations, são uma maneira controlada de gerenciar o esquema do banco de dados ao longo do tempo de vida da aplicação.
- Ela permite criar, modificar e remover tabelas e colunas e mantém-se como um registro destas operações.
- Através dela é possível rapidamente restaurar a estrutura do banco

Melhores praticas para se criar uma migration

- Definir nomes significativos, isso facilita a compreensão do que cada uma faz
- Manter migrations reversíveis através do método Down, isso garante que elas podem ser revertidas em caso de erro
- Nunca alterar as migrations após a execução em produção, em vez disso é melhor criar uma nova com a correção a ser feita
- Realizar um backup do banco de dados antes da execução, para ter uma salva guarda caso seja necessário um reversão

5. Qual é a diferença entre transações e savepoints no SQL Server? Como você usaria transações em um ambiente Laravel?

Resposta: Transações e Savepoints são conceitos relacionados ao controle de consistência e gerenciamento de alterações em um banco de dados.

- Uma transação no SQL Server é uma sequência de operações (inserção, atualização, exclusão, etc.) que são executadas como uma única unidade lógica.
- Um savepoint é um ponto intermediário dentro de uma transação que permite realizar rollbacks parciais. Ou seja, em vez de desfazer toda a transação, você pode reverter apenas parte das operações até o ponto definido pelo savepoint.

No Laravel implementaria uma transação com o método `DB::transaction()`, para executar uma transação através de um bloco de código que automaticamente faz um rollback em caso de falhas.

E como o Laravel não tem suporte nativo direto para Savepoints, eles podem ser gerenciados manualmente usando `DB::statement()` para comandos SQL e gerenciar os savepoints manualmente no banco.