# Laravel parte 2: Trabalhando com Autenticação, Relacionamentos e Testes

- Esse curso capacita a:
  - Proteger páginas e trabalhar com autenticação;
  - Aprender como lidar com requisições AJAX;
  - Criar relacionamento no modelo e atualizar o banco através de migrations;
  - Aplicar boas práticas e refatorar o código;
  - Testar as rotas e controllers com testes automatizados;

Aulas:

1. **Relacionamentos no Modelo:**
   - Gera um modelo e criar uma migration em conjunto:
     **php artisan make:model <nome> -m**
   - **php artisan migrate** [Executa as migrations novas];
   - Eloquent (ORM do Laravel) informa as relações entre os modelos através dos métodos: **belongsTo(), hasMany(), hasOne(), belongsToMany()**;

2. **Novo controller e View:**
   - **php artisan make:controller <nome>** [Gera um Controller];
     - Como criar efetivamente os objetos em um relacionamento usando o ORM Eloquent, por exemplo, para criar uma temporada a partir dá serie:
       **$temporada = $serie->temporadas()->create([‘numero’ => 1]);**

3. **Usando serviços para exclusão:**
   - **DB::transaction**, gerencia os casos de erro automaticamente, fazendo rollback em caso erro. Com **DB::beginTransaction** e **DB::commit,** um no começo e outro no fim da parte do código, há um controle maior, verificando e fazendo rollback de uma transação, mesmo sem que uma exceção seja lançada;
     - Como criar classes auxiliares. Essas classes podem ser criadas na pasta **Provider** ou **Helpers**, no nosso exemplo usamos a pasta **Services**.
     - É boa prática encapsular regras mais complexas dentro de classes;
     - Além disso, você pode injetar algum objeto dessas classes no método do controller;
     - Como lidar transações no Laravel. As **transações** garantem que todos os comandos enviados são executados de uma vez (ou desfeito de uma vez só). No Laravel, basta usar a classe fachada DB para abrir e consolidar a transação;
     - Alternativamente, pode-se usar o método **DB::transaction(..)**, que recebe uma função anômica:
       **DB::transaction (function () use (&$variavel) {});**

4. **Edição da série:**
    - Como usar o JavaScript (Fetch API) para enviar uma requisição e manipular o HTML programaticamente (DOM);

5. **Assistindo episódios:**
    - Como gerar e executar uma migration para adicionar uma nova coluna tabela;
    - Como trabalhar com checkboxes e um array de checkboxes no Laravel;
    - Usa-se **redirect->back()** para voltar para última página;
    - Que podemos enviar o ID de um modelo na requisição e no controller recebemos o objeto;
    - Como incluir uma view em outra, através de sub-views:
      **@include(...)** e **@includeWhen(...):**
      **@include('mensagem', ['mensagem' => $mensagem])**

6. **Autenticando o usuário:**
    - Middlewares são como filtros que podem ser executados antes (ou depois, manipulando a resposta) da requisição chegar ao Controller;
        - Que o Laravel já vem preparada para trabalhar com autenticação:
          **php artisan make:auth**
          Existem rotas (como **/login** ou **/logout**), páginas e controllers (**HomeController**) já prontos para serem utilizados;
        - Que existem várias formas de proteger as rotas:
            - No método do controller, usando **if;**
            - No arquivo web.php, chamando **->middleware('auth');**
            - No construtor da classe controller, usando **$this->middleware('auth');**
            - No arquivo kernel.php, no atributo **$middlewareGroups**;

7. **Protegendo rotas e ações:**
    - Como definir uma rota de logout (chamando **Auth::logout()**);
    - Como definir um menu de navegação (usando o elemento **nav** do HTML e as classes do Bootstrap);
    - Como verificar na view se o usuário está logado ou não:
        - Pode-se usar a diretiva **@auth** para mostrar o conteúdo apenas para usuários logados;
        - Pode-se usar a diretiva **@guest** para mostrar o conteúdo para usuários não logados;
    - Que um middleware é uma espécie de filtro ou interceptador, através dele, é possível executar um código para várias rotas (antes ou depois);

8. **Testes automatizados:**
    - Através de testes, garantimos a qualidade do nosso código;
    - Testes executáveis agilizam e automatizam o controle de qualidade;
    - O Laravel já vem preparado para escrever e rodar testes (existe uma pasta dedicada para testes, que integra com o PHPUnit);
    - Ao executar os testes, recebemos um relatório sobre o status;
    - Para os testes que usam o banco, devemos usar um banco dedicado (por exemplo, em memória). O Laravel já possui uma trait para atualizar o banco especificamente para testes;
    - Devemos ter alguns cuidados na hora do deploy da aplicação:
        - O comando **composer install** é utilizado para baixar todas as dependências do projeto;
        - O comando **php artisan key:generate** gera uma nova chave para criptografar os dados sensíveis;
    - No arquivo .env, a propriedade **APP_ENV** deve ser **prod** e **APP_DEBUG** deve ser false;