1 БИЛЕТ

Для разработки страницы с формой на Laravel, мы должны выполнить ряд шагов:

1. Создать базу данных и таблицу

В первую очередь, нам нужно создать базу данных и таблицу, в которую будут сохраняться данные с формы. Мы можем использовать команду Laravel Artisan для этого. Откройте терминал и выполните следующую команду:

php artisan make:migration create\_users\_table --create=users

Затем, откройте созданный файл миграции в папке database/migrations и определите поля таблицы, как показано ниже:

// database/migrations/[timestamp]\_create\_users\_table.php

public function up()

{

Schema::create('users', function (Blueprint $table) {

$table->increments('id');

$table->string('login');

$table->string('name');

$table->string('password');

$table->timestamps();

});

}

После этого, выполните миграцию для создания таблицы. Запустите следующую команду:

shell

php artisan migrate

2. Создать маршрут и контроллер

Затем, создадим маршрут для страницы с формой и соответствующий контроллер для обработки отправки данных на сервер. Откройте файл routes/web.php и добавьте следующий код:

PHP

// routes/web.php

Route::get('/form', 'FormController@index')->name('form');

Route::post('/submit', 'FormController@submit')->name('submit');

Затем, создайте контроллер FormController с помощью команды Artisan:

shell

php artisan make:controller FormController

Откройте созданный файл контроллера в app/Http/Controllers и добавьте следующие методы:

PHP

// app/Http/Controllers/FormController.php

use Illuminate\Http\Request;

use App\User;

class FormController extends Controller

{

public function index()

{

return view('form');

}

public function submit(Request $request)

{

// Валидация данных

$request->validate([

'login' => 'required',

'name' => 'required',

'password' => 'required',

]);

// Сохранение данных в базе данных

$user = new User;

$user->login = $request->login;

$user->name = $request->name;

$user->password = $request->password;

$user->save();

return redirect()->back()->with('success', 'Данные успешно сохранены!');

}

}

3. Создать представление формы

Теперь, создадим представление формы. Создайте новый файл form.blade.php в папке resources/views и добавьте следующий код:

html

<!-- resources/views/form.blade.php -->

@section('content')

@if($errors->any())

@foreach($errors->all() as $error)

<li>{{$error}}</li>

@endforeach

@endif

@if(session('success'))

{{ session('success') }}

@endif

<form action="{{ route('submit') }}" method="POST">

@csrf

<div>

<label for="login">Логин:</label>

<input type="text" name="login" id="login" value="{{ old('login') }}">

@error('login')

<div>{{ $message }}</div>

@enderror

</div>

<div>

<label for="name">Имя:</label>

<input type="text" name="name" id="name" value="{{ old('name') }}">

@error('name')

<div>{{ $message }}</div>

@enderror

</div>

<div>

<label for="password">Пароль:</label>

<input type="password" name="password" id="password">

@error('password')

<div>{{ $message }}</div>

@enderror

</div>

<button type="submit">Отправить</button>

</form>

Обратите внимание на использование директивы @csrf для защиты от атаки на межсайтовое подделывание запросов (CSRF).

Билет 2

Для создания главной страницы сайта на Laravel, следуйте следующим шагам:

1. Откройте директорию resources/views и создайте новую папку layouts, если ее еще нет. Внутри папки layouts создайте файл app.blade.php, который будет содержать общий макет шаблона для всех страниц.

2. В файле app.blade.php определите следующую структуру шаблона:

PHP

<!DOCTYPE html>

<html>

<head>

<title>Главная страница</title>

</head>

<body>

@include('layouts.header')

<div>

@yield('content')

</div>

@include('layouts.footer')

</body>

</html>

3. Создайте файл header.blade.php внутри папки layouts, который будет содержать шапку сайта:

PHP

<header>

<nav>

<ul>

<li><a href="news">Новости</a></li>

<li><a href="about">О нас</a></li>

<li><a href="catalog">Каталог</a></li>

</ul>

</nav>

</header>

4. Создайте файл footer.blade.php внутри папки layouts, который будет содержать подвал сайта:

PHP

<footer>

<p>Текст пожелания хорошего дня</p>

</footer>

5. Откройте директорию routes и отредактируйте файл web.php для определения маршрутов:

PHP

Route::get('/', function () {

return view('welcome');

});

Route::get('/news', function () {

return view('news');

});

Route::get('/about', function () {

return view('about');

});

Route::get('/catalog', function () {

return view('catalog');

});

6. Создайте файлы welcome.blade.php, news.blade.php, about.blade.php и catalog.blade.php в директории resources/views, которые будут содержать информационный контент по вашему усмотрению, но в пределах допустимых нормами. Например:

welcome.blade.php:

PHP

@extends('layouts.app')

@section('content')

<h1>Добро пожаловать на главную страницу!</h1>

@endsection

news.blade.php:

PHP

@extends('layouts.app')

@section('content')

<h1>Новости</h1>

<p>Здесь будут новости сайта.</p>

@endsection

about.blade.php:

PHP

@extends('layouts.app')

@section('content')

<h1>О нас</h1>

<p>Мы занимаемся... (информация о вашей компании/проекте).</p>

@endsection

catalog.blade.php:

PHP

@extends('layouts.app')

@section('content')

<h1>Каталог</h1>

<p>В нашем каталоге представлены... (описание продуктов/услуг).</p>

@endsection

Теперь вы можете открыть вашу главную страницу по адресу / и увидеть шапку сайта с работающими переходами на страницы "Новости", "О нас", "Каталог", а также текст пожелания хорошего дня. При переходе на каждую страницу, соответствующий контент будет отображаться в блоке @yield('content') шаблона app.blade.php.

Билет №3

Давайте создадим мини-проект на фреймворке Laravel с использованием стилей Bootstrap. Следуйте инструкциям ниже:

1. Установка Laravel:

Убедитесь, что у вас установлен Composer, затем выполните следующую команду в терминале:

shell

composer create-project --prefer-dist laravel/laravel feedback-form

2. Настройка маршрутов:

Отредактируйте файл routes/web.php:

PHP

Route::get('/', function () {

return view('welcome');

});

Route::get('/client/message', [\App\Http\Controllers\FeedbackController::class, 'showMessageForm']);

Route::post('/client/message', [\App\Http\Controllers\FeedbackController::class, 'processMessageForm'])->name('processMessageForm');

3. Создание контроллера:

Создайте контроллер с помощью следующей команды:

php artisan make:controller FeedbackController

Отредактируйте файл app/Http/Controllers/FeedbackController.php:

PHP

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class FeedbackController extends Controller

{

public function showMessageForm()

{

return view('message\_form');

}

public function processMessageForm(Request $request)

{

$request->validate([

'email' => 'required|email',

'message' => 'required|min:10',

]);

$data = [

'email' => $request->input('email'),

'message' => $request->input('message'),

];

dd($data);

}

}

4. Создание представления:

Создайте файл resources/views/message\_form.blade.php:

PHP

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Feedback Form</title>

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css">

</head>

<body>

<div class="container mt-5">

<form method="post" action="{{ route('processMessageForm') }}">

@csrf <div class="form-group">

<label for="email">Email:</label>

<input type="email" class="form-control" id="email" name="email" required>

</div>

<div class="form-group">

<label for="message">Message:</label>

<textarea class="form-control" id="message" name="message" rows="4" required></textarea>

</div>

<button type="submit" class="btn btn-primary">Submit</button>

</form>

</div>

</body>

</html>

5. Запуск веб-сервера:

Запустите веб-сервер Laravel с помощью следующей команды:

shell

php artisan serve

После этого вы сможете открыть ваш браузер и перейти по адресу http://127.0.0.1:8000/client/message, чтобы увидеть форму обратной связи.

Билет №4

### Шаг 1: Установка Laravel

Установите Laravel, если у вас его еще нет. Вы можете использовать Composer для этого:

Bash

composer create-project --prefer-dist laravel/laravel your-project-name

### Шаг 2: Создание миграции

Создайте миграцию для таблицы заявок:

Bash

php artisan make:model Application -m

В отредактированном файле database/migrations/create\_applications\_table миграции добавьте следующий код:

PHP

// database/migrations/YYYY\_MM\_DD\_create\_applications\_table.php

use Illuminate\Database\Migrations\Migration;

use Illuminate\Database\Schema\Blueprint;

use Illuminate\Support\Facades\Schema;

class CreateApplicationsTable extends Migration

{

public function up()

{

Schema::create('applications', function (Blueprint $table) {

$table->id();

$table->string('full\_name');

$table->string('email');

$table->string('article\_title');

$table->enum('educational\_institution', ['Школа', 'Техникум', 'Колледж']);

$table->string('city');

$table->timestamps();

});

}

public function down()

{

Schema::dropIfExists('applications');

}

}

Затем выполните миграцию:

Bash

php artisan migrate

### Шаг 3: Создание контроллера

Создайте контроллер для управления заявками:

Bash

php artisan make:controller ApplicationController

Отредактируйте созданный контроллер (app/Http/Controllers/ApplicationController.php) следующим образом:

PHP

// app/Http/Controllers/ApplicationController.php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use App\Models\Application;

class ApplicationController extends Controller

{

public function create()

{

return view('application.create');

}

public function store(Request $request)

{

$request->validate([

'full\_name' => 'required',

'email' => 'required|email',

'article\_title' => 'required|min:5|max:50',

'educational\_institution' => 'required|in:Школа,Техникум,Колледж',

'city' => 'required|regex:/^[^0-9]+$/',

]);

$application = new Application();

$application ->full\_name = $request->input('full\_name');

$application ->email= $request->input('email');

$application ->article\_title = $request->input('article\_title');

$application ->educational\_institution = $request->input('educational\_institution');

$application ->city= $request->input('city');

$application ->save();

return redirect()->route('application.create')->with('success', 'Заявка успешно отправлена!');

}

}

### Шаг 4: Создание маршрутов

Отредактируйте файл routes/web.php:

PHP

// routes/web.php

use App\Http\Controllers\ApplicationController;

Route::get('/application/create', [ApplicationController::class, 'create'])->name('application.create');

Route::post('/application/store', [ApplicationController::class, 'store'])->name('application.store');

create\_applications\_table

### Шаг 5: Создание представлений

Создайте представление для формы в resources/views/application/create.blade.php:

html

<!-- resources/views/application/create.blade.php -->

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Форма отправки заявки</title>

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">

</head>

<body>

<div class="container mt-5">

<h1>Форма отправки заявки</h1>

@if(session('success'))

<div class="alert alert-success">

{{ session('success') }}

</div>

@endif

@if($errors->any())

<div class="alert alert-danger">

<ul>

@foreach($errors->all() as $error)

<li>{{ $error }}</li>

@endforeach

</ul>

</div>

@endif

<form method="post" action="{{ route('application.store') }}">

@csrf

<div class="form-group">

<label for="full\_name">ФИО:</label>

<input type="text" name="full\_name" class="form-control" required>

</div>

<div class="form-group">

<label for="email">Email:</label>

<input type="email" name="email" class="form-control" required>

</div>

<div class="form-group">

<label for="article\_title">Название статьи:</label>

<input type="text" name="article\_title" class="form-control" required>

</div>

<div class="form-group">

<label for="educational\_institution">Учебная организация:</label>

<select name="educational\_institution" class="form-control" required>

<option value="Школа">Школа</option>

<option value="Техникум">Техникум</option>

<option value="Колледж">Колледж</option>

</select>

</div>

<div class="form-group">

<label for="city">Город:</label>

<input type="text" name="city" class="form-control" required>

</div>

<button type="submit" class="btn btn-primary">Отправить заявку</button>

</form>

</div>

<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>

<script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.0.7/dist/umd/popper.min.js"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>

</body>

</html>

### Шаг 6: Запуск приложения

Запустите веб-сервер:

Bash

php artisan serve

Откройте приложение в браузере по адресу http://localhost:8000/application/create и протестируйте форму.

Это простой пример, и в реальном проекте вы, вероятно, захотите добавить дополнительные функции, такие как защита от CSRF, стили Bootstrap и т.д.

> ОТВЕТЫ НА БИЛЕТЫ(канал): БИЛЕТ 5  
### Шаг 1: Установка Laravel  
  
Установите Laravel с помощью Composer, если у вас его еще нет:  
  
```bash  
composer create-project --prefer-dist laravel/laravel product-management  
cd product-management  
```  
  
### Шаг 2: Создание базы данных и миграции  
  
Создайте миграцию для таблицы "products":  
  
```bash  
php artisan make:migration create\_products\_table  
```  
  
Отредактируйте созданную миграцию в файле database/migrations/xxxx\_xx\_xx\_create\_products\_table.php следующим образом:  
  
```php  
use Illuminate\Database\Migrations\Migration;  
use Illuminate\Database\Schema\Blueprint;  
use Illuminate\Support\Facades\Schema;  
  
class CreateProductsTable extends Migration  
{  
public function up()  
{  
Schema::create('products', function (Blueprint $table) {  
$table->id();  
$table->string('name');  
$table->string('manufacturer');  
$table->decimal('price', 8, 2);  
$table->date('manufacture\_date');  
$table->date('expiry\_date');  
$table->timestamps();  
});  
}  
  
public function down()  
{  
Schema::dropIfExists('products');  
}  
}  
```  
  
Выполните миграцию для создания таблицы:  
  
```bash  
php artisan migrate  
```  
  
### Шаг 3: Создание модели Product  
  
Создайте модель Product:  
  
```bash  
php artisan make:model Product  
```  
  
Конечно, вместо валидации в контроллере мы можем использовать Laravel Form Request для управления валидацией. Давайте создадим Request файл для валидации полей продукта.  
  
### Шаг 4: Создание ProductRequest  
  
Создайте файл запроса с помощью Artisan команды:  
  
```bash  
php artisan make:request ProductRequest  
```  
  
Отредактируйте файл app/Http/Requests/ProductRequest.php следующим образом:  
  
```php  
<?php  
  
namespace App\Http\Requests;  
  
use Illuminate\Foundation\Http\FormRequest;  
  
class ProductRequest extends FormRequest  
{  
public function authorize()  
{  
return true;  
}  
  
public function rules()  
{  
return [  
'name' => 'required|string',  
'manufacturer' => 'required|string',  
'price' => 'required|numeric',  
'manufacture\_date' => 'required|date',  
'expiry\_date' => 'required|date|after:manufacture\_date',  
];  
}  
}  
```  
  
Добавьте маршруты и контроллер для отображения и создания продуктов:  
  
### Шаг 5: Создание контроллера и маршрутов  
  
```bash  
php artisan make:controller ProductController  
```  
  
Отредактируйте app/Http/Controllers/ProductController.php:  
  
```php  
<?php  
  
namespace App\Http\Controllers;  
  
use App\Models\Product;  
use App\Http\Requests\ProductRequest;  
use Illuminate\Http\Request;  
  
class ProductController extends Controller  
{  
public function index()  
{  
$products = Product::all();  
  
return view('products.index', compact('products'));  
}  
  
public function create()  
{  
return view('products.create');  
}  
  
public function store(ProductRequest $request)  
{  
$product = new Product();  
$product->name = $request->input('name');  
$product->manufacturer = $request->input('manufacturer');  
$product->price = $request->input('price');  
$product->manufacture\_date = $request->input('manufacture\_date');  
$product->expiry\_date = $request->input('expiry\_date');  
  
// Дополнительные действия перед сохранением, если нужно  
// Например, можно добавить логику проверки перед сохранением  
  
$product->save();  
  
return redirect()->route('products.index')->with('success', 'Product created successfully');  
}  
}  
```  
  
  
### Шаг 6: Создание представлений  
  
Создайте представления в папке resources/views/products/:  
  
1. index.blade.php для отображения списка продуктов.  
2. create.blade.php для создания нового продукта. > ОТВЕТЫ НА БИЛЕТЫ(канал): !!!!Продолжение Билет 5.  
  
Пример resources/views/products/create.blade.php:  
```blade  
<!-- resources/views/products/create.blade.php -->  
  
<!DOCTYPE html>  
<html lang="en">  
<head>  
<meta charset="UTF-8">  
<meta name="viewport" content="width=device-width, initial-scale=1.0">  
<title>Create Product</title>  
<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">  
</head>  
<body>  
<div class="container mt-5">  
<h2>Create Product</h2>  
  
@if ($errors->any())  
<div class="alert alert-danger">  
<ul>  
@foreach ($errors->all() as $error)  
<li>{{ $error }}</li>  
@endforeach  
</ul>  
</div>  
@endif  
  
<form action="{{ route('products.store') }}" method="post">  
@csrf  
  
<div class="form-group">  
<label for="name">Name</label>  
<input type="text" class="form-control" name="name" required>  
</div>  
  
<div class="form-group">  
<label for="manufacturer">Manufacturer</label>  
<input type="text" class="form-control" name="manufacturer" required>  
</div>  
  
<div class="form-group">  
<label for="price">Price</label>  
<input type="text" class="form-control" name="price" required>  
</div>  
  
<div class="form-group">  
<label for="manufacture\_date">Manufacture Date</label>  
<input type="date" class="form-control" name="manufacture\_date" required>  
</div>  
  
<div class="form-group">  
<label for="expiry\_date">Expiry Date</label>  
<input type="date" class="form-control" name="expiry\_date" required>  
</div>  
  
<button type="submit" class="btn btn-primary">Create Product</button>  
</form>  
</div>  
</body>  
</html>  
```  
  
### Шаг 7: Создание index.blade.php  
  
```blade  
<!-- resources/views/products/index.blade.php -->  
  
<!DOCTYPE html>  
<html lang="en">  
<head>  
<meta charset="UTF-8">  
<meta name="viewport" content="width=device-width, initial-scale=1.0">  
<title>Product List</title>  
<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">  
</head>  
<body>  
<div class="container mt-5">  
<h2>Product List</h2>  
  
@if (session('success'))  
<div class="alert alert-success">  
{{ session('success') }}  
</div>  
@endif  
  
<table class="table">  
<thead>  
<tr>  
<th>Name</th>  
<th>Manufacturer</th>  
<th>Price</th>  
<th>Manufacture Date</th>  
<th>Expiry Date</th>  
</tr>  
</thead>  
<tbody>  
@foreach ($products as $product)  
<tr>  
<td>{{ $product->name }}</td>  
<td>{{ $product->manufacturer }}</td>  
<td>{{ $product->price }}</td>  
<td>{{ $product->manufacture\_date }}</td>  
<td>{{ $product->expiry\_date }}</td>  
</tr>  
@endforeach  
</tbody>  
</table>  
<a href="{{ route('products.create') }}" class='btn btn-primary'>Добавить</a>  
</div>  
</body>  
</html>  
```  
  
Шаг 8: Регистрация маршрутов  
  
Отредактируйте файл routes/web.php:  
  
```php  
use App\Http\Controllers\ProductController;  
  
// Маршруты для отображения формы создания продукта и обработки данных  
Route::get('products/create', [ProductController::class, 'create'])->name('products.create');  
Route::post('products', [ProductController::class, 'store'])->name('products.store');  
  
// Маршрут для отображения списка продуктов  
Route::get('products', [ProductController::class, 'index'])->name('products.index');  
```

> ОТВЕТЫ НА БИЛЕТЫ(канал): Билет №6.  
  
Шаг 1: Установка Laravel  
```bash  
composer create-project --prefer-dist laravel/laravel feedback-form  
cd feedback-form  
```  
  
Шаг 2: Создание миграции и модели для отзывов  
```bash  
php artisan make:model Feedback -m  
```  
  
Отредактируйте миграцию в файле database/migrations/xxxx\_xx\_xx\_create\_feedbacks\_table.php:  
  
```php  
// database/migrations/xxxx\_xx\_xx\_create\_feedbacks\_table.php  
  
use Illuminate\Database\Migrations\Migration;  
use Illuminate\Database\Schema\Blueprint;  
use Illuminate\Support\Facades\Schema;  
  
class CreateFeedbacksTable extends Migration  
{  
public function up()  
{  
Schema::create('feedback', function (Blueprint $table) {  
$table->id();  
$table->string('name');  
$table->string('email');  
$table->text('message');  
$table->timestamps();  
});  
}  
  
public function down()  
{  
Schema::dropIfExists('feedbacks');  
}  
}  
```  
  
Затем выполните миграцию:  
```bash  
php artisan migrate  
```  
  
Шаг 3: Создание контроллера  
```bash  
php artisan make:controller FeedbackController  
```  
  
Отредактируйте контроллер в файле app/Http/Controllers/FeedbackController.php:  
  
```php  
// app/Http/Controllers/FeedbackController.php  
  
namespace App\Http\Controllers;  
  
use Illuminate\Http\Request;  
use App\Models\Feedback;  
  
class FeedbackController extends Controller  
{  
public function index()  
{  
return view('feedback.index');  
}  
  
public function store(Request $request)  
{  
$request->validate([  
'name' => 'required|string|max:255',  
'email' => 'required|email|max:255',  
'message' => 'required|string',  
]);  
  
$feedback = new Feedback();  
$feedback->name = $request->input('name');  
$feedback->email = $request->input('email');  
$feedback->message = $request->input('message');  
$feedback->save();  
  
return redirect()->route('feedback.index')->with('success', 'Отзыв успешно добавлен!');  
}  
}  
```  
  
Шаг 4: Создание маршрутов  
Отредактируйте файл routes/web.php:  
  
```php  
// routes/web.php  
  
use App\Http\Controllers\FeedbackController;  
  
Route::get('/', [FeedbackController::class, 'index'])->name('feedback.index');  
Route::post('/feedback', [FeedbackController::class, 'store']);  
```  
> ОТВЕТЫ НА БИЛЕТЫ(канал): !!!!Продолжение Билет 6.  
  
Шаг 5: Создание представления  
Создайте файл resources/views/feedback/index.blade.php:  
  
```blade  
<!-- resources/views/feedback/index.blade.php -->  
  
<!DOCTYPE html>  
<html lang="en">  
<head>  
<meta charset="UTF-8">  
<meta name="viewport" content="width=device-width, initial-scale=1.0">  
<title>Отзывы</title>  
<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">  
</head>  
<body>  
<div class="container mt-5">  
<div class="row">  
<div class="col-md-6 offset-md-3">  
<h2>Форма отзывов</h2>  
@if(session('success'))  
<div class="alert alert-success">  
{{ session('success') }}  
</div>  
@endif  
  
<form action="{{ url('/feedback') }}" method="post">  
@csrf  
<div class="form-group">  
<label for="name">Имя:</label>  
<input type="text" name="name" class="form-control" required>  
</div>  
<div class="form-group">  
<label for="email">Email:</label>  
<input type="email" name="email" class="form-control" required>  
</div>  
<div class="form-group">  
<label for="message">Сообщение:</label>  
<textarea name="message" class="form-control" required></textarea>  
</div>  
<button type="submit" class="btn btn-primary">Отправить отзыв</button>  
</form>  
</div>  
</div>  
</div>  
</body>  
</html>  
```

> ОТВЕТЫ НА БИЛЕТЫ(канал): Билет №7.  
  
Для создания мини-проекта на фреймворке Laravel с использованием стилей Bootstrap, выполните следующие шаги:  
1. **Установка Laravel:**  
Установите Laravel с использованием Composer, если у вас его нет:  
```bash  
   composer create-project --prefer-dist laravel/laravel your-project-name  
```  
2. Настройка базы данных:  
Отредактируйте файл .env в корне проекта, чтобы настроить подключение к базе данных:  
```env  
   DB\_CONNECTION=mysql  
DB\_HOST=your-database-host  
DB\_PORT=your-database-port  
DB\_DATABASE=your-database-name  
DB\_USERNAME=your-database-username  
DB\_PASSWORD=your-database-password  
```  
3. Создание миграций и моделей:  
Создайте миграции для таблицы с людьми:  
  
```bash  
   php artisan make:model Person -m  
```  
   Внутри созданной миграции в методе up() определите структуру таблицы или Отредактируйте миграцию в файле database/migrations/xxxx\_xx\_xx\_create\_feedbacks\_table.php:  
  
```php  
use Illuminate\Database\Migrations\Migration;  
use Illuminate\Database\Schema\Blueprint;  
use Illuminate\Support\Facades\Schema;  
  
class CreatePeopleTable extends Migration  
{  
public function up()  
{  
Schema::create('people', function (Blueprint $table) {  
$table->id();  
$table->string('name');  
$table->string('gender');  
$table->integer('age');  
$table->string('city')->nullable();  
// Добавьте другие поля, если они есть в вашей таблице  
$table->timestamps();  
});  
}  
  
public function down()  
{  
Schema::dropIfExists('people');  
}  
}  
```  
И затем выполните миграцию:  
```bash  
   php artisan migrate  
```  
4. Создание маршрутов:  
В файле routes/web.php определите маршруты для каждого запроса:  
  
```php  
   use App\Http\Controllers\PersonController;  
  
Route::get('/women-with-name-alina', [PersonController::class, 'womenWithNameAlina']);  
Route::get('/boys-with-age-under-16', [PersonController::class, 'boysWithAgeUnder16']);  
Route::get('/delete-people-without-city', [PersonController::class, 'deletePeopleWithoutCity']);  
```  
  
5. Создание контроллера:  
Создайте контроллер PersonController с помощью команды Artisan:  
```bash  
   php artisan make:controller PersonController  
```  
   Затем в контроллере определите методы для каждого запроса.  
```php  
   // app/Http/Controllers/PersonController.php  
  
namespace App\Http\Controllers;  
  
use App\Models\Person;  
use Illuminate\Http\Request;  
  
class PersonController extends Controller  
{  
public function womenWithNameAlina()  
{  
$women = Person::where('gender', 'female')->where('name', 'Alina')->get();  
return view('women', compact('women'));  
}  
  
public function boysWithAgeUnder16()  
{  
$boys = Person::where('gender', 'male')->where('age', '<=', 16)->orderBy('name')->get();  
return view('boys', compact('boys'));  
}  
  
public function deletePeopleWithoutCity()  
{  
Person::whereNull('city')->delete();  
return redirect()->back();  
}  
}  
```  
> ОТВЕТЫ НА БИЛЕТЫ(канал): !!!!Продолжение Билет 7.  
6. Создание представлений:  
Создайте представления для каждого запроса в папке resources/views.  
  
- resources/views/women.blade.php:  
```php  
blade  
@extends('layouts.app')  
  
@section('content')  
<h2>Women with Name Alina</h2>  
<ul>  
@foreach($women as $woman)  
<li>{{ $woman->name }}</li>  
@endforeach </ul>  
@endsection  
```  
   - resources/views/boys.blade.php:  
```php  
blade  
@extends('layouts.app')  
  
@section('content')  
<h2>Boys with Age Under 16</h2>  
<ul>  
@foreach($boys as $boy)  
<li>{{ $boy->name }}</li>  
@endforeach </ul>  
@endsection  
```  
   7. Обновление макетов:  
Обновите макеты в папке resources/views/layouts (например, app.blade.php), чтобы использовать стили Bootstrap.  
```php  
blade  
<!DOCTYPE html>  
<html lang="en">  
<head>  
<meta charset="utf-8">  
<meta name="viewport" content="width=device-width, initial-scale=1">  
<title>Laravel Bootstrap Project</title>  
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0/dist/css/bootstrap.min.css" rel="stylesheet">  
</head>  
<body>  
<div class="container mt-5">  
@yield('content')  
</div>  
</body>  
</html>  
```

> ОТВЕТЫ НА БИЛЕТЫ(канал): Билет №8.  
  
Шаг 1: Создание нового проекта Laravel  
  
```bash  
composer create-project --prefer-dist laravel/laravel ProductManager  
cd ProductManager  
```  
  
Шаг 2: Создание базы данных и миграции  
  
Создайте миграцию для таблицы товаров:  
  
```bash  
php artisan make:model Product -m  
```  
  
Отредактируйте созданный файл миграции в директории database/migrations для добавления нужных полей:  
  
```php  
// внутри файла миграции  
public function up()  
{  
Schema::create('products', function (Blueprint $table) {  
$table->id();  
$table->string('name');  
$table->string('category');  
$table->decimal('price', 8, 2);  
$table->timestamps();  
});  
}  
```  
  
Примените миграции:  
  
```bash  
php artisan migrate  
```  
  
Шаг 3: Создание модели и контроллера  
  
Создайте модель и контроллер для работы с продуктами:  
  
```bash  
php artisan make:controller ProductController  
```  
  
Отредактируйте созданный контроллер (app/Http/Controllers/ProductController.php), чтобы добавить методы для отображения формы и обработки данных:  
  
```php  
// ProductController.php  
  
use App\Models\Product;  
use Illuminate\Http\Request;  
  
class ProductController extends Controller  
{  
public function create()  
{  
return view('products.create');  
}  
  
public function store(Request $request)  
{  
$request->validate([  
'name' => 'required|max:20',  
'category' => 'required|in:Одежда,Обувь',  
'price' => 'required|numeric',  
]);  
  
$product = new Product;  
$product->name = $request->name;  
$product->category = $request->category;  
$product->price = $request->price;  
$product->save();  
  
return redirect()->route('products.create')->with('success', 'Товар успешно добавлен');  
}  
}  
```  
  
Шаг 4: Создание представления для формы  
  
Создайте представление для формы (resources/views/products/create.blade.php):  
  
```html  
<!-- resources/views/products/create.blade.php -->  
  
<!DOCTYPE html>  
<html lang="en">  
<head>  
<meta charset="UTF-8">  
<meta name="viewport" content="width=device-width, initial-scale=1.0">  
<title>Добавление нового товара</title>  
<link href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css" rel="stylesheet">  
</head>  
<body>  
<div class="container mt-5">  
<h2>Добавление нового товара</h2>  
  
@if(session('success'))  
<div class="alert alert-success">  
{{ session('success') }}  
</div>  
@endif  
  
<form action="{{ route('products.store') }}" method="post">  
@csrf  
  
<div class="form-group">  
<label for="name">Наименование:</label>  
<input type="text" class="form-control" id="name" name="name" >  
</div>  
  
<div class="form-group">  
<label for="category">Категория:</label>  
<select class="form-control" id="category" name="category">  
<option value="Одежда">Одежда</option>  
<option value="Обувь">Обувь</option>  
<option value="Шапка">Шапка</option>  
</select>  
</div>  
  
<div class="form-group">  
<label for="price">Стоимость:</label>  
<input type="text" class="form-control" id="price" name="price">  
</div>  
  
<button type="submit" class="btn btn-primary">Добавить товар</button>  
</form>  
</div>  
  
<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"></script>  
<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js"></script>  
<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"></script>  
</body>  
</html>  
```  
Шаг 5: Регистрация маршрутов  
  
Отредактируйте файл routes/web.php:  
  
```php  
// routes/web.php  
  
use App\Http\Controllers\ProductController;  
  
Route::get('/', [ProductController::class, 'create'])->name('products.create');  
Route::post('products/store', [ProductController::class, 'store'])->name('products.store');  
```  
> ОТВЕТЫ НА БИЛЕТЫ(канал): !!!!Продолжение Билет 8.  
  
6. Создание представлений:  
Создайте представления для каждого запроса в папке resources/views.  
  
- resources/views/women.blade.php:  
```blade  
     @extends('layouts.app')  
  
@section('content')  
<h2>Women with Name Alina</h2>  
<ul>  
@foreach($women as $woman)  
<li>{{ $woman->name }}</li>  
@endforeach </ul>  
@endsection  
```  
   - resources/views/boys.blade.php:  
```blade  
     @extends('layouts.app')  
  
@section('content')  
<h2>Boys with Age Under 16</h2>  
<ul>  
@foreach($boys as $boy)  
<li>{{ $boy->name }}</li>  
@endforeach </ul>  
@endsection  
```  
   7. Обновление макетов:  
Обновите макеты в папке resources/views/layouts (например, app.blade.php), чтобы использовать стили Bootstrap.  
```blade  
   <!DOCTYPE html>  
<html lang="en">  
<head>  
<meta charset="utf-8">  
<meta name="viewport" content="width=device-width, initial-scale=1">  
<title>Laravel Bootstrap Project</title>  
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0/dist/css/bootstrap.min.css" rel="stylesheet">  
</head>  
<body>  
<div class="container mt-5">  
@yield('content')  
</div>  
</body>  
</html>  
```

Билет №9  
  
1. Установка Laravel:  
   Убедитесь, что у вас установлен Composer, и выполните команду для создания нового проекта Laravel:

Bash

composer create-project --prefer-dist laravel/laravel YourProjectName

2. Настройка базы данных:  
   Отредактируйте файл .env в корне вашего проекта, установив параметры подключения к базе данных:

env

DB\_CONNECTION=mysql

DB\_HOST=127.0.0.1

DB\_PORT=3306

DB\_DATABASE=your\_database\_name

DB\_USERNAME=your\_database\_username

DB\_PASSWORD=your\_database\_password

3. Создание миграции и модели:  
   Создайте миграцию для таблицы товаров:

Bash

php artisan make:migration create\_products\_table

   Отредактируйте созданную миграцию в database/migrations с необходимыми полями

PHP

use Illuminate\Database\Migrations\Migration;

use Illuminate\Database\Schema\Blueprint;

use Illuminate\Support\Facades\Schema;

class CreateProductsTable extends Migration

{

public function up()

{

Schema::create('products', function (Blueprint $table) {

$table->id();

$table->string('name');

$table->text('description');

*// Добавьте другие поля товара здесь*

$table->timestamps();

});

}

public function down()

{

Schema::dropIfExists('products');

}

}

А затем выполните миграцию:

Bash

php artisan migrate

Создайте модель для товаров:

Bash

php artisan make:model Product

   В модели Product.php определите соответствующую таблицу и заполните ее данными.  
  
4. Создание контроллера:  
   Создайте контроллер, который будет отвечать за отображение страницы:

php artisan make:controller ProductController

   Отредактируйте контроллер в app/Http/Controllers/ProductController.php.

PHP

<?php

namespace App\Http\Controllers;

use App\Models\Product;

use Illuminate\Http\Request;

class ProductController extends Controller

{

public function index()

{

$products = Product::all();

return view('products.index', compact('products'));

}

}

5. Создание маршрута:  
   Определите маршрут в routes/web.php:

PHP

use App\Http\Controllers\ProductController;

Route::get('/products', [ProductController::class, 'index']);

6. Создание представления:  
   Создайте представление resources/views/products/index.blade.php:

blade

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Products</title>

<!-- Bootstrap CSS -->

<link href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css" rel="stylesheet">

</head>

<body>

<div class="container">

<div class="row">

@foreach($products as $product)

<div class="col-md-4 mb-4">

<div class="card">

<div class="card-body">

<h5 class="card-title">{{ $product->name }}</h5>

<p class="card-text">{{ $product->description }}</p>

<a href="{{ route('products.show', $product->id) }}" class="btn btn-primary">Просмотреть далее</a>

</div>

</div>

</div>

@endforeach

</div>

</div>

<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"></script>

</body>

</html>

7. Создание маршрута и метода для просмотра деталей товара:  
   В контроллере ProductController.php добавьте метод:

PHP

public function show($id)

{

$product = Product::findOrFail($id);

return view('products.show', compact('product'));

}

   Добавьте маршрут в routes/web.php:

PHP

Route::get('/products/{id}', [ProductController::class, 'show'])->name('products.show');

!!!!Продолжение 9-того!!!!

Создать ещё одно представление в view/products

resources/views/products/show.blade.php

blade

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>{{ $product->name }}</title>

<!-- Bootstrap CSS -->

<link href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css" rel="stylesheet">

</head>

<body>

<div class="container">

<div class="card">

<div class="card-body">

<h5 class="card-title">{{ $product->name }}</h5>

<p class="card-text">{{ $product->description }}</p>

<!-- Добавьте здесь остальные поля товара -->

</div>

</div>

</div>

<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"></script>

</body>

</html>

> ОТВЕТЫ НА БИЛЕТЫ(канал): Билет 10.  
  
### Шаг 1: Установка Laravel и настройка проекта  
  
1. Установите Laravel:   
  
```bash  
composer create-project --prefer-dist laravel/laravel your-project-name  
```  
  
2. Перейдите в каталог вашего проекта:  
  
```bash  
cd your-project-name  
```  
  
### Шаг 2: Создание модели и миграций  
  
1. Создайте миграции для таблицы с данными:  
  
```bash  
php artisan make:migration create\_people\_table  
```  
  
2. Отредактируйте миграцию в файле database/migrations/YYYY\_MM\_DD\_create\_people\_table.php,   
  
```php  
use Illuminate\Database\Migrations\Migration;  
use Illuminate\Database\Schema\Blueprint;  
use Illuminate\Support\Facades\Schema;  
  
class CreatePeopleTable extends Migration  
{  
public function up()  
{  
Schema::create('people', function (Blueprint $table) {  
$table->id();  
$table->string('name');  
$table->string('gender');  
$table->integer('age');  
$table->string('city');  
// Добавьте другие необходимые столбцы  
$table->timestamps();  
});  
}  
  
public function down()  
{  
Schema::dropIfExists('people');  
}  
}  
```  
  
3. Запустите миграции:  
  
```bash  
php artisan migrate  
```  
  
4. Создайте модель для работы с данными:  
  
```bash  
php artisan make:model Person  
```  
  
  
### Шаг 4: Контроллер  
  
1. Создайте контроллер:  
  
```bash  
php artisan make:controller PersonController  
```  
  
2. В контроллере PersonController определите методы для выполнения каждого из запросов.  
  
### Шаг 5: Создание представлений  
  
1. Создайте представления для каждого запроса в папке resources/views.  
  
### Шаг 6: Настройка маршрутов  
  
1. Определите маршруты в файле routes/web.php для каждого запроса.  
  
```php  
use App\Http\Controllers\PersonController;  
  
Route::get('/', function () {  
return view('welcome');  
});  
  
Route::get('/women-not-named-gulnaz', [PersonController::class, 'womenNotNamedGulnaz'])  
->name('women-not-named-gulnaz');  
  
Route::get('/people-under-18', [PersonController::class, 'peopleUnder18'])  
->name('people-under-18');  
  
Route::get('/delete-by-city-ending', [PersonController::class, 'deleteByCityEnding'])  
->name('delete-by-city-ending');  
  
Route::get('/records-except-id-4', [PersonController::class, 'recordsExceptId4'])  
->name('records-except-id-4');  
```  
> ОТВЕТЫ НА БИЛЕТЫ(канал): !!!!Продолжение Билет 10.  
### Шаг 7. Создание представлений  
  
### 1. Вывести всех женщин, которых не зовут Гульназ  
womenNotNamedGulnaz.blade.php  
```blade  
<!-- resources/views/womenNotNamedGulnaz.blade.php -->  
  
@extends('layouts.app')  
  
@section('content')  
<h1>Women Not Named Gulnaz</h1>  
<ul>  
@foreach($womenNotNamedGulnaz as $person)  
<li>{{ $person->name }} - {{ $person->gender }}</li>  
@endforeach  
</ul>  
@endsection  
```  
  
### 2. Вывести всех людей, чей возраст не превышает 18 лет  
peopleUnder18.blade.php  
```blade  
<!-- resources/views/peopleUnder18.blade.php -->  
  
@extends('layouts.app')  
  
@section('content')  
<h1>People Under 18</h1>  
<ul>  
@foreach($peopleUnder18 as $person)  
<li>{{ $person->name }} - {{ $person->age }} years old</li>  
@endforeach  
</ul>  
@endsection  
```  
  
### 3. Удалить из БД все поля, которые в столбце Город содержат окончание «тов»  
deleteByCityEnding.blade.php  
```blade  
<!-- resources/views/deleteByCityEnding.blade.php -->  
  
@extends('layouts.app')  
  
@section('content')  
<h1>Delete Records by City Ending</h1>  
<p>{{ $recordsDeleted }} records deleted.</p>  
@endsection  
```  
  
### 4. Выборка лишь тех записей, у которых id не равно значению 4  
recordsExceptId4.blade.php  
```blade  
<!-- resources/views/recordsExceptId4.blade.php -->  
  
@extends('layouts.app')  
  
@section('content')  
<h1>Records Except ID 4</h1>  
<ul>  
@foreach($recordsExceptId4 as $record)  
<li>{{ $record->name }} - {{ $record->id }}</li>  
@endforeach  
</ul>  
@endsection  
```  
  
### Шаг 5. Создать layouts/app  
```blade  
<!-- resources/views/layouts/app.blade.php -->  
  
<!DOCTYPE html>  
<html lang="en">  
<head>  
<meta charset="utf-8">  
<meta name="viewport" content="width=device-width, initial-scale=1">  
<title>Your Laravel App</title>  
  
<!-- Bootstrap CSS -->  
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css" rel="stylesheet">  
</head>  
<body>  
  
<nav class="navbar navbar-expand-lg navbar-light bg-light">  
<div class="container-fluid">  
<a class="navbar-brand" href="#">Your App Name</a>  
<button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarNav" aria-controls="navbarNav" aria-expanded="false" aria-label="Toggle navigation">  
<span class="navbar-toggler-icon"></span>  
</button>  
<div class="collapse navbar-collapse" id="navbarNav">  
<ul class="navbar-nav">  
<li class="nav-item active">  
<a class="nav-link" href="/">Home</a>  
</li>  
<!-- Add more navigation links as needed -->  
</ul>  
</div>  
</div>  
</nav>  
  
<div class="container mt-4">  
@yield('content')  
</div>  
  
<!-- Bootstrap JS (optional) -->  
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"></script>  
</body>  
</html>  
```  
> ОТВЕТЫ НА БИЛЕТЫ(канал): !!!!Продолжение Билет 10.  
  
### 5. Пределать Welcome.blade  
```blade  
<!-- resources/views/welcome.blade.php -->  
  
@extends('layouts.app')  
  
@section('content')  
<div class="jumbotron">  
<h1 class="display-4">Welcome to Your Laravel App</h1>  
<p class="lead">This is a simple Laravel application with Bootstrap styles.</p>  
<hr class="my-4">  
<p>Choose an option below:</p>  
<ul>  
<li><a href="{{ route('women-not-named-gulnaz') }}">Women Not Named Gulnaz</a></li>  
<li><a href="{{ route('people-under-18') }}">People Under 18</a></li>  
<li><a href="{{ route('delete-by-city-ending') }}">Delete by City Ending</a></li>  
<li><a href="{{ route('records-except-id-4') }}">Records Except ID 4</a></li>  
</ul>  
</div>  
@endsection  
```  
  
### Шаг 6. Заполнить контроллер  
Контроллер PersonController будет содержать методы для выполнения каждого из запросов. Давайте создадим простую версию:  
  
```php  
// app/Http/Controllers/PersonController.php  
  
namespace App\Http\Controllers;  
  
use App\Models\Person;  
use Illuminate\Http\Request;  
  
class PersonController extends Controller  
{  
public function womenNotNamedGulnaz()  
{  
$womenNotNamedGulnaz = Person::where('gender', 'female')->where('name', '!=', 'Гульназ')->get();  
return view('womenNotNamedGulnaz', compact('womenNotNamedGulnaz'));  
}  
  
public function peopleUnder18()  
{  
$peopleUnder18 = Person::where('age', '<=', 18)->get();  
return view('peopleUnder18', compact('peopleUnder18'));  
}  
  
public function deleteByCityEnding()  
{  
$recordsDeleted = Person::where('city', 'like', '%тов')->delete();  
return view('deleteByCityEnding', compact('recordsDeleted'));  
}  
  
public function recordsExceptId4()  
{  
$recordsExceptId4 = Person::where('id', '!=', 4)->get();  
return view('recordsExceptId4', compact('recordsExceptId4'));  
}  
}  
```

Билет №11.

### Шаг 1: Установка Laravel

Если у вас еще нет установленного Laravel, выполните команду для создания нового проекта:

Bash

composer create-project --prefer-dist laravel/laravel my-laravel-project

Перейдите в каталог проекта:

Bash

cd my-laravel-project

### Шаг 2: Создание маршрута и контроллера

Создайте маршрут в файле web.php (расположен в каталоге routes), который будет обрабатывать запросы по адресу /list:

PHP

// routes/web.php

use App\Http\Controllers\ListController;

Route::get('/list', [ListController::class, 'index']);

Теперь создайте контроллер для обработки этого маршрута:

Bash

php artisan make:controller ListController

### Шаг 3: Редактирование контроллера

Откройте файл ListController.php (расположен в каталоге app/Http/Controllers) и отредактируйте его следующим образом:

PHP

// app/Http/Controllers/ListController.php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class ListController extends Controller

{

public function index()

{

// Заглушка для записей. Замените этот массив на ваши реальные данные.

$posts = [

'Запись 1',

'Запись 2',

'Запись 3',

'Запись 4',

'Запись 5',

];

return view('list.index', compact('posts'));

}

}

### Шаг 4: Создание представления

Создайте представление index.blade.php в каталоге resources/views/list:

Откройте файл index.blade.php и добавьте следующий код:

html

<!-- resources/views/list/index.blade.php -->

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Список записей</title>

<!-- Подключение Bootstrap стилей -->

<link href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css" rel="stylesheet">

</head>

<body>

<div class="container mt-5">

<h1>Список записей</h1>

@foreach($posts as $key => $post)

<div class="post mb-3" style="background-color: {{ $key % 2 === 0 ? 'white' : 'green' }}">

{{ $post }}

</div>

@endforeach

</div>

<!-- Подключение Bootstrap скриптов (необходимо для некоторых функциональностей) -->

<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"></script>

</body>

</html>

Билет №12

### Шаг 1: Установка Laravel

Установите Laravel, если у вас его еще нет, используя Composer:

Bash

composer create-project --prefer-dist laravel/laravel название\_проекта

### Шаг 2: Создание миграции и модели

Отредактируйте миграцию в файле, который будет находиться в database/migrations, добавив необходимые поля (login, password, email):

PHP

// в файле database/migrations/xxxx\_xx\_xx\_create\_users\_table.php

public function up()

{

Schema::create('users', function (Blueprint $table) {

$table->id();

$table->string('login');

$table->string('email')->unique();

$table->string('password');

$table->timestamps();

});

}

Выполните миграцию:

Bash

php artisan migrate

### Шаг 3: Создание контроллера и представления

Создайте контроллер:

Bash

php artisan make:controller UserController

Отредактируйте контроллер в файле app/Http/Controllers/UserController.php:

PHP

// UserController.php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use App\Models\User;

use Illuminate\Support\Facades\Hash;

use Illuminate\Validation\Rule;

class UserController extends Controller

{

public function showRegistrationForm()

{

return view('registration');

}

public function register(Request $request)

{

$request->validate([

'login' => 'required',

'email' => 'required|email|unique:users,email',

'password' => 'required|between:5,10',

]);

// Хеширование пароля перед сохранением

$password = Hash::make($request->input('password'));

$user = new User();

$user->login = $request->input('login');

$user->email = $request->input('email');

$user->password = Hash::make($request->input('password'));

// Сохраняем экземпляр модели

$user->save();

return redirect()->back()->with('success', 'Регистрация прошла успешно!');

}

}

### Шаг 4: Создание маршрутов

Добавьте маршруты в файл routes/web.php:

PHP

// routes/web.php

use App\Http\Controllers\UserController;

Route::get('/registration', [UserController::class, 'showRegistrationForm']);

Route::post('/registration', [UserController::class, 'register']);

### Шаг 5: Создание представления

Создайте файл представления resources/views/registration.blade.php:

blade

<!-- resources/views/registration.blade.php -->

@extends('layouts.app')

@section('content')

<div class="container">

@if(session('success'))

<div class="alert alert-success">

{{ session('success') }}

</div>

@endif

<form method="POST" action="/registration">

@csrf

<label for="login">Login:</label>

<input type="text" name="login" required>

@error('login')

<div class="alert alert-danger">{{ $message }}</div>

@enderror

<label for="email">Email:</label>

<input type="email" name="email" required>

@error('email')

<div class="alert alert-danger">{{ $message }}</div>

@enderror

<label for="password">Password:</label>

<input type="password" name="password" required>

@error('password')

<div class="alert alert-danger">{{ $message }}</div>

@enderror

<button type="submit">Register</button>

</form>

</div>

@endsection

### Шаг 6: Обновление макета (layout)

Обновите файл макета resources/views/layouts/app.blade.php:

blade

<!-- resources/views/layouts/app.blade.php -->

<!DOCTYPE html>

<html>

<head>

<title>Laravel Bootstrap Registration</title>

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css">

</head>

<body>

<div class="container mt-5">

@yield('content')

</div>

</body>

</html>