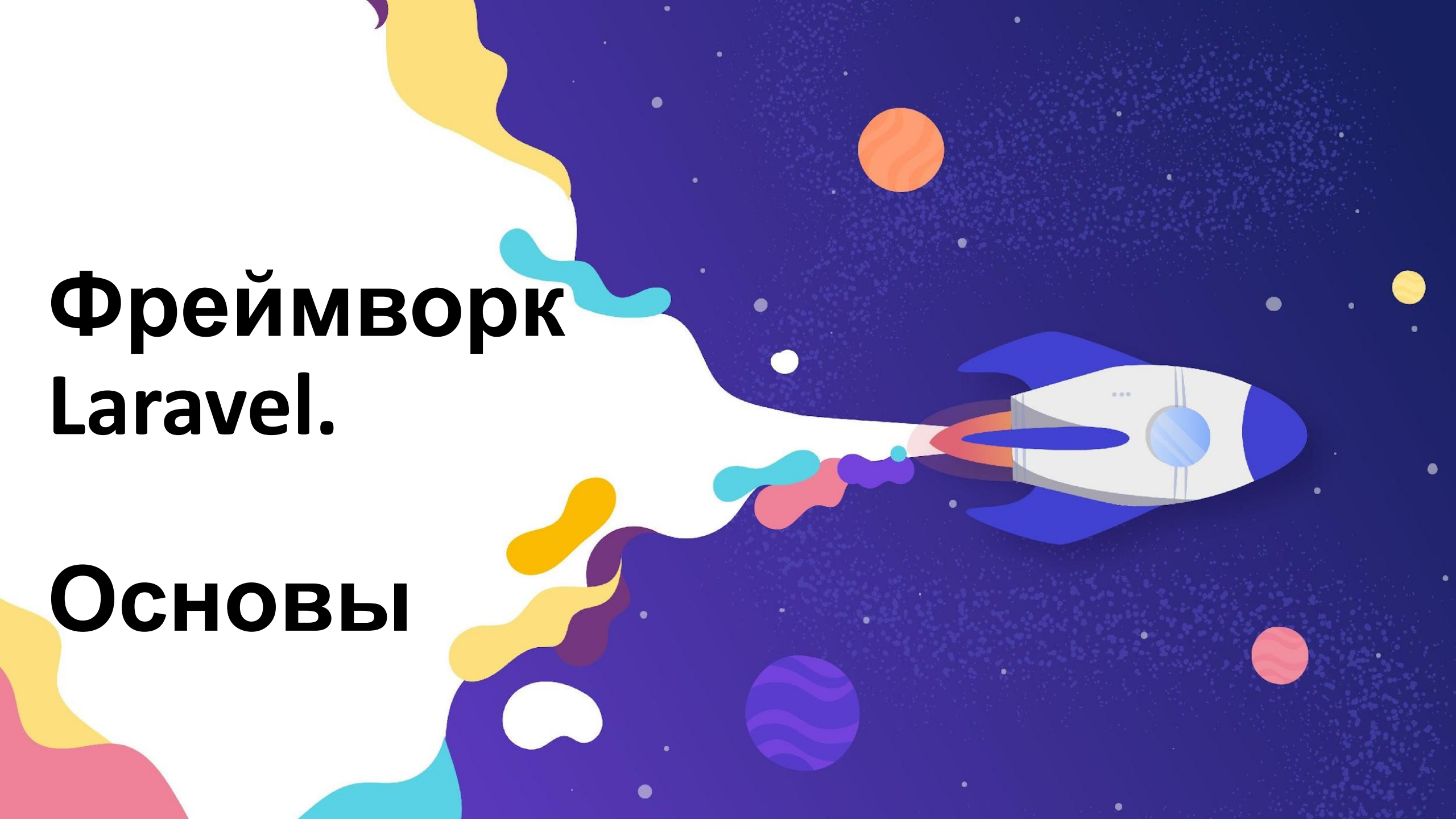
Фреймворк Laravel.

Основы



Фреймворк. Понятие, применение

Фреймворк - это инструмент, который облегчает и ускоряет процесс создания программного обеспечения, предоставляя готовые компоненты и функциональность, которую можно использовать в своих проектах.

Фреймворк включает в себя набор библиотек, инструментов и стандартов, которые позволяют разработчикам сосредоточиться на решении конкретных задач вместо того, чтобы писать все с нуля. Фреймворк определяет архитектуру приложения и предоставляет структуру для организации кода, что позволяет создавать приложения быстрее и более эффективно.

Кроме того, фреймворки часто включают в себя множество функций, таких как маршрутизация, обработка запросов и ответов, работа с базами данных, аутентификация, авторизация и т.д. Это позволяет разработчикам использовать готовые решения в своих проектах, что сокращает время разработки и снижает количество ошибок.

Некоторые популярные фреймворки веб-программирования, такие как Laravel, Symfony, Django и Ruby on Rails, предоставляют разработчикам готовые инструменты для создания веб-приложений, включая шаблонизацию, маршрутизацию, обработку форм, работу с базами данных и т.д.

Фреймворк. MVC

Архитектура MVC (Model-View-Controller) - это шаблон проектирования, который разделяет приложение на три компонента: модель, представление и контроллер. Каждый компонент выполняет свои функции и взаимодействует с другими компонентами через определенные интерфейсы.

В Laravel, которая является одним из самых популярных фреймворков PHP, архитектура MVC используется по умолчанию.

В целом, использование архитектуры MVC в Laravel позволяет лучше структурировать код приложения, повышает его модульность и облегчает его тестирование. Кроме того, разделение приложения на три компонента позволяет легче вносить изменения в каждый компонент отдельно, что упрощает поддержку и развитие приложения.

Фреймворк. MVC

Архитектура MVC (Model-View-Controller) - это шаблон проектирования, который разделяет приложение на три компонента: модель, представление и контроллер. Каждый компонент выполняет свои функции и взаимодействует с другими компонентами через определенные интерфейсы.

В Laravel, которая является одним из самых популярных фреймворков PHP, архитектура MVC используется по умолчанию.

В целом, использование архитектуры MVC в Laravel позволяет лучше структурировать код приложения, повышает его модульность и облегчает его тестирование. Кроме того, разделение приложения на три компонента позволяет легче вносить изменения в каждый компонент отдельно, что упрощает поддержку и развитие приложения.

Фреймворк. MVC

Модель (Model) - это компонент, отвечающий за работу с данными приложения. Он содержит логику работы с базой данных и определяет структуру данных, которые используются в приложении. В Laravel модели обычно размещаются в директории app/Models.

Представление (View) - это компонент, отвечающий за отображение данных приложения. Он определяет, как данные будут представлены пользователю, и содержит шаблоны и разметку. В Laravel представления обычно размещаются в директории resources/views.

Контроллер (Controller) - это компонент, отвечающий за обработку запросов пользователя и взаимодействие между моделью и представлением. Он содержит методы для получения данных из модели и передачи их в представление, а также для обработки пользовательских действий. В Laravel контроллеры обычно размещаются в директории app/Http/Controllers.

Фреймворк. MVC

В рамках архитектуры MVC в Laravel пользователь взаимодействует с приложением через маршруты, которые определяются в файле routes/web.php. Маршруты указываются в виде URL-адресов и связываются с определенными контроллерами и методами.

Пример:

Если есть маршрут /users, который вызывает метод index() в контроллере UserController, то пользователь может получить список всех пользователей, вызвав данный маршрут в браузере. Контроллер UserController получит список пользователей из модели User, передаст его в представление users/index.blade.php, которое отобразит список пользователей на странице.

Фреймворк Laravel. Структура

Рассмотрим основные директории и файлы, которые содержатся в структуре Laravel:

app - директория содержит все приложение PHP-кода, включая контроллеры, модели, сервис-провайдеры, и другие классы. Внутри этой директории находятся директории Http (контроллеры, маршруты), Providers (сервис-провайдеры), Models (модели) и директории с исходным кодом для вашего приложения.

bootstrap - содержит файлы для инициализации приложения и его настройки.

config - содержит все файлы конфигурации приложения, такие как настройки базы данных, логирования, кеша, и другие.

Фреймворк Laravel. Структура

database - содержит все файлы и директории, связанные с базой данных, такие как миграции и сиды.

public - содержит весь доступный публичный контент приложения, такой как изображения, JavaScript и CSS файлы, а также файл index.php, который является точкой входа в приложение.

resources - содержит ресурсы приложения, такие как шаблоны Blade, переводы, JavaScript и Sass файлы.

routes - содержит файлы маршрутизации, которые связывают URL-адреса с методами контроллера, определяя то, что должно быть выполнено, когда пользователь делает запрос.

Фреймворк Laravel. Структура

storage - содержит временные файлы, кеш, логи и другие файлы, которые используются приложением.

tests - содержит файлы тестирования приложения.

vendor - содержит зависимости PHP, установленные с помощью Composer.

.env - это файл конфигурации, содержащий настройки окружения, такие как ключ приложения, настройки базы данных и многое другое.

.gitignore - это файл, который содержит список файлов и директорий, которые не должны быть отслеживаемыми системой контроля версий Git.

Фреймворк Laravel. Структура app.

Директория app является одной из самых важных директорий в Laravel. В ней содержится основной код приложения и множество классов. Рассмотрим подробнее структуру директории app:

Console - содержит все команды Artisan, которые используются для выполнения задач из командной строки.

Exceptions - содержит классы исключений, которые могут быть брошены в вашем приложении.

Providers - содержит классы сервис-провайдеров, которые используются для регистрации сервисов в контейнере зависимостей приложения.

Фреймворк Laravel. Структура app.

Http - содержит контроллеры, маршруты, middleware, формы запросов и другие классы, связанные с HTTP-запросами.

- Controllers - содержит классы контроллеров, которые обрабатывают HTTP-запросы и возвращают HTTP-ответы.
- Middleware - содержит классы middleware, которые могут выполняться перед или после обработки HTTP-запроса.
- Requests - содержит классы форм запросов, которые используются для валидации входных данных, полученных от пользователя.

Фреймворк Laravel. Структура app.

Models - содержит классы моделей, которые используются для доступа к данным в базе данных.

Rules - содержит пользовательские правила валидации.

Events - содержит классы событий, которые используются для уведомления приложения о происходящих событиях.

Listeners - содержит классы слушателей событий, которые обрабатывают события, когда они происходят.

Jobs - содержит классы заданий, которые используются для выполнения длительных задач в фоне.

Фреймворк Laravel. Структура app.

Notifications - содержит классы уведомлений, которые используются для отправки уведомлений пользователям.

Policies - содержит классы политик, которые используются для определения прав доступа пользователей к определенным ресурсам.

Providers - содержит классы сервис-провайдеров, которые регистрируют сервисы в контейнере зависимостей приложения.

Repositories - содержит классы для работы с сущностями базы данных.

Services - содержит классы, которые содержат бизнес-логику приложения.

Фреймворк Laravel. DDD

DDD (Domain-Driven Design) - это методология проектирования программного обеспечения, которая ставит доменную модель в центр разработки. Она была разработана Эриком Эвансом в 2003 году и стала популярной в сообществе разработчиков благодаря своей эффективности в решении сложных задач и повышению качества разработки программного обеспечения.

Центральным элементом DDD является доменная модель - формальное описание основных концепций, правил и процессов, присущих определенной предметной области. Она представляет собой абстракцию реального мира, которую можно использовать для создания программного обеспечения.

В целом, DDD позволяет улучшить процесс разработки программного обеспечения, уменьшить количество ошибок и повысить качество конечного продукта.

Фреймворк Laravel. DDD

DDD обладает рядом особенностей, включая:

- Акцент на бизнес-логике. DDD сосредотачивается на проектировании программного обеспечения вокруг бизнес-процессов и бизнес-правил, что делает его особенно эффективным для сложных задач.
- Продвижение единой модели. DDD ставит задачу создания общей модели для всей команды разработчиков, что помогает повысить эффективность командной работы.
- Использование языка, понятного бизнесу. DDD рекомендует использовать термины и понятия, которые понятны бизнесу, что улучшает коммуникацию между разработчиками и бизнес-аналитиками.
- Разделение на слои. DDD разделяет программное обеспечение на различные слои, каждый из которых отвечает за конкретную функцию в приложении.

```
app/
├── Application/
|   ├── Commands/
|   ├── Exceptions/
|   ├── Services/
|   └── Transformers/
├── Domain/
|   ├── Entities/
|   ├── Events/
|   ├── Exceptions/
|   ├── Jobs/
|   ├── Repositories/
|   ├── Services/
|   └── ValueObjects/
├── Infrastructure/
|   ├── Persistence/
|   |   ├── Migrations/
|   |   └── Repositories/
|   └── Services/
├── Http/
|   ├── Controllers/
|   ├── Middleware/
|   └── Requests/
├── Providers/
├── Console/
|   └── Commands/
└── Resources/
    ├── lang/
    ├── views/
    └── assets/
```

app/Application - содержит классы приложения, которые содержат бизнес-логику и обеспечивают взаимодействие между доменной моделью и инфраструктурными сервисами. В этой директории могут быть следующие поддиректории:

- Commands - содержит классы команд, которые используются в приложении для выполнения различных задач.
- Exceptions - содержит исключения, связанные с приложением.
- Services - содержит классы сервисов, которые используются в приложении для выполнения различных задач.
- Transformers - содержит классы преобразователей, которые используются для преобразования данных из формата доменной модели в формат, подходящий для представления в пользовательском интерфейсе.

```
app/
├── Application/
|   ├── Commands/
|   ├── Exceptions/
|   ├── Services/
|   └── Transformers/
├── Domain/
|   ├── Entities/
|   ├── Events/
|   ├── Exceptions/
|   ├── Jobs/
|   ├── Repositories/
|   ├── Services/
|   └── ValueObjects/
├── Infrastructure/
|   ├── Persistence/
|   |   ├── Migrations/
|   |   └── Repositories/
|   └── Services/
├── Http/
|   ├── Controllers/
|   ├── Middleware/
|   └── Requests/
├── Providers/
├── Console/
|   └── Commands/
└── Resources/
    ├── lang/
    ├── views/
    └── assets/
```

Папка "Domain" - это один из основных слоев приложения, построенного по принципам DDD. Она содержит все классы и интерфейсы, относящиеся к доменной модели и бизнес-логике приложения. Структура папки "Domain" может выглядеть следующим образом:

- Entities содержит классы, которые представляют основные сущности доменной модели, такие как пользователи, продукты, заказы и т.д. Каждая сущность может иметь свои свойства и методы, связанные с ее функциональностью.
- Repositories содержит интерфейсы и классы, которые предоставляют абстракцию доступа к данным. Классы репозиториев выполняют операции чтения и записи сущностей в хранилище данных, скрывая детали реализации от остальных слоев приложения.

- Persistence содержит классы и компоненты, связанные с хранением данных в базе данных. Классы Eloquent Repository реализуют интерфейсы из слоя Domain и используют модели Eloquent для выполнения операций с базой данных. Миграции содержат определения структуры таблиц в базе данных, а DatabaseSeeder используется для наполнения базы данных начальными данными.
- Services содержит классы, которые реализуют вспомогательную бизнес-логику, связанную с инфраструктурными компонентами приложения. Они могут использоваться для отправки электронной почты, выполнения платежей, взаимодействия с внешними API и т.д.
- Providers содержит классы, которые регистрируют сервис-провайдеры в приложении Laravel. Сервис-провайдеры используются для регистрации зависимостей, настройки конфигурации и регистрации маршрутов в приложении.

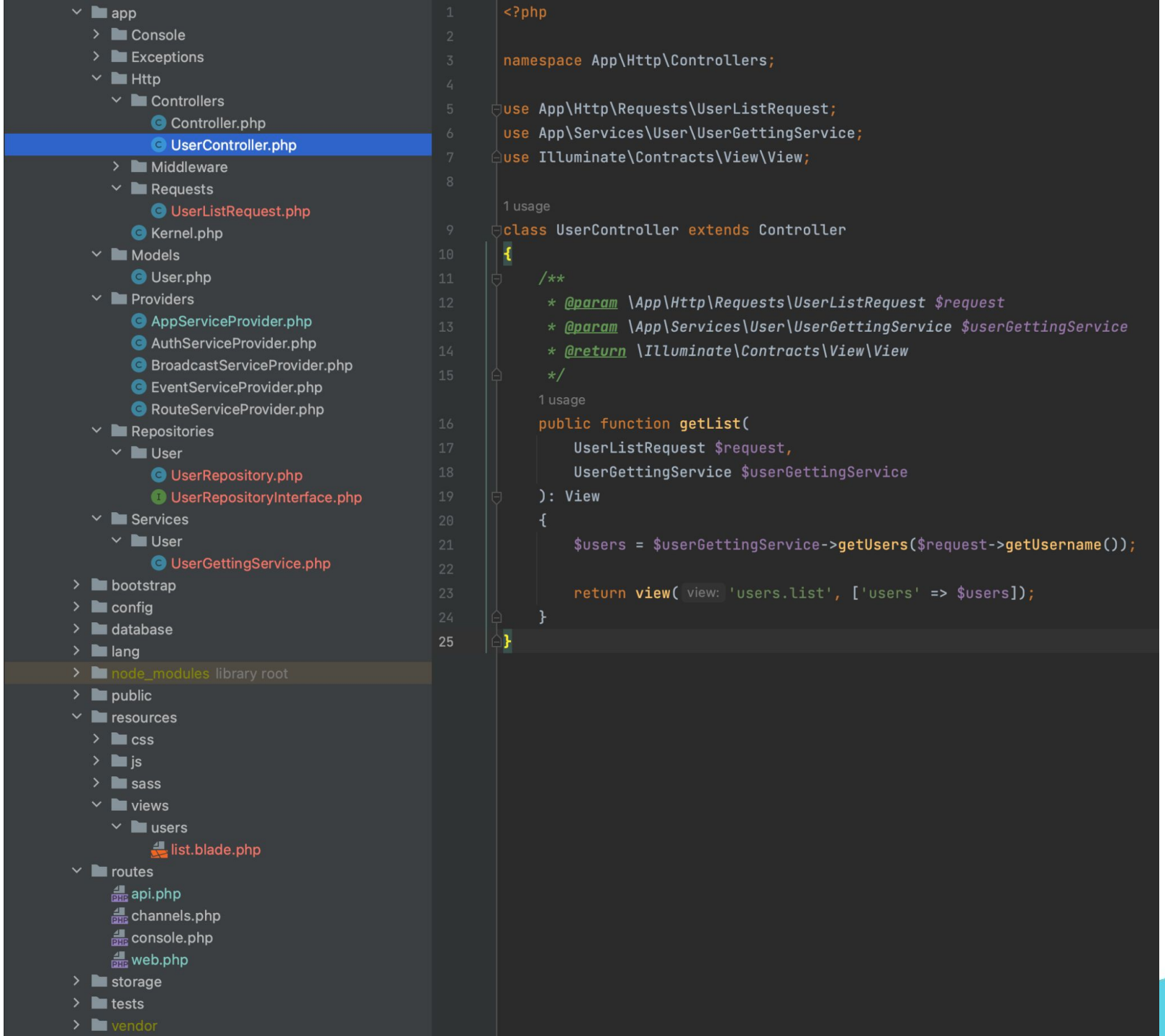
- Database содержит миграции, сиды и фабрики, а также сервис-провайдер, который регистрирует все зависимости, связанные с базами данных. Миграции используются для создания и обновления таблиц базы данных, сиды - для заполнения их начальными данными, а фабрики - для генерации тестовых данных. Сервис-провайдер регистрирует соединения с базами данных, репозитории и фабрики.
- Cache содержит сервис-провайдер, который регистрирует все зависимости, связанные с кэшированием. Это может быть любой кэш-драйвер, поддерживаемый Laravel, такой как Redis, Memcached и т.д.
- Logging содержит сервис-провайдер, который регистрирует все зависимости, связанные с логгированием. В Laravel используется мощный механизм логгирования, который позволяет записывать все события приложения в различные источники логов.

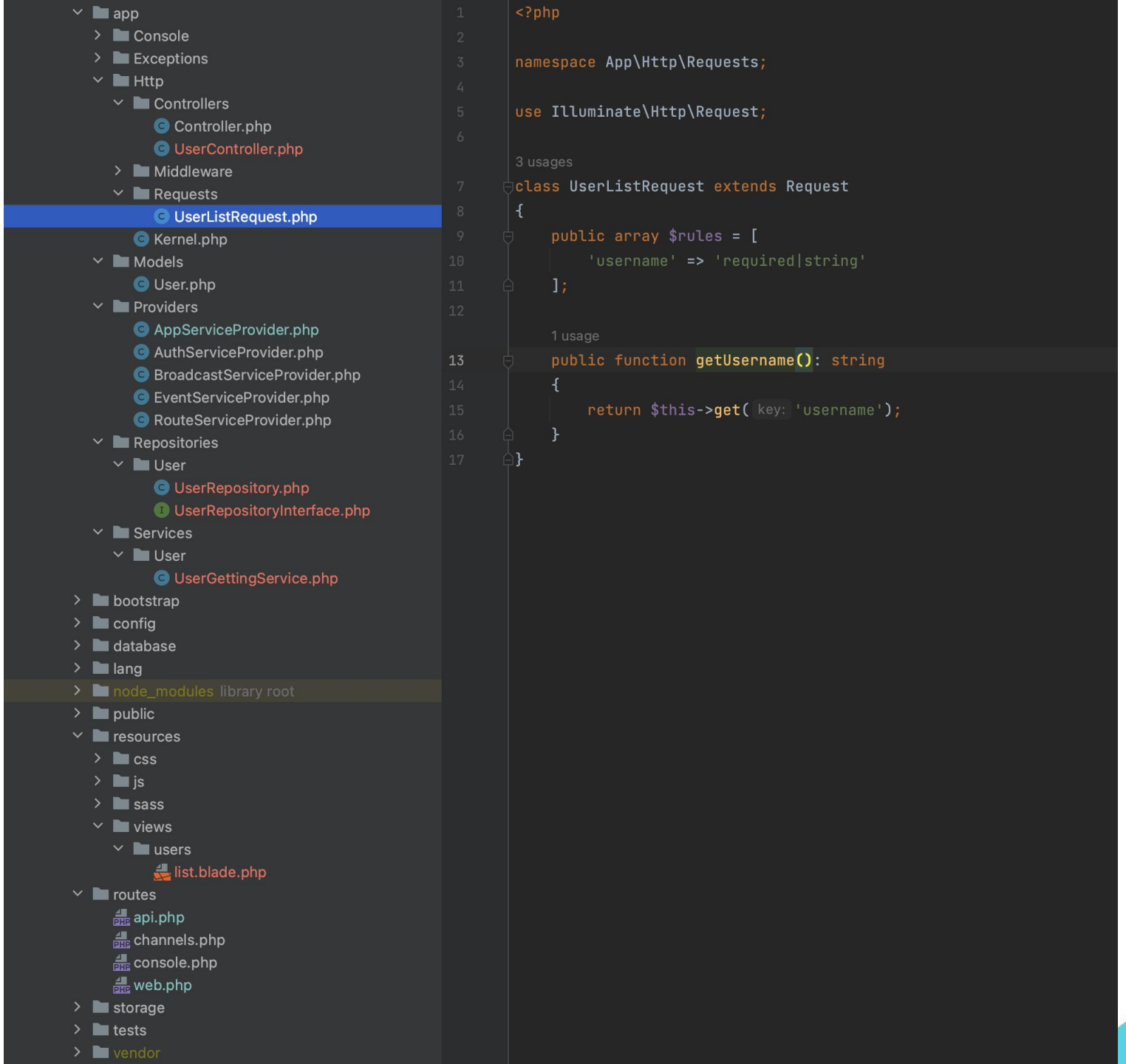
- Mail содержит сервис-провайдер, который регистрирует все зависимости, связанные с отправкой электронных писем. Laravel имеет встроенный мейлер, который позволяет отправлять письма через различные драйверы, такие как SMTP, Mailgun, Amazon SES и т.д.
- Queues содержит сервис-провайдер, который регистрирует все зависимости, связанные с очередями. Очереди используются для асинхронной обработки долгих операций, таких как отправка электронных писем, обработка изображений и т.д.

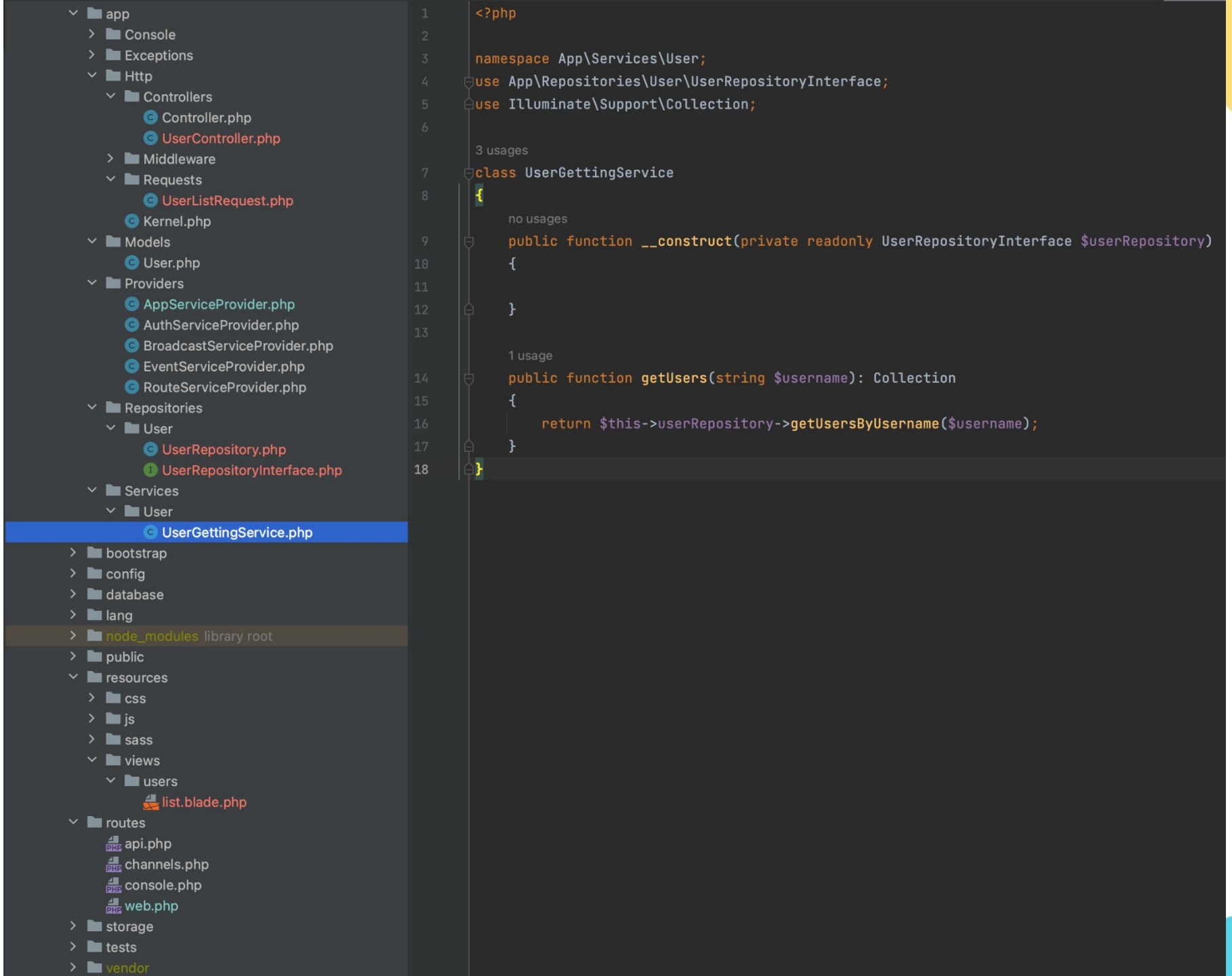
Фреймворк Laravel. Простой пример

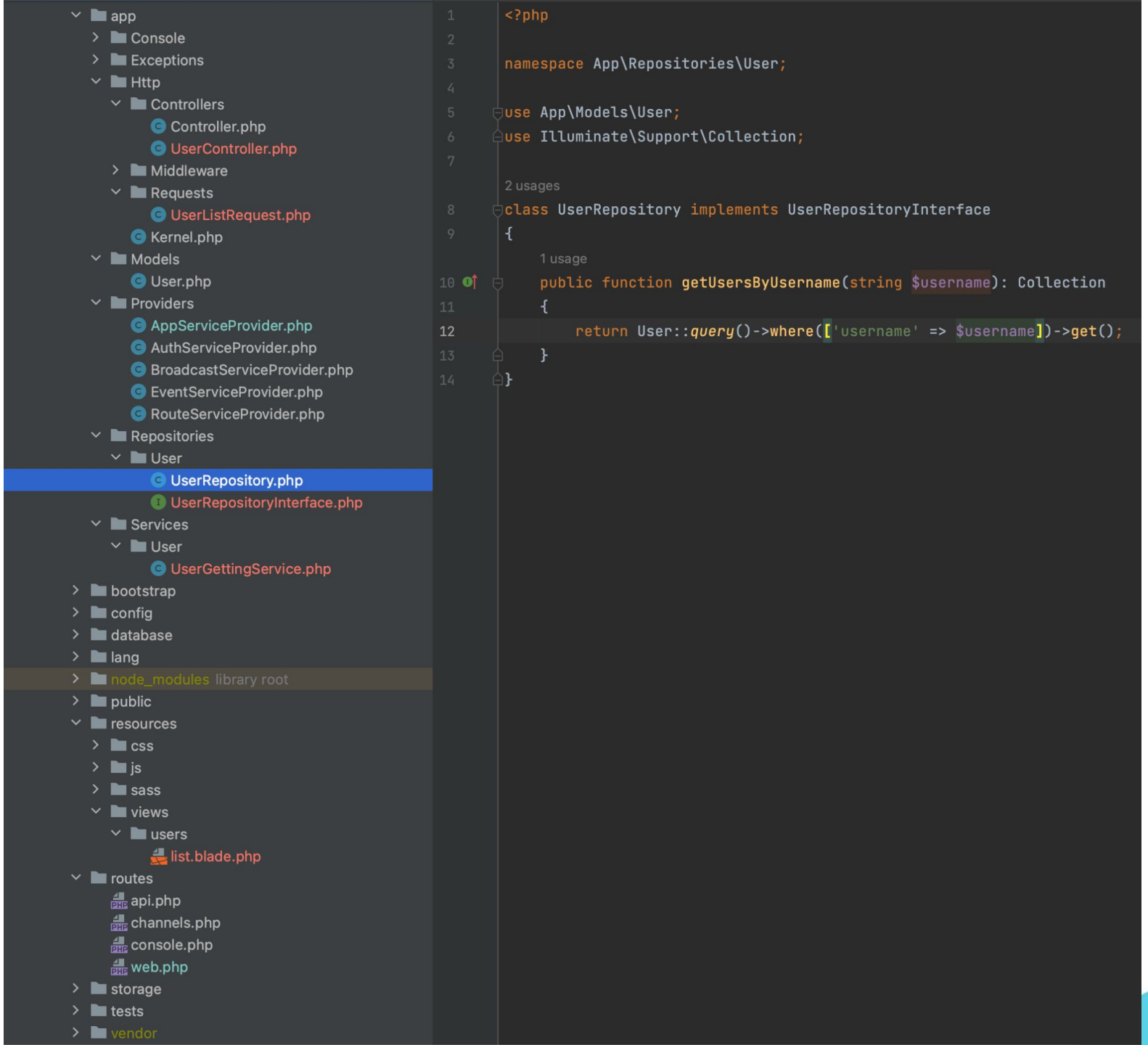
Задача: написать модуль, который выводит список пользователей на экран.
В GET запросе должен присутствовать параметр username, по которому будет осуществляться поиск.
Воспользоваться стандартным механизмом view фреймворка Laravel.

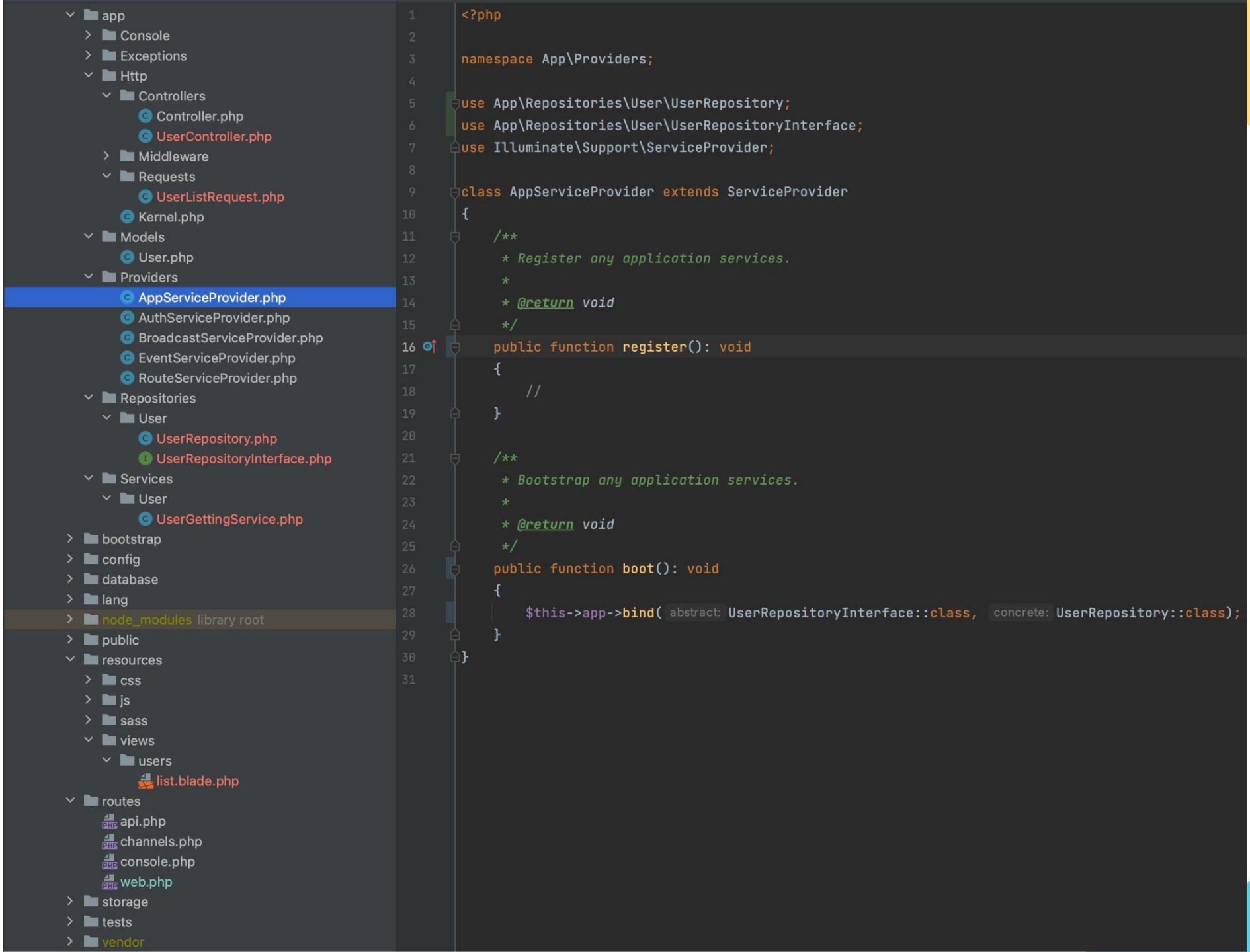
В первую очередь необходимо создать контроллер UserController, в котором задать метод getList. Далее создать Request, в котором указать правила валидации (например, что username обязателен). Создать UserGettingService, который в свою очередь вызовет UserRepository для того, чтобы получить данные из модели User. После этого создать view users/list.blade.php, в котором задать логику отображения. Далее - создать маршрут (файл routes/web.php), чтобы при вводе http://site.com/users происходил вызов UserController::getList().

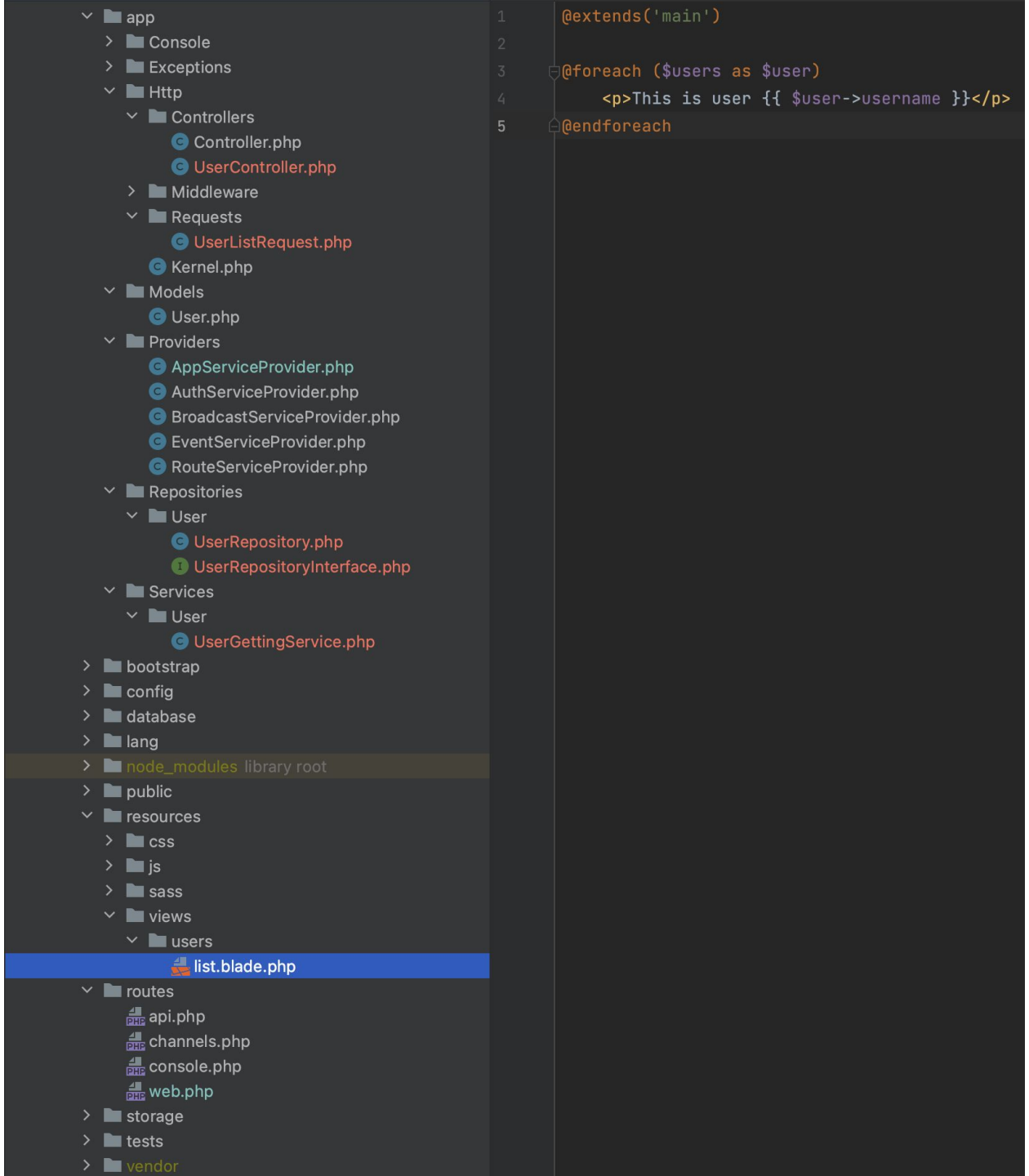

app
Console
Exceptions
Http
Controllers
Controller.php
UserController.php
Middleware
Requests
UserListRequest.php
Kernel.php
Models
User.php
Providers
AppServiceProvider.php
AuthServiceProvider.php
BroadcastServiceProvider.php
EventServiceProvider.php
RouteServiceProvider.php
Repositories
User
UserRepository.php
UserRepositoryInterface.php
Services
User
UserGettingService.php
bootstrap
config
database
lang
node_modules library root
public
resources
css
js
sass
views
users
list.blade.php
routes
api.php
channels.php
console.php
web.php
storage
tests
vendor
<?php
namespace App\Http\Controllers;
use App\Http\Requests\UserListRequest;
use App\Services\User\UserGettingService;
use Illuminate\Contracts\View\View;
1 usage
class UserController extends Controller
{
/**
* @param \App\Http\Requests\UserListRequest $request
* @param \App\Services\User\UserGettingService $userGettingService
* @return \Illuminate\Contracts\View\View
*/
1 usage
public function getList(
UserListRequest $request,
UserGettingService $userGettingService
): View
{
$users = $userGettingService->getUsers($request->getUsername());
return view(view: 'users.list', ['users' => $users]);
}
}

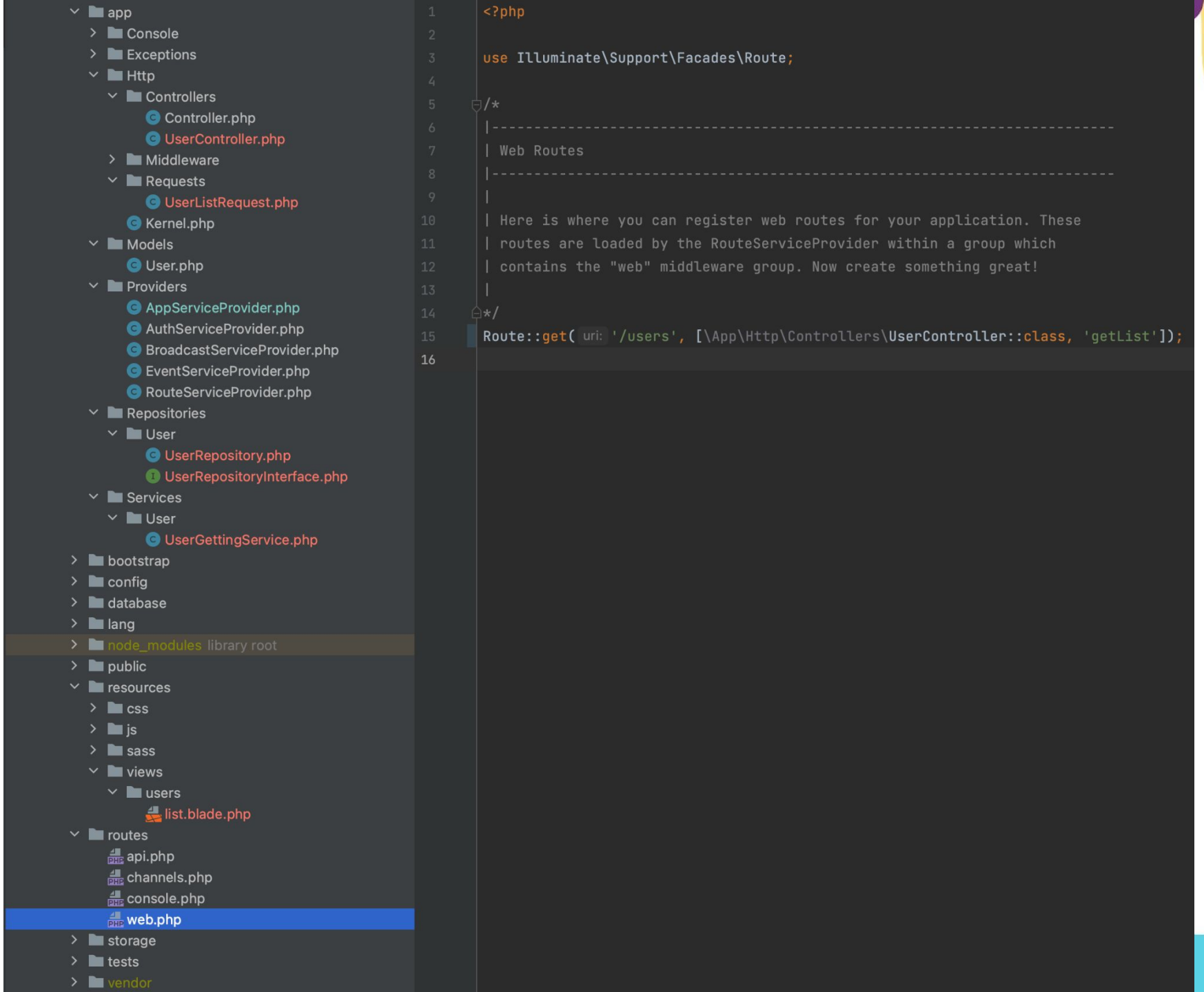

app
Console
Exceptions
Http
Controllers
Controller.php
UserController.php
Middleware
Requests
UserListRequest.php
Kernel.php
Models
User.php
Providers
AppServiceProvider.php
AuthServiceProvider.php
BroadcastServiceProvider.php
EventServiceProvider.php
RouteServiceProvider.php
Repositories
User
UserRepository.php
UserRepositoryInterface.php
Services
User
UserGettingService.php
bootstrap
config
database
lang
node_modules library root
public
resources
css
js
sass
views
users
list.blade.php
routes
api.php
channels.php
console.php
web.php
storage
tests
vendor
<?php
namespace App\Http\Requests;
use Illuminate\Http\Request;
3 usages
class UserListRequest extends Request
{
public array $rules = [
'username' => 'required|string'
];
1 usage
public function getUsername(): string
{
return $this->get(key: 'username');
}
}


app
Console
Exceptions
Http
Controllers
Controller.php
UserController.php
Middleware
Requests
UserListRequest.php
Kernel.php
Models
User.php
Providers
AppServiceProvider.php
AuthServiceProvider.php
BroadcastServiceProvider.php
EventServiceProvider.php
RouteServiceProvider.php
Repositories
User
UserRepository.php
UserRepositoryInterface.php
Services
User
UserGettingService.php
bootstrap
config
database
lang
node_modules library root
public
resources
css
js
sass
views
users
list.blade.php
routes
api.php
channels.php
console.php
web.php
storage
tests
vendor
<?php
namespace App\Services\User;
use App\Repositories\User\UserRepositoryInterface;
use Illuminate\Support\Collection;
3 usages
class UserGettingService
{
no usages
public function __construct(private readonly UserRepositoryInterface $userRepository)
{
}
1 usage
public function getUsers(string $username): Collection
{
return $this->userRepository->getUsersByUsername($username);
}
}


app
Console
Exceptions
Http
Controllers
Controller.php
UserController.php
Middleware
Requests
UserListRequest.php
Kernel.php
Models
User.php
Providers
AppServiceProvider.php
AuthServiceProvider.php
BroadcastServiceProvider.php
EventServiceProvider.php
RouteServiceProvider.php
Repositories
User
UserRepository.php
UserRepositoryInterface.php
Services
User
UserGettingService.php
bootstrap
config
database
lang
node_modules library root
public
resources
css
js
sass
views
users
list.blade.php
routes
api.php
channels.php
console.php
web.php
storage
tests
vendor
<?php
namespace App\Repositories\User;
use App\Models\User;
use Illuminate\Support\Collection;
2 usages
class UserRepository implements UserRepositoryInterface
{
1 usage
public function getUsersByUsername(string $username): Collection
{
return User::query()->where(['username' => $username])->get();
}
}


app
Console
Exceptions
Http
Controllers
Controller.php
UserController.php
Middleware
Requests
UserListRequest.php
Kernel.php
Models
User.php
Providers
AppServiceProvider.php
AuthServiceProvider.php
BroadcastServiceProvider.php
EventServiceProvider.php
RouteServiceProvider.php
Repositories
User
UserRepository.php
UserRepositoryInterface.php
Services
User
UserGettingService.php
bootstrap
config
database
lang
node_modules library root
public
resources
css
js
sass
views
users
list.blade.php
routes
api.php
channels.php
console.php
web.php
storage
tests
vendor
<?php
namespace App\Providers;
use App\Repositories\User\UserRepository;
use App\Repositories\User\UserRepositoryInterface;
use Illuminate\Support\ServiceProvider;
class AppServiceProvider extends ServiceProvider
{
/**
* Register any application services.
*
* @return void
*/
public function register(): void
{
//
}
/**
* Bootstrap any application services.
*
* @return void
*/
public function boot(): void
{
$this->app->bind(abstract: UserRepositoryInterface::class, concrete: UserRepository::class);
}
}


app
Console
Exceptions
Http
Controllers
Controller.php
UserController.php
Middleware
Requests
UserListRequest.php
Kernel.php
Models
User.php
Providers
AppServiceProvider.php
AuthServiceProvider.php
BroadcastServiceProvider.php
EventServiceProvider.php
RouteServiceProvider.php
Repositories
User
UserRepository.php
UserRepositoryInterface.php
Services
User
UserGettingService.php
bootstrap
config
database
lang
node_modules library root
public
resources
css
js
sass
views
users
list.blade.php
routes
api.php
channels.php
console.php
web.php
storage
tests
vendor
@extends('main')
@foreach ($users as $user)
This is user {{ $user->username }}
@endforeach


app
Console
Exceptions
Http
Controllers
Controller.php
UserController.php
Middleware
Requests
UserListRequest.php
Kernel.php
Models
User.php
Providers
AppServiceProvider.php
AuthServiceProvider.php
BroadcastServiceProvider.php
EventServiceProvider.php
RouteServiceProvider.php
Repositories
User
UserRepository.php
UserRepositoryInterface.php
Services
User
UserGettingService.php
bootstrap
config
database
lang
node_modules library root
public
resources
css
js
sass
views
users
list.blade.php
routes
api.php
channels.php
console.php
web.php
storage
tests
vendor
<?php
use Illuminate\Support\Facades\Route;
/*
|--
| Web Routes
|--
|
| Here is where you can register web routes for your application. These
| routes are loaded by the RouteServiceProvider within a group which
| contains the "web" middleware group. Now create something great!
|
*/
Route::get(uri: '/users', [\App\Http\Controllers\UserController::class, 'getList']);

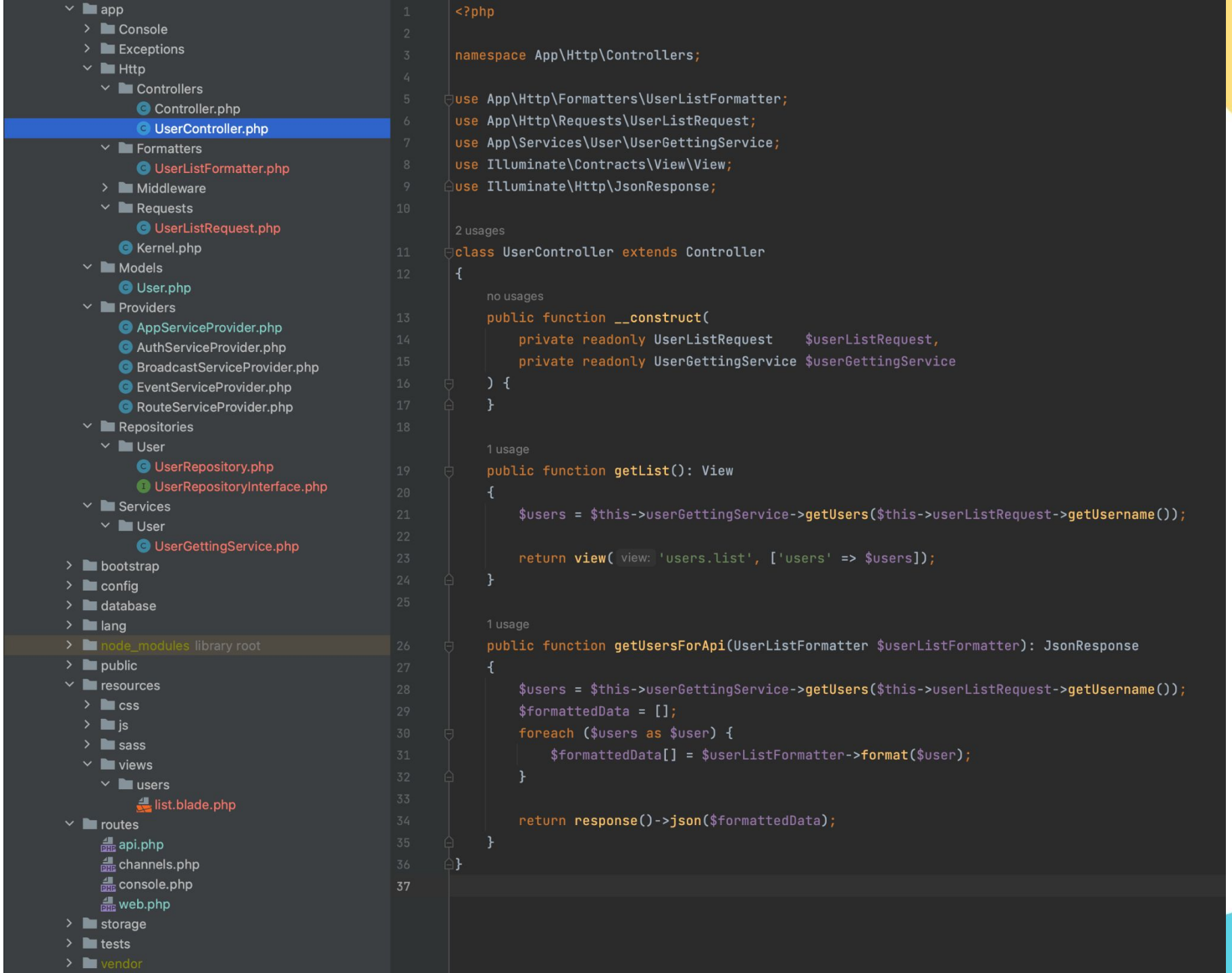
Фреймворк Laravel. Пример с API

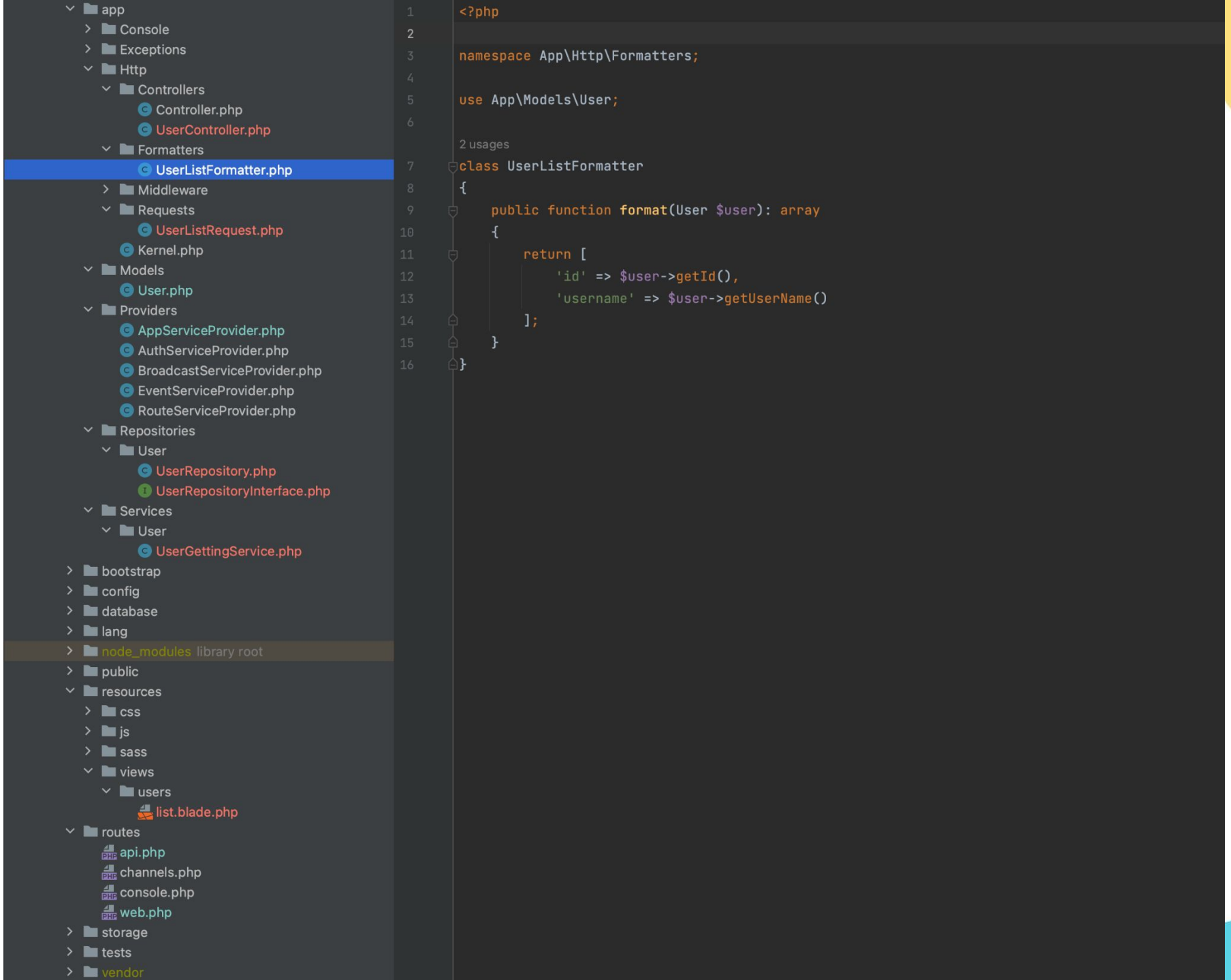
Задача: написать модуль, который возвращает список пользователей в формате JSON.
В GET запросе должен присутствовать параметр username, по которому будет осуществляться поиск.

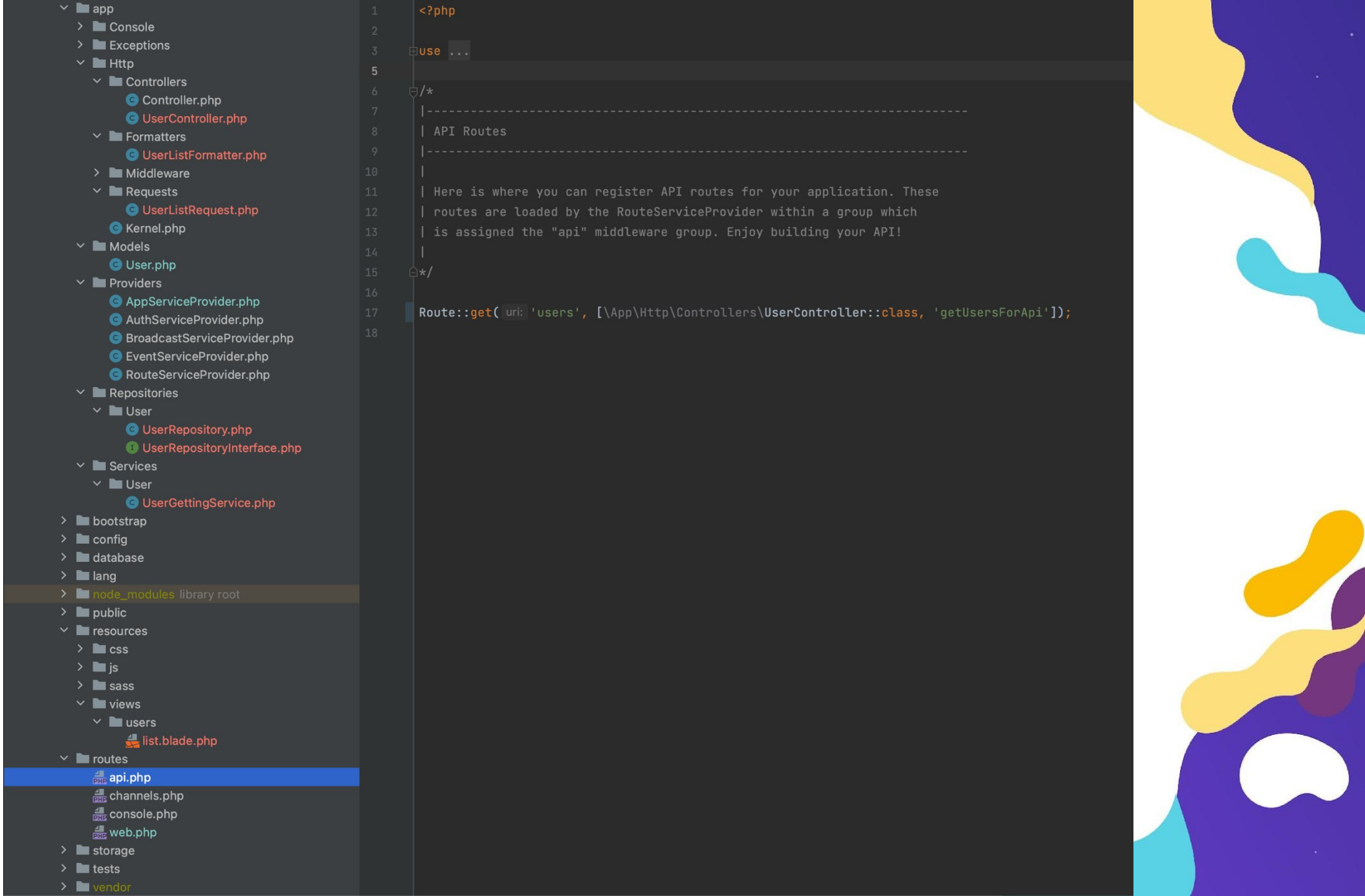
В данном примере можно воспользоваться уже имеющимся кодом. Создадим в UserController дополнительный метод getUsersForApi, который будет также вызывать метод у сервиса UserGettingService.

Однако, при работе по API необходимо заботиться о сохранении “контрактов” между вашим API и API источника запроса. Поэтому нам потребуется дополнительный элемент, который назовем UserListFormatter. Он будет отвечать за преобразование “сырых” данных в заранее определенный формат.

Т.к. маршрут принадлежит API, необходимо его создать в файле routes/api.php.
В итоге endpoint будет выглядеть так: http://site.com/api/users.


app
Console
Exceptions
Http
Controllers
Controller.php
UserController.php
Formatters
UserListFormatter.php
Middleware
Requests
UserListRequest.php
Kernel.php
Models
User.php
Providers
AppServiceProvider.php
AuthServiceProvider.php
BroadcastServiceProvider.php
EventServiceProvider.php
RouteServiceProvider.php
Repositories
User
UserRepository.php
UserRepositoryInterface.php
Services
User
UserGettingService.php
bootstrap
config
database
lang
node_modules library root
public
resources
css
js
sass
views
users
list.blade.php
routes
api.php
channels.php
console.php
web.php
storage
tests
vendor
<?php
namespace App\Http\Controllers;
use App\Http\Formatters\UserListFormatter;
use App\Http\Requests\UserListRequest;
use App\Services\User\UserGettingService;
use Illuminate\Contracts\View\View;
use Illuminate\Http\JsonResponse;
2 usages
class UserController extends Controller
{
no usages
public function __construct(
private readonly UserListRequest $userListRequest,
private readonly UserGettingService $userGettingService
) {
}
1 usage
public function getList(): View
{
$users = $this->userGettingService->getUsers($this->userListRequest->getUsername());
return view(view: 'users.list', ['users' => $users]);
}
1 usage
public function getUsersForApi(UserListFormatter $userListFormatter): JsonResponse
{
$users = $this->userGettingService->getUsers($this->userListRequest->getUsername());
$formattedData = [];
foreach ($users as $user) {
$formattedData[] = $userListFormatter->format($user);
}
return response()->json($formattedData);
}
}


app
Console
Exceptions
Http
Controllers
Controller.php
UserController.php
Formatters
UserListFormatter.php
Middleware
Requests
UserListRequest.php
Kernel.php
Models
User.php
Providers
AppServiceProvider.php
AuthServiceProvider.php
BroadcastServiceProvider.php
EventServiceProvider.php
RouteServiceProvider.php
Repositories
User
UserRepository.php
UserRepositoryInterface.php
Services
User
UserGettingService.php
bootstrap
config
database
lang
node_modules library root
public
resources
css
js
sass
views
users
list.blade.php
routes
api.php
channels.php
console.php
web.php
storage
tests
vendor
<?php
namespace App\Http\Formatters;
use App\Models\User;
2 usages
class UserListFormatter
{
public function format(User $user): array
{
return [
'id' => $user->getId(),
'username' => $user->getUserName()
];
}
}


app
Console
Exceptions
Http
Controllers
Controller.php
UserController.php
Formatters
UserListFormatter.php
Middleware
Requests
UserListRequest.php
Kernel.php
Models
User.php
Providers
AppServiceProvider.php
AuthServiceProvider.php
BroadcastServiceProvider.php
EventServiceProvider.php
RouteServiceProvider.php
Repositories
User
UserRepository.php
UserRepositoryInterface.php
Services
User
UserGettingService.php
bootstrap
config
database
lang
node_modules library root
public
resources
css
js
sass
views
users
list.blade.php
routes
api.php
channels.php
console.php
web.php
storage
tests
vendor
<?php
use ...
/*
|--
| API Routes
|--
|
| Here is where you can register API routes for your application. These
| routes are loaded by the RouteServiceProvider within a group which
| is assigned the "api" middleware group. Enjoy building your API!
|
*/
Route::get(uri: 'users', [\App\Http\Controllers\UserController::class, 'getUsersForApi']);

Фреймворк Laravel

Конец лекции