Implement the Repository Pattern with Eloquent and Dependency Injection

Objective

Learn how to:

1. **Decouple business logic from Eloquent** using the Repository Pattern.
2. **Use Dependency Injection (DI)** for cleaner, testable code.
3. **Swap data sources** (e.g., switch from MySQL to an API) without changing business logic.

---

1. Why Use the Repository Pattern?

✅ **Separation of concerns** (Business logic ≠ Database logic).
✅ **Easier testing** (Mock repositories instead of hitting the DB).
✅ **Flexibility** (Switch between Eloquent, APIs, or other data sources).

---

2. Project Structure

```
app/
├── Repositories/
│   ├── Interfaces/UserRepositoryInterface.php
│   ├── Eloquent/UserRepository.php
├── Providers/RepositoryServiceProvider.php
```

---

3. Step-by-Step Implementation

Step 1: Define the Repository Interface

PROF

```php
// app/Repositories/Interfaces/UserRepositoryInterface.php
namespace App\Repositories\Interfaces;

interface UserRepositoryInterface
{
    public function all();
    public function find(int $id);
    public function create(array $data);
    public function update(int $id, array $data);
    public function delete(int $id);
}
```

Step 2: Implement the Eloquent Repository

```php
// app/Repositories/Eloquent/UserRepository.php
namespace App\Repositories\Eloquent;

use App\Models\User;
use App\Repositories\Interfaces\UserRepositoryInterface;

class UserRepository implements UserRepositoryInterface
{
    protected $model;

    public function __construct(User $model)
    {
        $this->model = $model;
    }

    public function all()
    {
        return $this->model->all();
    }

    public function find(int $id)
    {
        return $this->model->findOrFail($id);
    }

    public function create(array $data)
    {
        return $this->model->create($data);
    }

    public function update(int $id, array $data)
    {
        $user = $this->model->findOrFail($id);
        $user->update($data);
        return $user;
    }

    public function delete(int $id)
    {
        return $this->model->destroy($id);
    }
}
```


Step 3: Bind Interface to Implementation (Service Provider)

```php
// app/Providers/RepositoryServiceProvider.php
namespace App\Providers;

use Illuminate\Support\ServiceProvider;
```

```php
use App\Repositories\Interfaces\UserRepositoryInterface;
use App\Repositories\Eloquent\UserRepository;

class RepositoryServiceProvider extends ServiceProvider
{
    public function register()
    {
        $this->app->bind(
            UserRepositoryInterface::class,
            UserRepository::class
        );
    }

    public function boot()
    {
        // Optional: Bind other repositories here
    }
}
```

Register the provider in `config/app.php`:

```php
'providers' => [
    // ...
    App\Providers\RepositoryServiceProvider::class,
],
```

---

4. Using the Repository in Controllers (Dependency Injection)

```php
// app/Http/Controllers/UserController.php
namespace App\Http\Controllers;

use App\Repositories\Interfaces\UserRepositoryInterface;

class UserController extends Controller
{
    protected $userRepository;

    // Inject the repository interface
    public function __construct(UserRepositoryInterface $userRepository)
    {
        $this->userRepository = $userRepository;
    }

    public function index()
    {
        $users = $this->userRepository->all();
        return view('users.index', compact('users'));
```

PROF

```
    }

    public function store(Request $request)
    {
        $data = $request->validate([...]);
        $user = $this->userRepository->create($data);
        return redirect()->route('users.index');
    }
}
```

---

5. Testing with Mock Repositories

```
// tests/Feature/UserControllerTest.php
use App\Repositories\Interfaces\UserRepositoryInterface;
use Mockery;

test('index returns all users', function () {
    $mockRepo = Mockery::mock(UserRepositoryInterface::class);
    $mockRepo->shouldReceive('all')->once()->andReturn(collect([]));

    $this->app->instance(UserRepositoryInterface::class, $mockRepo);

    $response = $this->get('/users');
    $response->assertStatus(200);
});
```

---

6. Advanced: Extending for Multiple Models

1. **Create a `BaseRepository`** (optional for DRY code):

PROF

```
// app/Repositories/Eloquent/BaseRepository.php
namespace App\Repositories\Eloquent;

use Illuminate\Database\Eloquent\Model;

class BaseRepository
{
    protected $model;

    public function __construct(Model $model)
    {
        $this->model = $model;
    }

    // Common methods (all, find, create, update, delete)
}
```

2. **Extend in `UserRepository`:**

```php
class UserRepository extends BaseRepository implements
UserRepositoryInterface
{
    public function __construct(User $model)
    {
        parent::__construct($model);
    }

    // Add custom methods (e.g., findByEmail)
}
```

Key Takeaways

✅ **Decouples business logic** from Eloquent.
✅ **Easy to test** (swap real DB for mock repositories).
✅ **Flexible** (switch to API/Redis storage later).