Random User API - Laravel Implementation

Overview

This project implements a Laravel application that:

1. Fetches random user data from the randomuser.me API on a scheduled basis
2. Stores the user data in a normalized database structure
3. Provides a RESTful API for querying the stored user data with filtering capabilities

Table of Contents

Project Structure

The project follows Laravel's standard MVC architecture:

- **Models**: Define the database structure and relationships
- **Controllers**: Handle API requests and responses
- **Commands**: Manage scheduled tasks for data fetching
- **Migrations**: Define the database schema

Key Files

- `app/Console/Commands/FetchRandomUsers.php` - Command for fetching random users
- `app/Console/Kernel.php` - Scheduler configuration
- `app/Models/User.php` - User model
- `app/Models/UserDetail.php` - User details model
- `app/Models/Location.php` - Location model
- `app/Http/Controllers/Api/UserController.php` - API controller
- `routes/api.php` - API route definitions
- `database/migrations/*` - Database migration files

Database Design

The database follows a normalized structure with three main tables:

Users Table

- `id` (Primary Key)
- `first_name`
- `last_name`
- `email` (Unique)
- `username` (Unique)
- `timestamps` (created_at, updated_at)

User Details Table

- `id` (Primary Key)
- `user_id` (Foreign Key)
- `gender`
- `date_of_birth`
- `phone`
- `cell`
- `picture_large`
- `picture_medium`
- `picture_thumbnail`
- `timestamps` (created_at, updated_at)

Locations Table

- `id` (Primary Key)
- `user_id` (Foreign Key)
- `street_number`
- `street_name`
- `city`
- `state`
- `country`
- `postcode`
- `latitude`

- `longitude`
- `timestamps` (created_at, updated_at)

Implementation Details

Scheduled Task

The scheduled task is implemented as a Laravel command that:

1. Runs every 5 minutes as configured in the `Kernel.php` file
2. Makes 5 separate API calls to the randomuser.me API
3. Processes and stores the retrieved user data in the database

The command supports custom parameters:

- `--count`: Number of users to fetch (default: 5)

Data Storage

User data is stored using Laravel's Eloquent ORM:

1. Each API response is processed to extract relevant data
2. Database transactions ensure data integrity
3. Error handling prevents partial data insertion

API Endpoint

The API endpoint (`/api/users`) supports:

1. Filtering by gender, city, and country
2. Limiting the number of records returned
3. Selecting which fields to include in the response

Setup Instructions

Follow these steps to set up the project:

1. Clone the Repository

bash Copy

```bash
git clone https://github.com/yourusername/random-user-api.git
cd random-user-api
```

2. Install Dependencies

bash Copy

```
composer install
```

3. Configure Environment

bash Copy

```
cp .env.example .env
php artisan key:generate
```

Edit `.env` file to configure your database connection:

Copy

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=random_user_api
DB_USERNAME=root
DB_PASSWORD=
```

4. Run Migrations

bash Copy

```
php artisan migrate
```

5. Test the Command

bash Copy

```
php artisan users:fetch
```

6. Configure Scheduler

To enable the scheduler, add this entry to your crontab:

Copy

```
* * * * * cd /path-to-your-project && php artisan schedule:run >> /dev/null 2>&1
```

7. Run the Application

bash Copy

```
php artisan serve
```

API Documentation

Endpoint: GET /api/users

Retrieves user data with filtering capabilities.

Query Parameters

Parameter	Type	Description	Example
gender	string	Filter users by gender (male/female)	?gender=female
city	string	Filter users by city	?city=London
country	string	Filter users by country	?country=France
limit	integer	Number of records to return (default: 10)	?limit=20
fields	string	Comma-separated list of fields to include	?fields=name,email

Response Format

json Copy

```
{
  "status": "success",
  "total": 25,
  "per_page": 10,
  "current_page": 1,
  "last_page": 3,
  "data": [
    {
      "id": 1,
      "name": "John Doe",
      "email": "john.doe@example.com",
      "username": "johndoe",
      "gender": "male",
      "city": "New York",
      "country": "United States"
    },
    // More user records...
  ]
}
```

Code Quality Measures

The project follows Laravel best practices and coding standards:

1. PSR Compliance

- PSR-4 autoloading standard
- PSR-1 and PSR-12 coding standards

2. Documentation

- PHPDoc comments on all methods and classes
- Clear and concise documentation of code functionality

3. Error Handling

- Try/catch blocks for robust error handling
- Database transactions to ensure data integrity
- Comprehensive logging of errors

4. Input Validation

- Request validation for API parameters
- Type-hinting for method parameters and return types

5. Database Optimization

- Proper indexing on filtered columns
- Eloquent relationships for efficient querying
- Database transactions for atomic operations

Future Enhancements

Potential improvements for the project:

1. **API Authentication**: Implement Laravel Sanctum for API authentication
2. **Rate Limiting**: Add rate limiting to prevent API abuse
3. **Caching**: Implement response caching for improved performance
4. **Tests**: Add unit and feature tests for reliability
5. **Data Deduplication**: Implement logic to prevent duplicate users
6. **Advanced Filtering**: Add more filtering options like age range, registration date, etc.
7. **API Versioning**: Implement API versioning for future compatibility