Complete Laravel Community App - Detailed Installation & Frontend Setup

🚀 Prerequisites Installation

1. Install PHP 8.1+ (Ubuntu/Debian)

bash

```bash
# Update system
sudo apt update && sudo apt upgrade -y

# Install PHP and required extensions
sudo apt install php8.1 php8.1-cli php8.1-common php8.1-mysql php8.1-zip php8.1-gd php8.1-mbstr

# Verify PHP installation
php -v
php -m | grep -E "(mysql|gd|mbstring|curl|xml)"
```

2. Install Composer

bash

```bash
# Download and install Composer globally
curl -sS https://getcomposer.org/installer | php
sudo mv composer.phar /usr/local/bin/composer
sudo chmod +x /usr/local/bin/composer

# Verify installation
composer --version
```

3. Install Node.js & NPM

bash

```bash
# Install Node.js via NodeSource repository
curl -fsSL https://deb.nodesource.com/setup_18.x | sudo -E bash -
sudo apt-get install -y nodejs

# Verify installation
node --version
npm --version
```

4. Install MySQL

bash

```
# Install MySQL Server
sudo apt install mysql-server -y

# Secure MySQL installation
sudo mysql_secure_installation

# Start and enable MySQL
sudo systemctl start mysql
sudo systemctl enable mysql

# Test MySQL connection
sudo mysql -u root -p
```

5. Install Redis (Optional - for caching)

bash

```
sudo apt install redis-server -y
sudo systemctl start redis-server
sudo systemctl enable redis-server
```

📦 Project Setup

Step 1: Create Laravel Project

bash

```
# Create new Laravel project
composer create-project laravel/laravel community-app "10.*"
cd community-app

# Set proper permissions
sudo chown -R $USER:www-data storage
sudo chown -R $USER:www-data bootstrap/cache
chmod -R 775 storage
chmod -R 775 bootstrap/cache
```

Step 2: Environment Configuration

bash

```bash
# Copy environment file
cp .env.example .env

# Generate application key
php artisan key:generate
```

Complete .env Configuration

env

```
APP_NAME="Community Social App"
APP_ENV=local
APP_KEY=base64:your-generated-key-here
APP_DEBUG=true
APP_URL=http://localhost:8000

LOG_CHANNEL=stack
LOG_DEPRECATIONS_CHANNEL=null
LOG_LEVEL=debug

# Database Configuration
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=community_app
DB_USERNAME=root
DB_PASSWORD=your_mysql_password

# Broadcasting
BROADCAST_DRIVER=pusher
CACHE_DRIVER=redis
FILESYSTEM_DISK=local
QUEUE_CONNECTION=database
SESSION_DRIVER=database
SESSION_LIFETIME=120

# Redis
REDIS_HOST=127.0.0.1
REDIS_PASSWORD=null
REDIS_PORT=6379

# Mail Configuration (using Mailtrap for testing)
MAIL_MAILER=smtp
MAIL_HOST=smtp.mailtrap.io
MAIL_PORT=2525
MAIL_USERNAME=your_mailtrap_username
MAIL_PASSWORD=your_mailtrap_password
MAIL_ENCRYPTION=tls
MAIL_FROM_ADDRESS="noreply@communityapp.com"
MAIL_FROM_NAME="${APP_NAME}"

# Pusher (for real-time features)
PUSHER_APP_ID=your_pusher_app_id
```

```
PUSHER_APP_KEY=your_pusher_app_key
PUSHER_APP_SECRET=your_pusher_app_secret
PUSHER_HOST=
PUSHER_PORT=443
PUSHER_SCHEME=https
PUSHER_APP_CLUSTER=mt1

# File Upload Configuration
UPLOAD_MAX_SIZE=20480
MAX_UPLOAD_FILES=10

# App Configuration
SANCTUM_STATEFUL_DOMAINS=localhost:8000,127.0.0.1:8000
SESSION_DOMAIN=localhost
```

Step 3: Database Setup

bash

```
# Login to MySQL
mysql -u root -p

# Create database and user
CREATE DATABASE community_app CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
CREATE USER 'community_user'@'localhost' IDENTIFIED BY 'SecurePassword123!';
GRANT ALL PRIVILEGES ON community_app.* TO 'community_user'@'localhost';
FLUSH PRIVILEGES;
EXIT;
```

Update .env with new credentials:

env

```
DB_DATABASE=community_app
DB_USERNAME=community_user
DB_PASSWORD=SecurePassword123!
```

🔧 Backend Installation

Step 4: Install PHP Dependencies

bash

```bash
# Core Laravel packages
composer require laravel/sanctum
composer require laravel/socialite

# File handling & Image processing
composer require intervention/image
composer require spatie/laravel-media-library
composer require league/flysystem-aws-s3-v3

# Real-time features
composer require pusher/pusher-php-server

# Search functionality
composer require laravel/scout
composer require algolia/algoliasearch-client-php

# Additional utilities
composer require spatie/laravel-permission
composer require barryvdh/laravel-cors
composer require laravel/telescope

# Development tools
composer require --dev barryvdh/laravel-debugbar
composer require --dev laravel/pint
```

Step 5: Publish and Configure Packages

bash

```bash
# Publish Sanctum configuration
php artisan vendor:publish --provider="Laravel\Sanctum\SanctumServiceProvider"

# Publish Telescope (optional - for debugging)
php artisan telescope:install
php artisan migrate

# Create storage link
php artisan storage:link
```

Step 6: Create Database Structure

Create Migration Files

bash

```bash
# User-related migrations
php artisan make:migration add_fields_to_users_table --table=users
php artisan make:migration create_user_profiles_table

# Social features
php artisan make:migration create_posts_table
php artisan make:migration create_comments_table
php artisan make:migration create_likes_table
php artisan make:migration create_friendships_table
php artisan make:migration create_follows_table

# Community features
php artisan make:migration create_groups_table
php artisan make:migration create_group_members_table
php artisan make:migration create_group_posts_table

# Events system
php artisan make:migration create_events_table
php artisan make:migration create_event_attendees_table

# Notifications
php artisan make:migration create_notifications_table
```

Complete Migration Examples

Users Migration Enhancement:

php

```php
<?php
// database/migrations/xxxx_add_fields_to_users_table.php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    public function up()
    {
        Schema::table('users', function (Blueprint $table) {
            $table->string('username')->unique()->after('id');
            $table->string('first_name')->after('username');
            $table->string('last_name')->after('first_name');
            $table->text('bio')->nullable()->after('email_verified_at');
            $table->string('avatar')->nullable()->after('bio');
            $table->string('cover_photo')->nullable()->after('avatar');
            $table->string('location')->nullable()->after('cover_photo');
            $table->string('website')->nullable()->after('location');
            $table->date('birth_date')->nullable()->after('website');
            $table->enum('privacy_level', ['public', 'friends', 'private'])->default('public')-
            $table->boolean('is_active')->default(true)->after('privacy_level');
            $table->timestamp('last_seen_at')->nullable()->after('is_active');
            $table->json('settings')->nullable()->after('last_seen_at');
        });
    }

    public function down()
    {
        Schema::table('users', function (Blueprint $table) {
            $table->dropColumn([
                'username', 'first_name', 'last_name', 'bio', 'avatar',
                'cover_photo', 'location', 'website', 'birth_date',
                'privacy_level', 'is_active', 'last_seen_at', 'settings'
            ]);
        });
    }
};
```

Posts Migration:

php

```php
<?php
// database/migrations/xxxx_create_posts_table.php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    public function up()
    {
        Schema::create('posts', function (Blueprint $table) {
            $table->id();
            $table->foreignId('user_id')->constrained()->onDelete('cascade');
            $table->foreignId('group_id')->nullable()->constrained()->onDelete('cascade');
            $table->text('content');
            $table->json('media_urls')->nullable();
            $table->enum('privacy_level', ['public', 'friends', 'private', 'group'])->default('
            $table->enum('post_type', ['text', 'image', 'video', 'link', 'poll'])->default('tex
            $table->boolean('is_pinned')->default(false);
            $table->boolean('comments_enabled')->default(true);
            $table->integer('like_count')->default(0);
            $table->integer('comment_count')->default(0);
            $table->integer('share_count')->default(0);
            $table->json('metadata')->nullable(); // For polls, links, etc.
            $table->timestamps();

            $table->index(['user_id', 'created_at']);
            $table->index('privacy_level');
            $table->index('group_id');
            $table->fullText('content');
        });
    }

    public function down()
    {
        Schema::dropIfExists('posts');
    }
};
```

Groups Migration:

php

```php
<?php
// database/migrations/xxxx_create_groups_table.php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    public function up()
    {
        Schema::create('groups', function (Blueprint $table) {
            $table->id();
            $table->string('name');
            $table->text('description')->nullable();
            $table->string('cover_photo')->nullable();
            $table->enum('privacy_type', ['public', 'private', 'secret'])->default('public');
            $table->foreignId('created_by')->constrained('users')->onDelete('cascade');
            $table->integer('member_count')->default(1);
            $table->json('settings')->nullable();
            $table->boolean('is_active')->default(true);
            $table->timestamps();

            $table->index('privacy_type');
            $table->index('created_by');
            $table->fullText(['name', 'description']);
        });
    }

    public function down()
    {
        Schema::dropIfExists('groups');
    }
};
```

Run Migrations

bash

```bash
# Run all migrations
php artisan migrate

# Check migration status
php artisan migrate:status
```

Step 7: Create Models with Relationships

Enhanced User Model

php

```php
<?php
// app/Models/User.php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Foundation\Auth\User as Authenticatable;
use Illuminate\Notifications\Notifiable;
use Laravel\Sanctum\HasApiTokens;
use Spatie\Permission\Traits\HasRoles;

class User extends Authenticatable
{
    use HasApiTokens, HasFactory, Notifiable, HasRoles;

    protected $fillable = [
        'username', 'email', 'password', 'first_name', 'last_name',
        'bio', 'avatar', 'cover_photo', 'location', 'website',
        'birth_date', 'privacy_level', 'settings'
    ];

    protected $hidden = [
        'password', 'remember_token',
    ];

    protected $casts = [
        'email_verified_at' => 'datetime',
        'birth_date' => 'date',
        'last_seen_at' => 'datetime',
        'settings' => 'array',
    ];

    // Relationships
    public function posts()
    {
        return $this->hasMany(Post::class)->latest();
    }

    public function comments()
    {
        return $this->hasMany(Comment::class);
    }
```

```php
public function likes()
{
    return $this->hasMany(Like::class);
}

// Friendship relationships
public function friendsOfMine()
{
    return $this->belongsToMany(User::class, 'friendships', 'requester_id', 'addressee_id')
                ->wherePivot('status', 'accepted')
                ->withPivot('created_at');
}

public function friendOf()
{
    return $this->belongsToMany(User::class, 'friendships', 'addressee_id', 'requester_id')
                ->wherePivot('status', 'accepted')
                ->withPivot('created_at');
}

public function friendRequests()
{
    return $this->belongsToMany(User::class, 'friendships', 'addressee_id', 'requester_id')
                ->wherePivot('status', 'pending')
                ->withPivot('created_at');
}

// Group relationships
public function groups()
{
    return $this->belongsToMany(Group::class, 'group_members')
                ->withPivot('role', 'joined_at', 'status')
                ->withTimestamps();
}

public function ownedGroups()
{
    return $this->hasMany(Group::class, 'created_by');
}

// Event relationships
public function events()
{
    return $this->hasMany(Event::class, 'created_by');
```

```
    }

    public function attendingEvents()
    {
        return $this->belongsToMany(Event::class, 'event_attendees')
                    ->withPivot('status')
                    ->withTimestamps();
    }

    // Helper methods
    public function friends()
    {
        return $this->friendsOfMine->merge($this->friendOf);
    }

    public function getFullNameAttribute()
    {
        return trim($this->first_name . ' ' . $this->last_name);
    }

    public function getAvatarUrlAttribute()
    {
        return $this->avatar ? asset('storage/' . $this->avatar) : asset('images/default-avatar
    }

    public function isFriendWith(User $user)
    {
        return $this->friends()->contains($user);
    }

    public function hasSentFriendRequestTo(User $user)
    {
        return $this->friendsOfMine()->wherePivot('addressee_id', $user->id)->wherePivot('statu
    }
}
```

Post Model

php

```php
<?php
// app/Models/Post.php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Post extends Model
{
    use HasFactory;

    protected $fillable = [
        'user_id', 'group_id', 'content', 'media_urls', 'privacy_level',
        'post_type', 'is_pinned', 'comments_enabled', 'metadata'
    ];

    protected $casts = [
        'media_urls' => 'array',
        'metadata' => 'array',
        'is_pinned' => 'boolean',
        'comments_enabled' => 'boolean',
    ];

    // Relationships
    public function user()
    {
        return $this->belongsTo(User::class);
    }

    public function group()
    {
        return $this->belongsTo(Group::class);
    }

    public function comments()
    {
        return $this->hasMany(Comment::class)->latest();
    }

    public function likes()
    {
        return $this->morphMany(Like::class, 'likeable');
```

```
    }

    public function shares()
    {
        return $this->hasMany(Share::class);
    }

    // Scopes
    public function scopePublic($query)
    {
        return $query->where('privacy_level', 'public');
    }

    public function scopeForUser($query, User $user)
    {
        return $query->where(function($q) use ($user) {
            $q->where('user_id', $user->id)
              ->orWhere('privacy_level', 'public')
              ->orWhere(function($subQuery) use ($user) {
                  $subQuery->where('privacy_level', 'friends')
                           ->whereHas('user.friendsOfMine', function($friendQuery) use ($user)
                               $friendQuery->where('addressee_id', $user->id);
                           });
              });
        });
    }

    // Helper methods
    public function isLikedBy(User $user)
    {
        return $this->likes()->where('user_id', $user->id)->exists();
    }

    public function getMediaUrlsAttribute($value)
    {
        $urls = json_decode($value, true) ?? [];
        return array_map(function($url) {
            return asset('storage/' . $url);
        }, $urls);
    }
}
```

Step 8: Create Controllers

Authentication Controller

php

```php
<?php
// app/Http/Controllers/API/AuthController.php

namespace App\Http\Controllers\API;

use App\Http\Controllers\Controller;
use App\Models\User;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;
use Illuminate\Support\Facades\Hash;
use Illuminate\Support\Facades\Validator;

class AuthController extends Controller
{
    public function register(Request $request)
    {
        $validator = Validator::make($request->all(), [
            'username' => 'required|string|max:50|unique:users',
            'email' => 'required|string|email|max:255|unique:users',
            'password' => 'required|string|min:8|confirmed',
            'first_name' => 'required|string|max:100',
            'last_name' => 'required|string|max:100',
        ]);

        if ($validator->fails()) {
            return response()->json([
                'success' => false,
                'message' => 'Validation errors',
                'errors' => $validator->errors()
            ], 422);
        }

        $user = User::create([
            'username' => $request->username,
            'email' => $request->email,
            'password' => Hash::make($request->password),
            'first_name' => $request->first_name,
            'last_name' => $request->last_name,
        ]);

        $token = $user->createToken('auth_token')->plainTextToken;

        return response()->json([
```

```
            'success' => true,
            'message' => 'User registered successfully',
            'user' => $user,
            'token' => $token
        ], 201);
    }

    public function login(Request $request)
    {
        $validator = Validator::make($request->all(), [
            'login' => 'required|string', // Can be email or username
            'password' => 'required|string',
        ]);

        if ($validator->fails()) {
            return response()->json([
                'success' => false,
                'message' => 'Validation errors',
                'errors' => $validator->errors()
            ], 422);
        }

        $loginField = filter_var($request->login, FILTER_VALIDATE_EMAIL) ? 'email' : 'username'

        if (!Auth::attempt([$loginField => $request->login, 'password' => $request->password])) {
            return response()->json([
                'success' => false,
                'message' => 'Invalid credentials'
            ], 401);
        }

        $user = Auth::user();
        $user->update(['last_seen_at' => now()]);
        $token = $user->createToken('auth_token')->plainTextToken;

        return response()->json([
            'success' => true,
            'message' => 'Login successful',
            'user' => $user,
            'token' => $token
        ]);
    }

    public function logout(Request $request)
```

```
    {
        $request->user()->currentAccessToken()->delete();

        return response()->json([
            'success' => true,
            'message' => 'Logged out successfully'
        ]);
    }

    public function user(Request $request)
    {
        return response()->json([
            'success' => true,
            'user' => $request->user()->load(['groups', 'friendsOfMine', 'friendOf'])
        ]);
    }
}
```

Post Controller

php

```php
<?php
// app/Http/Controllers/API/PostController.php

namespace App\Http\Controllers\API;

use App\Http\Controllers\Controller;
use App\Models\Post;
use App\Models\Like;
use App\Models\Comment;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Storage;
use Intervention\Image\Facades\Image;

class PostController extends Controller
{
    public function index(Request $request)
    {
        $user = auth()->user();

        $posts = Post::forUser($user)
                    ->with(['user', 'group', 'comments.user', 'likes'])
                    ->withCount(['likes', 'comments'])
                    ->latest()
                    ->paginate(15);

        // Add user interaction data
        $posts->getCollection()->transform(function ($post) use ($user) {
            $post->is_liked = $post->isLikedBy($user);
            return $post;
        });

        return response()->json([
            'success' => true,
            'posts' => $posts
        ]);
    }

    public function store(Request $request)
    {
        $validator = \Validator::make($request->all(), [
            'content' => 'required|string|max:5000',
            'media' => 'nullable|array|max:10',
            'media.*' => 'file|mimes:jpg,jpeg,png,gif,mp4,mov|max:20480',
```

```
        'privacy_level' => 'in:public,friends,private',
        'group_id' => 'nullable|exists:groups,id',
        'post_type' => 'in:text,image,video,link,poll'
    ]);

    if ($validator->fails()) {
        return response()->json([
            'success' => false,
            'message' => 'Validation errors',
            'errors' => $validator->errors()
        ], 422);
    }

    $mediaUrls = [];

    if ($request->hasFile('media')) {
        foreach ($request->file('media') as $file) {
            $filename = time() . '_' . uniqid() . '.' . $file->getClientOriginalExtension()
            $path = "posts/{$filename}";

            // Store original file
            Storage::disk('public')->put($path, file_get_contents($file));

            // Create thumbnail for images
            if (in_array($file->getClientOriginalExtension(), ['jpg', 'jpeg', 'png'])) {
                $thumbnailPath = "posts/thumbs/{$filename}";
                $image = Image::make($file)->resize(300, 300, function ($constraint) {
                    $constraint->aspectRatio();
                    $constraint->upsize();
                });
                Storage::disk('public')->put($thumbnailPath, $image->encode());
            }

            $mediaUrls[] = $path;
        }
    }

    $post = Post::create([
        'user_id' => auth()->id(),
        'content' => $request->content,
        'media_urls' => $mediaUrls,
        'privacy_level' => $request->privacy_level ?? 'public',
        'group_id' => $request->group_id,
        'post_type' => $request->post_type ?? 'text',
```

```
        ]);

        return response()->json([
            'success' => true,
            'message' => 'Post created successfully',
            'post' => $post->load(['user', 'group'])
        ], 201);
    }

    public function show(Post $post)
    {
        $user = auth()->user();

        // Check if user can view this post
        if (!$this->canUserViewPost($user, $post)) {
            return response()->json([
                'success' => false,
                'message' => 'Access denied'
            ], 403);
        }

        $post->load(['user', 'group', 'comments.user', 'likes.user']);
        $post->is_liked = $post->isLikedBy($user);
        $post->loadCount(['likes', 'comments']);

        return response()->json([
            'success' => true,
            'post' => $post
        ]);
    }

    public function like(Post $post)
    {
        $user = auth()->user();

        if (!$this->canUserViewPost($user, $post)) {
            return response()->json([
                'success' => false,
                'message' => 'Access denied'
            ], 403);
        }

        $existingLike = $post->likes()->where('user_id', $user->id)->first();
```

```
    if ($existingLike) {
        $existingLike->delete();
        $post->decrement('like_count');
        $liked = false;
    } else {
        $post->likes()->create(['user_id' => $user->id]);
        $post->increment('like_count');
        $liked = true;
    }

    return response()->json([
        'success' => true,
        'liked' => $liked,
        'like_count' => $post->fresh()->like_count
    ]);
}

public function comment(Request $request, Post $post)
{
    $user = auth()->user();

    if (!$this->canUserViewPost($user, $post) || !$post->comments_enabled) {
        return response()->json([
            'success' => false,
            'message' => 'Access denied'
        ], 403);
    }

    $validator = \Validator::make($request->all(), [
        'content' => 'required|string|max:1000',
    ]);

    if ($validator->fails()) {
        return response()->json([
            'success' => false,
            'errors' => $validator->errors()
        ], 422);
    }

    $comment = $post->comments()->create([
        'user_id' => $user->id,
        'content' => $request->content,
    ]);
```

```
        $post->increment('comment_count');

        return response()->json([
            'success' => true,
            'message' => 'Comment added successfully',
            'comment' => $comment->load('user')
        ], 201);
    }

    private function canUserViewPost($user, $post)
    {
        if ($post->privacy_level === 'public') return true;
        if ($post->user_id === $user->id) return true;
        if ($post->privacy_level === 'friends' && $post->user->isFriendWith($user)) return true
        if ($post->group_id && $user->groups->contains($post->group_id)) return true;

        return false;
    }
}
```

🎨 Frontend Installation & Setup

Step 9: Install Frontend Dependencies

bash

```bash
# Core Vue.js ecosystem
npm install vue@next @vitejs/plugin-vue vue-router@4 pinia

# UI Framework & Styling
npm install bootstrap@5.3.0 @popperjs/core
npm install sass
npm install @headlessui/vue @heroicons/vue

# HTTP Client & Real-time
npm install axios
npm install laravel-echo pusher-js

# Form handling & validation
npm install vee-validate @vee-validate/yup yup

# Date handling
npm install date-fns

# File upload
npm install vue-filepond filepond-plugin-image-preview filepond-plugin-file-validate-type

# Icons
npm install lucide-vue-next

# Utilities
npm install lodash
npm install @tailwindcss/forms @tailwindcss/typography
```

Step 10: Configure Vite

javascript

```javascript
// vite.config.js
import { defineConfig } from 'vite';
import laravel from 'laravel-vite-plugin';
import vue from '@vitejs/plugin-vue';
import { resolve } from 'path';

export default defineConfig({
    plugins: [
        laravel({
            input: ['resources/css/app.css', 'resources/js/app.js'],
            refresh: true,
        }),
        vue({
            template: {
                transformAssetUrls: {
                    base: null,
                    includeAbsolute: false,
                },
            },
        }),
    ],
    resolve: {
        alias: {
            '@': resolve(__dirname, 'resources/js'),
            '~bootstrap': resolve(__dirname, 'node_modules/bootstrap'),
        },
    },
    define: {
        __VUE_PROD_DEVTOOLS__: false,
    },
});
```

Step 11: Create Frontend Architecture

Main App Entry Point

javascript

```javascript
// resources/js/app.js
import { createApp } from 'vue';
import { createPinia } from 'pinia';
import router from './router';
import App from './App.vue';
import './bootstrap';

// Global styles
import '../css/app.css';
import 'bootstrap/dist/css/bootstrap.min.css';
import 'bootstrap/dist/js/bootstrap.bundle.min.js';

const app = createApp(App);
const pinia = createPinia();

app.use(pinia);
app.use(router);

// Global properties
app.config.globalProperties.$http = window.axios;

app.mount('#app');
```

Bootstrap Axios Configuration

javascript

```javascript
// resources/js/bootstrap.js
import axios from 'axios';
import Echo from 'laravel-echo';
import Pusher from 'pusher-js';

// Configure Axios
window.axios = axios;
window.axios.defaults.headers.common['X-Requested-With'] = 'XMLHttpRequest';
window.axios.defaults.baseURL = '/api';

// Add token to requests if available
const token = localStorage.getItem('auth_token');
if (token) {
    window.axios.defaults.headers.common['Authorization'] = `Bearer ${token}`;
}

// Request interceptor to add token
window.axios.interceptors.request.use(
    config => {
        const token = localStorage.getItem('auth_token');
        if (token) {
            config.headers.Authorization = `Bearer ${token}`;
        }
        return config;
```