Improving Search Result Load Times on TravelGay.com

Problem Overview

Users experience a 10-second delay when searching for hotels, while the API responds within 1 second. Our goal is to reduce this time to under 3 seconds.

Proposed Solutions with Code Examples

1. Optimize Database Access

a. Indexing

Action: Add indexes on frequently queried fields like `city`, `propertyID`, and `isTGApproved`.

Laravel Migration Example:

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

class AddIndexesToHotelsTable extends Migration
{
    public function up()
    {
        Schema::table('hotels', function (Blueprint $table) {
            $table->index('city');
            $table->index('propertyID');
            $table->index('isTGApproved');
        });
    }

    public function down()
    {
        Schema::table('hotels', function (Blueprint $table) {
            $table->dropIndex(['city']);
            $table->dropIndex(['propertyID']);
            $table->dropIndex(['isTGApproved']);
        });
    }
}
```

Explanation: This migration adds indexes to the `city`, `propertyID`, and `isTGApproved` columns to speed up queries filtering on these fields.

b. Query Optimization

Action: Refactor slow queries for efficiency.

Before Optimization:

```
// Inefficient: Fetches all hotels and filters in PHP
$hotels = Hotel::all()->filter(function ($hotel) use ($city) {
    return $hotel->city == $city;
});
```

After Optimization:

```
// Efficient: Filters hotels at the database level
$hotels = Hotel::where('city', $city)->get();
```

Explanation: Querying the database directly for hotels in the specified city reduces memory usage and processing time.

c. Caching

Action: Implement Redis/Memcached to cache frequently accessed data.

Laravel Caching Example:

```
use Illuminate\Support\Facades\Cache;

$cacheKey = 'hotels_in_' . $city;

$hotels = Cache::remember($cacheKey, now()->addMinutes(30), function () use ($city) {
    return Hotel::where('city', $city)->get();
});
```

Explanation: This caches the hotel data for a city, so subsequent requests within 30 minutes retrieve data from the cache instead of querying the database.

2. Improve Sorting Efficiency

a. Pre-Sort "TG Approved" Hotels

Action: Keep a pre-sorted list to eliminate runtime sorting.

Laravel Eloquent Example:

```
// Fetch TG Approved hotels
$tgApprovedHotels = Hotel::where('city', $city)
    ->where('isTGApproved', true)
    ->get();

// Fetch other hotels
$otherHotels = Hotel::where('city', $city)
    ->where('isTGApproved', false)
    ->get();

// Merge the collections
$hotels = $tgApprovedHotels->merge($otherHotels);
```

Explanation: Separating the queries ensures "TG Approved" hotels are listed first without the need for sorting after retrieval.

b. Efficient Sorting

Action: Use database-level sorting to speed up the process.

Example:

```
$hotels = Hotel::where('city', $city)
    ->orderBy('isTGApproved', 'desc')
    ->get();
```

Explanation: Sorting is handled by the database, which is optimized for such operations.

3. Enhance Data Conversion

a. Streamline Data Transformation

Action: Optimize data transformation to reduce overhead.

Before Optimization:

```
$hotelData = $hotels->map(function ($hotel) {
    return [
        'id' => $hotel->id,
        'name' => $hotel->name,
        'description' => $hotel->description,
        'images' => $hotel->images,
        // Additional processing...
    ];
});
```

After Optimization:

```
$hotelData = $hotels->makeHidden(['created_at', 'updated_at', 'unnecessary_field'])
    ->toArray();
```

Explanation: Using Eloquent's built-in methods to hide unnecessary fields and convert to an array simplifies the transformation.

b. Lightweight Formats

Action: Ensure data is transferred in lightweight formats like JSON.

Laravel Controller Example:

```
return response()->json($hotelData);
```

Explanation: JSON is a lightweight data-interchange format that is easy to parse and generate.

4. Asynchronous Loading

a. Load Essential Data First

Action: Display basic info immediately and load additional details asynchronously.

Backend Controller:

```
// Return essential hotel data
$hotels = Hotel::where('city', $city)
    ->select('id', 'name', 'isTGApproved')
    ->orderBy('isTGApproved', 'desc')
    ->get();

return response()->json($hotels);
```

Frontend JavaScript:

```
// Display basic hotel info
hotels.forEach(function(hotel) {
    displayBasicInfo(hotel);

    // Fetch additional details asynchronously
    fetch(`/api/hotels/${hotel.id}/details`)
        .then(response => response.json())
        .then(details => {
            updateHotelDetails(hotel.id, details);
        });
});
```

Explanation: Users see search results quickly, and detailed information loads in the background.

5. Optimize API Usage

a. Smaller Batches

Action: Reduce API request batch sizes to improve performance.

Example:

```
$hotelIds = getHotelIds(); // Assume this retrieves relevant hotel IDs
$chunks = array_chunk($hotelIds, 50); // Reduced batch size from 100 to 50

foreach ($chunks as $chunk) {
    // Send API requests with smaller chunks
    $apiResults = sendApiRequest($chunk);
    // Process results...
}
```

Explanation: Smaller batches may be processed faster by the API and reduce the likelihood of timeouts.

b. Limit API Calls

Action: Only request necessary data from the API.

Example:

```
// Determine which hotels need fresh data
$hotelIdsToUpdate = getHotelIdsNeedingUpdate();

// Fetch data only for these hotels
if (!empty($hotelIdsToUpdate)) {
    $apiResults = sendApiRequest($hotelIdsToUpdate);
    // Update local database with new data
}
```

Explanation: Minimizing API calls reduces external dependencies and improves performance.

6. Use Faster Frameworks (Optional)

a. Move Critical APIs to Lumen or Go

Action: Re-implement critical API endpoints using Lumen or Go.

Lumen Route Example:

```
// routes/web.php
$router->get('/hotels', 'HotelController@index');
```

Lumen Controller Example:

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

use Illuminate\Http\Request;
use App\Models\Hotel;

class HotelController extends Controller
{
    public function index(Request $request)
    {
        $city = $request->input('city');
        $hotels = Hotel::where('city', $city)
            ->orderBy('isTGApproved', 'desc')
            ->get();

        return response()->json($hotels);
    }
}
```

Explanation: Lumen is optimized for microservices and APIs, offering better performance than full-stack frameworks.

Assumptions

- The application uses a microservice architecture with load balancing.
- The database structure is flexible and can be optimized.
- The codebase allows modifications in data processing and API handling.
- Data from the third-party API is regularly updated in our database.

Expected Outcome

- **Reduced Load Times**: Achieve search result load times under 3 seconds.
- **Improved User Experience**: Faster responses enhance user satisfaction and increase conversion rates.
- **Competitive Edge**: Improved performance aligns with industry standards, making the platform more competitive.

This report outlines practical steps with code examples to optimize TravelGay.com's search process and deliver faster results for users.