Metro Train Prediction App - Performance Optimization Guide

Author: David Morrison

Project Repo: https://github.com/DavMorr/wmata-app

Overview

This document describes the performance-related features implemented in the Metro Train Prediction App based on the actual codebase. All information reflects existing implementation only.

Backend Performance Features

Service Provider Registration

WmataServiceProvider registers services as singletons:

php

```php
// app/Providers/WmataServiceProvider.php
public function register(): void
{
    $this->app->singleton(WmataApiService::class, function ($app) {
        $config = config('wmata');

        return new WmataApiService(
            apiKey: $config['api']['key'],
            baseUrl: $config['api']['base_url'],
            endpoints: $config['endpoints'],
            cacheConfig: $config['cache'],
            maxRequestsPerHour: $config['rate_limit']['max_requests_per_hour']
        );
    });

    $this->app->singleton(MetroDataService::class, function ($app) {
        return new MetroDataService(
            $app->make(WmataApiService::class)
        );
    });
}
```

Controller Implementation

MetroController implements consistent error handling:

php

```php
// app/Http/Controllers/Api/MetroController.php
public function getLines(): JsonResponse
{
    try {
        $lines = $this->metroService->getCachedLines();

        if (empty($lines)) {
            $this->metroService->syncAllMetroData();
            $lines = $this->metroService->getCachedLines();
        }

        return response()->json([
            'success' => true,
```

```
            'data' => $lines,
        ]);

    } catch (\Exception $e) {
        return response()->json([
            'success' => false,
            'error' => 'Failed to load lines: ' . $e->getMessage(),
        ], 500);
    }
}
```

Caching Implementation

WmataApiService Caching

The service implements caching for all WMATA API calls:

php

```
// Lines caching
public function getLines(): array
{
    $cacheKey = 'wmata.lines';

    return Cache::remember($cacheKey, $this->cacheConfig['lines_ttl'], function () {
        $response = $this->makeRequest($this->endpoints['lines']);

        return array_map(
            fn($line) => LineDto::fromArray($line),
            $response['Lines'] ?? []
        );
    });
}

// Station caching by line
public function getStationsForLine(string $lineCode): array
{
    $cacheKey = "wmata.stations.line.{$lineCode}";

    return Cache::remember($cacheKey, $this->cacheConfig['stations_ttl'], function () use
($lineCode) {
        $endpoint = $this->endpoints['stations'] . "?LineCode={$lineCode}";
        $response = $this->makeRequest($endpoint);

        return array_map(
            fn($station) => StationDto::fromArray($station),
            $response['Stations'] ?? []
        );
    });
}

// All stations caching
```

```php
public function getAllStations(): array
{
    $cacheKey = 'wmata.stations.all';

    return Cache::remember($cacheKey, $this->cacheConfig['stations_ttl'], function () {
        // Implementation details from actual code...
    });
}

// Predictions caching
public function getTrainPredictions(string $stationCode): array
{
    $singleStationCode = $this->extractFirstStationCode($stationCode);
    $cacheKey = "wmata.predictions.{$singleStationCode}";

    return Cache::remember($cacheKey, $this->cacheConfig['predictions_ttl'], function () use
($singleStationCode) {
        $endpoint = $this->endpoints['predictions'] . "/{$singleStationCode}";
        $response = $this->makeRequest($endpoint);

        return array_map(
            fn($train) => TrainPredictionDto::fromArray($train),
            $response['Trains'] ?? []
        );
    });
}
```

MetroDataService Caching

Frontend-optimized caching methods:

php

```php
// Frontend lines cache
public function getCachedLines(): array
{
    return Cache::remember('metro.lines.frontend', 3600, function () {
        return Line::all()->map(function ($line) {
            return [
                'value' => $line->line_code,
                'label' => $line->display_name,
            ];
        })->toArray();
    });
}

// Ordered stations cache
public function getOrderedStationsForLine(string $lineCode): array
{
    $cacheKey = "metro.stations.ordered.{$lineCode}";

    return Cache::remember($cacheKey, 3600, function () use ($lineCode) {
        $stations = Station::where('line_code_1', $lineCode)
            ->orWhere('line_code_2', $lineCode)
```

```
            ->orWhere('line_code_3', $lineCode)
            ->orWhere('line_code_4', $lineCode)
            ->get();

        $stationCodes = $stations->pluck('code')->toArray();

        $orderedPaths = StationPath::forLine($lineCode)
            ->whereIn('station_code', $stationCodes)
            ->ordered()
            ->get();

        if ($orderedPaths->isEmpty()) {
            Log::warning("No path data found for line {$lineCode}, using unordered stations");
            return $stations->map(function ($station) {
                return [
                    'value' => $station->code,
                    'label' => $station->name,
                ];
            })->toArray();
        }

        return $orderedPaths->map(function ($path) {
            return [
                'value' => $path->station_code,
                'label' => $path->station_name,
                'seq_num' => $path->seq_num,
                'distance_to_prev' => $path->distance_to_prev,
            ];
        })->toArray();
    });
}

// Cache integrity validation
public function validateCacheIntegrity(): bool
{
    $hasLines = Cache::has('metro.lines.frontend');
    $hasStations = Cache::has('wmata.stations.all');

    return $hasLines && $hasStations;
}
```

Database Structure

Table Indexes

From the migration files:

sql

```
-- Lines table (2025_06_04_184151_create_lines_table.php)
$table->string('line_code', 2)->primary();
$table->index('display_name');
```

```
-- Stations table (2025_06_04_184201_create_stations_table.php)
$table->string('code', 3)->primary();
$table->index('name');
$table->index(['lat', 'lon']);
$table->index('is_active');

-- Station addresses table (2025_06_04_184208_create_station_addresses_table.php)
$table->string('station_code', 3)->primary();
$table->index(['city', 'state']);
$table->index('zip_code');

-- Station paths table (2025_06_04_184214_create_station_paths_table.php)
$table->id();
$table->index(['line_code', 'seq_num']);
$table->index('station_code');
$table->unique(['line_code', 'station_code']);
```

Model Scopes

Station model includes query scopes:
php

```
// app/Models/Station.php
public function scopeOnLine($query, string $lineCode)
{
    return $query->where(function ($q) use ($lineCode) {
        $q->where('line_code_1', $lineCode)
          ->orWhere('line_code_2', $lineCode)
          ->orWhere('line_code_3', $lineCode)
          ->orWhere('line_code_4', $lineCode);
    });
}
```

StationPath model includes ordering:
php

```
// app/Models/StationPath.php
public function scopeForLine($query, string $lineCode)
{
    return $query->where('line_code', $lineCode);
}

public function scopeOrdered($query)
{
    return $query->orderBy('seq_num');
}
```

API Performance

Rate Limiting

WmataApiService implements rate limiting:

php

```php
private const RATE_LIMIT_KEY = 'wmata_api_rate_limit';

private function checkRateLimit(): bool
{
    $currentCount = Cache::get(self::RATE_LIMIT_KEY, 0);
    return $currentCount < $this->maxRequestsPerHour;
}

private function incrementRateLimit(): void
{
    $currentCount = Cache::get(self::RATE_LIMIT_KEY, 0);
    Cache::put(self::RATE_LIMIT_KEY, $currentCount + 1, 3600);
}
```

HTTP Client Configuration

Request configuration with retries:

php

```php
$response = Http::withHeaders([
    'api_key' => $this->apiKey,
    'Accept' => 'application/json',
])
->timeout(30)
->retry(3, 1000, function ($exception) {
    return $exception instanceof \Illuminate\Http\Client\ConnectionException;
})
->get($url);
```

Route Rate Limiting

API routes include throttling middleware:

php

```php
// routes/api.php
Route::prefix('metro')->middleware(['throttle:60,1'])->group(function () {
    Route::get('lines', [MetroController::class, 'getLines']);
    Route::get('stations/{lineCode}', [MetroController::class, 'getStationsForLine']);
    Route::get('predictions/{stationCode}', [MetroController::class, 'getTrainPredictions']);
    Route::post('sync', [MetroController::class, 'syncData']);
});
```

Frontend Implementation

Component Lifecycle Management

MetroTrainPredictor.vue implements timer management:

javascript

```
// Auto-refresh timer management
let refreshTimer = null

const onStationChange = () => {
  predictions.value = []

  if (refreshTimer) {
    clearInterval(refreshTimer)
    refreshTimer = null
  }

  if (selectedStation.value && selectedStation.value !== '') {
    fetchPredictions(selectedStation.value)

    refreshTimer = setInterval(() => {
      fetchPredictions(selectedStation.value)
    }, refreshInterval.value * 1000)
  }
}

// Cleanup on unmount
onUnmounted(() => {
  if (refreshTimer) {
    clearInterval(refreshTimer)
  }
})
```

API Service

metroApi.js implements consistent error handling:

javascript

```
class MetroApiService {
    async makeRequest(endpoint) {
        try {
            const response = await api.get(endpoint)

            if (!response.data.success) {
                throw new Error(response.data.error || 'API request failed')
            }

            return response.data.data
        } catch (error) {
            console.error('Metro API Error:', error)
            throw error
        }
```

```
    }

    async getTrainPredictions(stationCode) {
        try {
            const response = await api.get(`/metro/predictions/${stationCode}`)

            if (!response.data.success) {
                throw new Error(response.data.error || 'API request failed')
            }

            return response.data.data
        } catch (error) {
            console.error('Metro API Error:', error)
            throw error
        }
    }
}
```

Configuration

WMATA Configuration

Cache and rate limiting configuration from config/wmata.php:

php

```
'cache' => [
    'lines_ttl' => env('WMATA_CACHE_LINES_TTL', 86400),
    'stations_ttl' => env('WMATA_CACHE_STATIONS_TTL', 86400),
    'paths_ttl' => env('WMATA_CACHE_PATHS_TTL', 86400),
    'predictions_ttl' => env('WMATA_CACHE_PREDICTIONS_TTL', 15),
],

'rate_limit' => [
    'max_requests_per_hour' => env('WMATA_RATE_LIMIT', 1000),
],

'frontend' => [
    'predictions_refresh_interval' => env('WMATA_FRONTEND_REFRESH', 30),
],
```

Available Cache Keys

Based on actual implementation:

Cache Key	Purpose	TTL Source
`wmata.lines`	WMATA API lines	`lines_ttl`
`wmata.stations.all`	All stations from WMATA	`stations_ttl`

`wmata.stations.line.{lineCode}`	Stations for specific line	`stations_ttl`
`wmata.predictions.{stationCode}`	Train predictions	`predictions_ttl`
`metro.lines.frontend`	Frontend-formatted lines	3600 seconds
`metro.stations.ordered.{lineCode}`	Ordered stations for line	3600 seconds
`wmata_api_rate_limit`	Rate limiting counter	3600 seconds