Metro Train Prediction App - Performance Optimization Guide

Author: David Morrison

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

[Overview 1](#_Toc1142608474)

[Backend Performance Features 1](#_Toc433226305)

[Service Provider Registration 1](#_Toc779081915)

[Controller Implementation 1](#_Toc40869107)

[Caching Implementation 1](#_Toc917755978)

[WmataApiService Caching 1](#_Toc1169215995)

[MetroDataService Caching 1](#_Toc1911119318)

[Database Structure 1](#_Toc1622626137)

[Table Indexes 1](#_Toc1960239517)

[Model Scopes 1](#_Toc1506230093)

[API Performance 1](#_Toc1802223327)

[Rate Limiting 1](#_Toc833964724)

[HTTP Client Configuration 1](#_Toc360390577)

[Route Rate Limiting 1](#_Toc1595230192)

[Frontend Implementation 1](#_Toc890788560)

[Component Lifecycle Management 1](#_Toc1244891325)

[API Service 1](#_Toc1229508587)

[Configuration 1](#_Toc1727110561)

[WMATA Configuration 1](#_Toc33183138)

[Available Cache Keys 1](#_Toc1422915502)

## **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

*// 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

*// 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*

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

*// 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

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

$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

*// 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 |