Metro Transit App - Project Integration Guide

📋 Table of Contents

1. Project Overview
2. Repository Structure
3. Backend Integration (laravel-app/)
4. Frontend Integration (vue-app/)
5. API Communication Setup
6. Integration Checklist
7. Development Workflow
8. Testing Strategy

🎯 Project Overview

What We're Building

A Metro Transit application with **separated Laravel backend and Vue 3 frontend** that provides real-time train predictions using the WMATA API.

Key Features

- **Progressive Form Interface**: Line → Station → Real-time Predictions
- **Ordered Station Display**: Stations shown in proper sequence along metro lines
- **Real-time Updates**: Auto-refreshing predictions every 30 seconds
- **Cached Data Management**: Static data cached indefinitely, predictions cached 15 seconds
- **Rate Limiting & Fault Tolerance**: Comprehensive error handling

API Integration

- **Lines**: `GET /Rail.svc/json/jLines` (cached indefinitely)
- **Stations**: `GET /Rail.svc/json/jStations?LineCode={code}` (cached indefinitely)
- **Predictions**: `GET /StationPrediction.svc/json/GetPrediction/{stationCode}` (cached 15s)
- **Paths**: `GET /Rail.svc/json/jPath?FromStationCode={from}&ToStationCode={to}` (cached indefinitely)

📁 Repository Structure

```
project-root/
├── laravel-app/          # Laravel 12 Backend API
│   ├── app/
│   ├── database/
│   ├── routes/
│   ├── config/
│   └── .env
├── vue-app/              # Vue 3 Frontend
│   ├── src/
│   ├── components/
│   ├── package.json
│   └── .env
├── README.md
└── docker-compose.yml    # If using Docker
```

Benefits of This Structure

✅ **Clear separation** of backend and frontend concerns
✅ **Independent deployments** possible
✅ **Different teams** can work on each app
✅ **Technology flexibility** - can swap out either side
✅ **Easier testing** and development workflows

🔧 Backend Integration (laravel-app/)

File Structure to Create

```
laravel-app/
├── app/
│   ├── Models/
│   │   ├── Line.php
│   │   ├── Station.php
│   │   ├── StationAddress.php
│   │   └── StationPath.php
│   ├── DTOs/
│   │   ├── LineDto.php
│   │   ├── StationDto.php
│   │   ├── AddressDto.php
│   │   ├── TrainPredictionDto.php
│   │   └── StationPathDto.php
│   ├── Services/
│   │   ├── WmataApiService.php
│   │   └── MetroDataService.php
│   ├── Http/Controllers/Api/
│   │   └── MetroController.php
│   ├── Console/Commands/
│   │   └── SyncMetroData.php
│   └── Providers/
│       └── WmataServiceProvider.php
├── database/migrations/
│   ├── create_lines_table.php
│   ├── create_stations_table.php
│   ├── create_station_addresses_table.php
│   ├── create_station_paths_table.php
│   └── add_foreign_keys_to_lines_table.php
├── config/
│   └── wmata.php
└── routes/
    └── api.php (add metro routes)
```

Laravel Backend API Endpoints

The Laravel app will serve as a REST API:

```
GET /api/metro/lines
GET /api/metro/stations/{lineCode}
GET /api/metro/predictions/{stationCode}
GET /api/metro/path/{lineCode}
POST /api/metro/sync (admin)
```

Backend Environment Configuration (Safe Addition)

Add these WMATA-specific settings to your existing `laravel-app/.env` file:

```env
# WMATA API Configuration (add to existing .env)
WMATA_API_KEY=your_wmata_api_key_here
WMATA_BASE_URL=https://api.wmata.com
WMATA_TIMEOUT=30
WMATA_RETRY_ATTEMPTS=3

# WMATA Cache timing configuration
WMATA_CACHE_LINES_TTL=86400
WMATA_CACHE_STATIONS_TTL=86400
WMATA_CACHE_PATHS_TTL=86400
WMATA_CACHE_PREDICTIONS_TTL=15

# WMATA Rate limiting
WMATA_RATE_LIMIT=1000

# WMATA Frontend refresh interval (seconds)
WMATA_FRONTEND_REFRESH=30

# Optional: For better Metro app performance, consider changing:
# CACHE_STORE=redis  (currently you have CACHE_STORE=database)
```

Note: Keep all your existing settings unchanged. Your current cache and Redis configuration will work perfectly with the Metro app.

Quick Start Commands (Laravel with Sail)

```bash
# Navigate to Laravel app
cd laravel-app/

# Start Sail (if not already running)
sail up -d

# Install dependencies (if not already done)
sail composer install

# Create and run migrations
sail artisan make:migration create_lines_table
sail artisan make:migration create_stations_table
sail artisan make:migration create_station_addresses_table
sail artisan make:migration create_station_paths_table
sail artisan make:migration add_foreign_keys_to_lines_table

sail artisan migrate

# NOTE: Before running metro:sync, you must:
# 1. Add all Models (Line, Station, StationAddress, StationPath)
# 2. Add all DTOs (LineDto, StationDto, AddressDto, etc.)
# 3. Add all Services (WmataApiService, MetroDataService)
# 4. Add the Controller (MetroController)
# 5. Create and implement the SyncMetroData command
# 6. Register the WmataServiceProvider
# 7. Add your WMATA_API_KEY to .env

# After implementing all the above code:
sail artisan metro:sync  # This will work once everything is implemented
```

🎨 Frontend Integration (vue-app/)

Vue App Structure

```
vue-app/
├── src/
│   ├── components/
│   │   └── MetroTrainPredictor.vue
│   ├── services/
│   │   └── metroApi.js
│   ├── App.vue
│   └── main.js
├── package.json
└── .env
```

Frontend Environment Configuration (for Sail)

Add to `vue-app/.env`:

```env
# Laravel API Base URL (Sail default)
VITE_API_BASE_URL=http://localhost/api

# Development settings
VITE_APP_NAME="Metro Transit Predictor"
VITE_PREDICTIONS_REFRESH_INTERVAL=30
```

Vue Component Implementation

MetroTrainPredictor.vue

vue

```
<!-- vue-app/src/components/MetroTrainPredictor.vue -->
<template>
  <div class="metro-predictor">
    <h2>Metro Train Predictions</h2>

    <form @submit.prevent class="prediction-form">
      <!-- Line Selection -->
      <div class="form-group">
        <label for="line">Line:</label>
        <select
          id="line"
          v-model="selectedLine"
          @change="onLineChange"
          required
        >
          <option value="" disabled>Select a line</option>
          <option
            v-for="line in lines"
            :key="line.value"
            :value="line.value"
          >
            {{ line.label }}
          </option>
        </select>
      </div>

      <!-- Station Selection -->
      <div class="form-group" v-if="selectedLine && stations.length > 0">
        <label for="station">Station:</label>
        <select
          id="station"
          v-model="selectedStation"
          @change="onStationChange"
          required
        >
          <option value="" disabled>Select a station</option>
          <option
            v-for="station in stations"
            :key="station.value"
            :value="station.value"
          >
            {{ station.label }}
          </option>
        </select>
      </div>
    </form>

    <!-- Loading States -->
    <div v-if="loading.stations" class="loading">
      Loading stations for {{ getLineName(selectedLine) }} line...
    </div>
    <div v-if="loading.predictions" class="loading">
      Loading train predictions...
    </div>

    <!-- Train Predictions -->
    <div v-if="predictions && predictions.length > 0" class="predictions">
      <h3>
        Train arrival times for: {{ stationInfo.name }} ({{ stationInfo.code }})
      </h3>
      <ul class="prediction-list">
        <li
          v-for="(prediction, index) in predictions"
          :key="index"
          class="prediction-item"
        >
          <span class="line-name">{{ getLineName(prediction.line) }} line</span>
          to {{ prediction.destination }}:
          <span class="arrival-time" :class="getArrivalClass(prediction.minutes)">
            {{ formatArrivalTime(prediction.minutes) }}
          </span>
          <span class="car-count">({{ prediction.cars }} cars)</span>
        </li>
      </ul>
      <div class="last-updated">
        Last updated: {{ lastUpdated }}
        <span v-if="refreshInterval" class="refresh-info">
          (refreshes every {{ refreshInterval }}s)
        </span>
      </div>
    </div>

    <!-- No Predictions -->
    <div v-else-if="selectedStation && !loading.predictions" class="no-predictions">
      No train predictions available for this station.
    </div>
```

```
    <!-- Error Messages -->
    <div v-if="error" class="error">
      {{ error }}
    </div>
  </div>
</template>

<script setup>
import { ref, reactive, onMounted, watch, onUnmounted } from 'vue'
import { metroApi } from '../services/metroApi.js'

// Reactive state
const selectedLine = ref('')
const selectedStation = ref('')
const lines = ref([])
const stations = ref([])
const predictions = ref([])
const stationInfo = ref({})
const lastUpdated = ref('')
const error = ref('')
const refreshInterval = ref(30)

const loading = reactive({
  lines: false,
  stations: false,
  predictions: false
})

// Auto-refresh management
let refreshTimer = null

// Computed helpers
const getLineName = (lineCode) => {
  const line = lines.value.find(l => l.value === lineCode)
  return line ? line.label : lineCode
}

const formatArrivalTime = (minutes) => {
  if (minutes === 'BRD') return 'Boarding'
  if (minutes === 'ARR') return 'Arriving'
  return `${minutes} min${minutes !== '1' ? 's' : ''}`
}

const getArrivalClass = (minutes) => {
  if (minutes === 'BRD' || minutes === 'ARR') return 'arriving'
  const num = parseInt(minutes)
  if (num <= 2) return 'soon'
  if (num <= 5) return 'moderate'
  return 'later'
}

// API functions
const fetchLines = async () => {
  loading.lines = true
  error.value = ''

  try {
    const data = await metroApi.getLines()
    lines.value = data
  } catch (err) {
    error.value = `Failed to load lines: ${err.message}`
  } finally {
    loading.lines = false
  }
}

const fetchStations = async (lineCode) => {
  loading.stations = true
  error.value = ''

  try {
    const data = await metroApi.getStationsForLine(lineCode)
    stations.value = data
  } catch (err) {
    error.value = `Failed to load stations: ${err.message}`
    stations.value = []
  } finally {
    loading.stations = false
  }
}

const fetchPredictions = async (stationCode) => {
  loading.predictions = true
  error.value = ''

  try {
    const data = await metroApi.getTrainPredictions(stationCode)
    predictions.value = data.predictions
    stationInfo.value = data.station
    lastUpdated.value = new Date(data.updated_at).toLocaleTimeString()
    refreshInterval.value = data.refresh_interval || 30
  } catch (err) {
    error.value = `Failed to load predictions: ${err.message}`
    predictions.value = []
  } finally {
    loading.predictions = false
  }
}

// Event handlers
const onLineChange = () => {
  selectedStation.value = ''
  stations.value = []
  predictions.value = []

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

  if (selectedLine.value && selectedLine.value !== '') {
    fetchStations(selectedLine.value)
  }
}

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)
  }
}
```

```
// Lifecycle
onMounted(() => {
  fetchLines()
})

onUnmounted(() => {
  if (refreshTimer) {
    clearInterval(refreshTimer)
  }
})
</script>

<style scoped>
.metro-predictor {
  max-width: 800px;
  margin: 0 auto;
  padding: 20px;
  font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
}

.prediction-form {
  background: #f8f9fa;
  padding: 20px;
  border-radius: 8px;
  margin-bottom: 20px;
}

.form-group {
  margin-bottom: 16px;
}

.form-group label {
  display: block;
  margin-bottom: 8px;
  font-weight: 600;
  color: #333;
}

.form-group select {
  width: 100%;
  padding: 12px;
  border: 2px solid #ddd;
  border-radius: 6px;
  font-size: 16px;
  background: white;
  transition: border-color 0.3s;
}

.form-group select:focus {
  outline: none;
  border-color: #0056b3;
}

.loading {
  text-align: center;
  padding: 20px;
  color: #666;
  font-style: italic;
}

.predictions {
  background: white;
  border: 1px solid #ddd;
  border-radius: 8px;
  padding: 20px;
  margin-top: 20px;
}

.predictions h3 {
  margin-top: 0;
  color: #333;
  border-bottom: 2px solid #0056b3;
  padding-bottom: 10px;
}

.prediction-list {
  list-style: none;
  padding: 0;
  margin: 20px 0;
}

.prediction-item {
  display: flex;
  align-items: center;
  padding: 12px;
  border-bottom: 1px solid #eee;
  transition: background-color 0.2s;
}
```

```
.prediction-item:hover {
  background-color: #f8f9fa;
}

.prediction-item:last-child {
  border-bottom: none;
}

.line-name {
  font-weight: 600;
  margin-right: 8px;
}

.arrival-time {
  font-weight: bold;
  margin: 0 8px;
  padding: 4px 8px;
  border-radius: 4px;
}

.arrival-time.arriving {
  background-color: #dc3545;
  color: white;
}

.arrival-time.soon {
  background-color: #fd7e14;
  color: white;
}

.arrival-time.moderate {
  background-color: #ffc107;
  color: #333;
}

.arrival-time.later {
  background-color: #28a745;
  color: white;
}

.car-count {
  color: #666;
  font-size: 0.9em;
  margin-left: auto;
}

.last-updated {
  text-align: center;
  color: #666;
  font-size: 0.9em;
  margin-top: 16px;
  padding-top: 16px;
  border-top: 1px solid #eee;
}

.no-predictions {
  text-align: center;
  padding: 40px;
  color: #666;
  font-style: italic;
  background: #f8f9fa;
  border-radius: 8px;
  margin-top: 20px;
}

.error {
  background-color: #f8d7da;
  color: #721c24;
  padding: 12px;
  border-radius: 6px;
  margin: 16px 0;
  border: 1px solid #f5c6cb;
}

@media (max-width: 768px) {
  .metro-predictor {
    padding: 10px;
  }

  .prediction-item {
    flex-direction: column;
    align-items: flex-start;
    gap: 8px;
  }

  .car-count {
    margin-left: 0;
  }
}
</style>
```

Metro API Service (Updated for Your Axios Setup)

```
javascript
```

```
// vue-app/src/services/metroApi.js
import axios from 'axios'

// Use your existing axios instance
import api from '../api/index.js'

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 getLines() {
    return this.makeRequest('/metro/lines')
  }

  async getStationsForLine(lineCode) {
    return this.makeRequest(`/metro/stations/${lineCode}`)
  }

  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
    }
  }
}

export const metroApi = new MetroApiService()
```

Quick Start Commands (Vue)

```bash
# Navigate to Vue app
cd vue-app/

# Install dependencies (if not already done)
npm install

# Start Vue development server
npm run dev
```

🔗 API Communication Setup

CORS Configuration (Laravel with Sail)

Update `laravel-app/config/cors.php`:

```php
<?php

return [
    'paths' => ['api/*'],
    'allowed_methods' => ['*'],
    'allowed_origins' => [
        'http://localhost:5173',  // Vue dev server
        'http://127.0.0.1:5173',  // Alternative localhost
    ],
    'allowed_origins_patterns' => [],
    'allowed_headers' => ['*'],
    'exposed_headers' => [],
    'max_age' => 0,
    'supports_credentials' => true,  // Required for withCredentials: true
];
```

API Routes (Laravel)

Add to `laravel-app/routes/api.php`:

```php
<?php

use App\Http\Controllers\Api\MetroController;

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, 'getTrainPredicti
    Route::get('path/{lineCode}', [MetroController::class, 'getLinePath']);

    // Option 1: No auth (for development/admin CLI usage)
    Route::post('sync', [MetroController::class, 'syncData']);

    // Option 2: Add auth later when needed
    // Route::post('sync', [MetroController::class, 'syncData'])->middleware('auth');
});
```

✅ Integration Checklist

Phase 1: Backend Setup

- [ ] Copy all Laravel files to `laravel-app/` directory
- [ ] Add WMATA configuration to `laravel-app/.env`
- [ ] Create and run database migrations
- [ ] Register `WmataServiceProvider` in `config/app.php`
- [ ] Add API routes to `routes/api.php`
- [ ] Configure CORS for Vue app communication
- [ ] Test API endpoints with Postman/curl
- [ ] Run initial data sync: `php artisan metro:sync`

Phase 2: Frontend Setup

- [ ] Copy Vue component to `vue-app/src/components/`
- [ ] Create metro API service in `vue-app/src/services/`
- [ ] Add environment variables to `vue-app/.env`
- [ ] Import and use component in main Vue app
- [ ] Configure API base URL for Laravel backend

- [ ] Test frontend → backend communication

Phase 3: Integration Testing

- [ ] Test complete flow: Line selection → Station selection → Predictions
- [ ] Verify auto-refresh functionality (30-second interval)
- [ ] Test error handling (network failures, invalid selections)
- [ ] Verify station ordering (should be in proper sequence)
- [ ] Test responsive design on mobile devices

Phase 4: Production Readiness

- [ ] Set up environment-specific configuration
- [ ] Configure production CORS settings
- [ ] Set up SSL/HTTPS for production
- [ ] Configure caching strategies
- [ ] Set up monitoring and logging
- [ ] Create backup/restore procedures

🚀 Development Workflow

Daily Development (Sail)

```bash
# Terminal 1: Laravel API (Sail)
cd laravel-app/
sail up -d  # Start containers
# Laravel API available at: http://localhost

# Terminal 2: Vue Frontend
cd vue-app/
npm run dev
# Vue app available at: http://localhost:5173

# Terminal 3: Data sync (when needed)
cd laravel-app/
sail artisan metro:sync
```

Testing Communication (Sail)

```bash
# Test Laravel API directly (Sail)
curl http://localhost/api/metro/lines

# Access Vue app
# Visit: http://localhost:5173
```

Sync Metro Data (Sail)

```bash
cd laravel-app/
sail artisan metro:sync --validate  # Check cache integrity
sail artisan metro:sync            # Full sync including paths
```

🧪 Testing Strategy

Backend Testing (Sail)

```bash
cd laravel-app/

# Run feature tests
sail artisan test --filter MetroApiTest

# Test specific endpoints
sail artisan test tests/Feature/MetroControllerTest.php

# Access container shell for debugging
sail shell

# View logs
sail logs -f
```

Frontend Testing

```bash
cd vue-app/

# Run unit tests (if configured)
npm run test

# E2E testing with manual verification:
# 1. Select a line → verify stations load
# 2. Select a station → verify predictions load
# 3. Wait 30 seconds → verify auto-refresh
# 4. Change line → verify form resets properly
```

Integration Testing

- [ ] Verify API responses match expected format
- [ ] Test CORS functionality between apps
- [ ] Verify caching behavior (15s predictions, indefinite static data)
- [ ] Test rate limiting and error handling
- [ ] Verify station ordering matches physical metro layout

This structure makes it much cleaner to work with your separated Laravel and Vue applications! The integration focuses on API communication between the two apps rather than trying to bundle everything together.

Migration Files to Create

1. Create Lines Table

php

```php
// database/migrations/2025_01_01_000001_create_lines_table.php
use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

class CreateLinesTable extends Migration
{
    public function up()
    {
        Schema::create('lines', function (Blueprint $table) {
            $table->string('line_code', 2)->primary();
            $table->string('display_name', 50);
            $table->string('start_station_code', 3);
            $table->string('end_station_code', 3);
            $table->string('internal_destination_1', 3)->nullable();
            $table->string('internal_destination_2', 3)->nullable();
            $table->timestamps();

            $table->index('display_name');
        });
    }

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

2. Create Stations Table

php

```php
// database/migrations/2025_01_01_000002_create_stations_table.php
class CreateStationsTable extends Migration
{
    public function up()
    {
        Schema::create('stations', function (Blueprint $table) {
            $table->string('code', 3)->primary();
            $table->string('name', 100);
            $table->string('line_code_1', 2)->nullable();
            $table->string('line_code_2', 2)->nullable();
            $table->string('line_code_3', 2)->nullable();
            $table->string('line_code_4', 2)->nullable();
            $table->string('station_together_1', 3)->nullable();
            $table->string('station_together_2', 3)->nullable();
            $table->decimal('lat', 10, 8);
            $table->decimal('lon', 11, 8);
            $table->boolean('is_active')->default(true);
            $table->timestamps();

            $table->index('name');
            $table->index(['lat', 'lon']);
            $table->index('is_active');
        });
    }
}
```

3. Create Station Addresses Table

php

```php
// database/migrations/2025_01_01_000003_create_station_addresses_table.php
class CreateStationAddressesTable extends Migration
{
    public function up()
    {
        Schema::create('station_addresses', function (Blueprint $table) {
            $table->string('station_code', 3)->primary();
            $table->string('street', 255);
            $table->string('city', 100);
            $table->string('state', 2);
            $table->string('zip_code', 10);
            $table->string('country', 2)->default('US');
            $table->timestamps();

            $table->foreign('station_code')->references('code')->on('stations')->onDel
            $table->index(['city', 'state']);
            $table->index('zip_code');
        });
    }
}
```

4. Create Station Paths Table

php

```php
// database/migrations/2025_01_01_000004_create_station_paths_table.php
class CreateStationPathsTable extends Migration
{
    public function up()
    {
        Schema::create('station_paths', function (Blueprint $table) {
            $table->id();
            $table->string('line_code', 2);
            $table->string('station_code', 3);
            $table->string('station_name', 100);
            $table->integer('seq_num');
            $table->integer('distance_to_prev')->default(0);
            $table->timestamps();

            $table->index(['line_code', 'seq_num']);
            $table->index('station_code');
            $table->unique(['line_code', 'station_code']);

            $table->foreign('line_code')->references('line_code')->on('lines');
            $table->foreign('station_code')->references('code')->on('stations');
        });
    }
}
```

5. Add Foreign Keys to Lines Table

php

```php
// database/migrations/2025_01_01_000005_add_foreign_keys_to_lines_table.php
class AddForeignKeysToLinesTable extends Migration
{
    public function up()
    {
        Schema::table('lines', function (Blueprint $table) {
            $table->foreign('start_station_code')->references('code')->on('stations');
            $table->foreign('end_station_code')->references('code')->on('stations');
            $table->foreign('internal_destination_1')->references('code')->on('station
            $table->foreign('internal_destination_2')->references('code')->on('station
        });
    }
}
```

🔧 Backend Implementation

Models

Line Model

php

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

namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\BelongsTo;
use Illuminate\Database\Eloquent\Relations\HasMany;

class Line extends Model
{
    protected $primaryKey = 'line_code';
    public $incrementing = false;
    protected $keyType = 'string';

    protected $fillable = [
        'line_code',
        'display_name',
        'start_station_code',
        'end_station_code',
        'internal_destination_1',
        'internal_destination_2',
    ];

    public function startStation(): BelongsTo
    {
        return $this->belongsTo(Station::class, 'start_station_code', 'code');
    }

    public function endStation(): BelongsTo
    {
        return $this->belongsTo(Station::class, 'end_station_code', 'code');
    }

    public function stationPaths(): HasMany
    {
        return $this->hasMany(StationPath::class, 'line_code', 'line_code')->orderBy('
    }

    public function toSelectOption(): array
    {
        return [
            'value' => $this->line_code,
            'label' => $this->display_name,
        ];
    }
}
```

Station Model

php

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

namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\HasOne;
use Illuminate\Database\Eloquent\Casts\Attribute;

class Station extends Model
{
    protected $primaryKey = 'code';
    public $incrementing = false;
    protected $keyType = 'string';

    protected $fillable = [
        'code', 'name', 'line_code_1', 'line_code_2', 'line_code_3', 'line_code_4',
        'station_together_1', 'station_together_2', 'lat', 'lon', 'is_active',
    ];

    protected function casts(): array
    {
        return [
            'lat' => 'decimal:8',
            'lon' => 'decimal:8',
            'is_active' => 'boolean',
        ];
    }

    public function address(): HasOne
    {
        return $this->hasOne(StationAddress::class, 'station_code', 'code');
    }

    public function getLineCodes(): array
    {
        return array_filter([
            $this->line_code_1,
            $this->line_code_2,
            $this->line_code_3,
            $this->line_code_4,
        ]);
    }
    protected function coordinates(): Attribute
    {
        return Attribute::make(
            get: fn() => [
                'lat' => (float) $this->lat,
                'lng' => (float) $this->lon,
            ]
        );
    }

    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);
        });
    }

    public function toSelectOption(): array
    {
        return [
            'value' => $this->code,
            'label' => $this->name,
        ];
    }
}
```

StationAddress Model

php

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

namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\BelongsTo;

class StationAddress extends Model
{
    protected $primaryKey = 'station_code';
    public $incrementing = false;
    protected $keyType = 'string';

    protected $fillable = [
        'station_code', 'street', 'city', 'state', 'zip_code', 'country'
    ];

    public function station(): BelongsTo
    {
        return $this->belongsTo(Station::class, 'station_code', 'code');
    }

    public function getFormattedAttribute(): string
    {
        return "{$this->street}, {$this->city}, {$this->state} {$this->zip_code}";
    }
}
```

StationPath Model

php

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

namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\BelongsTo;

class StationPath extends Model
{
    protected $fillable = [
        'line_code', 'station_code', 'station_name', 'seq_num', 'distance_to_prev',
    ];

    protected function casts(): array
    {
        return [
            'seq_num' => 'integer',
            'distance_to_prev' => 'integer',
        ];
    }

    public function line(): BelongsTo
    {
        return $this->belongsTo(Line::class, 'line_code', 'line_code');
    }

    public function station(): BelongsTo
    {
        return $this->belongsTo(Station::class, 'station_code', 'code');
    }

    public function scopeForLine($query, string $lineCode)
    {
        return $query->where('line_code', $lineCode);
    }

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

DTOs (Data Transfer Objects)

LineDto

php

```php
// app/DTOs/LineDto.php
<?php

namespace App\DTOs;

class LineDto
{
    public function __construct(
        public string $displayName,
        public string $lineCode,
        public string $startStationCode,
        public string $endStationCode,
        public ?string $internalDestination1 = null,
        public ?string $internalDestination2 = null,
    ) {}

    public static function fromArray(array $data): self
    {
        return new self(
            displayName: $data['DisplayName'],
            lineCode: $data['LineCode'],
            startStationCode: $data['StartStationCode'],
            endStationCode: $data['EndStationCode'],
            internalDestination1: !empty($data['InternalDestination1']) ? $data['Inter
            internalDestination2: !empty($data['InternalDestination2']) ? $data['Inter
        );
    }

    public function toModel(): array
    {
        return [
            'line_code' => $this->lineCode,
            'display_name' => $this->displayName,
            'start_station_code' => $this->startStationCode,
            'end_station_code' => $this->endStationCode,
            'internal_destination_1' => $this->internalDestination1,
            'internal_destination_2' => $this->internalDestination2,
        ];
    }
}
```

StationDto

php

```php
// app/DTOs/StationDto.php
<?php

namespace App\DTOs;

class StationDto
{
    public function __construct(
        public string $code,
        public string $name,
        public ?string $stationTogether1,
        public ?string $stationTogether2,
        public ?string $lineCode1,
        public ?string $lineCode2,
        public ?string $lineCode3,
        public ?string $lineCode4,
        public float $lat,
        public float $lon,
        public AddressDto $address,
    ) {}

    public static function fromArray(array $data): self
    {
        return new self(
            code: $data['code'],
            name: $data['name'],
            stationTogether1: !empty($data['stationTogether1']) ? $data['stationTogeth
            stationTogether2: !empty($data['stationTogether2']) ? $data['stationTogeth
            lineCode1: $data['lineCode1'] ?? null,
            lineCode2: $data['lineCode2'] ?? null,
            lineCode3: $data['lineCode3'] ?? null,
            lineCode4: $data['lineCode4'] ?? null,
            lat: (float) $data['lat'],
            lon: (float) $data['lon'],
            address: AddressDto::fromArray($data['address']),
        );
    }

    public function toModel(): array
    {
        return [
            'code' => $this->code,
            'name' => $this->name,
            'station_together_1' => $this->stationTogether1,
            'station_together_2' => $this->stationTogether2,
            'line_code_1' => $this->lineCode1,
            'line_code_2' => $this->lineCode2,
            'line_code_3' => $this->lineCode3,
            'line_code_4' => $this->lineCode4,
            'lat' => $this->lat,
            'lon' => $this->lon,
        ];
    }
}
```

AddressDto

```php
// app/DTOs/AddressDto.php
<?php

namespace App\DTOs;

class AddressDto
{
    public function __construct(
        public string $street,
        public string $city,
        public string $state,
        public string $zip,
    ) {}

    public static function fromArray(array $data): self
    {
        return new self(
            street: $data['Street'],
            city: $data['City'],
            state: $data['State'],
            zip: $data['Zip'],
        );
    }
}
```

TrainPredictionDto

php

```php
// app/DTOs/TrainPredictionDto.php
<?php

namespace App\DTOs;

class TrainPredictionDto
{
    public function __construct(
        public string $car,
        public string $destination,
        public ?string $destinationCode,
        public string $destinationName,
        public string $group,
        public string $line,
        public string $locationCode,
        public string $locationName,
        public string $min,
    ) {}

    public static function fromArray(array $data): self
    {
        return new self(
            car: $data['Car'],
            destination: $data['Destination'],
            destinationCode: $data['DestinationCode'],
            destinationName: $data['DestinationName'],
            group: $data['Group'],
            line: $data['Line'],
            locationCode: $data['LocationCode'],
            locationName: $data['LocationName'],
            min: $data['Min'],
        );
    }

    public function toFrontend(): array
    {
        return [
            'line' => $this->line,
            'destination' => $this->destinationName,
            'minutes' => $this->min,
            'cars' => $this->car,
            'group' => $this->group,
        ];
```

```
      }
  }
```

StationPathDto

php

```php
// app/DTOs/StationPathDto.php
<?php

namespace App\DTOs;

class StationPathDto
{
    public function __construct(
        public string $lineCode,
        public string $stationCode,
        public string $stationName,
        public int $seqNum,
        public int $distanceToPrev,
    ) {}

    public static function fromArray(array $data): self
    {
        return new self(
            lineCode: $data['LineCode'],
            stationCode: $data['StationCode'],
            stationName: $data['StationName'],
            seqNum: (int) $data['SeqNum'],
            distanceToPrev: (int) $data['DistanceToPrev'],
        );
    }

    public function toModel(): array
    {
        return [
            'line_code' => $this->lineCode,
            'station_code' => $this->stationCode,
            'station_name' => $this->stationName,
            'seq_num' => $this->seqNum,
            'distance_to_prev' => $this->distanceToPrev,
        ];
    }
}
```

Services

WmataApiService

php

```php
// app/Services/WmataApiService.php
<?php

namespace App\Services;

use App\DTOs\LineDto;
use App\DTOs\StationDto;
use App\DTOs\TrainPredictionDto;
use App\DTOs\StationPathDto;
use App\Models\Line;
use Illuminate\Support\Facades\Http;
use Illuminate\Support\Facades\Log;
use Illuminate\Support\Facades\Cache;

class WmataApiService
{
    private const RATE_LIMIT_KEY = 'wmata_api_rate_limit';

    public function __construct(
        private string $apiKey,
        private string $baseUrl,
        private array $endpoints,
        private array $cacheConfig,
        private int $maxRequestsPerHour
    ) {}

    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'] ?? []
            );
        });
    }

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

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

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

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

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

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

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

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

    public function getStationPath(string $fromStationCode, string $toStationCode): ar
    {
        $cacheKey = "wmata.path.{$fromStationCode}.{$toStationCode}";

        return Cache::remember($cacheKey, $this->cacheConfig['paths_ttl'], function ()
            $endpoint = $this->endpoints['path'] . "?FromStationCode={$fromStationCode
            $response = $this->makeRequest($endpoint);

            return array_map(
                fn($pathItem) => StationPathDto::fromArray($pathItem),
                $response['Path'] ?? []
            );
        });
    }

    public function getLineCompletePath(string $lineCode): array
    {
        $line = Line::where('line_code', $lineCode)->first();

        if (!$line) {
            throw new \Exception("Line {$lineCode} not found");
        }

        return $this->getStationPath($line->start_station_code, $line->end_station_cod
    }

    private function extractFirstStationCode(string $stationCodes): string
    {
        $codes = explode(',', $stationCodes);
        return trim($codes[0]);
    }

    private function makeRequest(string $endpoint): array
    {
        if (!$this->checkRateLimit()) {
            throw new \Exception('API rate limit exceeded. Please try again later.');
        }

        try {
            $url = $this->baseUrl . $endpoint;

            Log::info('WMATA API Request', ['url' => $url]);

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

            if ($response->failed()) {
                Log::error('WMATA API request failed', [
                    'endpoint' => $endpoint,
                    'status' => $response->status(),
                    'body' => $response->body(),
                ]);

                throw new \Exception("API request failed with status: {$response->stat
            }

            $this->incrementRateLimit();
            return $response->json();

        } catch (\Exception $e) {
            Log::error('WMATA API error', [
                'endpoint' => $endpoint,
                'error' => $e->getMessage(),
            ]);

            throw $e;
        }
    }

    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);
    }
}
```

php

```php
// app/Services/MetroDataService.php
<?php

namespace App\Services;

use App\Models\Line;
use App\Models\Station;
use App\Models\StationAddress;
use App\Models\StationPath;
use Illuminate\Support\Facades\Cache;
use Illuminate\Support\Facades\Log;

class MetroDataService
{
    public function __construct(
        private WmataApiService $wmataApi
    ) {}

    public function syncAllMetroData(): array
    {
        $results = ['lines' => 0, 'stations' => 0, 'paths' => 0, 'errors' => []];

        try {
            // Sync lines first
            $lines = $this->wmataApi->getLines();
            foreach ($lines as $lineDto) {
                Line::updateOrCreate(
                    ['line_code' => $lineDto->lineCode],
                    $lineDto->toModel()
                );
                $results['lines']++;
            }

            // Sync all stations
            $allStations = $this->wmataApi->getAllStations();
            foreach ($allStations as $stationDto) {
                $station = Station::updateOrCreate(
                    ['code' => $stationDto->code],
                    $stationDto->toModel()
                );

                StationAddress::updateOrCreate(
                    ['station_code' => $station->code],
                    [
                        'street' => $stationDto->address->street,
                        'city' => $stationDto->address->city,
                        'state' => $stationDto->address->state,
                        'zip_code' => $stationDto->address->zip,
                    ]
                );
                $results['stations']++;
            }

            // Sync station paths for proper ordering
            $this->syncStationPaths($results);

        } catch (\Exception $e) {
            $results['errors'][] = $e->getMessage();
        }

        return $results;
    }

    private function syncStationPaths(array &$results): void
    {
        $lines = Line::all();

        foreach ($lines as $line) {
            try {
                $pathData = $this->wmataApi->getLineCompletePath($line->line_code);

                // Clear existing path data for this line
                StationPath::where('line_code', $line->line_code)->delete();

                // Insert new path data
                foreach ($pathData as $pathDto) {
                    StationPath::create($pathDto->toModel());
                    $results['paths']++;
                }

            } catch (\Exception $e) {
                $results['errors'][] = "Failed to sync path for line {$line->line_code
            }
        }
    }

    public function getOrderedStationsForLine(string $lineCode): array
```

```php
    {
        $cacheKey = "metro.stations.ordered.{$lineCode}";

        return Cache::remember($cacheKey, 3600, function () use ($lineCode) {
            $orderedPaths = StationPath::forLine($lineCode)->ordered()->get();

            if ($orderedPaths->isEmpty()) {
                Log::warning("No path data found for line {$lineCode}, using unordered
                return $this->getCachedStationsForLine($lineCode);
            }

            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();
        });
    }

    public function getCachedStationsForLine(string $lineCode): array
    {
        $cacheKey = "metro.stations.frontend.{$lineCode}";

        return Cache::remember($cacheKey, 3600, function () use ($lineCode) {
            $stationDtos = $this->wmataApi->getStationsForLine($lineCode);

            return array_map(function ($stationDto) {
                return [
                    'value' => $stationDto->code,
                    'label' => $stationDto->name,
                ];
            }, $stationDtos);
        });
    }

    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();
        });
    }

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

        return $hasLines && $has
```