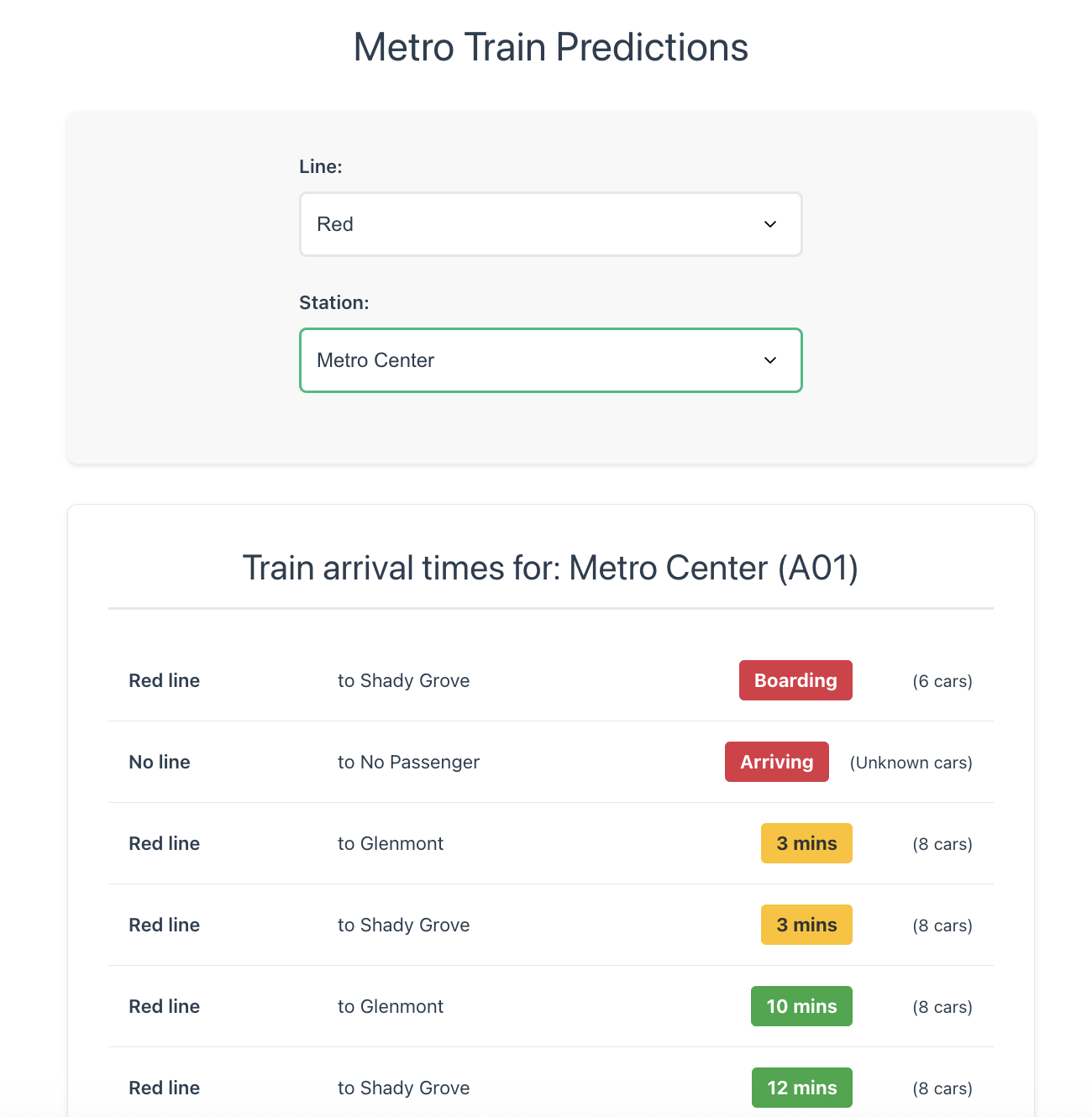
Metro Train Prediction App - Complete API Documentation

Author: David Morrison

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



## Table of Contents

[Table of Contents 1](#_Toc802204159)

[API Overview 1](#_Toc52477320)

[Base URL 1](#_Toc1321241684)

[API Design Principles 1](#_Toc381256228)

[Supported HTTP Methods 1](#_Toc181365585)

[Authentication & Rate Limiting 1](#_Toc846899374)

[Authentication 1](#_Toc1932814217)

[Rate Limiting 1](#_Toc1003157868)

[Rate Limit Headers 1](#_Toc160707937)

[Response Format 1](#_Toc1680997362)

[Standard Success Response 1](#_Toc66154680)

[Standard Error Response 1](#_Toc1561281131)

[HTTP Status Codes 1](#_Toc1546820273)

[Endpoints 1](#_Toc661263751)

[1. Get Metro Lines 1](#_Toc591069263)

[2. Get Stations for Line 1](#_Toc248418182)

[3. Get Train Predictions 1](#_Toc1877695225)

[4. Sync Metro Data (Administrative) 1](#_Toc1482796101)

[Error Handling 1](#_Toc576297420)

[Error Response Structure 1](#_Toc171609519)

[Common Error Scenarios 1](#_Toc1722372143)

[1. Invalid Line Code 1](#_Toc512429626)

[2. Station Not Found 1](#_Toc1952059325)

[3. Rate Limit Exceeded 1](#_Toc364692644)

[4. WMATA API Issues 1](#_Toc1883642075)

[5. Cache/Database Issues 1](#_Toc398895601)

[Data Models 1](#_Toc2084723367)

[Line Model 1](#_Toc1170754628)

[Station Model 1](#_Toc27158280)

[Prediction Model 1](#_Toc962321367)

[Station Info Model 1](#_Toc2105668805)

[Sync Results Model 1](#_Toc296992949)

[Usage Examples 1](#_Toc1760889126)

[Frontend Integration Flow 1](#_Toc771587782)

[Complete Vue.js Integration 1](#_Toc1621611787)

[Error Handling Example 1](#_Toc1709645150)

[SDK Integration 1](#_Toc1029946621)

[Vue 3 Integration 1](#_Toc1189394522)

[Node.js Integration 1](#_Toc1316302433)

[Python Integration 1](#_Toc1089108169)

[Performance & Caching 1](#_Toc1344390565)

[Caching Strategy 1](#_Toc113706368)

[Performance Characteristics 1](#_Toc1503926229)

[Optimization Tips 1](#_Toc1587065978)

## API Overview

### Base URL

<http://localhost/api> # Development (Laravel Sail)

<https://yourapp.com/api> # Production

### API Design Principles

* **RESTful Architecture** - Standard HTTP methods and status codes
* **JSON-First** - All requests and responses use JSON format
* **Consistent Structure** - Standardized response format across all endpoints
* **Progressive Enhancement** - Form-based workflow (Lines → Stations → Predictions)
* **Real-time Caching** - Smart caching strategy for different data types

### Supported HTTP Methods

* GET - Retrieve data (lines, stations, predictions)
* POST - Administrative operations (data synchronization)

## Authentication & Rate Limiting

### Authentication

Currently **no authentication required** for public endpoints. Future versions may implement:

* API keys for third-party integrations
* Laravel Sanctum for authenticated users

### Rate Limiting

* **General API**: 60 requests per minute per IP address
* **Metro Endpoints**: Applied via throttle:60,1 middleware
* **WMATA Backend**: 1000 requests per hour (configurable)

### Rate Limit Headers

http

X-RateLimit-Limit: 60

X-RateLimit-Remaining: 59

X-RateLimit-Reset: 1672531200

## Response Format

### Standard Success Response

json

{

"success": true,

"data": {

*// Response data varies by endpoint*

},

"meta": {

*// Optional metadata (pagination, totals, etc.)*

}

}

### Standard Error Response

json

{

"success": false,

"error": "Descriptive error message",

"code": "ERROR\_CODE" *// Optional error code*

}

### HTTP Status Codes

|  |  |  |
| --- | --- | --- |
| **Code** | **Meaning** | **Usage** |
| 200 | OK | Successful request |
| 400 | Bad Request | Invalid parameters (e.g., invalid line code) |
| 404 | Not Found | Resource not found (e.g., station doesn't exist) |
| 429 | Too Many Requests | Rate limit exceeded |
| 500 | Internal Server Error | Server-side error |

## Endpoints

### 1. Get Metro Lines

Retrieve all available metro lines for selection.

**Endpoint**

GET /api/metro/lines

**Parameters** None

**Response**

json

{

"success": true,

"data": [

{

"value": "RD",

"label": "Red"

},

{

"value": "BL",

"label": "Blue"

},

{

"value": "GR",

"label": "Green"

},

{

"value": "OR",

"label": "Orange"

},

{

"value": "SV",

"label": "Silver"

},

{

"value": "YL",

"label": "Yellow"

}

]

}

**Caching**

* **Cache Duration**: 1 hour
* **Cache Key**: metro.lines.frontend
* **Auto-refresh**: If cache is empty, triggers full data sync

**Example Request**

bash

curl -X GET "<http://localhost/api/metro/lines>" \

-H "Accept: application/json"

### 2. Get Stations for Line

Retrieve all stations for a specific metro line in geographic order.

**Endpoint**

GET /api/metro/stations/{lineCode}

**Parameters**

|  |  |  |  |
| --- | --- | --- | --- |
| **Parameter** | **Type** | **Required** | **Description** |
| lineCode | string | ✅ | Metro line code (RD, BL, GR, OR, SV, YL) |

**Response**

json

{

"success": true,

"data": [

{

"value": "A15",

"label": "Shady Grove",

"seq\_num": 1,

"distance\_to\_prev": 0

},

{

"value": "A14",

"label": "Rockville",

"seq\_num": 2,

"distance\_to\_prev": 4823

},

{

"value": "A13",

"label": "Twinbrook",

"seq\_num": 3,

"distance\_to\_prev": 2134

}

],

"meta": {

"line\_code": "RD",

"total\_stations": 27,

"ordered": true

}

}

**Response Fields**

* value: Station code for API calls
* label: Human-readable station name
* seq\_num: Geographic sequence along the line
* distance\_to\_prev: Distance to previous station (meters)

**Error Responses**

json

*// Invalid line code*

{

"success": false,

"error": "Invalid line code"

}

**Caching**

* **Cache Duration**: 1 hour
* **Cache Key**: metro.stations.ordered.{lineCode}
* **Fallback**: Unordered stations if path data unavailable

**Example Requests**

bash

*# Get Red Line stations*

curl -X GET "<http://localhost/api/metro/stations/RD>" \

-H "Accept: application/json"

*# Get Blue Line stations*

curl -X GET "<http://localhost/api/metro/stations/BL>" \

-H "Accept: application/json"

### 3. Get Train Predictions

Retrieve real-time train arrival predictions for a specific station.

**Endpoint**

GET /api/metro/predictions/{stationCode}

**Parameters**

|  |  |  |  |
| --- | --- | --- | --- |
| **Parameter** | **Type** | **Required** | **Description** |
| stationCode | string | ✅ | Station code (e.g., A01, B02, C03) |

**Response**

json

{

"success": true,

"data": {

"station": {

"code": "A01",

"name": "Metro Center"

},

"predictions": [

{

"line": "RD",

"destination": "Glenmont",

"minutes": "3",

"cars": "6",

"group": "1"

},

{

"line": "RD",

"destination": "Shady Grove",

"minutes": "8",

"cars": "8",

"group": "2"

},

{

"line": "BL",

"destination": "Franconia-Springfield",

"minutes": "BRD",

"cars": "6",

"group": "1"

},

{

"line": "OR",

"destination": "Vienna",

"minutes": "ARR",

"cars": "Unknown",

"group": "1"

}

],

"updated\_at": "2025-06-05T14:30:15.123Z",

"refresh\_interval": 30

}

}

**Prediction Fields**

* line: Metro line code (RD, BL, GR, OR, SV, YL)
* destination: Final destination station name
* minutes: Arrival time
  + Number (e.g., "3", "15"): Minutes until arrival
  + "BRD": Train is boarding
  + "ARR": Train is arriving
* cars: Number of cars or "Unknown" for end-of-line stations
* group: Train grouping (1 or 2, defaults to "1")

**Error Responses**

json

*// Station not found*

{

"success": false,

"error": "Station not found"

}

*// API error*

{

"success": false,

"error": "Failed to get predictions: API request failed with status: 500"

}

**Caching**

* **Cache Duration**: 15 seconds
* **Cache Key**: wmata.predictions.{stationCode}
* **Auto-refresh**: Frontend refreshes every 30 seconds

**Example Requests**

bash

*# Metro Center predictions*

curl -X GET "<http://localhost/api/metro/predictions/A01>" \

-H "Accept: application/json"

*# Union Station predictions*

curl -X GET "<http://localhost/api/metro/predictions/B03>" \

-H "Accept: application/json"

*# End-of-line station (may have "Unknown" cars)*

curl -X GET "<http://localhost/api/metro/predictions/A15>" \

-H "Accept: application/json"

### 4. Sync Metro Data (Administrative)

Synchronize all metro data from WMATA API. Administrative endpoint for data updates.

**Endpoint**

POST /api/metro/sync

**Parameters** None

**Response**

json

{

"success": true,

"message": "All Metro data synchronized successfully",

"results": {

"lines": 6,

"stations": 95,

"paths": 95,

"errors": []

}

}

**Error Response**

json

{

"success": false,

"error": "Sync failed: API request failed with status: 401",

"results": {

"lines": 6,

"stations": 0,

"paths": 0,

"errors": [

"Failed to sync path for line RD: API request failed with status: 401"

]

}

}

**Operations Performed**

1. Sync all metro lines from WMATA
2. Sync all stations with addresses
3. Generate station path sequences for ordering
4. Clear and rebuild caches

**Usage**

* **Development**: Manual sync as needed
* **Production**: Automated daily sync via cron
* **CLI Alternative**: php artisan metro:sync

**Example Request**

curl -X POST "<http://localhost/api/metro/sync>" \

-H "Accept: application/json" \

-H "Content-Type: application/json"

## Error Handling

### Error Response Structure

json

{

"success": false,

"error": "Human-readable error message",

"details": { *// Optional, for debugging*

"endpoint": "/api/metro/predictions/INVALID",

"timestamp": "2025-06-05T14:30:15Z"

}

}

### Common Error Scenarios

#### **1. Invalid Line Code**

GET /api/metro/stations/INVALID

json

{

"success": false,

"error": "Invalid line code"

}

#### 2. Station Not Found

GET /api/metro/predictions/INVALID

json

{

"success": false,

"error": "Station not found"

}

#### 3. Rate Limit Exceeded

json

{

"success": false,

"error": "Too many requests. Please try again later.",

"retry\_after": 60

}

#### 4. WMATA API Issues

json

{

"success": false,

"error": "Failed to load predictions: API request failed with status: 500"

}

#### 5. Cache/Database Issues

json

{

"success": false,

"error": "Failed to load lines: Database connection failed"

}

## Data Models

### Line Model

json

{

"value": "RD", *// 2-character line code*

"label": "Red" *// Human-readable name*

}

### Station Model

json

{

"value": "A01", *// 3-character station code*

"label": "Metro Center", *// Station name*

"seq\_num": 5, *// Geographic sequence (optional)*

"distance\_to\_prev": 1250 *// Distance in meters (optional)*

}

### Prediction Model

json

{

"line": "RD", *// Line code*

"destination": "Glenmont", *// Destination name*

"minutes": "3", *// "3", "15", "BRD", "ARR"*

"cars": "6", *// Car count or "Unknown"*

"group": "1" *// Train group (1 or 2)*

}

### Station Info Model

json

{

"code": "A01", *// Station code*

"name": "Metro Center" *// Station name*

}

### Sync Results Model

json

{

"lines": 6, *// Number of lines synced*

"stations": 95, *// Number of stations synced*

"paths": 95, *// Number of path entries synced*

"errors": [] *// Array of error messages*

}

## Usage Examples

### Frontend Integration Flow

javascript

*// 1. Load available lines*

const lines = await fetch('/api/metro/lines')

.then(r => r.json())

.then(data => data.data);

*// 2. User selects Red Line, load stations*

const stations = await fetch('/api/metro/stations/RD')

.then(r => r.json())

.then(data => data.data);

*// 3. User selects Metro Center, get predictions*

const predictions = await fetch('/api/metro/predictions/A01')

.then(r => r.json())

.then(data => data.data);

*// 4. Auto-refresh predictions every 30 seconds*

setInterval(async () => {

const updated = await fetch('/api/metro/predictions/A01')

.then(r => r.json())

.then(data => data.data);

updateUI(updated);

}, 30000);

### Complete Vue.js Integration

vue

<script setup>

import { ref, onMounted } from 'vue'

const selectedLine = ref('')

const selectedStation = ref('')

const lines = ref([])

const stations = ref([])

const predictions = ref([])

// Load lines on component mount

onMounted(async () => {

const response = await fetch('/api/metro/lines')

const data = await response.json()

lines.value = data.data

})

// Load stations when line changes

const onLineChange = async () => {

if (!selectedLine.value) return

const response = await fetch(`/api/metro/stations/${selectedLine.value}`)

const data = await response.json()

stations.value = data.data

selectedStation.value = ''

predictions.value = []

}

// Load predictions when station changes

const onStationChange = async () => {

if (!selectedStation.value) return

const response = await fetch(`/api/metro/predictions/${selectedStation.value}`)

const data = await response.json()

predictions.value = data.data.predictions

}

</script>

### Error Handling Example

javascript

async function getPredictions(stationCode) {

try {

const response = await fetch(`/api/metro/predictions/${stationCode}`)

const data = await response.json()

if (!data.success) {

throw new Error(data.error)

}

return data.data

} catch (error) {

console.error('Failed to get predictions:', error.message)

*// Handle specific error cases*

if (error.message.includes('Station not found')) {

showUserMessage('Station not found. Please select a different station.')

} else if (error.message.includes('Too many requests')) {

showUserMessage('Please wait a moment before refreshing.')

} else {

showUserMessage('Unable to load train predictions. Please try again.')

}

return null

}

}

## SDK Integration

### Vue 3 Integration

bash

*# Install axios for HTTP requests (if not already installed)*

npm install axios

javascript

*// services/metroApi.js (matches your actual implementation)*

import axios from 'axios'

import api from '../api/index.js' *// Your existing axios instance*

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) {

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

}

}

export const metroApi = new MetroApiService()

### Node.js Integration

javascript

*// server.js*

const express = require('express')

const axios = require('axios')

const app = express()

const METRO\_API\_BASE = '<http://localhost/api>'

app.get('/proxy/metro/predictions/:stationCode', async (req, res) => {

try {

const response = await axios.get(

`${METRO\_API\_BASE}/metro/predictions/${req.params.stationCode}`

)

res.json(response.data)

} catch (error) {

res.status(500).json({

success: false,

error: 'Failed to fetch predictions'

})

}

})

### Python Integration

python

import requests

import json

class MetroAPI:

def \_\_init\_\_(self, base\_url="http://localhost/api"):

self.base\_url = base\_url

def get\_lines(self):

response = requests.get(f"{self.base\_url}/metro/lines")

data = response.json()

return data['data'] if data['success'] else None

def get\_stations\_for\_line(self, line\_code):

response = requests.get(f"{self.base\_url}/metro/stations/{line\_code}")

data = response.json()

return data['data'] if data['success'] else None

def get\_predictions(self, station\_code):

response = requests.get(f"{self.base\_url}/metro/predictions/{station\_code}")

data = response.json()

return data['data'] if data['success'] else None

*# Usage*

metro = MetroAPI()

lines = metro.get\_lines()

red\_line\_stations = metro.get\_stations\_for\_line('RD')

metro\_center\_predictions = metro.get\_predictions('A01')

## Performance & Caching

### Caching Strategy

* **Static Data** (Lines, Stations): 1 hour cache
* **Dynamic Data** (Predictions): 15 second cache
* **Path Data**: 1 hour cache
* **Frontend Auto-refresh**: 30 seconds

### Performance Characteristics

* **Lines endpoint**: ~10ms (cached)
* **Stations endpoint**: ~20ms (cached, ordered)
* **Predictions endpoint**: ~100ms (WMATA API call or cache)
* **Sync endpoint**: ~30 seconds (full data refresh)

### Optimization Tips

1. **Cache predictions locally** for 15 seconds minimum
2. **Batch station requests** when possible
3. **Implement request deduplication** for rapid user interactions
4. **Use WebSockets** for real-time updates in advanced implementations

This API provides a robust, well-documented interface for building Metro transit applications with real-time predictions and proper error handling.