Metro Train Prediction App - Service Overview (updates 6/5/25)

Author: David Morrison

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

This document contains updates made after the initial version of this Metro Train Prediction App - Service Overview document was produced.

System Architecture

High-Level Architecture

```
┌──────────────────┐    ┌──────────────────┐    ┌──────────────────┐
│   Vue Frontend   │    │ Laravel Backend  │    │    WMATA API     │
│                  │    │                  │    │                  │
│ MetroTrainPred   │◄──►│ MetroController  │◄──►│ Rail Services    │
│ LineSelector     │    │                  │    │ Predictions      │
│ StationSelector  │    │  ┌────────────┐  │    │ Station Data     │
│ PredictionList   │    │  │MetroDataSvc│  │    │ Path Data        │
│                  │    │  │            │  │    │                  │
│ metroApi.js      │    │  │WmataApiSvc │  │    │                  │
└──────────────────┘    └──────────────────┘    └──────────────────┘
```

Service Layer Architecture

The application follows a layered architecture pattern:

1. **Controller Layer** (`MetroController`)
   a. Handles HTTP requests
   b. Manages response formatting
   c. Implements error handling
   d. Coordinates between services
2. **Service Layer**
   a. `MetroDataService`: Business logic and data management
   b. `WmataApiService`: WMATA API integration and caching
3. **Data Layer**
   a. Models: `Line`, `Station`, `StationPath`, `StationAddress`
   b. DTOs: `LineDto`, `StationDto`, `TrainPredictionDto`, `StationPathDto`

Service Layer Implementation

WmataApiService

```
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
    ) {}
```

Key Methods

1. Line Management

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

2. Station Management

```
public function getAllStations(): array
{
    $cacheKey = 'wmata.stations.all';
    return Cache::remember($cacheKey, $this->cacheConfig['stations_ttl'], function () {
        // Get the base stations list first
        $response = $this->makeRequest($this->endpoints['stations']);
        $allStations = [];
        $seenStations = [];

        // Process base station list and handle transfer stations
        foreach ($response['Stations'] ?? [] as $stationData) {
            $station = StationDto::fromArray($stationData);
            $allStations[] = $station;
            $seenStations[$station->code] = true;

            // Handle transfer stations
            if ($station->stationTogether1) {
                $seenStations[$station->stationTogether1] = false;
            }
            if ($station->stationTogether2) {
                $seenStations[$station->stationTogether2] = false;
            }
        }

        // Process line-specific stations
        $lines = $this->getLines();
        foreach ($lines as $line) {
            $lineStations = $this->getStationsForLine($line->lineCode);
            foreach ($lineStations as $station) {
                if (!isset($seenStations[$station->code]) || $seenStations[$station-
>code] === false) {
                    $allStations[] = $station;
                    $seenStations[$station->code] = true;
                }
            }
        }

        return $allStations;
    });
}
```

3. Path Management

```
public function getLineCompletePath(string $lineCode): array
{
    // Get all stations for this line
    $stations = $this->getStationsForLine($lineCode);

    if (empty($stations)) {
        throw new \Exception("No stations found for line {$lineCode}");
    }

    // Convert stations to path DTOs with sequence numbers and distances
    $pathDtos = [];
    $prevDistance = 0;

    foreach ($stations as $index => $station) {
        $pathDtos[] = new StationPathDto(
            lineCode: $lineCode,
            stationCode: $station->code,
            stationName: $station->name,
            seqNum: $index + 1,
            distanceToPrev: $index === 0 ? 0 : $prevDistance
        );

        // Calculate distance to next station
        if (isset($stations[$index + 1])) {
            $nextStation = $stations[$index + 1];
            $prevDistance = $this->calculateDistance(
                $station->lat,
                $station->lon,
                $nextStation->lat,
                $nextStation->lon
            );
        }
    }
```

```
        return $pathDtos;
    }
```

MetroDataService

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

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

        try {
            // Sync lines
            $this->syncLines($results);

            // Sync stations
            $this->syncStations($results);

            // Sync paths
            $this->syncStationPaths($results);
        } catch (\Exception $e) {
            $results['errors'][] = $e->getMessage();
        }

        return $results;
    }
}
```

Data Models and DTOs

Models

1. Line Model

```
class Line extends Model
{
    protected $primaryKey = 'line_code';
    public $incrementing = false;
    protected $keyType = 'string';
    protected $fillable = [
        'line_code',              // String(2): RD, BL, GR, OR, SV, YL
        'display_name',           // String(50): Red, Blue, Green, etc.
        'start_station_code',     // String(3): A15, J03, etc.
        'end_station_code',       // String(3): B11, G05, etc.
        'internal_destination_1', // String(3): Optional branch destination
        'internal_destination_2', // String(3): Optional branch destination
    ];
}
```

2. Station Model

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

    protected $fillable = [
        'code',                // String(3): A01, B02, C03
        'name',                // String(100): Metro Center, Union Station
        'line_code_1',         // String(2): Primary line code
        'line_code_2',         // String(2): Transfer line code
        'line_code_3',         // String(2): Transfer line code
        'line_code_4',         // String(2): Transfer line code
        'station_together_1',  // String(3): Connected platform code
        'station_together_2',  // String(3): Connected platform code
        'lat',                 // Decimal(10,8): Latitude coordinate
        'lon',                 // Decimal(11,8): Longitude coordinate
        'is_active',          // Boolean: Station operational status
    ];
}
```

3. StationPath Model

```
class StationPath extends Model
{
    protected $fillable = [
        'line_code',        // String(2): Line identifier
        'station_code',     // String(3): Station identifier
        'station_name',     // String(100): Station display name
        'seq_num',          // Integer: Geographic sequence number
        'distance_to_prev', // Integer: Distance in meters to previous station
    ];

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

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

Configuration

Environment Variables

```
# WMATA API Configuration
WMATA_API_KEY=your-api-key
WMATA_BASE_URL=https://api.wmata.com
WMATA_TIMEOUT=30
WMATA_RETRY_ATTEMPTS=3

# Cache Configuration
WMATA_CACHE_LINES_TTL=86400
WMATA_CACHE_STATIONS_TTL=86400
WMATA_CACHE_PATHS_TTL=86400
WMATA_CACHE_PREDICTIONS_TTL=15

# Rate Limiting
WMATA_RATE_LIMIT=1000

# Frontend Configuration
WMATA_FRONTEND_REFRESH=30
```

Cache Configuration

```
// config/wmata.php
return [
    'api' => [
        'key' => env('WMATA_API_KEY'),
        'base_url' => env('WMATA_BASE_URL', 'https://api.wmata.com'),
        'timeout' => env('WMATA_TIMEOUT', 30),
        'retry_attempts' => env('WMATA_RETRY_ATTEMPTS', 3),
    ],

    'endpoints' => [
        'lines' => '/Rail.svc/json/jLines',
        'stations' => '/Rail.svc/json/jStations',
        'predictions' => '/StationPrediction.svc/json/GetPrediction',
        'path' => '/Rail.svc/json/jPath',
    ],

    '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),
    ],
];
```

API Integration

Endpoints

1. GET /api/metro/lines

```
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);
    }
}
```

2. GET /api/metro/stations/{lineCode}

```
public function getStationsForLine(string $lineCode): JsonResponse
{
    try {
        if (!Line::where('line_code', $lineCode)->exists()) {
            return response()->json([
                'success' => false,
                'error' => 'Invalid line code',
            ], 400);
        }

        $stations = $this->metroService->getOrderedStationsForLine($lineCode);

        return response()->json([
            'success' => true,
            'data' => $stations,
            'meta' => [
                'line_code' => $lineCode,
                'total_stations' => count($stations),
                'ordered' => true,
            ],
        ]);
    } catch (\Exception $e) {
        return response()->json([
            'success' => false,
            'error' => 'Failed to load stations: ' . $e->getMessage(),
        ], 500);
    }
}
```

3. GET /api/metro/predictions/{stationCode}

```
public function getTrainPredictions(string $stationCode): JsonResponse
{
    try {
        $predictions = $this->wmataApi->getTrainPredictions($stationCode);
        $station = Station::find($stationCode);

        if (!$station) {
            return response()->json([
                'success' => false,
                'error' => 'Station not found',
            ], 404);
        }

        return response()->json([
            'success' => true,
            'data' => [
                'station' => [
                    'code' => $station->code,
                    'name' => $station->name,
                ],
                'predictions' => array_map(
                    fn($prediction) => $prediction->toFrontend(),
                    $predictions
                ),
                'updated_at' => now()->toISOString(),
                'refresh_interval' =>
config('wmata.frontend.predictions_refresh_interval'),
            ],
        ]);
    } catch (\Exception $e) {
        return response()->json([
            'success' => false,
            'error' => 'Failed to get predictions: ' . $e->getMessage(),
        ], 500);
    }
}
```

Frontend Integration

Real-time Updates

The frontend implements automatic refresh of predictions using the following pattern:

```
export default {
  data() {
    return {
      refreshInterval: null,
      predictions: [],
      loading: false,
      error: null
    }
  },

  methods: {
    async fetchPredictions() {
      this.loading = true;
      try {
        const response = await this.metroApi.getPredictions(this.stationCode);
        this.predictions = response.data.predictions;
        this.setupRefreshInterval(response.data.refresh_interval);
      } catch (error) {
        this.error = error.message;
      } finally {
        this.loading = false;
      }
    },

    setupRefreshInterval(interval) {
      if (this.refreshInterval) {
        clearInterval(this.refreshInterval);
      }
      this.refreshInterval = setInterval(() => {
        this.fetchPredictions();
      }, interval * 1000);
    }
  },

  beforeUnmount() {
    if (this.refreshInterval) {
      clearInterval(this.refreshInterval);
    }
  }
}
```

Error Handling

The frontend implements comprehensive error handling:

1. Network Errors

```
async function makeRequest(endpoint, options = {}) {
  try {
    const response = await fetch(endpoint, {
      ...options,
      headers: {
        'Accept': 'application/json',
        ...options.headers
      }
    });

    if (!response.ok) {
      throw new Error(`HTTP error! status: ${response.status}`);
    }
    const data = await response.json();
    if (!data.success) {
      throw new Error(data.error || 'Unknown error occurred');
    }

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

2. Loading States

```
<template>
  <div class="metro-predictor">
    <div v-if="loading" class="loading-overlay">
      <spinner-component />
    </div>

    <div v-if="error" class="error-message">
      {{ error }}
      <button @click="retry">Retry</button>
    </div>

    <div v-else class="predictions-list">
      <!-- Predictions content -->
    </div>
  </div>
</template>
```

Deployment and Maintenance

Environment Setup

1. **Prerequisites**
   - PHP 8.1+
   - Composer
   - Node.js 16+
   - MySQL 8.0+
   - Redis (optional, but recommended for production)

2. **Installation**

```
# Clone repository
git clone [repository-url]

# Install PHP dependencies
composer install

# Install Node.js dependencies
npm install

# Set up environment
cp .env.example .env
php artisan key:generate

# Configure database
php artisan migrate

# Build frontend assets
npm run build

# Cache configuration
php artisan config:cache
php artisan route:cache
```

Maintenance Tasks

1. Daily Tasks

```
# Clear old cache entries
php artisan cache:clear

# Sync metro data
php artisan metro:sync

# Verify cache integrity
php artisan metro:verify-cache
```

2. Monitoring

- Monitor API rate limits using the wmata_api_rate_limit cache key
- Check error logs in `storage/logs/laravel.log`
- Monitor cache hit rates
- Track prediction refresh performance

Troubleshooting

1. Common Issues

- **Rate Limiting**

  ```
  // Check current rate limit count
  Cache::get('wmata_api_rate_limit')
  ```

- **Cache Issues**

  ```
  // Clear specific cache keys
  Cache::forget('wmata.lines');
  Cache::forget('wmata.stations.all');
  ```

- **API Connectivity**

  ```
  // Test API connection
  php artisan metro:sync --validate
  ```

2. Performance Optimization

- Use Redis for caching in production
- Implement database indexes for frequently queried fields
- Monitor and optimize database queries
- Consider implementing request queuing for high-traffic periods

This documentation provides a comprehensive and accurate representation of the current codebase, including all recent updates and improvements.