Metro Train Prediction App - Component Integration Guide

Author: David Morrison

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

Overview

The **Metro Train Prediction App** uses a modular Vue 3 component architecture designed for reusability, maintainability, and clean separation of concerns. The component system provides a complete solution for Metro train predictions while allowing for easy customization and integration into other applications.

Design Principles

- **Single Responsibility** - Each component has one clear purpose
- **Composition over Inheritance** - Components are composed together rather than extended
- **Prop-driven Configuration** - Behavior controlled through props rather than internal state
- **Event-driven Communication** - Clean parent-child communication through events
- **Reactive Data Flow** - Vue 3 Composition API for optimal reactivity

Component Hierarchy

```
MetroTrainPredictor (Main Container)
```

```
├── LineSelector (Form Component)
├── StationSelector (Form Component)
├── LoadingState (Utility Component)
└── PredictionList (Display Component)
```

Component Architecture

Main Container Component

MetroTrainPredictor.vue

The primary orchestrator component that manages the complete prediction workflow.

vue

```vue
<template>
  <div class="metro-predictor">
    <h2>Metro Train Predictions</h2>

    <form @submit.prevent class="prediction-form">
      <LineSelector
        v-model="selectedLine"
        :lines="lines"
        @update:modelValue="onLineChange"
      />


      <StationSelector
        v-model="selectedStation"
        :stations="stations"
        @update:modelValue="onStationChange"
      />
    </form>


    <LoadingState :show="loading.stations">
      Loading stations for {{ getLineName(selectedLine) }} line...
    </LoadingState>


    <LoadingState :show="loading.predictions">
      Loading train predictions...
    </LoadingState>
```

```
    <PredictionList
      :predictions="predictions"
      :station-info="stationInfo"
      :selected-station="selectedStation"
      :loading="loading.predictions"
      :last-updated="lastUpdated"
      :refresh-interval="refreshInterval"
      :get-line-name="getLineName"
    />


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

Key Responsibilities:

- State management for the entire prediction workflow
- API communication coordination
- Auto-refresh timer management
- Error handling and display
- Component lifecycle management

Form Components

LineSelector.vue

Handles metro line selection with proper v-model support.

vue
```
<template>
  <div class="form-group">
    <label for="line">Line:</label>
    <select
      id="line"
      :value="modelValue"
      @change="$emit('update:modelValue', $event.target.value)"
      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>
</template>


<script setup>
defineProps({
  modelValue: {
    type: String,
    required: true
  },
  lines: {
    type: Array,
    required: true
  }
})


defineEmits(['update:modelValue'])
</script>
```

Props:

- `modelValue` (String, required) - Currently selected line code
- `lines` (Array, required) - Available lines array

Events:

- `update:modelValue` - Emitted when line selection changes

StationSelector.vue

Handles station selection with conditional rendering.

vue

```
<template>
  <div class="form-group" v-if="stations.length > 0">
```

```
    <label for="station">Station:</label>
    <select
      id="station"
      :value="modelValue"
      @change="$emit('update:modelValue', $event.target.value)"
      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>
</template>
```

Props:

- `modelValue` (String, required) - Currently selected station code
- `stations` (Array, required) - Available stations array

Events:

- `update:modelValue` - Emitted when station selection changes

Display Components

PredictionList.vue

Displays real-time train predictions with responsive design.

vue

```
<template>
  <div v-if="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>
        <span class="destination">to {{ prediction.destination }}</span>
        <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: {{ formatLastUpdated }}
      <span v-if="refreshInterval" class="refresh-info">
        (refreshes every {{ refreshInterval }}s)
      </span>
    </div>
  </div>
  <div v-else-if="selectedStation && !loading" class="no-predictions">
    No train predictions available for this station.
  </div>
</template>
```

Props:

- `predictions` (Array, required) - Train prediction data
- `stationInfo` (Object, required) - Station metadata
- `selectedStation` (String, required) - Current station code
- `loading` (Boolean) - Loading state indicator
- `lastUpdated` (String, required) - Last update timestamp
- `refreshInterval` (Number) - Auto-refresh interval
- `getLineName` (Function, required) - Line code to name converter

LoadingState.vue

Reusable loading indicator component.

vue

```
<template>
  <div v-if="show" class="loading">
    <slot></slot>
  </div>
</template>


<script setup>
defineProps({
  show: {
```

```
    type: Boolean,
    required: true
  }
})
</script>
```

Props:

- `show` (Boolean, required) - Controls visibility

Slots:

- `default` - Loading message content

Integration Patterns

Basic Integration

Simple Usage

vue

```
<template>
  <div id="app">
    <MetroTrainPredictor />
  </div>
</template>


<script setup>
import MetroTrainPredictor from './components/MetroTrainPredictor.vue'
</script>
```

With Custom Container

vue

```
<template>
  <div class="app-container">
    <header>
      <h1>My Transit App</h1>
    </header>

    <main class="main-content">
      <MetroTrainPredictor />
```

```
    </main>

    <footer>
      <p>Powered by WMATA API</p>
    </footer>
  </div>
</template>
```

Advanced Integration

Multiple Instances

vue

```
<template>
  <div class="multi-predictor">
    <div class="predictor-section">
      <h2>Home Station</h2>
      <MetroTrainPredictor key="home" />
    </div>

    <div class="predictor-section">
      <h2>Work Station</h2>
      <MetroTrainPredictor key="work" />
    </div>
  </div>
</template>
```

With External State Management

vue

```
<template>
  <MetroTrainPredictor />
</template>


<script setup>
import { provide } from 'vue'
import { useMetroStore } from './stores/metro'


// Provide store to child components
const metroStore = useMetroStore()
provide('metroStore', metroStore)
</script>
```

Component Composition

Custom Form Layout

vue

```
<template>
  <div class="custom-predictor">
    <div class="form-section">
      <div class="form-row">
        <LineSelector
          v-model="selectedLine"
          :lines="lines"
          @update:modelValue="onLineChange"
        />

        <StationSelector
          v-model="selectedStation"
          :stations="stations"
          @update:modelValue="onStationChange"
        />
      </div>

      <button @click="refreshPredictions" :disabled="!selectedStation">
        Refresh Now
      </button>
    </div>

    <PredictionList
      :predictions="predictions"
      :station-info="stationInfo"
      :selected-station="selectedStation"
      :loading="loading"
      :last-updated="lastUpdated"
      :refresh-interval="refreshInterval"
      :get-line-name="getLineName"
    />
  </div>
</template>
```

With Additional Features

vue

```
<template>
  <div class="enhanced-predictor">
```

```
    <!-- Favorites functionality -->
    <div class="favorites-bar">
      <button
        v-for="favorite in favorites"
        :key="favorite.code"
        @click="selectFavorite(favorite)"
        class="favorite-btn"
      >
        {{ favorite.name }}
      </button>
    </div>

    <!-- Main predictor -->
    <MetroTrainPredictor ref="predictor" />

    <!-- Additional controls -->
    <div class="controls">
      <button @click="addToFavorites" :disabled="!currentStation">
        Add to Favorites
      </button>
      <button @click="shareStation" :disabled="!currentStation">
        Share Station
      </button>
    </div>
  </div>
</template>
```

Data Flow

State Management Flow

```
User Action → Form Component → Parent State → API Call → Data Update → Display Component
```

Detailed Flow Diagram

```
1. User selects line
   ↓
2. LineSelector emits update:modelValue
   ↓
3. MetroTrainPredictor.onLineChange()
   ↓
```

```
4. Clear stations/predictions state
   ↓
5. API call: fetchStations(lineCode)
   ↓
6. Update stations array
   ↓
7. StationSelector becomes visible


8. User selects station
   ↓
9. StationSelector emits update:modelValue
   ↓
10. MetroTrainPredictor.onStationChange()
    ↓
11. API call: fetchPredictions(stationCode)
    ↓
12. Update predictions state
    ↓
13. Start auto-refresh timer
    ↓
14. PredictionList displays data
```

API Communication Pattern

javascript

```javascript
// MetroTrainPredictor.vue
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 = data.updated_at
    refreshInterval.value = data.refresh_interval || 30
  } catch (err) {
    error.value = `Failed to load predictions: ${err.message}`
    predictions.value = []
  } finally {
    loading.predictions = false
  }
```

```
}
```

Auto-refresh Lifecycle

javascript

```javascript
// Timer management in MetroTrainPredictor.vue
const onStationChange = () => {
  predictions.value = []

  // Clear existing timer
  if (refreshTimer) {
    clearInterval(refreshTimer)
    refreshTimer = null
  }

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

    // Set up auto-refresh
    refreshTimer = setInterval(() => {
      fetchPredictions(selectedStation.value)
    }, refreshInterval.value * 1000)
  }
}


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

Customization Guide

Props Customization

Custom API Service

vue

```vue
<template>
```

```
  <MetroTrainPredictor :api-service="customMetroApi" />
</template>


<script setup>
import { customMetroApi } from './services/customMetroApi'
</script>
```

Custom Refresh Intervals

vue

```
<template>
  <MetroTrainPredictor
    :default-refresh-interval="60"
    :min-refresh-interval="10"
  />
</template>
```

Event Handling

Custom Event Listeners

vue

```
<template>
  <MetroTrainPredictor
    @line-changed="onLineChanged"
    @station-changed="onStationChanged"
    @predictions-updated="onPredictionsUpdated"
    @error="onError"
  />
</template>


<script setup>
const onLineChanged = (lineCode) => {
  console.log('Line changed to:', lineCode)
  // Custom analytics, logging, etc.
}


const onStationChanged = (stationCode) => {
  console.log('Station changed to:', stationCode)
  // Update URL, save preference, etc.
}
```

```
const onPredictionsUpdated = (predictions) => {
  console.log('Predictions updated:', predictions.length)
  // Custom processing, notifications, etc.
}


const onError = (error) => {
  console.error('Metro error:', error)
  // Custom error handling, reporting, etc.
}
</script>
```

Slot Customization

Custom Loading States

vue

```
<template>
  <MetroTrainPredictor>
    <template #loading-lines>
      <div class="custom-loading">
        <spinner />
        <p>Loading metro lines...</p>
      </div>
    </template>

    <template #loading-stations>
      <div class="custom-loading">
        <spinner />
        <p>Finding stations...</p>
      </div>
    </template>

    <template #loading-predictions>
      <div class="custom-loading">
        <spinner />
        <p>Getting real-time data...</p>
      </div>
    </template>
  </MetroTrainPredictor>
</template>
```

Custom Error Messages

vue
```
<template>
  <MetroTrainPredictor>
    <template #error="{ error }">
      <div class="custom-error">
        <icon name="warning" />
        <h3>Oops! Something went wrong</h3>
        <p>{{ error }}</p>
        <button @click="retryAction">Try Again</button>
      </div>
    </template>
  </MetroTrainPredictor>
</template>
```

Method Access

Component References

vue
```
<template>
  <MetroTrainPredictor ref="predictorRef" />
  <button @click="manualRefresh">Refresh Now</button>
</template>


<script setup>
import { ref } from 'vue'


const predictorRef = ref(null)


const manualRefresh = () => {
  if (predictorRef.value) {
    predictorRef.value.refreshPredictions()
  }
}
</script>
```

Styling and Theming

CSS Custom Properties

The components use CSS custom properties for easy theming:

css

```css
:root {
  /* Colors */
  --color-text: #333333;
  --color-heading: #2c3e50;
  --color-background: #ffffff;
  --color-background-soft: #f8f9fa;
  --color-border: #dee2e6;

  /* Spacing */
  --spacing-xs: 0.25rem;
  --spacing-sm: 0.5rem;
  --spacing-md: 1rem;
  --spacing-lg: 1.5rem;
  --spacing-xl: 2rem;

  /* Typography */
  --font-family-base: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
  --font-size-sm: 0.875rem;
  --font-size-base: 1rem;
  --font-size-lg: 1.125rem;
  --font-weight-normal: 400;
  --font-weight-semibold: 600;
  --font-weight-bold: 700;

  /* Prediction Status Colors */
  --color-arriving: #dc3545;
  --color-soon: #fd7e14;
  --color-moderate: #ffc107;
  --color-later: #28a745;
}
```

Dark Theme Example

css

```css
[data-theme="dark"] {
  --color-text: #e9ecef;
```

```
  --color-heading: #f8f9fa;
  --color-background: #212529;
  --color-background-soft: #343a40;
  --color-border: #495057;
}
```

Custom Styling

Override Component Styles

CSS

```css
/* Custom metro predictor styling */
.metro-predictor {
  max-width: 1000px;
  margin: 0 auto;
  padding: 2rem;
  background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
  border-radius: 12px;
  box-shadow: 0 10px 30px rgba(0, 0, 0, 0.1);
}


.prediction-form {
  background: rgba(255, 255, 255, 0.95);
  backdrop-filter: blur(10px);
  border-radius: 8px;
  padding: 2rem;
}


.prediction-item {
  background: linear-gradient(90deg, #f8f9fa 0%, #ffffff 100%);
  border-left: 4px solid var(--color-moderate);
  margin-bottom: 0.5rem;
  transition: all 0.3s ease;
}


.prediction-item:hover {
  transform: translateX(4px);
  box-shadow: 0 4px 12px rgba(0, 0, 0, 0.1);
}
```

Responsive Breakpoints

css

```css
/* Mobile-first responsive design */
.metro-predictor {
  padding: 1rem;
}

@media (min-width: 768px) {
  .metro-predictor {
    padding: 2rem;
  }

  .prediction-form {
    display: grid;
    grid-template-columns: 1fr 1fr;
    gap: 2rem;
  }
}

@media (min-width: 1024px) {
  .prediction-list {
    display: grid;
    grid-template-columns: repeat(auto-fit, minmax(300px, 1fr));
    gap: 1rem;
  }
}
```

Animation Examples

Loading Animations

css

```css
@keyframes pulse {
  0%, 100% { opacity: 1; }
  50% { opacity: 0.5; }
}

.loading {
  animation: pulse 2s ease-in-out infinite;
}
```

```
@keyframes slideIn {
  from {
    opacity: 0;
    transform: translateY(-10px);
  }
  to {
    opacity: 1;
    transform: translateY(0);
  }
}

.prediction-item {
  animation: slideIn 0.3s ease-out;
}
```

Performance Optimization

Component Optimization

Lazy Loading

javascript

```
// Lazy load components for better initial load time
import { defineAsyncComponent } from 'vue'

const MetroTrainPredictor = defineAsyncComponent(() =>
  import('./components/MetroTrainPredictor.vue')
)

const PredictionList = defineAsyncComponent(() =>
  import('./components/metro/predictions/PredictionList.vue')
)
```

Computed Properties

javascript

```
// Use computed properties for expensive calculations
const sortedPredictions = computed(() => {
  return predictions.value.sort((a, b) => {
    // Sort by arrival time
```

```
    if (a.minutes === 'BRD') return -1
    if (b.minutes === 'BRD') return 1
    if (a.minutes === 'ARR') return -1
    if (b.minutes === 'ARR') return 1
    return parseInt(a.minutes) - parseInt(b.minutes)
  })
})
```

Memory Management

javascript

```
// Proper cleanup in components
onUnmounted(() => {
  // Clear timers
  if (refreshTimer) {
    clearInterval(refreshTimer)
    refreshTimer = null
  }

  // Clear large data structures
  predictions.value = []
  stations.value = []

  // Cancel pending requests
  if (abortController) {
    abortController.abort()
  }
})
```

API Optimization

Request Deduplication

javascript

```
// Prevent duplicate API calls
const requestCache = new Map()


const makeRequest = async (endpoint) => {
  if (requestCache.has(endpoint)) {
    return requestCache.get(endpoint)
  }

  const promise = api.get(endpoint)
```

```
  requestCache.set(endpoint, promise)

  try {
    const result = await promise
    return result
  } finally {
    // Clear cache after request completes
    setTimeout(() => {
      requestCache.delete(endpoint)
    }, 1000)
  }
}
```

Smart Caching

javascript

```
// Cache data with expiration
const dataCache = reactive({
  lines: { data: null, expires: 0 },
  stations: new Map(),
  predictions: new Map()
})


const getCachedData = (key, ttl = 60000) => {
  const cached = dataCache[key]
  if (cached && cached.expires > Date.now()) {
    return cached.data
  }
  return null
}


const setCachedData = (key, data, ttl = 60000) => {
  dataCache[key] = {
    data,
    expires: Date.now() + ttl
  }
}
```

Troubleshooting

Common Integration Issues

Component Not Rendering

Problem: MetroTrainPredictor component doesn't appear

Solutions:

javascript
```
// Check component import
import MetroTrainPredictor from './components/MetroTrainPredictor.vue'


// Verify component registration
export default {
  components: {
    MetroTrainPredictor
  }
}


// Check for JavaScript errors in console
console.error // Look for error messages
```

API Connection Issues

Problem: Components load but no data appears

Solutions:

javascript
```
// Verify API configuration
import { metroApi } from './services/metroApi'


// Test API directly
metroApi.getLines()
  .then(data => console.log('API working:', data))
  .catch(err => console.error('API error:', err))


// Check network requests in browser dev tools
```

Styling Issues

Problem: Components appear unstyled or incorrectly styled

Solutions:

css
```
/* Ensure CSS custom properties are defined */
:root {
  --color-text: #333;
  --color-background: #fff;
  /* ... other properties */
}


/* Check for CSS conflicts */
.metro-predictor * {
  box-sizing: border-box;
}
```

Performance Issues

Slow Loading

Symptoms: Components take long time to load data

Diagnostics:

javascript
```
// Add timing to API calls
const startTime = performance.now()
await metroApi.getLines()
const duration = performance.now() - startTime
console.log(`API call took ${duration}ms`)


// Monitor component render time
import { onMounted, onUpdated } from 'vue'


onMounted(() => {
  console.log('Component mounted at:', Date.now())
})


onUpdated(() => {
```

```
  console.log('Component updated at:', Date.now())
})
```

Memory Leaks

Symptoms: Browser memory usage increases over time

Solutions:

javascript
```
// Ensure proper cleanup
onUnmounted(() => {
  // Clear all timers
  clearInterval(refreshTimer)
  clearTimeout(debounceTimer)

  // Clear large arrays
  predictions.value = []
  stations.value = []

  // Remove event listeners
  window.removeEventListener('beforeunload', cleanup)
})
```

Development Debugging

Debug Mode

javascript
```
// Enable debug logging
const DEBUG = import.meta.env.DEV


const debugLog = (message, data) => {
  if (DEBUG) {
    console.log(`[Metro Debug] ${message}`, data)
  }
}


// Use throughout components
debugLog('Line changed', selectedLine.value)
debugLog('Predictions received', predictions.value)
```

Component State Inspection

vue

```
<template>
  <div>
    <!-- Production component -->
    <MetroTrainPredictor />

    <!-- Debug panel (dev only) -->
    <div v-if="$dev" class="debug-panel">
      <h3>Debug Info</h3>
      <pre>{{ debugInfo }}</pre>
    </div>
  </div>
</template>


<script setup>
const debugInfo = computed(() => ({
  selectedLine: selectedLine.value,
  selectedStation: selectedStation.value,
  predictionsCount: predictions.value.length,
  loading: loading,
  error: error.value
}))
</script>
```

This component integration guide provides comprehensive documentation for implementing, customizing, and troubleshooting the Metro Train Prediction App's Vue 3 component system.