Mixin 사용 튜토리얼

새 컴포넌트에서 PopupMixin과 HeatmapMixin을 사용하는 step-by-step 가이드.

> ***관련 문서***
>
> - *POPUP_MIXIN_API.md — PopupMixin 전체 메서드 레퍼런스*
> - *HEATMAP_MIXIN_API.md — HeatmapMixin 옵션 레퍼런스*
> - *COMPONENT_RUNTIME_GUIDE.md — 런타임 주입 속성 및 컴포넌트 간 통신*

목차

---

1. 사전 준비

컴포넌트 디렉토리 구조

```
Projects/{ProjectName}/page/components/{ComponentName}/
 ├─── scripts/
 |    ├─── register.js          ← 메인 로직 (이 파일을 작성)
 |    └─── beforeDestroy.js     ← 정리 로직
 ├─── views/
 |    └─── component.html       ← 팝업 HTML 템플릿
 ├─── styles/
 |    └─── component.css        ← 팝업 CSS
 └─── preview.html              ← 개발용 프리뷰
```

사용 가능한 전역 객체

객체	제공 메서드	설명
`Wkit`	`bind3DEvents`, `fetchData`	3D 이벤트, 데이터 패치
`PopupMixin`	`applyShadowPopupMixin`, `applyEChartsMixin`, `applyTabulatorMixin`	팝업 시스템
`HeatmapMixin`	`applyHeatmapMixin`	3D 히트맵 서피스
`fx`	`go`, `map`, `filter`, `each`, `reduce`	함수형 유틸리티
`wemb`	`configManager`	글로벌 설정

---

2. PopupMixin 튜토리얼

가상의 **Generator(발전기)** 컴포넌트를 만들면서 단계별로 설명합니다.

Step 1: 기본 뼈대

```
// register.js
const { bind3DEvents, fetchData } = Wkit;
const { applyShadowPopupMixin, applyEChartsMixin } = PopupMixin;
```

```
// ──────────────────────────────────────────────────
// 헬퍼 함수 (모든 컴포넌트 공통)
// ──────────────────────────────────────────────────

function extractTemplate(htmlCode, templateId) {
  const parser = new DOMParser();
  const doc = parser.parseFromString(htmlCode, 'text/html');
  const template = doc.querySelector(`template#${templateId}`);
  return template?.innerHTML || '';
}

function extractData(response, path = 'data') {
  if (!response?.response) return null;
  const data = response.response[path];
  return data !== null && data !== undefined ? data : null;
}

function hexToRgba(hex, alpha) {
  const r = parseInt(hex.slice(1, 3), 16);
  const g = parseInt(hex.slice(3, 5), 16);
  const b = parseInt(hex.slice(5, 7), 16);
  return `rgba(${r}, ${g}, ${b}, ${alpha})`;
}

// ──────────────────────────────────────────────────
// 진입점
// ──────────────────────────────────────────────────

initComponent.call(this);
```

> *`initComponent.call(this)` : 모든 컴포넌트 register.js의 마지막에 호출됩니다.* `this` *는 런타임이 주입하는 컴포넌트 인스턴스입니다.*

Step 2: 내부 상태 초기화

```
function initComponent() {
  // ─── 1. 내부 상태 ───
  this._defaultAssetKey = this.setter?.assetInfo?.assetKey || this.id;
  this._baseUrl = wemb.configManager.assetApiUrl.replace(/^https?:W/W//, '');
  this._locale = 'ko';
  this._popupTemplateId = 'popup-generator';  // HTML template#id
```

속성	출처	설명
_defaultAssetKey	this.setter.assetInfo	자산 식별키. 없으면 컴포넌트 id fallback
_baseUrl	wemb.configManager	API 서버 주소 (프로토콜 제거)
_locale	고정값	다국어 코드
_popupTemplateId	고정값	views/component.html 내 <template> id

Step 3: Config 통합 객체

모든 설정을 `this.config` 하나에 모읍니다.

```
// ─── 2. Config 통합 ───
this.config = {
  // 데이터셋 이름
  datasetNames: {
    assetDetail: 'assetDetailUnified',
    metricLatest: 'metricLatest',
    metricHistory: 'metricHistoryStats',
    modelDetail: 'modelDetail',
    vendorDetail: 'vendorDetail',
  },

  // API 엔드포인트 및 파라미터
  api: {
    trendHistory: '/api/v1/mhs/l',
    trendParams: {
      interval: '1h',
      timeRange: 24 * 60 * 60 * 1000,     // 24시간
      metricCodes: ['GEN.VOLTAGE', 'GEN.FREQUENCY'],
      statsKeys: [],
      timeField: 'time',
    },
    statsKeyMap: {
      'GEN.VOLTAGE': 'avg',
      'GEN.FREQUENCY': 'avg',
    },
  },

  // 상태 코드 매핑 (모든 컴포넌트 동일)
  statusMap: {
    ACTIVE:      { label: '정상운영', dataAttr: 'normal' },
    WARNING:     { label: '주의',     dataAttr: 'warning' },
    CRITICAL:    { label: '위험',     dataAttr: 'critical' },
    INACTIVE:    { label: '비활성',   dataAttr: 'inactive' },
    MAINTENANCE: { label: '유지보수', dataAttr: 'maintenance' },
    DEFAULT:     { label: '알 수 없음', dataAttr: 'normal' },
  },

  // ─── UI 영역별 설정 ───

  // 팝업 헤더
  header: {
    fields: [
      { key: 'name', selector: '.gen-name' },
      { key: 'locationLabel', selector: '.gen-zone' },
      { key: 'statusType', selector: '.gen-status', transform: statusTypeToLabel },
      { key: 'statusType', selector: '.gen-status', dataAttr: 'status', transform: statusTypeToDataAttr },
    ],
  },

  // 기본정보 테이블
  infoTable: {
    fields: [
      { key: 'name', selector: '.info-name' },
      { key: 'assetType', selector: '.info-type' },
      { key: 'assetModelName', selector: '.info-model', fallback: '-' },
      { key: 'usageCode', selector: '.info-usage', fallback: '-' },
```

```
      { key: 'locationLabel', selector: '.info-location' },
      { key: 'statusType', selector: '.info-status', transform: statusTypeToLabel },
      { key: 'installDate', selector: '.info-install-date', transform: formatDate },
    ],
    chain: {
      vendor: '.info-vendor',
      model: '.info-model',
    },
  },

  // 트렌드 차트
  chart: {
    tabs: {
      voltage:   { metricCode: 'GEN.VOLTAGE',   label: '전압', unit: 'V',  color: '#3b82f6', scale: 1.0 },
      frequency: { metricCode: 'GEN.FREQUENCY', label: '주파수', unit: 'Hz', color: '#22c55e', scale: 1.0 },
    },
    selectors: {
      container: '.chart-container',
      tabBtn: '.tab-btn',
    },
  },
};
```

Config 설계 원칙:

- `datasetNames` : API 호출에 사용할 데이터셋 이름
- `api` : 엔드포인트, 파라미터, 통계 키 매핑
- `statusMap` : 상태 코드 → UI 레이블/CSS 속성
- `header` / `infoTable` / `chart` : 각 UI 영역의 필드-셀렉터 바인딩

> *자세한 config 설계 이유는 WHY CONFIG.md 참조.*

Step 4: 데이터셋 정의

```
// ─── 3. 데이터셋 정의 ───
const { datasetNames, api } = this.config;
const baseParam = {
  baseUrl: this._baseUrl,
  assetKey: this._defaultAssetKey,
  locale: this._locale,
};

this.datasetInfo = [
  {
    datasetName: datasetNames.assetDetail,
    param: { ...baseParam },
    render: ['renderBasicInfo'],
    refreshInterval: 0,           // 0 = 최초 1회만
  },
  {
    datasetName: datasetNames.metricHistory,
    param: { ...baseParam, ...api.trendParams, apiEndpoint: api.trendHistory },
    render: ['renderTrendChart'],
    refreshInterval: 5000,        // 5초 주기 갱신
```

```
  },
];
```

필드	설명
datasetName	fetchData(page, datasetName, param) 호출에 사용
param	API 파라미터
render	데이터 수신 후 호출할 렌더 함수명 배열
refreshInterval	0: 최초 1회 / > 0: setInterval 주기 (ms)

Step 5: 함수 바인딩 + Mixin 적용

```
// ─── 4. 렌더링 함수 바인딩 ───
this.renderBasicInfo = renderBasicInfo.bind(this);
this.renderTrendChart = renderTrendChart.bind(this);

// ─── 5. Public Methods ───
this.showDetail = showDetail.bind(this);
this.hideDetail = hideDetail.bind(this);
this.stopRefresh = stopRefresh.bind(this);

// ─── 6. 3D 이벤트 바인딩 ───
this.customEvents = { click: '@assetClicked' };
bind3DEvents(this, this.customEvents);

// ─── 7. PopupMixin 적용 ───
const popupCreatedConfig = {
  chartSelector: this.config.chart.selectors.container,
  events: {
    click: {
      '.close-btn': () => this.hideDetail(),
      '.tab-btn': (e) => switchTab.call(this, e.target.dataset.tab),
    },
  },
};

const { htmlCode, cssCode } = this.properties.publishCode || {};
this.getPopupHTML = () => extractTemplate(htmlCode || '', this._popupTemplateId);
this.getPopupStyles = () => cssCode || '';
this.onPopupCreated = onPopupCreated.bind(this, popupCreatedConfig);

applyShadowPopupMixin(this, {
  getHTML: this.getPopupHTML,
  getStyles: this.getPopupStyles,
  onCreated: this.onPopupCreated,
});

applyEChartsMixin(this);

// ─── 8. destroyPopup 확장 ───
const _origDestroyPopup = this.destroyPopup;
const _ctx = this;
```

```
  this.destroyPopup = function () {
    _ctx.stopRefresh();
    _origDestroyPopup.call(_ctx);
  };

  console.log('[Generator] Registered:', this._defaultAssetKey);
}
// ← initComponent 끝
```

적용 순서가 중요합니다:

```
① applyShadowPopupMixin  ← 반드시 첫 번째
② applyEChartsMixin      ← 차트가 필요하면 (Shadow DOM 의존)
③ applyTabulatorMixin    ← 테이블이 필요하면 (Shadow DOM 의존)
④ destroyPopup 확장      ← 컴포넌트 고유 정리 로직 추가
```

Step 6: Public Methods (showDetail / hideDetail / stopRefresh)

이 3개 함수는 **모든 컴포넌트에서 동일한 패턴**입니다. 복사해서 사용합니다.

```
function showDetail() {
  this.showPopup();

  // 모든 데이터셋 1회 fetch
  fx.go(
    this.datasetInfo,
    fx.each(d => fetchDatasetAndRender.call(this, d))
  );

  // refreshInterval > 0인 데이터셋에 대해 주기적 갱신
  this.stopRefresh();
  fx.go(
    this.datasetInfo,
    fx.filter(d => d.refreshInterval > 0),
    fx.each(d => {
      d._intervalId = setInterval(
        () => fetchDatasetAndRender.call(this, d),
        d.refreshInterval
      );
    })
  );
}

function hideDetail() {
  this.stopRefresh();
  this.hidePopup();
}

function stopRefresh() {
  const datasetInfo = this.datasetInfo ?? [];
  fx.go(
    datasetInfo,
    fx.filter(d => d._intervalId),
    fx.each(d => {
      clearInterval(d._intervalId);
```

```
      d._intervalId = null;
    })
  );
}
```

Step 7: 데이터 Fetch 헬퍼

```
function formatLocalDate(date) {
  const y = date.getFullYear();
  const m = String(date.getMonth() + 1).padStart(2, '0');
  const d = String(date.getDate()).padStart(2, '0');
  const h = String(date.getHours()).padStart(2, '0');
  const min = String(date.getMinutes()).padStart(2, '0');
  const s = String(date.getSeconds()).padStart(2, '0');
  return y + '-' + m + '-' + d + ' ' + h + ':' + min + ':' + s;
}

function fetchDatasetAndRender(d) {
  const { datasetNames } = this.config;
  const { datasetName, param, render } = d;

  // metricHistory는 매 호출마다 timeFrom/timeTo 갱신
  if (datasetName === datasetNames.metricHistory) {
    const now = new Date();
    const from = new Date(now.getTime() - param.timeRange);
    param.timeFrom = formatLocalDate(from);
    param.timeTo = formatLocalDate(now);
  }

  fetchData(this.page, datasetName, param)
    .then(response => {
      const data = extractData(response);
      if (!data) return;
      fx.each(fn => this[fn](response), render);
    })
    .catch(e => console.warn(`[Generator] ${datasetName} fetch failed:`, e));
}
```

Step 8: 렌더링 함수

팝업 생성 콜백

```
function onPopupCreated({ chartSelector, events }) {
  chartSelector && this.createChart(chartSelector);
  events && this.bindPopupEvents(events);
}
```

기본정보 테이블

```
function renderBasicInfo({ response }) {
  const { data } = response;
  if (!data || !data.asset) return;
```

```
  const asset = data.asset;
  const { header, infoTable } = this.config;

  // 헤더 렌더링
  fx.go(header.fields, fx.each(field => renderField(this, asset, field)));

  // 테이블 렌더링
  fx.go(infoTable.fields, fx.each(field => renderField(this, asset, field)));

  // 제조사/모델 체이닝 (비동기)
  fetchModelVendorChain(this, asset, infoTable.chain);
}

function renderField(ctx, data, field) {
  const el = ctx.popupQuery(field.selector);
  if (!el) return;
  let value = data[field.key] ?? field.fallback ?? '-';
  if (field.transform) value = field.transform(value);
  if (field.dataAttr) {
    el.dataset[field.dataAttr] = value;
  } else {
    el.textContent = value;
  }
}

function fetchModelVendorChain(ctx, asset, chainConfig) {
  const { datasetNames } = ctx.config;
  const setCell = (selector, value) => {
    const el = ctx.popupQuery(selector);
    if (el) el.textContent = value ?? '-';
  };

  if (!asset.assetModelKey) return;

  fx.go(
    fetchData(ctx.page, datasetNames.modelDetail, {
      baseUrl: ctx._baseUrl, assetModelKey: asset.assetModelKey,
    }),
    (modelResp) => {
      const model = extractData(modelResp, 'data');
      if (!model) return;
      setCell(chainConfig.model, model.name);

      if (model.assetVendorKey) {
        fx.go(
          fetchData(ctx.page, datasetNames.vendorDetail, {
            baseUrl: ctx._baseUrl, assetVendorKey: model.assetVendorKey,
          }),
          (vendorResp) => {
            const vendor = extractData(vendorResp, 'data');
            if (vendor) setCell(chainConfig.vendor, vendor.name);
          }
        ).catch(() => {});
      }
    }
```

```
  ).catch(() => {});
}
```

트렌드 차트

```
function renderTrendChart({ response }) {
  const { data } = response;
  const { chart, api } = this.config;
  const { tabs, selectors } = chart;

  // 현재 활성 탭 (기본값: 첫 번째 탭)
  const activeTab = this._activeTab || Object.keys(tabs)[0];
  const tabConfig = tabs[activeTab];
  if (!tabConfig) return;

  const safeData = Array.isArray(data) ? data : [];

  // metricCode로 필터 → 시간별 값 추출
  const chartData = safeData.filter(row => row.metricCode === tabConfig.metricCode);
  const statsKey = api.statsKeyMap[tabConfig.metricCode] || 'avg';

  const times = [];
  const values = [];
  chartData.forEach(row => {
    times.push(row.time || '');
    const val = row.statsBody?.[statsKey];
    values.push(val != null ? +(val * tabConfig.scale).toFixed(1) : null);
  });

  const option = {
    tooltip: {
      trigger: 'axis',
      backgroundColor: 'rgba(26, 31, 46, 0.95)',
      borderColor: '#2a3142',
      textStyle: { color: '#e0e6ed', fontSize: 12 },
    },
    grid: { left: 50, right: 20, top: 30, bottom: 24 },
    xAxis: {
      type: 'category',
      data: times,
      axisLine: { lineStyle: { color: '#333' } },
      axisLabel: { color: '#888', fontSize: 10 },
    },
    yAxis: {
      type: 'value',
      name: tabConfig.unit,
      axisLine: { show: true, lineStyle: { color: tabConfig.color } },
      axisLabel: { color: '#888', fontSize: 10 },
      splitLine: { lineStyle: { color: '#333' } },
    },
    series: [{
      type: 'line',
      data: values,
      smooth: true,
      symbol: 'none',
```

```
      lineStyle: { color: tabConfig.color, width: 2 },
      areaStyle: {
        color: {
          type: 'linear', x: 0, y: 0, x2: 0, y2: 1,
          colorStops: [
            { offset: 0, color: hexToRgba(tabConfig.color, 0.3) },
            { offset: 1, color: hexToRgba(tabConfig.color, 0) },
          ],
        },
      },
    }],
  };

  this.updateChart(selectors.container, option);
}

function switchTab(tabName) {
  if (!this.config.chart.tabs[tabName]) return;
  this._activeTab = tabName;

  // 탭 버튼 active 상태 전환
  const buttons = this.popupQueryAll(this.config.chart.selectors.tabBtn);
  fx.each(btn => {
    btn.classList.toggle('active', btn.dataset.tab === tabName);
  }, Array.from(buttons || []));

  // 트렌드 데이터 재요청
  const { datasetNames } = this.config;
  const trendInfo = this.datasetInfo.find(d => d.datasetName === datasetNames.metricHistory);
  if (trendInfo) fetchDatasetAndRender.call(this, trendInfo);
}
```

변환 함수

```
function statusTypeToLabel(statusType) {
  // initComponent 내에서 .bind(this)로 바인딩되므로 this.config 접근 가능
  const { statusMap } = this.config;
  return (statusMap[statusType] || statusMap.DEFAULT).label;
}

function statusTypeToDataAttr(statusType) {
  const { statusMap } = this.config;
  return (statusMap[statusType] || statusMap.DEFAULT).dataAttr;
}

function formatDate(dateStr) {
  if (!dateStr) return '-';
  try {
    return new Date(dateStr).toLocaleDateString('ko-KR', {
      year: 'numeric', month: '2-digit', day: '2-digit',
    });
  } catch { return dateStr; }
}
```

beforeDestroy.js

```
// scripts/beforeDestroy.js
if (this.destroyPopup) this.destroyPopup();
```

Step 9: 결과 흐름 요약

```
3D 오브젝트 클릭
  → bind3DEvents가 '@assetClicked' 이벤트 수신
  → showDetail() 호출
    → showPopup()
      → Shadow DOM 생성 (최초 1회)
      → onPopupCreated() 콜백
        → createChart('.chart-container')
        → bindPopupEvents({ click: { ... } })
    → fetchDatasetAndRender() × N
      → fetchData() → renderBasicInfo() / renderTrendChart()
    → setInterval (refreshInterval > 0인 데이터셋)

닫기 버튼 클릭
  → hideDetail()
    → stopRefresh() - 모든 interval 정리
    → hidePopup()
      → destroyPopup() (확장된 버전)
        → stopRefresh() - 이중 안전
        → ECharts dispose + ResizeObserver disconnect
        → 이벤트 리스너 제거
        → Shadow DOM 호스트 제거
```

---

3. HeatmapMixin 튜토리얼

PopupMixin 컴포넌트에 3D 히트맵 기능을 추가하는 방법입니다.

어떤 컴포넌트에 HeatmapMixin이 필요한가?

컴포넌트 유형	PopupMixin	HeatmapMixin	예시
장비 상세 팝업 (차트/테이블)	O	X	PDU, UPS, SWBD
온/습도 관련 장비 (3D 온도맵 필요)	O	O	CRAC, TempHumiditySensor
대시보드 패널 (라벨 + 히트맵 토글)	X	O	ActionPanel

Step 1: Mixin 임포트 추가

```
const { applyShadowPopupMixin, applyEChartsMixin } = PopupMixin;
const { applyHeatmapMixin } = HeatmapMixin;  // ← 추가
```

Step 2: applyHeatmapMixin 호출

`applyShadowPopupMixin` + `applyEChartsMixin` 이후, `destroyPopup` 확장 이전에 호출합니다.

```
  // ... applyShadowPopupMixin + applyEChartsMixin 이후 ...

  // 3D 히트맵 서피스
```

```
  applyHeatmapMixin(this, {
    surfaceSize: { width: 20, depth: 20 },           // 또는 'auto'
    temperatureMetrics: ['SENSOR.TEMP', 'CRAC.RETURN_TEMP'],
  });

  // destroyPopup 확장 (변경 없음)
  const _origDestroyPopup = this.destroyPopup;
  const _ctx = this;
  this.destroyPopup = function () {
    _ctx.stopRefresh();
    _origDestroyPopup.call(_ctx);
  };
```

Step 3: 팝업 UI에 토글 버튼 추가

views/component.html의 팝업 내부에 히트맵 버튼을 추가합니다:

```
<button class="heatmap-btn">온도맵</button>
```

popupCreatedConfig에 이벤트 바인딩을 추가합니다:

```
  const popupCreatedConfig = {
    chartSelector: this.config.chart.selectors.container,
    events: {
      click: {
        '.close-btn': () => this.hideDetail(),
        '.tab-btn': (e) => switchTab.call(this, e.target.dataset.tab),
        '.heatmap-btn': () => this.toggleHeatmap(),  // ← 추가
      },
    },
  };
```

Step 4: 제공되는 메서드

`applyHeatmapMixin` 적용 후 인스턴스에 추가되는 메서드:

메서드	설명
`toggleHeatmap()`	ON/OFF 토글
`updateHeatmapWithData(dataPoints)`	외부에서 데이터 직접 주입
`updateHeatmapConfig(newOptions)`	런타임 옵션 변경
`destroyHeatmap()`	히트맵 정리

Step 5: 옵션 커스텀

기본값으로 충분한 경우가 많지만, 필요 시 옵션을 추가합니다:

```
applyHeatmapMixin(this, {
  // 필수
  temperatureMetrics: ['SENSOR.TEMP', 'CRAC.RETURN_TEMP'],

  // 서피스 크기 (기본: 'auto' - BoundingBox 기반 자동 계산)
  surfaceSize: { width: 20, depth: 20 },
```

```
  // 온도 범위 (기본: { min: 17, max: 31 })
  temperatureRange: { min: 20, max: 28 },

  // 색상 그라디언트 (기본: null → 내장 DEFAULT_GRADIENT)
  gradient: {
    0.0: '#0000FF',   // 최저
    0.5: '#00FF00',   // 중간
    1.0: '#FF0000',   // 최고
  },

  // 시각 효과
  displacementScale: 3,   // 높이 변위 (0이면 평면)
  baseHeight: 2,          // 기준 높이
  opacity: 0.75,          // 투명도

  // 데이터 갱신
  refreshInterval: 0,     // 0: 외부 주입 모드 / > 0: 자체 타이머
  onLoadingChange: (isLoading) => { /* 로딩 UI 처리 */ },
});
```

> *전체 옵션 상세는 HEATMAP_MIXIN_API.md 참조.*

---

4. 주의사항 및 트러블슈팅

Mixin 적용 순서

```
반드시 이 순서를 따릅니다:

① applyShadowPopupMixin   ← 첫 번째 (다른 Mixin의 전제조건)
② applyEChartsMixin       ← Shadow DOM 필요
③ applyTabulatorMixin      ← Shadow DOM 필요
④ applyHeatmapMixin        ← 독립적이지만 destroyPopup 확장 전에 호출
⑤ destroyPopup 확장        ← 마지막
```

순서를 어기면 `this.createChart`, `this.popupQuery` 등 메서드가 존재하지 않아 런타임 에러가 발생합니다.

HeatmapMixin 초기화 타이밍

```
applyHeatmapMixin(this, options)
  → toggleHeatmap(), updateHeatmapWithData() 등 메서드 바인딩
  → 아직 히트맵이 활성화된 것은 아님

사용자가 버튼 클릭
  → toggleHeatmap() 호출
  → 이 시점에서 Three.js 서피스 생성 + 렌더링 시작
```

`updateHeatmapConfig()` 는 `toggleHeatmap()` 이후에만 동작합니다. 활성화 전에 호출하면 내부 객체가 없어서 무시됩니다.

---

destroyPopup 확장 패턴

여러 Mixin이 `destroyPopup` 을 체인 확장합니다. 반드시 **현재** `this.destroyPopup` **을 캡처한 후** 래핑합니다:

```
// ✅ 올바른 패턴
const _origDestroyPopup = this.destroyPopup;  // 현재 시점의 함수 캡처
const _ctx = this;
this.destroyPopup = function () {
  _ctx.stopRefresh();              // 커스텀 정리
  _origDestroyPopup.call(_ctx);    // 원래 정리 체인 호출
};

// ❌ 잘못된 패턴 - 무한 재귀
this.destroyPopup = function () {
  this.stopRefresh();
  this.destroyPopup();  // 자기 자신 호출 → 무한 루프
};
```

팝업은 동시에 1개만

`showPopup()` 호출 시 이전에 열린 다른 컴포넌트의 팝업이 자동으로 닫힙니다. 이것은 의도된 동작입니다 (`_activePopupInstance` 싱글톤 관리).

Shadow DOM 내부 요소 접근

일반 `document.querySelector` 로는 Shadow DOM 내부 요소에 접근할 수 없습니다:

```
// ❌ null 반환
document.querySelector('.chart-container');

// ✅ Shadow DOM 내부 쿼리
this.popupQuery('.chart-container');
this.popupQueryAll('.tab-btn');
```

---

refreshInterval 설계

값	동작	용도
0	showDetail() 시 1회만 fetch	자산 기본정보 (거의 안 바뀜)
5000	5초마다 fetch	실시간 메트릭, 트렌드 차트
30000	30초마다 fetch	히트맵 데이터 (ActionPanel 통합 타이머)

`hideDetail()` → `stopRefresh()` 로 모든 interval이 정리됩니다. `destroyPopup` 확장에서도 `stopRefresh()` 를 호출하므로 이중 안전장치가 됩니다.

Tabulator CSS 자동 주입

`applyTabulatorMixin` 사용 시 Tabulator CSS가 Shadow DOM에 자동 주입됩니다:

- 경로: `client/common/libs/tabulator/tabulator_midnight.min.css`
- 테마: midnight (다크 모드)
- 최초 `createTable()` 호출 시 1회만 fetch

CSS 파일이 없거나 경로가 잘못되면 테이블이 스타일 없이 렌더링됩니다. 개발 서버에서 해당 경로가 접근 가능한지 확인하세요.

관련 문서

문서	내용
POPUP_MIXIN_API.md	PopupMixin 전체 메서드 레퍼런스
HEATMAP_MIXIN_API.md	HeatmapMixin 옵션 상세
COMPONENT_DEVELOPMENT_GUIDE.md	컴포넌트 개발 가이드
WKIT_API.md	bind3DEvents, fetchData 레퍼런스
FX_API.md	fx 함수형 유틸리티
WHY_CONFIG.md	Config 패턴 설계 이유
INSTANCE_LIFECYCLE_GC.md	메모리 관리

최종 업데이트: 2026-02-25