DXnavis 플러그인 분석 리포트

분석일: 2025-11-20
플러그인 버전: v2.0 (API 통합)
대상: Navisworks 2025 Add-in

📋 목차

🎯 개요

프로젝트 정보

- **이름**: DXnavis
- **유형**: Navisworks Manage 2025 Add-in Plugin
- **프레임워크**: .NET Framework 4.8
- **UI 기술**: WPF (XAML)
- **패턴**: MVVM (Model-View-ViewModel)

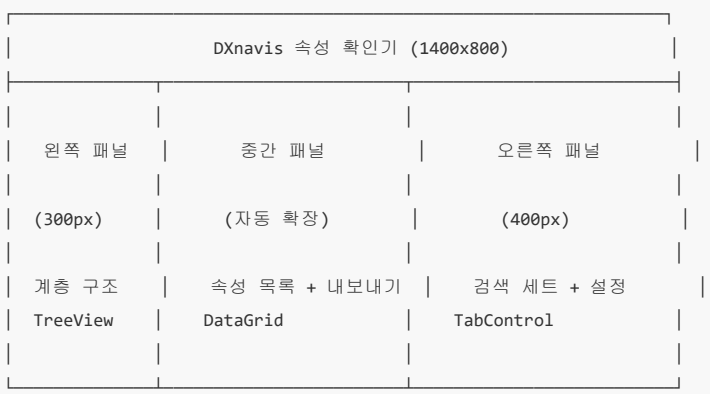
주요 기능

1. ✅ Navisworks 모델 계층 구조 시각화
2. ✅ 객체 속성 실시간 표시 및 검색
3. ✅ 계층 구조 CSV/JSON 로컬 저장 (v1.0)
4. ✅ API 서버 연동 및 자동 업로드 (v2.0)
5. ✅ 검색 세트 자동 생성
6. ✅ 프로젝트 자동 감지 및 리비전 관리

🎨 UI 구조 분석

레이아웃 구성

DXnavis는 **3패널 레이아웃**을 사용합니다:



1️⃣ 왼쪽 패널: 객체 계층 구조

파일: `Views/DXwindow.xaml`

구성 요소: - **TreeView**: 계층적 객체 표시 - 체크박스로 객체 선택 - `ObjectHierarchyRoot` 바인딩 - `Children` 재귀적 렌더링 - **버튼**: "모델 계층 로드" - Command: `LoadHierarchyCommand`

데이터 바인딩:

```
<TreeView ItemsSource="{Binding ObjectHierarchyRoot}">
    <HierarchicalDataTemplate ItemsSource="{Binding Children}">
        <CheckBox IsChecked="{Binding IsSelected, Mode=TwoWay}"/>
        <TextBlock Text="{Binding DisplayName}"/>
    </HierarchicalDataTemplate>
</TreeView>
```

2️⃣ 중간 패널: 속성 목록 및 내보내기

파일: `Views/DXwindow.xaml`

구성 요소:

A. 속성 DataGrid

```
<DataGrid ItemsSource="{Binding AllHierarchicalProperties}">
    <Columns>
        - 선택 (CheckBox)
        - 객체 (DisplayName)
        - 카테고리 (Category)
        - 속성 (PropertyName)
        - 값 (PropertyValue)
        - 권한 (ReadWriteStatus)
    </Columns>
</DataGrid>
```

B. v1.0 로컬 내보내기 섹션

- **버튼**: "전체 속성 CSV 저장" → `ExportAllToCsvCommand`
- **버튼**: "계층 구조 내보내기" → `ExportSelectionHierarchyCommand`

C. v2.0 API 서버 업로드 섹션 (신규)

- **프로젝트 정보 표시**:
- 프로젝트 코드: `{Binding ProjectCode}`
- 현재 리비전: `{Binding CurrentRevisionDisplay}`
- **버튼**: "🔍 프로젝트 감지 (CSV 선택)" → `DetectProjectCommand`
- **리비전 정보 입력**:
- 버전 태그: `{Binding VersionTag}`
- 설명: `{Binding RevisionDescription}`
- **버튼**: "⬆️ API 서버로 업로드" → `UploadToApiCommand`
- 활성화 조건: `IsEnabled="{Binding CanUpload}"`
- **ProgressBar**: 업로드 진행률 표시

3️⃣ 오른쪽 패널: TabControl

파일: *`Views/DXwindow.xaml`*

Tab 1: 🔍 검색 세트 생성

- **선택된 속성 개수**: `{Binding SelectedPropertiesCount}`
- **입력 필드**:
- 폴더 이름: `{Binding FolderName}`
- 세트 이름: `{Binding NewSetName}`
- **미리보기**: `{Binding SelectedPropertyInfo}`
- **버튼**: "검색 세트 생성" → `CreateSearchSetCommand`

Tab 2: ⚙️ 설정

- **API 서버 URL**: `{Binding ApiServerUrl}` (예: http://localhost:8000)
- **기본 사용자명**: `{Binding DefaultUsername}`
- **타임아웃 (초)**: `{Binding TimeoutSeconds}` (기본: 30)
- **배치 크기**: `{Binding BatchSize}` (기본: 100)
- **설정 파일 경로**: `{Binding SettingsFilePath}`
- **버튼**: "💾 설정 저장" → `SaveSettingsCommand`

🧩 ViewModel 분석

DXwindowViewModel

파일: *`ViewModels/DXwindowViewModel.cs`*
라인 수: 1,267줄
메서드 수: 22개

주요 프로퍼티 (INotifyPropertyChanged)

프로퍼티	타입	설명
`ObjectHierarchyRoot`	`ObservableCollection<HierarchyNodeViewModel>`	TreeView 루트 노드
`AllHierarchicalProperties`	`ObservableCollection<PropertyItemViewModel>`	DataGrid 속성 목록
`SelectedPropertiesCount`	`string`	선택된 속성 개수 표시
`ProjectCode`	`string`	v2.0: 감지된 프로젝트 코드
`CurrentRevisionDisplay`	`string`	v2.0: 현재 리비전 표시
`ExportStatusMessage`	`string`	진행 상태 메시지
`ExportProgressPercentage`	`double`	진행률 (0-100)
`IsUploading`	`bool`	업로드 중 플래그
`CanUpload`	`bool`	업로드 가능 여부
`ApiServerUrl`	`string`	설정: API 서버 URL
`DefaultUsername`	`string`	설정: 사용자명
`TimeoutSeconds`	`int`	설정: 타임아웃
`BatchSize`	`int`	설정: 배치 크기

핵심 메서드

1. 초기화 및 모니터링

- `DXwindowViewModel()` : 생성자, Command 초기화
- `StartMonitoring()` : Navisworks 선택 변경 모니터링 시작
- `StopMonitoring()` : 모니터링 중지

2. 객체 선택 및 속성 로드

- `OnSelectionChanged()` : 선택 변경 이벤트 (Debouncing 패턴)
- `LoadSelectedObjectProperties()` : 선택 객체 속성 UI 표시

3. 계층 구조 로드 및 TreeView

- `LoadModelHierarchyAsync()` : 전체 모델 계층 TreeView 로드
- `OnTreeNodeSelectionChanged()` : TreeView 노드 선택 시 속성 업데이트

4. 로컬 파일 내보내기 (v1.0)

- `ExportAllToCsvAsync()` : 전체 모델 속성 CSV 저장
- `ExportSelectionHierarchyAsync()` : 선택 객체 계층 구조 CSV/JSON 저장
- `SaveToFile()` : UI 속성 파일 저장

5. API 서버 연동 (v2.0)

- `DetectProjectFromCsvAsync()` : CSV에서 프로젝트 자동 감지
- `UploadHierarchyToApiAsync()` : API 서버에 계층 구조 업로드

6. 검색 세트 생성

- `CreateSearchSetFromSelectedProperty()` : 선택된 속성으로 Navisworks 검색 세트 생성

7. 설정 관리

- `LoadSettings()` : JSON 설정 파일 로드
- `SaveSettings()` : JSON 설정 파일 저장

🛠️ 서비스 레이어 분석

1. NavisworksDataExtractor

파일: `Services/NavisworksDataExtractor.cs`
라인 수: 470줄
메서드 수: 12개

핵심 역할: Navisworks 모델에서 계층 구조 및 속성 데이터 추출

주요 메서드

메서드	설명
`TraverseAndExtractProperties()`	재귀적 계층 탐색 및 속성 추출 (EAV 패턴)
`GetDisplayName()`	ModelItem 표시 이름 추출 헬퍼
`ExtractHierarchicalRecordsFromSelection()`	선택 객체로부터 `HierarchicalPropertyRecord` 일괄 추출
`ExtractHierarchy()`	TreeView 바인딩용 `HierarchyNodeViewModel` 생성
`ConvertToHierarchyNode()`	ModelItem → HierarchyNodeViewModel 재귀 변환
`ExtractProperties()`	단일 ModelItem 속성 리스트 추출
`FindModelItemById()`	ObjectId로 ModelItem 검색 (전체 모델 순회)

데이터 구조

HierarchicalPropertyRecord (EAV 패턴):

```
{
    ObjectId: Guid,
    ParentId: Guid,
    Level: int,
    DisplayName: string,
    Category: string,
    PropertyName: string,
    PropertyValue: string,
    ReadWriteStatus: string
}
```

2. HierarchyUploader

파일: `Services/HierarchyUploader.cs`
라인 수: 700줄
메서드 수: 27개

핵심 역할: v2.0 API 서버 통신 및 프로젝트 자동 감지

Phase E.1: TDD 구현 메서드

메서드	TDD 상태	설명
`SampleObjectIdsFromCsv()`	✅ Tested	CSV에서 최대 100개 객체 ID 샘플링
`StripPrefixes()`	✅ Tested	`DisplayString:`, `NamedConstant:` 접두사 제거
`DetectProject()`	✅ Tested	프로젝트 탐지 API 호출 (confidence threshold)

API 연동 메서드

메서드	HTTP Method	Endpoint	설명
`DetectProjectCodeFromCsv()`	-	-	CSV에서 Revit 소스 파일명 추출 → 프로젝트 코드 생성
`CheckProjectExistsAsync()`	GET	`/api/v1/projects/{code}`	프로젝트 존재 여부 확인
`GetLatestNavisworksRevisionAsync()`	GET	`/api/v1/projects/{code}/revisions`	최신 Navisworks 리비전 조회
`CreateRevisionAsync()`	POST	`/api/v1/projects/{code}/revisions`	새 리비전 생성
`UploadHierarchyCsvAsync()`	POST	`/api/v1/navisworks/projects/{code}/revisions/{num}/hierarchy`	CSV 파일 직접 업로드
`ExecuteFullWorkflowAsync()`	-	-	전체 워크플로우 orchestration

헬퍼 메서드

- `ParseCsvLine()` : CSV 라인 파싱 (따옴표 처리)
- `FindColumnIndex()` : 컬럼 인덱스 찾기 (여러 가능한 이름 지원)
- `GenerateProjectCode()` : 파일명 → 프로젝트 코드 변환
- `SanitizePropertyValue()` : 속성 값 정규화

3. FullModelExporterService

파일: `Services/FullModelExporterService.cs`

핵심 역할: 전체 모델 데이터 CSV 내보내기

4. HierarchyFileWriter

파일: `Services/HierarchyFileWriter.cs`

핵심 역할: 계층 구조 데이터 CSV/JSON 파일 작성

5. PropertyFileWriter

파일: `Services/PropertyFileWriter.cs`

핵심 역할: 속성 데이터 파일 작성

6. SetCreationService

파일: `Services/SetCreationService.cs`

핵심 역할: Navisworks 검색 세트 자동 생성

🔗 외부 연결 요소

1. DXBase 라이브러리 의존성

파일: `DXnavis.csproj`

```
<Reference Include="DXBase">
    <HintPath>..\DXBase\bin\Debug\netstandard2.0\DXBase.dll</HintPath>
</Reference>
```

사용 서비스: - `DXBase.Services.ConfigurationService` : 설정 파일 로드/저장 `csharp var settings = ConfigurationService.LoadSettings(); ConfigurationService.SaveSettings(settings);`

설정 파일 경로:

```
%AppData%\Roaming\DX_Platform\Config\settings.json
```

설정 모델:

```
{
    ApiServerUrl: "http://localhost:8000",
    DefaultUsername: "yoon",
    TimeoutSeconds: 30,
    BatchSize: 100
}
```

2. Navisworks API

참조 DLL:

```
<Reference Include="Autodesk.Navisworks.Api" />
<Reference Include="Autodesk.Navisworks.Automation" />
<Reference Include="Autodesk.Navisworks.Clash" />
<Reference Include="Autodesk.Navisworks.ComApi" />
<Reference Include="Autodesk.Navisworks.Controls" />
<Reference Include="Autodesk.Navisworks.Timeliner" />
```

주요 사용 클래스:

클래스	용도
`Autodesk.Navisworks.Api.Application`	현재 문서 접근
`Autodesk.Navisworks.Api.Document`	모델 데이터
`Autodesk.Navisworks.Api.ModelItem`	계층 객체
`Autodesk.Navisworks.Api.PropertyCategoryCollection`	속성 카테고리
`Autodesk.Navisworks.Api.DataProperty`	개별 속성
`Autodesk.Navisworks.Api.DocumentSelectionChanged`	선택 변경 이벤트
`Autodesk.Navisworks.Api.Search.SearchCondition`	검색 조건 생성
`Autodesk.Navisworks.Api.Clash.DocumentClash`	Search Sets 관리

3. FastAPI 서버 통신

HTTP 클라이언트: `System.Net.Http.HttpClient`

API 엔드포인트:

Endpoint	Method	Request Body	Response	설명
`/api/v1/projects/{code}`	GET	-	`ProjectResponse`	프로젝트 조회
`/api/v1/projects`	POST	`ProjectInfo`	`ProjectResponse`	프로젝트 생성
`/api/v1/projects/{code}/revisions`	GET	-	`List<RevisionInfo>`	리비전 목록
`/api/v1/projects/{code}/revisions`	POST	`RevisionInfo`	`RevisionResponse`	리비전 생성
`/api/v1/navisworks/projects/{code}/revisions/{num}/hierarchy`	POST	CSV File (Multipart)	`UploadResponse`	계층 업로드
`/api/v1/projects/detect-by-objects`	POST	`{objects: [unique_key]}`	`DetectResponse`	프로젝트 감지

요청 예시:

```
// 프로젝트 생성
var projectInfo = new {
    code = "PIPE_TEST",
    name = "배관테스트",
    created_by = "yoon"
};
var response = await httpClient.PostAsJsonAsync("/api/v1/projects", projectInfo);

// CSV 업로드
var content = new MultipartFormDataContent();
content.Add(new StreamContent(csvFile), "file", "hierarchy.csv");
await httpClient.PostAsync($"/api/v1/navisworks/projects/{code}/revisions/{num}/hierarchy", content);
```

4. 파일 시스템 연동

CSV 저장 경로:

```
// 기본 경로
string outputDir = Path.Combine(
    Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments),
    "DXnavis_Exports"
);

// 파일명 패턴
string fileName = $"hierarchy_{DateTime.Now:yyyyMMdd_HHmmss}.csv";
```

로그 파일:

```
DXnavis/
├── LOG-debug/
├── LOG-error/
└── LOG-prd/
```

🔄 데이터 플로우

워크플로우 1: 계층 구조 로드 및 표시

```
sequenceDiagram
    participant User
    participant UI as DXwindow
    participant VM as DXwindowViewModel
    participant Extractor as NavisworksDataExtractor
    participant API as Navisworks API

    User->>UI: "모델 계층 로드" 클릭
    UI->>VM: LoadHierarchyCommand
    VM->>API: Application.ActiveDocument
    API-->>VM: Document
    VM->>Extractor: ExtractHierarchy(rootItems)

    loop 모든 ModelItem
        Extractor->>API: item.Children 순회
        Extractor->>Extractor: ConvertToHierarchyNode(item)
    end

    Extractor-->>VM: List<HierarchyNodeViewModel>
    VM->>VM: ObjectHierarchyRoot = nodes
    VM-->>UI: PropertyChanged
    UI-->>User: TreeView 업데이트
```

워크플로우 2: v2.0 API 서버 업로드

```
sequenceDiagram
    participant User
    participant VM as DXwindowViewModel
    participant Uploader as HierarchyUploader
    participant FileWriter as HierarchyFileWriter
    participant API as FastAPI Server
    participant DB as PostgreSQL

    User->>VM: "🔍 프로젝트 감지" 클릭
    VM->>FileWriter: CSV 파일 선택
    VM->>Uploader: DetectProjectCodeFromCsv(csvPath)
    Uploader->>Uploader: CSV 파싱 (소스 파일명 찾기)
    Uploader->>Uploader: GenerateProjectCode("배관테스트.rvt")
    Uploader-->>VM: ProjectCode = "PIPE_TEST"
    VM->>VM: CanUpload = true

    User->>VM: 버전 태그 입력 (v1.0)
    User->>VM: "⬆️ API 업로드" 클릭
    VM->>Uploader: ExecuteFullWorkflowAsync()

    Uploader->>API: GET /projects/PIPE_TEST
    alt 프로젝트 없음
        Uploader->>API: POST /projects (생성)
        API->>DB: INSERT INTO projects
    end

    Uploader->>API: POST /projects/PIPE_TEST/revisions
    API->>DB: INSERT INTO revisions
    API-->>Uploader: revision_id, revision_number

    Uploader->>API: POST /navisworks/.../hierarchy (CSV Upload)
    API->>API: CSV 파싱 및 EAV 집계
    API->>DB: INSERT INTO unified_objects (batch)
    API-->>Uploader: {objects_count: 4317}

    Uploader-->>VM: WorkflowResult
    VM-->>User: "✅ 업로드 완료"
```

워크플로우 3: 검색 세트 생성

```
sequenceDiagram
    participant User
    participant VM as DXwindowViewModel
    participant Service as SetCreationService
    participant API as Navisworks API

    User->>VM: DataGrid에서 속성 체크
    VM->>VM: AllHierarchicalProperties 업데이트
    VM->>VM: SelectedPropertiesCount 갱신

    User->>VM: 폴더/세트 이름 입력
    User->>VM: "검색 세트 생성" 클릭
    VM->>Service: CreateSearchSet(folderName, setName, selectedProps)

    Service->>API: DocumentClash.SearchSets
    Service->>API: 폴더 생성 또는 기존 폴더 찾기
    Service->>API: 검색 조건 생성 (PropertyCondition)
    Service->>API: Search.FindAll(conditions)
    Service->>API: SavedItem 추가

    Service-->>VM: 생성된 세트 개수
    VM-->>User: "✅ 검색 세트 생성 완료"
```

🔌 통합 포인트

1. 플러그인 진입점

파일: `DX.cs`

```
[Plugin("DXnavis.DX", "YourCompany",
    DisplayName = "DXnavis 속성 확인기",
    ToolTip = "선택된 객체의 속성을 실시간으로 표시합니다")]
[AddInPlugin(AddInLocation.AddIn, LoadForCanExecute = true)]
public class DX : AddInPlugin
{
    private static Views.DXwindow _windowInstance; // 싱글톤

    public override int Execute(params string[] parameters)
    {
        // 창 싱글톤 관리
        if (_windowInstance != null && _windowInstance.IsLoaded)
        {
            _windowInstance.Activate();
            return 0;
        }

        _windowInstance = new Views.DXwindow();
        _windowInstance.Show(); // 모달리스
        return 0;
    }
}
```

특징: - **싱글톤 패턴**: 중복 창 방지 - **모달리스 창**: Navisworks 작업 계속 가능 - **DEBUG 모드**: 개발 시 항상 새 창 생성

2. 외부 라이브러리 통합

NuGet 패키지:

```
<PackageReference Include="Newtonsoft.Json" Version="13.0.4" />
<PackageReference Include="System.Text.Json" Version="7.0.0" />
<PackageReference Include="Microsoft.Bcl.AsyncInterfaces" Version="7.0.0" />
```

3. DXBase 통합

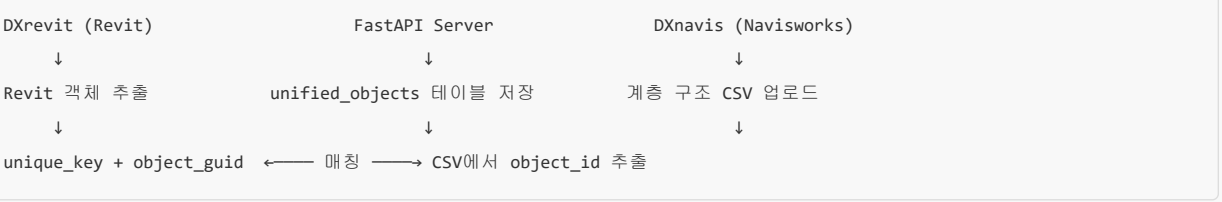
설정 로드 예시:

```
using DXBase.Services;

var settings = ConfigurationService.LoadSettings();
ApiServerUrl = settings.ApiServerUrl ?? "http://localhost:8000";
DefaultUsername = settings.DefaultUsername ?? Environment.UserName;
```

4. DXrevit 연동 (간접)

DXnavis와 DXrevit은 **FastAPI 서버를 통해 간접 연동**됩니다:



매칭 메커니즘: - Revit: `unique_key` 와 `object_guid` 제공 - Navisworks: CSV에서 `ObjectId` (GUID) 제공 - 서버: `object_guid` ↔ CSV `ObjectId` 매칭

📊 아키텍처 다이어그램

컴포넌트 다이어그램

```
graph TB
    subgraph "DXnavis Plugin (.NET Framework 4.8)"
        A[DX.cs<br/>Entry Point]
        B[Views/DXwindow.xaml<br/>WPF UI]
        C[DXwindowViewModel<br/>MVVM ViewModel]

        subgraph "Services Layer"
            D[NavisworksDataExtractor]
            E[HierarchyUploader]
            F[SetCreationService]
            G[FullModelExporterService]
            H[HierarchyFileWriter]
        end

        subgraph "Models"
            I[HierarchyNodeViewModel]
            J[PropertyItemViewModel]
            K[HierarchicalPropertyRecord]
        end
    end

    subgraph "External Dependencies"
        L[DXBase<br/>.NET Standard 2.0]
        M[Navisworks API<br/>COM Interop]
        N[FastAPI Server<br/>HTTP REST]
        O[File System<br/>CSV/JSON]
    end

    A --> B
    B --> C
    C --> D
    C --> E
    C --> F
    C --> G
    C --> H

    D --> M
    F --> M
    E --> N
    G --> O
    H --> O

    C --> L
    E --> L

    D --> I
    D --> J
    D --> K

    style A fill:#ff6b6b
```

```
    style C fill:#4ecdc4
    style E fill:#45b7d1
    style M fill:#ffa07a
    style N fill:#98d8c8
```

데이터 플로우 다이어그램

```
flowchart LR
    A[Navisworks 모델] -->|API 호출| B[NavisworksDataExtractor]
    B -->|계층 구조| C[HierarchyNodeViewModel]
    B -->|속성 리스트| D[PropertyItemViewModel]

    C --> E[TreeView UI]
    D --> F[DataGrid UI]

    F -->|선택| G[SetCreationService]
    G -->|생성| H[Navisworks Search Set]

    F -->|내보내기| I[HierarchyFileWriter]
    I --> J[CSV/JSON 파일]

    J -->|업로드| K[HierarchyUploader]
    K -->|HTTP POST| L[FastAPI Server]
    L --> M[(PostgreSQL DB)]

    style A fill:#00aa00
    style L fill:#0066cc
    style M fill:#336699
```

📝 요약

핵심 강점

1. ✅ **MVVM 패턴**: UI와 비즈니스 로직 완전 분리
2. ✅ **TDD 방식**: Phase E.1 핵심 메서드 테스트 완료
3. ✅ **v1.0 + v2.0 병행**: 로컬 파일 + API 서버 모두 지원
4. ✅ **Debouncing 패턴**: 빠른 선택 변경 시 성능 최적화
5. ✅ **싱글톤 패턴**: 중복 창 방지
6. ✅ **프로젝트 자동 감지**: CSV에서 Revit 소스 파일 추출

주요 통합 포인트

통합 대상	인터페이스	방향
DXBase	`ConfigurationService`	DXnavis → DXBase
Navisworks API	COM Interop	DXnavis → Navisworks
FastAPI Server	HTTP REST	DXnavis → FastAPI
File System	System.IO	DXnavis → File
DXrevit (간접)	FastAPI (unified_objects)	DXnavis ↔ FastAPI ↔ DXrevit

개선 제안

1. 🔄 **실시간 동기화**: SignalR을 통한 양방향 통신
2. 📊 **진행률 개선**: 더 세밀한 진행률 표시
3. 🔍 **검색 기능**: TreeView 내 객체 검색
4. 📦 **배치 업로드**: 여러 CSV 파일 동시 업로드
5. 🎨 **UI 개선**: Material Design 적용

문서 작성: 2025-11-20
분석 범위: DXnavis 전체 아키텍처
버전: v2.0 (API 통합)