AIGenStudio - Gemini CLI 완벽 가이드

프로젝트 개요

제공된 프로젝트 구조를 기반으로 모듈식 확장 가능한 AI 이미지 생성 플랫폼을 구축합니다.

기존 프로젝트 구조 (제공된 파일 기반)

```
aigen-studio/
├── .gitignore
├── .venv/
├── pyproject.toml
├── README.md
├── docs/
│   ├── api.md
│   ├── architecture.md
│   └── user_guide.md
├── models/                 # 모델 저장소 (ComfyUI 방식)
│   ├── Checkpoint/
│   │   ├── SD15/
│   │   ├── SDXL/
│   │   └── FLUX/
│   ├── LoRA/
│   │   ├── Characters/
│   │   ├── Styles/
│   │   ├── Concepts/
│   │   └── Poses/
│   ├── VAE/
│   ├── Embeddings/
│   ├── ControlNet/
│   └── Upscaler/
├── wildcards/              # 와일드카드 텍스트 파일
│   ├── animals.txt
│   ├── colors.txt
│   ├── styles.txt
│   └── objects.txt
├── outputs/                # 생성된 이미지 저장
│   ├── txt2img/
│   ├── img2img/
│   └── extras/
├── src/
│   └── aigen_studio/
│       ├── __init__.py
│       ├── main.py         # 애플리케이션 진입점
│       ├── api/            # FastAPI 엔드포인트
│       ├── core/           # 핵심 시스템
│       ├── models/         # 모델 관리 시스템
│       ├── llm/            # LLM 통합 시스템
│       ├── generation/     # 이미지 생성 엔진
│       ├── wildcards/      # 와일드카드 시스템
│       ├── embeddings/     # 임베딩 시스템
│       ├── ui/             # UI 컴포넌트
│       ├── pages/          # 페이지 모듈
│       ├── services/       # 비즈니스 로직
│       ├── state/          # 상태 관리
│       └── assets/         # 정적 파일
└── tests/                  # 테스트
```

모듈화 리팩토링 전략

1. 핵심 시스템 분리 (코어만 남기기)

src/aigen_studio/core/extension_system.py

python

```python
"""확장 시스템 - 기존 page_manager.py를 확장 가능하게 개선"""

import importlib
import asyncio
from pathlib import Path
from typing import Dict, List, Optional, Any
import json
from abc import ABC, abstractmethod

class ExtensionBase(ABC):
    """모든 확장의 기본 클래스"""

    def __init__(self):
        self.enabled = True
        self.config = {}
        self.dependencies = []

    @abstractmethod
    async def initialize(self, app) -> None:
        """확장 초기화"""
        pass

    @abstractmethod
    async def cleanup(self) -> None:
        """확장 정리"""
        pass

    def get_pages(self) -> Dict[str, Any]:
        """페이지 반환 - 기존 pages/ 구조 호환"""
        return {}

    def get_api_routes(self) -> List[Any]:
        """API 라우트 반환"""
        return []

    def get_ui_components(self) -> Dict[str, Any]:
        """UI 컴포넌트 반환"""
        return {}

class ExtensionManager:
    """확장 관리자 - 기존 page_manager.py 대체"""

    def __init__(self, app):
        self.app = app
        self.extensions: Dict[str, ExtensionBase] = {}
        self.extension_paths = [
            Path("src/aigen_studio/extensions"),  # 내장 확장
```

```
        Path("extensions")  # 사용자 확장
    ]

async def discover_and_load_extensions(self):
    """확장 자동 탐색 및 로드"""
    # 1. 내장 모듈들을 확장으로 변환
    await self._load_builtin_extensions()

    # 2. 외부 확장 로드
    for ext_path in self.extension_paths:
        if ext_path.exists():
            await self._scan_extension_directory(ext_path)

async def _load_builtin_extensions(self):
    """기존 모듈들을 확장으로 로드"""
    builtin_modules = {
        "model_manager": "aigen_studio.models",
        "llm_integration": "aigen_studio.llm",
        "wildcard_system": "aigen_studio.wildcards",
        "embedding_system": "aigen_studio.embeddings",
        "generation_engine": "aigen_studio.generation"
    }

    for name, module_path in builtin_modules.items():
        extension_class = self._create_extension_wrapper(module_path)
        if extension_class:
            self.extensions[name] = extension_class(self.app)
            await self.extensions[name].initialize(self.app)
```

2. 기존 모듈을 확장으로 변환

src/aigen_studio/extensions/model_manager/extension.py

python

```python
"""모델 관리자 확장 - 기존 models/ 모듈 래핑"""

from aigen_studio.core.extension_system import ExtensionBase
from aigen_studio.models.manager import ModelManager
from aigen_studio.models.scanner import ModelScanner
from aigen_studio.ui.components.model_selector import ModelSelector

class ModelManagerExtension(ExtensionBase):
    """모델 관리 확장"""

    def __init__(self):
        super().__init__()
        self.manager = None
        self.scanner = None

    async def initialize(self, app) -> None:
        """초기화"""
        settings = app.settings

        # 기존 모델 매니저 사용
        self.manager = ModelManager(settings.models_path)
        self.scanner = ModelScanner(settings.models_path)

        # 상태 저장소에 등록
        app.state.set_managers(model_manager=self.manager)

        # 백그라운드 스캔 시작
        asyncio.create_task(self._auto_scan_models())

    async def _auto_scan_models(self):
        """주기적 모델 스캔"""
        while True:
            await self.manager.scan_models()
            await asyncio.sleep(300)  # 5분마다

    def get_ui_components(self) -> Dict[str, Any]:
        """UI 컴포넌트 제공"""
        return {
            "model_selector": ModelSelector
        }

    def get_api_routes(self) -> List[Any]:
        """API 라우트 제공"""
        from aigen_studio.api.models import router
        return [router]
```

3. Canvas 확장 (새로 추가)

extensions/canvas-editor/

```
canvas-editor/
├── extension.json
├── __init__.py
├── extension.py
├── requirements.txt
├── static/
│   ├── js/
│   │   ├── canvas-core.js
│   │   ├── tools/
│   │   │   ├── brush.js
│   │   │   ├── shapes.js
│   │   │   └── eraser.js
│   │   └── layers/
│   │       └── layer-manager.js
│   └── css/
│       └── canvas.css
└── components/
    └── canvas_ui.py
```

extension.json

json

```
{
  "name": "Canvas Editor",
  "version": "1.0.0",
  "description": "Layer-based canvas editor for AI image generation",
  "author": "AIGenStudio",
  "main": "extension.py",
  "dependencies": [],
  "requirements": ["pillow>=10.0.0"]
}
```

static/js/canvas-core.js

javascript

```javascript
/**
 * AIGenStudio Canvas Editor Core
 * 기존 NiceGUI와 통합
 */

class AIGenCanvas {
  constructor(containerId, options = {}) {
    this.container = document.getElementById(containerId);
    this.options = {
      width: options.width || 512,
      height: options.height || 512,
      ...options
    };

    this.layers = [];
    this.activeLayer = null;
    this.tools = {};
    this.history = [];
    this.maxHistory = 50;

    this.init();
  }

  init() {
    // 캔버스 컨테이너 생성
    this.canvasContainer = document.createElement('div');
    this.canvasContainer.className = 'aigen-canvas-container';
    this.container.appendChild(this.canvasContainer);

    // 메인 캔버스 (합성용)
    this.mainCanvas = document.createElement('canvas');
    this.mainCanvas.width = this.options.width;
    this.mainCanvas.height = this.options.height;
    this.mainCanvas.className = 'aigen-main-canvas';
    this.canvasContainer.appendChild(this.mainCanvas);

    this.ctx = this.mainCanvas.getContext('2d');

    // 기본 레이어 추가
    this.addLayer('Background', { fillColor: 'white' });

    // 이벤트 설정
    this.setupEvents();

    // NiceGUI 통신 설정
    this.setupNiceGUIBridge();
  }
```

```
  // 레이어 관리
  addLayer(name, options = {}) {
    const layer = {
      id: `layer_${Date.now()}`,
      name: name,
      canvas: document.createElement('canvas'),
      visible: true,
      opacity: 1.0,
      blendMode: 'source-over',
      locked: false,
      ...options
    };

    layer.canvas.width = this.options.width;
    layer.canvas.height = this.options.height;
    const ctx = layer.canvas.getContext('2d');

    // 배경색 채우기
    if (options.fillColor) {
      ctx.fillStyle = options.fillColor;
      ctx.fillRect(0, 0, layer.canvas.width, layer.canvas.height);
    }

    this.layers.push(layer);
    this.activeLayer = layer;
    this.redraw();

    // NiceGUI로 이벤트 전송
    this.sendToNiceGUI('layer:added', {
      id: layer.id,
      name: layer.name
    });

    return layer;
  }

  // 레이어 병합
  mergeLayers(layerIds = null, includeHidden = false) {
    const canvas = document.createElement('canvas');
    canvas.width = this.options.width;
    canvas.height = this.options.height;
    const ctx = canvas.getContext('2d');

    const layersToMerge = layerIds
      ? this.layers.filter(l => layerIds.includes(l.id))
      : this.layers;

    layersToMerge.forEach(layer => {
      if (!includeHidden && !layer.visible) return;
```

```
      ctx.globalAlpha = layer.opacity;
      ctx.globalCompositeOperation = layer.blendMode;
      ctx.drawImage(layer.canvas, 0, 0);
    });

    return canvas;
  }

  // 출력 메서드들
  exportLayer(layerId) {
    const layer = this.layers.find(l => l.id === layerId);
    if (!layer) return null;

    return {
      id: layer.id,
      name: layer.name,
      dataURL: layer.canvas.toDataURL('image/png'),
      visible: layer.visible,
      opacity: layer.opacity
    };
  }

  exportAllLayers() {
    return this.layers.map(layer => this.exportLayer(layer.id));
  }

  exportMerged(options = {}) {
    const merged = this.mergeLayers(
      options.layerIds,
      options.includeHidden
    );
    return merged.toDataURL('image/png');
  }

  // img2img, preprocessor, mask 용 출력
  exportForProcessing(type) {
    const output = {
      type: type,
      timestamp: Date.now()
    };

    switch(type) {
      case 'img2img':
        output.image = this.exportMerged();
        output.metadata = {
          width: this.options.width,
          height: this.options.height
        };
```

```
            break;

        case 'mask':
            // 특정 레이어를 마스크로 사용
            const maskLayer = this.layers.find(l => l.name.includes('Mask'));
            if (maskLayer) {
                output.mask = maskLayer.canvas.toDataURL('image/png');
            }
            break;

        case 'preprocessor':
            // 전처리용 데이터
            output.layers = this.exportAllLayers();
            output.merged = this.exportMerged();
            break;
    }

    return output;
}

// NiceGUI 통신
setupNiceGUIBridge() {
    // NiceGUI의 JavaScript 실행 컨텍스트와 통신
    window.aiGenCanvas = this;

    // 커스텀 이벤트 리스너
    window.addEventListener('nicegui-canvas-command', (e) => {
        const { command, data } = e.detail;
        this.handleCommand(command, data);
    });
}

sendToNiceGUI(event, data) {
    // NiceGUI Python 백엔드로 데이터 전송
    if (window.nicegui) {
        window.nicegui.send({
            type: 'canvas-event',
            event: event,
            data: data
        });
    }
}

handleCommand(command, data) {
    switch(command) {
        case 'export':
            return this.exportForProcessing(data.type);
        case 'import':
            this.importImage(data.image, data.layerName);
            break;
```

```
            break;
        case 'clear':
            this.clearLayer(data.layerId);
            break;
      }
    }

    //도구 등록
    registerTool(name, toolClass) {
      this.tools[name] = new toolClass(this);
    }

    setActiveTool(name) {
      if (this.activeTool) {
        this.activeTool.deactivate();
      }

      this.activeTool = this.tools[name];
      if (this.activeTool) {
        this.activeTool.activate();
      }
    }
}

//기본 도구 클래스
class CanvasTool {
    constructor(canvas) {
      this.canvas = canvas;
      this.active = false;
    }

    activate() {
      this.active = true;
    }

    deactivate() {
      this.active = false;
    }
}

//브러시 도구
class BrushTool extends CanvasTool {
    constructor(canvas) {
      super(canvas);
      this.isDrawing = false;
      this.lastX = 0;
      this.lastY = 0;
      this.color = '#000000';
      this.size = 5;
    }
```

```
  activate() {
    super.activate();
    this.canvas.mainCanvas.addEventListener('mousedown', this.onMouseDown);
    this.canvas.mainCanvas.addEventListener('mousemove', this.onMouseMove);
    this.canvas.mainCanvas.addEventListener('mouseup', this.onMouseUp);
  }

  onMouseDown = (e) => {
    if (!this.active || !this.canvas.activeLayer) return;

    this.isDrawing = true;
    const rect = this.canvas.mainCanvas.getBoundingClientRect();
    this.lastX = e.clientX - rect.left;
    this.lastY = e.clientY - rect.top;

    // 히스토리 저장
    this.canvas.saveHistory();
  }

  onMouseMove = (e) => {
    if (!this.isDrawing || !this.active) return;

    const rect = this.canvas.mainCanvas.getBoundingClientRect();
    const currentX = e.clientX - rect.left;
    const currentY = e.clientY - rect.top;

    const ctx = this.canvas.activeLayer.canvas.getContext('2d');

    ctx.strokeStyle = this.color;
    ctx.lineWidth = this.size;
    ctx.lineCap = 'round';
    ctx.lineJoin = 'round';

    ctx.beginPath();
    ctx.moveTo(this.lastX, this.lastY);
    ctx.lineTo(currentX, currentY);
    ctx.stroke();

    this.lastX = currentX;
    this.lastY = currentY;

    this.canvas.redraw();
  }

  onMouseUp = (e) => {
    this.isDrawing = false;
  }
}
```

components/canvas_ui.py

python

```python
"""Canvas UI 컴포넌트 - NiceGUI 통합"""

from nicegui import ui
from pathlib import Path
import json
import asyncio

class CanvasEditor:
    """NiceGUI Canvas Editor 컴포넌트"""

    def __init__(self, width=512, height=512):
        self.width = width
        self.height = height
        self.canvas_id = f"canvas_{id(self)}"
        self.callbacks = {}

    def render(self):
        """캔버스 UI 렌더링"""
        with ui.card().classes('w-full'):
            # 툴바
            with ui.row().classes('gap-2 p-2'):
                # 도구 선택
                with ui.button_group():
                    ui.button(icon='brush', on_click=lambda: self.set_tool('brush'))
                    ui.button(icon='crop_square', on_click=lambda: self.set_tool('rect'))
                    ui.button(icon='circle', on_click=lambda: self.set_tool('circle'))
                    ui.button(icon='auto_fix_high', on_click=lambda: self.set_tool('eraser'))

                ui.separator().props('vertical')

                # 색상 선택
                self.color_picker = ui.color_picker(
                    value='#000000',
                    on_change=lambda e: self.set_color(e.value)
                )

                # 브러시 크기
                ui.slider(
                    min=1, max=50, value=5,
                    on_change=lambda e: self.set_brush_size(e.value)
                ).props('label').classes('w-32')

                ui.separator().props('vertical')

                # 레이어 컨트롤
                ui.button('레이어 추가', icon='add', on_click=self.add_layer)
                ui.button('레이어 병합', icon='merge_type', on_click=self.merge_layers)
```

```
        # 캔버스 영역
        with ui.row().classes('gap-4'):
            # 레이어 패널
            with ui.card().classes('w-48'):
                ui.label('레이어').classes('text-h6')
                self.layer_list = ui.column().classes('gap-1')

            # 캔버스
            ui.html(f'''
                <div id="{self.canvas_id}" style="border: 1px solid #ccc;">
                </div>
            ''').classes('flex-1')

            # 속성 패널
            with ui.card().classes('w-48'):
                ui.label('속성').classes('text-h6')

                ui.label('불투명도')
                ui.slider(min=0, max=100, value=100).props('label')

                ui.label('블렌드 모드')
                ui.select(
                    options=['normal', 'multiply', 'screen', 'overlay'],
                    value='normal'
                )

        # 하단 버튼
        with ui.row().classes('gap-2 mt-4'):
            ui.button(
                'img2img로 보내기',
                on_click=lambda: self.export_for('img2img')
            ).props('color=primary')

            ui.button(
                '마스크로 사용',
                on_click=lambda: self.export_for('mask')
            )

            ui.button(
                '전처리기로 보내기',
                on_click=lambda: self.export_for('preprocessor')
            )

        # 캔버스 초기화
        self._initialize_canvas()

    def _initialize_canvas(self):
        """JavaScript 캔버스 초기화"""
        # 정적 파일 로드
```

```
    extension_path = Path(__file__).parent.parent

    ui.add_head_html(f'''
        <link rel="stylesheet" href="/extensions/canvas-editor/static/css/canvas.css">
        <script src="/extensions/canvas-editor/static/js/canvas-core.js"></script>
        <script src="/extensions/canvas-editor/static/js/tools/brush.js"></script>
        <script src="/extensions/canvas-editor/static/js/tools/shapes.js"></script>
        <script src="/extensions/canvas-editor/static/js/layers/layer-manager.js"></script>
    ''')

    # 캔버스 인스턴스 생성
    ui.run_javascript(f'''
        const canvas = new AIGenCanvas("{self.canvas_id}", {{
            width: {self.width},
            height: {self.height}
        }});

        // 도구 등록
        canvas.registerTool('brush', BrushTool);
        canvas.registerTool('rect', RectangleTool);
        canvas.registerTool('circle', CircleTool);
        canvas.registerTool('eraser', EraserTool);

        // 기본 도구 설정
        canvas.setActiveTool('brush');

        // Python 콜백 연결
        window.nicegui = {{
            send: function(data) {{
                pywebview.api.canvas_event(JSON.stringify(data));
            }}
        }};

        window.canvasInstance = canvas;
    ''')

async def export_for(self, target: str):
    """특정 용도로 내보내기"""
    result = await ui.run_javascript(f'''
        window.canvasInstance.exportForProcessing("{target}")
    ''')

    # 이벤트 발생
    if target == 'img2img':
        ui.notify('img2img로 전송되었습니다', type='positive')
        # 실제로는 이벤트 버스로 전송
        await self.send_to_pipeline('img2img', result)
    elif target == 'mask':
        ui.notify('마스크로 설정되었습니다', type='positive')
```

```
            await self.send_to_pipeline('mask', result)
        elif target == 'preprocessor':
            ui.notify('전처리기로 전송되었습니다', type='positive')
            await self.send_to_pipeline('preprocessor', result)

    async def send_to_pipeline(self, pipeline_type: str, data):
        """파이프라인으로 데이터 전송"""
        from aigen_studio.state.store import app_state

        # 이벤트 버스로 전송
        app_state.emit(f'canvas:export_{pipeline_type}', {
            'type': pipeline_type,
            'data': data
        })
```

4. 성능 최적화 설정

src/aigen_studio/core/optimization.py

python

```python
"""성능 최적화 관리자"""

import torch
import argparse
from typing import Optional

class OptimizationManager:
    """통합 최적화 관리"""

    def __init__(self, args: argparse.Namespace):
        self.args = args
        self.use_xformers = False
        self.use_torch_sdp = True  # Scaled Dot Product Attention

    def setup_optimizations(self):
        """최적화 설정 적용"""
        # PyTorch 2.0 최적화
        if torch.__version__ >= "2.0":
            torch.backends.cuda.matmul.allow_tf32 = True
            torch.backends.cudnn.allow_tf32 = True
            torch.backends.cudnn.benchmark = True

        # xFormers 확인
        if self.args.xformers:
            try:
                import xformers
                import xformers.ops
                self.use_xformers = True
                self.use_torch_sdp = False
                print("✓ xFormers 활성화")
            except ImportError:
                print("⚠ xFormers 미설치, PyTorch SDP 사용")

        # 메모리 최적화
        if self.args.low_vram:
            torch.cuda.empty_cache()
            torch.cuda.set_per_process_memory_fraction(0.8)

    def optimize_model(self, model):
        """모델에 최적화 적용"""
        if self.use_xformers:
            self._apply_xformers(model)
        elif self.use_torch_sdp:
            self._apply_torch_sdp(model)

        if self.args.cpu_offload:
            self._apply_cpu_offload(model)
```

```
        return model

    def _apply_xformers(self, model):
        """xFormers 어텐션 적용"""
        if hasattr(model, 'enable_xformers_memory_efficient_attention'):
            model.enable_xformers_memory_efficient_attention()

    def _apply_torch_sdp(self, model):
        """PyTorch 2.0 SDP 어텐션 적용"""
        # PyTorch 2.0의 기본 최적화 사용
        if hasattr(torch.nn.functional, 'scaled_dot_product_attention'):
            # 이미 최적화됨
            pass

    def _apply_cpu_offload(self, model):
        """CPU 오프로딩 적용"""
        if hasattr(model, 'enable_sequential_cpu_offload'):
            model.enable_sequential_cpu_offload()
```

5. 실행 스크립트 개선

run.py

python

```python
#!/usr/bin/env python3
"""
AIGenStudio 실행 스크립트
기존 main.py를 확장하여 CLI 인자 지원
"""

import sys
import argparse
import asyncio
from pathlib import Path

# 프로젝트 경로 추가
sys.path.insert(0, str(Path(__file__).parent / "src"))

from aigen_studio.core.config import Settings
from aigen_studio.core.optimization import OptimizationManager
from aigen_studio.core.extension_system import ExtensionManager

async def main():
    # CLI 인자 파싱
    parser = argparse.ArgumentParser(description='AIGenStudio')

    # 성능 옵션
    parser.add_argument('--xformers', action='store_true',
                        help='Enable xformers optimization')
    parser.add_argument('--cpu-offload', action='store_true',
                        help='Enable CPU offloading for low VRAM')
    parser.add_argument('--low-vram', action='store_true',
                        help='Low VRAM mode')
    parser.add_argument('--precision', choices=['fp32', 'fp16', 'bf16'],
                        default='fp16', help='Model precision')

    # 서버 옵션
    parser.add_argument('--host', default='127.0.0.1')
    parser.add_argument('--port', type=int, default=8080)
    parser.add_argument('--share', action='store_true')

    # 확장 옵션
    parser.add_argument('--disable-extensions', nargs='+',
                        help='Disable specific extensions')
    parser.add_argument('--extension-dir', type=Path,
                        help='Additional extension directory')

    args = parser.parse_args()

    # 설정 로드
    settings = Settings()
```

```
    # 최적화 적용
    optimizer = OptimizationManager(args)
    optimizer.setup_optimizations()

    # 기존 main.py의 AIGenStudioApp 임포트
    from aigen_studio.main import AIGenStudioApp

    # 앱 인스턴스 생성 (args 전달)
    app = AIGenStudioApp()
    app.args = args  # CLI 인자 전달
    app.optimizer = optimizer

    # 확장 시스템 초기화
    app.extension_manager = ExtensionManager(app)
    await app.extension_manager.discover_and_load_extensions()

    # 앱 실행
    await app.startup()

    # NiceGUI 실행
    from nicegui import ui
    ui.run(
        title="AIGenStudio",
        host=args.host,
        port=args.port,
        reload=False,
        show=not args.share,
        dark=settings.dark_mode,
        storage_secret=settings.storage_secret
    )

if __name__ == "__main__":
    asyncio.run(main())
```