Python + Manim + Claude Architecture

Complete Design for DevForge Competition

PART 1: FINAL ARCHITECTURE DECISION

Decision: SINGLE SCENE + UNIFIED TIMELINE

Factor	Single Scene	Why It Wins
Manim Paradigm	Native to Manim	Manim designed for single scenes with sequential animations
Timeline Control	Unified = deterministic	Same JSON → same video every time
Rendering Speed	One pass	No scene switching overhead
Code Generation	Simple mapping	JSON directly → Python with self.play() calls
DevForge Requirements	✓ All met	Text → instructions → render → video
Educational Flow	Sequential	Math/physics/algorithms are inherently sequential

Locked in: Single Manim Scene class with unified timeline-based keyframe animation

---

PART 2: COMPLETE DATA FLOW

End-to-End Pipeline

```
┌─────────────────────────────────────────────────────────────┐
│ STAGE 1: USER INPUT (React Frontend)                        │
├─────────────────────────────────────────────────────────────┤
│ User enters natural language:                               │
│ "Animate the Pythagorean theorem with a growing square"     │
│                                                             │
│ POST /api/generate                                          │
│ { "description": "..." }                                    │
└────────────────┬────────────────────────────────────────────┘
                 │
                 ▼
┌─────────────────────────────────────────────────────────────┐
│ STAGE 2: LLM PARSING (Claude API via Python)                │
├─────────────────────────────────────────────────────────────┤
│ Python backend calls:                                       │
│ client.messages.create(                                     │
│   model="claude-3-5-sonnet",                                │
│   system_prompt=ANIMATION_SPEC_SCHEMA,                      │
│   user_message=description                                  │
│ )                                            │
│                                              │
│ Claude analyzes concept and returns structured JSON          │
└──────────────────────────┬──────────────────────────────────────────┘
                           │
                           ▼
┌─────────────────────────────────────────────────────────────────────┐
│ STAGE 3: JSON SPECIFICATION (Structured Output from Claude) │
├─────────────────────────────────────────────────────────────────────┤
│ Claude returns JSON with:                              │
│ - conceptType (e.g., "pythagorean_theorem")                  │
│ - totalDuration (seconds)                              │
│ - objects (what to draw)                               │
│ - timeline (when to animate)                           │
│ - scenes (logical grouping for UI)                     │
│                                              │
│ [See PART 3 for full schema]                           │
└──────────────────────────┬──────────────────────────────────────────┘
                           │
                           ▼
┌─────────────────────────────────────────────────────────────────────┐
│ STAGE 4: VALIDATION (Python Backend)                     │
├─────────────────────────────────────────────────────────────────────┤
│ Check:                                           │
│ ✓ All required fields present                          │
│ ✓ All object types supported                           │
│ ✓ All animation actions valid                          │
│ ✓ Timeline is chronologically ordered                        │
│ ✓ Object references exist                              │
│ ✓ Duration is reasonable (< 10 min)                          │
│                                              │
│ If FAIL: Return error to frontend                      │
│ If PASS: Store spec in database, proceed                     │
└──────────────────────────┬──────────────────────────────────────────┘
                           │
                           ▼
┌─────────────────────────────────────────────────────────────────────┐
│ STAGE 5: FRONTEND DISPLAY                              │
├─────────────────────────────────────────────────────────────────────┤
│ Send back to React:                                    │
│                                              │
│ ┌─ SCENE BREAKDOWN ────────────────────────────┐            │
│ │ Scene 1: Setup (0.0s - 1.5s)        │            │
│ │ ├─ Create right triangle            │            │
│ │ ├─ Label sides a, b, c              │            │
│ │                                     │            │
│ │ Scene 2: Demonstrate (1.5s - 5.0s)  │            │
│ │ ├─ Draw squares on each side         │              │
│ │ ├─ Show equation a² + b² = c²        │              │
│ │ ├─ Highlight growing squares         │              │
│ │                             │              │
│ │ Scene 3: Conclusion (5.0s - 6.0s)    │              │
│ │ └─ Display final verification        │              │
│ └──────────────────────────────────────────────────┘              │
│                                 │
│ ┌─ TIMELINE VISUALIZATION ──────────────────┐              │
│ │ 0s ──── 2s ──── 4s ──── 6s         │              │
│ │ ↓    ↓    ↓    ↓              │              │
│ │ [Scene 1][Scene 2         ][Scene 3] │              │
│ │ Create  Highlight          Verify │              │
│ └──────────────────────────────────────────────────┘              │
│                                 │
│ ┌─ ANIMATION KEYFRAMES (EXPANDABLE) ────┐              │
│ │ [+] Keyframe 1 @ 0.0s: create_triangle │              │
│ │ [+] Keyframe 2 @ 0.5s: label_sides    │              │
│ │ [+] Keyframe 3 @ 1.5s: draw_squares   │              │
│ │ ... (more keyframes)               │              │
│ └──────────────────────────────────────────────────┘              │
│                                 │
│ [Generate Video]  [Cancel]                   │
└────────────────────┬──────────────────────────────────────────────┘
        │
      User clicks
     "Generate Video"
        │
        ▼
┌───────────────────────────────────────────────────────────────────┐
│ STAGE 6: PYTHON CODE GENERATION                 │
├───────────────────────────────────────────────────────────────────┤
│ Backend transforms JSON → Python Manim code              │
│                                 │
│ Template: class GeneratedScene(Scene):              │
│          def construct(self):                   │
│            # Create objects                  │
│            # Execute timeline                  │
│            # Add wait times                  │
│                                 │
│ Pseudo-structure:                       │
│ ─────────────────────────────────────────────────────              │
│ from manim import *                       │
│                                 │
│ class GeneratedScene(Scene):                   │
│    def construct(self):                     │
│       # Objects from JSON                    │
│        triangle = Polygon([0,0], [3,0], [0,4])              │
│        label_a = Text("a", ...)                │
│        ...                                     │
│                                                │
│        # Timeline: loop through keyframes                     │
│        self.add(triangle)                      │
│        self.play(FadeIn(triangle), run_time=0.8)             │
│        self.play(Write(label_a), run_time=0.5)               │
│        self.wait(0.2)                          │
│        ...                                     │
│                                                │
│ File saved: /tmp/generated_scene_12345.py                   │
└──────────────────────────────┬──────────────────────────────┘
             │
             ▼
┌─────────────────────────────────────────────────────────────┐
│ STAGE 7: MANIM RENDERING                                    │
├─────────────────────────────────────────────────────────────┤
│ Execute: manim -pql generated_scene_12345.py GeneratedScene │
│                                                │
│ Manim:                                         │
│ 1. Initializes 1920x1080 canvas                     │
│ 2. Creates 3D camera                               │
│ 3. Adds lights and renders                          │
│ 4. Executes construct() method                      │
│ 5. Records EACH frame to disk (@ 60 FPS)                  │
│ 6. Encodes frames to MP4 using FFmpeg                    │
│ 7. Output: generated_scene_12345.mp4 (6 sec @ 60fps)        │
│                                                │
│ Rendering time: ~2-5 minutes for 6 second animation         │
│ Output size: ~15-30 MB                              │
│ Quality: 1920x1080 H.264                            │
└──────────────────────────────┬──────────────────────────────┘
             │
             ▼
┌─────────────────────────────────────────────────────────────┐
│ STAGE 8: VIDEO STORAGE & METADATA                           │
├─────────────────────────────────────────────────────────────┤
│ Store video:                                   │
│ /videos/spec_12345/output.mp4                       │
│                                                │
│ Database updates:                              │
│ video_outputs table:                           │
│ {                                              │
│   spec_id: "spec_12345",                       │
│   video_path: "/videos/spec_12345/output.mp4",          │
│   duration: 6.0,                               │
│   resolution: "1920x1080",                      │
│   file_size_mb: 25,                             │
│   status: "completed",                          │
│   created_at: timestamp,                        │
│   manim_code: "[full Python code as text]"      │
│ }                                               │
│                                                 │
│ animation_specs table:                          │
│ {                                               │
│   spec_id: "spec_12345",                        │
│   input_description: "Animate the Pythagorean...",  │
│   generated_json: "[full JSON spec]",           │
│   status: "completed"                           │
│ }                                               │
└──────────────────────┬──────────────────────────┘
             │
             ▼
┌─────────────────────────────────────────────────┐
│ STAGE 9: FRONTEND DELIVERY & DISPLAY            │
├─────────────────────────────────────────────────┤
│ Frontend polls: GET /api/status/spec_12345      │
│ Returns: { status: "completed", video_url: "..." }  │
│                                                 │
│ Frontend displays:                              │
│                                                 │
│ ┌─────────────────────────────────────────┐     │
│ │ FINAL ANIMATION VIDEO PLAYER            │     │
│ │ ┌─────────────────────────────────────┐ │     │
│ │ │ [■] ───────●─────────── 0:06        │ │     │
│ │ │ Play | Pause | Volume | Fullscreen  │ │     │
│ │ │ Download MP4 | Share | Delete       │ │     │
│ │ └─────────────────────────────────────┘ │     │
│ │                                         │     │
│ │ [View Generated Code] [Regenerate]      │     │
│ └─────────────────────────────────────────┘     │
│                                                 │
│ Expandable sections:                            │
│ [+] Generated Python Code (for reference)       │
│ [+] Scene Breakdown (what we showed earlier)    │
│ [+] Timeline Details (animation keyframes)      │
│ [+] Original JSON Spec (full spec)              │
└─────────────────────────────────────────────────┘
```

PART 3: JSON SPECIFICATION SCHEMA

Complete Schema: What Claude Generates

```
json
```

```
{
  "conceptType": "string",
  "title": "string",
  "description": "string",
  "domain": "math | physics | cs | biology | chemistry | other",
  "difficulty": "beginner | intermediate | advanced",
  "totalDuration": "float (seconds)",
  "fps": "int (typically 60)",

  "scenes": [
    {
      "id": "string (unique identifier)",
      "name": "string",
      "startTime": "float (seconds)",
      "endTime": "float (seconds)",
      "description": "string (what happens in this scene)"
    }
  ],

  "objects": [
    {
      "id": "string (unique)",
      "type": "string (MathTex | Text | Circle | Rectangle | Line | Arrow | Grid | Polygon | etc)",
      "content": "string (LaTeX for MathTex, text for Text, etc)",
      "initialState": {
        "position": "[x, y, z] or [0, 0, 0] default",
        "opacity": "float (0-1, default 0)",
        "scale": "float (default 1)",
        "color": "string (color name or hex)",
        "rotation": "float (degrees, default 0)"
      },
      "properties": {
        "fontSize": "int (if Text or MathTex)",
        "strokeWidth": "float (for shapes)",
        "fillColor": "string (for shapes)",
        "strokeColor": "string",
        "radius": "float (for Circle)",
        "width": "float (for Rectangle)",
        "height": "float (for Rectangle)",
        "vertices": "[[x1,y1], [x2,y2], ...] (for Polygon)"
      }
    }
  ],

  "timeline": [
    {
      "id": "string (unique keyframe id)",
      "time": "float (seconds, when animation starts)",
      "duration": "float (seconds, how long animation runs)",
      "action": "string (FadeIn | FadeOut | Write | Transform | MoveTo | Highlight | Rotate | Scale | etc)",
      "target": "string or [string] (object id or ids to animate)",
      "parameters": {
        "color": "string (if color change)",
        "newContent": "string (if morphing text/equation)",
        "direction": "float (angle in degrees, for movement)",
        "newPosition": "[x, y, z] (for MoveTo)",
        "newScale": "float (for Scale)",
        "angle": "float (degrees, for Rotate)",
        "highlightColor": "string (for Highlight)"
      },
      "easing": "string (linear | ease_in | ease_out | ease_in_out)",
      "sceneId": "string (optional, which scene this belongs to)"
    }
  ],

  "annotations": [
    {
      "id": "string",
      "text": "string (explanatory text for user)",
      "time": "float (when to show)",
      "position": "[x, y, z]"
    }
  ]
}
```

PART 4: SUPPORTED OBJECT TYPES & ANIMATIONS

Object Types (What Can Be Drawn)

```
Geometric Shapes:
├── Circle (radius, color, fill)
├── Rectangle (width, height, color)
├── Polygon (vertices list)
├── Line (start point, end point)
├── Arrow (start, end, color, width)
├── Grid (rows, cols, spacing)
└── Arc (radius, angle, color)

Text & Math:
├── Text (content, font_size, color)
├── MathTex (LaTeX equation, font_size, color)
```

```
└── Label (text + position, for labeling objects)

2D Graphs:
├── Axes (x_range, y_range, x_axis_label, y_axis_label)
├── FunctionGraph (function string, color, stroke_width)
├── ScatterPlot (points array)
└── BarChart (data array, colors)

3D Objects:
├── Sphere (radius, color)
├── Cube (side_length, color)
├── Cylinder (radius, height, color)
└── ThreeDAxes (x_range, y_range, z_range)

Composite:
├── Group (multiple objects together)
└── BreakableGroup (objects that can be separated)
```

Animation Actions (What Can Happen)

```
Appearance/Disappearance:
├── FadeIn (opacity 0 → 1)
├── FadeOut (opacity 1 → 0)
├── Write (text appears as if being written)
└── Unwrite (text disappears as if being erased)

Movement:
├── MoveTo (position A → position B)
├── Shift (move by offset)
└── Rotate (rotate around center)

Transformation:
├── Transform (morph shape A → shape B)
├── ReplacementTransform (replace object while transforming)
├── Scale (size change)
├── Stretch (stretch in one direction)
└── Highlight (change color/glow temporarily)

Composition:
├── Add (add object to scene)
├── Remove (remove object from scene)
└── Wait (pause, no animation)

Math-Specific:
├── PointAlong (move point along curve)
├── UpdateFromAlphaData (transform with alpha interpolation)
└── Indicate (flash/indicate an object)
```

PART 5: JSON EXAMPLE - PYTHAGOREAN THEOREM

Real Example: What Claude Would Generate

```json
```

```json
{
  "conceptType": "pythagorean_theorem",
  "title": "Pythagorean Theorem Visualization",
  "description": "Demonstrate the Pythagorean theorem with squares on each side of a right triangle",
  "domain": "math",
  "difficulty": "beginner",
  "totalDuration": 6.0,
  "fps": 60,

  "scenes": [
    {
      "id": "scene_1",
      "name": "Setup",
      "startTime": 0.0,
      "endTime": 1.5,
      "description": "Create right triangle and label sides"
    },
    {
      "id": "scene_2",
      "name": "Demonstration",
      "startTime": 1.5,
      "endTime": 5.0,
      "description": "Draw squares and show relationship"
    },
    {
      "id": "scene_3",
      "name": "Conclusion",
      "startTime": 5.0,
      "endTime": 6.0,
      "description": "Display equation and verification"
    }
  ],

  "objects": [
    {
      "id": "triangle",
      "type": "Polygon",
      "content": null,
      "initialState": {
        "position": [0, 0, 0],
        "opacity": 0,
        "scale": 1,
        "color": "BLUE",
        "rotation": 0
      },
      "properties": {
        "vertices": [[0, 0], [3, 0], [0, 4]],
        "strokeWidth": 3,
        "fillColor": "LIGHT_BLUE",
        "strokeColor": "BLUE"
      }
    },
    {
      "id": "side_a_label",
      "type": "Text",
      "content": "a",
      "initialState": {
        "position": [1.5, -0.5, 0],
        "opacity": 0,
        "scale": 1,
        "color": "BLACK"
      },
      "properties": {
        "fontSize": 32
      }
    },
    {
      "id": "side_b_label",
      "type": "Text",
      "content": "b",
      "initialState": {
        "position": [-0.5, 2, 0],
        "opacity": 0,
        "scale": 1,
        "color": "BLACK"
      },
      "properties": {
        "fontSize": 32
      }
    },
    {
      "id": "side_c_label",
      "type": "Text",
      "content": "c",
      "initialState": {
        "position": [1.8, 2.2, 0],
        "opacity": 0,
        "scale": 1,
        "color": "BLACK"
      },
      "properties": {
        "fontSize": 32
      }
    },
    {
      "id": "square_a",
      "type": "Rectangle",
      "content": null,
      "initialState": {
        "position": [1.5, -2.5, 0],
        "opacity": 0,
        "scale": 1,
        "color": "RED"
      },
      "properties": {
        "width": 3,
        "height": 3,
        "strokeWidth": 2,
        "fillColor": "RED",
        "strokeColor": "DARK_RED"
      }
    },
    {
      "id": "square_b",
      "type": "Rectangle",
      "content": null,
      "initialState": {
        "position": [-2.5, 2, 0],
        "opacity": 0,
        "scale": 1,
        "color": "GREEN"
      },
      "properties": {
        "width": 4,
        "height": 4,
        "strokeWidth": 2,
        "fillColor": "GREEN",
        "strokeColor": "DARK_GREEN"
      }
    },
    {
      "id": "square_c",
      "type": "Rectangle",
      "content": null,
      "initialState": {
        "position": [1.8, 2.2, 0],
        "opacity": 0,
        "scale": 1,
        "color": "PURPLE"
      },
      "properties": {
        "width": 5,
        "height": 5,
        "strokeWidth": 2,
        "fillColor": "PURPLE",
        "strokeColor": "DARK_PURPLE"
      }
    },
    {
      "id": "equation",
      "type": "MathTex",
      "content": "a^2 + b^2 = c^2",
      "initialState": {
        "position": [0, -3, 0],
        "opacity": 0,
        "scale": 1,
        "color": "BLACK"
      },
      "properties": {
        "fontSize": 48
      }
    }
  ],

  "timeline": [
    {
      "id": "kf_1",
      "time": 0.0,
      "duration": 0.8,
      "action": "FadeIn",
      "target": "triangle",
      "parameters": {},
      "easing": "ease_in_out",
      "sceneId": "scene_1"
    },
    {
      "id": "kf_2",
      "time": 0.8,
      "duration": 0.3,
      "action": "Write",
      "target": "side_a_label",
      "parameters": {},
      "easing": "linear",
      "sceneId": "scene_1"
    },
    {
      "id": "kf_3",
      "time": 1.1,
      "duration": 0.2,
      "action": "Write",
      "target": "side_b_label",
      "parameters": {},
      "easing": "linear",
      "sceneId": "scene_1"
    },
    {
      "id": "kf_4",
      "time": 1.3,
      "duration": 0.2,
      "action": "Write",
      "target": "side_c_label",
      "parameters": {},
      "easing": "linear",
      "sceneId": "scene_1"
    },
    {
      "id": "kf_5",
      "time": 1.5,
      "duration": 1.2,
      "action": "FadeIn",
      "target": ["square_a", "square_b", "square_c"],
      "parameters": {},
      "easing": "ease_in_out",
      "sceneId": "scene_2"
    },
    {
      "id": "kf_6",
      "time": 2.7,
      "duration": 0.5,
      "action": "Highlight",
      "target": "square_a",
      "parameters": {
        "highlightColor": "ORANGE"
      },
      "easing": "ease_in_out",
      "sceneId": "scene_2"
    },
    {
      "id": "kf_7",
      "time": 3.2,
      "duration": 0.5,
      "action": "Highlight",
      "target": "square_b",
      "parameters": {
```

```
      "highlightColor": "ORANGE"
    },
    "easing": "ease_in_out",
    "sceneId": "scene_2"
  },
  {
    "id": "kf_8",
    "time": 3.7,
    "duration": 0.5,
    "action": "Highlight",
    "target": "square_c",
    "parameters": {
      "highlightColor": "ORANGE"
    },
    "easing": "ease_in_out",
    "sceneId": "scene_2"
  },
  {
    "id": "kf_9",
    "time": 4.2,
    "duration": 0.5,
    "action": "Wait",
    "target": null,
    "parameters": {},
    "easing": "linear",
    "sceneId": "scene_2"
  },
  {
    "id": "kf_10",
    "time": 5.0,
    "duration": 0.8,
    "action": "Write",
    "target": "equation",
    "parameters": {},
    "easing": "linear",
    "sceneId": "scene_3"
  },
  {
    "id": "kf_11",
    "time": 5.8,
    "duration": 0.2,
    "action": "Wait",
    "target": null,
    "parameters": {},
    "easing": "linear",
    "sceneId": "scene_3"
  }
  ],

  "annotations": [
    {
      "id": "anno_1",
      "text": "Right triangle with sides a, b, and hypotenuse c",
      "time": 0.8,
      "position": [0, 3.5, 0]
    },
    {
      "id": "anno_2",
      "text": "The area of squares on shorter sides...",
      "time": 2.7,
      "position": [0, 3.5, 0]
    },
    {
      "id": "anno_3",
      "text": "...equals the area of square on hypotenuse",
      "time": 3.7,
      "position": [0, 3.5, 0]
    }
  ]
}
```

PART 6: JSON → PYTHON CODE GENERATION

Transformation Process

```
INPUT: JSON spec (from Claude)
   ↓
TRANSFORMATION RULES:
├── For each object:
│   └── Generate Manim object creation
├── For each timeline entry:
│   └── Generate self.play() call
└── Construct full Scene class
   ↓
OUTPUT: Python file with Scene class
```

Python Code Template

python

```
# Generated code structure (pseudo)

from manim import *

class GeneratedScene(Scene):
    def construct(self):
        # Camera and rendering settings
        self.camera.background_color = WHITE

        # STEP 1: Create all objects
        triangle = Polygon([0,0], [3,0], [0,4])
        triangle.set_color(BLUE).set_fill(LIGHT_BLUE)
        triangle.set_opacity(0)

        side_a_label = Text("a", font_size=32)
        side_a_label.move_to([1.5, -0.5, 0])
        side_a_label.set_opacity(0)

        # ... more object creation ...

        # STEP 2: Add objects to scene
        self.add(triangle, side_a_label, ...)

        # STEP 3: Execute timeline animations

        # Keyframe 1 @ 0.0s: FadeIn triangle
        self.play(FadeIn(triangle), run_time=0.8)

        # Keyframe 2 @ 0.8s: Write label
        self.play(Write(side_a_label), run_time=0.3)

        # ... more animations ...

        # Keyframe 11 @ 5.8s: Wait
        self.wait(0.2)
```

Key insight: Each timeline entry maps to a `self.play()` call with specific run_time and animation type.

PART 7: DATA STRUCTURES IN PYTHON BACKEND

How Python Processes This

```
python
```

```
# 1. Store JSON in database
spec_dict = json.loads(claude_response)
db.save_spec(spec_dict)

# 2. Validate
validator.validate_schema(spec_dict)
validator.check_all_actions_supported(spec_dict)

# 3. Generate Manim code
code_generator = ManimCodeGenerator(spec_dict)
python_code = code_generator.generate()

# 4. Write to file
with open(f'/tmp/scene_{spec_id}.py', 'w') as f:
    f.write(python_code)

# 5. Execute Manim
subprocess.run([
    'manim',
    '-pql',  # preview, low quality for speed
    f'/tmp/scene_{spec_id}.py',
    'GeneratedScene'
])

# 6. Move video to storage
shutil.move(
    'media/videos/.../GeneratedScene.mp4',
    f'/videos/spec_{spec_id}/output.mp4'
)

# 7. Update database with video path
db.update_status(spec_id, 'completed', video_path)
```

PART 8: FRONTEND DISPLAY LOGIC

What React Shows at Each Stage

```
Stage 1: Input
└─ Textarea for description
└─ [Generate] button

Stage 2: Processing (Spinner)
└─ "Parsing with Claude..."
└─ "Validating spec..."
```

```
└── "Generating Manim code..."
└
```