> name: presentation-image-generator description: Generate executive-ready presentation slides as AI-generated images using OpenAI's gpt-image-2 model. Use this skill whenever the user wants to create a presentation, slide deck, pitch deck, strategy slides, or any set of slides where each slide is rendered as a single polished image (rather than editable text/shapes). Trigger on phrases like "create a presentation about X", "build me a slide deck on Y", "I need slides for Z", "make a pitch deck", "design a strategy presentation", or any request that involves producing multiple slides as visual deliverables. This skill produces an HTML file that the user opens in their browser, pastes their OpenAI API key into, and uses to generate two design variants per slide with one-click download. ALWAYS use this skill for presentation creation requests — do NOT default to creating a .pptx file unless the user explicitly asks for an editable PowerPoint.

Presentation Image Generator

A skill for producing presentations where each slide is generated as a single AI-rendered image using OpenAI's `gpt-image-2` model. The deliverable is a self-contained HTML file the user opens locally; they paste their OpenAI API key, click "Start Generation", review two design variants per slide, and download the chosen image.

When to use this skill

Use whenever the user wants to create a presentation, slide deck, pitch deck, or strategy slides — unless they explicitly ask for an editable PowerPoint (.pptx) file. This skill produces image-based slides, not editable text/shape slides.

Typical trigger phrases: - "Create a presentation about [topic]" - "I need a slide deck on [topic]" - "Make me a pitch deck for [topic]" - "Design strategy slides for [topic]" - "Build slides explaining [topic]"

Workflow overview

The skill runs in three phases. Do not skip phases or merge them. Each phase ends with explicit user approval before moving on.

```
Phase 1: Clarify  →  Phase 2: Storyline (approve)  →  Phase 3: Generate HTML
```

Phase 1: Clarification

Before writing anything, ask the user the following questions in a single, concise message.

Required clarifications:

1. **Topic / content** — What is the presentation about? (If they already gave the topic, skip this.)
2. **Company / brand reference** — What company is this for? (Used to inform the visual style and tone of the slides. If none, ask whether to use a generic professional style or a specific reference company.)
3. **Number of slides** — How many slides? (Typical: 5–12. If they say "a few", default to 6.)
4. **Audience** — Who is the audience? (e.g., executive committee, board, sales team, investors, internal town hall.) This shapes tone and depth.
5. **Style preference** — One of: - **Executive strategy** (McKinsey/BCG/Bain-style: dense, framework-heavy, conclusion-driven action titles) - **Investor pitch** (cleaner, narrative-forward, more whitespace, key numbers prominent) - **Technical / architectural** (system diagrams, layered views, data flows) - **Transformation roadmap** (phased, journey-oriented, milestone-heavy) - **Other** — let the user describe it

Wait for the user's answers before proceeding. Do not draft the storyline until clarification is complete.

Phase 2: Storyline alignment

Once Phase 1 is complete, draft a storyline and present it to the user for approval. The storyline is the narrative spine of the deck.

For each slide, provide:

- **Slide number and action title** — A conclusion-oriented headline (not a topic label). Example: "Our cost-to-serve is 23% above peers — driven by three structural gaps" (good) vs. "Cost analysis" (bad).
- **2-sentence description** — What the slide argues and why it matters in the narrative.
- **3–4 key bullets** — The supporting points/data/callouts that should appear on the slide. These are also what will be shown on the left-hand side of the generated HTML for user reference next to the image variants.
- **Suggested visual framework** — One single framework from the allowed list: e.g., 2x2 matrix, phased roadmap, pyramid, value-chain map, swimlane, flywheel, layered architecture, etc. Each slide must use exactly one framework.

Present the storyline as a clean markdown list or table, then ask:

> "Does this storyline work, or would you like to adjust any slides (titles, framing, frameworks, bullets, or order)?"

Iterate until the user explicitly approves. Do not move to Phase 3 until the user says something like "looks good", "approved", "proceed", or equivalent.

Phase 3: Generate the HTML file

Once the storyline is approved, produce a single self-contained HTML file that the user will open in their browser.

What the HTML must do

1. Show an **API key input field** at the top where the user pastes their OpenAI API key. The key is held in memory only — never logged, never sent anywhere except `api.openai.com`.
2. Show a **"Start Generation" button** that triggers parallel generation of all slides at once (in batches of 5 if there are more than 5 slides — see the concurrency rules below).
3. For each slide, render a **row** with: - **Left side (~30% width):** slide number, action title, the 3–4 key bullets from the storyline, and the suggested framework. This is the reference content the user approved. - **Right side (~70% width):** two image variants side by side, each with a "Download" button below it. While generating, show a skeleton/loading state. Once both variants render, the user can click either download button to save the image as a PNG.
4. Use OpenAI's `gpt-image-2` model with `quality: "medium"` and `size: "2560x1440"` (exact 16:9, both edges divisible by 16, sits at the 2K threshold which is the recommended max for stable results).
5. Show an overall progress indicator (e.g., "Generated 6 of 8 slides...") and per-slide states. Surface API errors clearly if the key is invalid or a request fails.

Concurrency rules (important — read carefully)

The user wants generation to run in **parallel batches of 5 slides at a time**, not sequentially one-by-one. Each slide produces two variants, so a batch of 5 slides fires 10 API requests concurrently.

Behavior:

- **≤ 5 slides:** Fire all variant requests in parallel at once (up to 10 requests). Use `Promise.allSettled` so one failure doesn't block the others.
- **> 5 slides:** Process in waves of 5 slides each. Start wave 1 (slides 1–5, 10 requests in parallel). When all 10 either resolve or fail, start wave 2 (slides 6–10), and so on. This keeps the user from hitting rate limits while still being dramatically faster than one-at-a-time.
- Use `Promise.allSettled` within each wave so partial failures inside a wave don't block the rest of that wave or the next one.
- Update the UI **per individual promise** as it settles — do not wait for the whole wave before rendering individual variants. The wave boundary is only for pacing the next batch.
- After all waves complete, list any slides that failed and offer a "Retry failed slides" button that re-runs only those.

The exact prompt sent to OpenAI per image

For every image generation request, build a prompt by filling the bracketed placeholders below with the slide-specific content and the company name. **The Style input section is fixed — do not modify it.**

```
Content input: [Action title + 3–4 key bullets + 2-sentence description for this slide. Optionally end with: "Use a [framework] as the primary visual framework."]

Company reference: [Company name from Phase 1]

Style input: Create the slide as a single polished image in a strict 16:9 aspect ratio. Make it executive-ready, text-rich, and strategically structured, with dense but legible content and clear business language. The output should look like a professional strategy presentation slide, not a website, dashboard, HTML layout, app screen, or web-based design. Use the image model to generate the slide directly; do not use HTML, CSS, code, or browser-style UI elements. Build the slide around one dominant visual framework only, choosing the most appropriate structure for the message, such as a phased roadmap, capability maturity model, transformation journey map, operating-model blueprint, value-chain transformation map, swimlane roadmap, flywheel, pyramid, 2x2 matrix, org model, process flow, layered architecture, house-of-strategy framework, transformation portfolio map, or another single coherent infographic format. Do not combine multiple competing frameworks on the same slide, and do not overcrowd the slide with too many visual systems or text that becomes unreadably small. Add supporting text, callouts, labels, milestones, risks, enablers, KPIs, and decision points around the main framework where useful, but keep everything visually disciplined and anchored to the primary infographic. Leave generous whitespace for a strong action title at the top, written as a conclusion-oriented headline that communicates the key strategic message. Do not include a subtitle.
```

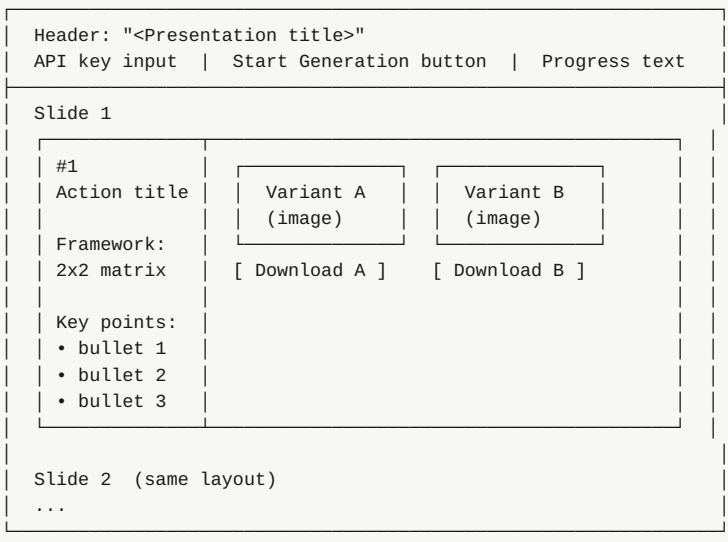
Building the HTML

1. Use the HTML scaffold and JS logic specified in the **HTML Template** section below.
2. Embed all slide data (titles, bullets, descriptions, the full prompt for each slide) as a JavaScript constant inside the HTML so the file is fully self-contained.
3. Save the HTML to `/mnt/user-data/outputs/presentation_<topic-slug>.html`.
4. Use `present_files` to share the file with the user. Briefly tell them: open the file, paste their OpenAI key, click Start Generation, then pick and download their preferred variants.

Keep the post-amble short — just the file link and the three-step instruction. Do not lecture about API keys or pricing unless the user asks.

HTML Template Specification

Structure



OpenAI API call

Endpoint: `https://api.openai.com/v1/images/generations`

Request body:

```
{
  "model": "gpt-image-2",
  "prompt": "<the full prompt for this slide>",
  "n": 1,
  "size": "2560x1440",
  "quality": "medium"
}
```

Headers:

```
Authorization: Bearer <user's API key>
Content-Type: application/json
```

The response returns base64-encoded PNG data in `response.data[0].b64_json`. Decode it to a data URL: `<b64_json>` and set it as the `src` of the `<img>` element.

Size rationale: `2560x1440` is exact 16:9 (1.7778:1), both edges divisible by 16 (160 × 90), and sits at the 2K threshold — the largest stable, non-experimental size for `gpt-image-2`. Do not substitute a different size without verifying it satisfies all `gpt-image-2` constraints: both edges divisible by 16, max edge ≤ 3840, aspect ratio ≤ 3:1, total pixels between 655,360 and 8,294,400.

Parallel batch generation logic

```
async function generateAllSlides(slides, apiKey) {
  const BATCH_SIZE = 5;
  for (let i = 0; i < slides.length; i += BATCH_SIZE) {
    const batch = slides.slice(i, i + BATCH_SIZE);
    // For each slide in the batch, fire BOTH variants in parallel.
    // Flatten to one big Promise.allSettled per batch so 10 calls
    // (5 slides x 2 variants) run concurrently.
    const tasks = batch.flatMap(slide => [
      generateVariant(slide, 'A', apiKey),
      generateVariant(slide, 'B', apiKey)
    ]);
    await Promise.allSettled(tasks);
    // Wave finished — UI has been updating per-promise as each settles.
  }
}

async function generateVariant(slide, variant, apiKey) {
  setSlideState(slide.number, variant, 'generating');
  try {
    const res = await fetch('https://api.openai.com/v1/images/generations', {
      method: 'POST',
      headers: {
        'Authorization': `Bearer ${apiKey}`,
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({
        model: 'gpt-image-2',
        prompt: slide.prompt,
        n: 1,
        size: '2560x1440',
        quality: 'medium'
      })
    });
    if (!res.ok) {
      const err = await res.json().catch(() => ({}));
      throw new Error(err.error?.message || `HTTP ${res.status}`);
    }
    const data = await res.json();
    const dataUrl = `${data.data[0].b64_json}`;
    setSlideImage(slide.number, variant, dataUrl);
  } catch (e) {
    setSlideError(slide.number, variant, e.message);
  }
}
```

Update the UI progressively: each variant should render as soon as its individual promise resolves, not wait for the whole batch. The `Promise.allSettled` boundary is just for batch pacing — not for UI updates.

Download logic

Each "Download" button below a variant should: 1. Use the existing base64 data URL as the `href` of a synthetic anchor. 2. Trigger a download with filename `slide_<N>_variant_<A|B>.png`.

Standard pattern:

```
function download(dataUrl, filename) {
  const a = document.createElement('a');
  a.href = dataUrl;
  a.download = filename;
  a.click();
}
```

Embedded slide data

Embed the storyline data as a JS constant near the top of the `<script>` block. Example shape:

```
const PRESENTATION_TITLE = "Q3 Strategy Review — Acme Corp";

const SLIDES = [
  {
    number: 1,
    title: "Our cost-to-serve runs 23% above peers — three structural gaps explain the gap",
    framework: "Waterfall + 3-bucket breakdown",
    bullets: [
      "Network footprint is 18% denser than peers (driving fixed cost)",
      "Service tier mix skews 2x toward premium SLA contracts",
      "Vendor consolidation lags peers by 4 years",
      "Combined impact: ~$240M annual cost premium"
    ],
    prompt: "Content input: ...\n\nCompany reference: Acme Corp\n\nStyle input: Create the slide as a single polished
image..."
  },
  // ... one entry per slide
];
```

The `prompt` field is the full prompt string (Content input + Company reference + Style input) for that slide. Building it at HTML-generation time keeps the JS simple — the browser just sends `slide.prompt` to OpenAI as-is.

Styling

Keep it clean, professional, restrained — the slides themselves are the visual product, not the page. Suggested palette:

- Background: `#f7f7f5` (warm off-white)
- Card background: `#ffffff`
- Borders: `#e5e5e0`
- Primary text: `#1a1a1a`
- Muted text: `#6b6b66`
- Accent (button, progress bar): `#2d5fdb` or any single accent color
- Font: system stack (`-apple-system, BlinkMacSystemFont, "Segoe UI", Roboto, sans-serif`)
- Generous padding, max-width container around 1400px for readability

Each slide row should have: - A small slide number badge in the top-left of the left column - Title in bold, ~20px - Framework label as small muted text - Bullets as a clean `<ul>` with comfortable line-height - Image placeholders that maintain a 16:9 aspect ratio while loading (skeleton shimmer is a nice touch but optional) - Download buttons styled as outline buttons below each image

States to handle

1. **Idle** — Before "Start Generation" is clicked. Show empty image placeholders with a "Waiting..." label.
2. **Generating** — Show a spinner or skeleton in each image area currently in-flight. Update overall progress text: "Generated N of M slides (X variants in flight)..."
3. **Variant complete** — That single image visible with its download button enabled. Each variant resolves independently as its API call returns.
4. **Variant error** — Show the error message in that variant's image area with a "Retry" button that re-fires just that one API call.

5. **All done** — Progress text changes to "All slides generated. Click any download button to save." If any slides have failed variants, show a "Retry failed slides" button at the top.

Error handling

- If no API key is entered, disable the Start button or show an inline message: "Paste your OpenAI API key to begin."
- If a 401 is returned, surface "Invalid API key" prominently and stop firing further requests in remaining batches.
- If a 429 is returned, wait 2s and retry once; if it fails again, surface the error and let the user retry the variant.
- If a 400 is returned (content policy or prompt issue), show the error message verbatim and offer a retry button.
- Catch network errors and show "Network error — check your connection".

Security note

The API key is held in a JS variable, used only for `fetch` calls to `api.openai.com`, and never written to localStorage or sent elsewhere. Include a small line of text near the input: "Your key stays in this browser tab and is sent only to OpenAI."

Self-contained constraint

The HTML must work offline-once-loaded (apart from the OpenAI calls themselves). No CDN scripts, no external CSS, no fonts that require network. Everything inline.

Prompt Engineering Guidance

Guidance for writing the `Content input:` portion of each per-slide prompt. The `Style input:` portion is fixed and must not be edited.

What goes in Content input

For each slide, the Content input should pack the model with everything it needs to render a meaningful, specific slide:

1. **The action title** — verbatim, since the model will place it at the top of the slide.
2. **The 3–4 key bullets** — these become the labels, callouts, data points, or framework elements on the slide.
3. **A 1–2 sentence framing** — what the slide is arguing.
4. **(Optional) An explicit framework directive** — "Use a [phased roadmap / 2x2 matrix / etc.] as the primary visual framework." This biases the model toward the framework chosen during Phase 2.

Good vs. bad Content input

Bad (too vague):

```
Content input: A slide about our growth strategy.
```

Result: Generic, predictable, low information density. Looks like a template.

Good (specific, dense, structured):

```
Content input: Action title: "Three growth vectors will deliver 2x revenue by 2028 — adjacent geographies lead the
contribution"

Framing: This slide quantifies the three concurrent growth plays and shows their relative contribution to the 2028
revenue target. The audience is the exec committee, who already knows the strategy categories but needs the magnitude
and sequencing.

Key elements to render:
• Geographic expansion: $1.2B contribution (EU + APAC entry, 2026–2028)
• Product extension: $700M contribution (two adjacent SKU lines)
• Enterprise upsell: $400M contribution (top-50 accounts, expanded contracts)
• Combined 2028 target: $4.1B (vs. $2.0B today)
• Key risk: regulatory approvals in two APAC markets gate ~30% of geographic upside

Use a stacked-contribution waterfall or growth-bridge chart as the primary visual framework.
```

Result: Specific numbers anchor the visual; named risks become callouts; framework directive avoids ambiguity.

Specific tips

- **Always give the action title verbatim.** Wrapping it in quotes helps the model treat it as the headline.
- **Numbers are anchors.** Whenever the bullets contain percentages, dollars, dates, or named entities, the model places them prominently. Vague slides ("we will grow significantly") render as vague visuals.
- **Name the framework once.** "Use a 2x2 matrix" or "Use a phased roadmap" is enough; don't describe the framework's mechanics.
- **Cap content density.** If a bullet list has more than ~5–6 items, the model will shrink the text to fit and the slide becomes unreadable. Consolidate into fewer, denser bullets in that case.
- **Don't ask for things the prompt forbids.** The style input bans subtitles, multiple frameworks, and web-style UI. Don't reintroduce them in Content input ("add a subtitle that says...", "show this as a dashboard...") — the model will either ignore you or produce a worse slide.

Important constraints (recap)

- **Model is `gpt-image-2`.** Not `gpt-image-1`, not any other variant. The model ID in the API call must be exactly `gpt-image-2`.
- **Size is `2560x1440`.** Exact 16:9, both edges divisible by 16, within the stable 2K range. Do not substitute non-16:9 sizes.
- **Quality is `medium`.** Not `low`, not `high`, not `auto`.
- **Parallel generation in batches of 5.** Never generate slide-by-slide. Use `Promise.allSettled` to run 10 requests at a time (5 slides × 2 variants), then move to the next batch.
- **One framework per slide.** Never combine multiple frameworks on a single slide. The prompt template enforces this; do not weaken it.
- **Action titles, not topic labels.** Every slide title is a conclusion. "Three structural gaps drive our 23% cost premium" beats "Cost gaps".
- **No subtitle on slides.** The prompt explicitly forbids subtitles. Do not add them to your storyline either.
- **Do not generate images yourself.** This skill produces HTML; the user's browser calls the OpenAI API. Do not attempt to call OpenAI from your own environment.
- **Do not create a .pptx fallback.** If the user wants editable PowerPoint, point them to the standard pptx skill instead.
- **Storyline must be approved before HTML generation.** If you skip the approval gate, the user loses the ability to course-correct cheaply.