**Installation:**

npm install -g yo generator-office

yo office  
**Choose a project type:** Office Add-in Task Pane project

**Choose a script type:** Javascript

**What do you want to name your add-in?** My Project/Powerpoint

**Which Office client application would you like to support?** PowerPoint

**PowerPoint Modification Process**

1. Office.js sends API instructions to PowerPoint.
2. PowerPoint converts these instructions into Open XML format.
3. Changes are applied dynamically within the PowerPoint file.
4. Office.js acts as an abstraction layer that manipulates these XML structures without directly modifying the raw XML.
5. Office.js updates the UI in real-time by sending high-level API commands, which PowerPoint translates into XML-based modifications.

Office.js does not directly edit Open XML but provides an interface to modify the PowerPoint object model efficiently.

**Phase 1: User Input**

Check for existing instructions - Extract comments, sticky notes, and slide notes.

If no instructions found then - ask the user using the PowerPoint Add-in (Text field).

Output: A structured JSON instruction with tasks.

**Phase 2: AI Task Breakdown & Planning**

Convert user instructions into structured tasks

* Send the JSON instruction to the LLM (Gemini 2.0 Flash)
* The LLM analyzes the task and splits it into sub-tasks if needed.
* The LLM decides the best Office.js API calls to achieve the task.
* Returns a structured execution plan in JSON format.

**Phase 3: Dynamic Code Generation**

* Convert structured tasks into executable Office.js code.
* Prompt the LLM to generate JavaScript functions using Office.js APIs.
* Output structured code that can be executed dynamically within PowerPoint.

**Phase 4: Code Validation and Error Handling**

* LLM-based validation before execution if issue found the generated Office.js code is sent back to the LLM for validation.

**Phase 5: Execute Office.js Code**

* Office.js applies the changes dynamically.

**Phase 6: after Execution Verification**

* The add-in retrieves updated slide properties.
* The data is sent back to the LLM for comparison with expected output.
* The LLM checks for differences and suggests corrections if needed.

**Phase 7: Human-in-the-Loop Feedback**

* Allow the user to approve changes or refine them further.

Instruction Agent - Parses user input and classifies tasks.

Code Generation Agent - Generates Office.js JavaScript code.

Execution Agent - Runs the generated code within PowerPoint.

Validation Agent - Ensures modifications align with user instructions.

Error Handling Agent - Detects execution errors and applies corrective measures.

**Backend Automation**

1. The user submits an instruction via the PowerPoint Add-in.
2. The backend processes the request using LLMs and AI agents.

* Executes validation checks before modifying slides.

1. Executes the generated Office.js code within PowerPoint.
2. Fetches the final slide state and sends it back for validation.
3. LLM verifies correctness and suggests corrections if necessary.
4. The user receives a response in the UI indicating success or requesting refinements.

┌────────────┐ HTTP Request ┌────────────────────┐

│ Office.js │ ──────────────────────▶ │ FastAPI/Flask (.py)│

│ Frontend │ JSON Payload │ Backend (AI Agents)│

└────────────┘ ⬅────────────────── └────────────────────┘

Response (JSON)