You defined **seventeen agents** for your lab-generation pipeline. Each performs one discrete function before passing its output to the next:

1. **Agent 1 – Requirements & Tasks Generator**
   • Input: Topic prompt.
   • Output: 3–5 requirements, each with 3–5 detailed tasks.
   • Purpose: Break the topic into a structured lab framework.

WORKFLOW:

01. Create a Lab Series
02. Job Role Mapping
03. Lab Series Proposal
04. Create steps for each task
05. Apply Challenge Format
06. Create Sample Data
07. Convert Steps to Guided Hints
08. Create Advanced Hints
09. Insert Help, Notes, Alerts
10. Improve Task Wording
11. Improve Requirements
12. Create a Challenge Lab Title
13. Improve the Overview
14. Create Summary Objectives
15. AI Tech Edit
16. Scripting
17. Update Proposal

PROMPTS:

01. Create a Lab Series

I want to create a series of 12 learning labs. Provide them one Lab at a time so I can add feedback and make adjustments:

- Each lab should consist of a Title.
- Each lab should consist of 4-5 different Requirements.
- Each lab should consist of 3-5 different Tasks per Requirement.
- The total time to perform all tasks within the lab should be approximately 30 minutes.
-

Using the 12 Labs, also generate a single sentence name for the Lab Series to fall under: OUTPUT: 1 lab series consisting of: - 12 Labs with: - 4-5 Requirments per lab consisting of: - 4-5 Tasks per Requirement The series should be based on the following topic:

02. Job Role Mapping

Tell me what type of job role would each lab relate the most to using the following job roles: administrator, architect, data engineer, developer, dev ops engineer. Or if none apply, suggest a job role please.

03. Lab Series Proposal

You are a Skillable Challenge Lab content reviewer.
Scan all uploaded **completed Lab Series files** and extract the data needed to populate the **Proposal Template.xlsx**.
After analyzing the entire series, return the output in **CSV-ready format** for direct export into Excel.

Extraction Requirements

Series-Level Fields

Identify once per series:

* **Challenge Series Name**
* **Brief Series Summary / Description**
* **Primary Job Role** (based on `@lab.Variable(GlobalJobRole)` token or contextual inference)

Per-Lab Fields

For every lab in the series, extract and include:

1. **Challenge Lab Number**
2. **Type of Lab** *(Guided / Advanced / Expert)*
3. **Challenge Title** *(from `>[challenge-title]:`)*
4. **Challenge Lab Overview** *(from `[overview]:` section)*
5. **Description / Objective** *(first paragraph of the overview in plain text)*
6. **Scorable Tasks** *(each "- " task line summarized as clean bullets separated by semicolons)*
7. **Exam Mapping** *(e.g., AZ-104, SC-200, or mark "None" if not applicable)*

Output Format

Return the results in **CSV format**, using this exact column order:

```
Challenge Series Name,Series Description,Job Role,Challenge Lab Number,Type of Lab,Challenge Title,Overview,Description/Objective,Scorable Tasks,Exam Mapping
```

Each lab entry should appear as a new row.
Preserve commas inside text by wrapping each field in double quotes (`"`).

Formatting Rules

* Keep any Skillable tokens (`@lab.Variable(...)`) intact.
* Remove Markdown syntax (`>`, `:::`, `**`, etc.).
* Condense long paragraphs while preserving full instructional meaning.
* Separate multiple tasks within a lab using `; ` so they stay in one Excel cell.
* If data is not available, insert `"Not Specified"`.

Final Step

After the CSV table, provide:

1. A brief **summary paragraph** describing how the series progresses across labs (e.g., from foundational to expert level).
2. A **one-sentence description** of the learning outcome for the full series.

04. Create steps for each task

PROMPT:
Scan the attached Lab document, and break each task into detailed step-by-step instructions. Include all examples as required steps.
You are given one or more Requirements, each containing several Tasks. For each Task:

- Create clear, numbered, step-by-step instructions.
- Every step must be concise, actionable, and ordered logically from start to finish.
- If any Task mentions sample data, example files, or setup, that is not already present in the instructions, add a preparatory Task with detailed steps to create that data or resource.

FORMATTING RULES:

- Each step starts with ">- "
- Leave one blank line between steps, with only ">"
- Preserve any provided formatting.
- Maintain consistency with the Challenge Lab structure.
- Integrate provided examples directly into the steps — **do not omit or rewrite them.**
- Do not combine or shorten multiple steps into one.
- Use verbs like Select, Enter, Create, Save.
- Duration target: 30–45 minutes to complete.

EXAMPLE USE PATTERN:

- Minimum 3 tasks per requirement.
- I need steps for each task. Detailed step-by-step.
- Include examples provided in the document.
- If any task needs sample data, add a task and steps to create it.

05. Apply Challenge Format

Convert this lab into the Challenge Lab Format using the following: **Challenge Lab Format.pdf**

- Remove all occurrences of "**Lab**" or "**Task**" from the instructions
- Replace with the supplied task bullet
- Convert into 1 continuous segment of markdown

06. Create Sample Data

PROMPT:
If any task requires input data, example files, or configuration values, insert a dedicated task titled **Create Sample Data**.

- Include clear, numbered steps showing exactly how to generate or obtain the data (for example: create a CSV file, prepare a text file, or access a sample API endpoint).
- The data must be:
    - Realistic
    - Usable in subsequent tasks
    - Stored in a consistent location.

07. Convert Steps to Guided Hints

Convert the Steps into Hints by using the following formatting:

• Wherever you see: "Expand this hint for guidance on", complete the sentence using the bulleted Task directly above as a reference.

Each set of steps will begin with:

:::ShowGuided(ShowGuided=Yes)
>[+hint] Expand this hint for guidance on...
>
>-

Each set of steps will end with:

>
:::

08. Create Advanced Hints

Use the Sample Markdown Elements file and enter the following: After each [+hint], using the Sample Markdown Elements file to add an Advanced Hint. Advanced Hints • If the outline contains: >[!knowledge] Want to learn more? Review the documentation on … → Search for certified reference links related to the topic in brackets. → Insert those official links so they can replace placeholders.

Advanced Hints • If the outline contains: >[!knowledge] Want to learn more? Review the documentation on … → Search for certified reference links related to the topic in brackets. → Insert those official links so they can replace placeholders.

09. Insert Help, Notes, Alerts

Scan the attached document and suggest for every requirement section:

1 Alert. (Insert this directly after the Task that would trigger the alert)
2 Useful Note blocks related to the tasks. (Insert this directly after the Task that the note would apply to)
1 Helpful tip. (Insert this directly after the Task that the Help/Tip would apply to)

- Organize them by Requirement section. and then format them according to Challenge Lab markdown specs.

- Don't leave ANY of the Markdown out or change any existing markdown. Only insert it. txt.

- Use the Sample Markdown Elements.pdf document for correct formatting.

10. Improve Task Wording

Improve my Task wording by:
- Splitting all Tasks where multiple functions are being performed, into their own individual tasks.
- Making the actual task more detailed than 3 or four word bullets.
- The Task should include any details required (possibly mentioned in the corresponding steps) to complete the Task.
- Remove any reference to "Task #:" and replace with "- "

11. Improve Requirements

Replace the Requirement Headers with actual titles.
- Use the tasks within that requirement section to produce the Requirement title.
- No punctuation
- Sentence case
- Short and precise

12. Create a Challenge Lab Title

When you see the following: ">[challenge-title]: Challenge Title Here"
Replace "*Challenge Lab Here*" with a **new title**.
The title should:

- **Begin with a verb**
- In P**resent Tense**
- The **verb can not end in "ing" if it starts the Title**.

Scan all of the Requirements (these begin with "**#**", and then create the single sentence title.

CHALLENGE TITLE RULE:

- Replace any placeholder ">[challenge-title]: <Lab Title>" with a new title.
- Title must start with a verb in present tense (no "ing").
- Derive from scanning all Requirement headers (those beginning with #) and summarize them in a single concise sentence.

For each lab, using all of the included excercises within that specific lab, create a new, one sentence long, Lab title that begins with a verb, but not using "ing".

FORMAT:
>[challenge-title]: Challenge Title

REQUEST:
Replace any placeholder with a new title. Title must start with a verb in present tense (no "ing"). Derive the title from scanning all Requirement headers (those beginning with #) and summarizing them in a single concise sentence.

13. Improve the Overview

Overview Rule:

[overview]: Context paragraph describing the scenario (include role and company variables if available).

Use this format and verbiage:

>[overview]:
>You are a @lab.Variable(GlobalDeveloper) at @lab.Variable(GlobalCompany), a company that needs to [INSERT WHY YOU WOULD BE COMPLETING THESE REQUIREMENT]. In this Challenge Lab you will [INSERT MORE DETAILED VERSION OF CHALLENGE LAB TITLE]. First, you will [Requirement 1], and then you will[Requirement 2]. Next, you will [Requirement 3], and then you will [Requirement 4]. Finally, you will [Requirement 5], and then you will [Requirement 6].

Replace [Requirement] placeholders with short summaries of the requirement sections.

14. Create Summary Objectives

Objectives
• After Summary.
• 3–5 bullets based on the Requirments included in the lab.
• Each starts with an action verb in second person (“Created,” “Used,” “Archived,” “Validate”).
• Past tense
• Replace the "Past tense list of requirements" with the new completed objectives.

Replace **CHALLENGE LAB TITLE?** with the actual new Challenge Lab Title.

FORMAT:
>[recap]:
>Congratulations, you have completed the **CHALLENGE LAB TITLE?** Challenge Lab.
>
>You have accomplished the following:
>
>- Past tense list of requirements.

15. AI Tech Edit

Take the following Markdown Challenge Lab document and perform these operations in order:

1. **Spelling Check**

* Identify and correct all spelling errors throughout the Markdown.
* Retain capitalization and special syntax like `@lab.Variable()`, code fences, and block identifiers (`:::`).
* Do not modify URLs, tokens, or variable names.

2. **Duplicate Word Check**

* Locate any repeated words (e.g., “the the”, “in in”) and remove duplicates while preserving original sentence meaning and spacing.

3. **Grammar Check**

* Fix improper grammar, punctuation, and sentence structure while maintaining the Skillable Challenge Lab instructional tone.
* Keep imperative task phrasing (“Select”, “Enter”, “Verify”).
* Do not convert bullet structures, hint blocks, or Markdown headings.

4. **Type Text Conversion**

* Scan the lab for all text or values that the learner is expected to *type or enter* (e.g., credentials, commands, URLs, filenames, parameter values).
* Wrap each of those entries in triple plus signs (`+++`), following the Skillable Markdown “Type Text” rule:

```
+++text-to-type+++
```
* Examples:

* Convert `Sign in as User1` → `Sign in as +++User1+++`
* Convert `Enter https://portal.azure.com` → `Enter +++https://portal.azure.com+++`
* Convert `Password: Passw0rd!` → `Password: +++Passw0rd!+++`
* Do not modify inline code blocks (```) or variable tokens (`@lab.Variable(...)`).

5. **Output Formatting**

* Deliver the corrected Markdown as clean, valid Skillable Challenge Lab format (preserve all `ShowGuided`, `ShowAdvanced`, `ShowActivity`, and `!INSTRUCTIONS` sections).
* Do not add or remove any structural Markdown.
* Final output should be ready to paste directly into Skillable Studio.

16. Scripting

You write a single verification script for Skillable Challenge Labs.

Scope: Only verify the tasks listed directly above the current "## Check your work @lab.ActivityGroup(...)" within this one Requirement. Do not verify anything from other Requirements.

Environment:
- Preferred shell: PowerShell 7+ on Windows. If the Requirement is clearly Linux/macOS, output Bash instead.
- Script must run with no external modules.

Output contract:
- Emit one self-contained script only (no prose).
- For each task, implement a deterministic check and label it exactly with the task text.
- Print one line per task: "PASS | <task>" or "FAIL | <task> | <actionable hint>".
- At end, print "SCORE <passed>/<total>".
- Exit code 0 only if all tasks pass; otherwise non-zero.

Design rules:
- Use small, readable helpers (e.g., Test-File, Test-Registry, Test-Command, Test-Service on Windows; file/service/command checks on Linux).
- Never make changes; read-only checks only.
- Keep timeouts short (≤5s per check).
- If a task is ambiguous, choose the most reliable, observable artifact (file, process/service, CLI query, API call, resource state).
- Avoid false positives. Prefer exact matches, case-insensitive where appropriate.
- Assume tokens like @lab.Variable(...) may appear in names; treat them as literal text unless replaced.

Inputs you must use:
- Requirement title: "<paste requirement title>"
- Tasks (verbatim bullets from this Requirement, top to bottom):
1) "<paste task 1>"
2) "<paste task 2>"

...
- Any concrete paths, names, commands, or IDs visible in the Requirement.

Deliverable:
- One script implementing the checks for these tasks, following the output contract.

EXAMPLE OF A CURRENTLY FUNCTIONING SCRIPT:

```
# Verifies the creation of the SyllableIndex.html file and some representative content.
# This must run on CL1.

# Set default return value
$result = $false

# Set this to $True for temporary troubleshooting to display any errors. Set to $False for
production.
# Create a lab variable 'debug' and set to true to debug all scripts in the lab.
$scriptDebug = '@lab.Variable(debug)'
if ($scriptDebug -eq 'True') {
$scriptDebug = $true
} else {
$scriptDebug = $false
}
if ($scriptDebug) {Write-Output "Debug mode is enabled."}

# Modify the code below to suit the needs of the validation being performed.

# Query variables. Modify these to match the requirements of the lab environment.
$file = 'D:\LabFiles\SyllableIndex.html'
$queryString = '*Syllable Splitter*'

# Add the commands for your scenario here:
$queryReturn = try {
[string](Get-Content $file) -like $queryString
} catch {$null}

# This is an example of optional debug output for an individual command. Insert this after
the command using the
# appropriate variable. This will only output if $ScriptDebug is set to $True
If ($scriptDebug) {Write-Output $queryReturn | Out-String}

# Perform your validation testing here:
If ($queryReturn) {
```

```
$result = $true
} Else {
$result = $false
}

# This is optional debug output. This will only output if $scriptDebug is set to $True
If ($scriptDebug) {Write-Output "Error output:`n`n$Error" | Out-String}

# Using the Return keyword here to return the True or False value.
Return $result
```

17. Update Proposal

Update Proposal with New Lab Titles and any additional details that might have changed.