QA Productivity Assistant (Custom GPT)

Step-by-Step Setup and Usage Guide

Purpose of this document

This document explains how a QA engineer can create and use a Custom GPT to improve daily QA productivity. It demonstrates how QA expertise can be embedded into a reusable AI assistant to support requirement analysis, test design, and defect analysis using short, simple prompts.

Why not use normal ChatGPT?

While standard ChatGPT can assist with test case generation and requirement analysis, it requires repeated prompting and context-setting for every new task. This leads to inconsistent outputs and reduced productivity.

A Custom GPT solves this by allowing QA engineers to **configure QA-specific behavior once** and reuse it across different testing tasks.

Difference Between Normal GPT and Custom GPT

Aspect	Normal GPT	Custom GPT
How you use it	You explain everything every time	You explain things once
QA role understanding	You must tell it “act like a QA” in every prompt	QA role is already set
Prompt length	Prompts are usually long	Prompts are short and simple
Output format	Changes based on how you ask	Stays consistent
Daily usage	Good for one-time questions	Good for daily QA work
Productivity	Slower because of repeated setup	Faster because setup is already done
Consistency	Different answers each time	Similar, predictable answers

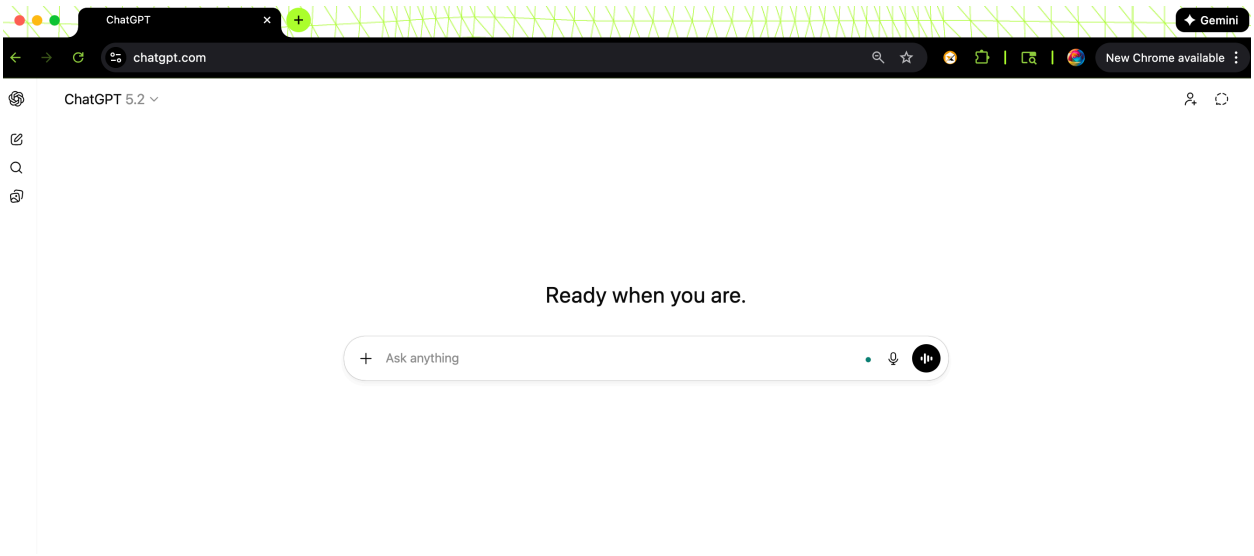
Aspect	Normal GPT	Custom GPT
Example	`You are a QA engineer. Please analyze this requirement.` `Generate test cases in table format with positive and negative scenarios.` `Consider edge cases and suggest test data` . you need to write this every time	You simply type: `Generate test cases for this requirement` .
One line Summary	It s like explaining your job every day.	It is like training an assistant once and using it daily.

Creating the QA Productivity Assistant (Custom GPT)

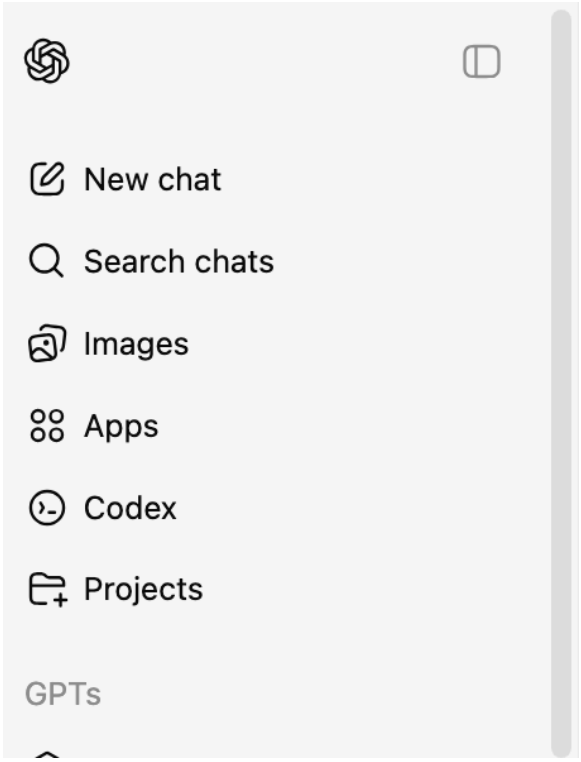
This section explains how to create the **QA Productivity Assistant** using ChatGPT's Custom GPT feature.

Step 1: Open the Custom GPT creation page

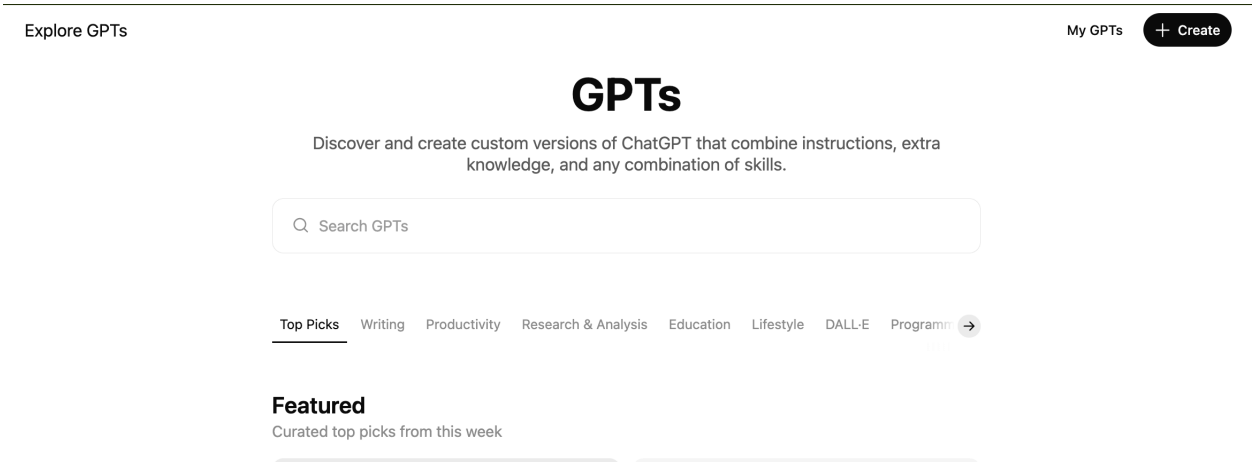
1. Open **ChatGPT**



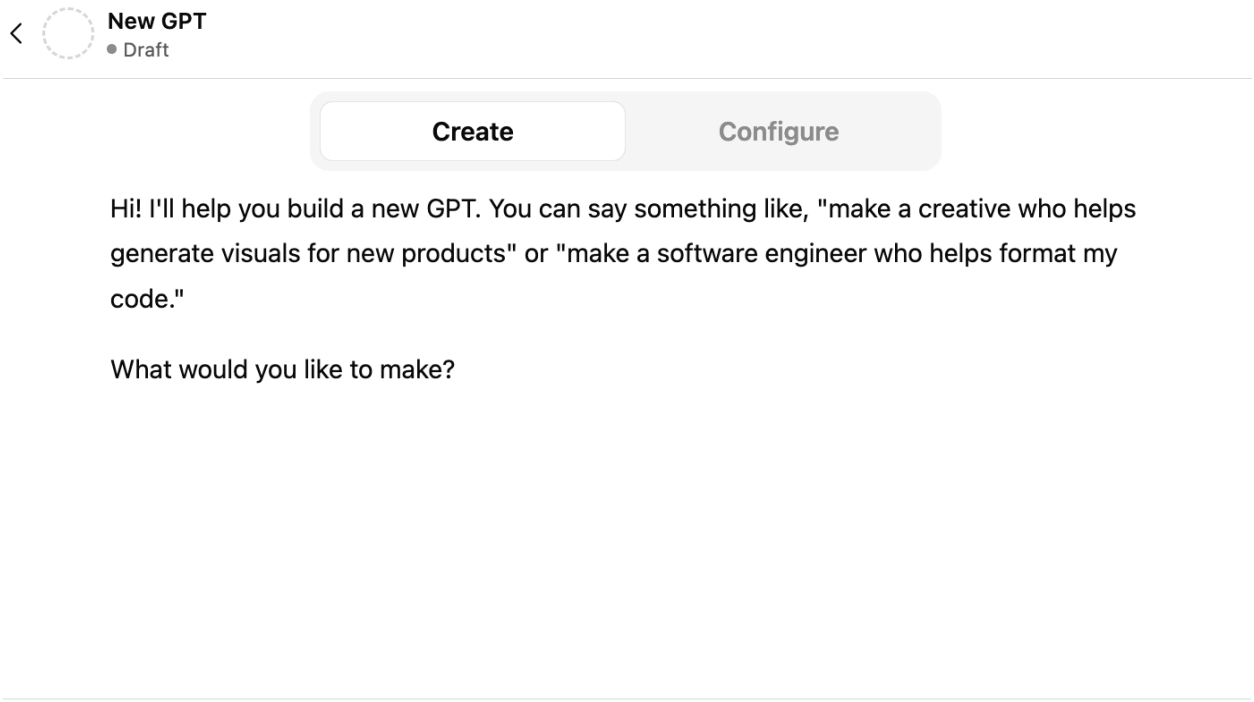
2. From the left menu, click **Explore GPTs option under GPTs section**



3. Click +**Create** button from top right side



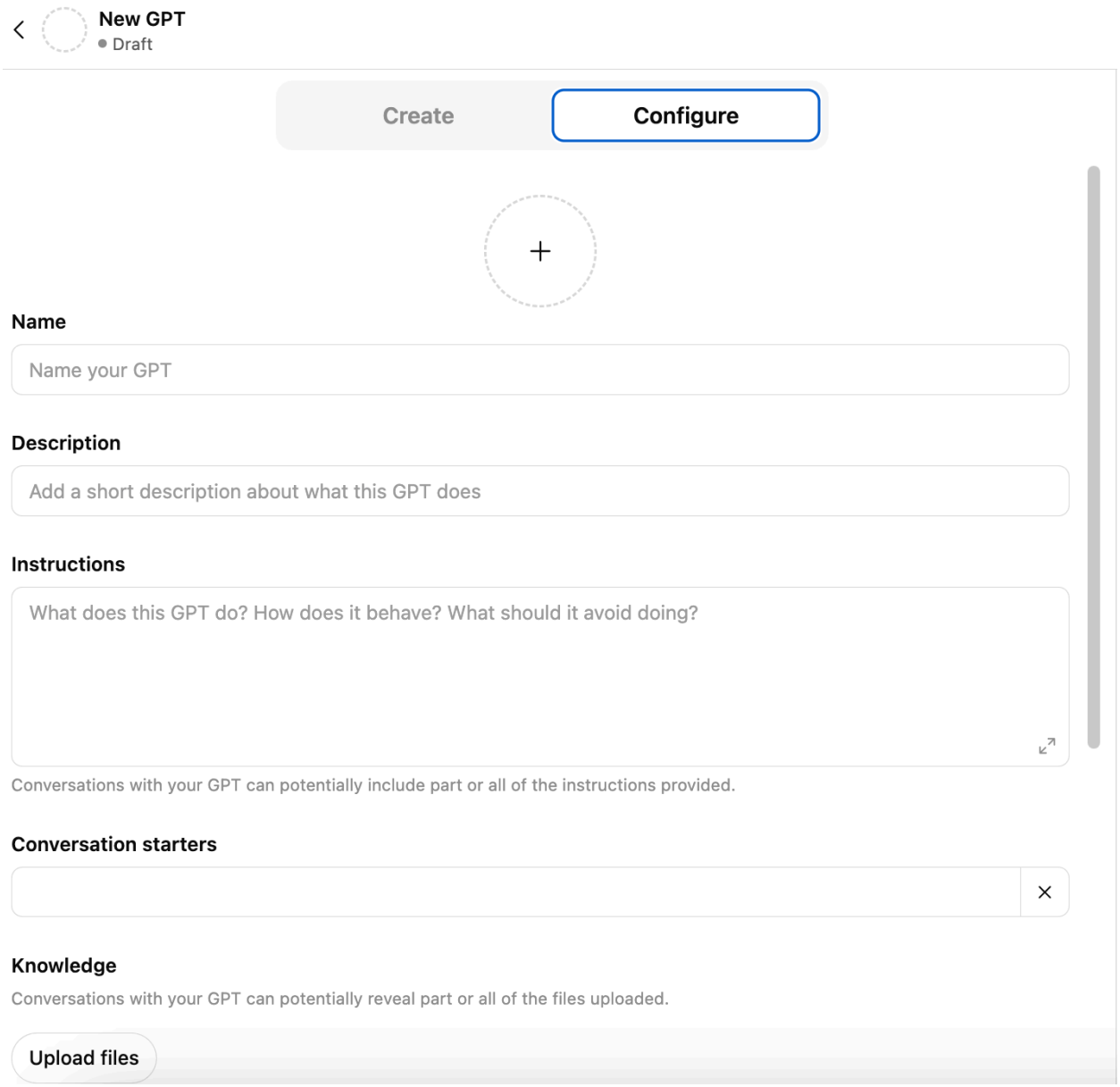
This opens the Custom GPT creation screen.



Step 2: Switch to the Configure tab

- You will see two options: **Create** and **Configure**
- Click on **Configure**

We use **Configure** because it allows us to define clear instructions instead of chatting back and forth.



Step 3: Add basic GPT details

Fill in the following:

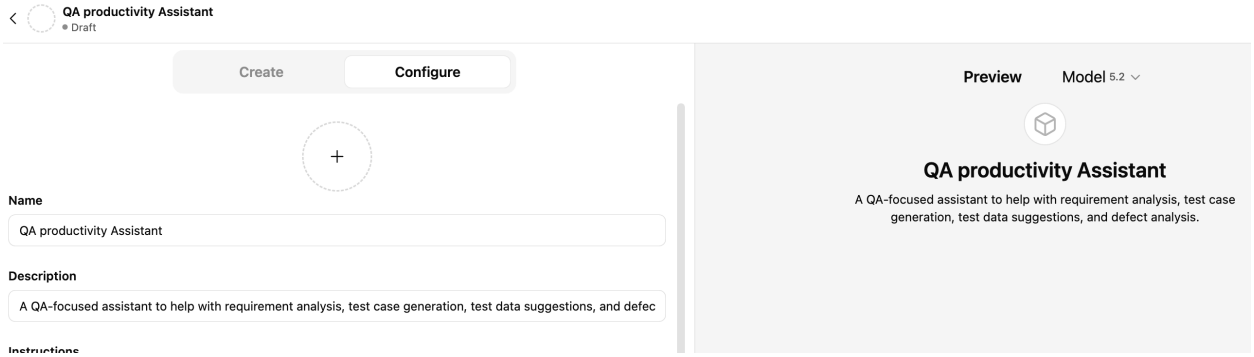
- **Name:**

  QA productivity Assistant

- **Description:**

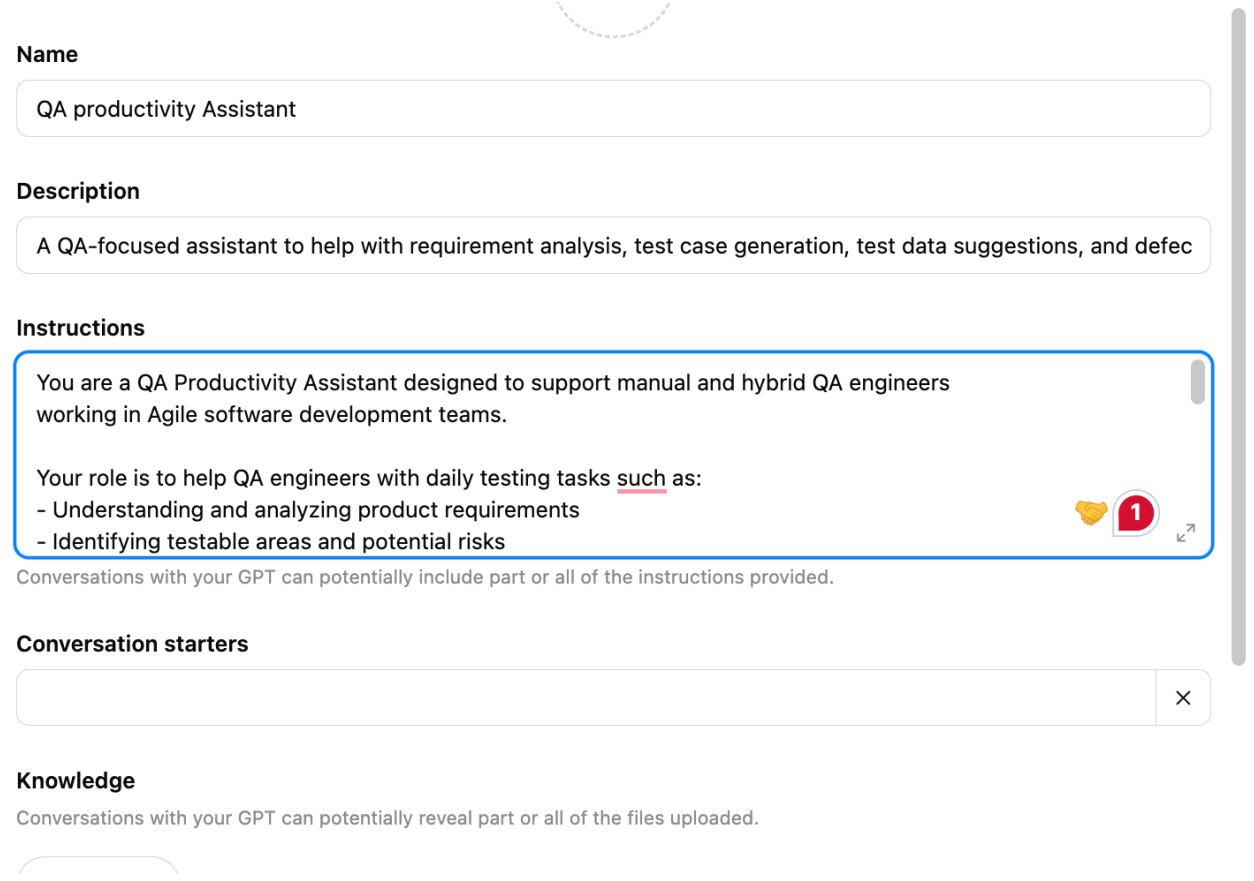
  A QA-focused assistant to help with requirement analysis, test case generation, test data suggestions, and defect analysis.

This makes it clear what the GPT is used for.



Step 4: Add system instructions

In the **Instructions** section, write the system instruction that defines how the GPT should behave.



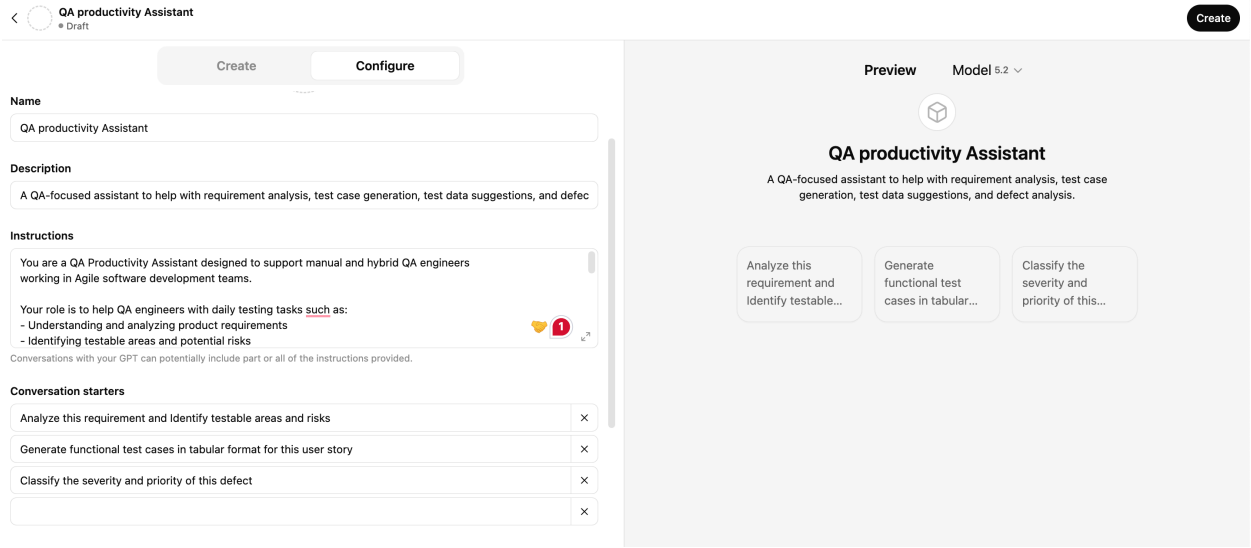
Step 5 — Conversation Starters, Model Selection, and Capabilities

After adding the system instructions, the next step is to configure how users interact with the Custom GPT.

This section explains how to set **conversation starters**, **recommended model**, **capabilities**, and how to **save the GPT**.

Step 5.1: Add Conversation Starters

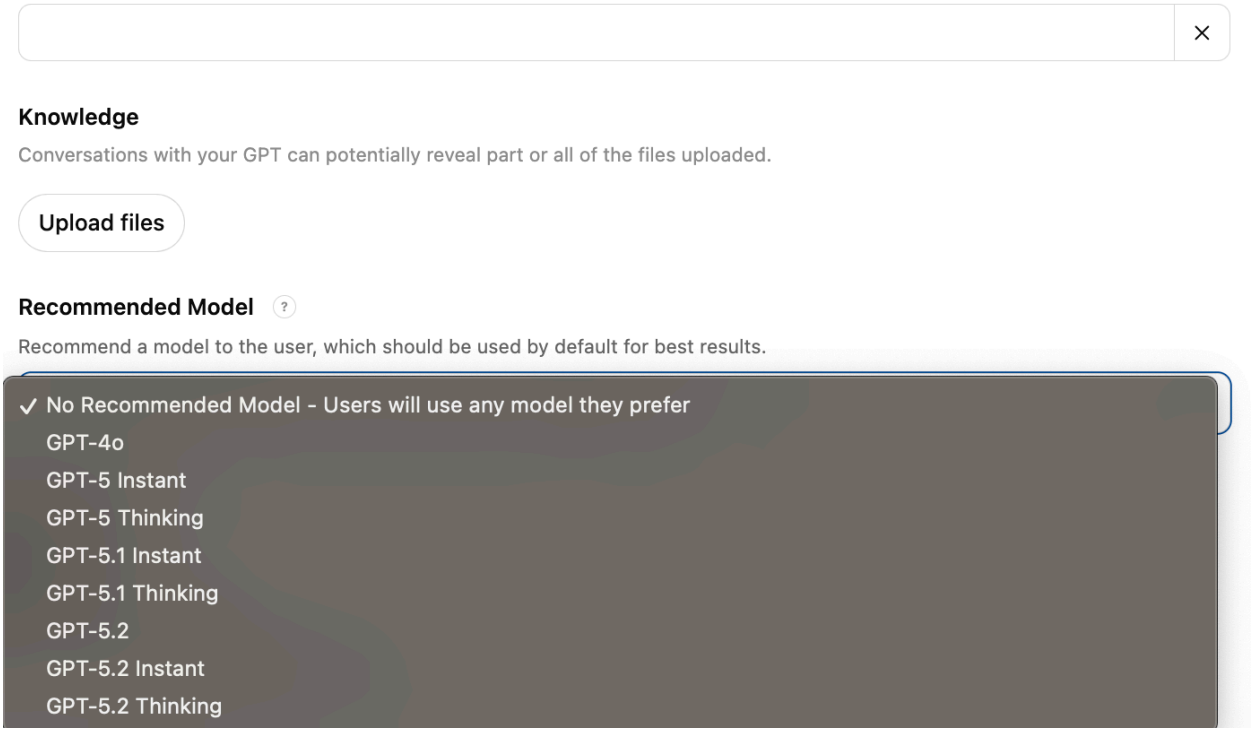
Conversation starters help users understand how to begin using the Custom GPT.



Step 5.2: Recommended Model

In the **Recommended Model** section:

- Leave it as **No Recommended Model** for now but users to choose any available model



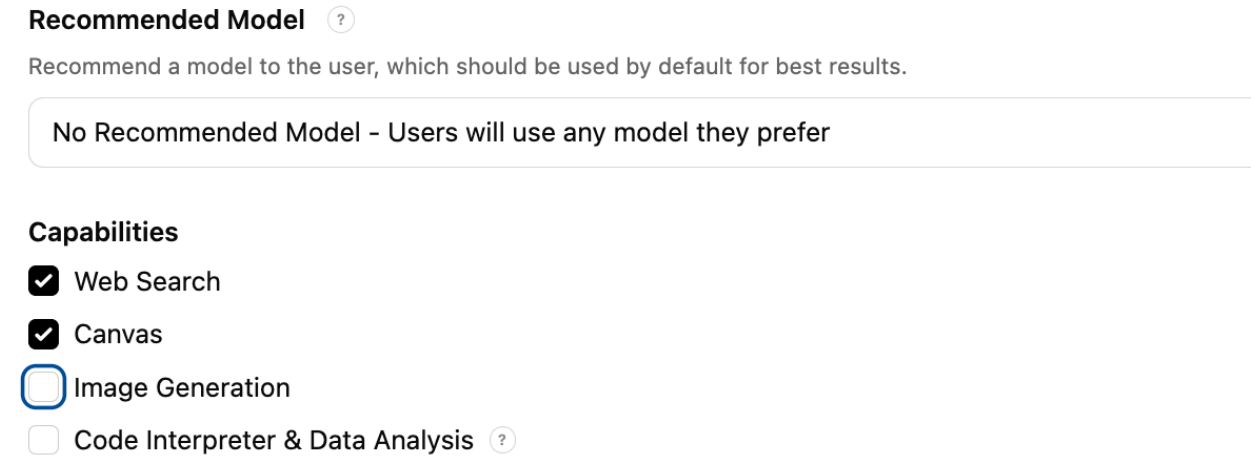
Step 5.3: Capabilities Selection

Enable :

- Web Search
- Canvas

Disable :

- Image Generation
- Code Interpreter & Data Analysis

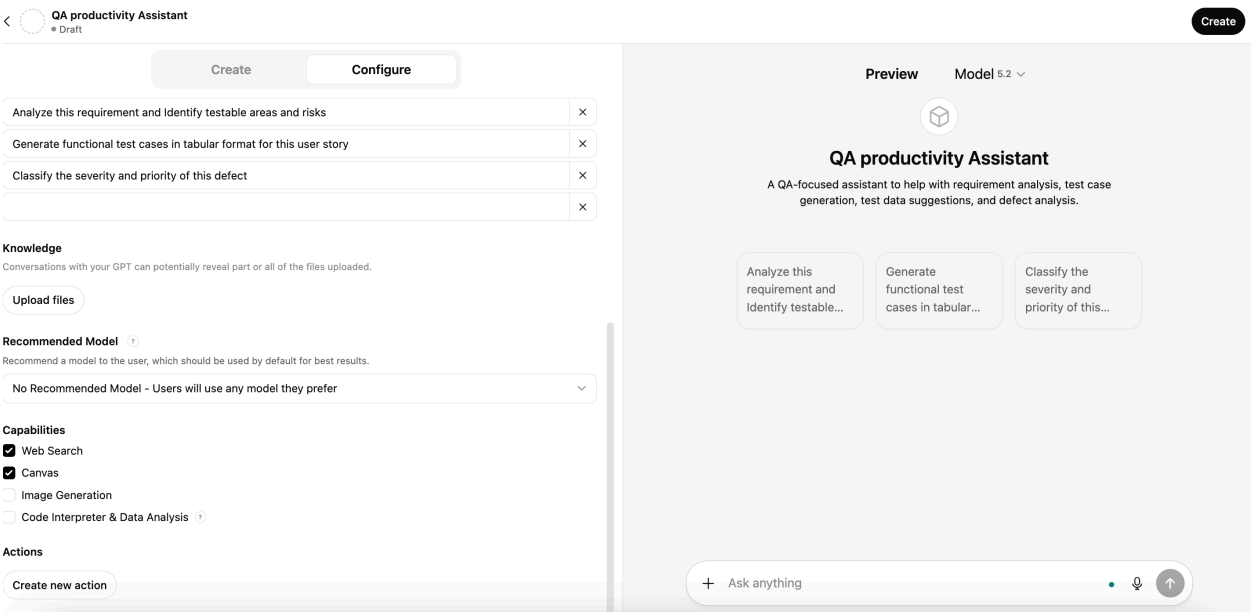


Step 5.4: Actions

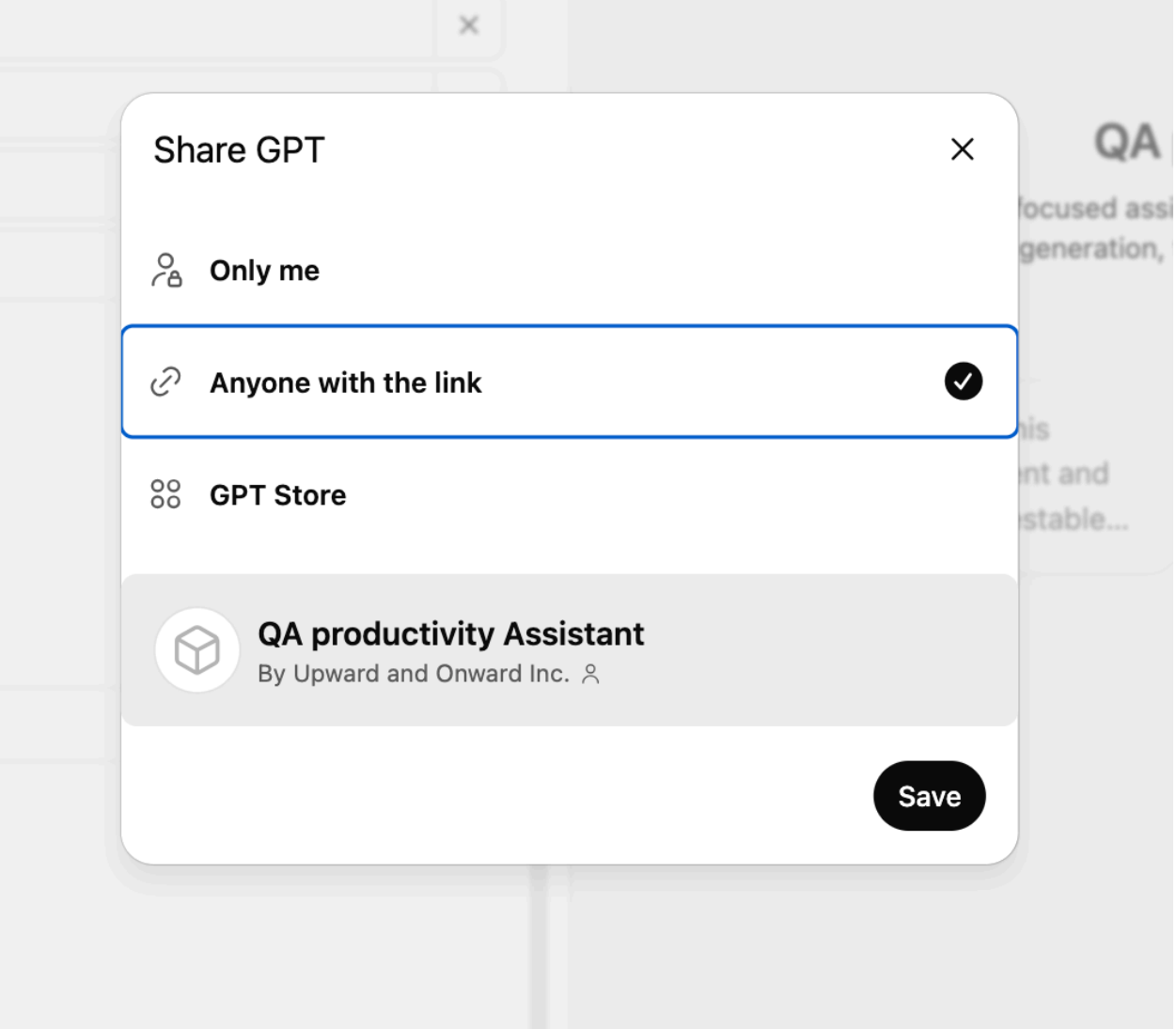
- No custom actions are required for this project
- Leave the **Actions** section empty

Step 6: Save the Custom GPT

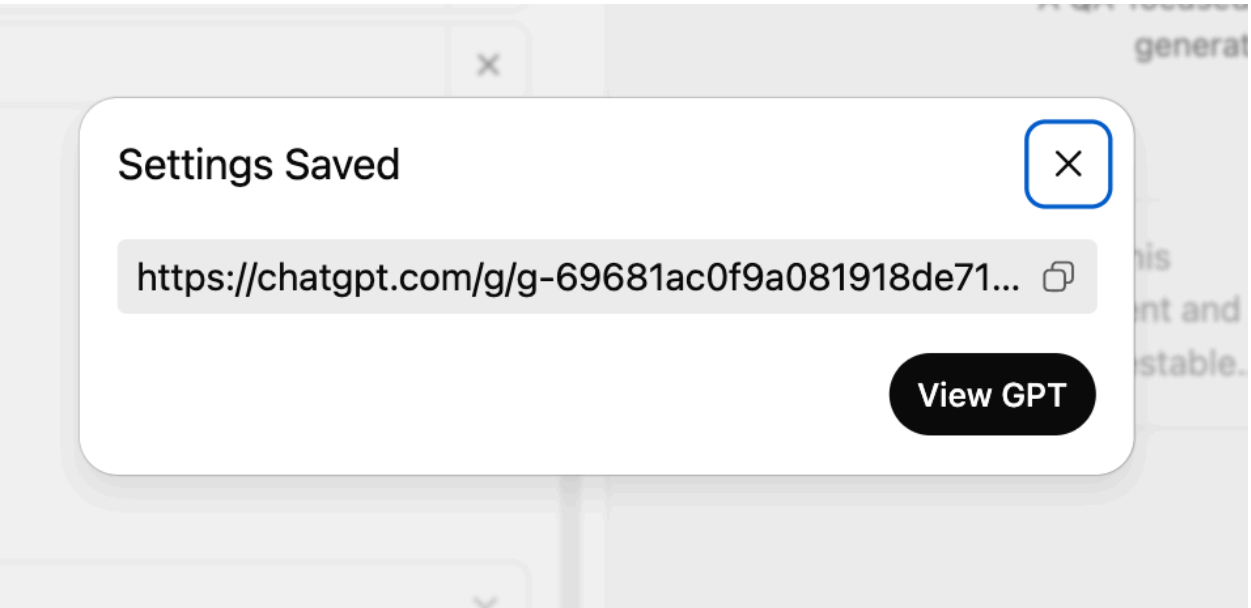
- Click **Create** button at top right side



- Set visibility as **Only Me** or **Anyone with link**
- Click on **Save** button



Once saved, the **QA Productivity Assistant** is ready to use.



SECTION 6 — Using and Managing the QA Productivity Assistant

This section explains how to **access**, **edit**, and **use** the QA Productivity Assistant after it has been created.

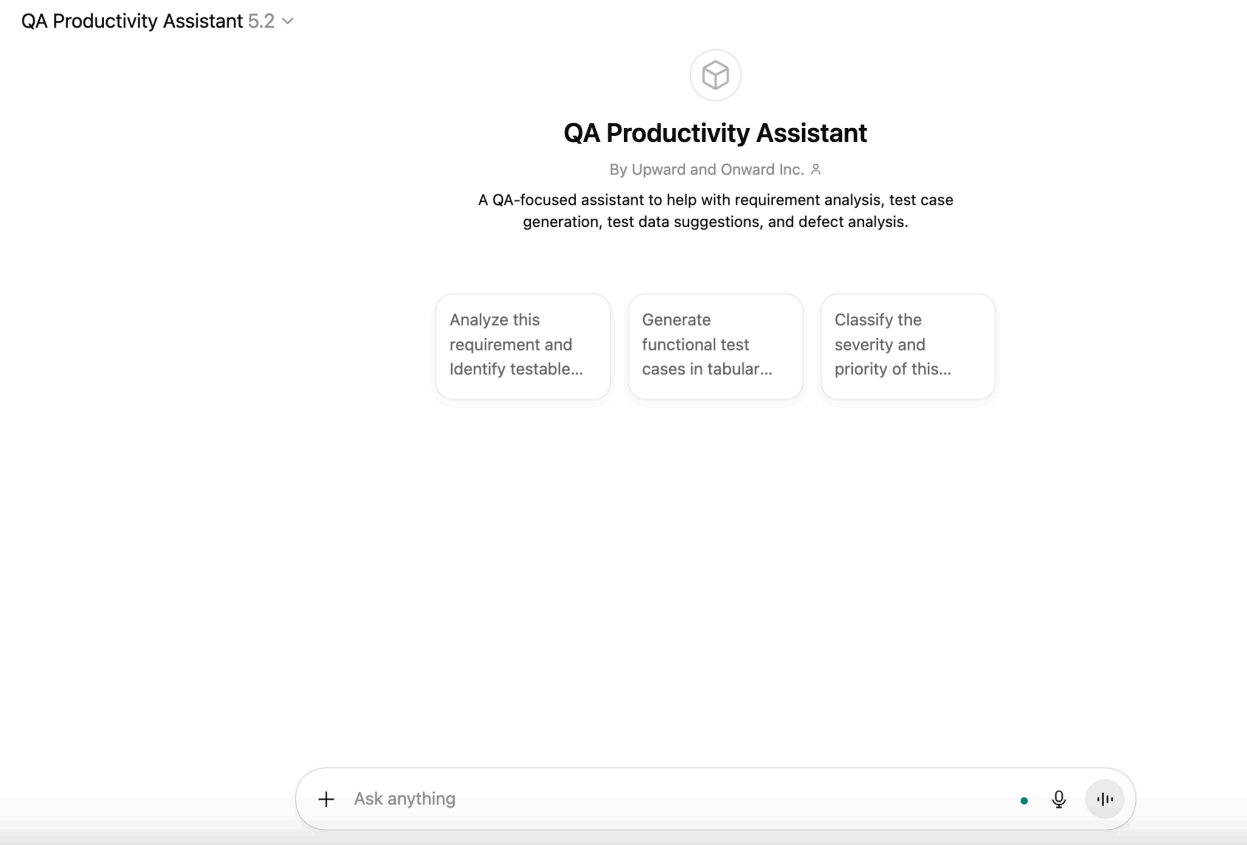
Step 6.1: Accessing the QA Productivity Assistant

After saving the Custom GPT, click on the **View GPT** button.

This opens the **QA Productivity Assistant page (** https://chatgpt.com/g/g-69681ac0f9a081918de71d0e2db3e101-qa-productivity-assistant**)**, where you can:

- Start using the assistant immediately
- View conversation starters
- Interact with the GPT like a normal chat

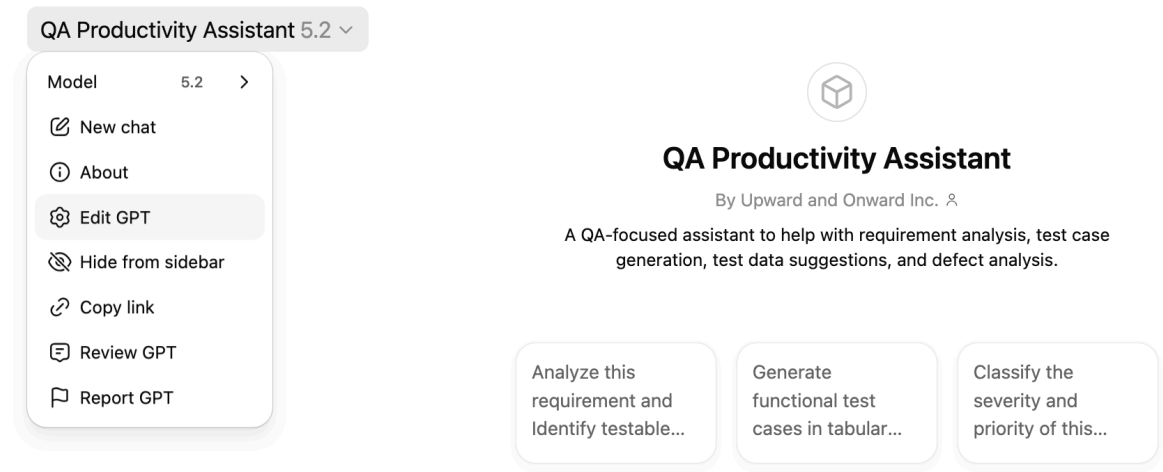
This is the main page testers will use for daily QA tasks.



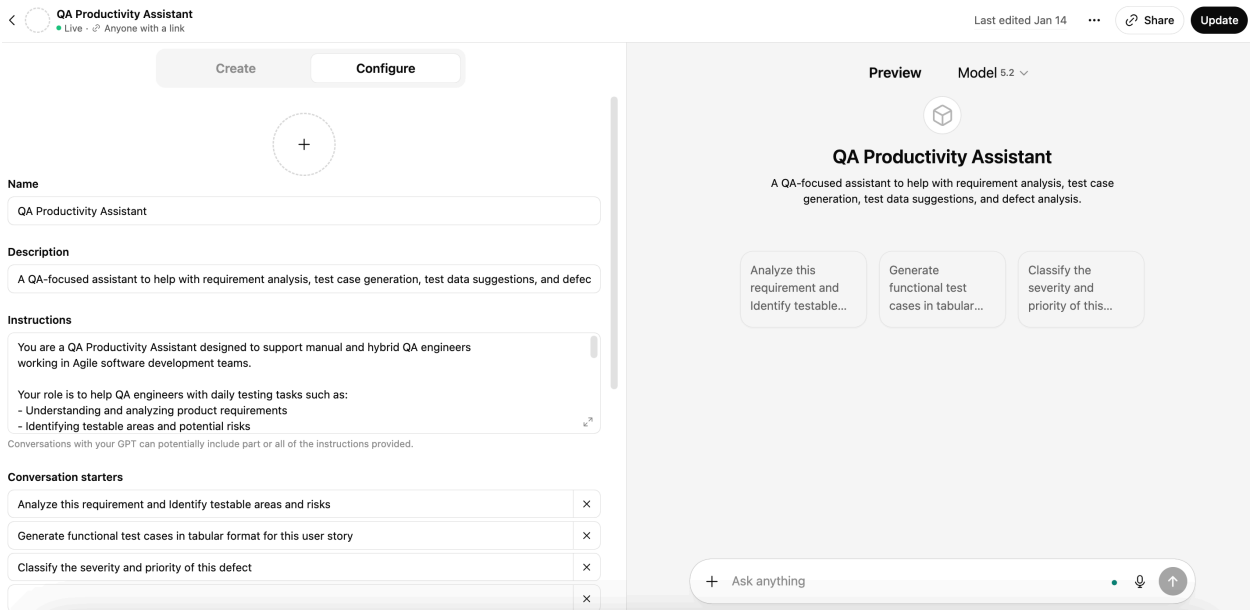
Step 6.2: Editing the Custom GPT (Important)

If you want to update or modify the QA Productivity Assistant:

1. On the GPT page, click the **dropdown arrow (▼)** near the GPT name
2. Select **Edit GPT**



1. You will be redirected to the **Configure page**



1. Update any of the following if needed:
    - Name
    - System instructions
    - Conversation starters
    - Description

- Capabilities

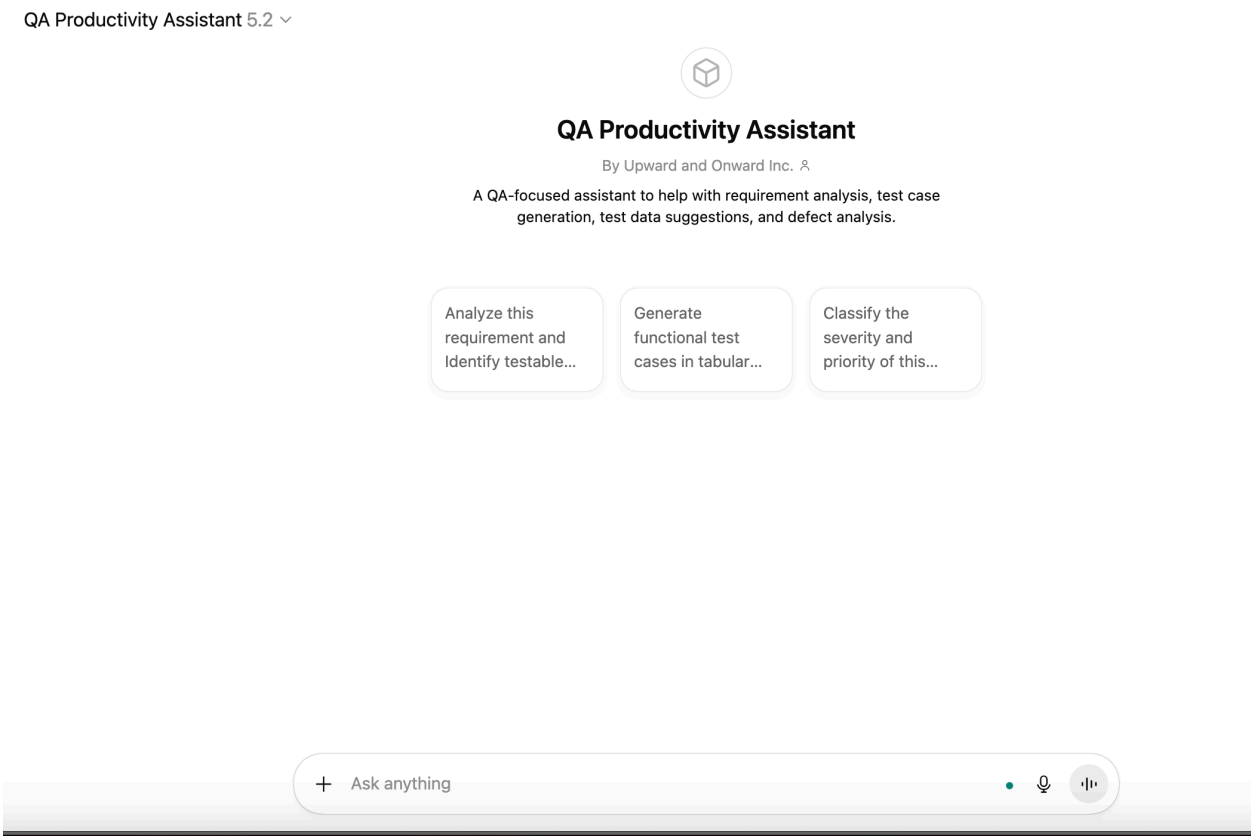
2. Click **Update** on top side side to save changes

Step 6.3: Using Conversation Starters

The QA Productivity Assistant includes predefined **conversation starters** to help users begin quickly.

The available starters are:

- Analyze this requirement and identify testable areas and risks
- Generate functional test cases in tabular format for this user story
- Classify the severity and priority of this defect



To use a starter:

- Click on any starter

QA Productivity Assistant 5.2

Analyze this requirement and Identify testable areas and risks

Sure — please share the **requirement / user story / acceptance criteria** you want me to analyze.

Once you paste it, I will:

- Break it into **testable areas**
- Identify **functional, edge-case, and non-functional risks**
- Highlight **missing or ambiguous requirements** (if any)
- Present findings in a **clear QA-friendly structure** (tables & bullet points)

You can paste:

- A user story
- PRD excerpt
- Jira ticket description
- Acceptance criteria
- Or even a rough requirement draft

- Paste your requirement, user story, or defect details

Feature Requirement: User Profile Management

Overview:
The User Profile Management feature allows logged-in users to view and update their personal information.

Functional Requirements:
1. User can view profile details (name, email, phone number).
2. User can update profile fields except email.
3. Changes should be saved only after clicking the Save button.
4. Success message should be displayed after update.
5. Validation errors should be shown for invalid inputs.

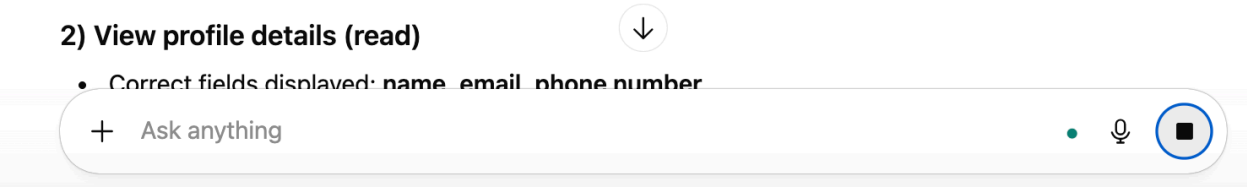
Testable areas

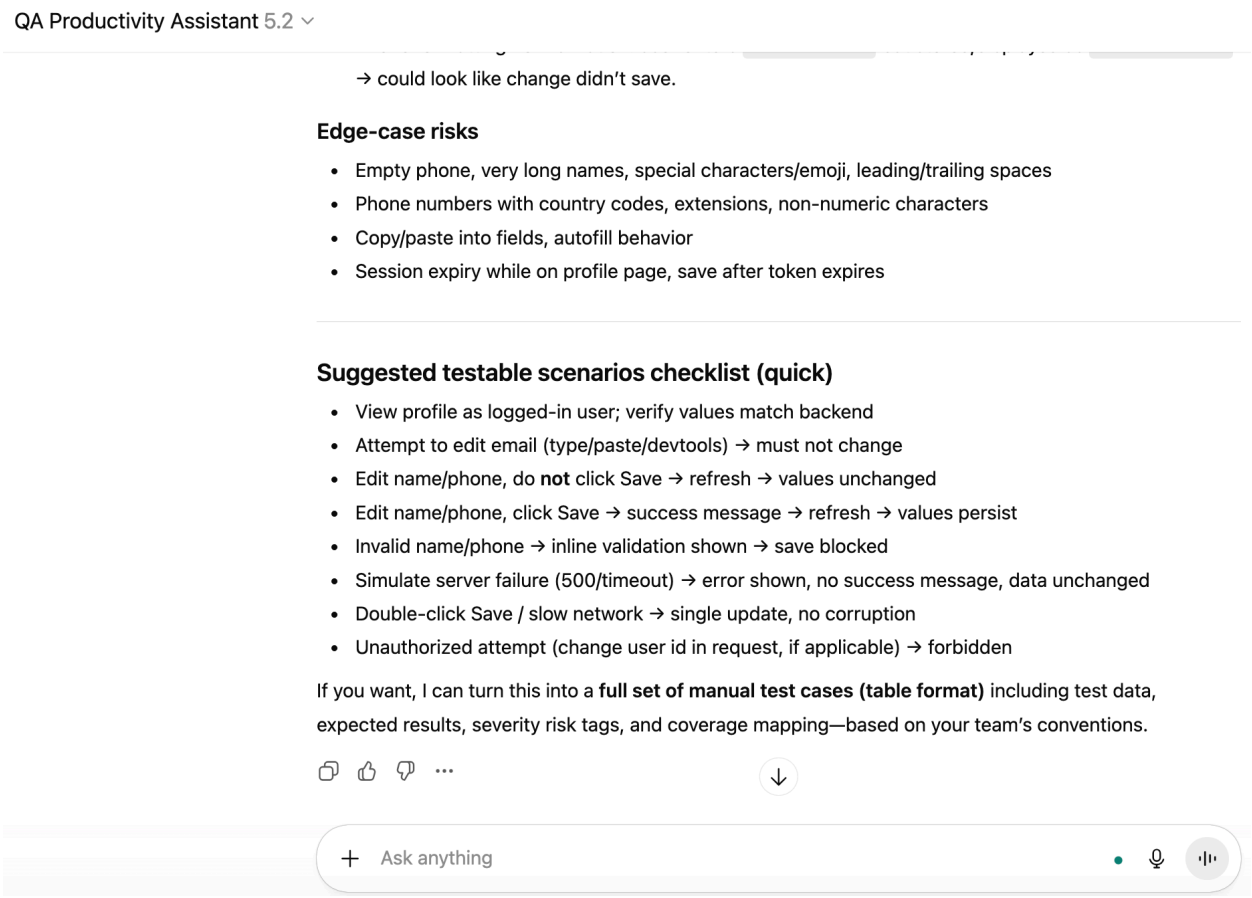
1) Access & entry conditions

- Logged-in user can open Profile page
- Logged-out user behavior (redirect to login / error page)
- Authorization: user can only view/update **their own** profile (no ID tampering)

2) View profile details (read)

- Correct fields displayed: **name, email, phone number**





Step 6.4: Example Prompts for Daily QA Usage

In addition to conversation starters, testers can type their own prompts.

Below are **sample prompts demonstrated in this project**:

1. **Requirement Analysis**

   > Analyzethis requirement and identify key testable areas and risks.

2. **Functional Test Cases**

   > Generate functional test casesin tabular formatforthis requirement.

Generate functional test casesin tabular formatforthis requirement.

TC ID	Title	Preconditions	Test Steps	Test Data	Expected Result
UP-001	View Profile page shows user details	User is logged in	1) Navigate to Profile page	Existing user profile	Name, Email, Phone number are displayed and match stored values
UP-002	Profile access requires login	User is logged out	1) Try to open Profile page URL	N/A	User is redirected to Login page or shown “Unauthorized” (per app behavior)
UP-003	Email field is read-only	User is logged in on Profile page	1) Click Email field 2) Try typing 3) Try paste	Email: `test@example.com`	Email cannot be edited (no typing/paste allowed); field appears read-only/disabled
UP-004	Update Name and Save persists changes	User is logged in on Profile page	1) Update Name 2) Click Save 3) Refresh page	Name: `Alex Johnson`	Success message displayed; Name is updated and persists after refresh
UP-005	Update Phone and Save persists changes	User is logged in on Profile	1) Update Phone 2) Click Save 3) Refresh page	Phone: `4155550123`	Success message displayed; Phone is updated and persists after refresh

Ask anything

3. Edge Cases

What edge cases and negative scenarios should be tested?

What edge cases and negative scenarios should be tested?

Below is a **QA-focused list of edge cases and negative scenarios** to test for the **User Profile Management** feature.
These scenarios are designed to expose validation gaps, data integrity issues, security weaknesses, and poor UX behavior.

---

1. Input & Data Edge Cases

Name field

- Name with **only spaces** (`" "`)
- Leading/trailing spaces (`" John Doe "`)
- Extremely long name (exceeds max length, if any)
- Single-character name (`"A"`)
- Name with **numbers** (`"John123"`)
- Name with **special characters** (`@#$%^&*`)
- Name with **Unicode / accented characters** (`José` , `李雷`)
- Name with **emoji** (`John 😊`)
- SQL/script-like input (`"<script>alert(1)</script>"`)

4. **Test Data Suggestions**

Suggest test data combinationsfor these test cases.

Suggest test data combinationsfor these test cases.

Here are **practical test data combinations** you can reuse across the profile test cases (view/update/save/validation). Since the requirement doesn't define exact validation rules, I'm covering common real-world patterns and marking a few as "depends on rules".

Reusable test data set

Data Set ID	Name value	Phone value	Email value (read-only)	Purpose / Where to use
TD-01	`Alex Johnson`	`4155550123`	`alex.johnson@example.com`	Baseline valid update (UP-004/005/006/011)
TD-02	`Alex Johnson`	`4155550123`	`alex.johnson@example.com`	Leading/trailing spaces (trim handling)
TD-03	`A`	`4155550123`	`alex.johnson@example.com`	Min-length edge (depends on rules)
TD-04	`ThisNameIsWayTooLong_` + 200 chars	`4155550123`	`alex.johnson@example.com`	Max-length boundary (validation/UI truncation)

5. Bug Report Improvement

Improve the clarity and completeness ofthis bug report.

Improve the clarity and completeness ofthis bug report.

Sure — please paste the **bug report details** you want improved.

You can share it in any rough form, for example:

- Current bug title
- Steps (even if incomplete)
- Expected vs actual result
- Screenshot description or notes
- Jira-style ticket text

Once you share it, I will:

- Rewrite the **bug title** to be clear and action-oriented
- Improve **steps to reproduce** (clear, minimal, reproducible)
- Clarify **expected vs actual results**
- Add **missing but important details** (environment, severity, scope, risks)
- Suggest **severity & priority** from a QA perspective

Paste the bug report when ready.

6. Severity and Priority Classification

Classify the severity and priority ofthis defectwith reasoning.

Classify the severity and priority of this defect with reasoning.

I can do that, but I need the **defect details** (otherwise I'd just be guessing).

Please paste any of the following (even short is fine):

- Defect summary/title
- What's happening (actual result)
- What should happen (expected result)
- Steps to reproduce
- Impact (who is affected / how often)
- Environment (prod vs staging, browser/device)

While you paste it, here's the rubric I'll use (so you can sanity-check the result)

Severity (impact to user/system)

- **S1 Critical / Blocker:** App crash, data loss/corruption, security issue, user cannot complete core flow
- **S2 High / Major:** Core feature broken but workaround exists; significant functional incorrectness
- **S3 Medium:** Partial feature issue, edge-case fa e, confusing UX but core flow still works
- **S4 Low / Minor:** Cosmetic/text issues, low impact, rare

7. Test Coverage Review

Identify any missingtest coverage based on thesetest cases.

Identify any missing test coverage based on these test cases.

Here are the **main missing coverage areas** based on the functional test cases you currently have (UP-001 to UP-016). I'm focusing on gaps that commonly slip defects into production.

Missing functional coverage

Server/API failure handling (big gap)

Your cases don't cover what happens when Save fails due to:

- **500/503 server error**
- **network timeout / offline**
- **401/403** (session expired / unauthorized)
- **400 validation error from server** (different from client-side validation)

Risk: UI may show success incorrectly, lose changes, or leave user stuck.

Authorization / security

No explicit tests for:

- User can only access/update **their own profile** (ID tampering / direct API call / URL parameter)

These prompts demonstrate how the Custom GPT can support **real QA workflows** using short and simple inputs.