PRACTICAL GUIDE · NOVEMBER 2026

Generative UI
A practical guide

Evaluate the use case and plan a controlled pilot

AUTHOR

Santiago Chiappa

Software Engineer at Kaizen Softworks

EXECUTIVE SUMMARY

Start with a task people struggle to complete, then test whether an adaptive interface helps them do it.

A useful pilot should show whether people get correct answers, how much effort they spend reaching them, and what it takes to support the feature.

Generative UI lets a model select or arrange interface elements in response to a request. Teams can approach this in three ways: selecting complete predefined views, combining approved components through structured instructions, or generating new interface code. This guide focuses on combining approved components.

Moving Generative UI into production brings challenges with consistency, reliability, security, performance, and debugging. Controls around the model help manage these risks, but they also affect flexibility and maintenance effort.

What this guide helps you *decide*

Use this guide to choose a first workflow, define how the feature should behave, and evaluate a pilot before expanding access. The decision criteria and exercises build on two Kaizen articles. They are proposed ways to apply the ideas, not findings from a production study.

READ	USE IT TO
The starting point · page 3	Understand the examples and identify an opportunity.
The pilot scope · page 5	Choose a task and set clear boundaries.
The behavior contract · page 6	Define what every useful answer must contain.
Production controls · page 7	Set limits and test that they work.
Evaluation · page 8	Compare results and investigate failures.
Release decision · page 9	Review the evidence and choose what to do next.

01 OPPORTUNITY

The *starting point*

How the interface works

With a component-based approach, the application turns a user's request into structured instructions for existing interface components. The model can choose how to present information, but only from the components the product team has approved. For a first pilot, keep that collection small: a reporting feature might start with a total, a comparison chart, and a detail table. Add components when users demonstrate a need for them.

A question worth testing

A fixed report could already answer the request below. Generative UI becomes worth exploring if users regularly need different comparisons and struggle to configure them. Before choosing a model, describe the improvement you want to test.

EXAMPLE REQUEST

"Compare support requests for the last two complete months and show which categories changed most."

IMPROVEMENT TO TEST

"A support lead can produce a correct category comparison *without asking an analyst* to prepare it."

This identifies the user, the task, and a result the team can observe. It doesn't assume that Generative UI is the right answer.

01 OPPORTUNITY

How a generated interface is *assembled*

01 THE REQUEST

> “Compare support requests for the last two complete months and show which categories changed most.”

02 STRUCTURED INSTRUCTIONS

```
{
  "components": [
    { "type": "Total", "period": "Sep–Oct" },
    { "type": "ComparisonChart",
      "groupBy": "category",
      "rank": "absolute" },
    { "type": "DetailTable", "rows": "top changes" }
  ]
}
```

03 APPROVED COMPONENTS

Total	APPROVED
Comparison chart	APPROVED
Detail table	APPROVED
~~Arbitrary interface code~~	~~REJECTED~~

The application validates each instruction against the catalog before anything renders.

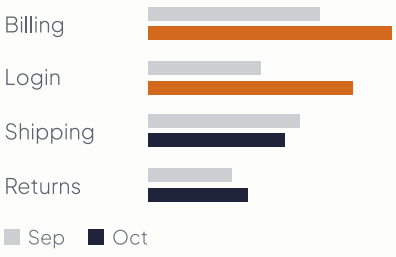
04 RENDERED INTERFACE

TOTAL · OCT

1,284

+94 vs. Sep

BY CATEGORY



TOP CHANGES

Login	+33
Billing	+26
Shipping	−6

WHEN ADAPTIVE PRESENTATION ADDS VALUE

Test the result by asking users to make a decision with it. If they repeatedly need the same comparison, a fixed report may be enough. If their follow-up questions vary, adaptive presentation may offer more value.

02 SCOPE

Define the *pilot scope*

Choose one group of users and one task they regularly perform. Start with something narrow enough that the team can recognize and investigate an incorrect answer.

DECISION	EVIDENCE TO COLLECT	WHAT IT MEANS FOR THE PILOT
User need	Examples of questions users already ask.	Include variations that occur in real work.
Current effort	Time spent preparing, checking, and revising answers.	Compare the complete task with today's method.
Correct answer	A person or query that can verify the result.	Choose tasks with answers you can independently check.
Consequences	What someone might do after seeing the result.	Start where mistakes can be caught before someone acts.
Repeat use	Which parts of the task stay the same.	Check whether a fixed report already meets the need.

Outside the first release

For the support-reporting example, start with read-only analysis of one approved dataset. Leave out actions such as changing ticket status, assigning staff, or messaging customers. Define which statuses count, how categories are assigned, and which timezone sets month boundaries.

Who owns each decision

Product defines the task and how success is measured. Design checks whether people can understand the experience. Engineering defines how the application accesses data. Also name who investigates incorrect results and who can pause the pilot.

CAPTURE THESE DECISIONS IN A SHORT STATEMENT AND REVIEW IT TOGETHER BEFORE IMPLEMENTATION

"For *[user group]*, support *[task]* using *[dataset]*. A successful result allows *[decision]*. The pilot excludes *[actions]*. We'll compare it with *[current method]*."

Specify the behavior of a *useful answer*

Before evaluating generated screens, agree on the rules every answer must follow: the **behavior contract**. A result may look good and still be wrong, such as a clear chart that shows the wrong months or hides missing records.

SITUATION	EXPECTED BEHAVIOR
The user requests the last two complete months.	Show the two calendar months used and state the timezone.
Category labels changed between periods.	Explain the mapping or ask the user to choose a comparable grouping.
One month has incomplete records.	Flag the missing data before presenting the comparison.
The previous value is zero.	Show the absolute change instead of a misleading percentage.
The user asks which categories changed most.	Explain whether the ranking uses absolute or percentage change.
The user asks a follow-up about one category.	Keep the same reporting period unless the user changes it.

Suggested rules for the example, not behaviors demonstrated in the Kaizen demos. Adapt them to your users' decisions.

Check the numbers, not just the layout

Verify calculations separately from presentation. With a known dataset, check which records were selected, the totals, and the ranking before reviewing the screen. One approach is to calculate totals in the application and let the model decide only how to present them.

Help users recover from mistakes

Users should be able to correct an assumption, change the reporting period, or return to a previous result. Retrying should not silently change the original question. A conventional report or export gives another way to finish the task.

04 PRODUCTION CONTROLS

Put production controls *around the model*

The application should enforce limits rather than relying on the model to follow instructions.

	CONTROL	WHAT TO IMPLEMENT
01	Stable interface	Keep navigation and critical actions in fixed positions.
02	Approved components	Allow the model to configure approved components, but reject arbitrary interface code.
03	Output checks	Validate data structures, sanitize content, and enforce permissions outside the model. Limit sensitive data shared with it.
04	Session records	Record the request, generated configuration, component and model versions, and outcome so the team can investigate failures.
05	Routing and alternatives	Use caching where appropriate, load content progressively, and provide another model or a conventional interface when generation fails.

Test the boundaries

Unsupported components	Request a component the application doesn't support. It should reject it and give a useful response.
Restricted data	Ask for records that belong to another team. Access should be refused.
Unexpected content	Check what happens when generated content contains an unexpected link.
Keyboard accessibility	Navigate results with a keyboard. Focus should move predictably, and charts shouldn't rely on color.
Failure investigation	Use a failed session to trace the question, data, and display. Track fallbacks separately from generation failures.

Each control introduces trade-offs. Restricting components reduces flexibility, and session records need access and deletion policies. Choose controls based on the workflow and its risks.

05 EVALUATION

Evaluate *the pilot*

Compare Generative UI with the method people use today. Give participants realistic tasks and measure whether they reach a correct, usable answer. A screen that appears quickly doesn't necessarily help someone finish faster.

Build a small set of real questions with answers verified against a fixed dataset. Run each more than once: every answer should follow the agreed rules, though the layout can vary.

TEST CASE	WHAT TO CHECK
A normal comparison	Correct periods, totals, rankings, and interpretation.
An ambiguous request	Whether the system asks for clarification or makes its assumption clear.
Missing or inconsistent records	Whether the result clearly explains the limitation.
A follow-up after several turns	Whether the response keeps the intended scope.
An unsupported request	Whether the user gets a clear way forward.
A slow or failed dependency	Whether the user can resume without losing context.

Measure the complete task

01

Correct task completion

Tasks users complete correctly without help.

02

Time to a verified answer

Including time spent checking and correcting.

03

Corrections or retries

How often users need to fix or repeat a request.

Cost per completed task

Include failed attempts, implementation, and ongoing support alongside model usage. Agree on an acceptable range before reviewing results.

Why failures happen

A misunderstood question, an incorrect calculation, and a confusing chart need different fixes. Keep failed examples, record the fix, and retest before expanding.

06 RELEASE DECISION

Make the *release decision*

Use the pilot results to decide whether to expand the feature, narrow its scope, or keep the existing interface. A successful demonstration gives you a reason to keep exploring. It doesn't prove the feature is ready for production.

QUESTION	EVIDENCE TO REVIEW
Does the feature help its intended users?	Task results compared with the current method.
Do answers follow the agreed rules?	Verified examples and an explanation of remaining errors.
Are access restrictions working?	Tests that attempt to access information outside each user's permissions.
Can users recover from failures?	An observed recovery task, including a failed dependency.
Can the team support the feature?	A named owner and a demonstrated investigation of a failed session.
Is the cost acceptable?	Estimated cost per successful task at the expected usage level.

If the system exposes restricted information or produces materially incorrect answers, pause expansion until the problem is addressed. For usability problems, consider whether a smaller audience or narrower task would make another test worthwhile. Record what evidence would change the decision and when the team will review it.

If the pilot provides a reason to invest further, expand *one area at a time*: another group of users, another question type, or another dataset.

Keep the original test cases. That way, the team can see whether a change breaks something that previously worked.

KAIZEN SOFTWORKS

Have a workflow you'd like to *test?*

We can help you define a first pilot, decide what to measure, and evaluate whether Generative UI is the right approach.

WWW.KZSOFTWORKS.COM

© 2026 KAIZEN SOFTWORKS