NAVA

Case study | Nava PBC

Form design approaches for downstream effects & nonlinear navigation

By Sawyer Hollenshead

Published September 4, 2018

This is part of a series of blog posts about Nava Public Benefit Corporation's partnership with the Centers for Medicare and Medicaid Services to design and build a new eligibility application for millions of Americans seeking health coverage on HealthCare.gov. Read more in this series.

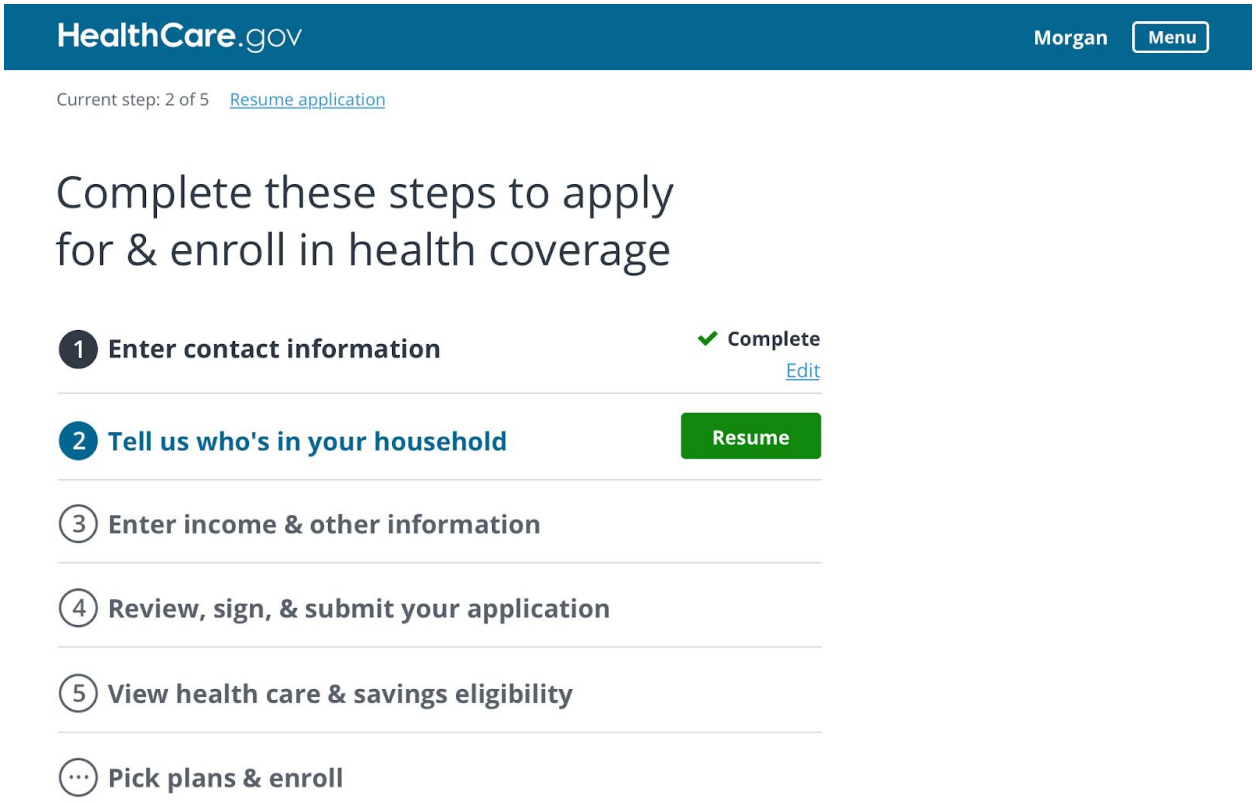
In a previous post, we discussed how we structured the complex eligibility form for HealthCare.gov, breaking it down across multiple pages. By breaking the form into digestible chunks, we were able to simplify the interface and allow applicants to focus on smaller bits of information at a time.

Due to the complex branching logic of the form we were building, where questions can vary widely depending on the applicant's answers, our team encountered our next challenge: How would we support navigating between different pages in a nonlinear way, to perform actions like changing an answer? And what happens when changing an answer causes questions or choices on subsequent pages to change? To solve this, we had to describe the challenges we were addressing, identify various scenarios within the form where these challenges might surface, and finally create design and technical solutions to address them.

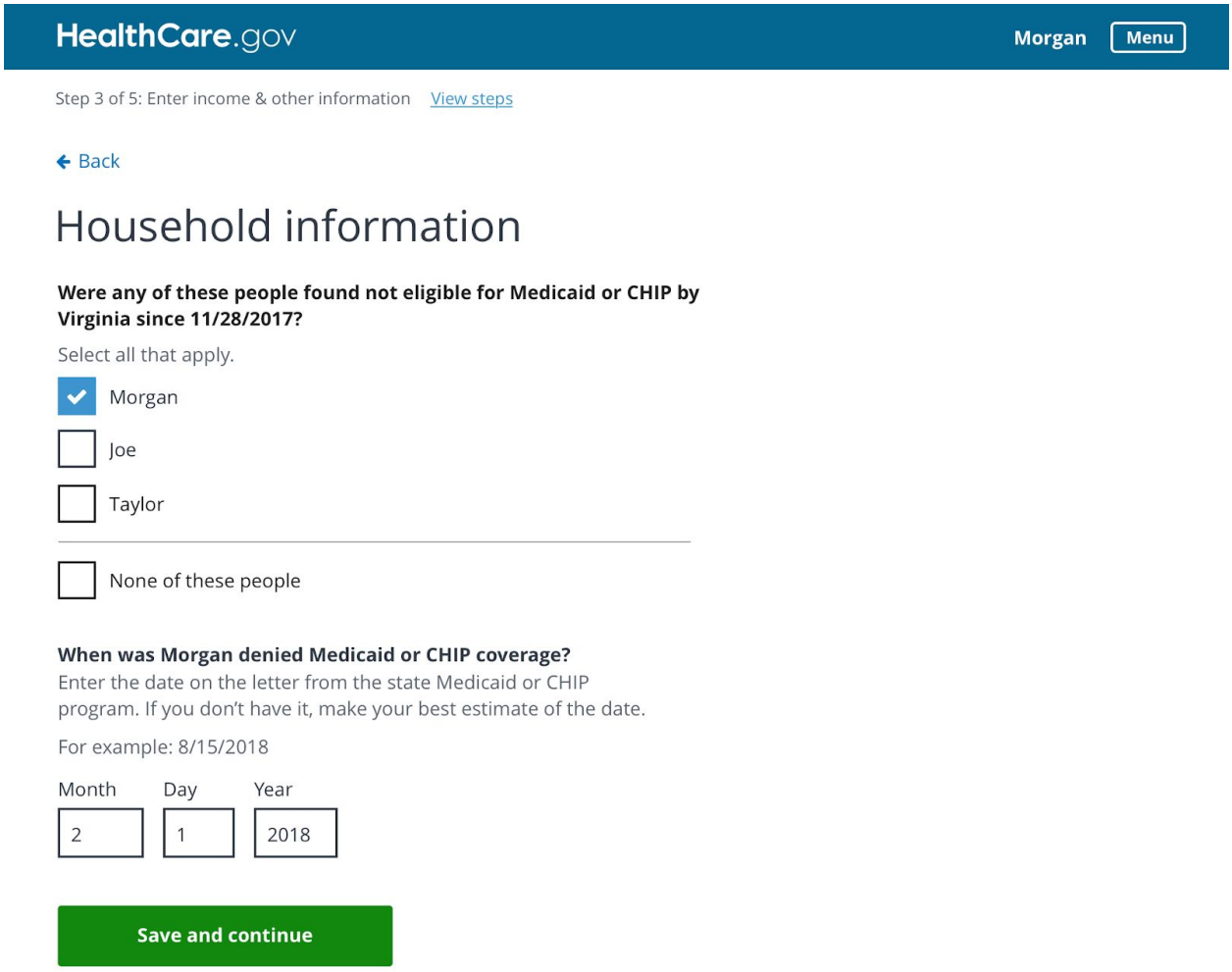
The approaches and ideas below are just that: ideas. They still require more research and validation, but we feel they are at a level of depth and polish worth sharing to the wider community to spur further discussion and iteration.

Components of the flow

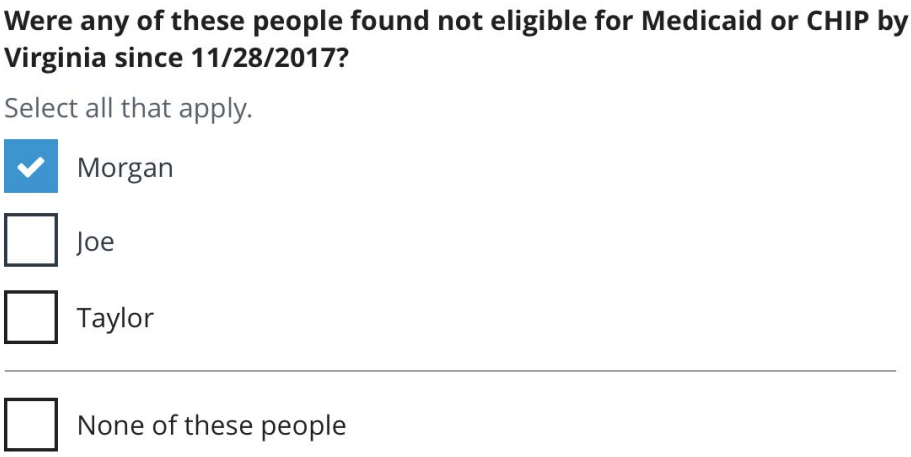
To understand the approaches we landed on, it'll be helpful to first understand the various components of the flow we proposed. At a high-level, the flow included:



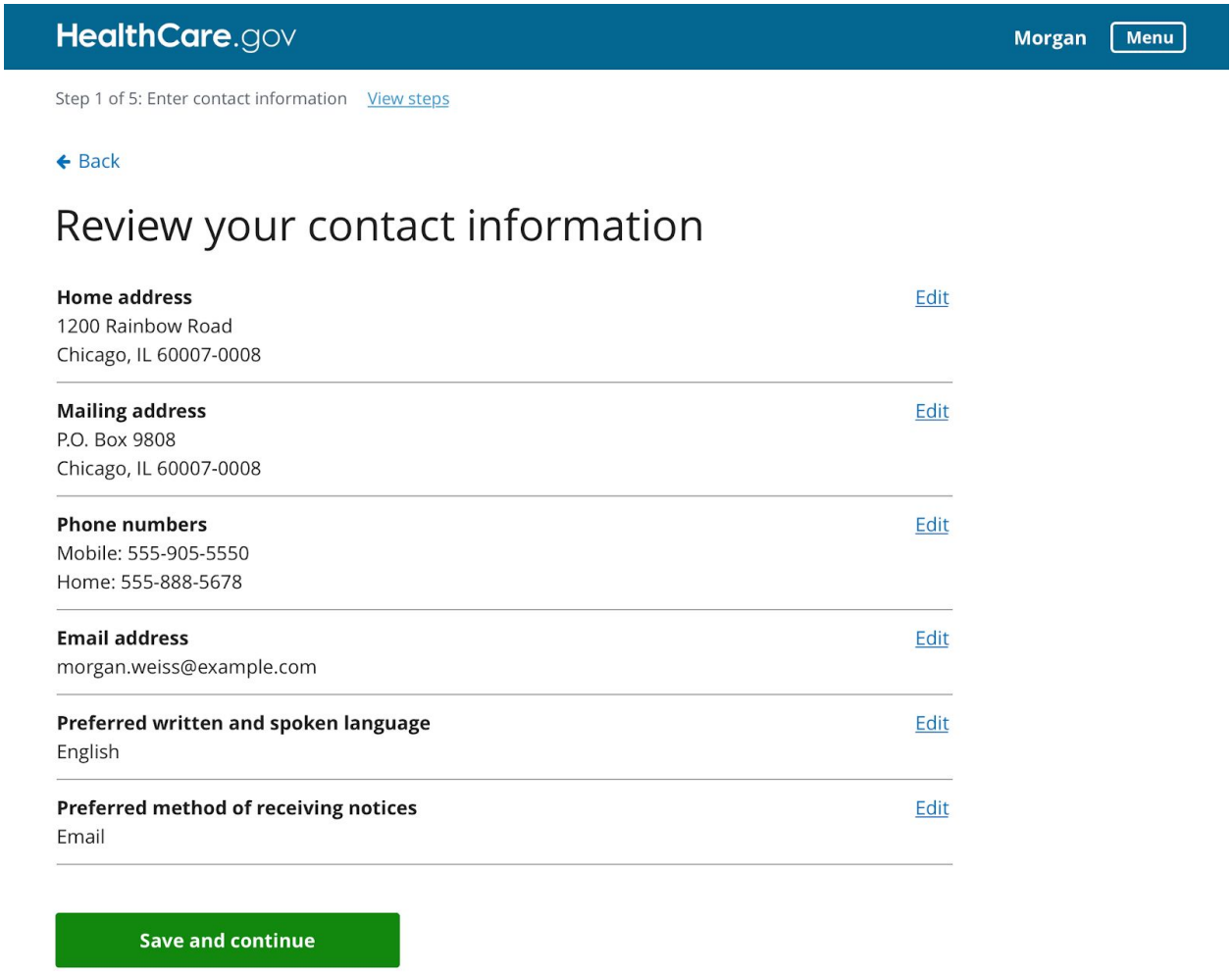
A **task list** served as an introduction screen, showing the various steps an applicant will go through to determine their eligibility. As an applicant completes each step, the task list updates accordingly. At any point while filling out the form, an applicant can click a "View steps" sub-navigation link to view their task list and navigate between completed and in-progress steps. The task list also serves as a landing page for applicants who return to the site to resume an in-progress application. This allows them to see where they last left off. This pattern was inspired by the Task List pattern proposed by GDS.



As mentioned earlier, the form's questions were displayed across multiple **questions pages**. Answers on one page informs what questions are asked on subsequent pages. A step in the task list is a group of questions pages.



A **dynamic choice list** is a type of checkbox or radio list whose choices are based on previously entered data. An example of this is a list of family members, which can vary in size, and is entered by the applicant in the first section of the form. Various questions in the form list family members in order to identify what circumstances apply to whom.



A **section review page** follows the last questions page of each step, and summarizes the information the applicant entered in that section, providing an opportunity to navigate back and change an answer or fix a mistake.

A framework for describing downstream effects

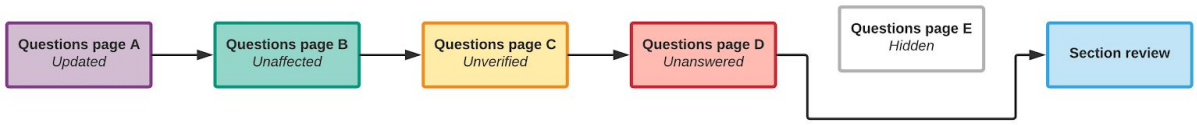
A downstream effect can occur when an applicant changes their answer, or adds/removes an item used in a dynamic choice list. This causes follow-up questions to change, either by making the questions hidden or visible in the flow, requiring a question to be re-answered, or changing the list of choices.

To describe the challenges and approaches associated with downstream effects, we found the following framework to be helpful, which describes the four different states of a follow-up question.

- 🛑**Unanswered**: This question has newly appeared downstream, wasn't previously answered, or it was previously optional but is now required. For example, adding a family member would result in questions about that family member's finances to appear downstream.
- ⚠**Unverified**: This question retained its previous answer, but its list of choices have changed. For example, adding a family member would result in this new family member appearing in follow-up questions that list family members as options.
- ✔️**Unaffected**: This question retained its previous answer and its options and validation rules were unaffected by the upstream change. For example, changing a family member's birthday doesn't affect the email you entered as a preferred contact method.
- 🙈**Hidden**: This question was previously visible but is no longer needed. For example, removing a family member would result in any downstream questions about that family member to be hidden as they are no longer relevant.

Approaches to downstream effects

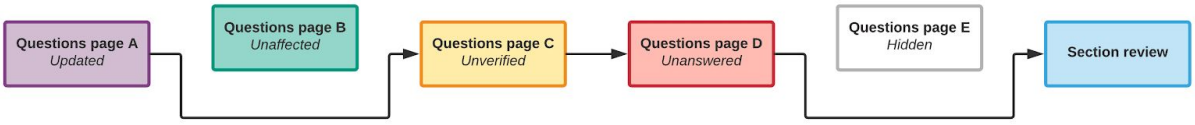
We proposed two approaches to downstream effects, each with their own tradeoffs.



A linear approach to downstream effects, bypassing only hidden question pages

The first was a **linear approach**, where after an applicant navigates back and changes their answer, they then proceed linearly through all follow-up questions regardless of whether those questions were affected by the change or not.

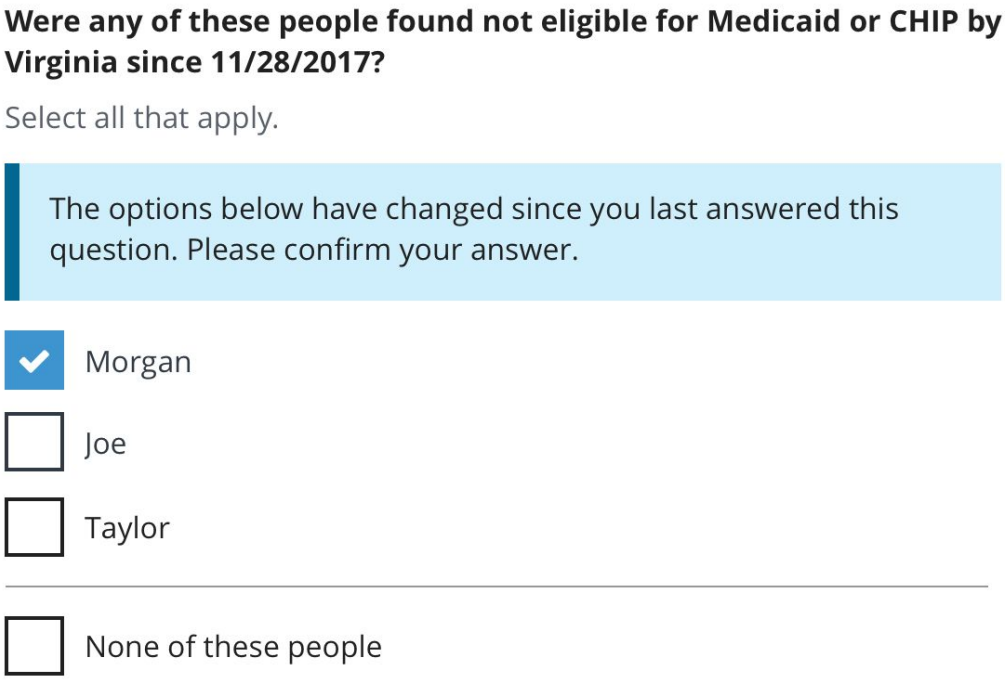
This approach was the most technically simple, and so was a good candidate for at least the MVP. A downside of this approach is that, if the form has many follow-up pages, it's possible the applicant will be forced to navigate back through many unaffected follow-up questions when they only needed to make a simple fix to a preceding question. We were okay with this tradeoff though. Our hypothesis was, even though it might be irritating to go through all the follow-up pages again, it wouldn't be a confusing experience since the applicant had gone through this flow already. This is in contrast to the second approach we explored.



A nonlinear approach to downstream effects, bypassing unaffected and hidden question pages

The second was a **nonlinear approach**, where after an applicant navigates back and changes their answer, they then proceed nonlinearly to the next page with an unanswered or unverified question, bypassing any pages with unaffected question. This approach has the benefit of not forcing the applicant to go through the entire flow again, and provides the shortest possible path to making a change and continuing the application. However, the technical implementation of such a nonlinear approach is non-trivial and further usability testing is necessary to validate whether this flow is intuitive for all applicants.

Another challenge we faced was letting an applicant know when a previously answered dynamic choice list has changed. For example, when a new family member is added, downstream lists now include that member. These questions would now be considered unverified, since some members may have been previously selected, but we can't be certain whether the new member should also be selected.



Example message above an unverified dynamic choice list, where the options have changed and "Morgan" was previously selected.

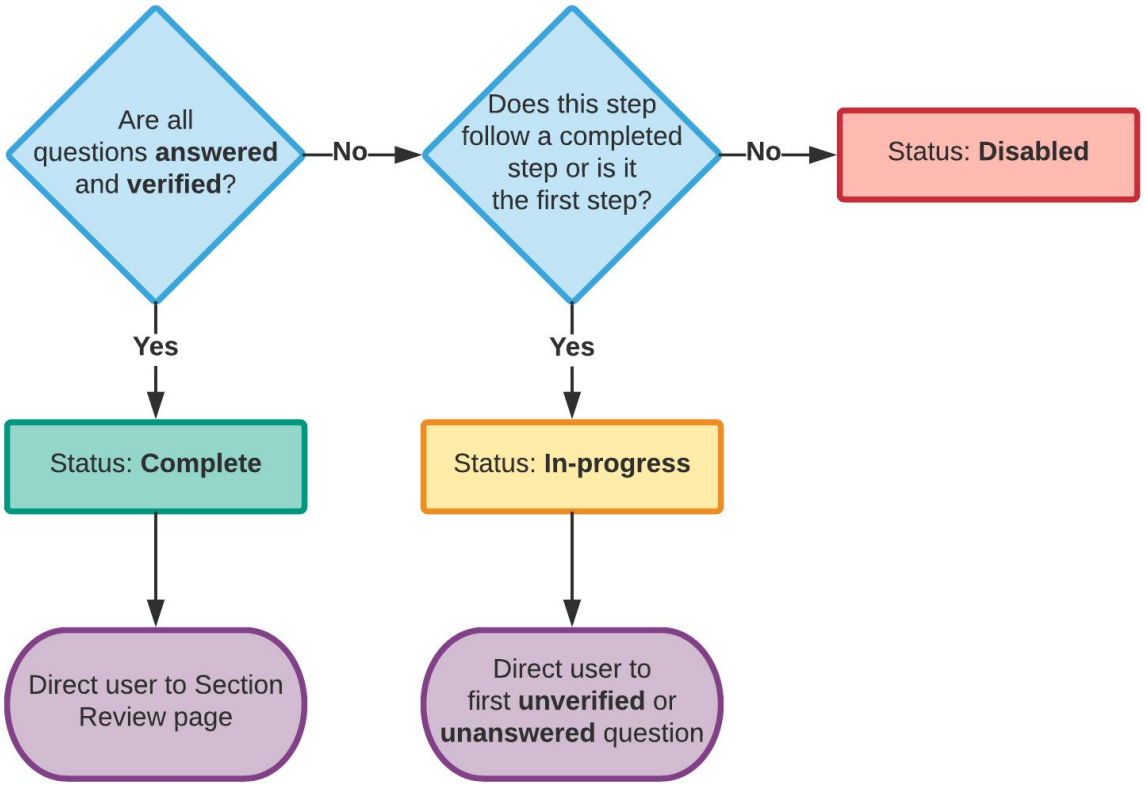
One approach to this is to clear the previous selections and prompt the applicant to re-select everyone who the question applies to. This would change the question from being unverified to unanswered. This felt too heavy handed, so we also explored how we might retain the previous answers while making it clear to the applicant that something about the list has changed. We did this by displaying a callout directly above the list, with a message mentioning that the options have changed since they last answered the question, and to confirm their answer. Further testing is needed to validate whether applicants notice this message.

Navigating in-progress applications

By including the task list and review pages as methods for navigating the form, and by supporting unique permalinks for each page of the form, there were situations where an applicant could jump from one section of the form to another unrelated section of the form. For example, they could change the URL in their browser or navigate to a previous step in the task list.

Since follow-up questions are influenced by preceding questions, we first had to identify which questions belonged within each step, and the status of each of those questions (using the question status framework described earlier). We then used that information to determine the status of each step:

- ✔️**Complete**: A step with 100% verified and answered questions
- ✏**In-progress**: A step directly following a completed step
- 🔒**Disabled**: Any step following an in-progress step



Decision tree for determining the status of a step and where applicants are directed when entering that step

One question we confronted was identifying where to take the applicant after they jump to a completed or in-progress step.

For in-progress steps, we proposed taking the applicant directly to the first unverified or unanswered question so that they can quickly continue where they left off.

For completed steps, rather than direct the applicant to the first question in the completed section, we proposed directing them to the section review page instead. From the section review page, they could view a summary of information they entered in that step, validate their information, and navigate to whichever question they want to change their answer to. This approach only works when the section review page displays a granular summary of the information entered. If it only displayed a high-level overview, it would be difficult for the applicant to identify how to change an answer to a question not surfaced in the summary.

How we think about "edge cases" in public services

For extreme cases that seem unlikely, it's often tempting to classify them as an edge case and sweep them under the rug. However, for public services depended upon by millions, improperly handling an "edge case" could mean the difference between someone being deemed eligible for health insurance or not. It's irresponsible not to consider all the different ways the service you are building could break down.

> It's irresponsible not to consider all the different ways the service you are building could break down.

Wherever possible, we attempted to put these safeguards in place so applicants could intuitively navigate through the experience without surprises or mistakes. If they do make a mistake, as we all occasionally do, we made sure to provide opportunities to fix them and continue where they left off. For complex forms like the one we were designing, it's often difficult to anticipate every way someone might try to navigate the form, and put safeguards in place on a case-by-case basis. Instead, by defining frameworks for thinking about these scenarios, designing reusable patterns for addressing them, and continually testing those patterns, we can feel more confident that the safeguards we put in place are indeed helping people reach the finish line.