Survey

Download our code

In this exercise, you will build a survey application.

It will ask the site visitor questions from a questionnaire, one per screen, moving to the next question when they submit.

Step Zero: Setup

Create a new virtual environment and activate it.

Install Flask and the Flask Debug Toolbar.

Make your project a Git repository, and add ***venv/*** and ***__pycache__*** to a new ***.gitignore*** file for your repository.

Step One: Surveys

We've provided a file, ***surveys.py***, which includes classes for ***Question*** (a single question on a survey, with a question, a list of choices, and whether or not that question should allow for comments) and ***Survey*** (a survey, which has a title, instructions, and a list of ***Question*** objects).

For the main part of this exercise, you'll only need to worry about the ***satisfaction_survey*** survey in that file. It does not include any questions that allow comments, so you can skip that for now. (Ignore the personality quiz and the ***surveys*** object; those come into play only in the Further Study).

Play with the ***satisfaction_survey*** in ipython to get a feel for how it works: it is an instance of the ***Survey*** class, and its ***.questions*** attribute is a list of instances of the ***Question*** class. You'll need to understand this structure well, so don't move on until you feel comfortable with it.

Step Two: The Start Page

For now, we'll keep track of the user's survey responses with a list in the outermost scope in your ***app.py***. To begin, initialize a variable called ***responses*** to be an empty list. As people answer questions, you should store their answers in this list.

For example, at the end of the survey, you should have in memory on the server a list that looks like this:

```
['Yes', 'No', 'Less than $10,000', 'Yes']
```

Next, let's handle our first request. When the user goes to the root route, render a page that shows the user the title of the survey, the instructions, and a button to start the survey. The button should serve as a link that directs the user to ***/questions/0*** (the next step will define that route).

Be sure to create a ***base.html*** and use template inheritance!

Step Three: The Question Page

Next, build a route that can handle questions — it should handle URLs like ***/questions/0*** (the first question), ***/questions/1***, and so on.

When the user arrives at one of these pages, it should show a form asking the current question, and listing the choices as radio buttons. Answering the question should fire off a POST request to ***/answer*** with the answer the user selected (we'll handle this route next).

Step Four: Handling Answers

When the user submits an answer, you should append this answer to your ***responses*** list, and then **redirect** them to the next question.

The Flask Debug Toolbar will be **very useful** in looking at the submitted form data.

Step Five: Thank The User

The customer satisfaction survey only has 4 questions, so once the user has submitted four responses, there is no new question to task. Once the user has answered all questions, rather than trying to send them to ***/questions/5***, redirect them to a simple "Thank You!" page.

Note: Don't Hardcode 5

It's possible that, in the future, this survey may include more than four questions, so don't hard-code 5 as the end. Do this in a way that can handle any number of questions.

Step Six: Protecting Questions

Right now, your survey app might be buggy. Once people know the URL structure, it's possible for them to manually go to ***/questions/3*** before they've answered questions 1 and 2. They could also try to go to a question id that doesn't exist, like ***/questions/7***.

To fix this problem, you can modify your view function for the question show page to look at the number in the URL and make sure it's correct. If not, you should redirect the user to the correct URL.

For example, if the user has answered one survey question, but then tries to manually enter ***/questions/4*** in the URL bar, you should redirect them to ***/questions/1***.

Once they've answered all of the questions, trying to access any of the question pages should redirect them to the thank you page.

Note: Clearing the list

Once this functionality is built, there's no way to reset the survey. Once it's complete, you can only see the start page and the thank you page.

By stopping and starting your server, you can complete the survey again (since every time the server starts, Flask reads the ***app.py*** and re-initializes ***responses*** to an empty list.

We'll fix this problem in Step Eight.

Step Seven: Flash Messages

Using ***flash***, if the user does try to tinker with the URL and visit questions out of order, flash a message telling them they're trying to access an invalid question as part of your redirect.

Step Eight: Using the Session

Storing answers in a list on the server has some problems. The biggest one is that there's only one list – if two people try to answer the survey at the same time, they'll be stepping on each others' toes!

A better approach is to use the session to store response information, so that's what we'd like to do next. **If you haven't learned about the session yet, move on to step 9 and come back to this later.**

To begin, modify your start page so that clicking on the button fires off a POST request to a new route that will set ***session["responses"]*** to an empty list. The view function should then redirect you to the start of the survey. (This will also take care of the issue mentioned at the end of Step Six.) Then, modify your code so that you reference the session when you're trying to edit the list of responses.

Note: Why is this a POST request?

Why are we changing "Start Survey" button from sending a GET request to sending a POST request? Feel free to ask for some support on this question.

Note: Appending to a list in the session

When it comes time to modify the session, watch out. Normally, you can append to a list like this:

```
fruits.append("cherry")
```

However, for a list stored in the session, you'll need to rebind the name in the session, like so:

```
fruits = session['fruits']
fruits.append("cherry")
session['fruits'] = fruits
```

Step Nine: Celebrate!

Good work!

Further Study

View the Further Study for this

Solution

See our solution