Individual Assignment 3

COMS W4170: User Interface Design (Fall 2024)

DUE: Friday, Oct. 18, 2024 at 11:59pm ET

(Last updated Oct. 4, 2024. Changes since first release highlighted in yellow)

Part 1: The Design Process	12 pts
Part 2: Journey Mapping	14 pts
Part 3: Storyboarding	16 pts
Part 4: Smoke-and-Mirrors Prototyping	16 pts
Part 5: Flask To-Do List App	28 pts
Part 6: Flask Pages and API	14 pts
TOTAL:	100 pts

SUBMISSION INSTRUCTIONS:

Please submit all of your files electronically via Gradescope.

For Parts 1–4, we ask you to submit a single PDF named **YourName_IA3.pdf**.

For Parts 5–6, we ask that you submit your files as a .zip file named **YourName_code_IA3.zip**. You should include a file named `README.txt` which includes the URL to your demo video (described below) and any special instructions for running your code.

For Parts 5–6, we also ask that you submit a single short (3 minutes max) YouTube or Panopto video of you demonstrating specific functionality in your app. You should use screen recording software such as QuickTime (free and installed by default on Mac) or Panopto (Columbia's official solution and free to university affiliates) to capture the video. If you must, you can also capture the video using your phone as long as the relevant information is legible in the video.

However you create the video, you must ensure that that it is viewable to others who have only the URL. On YouTube, you should change the video's sharing settings to make the video "unlisted" rather than private (making the video "unlisted" allows it to be viewable only to people who have the video's URL). On Panopto, you should configure the video's sharing settings to be viewable by anyone with the URL. If you host the video on your own server, it should have global read access at its URL. If in doubt, paste the URL into a private browser window and test it to ensure that it can be opened by anyone with the URL.

You should narrate the video with your own voice, walking through your functionality by demonstrating the following steps in order:

1. **Part 5: Flask To-Do List App**
   - Start the Flask server
   - Show the to-do list app open at the correct URL
   - Toggle the complete/incomplete status of two to-do items, then refresh the page to show that it is saved
   - Add a teammate to the dropdown menu and a new to-do item, then refresh the page to show that they are saved
   - Click the "Clear Completed" button, then refresh the page to show that it is saved
   - Click the "Reset" button, then refresh the page to show that the empty to-do list is saved
   - Restart the Flask server, then reload the to-do list app again at its URL
2. **Part 6: Flask Pages and API**
   - Using the navbar, click between the Announcements page and the To-Do List page
   - Navigate to `[Your URL]/api/announcements`
   - Navigate to `[Your URL]/api/announcements/0`
   - Navigate to `[Your URL]/api/todo`
   - Navigate to the URL that you added in Part 6, #5.

NOTE: We are randomly cross-checking videos with code to make sure that the functionality matches. As with any dishonest submission, submitting a video that does not match your code constitutes academic dishonesty.

If you have trouble submitting any of your files, please post a question to Ed Discussion and/or come to our office hours.

PART 1: THE DESIGN PROCESS

(12 points)

In this first Part, your role is to consider the following scenario as a design consultant to the founder of a startup company. You should analyze the scenario through the lens of the five stages of the design process as we have illustrated in class: Identify Problem, Understand Users, Design, Prototype, and Evaluate. The startup founder has not taken this course and is unfamiliar with the design process. They may have skipped some of the stages without realizing the consequences. As their consultant (or eventual replacement as the company's CEO!), your role is to help them find the flaws with their strategy and chart a more effective path forward. The scenario is as follows.

A startup founder believes they have identified a very promising idea for a new product: a smartwatch app for restaurant servers that can notify them when tables are requesting service and remind them to check up on tables every so often. Such an app could make life easier for restaurant servers since they would not have to keep looking around the restaurant unnecessarily to check for things like guests running out of water. Instead, each table would have a simple "call server" button. Such an app could also make life easier for restaurant managers since it would make it easier for them to train servers and keep guests happy.

The founder just asked their team to begin implementing the app, which would take at least four months. The founder wants to gain traction by having many restaurants adopt the app, so they want it to be developed as soon as possible. The founder figures that having a fully working app will enable them to demonstrate it to restaurant managers so that restaurant managers can sign on and adopt it.

The founder does not have any restaurant managers as paying customers yet, but they firmly believe that their idea has potential. They have already asked several restaurant managers if they would like such a product, and those managers said they would. The founder does not have any personal experience working at a restaurant.

1. Which stage of the design process is the startup founder currently acting in, given their actions and desires so far? Provide the reasoning for your answer.
   (1 pt.)

2. Which stage(s) of the design process has the startup founder skipped? Provide the reasoning for your answer.
   (2 pts.)

3. For *each* stage of the design process that the founder skipped, what is a potential consequence that the founder may face because they skipped it? Give a concrete example of something that might happen as a result. Do so for *each* stage that was skipped.
   (3 pts.)

4. For *__each__* stage of the design process that the founder skipped, what should the founder do to perform that stage properly?
   (3 pts.)

5. What should the output of *__each__* of those stages be?
   (3 pts.)

PART 2: JOURNEY MAPPING

(14 points)

In this Part, you will practice journey mapping as we did in class. For the exercise, we will create a journey map for an everyday scenario: booking and attending a doctor or dentist appointment, from the patient's perspective. You can choose whether it is a doctor or dentist appointment.

1. Create a journey map for this scenario in the simpler format we used during class (not the detailed spreadsheet version). Your journey map should include axes with labels, a neutral line, and at least four moments. Ensure your journey map includes a label summarizing each moment.
   (6 pts.)

2. Ensure your journey map includes an emoji or Bitmoji to show the person's feelings at that time. If you wish to use Bitmoji, we recommend using the Bitmoji browser extension on Google Chrome, which makes it easy to generate Bitmoji and save them to your computer.
   (2 pts.)

3. Ensure your journey map includes clear, visual labels for the highest point, lowest point, and major transitions on your journey map. Certain emojis, such as the glowing star emoji (🌟), cross mark emoji (❌), and checkered flag emoji (🏁), can work for this purpose.
   (2 pts.)

4. Let us now go back to our figure for the Design Process that we covered in class, the same figure that we referenced in Part 1. Describe how the Problem Statement point within the design process (as illustrated by a point in the figure) would incorporate the results from this journey map. Exactly how would the results of this journey map be used to form a problem statement?
   (2 pts.)

5. Describe how the Design stage of the design process would incorporate the results from this journey map. Give a concrete example of how the Design stage should play out given the journey map you created.
   (2 pts.)

PART 3: STORYBOARDING

(16 points)

In this Part, you will practice brainstorming diverse design solutions and creating a storyboard. We will brainstorm design solutions for the scenario from Part 2: booking and attending a doctor or dentist appointment.

1. Describe ***three very different*** design solutions ("hills") for improving the process of booking and attending a doctor or dentist appointment. Make your design solutions as different from each other as possible; they should explore very different "hills." If one involves an app, for example, try another that does not involve an app at all. You can use your journey map from Part 2 and consider the peak-end rule to brainstorm different ideas.
   (6 pts.)

We will now start creating a storyboard for what you think is the most promising design solution.

2. First, write the captions. The storyboard should have between three and seven panels, as shown in class, so for now, you should write between three and seven captions. The captions should tell the story: they should convey the setting, the problem/user need(s), and how the design solution would satisfy their needs, with an appropriate climax and denouement (resolution). They should be written in a narrative rather than descriptive format; for example:

   BAD: "A diagram of a person noticing a notification on their phone."
   GOOD: "Carla notices a notification on her phone."

   (3 pts.)

3. Now, create illustrations for each panel. As shown in class, it is fine to use stick figures or even collections of Post-It notes for your drawings. Choose the perspective (e.g., long shot vs. over-the-shoulder shot) that makes the most sense for each panel.
   (5 pts.)

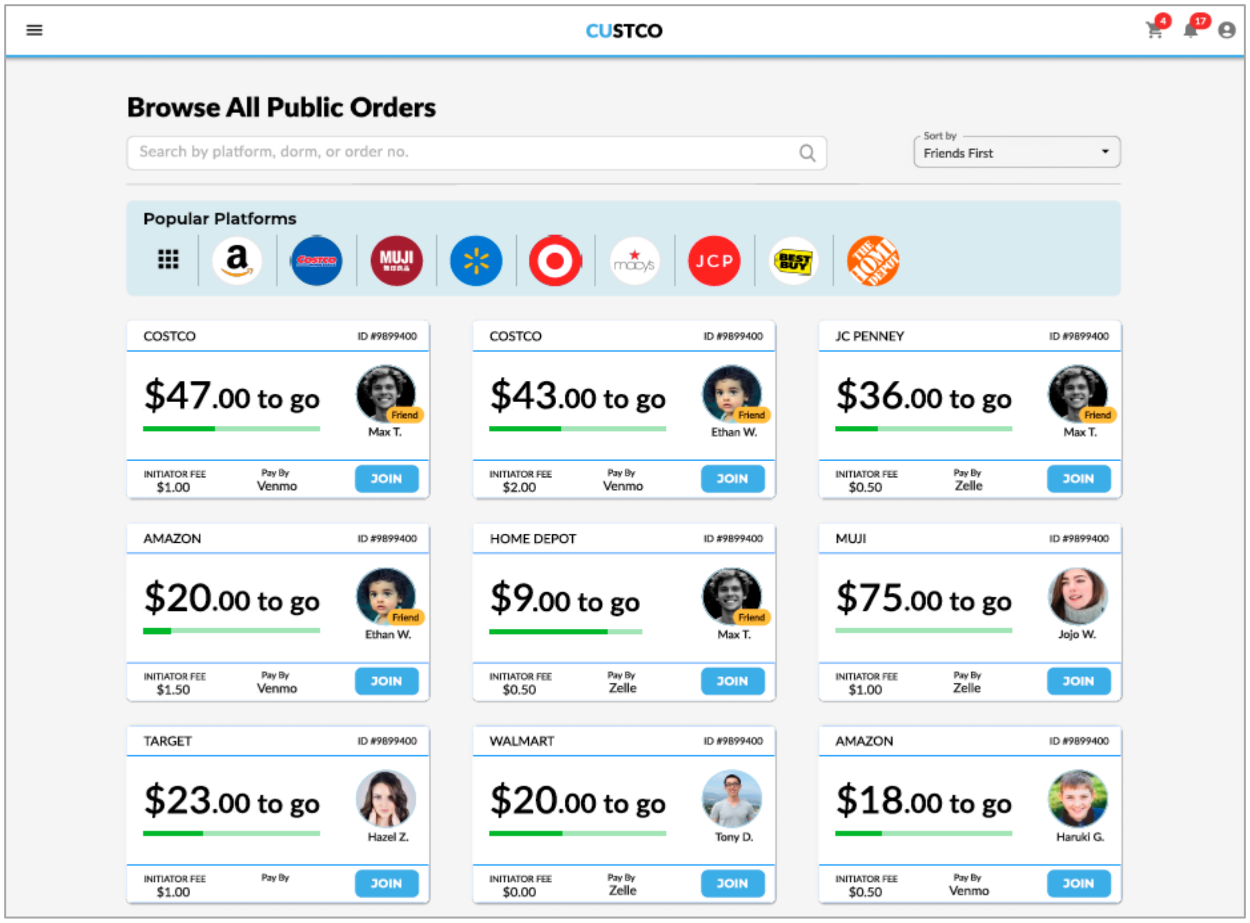
4. Ensure that the storyboard's panel illustrations highlight the actions (verbs) and feelings described in the captions.
   (2 pts.)

PART 4: SMOKE-AND-MIRRORS PROTOTYPING

(16 points)

In this Part, you will practice designing smoke-and-mirrors prototypes. We will explore two case studies based on student projects from past iterations of the course.

Our first case study is CUstco, a past project that we reviewed in Class 1. The concept for CUstco is an app that lets Columbia students pool orders to overcome minimum free shipping thresholds. Students can pool orders from Amazon, Costco, Home Depot, and other online retailers. The app is meant to give Columbia students an opportunity to easily find others to buy things together so that they can all save some money by removing the constraint of minimum free shipping thresholds. Users can start a group order on the platform or join others' orders. The initiator pays for the order, delivers the items to others joining (communicating with them if necessary to arrange exactly how to meet up later), and collects a small additional fee from others joining. Below are screenshots from the final app.



COSTCO ID #9899400
$47.00 to go
Friend
Max T.
INITIATOR FEE $1.00 Pay By Venmo JOIN

COSTCO ID #9899400
$43.00 to go
Friend
Ethan W.
INITIATOR FEE $2.00 Pay By Venmo JOIN

JC PENNEY ID #9899400
$36.00 to go
Friend
Max T.
INITIATOR FEE $0.50 Pay By Zelle JOIN

AMAZON ID #9899400
$20.00 to go
Friend
Ethan W.
INITIATOR FEE $1.50 Pay By Venmo JOIN

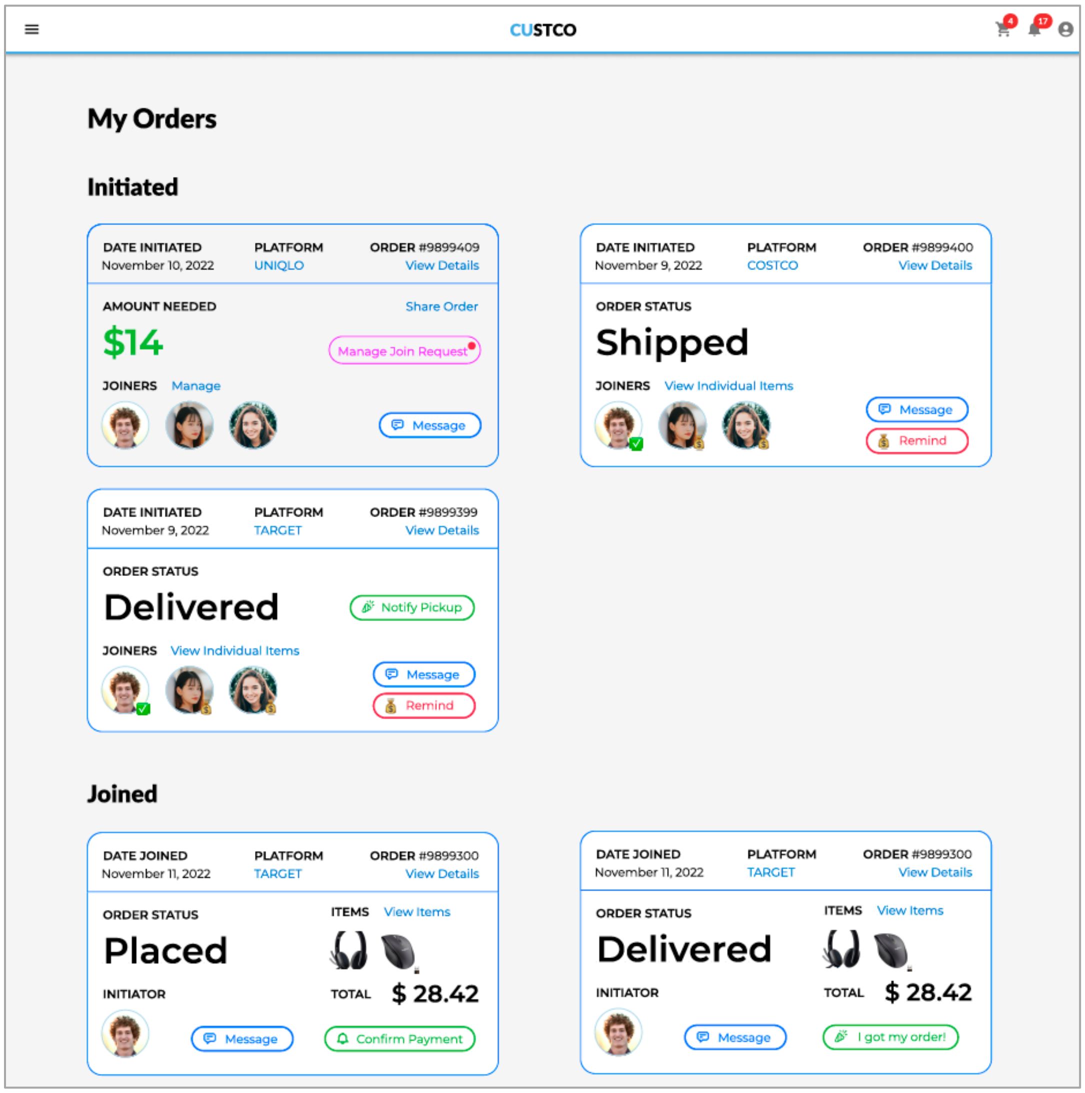
HOME DEPOT ID #9899400
$9.00 to go
Friend
Max T.
INITIATOR FEE $0.50 Pay By Zelle JOIN

MUJI ID #9899400
$75.00 to go
Jojo W.
INITIATOR FEE $1.00 Pay By Zelle JOIN

TARGET ID #9899400
$23.00 to go
Hazel Z.
INITIATOR FEE $1.00 Pay By JOIN

WALMART ID #9899400
$20.00 to go
Tony D.
INITIATOR FEE $0.00 Pay By Zelle JOIN

AMAZON ID #9899400
$18.00 to go
Haruki G.
INITIATOR FEE $0.50 Pay By Venmo JOIN



My Orders

Initiated

DATE INITIATED November 10, 2022 | PLATFORM UNIQLO | ORDER #9899409 View Details

AMOUNT NEEDED $14 | Share Order | Manage Join Request

JOINERS Manage | Message

DATE INITIATED November 9, 2022 | PLATFORM COSTCO | ORDER #9899400 View Details

ORDER STATUS Shipped

JOINERS View Individual Items | Message | Remind

DATE INITIATED November 9, 2022 | PLATFORM TARGET | ORDER #9899399 View Details

ORDER STATUS Delivered | Notify Pickup

JOINERS View Individual Items | Message | Remind

Joined

DATE JOINED November 11, 2022 | PLATFORM TARGET | ORDER #9899300 View Details

ORDER STATUS Placed | ITEMS View Items

INITIATOR | TOTAL $ 28.42

Message | Confirm Payment

DATE JOINED November 11, 2022 | PLATFORM TARGET | ORDER #9899300 View Details

ORDER STATUS Delivered | ITEMS View Items

INITIATOR | TOTAL $ 28.42

Message | I got my order!

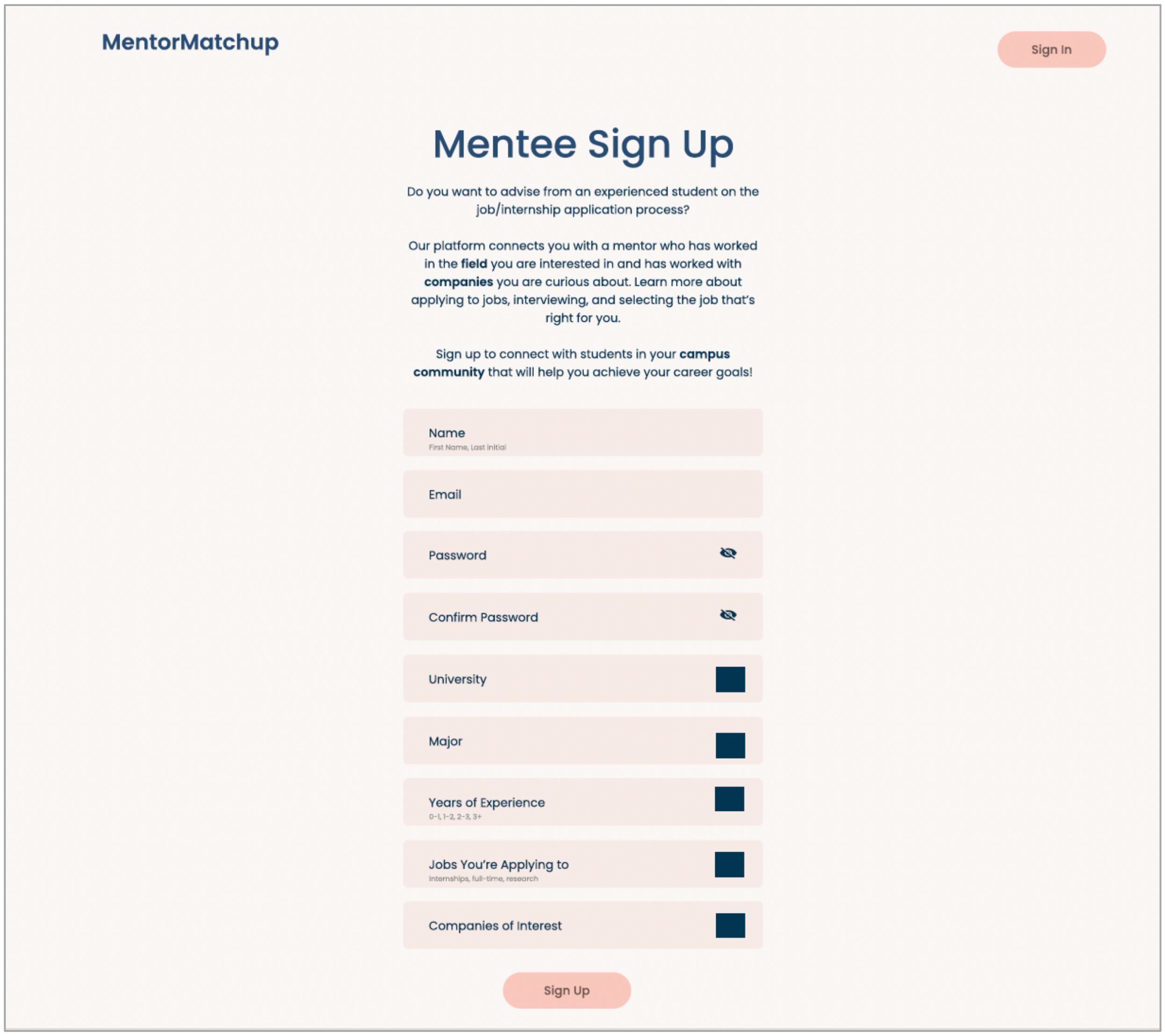
1. What should the design hypothesis be for CUstco?
   (1 pt.)

2. Name two important metrics for success for CUstco (i.e., ways of gauging whether the concept is effective—is a good "hill").
   (2 pts.)

3. Describe an effective smoke-and-mirrors prototype for the CUstco concept. If you were to "run" the prototype over a week, evaluating the concept with study participants, what would you be doing over that week to give participants the desired experience and collect the data you need to measure your metrics for success?
   (5 pts.)

Our second case study is Mentor Matchup, a project by Amelia Wissink, Kaya Dorogi, Kaitlyn Hazzard, and Grace Perrin. The concept for Mentor Matchup is a website that helps Columbia students feel more supported in their job and internship search process. That process is often a very stressful and isolating time for students.

Students join Mentor Matchup by entering a set of profile information, shown in the screenshot below. Mentor Matchup then uses this information to match students with each other, forming mentor–mentee pairs according to a specific matching algorithm. Students can then contact their matches and chat with them in the app, with helpful prompts appearing at the bottom to stimulate conversation. Below are some screenshots from the final app.


MentorMatchup

Sign In

Mentee Sign Up

Do you want to advise from an experienced student on the job/internship application process?

Our platform connects you with a mentor who has worked in the **field** you are interested in and has worked with **companies** you are curious about. Learn more about applying to jobs, interviewing, and selecting the job that's right for you.

Sign up to connect with students in your **campus community** that will help you achieve your career goals!

Name
First Name, Last Initial

Email

Password

Confirm Password

University

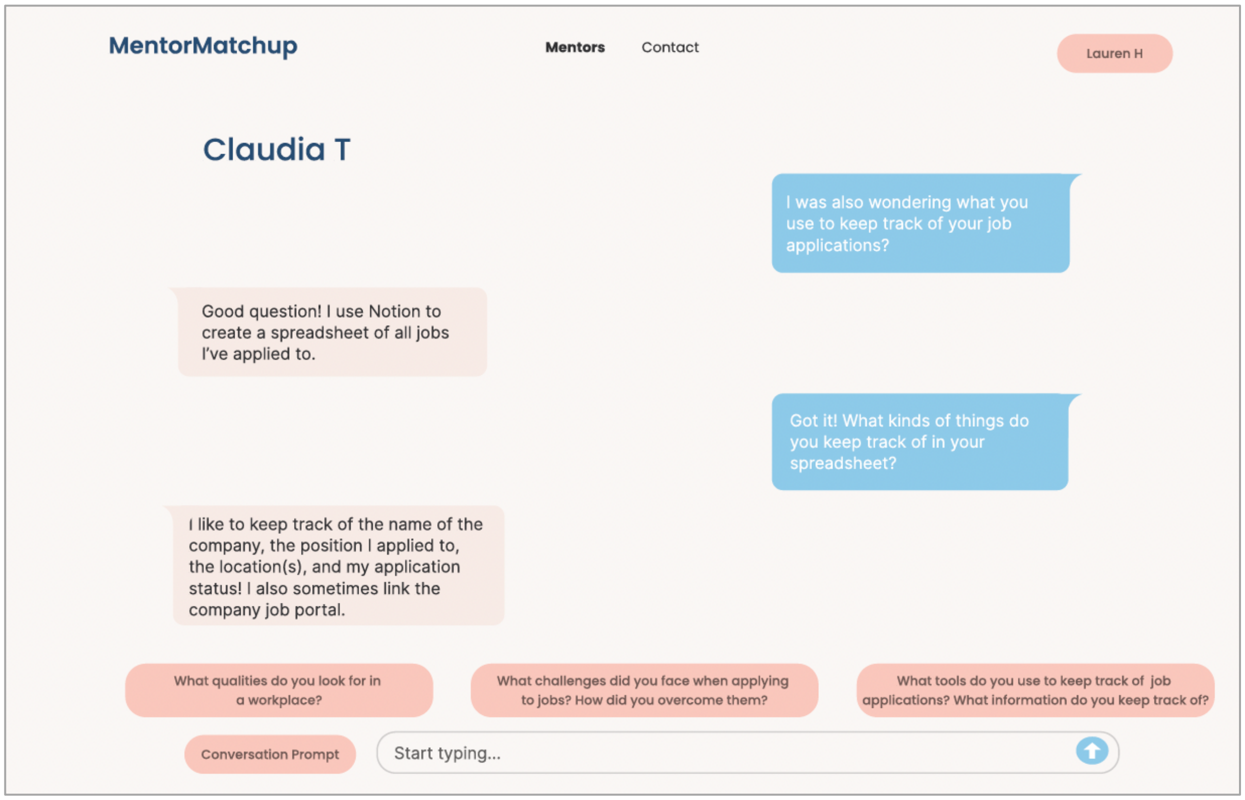
Major

Years of Experience
0-1, 1-2, 2-3, 3+

Jobs You're Applying to
Internships, full-time, research

Companies of Interest

Sign Up



Claudia T

I was also wondering what you use to keep track of your job applications?

Good question! I use Notion to create a spreadsheet of all jobs I've applied to.

Got it! What kinds of things do you keep track of in your spreadsheet?

I like to keep track of the name of the company, the position I applied to, the location(s), and my application status! I also sometimes link the company job portal.

What qualities do you look for in a workplace?
What challenges did you face when applying to jobs? How did you overcome them?
What tools do you use to keep track of job applications? What information do you keep track of?
Conversation Prompt
Start typing...

4. What should the design hypothesis be for Mentor Matchup?
   (1 pt.)

5. Name two important metrics for success for Mentor Matchup (i.e., ways of gauging whether the concept is effective—is a good "hill").
   (2 pts.)

6. Describe an effective smoke-and-mirrors prototype for the Mentor Matchup concept. If you were to "run" the prototype over a week, evaluating the concept with study participants, what would you be doing over that week to give participants the desired experience and collect the data you need to measure your metrics for success?
   (5 pts.)

PART 5: FLASK TO-DO LIST APP

(28 points)

In the Part, we will add a proper back-end—namely, Flask—to the To-Do List app from Individual Assignment 2. We will also add new pages to the app to take advantage of the new backend. As you port pieces of the to-do list app over to this new assignment, you are free to copy→paste sections of your code from prior assignments.

1. Our first step will be to integrate our code from Individual Assignment 2 into a starter Flask app template. The template will be the final Flask Announcements code from class, which is a great foundation for starting any Flask project. Download the starter code for this assignment from CourseWorks, and edit the root route ('/') of `server.py` to render the HTML to-do list layout for the to-do list from Individual Assignment 2. The CSS styling and JavaScript functionality do not need to be in place for this first step.
   (2 pts.)

2. Now, integrate your CSS styles from Individual Assignment 2 so your to-do list renders correctly, as you had designed it before.
   (1 pt.)

3. Integrate your JavaScript code from Individual Assignment 2 so your to-do list is fully functional as it was before.
   (1 pt.)

4. We will now work on migrating the to-do list data store from local storage (as it was in Individual Assignment 2) to our Flask database. First, remove all lines of code that load or save from local storage. You may want to leave comments in those parts of the code so you can remember where they were later.
   (1 pt.)

5. Next, add some hardcoded initial data to your to-do list so that, when the page is first loaded, the to-do list will already be populated with at least two to-do items assigned to two different people. At least one of the initial set of to-do items should be marked as incomplete, and at least one of them should be marked as completed already.

   Your hardcoded data should appear as a JSON dictionary in `server.py`, similar to how we created an initial JSON dictionary for the announcements page in class (lines 7–18 of `server.py`). Do not remove or overwrite the `data` variable; we will use that later as we combine the To-Do List page and the Announcements page together into a joint website. Instead, create a variable for storing the dictionary. The dictionary may include the initial set of teammate names (for the to-do list's dropdown menu) and the initial set of to-do items. Give the variable an intuitive name, such as `todo_list_data`.

The data here will not actually be rendered yet, but we will now be able to see your JSON format directly in your code.

For clarity, you should rename the existing `data` variable in `server.py` (i.e., the variable whose name is "`data`" and which holds the Announcements page data) to `announcements_data`.
(4 pts.)

6. It is now time to render the initial data that you created. If it is not present already, be sure that your to-do list HTML template includes code similar to the following, adapted from lines 12–14 of `announcements.html`:

```
<script>
    var data = {{todo_list_data|tojson}}
</script>
```

Here, the variable `todo_list_data` is the Python (Flask) data store from `server.py`, which you initialized with the hardcoded JSON dictionary. The variable `data` is a JavaScript variable that is copied from the Python variable and that you can access in your to-do list's JavaScript file. The variable is copied on page load, as line 13 of `announcements.html` does for the announcements page data. After that, it will not stay in sync with the Python variable by itself. Soon, in #7 below, you will keep it in sync every time you save new data by having your successful Ajax call return the updated copy of the Python variable, as lines 41–42 of `announcements.js` do for the announcements page.

For now, though, all you need to do is ensure the to-do list page renders the initial set of items from `server.py` (and only those items) on page load. That means you need to revise your JavaScript code to use the data copied from Python. The items should be rendered and sorted as before from Individual Assignment 2. Completed items should be rendered as completed.

Note that once you are done, it should still be possible for the user to add new to-do items to the rendered page as normal. Those new items will not be saved yet, of course, and will thus disappear if the page is refreshed. The new items should still be rendered on the page when they are added, though.
(3 pts.)

7. Now, add a Flask route to `server.py` for saving a teammate that is added to the dropdown menu. The Flask route will mimic the approach used by the existing `/post_announcement` route. Give the route an intuitive name such as `/add_teammate`. If there is not one already, add a helper function called `saveTeammate(…)` (or something very similar) to your JavaScript file for adding a

new teammate. Within that function, make an Ajax call to your server's new `/add_teammate` route to update the data store accordingly. You can mimic the approach used by the `saveAnnouncement(…)` function from the `announcements.js` file.

This saving functionality should now work, meaning any teammates added should persist if the user reloads the page.
(4 pts.)

8. Create another Flask route in `server.py` for saving a to-do item. You can mimic the approach you used in #7, but with a to-do item this time. This saving functionality should now work, meaning any to-do list items added should persist if the user reloads the page.
(4 pts.)

9. Similarly, create Flask route(s) that save the status of items marked as complete or incomplete, whenever the user clicks their corresponding checkbox. This saving functionality should now work, and items' complete/incomplete status should persist if the user reloads the page.
(4 pts.)

10. Add the necessary code for making the "Clear Completed" button's functionality persistent in the backend.
(2 pts.)

11. Last, add the necessary code for making the "Reset" button's functionality persistent in the backend. When the user resets the to-do list, the backend data store should be cleared and saved as empty.
(2 pts.)

PART 6: FLASK PAGES AND API

(14 points)

In this last Part, we will add links to connect the Announcements page and the To-Do List page to create a website that can serve as a team's central communication hub. We will also create our own API from those pages for others to query our data.

1. Add a navbar to the top of the Announcements page and the To-Do List page. The navbar should have links to both pages, making it easy to navigate between the pages. You can style the navbar however you like as long as it is centered and rendered above the rest of each page. The navbar should appear the same on both pages.
   (4 pts.)

2. We will now start creating our own API endpoints to allow others to query our Announcements page data. Specifically, we would like others to be able to navigate to the `http://127.0.0.1:5000/api/announcements` endpoint (replace `http://127.0.0.1:5000` with your server's actual URL) to retrieve a JSON string with all of the announcements data.

   Follow the example from the included `api_example.py` file to add a new route to `server.py` that returns the JSON string appropriately. You can try out the `api_example.py` server by running "`python api_example.py`" on the command line, then navigating to `http://127.0.0.1:5000/returnjson` in your browser (again, replace `http://127.0.0.1:5000` with your server's actual URL). This example's source code is very short and should be self-explanatory, but if you would like more information, we recommend the following Geeks4Geeks article, where we copied the example from:
   https://www.geeksforgeeks.org/how-to-return-a-json-response-from-a-flask-api/
   (3 pts.)

3. Now, create an API endpoint at `[Your URL]/api/announcements/<index>` to return only the JSON for the ith announcement, where i = `index`. The indices are zero-based. That is, `[Your URL]/api/announcements/0` should return the very first announcement that was posted, while `[Your URL]/api/announcements/1` should return the second announcement that was posted. You can define the Flask route similarly to how we defined the `/hello/<name>` route in our in-class Flask exercise.
   (3 pts.)

4. Similarly to #2, create an API endpoint at `[Your URL]/api/todo` to return a JSON string with the full set of to-do list items.
   (2 pts.)

5. Add one more API endpoint of your choice with the format `[Your URL]/api/todo/XXX` (where “`XXX`” can be anything) to return some subset of to-do list items, however you wish to define.
   (2 pts.)

Great work! As you see here, Flask servers are popular ways of implementing APIs. Here, you learned how to enable basic API read operations. You have already seen how to enable API write operations; in fact, you already implemented them! Your `/post_announcement` route, `/add_teammate` route (or whatever you named it), and other routes for writing to-do list data via POST are exactly that.