Building Web Apps, Flutterer, My CS Story 🎄

CS106AX: Programming Methodologies in JS and Python

First guest lecture of quarter! 💖

Ben Yan, Nov 21st 2025

flutterer

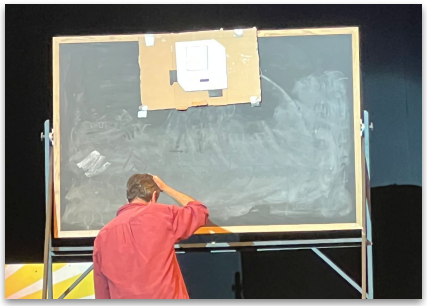
Ben Yan

Welcome to Flutterer! I'm Ben!

CS106AX Autumn 2025 🍁

Stanford | ENGINEERING
Computer Science

Welcome to Friday of Week 9! 🥳



🎃 🍁 Autumn

Week 9

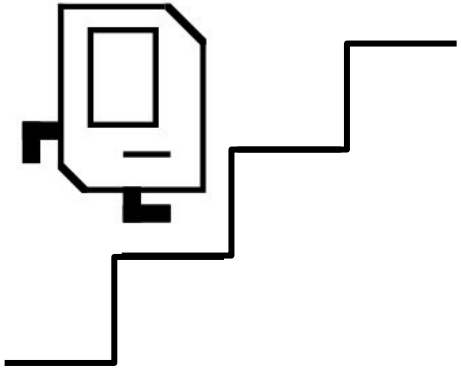
Winter ⛄ ⛷️

We're almost there – you're doing amazing!

Any fun break plans? Anyone going to Gaieties / Big Game?

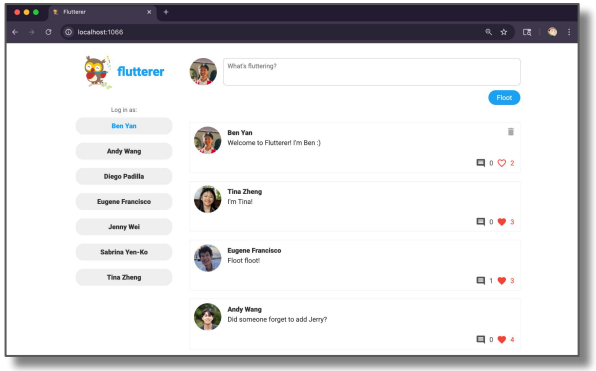
The Map For Today

1. Hello world! Building web apps



2. Coding: Persistent To-Do List

3. Assign8: Flutterer

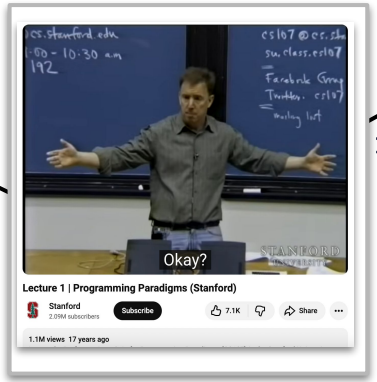


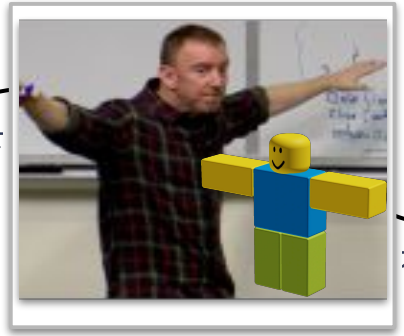
4. Lastly, Ben's CS story!

The good, the bad, the very bad, the very very good

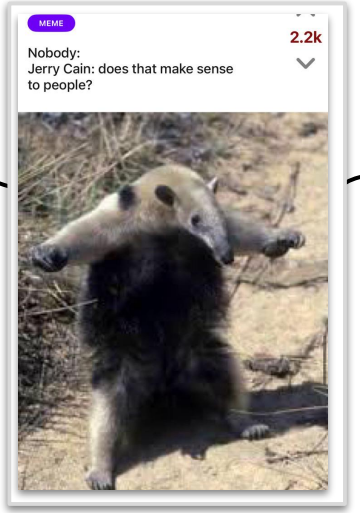
Tysm Jerry for letting me lecture today!! 💖

As gratitude—and **reflecting back on memories of his teaching, which I'll never forget**—perhaps the largest collage of Jerry ever assembled on a lecture slide.

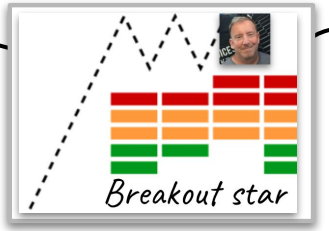




Once upon a time

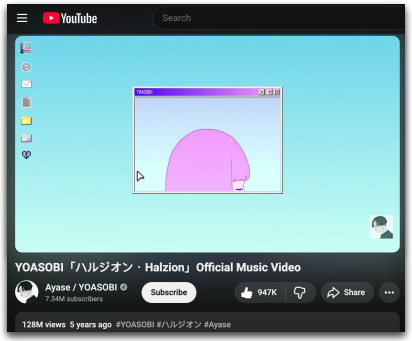


First photo with Jerry!



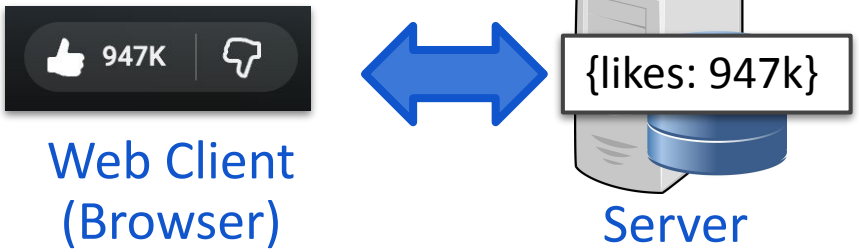
Building Web Applications

- **Thin-Client:** Most web apps minimize the amount of data needed for initial user experience and interaction – e.g., an initial HTML outline / skeleton
- **As new resources are needed** (e.g., images to load, videos, new content), **the website can download them from server in the background i.e. *asynchronously.***
- Once downloaded, JavaScript can update the DOM tree to include those resources on the webpage.



Key Idea: We need to **make sure the client and server as on the same page (get it?)**

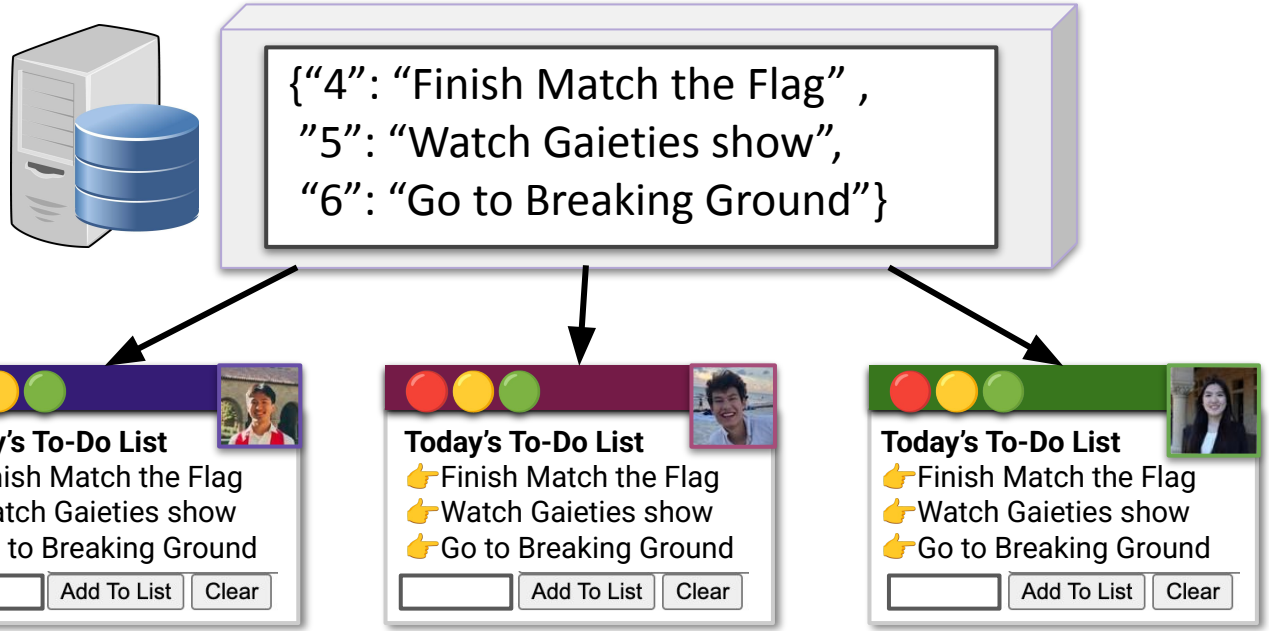
- The browser's presentation of data / information **should be in sync** with the information stored in the server.



Code Example: Persistent To-Do List

We'll revisit the **to-do list** from last week and **implement one key difference**!

➔ We'll require the server to store a **persistent copy of the list** that can be syndicated to every browser that ever connects to it.



Code Example: Persistent To-Do List

New Feature: We'll require the server to store a **persistent copy of the list** that can be syndicated to every browser that ever connects to it.

To fully realize the application, we need to define the structure and behavior of a **few endpoints—to communicate / synchronize state** between client and server.

Those endpoints are:

GET	**`/scripts/getListItems.py`**, which returns a JSON object of item-id/item-text pairs.	{"4": "Finish Match the Flag" , "5": "Watch Gaieties show"}
POST	**`/scripts/addListItem.py`**, which receives the text of a new item so it's shared with / stored by server.	{"4": "Finish Match the Flag" , "5": "Watch Gaieties show", **"6": "Go to Breaking Ground"**}
POST	**`/scripts/removeListItems.py`**, which receives a JSON list of item-ids the server should delete.	{"4": "Finish Match the Flag", "6": "Go to Breaking Ground"} *Just watched Gaieities!*

`index.html` of Persistent To-Do List

Very often, the **HTML is small and only includes elements that are always visible**,

- For instance, HTML has text input, 🔲 2 buttons, skeleton of unordered list.
- We use JavaScript and **`AsyncRequests`** to **fill in the HTML skeleton** / pull in list items, which can potentially vary with each page load.

`index.html` of Persistent To-Do List

Very often, the **HTML is small and only includes elements that are always visible**,

- For instance, HTML has text input, 2 buttons, skeleton of unordered list.
- We use JavaScript and **`AsyncRequests`** to **fill in the HTML skeleton** / pull in list items, which can potentially vary with each page load.

```
<!DOCTYPE html>
<html>
   <head>
      <meta charset="UTF-8">
      <title>To-Do List</title>
      <script src="async.js" type="text/javascript"></script
      <script src="todo.js" type="text/javascript"></script>
   </head>
   <body>
      Today's To-Do List:
      <ul id="to-do-list">
         <!-- This is where list items will be placed. -->
      </ul>
      New item: <input id="item-text" size="60"/>
      <button id="add-item" type="button">Add To List</button>
      <button id="clear-all-items" type="button">Clear</button>
   </body>
</html>
```

<body> section is what displays on the actual webpage

index.html of Persistent To-Do List

Very often, the **HTML is small and only includes elements that are always visible**,

- For instance, HTML has text input, 2 buttons, skeleton of unordered list.
- We use JavaScript and **`AsyncRequests`** to **fill in the HTML skeleton** / pull in list items, which can potentially vary with each page load.

```
<!DOCTYPE html>
<html>
   <head>
      <meta charset="UTF-8">
      <title>To-Do List</title>
      <script src="async.js" type="text/javascript"></script
      <script src="todo.js" type="text/javascript"></script>
   </head>
   <body>
      Today's To-Do List:
      <ul id="to-do-list">
         <!-- This is where list items will be placed. -->
      </ul>
      New item: <input id="item-text" size="60"/>
      <button id="add-item" type="button">Add To List</button>
      <button id="clear-all-items" type="button">Clear</button>
   </body>
</html>
```

<body> section is what displays on the actual webpage

⚫ **Empty Skeleton of Todo List**

💬 **Text Box for Input**

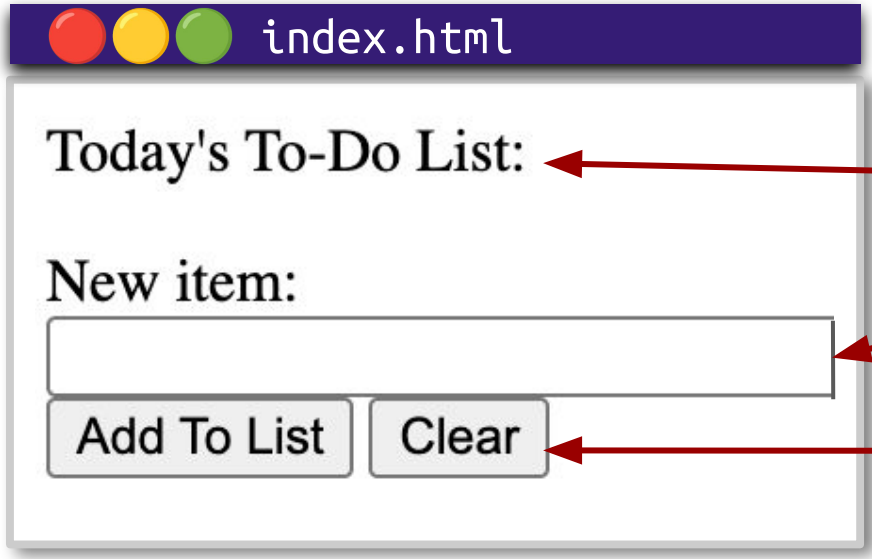
▶️ **Button for Adding to List**

🟧 **Button for Clearing List**

`index.html` of Persistent To-Do List

Very often, the **HTML is small and only includes elements that are always visible**,

- For instance, HTML has text input, 2 buttons, skeleton of unordered list.
- We use JavaScript and **`AsyncRequests`** to **fill in the HTML skeleton** / pull in list items, which can potentially vary with each page load.



Pretty barebones right? But neat

- Empty Skeleton of Todo List
- Text Box for Input
- Button for Adding to List
- Button for Clearing List

Let’s start coding it up!

🗂️ We’ll start with the **`getListItems.py`**, endpoint—server-side first, then client-side

Implement `getListItems` GET Endpoint

Server

File: /scripts/getListItems.py

```
def extractItems(name):                    # used by all
    with open(name) as infile:             three endpoints
        return json.loads(infile.read())

def publishPayloadAsJSON(response):
    responsePayload = json.dumps(response)
    print("Content-Length: " + str(len(responsePayload)))
    print("Content-Type: application/json")
    print()
    print(responsePayload)

info = extractItems("items.json")
publishPayloadAsJSON(info["items"])
```

items.json (data)

```
{"id": 7,
 "items": {"4": "Finish Match the Flag",
           "5": "Watch Gaieites show",
           "6": "Go to Breaking
                 Ground"
           }
}
```

✳ `"id"` key stores the id of the next item to be added

📚 `"items"` key stores an object mapping item ids to the text

This is the server-side endpoint that will be called by the onload **`AsyncRequest`**, to populate the list with its **current items**. The **server's response payload** will appear as:

{"4": "Finish Match the Flag" ,
"5": "Watch Gaieties show",
"6": "Go to Breaking Ground"}

Payload sent via HTTP to client

Implement `populateInitialList()`

Client

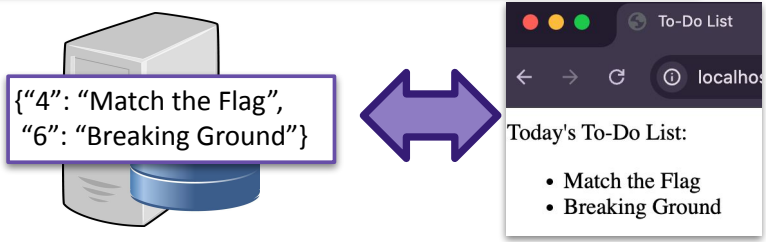
File: todo.js

```
function BootstrapToDoList() {
  let ul = document.getElementById("to-do-list");
  let addButton = document.getElementById("add-item");
  // other variables omitted for brevity

  AsyncRequest("/scripts/getListItems.py")
     .setSuccessHandler(populateInitialList)
     .send(); # notation provided to daisy chain method calls

  function populateInitialList(response) {
       while (ul.lastChild !== null) ul.removeChild(ul.lastChild);
       let items = JSON.parse(response.getPayload());
       for (let key in items) appendListItem(key, items[key]);
  }  // implementation of appendListItem on next slide
}
document.addEventListener("DOMContentLoaded", BootstrapToDoList);
```

The above JavaScript is a subset of the full ***controller***, but just enough to be clear **how the todo-list is populated with the current set of items** stored on the server.

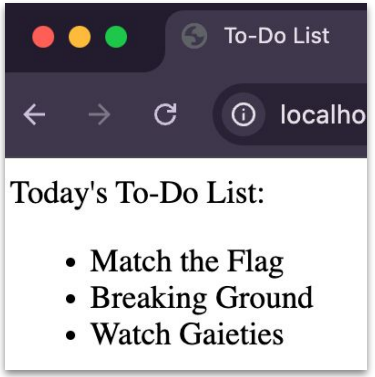


Implement `appendListItem()`

Client

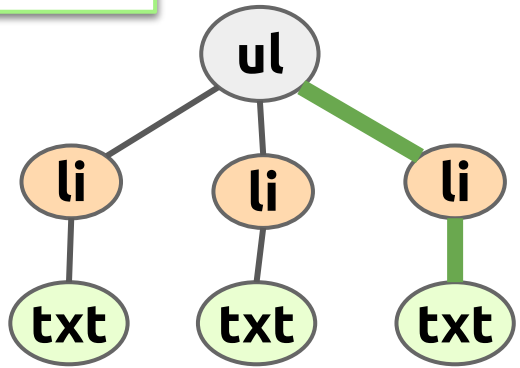
File: todo.js (part 2)

```
function appendListItem(itemid, text) {
    let li = document.createElement("li");
    let tn = document.createTextNode(text);
    li.setAttribute("id", "item-" + itemid);
    li.setAttribute("data-id", itemid);
    li.addEventListener("dblclick", onItemDoubleClick);
    li.appendChild(tn);
    ul.appendChild(li);
} // implementation of onItemDoubleClick on future slide
```



Similar to prior versions of to-do list app: we create a **text** node of the supplied text, and embed it in a new **<li>** node, which we hang from the running list **<ul>** node.

Key Difference: We add an **id** attribute so the node can be discovered in JS by **document.getElementById**, and a **data-id** attribute that stores the raw list ID like in the server.



Implement `onAddClick()`

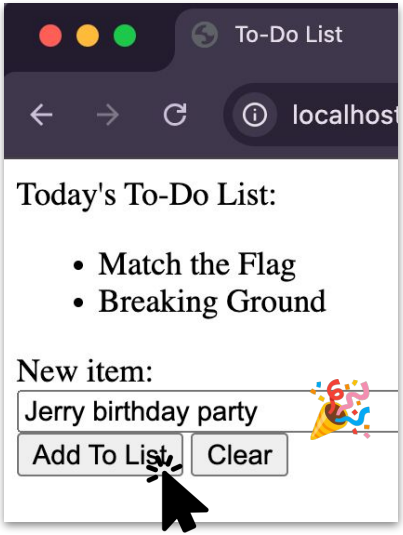
Client

File: todo.js (part 3)

```
function onAddClick(e) { // e argument is ignored
    let text = input.value.trim();
    input.value = "";
    if (text.trim() == 0) return; // nothing in text box

    AsyncRequest("/scripts/addListItem.py")
      .setMethod("POST") // defaults to "GET", so override
      .setPayload(JSON.stringify(text))
      .setSuccessHandler(addNewListItem)
      .send();
}

function addNewListItem(response) {
   let payload = JSON.parse(response.getPayload());
   appendListItem(payload.id, payload.item);
}
```



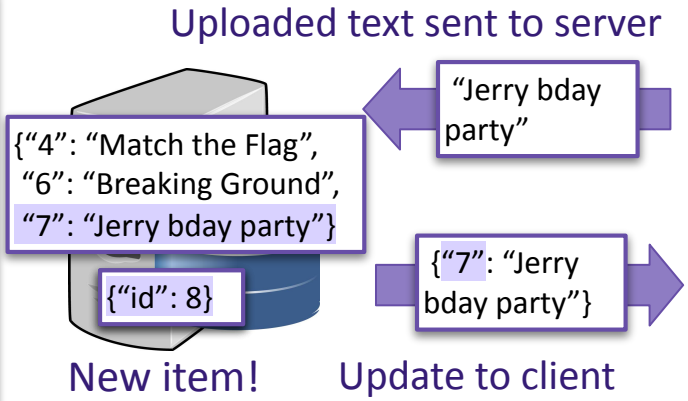
- **onAddclick** is invoked whenever the user clicks the **Add To List** button.
- We want this item to persist server-side, so we issue a **"POST" AsyncRequest**.
- We install a callback so, when the server responds, **a new <li> item is added to page.**

Implement `addListItem` POST Endpoint

Server

File: scripts/addListItem.py

```
requestPayload = json.loads(extractPayload())
info = extractItems("items.json")
id = info["id"]
newItem = {"id": id, "item": requestPayload}
info["id"] += 1
info["items"][id] = requestPayload
saveAllItems("items.json", info)
publishPayloadAsJSON(newItem)
```



- **addListItem.py** endpoint is invoked by **POST AsyncRequest** on previous slide.
- The upload **request to server includes a payload returned by extractPayload().** The payload is a string constant, and valid JSON provided it has the double quotes.

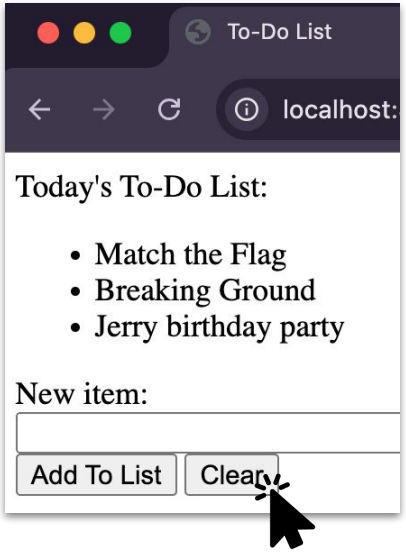
- Our endpoint attaches the next available ID to the uploaded text, **updates the data JSON to include the new {id, item} association**, and publishes a response back to the client whose payload includes the original text and assigned list ID ✅.

Implement `onclearClick()`

Client

File: todo.js (part 4)

```
function onClearClick(e) { // e argument is ignored
    let itemIDs = [];
    for (let i = 0; i < ul.childNodes.length; i++) {
       itemIDs.push(ul.childNodes[i].getAttribute("data-id"));
    }
    AsyncRequest("/scripts/removeListItems.py")
       .setPayload(JSON.stringify(itemIDs))
       .setMethod("POST")
       .setSuccessHandler(removeListItems)
       .send();
 }
```



- **onClearClick()** is invoked whenever the user clicks the **Clear** button.
- It collects all the text item IDs into an array, and POSTs that array to our server-side endpoint **`/scripts/removeListItems.py` to erase them from server data.**
- We install a **client callback removeListItems()** to be invoked once the server responds with which items were successfully deleted (generally all of them), which ones were not

Implement `removeListItems()`

Client

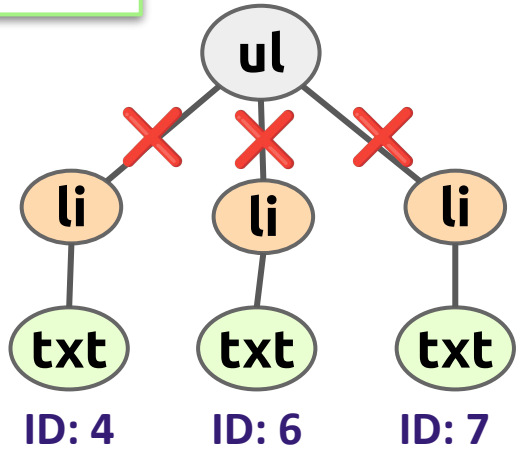
File: todo.js (part 5)

```
function removeListItems(response) {
  let itemsToDelete = JSON.parse(response.getPayload());
  for (let key in itemsToDelete) {
    if (!itemsToDelete[key]) continue; // skip if false
    let li = document.getElementById("item-" + key);
    li.parentNode.removeChild(li);
  }
}
```

Response Payload from Server 📬

```
{
 4: true,
 6: true,
 7: true
}
```

- The above is invoked once the **server `removeListItems.py` endpoint responds with a payload** structured as on the top right:
- **Client JS:** The for loop instructs **each of the relevant <li> nodes to remove itself from its <ul> parent**, thus removing them from the webpage list.

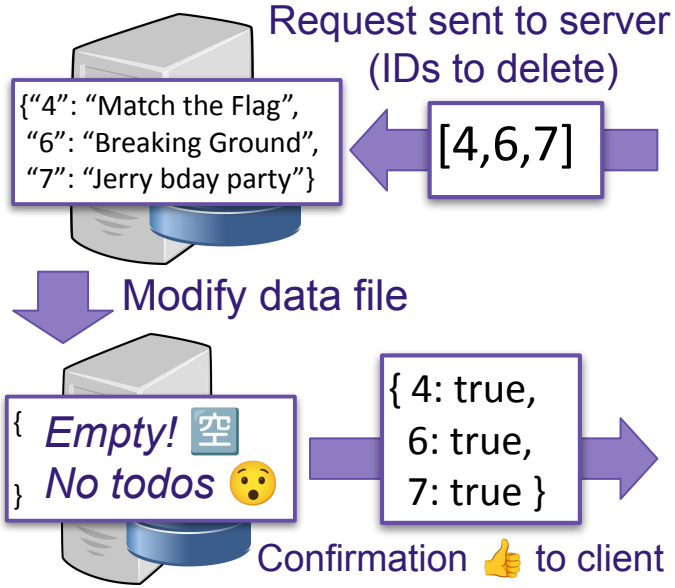


Implement `removeListItems` Endpoint

Server

File: scripts/removeListItems.py

```
itemsToDelete = json.loads(extractPayload())
info = extractItems("items.json")
results = {};
for itemID in itemsToDelete:
    results[itemID] = itemID in info["items"]
    if results[itemID]:
        del info["items"][itemID]
saveAllItems("items.json", info)
publishPayloadAsJSON(results)
```



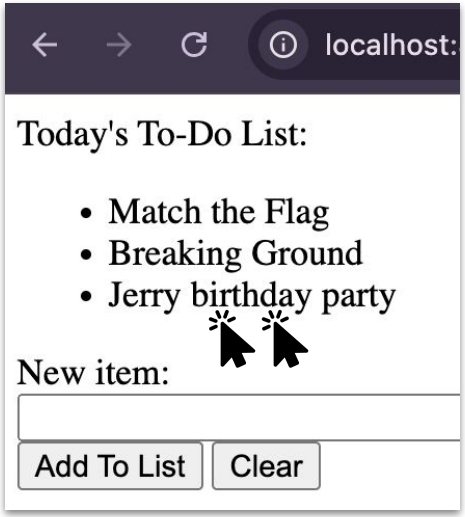
- ❑ This endpoint expects the request payload as an **array of item IDs to be removed**.
- ❑ **For each ID, it removes that item from the data file**—checking the ID is actually still present (e.g., if a second client deleted the item, and the first is out of sync)
- ❑ The script responds with **info on which items were truly deleted**, which ones were not.

Implement `OnItemDoubleClick()`

Client

File: todo.js (part 6)

```
function onItemDoubleClick(e) {
  let itemID = e.target.getAttribute("data-id");
  let itemIDs = [itemID];
  AsyncRequest("/scripts/removeListItems.py")
      .setMethod("POST")
      .setPayload(JSON.stringify(itemIDs))
      .setSuccessHandler(removeListItems)
      .send();
}
```



- The **onItemDoubleClick()** handler **uses the same endpoint OnClearClick()** does.
- But now, we extract the **item id right out of the element that was double-clicked**, embed that item in an array as **/scripts/removeListItems.py** expects, and post the request.
- Note we're able to **install the same success handler used by onClearClick()**, which was coded specifically **to handle both remove-all and remove-one user interactions**. Neat!

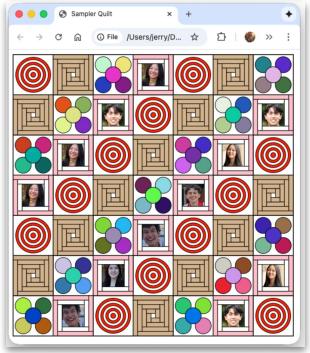
The Assignment Finale! A8: Flutterer 

Partners allowed! #powerOfFriendship 💙

Before we jump in, give yourself a pat on the back for

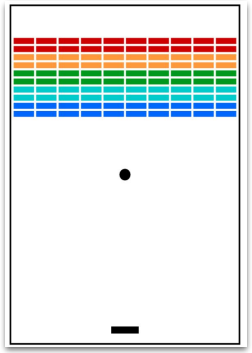
all the amazing work you've done already! 👏👏🎉

🌐 **JavaScript**

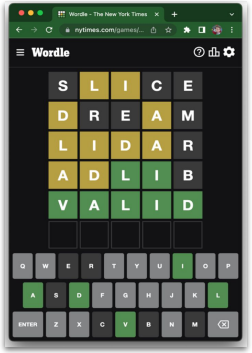


A1: JavaScript Programs

e.g., SamplerQuilt 🎨



A2: Breakout!



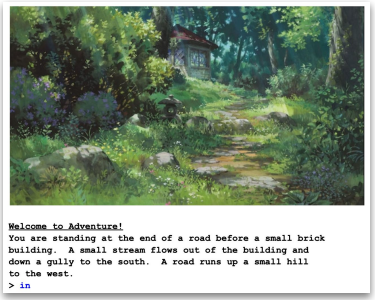
A3: Wordle

A4: Enigma Machine

🐍 **Python**

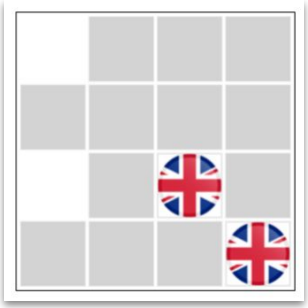
A5: Python Programs

i.e. Random Sentence, Reassemble 🧬

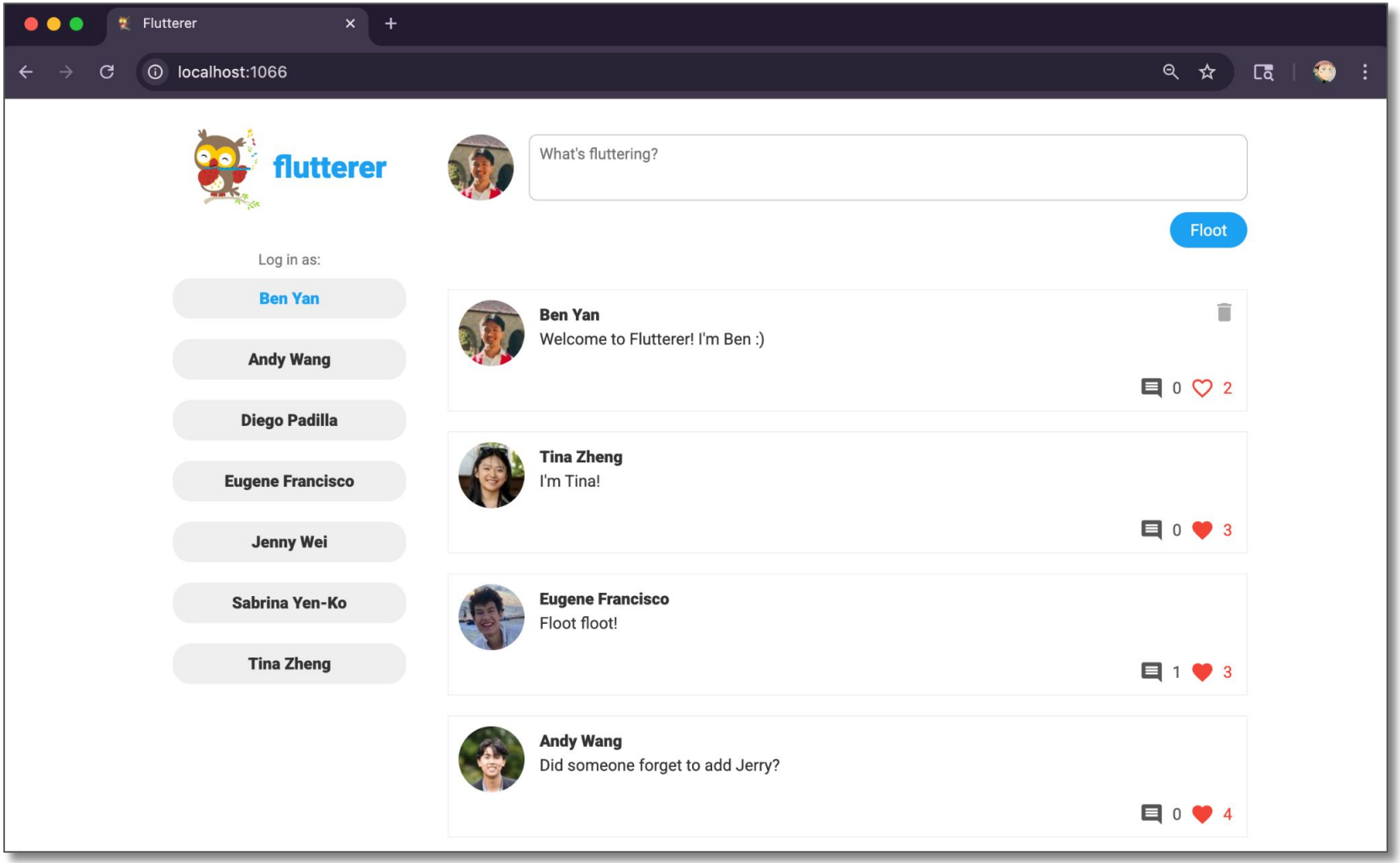


A6: Adventure!

And soon!



A7: Match the Flag


Flutterer
localhost:1066
flutterer
What's fluttering?
Floot
Log in as:
Ben Yan
Andy Wang
Diego Padilla
Eugene Francisco
Jenny Wei
Sabrina Yen-Ko
Tina Zheng
Ben Yan
Welcome to Flutterer! I'm Ben :)
0 2
Tina Zheng
I'm Tina!
0 3
Eugene Francisco
Floot floot!
1 3
Andy Wang
Did someone forget to add Jerry?
0 4

Python and JavaScript! The starter code

This is the first time you'll write Python and JS at the same time, **exciting right?**

- **Python for server side**, e.g., API/managing database, in `/server` folder
- **JavaScript for client side**, e.g., updating webpage, in `/client/js` folder

Client

- `flutterer.js` ✍️
- `floot_components.js` 🖌️
- `modal_components.js` 🖌️
- `general_components.js`
- `comment_components.js` 🖌️
- `sidebar_components.js` 🖌️
- `index.html, style.css`

Server

- `api.py` ✍️
- `database.py`
- `floot.py`
- `floot_comment.py`
- `response.py`
- `serve.py`
- `error.py`
- `test_api.py`

✍️ 2 files to implement (one Python, one JavaScript)

🖌️ 4 files to write small but meaningful code segments in

Part I: Implement Server / API Endpoints

You'll build an API – write a suite of functions / endpoints, each corresponding to a request from the client, i.e. creating a Floot, delete a Floot, get all Floots, etc.

GET	/api/floots
POST	/api/floots
GET	/api/floots/{id}
POST	/api/floots/{id}/delete
POST	/api/floots/{floot_id}/comment/{id}/delete
GET	/api/floots/{floot_id}/comments
POST	/api/floots/{floot_id}/comment
POST	/api/floots/{id}/like
POST	/api/floots/{id}/unlike

📂 For a **GET**, typically you **access the database** for requested info and send it back to client

e.g., when you load Instagram feed

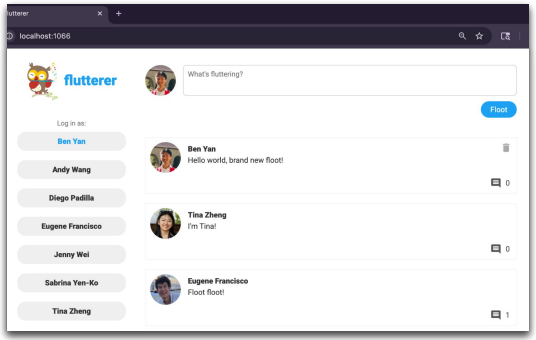
✍️For a **POST**, typically you get info from client, e.g., a new floot, and **modify the database**

e.g., when you post a new photo

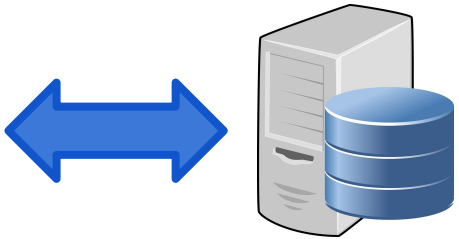
The Flutterer API endpoints! 😍 Note that **like button functionality is extra credit**

Part II: Implement the Client-Side Interators

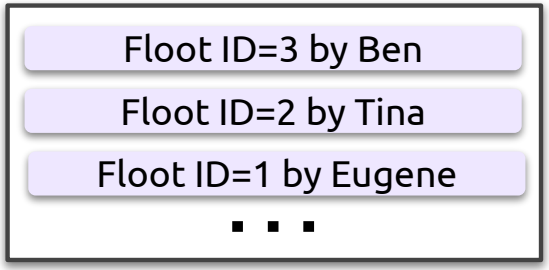
Next, you'll implement the frontend – the JavaScript code that supports the webpage at `localhost:[port-num]`, which becomes truly interactive when communicating with the active Python server (🪄 `python run_server.py`).



Web Client (Browser)



Server



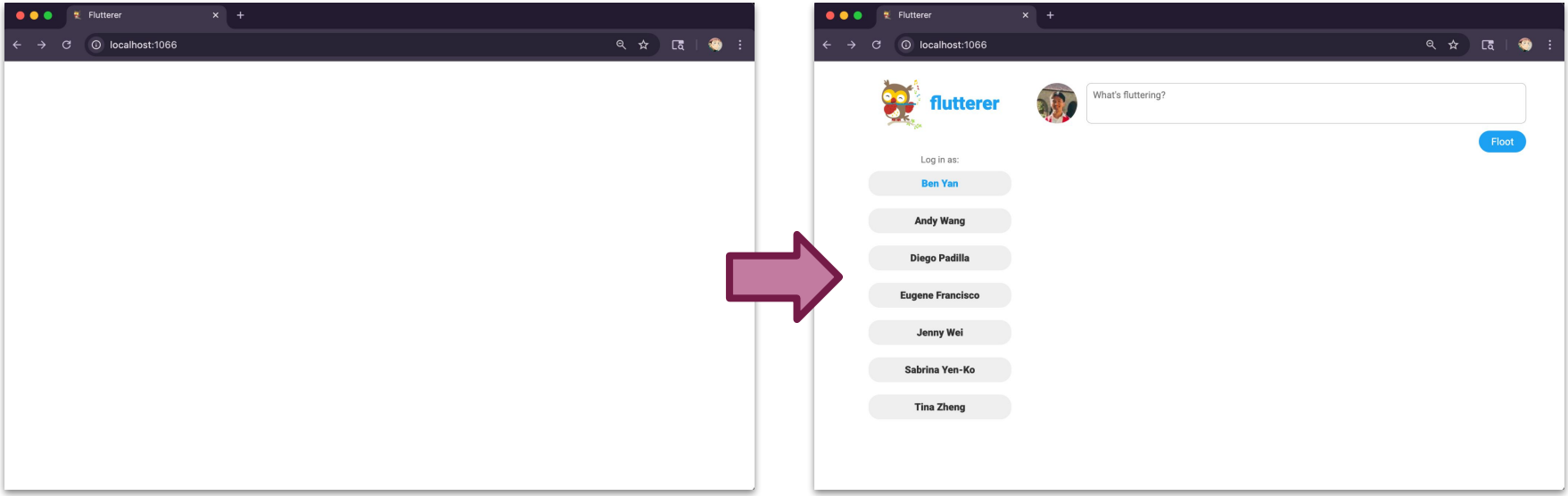
Server Database

Milestone 1: Reading over JavaScript files 🧩

Milestone 1 is just browsing the JS files, and getting a vibe of how it's organized, e.g., the main page-generating `flutterer.js`, the individual `_components.js` files

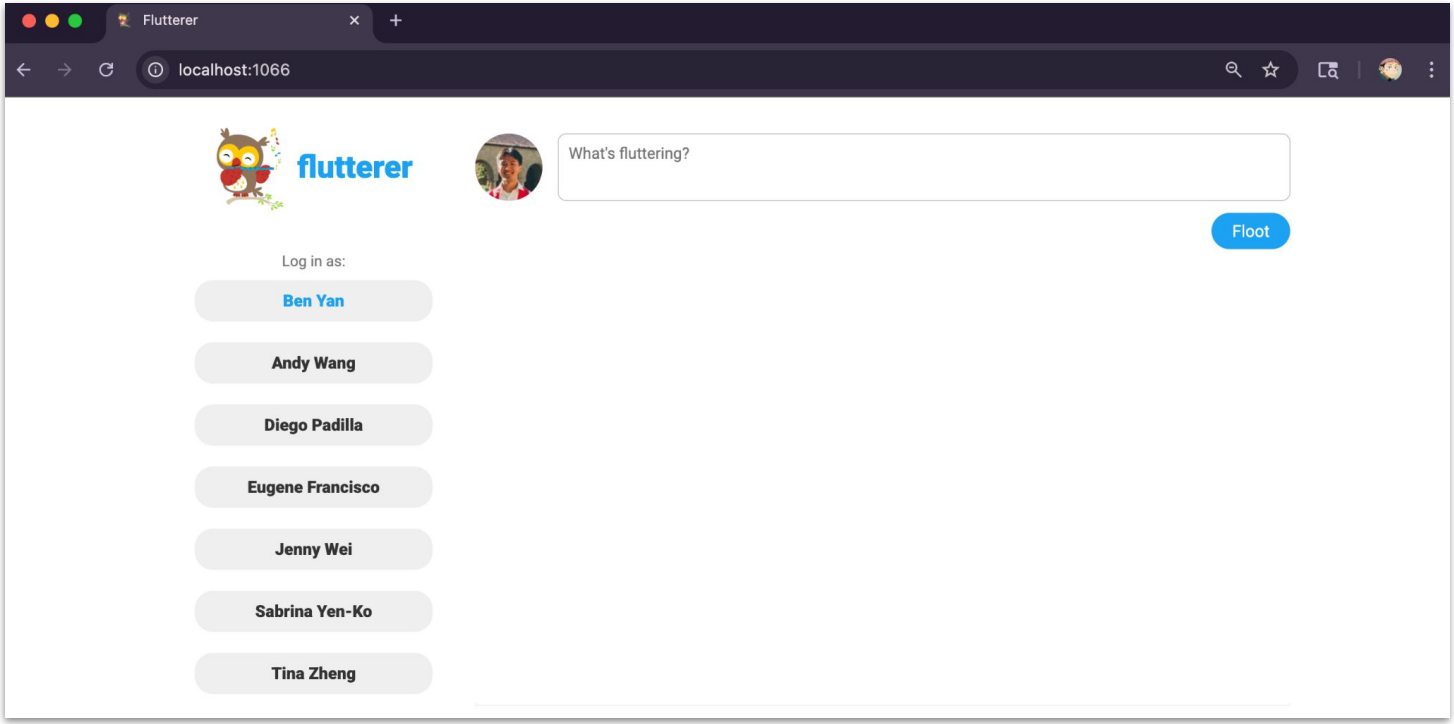
Milestone 2: Display Basic Interface

Implementing the **`MainComponent`** in flutterer.js, by adding nodes for the sidebar and currently hollow **`NewsFeed`**. And voila! From a blank screen to…



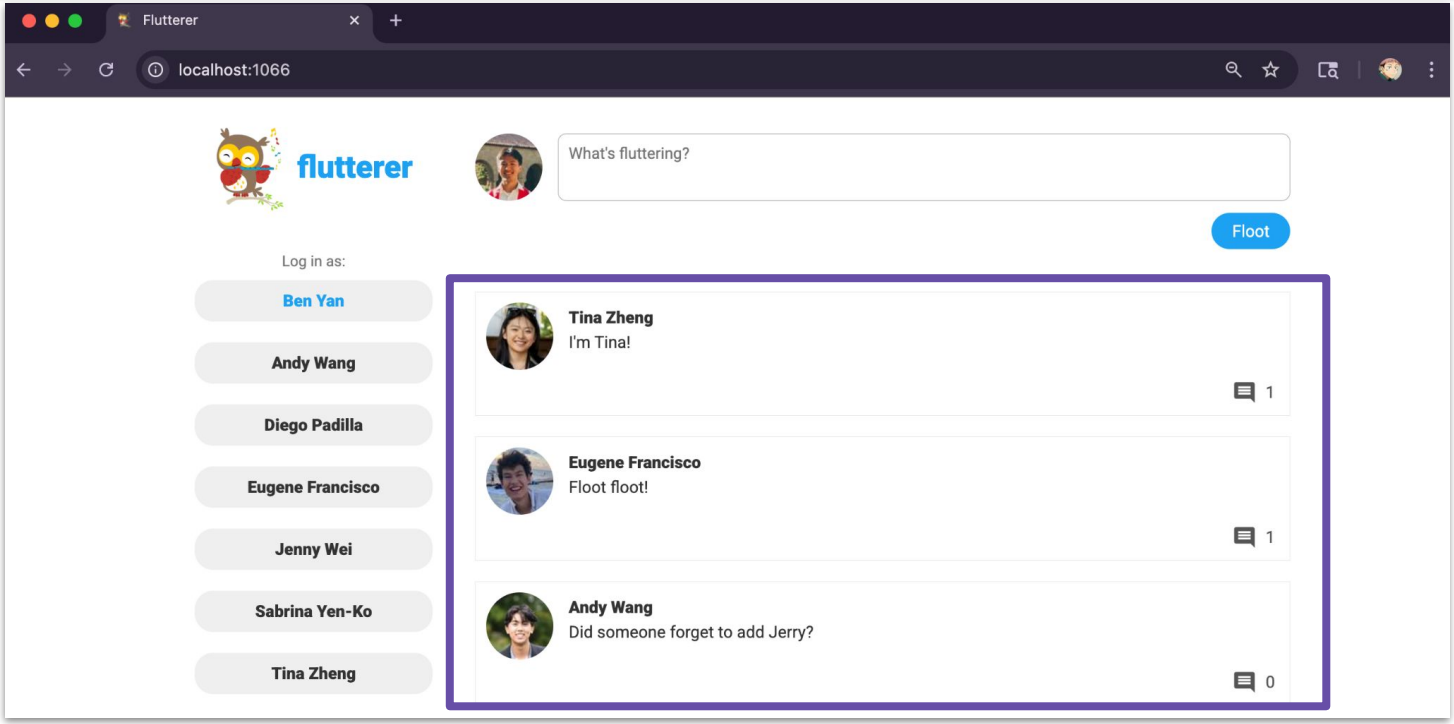
Note: There are **`Sidebar()`** and **`Newsfeed()`** functions you can call, then **`document.body.appendChild(MainComponent(user, floots, actions))`**

Milestone 3: NewsFeed ♻



Milestone 3: NewsFeed ♻️

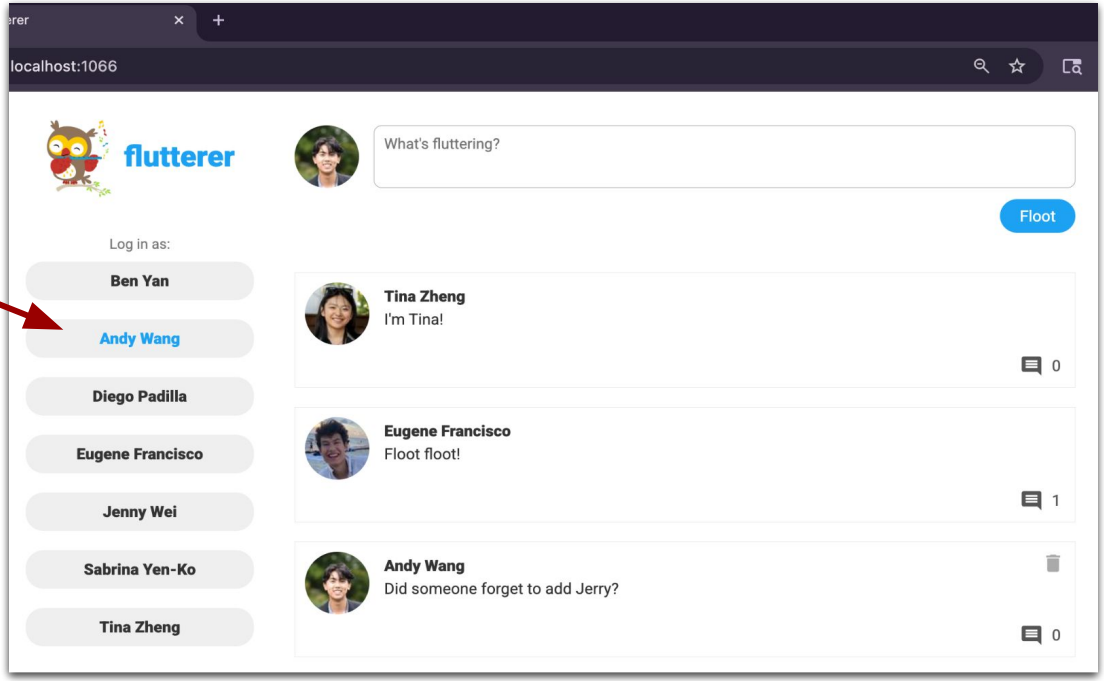
We load in the real NewsFeed, by making an **`AsyncRequest` to `/api/floots`** to **fetch all floots in the server**. Once the server responds, we update the webpage.



Milestone 4: User Switching 🎭

I made myself the starting / default user lol, which is a little self-centered I know. But with this milestone, **now you can switch to a different SL on the sidebar!**

👤 **Andy selected on sidebar**



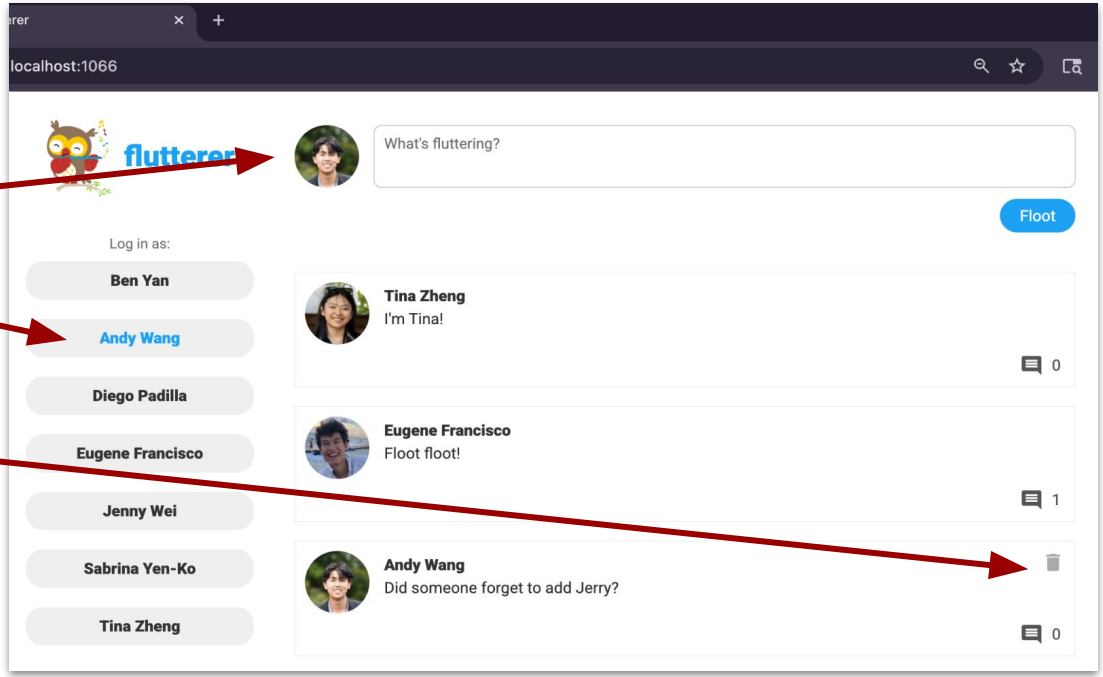
Milestone 4: User Switching 🎭

I made myself the starting / default user lol, which is a little self-centered I know. But with this milestone, **now you can switch to a different SL on the sidebar!**

📸 **The profile picture at top changes to Andy's!**

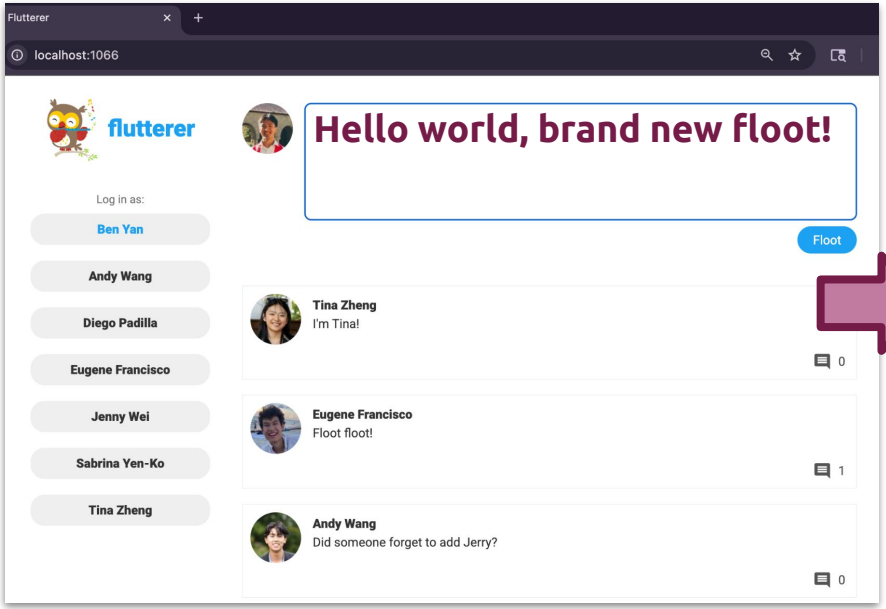
👤 **Andy selected on sidebar**

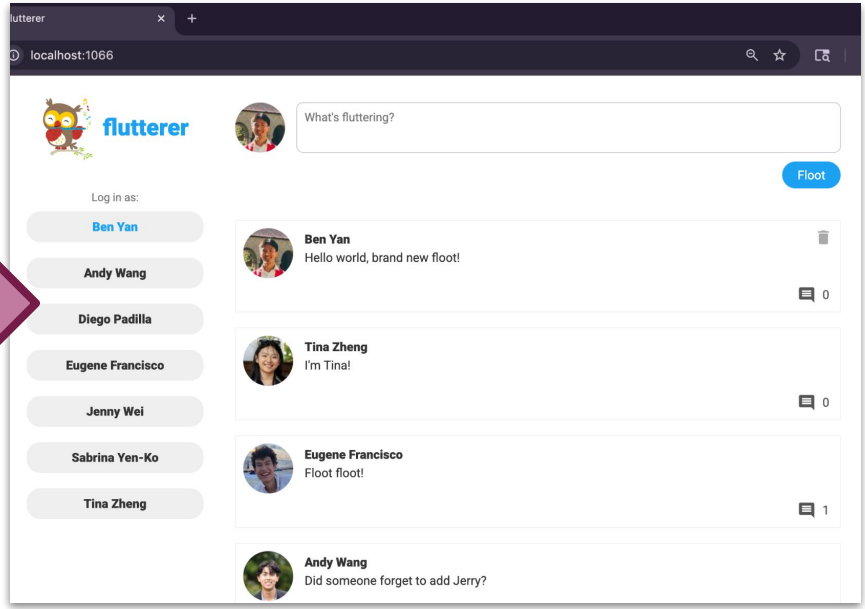
🔑 **Note changes in permissions** (we can now 🗑️ delete tweets by Andy, but not by any other user)



🎯 Aim: Pass a different user in `MainComponent(user,floots,actions)`

Milestone 5: Post New Floots



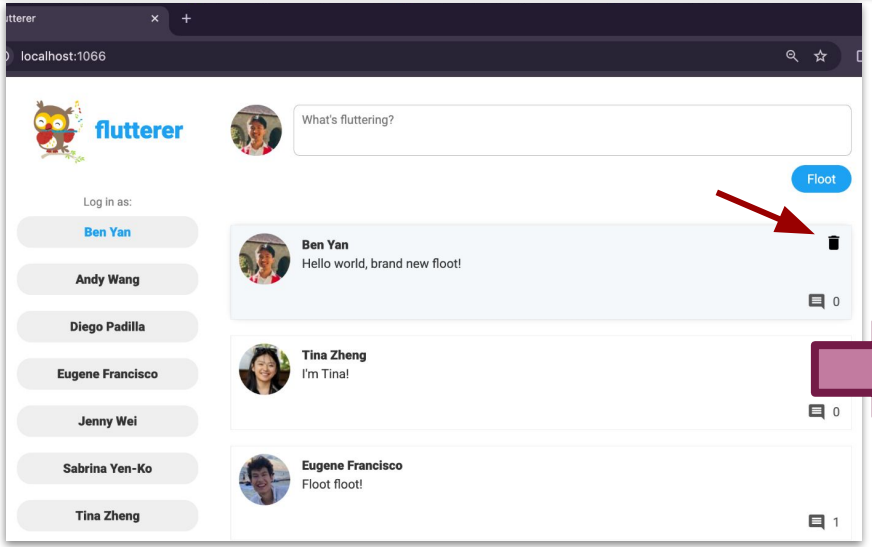


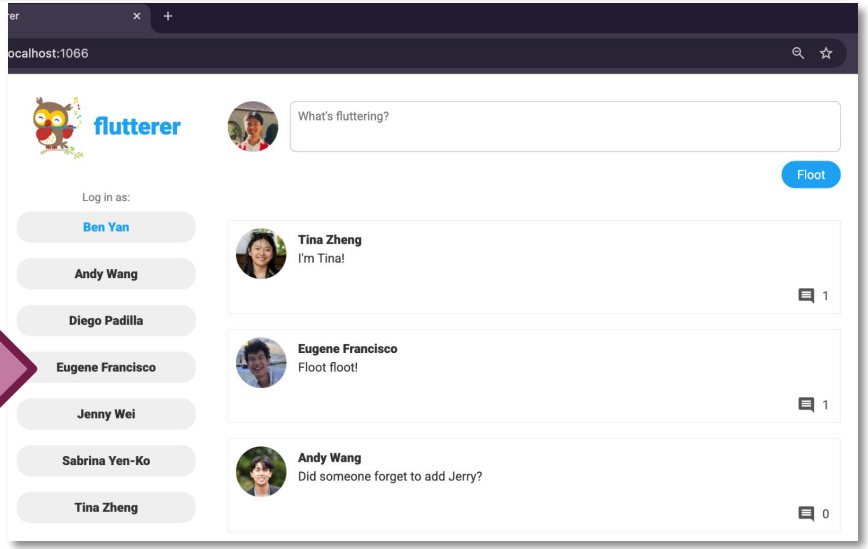
Now, when write in the textbox and click Floot , the JavaScript client should:

1. Send a **`POST`** request to update the server database **by adding new Floot**
2. When server confirms it's done, send a **`GET`** request for **updated list of all floots**
3. Finally, when server responds with the floots, update the webpage. Whew!

Milestone 6: Delete Floots

What if you want to backtrack? Now, by clicking the 🗑️, **you can delete your own Floots** (but not the Floots belonging to other users, in which the 🗑️ button should not appear)

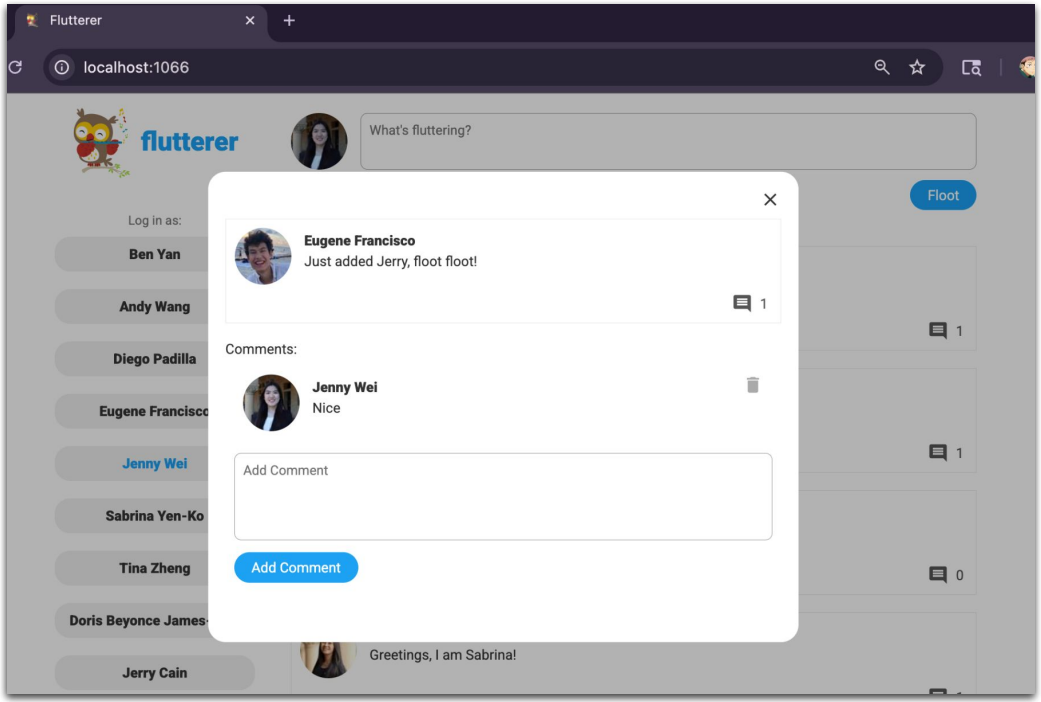




1. Send a **`POST`** request to update the server database to **delete that specific floot**
2. When server confirms it's done, send a **`GET`** request for **updated list of all floots**
3. Finally, when server responds with the floots, update the webpage.

Milestone 7: Show Comments in FlootModal 💬

By clicking the Floot, you can **open a pop-up or modal window with that Floot's comments** 🍵**.** You can depart the pop-up by clicking outside it or the ✖ button.

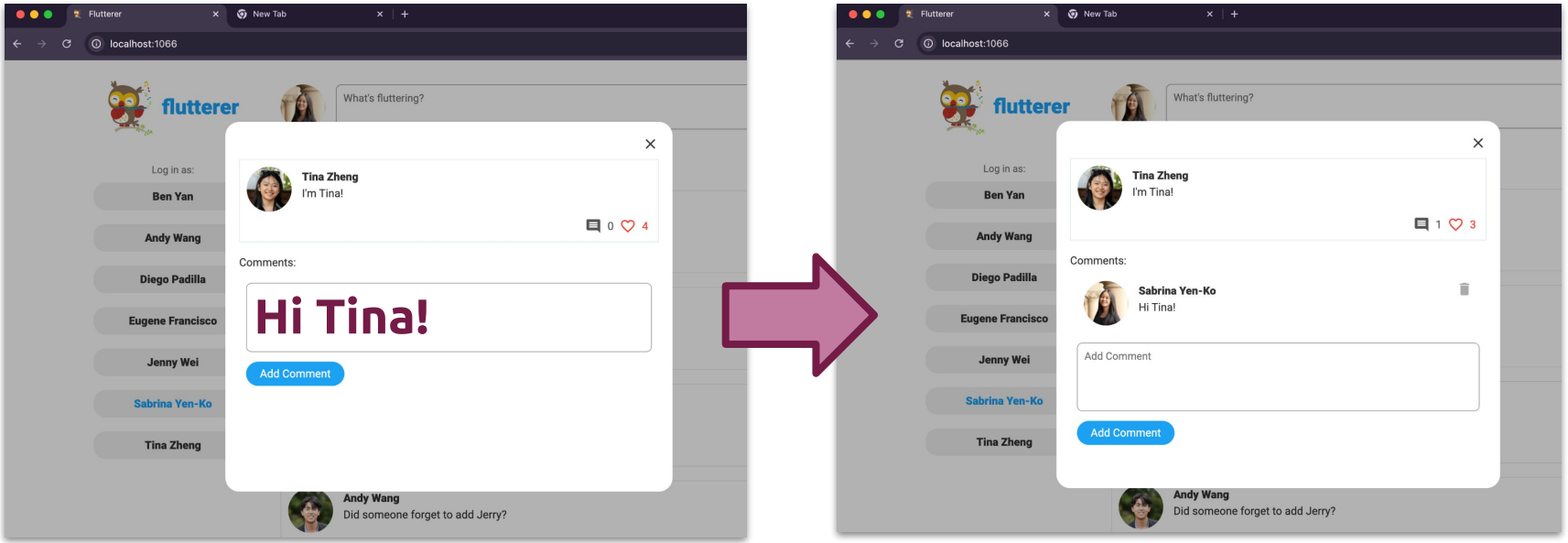


Jerry approves 👍

Jerry is fluttering 🦋

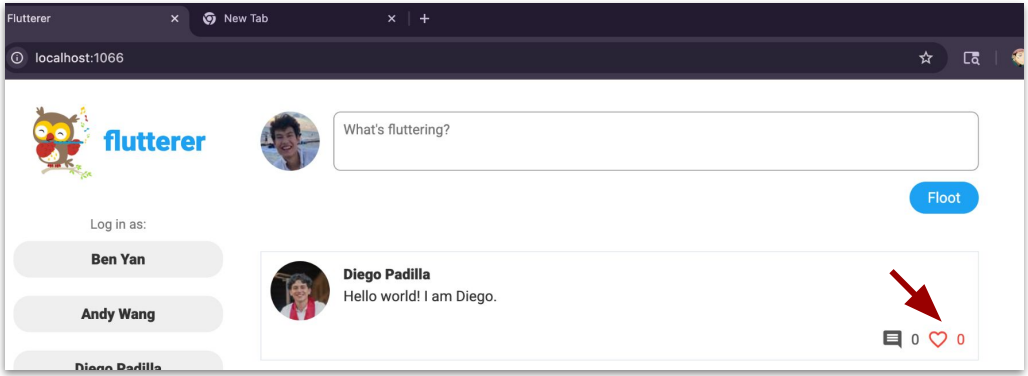
Milestone 8: Create and Delete Comments

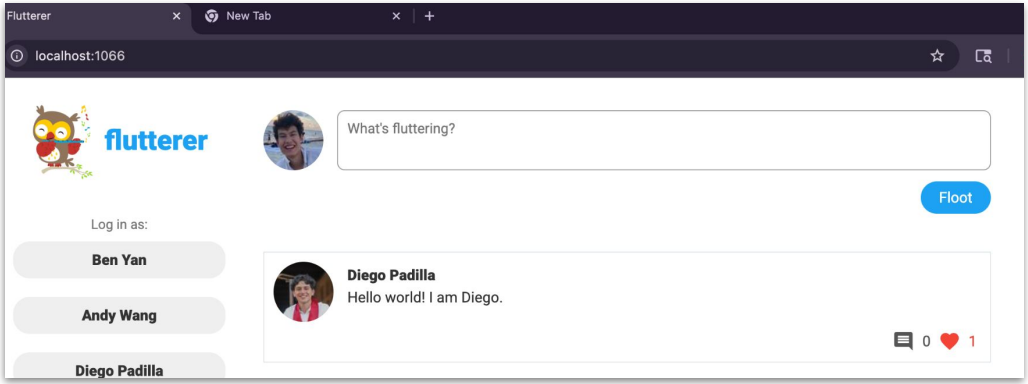
The final milestone is to allow adding / deleting comments in the pop-up window!



1. Send a `POST` request to update the server to **add/delete that specific comment**
2. When server confirms it's done, send a `GET` request for **updated list of all floots**
3. When server responds with floots, **update the webpage—and pop-up window!**

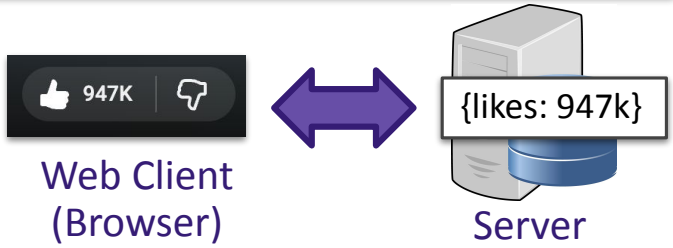
Extra Credit: Like Button for Floots





"It was only three months into the implementation of it that I realized just how big it was going to be."

– Engineer of Facebook Like Button



Server: Write 2 API endpoints to work with like counters data ❤️.

Client: Most of frontend code for like button is provided, but you'll have to wire up the pieces and make it all interactive & working.

🦋 Live Demo / 106AX Social Network on Assignment Page!

A gentle note that this is shared server across all students, so **please make sure your Floots are respectful**, in good nature, and in compliance with Stanford's Fundamental Standard. That is, don't use this platform for testing content moderation, thank you!

Happy to answer questions on Flutterer!

flutterer

✍️ Quick Announcement: Official Course Evals

They release at the start of Week 10 – **we'd really appreciate and value your honest, anonymous feedback there for CS 106AX when you have time!** ❤️

Your evaluations help the course, Jerry, Ben, and section leaders immensely.

- ➔ There's a **Course** tab to give feedback for Jerry / anyone or aspect of the course!
- ➔ There's a **TA / Section** tab to give feedback for me (Ben) specifically!
- ➔ In section this week, there was a **form at the end to give feedback to your SLs**!

Ben’s CS Story 

The good, the bad, the very bad, the very very good ❤️

🌲 Storyas a heap – a beautiful whirlwind blessed mess

Stanford (‘20→’24→‘26)

♣ Majored in **CS (AI)** ‘24
♣ Also majored in **Math** cuz I’m indecisive af 💅😫
♣ Also minored in **Creative Writing**, as poetry>>JavaScript

Summary: fun, but i got cooked

♠ MS CS ‘26 (AI again 😫)
♠ **My advisor is Jerry!** which has done wonders for my life

What I love Most: Teaching

💮 Starting teaching CS in 2023
💔💔 Rejected first 2 quarters I applied to be a graduate CS TA :(
❤️ 3rd time's a charm! **Head TA of CS106AX** last fall, back this fall 🎉
❤️Then **TAed for CS107**, love it!

❤️Side gig: Lecturer for a 1-unit wonder **CS 106S: Coding for Social Good***, ~110 students overall 💖

Work Stuff

♠ Systems SWE (NVIDIA); CURIS after rejections 1st & 2nd year

Life & Side Quests

♦ **From Minnesota** ☃️, and hope to live in SF / Bay! 🌉
♦ Lived in Lantana, Kimball, Roble, EVGR-A, JRo (kinda)
♦ **Oxford study abroad** senior winter! (Creative Writing 🦄)
♦ Wrote 2 terrible novels 📚
♦ **Language learning** (took 1st-year JAPANLNG sequence, just started learning Korean)
♦ Love manga, YOASOBI 🎶

*Last spring 🍀, **CS106S had a record ~40 students**, and somehow, from the course evaluations, it was the **highest-rated course in the entire Stanford CS department! My students are really too kind** 💚

The truth I guess …

`disassemble.py my_time_in_CS`

🎞️ time for a lowlight reel / noteworthy fails

- My first CS class (frosh Autumn 🍂) was CS109, **which looking back was kinda sus** – prereqs are CS106B, Math 51, and CS103 (soft)
- Nevertheless, Profs Lisa Yan & Jerry Cain made it very welcoming and it felt tough but doable. **But that wasn't the only CS class I took that quarter 😭.**
- Have you ever heard of a class called ***CS229*** 👏***: MACHINE*** 👏 ***LEARNING?*** 👏 **I wish I haven't. I got cooked.**
- Goodness, yeah, that still registers as one of the dumbest and most notorious mistakes I've made here.

- It was brutal. I'm an idiot. But surely I learned my lesson right —

There's More!

- My frosh year (on Zoom), I didn't realize there was a front page on the CS program sheet – and ended up taking CS107 before I took 106B. Wtf Ben
  - ➔ **How did that go?** Uhhhh
- Somehow, I was paddling above water (barely), until the bottom fell out on **Assignment 4: Into the Void**

Assignments	Submitted	Days Late	Functionality	Review
assign4: Into the void*	Sat Feb 20 23:54	3	25/92	permitted
assign5: Some Assembly Required	Sun Mar 07 23:59	2	41/84	

And I collapsed even further in Binary Bomb and Heap Alligator :(🐊

Overall Assignment percentage: 68.41%

broke down crying in OH over heap :(

- At the end of that winter quarter, I passed just half my units, which was quite demoralizing ngl. A lot of time was spent mounting a desperate comeback in 107
  - I think CS106B would have helped hmm 🤔😞


More!


- I was a **Physics** major at one point – if you can believe it, which ended spectacularly badly, partly due to Stats Mech.
- Still **surprised I got to the finish line** of my undergrad, definitely didn't seem that way for like a solid four years.

It's not flattering, but it's true that I took an entire year off from CS in undergrad, **taking no classes / mentally giving up.** It was probably the most hopeless I've felt in a lot of ways (besides CS)

kindness during a very rough time

Thanks, and let me know if I can help you make some progress today or tomorrow. I don't want you feeling like everyone around you has moved on and that you don't have access to help.

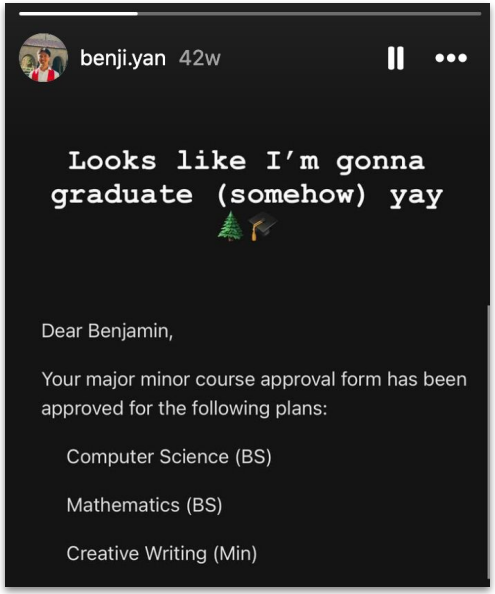
Jerry

- **Aight, that's most of it lol (for now)**, not trying to elevate academic self-sabotage into an art form :(

Hope 🦋

- **On the bright side**, I later took CS110 (predecessor to CS111) **with Jerry, and some friends #powerOfFriendship**. The topics (e.g., Unix filesystems, HTTP web servers) really appealed to me; the class & Jerry greatly healed my confidence.
- 3 years later or so, **I'm still figuring out what kind of CS I really want to do lol.** There's plenty of areas I've decided I won't do well in.

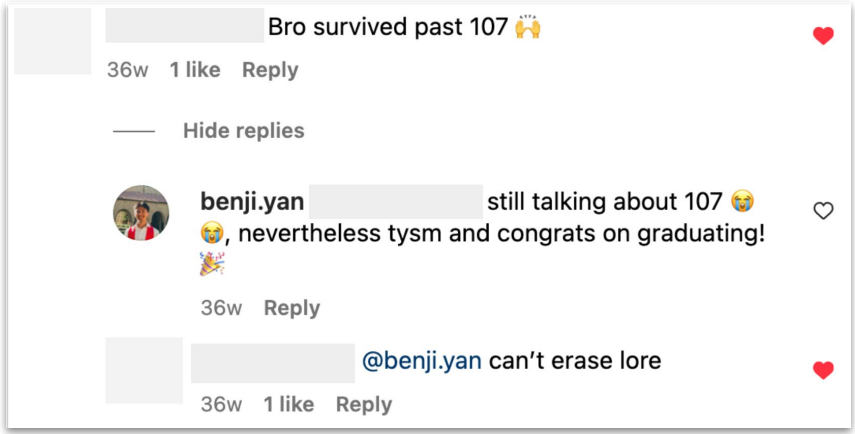
But my dream is to become a full-time lecturer in CS somewhere, someday. But we'll see haha, fingers crossed, praying to the stars that things work out 🌠.



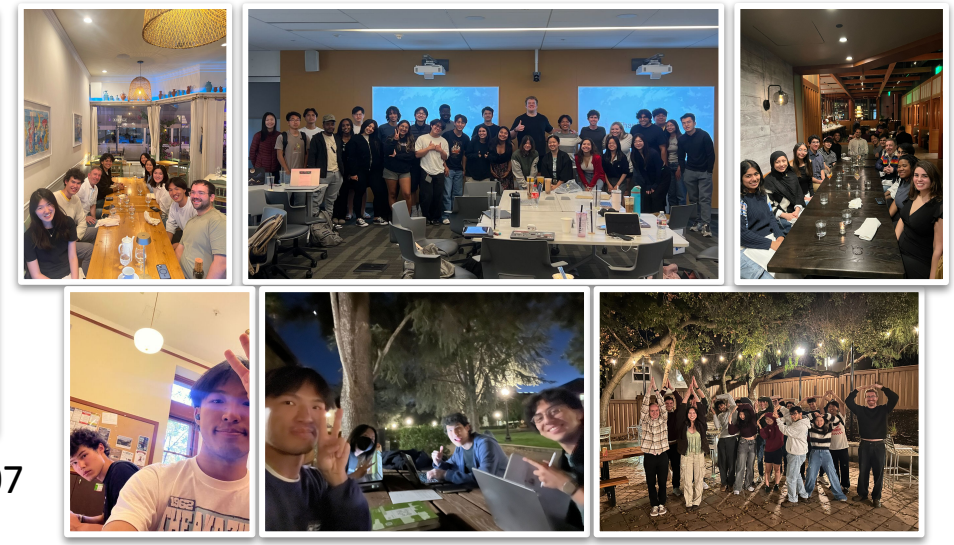
- I – and I'm sure many others – would not still be in my chosen field of study, if it weren't for **the compassion of students & others**.

I love teaching with CS106AX: **my students, the SLs (for AX, Diego, Eugene, Tina, Sabrina, Andy, Jenny, Kieran, Linda, Isaias, Aryan, Rachel), Jerry have given me so much joy 🌠.**

Hope 🦋



A friend **who really helped me power through** 107 when I needed it (context: my IG grad post 🎓)



```
The 4 levels begin now.  Good luck!
Level 1 disarmed. How about the next one?
Level 2 disabled. Keep going!
Level 3 disconnected. And lastly...
Level 4 deactivated. You are granted access to the master vault.
Congratulations! You solved all the levels!
Summarize your vault experience in one word:
```

Finally finished **Binary Bomb (as a 107 TA, just half a year ago 🌈)**; it only took 4 years!

So many blessings in the past years, and though I won't be able to TA 106AX again, **ty for making this time for special** 🥹

I have learned so much from teaching and from you.

So yeah, thank you!!!

I’m so grateful you’re here

And to the SLs, the CS106 courses would simply not be possible without you, and we’re indebted to your incredible work and care!




Thank you for being here, and have an amazing break! 🍁

- ➔ Enjoy the abundant mochi! 🍡
- ➔ AMA: Feel free to reach out after lecture today, today's pre-break office hours 3 – 5 PM, in the CoDa B45 office, or at **`bbyan@stanford.edu`**!
- ➔ Happy to chat about CS106AX, web stuff, and Flutterer, but also about:

- CS Coterm, CS or Math major
- Creative Writing Minor
- Studying abroad (esp. Oxford)
- CS classes / research / activities
- Bay Area & San Francisco 🌉
- How many points we will win by tmrw 🏈
- Language learning, Deuxlingo, etc.
- Anything really! (life, Stanford, etc.)