

Announcements

- 📅 **Milestone 1 (Project Pitch) Signups due tonight!** 🕒
weblab.is/milestone1
- 💬 Please give us feedback! weblab.is/feedback
 - 🥛🍪 And any random feels at weblab.is/milkandcookies
- 🎥 Workshop 3 re-record will be up ASAP
- 🦴 Project skeleton at weblab.is/skeleton
 - Includes some stuff we haven't learned yet (google auth)

Advanced CSS & Other Libraries

Stanley Zhao

Advanced CSS



Advanced CSS Topics

- CSS combinators
- Display types
- Content overflow
- Animations

Setup Playground

Let's use this playground to write our CSS: **play.tailwindcss.com**

Let's also add a simple skeleton to work with in the HTML section; we'll also add some code in the CSS section.

HTML

```
<div class="container">
  <section class="child" id="c1">
    <div>subchild 1</div>
  </section>
  <div class="child" id="c2">child 2</div>
  <div class="child" id="c3">child 3</div>
  <div class="child" id="c4">child 4</div>
</div>
```

CSS

```
.container {
}
```

CSS Combinators

Specifies relationship between CSS selectors. Examples of selectors include HTML tags, such as **div** or **p**

We have 4 different CSS combinators:

- descendant selector (space)
- child selector (>)
- adjacent sibling selector (+)
- general sibling selector (~)

Descendant selector (space)

Matches all elements that are **descendants of the specified element**

- In the below example, we're selecting all the **div** elements that are descendants of the **.container** element → can be deeply nested (**subchild 1 is also selected**)



CSS

```
.container div {  
  font-size: 20px;  
}
```

subchild 1

child 2

child 3

child 4

Child selector (>)

Matches all elements that are **direct children of the specified element**

- In the below example, only the elements denoted as **child** will be selected → further levels of nesting will not be selected



CSS

```
.container > div {  
  font-size: 20px;  
}
```

subchild 1

child 2

child 3

child 4

Adjacent Sibling Selector (+)

Selects a *single* element that is **directly after another specific element**

- In the below example, the **element directly after the element with an id of "c1"** is selected

```
● ● ● CSS
#c1 + div {
  color: red;
}
```

subchild 1

child 2

child 3

child 4

General Sibling Selector (~)

Selects *all* elements **after another specific element**

- In the below example, **all elements after the element with an id of "c1"** are selected

```
css

#c1 ~ div {
  color: red;
}
```

subchild 1

child 2

child 3

child 4

Display Types

The **display** property allows us to tell the browser how to display an element and its children on the page.

We've looked at:

- block
- flex

Values we'll be looking at:

- grid
- none

Update Playground

Let's update the code in our playground:

HTML

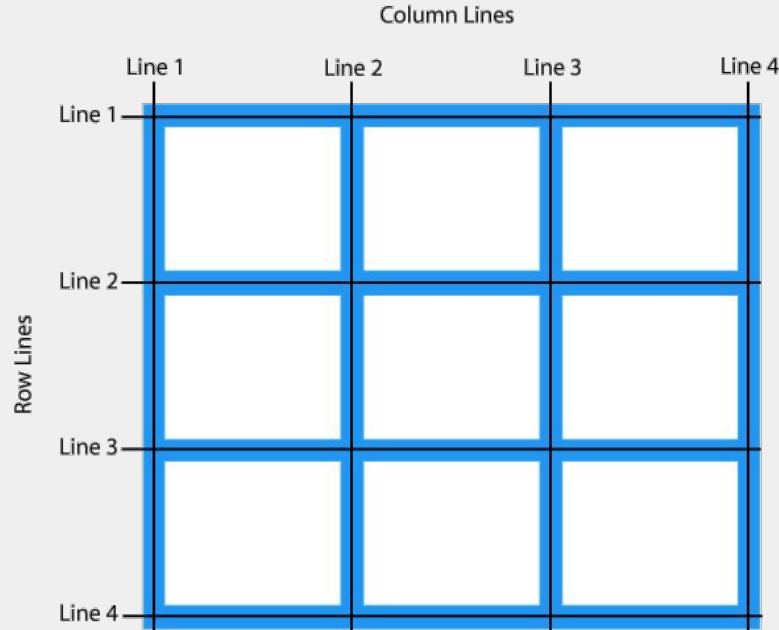
```
<div class="container">
  <div class="child" id="c1">child 1</div>
  <div class="child" id="c2">child 2</div>
  <div class="child" id="c3">child 3</div>
  <div class="child" id="c4">child 4</div>
</div>
```

CSS

```
.container {
}
```

display: grid

Tells the browser to display child elements in a two-dimensional grid layout



display: grid



CSS

```
.container {  
  display: grid;  
}
```

grid-auto-flow

Let's talk about **grid-auto-flow**

- **auto-flow** gives us the power to tell the browser how to automatically handle child elements that reach the end of our grid layout

grid-auto-flow

row instructs the browser to prioritize adding rows

- By default, **grid** will prioritize rows

columns instructs the browser to prioritize adding columns

```
CSS

.container {
  display: grid;
  grid-auto-flow: row;
}
```

```
CSS

.container {
  display: grid;
  grid-auto-flow: column;
}
```


grid-auto-flow

child 1
child 2
child 3
child 4

child 1

child 2

child 3

child 4

```
CSS

.container {
  display: grid;
  grid-auto-flow: row;
}
```

```
CSS

.container {
  display: grid;
  grid-auto-flow: column;
}
```

grid-template-rows/grid-template-columns

This property allows us to “template” rows/columns → # of rows/cols + width/height

grid-template-rows

Number of values → number of rows with specified height

- `grid-template-rows: 100px 100px;`
 - Creates a grid layout giving rows 1+2 of height 100px
 - *Subsequent rows will have a height to fit the element if a value isn't specified*

```

CSS

.container {
  display: grid;
  grid-template-rows: 100px 100px;
}
```

child 1

child 2

child 3

child 4

grid-template-columns

Number of values → number of columns

- `grid-template-columns: 100px 100px;`
 - Creates a grid layout with two columns of width 100px

```
css
.container {
  display: grid;
  grid-template-columns: 100px 100px;
}
```

child 1

child 2

child 3

child 4

display: none

Tells the browser to remove an element from the document → doesn't take up any space in the layout

- This is different from **visibility: hidden**, which hides the element but still takes up space in our layout

display: none

Tells the browser to remove an element from the document → doesn't take up any space in the layout

- This is different from **visibility: hidden**, which hides the element but still takes up space in our layout

child 2

child 3

child 4

child 3

child 2

child 4

```
● ● ● CSS
#c1 {
  display: none;
}
```

```
● ● ● CSS
#c1 {
  visibility: hidden;
}
```

Content Overflow

The **overflow** property allows us to tell the browser how to handle child elements that may exceed the size of a parent element.

Values we'll be looking at:

- visible
- hidden
- scroll
- auto

Update Playground

Let's update the code in our playground:

HTML

```
<div class="container">
  <p>the quick brown fox jumps over the lazy dog</p>
</div>
```

CSS

```
.container {
  width: 50px;
  height: 50px;
  background-color: gray;
}
```


overflow: visible

Default behavior → tells the browser to display the overflowing content

```

CSS

.container {
  width: 50px;
  height: 50px;
  background-color: gray;
  overflow: visible;
}

```

the
quick
brown
fox
jumps
over
the
lazy
dog

overflow: hidden

Tells the browser to clip the overflowing content → cannot scroll within the parent element.

```
CSS

.container {
  width: 50px;
  height: 50px;
  background-color: gray;
  overflow: hidden;
}
```

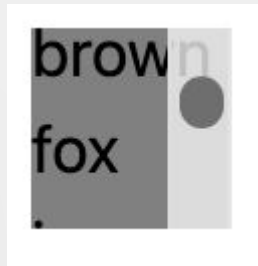


overflow: scroll

Tells the browser to display a scrollbar for the parent element → this **scrollbar will always be present**, even if there may not be any overflowing content.

```
CSS

.container {
  width: 50px;
  height: 50px;
  background-color: gray;
  overflow: scroll;
}
```



overflow: auto

Tells the browser to display a scrollbar for the parent element if needed → this **scrollbar will only be present if there is any overflowing content**.

```
css

.container {
  width: 50px;
  height: 50px;
  background-color: gray;
  overflow: auto;
}
```

overflow content



no overflow content



Animations

We'll explore some ways to give our HTML elements some movement using animations.

Animation topics we'll be looking at:

- Keyframes
- Calling our animation
- Duration
- Delay
- Timing functions

Update Playground

Let's update the code in our playground:

HTML

```
<div class="container">
  <p>the quick brown fox jumps over the lazy dog</p>
</div>
```

CSS

```
.container {
}
```

Keyframes

Describes the animation we're creating, and what will happen at different points of the animation.



```
@keyframes fadeIn{
  0% {
    opacity: 0;
  }
  100% {
    opacity: 1;
  }
}
```

- "fadeIn" is our animation name
- We indicate different points of our animation using percentages
 - 0% → start of the animation
 - 100% → end of the animation
- Inside the percentages we can indicate CSS properties that we'd like to animate.
 - In this case, we're animating opacity.

Calling Our Animation

Once we've created the keyframes for our animation, we can reference the name to call the animation.

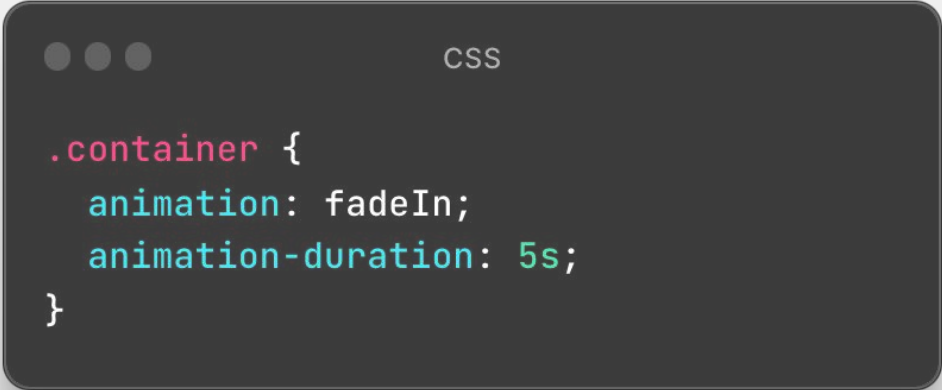


CSS

```
.container {  
  animation: fadeIn;  
}
```


Animation Duration

Tells the browser how long the animation should last.



```
.container {  
  animation: fadeIn;  
  animation-duration: 5s;  
}
```

Animation Delay

Tells the browser a delay before the animation is executed.

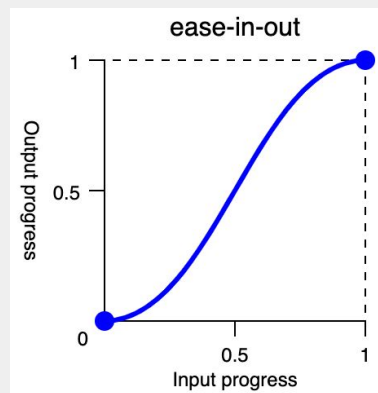
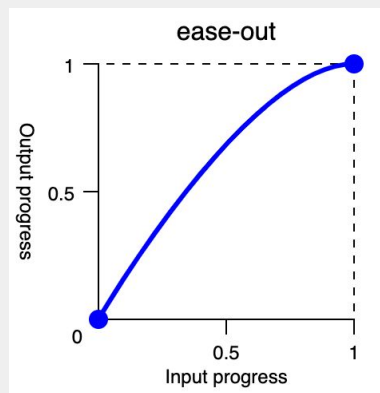
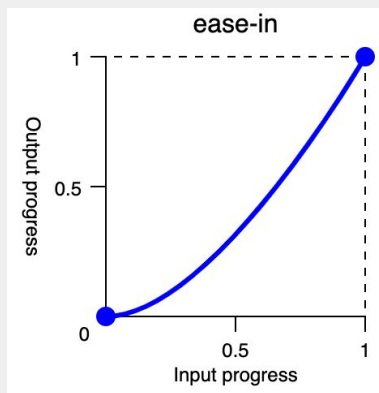
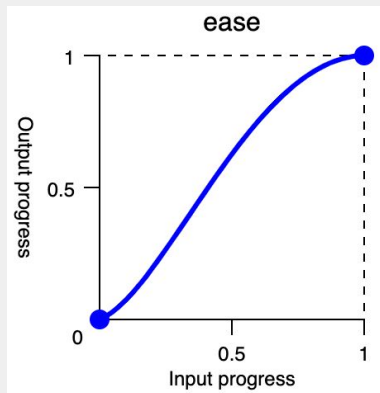


```
.container {  
  animation: fadeIn;  
  animation-duration: 5s;  
  animation-delay: 2s;  
}
```

Timing Functions

Change the speed at which our animation progresses

- **ease (default)**: slow start → fast middle → slow end
- **ease-in**: slow start
- **ease-out**: slow end
- **ease-in-out**: slow start → slow end
- **linear**: uniform speed



CSS Libraries (Tailwind)



What is TailwindCSS?

TailwindCSS is a **utility-first** CSS framework that utilizes pre-made classes to make development quicker and more efficient.

How is it a “**utility-first**” framework?

- TailwindCSS provides utility classes for styling → eliminates struggle of determining a naming scheme
 - Name of the class → what it does (its utility)
 - Ex: **m-4** → margin of 4px
- Pre-made classes make it easy to stay consistent with color choices, spacing, typography, and more.

What is special about TailwindCSS?

TailwindCSS is very **low level**; rather than providing classes that create components, you can create different components even with the same utility classes.

- For example, Bootstrap provides component templates → results in **very uniform, similar-looking components** that are hard to modify at a low level.

What is special about TailwindCSS?

TailwindCSS is very **low level**; rather than providing classes that create components, you can create different components even with the same utility classes.

- For example, Bootstrap provides component templates → results in **very uniform, similar-looking components** that are hard to modify at a low level.

Tailwind reduces CSS bundle sizes to the absolute minimum → Tailwind's compiler automatically discards any unused CSS → **smaller CSS bundle sizes = faster load times**

What is special about TailwindCSS?

TailwindCSS is very **low level**; rather than providing classes that create components, you can create different components even with the same utility classes.

- For example, Bootstrap provides component templates → results in **very uniform, similar-looking components** that are hard to modify at a low level.

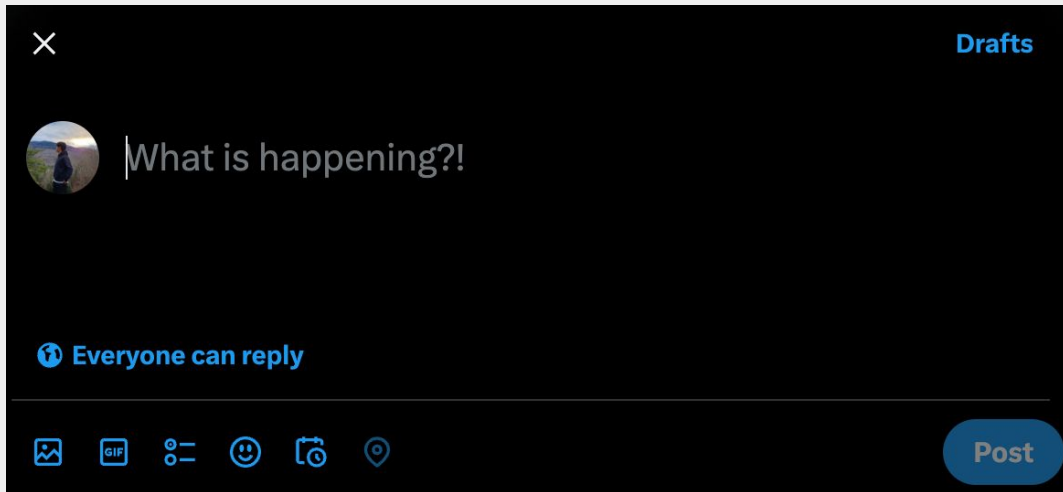
Tailwind reduces CSS bundle sizes to the absolute minimum → Tailwind's compiler automatically discards any unused CSS → **smaller CSS bundle sizes = faster load times**

Tailwind also emphasizes **responsive design** → Tailwind has utility selectors to help handle changes in CSS for different screen sizes.

Tailwind Playground

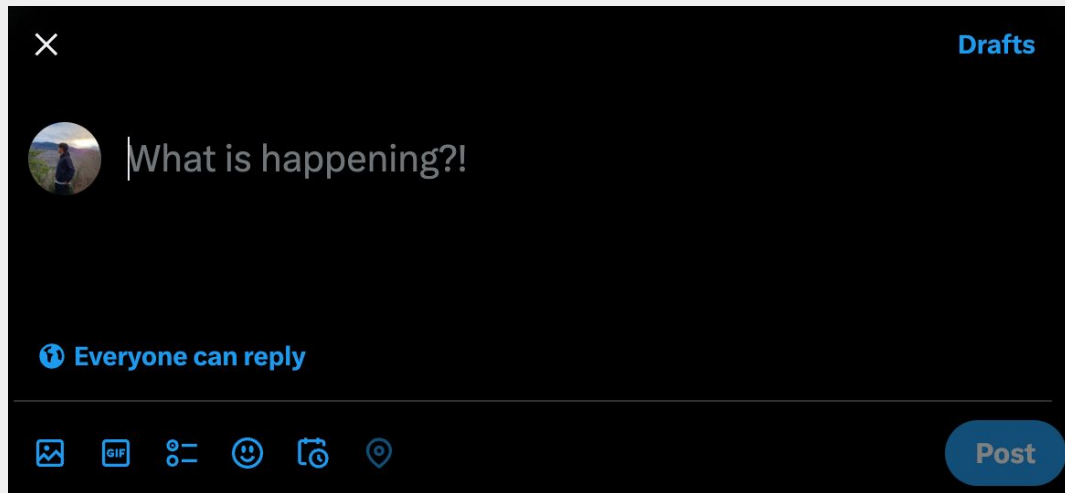
Let's open up the TailwindCSS playground if you haven't already:
play.tailwindcss.com

We're going to try and replicate the Tweet box using TailwindCSS



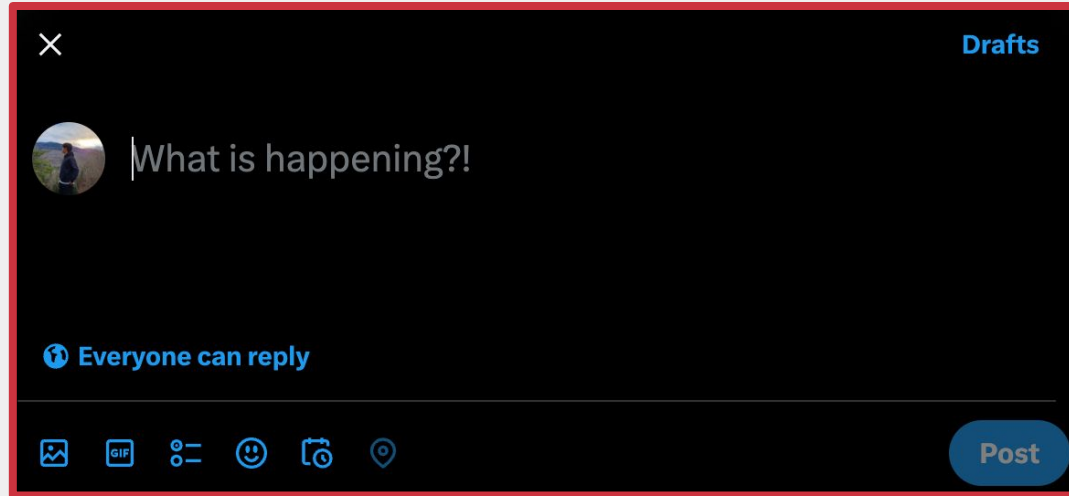
Identify structure

First, we should identify some key structural elements of what we're replicating.



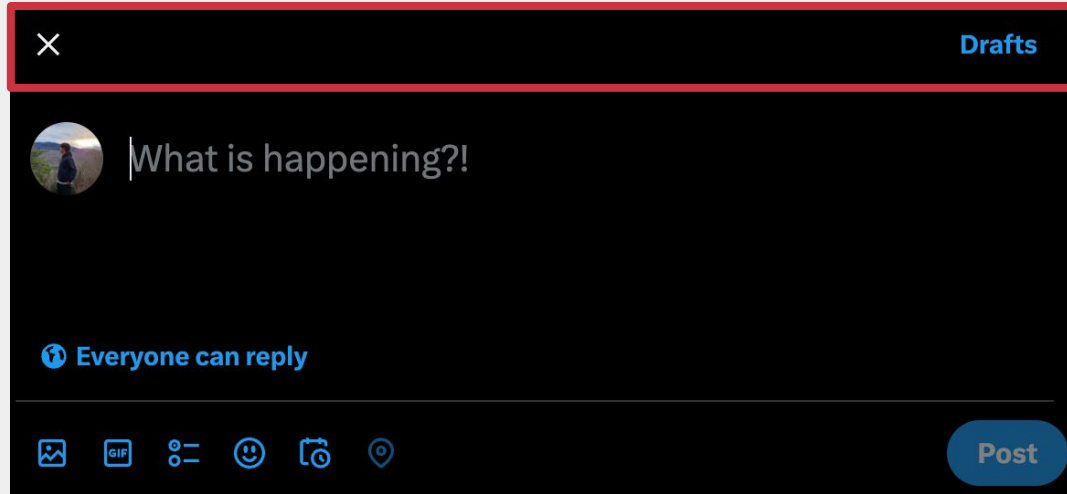
Identify structure

Container for all of the elements with some sort of padding



Identify structure

Bar at the top that contains the button to close the tweet, as well as a button to take us to our drafts.



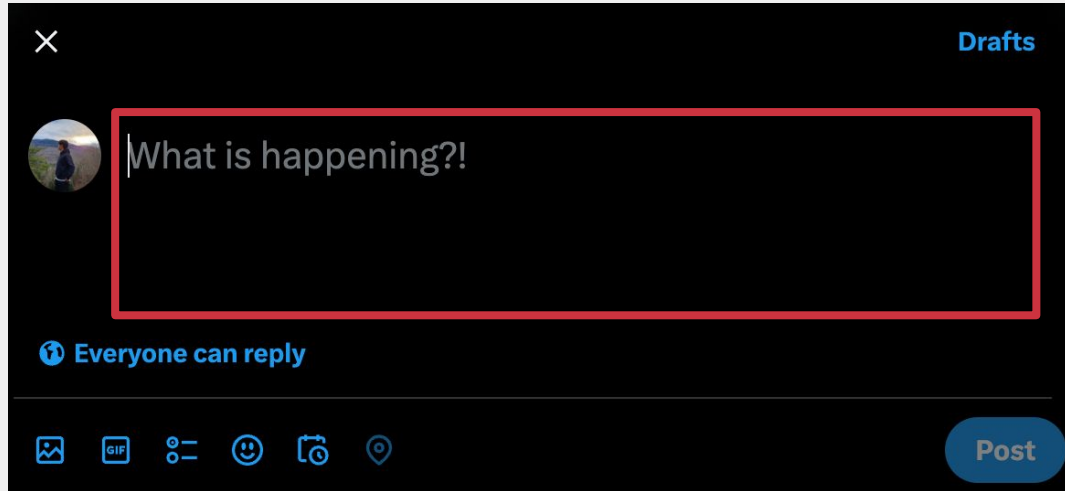
Identify structure

Profile picture of the person composing the tweet



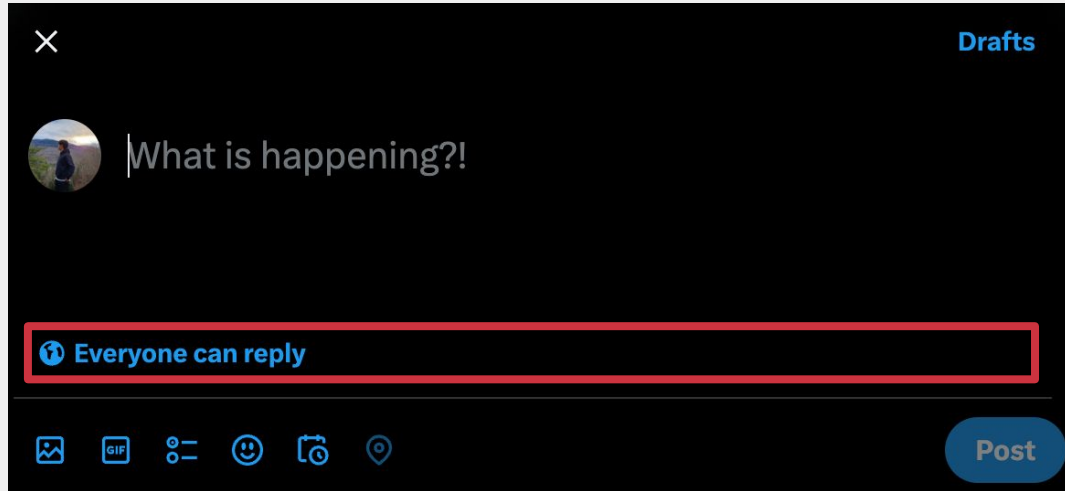
Identify structure

Text input area for writing tweet content



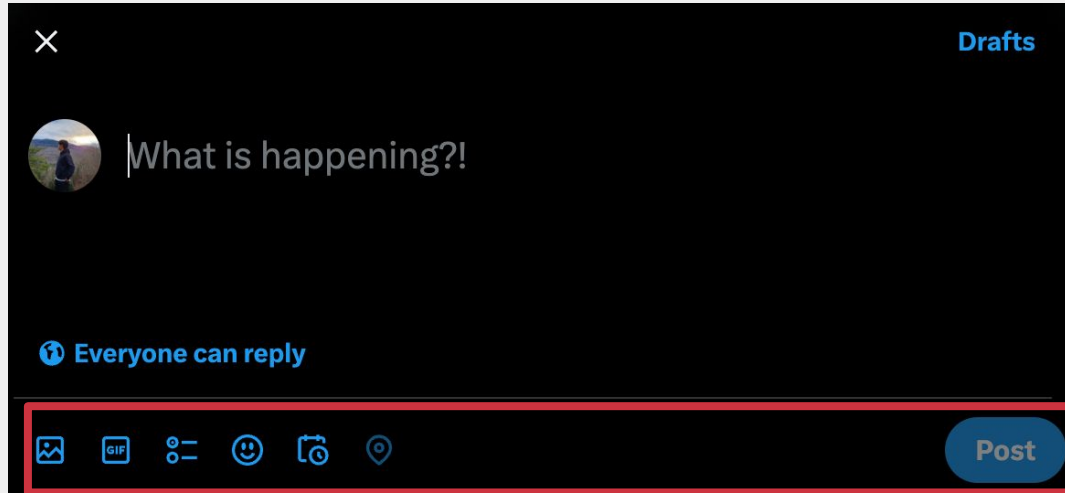
Identify structure

Text indicator for who can reply to the tweet

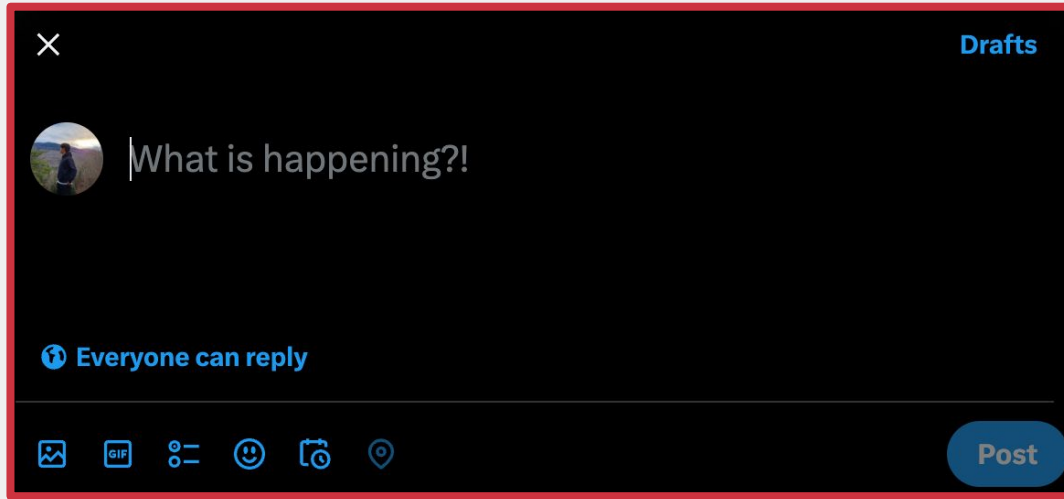


Identify structure

Bar at the bottom containing controls for the tweet



Build container



```
<section class="bg-gray-900 w-full p-4">  
  
</section>
```

Top bar



What should go inside the class for the **div** element in order to tell the browser to maximize space between “Close” and “Drafts”?

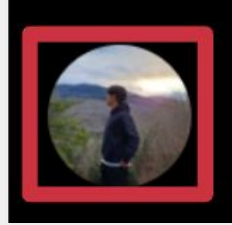
```
<section class="bg-gray-900 w-full p-4">
  <div class="">
    <p class="text-white">Close</p>
    <p class="text-blue-400 font-semibold">Drafts</p>
  </div>
</section>
```

Top bar



```
<section class="bg-gray-900 w-full p-4">
  <div class="flex justify-between">
    <p class="text-white">Close</p>
    <p class="text-blue-400 font-semibold">Drafts</p>
  </div>
</section>
```

Profile Picture + Text Content

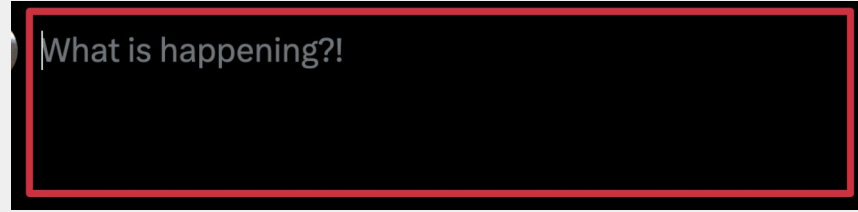


```

<!-- Container for PFP + text -->
<div class="flex gap-4 py-8 h-48">
  <!-- Profile Picture -->
  <div class="w-16 h-16 aspect-square bg-blue-400 rounded-full"></div>
</div>

```

Profile Picture + Text Content



```

<!-- Container for PFP + text -->
<div class="flex gap-4 py-8 h-48">
  <!-- Profile Picture -->
  <div class="w-16 h-16 aspect-square bg-blue-400 rounded-full"></div>
  <!-- Text input area -->
  <textarea
    class="w-full mt-4 bg-transparent text-white text-2xl focus:outline-none"
    placeholder="What is happening?!">
  </textarea>
</div>

```

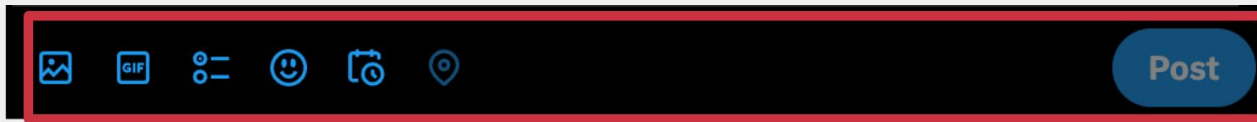
Who can reply indicator



 Everyone can reply

```
• • •  
  
<!-- Reply indicator -->  
<p class="font-bold text-blue-400">Everyone can reply</p>  
<!-- Horizontal line -->  
<hr class="border-gray-500 my-4"/>
```

Tweet Controls + Post Button Container



```

<!-- Tweet Controls + Post Button Container -->
<nav class="flex items-center justify-between">

</nav>

```

Tweet Controls Container



```

<!-- Tweet Controls + Post Button Container -->
<nav class="flex items-center justify-between">
  <!-- Tweet Controls Container -->
  <div class="flex gap-4 text-blue-400">
    <p>Image</p>
    <p>GIF</p>
    <p>Poll</p>
    <p>Emoji</p>
    <p>Location</p>
  </div>
</nav>

```


Post Button



```
...
<!-- Tweet Controls + Post Button Container -->
<nav class="flex items-center justify-between">
  ...
  <!-- Post Button -->
  <button class="text-white bg-blue-400 px-4 py-2 rounded-full font-
bold">Post</button>
</nav>
```

```

<section class="bg-gray-900 w-full p-4">
  <div class="flex justify-between">
    <p class="text-white">Close</p>
    <p class="text-blue-400 font-semibold">Drafts</p>
  </div>

  <div class="flex gap-4 py-8 h-48">
    <div class="w-16 h-16 aspect-square bg-blue-400 rounded-full"></div>
    <textarea
      class="w-full mt-4 bg-transparent text-white text-2xl focus:outline-none"
      placeholder="What is happening?!">
    </textarea>
  </div>

  <p class="font-bold text-blue-400">Everyone can reply</p>

  <hr class="border-gray-500 my-4"/>

  <nav class="flex items-center justify-between">
    <div class="flex gap-4 text-blue-400">
      <p>Image</p>
      <p>GIF</p>
      <p>Poll</p>
      <p>Emoji</p>
      <p>Location</p>
    </div>
    <button class="text-white bg-blue-400 px-4 py-2 rounded-full font-
bold">Post</button>
  </nav>
</section>

```

Final Result + Code → weblab.is/tailwind

