Week 3

Goals

- [ ] Intro to CSS
- [ ] Building a demo page
- [ ] Implementing the `KudosCard` component

A Note About Project Versions and Installations...

I knwo we had some mix-ups and discrepencies last session when trying to create a project.

So, we need to do some cleaning to ensure there is no confusion in the future with project names, versions, etc.

Let's take a few minutes to delete all prior projects relating to Kudos (`blueprint_kudos`, `kudos-me`, etc.)

For the last time (fingers crossed) let's create a React project configured to use TypeScript.

In your **root** directory, run:

```
npx create-react-app blueprint_kudos --template typescript
```

After running, you may be asked a few questions. Simply respond yes to all of them.

Then, to open your project in VSCode and start your live development server, run:

```
cd blueprint_kudos
npm install
npm start
```

Part 1: CSS Basics

Last week you saw some live changes made to the browser, and hopefully gained some understanding of why we really need tools like Node and git.

This week, we will be getting hands-on with one of the main languages of our app: CSS.

What is CSS?

CSS, which stands for Cascading Style Sheets, is a style sheet language used for describing the presentation of a document written in a markup language like HTML.

Basically, it makes websites look nice.

HTML

You may have heard of HTML, but what does it actually do?

HTML is the **structure** of a webpage. It's like the skeleton or framework that holds everything together.

Here is some brief info about HTML before we begin diving into CSS. Feel free to check out some tutorials/videos on HTML ifn you want to learn more tha what is in these docs.

How HTML Works

HTML uses **tags** (words in angle brackets) to define different parts of a webpage:

```
<h1>This is a heading</h1>
<p>This is a paragraph</p>
<button>Click me</button>
```

Cheatsheet

Use this as a cheatsheet for whenever you forgot/don't know which tags to use.

- `<h1>`, `<h2>`, `<h3>` - Headings (big to small)
- `<p>` - Paragraphs
- `<div>` - Container/box
- `<button>` - Clickable buttons
- `<img>` - Images

- `<a>` - Links

And remember: **HTML is the foundation - everything else builds on top of it!**

A Useful Analogy

A classic analogy that I will probably continue using throughout the next few sessions is this:

Our app is like a house...

HTML is the walls.

TypeScript is the door.

CSS is the paint.

We need all three to build a house, and in our case, this app.

But, specifically, CSS controls:

- **Colors** - text, backgrounds, borders
- **Layout** - where elements appear on the page
- **Typography** - fonts, sizes, spacing
- **Visual effects** - shadows, animations, hover effects

Let's see how this works in practice.

Classes

You may have heard of classes before... but CSS classes are slightly different.

CSS classes are reusable **style** definitions that you can **apply to HTML elements.** They're like templates for styling that you can use over and over again.

We can think of CSS classes like labels or tags that you apply to HTML elements to style them.

Imagine you have a box of different colored stickers:

- Yellow stickers for "important" items
- Red stickers for "urgent" items
- Green stickers for "completed" items

You stick these labels on different objects, and everyone knows what each color means.

CSS classes work the same way. You "stick" a class name on an HTML (wall) element, and the browser knows how to style it with CSS (paint).

Selectors

Before we can style anything, we need to tell CSS what we want to style. This is where selectors come in.

The purpose of selectors is to **tell CSS which HTML elements we want to style**

Let's create a simple demo TypeScript file to see this in action.

First though, let's create a directory (folder) to store our components:

CREATE: `src/components`

Then,

CREATE: `src/components/CSSBasicsDemo.tsx`

```
// Creating the demo component
export function CSSBasicsDemo() {
  return (
    <div className="demo-container"> {/* 1. demo-container class*/}
      <h2>CSS Basics Demo</h2>
      <div className="box-model-demo">
        <h3>Box Model</h3>
        <div className="box-example"> {/* 2. box-example class */}
          <div className="content">Content</div>
        </div>
      </div>
    </div>
  );
}
export default CSSBasicsDemo;
```

Now, let me add the CSS file. Notice how I'm targeting these elements using class names:

CREATE: `src/components/CSSBasicsDemo.css`

```
/* I'm targeting the demo-container class */
.demo-container {
  max-width: 800px;
  margin: 0 auto;
  padding: 20px;
```

```
}

/* I'm targeting the box-example class */
.box-example {
  width: 200px;
  height: 100px;
  padding: 20px;
  margin: 20px;
  border: 3px solid #3b82f6;
  background-color: #f8fafc;
}
```

See how this works? I'm using class names - those are the words after 'className=' in the HTML - to target specific elements. This is the most common way to style things in React.

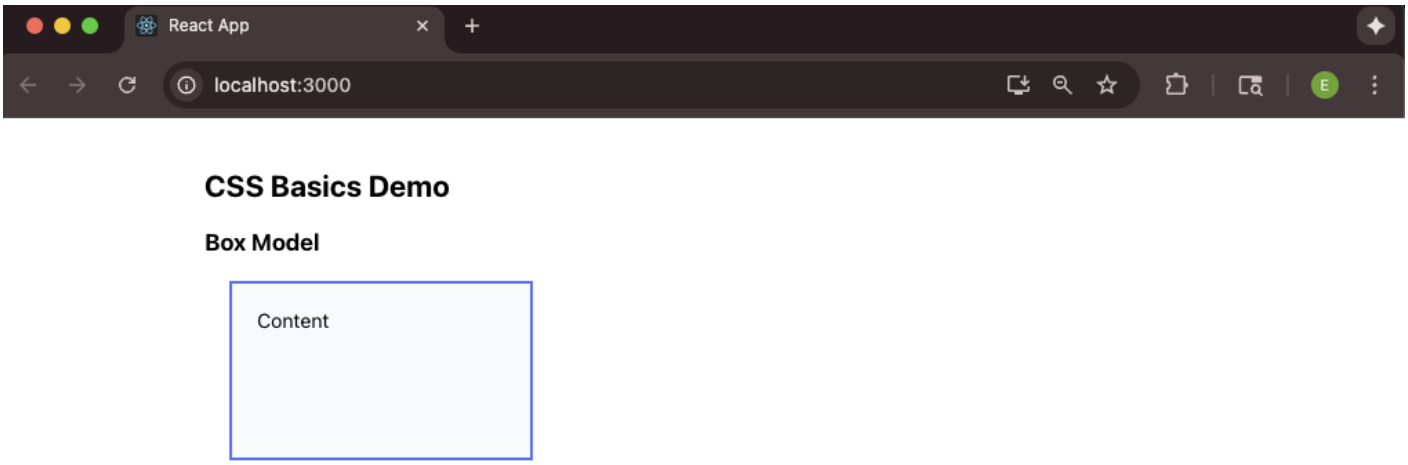
Recall: `src/App.tsx` is the main file in any React project, because it is where the main application structure is defined, and where each component is referenced.

So, let's update ours with our new components:

```
import React from 'react';
import { CSSBasicsDemo } from './components/CSSBasicsDemo'; // refernces .tsx demo file
import './components/CSSBasicsDemo.css'; // refernces .css demo file

function App() {
  return <CSSBasicsDemo />;
}

export default App;
```



If you're seeing a page that looks like this, then you've done everything correctly so far!

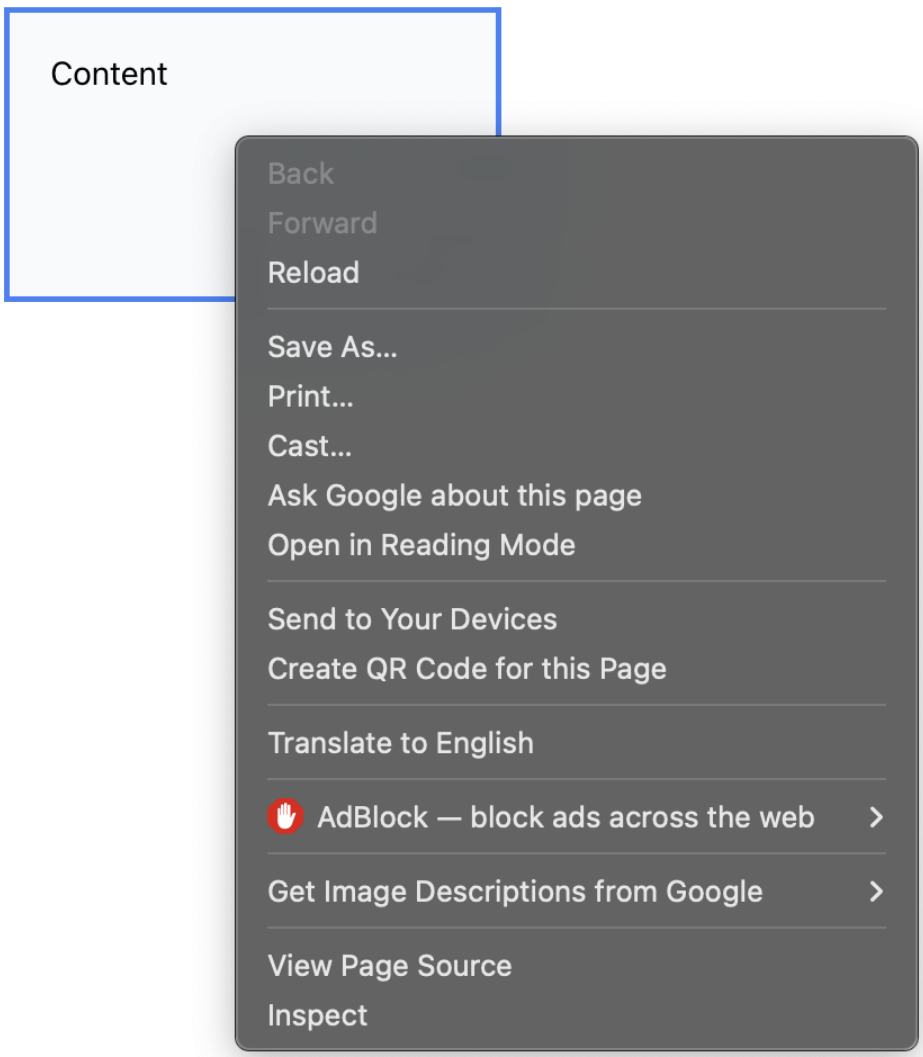
The Box Model

Note: This is a VERY important section, so be sure to ask any questions if you need to. The Box Model is absolutely crucial to understand.

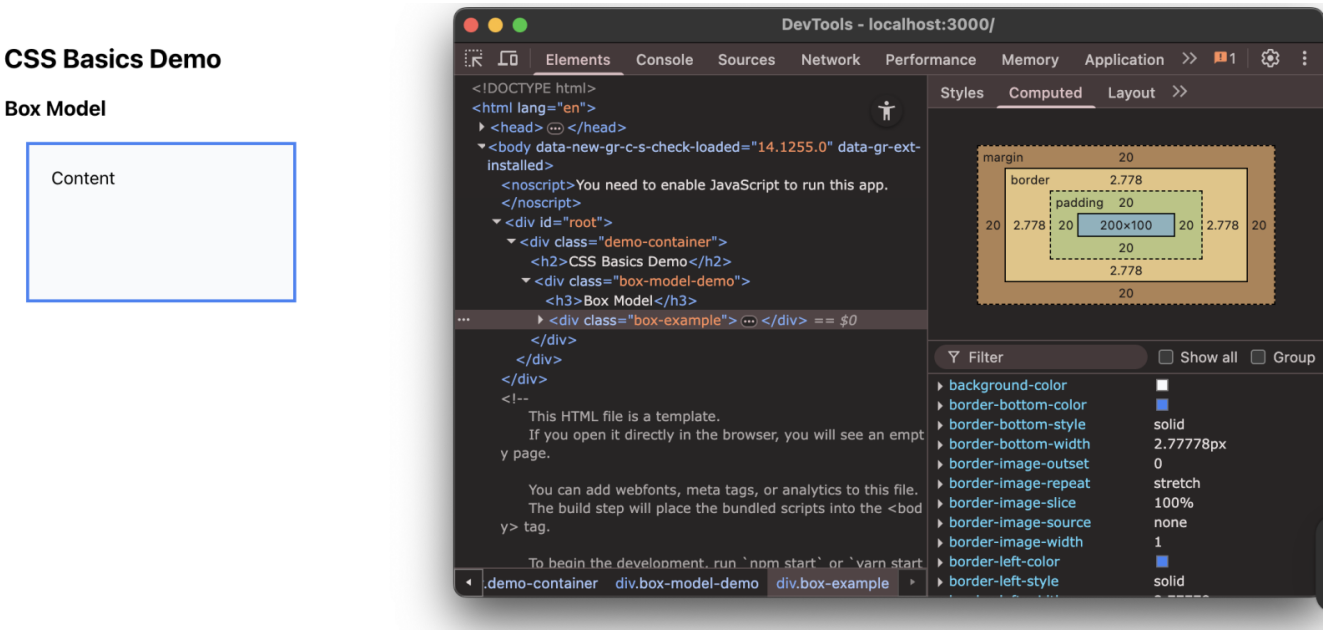
The Box Model describes that **every single element on a webpage is a box.**

To show you what I mean, let's use some developer tools (DevTools) made availible to us through `Inspect Element`.

To access your DevTools, right-click the object you want to examine (in our case, the blue box), and select `Inspect Element` from the dropdown.



You should now see a new window appear. It should look something like this:



Let me break down what's happening here:

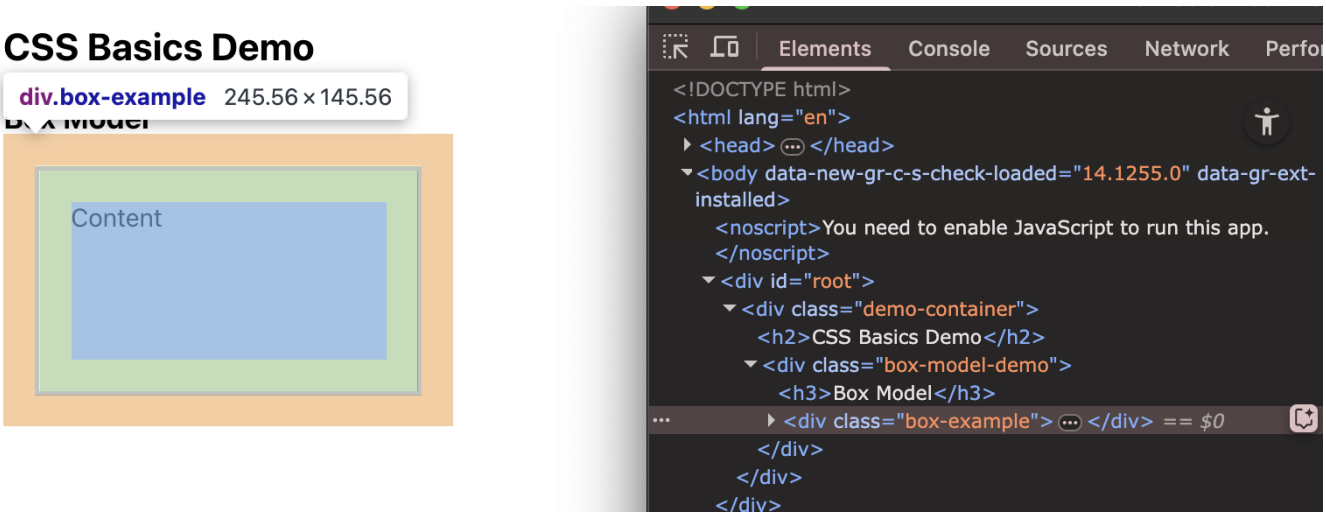
Each class is made up of properties which describe it in many ways.

In this example:

- The **content** is the actual text or image inside.
- **Padding** is the space inside the element, between the content and the border. That's the white space around the blue text.
- The **border** is the line around the element. That's the blue line you see.
- **Margin** is the space outside the element, between this element and other elements.

Let's see what heppns when we make a change to one of these properties...

First, hover over some of the `<div>`s so you can see exactly how your code is visualized on our webpage. Like this:



See how when my mouse is hovered over `".box-example"` it is highlighted on our webpage? This makes knwoing what our code does visually super clear.

We can see the properties of any element we want in the right panel that looks like this:

Hmm... some of those properties in the panel seem very familiar...

Remember how we gave our classes certain properties earlier in `src/components/CSSBasicsDemo.css`? We can now play around with their values to see their effect visually on our webpage.

Let's do some experimentation.

UPDATE: `src/components/CSSBasicsDemo.css`

```
.box-example {
  padding: 40px; /* Changed from 20px */
}
```

See how the content moved further from the border? That's padding.

Now let us change the margin...

```
.box-example {
  margin: 40px; /* Changed from 20px */
}
```

See how the entire box moved away from other elements? That's margin.

This is so important because when you're building layouts, you need to understand how these four things work together.

Most layout problems you'll face when first starting off come from not understanding the box model.

To close off this section, experimentsome more with the box-example properties and see the changes visually in your live server.

```
.demo-container {
  max-width: 800px;
  margin: 0 auto;
  padding: 20px;
}

/* Play around here and see the visual changes */
.box-example {
  width: 200px;
  height: 200px; /* 100 -> 200 */
  padding: 80px; /* 20 -> 40 -> 80 */
  margin: 60px; /* 20 -> 60 */
  border: 3px solid #3b82f6;
  background-color: #f8fafc;
}
```

Part 2: Building Our Kudos Card Component

Step 1: Planning Our Component

In software development, the first step is always to plan.

Since we already have an idea of what our end product looks like, and we now understand how our app is structured, **we can actually do this step without any pre-existing visualization!**

But, because we have an end product already, let's look back to it and create a quick list of components we will need to build.

[look at final Kudos example from Week 1]

One clear one that we should consider is the card layout - especially since it is the heart of our app.

Let's think about what this component needs:

- A recipient's name
- The kudos message
- Who gave the kudos
- When it was given
- What type it is (kudos or feedback)

This is a great example of a real-world component. It's not too simple, but not too complex either.

Let me create the basic structure first...

Step 2: Structuring the Component

Let's create a new file for our kudos card component.

Note: In React, it's common to have one component per file.

CREATE: `src/components/KudosCard.tsx`

```
import React from 'react';

interface KudosCardProps {
  recipient: string;
  message: string;
  giver: string;
  type: 'kudos' | 'feedback';
  date: string;
}

export function KudosCard({ recipient, message, giver, type, date }: KudosCardProps) {
  return (
    <div className="kudos-card">
      <div className="kudos-header">
        <h3 className="recipient-name">{recipient}</h3>
        <span className="kudos-type">{type}</span>
      </div>
```

```
      <p className="kudos-message">{message}</p>

      <div className="kudos-footer">
        <span className="giver-name">From: {giver}</span>
        <span className="kudos-date">{date}</span>
      </div>
    </div>
  );
}
```

Let me walk through what I'm doing here:

First, I'm defining the TypeScript interface. This tells us exactly what props this component expects. This is really helpful because if someone tries to use this component incorrectly, TypeScript will catch the error.

The component structure is semantic - I'm using meaningful HTML elements. The header contains the recipient name and type, the message is in a paragraph, and the footer has the giver and date.

Notice how I'm using className for all the styling. **This is how we connect our HTML to our CSS.**

Now let me add the CSS...

Step 3: Applying CSS Styles

I'm creating a separate CSS file for this component. This keeps things organized and makes the component reusable.

CREATE: `src/components/KudosCard.css`

```
/* src/components/KudosCard.css */

.kudos-card {

  /* Box Model */
  width: 100%; /* Full width of container */
  max-width: 400px; /* Maximum width */
  padding: 24px; /* Space inside the card */
  margin: 16px 0; /* Space above and below */
  border: 1px solid #e2e8f0; /* Light gray border */
  border-radius: 12px; /* Rounded corners */
```

```
  /* Layout */
  display: flex; /* Make it a flex container */
  flex-direction: column; /* Stack children vertically */
  gap: 16px; /* Space between children */

  /* Colors */
  background-color: #ffffff; /* White background */
  color: #1e293b; /* Dark gray text */

  /* Typography */
  font-family: 'Inter', -apple-system, sans-serif; /* Font family */
  font-size: 14px; /* Base font size */
  line-height: 1.5; /* Line spacing */

  /* Visual Effects */
  box-shadow: 0 2px 4px rgba(0, 0, 0, 0.05); /* Subtle shadow */
  transition: all 0.2s ease; /* Smooth transitions */
}
```

Let me break this down:

I'm using the box model we learned about: padding for internal space, margin for external space, border for the outline.

`max-width: 400px` means the card won't get wider than 400 pixels, but it can be smaller on mobile devices.

`display: flex` with `flex-direction: column` stacks the elements vertically.

`gap: 16px` puts consistent space between all the child elements.

I know we can really strecth our skills though, so let's keep adding props...

For the hover effect from before:

```
  /* ...prior code... */

.kudos-card:hover {
  transform: translateY(-2px);
  box-shadow: 0 8px 25px rgba(0, 0, 0, 0.1);
  border-color: #cbd5e1;
```

```
}
```

This creates a nice interactive effect. When you hover over the card, it should move up slightly and get a stronger shadow.

Now let me style the header...

```
/* ...prior code... */

.kudos-header {
  display: flex;
  justify-content: space-between;
  align-items: center;
  margin-bottom: 8px;
}

.recipient-name {
  font-size: 18px;
  font-weight: 600;
  color: #0f172a;
  margin: 0;
}

.kudos-type {
  padding: 4px 12px;
  border-radius: 20px;
  font-size: 12px;
  font-weight: 500;
  text-transform: uppercase;
  letter-spacing: 0.5px;
}
```

The header uses flexbox to put the name on the left and the type badge on the right.

The type badge has rounded corners (`border-radius: 20px`) and is styled like a pill.

Let me add different colors for different types...

```
/* ...prior code... */

.kudos-type.kudos {
```

```
  background-color: #dbeafe;
  color: #1e40af;
}

.kudos-type.feedback {
  background-color: #f3f4f6;
  color: #374151;
}
```

This uses CSS class combination. When an element has both `kudos-type` and `kudos` classes, it gets the blue styling. When it has `kudos-type` and `feedback`, it gets the gray styling.

Now let me style the message and footer...

```
/* ...prior code... */

.kudos-message {
  font-size: 15px;
  line-height: 1.6;
  color: #475569;
  margin: 0;
  flex-grow: 1;
}

.kudos-footer {
  display: flex;
  justify-content: space-between;
  align-items: center;
  padding-top: 12px;
  border-top: 1px solid #f1f5f9;
  font-size: 12px;
  color: #64748b;
}
```

The message uses `flex-grow: 1` which means it will take up all available space, pushing the footer to the bottom.

The footer has a subtle border on top to separate it from the message.

All together now...

```
/* src/components/KudosCard.css */

.kudos-card {

  /* Box Model */
  width: 100%; /* Full width of container */
  max-width: 400px; /* Maximum width */
  padding: 24px; /* Space inside the card */
  margin: 16px 0; /* Space above and below */
  border: 1px solid #e2e8f0; /* Light gray border */
  border-radius: 12px; /* Rounded corners */

  /* Layout */
  display: flex; /* Make it a flex container */
  flex-direction: column; /* Stack children vertically */
  gap: 16px; /* Space between children */

  /* Colors */
  background-color: #ffffff; /* White background */
  color: #1e293b; /* Dark gray text */

  /* Typography */
  font-family: 'Inter', -apple-system, sans-serif; /* Font family */
  font-size: 14px; /* Base font size */
  line-height: 1.5; /* Line spacing */

  /* Visual Effects */
  box-shadow: 0 2px 4px rgba(0, 0, 0, 0.05); /* Subtle shadow */
  transition: all 0.2s ease; /* Smooth transitions */

}

.kudos-card:hover {

  transform: translateY(-2px); /* Move up slightly on hover */
  box-shadow: 0 8px 25px rgba(0, 0, 0, 0.1); /* Stronger shadow on hover */
  border-color: #cbd5e1; /* Lighter border on hover */
}
```

```
.kudos-header {
  display: flex; /* Make it a flex container */
  justify-content: space-between; /* Space items apart */
  align-items: center; /* Center items vertically */
  margin-bottom: 8px; /* Space below header */
}

.recipient-name {
  font-size: 18px; /* Larger font size */
  font-weight: 600; /* Bold text */
  color: #0f172a; /* Very dark gray */
  margin: 0; /* Remove default margin */
}

.kudos-type {
  padding: 4px 12px; /* Space inside badge */
  border-radius: 20px; /* Pill-shaped */
  font-size: 12px; /* Small font size */
  font-weight: 500; /* Medium bold */
  text-transform: uppercase; /* All caps */
  letter-spacing: 0.5px; /* Space between letters */
}


.kudos-type.kudos {
  background-color: #dbeafe; /* Light blue background */
  color: #1e40af; /* Dark blue text */
}


.kudos-type.feedback {
  background-color: #f3f4f6; /* Light gray background */
  color: #374151; /* Dark gray text */
}


.kudos-message {
  font-size: 15px; /* Slightly larger font */
  line-height: 1.6; /* Good line spacing */
  color: #475569; /* Medium gray text */
```

```
  margin: 0; /* Remove default margin */
  flex-grow: 1; /* Take up available space */
}


.kudos-footer {
  display: flex; /* Make it a flex container */
  justify-content: space-between; /* Space items apart */
  align-items: center; /* Center items vertically */
  padding-top: 12px; /* Space above footer */
  border-top: 1px solid #f1f5f9; /* Light border on top */
  font-size: 12px; /* Small font size */
  color: #64748b; /* Light gray text */
}

.giver-name {
  font-weight: 500; /* Medium bold */
}


.kudos-date {
  color: #94a3b8; /* Very light gray */
}
```

Finally, let's create a demo page to show this in action...

Step 4: Creating the Demo Page

We need to update the App component to show our kudos cards...

```
// src/App.tsx
import React from 'react';
import { KudosCard } from './components/KudosCard';
import './components/KudosCard.css';

function App() {
  const sampleKudos = [
    {
      recipient: "Jane Smith",
      message: "Amazing work on the authentication refactor!  Your clean code and thorough
```

```
testing saved us so much time.",
      giver: "John Doe",
      type: "kudos" as const,
      date: "Mar 22, 2024"
    },
    {
      recipient: "Mike Johnson",
      message: "Thanks for staying late to help debug the payment integration. Your
persistence really paid off!",
      giver: "Sarah Wilson",
      type: "kudos" as const,
      date: "Mar 21, 2024"
    },
    {
      recipient: "Alex Chen",
      message: "Great job presenting the technical architecture to stakeholders. You explained
everything so clearly!",
      giver: "Emily Davis",
      type: "feedback" as const,
      date: "Mar 20, 2024"
    }
  ];

  return (
    <div className="app-container">
      <div className="app-content">
        <h1 className="app-title">
          - Kudos Cards Demo -
        </h1>

        <div className="cards-grid">
          {sampleKudos.map((kudos, index) => (
            <KudosCard
              key={index}
              recipient={kudos.recipient}
              message={kudos.message}
              giver={kudos.giver}
              type={kudos.type}
              date={kudos.date}
            />
```

```
          ))}
        </div>
      </div>
    </div>
  );
}

export default App;
```

Let me add some CSS for the app layout...

```
/* Add to App.css or create App.css */
.app-container {
  min-height: 100vh;
  background-color: #f8fafc;
  padding: 32px 0;
}

.app-content {
  max-width: 1200px;
  margin: 0 auto;
  padding: 0 16px;
}

.app-title {
  font-size: 48px;
  font-weight: 700;
  text-align: center;
  color: #0f172a;
  margin-bottom: 32px;
}

.cards-grid {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(350px, 1fr));
  gap: 24px;
}

@media (max-width: 768px) {
  .cards-grid {
```

```
    grid-template-columns: 1fr;
  }
}
```

Look at that! We have working kudos cards!!!

Notice how they respond to hover, how the different types have different colors, how the layout is clean and structured!

Revision #19
Created 1 October 2025 06:25:35 by Emilio Cardillo Schrader
Updated 28 October 2025 00:35:36 by Emilio Cardillo Schrader