2.4.10 Make the Grid Match the Mock-Up

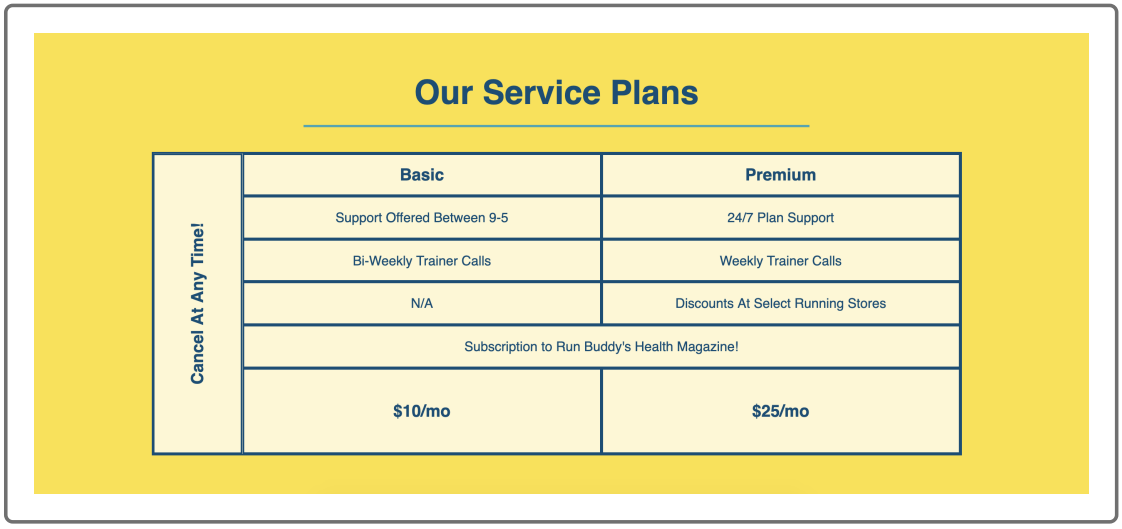
Let's use our new CSS Grid skills to implement the service plan table layout! We'll start by planning the build process.

First we'll take another look at the mockup, then add the HTML. We'll add CSS Grid properties for the grid container and grid items. Then we'll style the section and section content to match the mock-up.

IMPORTANT

> Remember that this section should be located after the "Meet the Trainers" section and before "Reach Out."

Here's the mock-up:



Add the HTML

We'll use the `<section>` tags with the `service-plans` id and `services` class that we added in the beginning of this lesson to contain the service plan table.

Let's replace the example `<div class="service-grid-container">` and corresponding nested `<div>`s and add an `h2` element the "Our Service Plans" heading as follows:

```
<section id="service-plans" class="services">

  <h2 class="section-title secondary-border">
    Our Service Plans
  </h2>

</section>
```

Now we'll add two more `<div>`s. We'll add a `<div>` wrapper under the heading that will be used later in the lesson to center the service plan table in the viewport. We'll also add the grid container by using a `<div>` with the class `service-grid-container`:

```
<div class="service-grid-wrapper">
  <div class="service-grid-container">

  </div> <!-- div service-grid-container -->
</div> <!-- div service-grid-wrapper -->
```

This isn't a lot to look at in the browser just yet, but we've laid the groundwork for the grid.

The next step will be to add the grid item elements. We'll do this by simply adding a direct child element, corresponding content, and box label for each box. Let's also add the `service-grid-item` class to each of these `<div>` elements for our grid properties. Just as we would do with a table, we'll start with the headers and then proceed with the body of the table.

Take your time and approach this step by step.

PAUSE

Is the order in which you place the HTML important?

Using the `grid-row` and `grid-column` properties, we can place our grid item anywhere we like explicitly. If we're relying on CSS Grid to place them implicitly for us, then the order of the HTML matters.

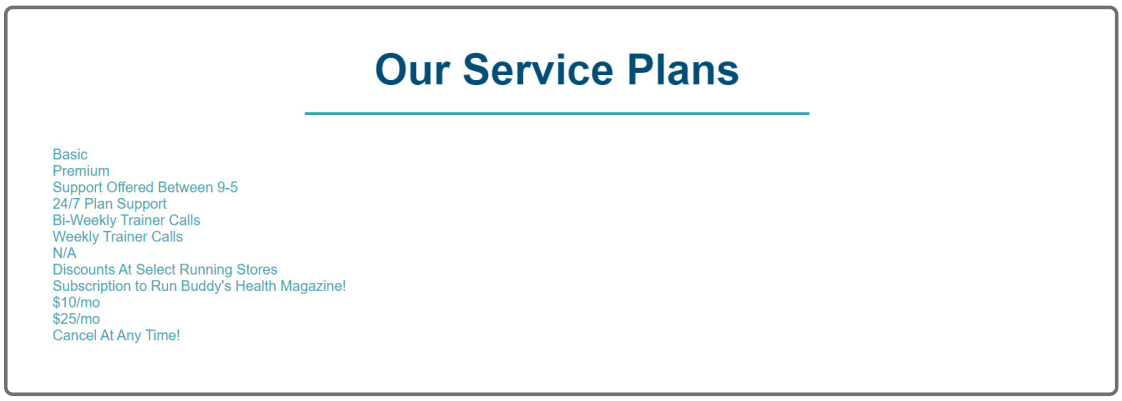
Hide Answer

The HTML inside your grid container should now look something like this:

```
<!-- Headers -->
<div class="service-grid-item">
  Basic
```

```
</div>
<div class="service-grid-item">
  Premium
</div>

<!-- Body -->
<div class="service-grid-item">
  Support Offered Between 9-5
</div>

<div class="service-grid-item">
  24/7 Plan Support
</div>

<div class="service-grid-item">
  Bi-Weekly Trainer Calls
</div>

<div class="service-grid-item">
  Weekly Trainer Calls
</div>

<div class="service-grid-item">
  N/A
</div>

<div class="service-grid-item">
  Discounts At Select Running Stores
</div>

<div class="service-grid-item">
  Subscription to Run Buddy's Health Magazine!
</div>

<div class="service-grid-item">
  $10/mo
</div>

<div class="service-grid-item">
  $25/mo
</div>

<div class="service-grid-item">
  Cancel At Any Time!
</div>
```

Now that we have our HTML in place, let's look at our work in the browser:

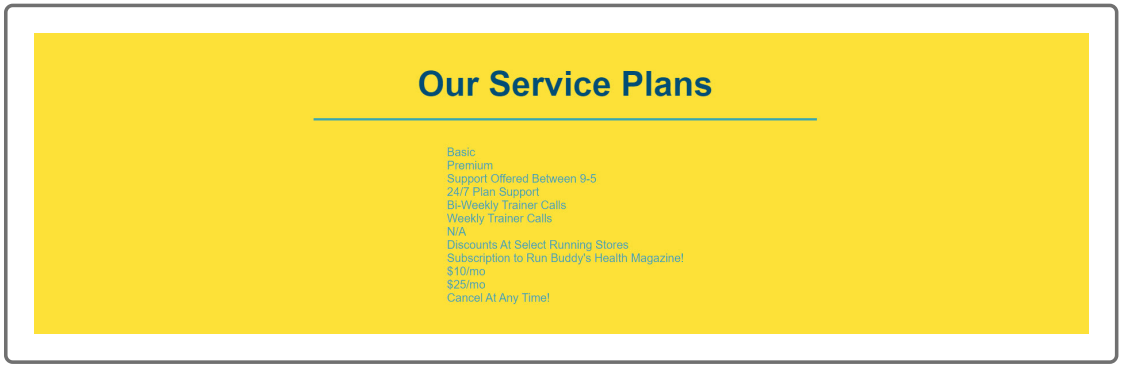


The design team wouldn't be thrilled with this look, but we can use our new CSS Grid skills to convert this unreadable list into an attractive chart. Let's get started!

Build the Grid

Let's proceed by defining the grid in the grid container. We'll set the background color and center the grid using flexbox:

```
/* SERVICE STYLES BEGIN */
.services {
  background: #fce138;
}

.service-grid-wrapper {
  display: flex;
  width: 100%;
  justify-content: center;
}
```



Now let's add CSS Grid by declaring it in the grid container.

PAUSE

How many columns and rows will our service table require?

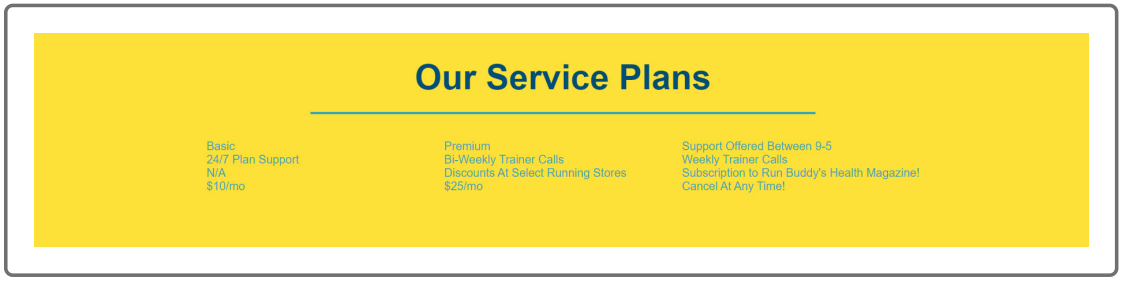
There are three columns and six rows.

Hide Answer

Let's add these columns and rows using the `grid-template-columns` and `grid-template-rows` properties and the `fr` value:

```
.service-grid-container {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  grid-template-rows: repeat(6, 1fr);
}
```

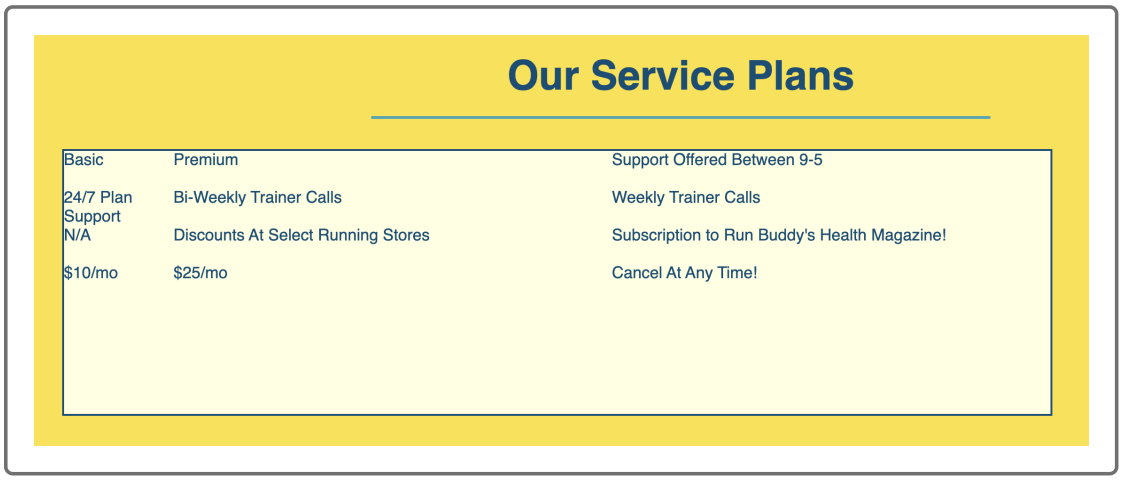
Let's save and render in the browser to view the state of our current grid:



Now add a few simple styles to the grid container to make the size more manageable and the background color a little easier on the eyes. Also add a border color and font color, and increase the font size:

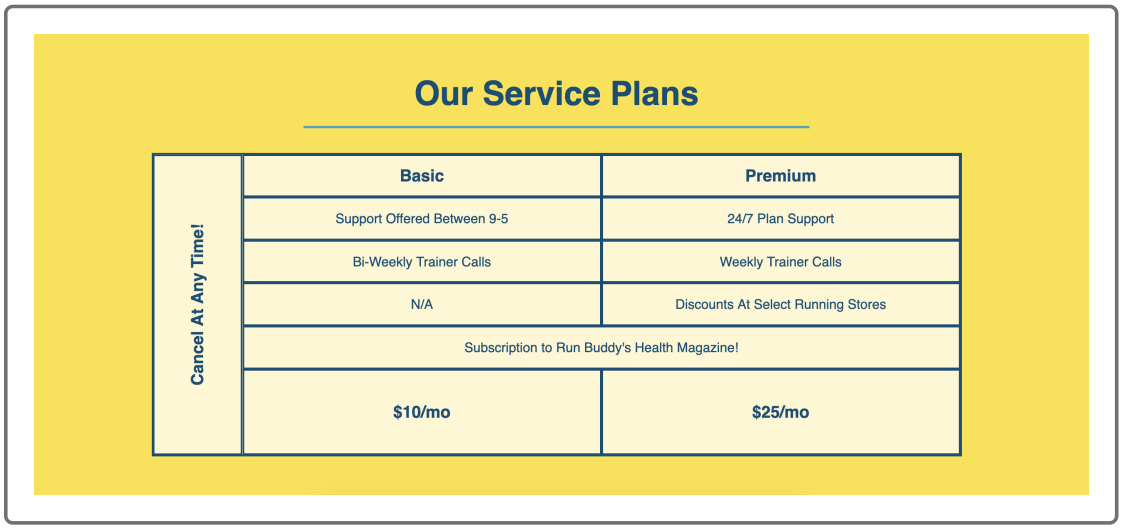
```
.service-grid-container {
  background: lightyellow;
  width: 80%;
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  grid-template-rows: repeat(6, 1fr);
  border: 2px solid #024e76;
  color: #024e76;
  font-size: 18px;
}
```

Now let's save and render our work to see the following:

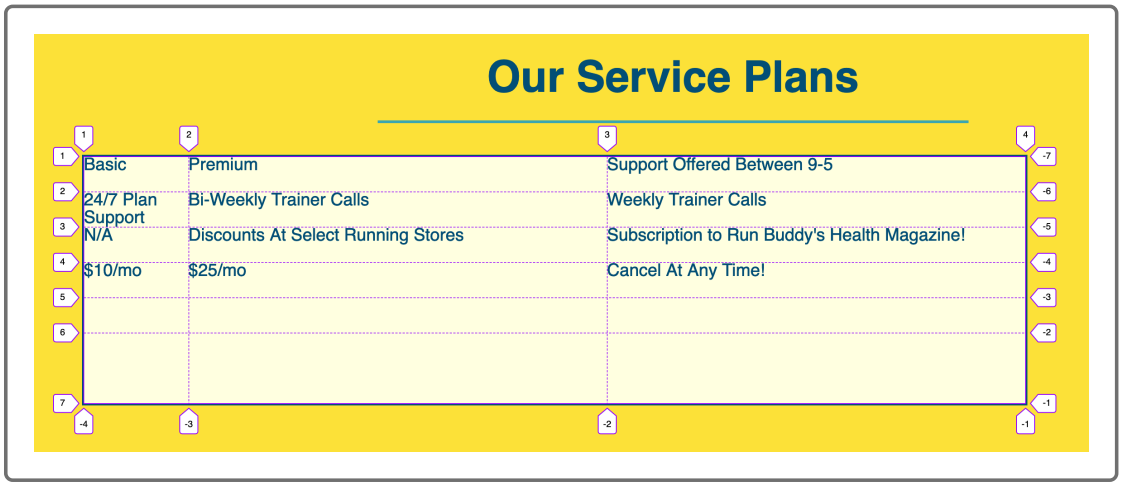


The grid is coming along nicely, but the grid items aren't in the correct order or format. Let's first try to fix the dimensions of the grid by changing the size of the columns and rows to reflect the mock-up.

Let's take another look at the mock-up:



With a grid overlay, we can see the relative dimensions of our grid columns and rows:



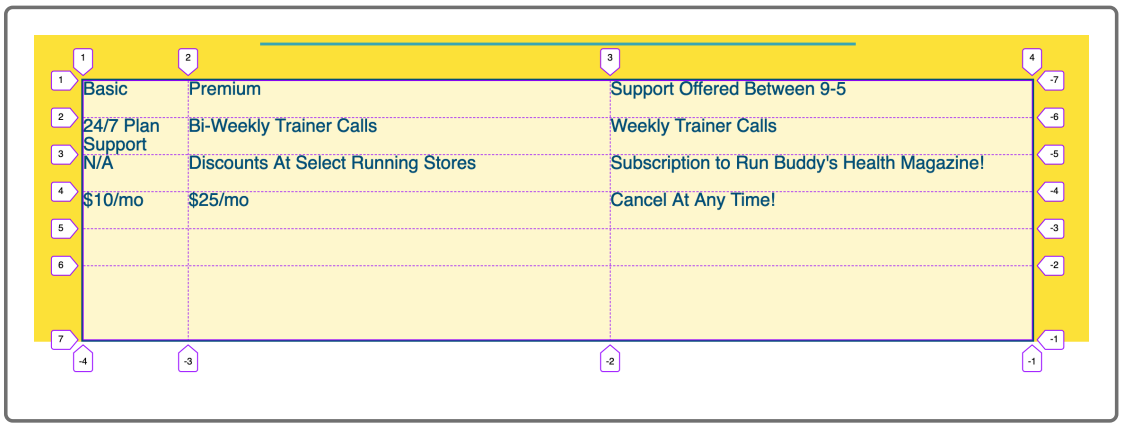
We can see that the first column should actually be closer to 1/4 of the size of the other columns. The bottom row is also double the height of the other rows. So how do we manipulate the size of the rows and columns in

the grid? We can designate each column or row by declaring the size in the property value.

Update the grid values to look like this now:

```
.service-grid-container {
  background: lightyellow;
  width: 80%;
  display: grid;
  /* repeat(iterator, size) */
  grid-template-columns: 1fr repeat(2, 4fr);
  grid-template-rows: repeat(5, 1fr) 2fr;
  border: 2px solid #024e76;
  color:  #024e76;
  font-size: 1.2rem;
}
```

Here's the grid with the rows and columns in their correct proportions. The first column is 1/4 the size of the other two columns and the bottom row is twice the height of the other rows:



Now that our basic formatting of the grid is done, its time to style our grid items.

Style the Grid Items

Good work! We're almost finished. In this next step, we'll use the `.service-grid-item` class selector to give each grid item some styling, including some padding, a border, and center positioning using flexbox.

Add the following to the style sheet:

```
.service-grid-item {
  padding: 15px 0;
  border: 2px solid #024e76;
  display: flex;
  align-items: center;
  justify-content: center;
  text-align: center;
}
```

Note that the `align-items` and `justify-content` are flexbox properties used to center the content in the grid item. This can be confusing because the grid properties look so similar. Keep in mind that the alignment properties for grid on the grid-item scope uses the `self` identifier, such as `align-self` or `justify-self`. Having the `display: flex;` declaration is also a key indicator that a flexbox property will be applied in this case. To center the text or content in the grid item, we must treat the grid item as a flexbox container and the content or text as the child element that will be centered.

Let's save and render this in the browser to see if we're getting a desired effect:

This looks very close to the mock-up except the positioning needs some work.

Position the Grid Items

To position the grid items correctly, we'll use the grid lines available in the Firefox inspector window with the Overlay Grid option.

Instead of assigning a class to every grid item individually, we'll use a column placement method, identifying each column and allowing the `grid-auto-flow` property to fill in the remaining available grid cells with the content available.

First, let's identify the three columns:

- Column one will contain the "Cancel At Any Time!" grid item.
- Column two will contain the Basic Service Plan grid items.
- Column three will contain the Premium Service Plan grid items.

Knowing this, let's use the mock-up and place our class attribute for cancel, basic, and premium with the grid items. For now, let's ignore the

"Subscription to Run Buddy's Health Magazine" grid item—we'll come back to that later.

Your HTML should now look like this:

```
<!-- Headers -->
<div class="service-grid-item basic">
  Basic
</div>
<div class="service-grid-item premium">
  Premium
</div>
<!-- Body -->
<div class="service-grid-item basic">
  Support Offered Between 9-5
</div>

<div class="service-grid-item premium">
  24/7 Plan Support
</div>

<div class="service-grid-item basic">
  Bi-Weekly Trainer Calls
</div>

<div class="service-grid-item premium">
  Weekly Trainer Calls
</div>

<div class="service-grid-item basic">
  N/A
</div>

<div class="service-grid-item premium">
  Discounts At Select Running Stores
</div>

<div class="service-grid-item">
  Subscription to Run Buddy's Health Magazine!
</div>

<div class="service-grid-item grid-price basic">
  $10/mo
```

```
</div>

<div class="service-grid-item grid-price premium">
  $25/mo
</div>

<div class="service-grid-item cancel">
  Cancel At Any Time!
</div>
```

Now let's add CSS rules using these class selectors to designate where the grid items will go by declaring each column in the grid:

```
/* specific match */
.service-grid-item.basic {
  grid-column: 2 / span 1;
}
```

Notice that this CSS selector has different syntax.

PAUSE

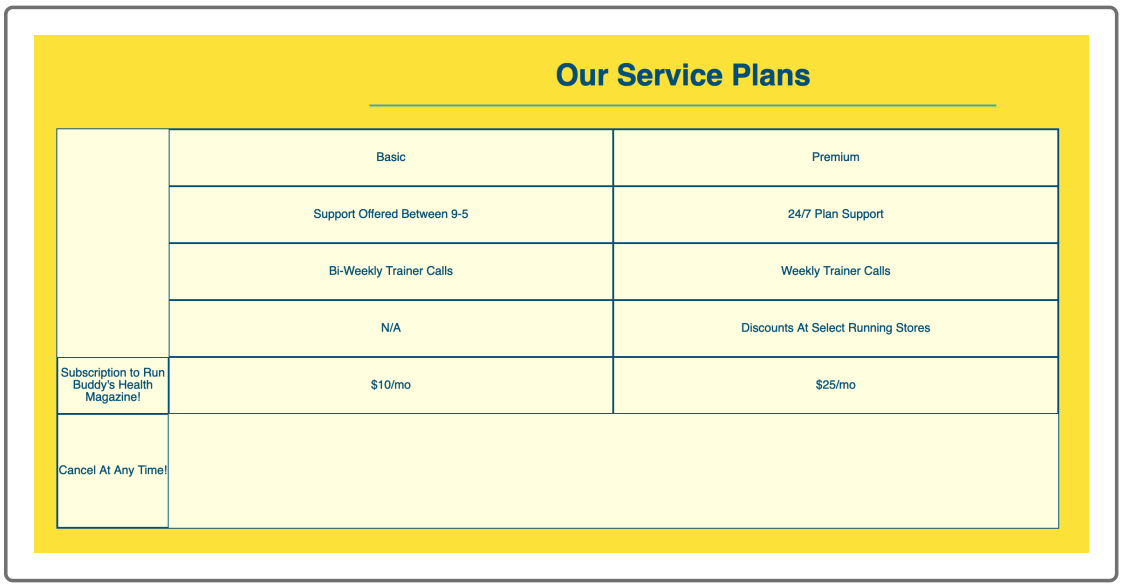
How is this selector, `.service-grid-item.basic`, functioning as it seems to have two different classes?

This class selector targets the HTML elements that have both classes. In this case, it's a great example of selecting a subset of a larger collection of elements.

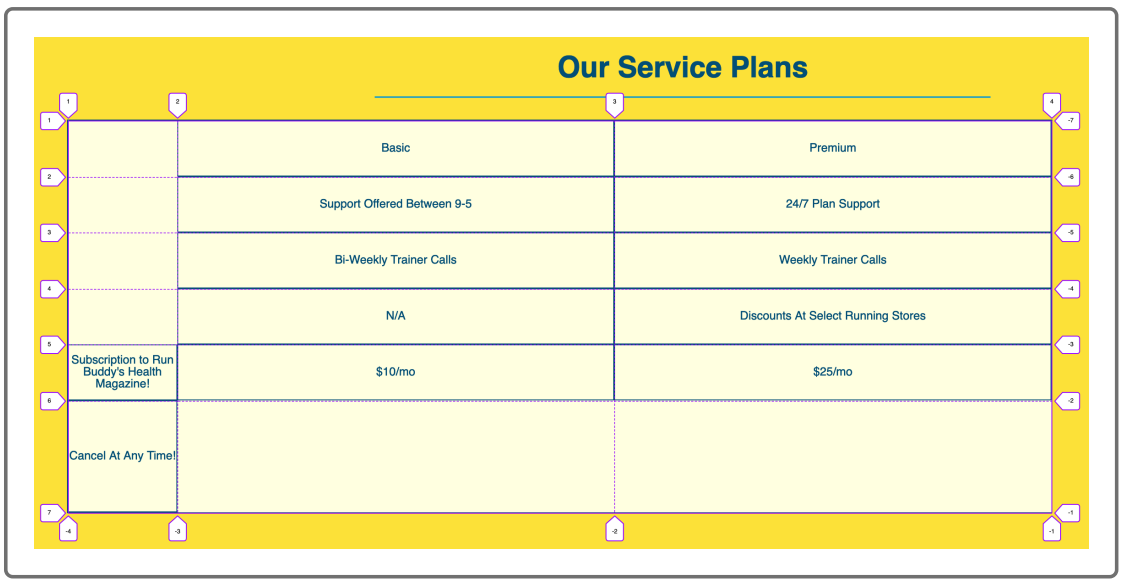
Hide Answer

In this first rule, we designated that the `basic` column will be located after the second grid line and span one grid cell. That makes it the second

column in the grid, as you can see here:



With the grid overlay, the grid looks like this:



Notice how the grid items with the class `basic` filled the designated column based on their order in the HTML. The premium column is also nearly done even though we didn't declare any specific rule.

PAUSE

What is the grid property that allowed the premium column to fill automatically?

The `grid-auto-flow` property filled empty space with grid items in accordance to their HTML order in the HTML file. By default, this property is set to a `row` value.

Hide Answer

Let's add a class to the grid item for the "Subscription to Run Buddy's Health Magazine" content so we can move its position:

```
<div class="service-grid-item both">
  Subscription to Run Buddy's Health Magazine!
</div>
```

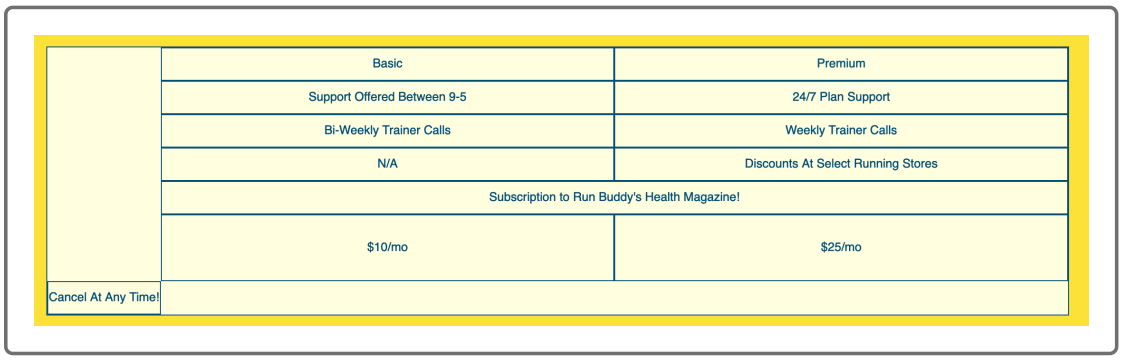
Create a rule for the `.both` class selector to span the grid item over the second and third columns according to the mock-up.

It should look like this:

```
.service-grid-item.both {
  grid-column: 2 / span 2;
}
```

In this rule, we designated that the grid item with the `.both` class attribute will span from grid line 2 for two grid cells and end on grid line 4.

Let's save and refresh the browser to if the results are satisfactory:



Excellent work! Our service table is coming along great.

PAUSE

How did the subscription grid item automatically position itself into the fifth row when only the column declaration was applied?

Before any grid position was designated to the subscription grid item, this item was removed from the grid flow and placed at the bottom. Once a position was assigned to this grid item, this item was returned to the natural grid flow and was placed by the `grid-auto-flow` property according to its order in the HTML.

Hide Answer

In the final step of this section, we'll position the "Cancel At Any Time!" grid item to span all the rows in the first column. Let's proceed by using a grid item property similar to one we used for the "Subscription" grid item except to span multiple rows, not columns, this time.

Your code should look like this:

```
.service-grid-item.cancel {
  grid-row: 1 / -1;
}
```

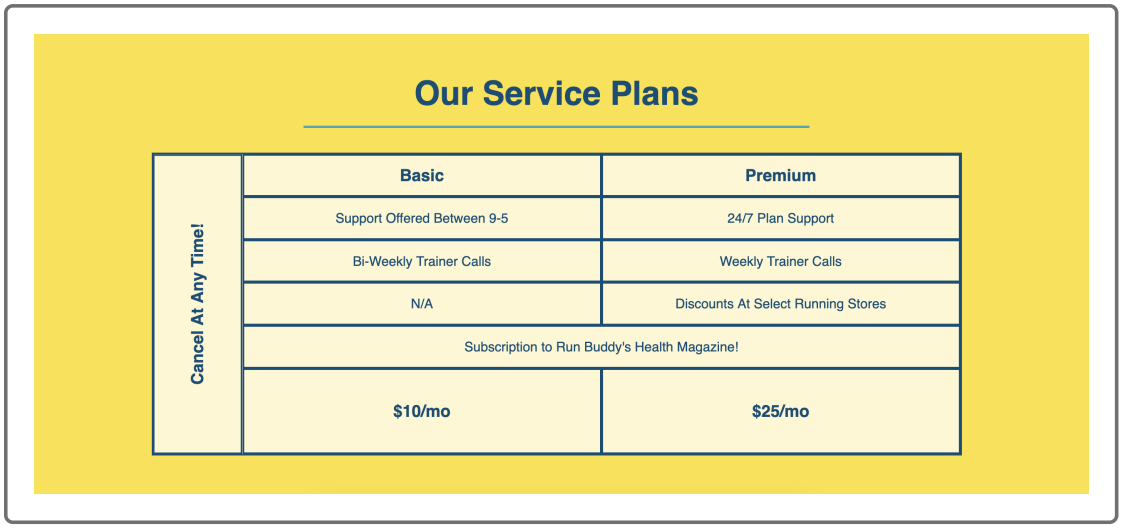
Note that the -1 property value was used to set the ending grid line. This is a great shortcut when we know that the grid item spans the entire grid.

So why did we not need to designate a column for this grid item? CSS Grid is calculating placement and autofills in the grid areas that aren't occupied. To be explicit, we could add the `grid-column` declaration to `1`, but that isn't necessary in this case because the only option for the subscription grid item isto span the length of the grid.

Great job so far! Now that our grid items are placed properly in a formatted grid according to the mock-up, the design team is very happy with the progress made so far. They just have a few recommendations to emphasize a few key fields in the table.

Add the Final Touches

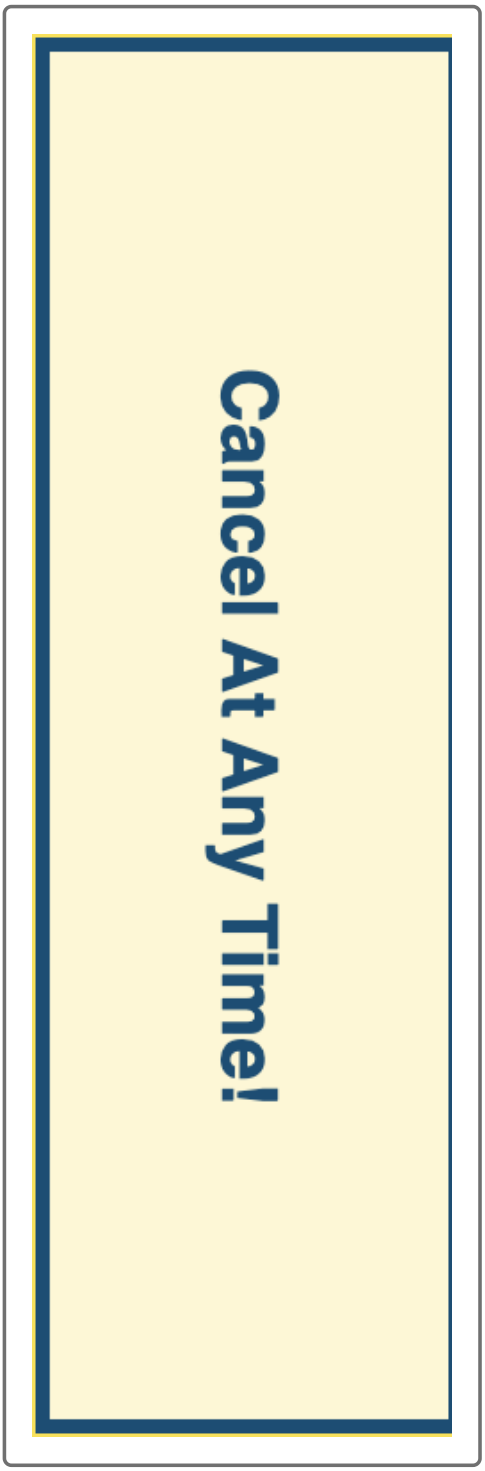
Let's take a peek at the mock-up and see what's left to be done:



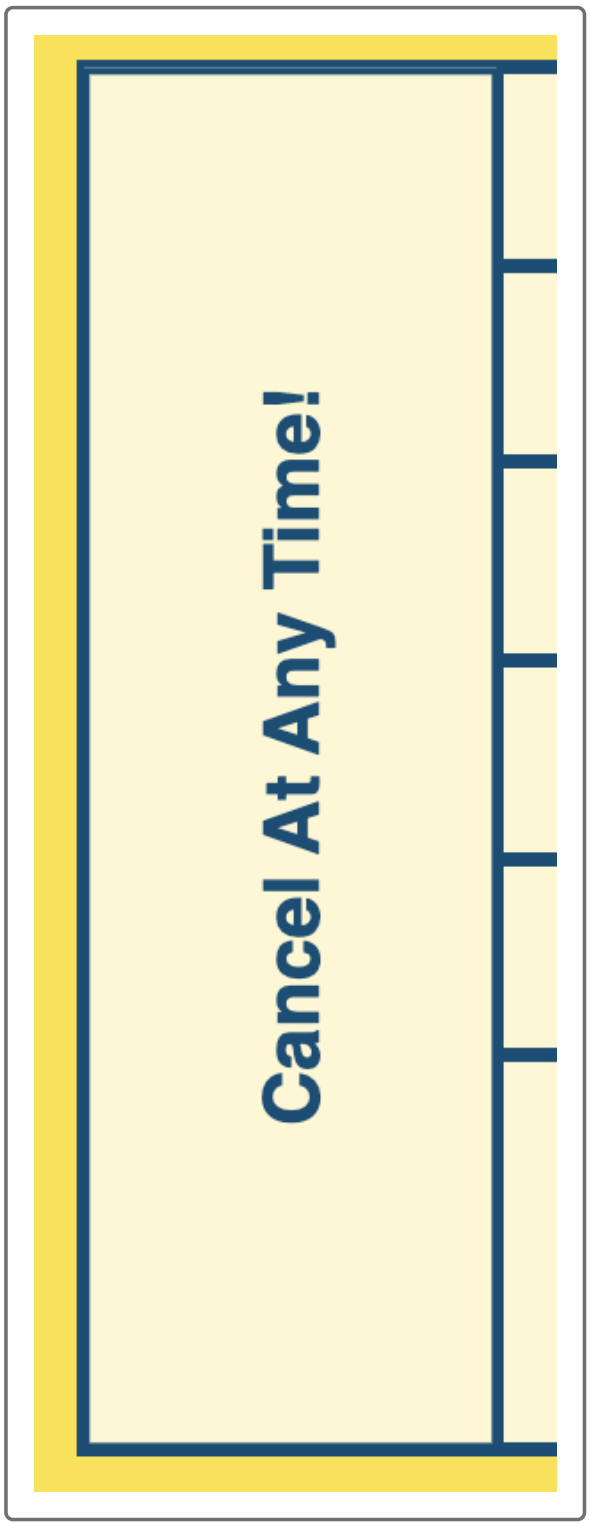
The design team has requested a special font change to the first column in the grid to vertically align the text with the column. Let's use the `writing-mode` property with the value of `vertical-lr` to see if this is a good option:

```
.service-grid-item.cancel {
  writing-mode: vertical-lr;
  grid-column: 1;
  grid-row: 1 / -1;
}
```

This should have the following result in the browser:

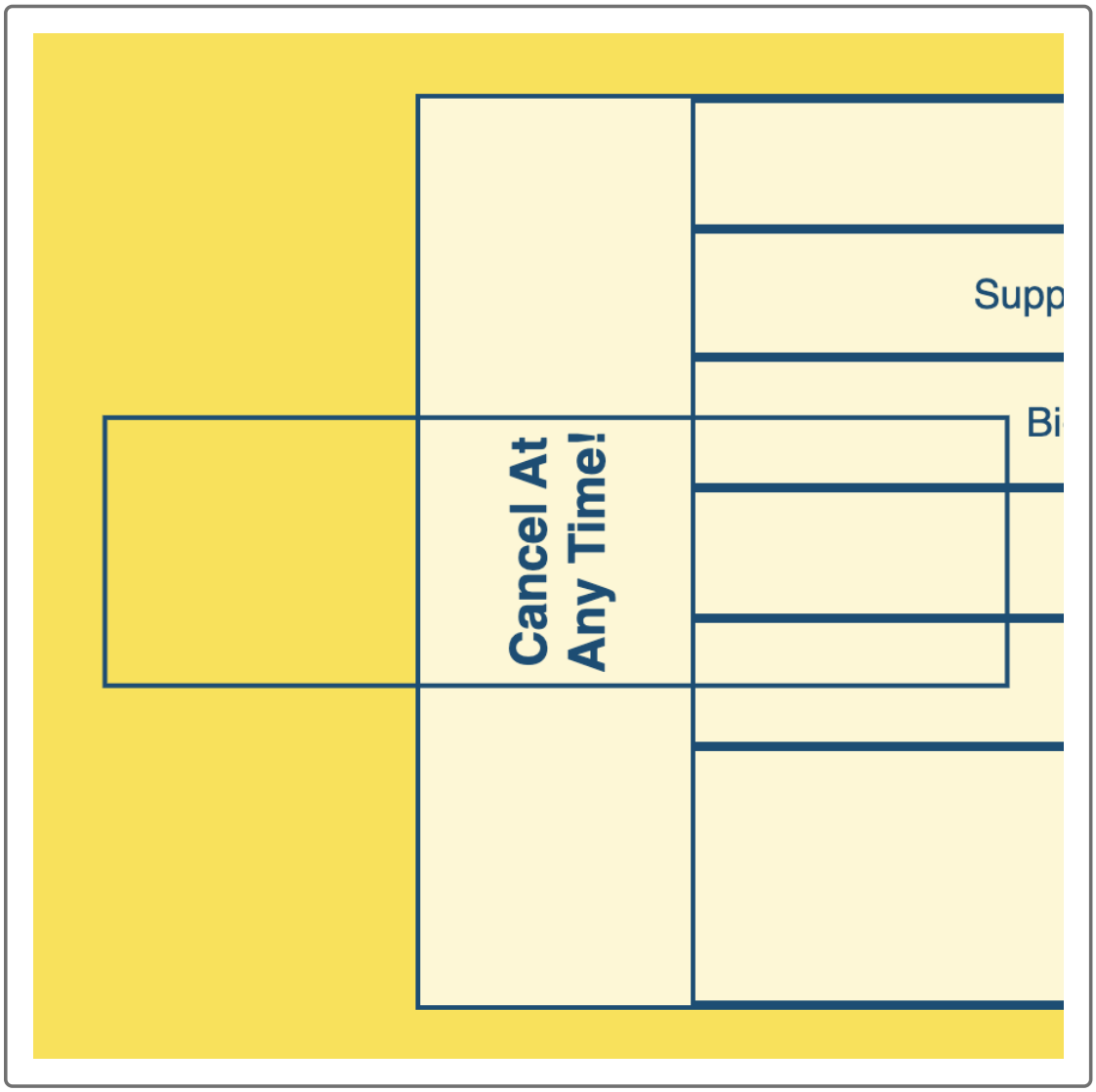


This looks promising. but it's not exactly what the design team wants. So let's keep working on it. We need to find a way to flip or rotate this text to face the correct direction. Fortunately for us, the `transform` property has a rotate value we can try. Let's add the following declaration, `transform: rotate(180deg);`, into the `.service-grid-item.cancel` rule to render the following in the browser:



Now the "Cancel" grid item looks great!

Wait a minute. Why why didn't we just rotate it from the beginning? Let's see what the "Cancel" grid item would have looked like if we had just rotated it by -90 degrees and removed the `writing-mode` property:



So this actually rotated the entire element and not just the font. Let's put the `writing-mode` property back into our rule and set the `transform` property value back to `rotate(180deg);`.

DEEP DIVE

DEEP DIVE

To learn more, see the **MDN web docs on the writing-mode property (https://developer.mozilla.org/en-US/docs/Web/CSS/writing-mode)** and the **MDN web docs on transformations (https://developer.mozilla.org/en-US/docs/Web/CSS/transform)**.

Next let's use some CSS properties to increase the headers, cancellation message, and pricing by first assigning the class attributes to the corresponding elements in the HTML, and then creating the related CSS rules with a font size of 1.5rem and a bold font-weight:

Add the following classes to `index.html`:

```
<!-- Headers -->
<div class="service-grid-item grid-head basic">
  Basic
</div>
<div class="service-grid-item grid-head premium">
  Premium
</div>
...
<div class="service-grid-item grid-price basic">
  $10/mo
</div>
<div class="service-grid-item grid-price premium">
  $25/mo
</div>
```

Add the following to `style.css`:

```
.grid-head, .grid-price, .service-grid-item.cancel {
  font-size: 1.5rem;
```

```
  font-weight: bold;
}
```

This rule uses three different selectors to declare font styling to emphasize their priority in the table.