# CSS Grid: PupSpa

Let’s practice what we’ve just learned about Native CSS Grid through building a classic, grid-based, responsive web page. Your friend has just opened a doggie daycare and grooming services shop, PupSpa. They would like a simple web page to let customers know about their services.

First, we’’ll build a grid layout for desktop and then modify it for mobile using a media query. Click the expansion arrow in the right hand corner of the browser to see the difference between the desktop and mobile versions. We’ve provided you with the complete index.html file and the beginnings of your CSS.

Let’s get started!

If you get stuck during this project or would like to see an experienced developer work through it, click “**Get Help**“ to see a **project walkthrough video**.

**Tasks**

**15/15Complete**

Mark the tasks as complete by checking them off

**1.**

First, let’s take a look at our **index.html** file. As you can see, we have several <div>s that contain the content of our website. Find the <div> with the class grid — this is our container grid and the properties applied to it will determine the structure for the content inside. Next, locate the <div>s that share the class box and the <div>s that share the class testimonial. Their positioning will change the most when the page is resized.

**2.**

Now let’s look at our CSS. We have provided a lot of the initial styling and now it is up to you to add the necessary CSS grid properties to make sure the content is properly laid out on the page.

In the .grid rule set in **style.css**, we’re going to lay out our grid and specify the number and size of our rows and columns. First, let’s initialize the grid by setting the display property to grid.

Hint

display: grid;

**3.**

Responsively size the grid rows using the grid-template-rows property in the .grid rule set. Set it equal to this value: 5% 30% 10% 30% 20% 5%. What changes do you see?

Hint

grid-template-rows: 5% 30% 10% 30% 20% 5%;

**4.**

Still inside of the .grid rule set, set grid-template-columns to six equal sections using the fr measurement.

When you run your code, don’’t panic! The CSS is just trying to organize our content based off of our instructions. In a later section, we’’ll go over how to style our <div>s so they take up the necessary column widths.

Hint

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

**5.**

Refactor the previous exercise’s code using the repeat() function. After that, refactor the grid-template-rows and grid-templates-columns code using the grid-template property.

Hint

grid-template-columns: repeat(6, 1fr);

grid-template: 5% 30% 10% 30% 20% 5% / repeat(6, 1fr);

**6.**

Now, let’s fix that broken page layout. Using grid-column-start and grid-column-end, style the <header> element so that it extends across all of the columns. Now, do the same for .banner, .about, and footer.

Hint

grid-column-start: 1;

grid-column-end: 7;

**7.**

Use grid-column to cause .address to span the first two columns, .hours to span the 3rd and 4th, and .call-us to span the 5th and 6th.

Hint

grid-column: 1 / span 2;```

```css

grid-column: 3 / span 2;```

```css

grid-column: 5 / span 2;```

**8.**

In the .one and .two rule sets, use grid-column to have each <div> span three columns in the 5th row.

Hint

grid-column: 1 / span 3;```

```css

grid-column: 4 / span 3;```

**9.**

Let’s give our content some wiggle room. Return to the .grid rule set, add a grid-column-gap property, and set its value to 20 pixels. Notice which parts of the page have changed.

Hint

grid-column-gap: 20px;

**10.**

Now let’s lay out our mobile version. For this grid, we want all of the content to occupy one column. Within the @media rule, inside the .grid rule set, add the grid-template-columns property and set its value to 100%.

Be sure to resize the browser on the right to a small screen width (less than 600px) to see this change.

Hint

Your declaration will look like:

grid-template-columns: 100%;

To learn more about responsive design, inluding media queries, check out our [Learn Responsive Web Design course](https://www.codecademy.com/learn/learn-responsive-design).

**11.**

We also need to specify additional rows. Add grid-template-rows and have its value be the same as the desktop version. Remember that each value is equal to a row.

To include more rows, split the fourth value into three equal values. Use the same process to turn the fifth row into two new rows. Try to practice your refactoring on this one!

Hint

grid-template-rows: 5% 30% 10% repeat(3, 10%) repeat(2, 10%) 5%;

**12.**

Use grid-row to define the starting row of the information blocks (.address, .hours, and .call-us) and testimonial blocks (.one and .two), and how many rows they will span.

Hint

grid-row: 4/ span 1;```

```css

grid-row: 5/ span 1;```

```css

grid-row: 6/ span 1;```

```css

grid-row: 7/ span 1;```

```css

grid-row: 8/ span 1;```

**13.**

We also have to specify that these blocks now only take up one column of space. Add grid-column: 1 / span 1; to their rule sets.

Hint

css grid-column: 1 / span 1;

**14.**

Rather than have gaps between columns, we now want to have gaps between the rows.

Return to the .grid rule set and add a property so that there is a 20 pixel gap between the rows.

Hint

grid-row-gap: 20px;

**15.**

But what if we don’t want there to be a gap between all of our rows? We can avoid the grid-row-gap rule by using negative margins. In header, .banner, and .about, set margin-bottom to -20px. For the footer, set margin-top to -20px.

Hint

margin-bottom: -20px

margin-top: -20px

CSS Grid: PupSpa