Create a grid layout (solution)
Below is a sample CSS solution file for the previous exercise.

```
.container {
  display: grid;
  max-width: 900px;
  min-height: 50vh;
  grid-template-columns: 100%;
  grid-template-rows: auto auto 1fr auto auto;
  grid-template-areas: "header" "left" "main" "right" "footer";
}

@media (min-width: 440px) {
  .container {
    grid-template-columns: 150px 1fr 150px;
    grid-template-rows: auto 1fr auto;
    grid-template-areas: "header header header" "left main right" "footer footer footer";
  }
}

.header {
  grid-area: header;
  padding: 10px;
  background-color: black;
  color: #fff;
  text-align: center;
}

.main {
  grid-area: main;
  padding: 25px;
}

.left {
  grid-area: left;
  background-color: peachpuff;
}

.right {
  grid-area: right;
}

.footer {
```

While reviewing the code, note the following:

The grid template areas are defined as **"header" "left" "main" "right" "footer"** but for a small device with a screen width of 440px or less, it is defined as **"header header header" "left main right" "footer footer footer"** using a media query.
The **grid-rows** property value also changes based on the media query.

The values for the number of rows you add for **grid-template-rows** and number of columns you add for **grid-template-columns** must match the dimensions of the grid-template-areas.
grid-area that has undefined rules will appear empty. (Does not happen with the example above.)

Each CSS rule specifies which grid area they belong to by using the **grid-area** CSS property.

The selectors of each rule used are element tags in HTML or classes, as we have used here.