Internship Report – Frontend Dev(Week3)

Name: Zainab
Father Name: Assad Qayyum
Date: 7th July, 2025
Internship Domain: Front-end Intern
Task: CSS Grid: Rows, Columns, grid-template areas

Task Overview: (Day1)

Today's task was to explore advanced CSS layout techniques using the CSS Grid. I focused on learning how to divide layouts into rows and columns, and how to define and use grid template areas to build clean and structured web page layouts.

Content Covered:

- Introduction to CSS Grid Layout
- Defining Rows and Columns using grid-template-rows and grid-template-column
- Naming Layout Areas with grid-template-areas
- Applying grid-area to assign elements to specific areas of the grid

Introduction:

CSS Grid is a powerful layout system in CSS that enables two-dimensional design using rows and columns. It is especially useful for creating responsive, flexible, and organized layouts. With Grid, developers can position elements easily without relying heavily on floats or complex nesting.

1. CSS Grid:

CSS Grid Layout is defined by applying display: grid to a container. It turns the container into a grid system where child elements (grid items) can be placed into rows and columns.

The element becomes a grid container when its display property is set to grid or inline-grid.

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

A grid consists of:

- A **Grid Container** - the parent (container) element
- **Grid Items** - the items inside the container

2. Rows:

Rows in a grid are defined using the grid-template-rows property. This allows us to define the height of each row.

Property	**Purpose**
grid-template-rows	Defines the height of rows
row-gap	Adds space between rows
grid-row-start	Specifies where a grid item starts vertically
grid-row-end	Specifies where a grid item ends vertically
grid-row	Shorthand for start/end

a) The row-gap Property

The row-gap property (or just gap when used with columns) controls the spacing between rows in the grid.

```
.container {
  row-gap: 20px;
}
```

- This adds 20px of space between each row, improving visual separation.
- We can also use gap: 20px 10px; for both rows and columns, first value is for rows, second for columns.

b) The grid-row-start and grid-row-end Properties:

These two properties are used on grid items to define which rows the item should span between.

```
.item {
  grid-row-start: 1;
  grid-row-end: 3;
}
```

- This makes the item span from row line 1 to line 3, which means it covers 2 rows.
- Grid lines start from 1 and are numbered at every track boundarynot row number, but line number.

c) The grid-row Shorthand

Instead of using grid-row-start and grid-row-end separately, you can combine them:

```
.item {
  grid-row: 1 / 3;   /*1 is row start and 3 is row end*/
}
```

Example:

```
grid-template-rows: 100px auto 1fr;
```

- 100px → The first row has a fixed height of 100 pixels.
- auto → The second row will automatically grow/shrink based on its content.
- 1fr → The third row will take up the remaining available space using a fractional unit (fr).

3. Columns:

Columns in CSS Grid are the vertical tracks used to structure content side by side. You define column sizes using the grid-template-columns property.

Property	Purpose
grid-template-columns	Defines column widths
column-gap	Adds space between columns
grid-column-start	Where a grid item starts horizontally
grid-column-end	Where a grid item ends horizontally
grid-column	Shorthand for start/end

a) The column-gap Property

The column-gap property defines the horizontal space between columns in a grid.

```
.container {
  column-gap: 20px;
} /* This adds 20px of spacing between columns.*/
```

To define both row and column gaps together:

```
gap: 20px 10px; /* 20px = row-gap, 10px = column-gap */
```

b) The grid-column-start and grid-column-end Properties

These properties control where a grid item begins and ends horizontally, allowing you to span it across multiple columns.

```
.item {
  grid-column-start: 1;
  grid-column-end: 4;
}
```

- This will stretch the item across columns 1 to 3 (ends before 4), effectively spanning 3 columns.
- Column lines start at 1 and are counted between the columns (not the columns themselves).

c) The grid-column Shorthand

You can use grid-column to write grid-column-start and grid-column-end in one line:

```
.item {
  grid-column: 1 / 4;
}
```

Example:

```
grid-template-columns: 200px 1fr 2fr;
```

- 200px → The first column has a fixed width of 200 pixels.
- 1fr → The second column takes up one fraction of the remaining space.
- 2fr → The third column takes up twice as much space as the second column

4. Grid Template Areas:

Grid-template-areas lets you visually define layout regions using names. Each name represents a part of the layout (like header, sidebar, content, etc.).

```
grid-template-areas:
"header header"
 "sidebar content"
 "footer footer";
```

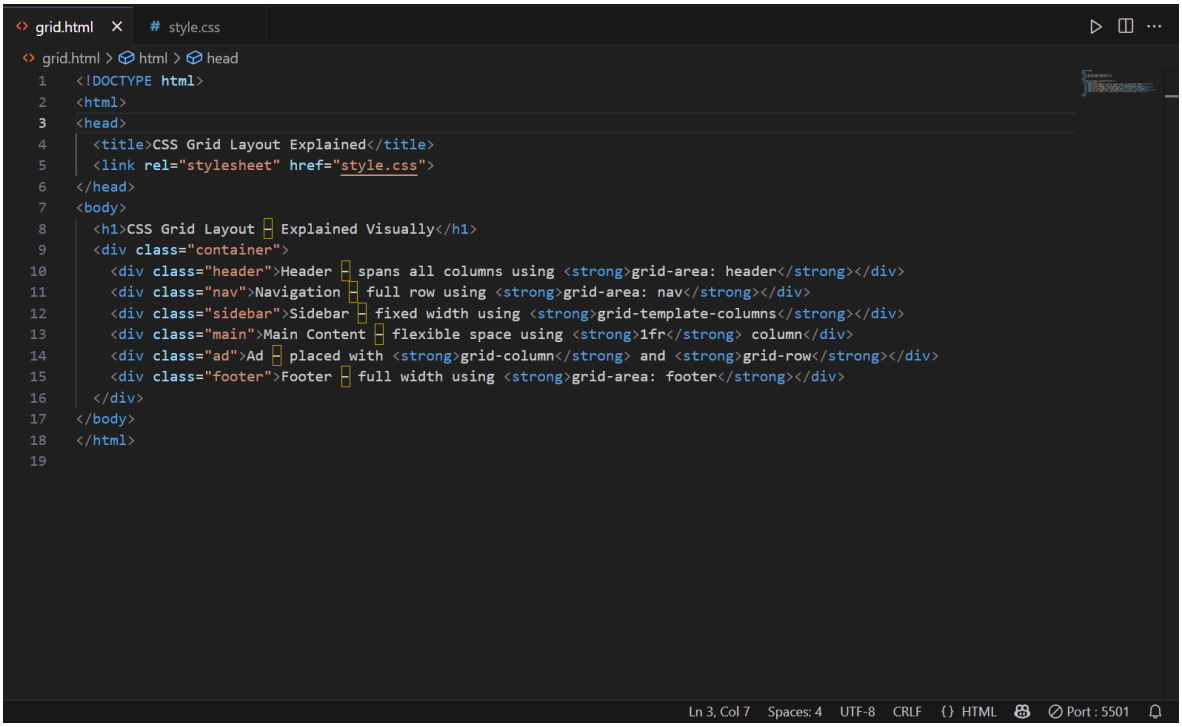
This is helpful for larger layouts to keep the design readable and organized.

Each grid item is assigned to an area using the grid-area property:

```
.header { grid-area: header; }
.sidebar { grid-area: sidebar; }
.content { grid-area: content; }
.footer { grid-area: footer; }
```

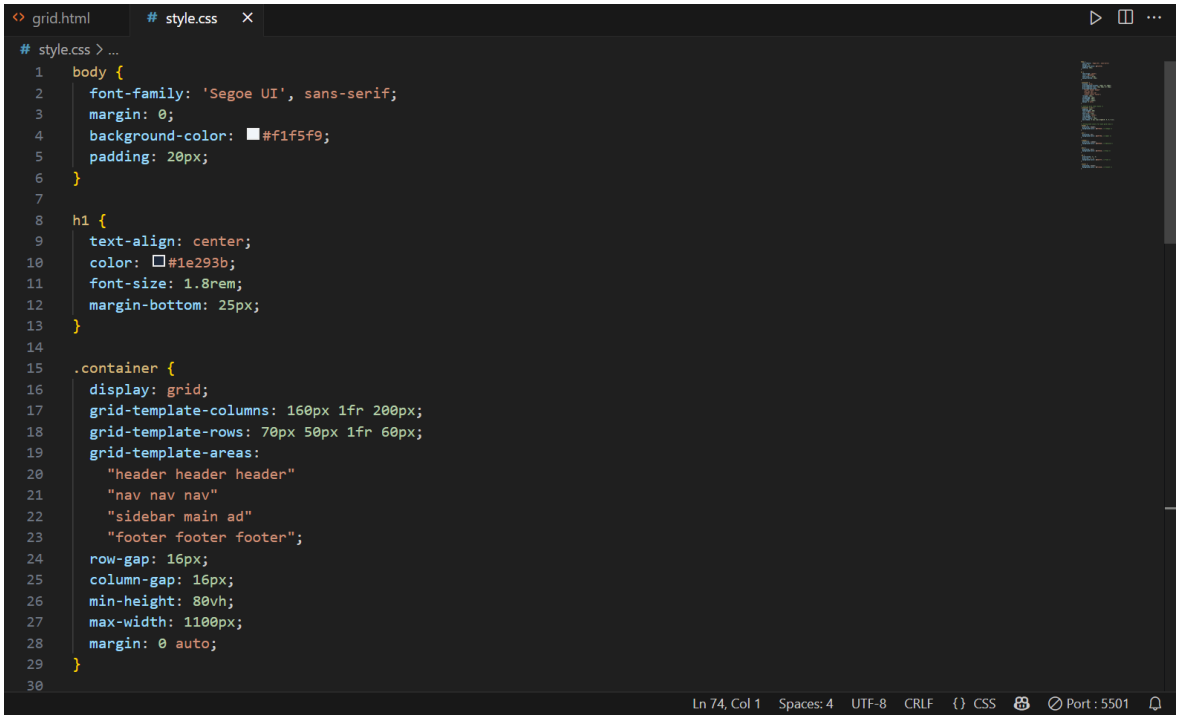
Practice Code:

Index.html:



```html
<!DOCTYPE html>
<html>
<head>
  <title>CSS Grid Layout Explained</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <h1>CSS Grid Layout – Explained Visually</h1>
  <div class="container">
    <div class="header">Header – spans all columns using <strong>grid-area: header</strong></div>
    <div class="nav">Navigation – full row using <strong>grid-area: nav</strong></div>
    <div class="sidebar">Sidebar – fixed width using <strong>grid-template-columns</strong></div>
    <div class="main">Main Content – flexible space using <strong>1fr</strong> column</div>
    <div class="ad">Ad – placed with <strong>grid-column</strong> and <strong>grid-row</strong></div>
    <div class="footer">Footer – full width using <strong>grid-area: footer</strong></div>
  </div>
</body>
</html>
```

Styles.css:



```css
body {
  font-family: 'Segoe UI', sans-serif;
  margin: 0;
  background-color: #f1f5f9;
  padding: 20px;
}

h1 {
  text-align: center;
  color: #1e293b;
  font-size: 1.8rem;
  margin-bottom: 25px;
}

.container {
  display: grid;
  grid-template-columns: 160px 1fr 200px;
  grid-template-rows: 70px 50px 1fr 60px;
  grid-template-areas:
    "header header header"
    "nav nav nav"
    "sidebar main ad"
    "footer footer footer";
  row-gap: 16px;
  column-gap: 16px;
  min-height: 80vh;
  max-width: 1100px;
  margin: 0 auto;
}
```

grid.html | style.css

style.css > ...

```
15  .container {
29  }
30
31  /* General grid item styles */
32  .container > div {
33    padding: 20px;
34    font-weight: 600;
35    color: #ffffff;
36    font-size: 1rem;
37    text-align: center;
38    line-height: 1.5;
39    border-radius: 10px;
40    box-shadow: 0 2px 12px rgba(0, 0, 0, 0.1);
41  }
42
43  /* Strong bold colors for each grid area */
44  .header {
45    grid-area: header;
46    background-color: #4f46e5; /* Indigo */
47  }
48
49  .nav {
50    grid-area: nav;
51    background-color: #d97706; /* Amber */
52  }
53
54  .sidebar {
55    grid-area: sidebar;
56    background-color: #059669; /* Emerald */
57  }
```

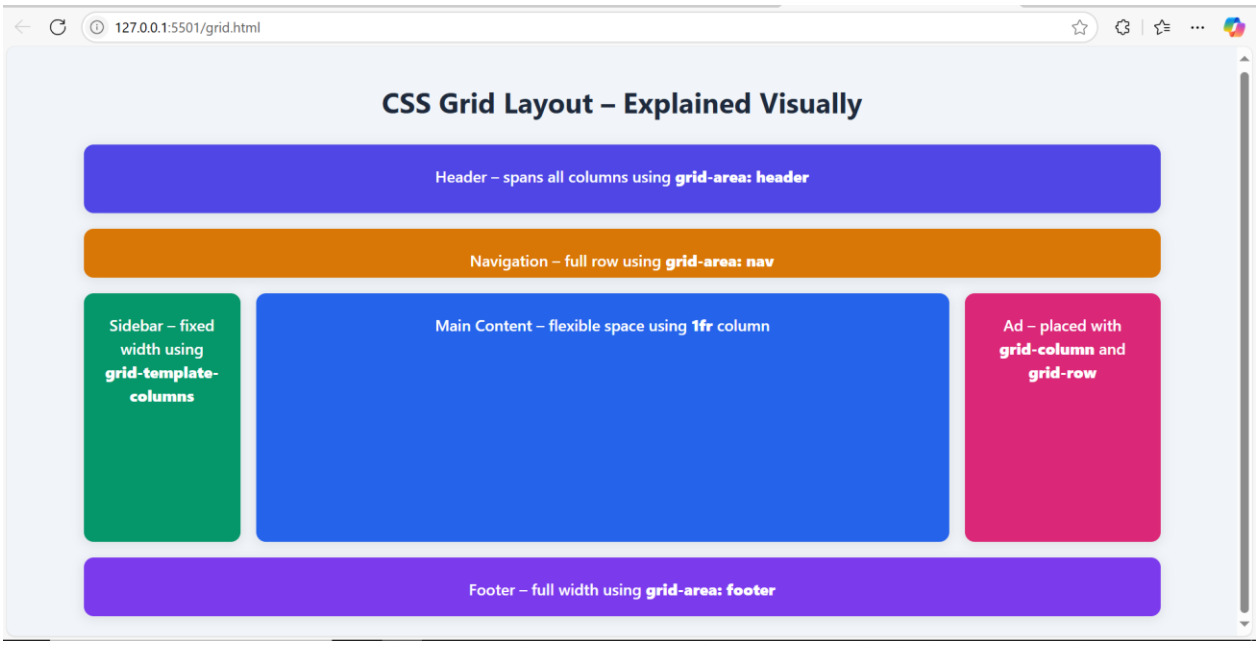
Ln 74, Col 1 Spaces: 4 UTF-8 CRLF {} CSS Port : 5501

grid.html | style.css

style.css > ...

```
49  .nav {
52  }
53
54  .sidebar {
55    grid-area: sidebar;
56    background-color: #059669; /* Emerald */
57  }
58
59  .main {
60    grid-area: main;
61    background-color: #2563eb; /* Blue */
62  }
63
64  .ad {
65    grid-column: 3 / 4;
66    grid-row: 3 / 4;
67    background-color: #db2777; /* Pink */
68  }
69
70  .footer {
71    grid-area: footer;
72    background-color: #7c3aed; /* Violet */
73  }
74
```

Ln 74, Col 1 Spaces: 4 UTF-8 CRLF {} CSS Port : 5501



Conclusion:

Today's task helped me understand how CSS Grid allows easy creation of two-dimensional layouts using rows and columns. Learning grid-template-areas added more control and readability to the layout. Overall, CSS Grid is a clean and powerful way to structure web pages without extra complexity.