Лекция: CSS Grid – Современная система макетов для веб-приложений

Цель лекции: Сформировать у студентов понимание модуля CSS Grid Layout, его основных концепций и практических навыков по созданию сложных, адаптивных макетов веб-страниц.

План лекции

1. **Введение в CSS Grid.**
    - Что такое CSS Grid?
    - Ключевые отличия от Flexbox.
    - Терминология: Grid Container, Grid Items, Grid Lines, Grid Tracks, Grid Cells, Grid Areas.
2. **Создание базового Grid-контейнера.**
    - Свойства `display: grid` и `display: inline-grid`.
    - Определение структуры: `grid-template-columns` и `grid-template-rows`.
    - Единицы измерения: `fr`, `auto`, `minmax()`, `repeat()`.
3. **Размещение элементов внутри Grid.**
    - Явное размещение по линиям: `grid-column-start`, `grid-column-end`, `grid-row-start`, `grid-row-end` и их шорткаты `grid-column` и `grid-row`.
    - Размещение по областям: `grid-template-areas` и `grid-area`.
    - Неявное размещение и поток элементов.
4. **Управление промежутками и выравниванием.**
    - Создание отступов между элементами: `gap` (`row-gap`, `column-gap`).
    - Выравнивание всего грида: `justify-content` и `align-content`.
    - Выравнивание отдельных элементов внутри своей ячейки: `justify-items` и `align-items`.
    - Индивидуальное выравнивание элемента: `justify-self` и `align-self`.
5. **Создание адаптивных макетов без медиа-запросов.**
    - Функция `auto-fill` и `auto-fit` в комбинации с `repeat()`.
    - Использование `minmax()` для создания гибких и отзывчивых tracks.
6. **Резюме и лучшие практики.**

Рассмотрение пунктов плана:

1. Введение в CSS Grid

Что такое CSS Grid? CSS Grid Layout — это двумерная система макетов для веба. Она позволяет располагать элементы на странице по строкам и столбцам одновременно, что делает ее идеальным

инструментом для создания сложных, структурных макетов всей страницы или ее крупных частей.

Ключевые отличия от Flexbox:

- **Измерность:** Flexbox — это *одномерная* система. Он работает либо по строке, либо по столбцу в один момент времени. Grid — *двумерная* система, управляет и строками, и столбцами вместе.
- **Назначение:** Flexbox идеален для выравнивания контента *внутри* компонента (например, навигационная панель, карточка товара). Grid идеален для макета *самой страницы* (расположение хедера, сайдбара, основного контента, футера).
- **Отношения:** Во Flexbox отношения между элементами сложные, размеры вычисляются на основе контента. В Grid отношения задаются на уровне контейнера, что дает больше контроля.

Терминология:

- **Grid Container (`display: grid`)** — элемент, к которому применяется `display: grid`. Он становится контейнером для всех своих прямых потомков, которые становятся...
- **Grid Items** — прямые потомки Grid Container.
- **Grid Lines** — разделительные линии, которые составляют структуру грида. Они бывают вертикальные (column lines) и горизонтальные (row lines). Нумерация начинается с 1.
- **Grid Tracks** — пространство между двумя соседними линиями. Это *столбец* или *строка*.
- **Grid Cell** — наименьшая единица грида, пересечение одной строки и одного столбца (как ячейка в таблице).
- **Grid Area** — прямоугольная область, состоящая из одной или нескольких смежных Grid Cells. Задается по линиям или по имени.

> **Пример:** Представьте себе шахматную доску. Сама доска — это Grid Container. Клетки на доске — это Grid Cells. Линии, разделяющие клетки — Grid Lines. Ряд клеток по горизонтали — это row track, по вертикали — column track. А область, занимаемая, например, ладьей (несколько клеток), — это Grid Area.

---

2. Создание базового Grid-контейнера

Активация Grid:

```
.container {
  display: grid; /* или display: inline-grid; */
}
```

Все прямые дочерние элементы `.container` мгновенно становятся grid-элементами.

Определение структуры:

```
.container {
  display: grid;
  grid-template-columns: 200px 1fr 300px; /* 3 колонки */
  grid-template-rows: 100px auto 150px; /* 3 строки */
}
```

- `grid-template-columns`: задает количество и ширину колонок.
- `grid-template-rows`: задает количество и высоту строк.

Единицы измерения:

- `fr` (fraction — доля) — гибкая единица, распределяющая доступное пространство.

```
grid-template-columns: 1fr 2fr 1fr; /* Средняя колонка в 2 раза шире крайних
*/
```

- `auto` — занимает все доступное пространство, исходя из контента.
- `minmax(min, max)` — задает диапазон размеров.

```
grid-template-columns: 200px minmax(300px, 1fr) 100px;
/* Вторая колонка не уже 300px и не шире 1fr */
```

- `repeat()` — функция для повторения шаблона.

```
grid-template-columns: repeat(3, 1fr); /* Эквивалентно 1fr 1fr 1fr */
grid-template-rows: repeat(2, 150px); /* Две строки по 150px */
```

---

3. Размещение элементов внутри Grid

По умолчанию элементы располагаются в гриде автоматически, занимая по одной ячейке в порядке их появления в HTML. Но мы можем управлять их позицией.

Размещение по линиям: Каждый элемент можно "привязать" к Grid Lines.

```
.item {
  grid-column-start: 1; /* Начинается с 1-й вертикальной линии */
  grid-column-end: 3;   /* Заканчивается на 3-й вертикальной линии */
  grid-row-start: 2;    /* Начинается с 2-й горизонтальной линии */
  grid-row-end: 4;      /* Заканчивается на 4-й горизонтальной линии */
}
```

Шорткаты:

```
.item {
  grid-column: 1 / 3;   /* start-line / end-line */
  grid-row: 2 / 4;
}
```

Можно использовать ключевое слово `span`:

```
.item {
  grid-column: 1 / span 2; /* Начать с линии 1 и занять 2 трека */
  grid-row: 2 / span 2;    /* Начать с линии 2 и занять 2 трека */
}
```

Размещение по областям (Grid Areas) — мощный и наглядный способ:

1. Даем имена областям в контейнере с помощью `grid-template-areas`:

   ```
   .container {
     display: grid;
     grid-template-areas:
       "header header header"
       "sidebar content content"
       "footer footer footer";
     grid-template-columns: 200px 1fr 1fr;
     grid-template-rows: auto 1fr auto;
   }
   ```

2. Привязываем элементы к этим именам с помощью `grid-area`:

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

   Этот метод очень нагляден и легко читается. Точка (`.`) означает пустую ячейку.

---

4. Управление промежутками и выравниванием

Промежутки (Gaps): Создает отступы *между* ячейками грида. Не отступы по краям контейнера.

```
.container {
  gap: 20px; /* Универсальный отступ */
  /* или */
  row-gap: 15px;
  column-gap: 30px;
}
```

Выравнивание всего грида (если грид меньше своего контейнера):

- `justify-content` — выравнивание по горизонтали (по главной оси, если направление строки).
- `align-content` — выравнивание по вертикали (по поперечной оси). Значения: `start`, `end`, `center`, `stretch`, `space-around`, `space-between`, `space-evenly`.

Выравнивание элементов внутри своих ячеек:

- `justify-items` — выравнивание элемента по горизонтали *внутри своей ячейки*.
- `align-items` — выравнивание элемента по вертикали *внутри своей ячейки*. Значения: `start`, `end`, `center`, `stretch` (по умолчанию).

Индивидуальное выравнивание одного элемента:

- `justify-self` — индивидуальное выравнивание по горизонтали.
- `align-self` — индивидуальное выравнивание по вертикали.

```
.special-item {
  justify-self: center; /* Этот элемент будет отцентрован в своей ячейке */
  align-self: end;      /* И прижат к нижнему краю */
}
```

---

5. Создание адаптивных макетов без медиа-запросов

Одна из самых мощных возможностей Grid — создание отзывчивых макетов "на лету".

Функции `auto-fill` и `auto-fit` в `repeat()`: Они работают в связке с `minmax()`.

```
.container {
  display: grid;
  grid-template-columns: repeat(auto-fill, minmax(250px, 1fr));
  gap: 15px;
}
```

- `minmax(250px, 1fr)`: каждая колонка будет не уже `250px` и не шире `1fr`.
- `repeat(auto-fill, ...)`: браузер будет *заполнять* строку как можно большим количеством колонок шириной минимум `250px`. Если останется свободное место, он его оставит пустым.
- `repeat(auto-fit, ...)`: браузер будет *растягивать* колонки, чтобы они заполнили всю доступную ширину строки.

> **Пример:** Представьте контейнер шириной 1000px. С `auto-fill` и `minmax(250px, 1fr)` браузер создаст 4 колонки (4 * 250px = 1000px). Если ширина контейнера увеличится до 1200px, `auto-fill` создаст 4 колонки по 300px каждая, а `auto-fit` — растянет 4 колонки, чтобы они заняли все 1200px. При уменьшении ширины ниже 500px, и тот, и другой вариант переведут макет в одну колонку. Это и есть адаптивность без медиа-запросов.

---

6. Резюме и лучшие практики

Резюме:

1. **CSS Grid — это мощная двумерная система** для создания макетов веб-страниц.
2. **Основная работа ведется в контейнере**, где определяется структура (столбцы и строки).
3. **Элементы размещаются** с помощью привязки к линиям или по именованным областям, что является очень удобным и читаемым методом.
4. **Система выравнивания** в Grid мощная и логичная, она разделяет выравнивание всей сетки и отдельных элементов.
5. **Grid отлично справляется с адаптивностью**, позволяя создавать сложные отзывчивые макеты с минимальным количеством кода, часто без медиа-запросов.

Лучшие практики:

- **Используйте Grid для макета страницы, Flexbox — для компонентов.** Они не конкуренты, а союзники.
- **Отдавайте предпочтение** `grid-template-areas` для сложных, но понятных макетов. Это делает CSS самодокументируемым.
- **Освойте функцию** `minmax()` для создания по-настоящему гибких и устойчивых интерфейсов.
- **Не бойтесь вкладывать Grid в Grid или Flexbox в Grid.** Это стандартная практика для построения современных интерфейсов.
- **Всегда проверяйте поддержку браузерами,** но на сегодняшний день CSS Grid поддерживается всеми современными браузерами (>98% глобального покрытия). Для легаси-проектов имейте запасной план.

CSS Grid — это фундаментальный сдвиг в подходе к веб-верстке. Освоив его, вы получаете в руки инструмент, который значительно ускоряет разработку, делает код чище, а макеты — более стабильными и мощными.