Comp 324/424 - Client-side Web Design

Spring Semester 2024 Week 9

Dr Nick Hayward

Dev week demo & assessment

Course total = 25 credits

- continue development of a web application
  - built from scratch
  - HTML5, CSS, plain JavaScript…
- continue design and development of initial project outline and design
- working app (as close as possible…)
  - **NO** content management systems (CMSs) such as Drupal, Joomla, WordPress…
  - **NO** PHP, Python, Ruby, C# & .Net, Java, Go, XML…
  - **NO** CSS frameworks, such as Bootstrap, Foundation, Materialize…
  - **NO** CSS preprocessors such as Sass…
  - **NO** template tools such as Handlebars.js &c.
- data may be implemented from either
  - self hosted (MongoDB, Redis…)
  - APIs
  - cloud services (Firebase…)
  - **NO** SQL…e.g. (you may **NOT** use MySQL, PostgreSQL &c.)
- outline research conducted
- describe data chosen for application
- show any prototypes, patterns, and designs

Dev week demo & assessment

DEV week assessment will include the following:

- brief presentation or demonstration of current project work
  - ~ 10 minutes per group
  - analysis of work conducted so far
    - e.g. during semester & DEV week
  - presentation and demonstration
    - outline current state of web app
    - explain what works & does not work
    - show implemented designs since project outline & mockup
    - show latest designs and updates
  - due Monday 18th March 2024 @ 4.15pm

HTML5, CSS, & JS - example - part 1

add grid layout - option 1

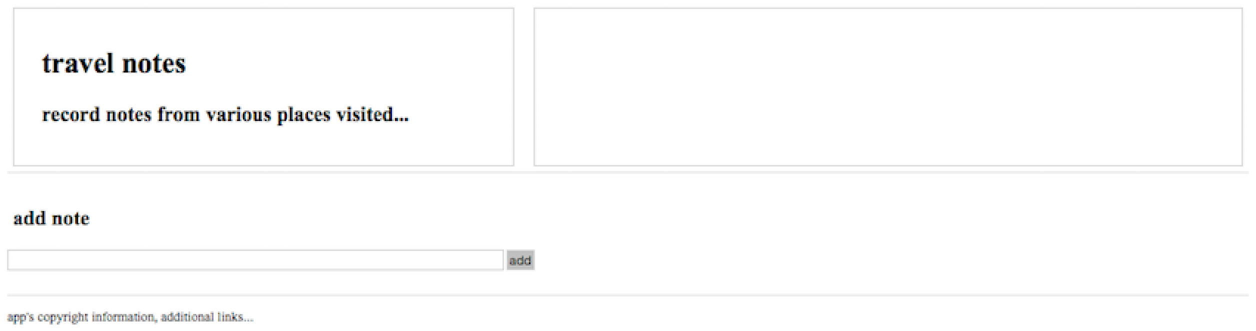
- update the layout of our Travel Notes application to include a grid layout
- apply this grid layout to the overall application
  - organisation and presentation of the notes
- remove the centred, fixed width for the `body` in our style.css stylesheet
- removes centre styling, results in content spanning full width of browser window
- add the grid layout to help us control this layout

```
<link rel="stylesheet" href="assets/styles/grid.css">
```

- then modify content categories, child elements to use new grid css

```
<!-- document header -->
<header>
  <div class="row">
    <div class="col-5">
      <h3>travel notes</h3>
      <h5>record notes from various places visited...</h5>
    </div>
    <div class="col-7"></div>
  </div>
</header>
```

Image - HTML5, CSS, & JS - grid layout


travel notes

record notes from various places visited...

add note

app's copyright information, additional links...

Figure 1: Grid Layout - Updated Header - option 1

HTML5, CSS, & JS - example - part 1

add grid layout - option 2

- alternative layout option

```
<!-- grid banner -->
<div class="banner">
    <!-- logo -->
    <div class="logo">
        <img src="/assets/images/travel-notes-logo2.png" alt="site logo" width="80" height="80">
    </div>
    <!-- document header -->
    <header class="site-header">
        <h3>travel notes</h3>
        <!--<h5>record notes from various places visited...</h5>-->
    </header>
    <!-- banner extras -->
    <div class="banner-extras">
    </div>
</div> <!-- end of grid banner -->
```

- a few extra *places* added to layout
  - logo, header, and banner extras

Image - HTML5, CSS, & JS - grid layout

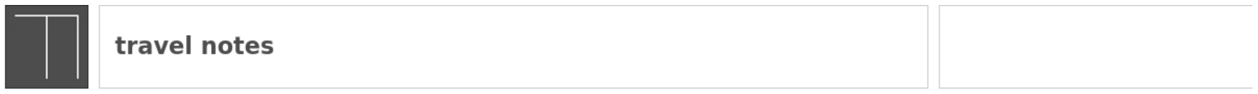


Figure 2: Grid Layout - Updated Header - option 2

HTML5, CSS, & JS - example - part 2

add grid layout - option 1

- update our main content to position the `note-input` and `note-controls`

```
<!-- note input -->
<section class="note-input">
  <div class="row">
    <div class="col-12">
      <h5>add note</h5>
      <input><button>add</button>
    </div>
  <div>
</section>
<!-- note controls for delete... -->
<section class="note-controls">
  <div class="row">
    <div class="col-12">
      <button id="notes-delete">Delete all</button>
    </div>
  </div>
</section>
```

- still need to amend style.css to remove additional fixed styling

---

Image - HTML5, CSS, & JS - grid layout 2

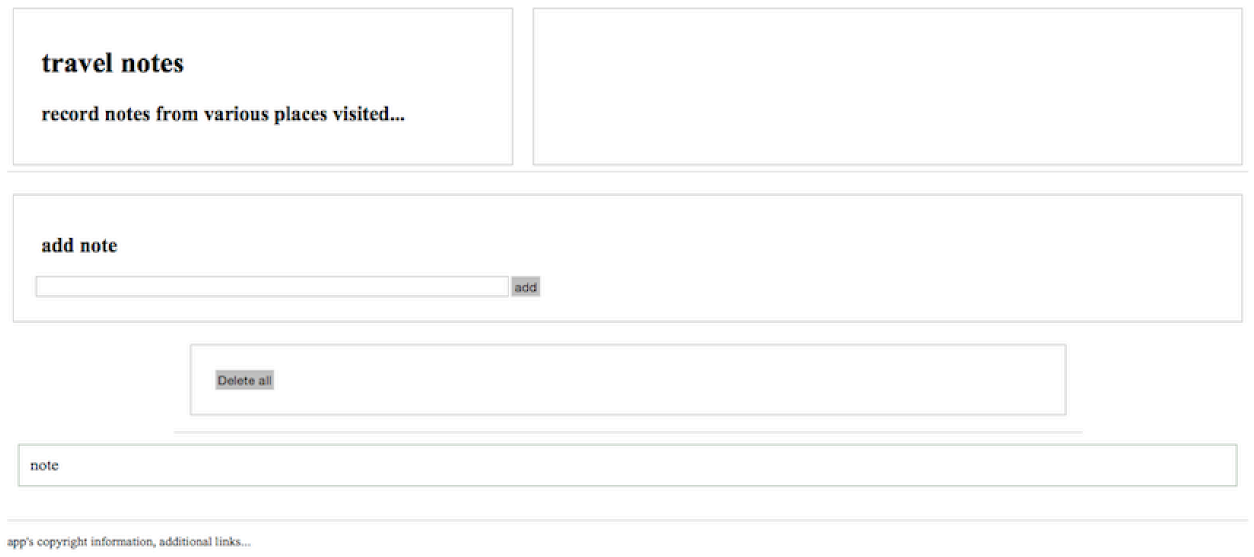

travel notes
record notes from various places visited...
add note
add
Delete all
note
app's copyright information, additional links...

Figure 3: Grid Layout - mixed grid and fixed - option 1

HTML5, CSS, & JS - example - part 2

add grid layout - option 2

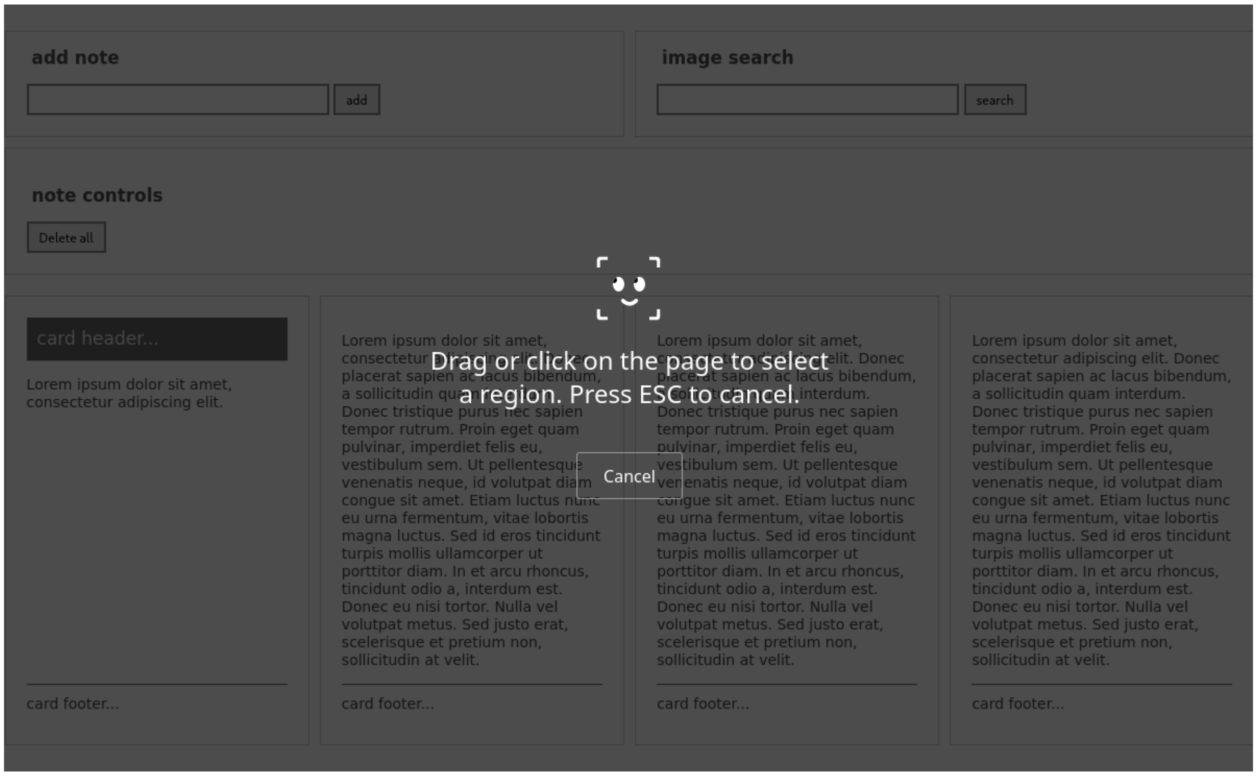
- modify `main` to include unique content

```html
<!-- document main - unique to current page -->
<main class="site-content">
    <div class="page-heading">
        <!-- note input -->
        <section class="note-input">
            <h5>add note</h5>
        <input type="text" id="input-note" />
        <button id="add-note">add</button>
        </section>
        <!-- image search -->
        <section class="image-search">
            <h5>image search</h5>
        <input type="text" id="input-image" />
        <button id="search-images">search</button>
        </section>
        <!-- note controls for delete... -->
        <section class="note-controls">
            <h5>note controls</h5>
        <button id="notes-delete" class="delete-all">Delete all</button>
        </section>
    </div><!-- end of page-heading -->
    <!-- note output -->
    <section class="note-output">
    </section>
</main>
```

- add `page-heading` with sections
  - note-input, image-search, note-controls

- add section for note-output
  - update dynamically with notes

Image - HTML5, CSS, & JS - grid layout 2



Lorem ipsum dolor sit amet, consectetur adipiscing elit.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec placerat sapien ac lacus bibendum, a sollicitudin quam interdum. Donec tristique purus nec sapien tempor rutrum. Proin eget quam pulvinar, imperdiet felis eu, vestibulum sem. Ut pellentesque venenatis neque, id volutpat diam congue sit amet. Etiam luctus nunc eu urna fermentum, vitae lobortis magna luctus. Sed id eros tincidunt turpis mollis ullamcorper ut porttitor diam. In et arcu rhoncus, tincidunt odio a, interdum est. Donec eu nisi tortor. Nulla vel volutpat metus. Sed justo erat, scelerisque et pretium non, sollicitudin at velit.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec placerat sapien ac lacus bibendum, a sollicitudin quam interdum. Donec tristique purus nec sapien tempor rutrum. Proin eget quam pulvinar, imperdiet felis eu, vestibulum sem. Ut pellentesque venenatis neque, id volutpat diam congue sit amet. Etiam luctus nunc eu urna fermentum, vitae lobortis magna luctus. Sed id eros tincidunt turpis mollis ullamcorper ut porttitor diam. In et arcu rhoncus, tincidunt odio a, interdum est. Donec eu nisi tortor. Nulla vel volutpat metus. Sed justo erat, scelerisque et pretium non, sollicitudin at velit.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec placerat sapien ac lacus bibendum, a sollicitudin quam interdum. Donec tristique purus nec sapien tempor rutrum. Proin eget quam pulvinar, imperdiet felis eu, vestibulum sem. Ut pellentesque venenatis neque, id volutpat diam congue sit amet. Etiam luctus nunc eu urna fermentum, vitae lobortis magna luctus. Sed id eros tincidunt turpis mollis ullamcorper ut porttitor diam. In et arcu rhoncus, tincidunt odio a, interdum est. Donec eu nisi tortor. Nulla vel volutpat metus. Sed justo erat, scelerisque et pretium non, sollicitudin at velit.

Drag or click on the page to select a region. Press ESC to cancel.

Figure 4: Grid Layout - mixed grid and fixed - option 2

CSS Basics - cascading rules - part 1

- CSS, or cascading style sheets, employs a set of **cascading** rules
- rules applied by each browser as a ruleset conflict arises
  - e.g. issue of **specificity**

```
p {
  color: blue;
  }
p.p1 {
  color: red;
  }
```

- the more specific rule, the class, will take precedence
- issue of possible duplication in rulesets

```
h3 {
  color: black;
}
```

```css
h3 {
  color: blue;
}
```

- **cascading** rules state the later ruleset will be the one applied
  - blue heading instead of black…

CSS Basics - cascading rules - part 2

- simple styling and rulesets can quickly become compounded and complicated
- different styles, in different places, can interact in complex ways
- a powerful feature of CSS
  - can also create issues with logic, maintenance, and design
- three primary sources of style information that form this cascade
  - 1. default styles applied by the browser for a given markup language
    * e.g. colours for links, size of headings…
  - 2. styles specific to the current user of the document
    * often affected by browser settings, device, mode…
  - 3. styles linked to the document by the designer
    * external file, embedded, and as inline styles per element

CSS Basics - cascading rules - part 3

- basic cascading nature creates the following pattern
  - browser's style will be default
  - user's style will modify the browser's default style
  - styles of the document's designer modify the styles further

CSS Basics - inheritance

- CSS includes inheritance for its styles
- descendants will inherit properties from their ancestors
- style an element
  - descendants of that element within the DOM inherit that style

```css
body {
  background: blue;
}
p {
  color: white;
}
```

- `p` is a descendant of `body` in the DOM
  - inherits background colour of the body
- this characteristic of CSS is an important feature
  - helps to reduce redundancy and repetition of styles
- useful to maintain outline of document's DOM structure
- most styles follow this pattern but not all
- margin, padding, and border rules for block-level elements **not inherited**

CSS Basics - reset options

- to help us reduce browser defaults, we can use a CSS reset
- reset allows us to start from scratch

- customise aspects of the rendering of our HTML documents in browsers
- often considered a rather controversial option
- considered controversial for the following primary reasons
  - accessibility
  - performance
  - redundancy
- use resets with care
- notable example of resets is Eric Meyer
  - discussed reset option in May 2007 blog post
- resets often part of CSS frameworks…

Demo - CSS Reset - Before

Browser default styles are used for

- `<h1>` , `<h3>` , and `<p>`
- Demo - CSS Reset Before

Demo - CSS Reset - After

Browser resets are implemented using the Eric Meyer stylesheet.

- Demo - CSS Reset After

CSS - a return to inline styles

- *inline* styles are once more gaining in popularity
  - helped by the rise of React &c.
- for certain web applications they are now an option
  - allow us to dynamically maintain and update our styles
- their implementation is not the same as simply embedding styles in HTML
  - dynamically generated
  - can be removed and updated
  - can form part of our maintenance of the underlying DOM
- inherent benefits include
  - no cascade
  - built using JavaScript
  - styles are dynamic

CSS - against inline styles

- CSS is designed for styling
  - this is the extreme end of the scale - in effect, styling is only done with CSS
- abstraction is a key part of CSS
  - by separating out concerns, i.e. CSS for styling, our sites are easier to maintain
- *inline* styles are too specific
  - again, abstraction is the key here
- some styling and states are easier to represent using CSS
  - psuedoclasses etc, media queries…
- CSS can add, remove, modify classes

- dynamically update selectors using classes

HTML5, CSS, & JS - example - part 3

add grid layout - option 1

- fix mixed rendering by removing width, margin, and padding for `.note-controls`

```
/* note controls */
.note-controls {
  border-bottom: 1px solid #dedede;
  display: none;
}
```

- continue to update Travel Notes app
  - modify output for notes
  - add further options for users

DEMO - Travel Notes - grid layout with media queries

CSS grid layout - part 1

intro

- grid designs for page layout, components…
  - increasingly popular over the last few years
  - useful for creating responsive designs
- quick and easy to layout a scaffolding framework for our structured content
- create boxes for our content
  - then position them within our grid layout
- content can be stacked in a horizontal and vertical manner
  - creating most efficient layout for needs of a given application
- another benefit of CSS grids is that they are framework and project agnostic
  - thereby enabling easy transfer from one to another
- columns will increase and decrease relative to the size of the browser window
- also set break points in our styles
  - helps to customise a layout relative to screen sizes, devices, aspect ratios…
  - helps us differentiate between desktop and mobile viewers

---

HTML5, CSS, & JS - example - part 3

add grid layout - option 2

- use CSS3 grids to structure page
  - add wrapper for grid in body
- content places for grid structure
  - banner, site-content, site-footer
  - e.g. banner for heading structure

Video - CSS grid

Layout considerations Layout and composition - up to 2:45

Source - Layout and composition - YouTube

CSS3 Grid - intro

- gid layout with CSS is useful for structure and organisation
  - applied to HTML page
- usage similar to table for structuring data
- in its basic form
  - enables developers to add columns and rows to a page
- grid layout also permits more complex, interesting layout options
  - e.g. overlap and layers...
- further information on MDN website,
  - MDN - CSS Grid Layout

CSS3 Grid - general concepts & usage

- grid may be composed of rows and columns
  - thereby forming an intersecting set of horizontal and vertical lines
- elements may be added to the grid with reference to this structured layout

Grid layout in CSS includes the following general features,

- additional tracks for content
  - option to create more columns and rows as needed to fit dynamic content
- control of alignment
  - align a grid area or overall grid
- control of overlapping content
  - permit partial overlap of content
  - an item may overlap a grid cell or area
- placement of items - explicit and implicit
  - precise location of elements &c.
  - use line numbers, names, grid areas &c.
- variable track sizes - fixed and flexible, e.g.
  - specify pixel size for track sizes
  - or use flexible sizes with percentages or new `fr` unit

CSS3 Grid - grid container

- define an element as a grid container using
  - `display: grid` or `display: inline-grid`
- any children of this element become *grid items*
  - e.g.

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

- we may also define other, child nodes as a `grid` container
  - any direct child nodes to a grid container are now defined as grid items

---

CSS3 Grid - what is a grid track?

- rows and columns defined with
  - `grid-template-rows` and `grid-template-columns` properties
- in effect, these define *grid tracks*
- as MDN notes,
  - "a *grid track* is the space between any two lines on the grid.""
  - (https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Grid_Layout/Basic_Concepts_of_Grid_Layout)
- so, we may create both row and column tracks, e.g.

```css
.wrapper {
  display: grid;
  grid-template-columns: 200px 200px 200px;
}
```

- `wrapper` class now includes three defined columns of width 200px
  - thereby creating three tracks
- *n.b.* a track may be defined using any valid length unit, not just `px` ...

HTML5, CSS, & JS - example - part 3

```css

div.wrapper {
    display: grid;
    grid-gap: 0;
    grid-template-rows: 80px auto 80px;
    grid-template-areas:
        "site-banner"
        "site-content"
        "footer";
    margin: 20px 5% 0 5%;
    padding: 0;
    height: calc(99vh - 20px);
}
```

add grid layout - option 2 - wrapper

CSS3 Grid - `fr` unit for tracks - part 1

- CSS Grid now introduces an additional length unit for tracks, `fr`
- `fr` unit represents fractions of the space available in the current grid container
  - e.g.

```css
.wrapper {
  display: grid;
  grid-template-columns: 1fr 1fr 1fr;
}
```

- we may also apportion various space to tracks, e.g.

```css
.wrapper {
  display: grid;
  grid-template-columns: 2fr 1fr 1fr;
}
```

- creates three tracks in the grid
  - but overall space effectively now occupies four parts
  - two parts for `2fr` , and one part each for remaining two `1fr`

CSS3 Grid - `fr` unit for tracks - part 2

- we may also be specific in this sub-division of parts in tracks, e.g.

```
.wrapper {
  display: grid;
  grid-template-columns: 200px 1fr 1fr;
}
```

- first track will occupy a width of `200px`
  - remaining two tracks will each occupy `1` fraction unit

CSS3 Grid - `repeat()` notation for `fr` - part 1

- for larger, repetitive grids, easier to use `repeat()`
  - helps define multiple instances of the same track
  - e.g.

```
.wrapper {
  display: grid;
  grid-template-columns: repeat(4, 1fr);
}
```

- this creates four separate tracks - each defined as `1fr` unit's width

---

CSS3 Grid - `repeat()` notation for `fr` - part 2

- `repeat()` notation may also be used as part of the track definition
  - e.g.

```
.wrapper {
  display: grid;
  grid-template-columns: 200px repeat(4, 1fr) 100px;
}
```

- this example will create
  - one track of `200px` width
  - then four tracks of `1fr` width
  - and finally a single track of `100px` width
- `repeat()` may also be used with multiple track definitions
  - thereby repeating multiple times
  - e.g.

```
.wrapper {
  display: grid;
  grid-template-columns: repeat(4, 1fr 2fr);
}
```

- this will now create eight tracks
  - the first four of width `1fr`
  - and the remaining four of `2fr`

CSS3 Grid - implicit and explicit grid creation

- in the above examples
  - we simply define tracks for the columns
  - and CSS grid will then apportion content to required rows
- we may also define an explicit grid of columns and rows
  - e.g.

```css
.wrapper {
  display: grid;
  grid-template-columns: repeat(2 1fr);
  grid-auto-rows: 150px;
}
```

- this slightly modifies an implicit grid to ensure each row is `200px` tall

---

CSS3 Grid - track sizing

- a grid may require tracks with a minimum size
  - and the option to expand to fit dynamic content
- e.g. ensuring a track does not collapse below a certain height or width
  - and that it has the option to expand as necessary for the content…
- CSS Grid provides a `minmax()` function, which we may use with rows
  - e.g.

```css
.wrapper {
  display: grid;
  grid-template-columns: repeat(2 1fr);
  grid-auto-rows: minmax(150px, auto);
}
```

- ensures each row will occupy a minimum of `150px` in height
  - still able to stretch to contain the tallest content
  - whole row will expand to meet the `auto` height requirements
  - thereby affecting each track in the row

HTML5, CSS, & JS - example - part 3

```css

div.banner {
    grid-area: site-banner;
    display: grid;
    grid-template-columns: 90px 1fr auto;
    grid-template-rows: 80px;
    grid-template-areas:
        "site-logo site-header banner-extras";
}
```

add grid layout - option 2 - banner

HTML5, CSS, & JS - example - part 3

add grid layout - option 2 - banner components

- various nested UI components
- banner
  - logo, site-header, banner-extras

```css
.logo {
    grid-area: site-logo;
    margin: 0;
}

.site-header {
    grid-area: site-header;
    margin: 0 5px 0 0;
    border: 1px solid #ccc;
    padding: 10px;
}

.banner-extras {
    grid-area: banner-extras;
    display: grid;
    grid-template-columns: 150px 150px;
    grid-template-areas:
        "extra-left extra-right";
    margin: 0 0 0 5px;
    border: 1px solid #ccc;
}
```

CSS3 Grid - grid lines

- a grid is defined using *tracks*
  - and not lines in the grid
- created grid also helps us with positioning by providing numbered lines
- e.g. in a three column, two row grid we have the following,
  - four lines for the three vertical columns
  - three lines for the two horizontal rows
- such lines start at the left for columns, and at the top for rows
- *n.b.* line numbers start relative to written script
  - e.g left to right for western, right to left for arabic…

CSS3 Grid - positioning against lines

- when we place an item in a grid
  - we use these lines for positioning, and not the tracks
- reflected in usage of
  - `grid-column-start` , `grid-column-end` , `grid-row-start` , and `grid-row-end` properties.

- items in the grid may be positioned from one line to another
    - e.g. column line 1 to column line 3
- *n.b.* default span for an item in a grid is one track,
    - e.g. define column start and no end - default span will be one track...
    - e.g.

```
.content1 {
  grid-column-start: 1;
  grid-column-end: 4;
  grid-row-start: 1;
  grid-row-end: 3;
}
```

CSS3 Grid - grid cell & grid area

grid cell

- a *cell* is the smallest unit on the defined grid layout
- it is conceptually the same as a cell in a standard table
- as content is added to the grid, it will be stored in one cell

grid area

- we may also store content in multiple cells
    - thereby creating *grid areas*
- grid areas must be rectangular in shape
- e.g. a grid area may span multiple row and column tracks for required content

---

CSS3 Grid - add some gutters

- gutters may be created using the *gap* property
    - available for either column or row
    - `column-gap` and `row-gap`
    - e.g.

```
.wrapper {
  display: grid;
  grid-template-columns: repeat(4, 1fr 2fr);
  column-gap: 5px;
  row-gap: 10px;
}
```

- *n.b.* any space used for gaps will be determined prior to assigned space for `fr` tracks

---

CSS3 Grid - working examples

- grid basic - page zones and groups
- grid basic - article style page
- grid layout - articles with scroll

HTML5, CSS, & JS - example - part 3

```
.site-content {
    grid-area: site-content;
    display: grid;
    grid-template-areas:
        "page-heading"
        "content";
}
```

add grid layout - option 2 - site content

HTML5, CSS, & JS - example - part 3

add grid layout - option 2 - site content components

- main app structure and components
- page-heading grouping for grid structure
  - note-input, image-search
  - note-controls

```
.note-input {
    grid-area: add-note;
    margin: 10px 5px 0 0;
    border: 1px solid #ccc;
    padding: 0 20px 20px 20px;
}

.image-search {
    grid-area: search-images;
    margin: 10px 0 0 5px;
    border: 1px solid #ccc;
    padding: 0 20px 20px 20px;
}

.note-controls {
    grid-area: note-controls;
    margin: 10px 0 0 0;
    border: 1px solid #ccc;
    padding: 20px;
}
```

- note-input & image-search rendered as 50/50 split
- note-controls moved to separate row in page-heading

HTML5, CSS, & JS - example - part 3

```
.site-footer {
    grid-area: footer;
    margin: 0;
    border-top: 1px solid #dddddd;
}
```

add grid layout - option 2 - site footer

- site banner and footer rendered equivalent to fixed
  - main site content uses internal scroll for page

CSS3 Grid - sample layouts

intro

- grid layout enables more complex and interesting layout options
  - overlap, layers…
- sample layouts using CSS grid structure
  - common layout options and designs
  - useful repetition of design
  - modify base layouts for various site requirements
- sample layouts
  - responsive layouts
  - auto placement for dynamic content and media
  - platform agnostic designs
  - useful with SPA, SVG, async patterns &c.

HTML5, CSS, & JS - example - part 4

add flex to grid layout

- an additional option to consider - flex layouts
  - aims to provide efficient way to align and proportion content
- known as **Flexbox Layout**
  - idea is to apportion width and height for content
  - proportions relative to container even when their size is unknown or dynamic
- flex layout could, in theory, replace a full grid layout
  - considered more a complement to overall grid structure
- defined flex container expands items to fill the container's available space
  - can also shrink them to prevent any possible overflow
- think of a flex layout as supporting multiple directions
  - direction agnostic
- many properties available for **flex**
  - focus upon styling flex container and any flex items

CSS - Flexbox

intro

- helps solve many issues that have continued to plague layout and positioning
- used with HTML elements and components
  - both client-side and cross-platform apps
- a few issues it tries to solve
  - vertical and horizontal alignment
  - defining a centred position for child elements relative to their parent
  - equal spacing and proportions for child nodes regardless of available space
  - equal heights and widths for varied content
  - & lots more…

CSS - Flexbox

basic usage

- for any app layout, we need to define specific elements as *flexible boxes*
- i.e. those allowed to use flexbox in a given app
  - e.g.

```css
section {
  display: flex;
}
```

- ruleset will define a `section` element as a parent flex container
  - child elements may now accept flex declarations
- initial declaration, `display: flex`
  - also includes default values for flexbox layout of child elements
- e.g. `<div>` elements in a section
  - by default now arranged as equal sized columns with the same initial height

---

CSS - Flexbox

axes

- elements arranged using flexbox are laid out on two axes
- main axis
  - axis running in the direction of the currently laid out flex items
  - e.g. rows or columns
  - start and end of axis = *main start* & *main end*
- cross axis
  - axis running perpendicular to the current main axis
  - start and end of axis = *cross start* & *cross end*
- each child element laid out inside flex container called a *flex item*

CSS - Flexbox

flex direction

- set a property for the flex direction
  - defines direction of flex items relative to main axis
  - i.e. layout direction for child elements
- default setting is `row`
  - direction will be relative to current browser language setting
  - e.g. for English language browsers = left to right

```css
section {
  flex-direction: column;
}
```

- override the default `row` setting
  - arrange child items in a column

```css
section {
  display: flex;
  flex-direction: column;
}
```

- ensures child flex items were laid out in a single column
- then override specific `section` elements
  - allow child flex items in a `row` direction

```
#tabs {
  flex-direction: row;
}
```

Image - CSS Flexbox

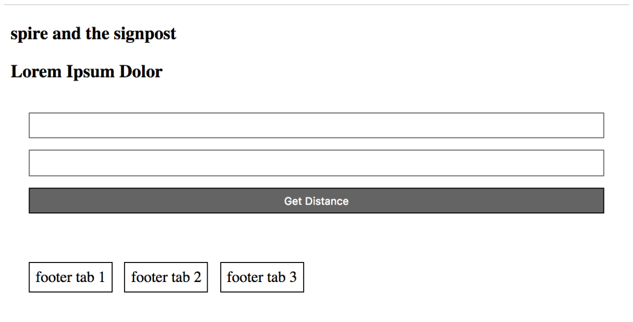


Figure 5: CSS Flexbox - flex direction

flex direction

CSS - Flexbox

flex item wrapping

- ensure child items do not overlap their parent flex container
  - add a declaration for `flex-wrap` to a required ruleset
  - e.g.

```
#tabs {
  flex-direction: row;
  flex-wrap: wrap;
}
```

Image - CSS Flexbox

without wrap

Image - CSS Flexbox

with wrap

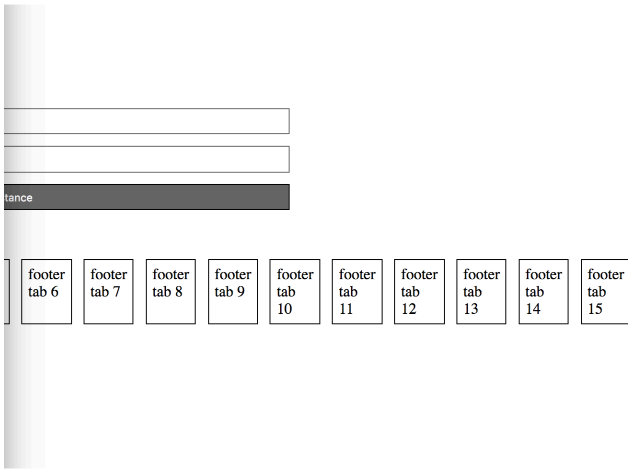


Figure 6: CSS Flexbox - no flex wrap

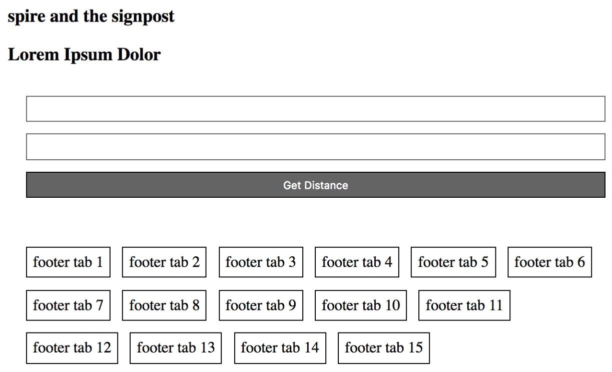


Figure 7: CSS Flexbox - flex wrap

Video - Flexbox

flexible design Examples of Modular UI Design

Source - Modular UI Design - YouTube

HTML5, CSS, & JS - example - part 5

add flex to grid layout - option 1

- we might specify CSS properties for a flex container

```
.flex-container {
    display: flex; /* defines container as flex */
    flex-direction: row; /* defines positioning of items added to container */
    flex-wrap: wrap; /* defines whether to wrap items to another line */
    justify-content: flex-start; /* defines start point and distribution of items */
}
```

- allows us to position our container starting at the left
  - items contained in a row
  - contained items wrapping to additional lines if necessary
- many additional options available for each property
- also add rulesets for specific styling of items within a flex container
- we could add properties to a flex item such as
  - specify the order of the flex items
  - whether a particular item can grow or shrink relative to content
  - default size of an item before any remaining space is distributed
  - individual alignment for a given item…

CSS - Flexbox

flex direction reverse

- also set flex direction to reverse
  - starts flex items from the right on an English language browser

```
#tabs {
  flex-direction: row-reverse;
  flex-wrap: wrap;
}
```

Image - CSS Flexbox

flex direction reverse

CSS - Flexbox

`flex-flow` shorthand

- also combine *direction* and *wrap* into a single declaration

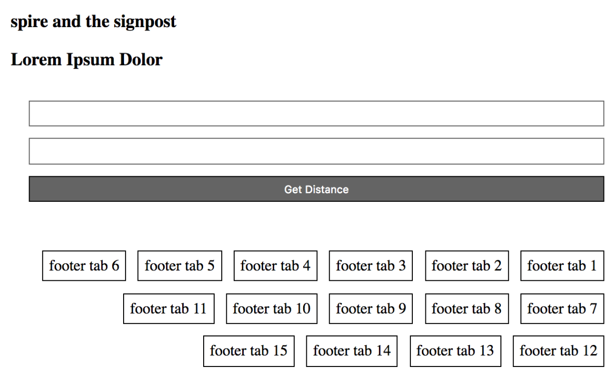


Figure 8: CSS Flexbox - flex direction reverse

- `flex-flow`
- now contain values for both `row` and `wrap`
- e.g.

```
#tabs {
  flex-flow: row wrap;
}
```

HTML5, CSS, & JS - example - part 6

add flex to grid layout - option 2

- flex container for option 2 design

```
/* note container - flex */
.note-output {
    display: flex;
    justify-content: space-between;
    flex-wrap: wrap;
    row-gap: 20px;/*applies to rows of items - not above first row... */
    padding-top: 20px;
}
```

- output notes section
  - organise single notes as flex items
  - add gap between rows of flex items
- justify content in container
  - notes start at left edge, end at right edge
  - space between evenly apportioned per note

CSS - Flexbox

sizing of flex items

- for each flex item, we may need to specify apportioned space in the layout
  - e.g. set space as an equal proportion for each flex item
  - we may add the following to a child item ruleset

```
div.fTab {
  flex: 1;
}
```

- defines each child flex item `<div class="fTab">`
    - occupy an equal amount of space within the given row
    - after considering margin and padding
- **n.b.** this value is proportional
    - doesn't matter if the value is 1 or 100 &c.
- define additional flex proportions for specific child items
    - e.g.

```
div.fTab:nth-child(odd) {
  flex: 2;
}
```

- each odd *flex-item* will now occupy twice available space
    - space in the current direction

Image - CSS Flexbox

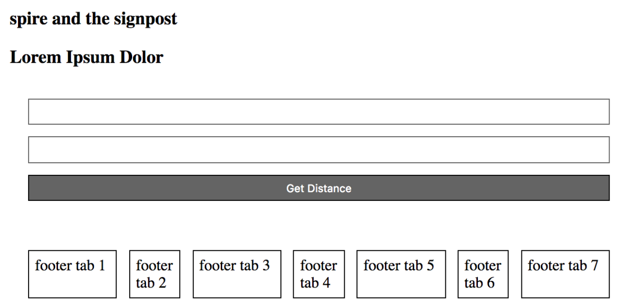


Figure 9: CSS Flexbox - flex item sizing

flex item sizing

CSS - Flexbox

minimum size

- then set a minimum size for a flex item
    - e.g.

```
div.fTab {
  flex: 1 100px;
}
```

- or a relative unit for the size

```
div.fTab {
  flex: 1 20%;
}
```

- each flex item will initially be given a minimum

  - e.g. `20%` of the available space
  - the remaining space will be defined relative to proportion units

---

Image - CSS Flexbox

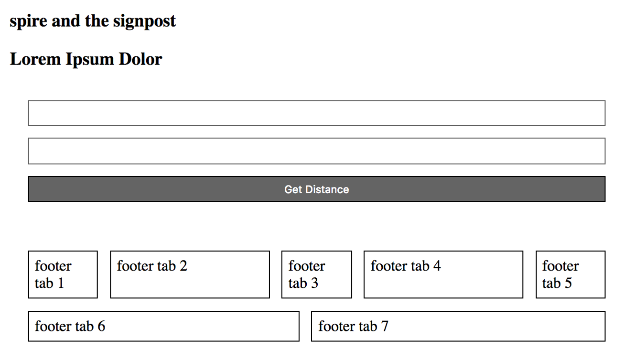


Figure 10: CSS Flexbox - flex item sizing - minimum size

flex item sizing

HTML5, CSS, & JS - example - part 7

add flex to notes

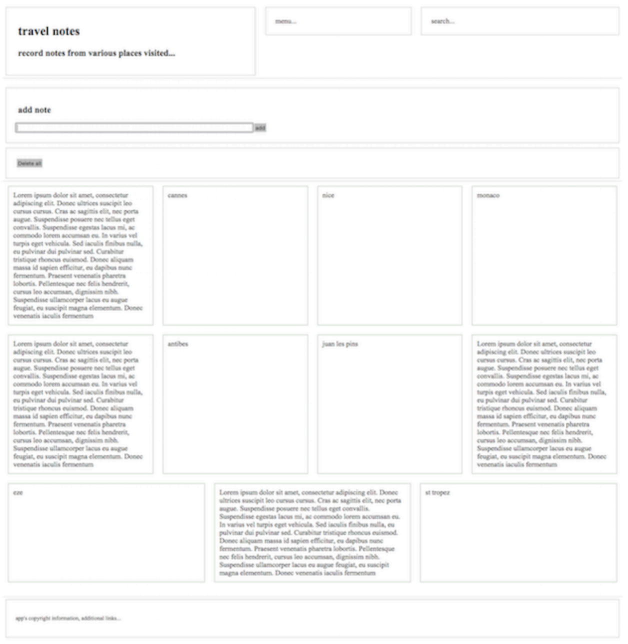
- flex container and items useful for organising and positioning our notes
- due to uncertainty about content, size, and general note requirements
  - flex positioning and styling removes the need for assumptions or fixed sizes
- we can start to modify the styling and rendering of our notes using flex

```
/* flex item */
.flex-item {
  flex-basis: 300px; /* default size before extra */
  flex-grow: 1; /* all items will be equal */
}
```

- gives us a default smallest size for each note
- then the ability for each note to grow to fill the row as required
- also work with responsive layouts
  - due to the minimum size and the option to grow for each item
  - and wrap flex items per flex container
- modify and update styles as we develop travel notes app

DEMO - Travel Notes - grid layout with flex notes

Image - HTML5, CSS, & JS - Flex Notes


travel notes

record notes from various places visited...

menu...

search...

add note

add

Delete all

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec ultrices suscipit leo cursus cursus. Cras ac sagittis elit, nec porta augue. Suspendisse posuere nec tellus eget convallis. Suspendisse egestas lacus mi, ac commodo lorem accumsan eu. In varius vel turpis eget vehicula. Sed iaculis finibus nulla, eu pulvinar dui pulvinar sed. Curabitur tristique rhoncus euismod. Donec aliquam massa id sapien efficitur, eu dapibus nunc fermentum. Praesent venenatis pharetra lobortis. Pellentesque nec felis hendrerit, cursus leo accumsan, dignissim nibh. Suspendisse ullamcorper lacus eu augue feugiat, eu suscipit magna elementum. Donec venenatis iaculis fermentum

cannes

nice

monaco

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec ultrices suscipit leo cursus cursus. Cras ac sagittis elit, nec porta augue. Suspendisse posuere nec tellus eget convallis. Suspendisse egestas lacus mi, ac commodo lorem accumsan eu. In varius vel turpis eget vehicula. Sed iaculis finibus nulla, eu pulvinar dui pulvinar sed. Curabitur tristique rhoncus euismod. Donec aliquam massa id sapien efficitur, eu dapibus nunc fermentum. Praesent venenatis pharetra lobortis. Pellentesque nec felis hendrerit, cursus leo accumsan, dignissim nibh. Suspendisse ullamcorper lacus eu augue feugiat, eu suscipit magna elementum. Donec venenatis iaculis fermentum

antibes

juan les pins

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec ultrices suscipit leo cursus cursus. Cras ac sagittis elit, nec porta augue. Suspendisse posuere nec tellus eget convallis. Suspendisse egestas lacus mi, ac commodo lorem accumsan eu. In varius vel turpis eget vehicula. Sed iaculis finibus nulla, eu pulvinar dui pulvinar sed. Curabitur tristique rhoncus euismod. Donec aliquam massa id sapien efficitur, eu dapibus nunc fermentum. Praesent venenatis pharetra lobortis. Pellentesque nec felis hendrerit, cursus leo accumsan, dignissim nibh. Suspendisse ullamcorper lacus eu augue feugiat, eu suscipit magna elementum. Donec venenatis iaculis fermentum

eze

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec ultrices suscipit leo cursus cursus. Cras ac sagittis elit, nec porta augue. Suspendisse posuere nec tellus eget convallis. Suspendisse egestas lacus mi, ac commodo lorem accumsan eu. In varius vel turpis eget vehicula. Sed iaculis finibus nulla, eu pulvinar dui pulvinar sed. Curabitur tristique rhoncus euismod. Donec aliquam massa id sapien efficitur, eu dapibus nunc fermentum. Praesent venenatis pharetra lobortis. Pellentesque nec felis hendrerit, cursus leo accumsan, dignissim nibh. Suspendisse ullamcorper lacus eu augue feugiat, eu suscipit magna elementum. Donec venenatis iaculis fermentum

st tropez

app's copyright information, additional links...

Figure 11: Grid Layout - flex notes

Image - HTML5, CSS, & JS - Flex Notes 2

Image - HTML5, CSS, & JS - Flex Notes 3

HTML5, CSS, & JS - example - part 8

add flex to notes

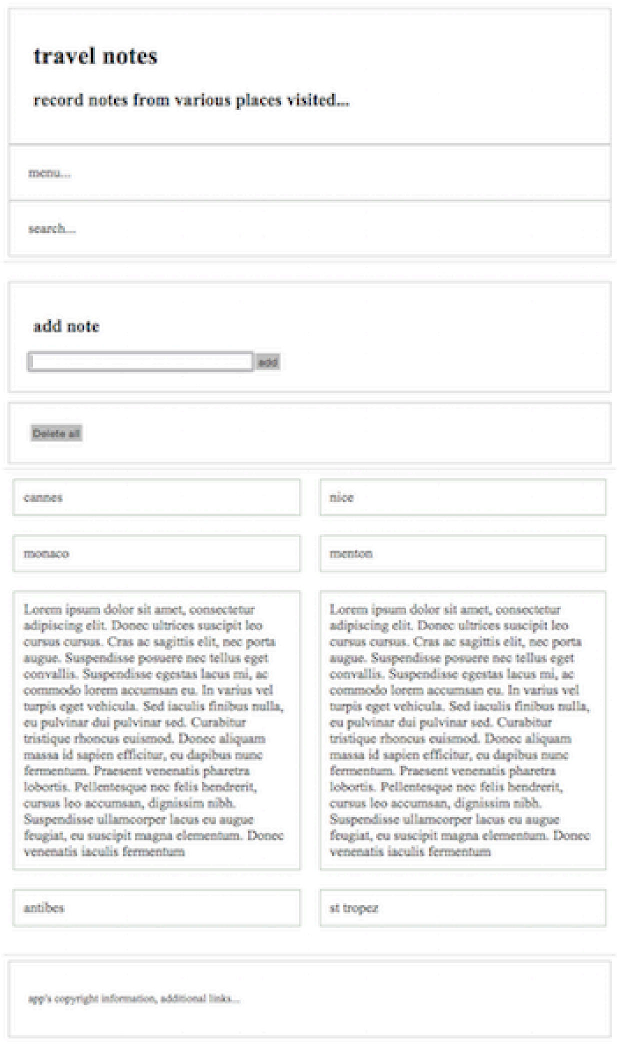
- Notes with Flex and Media Queries

HTML5, CSS, & JS - example - part 9

add flex to notes - option 2

- define styling for flex items in option 2 design
- note defined using card layout design
  - card-view, card-content

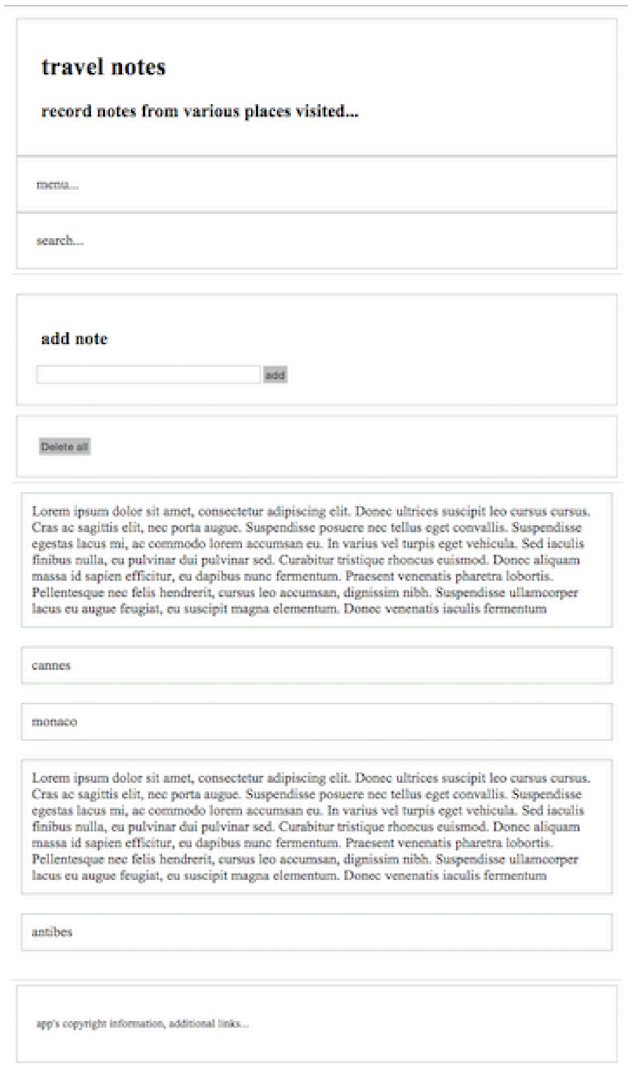
```css
/* note card - flex */
.card-view {
  display: flex;
  flex-direction: column;
    flex: 0 0 250px;
    border: 1px solid #CCCCCC;
    padding: 20px;
}
.card-content {
```


travel notes
record notes from various places visited...
menu...
search...
add note
add
Delete all
cannes
nice
monaco
menton
antibes
st tropez
app's copyright information, additional links...

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec ultrices suscipit leo cursus cursus. Cras ac sagittis elit, nec porta augue. Suspendisse posuere nec tellus eget convallis. Suspendisse egestas lacus mi, ac commodo lorem accumsan eu. In varius vel turpis eget vehicula. Sed iaculis finibus nulla, eu pulvinar dui pulvinar sed. Curabitur tristique rhoncus euismod. Donec aliquam massa id sapien efficitur, eu dapibus nunc fermentum. Praesent venenatis pharetra lobortis. Pellentesque nec felis hendrerit, cursus leo accumsan, dignissim nibh. Suspendisse ullamcorper lacus eu augue feugiat, eu suscipit magna elementum. Donec venenatis iaculis fermentum

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec ultrices suscipit leo cursus cursus. Cras ac sagittis elit, nec porta augue. Suspendisse posuere nec tellus eget convallis. Suspendisse egestas lacus mi, ac commodo lorem accumsan eu. In varius vel turpis eget vehicula. Sed iaculis finibus nulla, eu pulvinar dui pulvinar sed. Curabitur tristique rhoncus euismod. Donec aliquam massa id sapien efficitur, eu dapibus nunc fermentum. Praesent venenatis pharetra lobortis. Pellentesque nec felis hendrerit, cursus leo accumsan, dignissim nibh. Suspendisse ullamcorper lacus eu augue feugiat, eu suscipit magna elementum. Donec venenatis iaculis fermentum

Figure 12: Grid Layout - flex notes - medium



travel notes

record notes from various places visited...

menu...

search...

add note

add

Delete all

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec ultrices suscipit leo cursus cursus. Cras ac sagittis elit, nec porta augue. Suspendisse posuere nec tellus eget convallis. Suspendisse egestas lacus mi, ac commodo lorem accumsan eu. In varius vel turpis eget vehicula. Sed iaculis finibus nulla, eu pulvinar dui pulvinar sed. Curabitur tristique rhoncus euismod. Donec aliquam massa id sapien efficitur, eu dapibus nunc fermentum. Praesent venenatis pharetra lobortis. Pellentesque nec felis hendrerit, cursus leo accumsan, dignissim nibh. Suspendisse ullamcorper lacus eu augue feugiat, eu suscipit magna elementum. Donec venenatis iaculis fermentum

cannes

monaco

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec ultrices suscipit leo cursus cursus. Cras ac sagittis elit, nec porta augue. Suspendisse posuere nec tellus eget convallis. Suspendisse egestas lacus mi, ac commodo lorem accumsan eu. In varius vel turpis eget vehicula. Sed iaculis finibus nulla, eu pulvinar dui pulvinar sed. Curabitur tristique rhoncus euismod. Donec aliquam massa id sapien efficitur, eu dapibus nunc fermentum. Praesent venenatis pharetra lobortis. Pellentesque nec felis hendrerit, cursus leo accumsan, dignissim nibh. Suspendisse ullamcorper lacus eu augue feugiat, eu suscipit magna elementum. Donec venenatis iaculis fermentum

antibes

app's copyright information, additional links...

Figure 13: Grid Layout - flex notes - small

```
  flex: 1;
}
```

- card is flex container for child flex items
  - e.g. note content, header, footer &c.
- `flex` defines shorthand property
  - flex-grow, flex-shrink, flex-basis
  - note set to initial length of `250px`

CSS - Flexbox

flex item alignment

- Flexbox allows us to define alignment for flex items in each flex container
  - relative to the main and cross axes
- e.g. we might want to specify a centred alignment for flex items

```
#tabs {
  flex-direction: row;
  flex-wrap: wrap;
  align-items: center;
}
```

- `align-items: center`
  - causes flex item in flex container to be centred along the cross axis
  - however, they'll still maintain their basic dimensions
- also modify value for `align-items` to either `flex-start` or `flex-end`
- such values will align flex items to either start or end of cross axis

---

CSS - Flexbox

override align per flex item

- as with `flex`
  - also override alignment per flex item
  - using `align-self` property add a value for positioning
- e.g. a sample declaration might be as follows

```
div.fTab:nth-child(even) {
  flex: 2;
  align-self: flex-end;
}
```

CSS - Flexbox

justify content for flex item

- also specify `justify-content` for flex items in a flex container
  - property allows us to define position of a flex item relative to main axis
- default value is `flex-start`
- then modify it relative to one of the following
  - `flex-end`

    - `center`
    - `space-around`
        * distributes each flex item evenly along main axis with space at either end
    - `space-between`
        * same as `space-around` without space at either end...

CSS - Flexbox

alignment and order - part 1

- define alignment relative to each axis using a specific declaration
    - e.g. for the main we may use `justify-content`
    - for the cross axis we use `align-items`
- also modify layout order of flex items
    - without directly changing underlying source order
- use the following pattern to specify order

```css
div.fTab:first-child {
  order: 1;
}
```

- first flex item will now move to the end of the tab list

Image - CSS Flexbox

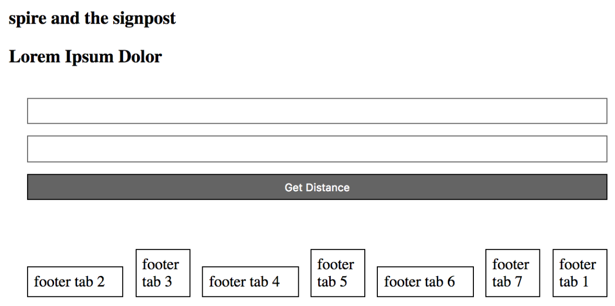


Figure 14: CSS Flexbox - flex item order 1

flex item order

CSS - Flexbox

alignment and order - part 2

- due to default order for flex items
    - by default, all flex items have an `order` value set to `0`
- higher the `order` value, later the item will appear in the list &c.
- items with the same order will revert to the order in the source code
- also possible to ensure certain items will always appear first
    - or at least before default `order` values

    - by using a negative value for the `order` declaration
    - e.g.

```
div.fTab:last-child {
  order: -1;
}
```

CSS - Flexbox

nesting flex containers and items - part 1

- Flexbox can also be used to create nested patterns and structures
    - e.g. we may set a flex item as a flex container for its child nodes
- we might add a banner to the top of a page

```
<section id="banner">
  <header id="page-header">
    <h3>spire and the signpost</h3>
    <h5>point to the stars...</h5>
  </header>
  <section id="search">
    <input type="text" id="searchBox"/>
    <button id="searchBtn">Search</button>
  </section>
</section>
```

CSS - Flexbox

nesting flex containers and items - part 2

- set `#banner` , `#page-header` , and `#search` as flex containers
    - e.g.

```
#search {
  display: flex;
}
```

- then specify various declarations for `#search`
    - e.g.

```
#search {
  display: flex;
  flex-direction: row;
  flex: 2;
  align-self: flex-start;
}
```

- includes values for itself and any child elements
    - if we then add some rulesets for the nested flex items
    - e.g.

```
#searchBox {
  flex: 4;
}
```

```
#searchBtn {
  flex: 1;
}
```

- we get a simple proportional split of `4:1` for the input field to the button

---

Image - CSS Flexbox

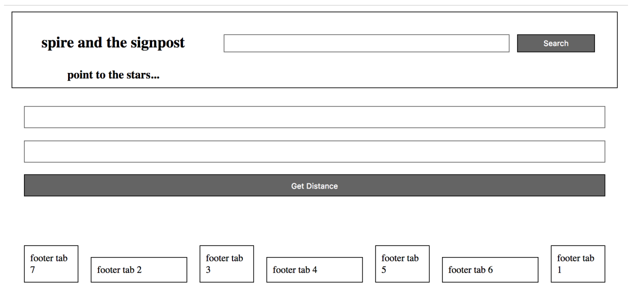


Figure 15: CSS Flexbox - nested flex containers

nested flex containers

HTML5, CSS, & JS - example - part 10

add flex to notes - option 2

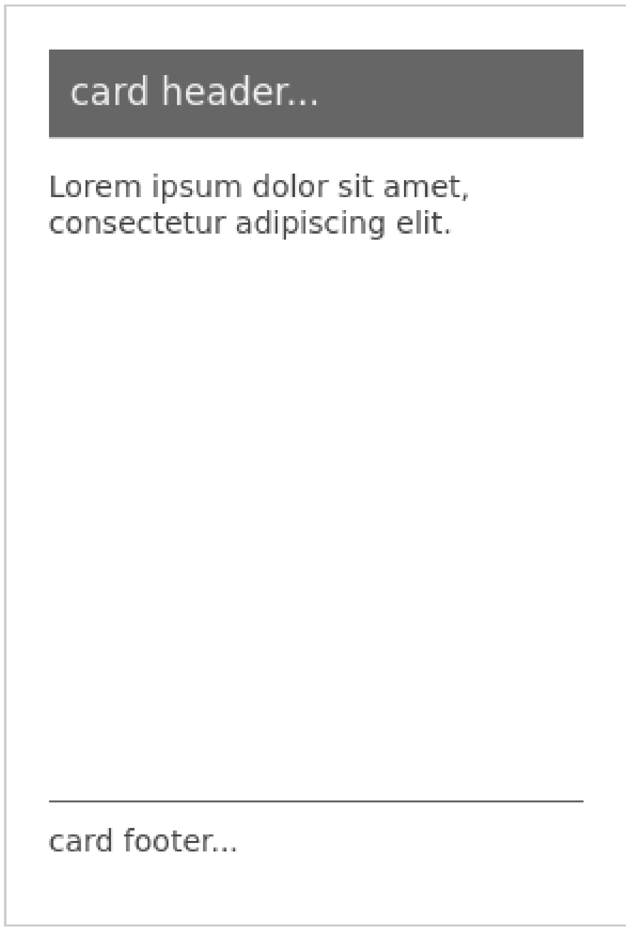
- define rulesets for child items
  - card-view header
  - card-view footer

```
.card-view header {
    padding: 10px;
    background-color: #666666;
    color: #EEEEEE;
    font-size: 17px;
}
.card-view footer {
    border-top: 1px solid #666666;
    padding: 10px 0;
}
```

- DEMO - Travel Notes - Version 3 - Grid

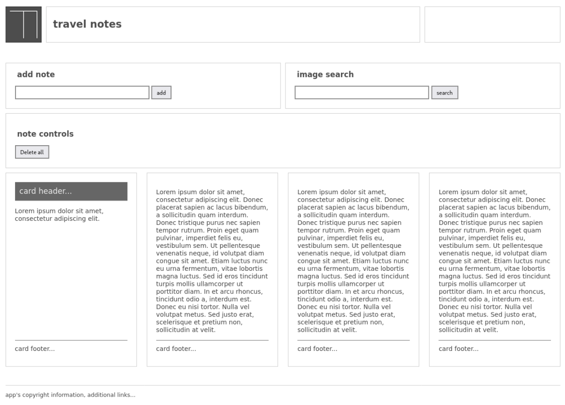
Image - HTML5, CSS, & JS - Flex Notes

Image - HTML5, CSS, & JS - Flex Notes



Lorem ipsum dolor sit amet, consectetur adipiscing elit.

Figure 16: Grid Layout - flex notes - card design



Lorem ipsum dolor sit amet, consectetur adipiscing elit.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec placerat sapien ac lacus bibendum, a sollicitudin quam interdum. Donec tristique purus nec sapien tempor rutrum. Proin eget quam pulvinar, imperdiet felis eu, vestibulum sem. Ut pellentesque venenatis neque, id volutpat diam congue sit amet. Etiam luctus nunc eu urna fermentum, vitae lobortis magna luctus. Sed id eros tincidunt turpis mollis ullamcorper ut porttitor diam. In et arcu rhoncus, tincidunt odio a, interdum est. Donec eu nisi tortor. Nulla vel volutpat metus. Sed justo erat, scelerisque et pretium non, sollicitudin at velit.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec placerat sapien ac lacus bibendum, a sollicitudin quam interdum. Donec tristique purus nec sapien tempor rutrum. Proin eget quam pulvinar, imperdiet felis eu, vestibulum sem. Ut pellentesque venenatis neque, id volutpat diam congue sit amet. Etiam luctus nunc eu urna fermentum, vitae lobortis magna luctus. Sed id eros tincidunt turpis mollis ullamcorper ut porttitor diam. In et arcu rhoncus, tincidunt odio a, interdum est. Donec eu nisi tortor. Nulla vel volutpat metus. Sed justo erat, scelerisque et pretium non, sollicitudin at velit.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec placerat sapien ac lacus bibendum, a sollicitudin quam interdum. Donec tristique purus nec sapien tempor rutrum. Proin eget quam pulvinar, imperdiet felis eu, vestibulum sem. Ut pellentesque venenatis neque, id volutpat diam congue sit amet. Etiam luctus nunc eu urna fermentum, vitae lobortis magna luctus. Sed id eros tincidunt turpis mollis ullamcorper ut porttitor diam. In et arcu rhoncus, tincidunt odio a, interdum est. Donec eu nisi tortor. Nulla vel volutpat metus. Sed justo erat, scelerisque et pretium non, sollicitudin at velit.

app's copyright information, additional links...

Figure 17: Grid Layout - flex notes - card view with space between

CSS grid layout - part 8

media queries

- often need to consider a mobile-first approach
- introduction of CSS3, we can now add **media queries**
- modify specified rulesets relative to a given condition
    - eg: screen size for a desktop, tablet, and phone device
- media queries allow us to specify a breakpoint in the width of the viewport
    - will then trigger a different style for our application
- could be a simple change in styles
    - such as colour, font etc
- could be a modification in the grid layout
    - effective widths for our columns per screen size etc…

Image - Grid Layout 4

grid test 2 - gutters
app's copyright information, additional links...

Figure 18: Grid Layout - Media Queries

CSS3 Grid - responsive layout

intro

- display a layout with a variety of patterns and structures, e.g.
  - single column for a phone
  - add a sidebar for a tablet of lower window resolution
  - full width view with dual sidebars &c.
- use responsive designs and structures for various games, media playback…
- responsive works with variety of markup
  - e.g. transform SVG designs

CSS3 Grid - responsive layout

page structure

- start with a sample page structure for a HTML page

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>CSS Grid - Responsive Layout</title>
    <link rel="stylesheet" type="text/css" href="./assets/style.css">
  </head>
  <body>
    <div class="wrapper">
            ...
        </div>
    </body>
</html>
```

CSS3 Grid - responsive layout

page structure - HTML5

- add some HTML5 markup for a `header` , `navigation` , `footer` , and some `main` content

```
<div class="wrapper">
    <header class="site-header">
        <h3>Spire & the Signpost</h3>
        <h5>Shine through the gloom, and point to the stars...</h5>
    </header>
    <nav class="site-nav">
        <ul>
            <li><a href="">Home</a></li>
            <li><a href="">Charts</a></li>
            <li><a href="">Data</a></li>
            <li><a href="">Views</a></li>
        </ul>
    </nav>
    <!-- use aside for tangentially related content for parent section... -->
    <aside class="content-side">
        <header>
            <h5>sidebar...</h5>
        </header>
    </aside>
    <main class="content">
        <article class="content-article">
            <header class="article-header">
                <h5>Welcome</h5>
            </header>
            <p>...</p>
        </article>
    </main>
    <section class="site-links">
        <h6>social links...</h6>
    </section>
    <footer class="site-footer">
        <h6>footer...</h6>
    </footer>
</div>
```

- demo - basic responsive

CSS3 Grid - responsive layout

CSS and structure - part 1

- for the page structure
  - need to define some template areas for our grid in the CSS
  - e.g.

```
/* CONTENT */
.content {
    grid-area: content;
}
```

- use such template area names
  - defined with the `grid-area` property
  - define a layout for the overall page or part of a page

CSS3 Grid - responsive layout

CSS and structure - part 2

- template areas may then be used with the parent for the grid structure
  - e.g. `wrapper` for the overall page

```
.wrapper {
    display: grid;
    grid-gap: 10px;
    grid-template-areas:
        "site-header"
        "site-nav"
        "content-side"
        "content"
        "site-links"
        "site-footer"
}
```

- `wrapper` class will display as a grid
  - with a gap between each area of the grid
  - has a single column in this example
  - includes the required order for the grid areas

CSS3 Grid - responsive layout

define media query

- current example would be suitable for a collapsed phone view
  - single column view
  - will also render for other resolutions and devices
- then add a media query for alternative layouts and displays
  - may be triggered using a check of current width for screen
  - check width of window…

```css
/* min 700 */
@media (min-width: 700px) {
    .wrapper {
        grid-template-columns: 1fr 3fr;
        grid-template-areas:
        "site-header site-header"
        "site-nav site-nav"
        "content-side content"
        "site-links site-footer"
    }
}
```

CSS3 Grid - responsive layout

specific media query

- add further media queries to handle various rendering requirements
  - e.g. add `height` property to fix footer at bottom of page

```css
@media (min-width: 700px) {
    .wrapper {
        grid-template-columns: 1fr 3fr;
        grid-template-rows: 120px 60px calc(98vh - 240) 60px;
        grid-template-areas:
        "site-header site-header"
        "site-nav site-nav"
        "content-side content"
        "site-links site-footer";
        height: 98vh;
    }
}
```

- specify height of current *viewport* using a relative unit, `vh`

- add `grid-template-rows` to define known heights for three of the four rows
- add a variant height for the main content
  - main content is only given a height corresponding to available space in viewer window
  - height achieved using the `calc()` function
- demo - responsive with specific media query

Resources

- MDN - CSS3 Grid
- W3 Schools - CSS Grid View
- Example Responsive UI Designs - YouTube
- MDN - CSS3 Grid
- Modular UI Design - YouTube
- W3 Schools - CSS Grid View
- MDN - CSS Flexbox
- W3 Schools - CSS Flexbox
- Various
  - Example Responsive UI Designs - YouTube
  - MDN - CSS3 Grid
  - Modular UI Design - YouTube