

Front-End Development

Labo 5 – Responsive Web Design, Grid & Transitions

J. Pieck, L. Van Houdt, S. Van Peborgh, T. Van Nuland

1 Contents

2 INTRODUCTION 3

3 SETTING UP STRUCTURE 4

4 MENU ANIMATION 5

4.1 LINK COLOR TRANSITION 5
4.2 LINK UNDERSCORE TRANSITION 6
4.3 MOBILE NAVIGATION 6

5 GRID 10

5.1 BASIC GRID 10
5.2 RESPONSIVE GRID 13

6 NEXT STEPS 21

7 ZIP-FILE INLEVEREN 21

8 REFLECTION 21

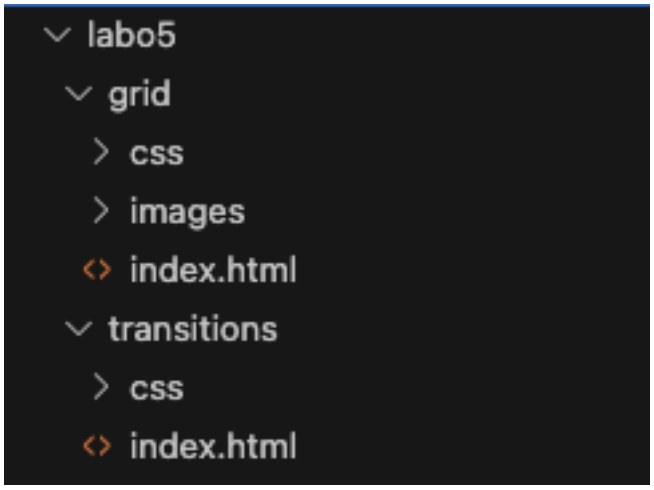
2 Introduction

Documentation	Teaching materials on Toledo: slides and code examples
Purpose	In this lab, we build on our existing CSS knowledge. We are introduced to a new component: CSS Grid. In addition, we explore the use of transitions and go deeper into responsive web design. The goal is to make web pages accessible and user-friendly on different screen sizes.
Learning Goals	At the end of this lab you will be able to: • Use CSS transitions to add simple animation effects to elements. • Apply CSS Grid to create flexible and clean layouts. The use of grid area to place elements concretely and determine their size. • Understanding the difference between Flexbox and Grid and using both in a targeted way. • Realize responsive web design using media queries, optimized grid and smoothly responsive configurations. • Modify an existing web page so that it works correctly and easily on different screen sizes.

3 Setting up structure

1. Create a project folder for this lab, for example `labo5/`.
2. Download and unzip the Toledo launch file in this folder.
3. Open `labo5/` in vscode

Your folder structure looks like this:



Use the Live Server extension in VS Code, which you already installed in lab 1, to open `transitions/index.html` in your browser. You will see a very simple, unstyled web page. With every change to your CSS file, the page will automatically refresh.

4 Menu Animation

In this exercise, you will create a modern navigation bar that uses CSS transitions, hover and click states, and that works responsively on different screen sizes.

4.1 Link Color Transition

- Open the project
  - Navigate to `transitions/index.html` and open this file in VS Code.
  - Launch the Live Server so you can view the page in your browser.
- Check the initial situation
  - In your browser, you should see a navigation bar with `<a>` link elements.
  - The links don't have an animation yet when you hover over them.



- Customize the CSS
  - Open the file `transitions/css/styles.css`.
  - Locate the existing CSS rules for `.nav links a`.
  - Add a new line for `.nav links a:hover`.

```
.nav-links a {
  ...
  transition: color 0.3s ease;
}

.nav-links a:hover {
  color: #ff7f50;
}
```

- Test your work

Your made your first transition!
When you hover over the navigation links, the hover state is activated. This starts a color transition: the link animates smoothly from the original color to the new color. The transition uses an ease interpolation and lasts 0.3 seconds.



What other values can you use instead of `ease` as an interpolation in a CSS transition?

4.2 Link Underscore Transition

Open the transitions `/css/styles.css` file and add the following lines at the bottom:

```
.nav-links a::after {
  content: "";
  position: absolute;
  left: 0;
  bottom: -4px;
  width: 0%;
  height: 2px;
  background: #ff7f50;
  transition: width 0.3s ease;
}

.nav-links a:hover::after {
  width: 100%;
}
```



Test your work in the browser and note what change this configuration makes:



Explain in your own words what the `::after` pseudo selector does?



This code creates an animation where a colored line appears under a link as soon as you hover over it, without any additional HTML elements. A typical application is with navigation bars or menus on modern websites.

Now experiment with a few other transitions yourself by means of hover:

- background
- border

4.3 Mobile navigation

Try this out first:

- Make your browser window narrower.
- You notice that the navigation disappears as soon as there is not enough space.
- The menu doesn't really disappear, but due to `flex` with `space-between` and the lack of wrapping, logo and links remain next to each other. As soon as the total width is larger than the screen, the `<a>` on the left slide out of the picture.

Solution:
A well-known way to solve this problem is to use a hamburger menu. The name comes from the shape of the icon, which resembles a stacked hamburger.

Step 1: Customize HTML
In `transitions/index.html` within the `<nav>` add the following element:

```
<div class="menu-toggle">&#9776;</div>
```



The code `☰` is an HTML character reference to unicode character for 3 horizontal lines

Step 2: Add CSS

In `transitions/css/styles.css`, add the following code:

```
.menu-toggle {
  display: none;
  font-size: 2rem;
  cursor: pointer;
}

@media (max-width: 48rem) {
  .menu-toggle {
    display: block;
  }
}
```



A media query (e.g. `@media(max-width: 768px)`) is a way in CSS to make your style rules dependent on the properties of the device or screen on which the website is viewed.

With media queries, you can not only check the screen width, but also things like orientation (portrait or landscape) or whether someone is using dark mode.

In our example, you'll now see that the hamburger icon appears as soon as the screen width is less than 768px. By default, this icon is not visible because the display has: none.

For smaller screens, the icon becomes visible by display: block.

Note: clicking on the icon does not yet show a menu. That's because clicking is an event that is normally handled with JavaScript (we'll get to that later!).

To simulate this purely with CSS, we are going to make the menu visible via a hover event.

Step 3: Expand CSS

Add the following code within the media query in `transitions/css/styles.css`:

```
.nav-links {
  display: none;
  flex-direction: column;
  background: #333;
  position: absolute;
  top: 70px;
  right: 0;
  width: 200px;
  padding: 1rem;
}
```



The current code does not work yet. We are still missing a pseudoclass configuration for the hover state of the navbar: when we hover over the navbar, .nav links should become visible by adjusting the display value. You can derive the correct value from the previously set properties.



Reflection: Explain in your own words how the .nav links will look on a mobile screen size.



This code puts the navigation links with `position: absolute` in the right place for mobile screens. By default, they are not visible because `display: none` is set. When you add the `:hover` pseudo-class, the display is adjusted, making the links visible again. Thanks to the use of `absolute`, the links also remain within the viewport.

5 Grid

In this assignment, you will create a modern CSS grid layout that automatically adjusts to the screen size. You create a section with multiple blocks. Each element contains a profile picture, a name, and a short text.

5.1 Basic Grid

- Open the project
  - Navigate to `grid/index.html` and open this file in VS Code.
  - Launch the Live Server so you can see the result in your browser.
- Add CSS to the grid `/css/styles.css file`
  - Copy the following code to provide the web page with a default styling and a grid layout

```
/* Default Styling */
body {
  font-family: system-ui, sans-serif;
  margin: 0;
  padding: 2rem;
  background: #f7f8fb;
  color: #222;
}

.testimonial {
  background: white;
  border-radius: 0.75rem;
  padding: 1rem;
  Box-Shadow: 0 2px 6px rgba(0, 0, 0, 0.08);
  text-align: center;
  transition: transform 0.3s ease, box-shadow 0.3s ease;
}

.testimonial img {
  width: 80px;
  height: 80px;
  object-fit: cover;
  border-radius: 50%;
  margin-bottom: 0.75rem;
  transition: box-shadow 0.3s ease;
```

```
}

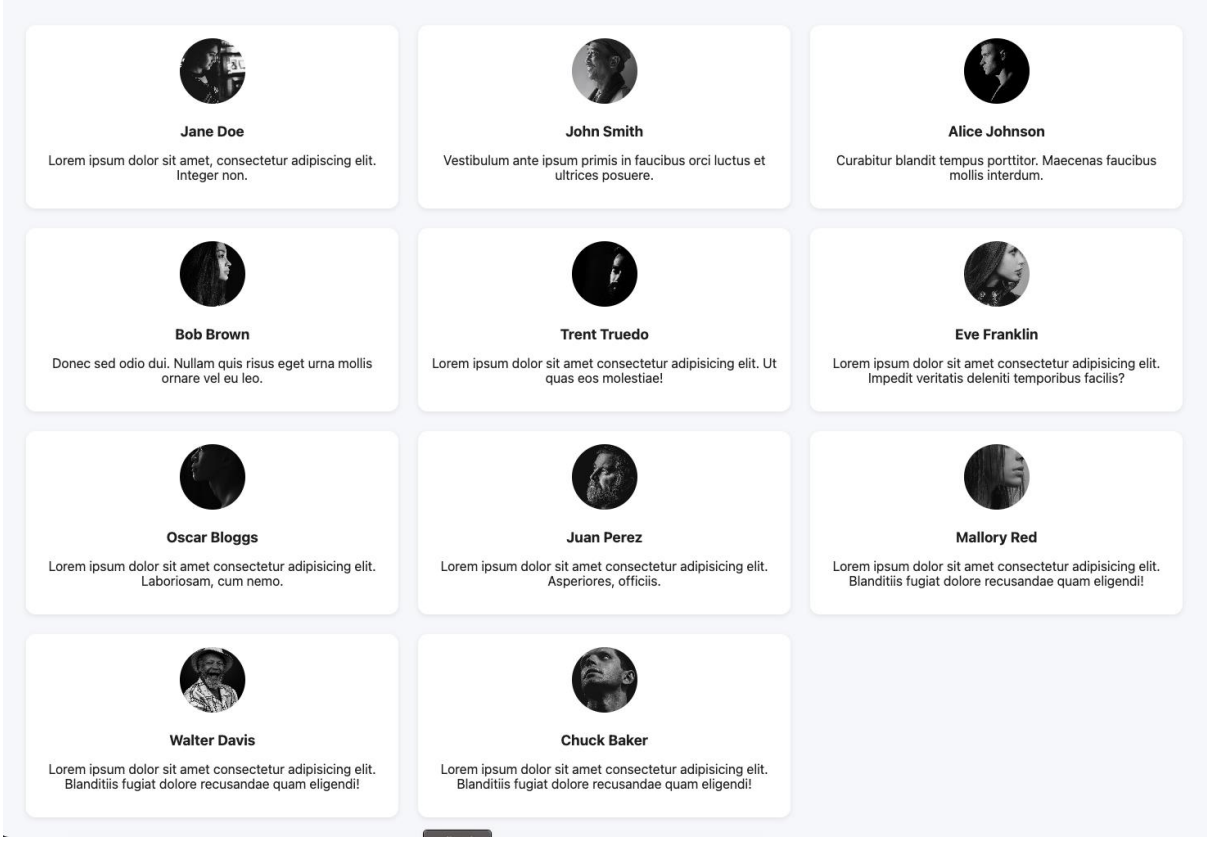
.testimonial h3 {
  margin: 0.5rem 0;
  font-size: 1.1rem;
}

.testimonial:hover {
  transform: translateY(-5px);
  Box-Shadow: 0 8px 20px RGB (0, 0, 0, 0.15);
}

.testimonial:hover img {
  box-shadow: 0 0 8px darkgray;
  transform: scale(1.1);
  transition: transform 0.3s ease, box-shadow 0.3s ease;
}

/* Grid Configuration */
.testimonials {
  display: grid;
  gap: 1.5rem;
  grid-template-columns: repeat(3, 1fr);
}
```

If everything is set up correctly, you should see the following result:



We were able to successfully configure a `CSS Grid` with the following properties:

```
display: grid
```

configures the CSS element as a grid.

```
gap: 1.5rem
```

provides some extra space between the cells

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

Divides the container into three equal columns. The unit fr (fractional unit) refers to a portion of the available space: after subtracting fixed widths, the remaining space is dynamically distributed among the columns.



What happens if you use `1fr 2fr 1fr`?



`1fr 1fr 1fr` can also be written as `repeat(3, 1fr)`. For the rest of this exercise, we use the repeat() notation, because it is clearer and gives less repetition in the code.

5.2 Responsive Grid

Reduce the width of your browser window to a minimum. Imagine that you are viewing this on a mobile device: this does not provide a good user experience, because all the text is too close together.

In this section, we're going to make the grid configuration responsive with a mobile-first approach.



```
Mobile first means that you first design and develop a
website for small screens (such as smartphones), and then
expand it to larger screens (tablets, laptops, desktops).

So you start with a simple, clear layout.

With media queries, you can add extra styles as the
screen gets bigger.
```

Mobile view
For our mobile implementation, we want to show only one column. This can be done in different ways and flexbox is certainly the right choice here. However, to keep the lab simple, we opt for a one-dimensional grid. We can modify the code by:

- delete the existing grid-template-columns configuration, or
- explicit grid-template-columns: 1fr; (this is the default value).

Because we take a mobile-first approach, we want to approach these settings as standard. So we don't provide a separate media query for the smallest supported screen size.

Desktop view
From a screen width of 75rem, we want to show three columns. To do this, add the following media query in `grid/css/styles.css`:

```
@media (min-width: 75rem) {
  .testimonials {
    grid-template-columns: repeat(3, 1fr);
  }
}
```



Test this configuration? Is this the most optimal user experience (UX)?

Delete the previous media query and replace it with the following two lines:

```
@media (min-width: 38rem) {
  .testimonials {
    grid-template-columns: repeat(2, 1fr);
  }
}

@media (min-width: 54rem) {
  .testimonials {
    grid-template-columns: repeat(3, 1fr);
  }
}
```

Congratulations, we have created a mobile-first design that adapts to the screen size step by step.

- Under 38em(phone) → the layout shows 1 column (default).
- From 38em (tablet) → the layout shows 2 columns.
- From 54em (desktop) → the layout shows 3 columns.

5.3 Grid Area

So far, we have created our layout by defining the number of columns and adjusting them responsively with media queries. This effect can also be achieved with a flexbox with wrapping configuration. The real strength of CSS Grid lies in the ability to precisely position elements using grid areas, without having to add additional HTML or wrapper elements.

We are now going to expand our existing testimonial section so that:

- Some blocks take up more space,
- The layout changes on different screen sizes,
- The HTML structure remains unchanged.

Step 1: Assign Grid Areas to existing features

In `grid/css/styles.css` you add at the bottom:

```
.testimonial:nth-child(1) { grid-area: one; }
.testimonial:nth-child(2) { grid-area: two; }
.testimonial:nth-child(3) { grid-area: three; }
.testimonial:nth-child(4) { grid-area: four; }
.testimonial:nth-child(5) { grid-area: five; }
```



This gives each of the first five elements a name within the grid.

Step 2: Replace standard grid layout (mobile first)

Customize your .testimonials container to take advantage of these areas:

```
.testimonials {
  display: grid;
  gap: 1.5rem;
  grid-template-columns: 1fr;
  grid-template-areas:
    "one"
    "two"
    "three"
    "four"
    "five"
    "six";
  padding-block: 2rem;
  width: min(95%, 70rem);
  margin-inline: auto;
}
```



All testimonials are still listed below each other by default (mobile first).

Step 3: Expand layout for larger screens, tablet first.

Add media queries to rearrange the layout **without changing the HTML**:

```
@media (min-width: 38rem) {
  .testimonials {
    grid-template-columns: repeat(3, 1fr);
    grid-template-areas:
      "one one five"
      "two three five"
      "four four six";
  }
}
```

Your layout will now automatically rearrange according to the screen width. Some testimonials take up more columns, others shift position.

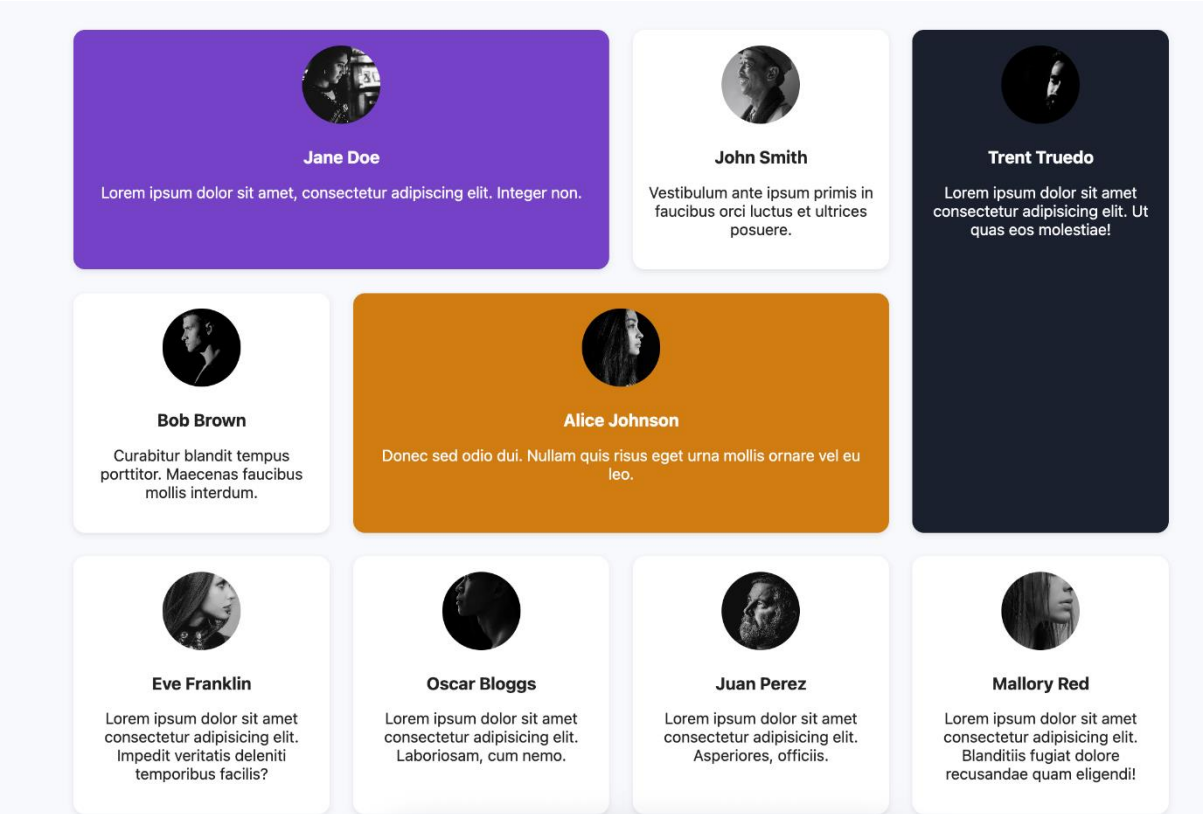
Step 4: Optimize for desktop environment:

```
@media (min-width: 54rem) {
  .testimonials {
    grid-template-columns: repeat(4, 1fr);
    grid-template-areas:
      "one one two five"
      "three four four five";
}
```

Extra tip: to make important elements stand out, you can style them visually:

```
.testimonial:nth-child(1) {
  background: hsl(263 55% 52%);
  color: white;
}
.testimonial:nth-child(4) {
  background: hsl(35 95% 42%);
  color: white;
}
.testimonial:nth-child(5) {
  background: hsl(219 29% 14%);
  color: white;
}
```

If correct, you will now see the following:



5.4 Bonus: Future of Responsive Web Design

The following section is optional and is not part of the compulsory subject matter. We do want to prepare you for what responsive web design will look like in the future.

While a media query is a plausible solution, **CSS Grid** (and CSS in general) also offers opportunities to create a responsive design without media queries. This can be done with `auto-fit`, `minmax()` and `clamp()`. The configuration of this is a bit more complex, but we will go through this step by step.

Step 1: Making padding smoothly responsive
With the following configuration below, the padding switches with the screen size:

- On small screens, the padding is never less than 0.75rem.
- On large screens, the padding can grow up to 1.25rem.
- In between, the padding scales proportionally with the width of the screen (2vw = 2% of the viewport width).

```
.testimonial {
 ..
 padding: clamp(0.75rem, 2vw, 1.25rem);
 ..
}
```

Step 2: Customize profile picture
We can also handle the height and width of the profile picture in a similar way so that it also behaves responsively.

```
.testimonial img {
  ..
  width: clamp(64px, 8vw, 96px);
  height: clamp(64px, 8vw, 96px);
  ..
}
```

Step 3: Configure Grid with auto-fit
Finally, we configure the grid with an auto-fit setting. This allows the grid to automatically fill as many columns as can fit in the available space.

```
.testimonials {
  display: grid;
  grid-template-columns: repeat(
    auto-fit,
    minmax(min(22rem, 100%), 1fr)
```

```
  );
  max-width: 80rem;
  margin-inline: auto;
  padding-inline: clamp(1rem, 3vw, 2rem);
  gap: clamp(1rem, 2vw, 1.5rem);
}
```

- `Gap` and `padding-inline` are in accordance with previous configurations, they make the configurations smoothly responsive.

- `repeat(auto-fit, minmax(min(22rem, 100%), 1fr))`: the grid automatically creates as many columns as fit. This automatically creates 1, 2, 3 or more columns, without media queries.
    - `minmax(min(22rem, 100%), 1fr)` means:
        - The column may shrink to a maximum of 100% (so 1 column on narrow screens), only expands from a minimum of 22REM wide, and can grow to 1FR (equal distribution of the remaining space).
- `max-width`: limits the total width of the grid container to 80REM (approximately 1280px at default setting). This prevents the content from becoming too wide on large screens, so that the readability remains better.

- `Margin-inline: Auto` ensures that the container is centered horizontally.
    - `inline` = left and right in the writing direction.
    - `Auto` distributes the remaining space evenly.

With these settings, the grid automatically scales with the screen width, the content remains neatly centered and easy to read, and **no media queries are** required.

6 Next steps

Continue experimenting with CSS Grid and CSS transitions:

- Look up Aligning items in CSS grid layout and apply them to some examples.
- Experimenting transitions: e.g. scale, rotate, and opacity in hover.
- https://cssgridgarden.com/

7 ZIP-file Inleveren

- Rename the main directory "labo5" as "naam_voornaam_labo5"
  e.g. "Doe_John_labo5"
- Via windows explorer (or finder on mac) compress the folder to a zip file.
- No .gzip, .7zip or .rar!
- You should now have a zip file naam_voornaam_labo5.zip
- You upload this file to the Toledo assignment

8 Reflection



When to use a `display: grid` versus `display: flex`?



What did you find most difficult in this lab? Which concepts are not yet entirely clear?