Talk

[Slide]

Hello, my name is Lauren Paul. I am a Web developer at University Relations. Most of the websites that I build are in the university CMS, Terminal Four. Building websites in T4 is interesting because it adds another layer of strategy for providing good user experience.

We often talk about a good user experience for our end visitors: the prospective students and parents, current students, faculty and staff, etc.

But we also need to provide a good experience for the people in T4 managing their content.

[Slide]

Overview

This talk is about how a Web design system using CSS Grid may help create a better experience for content creators in T4, and how to it could also provide some additional benefits to our end users.

I will not be teaching how to code CSS Grid. There are many great resources out there from real experts. **Just a disclaimer:** I am learning CSS Grid, I am not an expert.

I have only used CSS Grid on a small scale. I have not used CSS Grid at the global design system scale I am proposing in this talk. I still have a lot to learn before I am comfortable suggesting that this approach is the best. But as I'm learning about what CSS Grid can do, I am seeing opportunities for improving the systems we are building.

I will be covering what CSS Grid is, why use it, why use it with T4, and what supports it as well as how to handle older browsers.

[Slide]

Who is this talk for?

This talk is mainly for Web designers and T4 developers. I will be discussing some Web design and development strategies that are on the technical side, so apologies in advance.

But ultimately, CSS Grid will help the everyday T4 user. Those who:

- create content for T4
- update content in T4
- and make everyday decisions on how to deliver content to website visitors

While T4 content managers may not need to know the foundation for how the website is built, it might be beneficial for everyone to learn the different strategies we need to consider when building a design system in T4.

[Slide]

What is CSS Grid?

CSS Grid is a new specification for Web design layout that provides a two-dimensional grid for an element and it's child elements. It is part of the CSS language we use to style websites. I will be calling it CSS Grid throughout this talk, to prevent confusion when I talk about just a grid.

Two-dimensional means that CSS Grid can specify a grid with both rows

[Slide]

and columns.

[Slide]

The rows and columns form a grid, and elements can be placed in the grid as needed.

[Slide]

Elements can be placed in a single cell,

[Slide]

span all columns to fit the width of the space,

[Slide]

span all rows to fit the height of the space,

[Slide]

and they can even share the space of other elements, if overlapping is something you are going for.

[Slide]

You can also do cool things like set a column to a fixed width, then assign the remaining columns flexible widths relative to the remaining space.

In case you are wondering why you'd choose CSS Grid over Flexbox (which is another new Web design technology), Flexbox is one dimensional, so it assigns styles for rows, or columns. But not both. Flexbox is incredibly useful, and I use it a lot, but it has limitations when creating a full page layout.

In the past, Web designers and developers have used imperfect methods for Web page layouts, such as:

[Slide: Warning! Web jargon!]

- Positioning
- Floats and clear
- Display: table
- and (EEK!) actual tables.

We had hacks, or used javascript, to fix the imperfections and strongarm the Web design to do what visually made sense. This was often at the expense of proper Web semantics, clean and lean code, or even achieving the layout we were going for.

[Slide]

Why use CSS Grid?

- Solves layout problems
- Creates mixed grids that can be redefined
- Elements can be reordered in CSS
- Reduces code bloat
- And it removes dependency on framework grid systems

[Slide]

CSS Grid solves many layout problems these older techniques couldn't solve, like equal height columns.

[Slide]

Another benefit is that you can create a mixed grid with elements spanning rows and columns for one screen size,

[Slide]

then use media queries to redefine the grid for another screen size.

Not only that, you can reorder the grid's elements without touching the HTML. This keeps the Web document semantic.

Because you can keep the Web semantics and structure separate from the visual presentation, this reduces the amount of code in the HTML, keeping it lean and clean.

To achieve a grid layout, we used to use frameworks like Bootstrap that needed classes specific to the grid framework. Now that we have CSS Grid to handle this in the CSS, we have removed our dependency on framework grid systems. We can have one less http request, speeding up the load time on your website.

[Slide]

So why am I excited about using CSS Grid with T4?

I think it will:

- Improve T4 user experience
- Reduce T4 content types and navigation objects
- Easier to maintain

I've been thinking of ways of using CSS Grid to improve the T4 user's experience. One problem I've been trying to solve is: how can a T4 user add a grid element, such as a staff member listing in a three column grid, without having to wrap those elements in a container that defines the row.

[Slide]

Currently, I build grid content types without CSS Grid, and they require a wrapper. To achieve this, a T4 navigation object call is required to grab those grid element content types and wrap them in a container. This only works if the grid content types are hidden. The T4 navigation object will make the grid items visible once they are put in the wrapper.

I don't like having an extra step in order to create a grid of items. It's also confusing when the T4 user adds a content type to a new page, and misses the step to add the T4 navigation object call. We want to avoid having to do this

I've seen other developers use javascript to grab a group of grid elements and wrap them in a container. This removes the burden from the T4 user, but it relies on javascript for presentation.

[Slide]

If the whole design system uses CSS Grid, you remove the need to wrap a group of grid items in a wrapper. The whole page (or at least the part that contains your content) will be a grid, and the content types will be assigned grid placement in the CSS.

This means a T4 user can add a content type for a grid element, and it will just work!

Also, because you no longer have to use a T4 navigation object to call that grid item, you may not need a unique T4 content type built for that content at all! The T4 developer could create one T4 content type with basic text editor and a place to add a CSS Class. The CSS Class will determine the layout for that content piece.

[Slide]

This slide shows a generic content type built by my colleague, Jesse Oremland. It has just a text editor and a field for a class.

If this kind of content type existed for a T4 site, the T4 content manager would need to understand the classes available in the design system, and how to choose the one that fits their layout needs. This information could be provided with site-specific training and/or by referencing their styleguide.

We want to reduce the amount of content types and navigation objects in T4. The less items in T4, the faster T4 will be!

[Slide]

Support for CSS Grid

If you are wondering why I haven't yet implement many of these ideas, it is because it is a new technology. I've been trying to find the time to learn about CSS Grid, but it's been slow going for me. I was also concerned about Web browser support for this new technology, and how to account for older browsers with no support.

[Slide]

Current browser support: All modern browsers support CSS Grid, except IE11 and Opera Mini.

IE11 supports an older version of CSS Grid, so it's considered partially supported. IE11 will never be updated to the modern CSS Grid specifications, but this browser will soon be obsolete.

So what happens when a Web visitor is on an old browser that doesn't support CSS Grid? If there is no other CSS that provides an alternative layout, the grid elements stack vertically.

In the Web industry, this is perfectly acceptable. Visitors using old browsers are usually minimal, and due to the emerging new CSS techniques that are frequently used, they are probably used to seeing simpler layouts. The goal is to have the content available, even if the layout is simplified.

Opera mini is a mobile Web browser, so the stacked layout should not be much of a problem.

Personally, I also do not want to encourage anyone using an old browser as they can be a security risk. Update your Web browsers, everyone!

If CSS Grid reordered the elements, these older browsers will not follow those rules, so the re-ordered elements will be in the same order as the semantic content structure.

You can use other CSS techniques, like display: table, floats and clear, or flexbox, to provide fallback layouts, but the technique you choose will vary depending on a number of factors. We don't have time in this talk to go over these fallbacks, and again, I am not an expert, so at the bottom of this presentation I provide a link to a really good article by Rachel Andrew.

[Slide]

I also have links to the CSS Grid specification, the MDN Web Docs CSS Grid guides, and a great CSS Grid resource site, also by Rachel Andrew.

[Slide]

I would also love to keep this conversation going with the VCU Web community, so meet me on Slack! This talk came out of a Slack conversation, and it's a good place for everyone to bounce ideas off of each other and learn together.

Thanks everyone, see you next time!