Hello, my name is Tatyana and I will make the presentation about Responsive Web Design, Flexbox and CSS Grid Layout. Let's start with Responsive Web Design.

**What is Responsive Web design?**

Responsive Web Design is about using HTML and CSS to automatically resize a website to make it look good on all devices (desktops, tablets, and phones).

In product design we should follow the “mobile first” rule. The “mobile first” approach tells us that the first step is designing the optimal view for mobile devices so that we can then use media queries to adapt layouts to larger devices.

We have main ingredients of RWD. These are viewport, **grid-view, flexible images and media queries.**

## The first ingredient of RWD is viewport.

Before tablets and mobile phones appeared, web pages were designed for computer screens only, and it was common for web pages to have a static design and a fixed size.

Then, when we started surfing the internet using tablets and mobile phones, fixed size web pages were too large to fit the viewport. To fix this, browsers on those devices scaled down the entire web page to fit the screen. HTML5 introduced a method to let web designers take control over the viewport by means of <meta> tag.

You should include the following <meta> viewport element in all your web pages.

The <meta> viewport element gives the browser instructions how to control the page's dimensions and scaling.

The width=device-width part sets the width of the page to follow the screen-width of the device (which will vary depending on the device).

The initial-scale=1.0 part sets the initial zoom level when the page is first loaded by the browser.

**The second ingredient of RWD is grid-view.**

Many web pages are based on a grid-view, which means that the page is divided into columns**.** Using a grid-view is very helpful when designing web pages. It makes it easier to place elements on the page.

A responsive grid-view often has 12(twelve) columns, and has a total width of 100%, and will shrink and expand as you resize the browser window.

**The third ingredient of RWD is flexible images.**

One of the most challenging things in responsive web design is dealing with images. Responsive images are images that scale nicely to fit any browser size.

If the width property is set to a percentage and the height is set to "auto", the image will be responsive and scale up and down.

If the max-width property is set to 100%, the image will scale down if it has to, but never scale up to be larger than its original size.

The fourth ingredient of RWD is media queries. With media queries you can define completely different styles for different browser sizes.

**Ok, the next topic is Flexbox and Grid Layout**

[CSS Grid Layout](https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Grid_Layout) and Flexbox generally act in the same way with regards to overwriting other methods. We might want to use flexbox as a fallback for grid layout, as there is better support for flexbox in older browsers.

**Flex and Grid – what’s the difference?**

A common question is to ask what the difference is between Flexbox and Grid Layout — why do we have two specifications that sometimes appear to be doing the same thing?

First we used tables, then floats, positioning and inline-block, but all of these methods were essentially. Flexbox helped out, but it's intended for simpler one-dimensional layouts, not complex two-dimensional ones (Flexbox and Grid actually work very well together).

Grid is the very first CSS module created specifically to solve the layout problems we've all been hacking our way around for as long as we've been making websites.

**Flexible Box Layout**

The Flexible Box Layout Module, makes easier to design flexible responsive layout structure without using float or positioning.

**Please have a look at this figure, explaining the main idea behind the Flex layout.**

* main axis - The main axis of a flex container is the primary axis along which flex items are laid out. Beware, it is not necessarily horizontal; it depends on the flex-direction property (see below).
* main-start | main-end - The flex items are placed within the container starting from main-start and going to main-end.
* main size - A flex item's width or height, whichever is in the main dimension, is the item's main size. The flex item's main size property is either the ‘width’ or ‘height’ property, whichever is in the main dimension.
* cross axis - The axis perpendicular to the main axis is called the cross axis. Its direction depends on the main axis direction.
* cross-start | cross-end - Flex lines are filled with items and placed into the container starting on the cross-start side of the flex container and going toward the cross-end side.
* cross size - The width or height of a flex item, whichever is in the cross dimension, is the item's cross size. The cross size property is whichever of ‘width’ or ‘height’ that is in the cross dimension.

**To start using the Flexbox model you need to define a flex container at first.**

The flex container becomes flexible by setting the display property to flex.

**The flex container properties are:** [flex-direction](https://www.w3schools.com/css/css3_flexbox.asp#flex-direction), [flex-wrap](https://www.w3schools.com/css/css3_flexbox.asp#flex-wrap), [flex-flow](https://www.w3schools.com/css/css3_flexbox.asp#flex-flow), [justify-content](https://www.w3schools.com/css/css3_flexbox.asp#justify-content), [align-items](https://www.w3schools.com/css/css3_flexbox.asp#align-items), [align-content](https://www.w3schools.com/css/css3_flexbox.asp#align-content).

The flex-direction property defines in which direction the container wants to stack the flex items (column, column-reverse, row, row-reverse). The column value stacks the flex items vertically (from top to bottom). The row value stacks the flex items horizontally (from left to right).

The flex-wrap property specifies whether the flex items should wrap or not, if there is not enough room for them on one flex line. The wrap value specifies that the flex items will wrap if necessary (wrap, nowrap, wrap-reverse).

The flex-flow property is a shorthand property for setting both the flex-direction and flex-wrap properties.

The justify-content property is used to align the flex items (center, flex-start, flex-end, space-around, space-between).

The align-items property is used to align the flex items vertically (center, flex-start, flex-end, stretch, baseline).

The align-content property is used to align the flex lines (center, space-around, space-between, stretch, flex-start, flex-end).

The direct child elements of a flex container automatically becomes flexible (flex) items.

**The flex item properties are:** [order](https://www.w3schools.com/css/css3_flexbox.asp#order), [flex-grow](https://www.w3schools.com/css/css3_flexbox.asp#flex-grow), [flex-shrink](https://www.w3schools.com/css/css3_flexbox.asp#flex-shrink), [flex-basis](https://www.w3schools.com/css/css3_flexbox.asp#flex-basis), [flex](https://www.w3schools.com/css/css3_flexbox.asp#flex), [align-self](https://www.w3schools.com/css/css3_flexbox.asp#align-self)

The order property specifies the order of the flex items. The first flex item in the code does not have to appear as the first item in the layout. The order value must be a number, default value is 0.

The flex-grow property specifies how much a flex item will grow relative to the rest of the flex items. The value must be a number, default value is 0.

The flex-shrink property specifies how much a flex item will shrink relative to the rest of the flex items. The value must be a number, default value is 1.

The flex-basis property specifies the initial length of a flex item.

The flex property is a shorthand property for the flex-grow, flex-shrink, and flex-basis properties.

The align-self property specifies the alignment for the selected item inside the flexible container. The align-self property overrides the default alignment set by the container's align-items property vertically (center, flex-start, flex-end, stretch, baseline).

**It is an example of Flexible Box Layout.**

We can see example how to create layout using Flexible Box technology.

## Grid Layout

CSS Grid Layout Module offers a grid-based layout system, with rows and columns, making it easier to design web pages without having to use floats and positioning.

Grid layout consists of a parent element, with one or more child elements.

HTML element becomes a grid container by setting the display property to grid or inline-grid.

All direct children of the grid container automatically become grid items.

**Please have a look at this figure, explaining the main idea behind the Grid layout.**

The vertical line of grid items are called columns.

The horizontal line of grid items are called rows.

The space between each column/row are called gaps

The line between columns are called column lines.

The line between rows are called row lines.

**Properties for the Grid Container**: [display](https://css-tricks.com/snippets/css/complete-guide-grid/#prop-display), [grid-template-columns](https://css-tricks.com/snippets/css/complete-guide-grid/#prop-grid-template-columns-rows), [grid-template-rows](https://css-tricks.com/snippets/css/complete-guide-grid/#prop-grid-template-columns-rows), [grid-template-areas](https://css-tricks.com/snippets/css/complete-guide-grid/#prop-grid-template-areas), [grid-template](https://css-tricks.com/snippets/css/complete-guide-grid/#prop-grid-template), [grid-column-gap](https://css-tricks.com/snippets/css/complete-guide-grid/#prop-grid-column-row-gap), [grid-row-gap](https://css-tricks.com/snippets/css/complete-guide-grid/#prop-grid-column-row-gap), [grid-gap](https://css-tricks.com/snippets/css/complete-guide-grid/#prop-grid-gap), [justify-items](https://css-tricks.com/snippets/css/complete-guide-grid/#prop-justify-items), [align-items](https://css-tricks.com/snippets/css/complete-guide-grid/#prop-align-items), [place-items](https://css-tricks.com/snippets/css/complete-guide-grid/#prop-place-items), [justify-content](https://css-tricks.com/snippets/css/complete-guide-grid/#prop-justify-content), [align-content](https://css-tricks.com/snippets/css/complete-guide-grid/#prop-align-content), [place-content](https://css-tricks.com/snippets/css/complete-guide-grid/#prop-place-content), [grid-auto-columns](https://css-tricks.com/snippets/css/complete-guide-grid/#prop-grid-auto-columns-rows), [grid-auto-rows](https://css-tricks.com/snippets/css/complete-guide-grid/#prop-grid-auto-columns-rows), [grid-auto-flow](https://css-tricks.com/snippets/css/complete-guide-grid/#prop-grid-auto-flow), [grid](https://css-tricks.com/snippets/css/complete-guide-grid/#prop-grid).

The grid-template-columns property defines the number of columns in your grid layout, and it can define the width of each column.

The grid-template-rows property defines the height of each row.

We can adjust the gap size by using one of the following properties:

grid-column-gap, grid-row-gap, grid-gap. The grid-gap property is a shorthand property for the grid-column-gap and the grid-row-gap properties.

The justify-content property is used to align the whole grid inside the container (start, center, end, space-between, space-around, space-evenly).

The align-content property is used to vertically align the whole grid inside the container (start, center, end, space-between, space-around, space-evenly).

**Properties for the Grid Items:** [grid-column-start](https://css-tricks.com/snippets/css/complete-guide-grid/#prop-grid-column-row-start-end), [grid-column-end](https://css-tricks.com/snippets/css/complete-guide-grid/#prop-grid-column-row-start-end), [grid-row-start](https://css-tricks.com/snippets/css/complete-guide-grid/#prop-grid-column-row-start-end), [grid-row-end](https://css-tricks.com/snippets/css/complete-guide-grid/#prop-grid-column-row-start-end) ,[grid-column](https://css-tricks.com/snippets/css/complete-guide-grid/#prop-grid-column-row), [grid-row](https://css-tricks.com/snippets/css/complete-guide-grid/#prop-grid-column-row), [grid-area](https://css-tricks.com/snippets/css/complete-guide-grid/#prop-grid-area), [justify-self](https://css-tricks.com/snippets/css/complete-guide-grid/#prop-justify-self), [align-self](https://css-tricks.com/snippets/css/complete-guide-grid/#prop-align-self), [place-self](https://css-tricks.com/snippets/css/complete-guide-grid/#prop-place-self)

A grid container contains grid items.

By default, a container has one grid item for each column, in each row, but you can style the grid items so that they will span multiple columns and/or rows.

The grid-column property defines on which column(s) to place an item.

You define where the item will start, and where the item will end.

The grid-row property defines on which row to place an item.

You define where the item will start, and where the item will end.

The grid-area property can be used as a shorthand property for the grid-row-start, grid-column-start, grid-row-end and the grid-column-end properties.

The grid-area property can also be used to assign names to grid items.

Named grid items can be referred to by the grid-template-areas property of the grid container. Each row is defined by apostrophes (' '). The columns in each row is defined inside the apostrophes, separated by a space.

The first item in the HTML code does not have to appear as the first item in the grid. The Grid Layout allows us to position the items anywhere we like.

**It is an example of Grid Layout.**

We can see example how to create layout using Grid technology.

**Does CSS Grid Replace Flexbox?**

No. They can work together: a grid item can be a flexbox container. A flex item can be a grid container.

But here're some things Grid is specifically better at than Flexbox:

Making whole page layouts. It's better if you consider layout performance.

Grid-gap is wonderful, as it helps to create X columns with Y gap between them.

Requiring less media query intervention with really powerful functionality like: minmax(), repeat(), auto-fit and auto-fill.

I also prepared useful links, you can visit these links.

Thank you.