CSS : The Complete Guide

**1.Getting Started**

In this course I am going to learn about CSS from the ground up and not just in theory, but also by building an entire course project with CSS.I will start with the basics and gradually enhance it with more and more CSS features.

CSS stands for Cascading Style Sheets and the core idea behind it is that it makes your web page look good. A web page typically consists of HTML for structuring your web page, adding content and showing it on the web page.HTML is absolutely required, there are no pages without it.CSS on the other hand is optional, because it allows you to style the HTML content. It allows you to add colors, shadows, all kind of visual effects, position elements differently, so basically it turns it in an exciting and good looking website.

So we had a look at what you can do with CSS, now let's have a brief look at its history, a very brief one I promise.CSS was first introduced in 1996, we refer to this as CSS version 1, that was the first time you finally got tools to improve the look of your web page. Now only two years later, we got CSS version 2. The latest version of CSS actually is CSS version 3, that is the one this course uses and it's still in development.

CSS, towards the development of CSS, changed. Rather than focusing on different versions, they now split up CSS into a couple of modules which are organized by the feature they cover. So things like modules on coloring text, modules which focus on shadows, modules which focus on animations. We've got different versions of these modules and new modules and module versions are continuously added in the future , simply to keep up with modern developments and to keep CSS dynamic. Here's an overview of the different working groups and their results, who are working on improving CSS and developing new features.

For that I will use two tools. The first tool is a code editor, Visual Studio Code because which I think it's really nice to use and the second tool is Chrome as a browser. Again, you can use any browser of your choice but the main reason why I am using Chrome is because Chrome comes with really powerful developer tools and we will dive into these developer tools throughout the course.

**2.Diving Into the basics of CSS**

In order to learn the basics of CSS and many other features we will start with the project.

So to get started, I have the index.html file and the favicon.png file. Now the favicon is only there so that have it, it's not related to CSS, the index.html file is the important one, it's the first piece of the course project I am going to build. If you open it with Visual Studio Code , then you can see that we get a normal HTML5 skeleton inside of that file. We've got a doctype, we've got the HTML opening and closing tag, then we get a head section where we set some meta tags as well as a title and since we're building a fictional web hosting company or the website of this company, we named it uHost and you have the body tag.

Now the body tag of course holds the content of our web page and right now this really is just a main tag, a section tag and in there, this hi tag. So not too exciting but we'll add more and more throughout this course of course.

A screenshot of a cell phone

Description automatically generated

So let's add some CSS to this page. There are three different ways of adding CSS, the first way you can use is inline styling. Now let's say we want to style this section here, we can add an inline style by adding the style attribute to the section tag, the style attribute is a normal HTML attribute which you can actually add to pretty much any HTML element. Now you can simply set a CSS style by now applying a so-called CSS declaration inside of that style attribute or inside of that string you pass as an argument I could say. A declaration simply means that you define what you want to style and how you want to style it.

Now there is one other way of including styles though and that is using an external stylesheet, so let's do that now. For that I'll add a new file to the project, you can create it in your Windows Explorer of course or directly in your IDE as I do and in that new file, we will store the CSS code, hence the file should end with .css because it contains CSS code. The name is up to you, I will name it main because it refers to the main page of our web application. Now in that file, you write your CSS rules. Remember, a rule is this part which is composed of a selector, a property and a value. So you grab that rule and then you put it into that CSS file without the style tags, that's important, we're not in a HTML file, so we don't need style tags.

The third way of including CSS is the recommended way because by using an extra stylesheet, you can have a clear separation of your HTML and your CSS code which is especially useful as your CSS code grows and it would bloat your head section at some point and additionally if you use the same stylesheet in multiple pages let's say, then your browser can cache the stylesheet and doesn't need to re-download it for every new page, whereas if you include your styles in the head section, you increase the file size of your HTML file and the browser needs to re-download it since it's part of the HTML page for every new page which can be slower. That is why in this course, we will use an external stylesheet even if we only have one rule as in this case.

A screenshot of a cell phone

Description automatically generated

CSS selectors define the elements to which a set of CSS rules apply. There are more types of selectors such as:

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

You can open the developer tools by pressing F12 on Windows or you open it from the menu of your browser. Now in the developer tools, you get a couple of options in case you never worked with that. If you clicked on elements, you'll see your HTML code so to say and you can select elements by expanding and clicking there and then at the bottom, you see which styles are getting applied to these elements. You can also select that element from within your page by right-clicking on it and choosing inspect, this also works if the developer tools are closed, they will open then. If they are opened, it will just jump to that element and select it and you can also pick that tool here on the top left of the developer tools and now click on element in your DOM to also quickly inspect it.

In CSS, inheritance controls what happens when no value is specified for a property on an element. CSS properties can be categorized in two types: inherited properties, which by default are set to the computed value of the parent element and non-inherited properties, which by default are set to initial value of the property.

The other selectors that we will look at are combinators, because they combine other selectors in a way that gives them a useful relationship to each other and the location of content in the document.

Now there are more combinators, four very important ones to be precise. Now combinators allow us to be more clear about our rules and select elements by passing more information to the selector. Now you can combine multiple selectors, not just two as a side note and as I mentioned, you can combine them with four important types of combinators, which are :

A screenshot of a social media post

Description automatically generated

A screenshot of a social media post

Description automatically generated

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

**3.Diving Deeper into CSS**

We will dive into the box model, a core part of CSS, really important to know , so that is what we'll start with.

A screenshot of a cell phone

Description automatically generated

In HTML we have two different element types, block level elements and inline elements. Block level/ inline elements describe the way the elements behave in the so-called document flow (so the elements’ positioning on our website). Each block level element always uses the entire space available in the line where it is positioned and each block level element also starts in a new line.

On the other hand, an inline element doesn’t behave like that. These elements just use the space they need to display the actual content and after this content is displayed, the following inline element is placed next to it (in case the remaining space on that line is sufficient).

We will use those elements in our project, which are as in the next picture:

A screenshot of a cell phone

Description automatically generated

A CSS pseudo-class is a keyword added to a selector that specifies a special state of the selected elements. They let you apply a style to an element not only in relation to the content of the document tree, but also in relation to external factors like the history of the navigator (:visited, for example), the status of its content (like :checked on certain form elements), or the position of the mouse (like :hover, which lets you know if the mouse is over an element or not).

Display: none vs visibility: hidden

display: none; - this value removes the element to which you apply it from the document flow. This means that the element is not visible and it also doesn't "block its position". Other elements can (and will) take its place instead.

visibility: hidden; - this value only hides an element but other elements don’t fill the empty spot.

**5.More on Selectors and CSS Features**

We used CSS class selectors, they're very useful, we can define reuseable rules with them and apply them to any element which should receive the rule.

A screenshot of a cell phone

Description automatically generated

In the HTML snippet , we see that we got two elements and the first one actually got more than one class applied to it, it got class 1 and class 2 and the two classes are simply separated by a whitespace between them and the second element is a normal link which has just one class but we'll select it in a special way.

So for the first element, important, you can use multiple classes on one element, so the two rules we have here on the left would both be applied independently to that element because that element has the same class. if you got two classes applied to the same element and they both happened to set a certain style on the element, then the normal specificity and order rules will apply, the order in this class list here is not important regarding this.

Now for the second element, the anchor tag, we actually use a combinator ( combined selector here somehow ).

In order to put that in practice we have :

<body>

    <nav>

        <a href="#intro" class="active">Intro</a>

        <a href="#outro">Outro</a>

    </nav>

    <section id="intro" class="main-section highlighted">

        <p>This is the intro section.</p>

    </section>

    <section id="outro" class="main-section">

        <p>This is the outro section.</p>

    </section>

</body>

If both classes of the intro section happen to define a style for the same property, the normal specificity and order rules apply and therefore the class which is defined later in the file would win.

.main-section {

    height: 800px;

    border: 1px solid red;

    padding: 16px;

}

.highlighted {

    border: 2px solid blue;

}

If we have those two in this particulary order then our border will have 2px and will be red because it takes the values from the last one.We can see that in the next screenshot:

A screenshot of a computer

Description automatically generated

If we change their order (first written being .highlight and then .main-section) the border will have 1px and the color red.

A close up of a logo

Description automatically generated

The anchor tag has #intro and class=”active”. For the active class we want to apply whatever styles we're defining and they will be seen when the intro class is active. In order to demonstrate that I used the next CSS code:

a {

    text-decoration: none;

    color: white;

    margin: 10px;

}

a.active {

    color: purple;

}

Which can be seen in this screenshot:

A picture containing bird

Description automatically generated

CSS Class Selector vs ID Selector

A screenshot of a cell phone

Description automatically generated

There is one special pseudo class and that's the “not” pseudo class. This is an interesting pseudo class because it allows us to basically reverse a certain rule or exclude a certain selector.

:not(p) {

    color: blue;

}

In this example, we would apply a certain style to anything that's not a paragraph, so it will exclude the selector you pass between the parentheses.

In order to see that better, in our previous exemple we will add the not pseudo class in the CSS code in order to select all the a’s that don’t have an active class and turn them the color yellow:

a:not(.active) {

    color: yellow;

}

In this case Outro became yellow but Intro remained the color that we put for the active class:

A screenshot of a cell phone

Description automatically generated

**5.Practicing the basics**

For our project we added some extra code with three plans for the website.

<div>

                <article class="plan">

                    <h1>FREE</h1>

                    <h2>$0/month</h2>

                    <h3>For hobby projects or small teams.</h3>

                    <ul>

                        <li>1 Workspace</li>

                        <li>Unlimited Traffic</li>

                        <li>10GB Storage</li>

                        <li>Basic Support</li>

                    </ul>

                    <div>

                        <button>CHOOSE PLAN</button>

                    </div>

                </article>

                <article class="plan">

                    <h1>RECOMMENDED</h1>

                    <h1>PLUS</h1>

                    <h2>$29/month</h2>

                    <h3>For ambitious projects.</h3>

                    <ul>

                        <li>5 Workspaces</li>

                        <li>Unlimited Traffic</li>

                        <li>100GB Storage</li>

                        <li>Plus Support</li>

                    </ul>

                    <div>

                        <button>CHOOSE PLAN</button>

                    </div>

                </article>

                <article class="plan">

                    <h1>PREMIUM</h1>

                    <h2>$99/month</h2>

                    <h3>Your enterprise solution.</h3>

                    <ul>

                        <li>10 Workspaces</li>

                        <li>Unlimited Traffic</li>

                        <li>Unlimited Storage</li>

                        <li>Priority Support</li>

                    </ul>

                    <div>

                        <button>CHOOSE PLAN</button>

                    </div>

                </article>

            </div>

We added to each article a class ( plan ) so it can help us style them.We did that using the next CSS code:

.plan {

    background: #d5ffdc;

    text-align: center;

    padding: 16px;

    margin: 8px;

    display: inline-block;

    width: 30%;

    vertical-align: middle;

}

Also, we will put a text-align: center; to the section-title class and all the new things that we did can be seen on the next screenshot:

A screenshot of a cell phone

Description automatically generated

We made the middle plan bigger and added recommended in order to make people choose it.

Now we will work with the middle plan more in order to make it way more visible. For that we will add a second class to the second article named plan—highlighted and also a new class to the h1 with Recommended named plan\_\_annotation.

Then, we will style the middle plan by using CSS code to the new classes that we just defined:

.plan--highlighted {

    background: #19b84c;

    color: white;

    box-shadow: 2px 2px 2px 2px rgba(0, 0, 0, 0.5);

}

We used the box-shadow in order to give our box some shadow on the bottom and the ight part of it. We also used rbga so we can give it a black-transparent color for a better effect.

.plan\_\_annotation {

    background: white;

    color: #19b84c;

    padding: 8px;

    box-shadow: 2px 2px 2px 2px rgba(0, 0, 0, 0.5);

    border-radius: 8px;

}

For the Recommended we changed it a little to make it even more remarkable.All of the adjustments that we did can be seen in the following screenshot:

A screenshot of a cell phone

Description automatically generated

Then we changed some more things in the plan by adding more classes and styling them with CSS code. One big thing that we styled was for the buttons, because when we would press one of them we could see an outline which I wanted to get rid of.

For that ( I added a class for each button named button ) I added a CSS code :

.button:focus {

    outline: none;

}

So, in order to explain it in practice I put a picture for the button before applying the outline:none; and after it:

A screenshot of a cell phone

Description automatically generated A picture containing player, ball

Description automatically generated

Then, we added a new section to our code where we used things that we have already seen before. The section is looking like this:

A picture containing device, drawing

Description automatically generated

For our website we also wanted a footer. To achive that I wrote some HTML code :

<footer class="main-footer">

        <nav>

            <ul class="main-footer\_\_links">

                <li class="main-footer\_\_link">

                    <a href="#">Support</a>

                </li>

                <li class="main-footer\_\_link">

                    <a href="#">Terms of Use</a>

                </li>

            </ul>

        </nav>

    </footer>

And we styled it by writing the next CSS code :

.main-footer {

    background: black;

    padding: 32px;

    margin-top: 48px;

}

.main-footer\_\_links {

    list-style: none;

    margin: 0;

    padding: 0;

    text-align: center;

}

.main-footer\_\_link {

    display: inline-block;

    margin: 0 16px;

}

.main-footer\_\_link a {

    color: white;

    text-decoration: none;

}

.main-footer\_\_link a:hover,

.main-footer\_\_link a:active {

    color: #ccc;

}

Now, the footer looks like this :



We will make a shared.html file where we cut and paste the main, button, header and footer rules from our main.css. Then, we put the shared file in the head with :

<link rel="stylesheet" href="shared.css">

Everything will look the same as it was before, but we just put those things in the shared file just so we can use them easier to the other pages that we are going to create. To put that in practice, I created a new subfolder named packages where I added an index.html file where I copied all the HTML code from our first page but without the main part ( so just the header and the footer remain the same for all the pages ) . In this case, when we will press the packages web page on our main web page we will get this:

A picture containing screenshot, game, monitor, holding

Description automatically generated

Then, we put some packages inside it and some styles that will look like this:

A screenshot of a social media post

Description automatically generated

After that we style them a little so we will have blocks for each plan.

The float CSS property places an element on the left or right side of its container, allowing text and inline elements to wrap around it. The element is removed from the normal flow of the page, though still remaining a part of the flow (in contrast to absolute positioning).

A screenshot of a cell phone

Description automatically generated

**6.Positioning Elements with CSS**

A screenshot of a cell phone

Description automatically generated A screenshot of a cell phone

Description automatically generated

In order to understand better the positioning we have inside this body element , three divs for example, three block level elements in our case and as we know, the general behavior of these block level elements is the way it is displayed right here, occupying the entire space available in the row where they are positioned and because of that, being displayed one after another.

This basically is due to the fact that these are block level elements and that these elements are following the document flow which is basically the flow of a normal HTML document.

There is already some property applied automatically which makes sure that this document flow is followed or will be followed by the elements.The position property by default is static.

We can move them in which way we want but not with this default value, for that you have to specify a position property with a value of absolute, relative, fixed or sticky.

Now sticky is great out right here because sticky is a rather new value for this property and therefore the browser support is not the best, the general takeaway of this slide is that we need the position property in combination with a value different, that's really important, different from the default value which is static to be able to change the actual position of our elements on the website.

That's not everything of course because with that, with these values, we just specify that we want to change the position but we of course also have to specify how we want to change the position, so where are these elements should be displayed.

In order to make our navigation bar to stay fixed even though we are scrolling down on our web page, we added to our main-header the following CSS declarations:

position: fixed;

top: 0;

left: 0;

After that we want to set a background image on the packages website which will be a fixed image. For that we added a div in our HTML code before header:

<div class="background"></div>

And then we styled it in CSS( we also downloaded an image for it ):

.background {

    background: url("../images/plans-background.jpg");

    width: 100%;

    height: 100%;

    position: fixed;

    z-index: -1;

}

Z-index is used in order to put the image in the background of the packages. We gave him the value -1 for that and now our packages website looks like this :

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

**7. Understanding Background Images and Images**

A screenshot of a social media post

Description automatically generated

For our project we tried the next background properties :

background: url("freedom.jpg");

background-size: cover;

background-repeat: no-repeat;

background-position: left 10% bottom 30%;

background-origin: border-box;

background-clip: border-box;

Which can be written easier as in the following line :

background: url("freedom.jpg") left 10% bottom 20%/cover no-repeat border-box;

Those modifications can be seen in the following screenshot :

A person with a sunset in the background

Description automatically generated

Next for our project we wanted to change our logo by adding an image to the main-header\_\_brand in the HTML code.

<img src="images/uhost-icon.png" alt="uHost - Your favorite hosting company">

Then, we want to style it by writing some CSS code. If we add a height to our main-header\_\_brand of 22 px, in order to make our icon be the same size as the elements in it, we will see that that nothing changes. The reason is that by setting a height on a container the image will not be affected because of its default behavior.

The default behavior is if you enter an image tag and you point to an image, then the default height and width of the original image will be used and will be entered into your document, no matter which width and height you set for the surrounding element. So in our case here if we have a look at our image, you'll see it it has dimensions of 186 x 190.

In order to change that, we will display the elements in the main header brand as inline-block and set a hight of 22px.

Then we select the image and set a height of 100% in order to make the logo be the same size of the other elements with the next CSS code:

.main-header\_\_brand img {

    height: 100%; \*/

}



Let's add a customers page and for that, let's first of all create a new folder in our project, I'll name it customers and in there, I'll create an index.html file and a customers.css file. Then I will copy the code for both of the files and also we have 2 photos of customers which I will put in the images folder.

Then, we want to style the customer page by adding some CSS code :

.testimonial\_\_image-container {

    width: 65%;

    display: inline-block;

    vertical-align: middle;

    box-shadow: 3px 3px 5px 3px rgba(0, 0, 0, 0.4);

}

.testimonial\_\_image {

    width: 100%;

    vertical-align: top;

}

.testimonial\_\_info {

    text-align: right;

    padding: 14px;

    display: inline-block;

    vertical-align: middle;

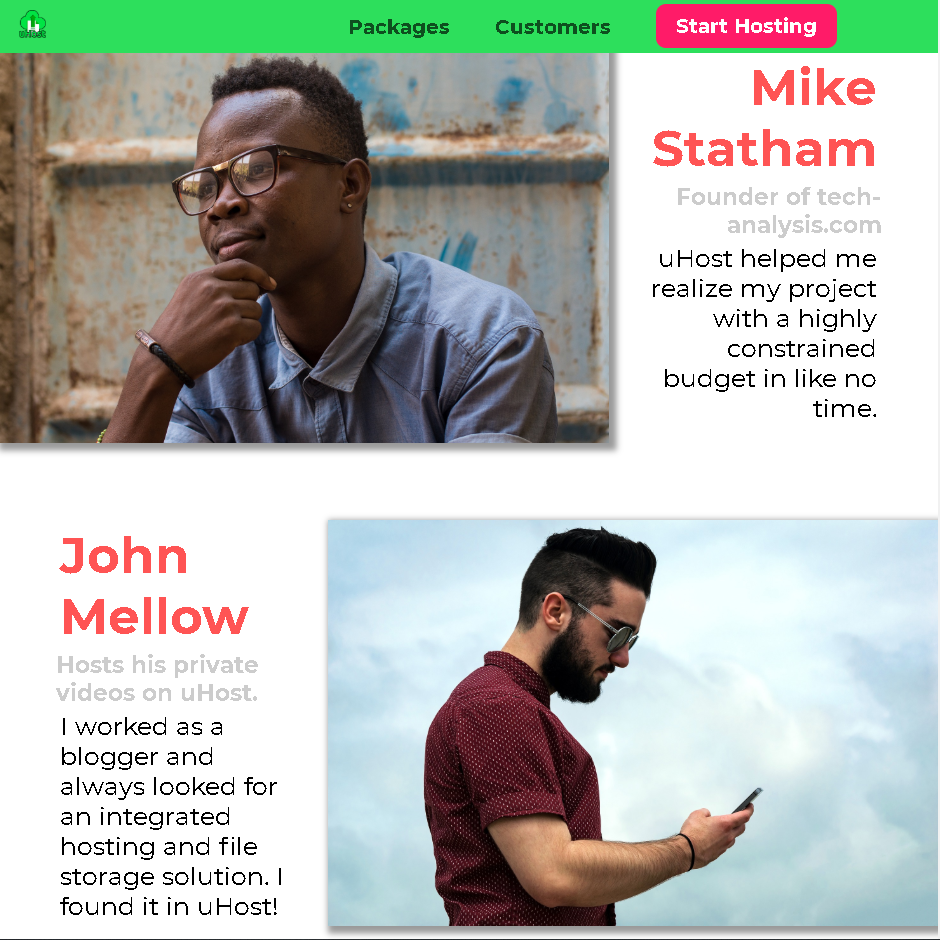
    width: 30%;

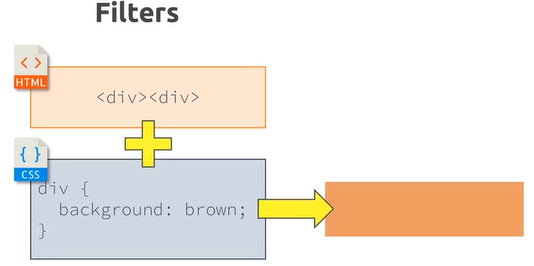
}

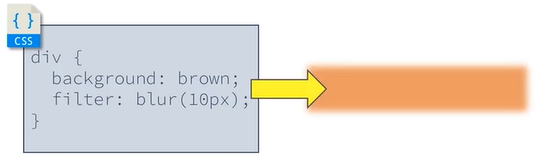
We used the display: inline-block in order to put the photo and info of each customer on the same line and we added a width of 30% for the info and 65% for the image. Then, we used the vertical-align: middle to the image container and the info and the text actually is positioned next to our testimonial image. We also wanted to add a little shadow with the box-shadow: 3px 3px 5px rbga(0, 0, 0, 0.4).

Now we also got that whitespace at the bottom of the image, which is simply an inline element bug and we can get rid of it by going to our testimonial image and setting vertical align on the image, not on the container but on the image to top or bottom.

The final result of the customers web page can be seen in the following screenshot:



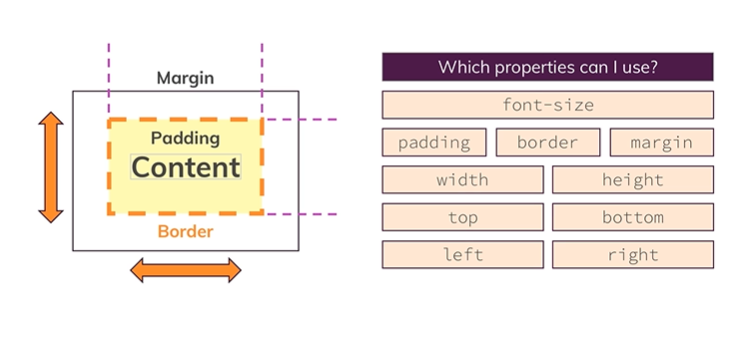




To our project, we also added 3 SVGs to the section key-features as pictures for each key-feature and we also added in our CSS code to the key feature image a padding of 20px, so our SVGs will fit better in the circles:



**8. Sizing and Units**



We have three rules to remember for the percentage unit work.



Now let's dive into it right now and the starting point of course is an element. So this is the element we applied a percentage unit to, for example to the width property. Now the important thing is that this is a special element because it has this declaration also applied, position fixed, the position property definitely has an impact on the way the percentage unit behaves.

The reference point for such an element with a percentage unit is called the containing block, this containing block is simply for example an element, a parent element for example which has a width of let's say 100 pixels applied. If this parent is the containing block ,well then our child, so our element right here, would have a width of let's say 10 pixels if we applied a width of 10% to it, so 10% width of a child would be 10 pixels if the containing block has a width of 100 pixels.

Now the issue is that the containing block depends on the position property applied to our element right here and in this case the position property or the value for this property that we applied is fixed, then the containing block is not an element, it's the viewport.



The second rule also starts with an element, this is again a special element because this time, we again have a percentage unit applied but we don't have position fixed but position absolute as a declaration right here.

Now with the different position value, the containing block also changed. Previously, we had the viewport, now we have an ancestor. The percentage unit of our element refers to this ancestor's content plus the padding, that's really important.

So, if we again have a percentage unit for the width, for our element, we will take the width of the ancestor's content and padding as the containing block. The containing block for an element with the position absolute declaration applied is the closest ancestor which is not position static.



For the third rule, we have an element with position either static or relative. In this case, the containing block again is an ancestor, like we had it before but this time, it's only the content of the ancestor, so not content and padding like we had it before. The ancestor is the closest one which is a block level element.

Em units for the font-size property will be relative to the font-size of the parent element. Em units on other properties than font-size will be relative to the font-size of the current element. Rem units sizes will always be relative to the font-size of the root html element.

Four new “viewport-relative” units appeared in the CSS specifications – vw, vh, vmin, and vmax – work similarly to existing length units like px or em, but represent a percentage of the current browser viewport.

Viewport Width ( vw ) – A percentage of the full viewport width. 10vw will resolve to 10% of the current viewport width. The difference between % and vw is most similar to the difference between em and rem. A % length is relative to local context (containing element) width, while a vw length is relative to the full width of the browser window.

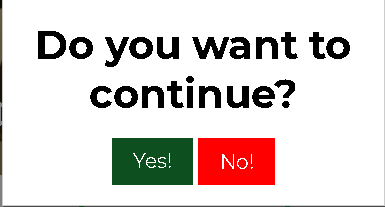
Viewport Height ( vh ) – A percentage of the full viewport height. 10vh will resolve to 10% of the current viewport height.

Viewport Minimum ( vmin ) – A percentage of the viewport width or height, whichever is smaller. 10vmin will resolve to 10% of the current viewport width in portrait orientations, and 10% of the viewport height on landscape orientations.

Viewport Maximum ( vmax ) – A percentage of the viewport width or height, whichever is larger. 10vmax will resolve to 10% of the current viewport height in portrait orientations, and 10% of the viewport width on landscape orientations.

**9. Working with JavaScript and CSS**

Next, on our project we will add some HTML and CSS code for a modal which will look like this:



Then, we want to add a JavaScript file to our project by creating a file named “shared.js” and then add right before closing the body section the next HTML code:

<script src="shared.js"></script>

In the JavaScript file we added some code in order to make events when we click one of the three buttons for ‘Choose A Plan’ which will take us to the modal and also would activate the backdrop:

var backdrop = document.querySelector('.backdrop');

var selectPlanButtons = document.querySelectorAll('.plan button');

var modal = document.querySelector('.modal');

for(var i = 0; i < selectPlanButtons.length ; i++){

    selectPlanButtons[i].addEventListener('click', function() {

        modal.style.display = 'block';

        backdrop.style.display = 'block';

    });

}

For the modal we also added 2 events: one for when you click the ‘No’ button and it will just get us back to our normal page and one for when you click on the backdrop doing the same. For that we wrote the next JavaScript code:

backdrop.addEventListener('click' , closeModal);

No.addEventListener('click' , closeModal);

function closeModal(){

    backdrop.style.display = 'none';

    modal.style.display = 'none';

}

We added also some HTML and CSS code for the mobile navigator which we wanna improve a little later and we added a side navigation bar which is activated by pressing a button that has 3 vertical lines. Then, in our JavaScript file we added some code so whenever you press the button it will open the side navigation bar and then close it either if you press on the same button or on the backdrop.

var toggleButton = document.querySelector('.toggle-button');

var mobileNav = document.querySelector('.mobile-nav');

toggleButton.addEventListener('click', function(){

    mobileNav.style.display = 'block';

    backdrop.style.display = 'block';

})

backdrop.addEventListener('click' , function(){

    mobileNav.style.display = 'none';

    closeModal();

});





Then, we added all of those to the Packages and Customers page and edited some JavaScript code so the side navigation bar works properly on every page.

**10. Making our Website Responsive**

In this module we will make our project more responsive for each type of device .We added the meta :

<meta name="viewport" content="width=device-width, initial-scale=1.0">

And now we can work on the responsive part.

Media queries are useful when you want to modify your site or app depending on a device's general type (such as print vs. screen) or specific characteristics and parameters (such as screen resolution or browser viewport width).

A media query is composed of an optional media type and any number of media feature expressions. Multiple queries can be combined in various ways by using logical operators. Media queries are case-insensitive.

A media query computes to true when the media type (if specified) matches the device on which a document is being displayed and all media feature expressions compute as true. Queries involving unknown media types are always false.

We used two media queries in our CSS code :

@media (min-width: 40rem) {

  #product-overview {

    height: 40vh;

    background-position: 50% 25%;

  }

  #product-overview h1 {

    font-size: 3rem;

  }

}

@media (min-width: 60rem) {

  #product-overview {

    height: 50vh;

    background-position: 50% 25%;

  }

  #product-overview h1 {

    font-size: 5rem;

  }

}

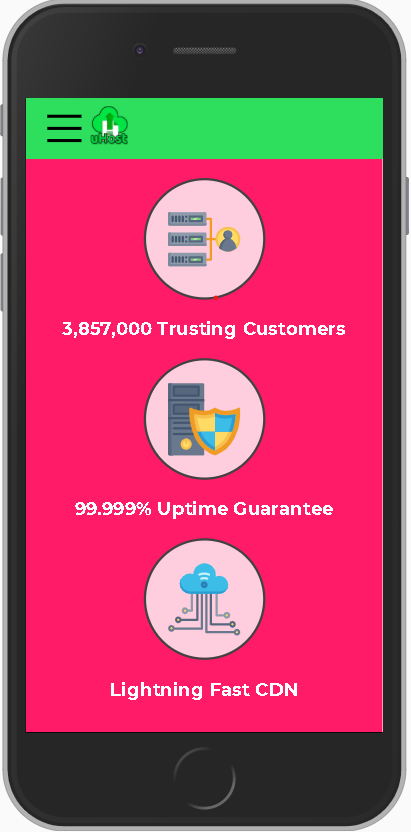
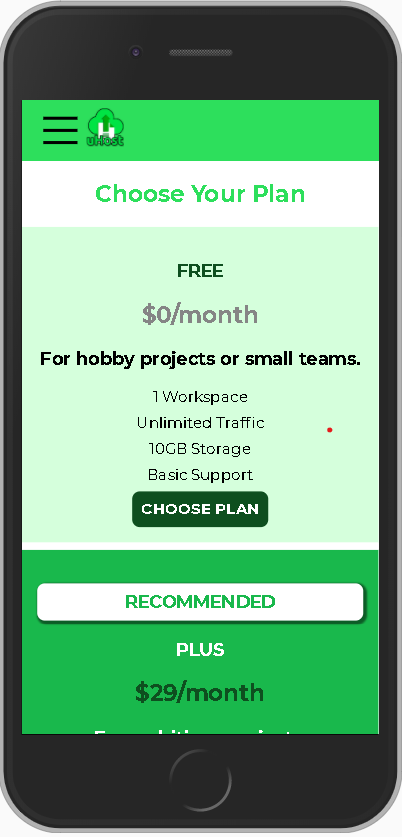
They will only work when the minimum width is 40 rem and 60rem and will just apply the things that we wrote inside of them. That will help our website look better on more devices.

Usually, the CSS code that doesn’t include media query should be for mobile version because nowadays most people open websites on their phone instead of a laptop/pc. This means that we will change some things from our code to make it look better for the mobile version and then add media queries for the pc version.

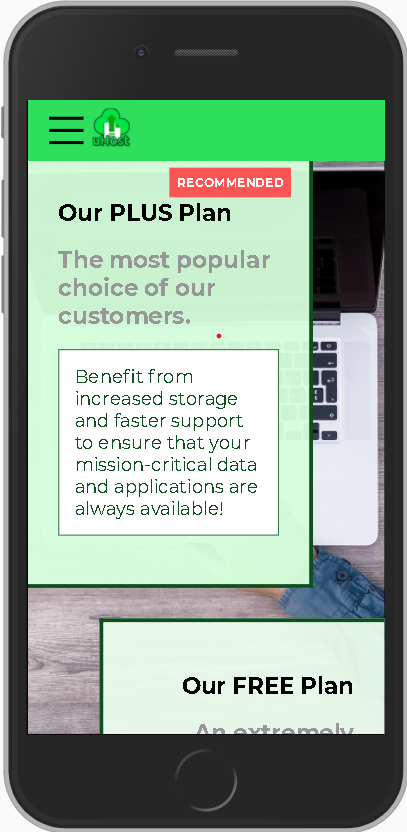
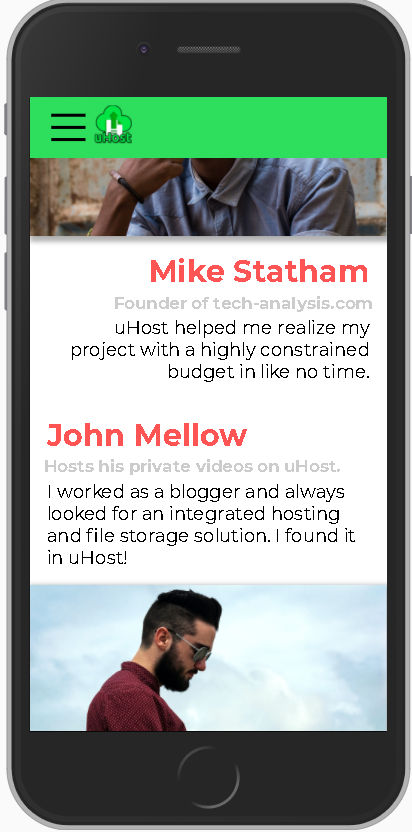
Now, for the phone version we took out the display: inline-block; so they will display one below another, changed the width to 100% and took out the box shadow for the highlighted plan.

Then, we created a media query with min-width:40 where we put back all the elements that we got out in order to make it look as it was on the desktop version and also added a few extra things like a minimum width and a maximum width for the plans in order to look better.

We did the same for the key features and the navigation bar so it will look way better for different types of devices



Now, we are going to focus on the other two web pages that we have in order to make them responsive for each type of device by adding more media queries to each of them.



**11.Adding and Styling forms**

It is time to add the start-hosting webpage to our website. For that we have a new folder called start-hosting , where we have a HTML and a CSS file. In the HTML file we will add all the elements that are shared between the webpages and also a form type and a title which we will style in this module.



Firstly, we will select the label, input and select in the form by the next CSS code and style them a little:

.signup-form label,

.signup-form input,

.signup-form select {

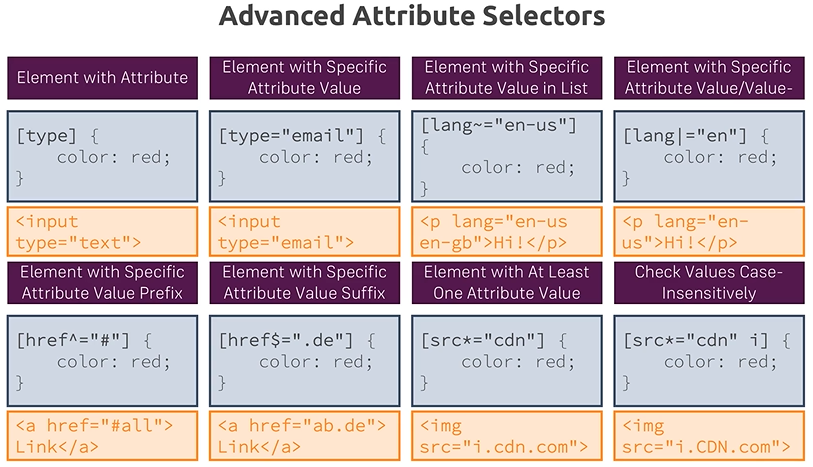
    display: block;

    margin-top: 1rem;

    width: 100%;

}





After that, we want to make the form look better on every type of device. For that we added some CSS code:

.signup-form {

    padding: 1rem;

}

.signup-form label {

    font-weight: bold;

}

.signup-form input[id\*="terms"],

.signup-form input[id\*="terms"] + label  {

    display: inline-block;

    width: auto;

}

.signup-form button[type="submit"] {

    display: block;

    margin-top: 1rem;

}

@media (min-width: 40rem) {

    .signup-form {

        margin: auto;

        width: 25rem;

    }

}





Then, we worked with the elements in the form in order to make it look better and also to have some requirments. For that we added this CSS code:

.signup-form input.invalid,

.signup-form select.invalid,

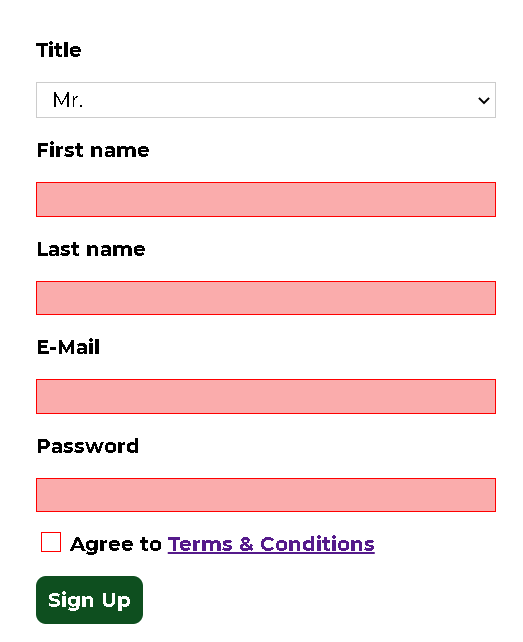
.signup-form :invalid {

    border-color: red !important;

    background: #faacac;

}

Which will make our labels have the border color and background different if they are not filled or if they don’t follow the requirments ( exemple: the e-mail address if it is not a valid one ):



And then we also saw that we can change the button “Sign Up” to not work if the form is not filled as it should. For that, we can make it work manually by adding disabled to the button ( which will make it at first invalid ) , and then add some CSS code in order to make it clear to the user that the button can not be pressed:

.button[disabled] {

  cursor: not-allowed;

  border: #a1a1a1;

  background: #ccc;

  color: #a1a1a1;

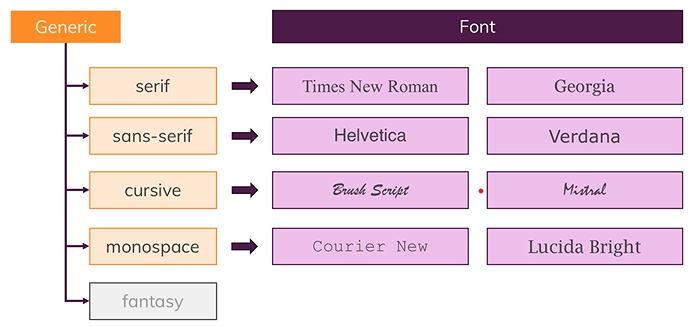
}

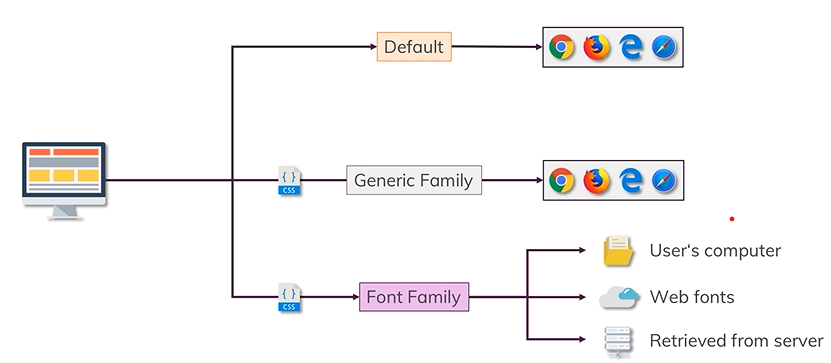


Also if you put the mouse cursor over it, it will show the not-allowed cursor.

This technique is not the best one to use and all of this can be implemented with JavaScript code.

**12. Working with Text and Fonts**





Now the default behavior is basically defined by the browser which means that we didn't specify any kind of font family in our CSS code because with that, the browser will simply choose the standard font that we applied and use the font family that is selected in the browser.

One alternative we have is this one, we can define a generic family in our CSS code. With that we could choose different font families for the different generic families in the browser settings ( exemple : sans-serif ).

The third option will be that we specify a font family, in contrast to that, this will not be influenced directly by the browser settings because in the browser settings, we have different font families available but we could also select a font family in our CSS code which is not available by default in the browsers of our users. Because of that, we have three different options again, where to retrieve these font families from.

The first option is the font family we specify in the CSS code is saved locally on the user's computer. Now as you can already imagine, this can lead to a lot of problems because the user's computer is a local computer and therefore, we cannot control which font families are installed, which ones are not, so the chances are quite high that our website, at least the fonts in our website will look different for a lot of users.

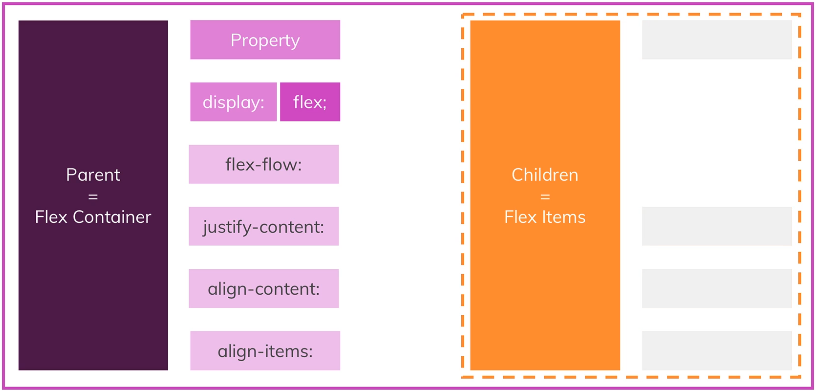
The second option, we can use web fonts and this is what we actually do in our project, we import our fonts that we used on the website from Google fonts with a so-called web font, so basically, we retrieve the font from a third party.

And then we have a third option, the third option simply means that we retrieve the fonts that we want to apply, the font family from a server and this could be for example our own server, so the server where our website is hosted. Therefore, we again have the control of the font families and make sure that the user can display these families, nevertheless if these are available in the browser or if these are saved to the computer and with that, we can ensure that the website looks as intended for most of our users.

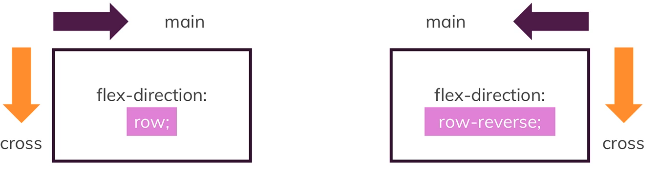
For the Google Fonts, there are more ways to add them to our code, but the best one would be to go on fonts.google.com, choose a font that we want to have to our website, click on it and choose which style you would like to have ( because each font can have different styles ). Then, go to embed, click on @import and there you will have an url with a @import in front of it which we will copy and paste it on the CSS file that we want to use ( in our case we will put them on shared.css file in order to have them on all of our webpages ) and then whenever we want to use one of the fonts that we chose we will just write a syntax like :

font-family: "Font-You-Imported", \*Generic Family\*;

**13.Adding Flexbox to our Project**

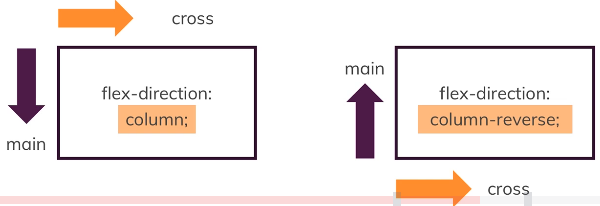


Flexbox allows us to change the way our elements are displayed and to use flexbox, we simply have to add a specific property to our element. This property is the display property. If we apply display flex to an element, this element will be turned into a so-called flex container. Inside our flex container, we should have other nested elements the so-called children and these children are also named flex items.



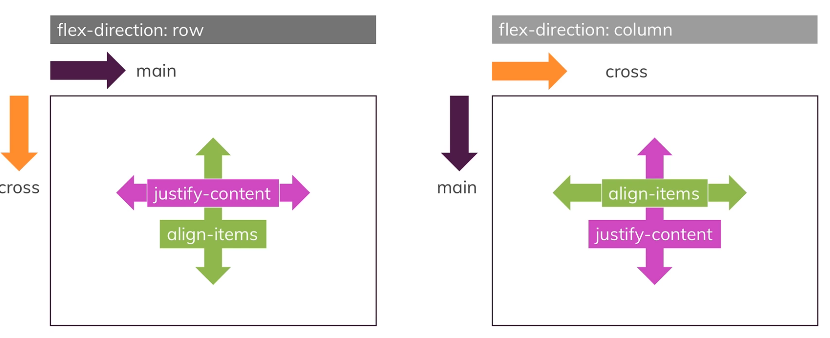
We had our flex container and we applied the flex direction property with a value of row. Now, our starting point for the main axis is the left top corner of our element basically or of our website. if the main axis goes from the top left to the top right, well then the cross axis which, that's important, always has the same starting point as the main axis goes from the top left to the bottom left corner. They are displayed in a row, so along the main axis and from left to right.

In addition to row, we also applied another value, we applied flex direction row reverse and if we think about the way row behaves, so from the top left to the top right for the main axis, then row reverse simply behaves the other way round. It starts in the right top corner and goes to the left top corner and the cross axis starts also in the right top corner and goes to the right bottom corner.

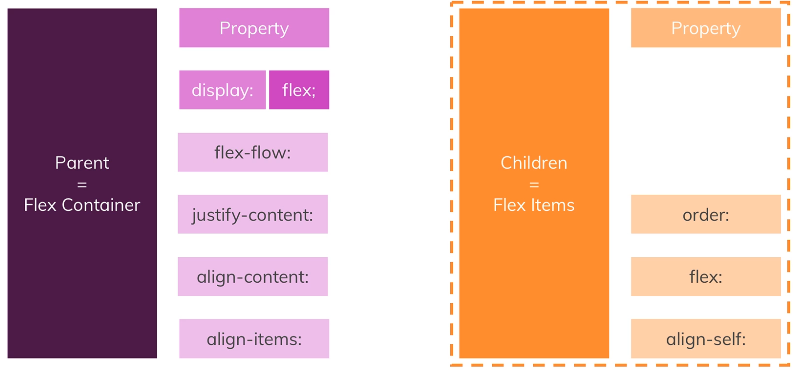


if we again take our flex container and now apply flex direction column, the starting point is exactly the same that we had before for flex direction row, it's still the top left corner but now the main axis goes from top to bottom and the cross axis goes from left to right.

If the flex direction is column, then the starting point is again the top left corner but the main axis now goes as the direction indicates, along the column and if we have column reverse, then the starting point is the bottom left corner and the main axis is again the column.

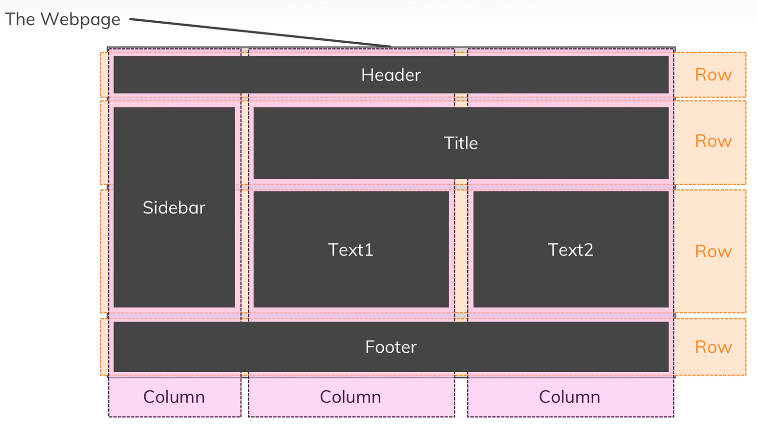


To center our box we use the align-items property to align our item on the cross axis, which in this case is the block axis running vertically. We use justify-content to align the item on the main axis, which in this case the inline axis running horizontally. For the flex-direction: column they just switch.



For the next steps, we will just change the items that we had the display: inline-block to display: flex so the code will be shorter and way better.

**14.Using CSS Grid**



Let's consider a normal web page like this one, where we have a couple of different elements or areas. We may have a header, sidebar, the main content with some elements and a footer let's say. Now if we look close, we could divide this up into a grid of rows and columns. We could think of this page being structured along this grid and actually, this is how you work in many design tools where you can also use a grid to position elements or when you work with CSS frameworks like Bootstrap, they also always ship with a grid which makes the positioning of elements easier.

In order to understand this concept easier we will have a simple code of a container which has 4 elements. We set the display: grid and also how the rows and columns will be grided.

.container {

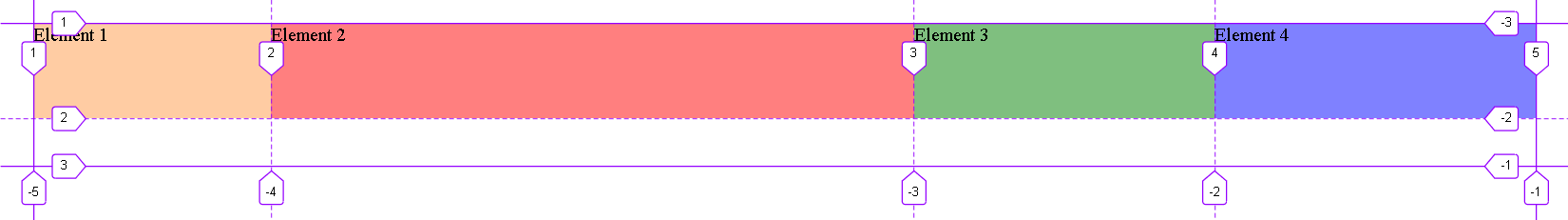
    margin: 20px;

    display: grid;

    grid-template-columns: 200px 2fr 20% 1fr;

    grid-template-rows: 5rem 2.5rem;

}



In order to see the elements better and to have the line number displayed, I used Firefox.

Then, we will try to see how the commands work by modifying how the third element will be:

.el3 {

    background: rgba(0, 128, 0, 0.5);

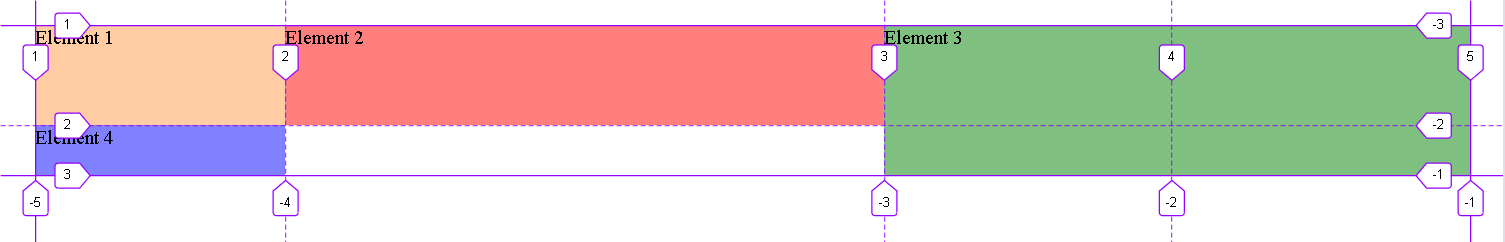
    grid-column-start: 3;

    grid-column-end: 5;

    grid-row-start: 1;

    grid-row-end: 3;

}



We can see that we made the Element 3 start from the column number 3 and end to the column number 5 and also the width to be from row number one to row number 3.

Then, we have the grid-gap which will put a gap between rows or/and columns. We can also define some areas and use them in order to be easier to put each element where it should be:

grid-gap: 10px;

grid-template-areas: "header header header header"

                     "side side main main"

                     "footer footer footer footer";

Then in order to put one of the elements in the area that we want we will just use grid-area: [the area that we want it to be]. As an example we have :

.el1 {

    background: rgba(255, 154, 72, 0.5);

    grid-area: side;

}

.el2 {

    background: rgba(255, 0, 0, 0.5);

    grid-area: main;

}

.el3 {

    background: rgba(0, 128, 0, 0.5);

    grid-area: header;

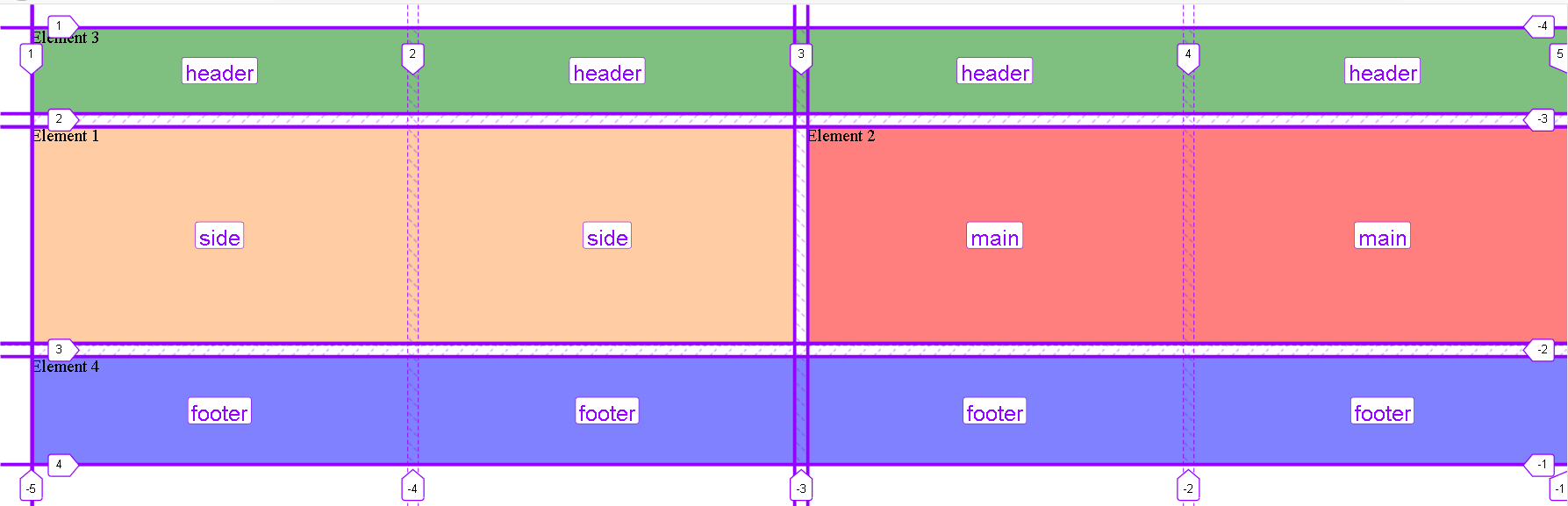
}

.el4 {

    background: rgba(0, 4, 255, 0.5);

    grid-area: footer;

}



Now, we will implement the grid in our project because we can see that on each page we have the header, main section and the footer. Since the header is fixed we can still use the grid for the main section and footer.

For that we will give the body the next code:

display: grid;

  grid-template-rows: 3.5rem auto 8rem;

  grid-template-areas: "header"

                      "main"

                      "footer";

height: 100%;

And for main and footer we will add the next code:

grid-area: main; - for the main part

grid-area: footer; - for the footer

This way the webpage looks exactly the same it did before but we just implemented the grid into it for the code to be easier to understand.

In case we have items that can be added to our grid we can use the auto commands.

Grid-auto-rows/columns - specifies the size of any auto-generated grid tracks (aka implicit grid tracks). Implicit tracks get created when there are more grid items than cells in the grid or when a grid item is placed outside of the explicit grid.

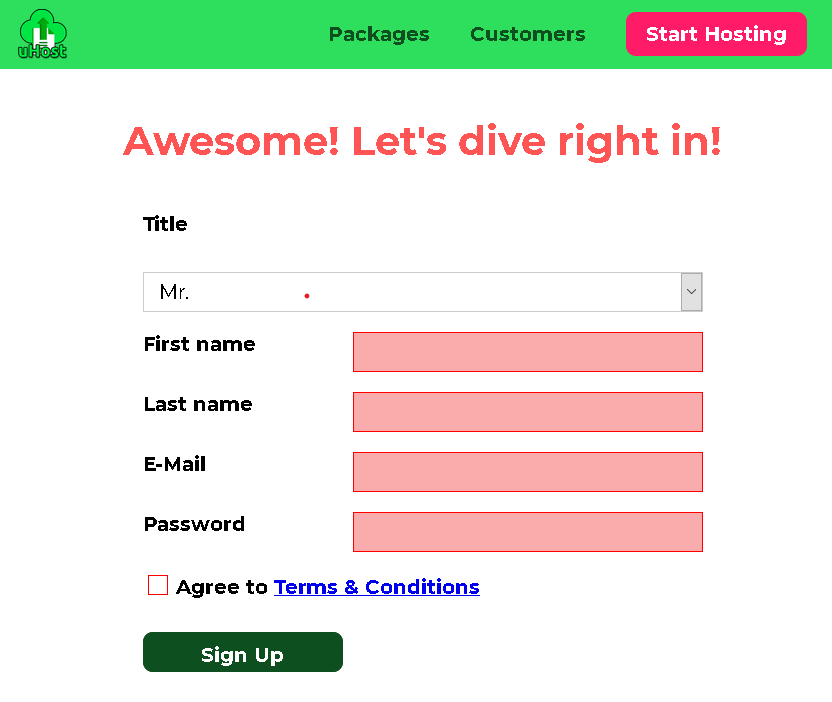
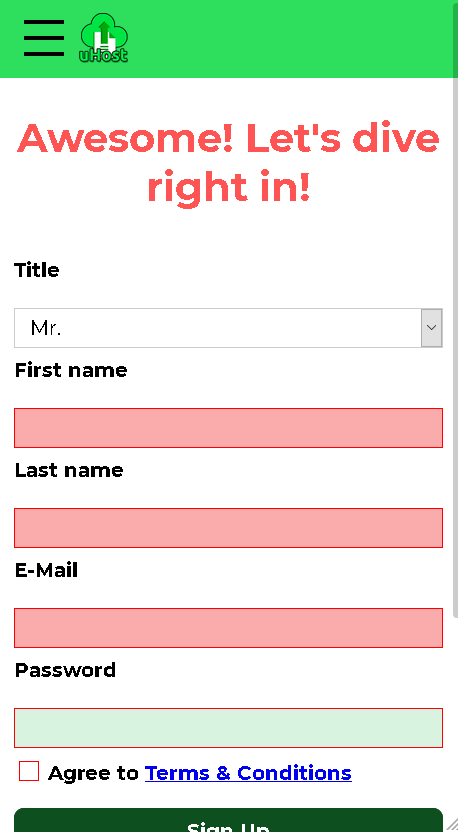
Grid-auto-flow - If you have grid items that you don’t explicitly place on the grid, the auto-placement algorithm kicks in to automatically place the items. This property controls how the auto-placement algorithm works. The values for it are : row – tells the auto-placement algorithm to fill in each row in turn, adding new rows as necessary ; column – tells the auto-placement algorithm to fill in each column in turn, adding new columns as necessary ; dense – tells the auto-placement algorithm to attempt to fill in holes earlier in the grid if smaller items come up later.

Auto-fill - fills the row with as many columns as it can fit. So it creates implicit columns whenever a new column can fit, because it’s trying to fill the row with as many columns as it can. The newly added columns can and may be empty, but they will still occupy a designated space in the row.

Auto-fit - fits the currently available columns into the space by expanding them so that they take up any available space. The browser does that after filling that extra space with extra columns (as with auto-fill ) and then collapsing the empty ones.

On our project, we will use grid for the Start Hosting page in order to organize the form.

The only changes that we had is that for the desktop version we created 2 columns so the form will have on the same row the label and the place to fill it. For both the phone and desktop version we added some gap for the grid so it looks a little better.

**15.Transforming Elements with CSS Transforms**

The transform property allows you to visually manipulate an element by skewing, rotating, translating, or scaling. The values used for the transform are:

- scale(): Affects the size of the element. This also applies to the font-size, padding, height, and width of an element, too. It’s also a shorthand function for the scaleX and scaleY functions.

- skewX() and skewY(): Tilts an element to the left or right, like turning a rectangle into a parallelogram. skew() is a shorthand that combines skewX() and skewY by accepting both values.

- translate(): Moves an element sideways or up and down.

- rotate(): Rotates the element clockwise from its current position.

- perspective(): Doesn’t affect the element itself, but affects the transforms of descendent elements’ 3D transforms, allowing them all to have a consistent depth perspective.

In our project we will transform our “Recommended” badge from the Packages so it looks better. For that we will add some css code to the package\_\_badge and also we will add overflow : hidden in the package :

.package\_\_badge {

transform: rotateZ(45deg) translateX(3.5rem) translateY(-2rem);

    width: 12rem;

    text-align: center;

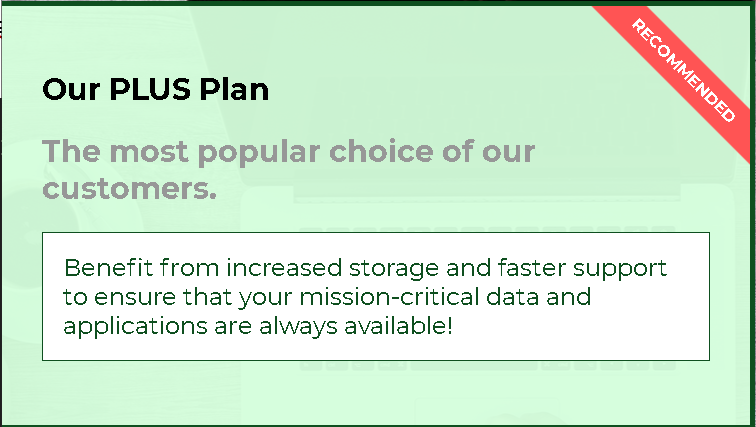
    transform-origin: center;

}

.package{

overflow: hidden;

}



Next, on the Customers page we will add some transformation for the images with the next CSS code:

.testimonial\_\_image {

    transform: skew(-20deg) scale(1.3);

}

.testimonial\_\_image-container {

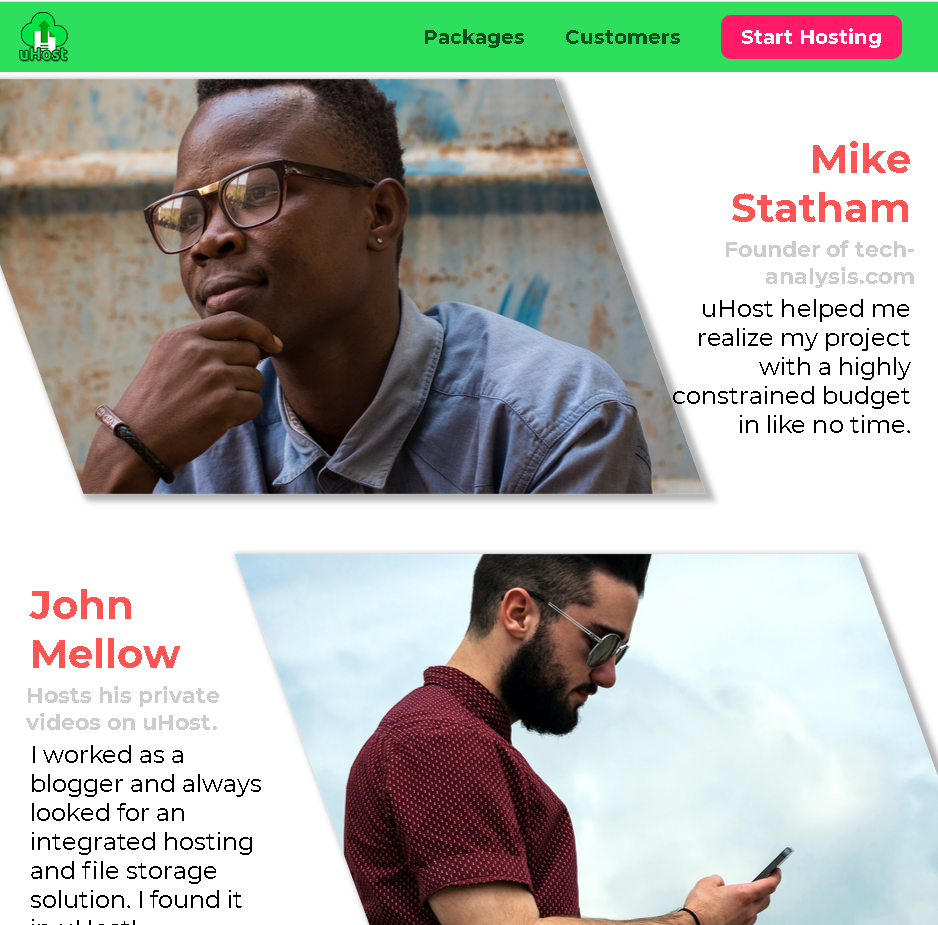
    width: 60%;

    transform: skew(20deg);

    overflow: hidden;

}

This code will make the photos to be in the shape close to a diamond, giving a better aspect to the page. In order for them to be fully in that so called diamond we also used the transform scale so the photo’s width and length will be bigger.



**16.Transitions and Animations in CSS**

For that we will use the transition command:

transition: WHAT DURATION DELAY TIMING-FUNCTION;

For example, transition: opacity 200ms 1s ease-out; can be translated to: "Animate any changes in the opacity property (for the element to which the transition property is applied) over a duration of 200ms. Start fast and end slow, also make sure to wait 1s before you start".

In our project, we will add an animation to our modals for the choose plan. For that we wrote the next CSS code in main file:

.modal {

  opacity: 0;

  transform: translateY(-3rem);

  transition: opacity 0.2s ease-out, transform 0.5s ease-out;

}

And in the shared CSS file we will add the next CSS code :

.open {

    opacity: 1 !important;

    transform: translateY(0) !important;

}

So now, when we will click on the Choose Plan on our main page, the modal with the “Do you want to continue?” will look like it is sliding easily down and if we press on No it will also go back to the main page with the same animation in reverse.

Then, for the side-bar menu that we have on the mobile version we wanted to add another animation that makes the menu slide when we click the three vertical lines. In this case, we will remove display: none and add a translate of the X axis to -100% so the menu will actually be out of our webpage. When we click the burger menu the open will move its position back to the webpage but now it will come with the animation that we set using the next CSS code:

.mobile-nav {

  transform: translateX(-100%);

  transition: transform 300ms ease-out;

}

Then in order to create an animation, for our project we added :

@keyframes wiggle {

  from {

    transform: rotateZ(0);

  }

  to {

    transform: rotateZ(10deg);

  }

}

And then for our Start Hosting button we will add this animation.

.main-nav\_\_item--cta {

  animation: wiggle 200ms 3s 8;

}

The animation: wiggle 200ms 3s 8; can be translated to “Play the wiggle keyframe set ( animation ) over a duration of 200ms. Make sure you wait 3 seconds before you start and play 8 animations “

For this we can also add alternate after 8 in our example will start it from the initial. We can also add forward which will keep the final value of our animation to it.

In order to use more keyframes for our animation we will put instead of from and to 0% and 100%. After that we can just add another frames which will apply at let’s say 50% or whatever part of it that we want.

In our Customers page we will add another animation in order to flip the images of the customers whenever you hover on them. For that we created another keyframe to use :

@keyframes flip-customer {

    0% {

      transform: rotateY(0) skew(20deg);

    }

    50% {

      transform: rotateY(160deg) skew(20deg);

    }

    100% {

      transform: rotateY(360deg) skew(20deg);

    }

}

In this case, we added 3 different keyframes so the animation will look way better. Then we are going to add this animation whenever we hover on the image :

#customer-1:hover .testimonial\_\_image-container {

  animation: flip-customer 1s ease-out forwards;

}

Now, in our Packages page we will add also an animation whenever we hover over a plan which will make it move a little up. For that we will create another keyframes:

@keyframes push-up {

    0% {

        transform: translateY(0);

    }

    33% {

        transform: translateY(-1.5rem);

    }

    100% {

        transform: translateY(-1rem);

    }

}

In this case we used the 0% which is the initial state ( that we already had in our webpage ) , then for 33% we pushed it up by 1.5rem and then we used for 100% to come down a little bit to just 1rem from the initial value. After that we used it as in the next CSS code:

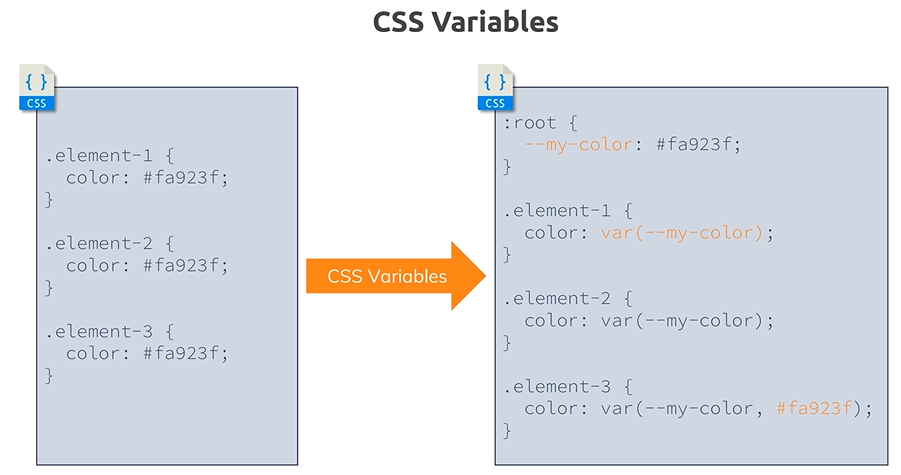
.package:hover,

.package:active {

    animation: push-up 1s ease-out forwards;

}

**17.** **Writing Future-Proof CSS Code**

****

Consider this case; we get CSS code where we get different selectors and let's say they share some common functionality, like a color which you reuse. This could also be some unit though, this could be something like 1rem which you use again and again, CSS variables are a relatively new feature which allow you to put that reused value into a variable which you defined with this syntax, --my-color.

Now this is a property which kind of exists, the browser understands that this now should become a custom variable and then you can reuse that variable with that var function. Also note that the variable here is defined in that root pseudo selector, which in the end refers to the entire document or to every element, you could also place it on a specific element selector but then this variable would only be usable within that selector.

We will apply this to our project because we used the same colors in a lot of places so it will be easier for us to write:

:root {

  --dark-green: #0e4f1f;

  --highlight-color: #ff1b68;

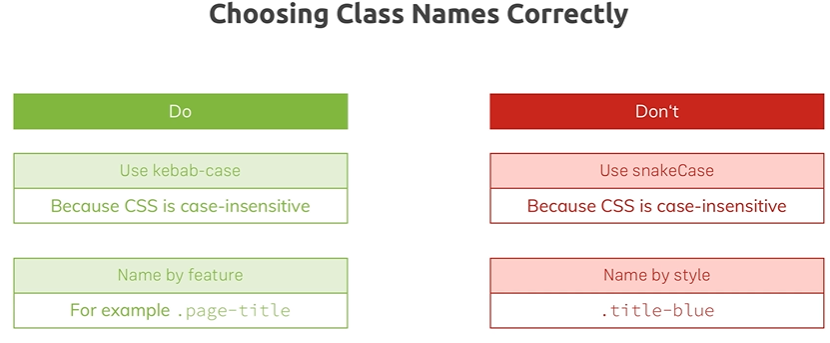
}

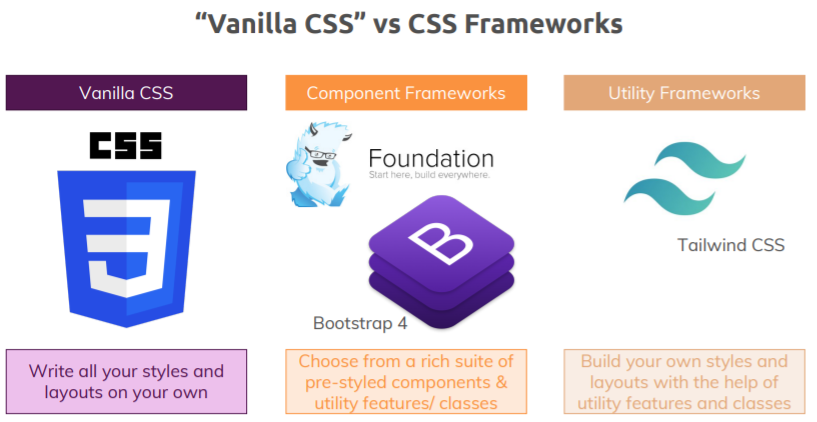
Then, we will just change from the hex code to the var(--dark-green) for example.

A polyfill is a piece of code (usually JavaScript on the Web) used to provide modern functionality on older browsers that do not natively support it.

For example, a polyfill could be used to mimic the functionality of an HTML Canvas element on Microsoft Internet Explorer 7 using a Silverlight plugin or mimic support for CSS rem units, or text-shadow, or whatever you want.

The reason why polyfills are not used exclusively is for better functionality and better performance.





****

**18.** **Introducing Sass (Syntactically Awesome Style Sheets)**

Sass is the most mature, stable, and powerful professional grade CSS extension language in the world. You can think of it like this: during development, you as a developer, want to save time and you want to write clean reusable code that's easy to maintain and to adjust. With CSS, you can quickly end up with huge CSS files. Now managing this can be cumbersome and you also find yourself repeating certain things a lot, so it would be nice if you had a more condensed, easier syntax, maybe something that allows you to nest CSS selectors into each other for example and SASS simply offers such features

The important thing is that you can only use it during development, so it needs to be compiled to normal CSS before you can ship it to production because the browser doesn't understand SASS. So the idea is that you as a developer can write SASS/SCSS code and then you let some tool compile it to normal CSS.

In order to use it we will have to install ruby first and then to install SASS using cmd. After that we will create a new file named main.scss and we will copy our css code. After that if we will open cmd and write : “sass main.scss main.css” it will just convert our sass code.

If we are going to update the code in order to not write that in cmd everytime we can just keep the cmd opened and write “sass --watch main.scss:main.css” and it will automatically update everything.