**About recent project:**

1. **Weather APP**

The Weather App I built allows users to input any city and country and it will give them current weather for that location. I build it with React, Bootstrap and custom CSS. The data if fetched from OpenWeather API with an async/await call. I use a lot of SVGs for silhouettes of the cities and for background that changes depending on weather at the selected location. It’s deployed to GitHub pages and live demo also on my personal website.

**SVG’s** (Scalable Vector Graphics) can be created and edited with any text editor. SVG images can be searched, indexed, scripted, and compressed.

**PROS**: **small size -** SVGs take up much less space than images in raster format;  
**– flexibility. With** [**CSS**](http://en.wikipedia.org/wiki/Cascading_Style_Sheets), you can quickly change the graphics settings on the site, such as background color or the position of the logo on the page. To do this, you can edit the file in any text editor;  
– **scalability** without changing the image quality. It is widely used for devices with [Retina](http://en.wikipedia.org/wiki/Retina) screens etc;  
– it’s possible to **view it in any browser** (IE, Chrome, Opera, Firefox, Safari, etc.).

**CONS:** – the file size is growing very fast, if the object consists of a large number of small elements;  
– it’s impossible to read a part of the graphic object, only the entire object and it slows you down.

**Async/Await** there’s a special syntax to work with promises in a more comfortable fashion, called “async/await”.

**ASYNC:** The word “async” before a function means one simple thing: a function always returns a promise.

So, async ensures that the function returns a promise, and wraps non-promises in it. Simple enough, right? But not only that. There’s another keyword, await, that works only inside async functions, and it’s pretty cool.

async function f() {

return 1;

}

**AWAIT:** The keyword await makes JavaScript wait until that promise settles and returns its result.

// works only inside async functions

let value = await promise

Here’s example with a promise that resolves in 1 second:

async function f() {

let promise = new Promise((resolve, reject) => {

setTimeout(() => resolve("done!"), 1000)

});

let result = await promise; // wait till the promise resolves (\*)

alert(result); // "done!"

}

f();

The function execution “pauses” at the line (\*) and resumes when the promise settles, with result becoming its result. So the code above shows “done!” in one second.

Let’s emphasize: await literally makes JavaScript wait until the promise settles, and then go on with the result. That doesn’t cost any CPU resources, because the engine can do other jobs meanwhile: execute other scripts, handle events etc.

It’s just a more elegant syntax of getting the promise result than promise.then, easier to read and write.

## [Summary](https://javascript.info/async-await" \l "summary)

The async keyword before a function has two effects:

Makes it always return a promise.

Allows to use await in it.

The await keyword before a promise makes JavaScript wait until that promise settles, and then:

If it’s an error, the exception is generated, same as if throw error were called at that very place.

Otherwise, it returns the result, so we can assign it to a value.

Together they provide a great framework to write asynchronous code that is easy both to read and write.

With async/await we rarely need to write promise.then/catch, but we still shouldn’t forget that they are based on promises, because sometimes (e.g. in the outermost scope) we have to use these methods. Also Promise.all is a nice thing to wait for many tasks simultaneously.

1. **CSS3 Animations**

I have built multiple animation projects using CSS, especially tried to take advantage of the new CSS3 features. Examples can be seen in my ‘button hover effects’ and ‘social media button effects’ projects. My favourite CSS3 features are:

1. **Animation + transition, @keyframes**
2. **Calc()**
3. **::before and ::after pseudo elements**

This is especially beneficial in cases where you need extra box shadows or other visual elements that would require extra spans or divs. In the end, you get a more minimal and semantically correct HTML.

1. **Gradients**

give web designers the power to create smooth transitions between colors without having to resort to images. CSS gradients also look great on retina displays, because they are generated on the fly. They can be linear or radial, and can be set to repeat.

1. **Box Sizing**

[CSS](https://developer.mozilla.org/en-US/docs/Web/CSS) property defines how the [user agent](https://developer.mozilla.org/en-US/docs/Glossary/user_agent) should calculate the total width and height of an element.

By default in the [CSS box model](https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Box_Model/Introduction_to_the_CSS_box_model), the width and height you assign to an element is applied only to the element's content box. If the element has any border or padding, this is then added to the width and height to arrive at the size of the box that's rendered on the screen. This means that when you set width and height, you have to adjust the value you give to allow for any border or padding that may be added.

The box-sizing property can be used to adjust this behavior:

**content-box** gives you the default CSS box-sizing behavior. If you set an element's width to 100 pixels, then the element's content box will be 100 pixels wide, and the width of any border or padding will be added to the final rendered width.

**border-box** tells the browser to account for any border and padding in the values you specify for an element's width and height. If you set an element's width to 100 pixels, that 100 pixels will include any border or padding you added, and the content box will shrink to absorb that extra width. This typically makes it much easier to size elements.

1. **Media Queries**

It used to be that you had a regular website, wide enough to fit the smallest monitor resolution used at the time, and a separate mobile website. These days, we build sites that are responsive and which adapt to the type of device, orientation and resolution. all you need to do is to enclose CSS styles in a block guarded by a [code>@media](mailto:code) rule. Each [code>@media](mailto:code) block is activated when one or more conditions are met.

1. **3D transforms**

With CSS3 you can create an imaginary 3D space. To activate 3D space, an element needs perspective. This can be done in two ways: using the transform property, with the perspective attribute value.

1. **Flexbox**

The CSS3 flexbox feature makes it much easier for web developers to arrange HTML elements vertically or horizontally. Website layouts designed with flexbox can respond to users with various screen widths, including mobile devices. To access the feature in CSS, the display property must be set to flex, as seen below:display: flex;

HTML elements that are children of an element with the "main" class are now flex items and can be arranged using flexbox properties. Here are two such properties:

*flex-wrap*: arranges flex items into a single line or wraps them across multiple lines of a webpage layout.

*justify-content*: can orient flex items in a number of different positions on a webpage, including the center.

## Flex Direction

You can also set the direction of each flex item inside the flex-container using the flex-direction property.

There are 4 available values for flex-direction property:

row – This is the default value. This will set the direction of flex-items from left-to-right and top-to-bottom.

row-reverse – sets the direction of flex-items from right to left.

column – sets the flex items vertically.

column-reverse – sets the flex items vertically but in a reverse manner.

## Background-image

The CSS3 background-image property is used to set a background image for an HTML element.

For example, consider the following CSS:

background-image: url("http://image-gallery.io/mountain-scene.png");

## Background-size

The CSS background-size property controls the size of an HTML element's background image. Used along with the background-image property, background-size values control how a background image is proportioned and scaled. One such value is cover as seen in the following CSS:

body {

background-image: url("http://image-gallery.io/mountain-scene.png");

background-size: cover;

}

1. **Background-clip**

Do you remember the days when you needed [**Photoshop**](https://speckyboy.com/category/photoshop/) to create image-based text? Photoshop isn’t needed near as much, as CSS3 introduced a property called background-clip that does exactly the same thing.

You can now clip an image with text using the background-clip property. First, you need the div element for the image to be placed above the h1 or the text element you want to clip. Now by using background-clip property, you can clip the image with the text. Make sure that the text-fill-color is transparent

## Blend Mode

Using the background-blend-mode property, you can [**blend images with colors**](https://speckyboy.com/css-image-filter-toolbox/). Just like in Photoshop, you can use different color effect variation properties (blend modes) such as screen, overlay, darken, lighten, color-dodge, color-burn, hard-light, soft-light, difference, exclusion, hue, saturation, color, and luminosity. Of course, using the normal property will just retain the original.

**HTML5 features:**

1. <figure>
2. <header><footer>
3. <audio><video>
4. Data attribute

<div data-custom-attr="My Value"> </div>

1. Required attribute on form:

<input type="text" name="someInput" required>

<input type="text" name="someInput" required="required"

**About current job – read from CV**

What are the responsibilities of a web developer?

“A web developer designs, develops, enhances, tests and deploys web applications with an end goal of creating engaging and user-friendly site layout and function. A developer gathers and defines requirements, maintains websites, troubleshoots and fixes bugs, follows best practices and collaborates with other teams.”

What sparked your interest in web development?

“As a previous marketing coordinator, I frequently used an online tool to create and customize email campaigns. I found the drag-and-drop feature buggy and discovered that I could achieve the look I was going for if I edited the HTML instead. I became fascinated the ability a web developer has to be imaginative, artistic and technical, while developing a product that benefits and inspires others.”

What is W3C and why is it important?

“W3C stands for World Wide Consortium and it is an international community that focuses on developing and standardizing the web. As a web developer, enforcing these standards ensures that web content is accessible in all browsers to reach all audiences, as well as optimizes the user experience. For example, using W3C-compliant CSS and XML allows every website to function similarly, but also improves SEO.”

Explain how you optimize and reduce web application load time.

“Although there are numerous techniques a web developer can use in reducing load time, I always rely on optimizing images, enabling browser caching and minimizing HTTP requests. My go-to tool for evaluating site speed is Google PageSpeed Insights. In one instance, I successfully reduced page load time from 2.1 seconds to .7 seconds by soley enabling browser caching.”

What steps do you take to balance demanding client requirements?

 “In balancing demanding requirements as a web developer, I take steps to fully understand what is expected of me, prioritize my tasks and keep an open line of communication with the client. Keeping a client pleased is a high priority, although I am not interested in producing buggy code and taking ineffective shortcuts to damage the final product. I keep the client informed of my progress and work efficiently to complete the task.”