



**Politechnika
Warszawska**

Unia Europejska
Europejski Fundusz Społeczny



Projekt „NERW 2 PW. Nauka – Edukacja – Rozwój – Współpraca” współfinansowany jest ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

Od HTMLa do PostGISa Samouczek do laboratorium

dr inż. Michał Okulewicz

Zadanie 10 pn. „Modyfikacja programów studiów na kierunkach prowadzonych przez Wydział Matematyki i Nauk Informatycznych”, realizowane w ramach projektu „NERW 2 PW. Nauka – Edukacja – Rozwój – Współpraca”, współfinansowanego jest ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

Introduction

The purpose of this set of tutorials is to support the technical part of the [From HTML to PostGIS](#) course conducted at the [Warsaw University of Technology](#).

HTML

Hyper Text Markup Language forms the backbone of any website, as it is the language describing the structure of the document. It is a text language with a tree structure. It gives semantic meaning to the parts of the document/application, by utilizing meaningful tag names like `<section>`, `<nav>`igation or ``phasis.

HTML allows for creation of rich text documents, including headers, paragraphs, figures, tables, lists etc.

Getting started

Let's start with creating a basic empty HTML website

```
<!DOCTYPE html> <!-- This specifies this document to be an HTML5 -->
<html lang="en">
  <head>
    <title>This will be shown in the browser tab</title>
    <meta charset="utf-8" />
  </head>
  <body>
    <!-- Website content -->
    <!-- By the way, this is an HTML comment :) -->
  </body>
</html>
```

You may validate this example in through the official [W3C Validator](#)

As you can observe HTML consists of `<elements>` and their key-value details `attribute=value`.

The elements within the HTML standard can be broadly categorized into inline and block elements.

Inline elements

Inline elements are generated as a part of text. They are mainly utilized to mark the function of particular parts of the text.

Examples:

Element	Role	Usually rendered as
<code><h1></code>	A header of the highest order	large text
<code><h6></code>	A header of the smallest order	bold text
<code></code>	A deleted or obsolete part of the text	stroked-through text

Element	Role	Usually rendered as
<code></code>	An important part of the text	bold font
<code></code>	A stressed part of the text	cursive script
<code></code>	A generic part of the text (to be customized by class name)	like surrounding text

Block elements

Block elements take up a rectangular space, taking the whole width of the website (usually without margins). By default they are generated as wall-of-text, so directly one after another.

Examples:

Element	Role
<code><p></code>	Marks a single paragraph
<code><section></code>	A very important part part of the text
<code><nav></code>	An important part of the text
<code><div></code>	A generic block element (to be customized by id or class name)

A complete list of HTML elements can be found within [HTML5 living standard description](#)

Example

Let's consider a following example

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title>Selection from The Lord of The Rings</title>
</head>
<body>
  <h1>Some quotes from &ldquo;The Lord of the Rings&rdquo;</h1>
  <h2>Life</h2>
  <blockquote
cite="https://en.wikiquote.org/wiki/The_Fellowship_of_the_Ring">
    It's a <em>dangerous</em> business, <del>Pippin</del> Frodo, going out
    your door.
    You step onto the road, and if you don't keep your feet,
    there's no knowing where you might be swept off to.
  </blockquote>
  <h2>Afterlife</h2>
  <blockquote cite="https://en.wikiquote.org/wiki/The_Return_of_the_King">
  <p>
    PIPPIN: <i>I didn't think it would end this way.</i><br />
    GANDALF: <i>End? No, the journey doesn't end here.
    Death is just another path, one that we all must take.
```

```
The grey rain-curtain of this world rolls back,  
and all turns to silver glass, and then you see it</i><sup>1</sup>.<br  
</i>  
  
PIPPIN: <i>What? Gandalf? See what?</i><br />  
GANDALF: <i>White shores, and beyond,  
a far green country under a swift sunrise.</i><br />  
PIPPIN: <i>Well, that isn't so bad.</i><br />  
GANDALF: <i>No. No, it isn't.</i>  
</blockquote>  
<hr />  
<small><sup>1</sup>Valinor</small>  
</body>  
</html>
```

This example will be rendered like this (in Google Chrome):

Some quotes from “The Lord of the Rings”

Life

It's a *dangerous* business, ~~Pippin~~Frodo, going out your door. You step onto the road, and if you don't keep your feet, there's no knowing where you might be swept off to.

Afterlife

PIPPIN: *I didn't think it would end this way.*

GANDALF: *End? No, the journey doesn't end here. Death is just another path, one that we all must take. The grey rain-curtain of this world rolls back, and all turns to silver glass, and then you see it¹.*

PIPPIN: *What? Gandalf? See what?*

GANDALF: *White shores, and beyond, a far green country under a swift sunrise.*

PIPPIN: *Well, that isn't so bad.*

GANDALF: *No. No, it isn't.*

¹Valinor

[example-00-html/formatting-no-css.html](#) contains the HTML code of the example.

Further reading

See also: [Basic HTML template explained](#)

Explanation about JS script location should be taken with a pinch of salt, as we are mostly discussing SPA applications.

On a less serious note:

Beware of people using HTML and semantics in the same sentence (in Polish and with strong language): [Pasta o HTMLu](#).

For a more international experience checkout the [XKCD explanations wiki for 1144: Tags](#)

CSS

As could be observed in HTML example, while we are able to create a text document there was not much that we could do for it's formatting. Besides, default formatting of certain elements is only a recommendation and might differ between browsers.

Therefore, if we want to provide a similar experience for the users of graphic browsers we need to use Cascade Style Sheets (CSS) for specifying the formatting and the layout of HTML page elements.

Getting started

The CSS syntax consists of the following elements:

```
selector {  
  property: value[-with-unit];  
}
```

In order to utilize a CSS document together with HTML document an information about CSS must be added in the `<head>` part of HTML

```
<!DOCTYPE html>  
<html>  
  <head>  
    <!-- other metadata -->  
    <link rel="stylesheet" href="css/style.css">  
  </head>  
  <body>  
    <!-- Website content -->  
  </body>  
</html>
```

If we create a `css/style.css` style with such content

```
h1 {  
  font-family: Arial, Helvetica, sans-serif;  
}
```

and apply it to the HTML example, we will get the following result

Some quotes from “The Lord of the Rings”

Life

It's a *dangerous* business, Pippin Frodo, going out your door. You step onto the road, and if you don't keep your feet, there's no knowing where you might be swept off to.

Afterlife

PIPPIN: *I didn't think it would end this way.*

GANDALF: *End? No, the journey doesn't end here. Death is just another path, one that we all must take. The grey rain-curtain of this world rolls back, and all turns to silver glass, and then you see it¹.*

PIPPIN: *What? Gandalf? See what?*

GANDALF: *White shores, and beyond, a far green country under a swift sunrise.*

PIPPIN: *Well, that isn't so bad.*

GANDALF: *No. No, it isn't.*

1) Valinor

As can be observed the font family of the header changes from some serif typeface (probably Times New Roman) into the Arial.

Formatting

Let's investigate some other ways in which we can set the style for the text.

Suppose that we create the following stylesheet

```
h1, h2 { /* font style defined for both h1 and h2 */
  font-family: Arial, Helvetica, sans-serif;
}

h1 { /* font size larger by 1.5 factor than in parent element */
  font-size: 150%;
}

h2 { /* font size larger by 1.2 factor than in parent element */
  font-size: 120%;
}

blockquote { /* text aligned */
  text-align: justify;
}

.hero__name { /* selector for elements with given HTML class */
```

```
    font-variant: small-caps; /* generates text as small caps */
}

.hero__name--wizard { /* gives a more fancy look to wizard's name */
    font-family: Harrington, fantasy;
}
```

defining `font-family` sequence should be ordered from most specific, to most commonly available font ending with a generic family which will be supported by the browser with one of the default fonts of a given style

observe that `hero__name` and `hero__name--wizard` tell us about the function of those elements, and there is nothing related to their visual aspect (which can be changed by designer at the client request without any confusion for developers)

In order for the above approach to work correctly we must apply the classes to HTML elements. So the dialog between Pippin and Gandalf will now look like this.

```
<span class="hero__name">Pippin</span>: <i>I didn't think it would end this way.
</i><br />
<span class="hero__name--wizard">Gandalf</span>: <i>End? No, the journey doesn't
end here.
Death is just another path, one that we all must take.
The grey rain-curtain of this world rolls back,...
```

We have introduced a generic `` element and explained it's function with appropriate class names.

The final result will look like this:

Some quotes from “The Lord of the Rings”

Life

It's a *dangerous* business, Pippin Frodo, going out your door. You step onto the road, and if you don't keep your feet, there's no knowing where you might be swept off to.

Afterlife

PIPPIN: *I didn't think it would end this way.*

Gandalf: *End? No, the journey doesn't end here. Death is just another path, one that we all must take. The grey rain-curtain of this world rolls back, and all turns to silver glass, and then you see [it](#)¹.*

PIPPIN: *What? Gandalf? See what?*

Gandalf: *White shores, and beyond, a far green country under a swift sunrise.*

PIPPIN: *Well, that isn't so bad.*

Gandalf: *No. No, it isn't.*

¹*Valinor*

Some other available CSS properties for formatting:

Property	Meaning	Possible values
color	foreground (text and its decorations) color	#af1245 red rgba(175,18,76,0.6)
background	complex style for setting background color, background image and its position	#af1245 url('background.png') repeat-x fixed center center
border	width, type of border line, color; has variants for each edge and corner	solid 1px black
font	font style, variant, weight, size, line height and family	12pt Arial, sans-serif
text-align	text alignment	left center right justify
text-decoration	lines under, over and through the text and their styles	underline dotted red
text-transform	letter size transformations	uppercase

Layout

However, from the point of view of application design a key feature of CSS is the ability to organize layout of the website.

This time let's start with a screenshot and then we will decompose it into particular HTML elements and CSS properties.

- Articles
- Gallery
- Content

Articles

<p>Artificial intelligence</p> <p>AI is intelligence demonstrated by machines, unlike the natural intelligence displayed by humans and animals, which involves consciousness and emotionality.</p>	<p>Pop music</p> <p>Pop is a genre of popular music that originated in its modern form during the mid-1950s in the United States and the United Kingdom. The terms popular music and pop music are often used interchangeably, although the former describes all music that is popular and includes many disparate styles.</p>	<p>Arizona</p> <p>Arizona is a state in the Southwestern region of the United States. It is also usually considered part of the Mountain states. Arizona is the 48th state and last of the contiguous states to be admitted to the Union, achieving statehood on February 14, 1912.</p>
<p>Vietnam mouse-deer</p> <p>The Vietnam mouse-deer (<i>Tragulus versicolor</i>), also known as the silver-backed chevrotain, is an even-toed ungulate in the family <i>Tragulidae</i> known only from Vietnam.</p>	<p>Attack helicopter</p> <p>An attack helicopter is an armed helicopter with the primary role of an attack aircraft, with the offensive capability of engaging ground targets such as enemy infantry, military vehicles and fortifications. Due to their heavy armament they are sometimes called helicopter gunships.</p>	<p>Anaconda</p> <p>Anaconda is a distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment.</p>

Gallery



Content

xkcd, sometimes styled XKCD, is a webcomic created in 2005 by American author Randall Munroe. The comic's tagline describes it as "A webcomic of romance, sarcasm, math, and language". Munroe states on the comic's website that the name of the comic is not an initialism but "just a word with no phonetic pronunciation".



Fig. 3. Source: <https://xkcd.com/303/>

The subject matter of the comic varies from statements on life and love to mathematical, programming, and scientific in-jokes. Some strips feature simple humor or pop-culture references. It has a cast of stick figures, and the comic occasionally features landscapes, graphs, charts, and intricate mathematical patterns such as fractals. New cartoons are added three times a week, on Mondays, Wednesdays, and Fridays.

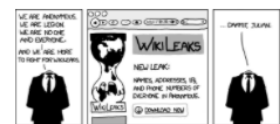


Fig. 3. Source: <https://xkcd.com/834/>

Munroe has released four spinoff books from the comic. The first book, chronologically, published in 2010 and entitled xkcd: volume 0 was a series of select comics from his website. His 2014 book What If? is based on his blog of the same name that answers unusual science questions from readers in a light-hearted way that is scientifically grounded. The What If column on the site is updated with new articles from time to time. His 2015 book Thing Explainer explains scientific concepts using only the one thousand most commonly used words in English. A fourth book, How To, which is described as "a profoundly unhelpful self-help book," was released on September 3, 2019.



Fig. 3. Source: <https://xkcd.com/862/>

This example utilizes 4 ways of dealing with layout:

- the menu and the rest of the content are managed by `position` property and particular coordinates
- the articles are displayed as a `grid`
- the photo gallery is displayed as `flex`
- the figures with xkcd comic strips are defined as floating elements

The complete source code is available in [example-01-css-layout](#)

Utilizing position property

Utilizing `position` was the first concept which arrived as a way to create websites without relying on frames.

In this example we are utilizing two styles `fixed` and `absolute`. Fixed means that the elements are positioned in relation to the browser window. Absolute means that they are positioned in relation to the parent container.

Read more about the [position property in the CSS level 3 specification](#)

First let's observe the HTML setup

```
<nav id="page-menu">
  ...
</nav>
<section id="page-body">
  ...
</section>
```

and then the styles that have been applied to it:

```
#page-menu {
  position: fixed;
  width: 200px;
}

#page-body {
  position: absolute;
  left: 200px;
  right: 1em;
}
```

In this scenario we are setting particular individual elements, hence the utilization of id selectors (e.g. `#page-menu`).

The effect of this styles is best observed if we scroll the page:

[Articles](#)[Gallery](#)[Content](#)

Vietnam mouse-deer

The Vietnam mouse-deer (*Tragulus versicolor*), also known as the silver-backed chevrotain, is an even-toed ungulate in the family *Tragulidae* known only from Vietnam.

Attack helicopter

An attack helicopter is an armed helicopter with the primary role of an attack aircraft, with the offensive capability of engaging ground targets such as enemy infantry, military vehicles and fortifications. Due to their heavy armament they are sometimes called helicopter gunships.

Anaconda

Anaconda is a distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment.

Gallery



The menu stayed in the same place, while the rest of the content moved.

Please note: instead of utilizing the absolute position we could have achieved the same visual result with padding or margins.

Utilizing grid and flex

A more contemporary method is to utilize compose the general layout on the basis of grid and when needed create a layout easily adjusting to the devices of different size with utilization of `grid` and `flex` display style of the elements.

Both of those layouts need a similar structure of HTML with the container and items pattern:

```
<div class="article-organizer">
  <div class="article-item container--minor">
  </div>
  <div class="article-item container--minor">
  </div>
  ...
</div>

<div class="gallery-organizer">
  <div class="gallery-item container--minor">
  </div>
  <div class="article-item container--minor">
  </div>
  ...
</div>
```

You may also observe the ability to set multiple classes to a single element, hence separating the particular aspects of being a container in `grid` or `flex` with the visual aspects of being regarded as an

item of some container.

Here you can observe some of the properties responsible for achieving the result observed previously For grid layout `display` and `grid-template-columns` are the basic properties (and most crucial in this context), defining that we are going to have a grid structure with 3 columns each of the occupying 30% of available horizontal space.

For flexbox layout the important property is `flex-wrap` defining what should be done when the content of flex box is too large to fit in width.

```

.article-organizer {
  display: grid;
  grid-template-columns: 30% 30% 30%;
  gap: 2em 1em;
  justify-content: space-evenly;
  margin-bottom: 0.5em;
}


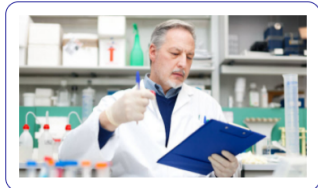



.gallery-organizer {
  display: flex;
  flex-wrap: wrap;
  gap: 1em;
  justify-content: space-evenly;
  margin-top: 0.5em;
}
    
```

The effect of those styles could be best observed when we stretch the website:

Articles

<p>Artificial intelligence</p> <p>AI is intelligence demonstrated by machines, unlike the natural intelligence displayed by humans and animals, which involves consciousness and emotionality.</p>	<p>Pop music</p> <p>Pop is a genre of popular music that originated in its modern form during the mid-1950s in the United States and the United Kingdom. The terms popular music and pop music are often used interchangeably, although the former describes all music that is popular and includes many disparate styles.</p>	<p>Arizona</p> <p>Arizona is a state in the Southwestern region of the United States. It is also usually considered part of the Mountain states. Arizona is the 48th state and last of the contiguous states to be admitted to the Union, achieving statehood on February 14, 1912.</p>
<p>Vietnam mouse-deer</p> <p>The Vietnam mouse-deer (<i>Tragulus versicolor</i>), also known as the silver-backed chevrotain, is an even-toed ungulate in the family <i>Tragulidae</i> known only from Vietnam.</p>	<p>Attack helicopter</p> <p>An attack helicopter is an armed helicopter with the primary role of an attack aircraft, with the offensive capability of engaging ground targets such as enemy infantry, military vehicles and fortifications. Due to their heavy armament they are sometimes called helicopter gunships.</p>	<p>Anaconda</p> <p>Anaconda is a distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment.</p>

Gallery

				
---	---	---	---	---

You may observe how the elements in grid were stretched in order to maintain their desired width, while flex box elements all ended up in a single row.

Utilizing floating elements

Finally, the xkcd figure elements have been placed with the use of `float` property. That way, the elements that come next in HTML surround floating elements. As there is no general way to do the `:nth-of-class()` we are defining our own `--even` class modifier.

```
<figure class="text-illustration-figure">
  
  <figcaption>Fig. 3. Source: <a
href="https://xkcd.com/834/">https://xkcd.com/834/</a></figcaption>
</figure>
<p class="content-text">xkcd, sometimes styled XKCD, is a webcomic created in 2005
by American author Randall Munroe. The..</p>
```

```
.text-illustration-figure {
  text-align: center;
  float: right;
}

.text-illustration-figure--even {
  text-align: center;
  float: left;
}
```

Further reading

1. More about [CSS class naming conventions](#)

The example for 5-year old is very nice. However, it brings the wrong message as to the modifier part in BEM, as it should be related to the purpose of modification rather than the resulting style. So it would be better if it is `.stick-man--ill` or `.stick-man--cold` instead of `.stick-man--red` or `.stick-man--blue`.

Yes, it is the semantics thing. AGAIN.

2. How to create CSS which handles both mobile and desktop devices? (apart from flexible layouts). Learn about [media queries](#)!
3. Grid layout [tutorial](#) and [cheat sheet](#)

Client side JavaScript

For the purpose of this tutorial JavaScript's purpose within a browser environment could be summarized by 6 features:

1. Operating on the HTML's Document Object Model (DOM)

2. Getting feedback from the user through various events (mouse movements and buttons, keyboard keys, tapping, drag&drop)
3. Validating form data (somewhat obsolete due to existence of `pattern` attribute in HTML5)
4. Working with multimedia elements of HTML5 (`canvas`, `audio`, `video`)
5. HTTP data exchange in the background (e.g. AJAX calls)
6. Management of threads (Web Workers)

This tutorial is going to cover the 4th one of the above.

Getting started

In order to use JavaScript within our application it has to be linked to the website. The most proper way to do this is through the `<head>` section in HTML document:

```
<!DOCTYPE html>
<html>
  <head>
    <!-- other metadata -->
    <script type="text/javascript" src="script.js"></script>
  </head>
  <body>
    <!-- Website content -->
  </body>
</html>
```

Within the `script.js` we might place something like:

```
console.info('Hello!')
```

Document Object Model

The basic ability of client side JavaScript is operating on the elements of the website through the [Document Object Model \(DOM\)](#).

Let's consider a case in which we will automatically generate a clickable table of contents on the basis of simple document structure.

Let's assume that we have an HTML document with the body content like this:

```
<body>
  <h1>Introduction</h1>
  <p>The purpose of this example is to show the DOM</p>
  <h1>Discovering the existing structure</h1>
  <p>In this chapter you will learn how to traverse
    the existing structure of the HTML document.
  <h2>Searching for elements by element type</h2>
  <p>In this chapter you will learn about the getElementsByTagName()
```

```
</code>
    and similar methods.</p>
    <h2>Searching for elements by class names</h2>
    <p>In this chapter you will learn about the <code>getElementsByClassName()</code>
</code>
    and similar methods.</p>
    <h2>Searching for particular elements</h2>
    <p>In this chapter you will learn about the <code>getElementById()</code>
    and similar methods.</p>
    <h1>Creating new elements</h1>
    <p>In this chapter you will learn how to utilize
    <code>createElement()</code>,
    <code>createTextNode()</code>
    and <code>appendChild()</code> methods</p>
</body>
```

Let's also support it with the following CSS:

```
#menu {
  position: absolute;
  left: 0px;
  width: 300px;
  top: 0px;
  bottom: 0px;
  overflow: auto;
}

#content {
  position: absolute;
  left: 300px;
  right: 0px;
  top: 0px;
  bottom: 0px;
  padding: 1em 1em 1em 1em;
  overflow: auto;
  text-align: justify;
}
```

Let's start with properly initializing our menu generator and creating `menu` and `content` containers for the desired output and original document.

```
document.addEventListener('DOMContentLoaded', buildMenu, false);

function buildMenu() {
  let menu = document.createElement('div');
  let content = document.createElement('div');
  menu.id = 'menu';
  content.id = 'content';
  content.innerHTML = document.body.innerHTML;
```

```
document.body.innerHTML = '';
document.body.appendChild(menu);
document.body.appendChild(content);

let menuList = document.createElement('ol');
menu.appendChild(menuList);
menuList = traverseContentAndGenerateMenu(content, menuList);
}
```

We have access to the object `document` representing the HTML document and providing methods for elements creation. We have utilized the `createElement(name)` method in order to generate 2 `div`s and we set their `ids` to the ones matching selectors in CSS.

Additionally we moved the content of body by assigning and clearing its `innerHTML` property.

Finally, we have added created `div`s as children of the `body`, thus making them a part of document and visible for the user.

The resulting HTML will look like this:

```
<body>
  <div id="menu">
  </div>
  <div id="content">
    <h1>Introduction</h1>
    <p>The purpose of this example is to show the DOM</p>
    <h1>Discovering the existing structure</h1>
    <p>In this chapter you will learn how to traverse
      the existing structure of the HTML document.
    <h2>Searching for elements by element type</h2>
    <p>In this chapter you will learn about the getElementsByTagName()
    </code>
      and similar methods.</p>
    <h2>Searching for elements by class names</h2>
    <p>In this chapter you will learn about the getElementsByClassName()
    </code>
      and similar methods.</p>
    <h2>Searching for particular elements</h2>
    <p>In this chapter you will learn about the getElementById()
    </code>
      and similar methods.</p>
    <h1>Creating new elements</h1>
    <p>In this chapter you will learn how to utilize
      createElement(),
      createTextNode()
      and appendChild() methods</p>
  </div>
</body>
```

Now it will be the goal of `traverseContentAndGenerateMenu()` to actually create the menu items.


```
function traverseContentAndGenerateMenu(content, menuList) {
  let header = content.firstChild;
  let lastLevel = 1;
  do {
    if (header.tagName.toLowerCase().startsWith('h')) {
      let currentLevel = Number(header.tagName[1]);
      if (currentLevel > lastLevel) {
        menuList = nestElementsOnTheBasisOfCurrentLevel(currentLevel,
lastLevel, menuList);
      } else if (currentLevel < lastLevel) {
        menuList = getBackToLastLevelInMenu(lastLevel, currentLevel,
menuList);
      }

      const headerText = header.innerHTML;
      let idFromHeader = generateIdFromText(headerText);
      header.id = idFromHeader;
      generateMenuItem(headerText, idFromHeader, menuList);

      lastLevel = currentLevel;
    }
    header = header.nextElementSibling;
  } while (header);
  return menuList;
}
```

This method traverses the children of content `div`, verifying if the element is a header by reading its `tagName` property. It moves through the structure of the content with iterator approach, initializing it with `content.firstChild` and getting the subsequent elements with `header.nextElementSibling`, until there are no more siblings to be found.

It needed a few helper functions which are listed below:

```
function nestElementsOnTheBasisOfCurrentLevel(currentLevel, lastLevel, menuList) {
  let levelDiff = currentLevel - lastLevel;
  while (levelDiff-- > 0) {
    let tempMenuList = document.createElement('ol');
    menuList.appendChild(tempMenuList);
    menuList = tempMenuList;
  }
  return menuList;
}

function getBackToLastLevelInMenu(lastLevel, currentLevel, menuList) {
  let levelDiff = currentLevel - lastLevel;
  while (levelDiff-- > 0) {
    menuList = menuList.parentElement;
  }
  return menuList;
}
```

```
function generateIdFromText(headerText) {
  let nonCharRegex = /^[^a-z]/g;
  let idedValue = headerText.toLowerCase().replace(nonCharRegex, '-');
  return idedValue;
}
```

and the `generateMenuItem` function responsible for the actual generation of menu items (observe how `createTextNode()` differs from `createElement()` function):

```
function generateMenuItem(headerText, headerId, menuList) {
  let menuItem = document.createTextNode(headerText);
  let menuItemLi = document.createElement('li');
  let menuItemA = document.createElement('a');
  menuItemA.href = '#' + headerId;
  menuItemA.appendChild(menuItem);
  menuItemLi.appendChild(menuItemA);
  menuList.appendChild(menuItemLi);
}
```

A complete code can be found in [example-02-js-dom](#), and the generated HTML and its render look similar to this:

```
<body>
  <div id="menu">
    <ol>
      <li><a href="#introduction">Introduction</a></li>
      <li><a href="#discovering-the-existing-structure">Discovering the
existing structure</a></li>
      <ol>
        <li><a href="#searching-for-elements-by-element-type">Searching
for elements by element type</a></li>
        <li><a href="#searching-for-elements-by-class-names">Searching for
elements by class names</a></li>
        <li><a href="#searching-for-particular-elements">Searching for
particular elements</a></li></ol>
      <li><a href="#creating-new-elements">Creating new elements</a></li>
    </ol>
  </div>
  <div id="content">
    <h1 id="introduction">Introduction</h1>
    <p>The purpose of this example is to show the DOM</p>
    <h1 id="discovering-the-existing-structure">Discovering the existing
structure</h1>
    ...
  </div>
</body>
```

1. Introduction
2. Discovering the existing structure
 1. Searching for elements by element type
 2. Searching for elements by class names
 3. Searching for particular elements
3. Creating new elements

Introduction

The purpose of this example is to show the DOM

Discovering the existing structure

In this chapter you will learn how to traverse the existing structure of the HTML document.

Searching for elements by element type

In this chapter you will learn about the `getElementsByTagName()` and similar methods.

Searching for elements by class names

In this chapter you will learn about the `getElementsByClassName()` and similar methods.

Searching for particular elements

In this chapter you will learn about the `getElementById()` and similar methods.

Creating new elements

In this chapter you will learn how to utilize `createElement()`, `createTextNode()` and `appendChild()` methods

Debugging

In case you want to verify what exactly is happening in your JavaScript code you may use developers tools, which are part of most browsers (Chrome example given - available on pressing **F12** and tab **Sources**):

The screenshot shows a browser window with the page content:

Introduction

The purpose of this example is to show the DOM

Discovering the existing structure

In this chapter you will learn how to traverse the existing structure of the HTML document.

 The developer tools are open to the 'Sources' tab. The file explorer on the left shows the file structure:

- top
- file:///
 - C:/Users/micha.ok
 - css
 - scripts
 - document.html

 The code editor shows the following JavaScript code:


```

21 let header = content.firstChild; header = h1#introduction {align: ""}
22 let lastLevel = 1; lastLevel = 1
23 do {
24   if (header.tagName.toLowerCase().startsWith('h')) { header = h1#i
25     let currentLevel = Number(header.tagName[1]);
26     if (currentLevel > lastLevel) { lastLevel = 1
27       menuList = nestElementsOnTheBasisOfCurrentLevel(currentLevel
28     } else if (currentLevel < lastLevel) {
29       menuList = getBackToLastLevelInMenu(lastLevel, currentLeve
30     }
31
32     const headerText = header.innerHTML; header = h1#introduction
33     let idFromHeader = generateIdFromText(headerText);
34     header.id = idFromHeader; header = h1#introduction {align: ""
35     generateMenuItem(headerText, idFromHeader, menuList); menuLis
36
37     lastLevel = currentLevel;
38   }
39   header = header.nextElementSibling;
40
    
```

 The debugger sidebar on the right shows:

- Debugger paused
- Call Stack:
 - traverseContentAndGenerate... scriptjs:37
 - buildMenu scriptjs:17
- Scope:
 - Block:
 - currentLevel: 1
 - headerText: "Introduction"
 - idFromHeader: "introducio...
 - Local:
 - content: div#content
 - header: h1#introduction
 - lastLevel: 1
 - menuList: ol

Events

Sample mouse events are discussed within **Drawing on canvas**.

Drawing on canvas

HTML5, among other features, introduced a method of drawing within the browser environment on the element called `<canvas>`. This brought the final demise of the Flash based applications and allowed to create JS versions of such timeless games like [Asteroids](#) or [Digger](#).

As any proper artist, in order to draw in the browser we need to prepare our... well... canvas (in HTML):

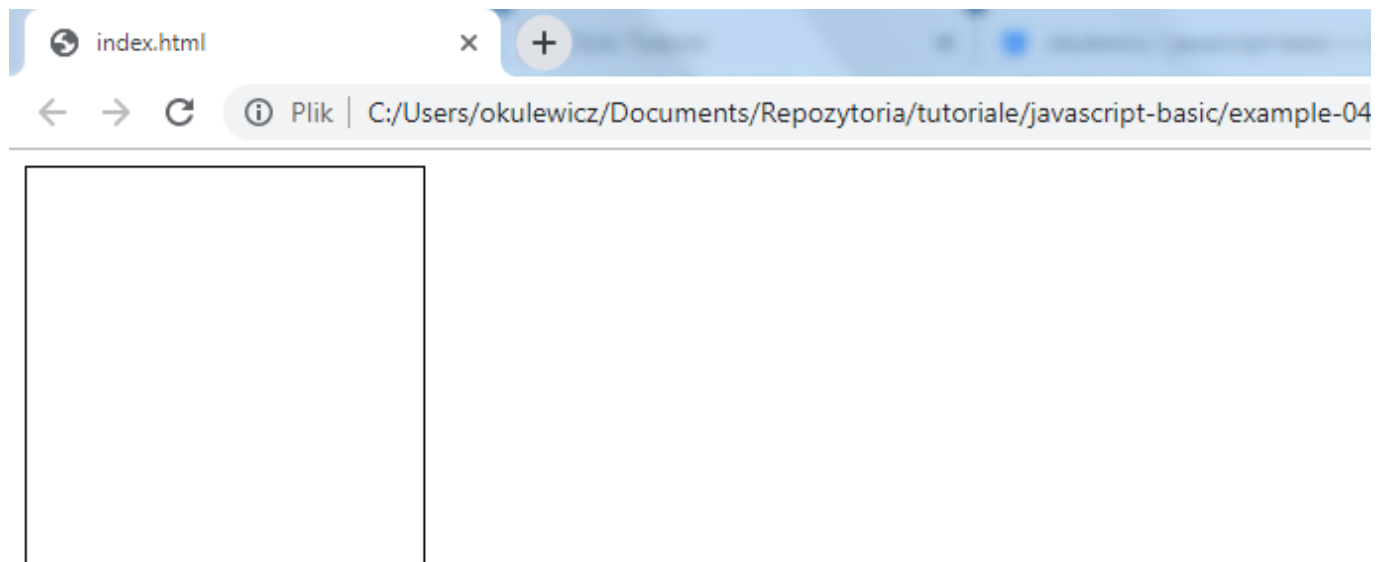
```
<!DOCTYPE html>
<html>
  <head>
    <!-- other metadata -->
    <link rel="stylesheet" type="text/css" href="style.css" />
    <script type="text/javascript" src="script.js"></script>
  </head>
  <body>
    <div id="canvas-container" class="canvas-container">
      <canvas id="canvas-element" class="canvas-element" width="200"
height="200"></canvas>
    </div>
  </body>
</html>
```

The `width` and `height` attributes of the canvas define the *virtual* units of the canvas. The size of the actual element needs to be defined through the CSS (`style.css` file in the example).

```
.canvas-container {
  border: 1px solid black; /* just to see the border */
}

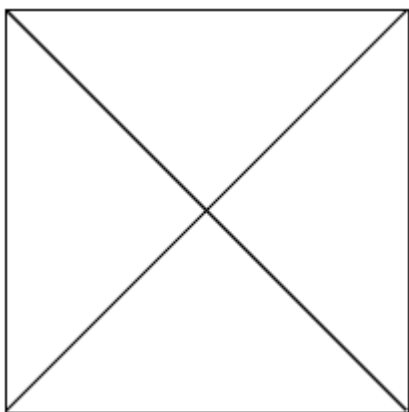
.canvas-element, .canvas-container {
  width: 200px;
  height: 200px;
}
```

Currently nothing interesting should have happend yet, and the result should be similar to the following screenshot:



As we do have all necessary elements in place it is time to focus on the important part: the actual JavaScript code. The actual drawing will happen on the object called context, and for the purpose of this tutorial we are going to focus on 2D drawing only (but true 3D is a possibility!).

Lets starting with drawing a simple figure, like a large X on the whole canvas element:



Such a product can be created by a following piece of code:

```
function initializeApplication() {
  var ctx = document
  //querying for an element with a given ID
  .getElementById("canvas-element")
  //getting a 2-dimensional drawing context
  .getContext('2d');
  //drawing the first line
  ctx.moveTo(0,0);
  ctx.lineTo(199,199);
  ctx.stroke();
  //drawing the second line
  ctx.moveTo(199,0);
  ctx.lineTo(0,199);
  ctx.stroke();
}
```

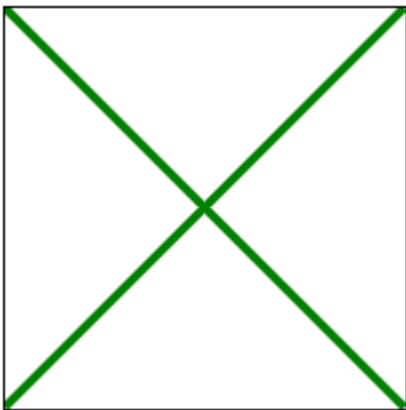
```
}

/* Attaching initialization handler after the HTML document is loaded */
document.addEventListener("DOMContentLoaded", initializeApplication);
```

Supposing we would like to add a little bit more colors? Than we need to set the style of our strokes (or fills). Let's try the following code for the body of `initializeApplication` function:

```
ctx.moveTo(0,0);
ctx.lineTo(199,199);
ctx.strokeStyle = 'blue';
ctx.lineWidth = 2;
ctx.stroke();
ctx.moveTo(199,0);
ctx.lineTo(0,199);
ctx.strokeStyle = 'green';
ctx.lineWidth = 4;
ctx.stroke();
```

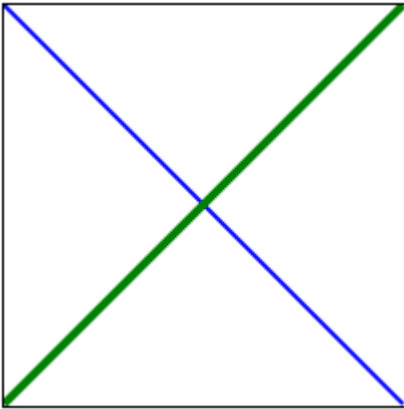
The result should be somewhat unexpected and unsatisfying:



So what went wrong? The answer is in the way we handled the context. In order to get different colors for those two strokes, we need to have two different paths. You may check now the following code:

```
ctx.beginPath();
ctx.moveTo(0,0);
ctx.lineTo(199,199);
ctx.strokeStyle = 'blue';
ctx.lineWidth = 2;
ctx.stroke();
//opening new path will preserve the style
ctx.beginPath();
ctx.moveTo(199,0);
ctx.lineTo(0,199);
ctx.strokeStyle = 'green';
ctx.lineWidth = 4;
ctx.stroke();
```

And the result? Should be the following:



Let's finish with adding user interaction into the whole picture. Now the center of X formed by the green and blue lines should follow the location of the mouse cursor (obviously, while over canvas).

Instead of drawing on the canvas after loading the document, we need to handle mouse movement over the canvas. Please observe, that this time we will need to handle an `event` argument within the event handler.

```
function drawX(event) {
  //actual drawing will happen here
}

function initializeApplication() {
  //getting the HTML element
  var canvas = document.getElementById("canvas-element");
  //attaching a mouse move handler - drawX
  canvas.addEventListener("mousemove", drawX);
}

document.addEventListener("DOMContentLoaded", initializeApplication);
```

For the sake of actual drawing we will need to capture two properties:

- the location of the canvas within the website
- the location of the mouse within the website

Note Theoretically there exist `offsetX` and `offsetY` properties giving us direct access to mouse location within the container, but as of this moment they are not considered stable (see: `offsetX`).

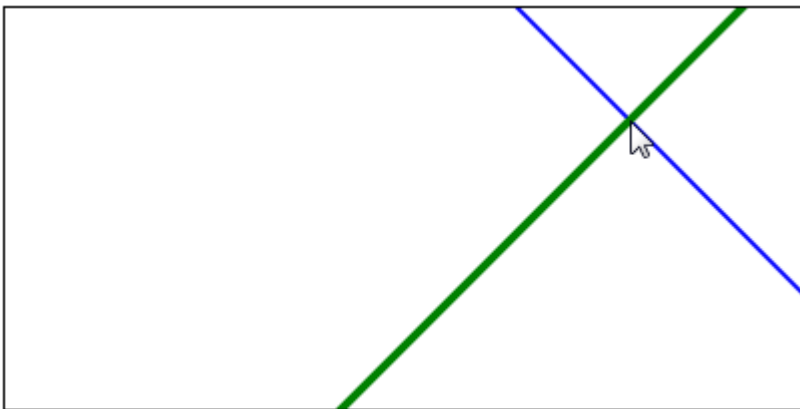
Anyway, in order to follow the mouse movement with our colorful X, we may use the following code within the `drawX` function:

```
//get all the necessary properties (this is THE canvas in drawX)
var x = event.clientX - this.getBoundingClientRect().left;
var y = event.clientY - this.getBoundingClientRect().top;
var width = this.getBoundingClientRect().width;
```

```
var height = this.getBoundingClientRect().height;
var safeDistance = Math.max(width,height);

//please observe that now we can safely change dimensions of canvas
//(however, simultaneously in HTML and CSS) and the result would still be
appropriate
var ctx = this.getContext('2d');
ctx.clearRect(0,0,width,height);
ctx.beginPath();
ctx.moveTo(x-safeDistance,y-safeDistance);
ctx.lineTo(x+safeDistance,y+safeDistance);
ctx.strokeStyle = 'blue';
ctx.lineWidth = 2;
ctx.stroke();
ctx.beginPath();
ctx.moveTo(x+safeDistance,y-safeDistance);
ctx.lineTo(x-safeDistance,y+safeDistance);
ctx.strokeStyle = 'green';
ctx.lineWidth = 4;
ctx.stroke();
```

As a reminder: `this` keyword in the context of event handler means the element that raised the event (i.e. canvas in this case). Meanwhile, `event` argument as an event raised by mouse movement has the properties connected with mouse position and mouse buttons (not used in this example).



See also:

- `getBoundingClientRect()` - <https://developer.mozilla.org/pl/docs/Web/API/Element/getBoundingClientRect>
- `MouseEvent` - <https://developer.mozilla.org/en-US/docs/Web/API/MouseEvent>

for more details.

[example-04-drawing-on-canvas](#) holds the final version of the code.

I also encourage you too check a more complex example of a [simple point & click game](#)

Leaflet API

After introducing plain JavaScript operations we may introduce one of the custom JavaScript map APIs: Leaflet. The other two most commonly used are: OpenLayers and GoogleMaps API. OpenLayers is an open source library best suited for professional GIS applications, while GoogleMaps for easy to write visually pleasing applications, with short lifespan (due to volatile nature of Google Maps API versions).

Leaflet API is also an open source library which can be downloaded [from here](#), intended to be [simple \(which means limited functionality of the core lib\)](#), but at the same type efficient and simple to use.

Basic map example

In order to place a simple map in our web application we need to setup leaflet in the `<head>` section and create and style an HTML container (like `<div>`) for the map to be rendered.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Basic map</title>
  <link rel="stylesheet" href="libs/leaflet.1.7.1/leaflet.css" />
  <script src="libs/leaflet.1.7.1/leaflet.js"></script>
  <script src="scripts/script.js"></script>
  <style>
    #map-panel {
      position: absolute;
      left: 0;
      top: 0;
      right: 0;
      bottom: 0;
    }
  </style>
</head>
<body>
  <div id="map-panel">
  </div>
</div>
</body>
</html>
```

Having this `<div>` container we may setup and display the map like this in JavaScript:

```
const mapContainerId = 'map-panel';
let map;

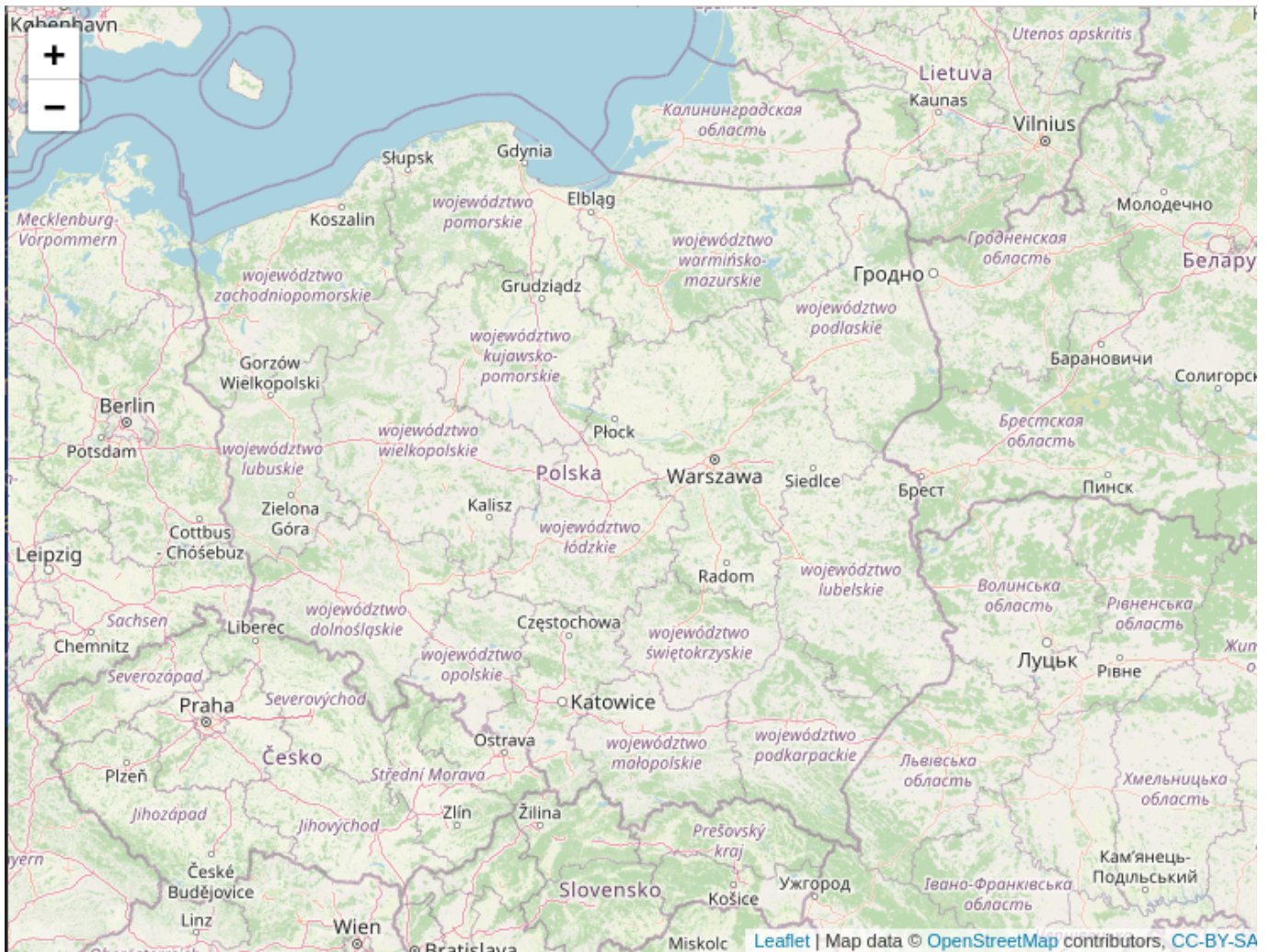
document.addEventListener('DOMContentLoaded', initializeApplication, false);

function initializeApplication() {
  loadMap();
}

function loadMap() {
```

```
//add map and set its center to specific location
map = L.map(mapContainerId).setView([52.05, 20.00], 6);
//add OSM as background layer
let osmLayer = L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
  attribution: 'Map data &copy; <a href="https://www.openstreetmap.org/">' +
    'OpenStreetMap</a> contributors, ' +
    '<a href="https://creativecommons.org/licenses/by-sa/2.0/">CC-BY-SA</a>',
  maxZoom: 19,
  id: 'osm.tiles'
});
osmLayer.addTo(map);
}
```

The above code utilizes [Open Street Map](#) data and centers the map roughly around the territory of Poland:



The code is available as the [example-03-leaflet/basic-map](#)

Presenting external data on the map

In order to enrich the map with some additional data we shall utilize Polish geoportal data about the protected environment sites like national and landscape parks (and enhance previous `loadMap()` function).

Checkout [GDOS WMS GetCapabilities](#) in order to see all available layers.

```
function loadMap() {
  //add map and set its center to specific location
  map = L.map(mapContainerId).setView([52.05, 20.00], 6);
  //add OSM as background layer
  let osmLayer = L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
    attribution: 'Map data &copy; <a href="https://www.openstreetmap.org/">' +
      'OpenStreetMap</a> contributors, ' +
      '<a href="https://creativecommons.org/licenses/by-sa/2.0/">CC-BY-SA</a>',
    maxZoom: 19,
    id: 'osm.tiles'
  });
  osmLayer.addTo(map);

  var gdosWMSNatura2000ock = L.tileLayer.wms("https://sdi.gdos.gov.pl/wms", {
    layers: ['GDOS:ObszaryChronionegoKrajobrazu'],
    format: 'image/png',
    transparent: true,
    attribution: "GDOS"
  });

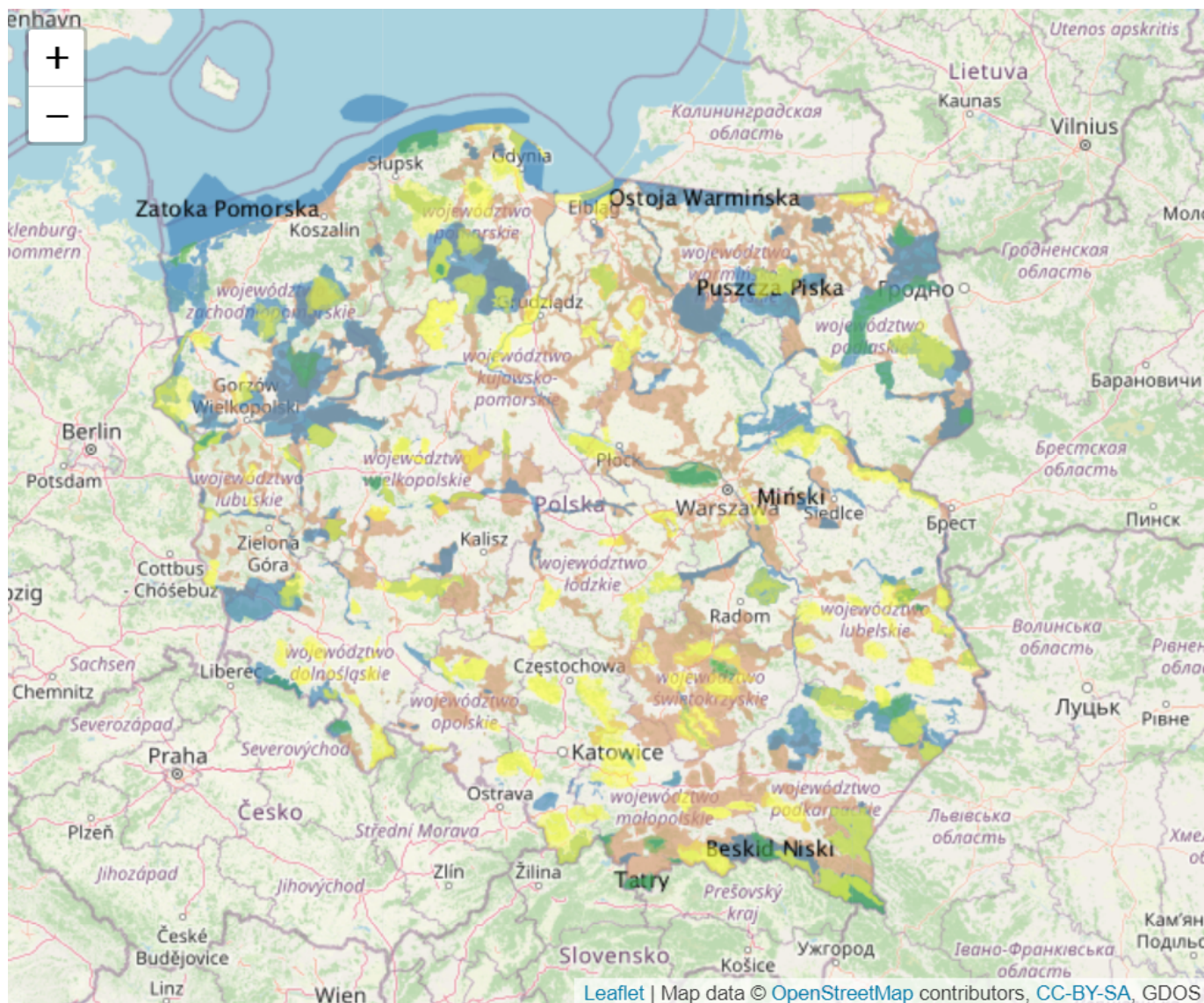
  var gdosWMSNatura2000oso = L.tileLayer.wms("https://sdi.gdos.gov.pl/wms", {
    layers: ['GDOS:ObszarySpecjalnejOchrony'],
    format: 'image/png',
    transparent: true,
    attribution: "GDOS"
  });

  var gdosWMSNationalParks = L.tileLayer.wms("https://sdi.gdos.gov.pl/wms", {
    layers: ['GDOS:ParkiNarodowe'],
    format: 'image/png',
    transparent: true,
    attribution: "GDOS"
  });

  var gdosWMSLandscapeParks = L.tileLayer.wms("https://sdi.gdos.gov.pl/wms", {
    layers: ['GDOS:ParkiKrajobrazowe'],
    format: 'image/png',
    transparent: true,
    attribution: "GDOS"
  });

  var gdosWMSLayerGroup = L.layerGroup()
    .addLayer(gdosWMSNatura2000ock)
    .addLayer(gdosWMSNatura2000oso)
    .addLayer(gdosWMSNationalParks)
    .addLayer(gdosWMSLandscapeParks);
  gdosWMSLayerGroup.addTo(map);
}
```

The result should be similar to this:



The code is available as the [example-03-leaflet/external-wms-map] ([https://bitbucket.org/okulewicz/javascript-basic/src/master/example-03-leaflet/external-wms-map /](https://bitbucket.org/okulewicz/javascript-basic/src/master/example-03-leaflet/external-wms-map/))

Adding custom data to the map

Alternatively we may add some custom markers with simple popups.

```
const mapContainerId = 'map-panel';
let map;

document.addEventListener('DOMContentLoaded', initializeApplication, false);

function initializeApplication() {
  loadMap();
}

function loadMap() {
  //add map and set its center to specific location
  map = L.map(mapContainerId).setView([52.05, 20.00], 6);
```

```

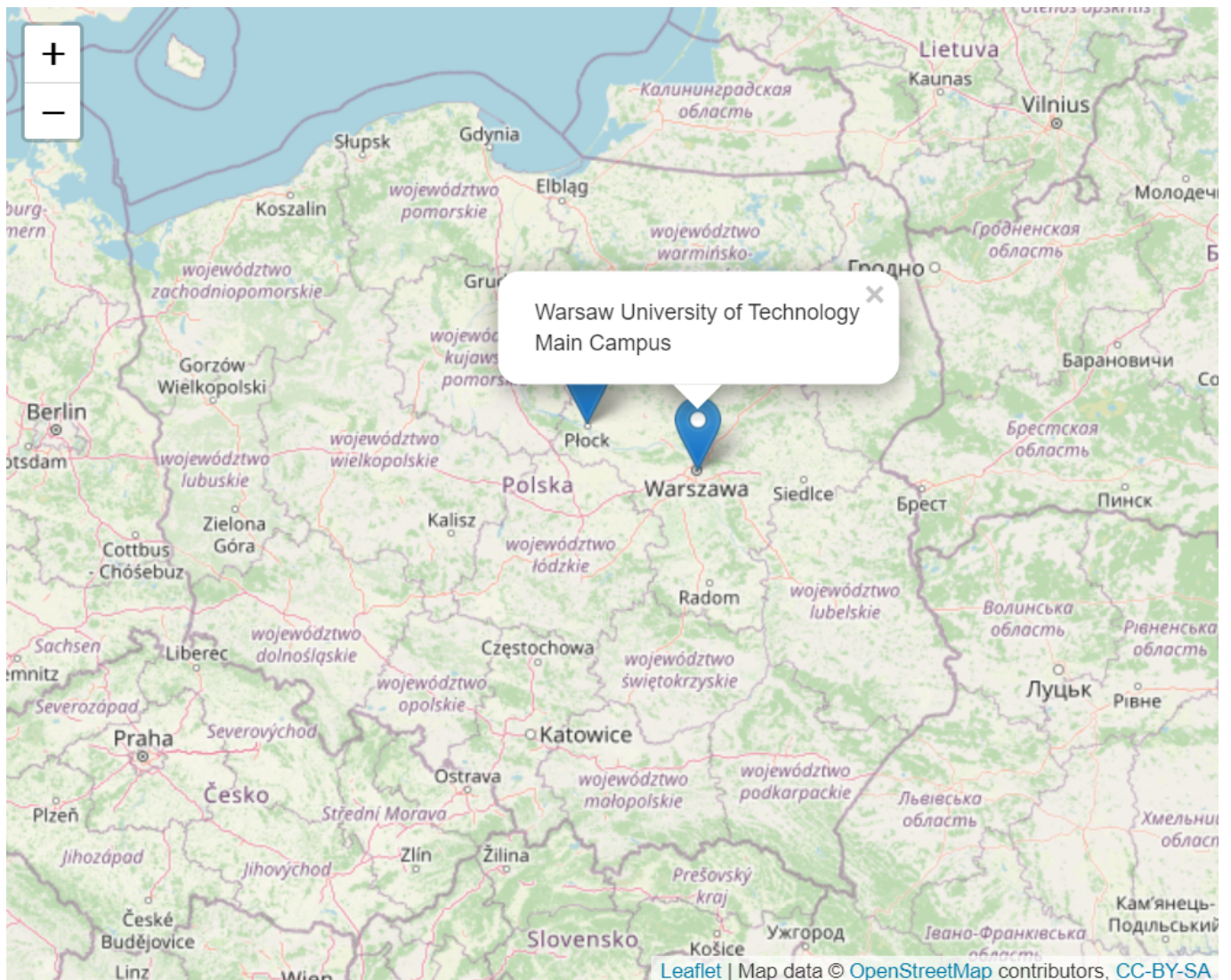
//add OSM as background layer
let osmLayer = L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
  attribution: 'Map data &copy; <a href="https://www.openstreetmap.org/">' +
    'OpenStreetMap</a> contributors, ' +
    '<a href="https://creativecommons.org/licenses/by-sa/2.0/">CC-BY-SA</a>',
  maxZoom: 19,
  id: 'osm.tiles'
});
osmLayer.addTo(map);

let marker1 = L.marker([52.2217988, 21.0071782]);
let marker2 = L.marker([52.5617037, 19.6790427]);
marker1.bindPopup('Warsaw University of Technology<br />Main Campus');
marker2.bindPopup('Warsaw University of Technology<br />City of Płock
Division');

marker1.addTo(map);
marker2.addTo(map);
}

```

The result after clicking on one of the markers should be the following:



The code is available as the [example-03-leaflet/custom-markers-map](#)

Further reading

- [Leaflet tutorials](#)
- [Leaflet documentation](#)
- [Leaflet plugins](#)

REST APIs

After learning the basics of client side JavaScript it is time to introduce the backend (server) aspect of the web applications. Currently the most common approach is to use Representational state transfer (REST) API, which takes advantage of the HTTP protocol, and accesses the server as a set of resources, somewhat resembling a CRUD model known from SQL. Additionally a mature REST service would also be self-describing, however we are not going to get into details about that (please checkout the term [HATEOAS](#)).

The philosophy of REST websites is based on the construction of a server application in order to operate on it as an access to resources through appropriate URI addresses.

HTTP request	URI address	Meaning
GET	http://myapp.domain.org/api/users	Gets a list of users
GET	http://myapp.domain.org/api/users/1	Gets the user with ID 1
POST	http://myapp.domain.org/api/users	Request to create a user (with user info sent in a body of the message)
PUT	http://myapp.domain.org/api/users/1	Request to update a user with id=1 (with user info sent in a body of the message)
DELETE	http://myapp.domain.org/api/users/1	Deletes the user with id=1

A good API uses HTTP protocol messages to inform about the status of the operation, for example:

HTTP request	HTTP message code	Meaning
GET	200 - OK	The resource exists and has been returned
GET	401 - Unauthorized	The requested resource exists but cannot be returned due to lack of permission
GET	404 - Not Found	The requested resource does not exist
POST	201 - Created	The requested resource has been created

Java Example: Micronaut

To prepare REST services in Java, we need one of the libraries that allow us to run a web server and map URI addresses to classes that support them.

Micronaut is one of the simpler frameworks supporting REST APIs.

To be able to use it, we need to perform [few configuration steps](#).

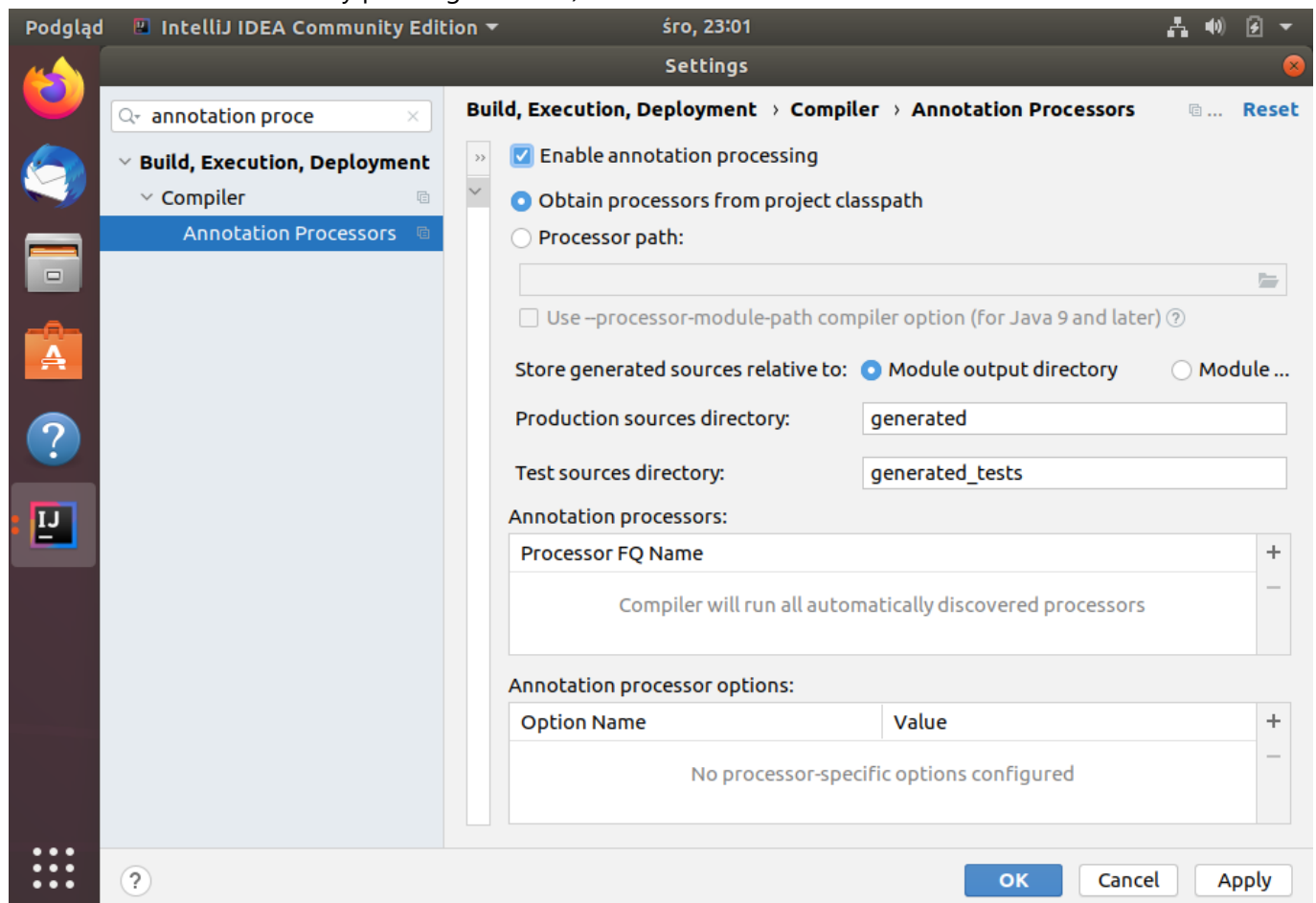
By following the instructions from [SDK MAN](#):

```
#install curl to download SDK installer
sudo apt-get update
sudo apt-get -y install curl
#download SDK installer
curl -s "https://get.sdkman.io" | bash
#install micronaut CLI (command line interface)
source "~/.sdkman/bin/sdkman-init.sh"
sdk install micronaut
#create application (first navigate to location of your choice)
mn create-app FirstMicronautOnTheMoon
```

For this part we are going to use [IntelliJ IDEA Community Edition](#), which will give us enough support for Java development.

However for HTML, CSS and JavaScript development I recommend [Visual Studio Code](#).

After importing the project `FirstMicronautOnTheMoon` in IntelliJ, turn on the annotation support (search for [Annotation Processors](#) by pressing `2xshift`):



Example

From the point of view of web applications the most important elements of the server is a controller. Controller is responsible for mapping and deserializing HTTP requests into particular methods.

```
package edu.html2postgis.controllers;

import io.micronaut.http.HttpResponse;
import io.micronaut.http.annotation.Controller;
import io.micronaut.http.annotation.Get;
import edu.html2postgis.dto.User;

//this annotation defines the address where the controller will be made available
@Controller("/users")
public class UserController {
    //this annotation marks the HTTP method mapped to the Java method
    @Get()
    HttpResponse getUsers() {
        //we assume that there is a static getAllUsersList()
        //HttpResponse.ok() generates HTTP status 200 and wraps serialized
        //users list in HTTP response
        return HttpResponse.ok(User.getAllUsersList());
    }
}
```

The implementation for `User` class may look something like this

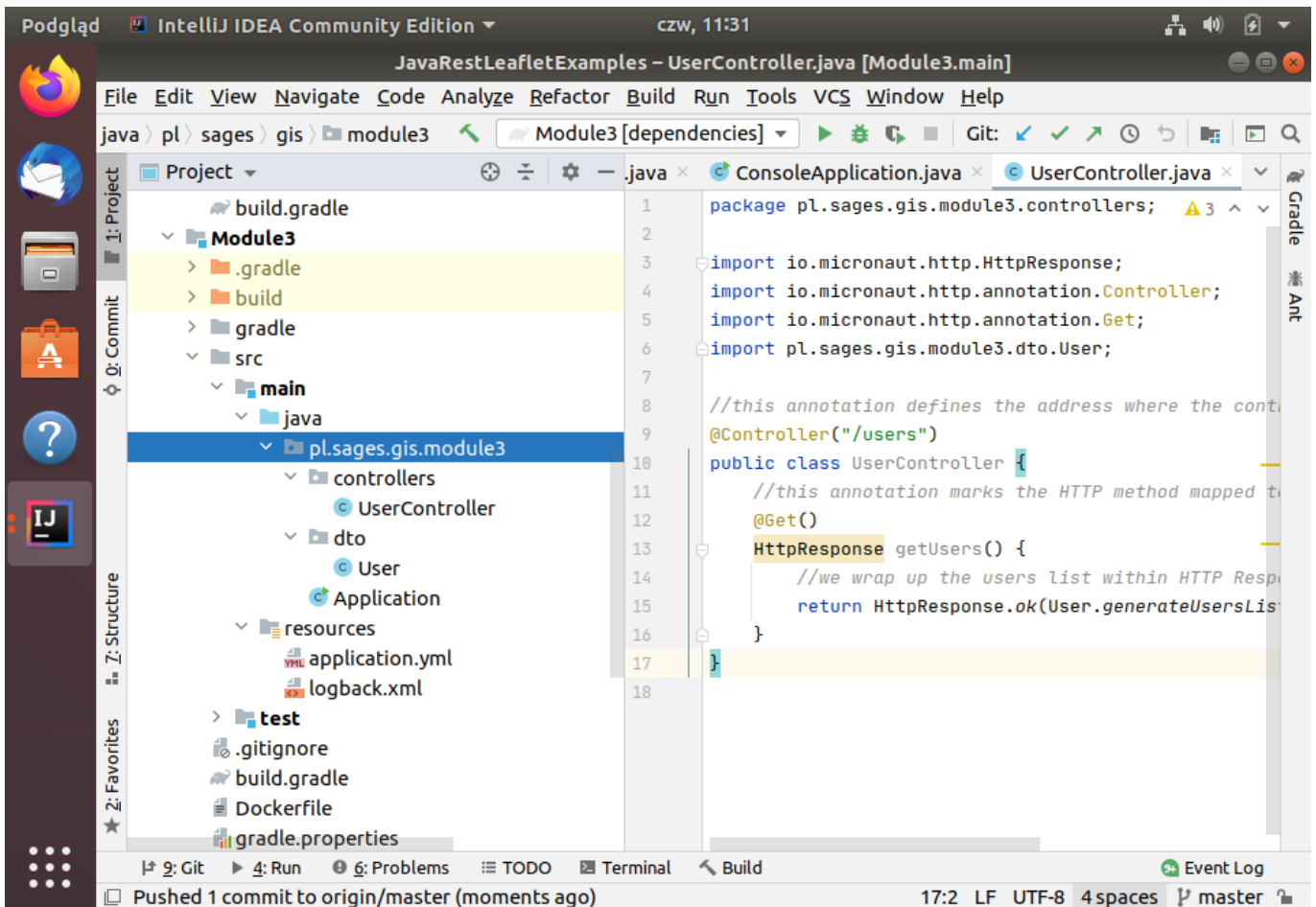
```
package edu.html2postgis.controllers;

import java.util.*;

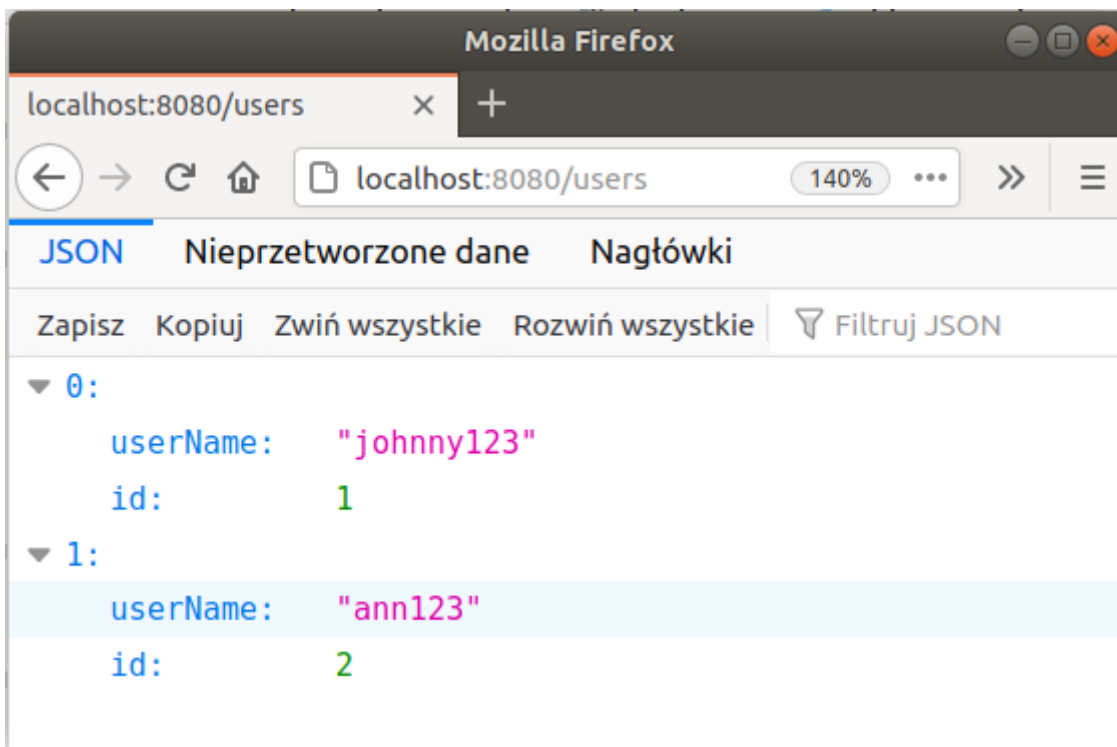
public class User {
    //...rest of the User class code
    public static List<User> getAllUsersList() {
        List<User> users = new ArrayList<>();
        tryAddUserToList(users, 1L, "johnny123");
        tryAddUserToList(users, 2L, "ann123");
        return users;
    }

    private static void tryAddUserToList(List<User> users, long id, String name) {
        try {
            User user = new User(id,name);
            users.add(user);
        } catch (IllegalArgumentException ex) {
            System.err.println(ex.getMessage());
        }
    }
}
```

View of out project in IntelliJ will look similar to this:



If we ran our application and call `http://localhost:8080/users` address in the browser we should see a response like this:



In case we want to add static resources (HTML, CSS, JavaScript, images files) to our web application we need to [allow server to serve static files](#) we need to add `router` section to `src/main/resources/application.yml` configuration file:

```
micronaut:
#rest of the code is hidden
  router:
    static-resources:
      default:
        enabled: true
        mapping: "/*"
        paths: "classpath:public"
```

Then we add `public` subdirectory in `src/main/resources/` and, in the case of this example an `index.html` with the following content:

```
<!DOCTYPE html>
<html lang="pl">
<head>
  <meta charset="UTF-8">
  <title>Main page of the Micronaut based web application</title>
</head>
<body>
  <!-- link to users controller -->
  <a href="/users">Users list [JSON]</a>
</body>
</html>
```

.NET Core WebAPI

The structure of REST API based web applications in .NET is quite similar.

We are using .NET Core to make the application independent of the operating system

In order to create .NET Core application install and register snap alias for newest .NET Core SDK (Ubuntu version below)

```
sudo snap install dotnet-sdk --classic`
sudo snap alias dotnet-sdk.dotnet dotnet
```

I also have had to add a symbolic link from `$HOME/.local/bin/dotnet` to `/snap/dotnet-sdk/current/dotnet`

In order to create a new .NET Core WebAPI project we run:

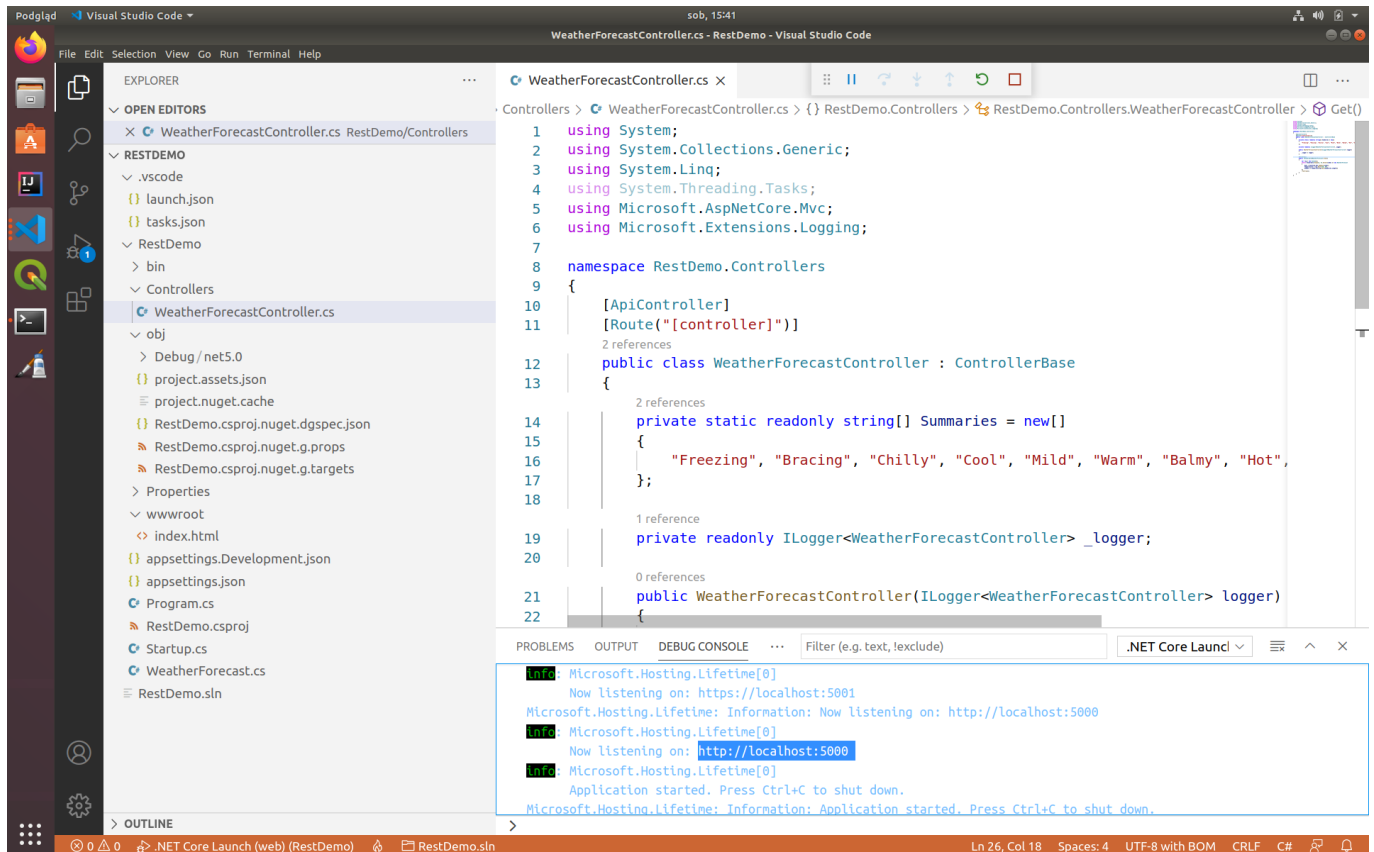
```
dotnet new webapi -n RestDemo
```

Open the `RestDemo` folder with Visual Studio code. Then you may run the application from the debug panel.

In case the necessary files to run and debug the application were not generated delete `.vscode` and all its contents and try running `.NET: Generate Assets for Build and Debug` from `View > Command Palette`.

Example

The `dotnet new webapi` command generates a sample project structure containing exemplar .NET Core Web API project.



A controller in .NET Core has a following structure

```

using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Logging;

namespace RestDemo.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class WeatherForecastController : ControllerBase
    {
        private static readonly string[] Summaries = new[]
        {
            "Freezing", "Bracing", "Chilly", "Cool", "Mild", "Warm", "Balmy",
            "Hot", "Sweltering", "Scorching"
        };
    };

```

```
private readonly ILogger<WeatherForecastController> _logger;

public WeatherForecastController(ILogger<WeatherForecastController>
logger)
{
    _logger = logger;
}

[HttpGet]
public ActionResult<IEnumerable<WeatherForecast>> Get()
{
    var rng = new Random();
    return Ok(Enumerable.Range(1, 5).Select(index => new WeatherForecast
    {
        Date = DateTime.Now.AddDays(index),
        TemperatureC = rng.Next(-20, 55),
        Summary = Summaries[rng.Next(Summaries.Length)]
    })
    .ToArray());
}
}
```

- `[ApiController]` attribute marks the class as being a controller
- `[Route("[controller]")]` specifies that the controller will be available under the URI address generated from the class name `WeatherForecastController` > `/weatherforecast`
- `[HttpGet]` attribute marks the method used to serve HTTP Get requests
- `Ok()` method wraps its argument in an HTTP response and sets the response HTTP status code to `200` (other examples include `NotFound()`, `Created()` or `BadRequest()`)

Further reading on REST APIs

[REST Services Maturity Model](#)

AJAX calls

Now, when we have established almost all the necessary elements to create our first web application, it is time to introduce something which will bring all the following things together:

- HTML and CSS for creating the interface
- JavaScript for the application behavior
- REST APIs for providing access to the data and computational resources of the server

In order to tie the frontend together with the backend we need a way to send data between them. For that we need AJAX (the name comes from Asynchronous JavaScript and XML), which provides the ability to make HTTP requests in the background (without blocking the browser interface).

Although the name comes from XML any type of data can be sent with those requests and the most popular format today is JSON as it is most natural to be processed with JavaScript (as JSON stands for:

JavaScript Object Notation)

First let's introduce the example of Fetch API, which provides an interface to make the AJAX calls

```
let call = fetch('https://nominatim.openstreetmap.org/search' +
  '?street=1 Plac Politechniki' +
  '&city=Warsaw' +
  '&country=Poland' +
  '&postalcode=00-661' +
  '&format=json')
;
call
  .then(response => {
    if (response.ok) {
      return response.json();
    } else {
      //handle bad HTTP status
    }
  })
  .then(showLocation);

function showLocation(data) {
  if (data.length > 0) {
    let lat = data[0]["lat"];
    let lng = data[0]["lon"];
    console.info(lat);
    console.info(lng);
    let marker = L.marker([lat, lng]);
    marker.addTo(map);
  } else {
    //handle no data
  }
}
```

This simple example shows us the following elements:

- the first argument of `fetch()` function is the URL to send request to
- `fetch()` returns promise (a future object)
- this first promise exposes the header of HTTP response when ready
- from the first promise we can get the content by calling `.text()` or `.json()` method
- we finally handle the second promise with the actual data returned from Nominatim API (which does geocoding on the basis of Open Street Map data)

Slightly more complex example would be the case when we would like to send the obtained `lat` and `lng` data to our custom REST endpoint `/location`.

```
fetch('/location',{
  method: 'POST',
  body: JSON.stringify({lat: lat, lng: lng}),
  headers: {
```

```

        'Content-Type': 'application/json'
    },
    //this will come in handy in case you need to send cookies
    credentials: 'include',
    //this will come in handy when you need to deal with cross-origin requests
    mode: 'cors'
  })
});

```

Single Page Application example

Let's create a complete SPA application with the usage of Java Micronaut framework. We start with creating the app from the command line with:

```
mn create-app POIManager
```

Then we set up the static resources and download the Leaflet library as discussed before and create the following HTML, CSS and JavaScript files.

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta charset="UTF-8">
  <title>Point of interest manager</title>
  <link rel="stylesheet" href="libs/leaflet.1.7.1/leaflet.css" />
  <link rel="stylesheet" href="css/style.css" />
  <script src="libs/leaflet.1.7.1/leaflet.js"></script>
  <script src="scripts/script.js"></script>
</head>
<body>
<div class="general-grid">
  <div id="map-panel">
  </div>
  <div id="new-poi-panel">
    <form id="new-poi-form">
      <div id="new-poi-form--container" class="form-grid">
        <div class="no-such-place-error-inactive grid-item-two-column">No such
place</div>
        <label for="input-country">Country</label><input id="input-country"
name="country" placeholder="Poland" />
        <label for="input-city">City</label><input id="input-city"
name="city" placeholder="Warszawa" />
        <label for="input-zipcode">Zipcode</label><input id="input-zipcode"
name="zipcode" placeholder="00-661"/>
        <label for="input-street">Street</label><input id="input-street"
name="street" placeholder="Plac Politechniki" />
        <label for="input-house">House number</label><input id="input-house"

```

```
name="house" placeholder="1" />
    <button class="grid-item-two-column">Place POI</button>
  </div>
</form>
</div>
</div>
</body>
</html>
```

```
.general-grid {
  display: grid;
  grid-template-rows: 1fr fit-content(10ch);
  row-gap: 1em;
  position: absolute;
  top: 0;
  bottom: 1em;
  left: 0;
  right: 0;
}

.form-grid {
  display: grid;
  grid-template-columns: fit-content(10em) fit-content(5em);
  row-gap: 0.5em;
  column-gap: 0.5em;
  justify-content: center;
  align-items: center;
}

.grid-item-two-column {
  grid-column: 1 / span 2;
}

.no-such-place-error-inactive {
  display: none;
}

.no-such-place-error-active {
  border: 1px solid darkred;
  background-color: lightpink;
  text-align: center;
}
```

```
const poiFormId = 'new-poi-form';
const mapContainerId = 'map-panel';
let map;

document.addEventListener('DOMContentLoaded', initializeApplication, false);
```

```
function initializeApplication() {
  let form = document.getElementById(poiFormId);
  form.onsubmit = handleNewPOI;
  loadMap();
}

function handleNewPOI() {
  let form = this;

  clearErrors();
  let response = fetch('https://nominatim.openstreetmap.org/search' +
    '?street=' + this.house.value.trim() + ' ' + this.street.value.trim() +
    '&city=' + this.city.value.trim() +
    '&country=' + this.country.value.trim() +
    '&postalcode=' + this.zipcode.value.trim() +
    '&format=json');
  ;
  response
    .then(r => {
      if (r.ok) {
        return r.json();
      }
    })
    .then(showLocation);
  return false;

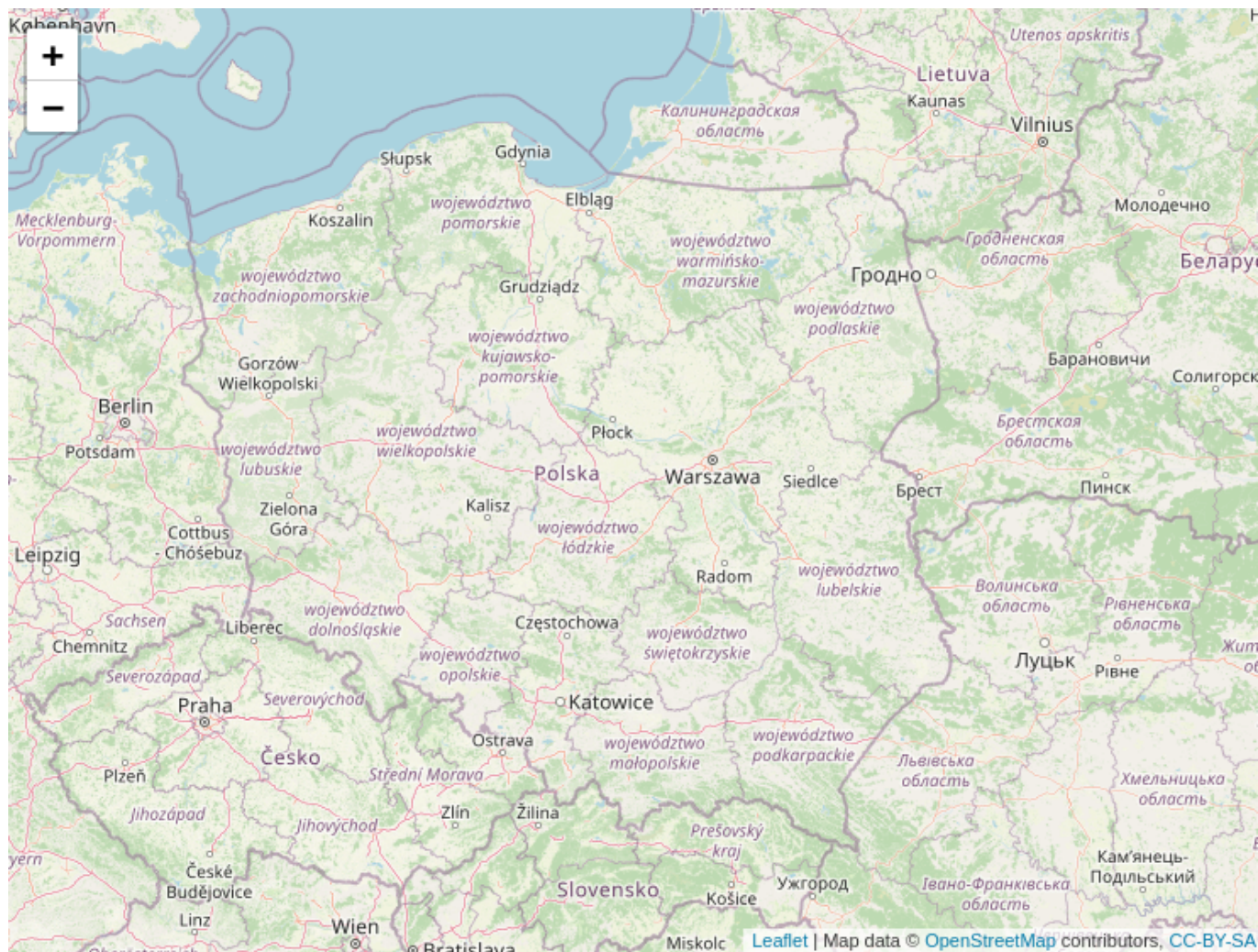
  function clearErrors() {
    let errorElements = form.getElementsByClassName('no-such-place-error-
  active');
    for (let i = 0; i < errorElements.length; ++i) {
      let errorElement = errorElements[i];
      errorElement.classList.add('no-such-place-error-inactive');
      errorElement.classList.remove('no-such-place-error-active');
    }
  }

  function showLocation(data) {
    if (data.length > 0) {
      let lat = data[0]["lat"];
      let lng = data[0]["lon"];
      let marker = L.marker([lat, lng]);
      marker.addTo(map);
    } else {
      let errorElements = form.getElementsByClassName('no-such-place-error-
  inactive');
      for (let i = 0; i < errorElements.length; ++i) {
        let errorElement = errorElements[i];
        errorElement.classList.add('no-such-place-error-active');
        errorElement.classList.remove('no-such-place-error-inactive');
      }
    }
  }
}
}
```



```
function loadMap() {
  //add map and set its center to specific location
  map = L.map(mapContainerId).setView([52.05, 20.00], 6);
  //add OSM as background layer
  L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png ', {
    attribution: 'Map data &copy; <a href="https://www.openstreetmap.org/">' +
      'OpenStreetMap</a> contributors, ' +
      '<a href="https://creativecommons.org/licenses/by-sa/2.0/">CC-BY-SA</a>',
    maxZoom: 19,
    id: 'osm.tiles'
  }).addTo(map);
}
```

Which results with the following interface:



Country	<input type="text" value="Poland"/>
City	<input type="text" value="Warszawa"/>
Zipcode	<input type="text" value="00-661"/>
Street	<input type="text" value="Plac Politechniki"/>
House number	<input type="text" value="1"/>
<input type="button" value="Place POI"/>	

Our goal is to create application for storing and presenting Points of Interest (POI).

Therefore we will need to provide them from the server through the controller

```
package POIManager.controller;  
  
import POIManager.dto.POI;  
import POIManager.service.POIService;  
import io.micronaut.http.HttpResponse;  
import io.micronaut.http.annotation.Controller;  
import io.micronaut.http.annotation.Get;  
import io.micronaut.http.annotation.Post;
```

```

import java.util.List;

@Controller("/poi")
public class POIController {
    private final POIService poiService;

    public POIController(POIService poiService) {
        this.poiService = poiService;
    }

    @Get
    HttpResponseMessage<List<POI>> getPOIs() {
        return HttpResponseMessage.ok(this.poiService.getPOIs());
    }

    @Post
    HttpResponseMessage<POI> createPoi(POI poi) {
        POI createdPoi = this.poiService.addPoi(poi);
        if (createdPoi != null) {
            return HttpResponseMessage.ok(createdPoi);
        } else {
            return HttpResponseMessage.unprocessableEntity();
        }
    }
}

```

Additionally JavaScript `loadMap()` function code needs to be extended with `fetch` call and `placeMarker(datum)` function in order to present POIs downloaded from the server

```

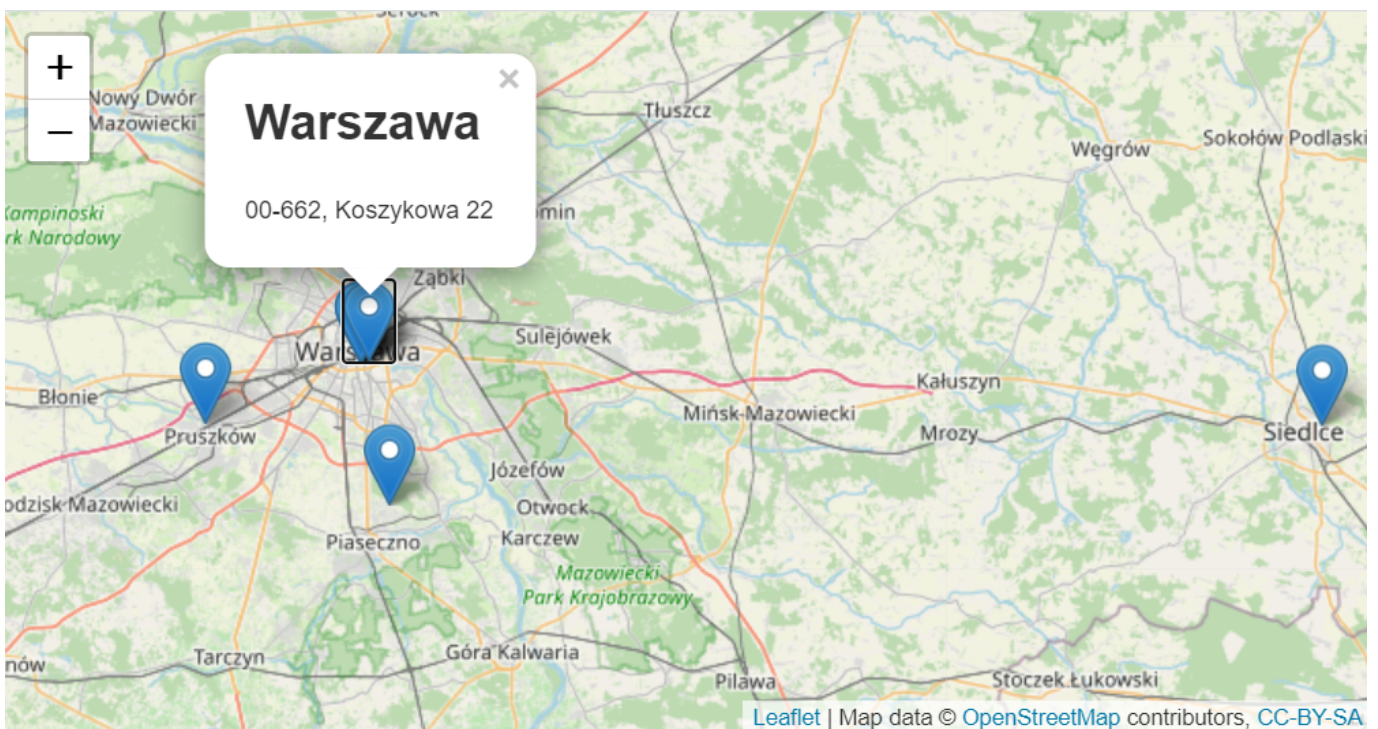
function loadMap() {
    //add map and set its center to specific location
    map = L.map('map-panel').setView([52.05, 20.00], 6);
    //add OSM as background layer
    L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png ', {
        attribution: 'Map data &copy; <a href="https://www.openstreetmap.org/">' +
            'OpenStreetMap</a> contributors, ' +
            '<a href="https://creativecommons.org/licenses/by-sa/2.0/">CC-BY-SA</a>',
        maxZoom: 19,
        id: 'osm.tiles'
    }).addTo(map);

    fetch('/poi')
        .then(r => r.json())
        .then(function (data) {
            for (let i = 0; i < data.length; ++i) {
                placeMarker(data[i]);
            }
        })
}

```

```
function placeMarker(datum) {
  let lat = datum["lat"];
  let lng = datum["lon"];
  let marker = L.marker([lat, lng]);
  marker.addTo(map);
  let message = '<h1>' + datum['city'] + '</h1>'
    + '<p>' + (datum['zipcode'] != null ? datum['zipcode'] : '00-000') +
    (datum['street'] != null ? (',' + datum['street']) : '') +
    (datum['house'] != null ? (' ' + datum['house']) : '') + '</p>';
  marker.bindPopup(message);
  console.info(message);
}
```

The result would be as follows:



Country	<input type="text" value="Polska"/>
City	<input type="text" value="Warszawa"/>
Zipcode	<input type="text" value="00-662"/>
Street	<input type="text" value="Koszykowa"/>
House number	<input type="text" value="22"/>
<input type="button" value="Place POI"/>	

As we are going to store data on the server we will set up a database for the application:

We assume that we already have **PostgreSQL** and **PostGIS** installed.

- create a user in the system
- create a role in DB for this user (passwords need to match)
- create a database for this role

- add schema to store spatial extension
- install spatial extension to schema

```
sudo adduser html2postgis
sudo -u postgres psql -c 'CREATE ROLE html2postgis WITH LOGIN CREATEDB ENCRYPTED
PASSWORD `webapp`;'
sudo -u postgres psql -c 'CREATE DATABASE html2postgis OWNER html2postgis;'
sudo -u html2postgis psql -d html2postgis -c 'CREATE SCHEMA postgis;'
sudo -u postgres psql -d html2postgis -c 'CREATE EXTENSION postgis SCHEMA
postgis;'
```

And we will also run the following query in order to initialize storage for our POIs:

```
CREATE TABLE IF NOT EXISTS public.poi (
  id serial NOT NULL PRIMARY KEY,
  country varchar NOT NULL,
  city varchar NOT NULL,
  zipcode varchar NOT NULL,
  street varchar NOT NULL,
  house varchar NOT NULL,
  geom geometry(Point,4326) NOT NULL
);
```

Additionally `build.gradle` of the micronaut project must be extended to include libraries for handling PostgreSQL connection.

```
dependencies {
  compile group: 'org.postgresql', name: 'postgresql', version: '42.2.14'
```

In the provided [example-05-spa-map-application](#) necessary creation of the table happens during the start of application.

Let's go back to the Java part of our micronaut project. Apart from the `POIController` we will need a service providing the objects from the database and Data Transfer Object representing POI.

The structure and the crucial part of this service can be observed within in the project structure:

```

13 import java.util.ArrayList;
14 import java.util.List;
15
16 @Singleton
17 @DefaultImplementation
18 public class POIDBService implements POIService {
19     public static final String CREATE_DB_SQL = "CREATE TABLE IF NOT EXISTS public.poi (" +
20         "    id serial NOT NULL PRIMARY KEY," +
21         "    country varchar NOT NULL," +
22         "    city varchar NOT NULL," +
23         "    zipcode varchar NOT NULL," +
24         "    street varchar NOT NULL," +
25         "    house varchar NOT NULL," +
26         "    geom geometry(Point,4326) NOT NULL" +
27         ");";
28
29     public static final String INSERT_POI_SQL = "INSERT INTO public.poi (country, city, zipcode, street, house, geom)" +
30         " VALUES (?, ?, ?, ?, ?, ST_SetSRID(ST_Point(?, ?), 4326)) RETURNING id;";
31
32     public static final String SELECT_POI_SQL = "SELECT id, country, city, zipcode, street, house," +
33         " ST_X(geom) as lon, ST_Y(geom) as lat FROM public.poi;";
34
35     private final DBConnector connector;
36
37     public POIDBService(DBConnector connector) {
38         this.connector = connector;
39     }
40
41     @PostConstruct
42     public void initializeDB() {
43         try {
44             Connection conn = connector.getConnection();
45             PreparedStatement ps = conn.prepareStatement(CREATE_DB_SQL);
46             ps.execute();
47         } catch (SQLException ex) {
48             ex.printStackTrace(System.err);
49         }
50     }
51
52     @Override
53     public List<POI> getPOIs() {
54         List<POI> pois = new ArrayList<>();
55         try {
56             Connection conn = connector.getConnection();
57             PreparedStatement ps = conn.prepareStatement(SELECT_POI_SQL);
58             ResultSet rs = ps.executeQuery();
59         } catch (SQLException ex) {
60             ex.printStackTrace(System.err);
61         }
62     }
63 }

```

Please note, that this service is marked as `@Singleton` and `@DefaultImplementation` of the `POIService` interface. Hence during the initialization of the controller, an object of `POIDBService` would be passed to the constructor of `POIController`.

Having the `POIController` supported by the `poiService.addPoi(poi)` method, we can use HTTP Post method to send new data to the server from JavaScript application:

```

let response = fetch('https://nominatim.openstreetmap.org/search' +
    '?street=' + poi.house + ' ' + poi.street +
    '&city=' + poi.city +
    '&country=' + poi.country +
    '&postalcode=' + poi.zipcode +
    '&format=json');
let jsonResponse = response
    .then(r => {
        if (r.ok) {
            return r.json();
        }
    });
jsonResponse.then(getPostPOIfunction(poi));

function getPostPOIfunction(poi) {
    return function (data) {
        if (data.length > 0) {
            poi.lat = data[0]["lat"];
            poi.lon = data[0]["lon"];
            fetch('/poi', {
                method: 'POST',
                body: JSON.stringify(poi),

```

```
        headers: {
            'Content-Type': 'application/json'
        }
    }).then(r => r.json())
        .then(placeMarker);
    }
};
}
```

In the example above, first the nominatim service is called to geocode the given address and if the geocoding returns at least one result, the first one is taken as the correct one. Then we extend `poi` definition with latitude and longitude and POST them to the `/poi` endpoint, hence `POIController.createPoi(POI poi)` method.

For the details of JavaScript and Java structure of this application solution please analyze the sources of the provided [example-05-spa-map-application](#)