**11. Projekt indywidualny - tydzień 3 z 4**

**Wyzwania:**

* Przypomnisz sobie informacje dot. formularzy.
* Lepiej poznasz rodzaje kontrolek oraz ich atrybuty.
* Dowiesz się jak zaimplementować walidację formularza.

**Wstęp**

Jesteśmy już w 3 tygodniu Twojego piewszego większego projektu. Przyszedł już czas na poruszenie tematu, bez którego aplikacje internetowe nie miałyby sensu — **formularzy**. I znowu temat może wydawać się dość prosty, bo nie jest problemem zrobienie kilku pól. Ale kiedy chcemy zrobić coś bardziej zaawansowanego, to niezbędna okazuje się wiedza na temat działania kontrolek formularzy.

W tym module omówimy sobie większość kontrolek formularzy, ich właściwości, oraz jak je stylować. Dodatkowo zobaczysz, na czym polega walidacja formularzy, i w jaki sposób możemy ją przeprowadzić.

**11.1. Wrapper formularza**

Zacznijmy od podstaw, czyli budowy wrappera <form></form>, którego atrybuty będą wpływały na działanie naszego formularza.

**Budowa formularza**

Każdy formularz składa się z dwóch głównych elementów — wrappera i kontrolek.

**Wrapper formularza** jest tagiem <form>, w którego wnętrzu znajdują się pola formularza. Jego użycie pozwala nam oddzielić dane pomiędzy innymi formularzami znajdującymi się na stronie. W ten sposób zabezpieczamy sytuację, gdy mamy tak samo brzmiące pola w kilku formularzach, ale wysłać do serwera chcemy tylko jeden z nich.

**Kontrolki formularza** to wszelkiego rodzaju pola i przyciski, dzięki którym jesteśmy w stanie wprowadzić dane do formularza. W następnym submodule zostaną zaprezentowane najważniejsze z nich. Warto też zaznaczyć, że kontrolki mogą pełnić również funkcję zatwierdzającą (przycisk do wysłania).

Wewnątrz formularza warto też umieszczać inne elementy, np. divy, które pozwolą nam na grupowanie pól z opisami i ułatwią późniejsze stylowanie.

Poniżej znajduje się przykład formularza kontaktowego.

<form action="" method="POST">

<div class="field">

<label>Imię i nazwisko</label>

<input type="text" name="user-name">

</div>

<div class="field">

<label>Wiadomość</label>

<textarea name="message"></textarea>

</div>

<button type="submit">Wyślij</button>

</form>

Jak zauważysz w powyższym przykładzie, formularz rozpoczyna się od wrappera zawierającego dwa atrybuty: action i method.

Pierwszy z nich mówi, **GDZIE** mają zostać przekazane dane z formularza. Może to być adres URL skryptu, który przetworzy zapytanie. Pozostawiając je puste zakładamy, że dane mają zostać przekazane pod adres, na którym aktualnie się znajdujemy.

Drugi parametr — method — określa, **W JAKI SPOSÓB** dane zostaną przekazane. Dysponujemy dwiema metodami, GET i POST, które za chwilę wyjaśnimy.

Kolejną rzeczą, na którą należy zwrócić uwagę, jest to, że każde **pole** zawiera atrybut name, dzięki, któremu będzie można zidentyfikować, z jakiego pola zostały przekazane dane.

I ostatnim elementem jest przycisk, za pomocą którego wyślemy dane z formularza.

**Działanie formularzy**

Na razie zajmujemy się jedynie wyglądem formularza. Jego pełne działanie będzie wymagało stworzenia backendu — w najprostszej formie byłby to np. skrypt na serwerze, który będzie potrafił odebrać dane wysłane za pomocą formularza i wysłać maila. Nie jest to jednak elementem tego projektu, i dlatego poniżej tylko pokrótce opisujemy jak argumenty tagu <form> wpływają na wysyłkę formularza.

**Formularz kontaktowy na GitHub Pages**

Nasz projekt będziemy publikować na GitHub Pages, który nie pozwala na zamieszczanie skryptów działających po stronie serwera. Jeśli jednak, w innych projektach w swoim portfolio, zechcesz umieścić działający formularz kontaktowy, możesz skorzystać z [Formspree](https://formspree.io/" \t "_blank) — darmowej platformy, która pozwala na wysyłanie maili na podstawie wypełnionego formularza.

Samo działanie większości formularzy wygląda następująco: użytkownik wypełnia formularz danymi, a następnie zatwierdza go klikając przycisk typu submit. W przypadku, jeśli ustawimy polom atrybuty związane z walidacją (która domyślnie w HTML5 jest włączona), zostanie uruchomiony proces sprawdzania poprawności wypełnionych pól. Gdy formularz pomyślnie przejdzie walidację, zostanie wywołana akcja przekazania danych, z wykorzystaniem zadeklarowanej metody.

I tutaj warto się zatrzymać, aby zrozumieć, jakie mamy metody wysyłania formularzy.

**GET**

Metoda GET polega na przekazaniu danych poprzez adres URL. Formularz przejdzie do adresu zadeklarowanego w polu action, doklejając do niego pola formularza razem z zawartością. Np. https://kodilla.com/formularz?user-name=Michal&message=Hej%20przyjaciele

**POST**

Metoda POST odwoła się bezpośrednio do adresu z action, z tym, że dane przekaże w niewidocznym dla użytkownika nagłówku zapytania. Ta metoda jest z reguły bezpieczniejsza.

Więcej na temat przetwarzania zapytań, w tym tych od formularzy, dowiesz się w następnych modułach.

Jeśli masz wątpliwości do powyższego materiału, to - zanim zatwierdzisz - zapytaj na czacie :)

Zapoznałe(a)m się!

**11.2. Typy kontrolek**

Czas poznać elementy, którymi dysponujemy przy tworzeniu formularzy. W tym submodule omówimy tylko te najczęściej wykorzystywane. Pełną listę elementów możesz znaleźć w [dokumentacji MDN](https://developer.mozilla.org/en-US/docs/Web/HTML/Element#Forms).

**Input — pole tekstowe**

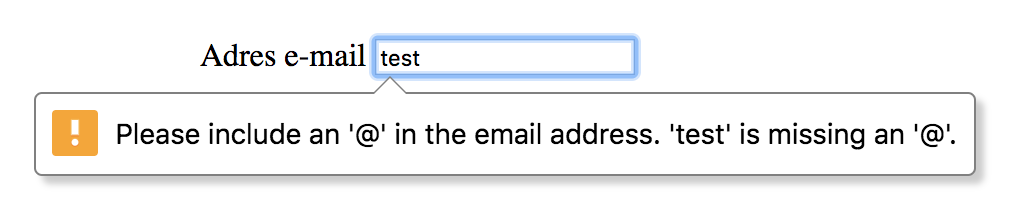
Pole, które zawiera każdy formularz. Służy ono do wprowadzania zazwyczaj krótkiej, jednolinijkowej treści.

[](https://uploads.kodilla.com/bootcamp/wdp/wdp11/input-text.png)

<input type="text" name="user-name">

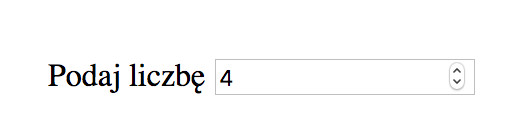
Zauważ, że jako typ podaliśmy text, co oznacza, że pole jest tekstowym. Jednak nic nie szkodzi na przeszkodzie, żeby zrobić z niego pola typu:

email — gdzie już dodamy od razu regułę walidacji, wymagającą podania prawidłowego adresu email, gdy pole posiada też atrybut required.

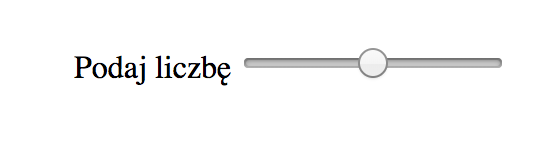
[](https://uploads.kodilla.com/bootcamp/wdp/wdp11/input-email.png)

password — pole z ukrywaną zawartością. Wszystko, co wprowadzimy, będzie ukryte za gwiazdkami/kropkami.

number — w którym będziemy mogli wprowadzać wartości liczbowe. Charakteryzują go również strzałki po prawej stronie, służące do zwiększania lub zmniejszania wartości. Dodając atrybuty min i max możemy określić zakres wprowadzanej liczby, a stosując atrybut step zdefiniujemy wielkość pojedynczego przeskoku, np. jeśli chcemy za pomocą przycisków zmieniać wartość w polu o 4.

[](https://uploads.kodilla.com/bootcamp/wdp/wdp11/input-number.png)

range — stworzy nam kontrolkę suwaka, która działa podobnie jak pole number. Aby określić zakres wartości oraz skok, również musimy skorzystać z atrybutów min, max i step.

[](https://uploads.kodilla.com/bootcamp/wdp/wdp11/input-range.png)

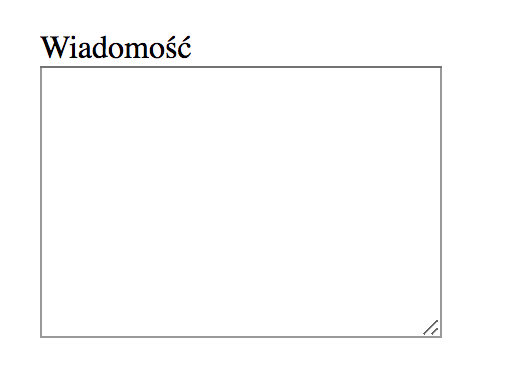
To tylko kilka przykładów — pełną listę typów inputów znajdziesz w [dokumentacji MDN](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input#Form_%3Cinput%3E_types).

**UWAGA!** W przypadku tych pól możemy ustawić atrybut value, który ustawi domyślną wartość dla tego pola.

**Textarea — tekst wielolinijkowy**

Pole, które umożliwia wprowadzenie długiego tekstu, np. treści wiadomości. Nie zawiera ono żadnych typów — jest tylko tekstowe.

<textarea name="message"></textarea>

[](https://uploads.kodilla.com/bootcamp/wdp/wdp11/textarea.png)

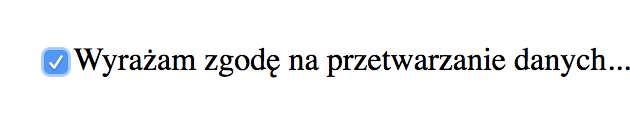
**UWAGA!** W przypadku tego pola, domyślną zawartość ustawiamy poprzez wprowadzenie jej między klamry znacznika.

<textarea name="message">Moja wiadomość</textarea>

**Checkbox**

Klasyczne pole wielokrotnego wyboru. Możemy je zaznaczyć i odznaczyć. Może istnieć samodzielnie lub jako grupa (w tym przypadku muszą mieć ten sam name, ale różne value).

<input type="checkbox" name="zgoda">

[](https://uploads.kodilla.com/bootcamp/wdp/wdp11/input-checkbox.png)

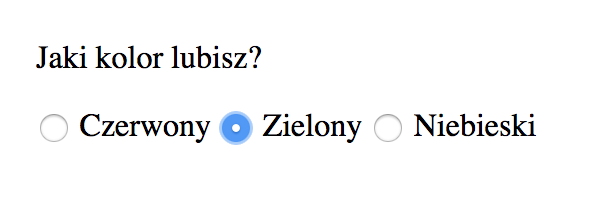
**Radio button**

Pole jednokrotnego wyboru występujące w grupie o tym samym name. Co istotne, raz kliknięta opcja nie może być odkliknięta, a jedynie zmieniona na inną. Różnicowanie polega na użyciu różnego value.

<input type="radio" name="kolor" value="red">

<input type="radio" name="kolor" value="green">

<input type="radio" name="kolor" value="blue">

[](https://uploads.kodilla.com/bootcamp/wdp/wdp11/input-radio.png)

**Select**

Rozwijana lista, umożliwiająca wybranie domyślnie jednego elementu. Tworzymy kontrolkę <select></select>, w którą wpisujemy kolejne opcje wyboru <option></option>. Domyślnie zaznaczona jest pierwsza opcja.

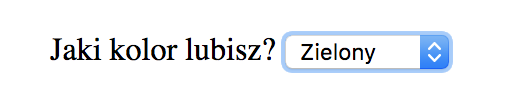
<select name="kolor">

<option value="red">Czerwony</option>

<option value="green">Zielony</option>

<option value="blue">Niebieski</option>

</select>

[](https://uploads.kodilla.com/bootcamp/wdp/wdp11/select.png)

Istnieje dodatkowo wrapper, który umożliwia nam wizualne pogrupowanie opcji. Wystarczy owinąć je elementem <optgroup></optgroup>, gdzie w atrybucie label podamy opis grupy.

<select name="car">

<optgroup label="Szwedzkie">

<option value="volvo">Volvo</option>

<option value="saab">Saab</option>

</optgroup>

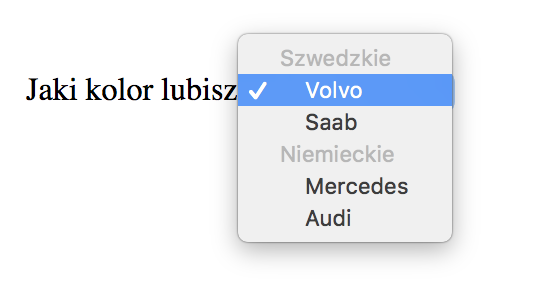
<optgroup label="Niemieckie">

<option value="mercedes">Mercedes</option>

<option value="audi">Audi</option>

</optgroup>

</select>

[](https://uploads.kodilla.com/bootcamp/wdp/wdp11/select-optgroup.png)

**Pole wyboru pliku**

Umożliwia nam wybranie pliku, którzy ma zostać wysłany do serwera.

**UWAGA!** Przekazywanie pliku możliwe jest wyłączenie za pomocą metody POST oraz wymaga dodania do formularza atrybutu enctype="multipart/form-data".

<form method="post" enctype="multipart/form-data">

<div>

<label>Wybierz plik do wgrania</label>

<input type="file" name="file">

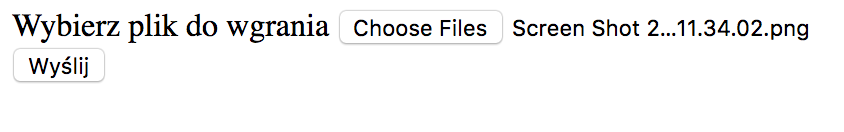
</div>

<div>

<button>Wyślij</button>

</div>

</form>

[](https://uploads.kodilla.com/bootcamp/wdp/wdp11/input-file.png)

Dodatkowo za pomocą atrybutu accept możemy podać akceptowane rozszerzenia pliku lub jego typ (MIME type).

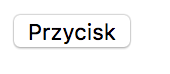
<input type="file" accept=".png, .jpeg">

<input type="file" accept="image/png, image/jpeg">

**Button**

Aby zatwierdzić wysłanie formularza, potrzebny nam jest przycisk z etykietą. Domyślnie przycisk w formularzu "submituje" go, jednak przycisków możemy używać dowolnie. Najważniejsze jest określenie jego typu (type).

<button type="submit">Przycisk</button>

[](https://uploads.kodilla.com/bootcamp/wdp/wdp11/button.png)

Jednak w identyczny sposób możemy stworzyć przycisk wykorzystując do tego <input> z typem submit. Etykietą przycisku będzie jego value.

<input type="submit" value="Przycisk">

[](https://uploads.kodilla.com/bootcamp/wdp/wdp11/input-submit.png)

**Różnica pomiędzy buttonem a inputem typu submit**

Jeśli potrzebujemy więcej elastyczności i możliwości kontroli zawartości przycisku, np. chcemy dodać ikonkę do niego, to lepiej skorzystać z opcji <button></button>.

Z drugiej jednak strony, jeśli chcemy mieć możliwość przekazania dodatkowej wartości po kliknięciu przycisku, to możemy skorzystać z opcji z <input>. Przykładem sytuacji może być formularz, w którym określamy dwie różne akcje — zapis lub usunięcie obiektu. Możemy do tego formularza dodać dwa przyciski:

<input type="submit" name="akcja" value="zapisz">

<input type="submit" name="akcja" value="usun">

W zależności od tego, w który przycisk klikniemy, taką wartość przyjmie pole o nazwie akcja.

**Ukryte pole**

Jeśli chcemy w formularzu przekazać ukryte dane, które mają się nie pojawiać użytkownikowi, wtedy możemy skorzystać z pola typu hidden.

<input type="hidden" name="nazwa\_pola" value="1">

**Label**

Jest to etykieta dla pola, która je opisuje. Często służy jako dodatkowy element interfejsu do zaznaczenia pola.

<label>Wybierz plik do wgrania</label>

Umieszaczając w jego wnętrzu jakieś pole, np. checkbox czy radio, sprawimy, że po jego kliknięciu pole nam się zaznaczy, a przypadku pól tekstowych stanie się aktywne (:focus).

<label>

<input type="checkbox" name="zgoda">

Wyrażam zgodę na przetwarzanie danych osobowych...

</label>

Jeśli jednak chcemy mieć wydzielone pole od etykiety, wtedy możemy je powiązać z pomocą atrybutu for i podać id pola, z którym ma być powiązane.

<input type="checkbox" name="zgoda" id="poleZgody">

<label for="poleZgody">Wyrażam zgodę na przetwarzanie danych...</label>

Jeśli masz wątpliwości do powyższego materiału, to - zanim zatwierdzisz - zapytaj na czacie :)

Zapoznałe(a)m się!

**11.3. Właściwości kontrolek**

Poza wspomnianymi atrybutami warto również zapoznać się z listą najpopularniejszych atrybutów stosowanych przy różnych typach pól.

| **Atrybut** | **Obsługiwane pola** | **Działanie** |
| --- | --- | --- |
| name | wszystkie pola | Każde pole musi zawierać nazwę, abyśmy mogli identyfikować jego zawartość. Jego brak skutkować będzie brakiem wysłania jego wartości. |
| value | wszystkie pola z wyjątkiem textarea | Ustawia wartość pola |
| selected | option | Określa,czy opcja jest wybrana |
| checked | checkbox, radio | Określa, czy checkbox/radio jest zaznaczony |
| maxlenght | textarea i pola tekstowe | Ogranicza możliwą liczbę wprowadzonych znaków |
| autocomplete | textarea i pola tekstowe | Określa, czy można uzupełnić pole wcześniej wprowadzonymi wartościami, jeżeli takowe są zapisane w przeglądarce |
| multiple | pole wyboru pliku, select | Określa, czy możliwe jest wybranie kilku opcji/plików |
| placeholder | pola tekstowe, textarea | Ustawia tekst pokazujący się, jeśli pole jest niewypełnione |
| required | wszystkie pola | Określa, czy wymagane jest wypełnienie pola przed wysłaniem formularza |
| readonly | wszystkie pola | Blokuje możliwość edycji pola |
| disabled | wszystkie pola | Blokuje możliwość edycji pola oraz jego wysyłki w formularzu |

**Różnica pomiędzy readonly a disabled**

Jeśli zależy Ci na pokazaniu pola użytkownikowi, ale nie chcesz dawać mu możliwości jego edycji, możesz wybrać jedną z dwóch opcji: readonly lub disabled. Jedno i drugie ma podobne działanie. Różnica polega na tym, że to, co ma readonly, zostanie wysłane jak normalne pole, a to, co jest disabled, zostanie pominięte.

Jeśli masz wątpliwości do powyższego materiału, to - zanim zatwierdzisz - zapytaj na czacie :)

Zapoznałe(a)m się!

**11.4. Stylowanie kontrolek**

Bardzo ważne jest odpowiednie dostosowanie wyglądu kontrolek do designu lub ogólnego stylu strony. Poniższe porady pozwolą Ci uniknąć wielu problemów, które napotykają początkujący Web Developerzy.

**Inputy tekstowe i textarea**

Najprostsze do ostylowania są z pewnością inputy tekstowe i textarea. Można nimi zarządzać jak każdym blokiem, ustawiać im kolor tła, kolor czcionki, zmieniać padding itp.

input[type="text"] {

padding: 5px 10px;

font-size: 14px;

background-color: #fafafa;

border: none;

color: #000000;

outline: none;

}

Warto zwracać uwagę, by przekazywać w CSS typ pola, które stylujemy, żeby uniknąć konfliktów z inputami różnych typów np. tekstu z checkboksem lub przyciskiem.

input[type="text"],

input[type="email"] {

...

}

Pewnie zastanawiasz się też, jak to się dzieje, że kiedy jesteś w polu i wprowadzasz tekst, to widać dookoła niebieską obwódkę. Jest to outline, który możesz wyłączyć, używając w CSS reguły outline: none. Pamiętaj jednak, że warto w jakiś sposób wyróżnić aktywne pole — możesz do tego wykorzystać style dla stanu kontrolki, które opisujemy poniżej.

Dodatkowo w przypadku textarea domyślnie mamy ustawioną możliwość zmiany wielkości pola. To ten mały trójkącik w prawym dolnym rogu pola, który wystarczy przeciągnąć, aby zmiana się dokonała. Jak zablokować tą możliwość? Dzięki regule resize: none.

**Checkboksy i radio buttony**

Stylowanie inputów tekstowych i textarea było rzeczą dość prostą, jednak stylowanie checkboksów i radio buttonów jest wyzwaniem. A to wszystko dlatego, że te kontrolki **nie reagują na stylowanie** z wyjątkiem pozycjonowania i widoczności.

Przychodzi zatem pytanie, w jaki sposób zrobić ładnie wyglądającą kontrolkę. Tutaj skorzystamy z wiedzy o stosowaniu labelów oraz stanów (:checked) pól typu checkbox czy radio.

Zwróć uwagę w powyższym przykładzie na te aspekty:

* nasz input[type="checkbox"] ukrywamy nadając mu opacity:0,
* pozycja absolutna sprawi, że input nałoży się na .icon,
* wymiary checkboksa muszą być takie same, jak wymiary .icon,
* wykorzystujemy box-sizing: border-box, aby szerokość obramowania wliczała się w width i height.

Dzięki temu podejściu, checkbox będzie przezroczysty i będzie przykrywał .icon. Jest to potrzebne dla systemów, w których kliknięcie w <label> nie zaznacza checkboksa.

W ten sposób stworzyliśmy działającą własną kontrolkę dla checkboksa. Analogicznie możemy zrobić to z radio buttonem. Pamiętaj tylko, że radio button najczęściej przyjmuje kształt koła, a checkbox kwadratu.

**Selecty**

Podobnie jak checkboksy i radio buttony, selecty również musimy stylować w nieco bardziej skomplikowany sposób. Dzieje się tak przede wszystkim dlatego, że przeglądarki narzucają im swoje style (każda przeglądarka używa własnych styli). Style te noszą nazwę webappearance i przed rozpoczęciem stylowania należy je wyłączyć.

select {

-webkit-appearance: none;

-moz-appearance: none;

appearance: none;

}

W ten sposób nie tylko usunęliśmy strzałki, które narzucała nam przeglądarka, ale również część styli. Dlatego powinniśmy zacząć właśnie od ich uzupełnienia:

select {

-webkit-appearance: none;

-moz-appearance: none;

appearance: none;

background-color: #ffffff;

border: 1px solid #000000;

padding: 5px 30px 5px 10px;

font-size: 14px;

display: block;

border-radius: 0;

}

Specjalnie zostawiliśmy po prawej stronie dodatkowy padding 20px, aby zmieścić tam strzałkę, którą dodamy za pomocą obrazku tła. Finalny efekt będzie następujący:

W ten sposób zrobiliśmy własne style dla selecta.

**Stany kontrolek**

Stylując kontrolki warto też pamiętać o ich wyglądzie dla różnych stanów. Im więcej stanów mamy obsłużonych dla każdej kontrolki, tym mniej może nas zaskoczyć przy implementacji i użytkowaniu formularza.

**Stany, które warto obsłużyć:**

:disabled i :readonly, kiedy pole jest zablokowane.

:focus, kiedy pole jest aktywne i wprowadzamy do niego treści.

:hover — nie jest wymagany, ale kiedy najedziemy na pole, za pomocą stanu :hovermożemy pokazać, że jest ono aktywne.

:checked, kiedy pole jest zaznaczone.

:valid lub :invalid, kiedy walidacja poprawności pola przeszła lub nie przeszła pomyślnie.

Jeśli masz wątpliwości do powyższego materiału, to - zanim zatwierdzisz - zapytaj na czacie :)

Zapoznałe(a)m się!

**11.5. Walidacja formularzy**

Skoro jesteśmy przy temacie formularzy, grzechem będzie nie wspomnieć o walidacji. Jeśli już chcemy przekazać do serwera dane, to warto przed wykonaniem jakiegokolwiek zapytania sprawdzić, czy przesłane dane są prawidłowe. Ten proces nazywamy właśnie walidacją formularza.

Walidację możemy przeprowadzić w dwojaki sposób — dzięki wbudowanym funkcjom HTML5 oraz wsparcia przeglądarki, lub poprzez napisanie własnego walidatora.

**Walidacja HTML5**

Każdy formularz i pole przyjmuje w każdym momencie jeden z dwóch stanów: :valid lub :invalid. W zależności od tego, w jaki sposób są wypełnione pola i czy ich zawartość jest zgodna z wymaganiami walidacji, taki stan otrzyma cały formularz.

W jaki sposób dodać wspomniane reguły? Za pomocą atrybutów w polach:

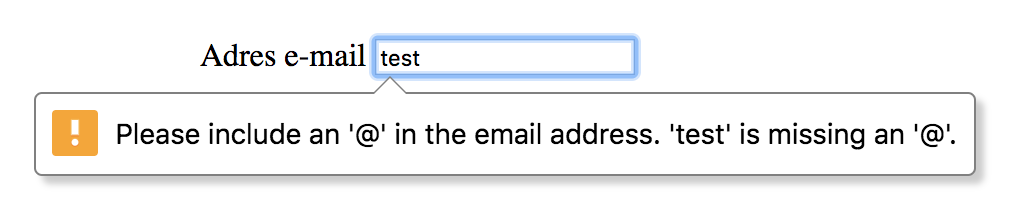
required — pojawił się już wcześniej wielokrotnie i wymaga, żeby pole nie było puste.

min i max — określają minialną i maksymalną wartość pola liczbowego.

minlength i maxlength — określają minialną i maksymalną długość wartości pola tekstowego.

pattern — określa reguły za pomocą wyrażeń regularnych (regex). Więcej na ten temat znajdziesz w [dokumentacji](https://developer.mozilla.org/en-US/docs/Learn/HTML/Forms/Form_validation#Validating_against_a_regular_expression)

Po ustawieniu takich atrybutów, po próbie zasubmitowania formularza w błędnych polach zobaczymy komunikaty walidacji.

[](https://uploads.kodilla.com/bootcamp/wdp/wdp11/input-email.png)

**Własne reguły walidacji w JS**

Jeśli jednak chcielibyśmy napisać własną walidację i sterować nią za pomocą JavaScriptu, należy zacząć od wyłączenia HTMLowej walidacji dodając do <form> atrybut novalidate.

<form novalidate>

<label for="mail">

<span>Podaj swój adres e-mail:</span>

<input type="email" id="mail" name="email\_address">

<span class="error"></span>

</label>

<button>Zapisz</button>

</form>

Następnie w JavaScripcie powinniśmy wykryć próbę submitowania.

document.querySelector('form').addEventListener('submit', function(event) {

...

})

A następnie określić, czy formularz powinien być zwalidowany, czy nie.

document.querySelector('form').addEventListener('submit', function(event) {

var isFormValidate = true;

// Tutaj napiszemy reguły walidacji

return !isFormValidate ? event.preventDefault() : true;

})

Spróbujmy teraz napisać walidację pola o nazwie email\_address:

document.querySelector('form').addEventListener('submit', function(event) {

var isFormValidate = true;

var emailAddressInput = event.target.querySelector('input[name="email\_address"]')

if(emailAddressInput.value.indexOf('@') < 0) {

isFormValidate = false;

emailAddressInput.parentElement.querySelector('.error').innerHTML = 'Błędny adres e-mail';

}

return !isFormValidate ? event.preventDefault() : true;

})

W ten sposób napisaliśmy prostą walidację formularza, którą możemy rozbudowywać.