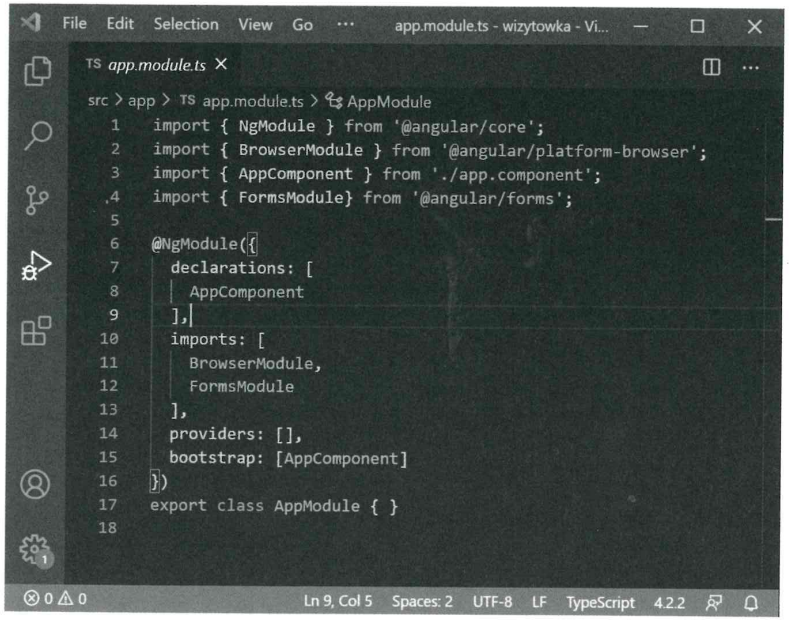
2.7. Zdarzenia i formularze

Angular zapewnia także obsługę i walidację formularzy.

2.7.1. Przygotowanie formularza

Aby móc rozpocząć pracę z formularzami, należy dodać *FormsModule*. Import modułu przeprowadzamy w pliku *app.module.ts* (rysunek 2.79).



Rysunek 2.79. Dodanie FormsModule

Następnym krokiem jest utworzenie formularza. Do jego budowy wykorzystujemy standardowe znaczniki języka HTML.

Kod formularza jest pokazany na listingu 2.99. Został umieszczony w szablonie widoku (plik *app.component.html*). Tworząc go, należy dopilnować, aby jego pola zostały nazwane, w przeciwnym razie wystąpią błędy kompilacji.

Listing 2.99. Kod HTML formularza

```
<form>
  <div class="formularz">
    <h1>Zamówienie:</h1>
    <label>
      <span>Imię i nazwisko:</span>
```



```
    </label>
    <label>
      <span>Email:</span>
      <input type="text" class="wpis" id="email" name="email">
    </label>
    <label>
      <span>Produkt:</span>
      <select id="produkt" class="produkt" name="produkt">
        <option value="pralka">pralka</option>
        <option value="telewizor">telewizor</option>
        <option value="lodowka">lodówka</option>
        <option value="laptop">laptop</option>
      </select>
    </label>
    <label>
      <span>Ilość:</span>
      <input type="number" class="wpis" id="ilosc" name="ilosc">
    </label>
    <label>
      <span>Wiadomość:</span>
      <textarea class="wiadomosc" id="wiadomosc" name="wiadomosc"></textarea>
      <button type="submit" class="przycisk">Zamów</button>
    </label>
  </div>
</form>
```

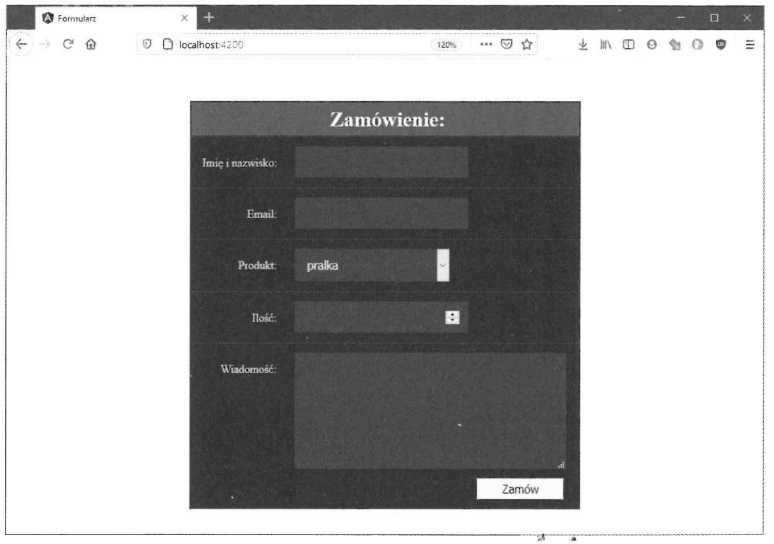
Formularz został sformatowany za pomocą arkusza stylów, pokazanego na listingu 2.100 (plik *app.component.css*).

Listing 2.100. Arkusz stylów formularza

```
* {
    margin: 0;
    padding: 0;
}

body {
    font: 100% normal Arial, Helvetica, sans-serif;
}

form, input, select, textarea {
    margin: 0;
    padding: 0;
    color: #ffffff;
}
```

```
div.formularz h1 {
    text-align: center;
    color: #ffffff;
    font-size: 24px;
    padding: 5px 0 5px 5px;
    border: 1px solid #161712;
}

div.formularz {
    margin: 0 auto;
    width: 450px;
    background: #3d3939;
    position: relative;
    top: 40px;
    border: 1px solid #262626;
}

div.formularz .wpis {
    padding: 10px 10px;
    width: 180px;
    background: #262626;
    border: 1px double #171717
}

div.formularz .produkt {
    padding: 10px 10px;
    width: 180px;
    background: #262626;
    border: 1px double #171717
}

div.formularz .wiadomosc {
    padding: 7px 7px;
    width: 300px;
    height: 120px;
    background: #262626;
    border: 1px double #171717;
    overflow: hidden;
}

div.formularz .przycisk {
    margin: 0 0 10px 0;
    padding: 4px 7px;
    position: relative;
    top: 10px;
    left: 332px;
    width: 100px;
}

div.formularz label {
    width: 100%;
    display: block;
    background: #1b1616;
    border-top: 1px solid #262626;
    border-bottom: 1px solid #161712;
    padding: 10px 0 10px 0;
}

div.formularz label span {
    display: block;
    color: #d0dbdf;
    font-size: 13px;
    float: left;
    width: 100px;
    text-align: right;
    padding: 12px 20px 0 0;
}
```


Wygląd i układ pól formularza jest pokazany na rysunku 2.80.



2.7.2. Implementacja formularza

Za zarządzanie formularzem odpowiada **dyrektywa ngForm**. Jej zastosowanie pozwoli uzyskać dostęp do wartości wpisanych do pól formularza. Aby zainicjować dyrektywę, przypisujemy ją do zmiennej szablonowej o dowolnej nazwie. Z użyciem tej nazwy będziemy się odwoływać do formularza. Linijka ze znacznikiem `<form>` przyjmuje postać `<form #formularz="ngForm">`.

Aby móc zacząć rejestrować pola formularza, inicjujemy je za pomocą **dyrektywy ngModel**. Przypisanie wartości spowoduje wyświetlenie jej w polu formularza.

Aby sprawdzić działanie formularza, definiujemy zdarzenie `click` na przycisku `Zamów`. Jego kliknięcie spowoduje wypisanie tekstu `Wszystko działa` w konsoli przeglądarki.

Kod po zmianach jest pokazany na listingu 2.101.

Listing 2.101. Implementacja formularza

```
<form #formularz="ngForm">
  <div class="formularz">
    <h1>Zamówienie:</h1>
    <label>
      <span>Imię i nazwisko:</span>
      <input type="text" class="wpis" id="imieinazwisko" ngModel
name="imieinazwisko">
    </label>
    <label>
      <span>Email:</span>
      <input type="text" class="wpis" id="email" ngModel="Proszę podać email"
name="email">
    </label>
    <label>
      <span>Produkt:</span>
      <select id="produkt" class="produkt" ngModel name="produkt">
        <option value="pralka">pralka</option>
        <option value="telewizor">telewizor</option>
        <option value="lodowka">lodówka</option>
        <option value="laptop">laptop</option>
      </select>
    </label>
    <label>
      <span>Ilość:</span>
      <input type="number" class="wpis" id="ilosc" ngModel name="ilosc">
    </label>
    <label>
      <span>Wiadomość:</span>
```

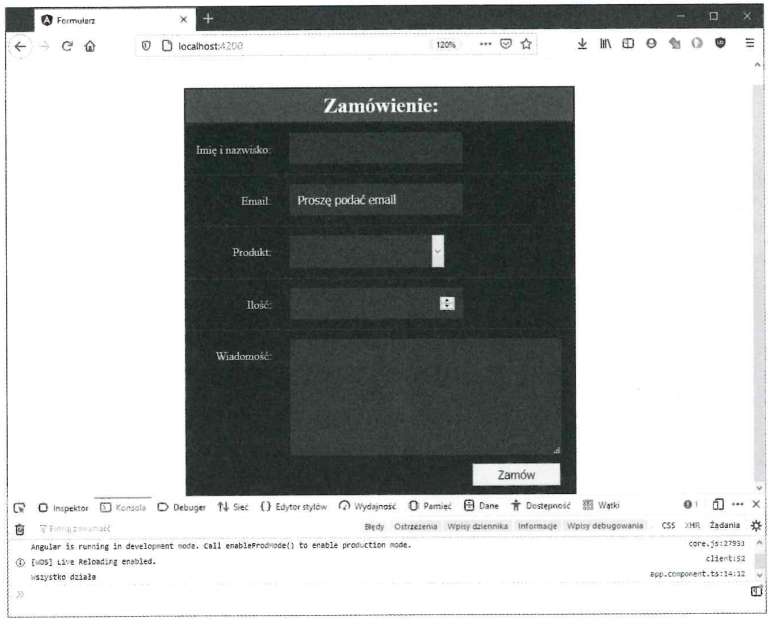


```
      <textarea class="wiadomosc" id="wiadomosc" ngModel name="wiadomosc"></
textarea>
      <button type="submit" class="przycisk" (click)="onSubmit()">Zamów</
button>
    </label>
  </div>
</form>
```

Aby formularz działał, w pliku *app.component.ts*. musi zostać zdefiniowana metoda `onSubmit`. Należy dodać kod:

```
onSubmit() {
   console.log("Wszystko działa");
}
```

Po tych wszystkich zmianach w polu *Email* zostaje wypisana informacja *Proszę podać email*, a w konsoli, po kliknięciu przycisku *Zamów* — zdefiniowany tekst. Formularz działa poprawnie (rysunek 2.81).



Rysunek 2.81. Implementacja formularza

Aby rozpocząć przechwytywanie wartości wpisanych w polach formularza, należy **skorzystać z wiązania zdarzeń**, które definiujemy w polu `input` formularza i pliku *app.component.ts* komponentu.

Kod formularza jest pokazany na listingu 2.102, a zawartość pliku *app.component.ts* — na

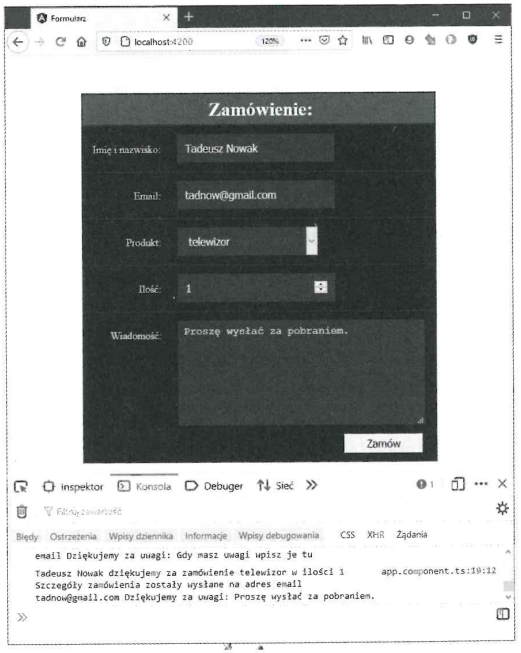
Listing 2.102. Rejestrowanie wartości wpisanych w polu formularza

```
<form #formularz="ngForm">
  <div class="formularz">
    <h1>Zamówienie:</h1>
    <label>
      <span>Imię i nazwisko:</span>
      <input type="text" class="wpis" id="imieinazwisko"
[(ngModel)]="imieinazwisko" name="imieinazwisko">
    </label>
    <label>
      <span>Email:</span>
      <input type="text" class="wpis" id="email" [(ngModel)]="email"
name="email">
    </label>
    <label>
      <span>Produkt:</span>
      <select id="produkt" class="produkt" [(ngModel)]="produkt"
name="produkt">
        <option value="pralka">pralka</option>
        <option value="telewizor">telewizor</option>
        <option value="lodowka">lodówka</option>
        <option value="laptop">laptop</option>
      </select>
    </label>
    <label>
      <span>Ilość:</span>
      <input type="number" class="wpis" id="ilosc" [(ngModel)]="ilosc"
name="ilosc">
    </label>
    <label>
      <span>Wiadomość:</span>
      <textarea class="wiadomosc" id="wiadomosc" [(ngModel)]="wiadomosc"
name="wiadomosc"></textarea>
      <button type="submit" class="przycisk" (click)="onSubmit()">Zamów</
button>
    </label>
  </div>
</form>
```


Listing 2.103. Obsługa formularza — plik app.component.ts

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'formularz';

  imieinazwisko: string = "";
  email: string = "Proszę podać email";
  produkt: string = "";
  ilosc: number = 0;
  wiadomosc: string = "Jeśli masz uwagi, wpisz je tu";

  onSubmit() {
    console.log(this.imieinazwisko, 'dziękujemy za zamówienie', this.produkt,
'w ilości', this.ilosc, 'Szczegóły zamówienia zostały wysłane na adres email',
this.email, 'Dziękujemy za uwagi:', this.wiadomosc);
  }
}
```

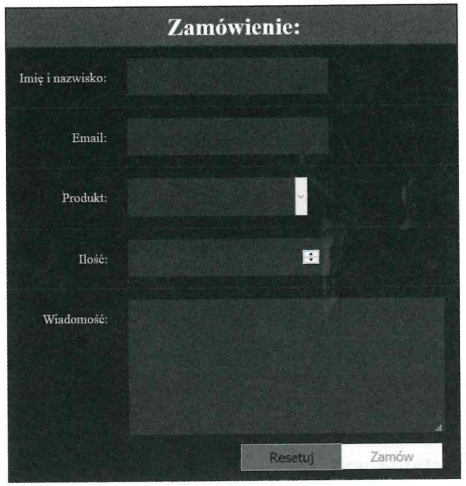
Po uzupełnieniu wszystkich pól formularza ich wartość zostanie wyświetlona w konsoli (rysunek 2.82).



Aby do formularza dodać funkcjonalność resetowania, należy skorzystać z metody `resetForm()`. W tym celu wstawiamy nowy przycisk i przypisujemy do niego metodę (nie zapominamy o aktualizacji arkusza stylów — właściwość `left` klasy `div.formularz .przycisk` zmieniamy z `332px` np. na `220px`). Przykładowy kod realizujący to zadanie mógłby wyglądać tak:

```
<button type="button" class="przycisk" style="background-color: red;"
(click)="formularz.resetForm()">Resetuj</button>
```

Uaktualnienie kodu spowoduje dodanie czerwonego przycisku *Resetuj*, którego kliknięcie poskutkuje wyczyszczeniem wszystkich pól formularza (rysunek 2.83).



Rysunek 2.83. Resetowanie pól formularza

2.7.3. Walidacja formularzy

Formularze tworzone w Angularze możemy walidować. To oznacza, że mamy możliwość sprawdzenia zawartości pola formularza. Kod formularza weryfikujący zawartość pól jest pokazany na listingu 2.104.

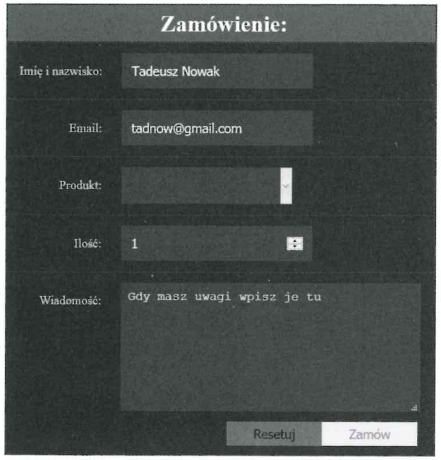
Listing 2.104. Walidacja formularzy

```
<form #formularz="ngForm">
  <div class="formularz">
    <h1>Zamówienie:</h1>
    <label>
      <span>Imię i nazwisko:</span>
      <input type="text" class="wpis" id="imieinazwisko"
[(ngModel)]="imieinazwisko" name="imieinazwisko" required
#imieinazwiskoForm="ngModel">
      <div class="alert" *ngIf="imieinazwiskoForm.invalid && imieinazwiskoForm.
touched">Pole imię i nazwisko jest wymagane</div>
```



```
    </label>
    <label>
      <span>Email:</span>
      <input type="text" class="wpis" id="email" [(ngModel)]="email"
name="email" required pattern=".+@.+" #emailForm="ngModel">
      <div class="alert" *ngIf="emailForm.invalid && emailForm.touched"> </div>
      <div class="alert" *ngIf="emailForm.touched && emailForm.errors?.
pattern">To nie jest email</div>
    </label>
    <label>
      <span>Produkt:</span>
      <select id="produkt" class="produkt" [(ngModel)]="produkt" name="produkt"
required>
        <option value="pralka">pralka</option>
        <option value="telewizor">telewizor</option>
        <option value="lodowka">lodówka</option>
        <option value="laptop">laptop</option>
      </select>
    </label>
    <label>
      <span>Ilość:</span>
      <input type="number" class="wpis" id="ilosc" name="ilosc"
[(ngModel)]="ilosc" min="1" max="5">
    </label>
    <label>
      <span>Wiadomość:</span>
      <textarea class="wiadomosc" id="wiadomosc" [(ngModel)]="wiadomosc"
name="wiadomosc"></textarea>
      <button type="button" class="przycisk" style="background-color: red;"
(click)="formularz.resetForm()">Resetuj</button>
      <button type="submit" class="przycisk" [disabled]="!formularz.form.
valid">Zamów</button>
    </label>
  </div>
</form>
```

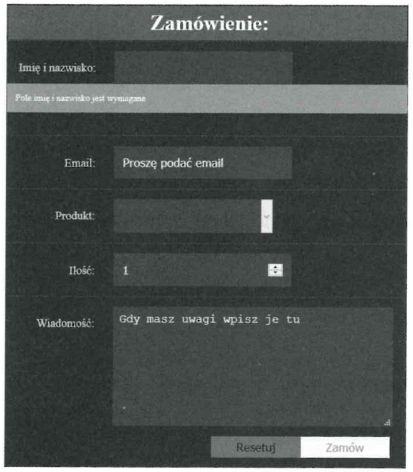
Użycie słowa kluczowego `required` powoduje, że dany element formularza jest wymagany. Nie będzie można go przesłać, jeśli pole nie zostanie uzupełnione. Dba o to kod, który znajduje się w definicji przycisku *Zamów*. Jest on nieaktywny, dopóki wymagane pola (`imieinazwisko`, `email` oraz `produkt`) są puste (rysunek 2.84).



Rysunek 2.84. Wysłanie formularza jest niemożliwe, ponieważ wymagane pole Produkt nie zostało określone

Pole `ilosc` zawiera dwa walidatory: `min` oraz `max`, których użycie powoduje, że liczba produktów została ograniczona do przedziału od 1 do 5.

Zastosowanie zmiennych szablonowych `#imieinazwiskoForm` oraz `#emailForm` wraz z przypisaniem ich do `ngModel` pozwala wyświetlić dodatkowe pola ostrzegawcze. Te dodatkowe okna są wyświetlane w momencie wybrania pola i opuszczenia go. Tę funkcjonalność zapewniają klasy `invalid` oraz `touched`. Ich działaniem steruje dyrektywa `ngIf`. Komunikaty zostały wystylizowane za pomocą klasy `.alert {font-size: 10px; padding: 10px; background-color: #f44336; color: white; margin-bottom: 15px;}` (rysunek 2.85).



Rysunek 2.85. Dodatkowe alerty

Wszystkie dostępne stany pól formularza są pokazane w tabeli 2.3.

Tabela 2.3. Klasy określające stan weryfikacji danych formularza HTML

Klasa	Opis
`valid`	Formularz został poprawnie wypełniony
`invalid`	Formularz nie został poprawnie wypełniony
`submitted`	Formularz został zatwierdzony
`touched`	Pole formularza zostało wybrane przez kliknięcie lub klawiszem *TAB*
`untouched`	Pole formularza nie zostało wybrane
`dirty`	W polu formularza było coś wpisane (mogło zostać później wykasowane)
`pristine`	W polu formularza nic nie było wpisane (jest dziewicze)

Dostępne są jeszcze stany:

- `status` — status danego pola (czy dane pole — nie cały formularz — jest typu `invalid`, czy `valid`),
- `disabled` — czy dane pole jest typu `disabled` (zwraca `true/false`),
- `errors` — indeks wskazujący błąd, przez który pole przyjmuje stan `invalid`,
- `value` — przechowuje wartość pola `input` wpisaną przez użytkownika.

W polu `email` dodatkowo zostało zawarte wyrażenie `pattern=".+@.+"`, które sprawdza, czy ustalony wzorzec pasuje do adresu email. Sprawdzane jest wystąpienie znaku `@` oraz to, czy przed tym znakiem i po nim występuje tekst.

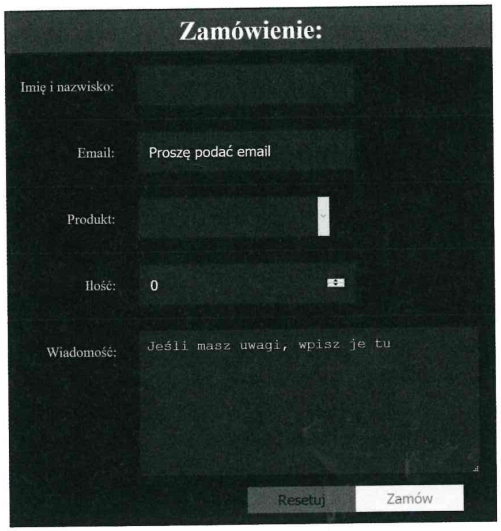
CIEKAWOSTKA

Wyrażenie, które w 99,99% gwarantuje walidację adresu email, ma postać (na podstawie RFC 5322 — *https://tools.ietf.org/html/rfc5322*):

```
(?:[a-z0-9!#$%&'*+/=?^_`{|}~-]+(?:\.[a-z0-9!#$%&'*+/=?^_`{|}~-]+)*|"(?:[\
x01-\x08\x0b\x0c\x0e-\x1f\x21\x23-\x5b\x5d-\x7f]|\\[\x01-\x09\x0b\x0c\
x0e-\x7f])*")@(?:(?:[a-z0-9](?:[a-z0-9-]*[a-z0-9])?\.)+[a-z0-9](?:[a-z0-
9-]*[a-z0-9])?|\[(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.){3}
(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?|[a-z0-9-]*[a-z0-9]:(?:[\x01-\x08\
x0b\x0c\x0e-\x1f\x21-\x5a\x53-\x7f]|\\[\x01-\x09\x0b\x0c\x0e-\x7f])+)\])
```

choć o wiele łatwiej jest użyć formy: `<input type="email" placeholder="Proszę podać email" />`.

Ponieważ zamieszczone w tabeli stany pola formularza są klasami, można im nadać style za pomocą arkusza stylów. Dodanie np. stylu `.ng-invalid {border: 2px solid blue};` (stan poprzedzamy ciągiem `ng-`) spowoduje, że wszystkie wymagane pola będą obramowane niebieską ramką (rysunek 2.86).



Rysunek 2.86. Użycie klasy .ng-invalid

Zadanie 2.25.

Stwórz ankietę składającą się z trzech pól:

- *Imię i nazwisko* — pole typu `text`,
- *W skali od 1 (najniżej) do 5 (najwyżej) oceń jakość obsługi klienta* — lista rozwijana z polami od *1* do *5*,
- *Status zawodowy: uczeń/student, pracujący, bezrobotny, na rencie/emeryturze* — pola typu `radio`.

Poniżej umieść przycisk *Zatwierdź*. Wysłanie ankiety ma być możliwe po wpisaniu wartości we wszystkich polach. Wynik wyświetl w konsoli.

Pytania kontrolne

1. Jak zaimplementować formularz?
2. W jakim celu wykonuje się walidację formularza?
3. W jaki sposób można zweryfikować informacje wprowadzone w polach formularza?

3

Środowisko uruchomieniowe — platforma Node.js

Node.js to środowisko uruchomieniowe dla programów napisanych w JavaScripcie, które pozwala je uruchamiać poza przeglądarką.

3.1. Platforma Node.js

3.1.1. Czym jest Node.js?

Platforma Node.js jest oparta na **silniku V8**, stworzonym przez firmę Google i używanym w przeglądarce Google Chrome. Dodatkowo w skład platformy weszła biblioteka **libuv**, która zapewnia **asynchroniczność**.

Do stworzenia Node.js użyto języka C++. Zrobiono tak z dwóch powodów. Po pierwsze, silnik V8, który jest jednym z filarów narzędzia, również został napisany z wykorzystaniem tego języka. Po drugie, Node.js zyskał w ten sposób możliwość rozbudowania swoich funkcji o zewnętrzne rozszerzenia i biblioteki.

CIEKAWOSTKA

Kod źródłowy silnika V8 jest dostępny na licencji open source, co oznacza, że twórca zezwala na jego bezpłatne rozpowszechnianie i na wprowadzanie w nim zmian. Taki sposób licencjonowania pozwolił wykorzystać go w Node.js.

Zadaniem silnika jest kompilacja i wykonanie kodu napisanego w JavaScripcie *just in time* —