Dokumentacja do projektu

„Komunikacja frontendu i backendu w NodeJS”

projekt systemu informatycznego na zaliczenie laboratorium „Praktyczne aspekty projektowania systemów informatycznych”

Wykonanie:
1) Anna Drobek
2) Radosław Biereta

Spis treści

1. Wstęp

Projekt systemu informatycznego na zaliczenie laboratorium z laboratoriów „Praktyczne aspekty projektowania systemów informatycznych” został wykonany w oparciu o aplikacje napisane w NodeJS. Wykonany system informatyczny zbudowany jest w oparciu o dwie aplikacje tj. backend i frontend które docelowo mają pokazać komunikację backendu i frontendu poprzez wyświetlanie godziny obranej z backendu i frontendu. Aplikacje komunikują się wzajemnie ze sobą w oparciu o protokół http w jednym obrazie dockerowym. Praca równoległa jest realizowana za pomocą menedżera pm2 dla aplikacji. Budowanie obrazów dockerowych jest realizowane w oparciu o CI/CD udostępniane przez Github actions. Obrazy dockerowe przechowywane są w repozytorium Docker Hub.

2. Środowisko

2.1 Aplikacja backend_http.js

Aplikacja została wykonana w oparciu o standardowe biblioteki wchodzące w skład NodeJS w wersji 16.18.1 do pobrania z https://nodejs.org/dist/v16.18.1/. Aplikacja pomimo możliwości dostarczanych przez samo środowisko NodeJS i mnogości bibliotek do komunikacji w oparciu o protokoły http/https została wykonana w filozofii pure-code która cechuje się niewykorzystywaniem gotowych frameworków.

2.2 Aplikacja guifrontend.js

Aplikacja została wykonana w oparciu o standardowe biblioteki wchodzące w skład NodeJS w wersji 16.18.1 do pobrania z https://nodejs.org/dist/v16.18.1/. Dodatkowo zostały użyte biblioteki EXPRESS, EJS,PATH.

2.2.1 EXPRESS

Express.js to bardzo popularna biblioteka NodeJS typu open source wspomagająca pracę backend developerów. Została wypuszczona w 2010 r. jako darmowe oprogramowanie. Pozwala tworzyć proste strony www oraz rozbudowane hybrydowe aplikacje webowe, które można uruchamiać w przeglądarce. Cechuje się dużą lekkością i wydajnością, wymaga znajomości jedynie Java Script i HTML oraz służy organizacji projektu po stronie serwera w oparciu o architekturę MVC (Model-View-Controller). Stanowi część składową technologii MEAN Software Stack napisanej w Java Script, w której skład wchodzą m.in. Angular.js, Node.js i MongoDB.

2.2.2 EJS

Biblioteka EJS (*ang. Embedded JavaScript templating*) jest jednym z wielu silników renderowania dla Express. Zwalnia on z konieczności ręcznego zarządzania szablonami „stron”. Wymaga podania folderu w którym zapisane będą widoki. Najważniejszą cechą EJS jest możliwość mieszania elementów deklaratywnych i imperatywnych wewnątrz widoków, w tym m.in. deklarowanie zmiennych lokalnych. Ogranicznikami struktury imperatywnej są znaczniki <% zmienna %>, wypisanie wartości możliwe jest dzięki znacznikom <%= zmienna %> natomiast wypisanie wartości zakodowanej (tzw. HTML encoding) obsługuje znacznik <%- zmienna %>

2.2.3 PATH

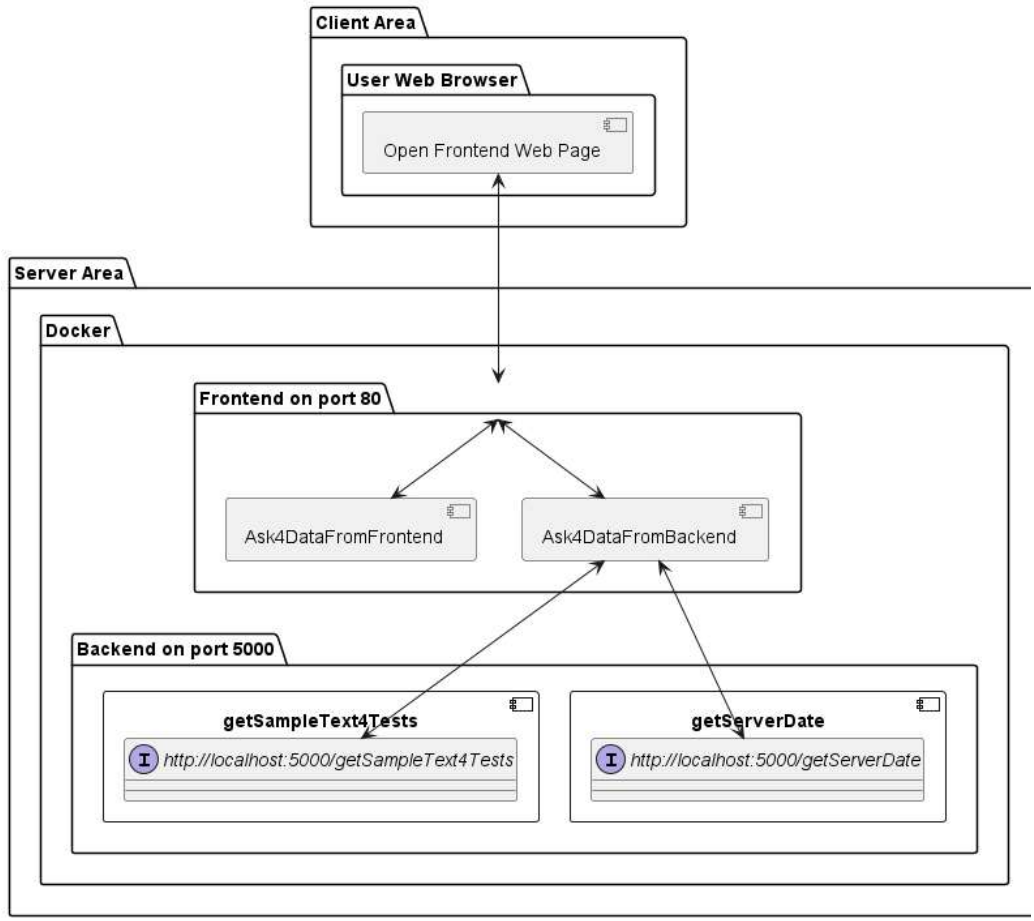
Biblioteka path zawiera szereg funkcji ułatwiających manipulowanie danymi zgromadzonymi lokalnie na systemie plików. W przypadku frontendu biblioteka jest używana do ładowania statycznych kodów css oraz obrazków wykorzystywanych przez bibliotekę ejs w celu wyrenderowania gotowej strony wyświetlanej użytkownikowi końcowemu.

Kody aplikacji przechowywane są w publicznych repozytoriach na github.com. Stosownie do elementu są to:
- https://github.com/annadrobek/nodeJS_backend/
- https://github.com/annadrobek/nodeJS_frontend/

Gotowe obrazy są przechowywane w repozytorium https://hub.docker.com/u/annadrobek

3. Przepływy informacji

Aplikacje backend_http.js oraz guifrontend.js komunikują się ze sobą w oparciu o protokół http. Backend oczekuje na połączenia na porcie 5000/TCP natomiast frontend symuluje działanie serwera www wystawiając nasłuchiwanie na porcie 80/TCP który to jest używany przez przeglądarki internetowe do otwierania stron internetowych.



Rysunek 1: Przepływ informacji w projekcie

Backend posiada zakodowane 2 metody w swoim API typu GET

1) getSampleText4Tests

2) getServerDate

3.1. getSampleText4Tests

Metoda getSampleText4Tests służy do generowania pseudolosowych ciągów znaków o długości wg zadanych parametrów. W tym przypadku od 5 do 20 znaków

```
if (req.url === "/getSampleText4Tests" && req.method === "GET") {
      var sampleText = makeSampleText(between(5, 20));
      res.writeHead(200, {
         "Content-Type": "application/json"
      });
```

W tym celu korzysta z dwóch funkcji

```
function makeSampleText(length) {
var result = '';
   var characters = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789';
   var charactersLength = characters.length;
   for (var i = 0; i < length; i++) {
      result += characters.charAt(Math.floor(Math.random() * charactersLength));
   }
   return result;
}
```

oraz

```
function between(min, max) {
   return Math.floor(
      Math.random() * (max - min + 1) + min
   )
}
```

Metoda getSampleText4Tests jest napisana do celów czysto testowych z wykorzystaniem frameworku testowania aplikacji NodeJS o nazwie mocha - https://mochajs.org/

3.2. getServerDate

Metoda getServerDate służy do pobierania daty w precyzji do milisekund.

```
if (req.url === "/getServerDate" && req.method === "GET") {
      var date = returnDate();
      res.writeHead(200, {
        "Content-Type": "application/json"
      });
```

korzysta z funkcji

```
function returnDate() {
   let date_ob = new Date();
   var miliseconds = date_ob.getMilliseconds()
   if (miliseconds <= 9) {miliseconds = '00' + miliseconds};
   if (miliseconds <= 99 && miliseconds >= 10) {miliseconds = '0' + miliseconds
   };
   var seconds = date_ob.getSeconds();
   if (seconds <= 9) {seconds = '0' + seconds;};
   var minutes = date_ob.getMinutes();
   if (minutes <= 9) {minutes = '0' + minutes;};
   var hour = date_ob.getHours();
   if (hour <= 9) {hour = '0' + hour;};
   var year = date_ob.getFullYear();
   var month = date_ob.getMonth() + 1;
   if (month <= 9) {month = '0' + month;};
   var day = date_ob.getDate();
   if (day <= 9) {day = '0' + day;};
   return day + '.' + month + '.' + year + ' ' + hour + ':' + minutes + ":" + seconds + ":" + miliseconds;
}
```

3.3 Router / frontendu aplikacyjnego

Działanie frontendu opiera się o routery biblioteki express. Zasadą działania routera jest podjęcie stosowanej akcji względem żądania płynącego od strony klienta.

```
app.get("/", function(req, res1) {
   const http = require('http');
```

```
  let request = http.get('http://localhost:5000/getServerDate', (res) => {
    let data = '';
    if (res.statusCode !== 200) {
      console.error('Did not get an OK from the server.');
      res.resume();
      return;
    }
    res.on('data', (chunk) => {
      data += chunk;
    });
    res.on('close', () => {
      var data1 = returnDate();
      console.log('Frontend date: ' + data1);
      console.log('Backend date: ' + data);
      res1.render("pages/index", {
        serverdate: data,
        frontenddate: data1
      });
    });
  });
});
```

Do uzyskaniu żądania otworzenia strony głównej frontendu zapisanego w postaci app.get(„/”, fumction(req, res) wykonywana jest asynchronicznie anonimowa funkcja, która odpytuje backend poprzez wywołanie żądania http.get(‘http://localhost:5000/getServerDate)’(res)) oraz przyporządkowanie podejmowanych akcji do obiektu res. Aplikacja frontendowa nie posiada pełnej obsługi wszystkich możliwości pełnoprawnego serwera www. Obsługuje tylko 3 przypadki. Pierwszy przypadek jeżeli zostanie zwrócony kod odpowiedzi z backendu inny niż 200 wówczas ma przerwać działanie. W przypadku otrzymania kodu 200 ma przejść do przyporządkowywania do zmiennej data danych pochodzących z backendu. Finalnie ma pobrać datę z aplikacji na froncie. Dwie pobrane daty są parametrami z którymi ma być wyrenderowana strona index.html i zaprezentowana użytkownikowi końcowemu. Dodatkowo pobrane backenddate i frontenddate mają być wypisane w konsoli.

4. Testowanie

Testowanie jako nieodłączny element procesu CI/CD jest realizowane przy użyciu frameworku mocha służącego do testowania aplikacji NodeJS

5. Budowanie i osadzanie środowiska

5.1 Budowanie ręczne

Do budowania ręcznego konieczne jest posiadanie środowiska docker. Aby zbudować obraz dockerowy należy uruchomić polecenie

```
docker build . --file Dockerfile --tag annadrobek/laboratorium:latest
```

a następnie

```
docker login -u $DOCKER_USER -p $DOCKER_PASSWORD
```

aby finalnie móc wykonać polecenie

```
docker push annadrobek/laboratorium:latest
```

5.2 Budowanie automatyczne

Budowanie automatyczne jest zrealizowane w oparciu o GitHub Actions oraz stworzony w tym celu workflow ci_cd.yml

```
name: Docker Image CI & CD
on:
  push:
    branches: [ "main" ]
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v3
    - name: Docker login
      env:
        DOCKER_USER: ${{ secrets.DOCKERHUBUSER }}
        DOCKER_PASSWORD: ${{ secrets.DOCKERHUBPASSWORD }}
      run: echo $DOCKER_PASSWORD | docker login -u $DOCKER_USER --password-stdin
    - name: Build the Docker image
      run: docker build . --file Dockerfile --tag annadrobek/laboratorium:latest
    - name: Push the Docker image
      run: docker push annadrobek/laboratorium:latest
```

Workflow zakłada uruchomienie procesu budowania i wysyłania do Docker Hub obrazu kontenera przy każdorazowym uaktualnianiu repozytorium frontendu.