Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Звіт до комп’ютерного практикуму №4

З дисципліни «Основи Back-end технологій»

Прийняв:
Зубко Роман Анатолійович

Виконав:
Студент 3 курсу,
Котенко Ярослав
гр. ІМ-13

2024 р.

Лабораторна робота №4. NodeJS.

Створення серверу за допомогою express. Обробка маршрутів. Шаблонізація.

Завдання.

Розробити веб-застосунок для отримання даних про погоду

• В шаблоні потрібно сформувати меню посилань з назвами міст (одним із пунктів меню має бути пункт із зазначенням місцезнаходження автора)

• Формат рядка запиту для отримання даних про погоду:

/weather/{city}, де city - назва вибраного міста

• Дані про погоду можна отримати відправкою запиту на OpenWeatherMap

• Advanced. Отримати дані про погоду в місцезнаходженні користувача за таким URI: /weather/

Хід роботи

1. Сформував меню посилань з назвами міст.

Код програми

index.js є головним файлом. У ньому відбувається підключення всіх необхідних залежностей для функціонування застосунку, а також налаштовані маршрути**.**

Файл: *index.js*

```
JS index.js > ...
1   import express from 'express'
2   import { create } from 'express-handlebars'
3   import path from 'node:path'
4   import { fileURLToPath } from 'node:url'
5   import bodyParser from 'body-parser'
6
7   import mainRoute from './routes/main.js'
8   import weatherRoute from './routes/weather.js'
9
```

```
const __filename = fileURLToPath(import.meta.url)
const __dirname = path.dirname(__filename)

const PORT = process.env.PORT || 5000
const app = express()
const hbs = create({
  defaultLayout: 'main',
  extname: 'hbs',
})

app.engine('hbs', hbs.engine)
app.set('view engine', 'hbs')
app.set('views','views')
app.use(express.static(path.resolve(__dirname, 'public')))
app.use(bodyParser.json())

app.use('/main', mainRoute)
app.use('/weather', weatherRoute)

const start = () => {
  app.listen(PORT, () => {
    console.log(`Server is running on port: ${PORT}`)
  })
}

start()
```

Файл: *main.js*

```
routes > JS main.js > ...
import { Router } from 'express'
import Controller from './controller/Controller.js'

const router = Router()

router.get('/', Controller.getMainPage)

export default router
```

У цьому файлі описаний маршрут головної сторінки, де представлений список міст для вибору.

```
routes > controller > JS Controller.js > Controller > getMainPage > title
1   export default class Controller {
2     static location = ''
3     static getMainPage(req, res) {
4       res.render('home', {
5         title: 'Main'
6       })
7     }
8     static async getCityPage(req, res) {
9       try {
```

Я використовував клас **Controller**, де описуються всі методи, які будуть використовуватися в роутах. У файлі main.js я використовував метод **getMainPage**.

Файл: *home.hbs*

```
views > home.hbs > div.container > div.main-menu > ul > li > a.location
1   <div class="container">
2     <div class="main-menu">
3       <h1><i class="fas fa-sun"></i> Weather App</h1>
4       <h2>Select a city</h2>
5       <ul>
6         <li><a href="/weather/Kyiv">Kyiv</a></li>
7         <li><a href="/weather/Lviv">Lviv</a></li>
8         <li><a href="/weather/Odesa">Odesa</a></li>
9         <li><a href="/weather/Cherkasy">Cherkasy</a></li>
10        <li><a href="/weather/Kharkiv">Kharkiv</a></li>
11        <li><a href="/weather/Dnipro">Dnipro</a></li>
12        <li><a href="/weather/" class="location">My location</a></li>
13      </ul>
14    </div>
15  </div>
```

Я скористався шаблонізатором **express-handlebar**s, що дозволило мені описати структуру всіх сторінок лише один раз у головному макеті **main.hbs**. В інших сторінках я лише описав саме тіло сторінки.

Макет:

Файл: *main.hbs*

views > layouts > main.hbs > ...

```
1  {{>header}}
2  <body>
3    {{{body}}}
4  {{>footer}}
5  </body>
6  </html>
```

Partials:

Файл: *header.hbs*

views > partials > header.hbs > html > head

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4    <meta charset="UTF-8">
5    <meta name="viewport" content="width=device-width, initial-scale=1.0">
6    <link rel="stylesheet" href="/index.css">
7    <link rel="stylesheet" href="/city.css">
8    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.0.0-beta3/css/all.min.css" integrity="sha
9    <title>{{title}}</title>
10 </head>
```

Файл: *footer.hbs*

views > partials > footer.hbs > script

```
1  <script src="/app.js"></script>
```

Скріншоти виконання коду

2. Створив формат рядка запиту для отримання даних про погоду в обраному місті: /weather/{city}. Дані про погоду отримав відправкою запиту на **Weather API**.

Код програми

Файл: *weather.js*

У цьому файлі міститься логіка для визначення погоди у обраному місті та генерування сторінки з даними про погоду у вибраному місті.

routes > JS weather.js > ...

```
import { Router } from 'express'
import Controller from './controller/Controller.js'

const router = Router()
router.get('/', Controller.getLocation)
router.get('/:city', Controller.getCityPage)

export default router
```

```
static async getCityPage(req, res) {
  try {
    let city = req.params.city
    if(!city) city = this.location
    const weatherResponse = await Controller.defineWeather(city)
    const weatherParams = {
      temp: weatherResponse.current.temp_c,
      press: weatherResponse.current.pressure_mb,
      hum: weatherResponse.current.humidity,
      img: weatherResponse.current.condition.icon
    }
    res.render('city', {
      title: city,
      cityName: city,
      weatherParams,
    })
  } catch (e) {
    console.log(e)
  }
}
```

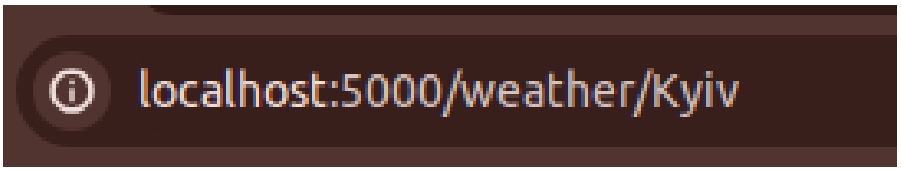
```
static async defineWeather(cityName) {
  const API_KEY = '0cc11595e89649cb8c4232321230901'
  const weatherResult = await fetch(`https://api.weatherapi.com/v1/forecast.json?key=${API_KEY}&q=${cityName}`)
  if (!weatherResult.ok) {
    throw new Error(`Failed to fetch weather data: ${response.status} ${response.statusText}`)
  }
  return weatherResult.json()
}
```

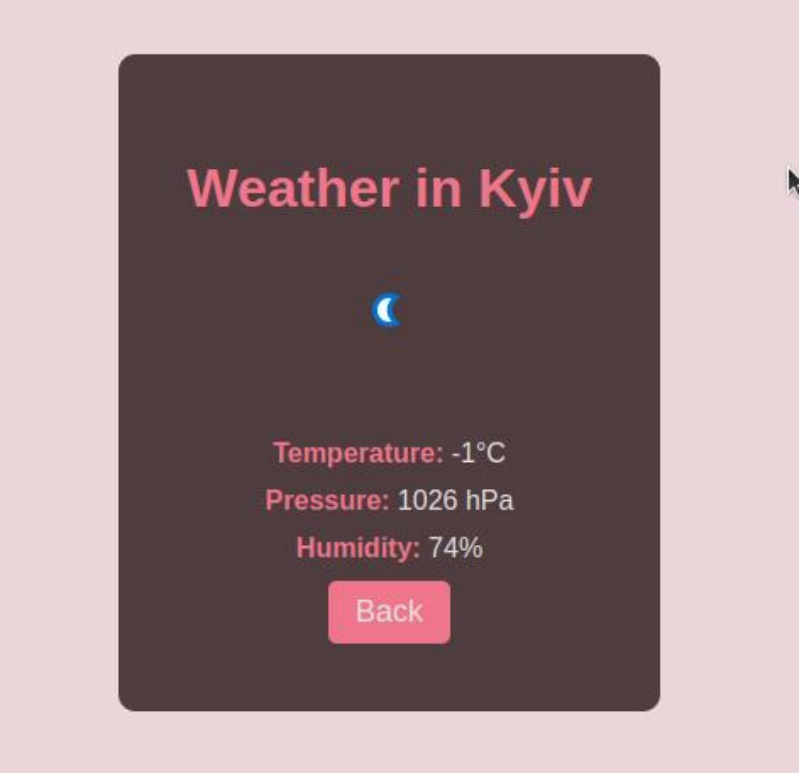
Файл: *city.hbs*

views > city.hbs > div.container > div.city-weather > div.weather-image > img

```
<div class="container">
  <div class="city-weather">
    <h1>Weather in {{cityName}}</h1>
    <div class="weather-image">
      <img src="{{weatherParams.img}}" alt="Sunny">
    </div>
    <div class="weather-info">
      <p><strong>Temperature:</strong> {{weatherParams.temp}}°C</p>
      <p><strong>Pressure:</strong> {{weatherParams.press}} hPa</p>
      <p><strong>Humidity:</strong> {{weatherParams.hum}}%</p>
    </div>
    <a href="/main" class="back-button">Back</a>
  </div>
</div>
```

Скріншоти виконання коду





3. Реалізував отримання даних про погоду у місцезнаходженні користувача за наступним URI: /weather/

Код програми

Файл: *weather.js*

```
routes > JS weather.js > ...
1  import { Router } from 'express'
2  import Controller from './controller/Controller.js'
3
4  const router = Router()
5  router.get('/', Controller.getLocation)
6  router.get('/:city', Controller.getCityPage)
7
8  export default router
```

У цьому файлі я також впровадив функціонал для обробки маршруту /weather/.

```
static getLocation(req,res) {
  Controller.location = req.query.city
  setTimeout(() => Controller.getCityPage(req,res), 1000)
}
```

У контролері я створив метод **getLocation**, який очікує дані про місцезнаходження користувача і лише після їх отримання генерує сторінку з даними.

Клієнтський код, який визначає місцезнаходження:

Файл: *app.js*

```
public > JS app.js > getLocation
const API_DATA_KEY = 'e87ac4483fb3469f9b44372a0e2936ca'
const locationButton = document.querySelector('.location')

async function getLocation() {
  return new Promise((resolve, reject) => {
    if (navigator.geolocation) {
      navigator.geolocation.getCurrentPosition(position => resolve(position), error => reject(error))
    } else {
      reject(new Error("Geolocation is not supported by this browser."))
    }
  });
}
```

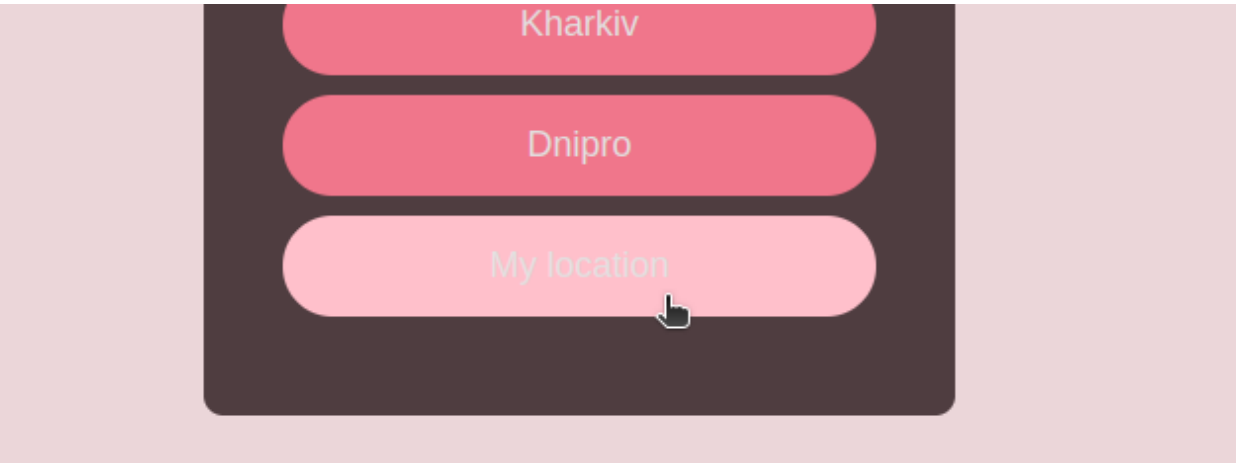
```
async function showPosition(position) {
  const latitude = position.coords.latitude
  const longitude = position.coords.longitude
  const url = `https://api.opencagedata.com/geocode/v1/json?q=${latitude}+${longitude}&key=${API_DATA_KEY}`
  try {
    const response = await fetch(url)
    const data = await response.json()
    const city = data.results[0].components.city
    return city
  } catch (error) {
    console.error('Error getting location:', error)
    throw new Error("Error getting location")
  }
}
```

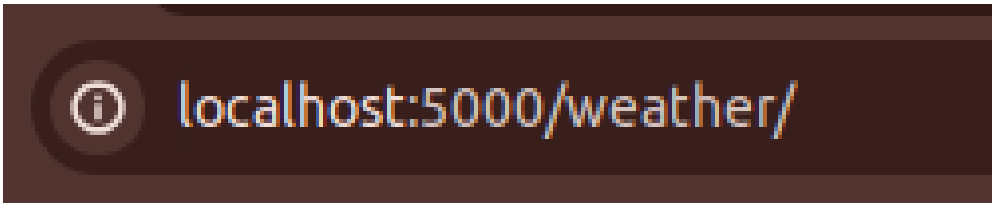
Тут я скористався додатковим сервісом для визначення міста по координатам: **opencagedata**.

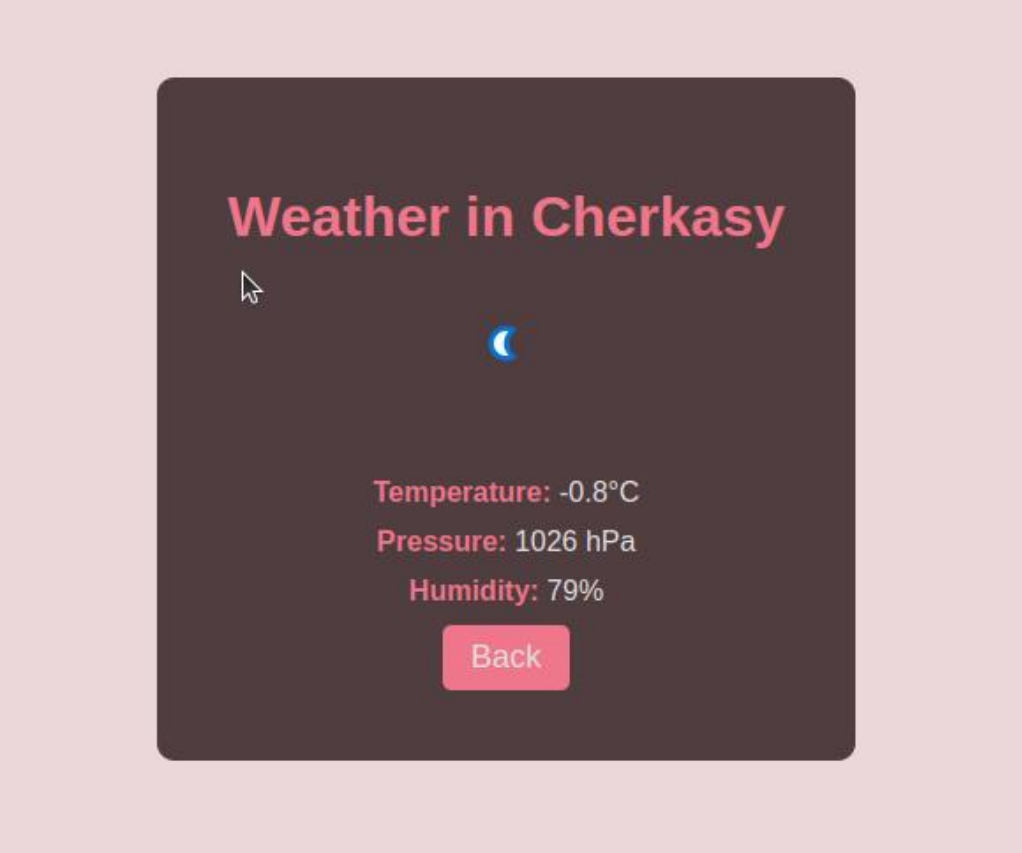
```
if (locationButton) {
  locationButton.addEventListener('click', async (event) => {
    try {
      const userLocation = await getLocation()
      const city = await showPosition(userLocation)
      fetch(`http://localhost:5000/weather?city=${city}`)
      .catch(err => {
        alert(err)
      })
    } catch (error) {
      alert(error.message)
    }
  });
}
```

Відправив місцезнаходження на сервер.

Скріншоти виконання коду







Висновок:

В результаті виконання лабораторної роботи я здобув невеликий досвід у розробці веб-застосунків, зокрема, в роботі з взаємодією клієнта та сервера для отримання даних про погоду. Я оволодів навичками роботи з API, зокрема, WeatherAPI, із якого можна отримати потрібну інформацію про погоду за допомогою відповідних запитів. Крім того, я навчився використовувати геолокацію для визначення місцезнаходження користувача та взаємодії з відповідними сервісами, наприклад, opencagedata. Загалом, ці завдання допомогли мені розвинути навички веб-розробника та збагатити мої знання у сфері веб-технологій.