Залікова робота з курсу “Основи *web*-програмування”

Студента 3 курсу групи МП-31 Захарова Дмитра Олеговича

5 грудня 2023 р.

Білет #5

Питання 1

Умова. Що таке асинхронність у *Javascript*? Які методи використовуються для управління асинхронним кодом?

Відповідь. Зазвичай, базові програми працюють таким чином, що є деяка послідовність операцій, котрі виконуються послідовно. Для застосунків це не завжди зручно і, головне, не завжди швидко. **Асинхронність** дозволяє виконувати деякі операції *паралельно*.

Як базовий приклад – нехай нам потрібно зробити API для вебсайту, де, наприклад, ми приймаємо великий *.pdf* файл, і нам потрібно якимось чином його обробити. Обробка – операція дорога і ми не можемо дозволити одразу прийняти запит і його обробити (інакше після 5-10 запитів наша API повністю зависне). Тому, замість цього, ми можемо створити чергу і запустити декілька процесів для обробки кожного. Таким чином, ми зможемо використати асинхронність.

Для управління асинхронністю в *Javascript* використовуються два ключових слова: `async` та `await`. `async` використовується для позначення методу або функції, що буде використовуватися у асинхронному режимі. Функція в такому разі не буде повертати значення безпосередньо, а так званий `Promise` – тобто обгортку навколо значення.

Наприклад, давайте створимо наступну функцію:

```
1 const NUMBERS_TO_DISPLAY = 10
2 const MILISECONDS_TO_WAIT = 1000
3
4 async function fn(a) {
```

```
5       await new Promise(r => setTimeout(r, MILISECONDS_TO_WAIT * (
        NUMBERS_TO_DISPLAY - a)))
6       return a
7   }
8
9   for (var i = 0; i < NUMBERS_TO_DISPLAY; i++) {
10      fn(i).then(number => console.log(number))
11  }
```

Лістинг 1: Створення асинхронної функції

Коротке пояснення: ми проходимось по числам від 0 до `NUMBERS_TO_DISPLAY-1`. Далі викликаємо асинхронну функцію `fn` для кожного числа `k`, на початку котрої чекаємо `NUMBERS_TO_DISPLAY-k` секунд (тобто, для числа 0 ми чекаємо `NUMBER_TO_DISPLAY` секунд, для наступного числа на одну секунду меньше тощо). Для очікування вже бачимо ключове клово `await`, але поки на нього не звертатимемо увагу.

Якщо б функція не була асинхронною, то ми б спочатку зачекали 10 секунд і вивели б 0, потім ще 9 секунд і вивели б 1 і так далі до 9. Тобто сумарно ми б чекали 55 секунд.

Якщо ж ми використовуємо асинхронність, то ми запускаємо усі процеси і усі числа будуть виведені через 10 секунд (час найбільшого очікування з усіх процесів). Причому, не у порядку зростання, як б було при послідовному виконанні, а навпаки – спочатку найбільше (оскільки треба чекати лише 1 секунду) і до 0. Для виведення ми використали метод `.then`, в котрий ми можемо передати анонімну функцію, що виконається при завершенні відповідного процесу.

Розглянемо тепер ключове слово `await`. Трошки модифікуємо нашу програму наступним чином:

```
1   const NUMBERS_TO_DISPLAY = 10
2   const MILISECONDS_TO_WAIT = 1000
3
4   async function fn(a) {
5       await new Promise(r => setTimeout(r, MILISECONDS_TO_WAIT * (
        NUMBERS_TO_DISPLAY - a)))
6       return a
7   }
8
9   async function main() {
10      for (var i = 0; i < NUMBERS_TO_DISPLAY; i++) {
11          let result = await fn(i)
12          console.log(result)
13      }
14  }
15
16  main()
```

Лістинг 2: Використання ключового слову `await`

Тепер, для кожної цифри ми будемо очікувати результат з функції `fn`. Це, по своїй суті, буде відповідати послідовному виконанню операцій і ми будемо чекати 55 секунд. Часто це використовується тоді, коли ми не можемо розпаралелити процеси і нам потрібно дочекати конкретний результат.

Також, іноді асинхронні функції можуть повертати помилки і їх треба оброблювати. Приклад можна побачити знизу:

```
1  const NUMBERS_TO_DISPLAY = 10
2  const MILISECONDS_TO_WAIT = 1000
3
4  async function fn(a) {
5      throw new Error('Error in fn')
6      return a // This will never be returned
7  }
8
9  async function main() {
10     for (var i = 0; i < NUMBERS_TO_DISPLAY; i++) {
11         fn(i).then(result => {
12             console.log(result)
13         }).catch(error => {
14             console.log('Hey, I know how to handle errors in
       asynchronous js!')
15             console.log(error)
16         })
17     }
18 }
19
20 main()
```

Лістинг 3: Обробка помилок

На виході отримаємо виведення помилок у консоль разом із повідомленням "*Hey, I know how to handle errors in asynchronous js!*".

Питання 2

Умова. Медіа запити в *CSS*.

Відповідь. Найбільш широке використання медіа запитів – це додавання адаптивності до веб-сайтів. Тобто, ми можемо задавати умови на розмір екрану, і відповідно до цього, задавати унікальні стилі (зменшити розмір шрифта або зробити інше вирівнювання на сторінці тощо). Синтаксис команди має наступний вигляд:

```
@media <media-query-list> { <rule-list> }
```

Для демонстрації надамо наступну програму:

```html
<!doctype html>

<html lang="en">
<head>
    <meta charset="utf-8">
    <title>Programming exam</title>
</head>
<style>
    body { background-color: rgb(230.0, 230.0, 230.0); }
    h1   { color: rgb(20.0, 20.0, 20.0); }

    @media screen and (max-width: 900px) {
        .small-screen {
            color: red;
        }
    }
    @media screen and (min-width: 900px) {
        .huge-screen {
            color: red;
        }
    }
</style>
<body>
    <!-- Will be displayed based on the screen's size -->
    <h1 class="huge-screen"> Huge screen here! </h1>
    <h1 class="small-screen"> Small screen here! </h1>

    <!-- This launches a .js script for the first exam question -->
    <script src="main.js"></script>
</body>
</html>
```

Лістинг 4: Повна `html` сторінка з використанням `@media` запитів

Вебсайт складається з двох чорних заголовків: "*Huge screen here!*", "*Small screen here!*". Якщо екран маленький, то червоним помітиться напис "*Small screen here!*", а якщо великий – "*Huge screen here!*". Це досягається за допомогою медіазапиту:

```css
@media screen and (max-width: 900px) {
    .small-screen {
        color: red;
    }
}
@media screen and (min-width: 900px) {
    .huge-screen {
        color: red;
    }
}
```

Лістинг 5: Використання `@media` запитів

За допомогою умови `@media screen and (max-width: 900px) { ... }` ми кажемо *CSS*'у, що те, що в дужках, має застосовуватись лише за умови, що ширина екрану має значення до 900 пікселів. Відповідно, умова на `min-width` позначає, що зміни відбуваються **від** ширини екрану в 900 пікселів.