**2 Programowanie asynchroniczne.**

**Async/await.**

**Wstęp**

W poprzednim szkoleniu poznałeś, czym są obiekty typu Promise oraz jakie jest ich zastosowanie. Aby dopełnić tę wiedzę, zapoznamy się teraz z niejako zagadnieniem powiązanym do wcześniejszego - async oraz await. Jest to wygodne i efektywne rozwiązanie dla zarządzania promise.

**Funkcje Async**

Słowo kluczowe async umieszcza się przed definicją dowolnej funkcji, np.

|  |
| --- |
| async function f(){  return 7;  } |

Tak dodany async do sygnatury funkcji oznacza jedną rzecz. Każda zwracana przez taką funkcję wartość będzie traktowana jako Promise. Bądź świadomy, że dzięki async taki obiekt powstaje niejawnie, a zapis jest równoważny do następującego:

|  |
| --- |
| async function f() {  return Promise.resolve(7); } |

Z racji tego, iż każda funkcja z przydomkiem async zwraca obiekt Promise, możemy bez problemu zacząć wykorzystywać funkcję then() i odpowiednio zarządzać kodem konsumującym.

|  |
| --- |
| async function f(){  return 7; }  let promise = f(); promise.then(alert); // 7 |

Przyznaj, że bardzo proste i intuicyjne zastosowanie (o ile zapoznałeś się z wcześniejszymi materiałami i Promise). Async więc oszczędza nam dodatkowego pisania i tworzenia nadmiarowych linii kodu, gdy chcemy zwrócić Promise z danej funkcji.

Jednak to nie wszystkie korzyści płynące z takiego rozwiązania. Proces zwracania i zarządzania obiektami Promise zyskuje na sile, gdy async połączymy z funkcjonalnością następnego keyword - await.

**Await**

Syntax:

|  |
| --- |
| let value = await promise; |

Await ma następujące działanie - wstrzymuje wykonanie funkcji w linii, w której wykorzystujemy ten keyword i wznawia pracę dopiero, gdy kod produkujący promise-a zwróci odpowiedni result do konsumera.

Await więc ma niemal identyczne działanie do .then(), z tą różnicą, że jest bardziej eleganckim i ładniejszym sposobem na obsługiwanie flow programu.

|  |
| --- |
| async function f() {   let promise = new Promise((resolve, reject) => {  setTimeout(() => resolve("Job done!"), 1000)  });   let result = await promise; // czekaj, aż promise zwróci result  alert(result); // "Job done!" }  f(); |

|  |
| --- |
| **Uwaga!**  Będę zwracał na to uwagę w dalszej części szkolenia, ale już teraz chciałbym Cię przestrzec przed tym, iż await może być wykorzystywany tylko wewnątrz funkcji async. Niemożliwe jest jego użycie wewnątrz “zwykłej funkcji”.  Poniżej przykład, który, z racji właśnie umieszczenia await w non-async funkcji, zakończy się błędem interpretacji: |

|  |
| --- |
| function f() {  let promise = Promise.resolve(1);  let result = await promise; // Syntax error } |

**Pierwszy program z wykorzystaniem async/await**

Tak jak zdążyłem Ci już wspomnieć, async i await są dobrą i zgrabną alternatywą dla Promise. Ich użycie upraszcza kod i znacząco zwiększa czytelność naszego rozwiązania. Kojarzysz przykład API ze wcześniejszego szkolenia, gdzie z JSON-a pobieraliśmy nazwę użytkownika, a **następnie** czytaliśmy terminarz spotkań? Niżej postaramy się zrealizować to samo, ale przy użyciu async i await.

|  |
| --- |
| async function getMeetings() {  let userResponse = fetch('/devs-mentoring-api/courses/student-details/Jan.json');  let userResponse = await response.json(); // wyciągnięcie jsona z response (dane userów)    let calendarResponse = fetch(`/devs-mentoring-api/calendar/${user.uuid}`); // pobranie response zawierającego spotkania  let meetings = await calendarResponse.json; // wyciągnięcie jsona z response (spotkania)    // niżej operacje na obiekcie JSON - meetings  ... } |

Rzućmy jeszcze okiem na to samo, ale zrealizowane w inny sposób (przykład ze wcześniejszego szkolenia):

|  |
| --- |
| fetch('/devs-mentoring-api/courses/student-details/Jan.json')  .then(response => response.json) // wyciągnięcie jsona z response (dane userów)  .then(user => fetch(`/devs-mentoring-api/calendar/${user.uuid}`) // pobranie response zawierającego spotkania  .then(response => response.json) // wyciągnięcie jsona z response (spotkania)  .then(meetings => ...) // tutaj możemy zarządzać spotkaniami |

Przyznaj, że rozwiązanie z wykorzystaniem async/await jest o wiele bardziej przyjazne dla oka. Cechuje się schludnością i starannością. Mam nadzieję, że właśnie przez takie porównanie zachęcę Cię do korzystania z async i await.

**Jak obsługiwać błędy?**

No dobrze, poznaliśmy, jak odpowiednio obsługiwać zwracane rezultaty przez Promise, ale nie zahaczyliśmy jeszcze o możliwość rejectowania operacji i obsługiwania Errorów (do tej pory realizowaliśmy to przez wywołanie reject umieszczanego w kodzie produkcyjnym).

Aby zrealizować to założenia w funkcji async, wykorzystamy klauzulę throw, która umożliwi nam rzucenie wyjątkiem - Errorem i otrzymanie efektu podobnego do wywołania funkcji reject.

|  |
| --- |
| async function f() {  throw new Error("Whoops!"); } |

Informacja ta jest na tyle istotna, że dzięki niej wiemy, jak obsługiwać ewentualnie zwracane błędy, wykorzystując funkcje async. Użyjemy bowiem bloku try… catch().

|  |
| --- |
| async function f() {  try {  let response = await fetch('example.com');  }  catch(err) {  alert(err); // TypeError: failed to fetch  } }  f(); |

To, co wyżej możemy zapisać również w poniższy sposób:

|  |
| --- |
| async function f() {  let response = await fetch('http://no-such-url'); }  f().catch(alert); // TypeError: failed to fetch // (\*) |

Oba przykłady zwrócą nam ten sam rezultat - to, z jakiego rozwiązania skorzystamy zależy tylko do nas - programistów.

**Podsumowanie**

Keyword async umieszczany przed nazwą funkcji niesie ze sobą dwa rezultaty:

* Zawsze zwraca obiekt typu promise
* Umożliwia wykorzystanie await

Await natomiast…

* Czeka dopóki promise nie zwróci odpowiedniego rezultatu (wtedy wywoływana jest funkcja resolve)
* Jeżeli natomiast zostanie zwrócony błąd (reject), to generowany jest wyjątek, który możemy obsłużyć blokiem try… catch() lub funkcją .catch().

Dzięki wykorzystaniu async/await, nasz kod staje się o wiele bardziej czytelny i efektywny. Dzięki nim ograniczamy konieczność wykorzystania .then(). Nie zapominajmy jednak, że nie zwalnia nas to z konieczności wykorzystywania promise-ów. Async/await są jedynie sposobami na zarządzanie takim typem obiektów.