Resurse suplimentare
pe care recomand să le parcurgi

Async/Await: Sintaxa Modernă pentru Asincronicitate

Async/await, introdus în ES2017 (ES8), este o modalitate de a scrie cod asincron care arată și se comportă aproape ca și codul sincron. Este construit peste promises și face codul mai ușor de citit și de întreținut.

Funcții Async

O funcție async este declarată folosind cuvântul cheie `async`:

```
async function asyncFunction() {
  // cod
  return valoare // va fi împachetată automat într-un Promise
}
```

O funcție async va returna întotdeauna un Promise. Dacă funcția returnează o valoare, Promise-ul va fi rezolvat cu acea valoare. Dacă funcția aruncă o eroare, Promise-ul va fi respins cu acea eroare.

```
async function greet() {
  return 'Salut, lume!'
}

greet().then((message) => {
  console.log(message) // "Salut, lume!"
})
```

Operatorul Await

Operatorul `await` poate fi folosit doar în interiorul unei funcții async și face ca JavaScript să aștepte până când Promise-ul este rezolvat sau respins:

```
async function asyncFunction() {
  const result = await promise
  // codul continuă doar după ce promisiunea este rezolvată
  return result
}
```

Iată un exemplu concret:

```
function waitSeconds(seconds) {
  return new Promise((resolve) => {
    setTimeout(() => {
      resolve(`Au trecut ${seconds} secunde`)
    }, seconds * 1000)
  })
}

async function executeSequential() {
  console.log('Început')

  const result1 = await waitSeconds(2)
  console.log(result1)

  const result2 = await waitSeconds(1)
  console.log(result2)

  console.log('Sfârșit')
}

executeSequential()
```

Output:

```
Început
Au trecut 2 secunde
Au trecut 1 secunde
Sfârșit
```

Convertirea Promises în Async/Await

Să vedem cum putem transforma un lanț de Promises în syntax async/await:

Cu Promises:

```
function getUserData(userId) {
  return getUser(userId)
    .then((user) => {
      return getUserOrders(user)
    })
    .then(({ user, orders }) => {
      return { user, orders }
    })
    .catch((error) => {
      console.error('A apărut o eroare:', error)
      throw error
    })
}
```

Cu Async/Await:

```
async function getUserData(userId) {
  try {
    const user = await getUser(userId)
    const orders = await getUserOrders(user)
    return { user, orders }
  } catch (error) {
    console.error('A apărut o eroare:', error)
    throw error
  }
}
```

Observați cum codul cu async/await este mai ușor de citit și de înțeles, deoarece urmează un flux mai natural, asemănător codului sincron.

Gestionarea Erorilor cu Try/Catch

Cu async/await, gestionăm erorile folosind blocuri try/catch familiare:

```
async function getUser(id) {
  try {
    // Simulăm o cerere asincronă care ar putea eșua
    const user = await getUserFromDatabase(id)
    return user
  } catch (error) {
    console.error('Eroare la preluarea utilizatorului:', error)
    // Poți alege să arunci din nou eroarea sau să returnezi o valoare implicită
    return { error: true, message: error.message }
  }
}
```

Execuția Paralelă cu Async/Await

Pentru a executa mai multe operațiuni asincrone în paralel, putem combina async/await cu `Promise.all()`:

```
async function fetchAllResources() {
  try {
    const [users, products, categories] = await Promise.all([
      getUsers(),
      getProducts(),
      getCategories(),
    ])

    console.log('Utilizatori:', users)
    console.log('Produse:', products)
    console.log('Categorii:', categories)

    return { users, products, categories }
  } catch (error) {
    console.error('A apărut o eroare:', error)
    throw error
  }
}
```

Modele de Implementare Practică

Crearea Unei Secvențe de Operațiuni Asincrone

```
// Simularea unor operaţiuni asincrone
function authenticate(credentials) {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      if (credentials.username === 'admin' && credentials.password === 'pass') {
        resolve({ token: 'jwt_token_example', userId: 123 })
      } else {
        reject(new Error('Autentificare eşuată'))
      }
    }, 1000)
  })
}

function getUserDetails(userId) {
  return new Promise((resolve) => {
    setTimeout(() => {
      resolve({
        id: userId,
        name: 'John Doe',
        email: 'john@example.com',
      })
    }, 1000)
  })
}

function getUserPermissions(user) {
  return new Promise((resolve) => {
    setTimeout(() => {
      resolve({
        userId: user.id,
        permissions: ['read', 'write', 'delete'],
      })
    }, 1000)
  })
}

// Implementare cu async/await
async function initializeUserSession(credentials) {
  try {
    console.log('Iniţializare sesiune...')

    const authData = await authenticate(credentials)
    console.log('Autentificare reuşită:', authData)

    const user = await getUserDetails(authData.userId)
    console.log('Detalii utilizator:', user)

    const permissions = await getUserPermissions(user)
    console.log('Permisiuni utilizator:', permissions)

    return {
      user,
      permissions,
      token: authData.token,
    }
  } catch (error) {
    console.error('Eroare la iniţializarea sesiunii:', error)
    throw error
  }
}

// Utilizare
initializeUserSession({ username: 'admin', password: 'pass' })
  .then((session) => {
    console.log('Sesiune iniţializată cu succes:', session)
  })
  .catch((error) => {
    console.error('Nu s-a putut iniţializa sesiunea:', error.message)
  })
```

Implementarea Unui Sistem de Retry

```javascript
async function retryOperation(operation, maxRetries = 3, delay = 1000) {
  let lastError

  for (let attempt = 1; attempt <= maxRetries; attempt++) {
    try {
      return await operation()
    } catch (error) {
      console.warn(`Încercarea ${attempt} a eșuat: ${error.message}`)
      lastError = error

      if (attempt < maxRetries) {
        console.log(`Așteptare ${delay}ms înainte de reîncercare...`)
        await new Promise((resolve) => setTimeout(resolve, delay))
        // Creștem timpul de așteptare pentru următoarea încercare (backoff exponențial)
        delay *= 2
      }
    }
  }

  throw new Error(
    `Operațiunea a eșuat după ${maxRetries} încercări: ${lastError.message}`
  )
}

// Exemplu de utilizare
async function unstableOperation() {
  // Simulăm o operațiune care eșuează aleatoriu
  const random = Math.random()
  if (random < 0.7) {
    throw new Error('Operațiune eșuată din cauza unei erori aleatorii')
  }
  return 'Operațiune reușită!'
}

async function testRetry() {
  try {
    const result = await retryOperation(unstableOperation)
    console.log('Rezultat final:', result)
  } catch (error) {
    console.error('Eroare finală:', error.message)
  }
}

testRetry()
```