JavaScript - Promises, Async-Await, Requests

Async vs Sync

- Async vs Sync

```
const fs = require('fs')
const content = 'Logging to a file'
try {
    fs.writeFileSync('test.txt', content)
    console.log('logs completed')
} catch (err) {
    throw err
}
```

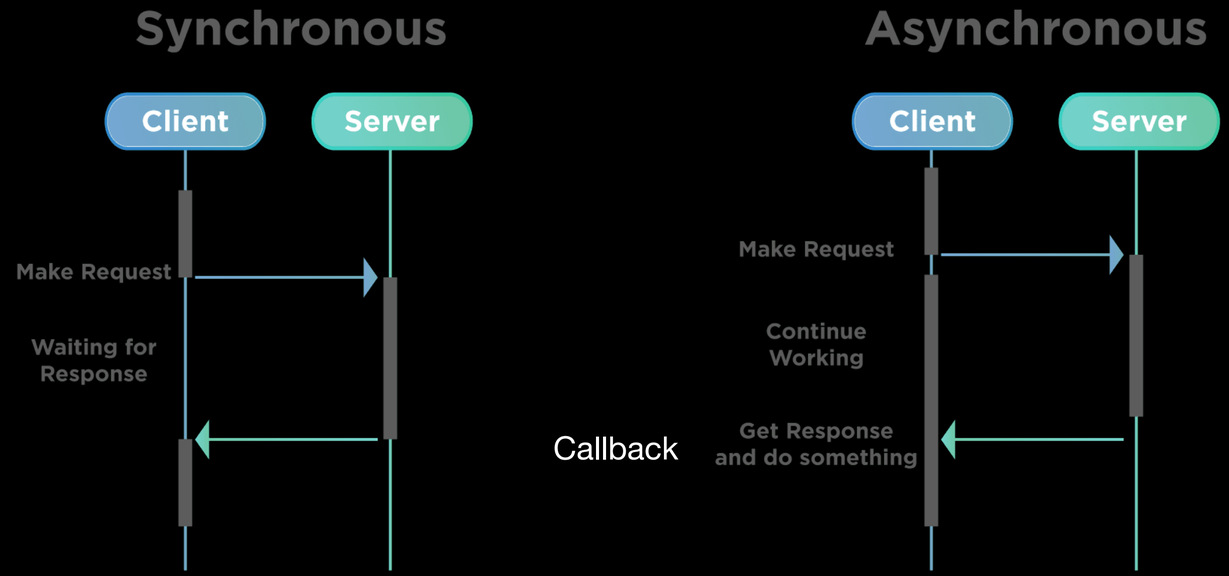
```
const fs = require('fs')
const content = 'Logging to a file'
fs.writeFile('test.txt', (content, err) => {
    if (err) {
        throw err
    }
    console.log('logs completed')
})
```

Callback

- Fundera: Vad är skillnaden mellan dessa i ”flödet”?

Async vs Sync

- *IO* funktioner: http-request, databas anrop, filöppning, timer, DOM API, etc…
- *Synchronous (blocking)* vs *Asynchronous (non-blocking)*



- Fundera: Vad är fördelen med asynkron funktion?

Async vs Sync
setTimeout

- Demo
  - Fundera: vad kommer hända?

```
const callback = () => {
  console.log("callback called");
};

const main = () => {
  console.log("first line")
  setTimeout(callback, 1000)
  console.log("last line")
}
```

Async vs Sync

- *Callback hell*

```
chooseToppings(function(toppings) {
  placeOrder(toppings, function(order) {
    collectOrder(order, function(pizza) {
      eatPizza(pizza)
    }, failureCallback)
  }, failureCallback)
}, failureCallback)
```

Async vs Sync
Callback hell

- Demo
  - Fundera: vad kommer hända?

```
const main = () => {
  console.log("1 line")
  setTimeout(() => {
    console.log("2 line")
    setTimeout(() => {
      console.log("3 line")
      setTimeout(() => {
        console.log("4 line")
      }, 2000)
    }, 3000)
  }, 4000)
  console.log("5 line")
}
```

Promise

- Promise - Ett löfte om något som ska göras *asynkront* (ska göras i framtiden); Löftet kan:
  - *resolvas* (löftet blir uppfyllt)
  - *rejectas* (löftet blir brutet)

Promise

- Promise skapades för att lösa *callback hell* (ES6)

```
chooseToppings(function(toppings) {
  placeOrder(toppings, function(order) {
    collectOrder(order, function(pizza) {
      eatPizza(pizza)
    }, failureCallback)
  }, failureCallback)
}, failureCallback)
```

```
chooseToppings()
.then(toppings => placeOrder(toppings))
.then(order => collectOrder(order))
.then(pizza => eatPizza(pizza))
.catch(failureCallback)
```

Promise

- Promise en inbyggd klass
  - Konstruktorn ges *executor* som tar
    - *resolve* - löftet uppfylls
    - *reject* - löftet bryts
  - När konstruktor kallas kör JS motorn *executor* med sin egna *resolve* och *reject*
  - *resolve*/*reject kan döpas om till något annat*

```
const executorFunction = (resolve, reject) => {
  if (someCondition) {
      resolve('I resolved!');
  } else {
      reject('I rejected!');
  }
}
const myFirstPromise = new Promise(executorFunction);
```

Promise

- Promise *resolvas* eller *rejectas* baserat på someCondition (*sync*)
  - I verkligheten baseras det på resultat av asynkrona operationer

```
const executorFunction = (resolve, reject) => {
  if (someCondition) {
      resolve('I resolved!');
  } else {
      reject('I rejected!');
  }
}
const myFirstPromise = new Promise(executorFunction);
```

Promise

- Demo - *synkron* för enkelhetens skull

```
const inventory = {
  sunglasses: 1200,
  pants: 1088,
  bags: 1344
};

const myExecutor = (resolve, reject) => {
  if (inventory.sunglasses > 0) {
    resolve('Sunglasses order processed.')
  } else {
    reject('That item is sold out.')
  }
}

const orderSunglasses = () => {
  return new Promise(myExecutor)
}

const orderPromise = orderSunglasses()
console.log(orderPromise)
```

Promise

then

- then - metod i Promise som tar 2 callbacks
  - handleSuccess - konsumerar *resolved* resultat
  - handleFailure - konsumerar *rejected* resultat
  - *handleSuccess*/*handleFailure kan döpas om*

```
let prom = new Promise((resolve, reject) => {
  let num = Math.random();
  if (num < .5 ){
    resolve('Yay!');
  } else {
    reject('Ohhh noooo!');
  }
});

const handleSuccess = (resolvedValue) => {
  console.log(resolvedValue);
};

const handleFailure = (rejectionReason) => {
  console.log(rejectionReason);
};

prom.then(handleSuccess, handleFailure);
```

Promise

- En funktion som tar 2 argument och bara använder första, kan skippa andra

```
const test = function(name, helloMsg, goodbyeMsg) {
  console.log(helloMsg + " " + name);
}

test("Patrik", "Hello", "Goodby") // Hello Patrik
test("Patrik", "Hello") // Hello Patrik
test("Patrik") // undefined Patrik
```

```
const promise = new Promise((resolve, reject) => resolve("Yay!"))
promise.then(value => console.log(value)) // Yay!

const promise = new Promise(resolve => resolve("Yay!"))
promise.then(value => console.log(value)) // Yay!
```

Promise
then

- Demo - nu asynkron för svårighetens skull

```
const test = num => {
  const promise = new Promise(resolve => setTimeout(() => resolve(num), 4000))
    .then(val => new Promise(resolve => setTimeout(() => resolve(val + 1), 3000)))
    .then(val => new Promise(resolve => setTimeout(() => resolve(val + 1), 2000)))
  return promise
}

const prom = test(1)
prom.then(val => console.log(val))
```

- Fundera: Vad loggas på sista raden, och när?

Promise

then

- Demo - bygg async API

```
const asyncAdd = num =>
  new Promise(resolve => setTimeout(() => resolve(num + 1), 1000))

const asyncSub = num =>
  new Promise(resolve => setTimeout(() => resolve(num - 1), 1000))
```

Promise
catch

- catch - metod i Promise som tar 1 callback
  - onRejected - konsumerar *rejected* resultat

```
prom
  .then((resolvedValue) => {
    console.log(resolvedValue);
  })
  .catch((rejectionReason) => {
    console.log(rejectionReason);
  });
```

Promises
catch

- Demo - build async API

```
const asyncAdd = num =>
  new Promise((resolve, reject) => {
    if(num > 5) {
      reject(`to big value: ${num}`)
    } else {
      setTimeout(() => resolve(num + 1), 1000)
    }
  })

const asyncSub = num =>
  new Promise((resolve, reject) => {
    if(num < 0) {
      reject(`to small value: ${num}`)
    } else {
      setTimeout(() => resolve(num - 1), 1000)
    }
  })
```

Async-Await

- ES8 introducerade async … await
  - Promise avhjälper *callback hell*
  - async … await avhjälper det *asynchrona utseendet*!

Async-Await

- Det viktiga är async deklarationen

```
const giveHelloIn5Sec = async () =>
  ...
}

async function giveHelloIn5Sec() {
  ...
}

const giveHelloIn5Sec = async function() {
  ....
}
```

Async-Await

- Demo - en async function returnerar
  - Om inget anges - en Promise som resolvar undefined
  - Om ett vanligt värde - en Promise som resolvar till det värdet
  - Om en Promise - samma Promise

Async-Await

- Nu kan vi förenkla vår Promise baserade kod

```
function withConstructor(num) {
  return new Promise((resolve, reject) => {
    if (num === 0) {
      reject("Error: Zero")
    } else {
      resolve("Not zero!")
    }
  })
}
```

```
async function withAsync(num) {
  if (num === 0) {
    throw Error("Error: Zero")
  } else {
    return "Not zero!"
  }
}
```

Async-Await
await

- async funktioners kraft kommer från await
  - await - kan endast användas i en async function
  - await - returnerar det resolvade värdet från Promise
  - await - stoppar async funktionens exekvering till den resolvas (non-blocking)

Async-Await
await

- Demo - async funktioners kraft kommer från await

```
const asyncAdd = async (num) =>
  new Promise(resolve => {
    setTimeout(() => resolve(num + 10), 1000)
  })

const asyncSub = async (num) =>
  new Promise(resolve => {
    setTimeout(() => resolve(num - 5), 1000)
  })

const main = async (num) => {
  const x = await asyncAdd(num)
  const y = await asyncSub(x)
  return y;
}
```

Async-Await
try catch

- Fångar både sync och async fel!

```
async function hostDinnerParty() {
  try {

  } catch (error) {

  }
}
```

Async-Await
try catch

- Demo

```
const asyncMul = async num =>
  new Promise((resolve, reject) => {
    if(num > 10) {
      reject(`to big value: ${num}`)
    } else {
      setTimeout(() => resolve(num * num), 1000)
    }
  })

const asyncDiv = async num =>
  new Promise((resolve, reject) => {
    if(num < 0) {
      reject(`to small value: ${num}`)
    } else {
      setTimeout(() => resolve(num / 10), 1000)
    }
  })

const tryCatchEx2 = async (num) => {
  try {
    const x = await asyncMul(num)
    const y = await asyncMul(x)
    const z = await asyncDiv(y)
    console.log(`successful computed value: ${z}`)
  } catch(err) {
    console.error(err)
  }
}
```

Requests
Query

- key-value par =

```
'https://api.datamuse.com/words?key=value'
```

- Flera parameterar &

```
'https://api.datamuse.com/words?key=value&anotherKey=anotherValue'
```

- Server tar emot parametrar och hämtar den data som efterfrågas

Requests
Ajax

- XMLHttpRequest (XHR) API

creates new object

```
const xhr = new XMLHttpRequest();
const url = 'http://api-to-call.com/endpoint';

xhr.responseType = 'json';
xhr.onreadystatechange = () => {
  if (xhr.readyState === XMLHttpRequest.DONE) {
      // Code to execute with response
    }
};

xhr.open('GET', url);
xhr.send();
```

handles response

opens request and sends object

Requests

Ajax

- Demo

```
const xhr = new XMLHttpRequest();
const url = 'https://api.datamuse.com/words?rel_rhy=test'
xhr.responseType = 'json'
xhr.onreadystatechange = () => {
  if (xhr.readyState === XMLHttpRequest.DONE) {
    console.log(xhr.response)
  }
}
xhr.open('GET', url)
xhr.send()
```

Requests
fetch

- fetch (W3C standard)

```
fetch('http://api-to-call.com/endpoint').then(response => {
  if (response.ok) {
    return response.json();
  }
  throw new Error('Request failed!');
}, networkError => console.log(networkError.message)
).then(jsonResponse => {
    // Code to execute with jsonResponse
});
```

sends request

converts response object to JSON

handles errors

handles success

Requests

fetch

- Demo

```
fetch('https://api.datamuse.com/words?rel_rhy=test')
.then(response => {
  if (response.ok) {
    return response.json()
  }
  throw new Error('Request failed!')
}, networkError => {
  console.log(networkError.message);
})
.then(jsonResponse => {
  console.log(jsonResponse)
})
```