

# Asynchronicity with Promises

Alex Thalhammer

# Inhalt


- Http access
- DEMO: First solution
- Promises
- DEMO: Better solution



ANGULAR  
**ARCHITECTS**  
INSIDE KNOWLEDGE



SOFTWARE  
**ARCHITECT**

A vintage black rotary telephone is the central focus, with its handset resting on top. The rotary dial is clearly visible, showing numbers 1 through 0. In the foreground, a black fountain pen with gold-colored accents lies horizontally on a piece of paper with handwritten notes. The background is a warm, yellowish glow, suggesting a desk lamp. A semi-transparent dark rectangle is overlaid on the middle of the image to contain the text.

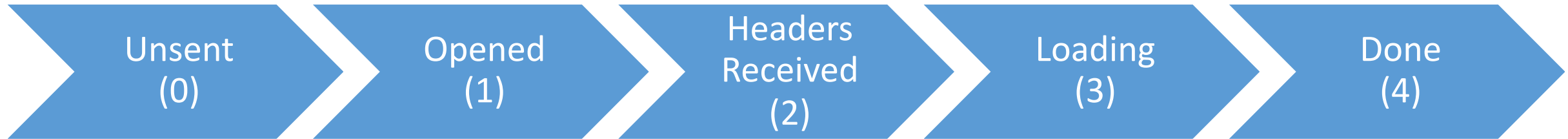
HTTP access as example for  
asynchronous operations



XHR

XmlHttpRequest
onreadystatechange readyState responseText responseXML status statusText
open() send() abort() setRequestHeaders() getResponseHeaders()

# Readystate



ANGULAR  
**ARCHITECTS**  
INSIDE KNOWLEDGE



SOFTWARE  
**ARCHITECT**

# XHR

```
function loadData() {  
    var xmlhttp;  
    xmlhttp = new XMLHttpRequest();  
  
    [...]  
}
```



ANGULAR  
**ARCHITECTS**  
INSIDE KNOWLEDGE



SOFTWARE  
**ARCHITECT**

# XHR

```
function loadData() {  
    var xmlhttp;  
    xmlhttp = new XMLHttpRequest();  
  
    xmlhttp.onreadystatechange = function() {  
        if (this.readyState == 4 && this.status == 200) {  
            console.debug(this.responseText);  
        }  
    };  
  
    [...]  
}
```



ANGULAR  
**ARCHITECTS**  
INSIDE KNOWLEDGE



SOFTWARE  
**ARCHITECT**

# XHR

```
function loadData() {  
    var xmlhttp;  
    xmlhttp = new XMLHttpRequest();  
  
    xmlhttp.onreadystatechange = function() {  
        if (this.readyState == 4 && this.status == 200) {  
            console.debug(this.responseText);  
        }  
    };  
  
    xmlhttp.open("GET", "http://...", true);  
    xmlhttp.send();  
}
```



ANGULAR  
**ARCHITECTS**  
INSIDE KNOWLEDGE



SOFTWARE  
**ARCHITECT**



# DEMO



Promises

# Promises

- Defines something that is not known (yet)
- States:
  - pending
  - fullfiled
  - rejected



# Promises

```
function loadData() {  
    return new Promise((resolve, reject) => {  
        //Async:  
        resolve(4711);  
        //reject("err!");  
    });  
}
```



# Promises

```
function loadData() {  
  return new Promise((resolve, reject) => {  
    //Async:  
    resolve(4711);  
    //reject("err!");  
  });  
}
```

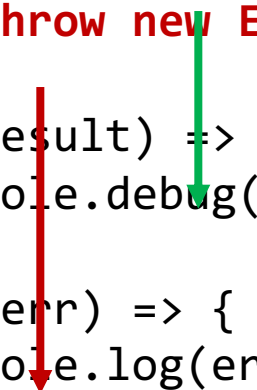
**Excuted immediately!**

```
loadData()  
  .then((result) => {  
    console.log(result);  
  })  
  .catch((err) => {  
    console.log(err);  
  });
```



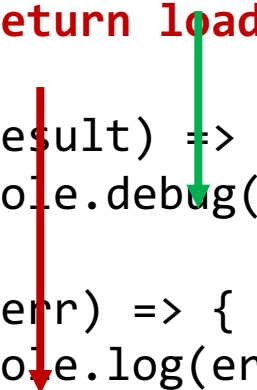
# Promise Chaining

```
loadData()  
  .then((result) => {  
    return 'XYZ';  
    // throw new Error();  
  })  
  .then((result) => {  
    console.debug(result == 'XYZ');  
  })  
  .catch((err) => {  
    console.log(err);  
  });
```



# Promise Chaining

```
loadData()  
  .then((result) => {  
    return loadData(...); // success  
    // return loadData(...); // no success  
  })  
  .then((result) => {  
    console.debug(result == 'XYZ');  
  })  
  .catch((err) => {  
    console.log(err);  
  });
```



# DEMO





# Alternative: Observables

We will take a closer look at observables in RxJS inputs / labs