HOCHSCHULE FULDA
UNIVERSITY OF APPLIED SCIENCES

WEB-APPLIKATIONEN

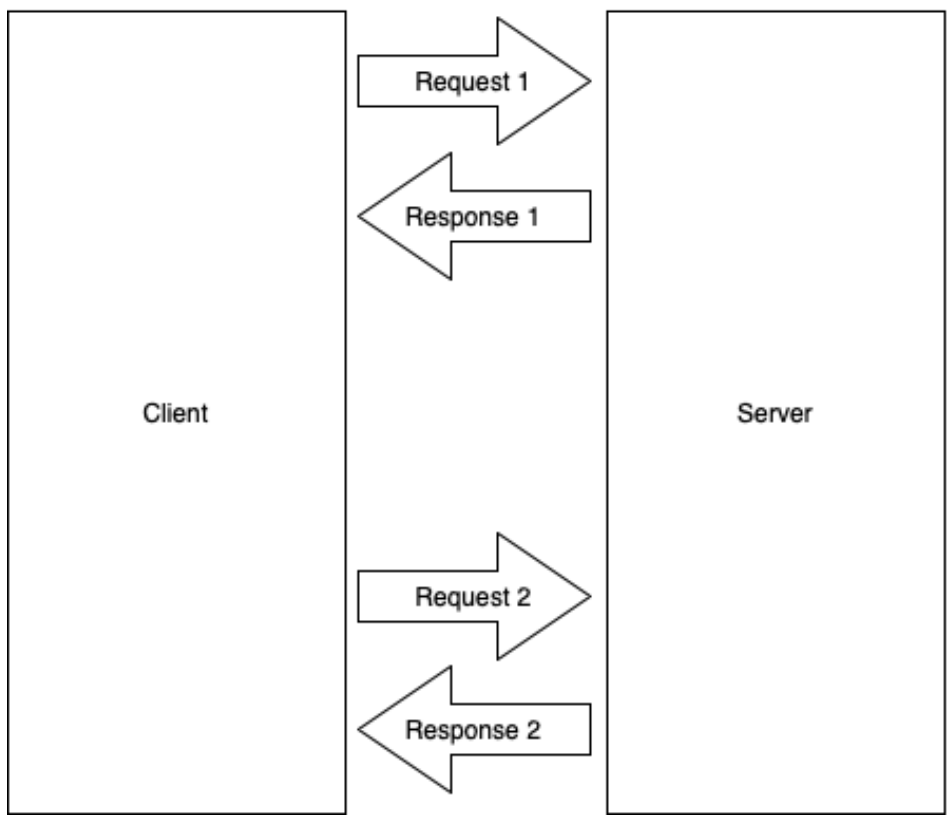
ASYNCHRONE REQUESTS

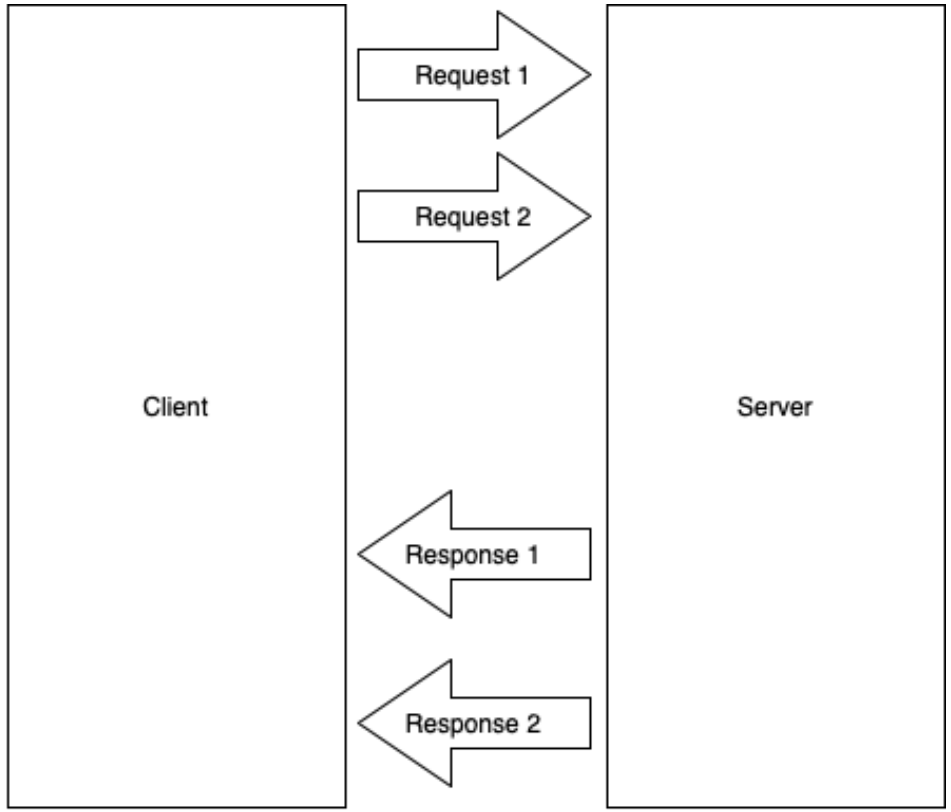
AI
Manuel Richardt, Dominik Rupprecht
FB **ANGEWANDTE INFORMATIK**

SYNCHRONE UND ASYNCHRONE REQUESTS

1. SYNCHRONE REQUESTS
   - Beziehen sich auf Echtzeitkommunikation, bei der jede Nachricht sofort empfangen und ggf. verarbeitet und beantwortet wird.
   - Beispiel: Telefon
2. ASYNCHRONE REQUESTS
   - Ein Programm kann Informationen von einer anderen Software anfordern und während des Wartens auf eine Antwort andere Dinge tun.
   - Beispiel: E-Mail

KOMMUNIKATION ZWISCHEN CLIENT UND SERVER





Manuel Richardt, Dominik Rupprecht
AI FB **ANGEWANDTE INFORMATIK**

XMLHTTPREQUEST

1. WURDE VON MICROSOFT (1998) ENTWICKELT UND VON MOZILLA, APPLE UND GOOGLE ÜBERNOMMEN
2. IM W3C STANDARDISIERT
3. UNTERSTÜTZT NICHT NUR HTTP PROTOKOLLE
4. KANN JEDE ART VON DATEN ENTHALTEN (NICHT NUR XML)
5. ASYNCHRONER REQUEST UND RESPONSE
   - Wenn Server antwortet werden die entsprechenden Daten im Client angezeigt

ALTERNATIVEN

1. AJAX (AB 2008)
2. FETCH (AB 2015)
3. AXIOS (AB 2017)

FETCH

1. FETCH IST DIE „NEUERE“ (2015) UND „EINFACHERE“ FORM VON XMLHTTPREQUEST
2. GENERISCHE DEFINITION VON REQUEST UND RESPONSE OBJEKTEN
3. UNTERSCHEIDUNG ZWISCHEN FETCH AUF CLIENT- ODER FETCH AUF SERVER-SEITE
4. ÜBER FETCH KÖNNEN CRUD ANFRAGEN DES CLIENTS REALISIERT WERDEN

ASYNC / AWAIT

- Funktion muss als async gekennzeichnet werden
- await zwingt JS auf eine Antwort zu warten bevor weiterer Code ausgeführt wird

```
 8  router.get('/await', async function(req, res, next) {
 9    console.log('1');
10    const response = await fetch('http://localhost:3000/users')
11    console.log('2');
12    const users = await response.json();
13    console.log('3');
14    res.render('index', { title: 'Express' });
15  });
```

PROBLEME AUSGABE DEBUGGING-KONSOLE TERMINAL PORTS

```
[nodemon] starting `node ./bin/www`
1
2
3
GET /await 200 45.701 ms - 218
GET /stylesheets/style.css 200 1.650 ms - 111
```

AI
Manuel Richardt, Dominik Rupprecht
FB **ANGEWANDTE INFORMATIK**

THEN

- then-Funktion wird ausgelöst, wenn Antwort erhalten wurde
- Code außerhalb der then-Funktion wird ggf. zuvor durchlaufen

```
16  router.get('/then', function(req, res, next) {
17    console.log('1');
18    fetch('http://localhost:3000/users')
19    .then((response) => {
20    console.log('2');
21      response.json()
22      .then ((users) => {
23        console.log('3');
24      })
25    })
26    console.log('4');
27    res.render('index', { title: 'Express' });
```

PROBLEME AUSGABE DEBUGGING-KONSOLE TERMINAL F

```
GET /stylesheets/style.css 304 1.027 ms - -
1
4
GET /then 200 3.604 ms - 218
2
3
```

FETCH (GET)

› Beispiel: Server fragt API eines anderen Servers an

› Der Server unterstützt „fetch“ standardmäßig nicht, daher muss zusätzliche Middleware installiert werden

```
npm install node-fetch
```

› Beispielcode in einer Routen-Datei:

› Sendet über API eine Anfrage an den Server.

› Wenn dieser Antwortet wird dessen Antwort (value) an den Browser gesendet

```
const fetch = (...args) => import('node-fetch')
  .then(({default: fetch}) => fetch(...args));

router.get('/test', function(req, res, next) {
  const url = 'https://api.chucknorris.io/jokes/random';
  fetch(url, {
    method: 'get',
    headers: {'Content-Type': 'application/json'}
  })
  .then(res => res.json())
  .then(response => res.send(response.value))
})
```

POST REQUEST IN FETCH

- Daten
- HTTP-Methode
- Body
- Headers
- Response

```
<script>
  let todo = {
    userId: 123,
    title: "loren impsum doloris",
    completed: false
  }

  fetch('https://jsonplaceholder.typicode.com/todos', {
    method: 'POST',
    body: JSON.stringify(todo),
    headers: { 'Content-Type': 'application/json' }
  }).then(res => res.json())
    .then(json => console.log(json))
    .catch(err => console.log(err));
</script>
```

AI
Manuel Richardt, Dominik Rupprecht
FB **ANGEWANDTE INFORMATIK**

PUT REQUEST IN FETCH

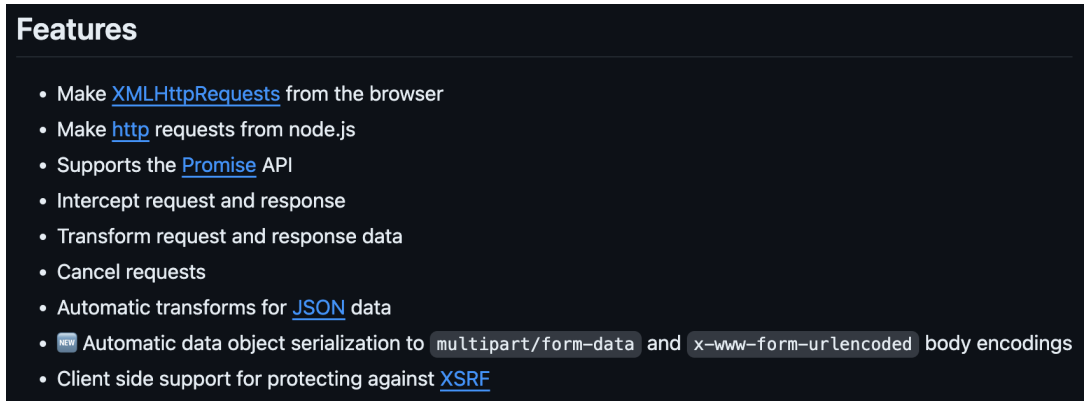
```
const someData = {
  title: document.querySelector(TitleInput).value,
  body: document.querySelector(BodyInput).value
}
fetch('https://jsonplaceholder.typicode.com/todos', {
  method: 'PUT',
  headers: { 'Content-type': 'application/json; charset=UTF-8' },
  body: JSON.stringify(someData)
})
  .then(response => response.json())
  .then(data => console.log(data))
  .catch(err => console.log(err))
```

AI
Manuel Richardt, Dominik Rupprecht
FB **ANGEWANDTE INFORMATIK**

DELETE REQUEST IN FETCH

```
fetch('https://jsonplaceholder.typicode.com/todos/1', {
  method: 'DELETE',
  headers: { 'Content-type': 'application/json; charset=UTF-8' }
})
  .then(response => response.json())
  .then(data => console.log(data))
  .catch(err => console.log(err))
```

AXIOS (GET)



› Beispiel: Server fragt API eines anderen Servers an

› Der Server unterstützt „axios“ standardmäßig nicht, daher muss zusätzliche Middleware installiert werden

```
npm install axios
```

› Beispielcode in einer Routen-Datei:

› Sendet über API eine Anfrage an den Server.

› Wenn dieser Antwortet wird dessen Antwort (response) an den Browser gesendet

```
const axios = require('axios');

/* GET users listing. */
router.get('/axios', function(req, res, next) {
  axios.get('http://localhost:3000/users')
        .then(response => {
            const users = response.data;
            res.render('blog', { title:"test", data:users });
        })
        .catch(error => {
            res.render('error');
        });
});
```

Manuel Richardt, Dominik Rupprecht
AI FB ANGEWANDTE INFORMATIK

AXIOS POST

```
router.post('/axios', function(req,res,next) {
    axios.post('http://localhost:3000/users', req.body)
      .then(function (response) {
        res.status(response.status).json(response.data);
      })
      .catch(function (error) {
        res.status(error.response.status).json(error.response.data);
      });
});
```

AXIOS PUT

```
router.put('/axios/:id', function(req,res,next) {
    axios.put('http://localhost:3000/users/'+req.params.id, req.body)
      .then(function (response) {
        res.status(response.status).json(response.data);
      })
      .catch(function (error) {
        res.status(error.response.status).json(error.response.data);
      });
});
```

AI
Manuel Richardt, Dominik Rupprecht
FB **ANGEWANDTE INFORMATIK**

AXIOS DELETE

```
router.delete('/axios/:id', function(req,res,next) {
  axios.delete('http://localhost:3000/users/'+req.params.id)
    .then(function (response) {
      res.status(response.status).json(response.data);
    })
    .catch(function (error) {
      res.status(error.response.status).json(error.response.data);
    });
});
```

AI
Manuel Richardt, Dominik Rupprecht
FB **ANGEWANDTE INFORMATIK**

FETCH UND AXIOS IM BROWSER

› Fetch

```
fetch('https://official-joke-api.appspot.com/jokes/programming/random')
.then(res => res.json())
.then(result => console.log(result))
```

› Axios

```
<script src="https://unpkg.com/axios/dist/axios.min.js"></script>
<script>
    axios.get('https://api.chucknorris.io/jokes/random')
  .then((response) => {
    document.getElementById("demo").innerHTML = response.data.value;
  });
</script>
```

AI
Manuel Richardt, Dominik Rupprecht
FB **ANGEWANDTE INFORMATIK**

ZUSAMMENFASSUNG

1. ASYNCHRONE KOMMUNIKATION ZWISCHEN CLIENT UND SERVER KANN ZUM NACHLADEN VON INHALTEN GENUTZT WERDEN OHNE DAS DER BENUTZER EINE INTERAKTION AUSFÜHREN MUSS.
2. ÜBER FETCH ODER AXIOS IST DAS ANSPRECHEN EINES RESTFUL WEBSERVICES INNERHALB EINES BROWSERS MÖGLICH.
3. FETCH UND AXIOS KÖNNEN SOWOHL IM CLIENT, ALS AUCH IM SERVER EINGEBUNDEN WERDEN.