WebSocket API in Stud.IP

Marcus Eibrink-Lunzenauer - elan e.V.

SET 26

Warum?

Warum?

- Und was wollen wir damit?

Warum?

- Und was wollen wir damit?
- Wenn wir das erst einmal haben, findet sich alles.

Warum?

- Und was wollen wir damit?
- Wenn wir das erst einmal haben, findet sich alles.
- Wieder mal typisch: Direkt über Technik sprechen, ohne vorher über UX zu sprechen.

Deshalb:

Deshalb:

- Grundlagen

Deshalb:

- Grundlagen
- Probleme

Deshalb:

- Grundlagen
- Probleme
- Ideen

Deshalb:

- Grundlagen
- Probleme
- Ideen
- Workshop

Ein kleiner Exkurs …

Göttingen 2008

(noch vor Stud.IP v2!)

Was ist eigentlich Stud.IP?

Organisation

Information

Stundenplan

Dateien

Kursmanagement

Daten für ...

… Räume

... Personen

... Veranstaltungen

aber nur die Dozenten: *Lernen*

Was soll Stud.IP werden?

Dozenten: „nichts anderes“

Studierende: ...

... Lernunterstützung

... offen

... **interaktiv**

... kommunikativ

... vernetzt

Ziel Interaktion

a.) Identität

ohne Identität keine Interaktion

Selbstdarstellung

b.) Vernetzung

Bsp. Lerngruppen

Veranstaltungen für alle

Bsp. Buddies

»Freundeliste«

Beziehungsnetzwerke

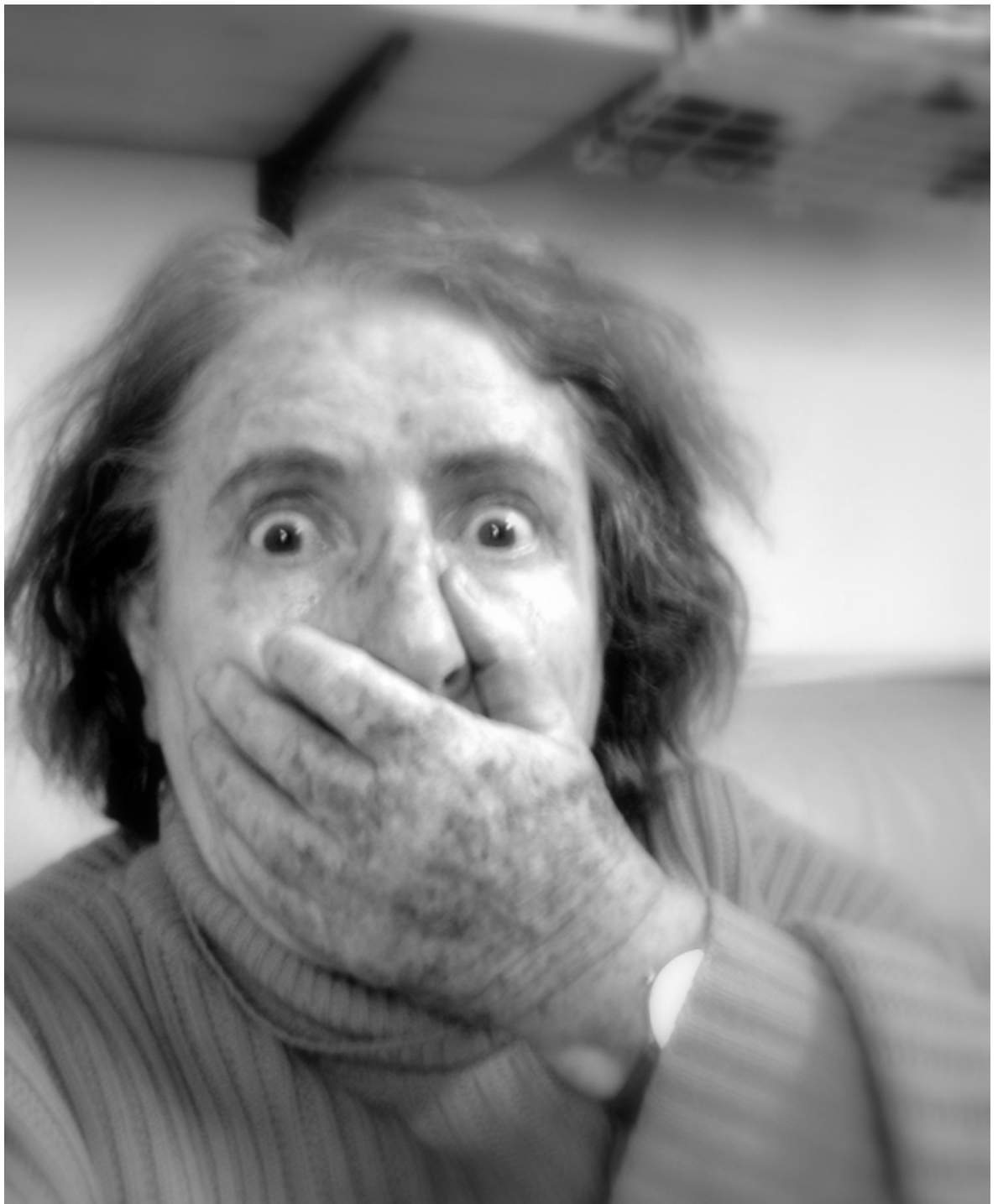
Beziehungspflege

c.) »presence«

Bsp. Präsenzinformation

Wer ist gerade da?

Was machen sie?



Warum überhaupt *WebSockets*

Warum überhaupt *Real-Time Updates*

„Der Server kann nicht von sich aus antworten, es sei denn, ich frage ihn zuerst.“

Technologien

Technologien

- WebSockets API

Technologien

- **WebSockets API**
- **Server-sent events (EventSource)**

Technologien

- **WebSockets API**
- **Server-sent events (EventSource)**
- Polling

Technologien

- **WebSockets API**
- **Server-sent events (EventSource)**
- Polling
- Long Polling

Technologien

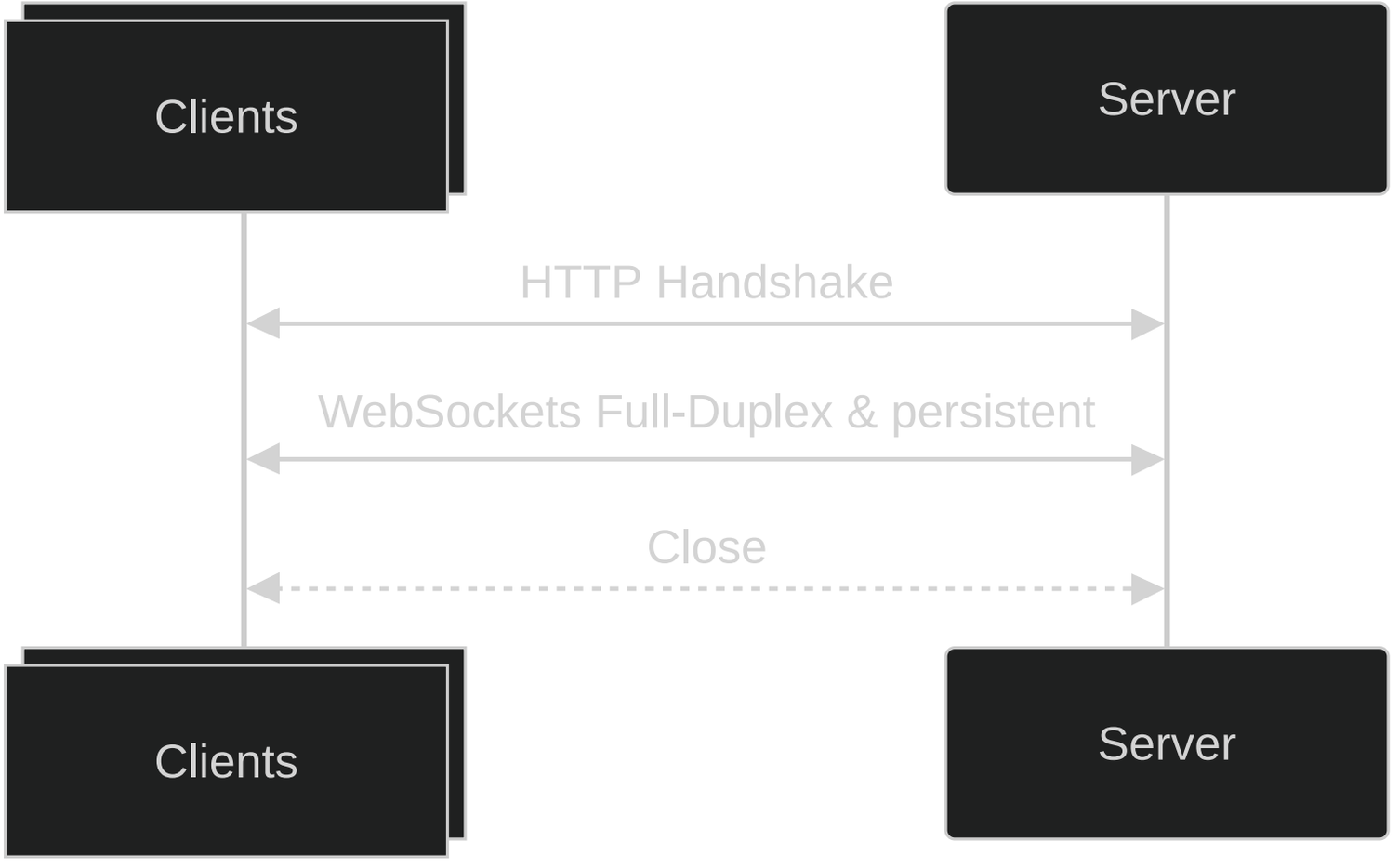
- **WebSockets API**
- **Server-sent events (EventSource)**
- Polling
- Long Polling
- WebRTC

WebSocket API

> *The WebSocket API makes it possible to open a two-way interactive communication session between the user's browser and a server. With this API, you can send messages to a server and receive responses without having to poll the server for a reply.*

https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API

Diagramm


Clients
Server
HTTP Handshake
WebSockets Full-Duplex & persistent
Close
Clients
Server

WebSocket Client

```
const uri = 'wss://studip.example.de';
const websocket = new WebSocket(uri);

// open event
websocket.addEventListener(
    'open',
    () => console.log('Connected.'),
);

// send messages
const message = { foo: 17 };
websocket.send(JSON.stringify(message));

// receive messages
websocket.addEventListener(
    'message',
    ({ data }) => console.log(`Received: ${data}`,
);
```

WebSocket Client

```
const uri = 'wss://studip.example.de';
const websocket = new WebSocket(uri);

// open event
websocket.addEventListener(
    'open',
    () => console.log('Connected.'),
);

// send messages
const message = { foo: 17 };
websocket.send(JSON.stringify(message));

// receive messages
websocket.addEventListener(
    'message',
    ({ data }) => console.log(`Received: ${data}`,
);
```

WebSocket Client

```
const uri = 'wss://studip.example.de';
const websocket = new WebSocket(uri);

// open event
websocket.addEventListener(
    'open',
    () => console.log('Connected.'),
);

// send messages
const message = { foo: 17 };
websocket.send(JSON.stringify(message));

// receive messages
websocket.addEventListener(
    'message',
    ({ data }) => console.log(`Received: ${data}`,
);
```

WebSocket Client

```
const uri = 'wss://studip.example.de';
const websocket = new WebSocket(uri);

// open event
websocket.addEventListener(
    'open',
    () => console.log('Connected.'),
);

// send messages
const message = { foo: 17 };
websocket.send(JSON.stringify(message));

// receive messages
websocket.addEventListener(
    'message',
    ({ data }) => console.log(`Received: ${data}`,
);
```

WebSocket Events

- open
- message
- error
- close
- window.pageshow
- window.pagehide

WebSocket Server

```
import { WebSocketServer } from 'ws';
import { createServer } from 'http';

const server = createServer();
const wss = new WebSocketServer({ server });

wss.on('connection', (ws) => {
  console.log('Client connected');

  ws.on('message', (message) => {
    const echoData = message.toString().slice(0, 100);
    ws.send(echoData);
  });

  ws.on('close', () => console.log('Client disconnected');
});

server.listen(8080);
```

WebSocket Server

```
1  import { WebSocketServer } from 'ws';
2  import { createServer } from 'http';
3
4  const server = createServer();
5  const wss = new WebSocketServer({ server });
6
7  wss.on('connection', (ws) => {
8    console.log('Client connected');
9
10   ws.on('message', (message) => {
11     const echoData = message.toString().slice(0, 100);
12     ws.send(echoData);
13   });
14
15   ws.on('close', () => console.log('Client disconnected'));
16 });
17
18 server.listen(8080);
```

WebSocket Server

```
import { WebSocketServer } from 'ws';
import { createServer } from 'http';

const server = createServer();
const wss = new WebSocketServer({ server });

wss.on('connection', (ws) => {
  console.log('Client connected');

  ws.on('message', (message) => {
    const echoData = message.toString().slice(0, 100);
    ws.send(echoData);
  });

  ws.on('close', () => console.log('Client disconnected');
});

server.listen(8080);
```

WebSocket Server

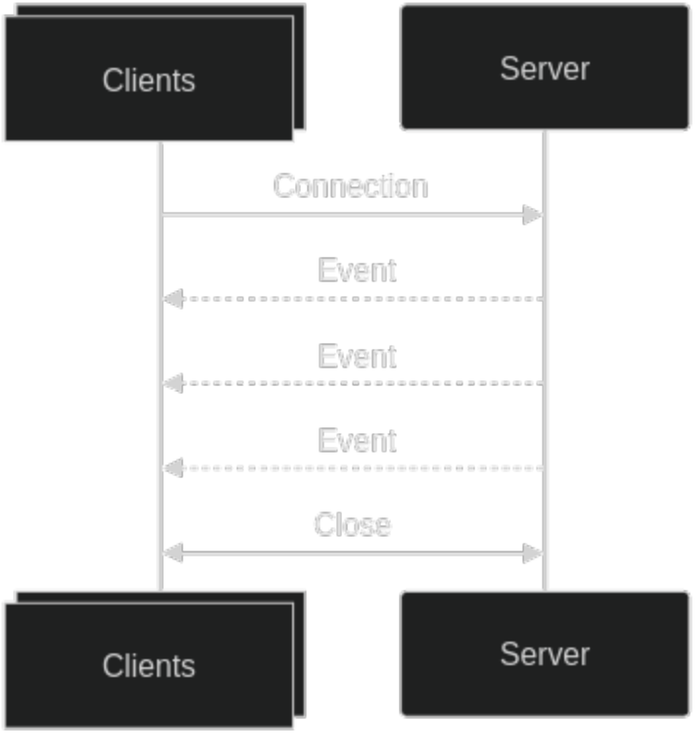
```
1  import { WebSocketServer } from 'ws';
2  import { createServer } from 'http';
3
4  const server = createServer();
5  const wss = new WebSocketServer({ server });
6
7  wss.on('connection', (ws) => {
8    console.log('Client connected');
9
10   ws.on('message', (message) => {
11     const echoData = message.toString().slice(0, 100);
12     ws.send(echoData);
13   });
14
15   ws.on('close', () => console.log('Client disconnected');
16 });
17
18 server.listen(8080);
```

EventSource API

An EventSource instance opens a persistent connection to an HTTP server, which sends events in text/event-stream format. The connection remains open until closed by calling EventSource.close().

https://developer.mozilla.org/en-US/docs/Web/API/EventSource

Diagramm


Clients
Server
Connection
Event
Event
Event
Close
Clients
Server

EventSource Client

```
const evtSource = new EventSource('sse-demo.php');

// Receive messages
evtSource.onmessage = ({ data }) => console.log(`Received: ${data}`);

// Receive custom messages
evtSource.addEventListener('ping', ({ data }) => console.log(`Got pinged: ${

// Handle errors
evtSource.onerror = (err) => console.error('EventSource failed:', err);

// Close connection
evtSource.close();
```

EventSource Client

```
const evtSource = new EventSource('sse-demo.php');

// Receive messages
evtSource.onmessage = ({ data }) => console.log(`Received: ${data}`);

// Receive custom messages
evtSource.addEventListener('ping', ({ data }) => console.log(`Got pinged: ${
)

// Handle errors
evtSource.onerror = (err) => console.error('EventSource failed:', err);

// Close connection
evtSource.close();
```

EventSource Client

```
const evtSource = new EventSource('sse-demo.php');

// Receive messages
evtSource.onmessage = ({ data }) => console.log(`Received: ${data}`);

// Receive custom messages
evtSource.addEventListener('ping', ({ data }) => console.log(`Got pinged: ${
)

// Handle errors
evtSource.onerror = (err) => console.error('EventSource failed:', err);

// Close connection
evtSource.close();
```

EventSource Client

```
const evtSource = new EventSource('sse-demo.php');

// Receive messages
evtSource.onmessage = ({ data }) => console.log(`Received: ${data}`);

// Receive custom messages
evtSource.addEventListener('ping', ({ data }) => console.log(`Got pinged: ${
// Handle errors
evtSource.onerror = (err) => console.error('EventSource failed:', err);

// Close connection
evtSource.close();
```

EventSource Client

```
const evtSource = new EventSource('sse-demo.php');

// Receive messages
evtSource.onmessage = ({ data }) => console.log(`Received: ${data}`);

// Receive custom messages
evtSource.addEventListener('ping', ({ data }) => console.log(`Got pinged: ${

// Handle errors
evtSource.onerror = (err) => console.error('EventSource failed:', err);

// Close connection
evtSource.close();
```

EventSource Server

```php
header("X-Accel-Buffering: no");
header("Content-Type: text/event-stream");
header("Cache-Control: no-cache");

while (true) {
    printf(
        'event: ping\ndata: {"time": "%s"}\n\n',
        date(DATE_ISO8601),
    );
    if (ob_get_contents()) {
        ob_end_flush();
    }
    flush();
    if (connection_aborted()) {
        break;
    }
    sleep(1);
}
```

EventSource Server

```php
header("X-Accel-Buffering: no");
header("Content-Type: text/event-stream");
header("Cache-Control: no-cache");

while (true) {
    printf(
        'event: ping\ndata: {"time": "%s"}\n\n',
        date(DATE_ISO8601),
    );
    if (ob_get_contents()) {
        ob_end_flush();
    }
    flush();
    if (connection_aborted()) {
        break;
    }
    sleep(1);
}
```

EventSource Server

```php
header("X-Accel-Buffering: no");
header("Content-Type: text/event-stream");
header("Cache-Control: no-cache");

while (true) {
    printf(
        'event: ping\ndata: {"time": "%s"}\n\n',
        date(DATE_ISO8601),
    );
    if (ob_get_contents()) {
        ob_end_flush();
    }
    flush();
    if (connection_aborted()) {
        break;
    }
    sleep(1);
}
```

EventSource Server

```php
header("X-Accel-Buffering: no");
header("Content-Type: text/event-stream");
header("Cache-Control: no-cache");

while (true) {
    printf(
        'event: ping\ndata: {"time": "%s"}\n\n',
        date(DATE_ISO8601),
    );
    if (ob_get_contents()) {
        ob_end_flush();
    }
    flush();
    if (connection_aborted()) {
        break;
    }
    sleep(1);
}
```

EventSource Server

```php
header("X-Accel-Buffering: no");
header("Content-Type: text/event-stream");
header("Cache-Control: no-cache");

while (true) {
    printf(
        'event: ping\ndata: {"time": "%s"}\n\n',
        date(DATE_ISO8601),
    );
    if (ob_get_contents()) {
        ob_end_flush();
    }
    flush();
    if (connection_aborted()) {
        break;
    }
    sleep(1);
}
```

Probleme

Technisch

Technisch

- Session Locking

Technisch

- Session Locking
- Memory Leaks

Technisch

- Session Locking
- Memory Leaks
- DB Connections

Technisch

- Session Locking
- Memory Leaks
- DB Connections
- Worker Process Exhaustion

Higher-Level Probleme

Higher-Level Probleme

- Events können nicht an Ort und Stelle ausgelöst werden.

Higher-Level Probleme

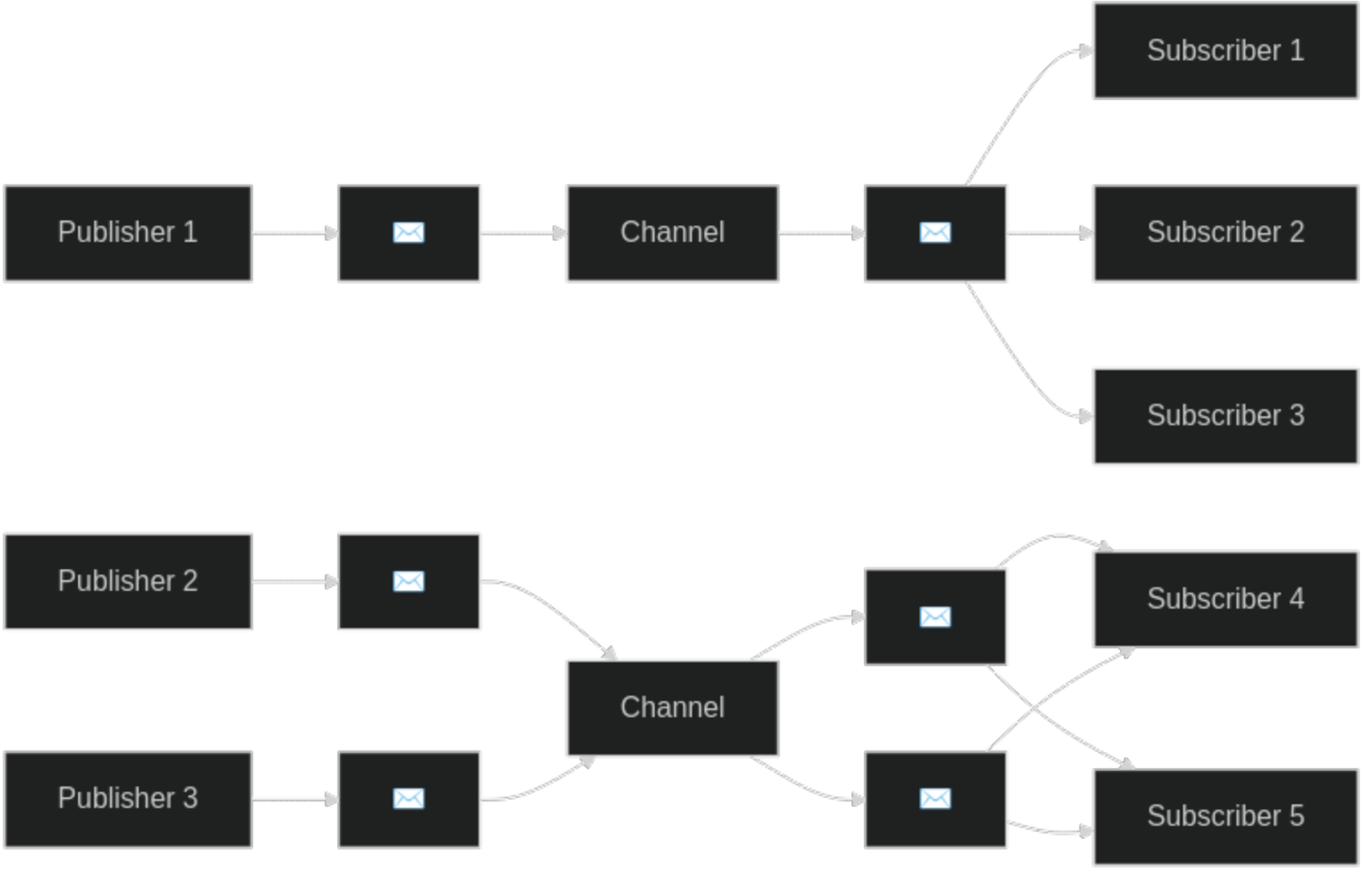
- Events können nicht an Ort und Stelle ausgelöst werden.
- Wie wir PHP einsetzen ist die Umsetzung technisch unmöglich.

Tech-Stack aufbohren?

Abstraktion einführen!

Pub/Sub

Idee


Publisher 1
Channel
Subscriber 1
Subscriber 2
Subscriber 3
Publisher 2
Publisher 3
Channel
Subscriber 4
Subscriber 5

Vorteile von Pub/Sub

Vorteile von Pub/Sub

- Lose Kopplung zwischen den Komponenten

Vorteile von Pub/Sub

- Lose Kopplung zwischen den Komponenten
- Hohe Skalierbarkeit

Vorteile von Pub/Sub

- Lose Kopplung zwischen den Komponenten
- Hohe Skalierbarkeit
- Sprach- und protokollunabhängig

Vorteile von Pub/Sub

- Lose Kopplung zwischen den Komponenten
- Hohe Skalierbarkeit
- Sprach- und protokollunabhängig
- Asynchrone, ereignisgesteuerte Kommunikation

Vorteile von Pub/Sub

- Lose Kopplung zwischen den Komponenten
- Hohe Skalierbarkeit
- Sprach- und protokollunabhängig
- Asynchrone, ereignisgesteuerte Kommunikation

Deshalb ist Pub/Sub ein **gutes Konzept**, um über die Web APIs zu abstrahieren.

Konkrete Beschränkungen

Konkrete Beschränkungen

- Mit Pub/Sub können die Web APIs in PHP handhabbar gemacht werden.

Konkrete Beschränkungen

- Mit Pub/Sub können die Web APIs in PHP handhabbar gemacht werden.
- Der PHP-Code verwendet dieselben Architekturkonzepte wie bisher.

Konkrete Beschränkungen

- Mit Pub/Sub können die Web APIs in PHP handhabbar gemacht werden.
- Der PHP-Code verwendet dieselben Architekturkonzepte wie bisher.
- Echtzeitbenachrichtigungen werden an den Broker geschickt, der die Kanäle zur Verfügung stellt. Das unterscheidet sich nicht von bisherigem Code.

Konkrete Beschränkungen

- Mit Pub/Sub können die Web APIs in PHP handhabbar gemacht werden.
- Der PHP-Code verwendet dieselben Architekturkonzepte wie bisher.
- Echtzeitbenachrichtigungen werden an den Broker geschickt, der die Kanäle zur Verfügung stellt. Das unterscheidet sich nicht von bisherigem Code.

Nachteil: Pub/Sub ist kein bidirektionales Werkzeug. Nachrichten fließen immer vom Publisher zum Subscriber.

Pub/Sub in Stud.IP

Was wird benötigt?

Software zum Publizieren von Nachrichten im PHP-Code des Stud.IP-Kern

Was wird benötigt?

Ein Tool, der Broker, dem die Publisher Nachrichten schicken und der diese an die Subscriber weiterleitet.

Was wird benötigt?

Software, die die Verbindung zum Broker aufnimmt und Channels abonniert

Was kann ich damit machen?

Data Broadcast

Data Sync

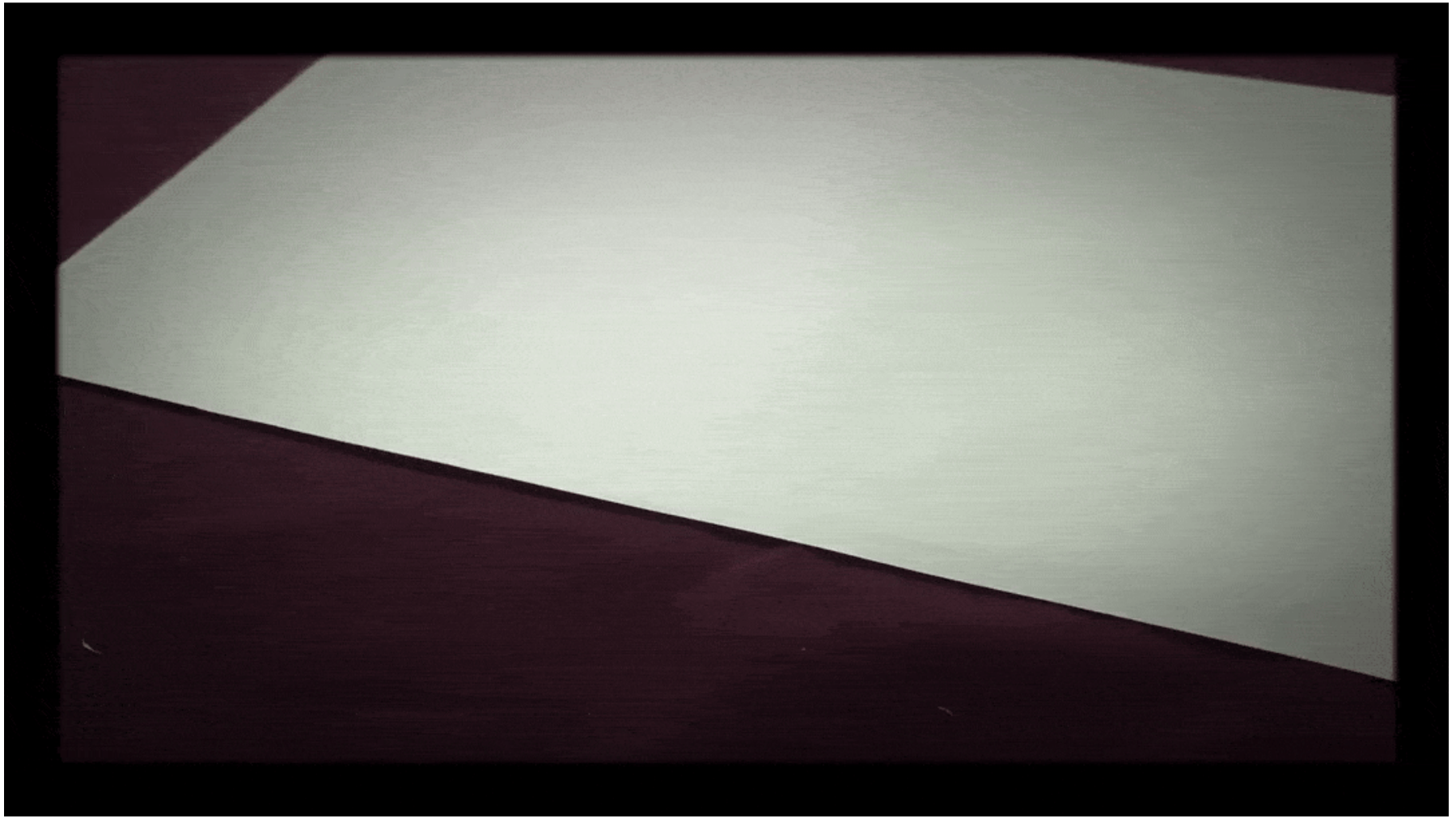
Presence

Collaboration

Synchrone Kommunikation

Alerts/Notifications

Workshop!



8 Ideen / 8 Minuten

Pair

Share

Vielen Dank