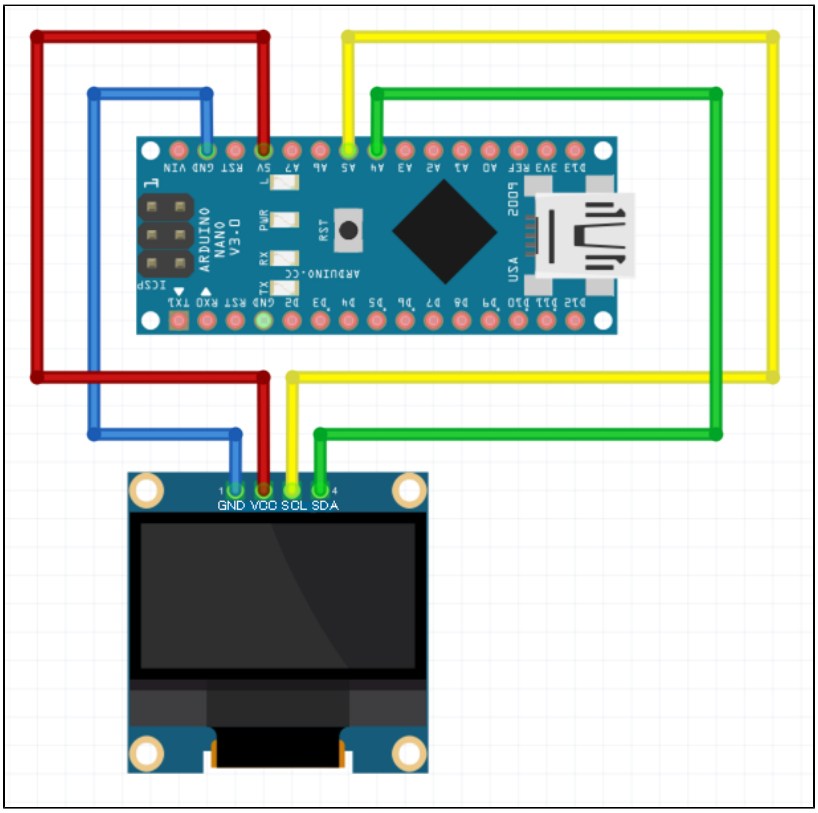
Praxissession 5 - IoT Session

IoT Session

> In der heutigen Session sollen Sie ein kleines "IoT-Device" zusammenbauen und mit einem Smart-Contract in der Ethereum Testkette Ropsten interagieren lassen.

Zusammenbau der Hardware

- Packen Sie den Arduino Nano und das kleine Oled-Display aus.
- Verkabeln Sie beide Geräte analog des Schaubildes.



Pin Display	Pin Arduino Nano
GND	GND
VCC	+5V
SCL	A5
SDA	A4

Updaten der Firmware

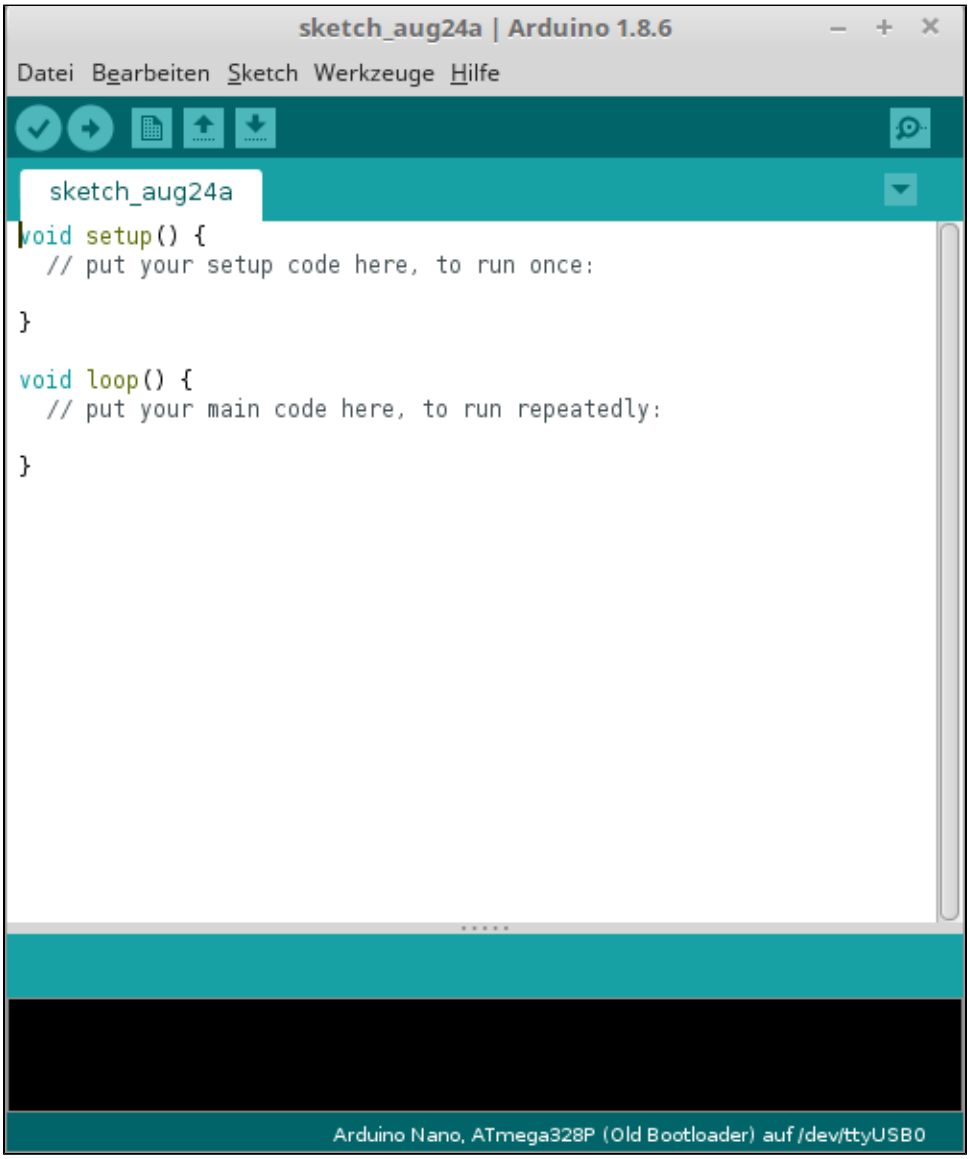
> Nach dem Zusammenbau wollen wir nun die Hardware erst einmal auf ihre Funktionsfähigkeit testen. Da wir den Nano mittels Javascript ansteuern wollen, müssen wir zu allererst eine neue Firmware auf den Nano hochladen.

- Verbinden Sie den Nano mit ihrem PC über das USB-Kabel.
- Öffnen Sie zum Starten der Arduino IDE ein neues Terminalfenster und tippen Sie folgenden Befehl ein.

Terminalfenster / Konsole

```
$ /./opt/arduino/current/arduino
```

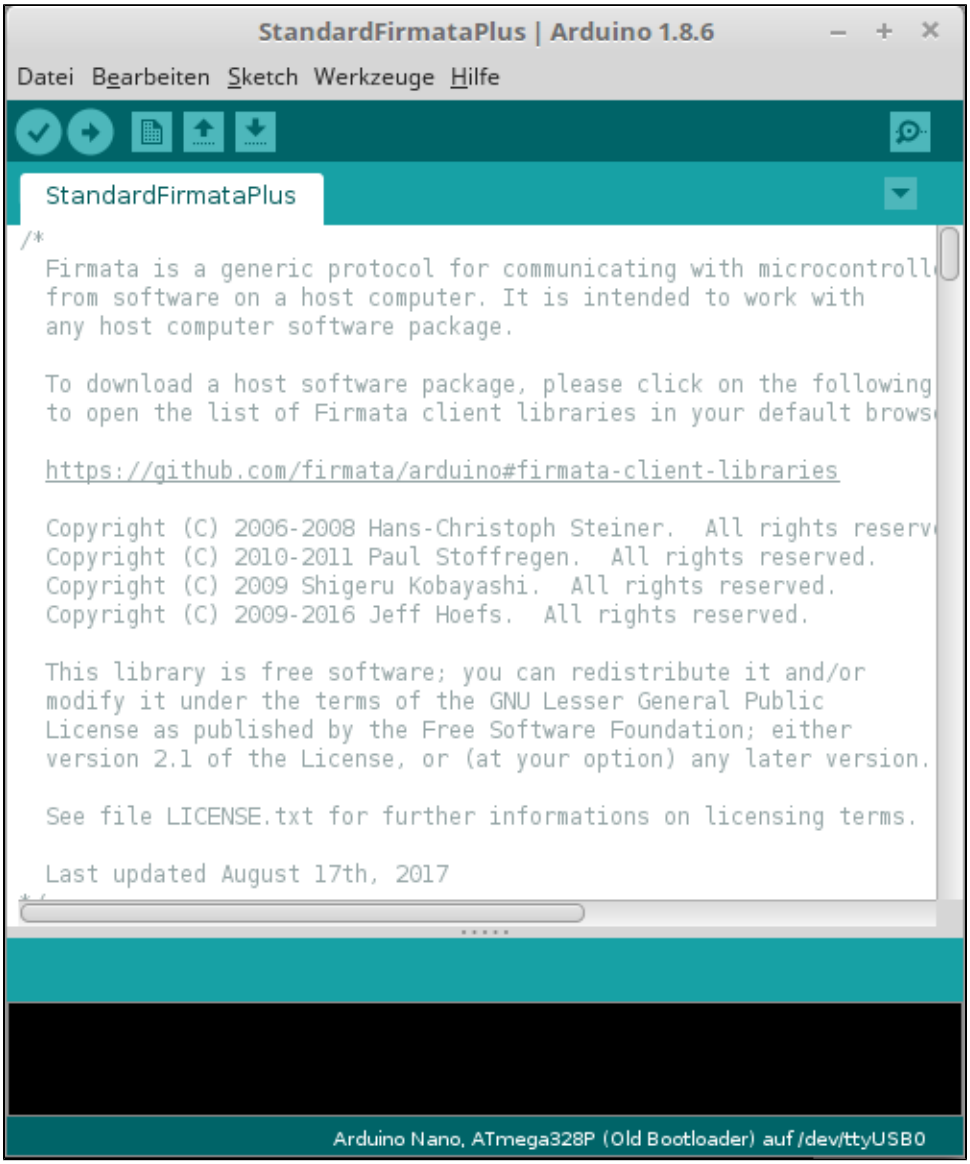
- Wählen Sie unter "**Werkzeuge-Board Arduino Nano**"
- Wählen Sie unter "**Werkzeuge-Prozessor ATMega328P (Old Bootloader)**"
- Wählen Sie unter "**Werkzeuge-Port . /dev/ttyUSB0**"



- Öffnen Sie den Dateiexplorer und legen Sie in ihrem "Persönlichen Ordner" (**/vol/vol_home_bcgst/bcgst**<XXX>) einen neuen Ordner "**iotsession** " an und speichern Sie die folgende Datei in diesem Ordner.
- Dazu mit **rechter Maustaste** auf den folgenden Link klicken, **Ziel Speichern unter** wählen, **Persönlicher Ordner iotsession**, **Speichern**

StandardFirmataPlus.ino

- Öffnen Sie in der Arduino IDE die soeben gespeicherte Datei (**Datei Öffnen**) und bestätigen Sie das PopUp mit "**Ok**". Es öffnet sich ein zweites Arduino IDE - Fenster mit dem Firmware-Skript.



- Kompilieren Sie das Script, indem Sie auf den Haken oben links "**Überprüfen**" klicken.
- Wenn das Kompilieren erfolgreich war, laden Sie die Firmware auf den Arduino Nano, indem Sie auf den Pfeil oben links "**Hochladen**" klicken. Der Nano sollte dabei rot und grün leuchten und nach erfolgreichem Hochladen sollte nur noch eine LED grün leuchten. Anschließend können Sie können die Arduino IDE schließen.

1. Testprogramm

- Öffnen Sie ein neues Terminalfenster navigieren Sie in den "**iotsession**" Ordner und installieren Sie alle JavaScript Bibliotheken, die wir im Laufe der Session zur Programmierung unseres Arduino Geräts verwenden werden http://johnny-five.io/.

Terminalfenster / Konsole

```
$ cd iotsession
$ npm init -y
$ npm install johnny-five --save
$ npm install web3@1.0.0-beta.35 --save
$ npm install oled-js --save
$ npm install png-to-lcd --save
$ npm install oled-font-5x7 --save
```

- Während der Installation können ein paar Warnungen in der Ausgabe erscheinen. Das ist in Ordnung. Sollten Fehlermeldungen (rot) erscheinen, wenden Sie sich bitte an den Dozenten.
- Öffnen Sie den **Komodo Editor** über das Startmenü und navigieren Sie in den "**iotsession**" Ordner und legen Sie eine neue Datei **"led.js"** an und kopieren Sie folgenden Code in die Datei.

led.js

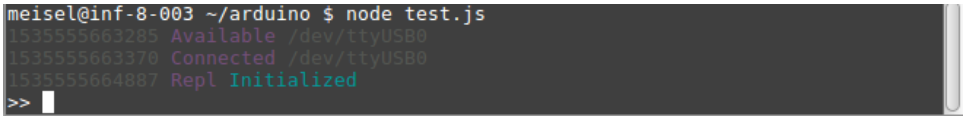
```
var Five = require("johnny-five");
var board = new Five.Board();

board.on("ready", function() {
  var led = new Five.Led(13);
  led.blink(100);
});
```

- Speichern Sie die Datei.
- Führen Sie den Test aus, indem Sie die soeben angelegte Datei im Terminalfenster starten. Die rote LED ganz rechts sollte nun blinken. Beenden Sie den Test in dem Sie im Terminalfenster zwei mal **Strg+C** drücken.

Terminalfenster / Konsole

```
$ node led.js
```



2. Testprogram

- Nun wollen wir das Oled-Display testen. Legen Sie dazu im Komodo-Editor im Ordner "**iotsession**" eine neue Datei "**oled.js**" an.
- Kopieren Sie den folgenden Code in die Datei, speichern Sie und testen Sie das Programm.

oled.js

```
var Five = require('johnny-five');
var Oled = require('oled-js');
var Font = require('oled-font-5x7');

var board = new Five.Board();

board.on('ready', () => {

  // oled configuration (address may differ 0x3B)
  const opts = {width: 128, height: 64, address: 0x3C};
  var oled = new Oled(board, Five, opts);

  oled.turnOnDisplay();
  oled.setCursor(1, 1);
  oled.writeString(Font, 3, 'Hello World', 1, true, 2);

});
```

```
$ node oled.js
```

Blockchain Anbindung

Nun möchten wir den Nano an die Blockchain anbinden und auf Events reagieren. Dazu soll er im Oled-Display immer die aktuelle Blocknummer und den aktuellen Blockhash anzeigen, sobald ein neuer Block veröffentlicht wird.

- Legen Sie mit dem Editor im Ordner "**iotsession**" eine neue Datei "**blockwatcher.js**" an und fügen Sie folgenden Code ein:

blockwatcher.js

```
var Web3JS = require ('web3');
var Five = require("johnny-five");
var OledJs = require('oled-js');
var Font = require('oled-font-5x7');

// connect to blockchain client
var web3ws = new Web3JS(new Web3JS.providers.WebsocketProvider("wss://ropsten.infura.io/ws"));

// load board ressources
var board = new Five.Board();
var oled;

board.on("ready", function() {

    // oled parameters
    const opts = {width: 128, height: 64, address: 0x3C};

    // initialize display
    oled = new OledJs(board, Five, opts);

    // clear the display on startup
    oled.clearDisplay(true);
    oled.update();

    // wait for blockheader
    listenToBlockchain();
});

// listen for blockchain events (only works on web socket connections)
function listenToBlockchain() {

    // event listener
    web3ws.eth.subscribe('newBlockHeaders')
    .on("data", function(blockHeader){
        console.log("Block: "+blockHeader.number);
        output(blockHeader);
    })
    .on("error", function(e){
        console.log("FEHLER: "+e);
    });
}

// output function
function output(block) {
    oled.setCursor(1, 1);
    oled.writeString(Font, 1, ''+block.number, 1, false, 2);
    oled.setCursor(1, 15);
    oled.writeString(Font, 1, ''+block.hash, 1, true, 2);
}
```

- Testen Sie die Anwendung.

```
$ node blockwatcher.js
```

IoT und Smart Contracts

Zum Abschluss der Praxissessions wollen wir nun eine Art Werbebannervermietung auf Blockchainbasis realisieren. Ihre kleinen Oled-Displays dienen dazu als Werbebanner, die von Jemandem (natürlich gegen eine angemessene Bezahlung) gemietet werden können und mit entsprechenden Inhalten des Mieters versehen werden sollen. Den Smart-Contract, der die Vermietung und das Hochladen der Werbebilder realisiert, haben wir schon implementiert und ist unter folgenden URL verfügbar.

http://141.55.229.170:3000/

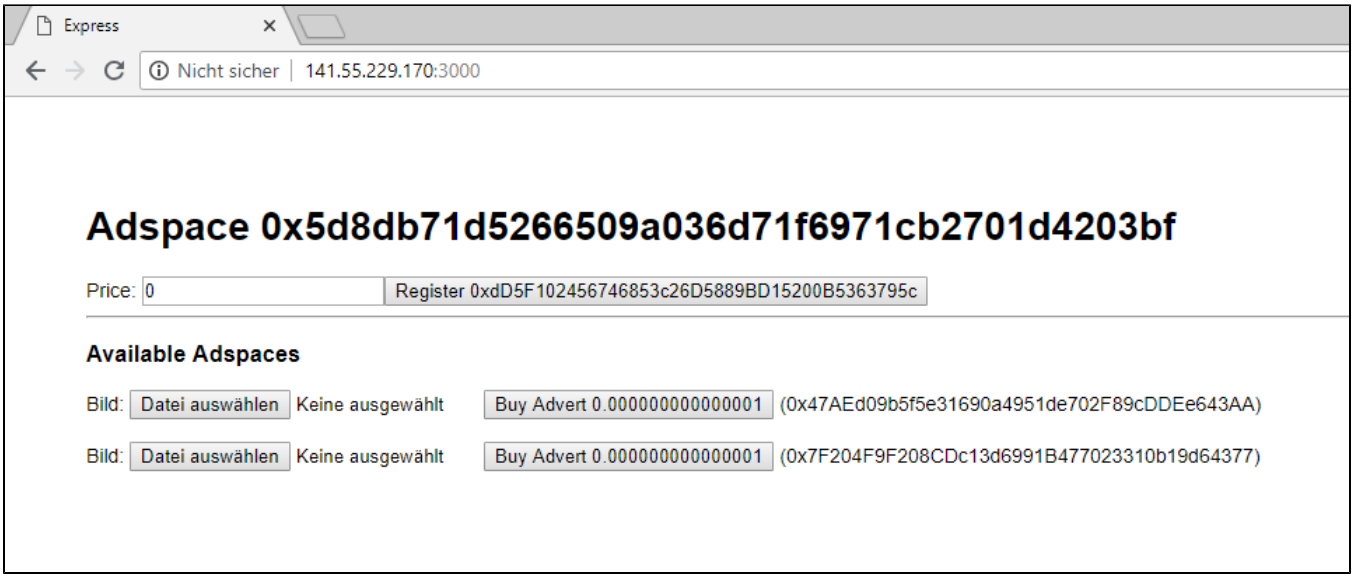
Den Quellcode des Smart Contracts finden Sie hier: Adplace.sol

Einleitung

- Um Ihr Oled-Display vermietbar zu machen, müssen Sie es beim obigen Smart Contract registrieren.
- Als Identifikationsnummer für ihr Display wird die **Adresse des Accounts** benutzt mit, mit dem Sie sich beim Smart Contract registriert haben.
- Um sich beim SC über das Webfrontend zu registrieren, benötigen Sie wieder das **Meta Mask Plugin**.
- Installieren Sie das Meta Mask Plugin analog der Anleitung falls Sie die Praxissession 2a - Ethereum Grundlagen verpasst haben sollten.
- Stellen Sie Meta Mask auf das Testnetzwerk "**Ropsten**" um und organisieren Sie sich Testether

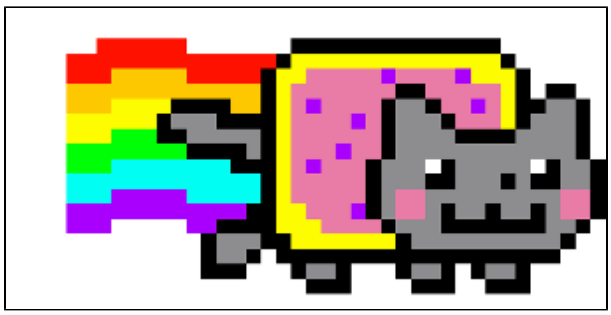
Registrierung beim Smart Contract

- Wenn Sie erfolgreich Meta Mask installiert haben, sollte das Webfrontend des SC nun so oder so ähnlich aussehen.



- In der oberen Leiste gibt es nun die Möglichkeit zum Registrieren über den Button "**Register 0x...**".
- Zunächst müssen Sie den Preis für ihre Werbefläche in **Wei** eingeben und anschließend auf den Button klicken. Bestätigen Sie die Transaktion und warten Sie, bis die Transaktion gemined wurde (dazu erscheint ein PopUp von Meta Mask).
- Aktualisieren Sie anschließend das Webfrontend.
- Nun sollte in der oberen Leiste nur noch ein Button zum Aktualisieren des Preises stehen und außerdem sollte ihre Adresse in der Liste der verfügbaren Werbeplätze erscheinen.
- Laden Sie nun ein Testbild in die Blockkette welches mit ihrer Adresse assoziiert wird.

- Dazu können Sie z.B. eines der folgende Testbilder verwenden:



  oder Sie wählen ein eigenes Bild vom Typ .png aus. Beachten Sie, dass es möglichst im Seitenverhältnis 2:1 ist und dass das Display nur zwei Farben anzeigen kann "Weiß und Schwarz" und dass Transparenz als "Schwarz" angezeigt wird.
- Klicken Sie nun in der Zeile ihrer Adresse auf "**Datei auswählen**" und wählen Sie ein entsprechendes Bild und anschließend auf "**Buy Advert 0.....**".
- Bestätigen Sie die Transaktion.

Anzeige ihres Werbebildes

- Nun sollen Sie sich das soeben in die Blockchain geladene Bild auf ihrem Oled-Display anzeigen lassen. Dazu müssen wir eine Verbindung zum Smart Contract in der Ropsten Blockchain herstellen. Dazu benötigen Sie die Adresse und das ABI (Interface) des Contracts. Beide Informationen sind in der folgenden Datei einhalten, die Sie sich bitte in ihren Ordner "**iotsession**" kopieren.

  Adplace.json

- Erstellen Sie nun im Ordner "**iotsession**" eine neue Datei "**viewadvert.js**" und fügen Sie den folgenden Quellcode ein. Wichtig ist, dass Sie:

  in **Zeile 8** als "**adspaceid**" ihre **eigene Meta Mask Adresse** eintragen.

viewadvert.js

```
var Web3JS = require ('web3');
var five = require("johnny-five");
var fs = require('fs');
var OledJs = require('oled-js');
var ptl = require('png-to-lcd');
var font = require('oled-font-5x7');

const adspaceid = '0xABC...';

var web3ws = new Web3JS(new Web3JS.providers.WebsocketProvider("wss://ropsten.infura.io/ws"));
var oled;

// load board ressources
var board = new five.Board();
board.on("ready", function() {

    //oled parameters
    const opts = {
        width: 128,
        height: 64,
        address: 0x3C
    };

    //initialize display
```

```
    oled = new OledJs(board, five, opts);
    oled.clearDisplay(true);
    oled.update();

    //wait for sale event
    listenToBlockchain();
});

// load blockchain
function listenToBlockchain() {
    loadContract((contract)=>{
        loadImageFromBlockchain(contract, adspaceid);
    });
}

// load contract
async function loadContract(cb) {
    // load contract information
    var contractData = JSON.parse(fs.readFileSync('Adplace.json'));

    // load network id of remote client
    var networkId = await web3ws.eth.net.getId();

    // load contract and call callback
    cb(new web3ws.eth.Contract(contractData.abi,contractData.networks[networkId].address));
}

// update function
async function loadImageFromBlockchain(contract, adspaceid) {

    // load encoded image data from contract
    var b64string = await contract.methods.adverts(adspaceid).call();

    // decode image data and write image file
    fs.writeFileSync('advert.png', Buffer.from(b64string, 'base64'));

    // update display
    updateDisplay('advert.png',false);
}

// function update display
function updateDisplay(file,dither) {
    ptl(file,dither,function(e,b){
        oled.buffer = b;
        oled.update();
    });
}
```

- Testen Sie die Anwendung und prüfen Sie, ob das Bild angezeigt wird.

```
$ node viewadvert.js
```

Live Aktualisierung der Werbeflächen

- Im letzten Schritt möchten wir nun noch unsere Software auf Events des Smart Contracts der Ropsten Blockchain lauschen lassen und unser Display immer dann automatisch aktualisieren, wenn jemand eine neue Anzeige hochgeladen hat.
- Erstellen Sie dazu ein neue Datei "**advertwatcher.js**" im Ordner "**iotsession**" und fügen Sie den folgenden Quellcode ein. Achten Sie wieder darauf:

  in **Zeile 8** als "**adspaceid**" ihre **eigene Meta Mask Adresse** einzutragen.

advertwatcher.js

```
var Web3JS = require ('web3');
var five = require("johnny-five");
var fs = require('fs');
var OledJs = require('oled-js');
```

```
var ptl = require('png-to-lcd');
var font = require('oled-font-5x7');

const adspaceid = '0xABC...';

var web3ws = new Web3JS(new Web3JS.providers.WebsocketProvider("wss://ropsten.infura.io/ws"));
var oled;

// load board ressources
var board = new five.Board();
board.on("ready", function() {

    //oled parameters
    const opts = {
        width: 128,
        height: 64,
        address: 0x3C
    };

    //initialize display
    oled = new OledJs(board, five, opts);
    oled.clearDisplay(true);
    oled.update();

    //wait for sale event
    listenToBlockchain();
});

// load blockchain
function listenToBlockchain() {
    loadContract((contract)=>{
        listenForSaleEvent(contract, adspaceid);
    });
}

// load contract
async function loadContract(cb) {
    // load contract information
    var contractData = JSON.parse(fs.readFileSync('Adplace.json'));

    // load network id of remote client
    var networkId = await web3ws.eth.net.getId();

    // load contract and call callback
    cb(new web3ws.eth.Contract(contractData.abi,contractData.networks[networkId].address));
}

// update function
async function listenForSaleEvent(contract, adspaceid) {

    // contract event listener
    contract.events.Sale().on('data', async function(event){

        // check for own events
        if (event.returnValues._id == adspaceid) {

            // load encoded image data from contract
            var b64string = await contract.methods.adverts(adspaceid).call();

            // decode image data and write image file
            fs.writeFileSync('advert.png', Buffer.from(b64string, 'base64'));

            // update display
            updateDisplay('advert.png',false);
        }

    // error handler
    }).on('error', function(e){
        console.log("Fehler: \n"+e);
    });
}
```

```
// function update display
function updateDisplay(file,dither) {
    ptl(file,dither,function(e,b){
        oled.buffer = b;
        oled.update();
    });
}
```

Starten Sie nun den Advertwatcher und testen Sie ihn, indem Sie über das Webfrontend des Smart Contracts auf ihre Anzeigefläche ein neues Bild hochladen. Wenn Sie noch Zeit haben, können Sie ja auch den anderen Teilnehmern ein paar Bilder auf deren Displays hochladen, falls diese den Advertwatcher auch schon fertig bekommen und gestartet haben.