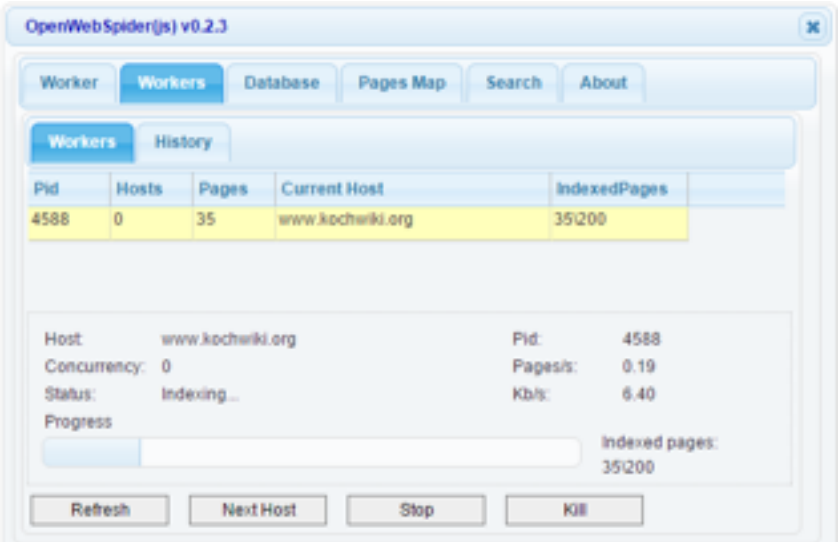
PoC:

1. Datenbank durch den Crawler füllen
Die Rezepte auf der Webseite „chefkoch.de“ werden durch einen Crawler in der Datenbank gespeichert.
Exit:
Der Crawler funktioniert einwandfrei und die Rezepte können konsistent gespeichert werden
Fail:
Die Rezepte sind nicht konsistent und sind somit nicht vergleichbar.
Fallback:
Die Datenbank wird per Hand gefüllt und kann durch den User erweitert werden.

Ergebnis:

Der vierte PoC konnte nicht erfolgreich umgesetzt werden.
Es hat sich nach einer kurzen Recherche herausgestellt, dass wir die Rezepte auf „chefkoch.de“ nicht für unsere Zwecke benutzen dürfen. Deshalb mussten wir auf deine freie Rezeptdatenbank zurückgreifen (kochwiki.org).
Als Datenbank haben wir MongoDB eingesetzt und als Webcrawler kam Openwebspider zum Einsatz. Dieser Crawler hat eine Anbindung zu MySQL und MongoDB und bietet zusätzlich eine grafische Benutzeroberfläche.
Mithilfe des Crawlers wurden dann Rezepte von „kochwiki.org“ in die Datenbank importiert.
Das Problem war hierbei, dass der gesamte Inhalt einer Rezeptseite ohne Formatierung gespeichert wurde. Dadurch ist es sehr schwer den Inhalt nach bestimmten Kriterien zu filtern, beispielsweise die Zutaten von den Kochanweisungen abzugrenzen.
Aus zeitlichen Gründen haben wir uns dann entschieden, manuell Rezepte in die Datenbank einzutragen.




```
{
    "_id" : ObjectId("563b5ecd978ee7ec112b1d48"),
    "host" : "www.kochwiki.org:80",
    "page" : "/wiki/Belgische_K%C3%A4seb%C3%A4llchen",
    "title" : " Belgische Käsebällchen - Koch-Wiki",
    "text" : " Belgische Käsebällchen aus dem Koch-Wiki (kochwiki.org) Wechseln zu: Navigat
terverarbeitung. Tipp [ Bearbeiten ] Als Vorspeise nicht mehr als 2 bis 3 Käsebällchen pro
ortale Von A bis Z Zufälliges Rezept Cocktails Kochtechniken Küchengeräte Mitmachen Portal
    "cache" : "",
    "anchor_text" : " Belgische Käsebällchen",
    "level" : 1,
    "date" : ISODate("2015-11-05T13:51:09.358Z"),
    "md5" : "b4d6f078d772b3d3c19cd716469b057b"
}
```

2. Kommunikation
Die Kommunikation zwischen Client, Server und Datenbank soll reibungslos funktionieren.

Exit:
Die Kommunikation funktioniert ohne Probleme

Fail:
Es treten Probleme bei der Kommunikation auf.

Fallback:
Nachschlagen in Literatur und mit den Programmiersprachen vertraut machen.

Risiko → Netzwerkfehler bei der Client-Server-Kommunikation, Auswahl der falschen Programmiersprache

Ergebnis
Um diesen PoC durchzuführen, wurde zunächst die MongoDB-Datenbank installiert und der Server mittels Node.js programmiert. Der Server soll eine POST und eine GET Methode beinhalten. Es sollen somit sowohl Daten vom Client entgegen genommen werden als auch Daten an dem Client mittels HTTP verschickt werden. Die Daten sind im JSON-Format.
Der Client wird mit Java und die Post und Get Aufrufe werden in Methoden realisiert. Zunächst wird eine Verbindung zu dem Server aufgebaut, um dem Post auszuführen. Hierbei gibt es einen Output und einen Input. Der Output stellt den Request dar und der Input, welcher auf die Antwort des Servers wartet, den Response. Die Daten werden ausgelesen und und dem Client in der Konsole zu verfügung gestellt. Bei der Get Anfrage wird nur ein Input benötigt, da der Response ohne Datentransfer von statten gehen kann. Dieser PoC ist erfolgreich abgeschlossen worden:

Client

```
import java.net.*;
import java.io.*;

public class Pocs {

	public static void main(String[] args) {
		String url = "http://localhost:8080/test";
		String urlParameters = "{ \"test\": \"blub\", \"key\":
\"1\" }";
		String testPost, testGet;

		testPost = Post(url, urlParameters);
		System.out.printf("%s", testPost);
		testGet = Get(url);
		System.out.printf("%s", testGet);
	}
```

```
      public static String Post(String targetURL, String urlParameters){
            URL url;
            HttpURLConnection connection = null;
            try {
                //Verbindung wird erstellt (POST)
                url = new URL(targetURL);
                connection = (HttpURLConnection)url.openConnection();
                connection.setRequestMethod("POST");
                connection.setRequestProperty("Content-Type",
"application/json");

                connection.setRequestProperty("Content-Length", "" +
Integer.toString(urlParameters.getBytes().length));

                connection.setUseCaches (false);
                connection.setDoInput(true);
                connection.setDoOutput(true);

                //Request wird gesendet, Daten werden übergeben
                DataOutputStream output = new DataOutputStream
(connection.getOutputStream ());
                output.writeBytes (urlParameters);
                output.flush ();
                output.close ();

                //Response wird erwartet und die übertragenden Daten
werden gespeichert
                InputStream input = connection.getInputStream();
                BufferedReader tempBuffer = new BufferedReader(new
InputStreamReader(input));
                String line;
                StringBuffer response = new StringBuffer();

                while((line = tempBuffer.readLine()) != null) {
                  response.append(line);
                  response.append('\r');
                }

                tempBuffer.close();
                return response.toString();

            } catch (Exception e) {

                e.printStackTrace();
                return null;

            } finally {
```

```
                if(connection != null) {
                      connection.disconnect();
                }
          }
     }

     public static String Get(String targetURL){
          URL url;
          HttpURLConnection connection = null;
          try {
              //Verbindung wird erstellt (GET)
              url = new URL(targetURL);
              connection = (HttpURLConnection)url.openConnection();
              connection.setRequestMethod("GET");
              connection.setRequestProperty("Content-Type",
"application/json");

              //Response wird erwartet und die übertragenden Daten
werden gespeichert
              InputStream input = connection.getInputStream();
              BufferedReader tempBuffer = new BufferedReader(new
InputStreamReader(input));
              String line;
              StringBuffer response = new StringBuffer();

              while((line = tempBuffer.readLine()) != null) {
                response.append(line);
                response.append('\r');
              }

              tempBuffer.close();
              return response.toString();

          } catch (Exception e) {

              e.printStackTrace();
              return null;

          } finally {

                if(connection != null) {
                      connection.disconnect();
                }
          }
     }
}
```

Server

```
var http = require('http');
var express = require('express');
var bodyparser = require('body-parser');

var MongoClient = require('mongodb').MongoClient;
var assert = require('assert');

var jsonParser = bodyparser.json();
var app = express();
var server = http.createServer(app);
app.use(jsonParser);

var url = 'mongodb://localhost:27017/test';
MongoClient.connect(url, function(err, db) {
  assert.equal(null, err);
  console.log("Connected correctly to server.");
  db.close();
});

app.post('/test', function(req, res){
    var test = req.body;
    MongoClient.connect(url, function(err, db) {
        var collection = db.collection("test");
        collection.insert(test, function(err, res){
            if (err) {
                console.log(err);
            } else {
                console.log('blub');
                console.log("Post ist angekommen");
                console.log(test.test);
        }
        db.close();
        });
        res.send(test);
    });
});

app.get('/test', function(req, res){
    MongoClient.connect(url, function(err, db){
        var collection = db.collection("test");
        collection.findOne({key: "1"}, function(err, item){
            if(err){
                console.log("err");
            } else {
                console.log(item);
                res.send(item);
            }

        });

    });

});



app.listen(8080);
console.log('Port: 8080');
```

3. Einkaufszettel Generierung
Aus den Zutaten der Rezepte soll eine Einkaufsliste erstellt werden.

Exit:
Die Einkaufsliste wird korrekt erstellt.

Fail:
Die Einkaufsliste wird nicht korrekt erstellt.

Fallback:
Nachschlagen in Literatur und überlegen ob JSON das richtige Format ist.

Ergebnis:
Aufgrund des Zeitmangels konnte dieser PoC zur Deadline nicht realisiert werden. Grund dafür ist die Einarbeitung in den JSON Parser von Java, welche viel Zeit in Anspruch genommen hat als eigentlich veranschlagt.
Um die Daten zu ermitteln, muss das JSON Objekt gefiltert und ausgewertet werden. Hierbei wird die Zutat im JSON Objekt als Key und die Menge als Value gespeichert. Die Filterung im Client der einzelnen Zutaten hat funktioniert, jedoch können die Werte der Daten noch nicht ermittelt werden.

```
import org.json.simple.*;

	public static void einkaufliste(){

		String jsonArrayString = "[ {\"Zutat1\": \"1\"}, {\"Zutat2\": 
\"2\"} ]";
		Object obj=JSONValue.parse(jsonArrayString);
		JSONArray array=(JSONArray)obj;
		System.out.println(array.get(1));
	}
```