Bachelorthesis

Development of a Web-Application for executing Job-Scripts on an IBM-Mainframe in context of Education

Bachelorarbeit gemäß § 17 der Allgemeinen Prüfungsordnung vom 01.08.2008
im Bachelorstudiengang Informationsmanagement und Unternehmenskommunikation
an der Hochschule für angewandte Wissenschaften Neu-Ulm

Erstkorrektor	Prof. Dr. Phillipp Brune
Betreuer	Dr. Kevin Henrichs
Verfasser	Marc Morschhauser (Matr.-Nr.: 204041)

Thema erhalten	01.01.2020
Arbeit abgegeben	01.05.2020

Unterschrift des Studierenden

Unterschrift und Firmenstempel der Ausbildungsstelle

In association with:

HNU HOCHSCHULE NEU-ULM UNIVERSITY OF APPLIED SCIENCES

TALENTSCHMIEDE

Academic Mainframe Consortium

Gruendermaschine.com
We fight for Your Ideas.

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

1 Abstract

Hier kommt der/die/das Abstract.

List of Figures

2 List of abbreviations

IoT	Internet of Things
I/O	Input / Output
OS	Operating System
ULE	Ubiquitous Learning Environment
JCL	Job Control Language
JES	Job Entry Subsystem
TSO	Time Sharing Option
USS	Unix System Services
OLTP	Online Transaction Processing
LPAR	Logical Partitions
LAN	Local Area Network
FTP	File Transfer Protocol
SSH	Secure Shell
SFTP	Secure File Transfer Protocol
VPN	Virtual Private Network

Contents

3 Introduction

Big Data, Cloud, Blockchain, Internet of Things (IoT) - Digitalization is steadily proceeding and with it the amount of data that has to be processed every day. Many web-services at this point are dependent on flexibility in first place. Big server farms facilitate an unprecedented agility to operators of online-businesses referring to their resource-planning. Urgent situations with high data traffic can be handled by adding physical or also virtual servers within minutes. However there are services where minutes make the difference between millions of euros. Those services demand for information technology, which not only has a high data throughput but also ensures the highest availability. The IBM Mainframe in its newest construction provides an Input / Output (I/O) of 288GB per second while being available 99,999% of the time running. This makes the Mainframe a cornerstone in present finance-sector's IT-management. Due to it's security and it's high speed it also finds it's use in other big industries - thus also insurances, the aerospace industry or big retail companies benefit from this supercomputer.
But since the IT improved considerably over the last decades, the Mainframe was presumed dead and many companies discontinued training their staff to maintain this technology. Contrary to their expectations they are dependent on it till today and they will presumably be in the remote future. What is left is a big gap in the division of skilled professionals that is hard to close.
The Goethe-University in Frankfurt, Germany, attends to participate in closing this gap. Therefore the university procured one of the aforementioned IBM supercomputers to train their students dealing with it.

This Bachelor Thesis approaches a Front-End-Application, which will be running on a Linux Virtual-Machine but executing tasks on the mainframe's Operating System (OS) called z/OS. By building this bridge between the Linux VM and the z/OS, students will be able to get in touch with the Mainframe located in Frankfurt University through a web-browser and as a consequence experience how the system is working.

The resources for this activity are allocated by the *Talentschmiede AG* - a Frankfurt located IT consultancy which is in close contact to the finance sector and knows and cares about the gap of skilled professionals. Further it is supported by the *Academic Mainframe Consortium e.V.* - an association founded by Mainframe Experts which are also willing to acquire new educated staff in the division of Mainframe.

4 Related Work

While a malicious tongue once suggested that " the last mainframe will be unplugged in 1996 " , it reconsidered when IBM introduced it's latest version of it in 2008 [12]. Till the present day the mainframe is the backbone of the financial markets worldwide and just as important for other big industries[1] [11]. The view on the mainframe technology as an IT-dinosaur has to change and the need of adding the mainframe technology to the IS Curriculum in a wide range was overdue already 10 years before [18] [19] [4]. Sharma et al. analysed the need of Large System Education regarding to mainframe education and they are investigating " the academic response to the need for large systems specialists " [15]. In [3] A. Corridori addresses the concerns and the opportunities that come with adding mainframe content to universities curricula and also previews ways to do this. How the economy and with it, the labor market can benefit from it is stated in [16].

The potentials of present digital media used to educate in a wide range is discovered by Cope et al. [2]. It is described that "the learner's relationship to knowledge and the processes of pedagogy have not changed in any significant way" but through technology " the educational paradigm has changed ". Vicki Jones et al. also address the integration of modern information technology in everyday's life and the advantages of this change regarding education [8]. Here it is stated that " Adaptive learning can offer great advantages in providing students with specific and personalised knowledge as and when required ". While the term " Adaptive Learning " is responsive to the methods that are used to transfer knowledge [13] , " Ubiquitous Learning Environment (ULE) " describes an environment with the possibility of learning everywhere and anytime[6]. Hwang et al. emphasize the fact, that ULEs are an innovative approach for teaching complex topics [7]. To establish a ULE there are many technologies needed [14], this thesis is supposed to do a first step in this direction to learn on the Mainframe anywhere and anytime.

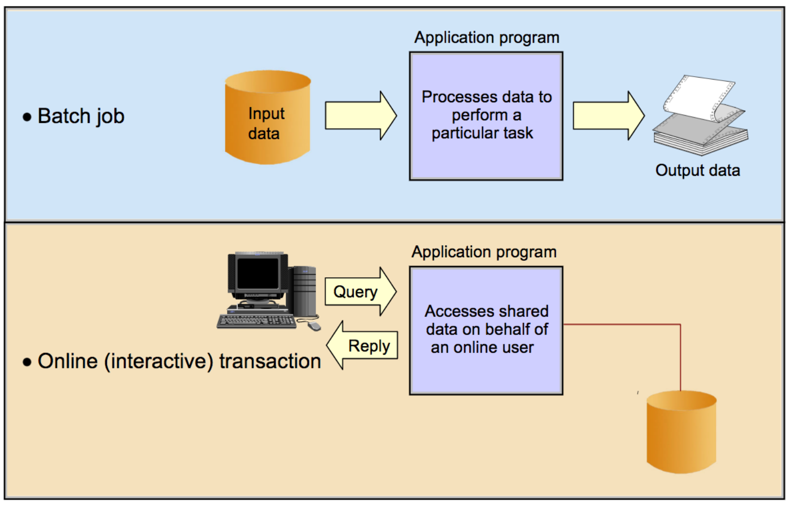
Kiefer, COBOL as a modern language [10]
Khadka, How do professionals perceive legacy systems and software modernization? [9]
Vinaja, 50th anniversary of the mainframe computer: a reflective analysis [17]

......

5 An overview on the IBM Mainframe

Prior to entering the main issue of implementing the web-application, this chapter introduces a few mainframe terminologies to assure a full understanding of the steps that take place in the following chapters.
Because, while many people just take the easy way by calling almost every computer a *server*, and the term *mainframe* is often just used to point out that this is the largest server in use, in this thesis the term *mainframe* describes the IBM Supercomputer which is capable of "supporting thousands of applications and input/output devices to simultaneously serve thousands of users" [5].
So the most common utilization of the mainframe is divided in two categories:

- Online Transaction Processing (OLTP), incl. web-based applications
- Batch Processing



Batch job
Input data
Application program
Processes data to perform a particular task
Output data
Application program
Query
Reply
Accesses shared data on behalf of an online user
Online (interactive) transaction

Figure 1-3 Typical mainframe workloads

Figure 1: Typical workloads

5.1 Online Transaction Processing

One of the core functions of many businesses is OLTP. For this case the mainframe serves a large amount of applications to make it possible to execute thousands of transactions in short time while handling not only a huge amount of different transaction types, but also to do this with many different users at a glance.
For end-users those transaction processes are often commonly known, while they appear in everyday's life, such as:

- Credit card payments in supermarkets
- ATM transactions
- Online purchasings

So in many cases you are already using a mainframe without even taking any note. Since these transaction processes are generally carried out through web-technology they are not to be consolidated in this thesis.

5.2 Batch Processing

The other main function of the mainframe is processing data in a batch. Eventually terabytes of. These processes are generally done without much user interaction. A batch job simply gets committed and processes the data that is determined in the job-statement.

While we dive deeper into the Job-Creation-Language that leads to those processes in chapter 7, it is important for now to understand the concept of these jobs that can be imagined equivalent to a UNIX script file or a Windows command file. The difference is, that z/OS batch-job processes millions of records.
A famous and easy understandable example for a classical batch-job is the payroll of a big-company. z/OS here takes up a core function by bringing together a huge amount of data consisting of personal data, data referring to working hours and salaries. The mainframe is predestined to work this out on a safe and reliable basis.

But of course there are also people included in this process that have to be reliable as well but in addition also have to govern this whole process what makes them exactly the target group of the application that will be developed in this thesis.

5.3 Separation of duties

To define the target grouper in more detail, this chapter will give a quick overview on people that have contact to the mainframe in daily work life.
distinguish between:

- System programmers
- System administrators (for example, DBA, storage, network, security, and performance)

- Application designers and programmers
- **System operators (explain)**
- Production control analysts

5.4 Mainframe Operation Systems

Another ambit where insights are required for further understanding of the application is the ambit of the different Operation Systems that can be run on the mainframe.

– z = zero Downtime?! –

- **z/OS** is the IBM operating system that runs on most z-system mainframe installations and by that takes the leading role in the field of mainframes. It isn't set apart a lot from normal operating systems as it has the same structure of three layers, consisting of Hardware, Operating System and User Processes. Similar to modern windows-systems there are some kind of subsystems, which run between the OS and the user processes. The most important one in the case of z/OS are the **??** for the background operations (batch processing), the **??** for the foreground operations (interactive) and the **??**, which represent a posix-compatible unix subsystem.
- **z/VSE**
- **z/TPF**
- **z/VM**
- **Linux for zSeries**

5.5 Virtualisation with z/VM

As an aid to consolidation, the mainframe offers software virtualization, through z/VM. z/VM?s extreme virtualization capabilities, which have been perfected since its introduction in 1967, make it possible to virtualize thousands of distributed servers on a single server, resulting in the significant reduction in the use of space and energy.

z/Virtual Machine (z/VM) has two basic components: a control program (CP) and a single-user operating system (CMS). As a control program, z/VM is a hypervisor because it runs other operating systems in the virtual machines it creates. Any of the IBM mainframe operating systems such as z/OS, Linux on System z, z/VSE, and z/TPF can be run as guest systemsin their own virtual machines, and z/VM can run any combination of guest systems.

—- Explain and show z/VM Constellation, LPARs etc —-

more in : `http://www.redbooks.ibm.com/redbooks/pdfs/sg247603.pdf`

6 Research gap

While the web-access to z/OS for OLTP is very established because of the usage, that is often conducted by end-users or computers, that are not located in immediate proximity to the server, the access to the JES-spool through web-applications is less common, since the system-operators are usually working in ultimate contact to the mainframe.
Certainly there are operators handling batch-processes through a web front-end due to flexibility and simplicity reasons. But while these applications are to simplify the process of managing the workload in daily-business, this thesis tries to establish an environment where the jobs have to be done in full amplitude, but on a test-system, implemented just for educational reasons.
This setup will have the ambitions of getting people into JCL quite quickly on the one hand and give them the opportunity to train their abilities and, in this way reduce potential uncertainties towards working on large-scale systems, on the other hand. With this method a pertinent and long-lasting learning effect is to be achieved in short time.

7 Dive into JCL

—- Important to understand JCL for accessing JES Spool?! —-

To get familiar with JCL, this chapter will give an introduction in how these Statements are working. There is talk about "Statements", because JCL is used to tell the OS what to do. Each Statement is an independent work unit, known as "Job" - therefore this language is called "Job Control Language". Each Job consists of instructions that are either typed in by an operator or they are stored and get transmitted to the computer.

7.1 Job Control Environment

So to understand how a job is executed, it is important to know which components are needed for this process.

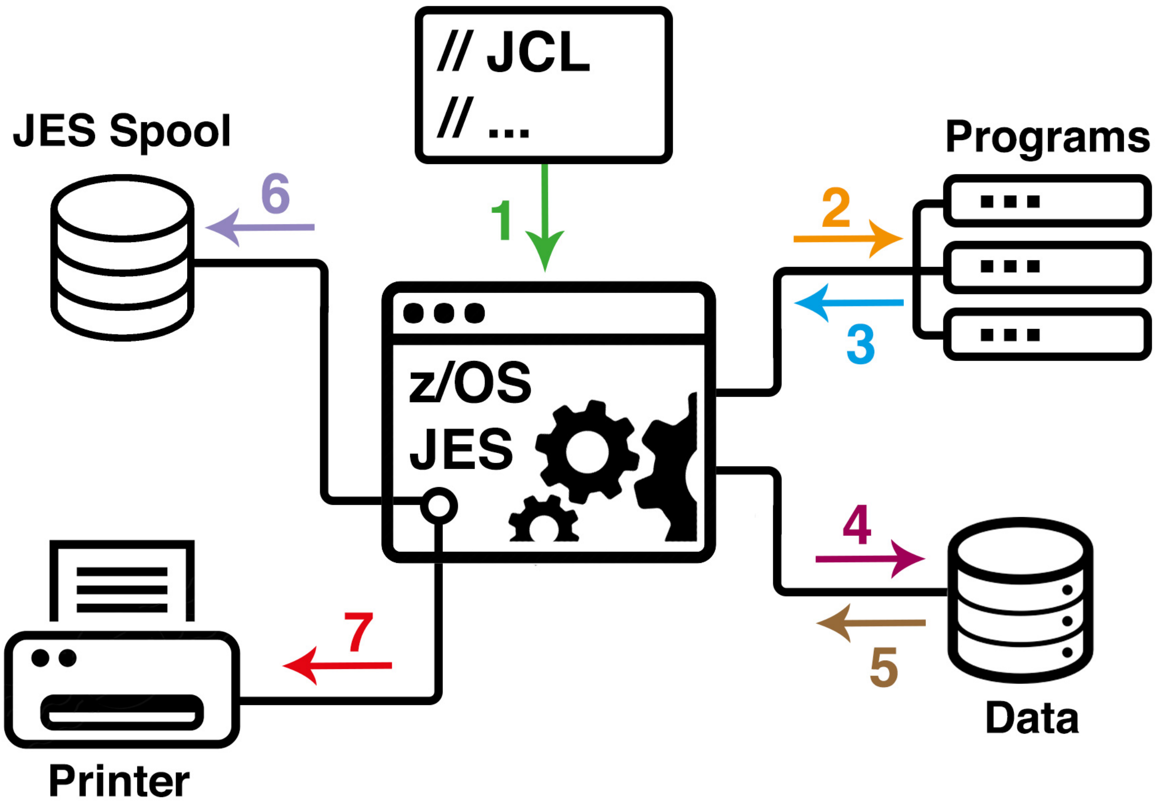


Figure 2: JCL Functionality, Source: IBM

On figure 2 you can see a job process described in 7 steps.

1. JCL submit

2. JCL requests program
3. Program gets loaded
4. Resources for program get allocated through JCL
5. Resources get provided to program
6. Program writes output to JES Spool
7. Output to gets transferred to printer as requested

The z/OS has written (hard coded) application programs which are not associated with any physical resources. Those programs are just names, which include internal file-names. Through JCL those programs can be opened for reading and writing during execution. The Job Entry Subsystem (JES) is there to evaluate and accept or not accept a job. If the syntax is right and the job is accepted the JES runs it on the OS and controls this process. The results get transferred to an output unit.

7.2 Job structure

-each statement 80characters
-JCL is introduced with //

7.2.1 EXEC-Statements

Within a job, there are working executions introduced through an "EXEC-statement". Every job needs at least one execution, but there can obviously be many more in addition. If a job has no execution it stops.

```
//STEP0001 EXEC PGM=IEBGENER
...
//STEP0002 EXEC PROC=PRDPROC1
//STEP0003 EXEC PRDPROC2
```

7.2.2 DD-Statements

- SYSUT1
- SYSUT2
- SYSIN

- SYSPRINT

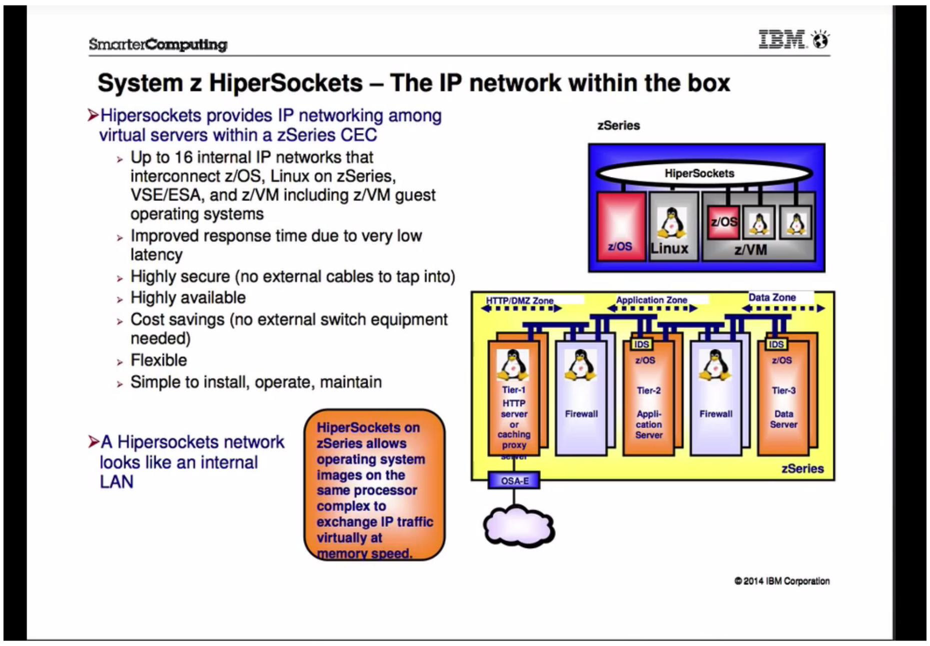
-JCL links the program file names with physical resources (e.g. data set names or unix file names).
–JCL is used to process programs in the background ("batch")
- And to process programs in the foreground("started task")
-JCL instruct z/OS -> Start / submit - JCL SUBMIT Statement will result in batch process of one or more programs (BACKGROUND)
- JCL START will result in FOREGROUND processing of a processing program

7.2.3 Background / Foreground jobs

Every Batch Job must contain JOB-statement & EXEC statement
->JOB statement highlight the beginning of a batch job & assigns a name to the job
JCL started tasks do not require a JOB Statement ->both have at least one EXEC statement -> marks the beginning of a job step , assigns name to the step & identifies the program or procedure to be executed in the step.

8 Access possibilities

In order to execute JCL-Jobs on z/OS from the Linux host, an access point is required to submit the data. This chapter is to reveal how the z/VM connects the Linux-host and the z-OS Target-system in this case and how this connection can be used to transfer data among them.
To enable the communication between different applications, a protocol is needed. The protocol that is used in the System Z is the **TCP!** (**TCP!**)/**IP!** (**IP!**), which is provided through the HiperSockets function. HiperSockets is a technology developed by IBM to enable high-speed communication between Logical Partitions (LPAR) with a hypervisor or between applications inside a z/VM as it is the case here.
You could imagine a HiperSockets-network like an internal Local Area Network (LAN), linking all partitions for internal communication.


SmarterComputing

System z HiperSockets – The IP network within the box

- Hipersockets provides IP networking among virtual servers within a zSeries CEC
  - Up to 16 internal IP networks that interconnect z/OS, Linux on zSeries, VSE/ESA, and z/VM including z/VM guest operating systems
  - Improved response time due to very low latency
  - Highly secure (no external cables to tap into)
  - Highly available
  - Cost savings (no external switch equipment needed)
  - Flexible
  - Simple to install, operate, maintain
- A Hipersockets network looks like an internal LAN

HiperSockets on zSeries allows operating system images on the same processor complex to exchange IP traffic virtually at memory speed.

© 2014 IBM Corporation

Figure 3: TCP/IP Connection / Hipersockets

—- Explain Hipersockets / TCP/IP Connection —-

A way we can use this connection to transfer jcl-jobs is:

Transmission Control Protocol

The Transmission Control Protocol (TCP) provides a reliable vehicle for delivering packets between hosts on an internet. TCP takes a stream of data, breaks it into datagrams, sends each one individually using IP, and reassembles the datagrams at

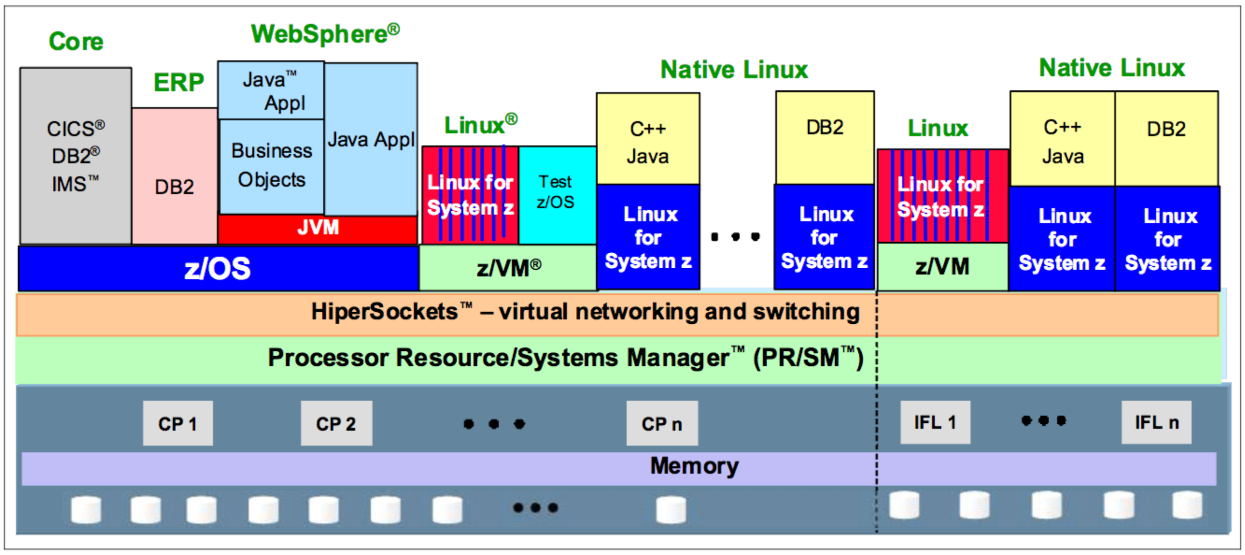


Figure 4: workloads

the destination node. If any datagrams are lost or damaged during transmission, TCP detects this and resends the missing datagrams. The received data stream is a reliable copy of the transmitted data stream.
—- Explain TCP/IP Connection in own words —-

8.1 Access through Java with FTP

What becomes possible through the TCP/IP in System Z is the File Transfer Protocol (FTP) Server, an IP-application used to transfer files between any kind of platform. The z/OS FTP-Server is a bit different from normal FTP-Servers as it provides not only the possibility to transfer files and get access to z/OS System Services, but it also provides access to the Job Entry Subsystem which is, as mentioned in chapter 7, needed to submit JCL-Jobs.

With the help of Java you can use the FTP server to get access to a number of JES functions, including the following:

- Submitting a job
- Displaying the status of jobs
- Receiving the spool output of a job (JCL messages and SYSOUT)

- Deleting a job
- Submitting a job and automatically receiving output

—- Importance of output for Learning environment —-

8.2 Access via SSH/ SFTP

While FTP gives us the opportunity to get access to the JES-Spool to run JCL-Jobs, this is not really a safe way to work this out.
To get a safe communication, it needs encryption what can be established through the SSH-protocol - making the FTP a Secure File Transfer Protocol.

The Secure Shell Protocol

The SSH protocol (also referred to as Secure Shell) is a method for secure remote login from one computer to another. It provides several alternative options for strong authentication, and it protects the communications security and integrity with strong encryption. It is a secure alternative to the non-protected login protocols (such as telnet, rlogin) and insecure file transfer methods (such as FTP).
—- Explain SSH in own words —-

8.3 Co:Z SFTP-Server

—- Describe Co:Z —-
A tool named Co:Z SFTP, an open-source product developed by Dovetailed Technologies, will help to establish a safe connection with before mentioned techniques. It works as a port of OpenSSH SFTP for z/OS and therefore enables the access to z/OS datasets and spool files.

9 Requirements

To get access to z/OS, a system constellation as seen on Fig. 5 will be implemented. A z/VM running Linux (LPAR 1) will serve as a web-server. Here the application is hosted and can be accessed through the user's web-browser. Due to the hipersockets network, that links LPAR 1 to the z/OS (LPAR 2), whereupon an SFTP-Server can be running, a stable and safe connection can be established. As before mentioned, the Co:Z SFTP-Server is deployed here, to get direct access to the JES-Spool. Through this constellation, students will be able to submit jobs directly on the mainframe, from any given device that they are using to access the internet.

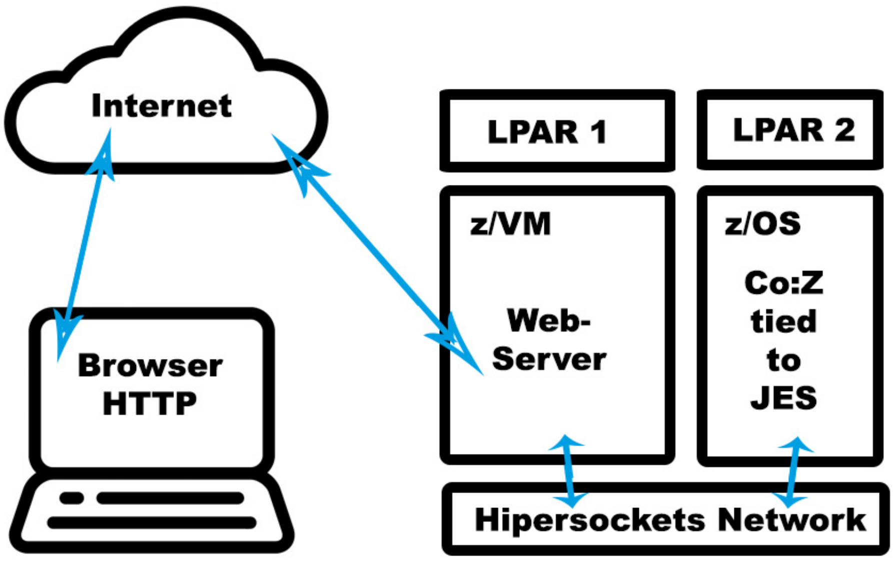


Figure 5: constellation

`http://www.redbooks.ibm.com/redbooks/pdfs/sg247603.pdf` from p.137

—- Ubuntu / Apache Server etc.?! Implementation part —-
—- Node.js website / hosting / Electron —-

10 Design

In this chapter, a framework, based on the node-stack, is used to implement the web-application. This framework, called Express, gives the opportunity to develop the javascript-application within an MVC-pattern, that can run on our zLinux and can be accessed through the browser. For local development, the mainframe access takes place through a Virtual Private Network (VPN) that is provided by the Frankfurt University. Later this application will be installed on zLinux, so the backend can access z/OS directly through the hipersockets network.
Since the Co:Z-SFTP-Server in its handling, differs slightly from usual SFTP-Servers, becoming noticeable through the commands that have to be executed, the following sub-chapters provide a brief glimpse on how the actions would be executed through a terminal. But it is avoided to go too much into detail here, as the focus is on the realisation through javascript, as these are the functions, which are triggered in the application in the end.

For further interests, the full Co:Z documentation can be read on http://dovetail.com/.

The following chapter will describe the whole application on the basis of the MVC-pattern on Fig. 6. A quick overview should help to get to know the functionalities straightforward. How the functions are implemented exactly will be described in the following subchapters. The operations that are listed on Fig. 6 all take place after the user is logged in.

From Left to Right the first entry point is the HTTP Request to the *routes.js*-file. This happens, when the user is logged in and gets redirected to the home view, which loads the *home.jade*-file. *Jade* is a template engine, used for serverside rendering of HTML-files in NodeJS. In this case home.jade renders a file called *index.html*. This is the front-end, that the user will see and will interact with.(Fig. 7, 8, 9, 10)
The two other routes that are called here pass the userdata for accessing the mainframe to functions in the *sftpManager.js* which works as model and is running as backend on the server. This model is the heart of this application as it contains all the logic to exchange data not only between the user and the database but also between the user and the mainframes' z/OS via SFTP.
The *myController.js* works as a middleware between front- and backend and therefore is used to manage responses from the mainframe and update the user on this operations.

After the functionality of the applications is clear, chapter 11 will introduce a few thoughts on how the users can learn from this program and how those thoughts are transposed.

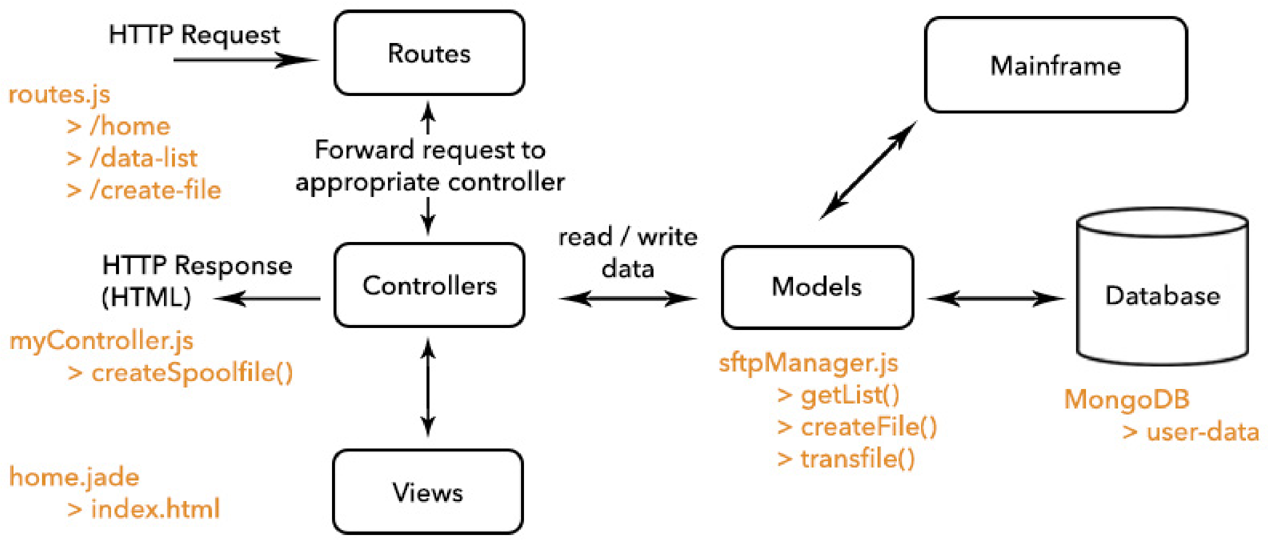


Figure 6: Model-View-Controller Scheme

10.1 Registration / Login

To be able to work on the IBM Machine, the students have to be added as authenticated users on the mainframe first. This has to be done by a mainframe administrator before. With given user-data they are ready to register for and use the Mainframe-Self-Service. An open-source login template, implemented by Stephen Braitsch (https://github.com/braitsch/node-login) will help to get started with the user registration and log-in. This template is built on the MongoDB database, which is a simple non-relational database also known as NoSQL-database. By registration this template saves the username and the password along with some other userdata into this database. There should not be too much detail about the login template here, as it is not part of the thesis. The documentation on its function can be read on the before mentioned github project of Stephen Braitsch.

The favor of this node-login is that when the user logs in its userdata is not only accessible from the database but also from an array called *session*. This session just exists as long as the user is logged in and gets destroyed when he logs out. From this session we take our userdata to access the mainframe.

For the next sections, an SFTP-Client is implemented to get a connection to Co:Z-SFTP. The node package *ssh2.js* by Brian White makes it possible to use the openSSH protocol as it is also used by Co:Z-SFTP. To access z/OS via SFTP we use the ssh2-package by requiring it on line 1. Then we establish a variable *connSettings* which contains all the requirements to access the mainframes' z/OS through SFTP and get access to the JES-Spool to read and write data. The host called here is the IP-Address of the Co:Z Server, port 22 is the port that is usually used for guest-access via SFTP. The guest-access is required due to the web-server running not in the same LPAR as the z/OS. In line 5 and 6 in the following code you can see that a request (req) is made to the session array to read the username and the password stored here as long as the user is logged in. How this data is passed to the functions can be seen in the chapters 10.3 and 10.4.

```
1 var Client = require('ssh2').Client;
2 var connSettings = {
3      host: '141.2.192.32',              // Mainframes z/OS
4      port: 22,                          // Guest-SFTP Port
5      username: req.session.user.user,   // Request sessions' username
6      password: req.session.user.pass    // Request sessions' password
7 };
```

10.2 Welcome

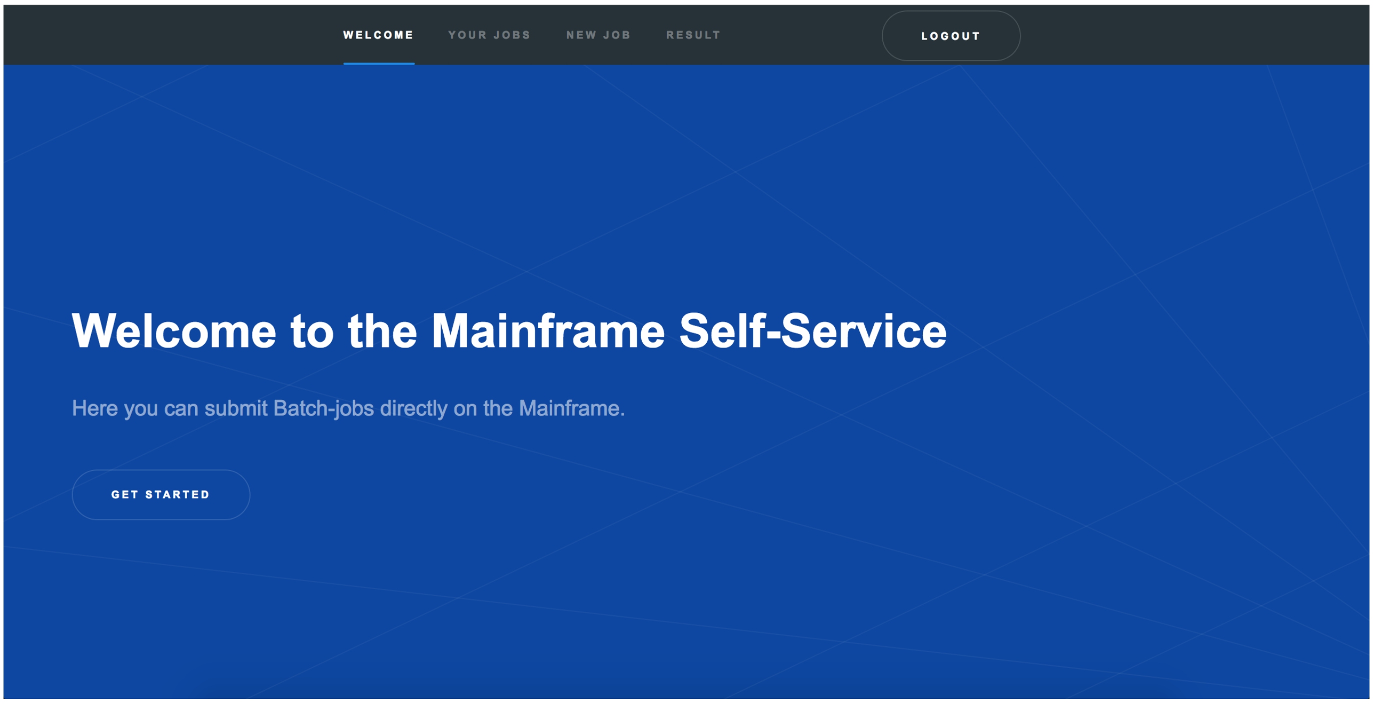
When a user is logged in it gets provided a one-page-layout program, which is devided into four sections. These sections and their functions are explained in the following.
As before mentioned the entry point is the *routes.js*-file. Here the home.jade file is called after login to render the index.html. This is done by the following code.

For a better understanding, the Filename is always listed before the code. As long as it doesn't change throughout the code-listings, it is the same file as before.

```
1  // routes.js
2
3    app.get('/home', function(req, res) {
4      if (req.session.user == null){
5    // if user is not logged-in redirect back to login page //
6        res.redirect('/');
7      } else{
8        res.render('home', {
9          title : 'Control Panel',
10         countries : CT,
11         udata : req.session.user
12       });
13       }
14   });
```

On line 11 you can see that the sessions' userdata is already requested when the program

is loaded. If there is something wrong the user gets redirected to the login-page.
The first section simply welcomes the user to the self service. This is the view, the user sees, when entering the program.



Welcome to the Mainframe Self-Service

Here you can submit Batch-jobs directly on the Mainframe.

Figure 7: Welcome View

10.3 Your Jobs

In this section the user should get an overview on the jobs listed in his JES-Spool. Therefore the index.html contains a simple empty div-container that gets filled with a Table of job-files if he submitted jobs so far.

```
// index.html

        <!-- Here the JES-Spool is listed -->
        <div id="dirout" style="max-height:800px;overflow:auto;">
        </div>
```

To create this table with the wanted content it takes a few steps. In *home.js*, a file that is part of the node-login and therefore is not listed on the figure above, we do a jQuery GET request, when the document is loaded.

```
// home.js
```

```

$(document).ready(function(){

//Get Job list
  $.get("/data-list", function(list) {
  // give the content 'list' in form of html to a div with the ID 'dirout'
    $("#dirout").html(list);
  });
```

This requests data from the */data-list* route. Here it requests a variable *list* which it wants to give to the #dirout-div in form of html.
The route that is requested looks like this:

```
1  // routes.js
2
3  // Establish the sftpManager as variable to use it in route
4  var SF = require('./modules/sftpManager');
5
6  // Route for getting job-list
7    app.get('/data-list', function(req, res) {
8    // pass user-data to function getList()
9        SF.getList(req, res, connSettings = {
10         host: '141.2.192.32',
11         port: 22,
12         username: req.session.user.user,
13         password: req.session.user.pass,
14       });
15
16   });
```

Here the *sftpManager.js* is defined as a variable to use it in the route (line 1). The route itself uses this variable to get the sftpManager to trigger the function getList() along with the user-data located in *connSettings*.

The Jobs that were submitted, are located in a directory on z/OS called "//-JES" from here they can be passed to the container in the form of a table.
The command that is usually used to read data in this directory on the Co:Z SFTP-Server is:

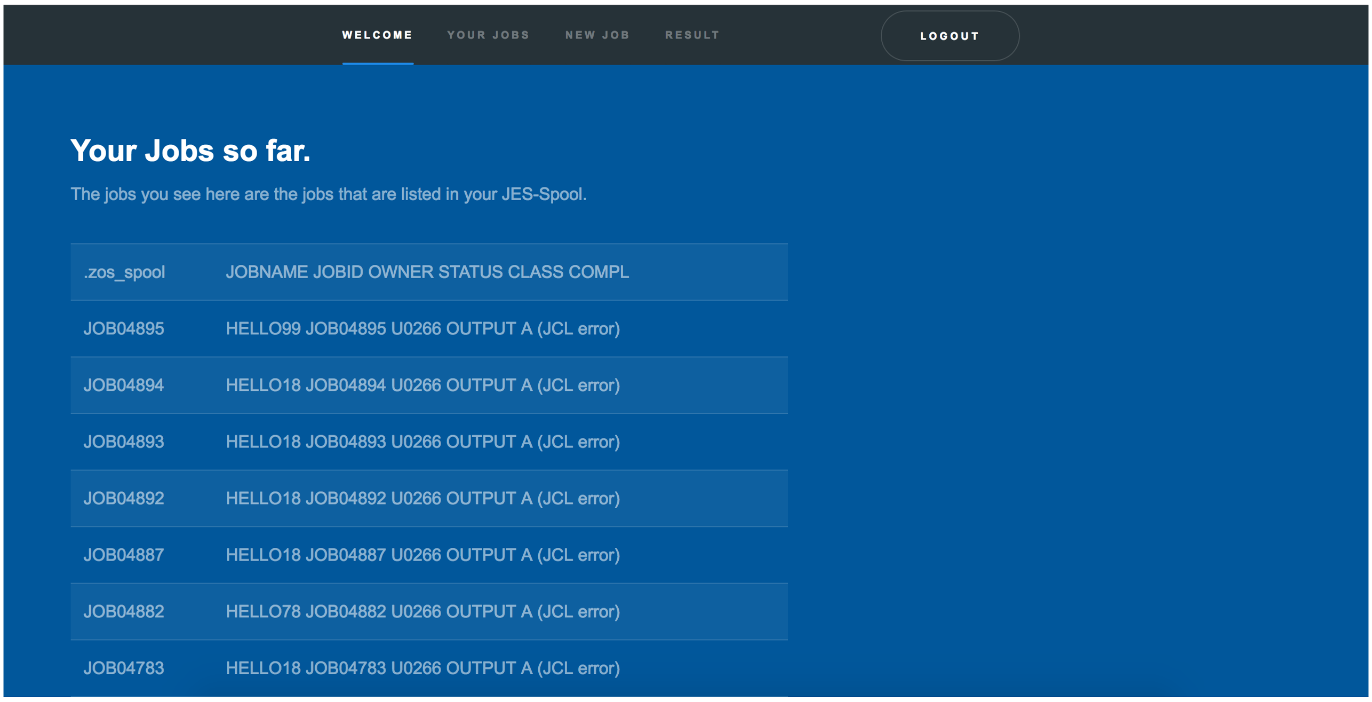
```
sftp>  ls -al //-JES
```

This command leads to a list of all spool-files and some information to them in form of a table in a shell or terminal. To trigger this kind of listing, the ssh2-library provides the command *readdir* to list the requested directory.

With the following function in the sftpManager, the jobs are passed to the application.

```
//sftpManager.js

  getList(req, res) {
    var remotePathToList = '//-JES';          //JES-Spool Listing

    var conn = new Client();         // Establish SFTP Connection
    conn.on('ready', function() {
        conn.sftp(function(err, sftp) {
          if (err) {
            //send error to myController if something went wrong
            res.send(400);
          } else {

    // read the //-JES directory
              sftp.readdir(remotePathToList, function(err, list) {
                     if (err) throw err;

                     // Return list
                     var rows = list;

    //Creata a table and fill with job-list
                     var jobs = "<table id='jobtable' border='1|1'>";
                     for (var i = 0; i < rows.length; i++) {
                         jobs+="<tr>";
                         jobs+="<td>"+rows[i].filename+"</td>";
                         jobs+="<td>"+rows[i].longname+"</td>";

                         jobs+="</tr>";
                     }
                     jobs+="</table>";

    // send the job-table
                     res.send(jobs);

                    // We're just calling the list once,
                    // so the connection has to be ended now
                     conn.end();
              });
            }
        });
    }).connect(connSettings);
  },
```

Through this, the jQuery GET request mentioned at the beginning gets a variable *list* that contains all spool-files in form of a html table. This table is given to the #dirout-div and is shown in the browser.



Your Jobs so far.

The jobs you see here are the jobs that are listed in your JES-Spool.

.zos_spool	JOBNAME JOBID OWNER STATUS CLASS COMPL
JOB04895	HELLO99 JOB04895 U0266 OUTPUT A (JCL error)
JOB04894	HELLO18 JOB04894 U0266 OUTPUT A (JCL error)
JOB04893	HELLO18 JOB04893 U0266 OUTPUT A (JCL error)
JOB04892	HELLO18 JOB04892 U0266 OUTPUT A (JCL error)
JOB04887	HELLO18 JOB04887 U0266 OUTPUT A (JCL error)
JOB04882	HELLO78 JOB04882 U0266 OUTPUT A (JCL error)
JOB04783	HELLO18 JOB04783 U0266 OUTPUT A (JCL error)

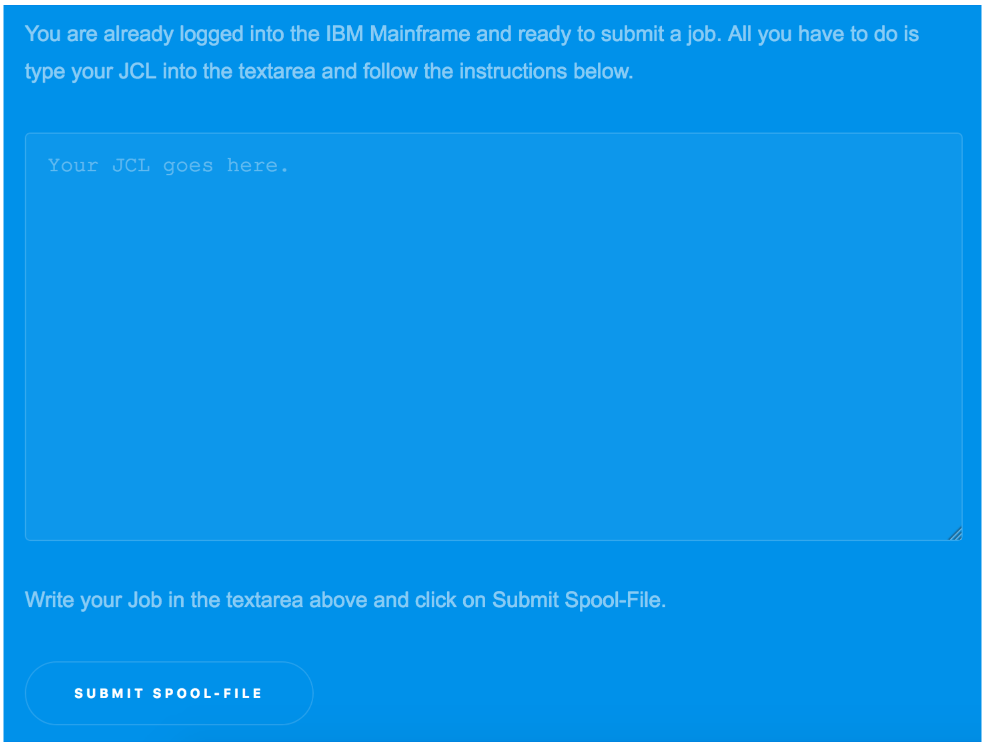
Figure 8: List of JES-Spool files

10.4 Submit a new Job

As FTP is short for File-Transfer-Protocol, there first has to be a file to be transferred. Node for this situation provides a module called *file-system*. With this module it is possible to create a .txt file through a textarea.

```
// index.html

        <div class="field">
          <textarea name="content" id="JCLcontent"
          placeholder="Your JCL goes here." rows="10" cols="50"></textarea>
        </div>
        <div class="field" style="margin-top:30px;">
          <p id="filesuccess">Write your Job in the textarea above
          and click on Submit Spool-File.</p>
          <button id="createSpool" onclick="createSpoolfile()">
          Submit Spool-File</button>
          <div id="messages"></div>
        </div>
```

Here the JCL-code can be written. Beneath the textarea, there is a Button saying


You are already logged into the IBM Mainframe and ready to submit a job. All you have to do is type your JCL into the textarea and follow the instructions below.

Write your Job in the textarea above and click on Submit Spool-File.

Figure 9: Textarea for typing JCL-Jobs

"Submit Spool-File" that has the onclick-event "createSpoolfile()", which triggers the following function.

```
//myController.js

function createSpoolfile() {
  var filecontent = $("#JCLcontent").val();
  var filesuccess = document.getElementById("filesuccess");
  var resultinfo = document.getElementById("resultinfo");

  //Check if textarea contains content
  if (!filecontent) {

    filesuccess.style.color = "#ff9800";
    filesuccess.innerHTML = "Please write a Job first.";

  } else {

    filesuccess.style.color = "#00e676";
    filesuccess.innerHTML = "Your Job is on its way to the mainframe. Scroll down to see

  //Define value of textarea as json-object
  var json = {
    content: filecontent,
  };

  // if it does call create-file in routes
  $.ajax({
    type: 'POST',
    url: '/create-file',
    data: json,

    ... to be continued in chapter 10.5.
```

The function createSpoolfile() that is triggered through the button first checks if the textarea even contains content. If it does not, it advises the user to "write a job first". If the textarea has content, the function here defines this value as a json object and does an ajax-call. This ajax-call of the type POST sends the information as a json-object to the url: /create-file in routes.js.

```
//routes.js

  // Route for creating and submitting file
  app.post('/create-file', function(req, res) {
    SF.createFile(req, res, connSettings = {
      host: '141.2.192.32',
      port: 22, // Guest-port is 22
      username: req.session.user.user,
      password: req.session.user.pass,
    });
  });
```

This route again triggers a function in the sftpManager.js and sends the data for the connection to the mainframe. The difference to the above route is that here we have a POST-routing which means we don't request data but we send data to the mainframe. This happens in the function createFile.

```
//sftpManager.js

  createFile(req, res) {
    var filedir = './Spool-files/' + req.session.user.user;
    var filepath = './Spool-files/' + req.session.user.user + '/JCL.txt';
    var fileContent = req.body.content;

    if (fileContent != '') {

```

```
10       var dir = './Spool-files/' + req.session.user.user;
11 
12       //Create User-folder for JCL files
13       if (!fs.existsSync(dir)){
14         fs.mkdirSync(dir);
15       };
16 
17       // When folder exists, create textfile
18       fs.writeFile(filepath, fileContent, (err) => {
19         if (err) {
20           throw err;
21         } else {
22           this.transFile(req, res, connSettings);
23         }
24       });
25     }
26   },
```

createFile() first checks if the user already has an own directory for his Spool-files to be submitted. If not it creates a subdirectory in the directory 'Spool-files' with the users name. Here it writes a .txt-file with the value of the textarea. After that it triggers the transFile()-function which is also located in the sftpManager. This time it needs no routing because the createFile()-function already has the connSettings variable and passes it to the transFile()-function (line 22).

As the transfer mode is set to binary by default, the .txt file would not be recognized by the JES-Spool if it would be submitted now. To change this, it is necessary to change the transfer mode to *text*. The Co:Z transfer mode can be changed by executing:

```
sftp>  ls /+mode=text
```

/+mode=text is a subdirectory. that simply has to be read. Like in the section before, the *readdir*-function is used to read this subdirectory and with this, change the transfer-mode. Now the file is ready to be transferred to the JES-SPOOL. The command for this action would usually be:

```
put <local file-path> //-JES.INTRDR/MYJOB
```

The ssh2 client executes this by placing a readStream (get) to get the file that has just been created and a writeStream (put) to send this file to the JES subdirectory *MYJOB*.

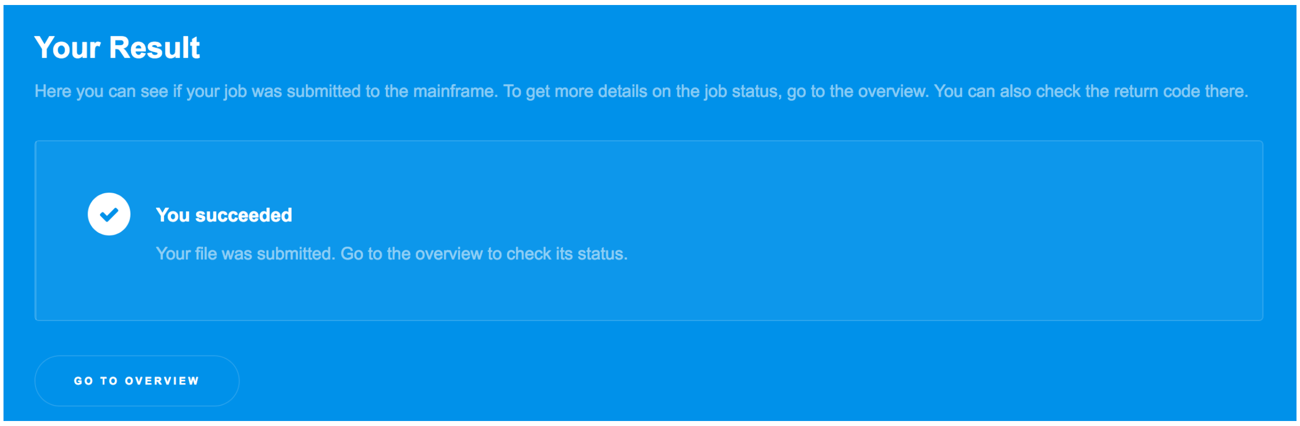
```
//sftpManager.js

  transFile(req, res) {
    var changeMod = '/+mode=text'

    var conn = new Client();
      conn.on('ready', function() {
        console.log('Client -> ready');
          conn.sftp(function(err, sftp) {
                if (err) {
                  //send error to myController if something went wrong
                  res.send(400);
                } else {

                  // change Transfer Mode with ls /+mode=text
                  sftp.readdir(changeMod, function(err, list) {
                          if (err) throw err;
                          // List the Changing in the console
                          console.log('mode -> text');
                          // This time no connection closing, to get the output-action
                  });

                  var fs = require("fs"); // Use node filesystem

                  //Read the textfile and send it to subdirectory, called "myjob"
                  var readStream = fs.createReadStream( './Spool-files/' + req.session.us
                  var writeStream = sftp.createWriteStream( '//-JES.INTRDR/MYJOB' );

                  writeStream.on('close',function () {
                  });

                  writeStream.on('end', function () {
                      console.log( "sftp connection closed" );
                      conn.close();
                  });

                  // initiate transfer of file
                  readStream.pipe( writeStream );
                  //send success to myController
                  res.send(200);
                  console.log( "file transferred succesfully" );
                }
              });
          }).connect(connSettings);
  },
};
```

10.5 Result

After this all has happened *myController.js* gets a response in form of 200=success or 400=error and performs the following tasks.

```
//myController.js

function createSpoolfile() {

  [...]

  $.ajax({
    type: 'POST',
    url: '/create-file',
    data: json,
    success: function() {
      resultinfo.style.display = "none";
      document.getElementById("goodresult").style.display = 'block';
    },
    error: function() {
      resultinfo.style.display = "none";
      document.getElementById("badresult").style.display = 'block';
    }
  });
```

Depending on the job submit was successful or not, predefined div containers are shown instead of the usual resultinfo-div that advise the user to check the connection in case of an error or show him a success message like this one in case he or she succeeded.


Your Result

Here you can see if your job was submitted to the mainframe. To get more details on the job status, go to the overview. You can also check the return code there.

You succeeded

Your file was submitted. Go to the overview to check its status.

Figure 10: Success Message

By clicking the button *Go to overview* the page refreshes, what triggers the functions of chapter 10.3 again to get a list containing the new job.

11 Learning Effect

12 Proof of Concept

13 Evaluation

References

[1] Elias G Carayannis and Yiannis Nikolaidis. Enterprise networks and information and communication technologies (ict) standardization.

[2] Bill Cope and Mary Kalantzis. Ubiquitous learning: An agenda for educational transformation. *Ubiquitous learning*, pages 3–14, 2009.

[3] A Corridori. Ways to include enterprise computing content in current curriculum paper presented at the enterprise computing conference. 2009.

[4] David Douglas and Christine Davis. Enterprise computing: Bridging the gap to generation-y with rdz and zlinux web. Enterprise Computing Conference, 2009.

[5] Mike Ebbers, Wolfgang Bosch, Hans Joachim Ebert, Helmut Hellner, Jerry Johnston, Marco Kroll, Wilhelm Mild, Wayne O'Brien, Bill Ogden, Ingolf Salm, et al. *Introduction to the New Mainframe: IBM Z/VSE Basics*. IBM Redbooks, 2016.

[6] Gwo-Jen Hwang, Tsai Chin-Chung, and Stephen JH Yang. Criteria, strategies and research issues of context-aware ubiquitous learning. *Journal of Educational Technology &amp; Society*, 11(2), 2008.

[7] Gwo-Jen Hwang, Tzu-Chi Yang, Chin-Chung Tsai, and Stephen JH Yang. A context-aware ubiquitous learning environment for conducting complex science experiments. *Computers &amp; Education*, 53(2):402–413, 2009.

[8] Vicki Jones and Jun H Jo. Ubiquitous learning environment: An adaptive teaching system using ubiquitous technology. In *Beyond the comfort zone: Proceedings of the 21st ASCILITE Conference*, volume 468, page 474. Perth, Western Australia, 2004.

[9] Ravi Khadka, Belfrit V Batlajery, Amir M Saeidi, Slinger Jansen, and Jurriaan Hage. How do professionals perceive legacy systems and software modernization? In *Proceedings of the 36th International Conference on Software Engineering*, pages 36–47. ACM, 2014.

[10] Charles Kiefer. Cobol as a modern language. 2017.

[11] S Lohr. Ibm seeks to make the mainframe modern technology. *New York Times*, 3, 2006.

[12] Steve Lohr. Why old technologies are still kicking. *New York Times*, 4, 2008.

[13] Carol Midgley. *Goals, goal structures, and patterns of adaptive learning*. Routledge, 2014.

[14] Ken Sakamura and Noboru Koshizuka. Ubiquitous computing technologies for ubiquitous learning. In *Wireless and Mobile Technologies in Education, 2005. WMTE 2005. IEEE International Workshop on*, pages 11–20. IEEE, 2005.

[15] Aditya Sharma and Marianne C Murphy. Teach or no teach: Is large system education resurging? *Information Systems Education Journal*, 9(4):11, 2011.

[16] Aditya Sharma, Cameron Seay, and Marilyn K McClelland. Alive and kicking: Making the case for mainframe education marianne c. murphy mmurphy@ nccu. edu.

[17] Robert Vinaja. 50 th aniversary of the mainframe computer: a reflective analysis. *Journal of Computing Sciences in Colleges*, 30(2):116–124, 2014.

[18] I Wallis and B Rashed. Who's watching the mainframe. *IBM Systems Journal*, 2007.

[19] W Wong. Old tech skills again in demand: Mainframe computing jobs vacant as the baby boomers who set up systems begin retiring with few educated to fill the spots. *Chicago Tribune*, 2009.

14 Appendices