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:

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

DD Data Definition

JES Job Entry Subsystem

TCP/IP Transmission Control Protocol / Internet Protocol

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

MVS Multiple Virtual Storage

CP Control Program

CMS Content Management System

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 web 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 [14]. Till the present day the mainframe is the backbone of the financial markets worldwide and just as important for other big industries[1] [13]. 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 [20] [21] [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 " [17]. 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 [18].

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 [9]. 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 [15] , " Ubiquitous Learning Environment (ULE) " describes an environment with the possibility of learning everywhere and anytime[7]. Hwang et al. emphasize the fact, that ULEs are an innovative approach for teaching complex topics [8]. To establish a ULE there are many technologies needed [16], 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 [12]
Khadka, How do professionals perceive legacy systems and software modernization? [11]
Vinaja, 50th anniversary of the mainframe computer: a reflective analysis [19]

.....

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].

The most common utilization of the mainframe is divided into two categories:

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

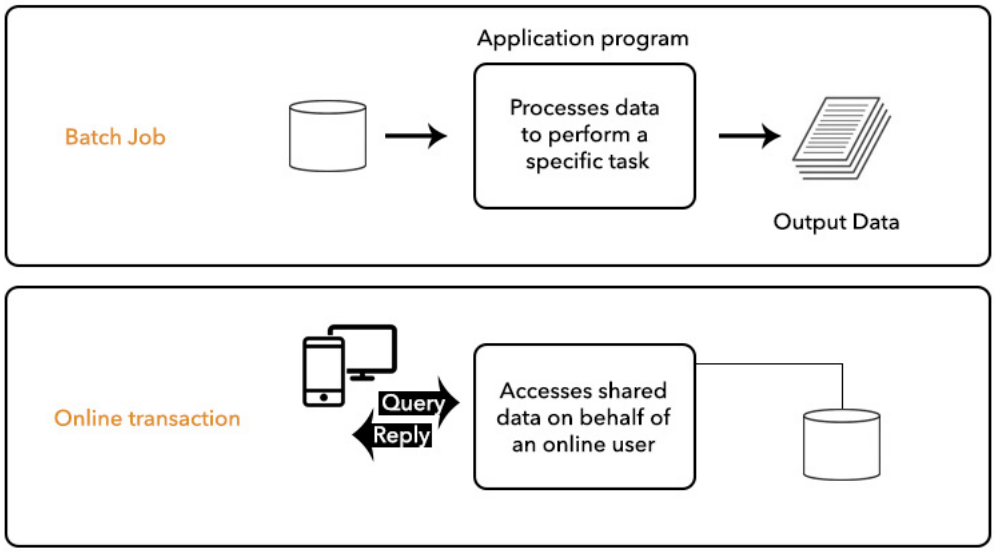

Figure 1: Typical workloads

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 to also 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 group of this thesis in more detail, this chapter will give a quick overview on the future users. For this, a few staff-roles will be explained, which are separated in their duties in large mainframe installations.
Kettner et al. [10] distinguish between the "members of the system programming staff" and the "system programming department and other IT departments in the IT organisation" and describe this separation as an "audit requirement" that is to avoid having one person with too much power in the system.
A few of the duties that are described by IBM [5] are the following:

- **System programmers** - install, customize, and maintain the operating system, and also install or upgrade products that run on the system
- **System administrators** - perform more of the day-to-day tasks related to maintaining the critical business data that resides on the mainframe (for example, DBA, storage, network, security, and performance)
- **Application designers and programmers** - design, build, test, and deliver mainframe applications for the company's end users and customers
- **System operators** - monitor and control the operation of the mainframe hardware and software
- **Production control analysts** - making sure that batch workloads run to completion - without error or delay

So the main focus here is on the the *production control analysts* while the keywords are *without error or delay*. The upcoming program will help future users to strengthen their abilities to write batch jobs till they are ready to work this out in a large mainframe installation without making any mistakes.

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 *z-Series*.
IBM *z-Series* describes a mainframe architecture for large scale systems where the *z* stands for *zero Downtime*.
The following operating systems are designed for this kind of mainframe architecture:

- **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 Job Entry Subsystem (JES) for the background operations (batch processing), the Time Sharing Option (TSO) for the foreground operations (interactive) and the Unix System Services (USS), which represent a posix-compatible unix subsystem.

- **z/VSE** is the system that is used for OLTP. It is based on web solutions to integrate OLTP in a heterogeneous environment.
- **z/TPF** works as a transaction processor and a database in one operating system. It is used as OS for high data-throughput as needed for example in airline IT-systems where it also has to be available 24/7.
- **Linux for zSeries** is a special Linux OS, compiled to be run on IBM mainframes, especially the IBM z-Series and the IBM LinuxONE servers.
- **z/VM** This system allows running thousands of distributed servers on one server at a time, which becomes possible through its huge virtualization capabilities. With its two components, a Control Program (CP) and a single-user operating system (a Content Management System (CMS)) it can run any of the IBM operating systems mentioned above but also guest systems such as Ubuntu, which will be the operating system to be used as a server for this application.

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 production control analysts are usually working in ultimate contact to the mainframe. The future users therefore shell be able to access a front-end running on a server that is located in the mainframe and from here execute tasks that take place on this mainframes' z/OS.

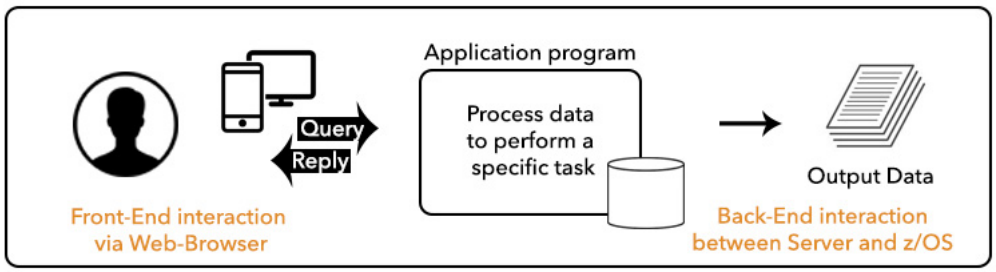

Figure 2: Scheme of usage

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.
To gain added value on an educational basis there will be some features that help the users understand the functions of the Batch-jobs as well as writing the jobs.

This setup will have the ambitions of getting people into the Job Control Language (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

To get familiar with JCL, this chapter will give an introduction on 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.

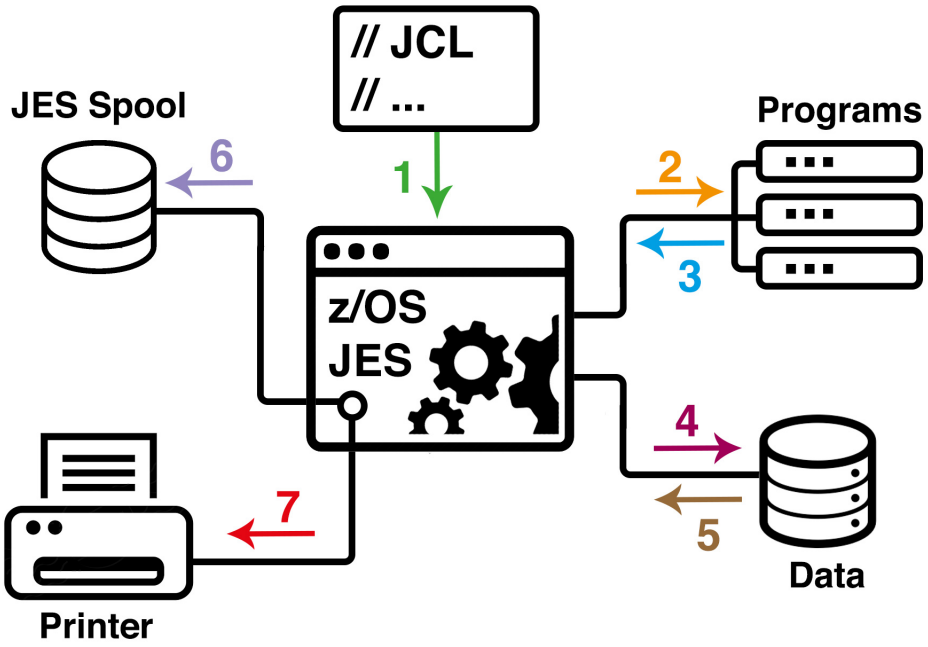

Figure 3: JCL Functionality, Source: IBM

On figure 3 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 (e.g. IEBGENER), which include internal file-names. Through JCL those programs can be opened for reading and writing during execution.
The 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

The job creation is defined of its structure. In general a job consists of two major parts -the job card, also known as job statement and the job steps or also called job body. They are all introduced with a double slash.

7.2.1 The Job-statement

The job statement identifies the job as JCL and calls the unit of work for the operating system to perform.

```
e.g.
//HELLO18 JOB 2
//...
```

7.2.2 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
...
//STEP0006 EXEC PROC=PRDPROC1
//STEP0007 EXEC PROC=PRDPROC2
```

7.2.3 DD-Statements

Through Data Definition (DD) statements the data-sets that are used while running the program are defined. There has to be a DD statement for every data set that is used or created within a job.

```
1 //SYSIN DD DUMMY
2 //SYSPRINT DD SYSOUT=X
3 //SYSUT1 DD *?Hello world!
4 /*
5 //SYSUT2 DD DISP=(NEW,CATLG),DSN=<USERID>.HELLO.DATA
```

- **SYSIN** Specifies the control dataset
- **SYSPRINT** Specifies a message output dataset
- **SYSUT1** Specifies the input dataset(s)
- **SYSUT2** Specifies the output dataset

As it can be seen here the control datasat is defined as a dummy. After that there is a message output dataset defined with the input *Hello World!* that is displayed afterwards.

7.3 The Job Entry Subsystem

The purpose of the Job Entry Subsystem is to automate the batch jobs. It is a special system component provided by the z/OS which is responsible for controlling and processing all the jobs running on the system (incl. TSO).

The JES therefore handles the JCL in three steps as follows:

- **Preprocessing** Reading + interpreting the JCL-records and providing the required datasets
- **Processing** Performing the job steps
- **Postprocessing** The data created through the job gets printed or it gets supplied to the user for further processing

So while the JES is that part of z/OS which is responsible for reading and processing the JCL that shall be transmitted from the front-end it should work as a contact system to be addressed by the application. The next chapters will introduce the way how to contact the Job-Entry-Subsystem to be able to submit jobs directly to it.

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 Transmission Control Protocol / Internet Protocol (TCP/IP), which is provided through the HiperSockets network. 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.
You could imagine a HiperSockets-network like an internal Local Area Network (LAN), linking all partitions for internal communication.

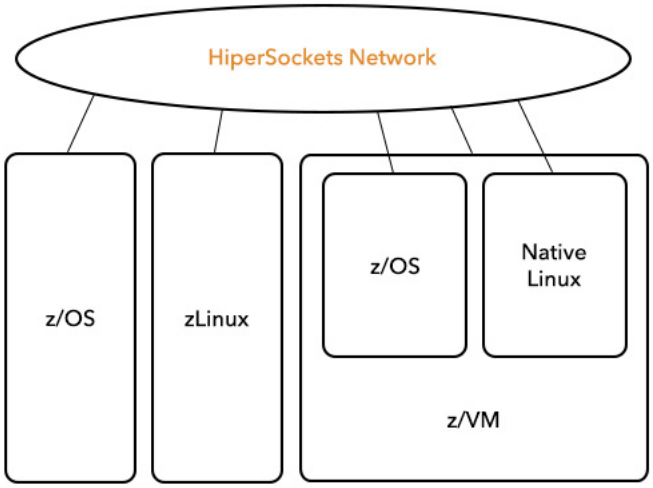

Figure 4: Hipersockets

—- 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 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

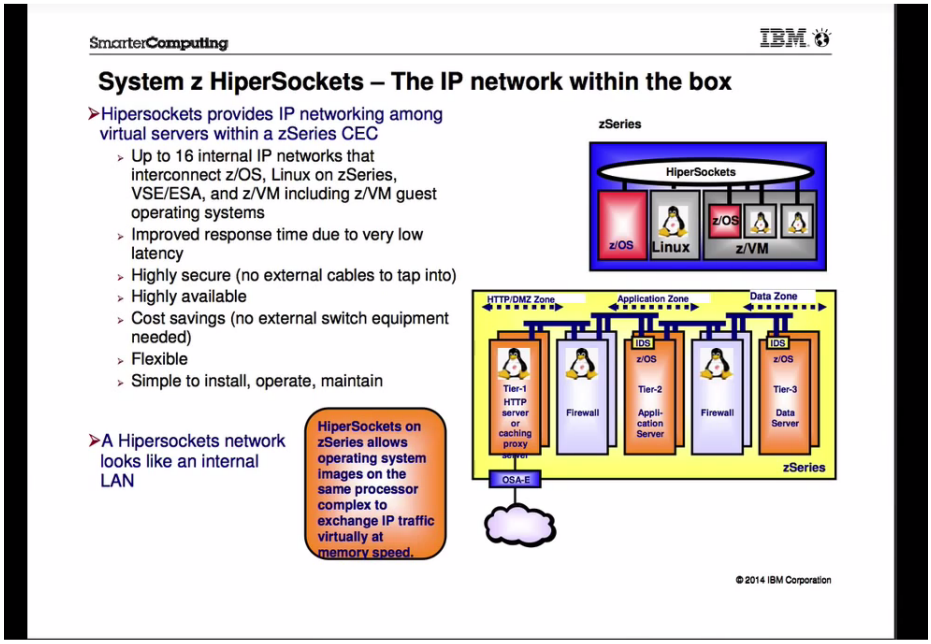

Figure 5: TCP/IP Connection / Hipersockets

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)

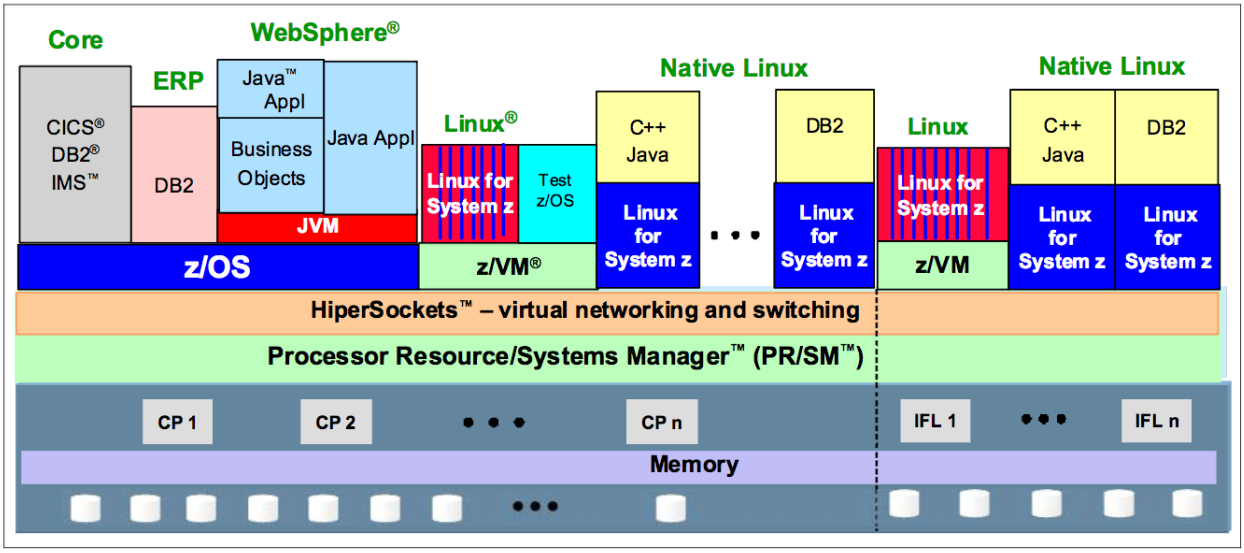

Figure 1-1 Multiple workloads on the mainframe

Figure 6: workloads

- 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. 7 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.

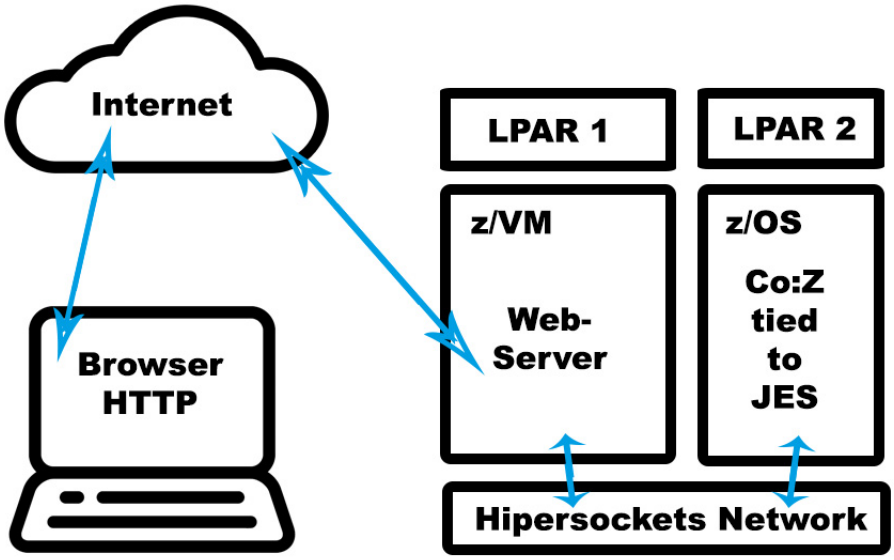

Figure 7: constellation

`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. 8. 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. 8 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. 9, 10, 11, 12)
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.

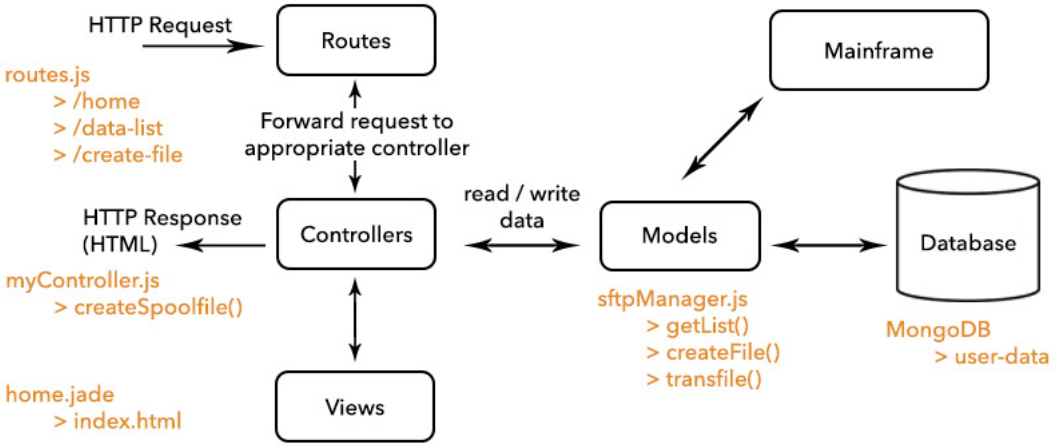

Figure 8: Model-View-Controller Scheme

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.

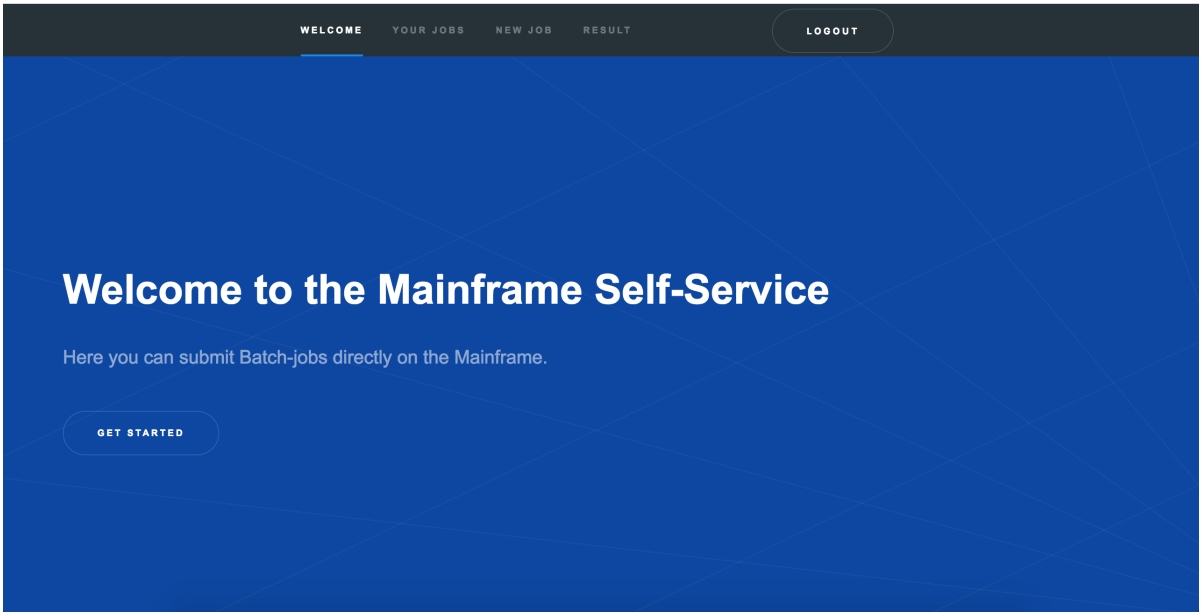

Figure 9: Welcome View

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) {
```

```
7   // give the content 'list' in form of html to a div with the ID 'dirout'
8     $("#dirout").html(list);
9   });
```

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:

```
1  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.

```
1  //sftpManager.js
2
```

```
3   getList(req, res) {
4     var remotePathToList = '//-JES';          //JES-Spool Listing
5
6     var conn = new Client();          // Establish SFTP Connection
7     conn.on('ready', function() {
8         conn.sftp(function(err, sftp) {
9           if (err) {
10            //send error to myController if something went wrong
11            res.send(400);
12          } else {
13
14    // read the //-JES directory
15             sftp.readdir(remotePathToList, function(err, list) {
16                    if (err) throw err;
17
18                    // Return list
19                    var rows = list;
20
21    //Creata a table and fill with job-list
22                    var jobs = "<table id='jobtable' border='1|1'>";
23                    for (var i = 0; i < rows.length; i++) {
24                        jobs+="<tr>";
25                        jobs+="<td>"+rows[i].filename+"</td>";
26                        jobs+="<td>"+rows[i].longname+"</td>";
27
28                        jobs+="</tr>";
29                    }
30                    jobs+="</table>";
31
32    // send the job-table
33                    res.send(jobs);
34
35                  // We're just calling the list once,
36                  // so the connection has to be ended now
37                    conn.end();
38              });
39           }
40        });
41    }).connect(connSettings);
42  },
```

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.

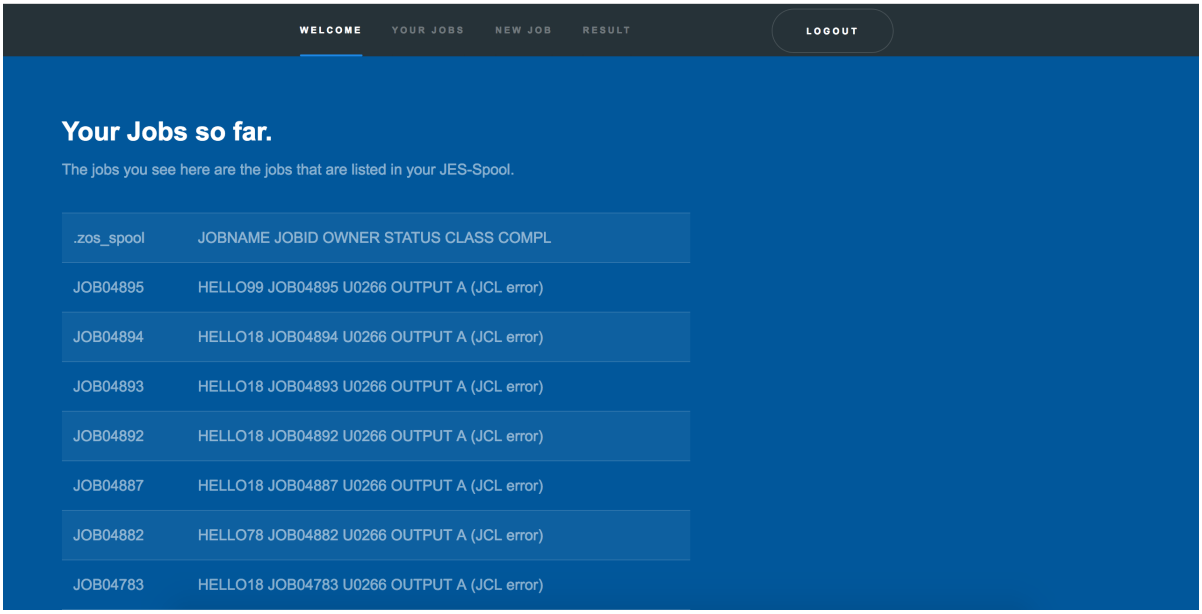

Figure 10: List of JES-Spool files

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

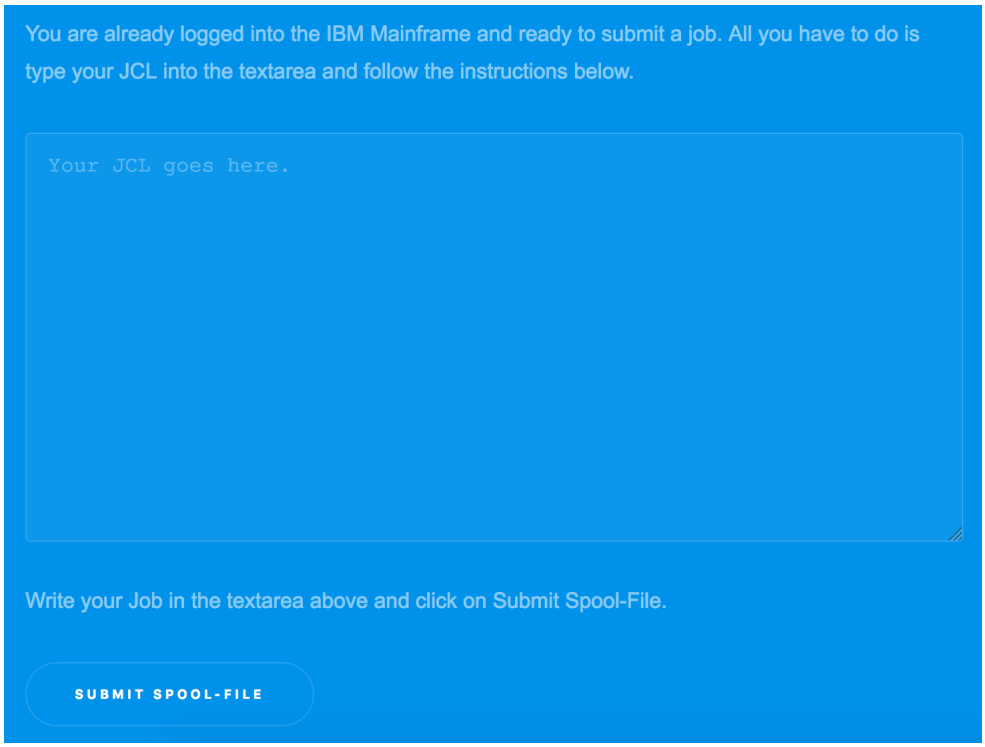

Figure 11: Textarea for typing JCL-Jobs

```
1  //myController.js
2
3  function createSpoolfile() {
4    var filecontent = $("#JCLcontent").val();
5    var filesuccess = document.getElementById("filesuccess");
6    var resultinfo = document.getElementById("resultinfo");
7
8    //Check if textarea contains content
9    if (!filecontent) {
10
11     filesuccess.style.color = "#ff9800";
12     filesuccess.innerHTML = "Please write a Job first.";
13
14   } else {
15
16     filesuccess.style.color = "#00e676";
17     filesuccess.innerHTML = "Your Job is on its way to the mainframe.
18         Scroll down to see the result.";
19
20   //Define value of textarea as json-object
21   var json = {
22     content: filecontent,
23   };
24
```

```
25  // if it does call create-file in routes
26  $.ajax({
27    type: 'POST',
28    url: '/create-file',
29    data: json,
30
31    ... 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.

```
1  //routes.js
2
3    // Route for creating and submitting file
4    app.post('/create-file', function(req, res) {
5      SF.createFile(req, res, connSettings = {
6        host: '141.2.192.32',
7        port: 22, // Guest-port is 22
8        username: req.session.user.user,
9        password: req.session.user.pass,
10     });
11   });
```

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.

```
1  //sftpManager.js
2
3    createFile(req, res) {
4      var filedir = './Spool-files/' + req.session.user.user;
5      var filepath = './Spool-files/' + req.session.user.user + '/JCL.txt';
6      var fileContent = req.body.content;
7
8      if (fileContent != '') {
9
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 transfermode. 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
```

```
12                  res.send(400);
13                } else {
14
15                  // change Transfer Mode with ls /+mode=text
16                  sftp.readdir(changeMod, function(err, list) {
17                         if (err) throw err;
18                         // List the Change in the console
19                         console.log('mode -> text');
20                  });
21
22                  var fs = require("fs"); // Use node filesystem
23
24                  //Read the textfile and send it to
25                  //subdirectory, called "myjob"
26                  var readStream =
27                     fs.createReadStream( './Spool-files/'
28                     + req.session.user.user + '/JCL.txt' );
29                  var writeStream =
30                     sftp.createWriteStream( '//-JES.INTRDR/MYJOB' );
31
32                  writeStream.on('close',function () {
33                  });
34
35                  writeStream.on('end', function () {
36                      console.log( "sftp connection closed" );
37                      conn.close();
38                  });
39
40                  // initiate transfer of file
41                  readStream.pipe( writeStream );
42                  //send success to myController
43                  res.send(200);
44                  console.log( "file transferred succesfully" );
45                }
46              });
47            }).connect(connSettings);
48    },
49  };
```

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.

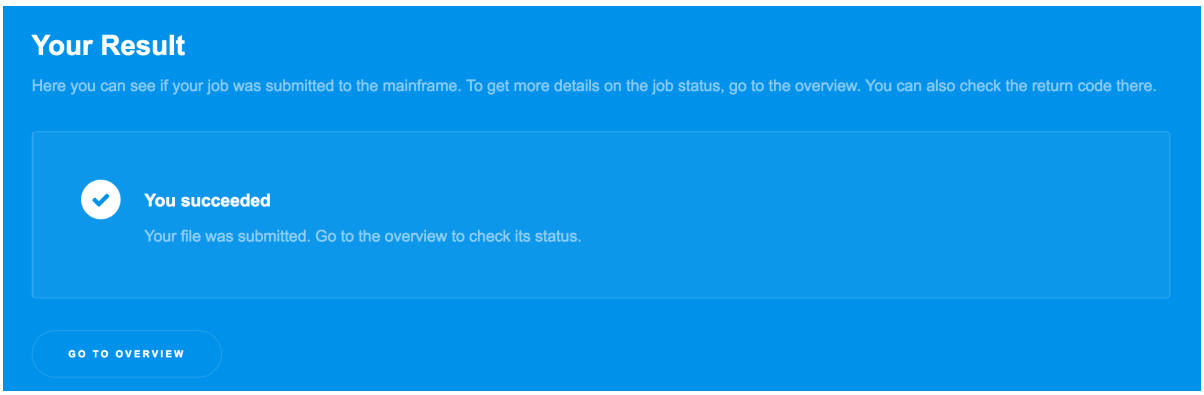

Figure 12: Success Message

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

Since the application in first place is designed to give students and system operators in general the ability to learn submitting batch-jobs, the functionality itself is not enough. This chapter will describe three techniques to give the user a learning effect.
The following functions will take place in a file called *content-functions.js* that is running on the frontend and is written in jQuery.

11.1 Information on MVS Utilities

Before creating a job, the user shall have the option to know, which effect it will have after submitting. For this the user can choose between the different Multiple Virtual Storage (MVS) Utilities the mainframe offers and get an overview on what this utility is able to do.
In this case, a select-field containing those nine utilities will on change reveal details regarding to this program. The informations on the utilities come from the essential mainframe guide by Mitra Gopaul [6].

```
1  //index.html
2
3  <div class="select-wrapper" style="margin-bottom:20px;">
4  <select id="mySelect" class="jobhint">
5    <option value="" selected disabled hidden>
6      Choose MVS Utility to get some introductions.</option>
7    <option name="step1">IEBGENER</option>
8    <option name="step2">IEBPTPCH</option>
9    <option name="step3">IEBCOPY</option>
10   <option name="step4">IEBUPDTE</option>
11   <option name="step5">IEBCOMPR</option>
12   <option name="step6">IEHPROGM</option>
13   <option name="step7">SORT</option>
14   <option name="step8">IEWL</option>
15   <option name="step9">IGYCRCTL</option>
16 </select>
17 </div>
```

The *index.html* contains a select-field where the user can choose the program he wants to use for his job.
If he chooses to the program IEBGENER for example, the select field reveals the information that is written in the div with the ID IEBGENER that is shown below.

```
//index.html

<div id="IEBGENER" class="infobox">
  <h3>IEBGENER</h3>
  <p>This utility has many functions. They are:</p>
  <p><b>1. TRANSFER DATA</b></p>
  <ul style="margin-left:20px;">
    <li>From a sequential dataset to a member of a
      partitioned dataset.</li>
    <li>From a member of a partitioned dataset to a
      sequential dataset.</li>
    <li>From a tape file to a disk file.</li>
    <li>From a disk file to a tape file.</li>
  </ul>
  <p><b>2. PRINT A FILE.</b> IEBGENER can send the contents of a
    sequential or PDS file to a printer.</p>
  <p><b>3. REORGANIZE A FILE.</b> IEBGENER can be used to expand a
    partitioned dataset. Also, it can be used to produce an "edited"
    sequential dataset or PDS member. Some of the
      editing functions are:</p>
  <ul style="margin-left:20px;">
    <li>Rearrange or omit data fields in a record.</li>
    <li>Replace data with literal values.</li>
    <li>Convert data from packed decimal to unpacked decimal and</li>
    <li>from unpacked decimal to packed decimal</li>
  </ul>
  <p><b>4. CHANGE FILE CHARACTERISTICS.</b> IEBGENER can be
    used to "reblock" a file. For example a file with LRECL=80,
    BLKSIZE=800 can be copied to a file with LRECL=80, BLKSIZE=8000.</p>
</div>
```

The revealing of this div is accomplished through the following jQuery-function.

```
//conent-functions.js

// Change infoboxes on select
$(function() {
      $('#mySelect').change(function(){
          $('.infobox').hide();
          $('.infosteps').hide();
          $("#stepside").addClass('split');
          $('#' + $(':selected',this).attr('name')).show();
          $('#' + $(this).val()).show();
      });
  });
```

After checking which utility is selected this function first hides infoboxes if there were any selected before. Than it reveals the content of the div-container above.

As it can be seen here there are not only the infoboxes that are shown on change but also the additional steps that have to be done to get this job running.
As these steps are slightly different for every program they are also changed by selecting another utility. For this the above function uses the attribute *name* and searches for a div with the deposited value as ID.
By adding the class *split* to the container containing the textarea, those steps to implement the job are shown directly next to the textarea. Details to the steps become evident by hovering the specific step as you can see on Fig. 13

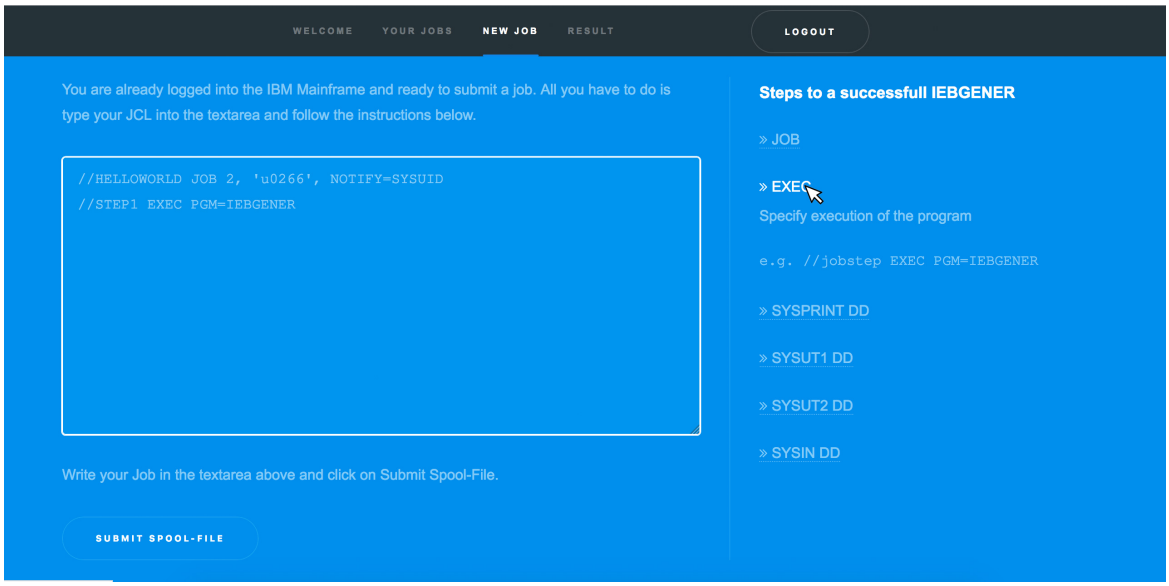

Figure 13: Reveal job-step by hovering

11.2 Return Codes

After a job was submitted, a few jobs provide a code that describes the status of the process. The user here again will have the ability to read those return codes by selecting the utility he chose based on the job he submitted before. With this he can find out whether his job was successful or not and if not, what went wrong. This option should give the possibility to make mistakes but learn from them immediately.

```
1 //index.html
2
```

```
<div id="codearea">
  <h5>Return Codes</h5>
  <h7>A few utilities provide a return code after you submitted the
   job. Their meaning can be looked up here.</h7>
  <select id="codeSelect">
    <option value="" selected disabled hidden>Get the return codes
     to a specific utility.</option>
    <option value="code1">IEBGENER</option>
    <option value="code2">IEBPTPCH</option>
    <option value="code3">IEBCOPY</option>
    <option value="code4">IEBUPDTE</option>
    <option value="code5">IEBCOMPR</option>
  </select>

  //for example the IEBGENER return codes:
  <div class="codes" id="code1">
     <ul>
      <li><b>00</b> The processing is successfully completed.</li>
      <li><b>04</b> The processing is completed but a
       warning message is issued.</li>
      <li><b>08</b> The processing is terminated.
       Processing of header labels only is requested.</li>
      <li><b>12</b> The program terminated;
       a fatal error occurred.</li>
      <li><b>16</b> The program terminated;
       a user routine returned error code 16.</</li>
    </ul>
  </div>
```

Similar to the infoboxes, a jQuery function reveals the return codes requested by the user based on the job utility he has chosen.

```
//content-functions.js

// Change return codes on select
$(function() {
      $('#codeSelect').change(function(){
          $('.codes').hide();
          $('#' + $(this).val()).show();
      });
  });
```

The content again originates from [6]. The return codes for a IEBGENER for example are shown as followed:

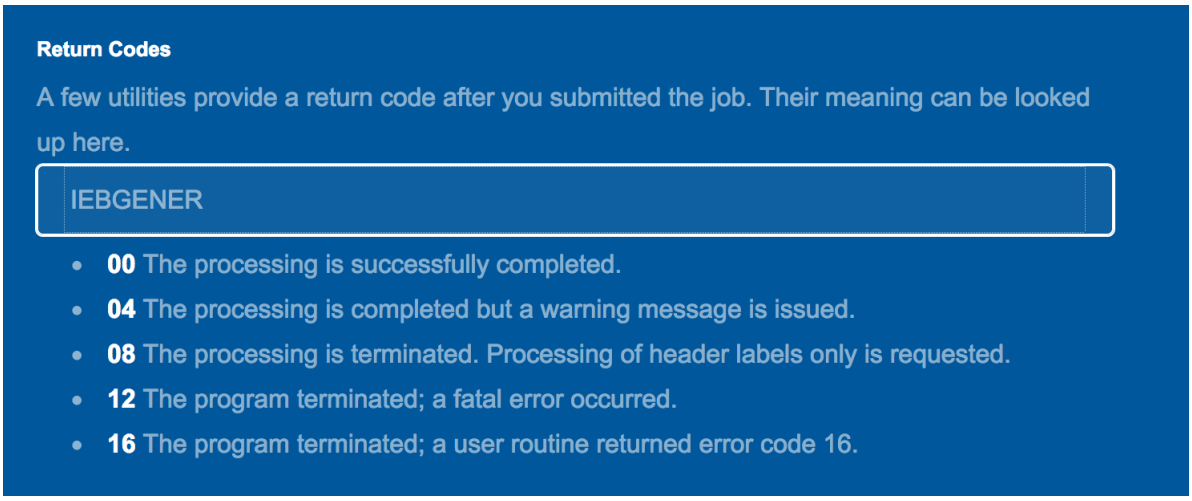

Figure 14: Return codes for IEBGENER

11.3 Gamification

Last but not least the users progress shall be tracked throughout the whole time he or she uses this program. The fact that all the jobs that have been submitted are located in one subdirectory and are shown in this program gives the possibility to count those jobs and based on this classify the training progress into several levels.
—— Describe gamification deeper — with sources ————
To let the user know which level it currently is in or which level it can expect within the next few steps, an area right beneath the job list is implemented. Here a few sentences are written down in combination with an icon to each level to encourage the user to use the program more often and level up.

```
1  //index.html
2
3  <section>
4      <div class="content">
5        <div class="inner">
6          <h2 id="lvlprogress">Training progress</h2>
7          <p id="joblistout">Show me your jobs and I tell you
8           how good you are.</p>
9            <ul class="actions">
10           <li><a href="#two" class="button">Create a New Job</a></li>
11           </ul>
12       </div>
13     </div>
14     <a id="lvllink" class="image"><img id="lvlicon"
15       src="assets/css/images/apprentice.svg" alt="Level image data"
16    data-position="center center" /></a>
17 </section>
```

To change this area based on the users progress the following function counts the jobs that are listed in the users' JES-Spool and changes the content of the area above.

```
// Training progress
$( document ).ready(function() {
  var existCondition = setInterval(function() {
    if ($('#jobtable').length) {
    console.log('ready to count');
    clearInterval(existCondition);

      var i = $('#jobtable >tbody >tr').length;
      var g = $('#joblistout'); //Change text
      var l = $('#lvlprogress'); //Change level
      var p = $('#lvlicon'); //Change icon
      var u = $('#lvllink'); //Change link that contains icon


      if (i == 0) {
      g.text("Oh welcome rookie. You didn't submit any Jobs yet.
          Go ahead and learn JCL by submitting your first Job.");
      l.text("Traingslevel: Rookie")
    } else if (i == 1) {
      g.text("Congratiulations you created your first job.");
      l.text("Traingslevel: Rookie")
    } else if (i < 5) {
      var n = 5-i;
      g.text("Wow. You have submitted " + i + " Jobs so far.
          Only " + n + " Jobs left to be an apprentice.");
      l.text("Traingslevel: Rookie")
    } else if (i == 5) {
      g.text("Congratiulations. You have submitted " + i + " Jobs.
          You are an apprentice now.");
      l.text("Traingslevel: Apprentice")
    } else if (i > 5, i < 10) {
      var n = 10-i;
      g.text("Wow. You have submitted " + i + " Jobs so far.
          Only " + n + " Jobs left to be a journeyman.");
      l.text("Traingslevel: Apprentice")
    } else if (i == 10) {
      g.text("Congratiulations. You have submitted " + i + " Jobs.
          You can call yourself a journeyman now.");
      l.text("Traingslevel: Journeyman")
      u.attr('style', 'background-image:
            url("assets/css/images/journeyman.svg")');
```

```
42      p.attr("src", "assets/css/images/journeyman.svg");
43    } else if (i > 10, i < 20) {
44      g.text("You have submitted " + i + " Jobs so far.
45          Go ahead journeyman. Become a master.");
46      l.text("Traingslevel: Journeyman")
47      u.attr('style', 'background-image:
48            url("assets/css/images/journeyman.svg")');
49      p.attr("src", "assets/css/images/journeyman.svg");
50    } else if (i == 20) {
51      g.text("Teach me master.
52          That is what people will say to you from now.");
53      l.text("Traingslevel: Master")
54      u.attr('style', 'background-image:
55            url("assets/css/images/master.svg")');
56      p.attr("src", "assets/css/images/master.svg");
57    } else if (i > 20, i < 30) {
58      var n = 30-i;
59      g.text("The Batch is yours, master.
60          You are on a good way to become a ninja.
61          For this you need to submit " + n + " more jobs.");
62      l.text("Traingslevel: Master")
63      u.attr('style', 'background-image:
64            url("assets/css/images/master.svg")');
65      p.attr("src", "assets/css/images/master.svg");
66    } else if (i == 30) {
67      g.text("Master Splinter would be proud of you.
68          You are a Batch-ninja now.");
69      l.text("Traingslevel: Ninja")
70      u.attr('style', 'background-image:
71            url("assets/css/images/ninja.svg")');
72      p.attr("src", "assets/css/images/ninja.svg");
73    } else if (i > 30, i < 50) {
74      g.text("Once a ninja - always a ninja. Keep submitting.
75          One day you will be a sensei.");
76      l.text("Traingslevel: Ninja")
77      u.attr('style', 'background-image:
78            url("assets/css/images/ninja.svg")');
79      p.attr("src", "assets/css/images/ninja.svg");
80    } else if (i >= 50) {
81      g.text("Konichiwa Sensei.");
82      l.text("Traingslevel: Sensei")
83      u.attr('style', 'background-image:
84              url("assets/css/images/sensei.svg")');
85      p.attr("src", "assets/css/images/sensei.svg");
86    }
87   }
88  }, 100); // check every 100ms
```

```
89 });
```

As this function reveals, it first waits, till the table from chapter 10.3 has loaded. For this an interval is set which is looking for the table every 100ms. When the table has loaded it changes the content in the training progress area in index.html based on how many jobs the user submitted so far.
The user here can pass six different levels - Rookie, Apprentice, Journeyman, Master, Ninja and Sensei.
A user that submitted 9 jobs for example is in the level Apprentice. If he submits one more job he will be a journeyman.

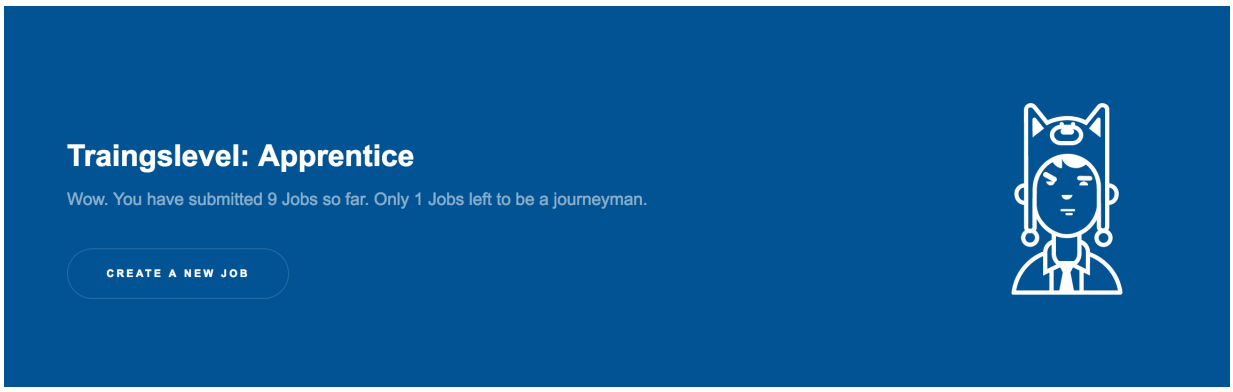

Figure 15: Training progress: Level Apprentice

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] V Mitra Gopaul. *IBM Mainframe Programmer's Desk Reference*. McGraw-Hill, Inc., 1993.

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

[8] 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 & Education*, 53(2):402–413, 2009.

[9] 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.

[10] John Kettner, Mike Ebbers, Wayne O'Brien, and Bill Ogden. Introduction to the new mainframe: z/os basics. *IBM Redbooks, NY*, 2011.

[11] 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.

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

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

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

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

[16] 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.

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

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

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

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

[21] 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