



# Summer Fellowship Report

On

**Python Web-Program for Koha Barcode and Spine Label**

Submitted by

**Rahul Paknikar**

Under the guidance of

**Dr. Manju Naika**

Chief Library Officer, Central Library

IIT Bombay

July 12, 2019

# Acknowledgment

It is really a pleasure to acknowledge the help and support that has gone to in making this successful project. I express sincere gratitude to Prof. Kannan Moudgalya and FOSSEE Team at IIT-Bombay for providing me with this opportunity to work on this project and also having faith in my abilities. I would like to thank my project mentors Dr. Manju Naika, Mr. Pravin Ghorpade at Central Library and project head Dr. Bella Tony, for being instrumental in helping me achieve the desired outcome and in understanding the objective of the work. I would like to mention special thanks to the open source community to provide me with the necessary tools and knowledge base required to complete this project. I shall mention special thanks to the staff of Central Library for making my working environment comfortable and providing me all the facilities required to carry out this project.

# Contents

|           |  |           |
|-----------|--|-----------|
| <b>1</b>  | <b>Introduction</b>                            | <b>4</b>  |
| <b>2</b>  | <b>Limitations with Existing System</b>        | <b>5</b>  |
| <b>3</b>  | <b>Methodology</b>                             | <b>6</b>  |
| <b>4</b>  | <b>Features</b>                                | <b>7</b>  |
| <b>5</b>  | <b>Requirements and Installation</b>           | <b>8</b>  |
| 5.1       | Server Setup . . . . .                         | 8         |
| 5.2       | Setup for Client-Processing . . . . .          | 9         |
| <b>6</b>  | <b>Running the Server</b>                      | <b>13</b> |
| 6.1       | Apache - The Deployment Server . . . . .       | 13        |
| 6.2       | Django WSGI - The Development Server . . . . . | 13        |
| <b>7</b>  | <b>User Access</b>                             | <b>14</b> |
| 7.1       | Accessing Home Page . . . . .                  | 14        |
| 7.2       | Staff Login . . . . .                          | 15        |
| 7.3       | Search . . . . .                               | 15        |
| 7.4       | Data Selection . . . . .                       | 16        |
| <b>8</b>  | <b>Admin Access</b>                            | <b>18</b> |
| 8.1       | Accessing Admin Page . . . . .                 | 18        |
| 8.2       | Admin Login . . . . .                          | 18        |
| 8.3       | Admin Actions . . . . .                        | 19        |
| 8.4       | Accessing Documentation . . . . .              | 19        |
| <b>9</b>  | <b>Printing the Labels</b>                     | <b>20</b> |
| 9.1       | Barcode Label Printing . . . . .               | 20        |
| 9.2       | Spine Label Printing . . . . .                 | 21        |
| 9.3       | Setting the Print Spooler . . . . .            | 21        |
| <b>10</b> | <b>Custom Fonts</b>                            | <b>23</b> |
| 10.1      | Serving Font-Files . . . . .                   | 23        |
| 10.2      | Encoding of Font-Files . . . . .               | 23        |
| 10.3      | Waiting for Server Response . . . . .          | 24        |

# List of Figures

|      |  |    |
|------|--|----|
| 5.1  | DataTable Plugin - Table Rendering . . . . .                   | 10 |
| 5.2  | JsBarcode Library - Encoded Barcode Image Generation . . . . . | 11 |
| 5.3  | pdfMake Library - PDF Generation . . . . .                     | 12 |
| 7.1  | Browser with given Server IP . . . . .                         | 14 |
| 7.2  | Home Page - Staff Login . . . . .                              | 15 |
| 7.3  | Barcode Search (Authenticated Users only) . . . . .            | 15 |
| 7.4  | Data Selection and Verification . . . . .                      | 16 |
| 8.1  | Admin Login . . . . .  | 18 |
| 8.2  | Admin Actions . . . . .  | 19 |
| 8.3  | Accessing Documentation . . . . .                              | 19 |
| 9.1  | Barcode Label PDF . . . . .                                    | 20 |
| 9.2  | Spine Label PDF . . . . .                                      | 21 |
| 9.3  | Print Spooler - Page Size Setup . . . . .                      | 22 |
| 9.4  | Print Spooler - Graphics Setup . . . . .                       | 22 |
| 10.1 | Serving of Font Files at Server-Side . . . . .                 | 23 |
| 10.2 | Ajax - Asynchronous Call to Server . . . . .                   | 24 |
| 10.3 | Await - Wait for Promise Object (Server Response) . . . . .    | 24 |

# Chapter 1

## Introduction

The existing system at the Central Library, IIT-Bombay relies on a Desktop-based software for printing of the Barcode Labels and Spine Labels for the Books, Records and Thesis. The printing software is made available along with a printer which is proprietary in nature and designed specifically for that printer. The data for the printing of Barcode and Spine Labels are provided through the Koha Database of the Central Library.

The process of setting up the print of these Labels is very cumbersome as it involves many tasks to be performed. The tasks include connecting to the database and making data available in an Excel sheet. This Excel sheet then needs to be linked with that proprietary software, manual adjustment of data layout on the label has to be done (if required), setting of font style and then printing those labels.

The primary objective of this project is to design a solution in such a way that it will be independent of Operating Systems and any printer interface. Other objectives involve making the solution portable and accessible from any location, not confined to a physical location, within the Intranet of the Central Library and to reduce the number of manual task to be performed with this process. Hence to provide a solution to above requirements, a Browser-Based approach is used i.e. a Website.

## Chapter 2

# Limitations with Existing System

- The existing software is confined to a physical location where the printer is placed and connected to a Desktop.
- The software needs to be installed separately on each computer and also the Excel sheets need to be linked on every computer where the printer will be associated for printing purposes.
- The softwares used are proprietary in nature and are not open-sourced.
- It is designed specifically to be compatible **ONLY** with their own printers and hence if printer is changed, then the software will fail.
- The current softwares used and the entire process of printing the labels are OS specific i.e. will work for Microsoft Windows Operating System.

# Chapter 3

## Methodology

The method to overcome above limitations is to make the entire solution Browser-Based which can be accessed via the Intranet of Central Library. It involves the deployment of Client-Server-Database Architecture where the Client will be the Browser. This solution involves following tasks to be performed for printing those labels:

- Search the Koha Database for the Barcode number. There are two options provided for the searching task:
  - A Single Barcode Search
  - A Range of Barcode Searches
- Select the data to be printed and verify it (if required).
- Print the data based on the requirement of user. There are two options provided for the requirement:
  - Barcode Label Printing
  - Spine Label Printing
- Verify the Settings of the Printer Interface. It involves two steps:
  - Setup of the page size (for Godex Printer : 91 mm x 18 mm).
  - Setup of the Graphics option (for Godex Printer : None).

**Note that this step is not within the scope of this project. However, it has been included to warn the user of the Printer Interface issues that can arise due to which the desired printing action won't be performed.**

# Chapter 4

## Features

- This solution is not confined to a physical location and is portable making it independent of printer connection.
- It does not require any installations to be made separately on every computer where the printer will be associated.
- This solution is Free/Libre and Open-Sourced, Flexible and Customisable.
- Its design is independent of printer interface and any other proprietary softwares i.e. is generic in nature and can work with any Barcode printers.
- This solution and its design of the task to print those labels are Operating System independent.
- Its architecture has been deployed entirely offline. Hence, it will function correctly even in the absence of Internet connectivity without any considerable bandwidth delay and consumption.



# Chapter 5

## Requirements and Installation

This system involves following technologies for its implementation:

- **Frontend**
  - HTML
  - CSS - Bootstrap Framework
  - JavaScript - JQuery Library
  - Ajax
- **Backend**
  - Python version 3
  - Django Web Framework
- **Servers**
  - Apache 2.0 Server with WSGI Interface
  - Django WSGI Server (Optional)

### 5.1 Server Setup

The Apache modules and the underlying Python tool will be installed globally i.e. using "root" permission; wherein the Django-WSGI Interface and entire Django App will be served by Apache in a Virtual Environment. Following are the setup on Linux (Ubuntu) based Operating System:

- **Recommended Python version  $\geq$  3.5. Install Python first.**
- **Python-MySQL Connectors:**

Open terminal and run the command as:

```
$ sudo apt-get install python3-dev  
$ sudo apt-get install libmysqlclient-dev
```

- **Apache-WSGI Server Setup:**

Open terminal and run the command as:

```
$ sudo apt-get update  
$ sudo apt-get install apache2  
$ sudo apt-get install libapache2-mod-wsgi-py3
```

- **PIP installation for Python:**

Open terminal and run the command as:

```
$ sudo apt-get install python3-pip
```

(Note that from here on pip points to the reference of Python3)

- **Installing Python Packages:**

Following packages are required which can be installed through the *pip*.

- **virtualenv**
- **django (Version  $\geq$  2.2)**
- **mysqlclient**
- **django-mysql**
- **django-decorator-include (Version  $\geq$  2.1)**
- **Sphinx (Version  $\geq$  2.1.1)**
- **recommonmark (Version  $\geq$  0.5.0)**
- **django-docs (Version  $\geq$  0.3.1)**

## 5.2 Setup for Client-Processing

Client-Side Processing involves Rendering of Koha data in a table, Barcode creation and PDF generation. Though these processes are done at Client-Side, the underlying plugins and library are served by the Server as static files to the Client. The description of each library is as follows:

- **DataTable Plugin (Version  $\geq$  1.10.18) :**

This plugin provides several APIs that facilitate the interaction of the Client with the Koha Data being searched. The implementation requires:

- CSS
  - \* jquery.dataTables.min.css
  - \* buttons.dataTables.min.css
  - \* select.dataTables.min.css

- JavaScript
  - \* jquery.dataTables.min.js
  - \* dataTable.select.min.js
  - \* dataTable.buttons.min.js
  - \* buttons.print.min.js

```

* Renders the Table using DataTable API
*
* @param {string} table_id - id of HTML Table Tag where the Table is to be rendered
* @param {array} tableData - An array of objects where each object is a row of the Table to be rendered
* @param {array} column_heads - An array of objects where each object is a column heading
* @param {string} storageKey - A storage key of the Local Storage
* @returns {DataTable object} table - DataTable Plugin| object representing the Table drawn
*/
function renderTable(table_id, tableData, column_heads, storageKey) {
  var table = $("#" + table_id).DataTable( {
    data: tableData,
    columns: column_heads,
    select: true,
    dom: 'Bfrtip',
    lengthMenu: [10,25,50,100],
    buttons: [
      'pageLength',
      'copy',
      'selectAll',
      'selectNone',
      {
        text: 'Delete Table',
        action: function (event, dt, node, config) {
          emptyTableData(storageKey);
        }
      },
      {
        text: 'Delete Row',
        action: function (event, dt, node, config) {
          table.rows('.selected').remove().draw();
        }
      },
      {
        text: 'Print Barcode',
        action: function (event, dt, node, config) {
          print("barcode_data");
        }
      },
      {
        text: 'Print Spine',
        action: function (event, dt, node, config) {
          print("spine_data");
        }
      }
    ],
  });
  return table;
}

```

Figure 5.1: DataTable Plugin - Table Rendering

Figure 5.1 shows the implementation of the DataTable Plugin by creating a JavaScript function named as *renderTable()*. It requires the HTML table ID

on which the CSS styling and data rendering will take place. The buttons include the custom options as per the requirements of the design.

- JsBarcode Library :

This library converts the text into a Barcode image. As per Barcode standards and the encoding used at the Central Library of IIT-Bombay, the implementation requires '*code39*' standard as follows:

- JavaScript
  - \* JsBarcode.code39.min.js

```
/**
 * Generates the Barcode for a given number.
 *
 * @param {integer} number - The number whose barcode is to be generated.
 * @returns {data URL} dataURL - a representation of the image in the format specified by the
 * 'image/png'.
 */
function genBarcode(number) {
  // Create a canvas for barcode
  var canvas = document.createElement('canvas');

  // Create a barcode on that canvas with the given number using JsBarcode API
  JsBarcode(canvas, number, {
    ...
    displayValue: false
  });

  // Convert Barcode to a string format readable by makePDF API
  dataURL = canvas.toDataURL('image/png');

  canvas = null;
  return dataURL;
}
```

Figure 5.2: JsBarcode Library - Encoded Barcode Image Generation

Figure 5.2 shows the implementation of JsBarcode Library by creating a JavaScript function named as '*genBarcode()*'. The output of JsBarcode is an image which requires HTML canvas for the drawing of that Barcode image. However, the image format is not acceptable by pdfMake Library and hence needs to be converted to a **Base64 encoding**.

- pdfMake Library

This library is at the core of this solution. It takes the data from the DataTable Plugin, the encoded image from the JsBarcode and generates the PDF. The implementation requires:

- JavaScript
  - \* pdfmake.min.js
  - \* vfs\_fonts.js

```
/**
 * Creates the printable data from the Table. If any row is selected, then only that row will be considered.
 * Otherwise, entire Table is considered for printing. With the printable data, it generates a PDF in another
 * tab where the file can be downloaded or printed directly.
 *
 * @param {string} type - Indicates the Table to be considered i.e. either Barcode Table or Spine Table
 */
async function print(type) {
    var selectedData;
    selectedData = table.rows('.selected').data();
    if (selectedData.length == 0) {
        selectedData = table.rows().data();
    }
}

// If number of objects in selectedData is odd, then duplicate last object and push it into the same array
// This is done to make the PDF compatible with the Printer and reduce wastage of paper.
if (selectedData.length % 2 != 0) {
    selectedData.push(selectedData[selectedData.length-1]);
}

window.pdfMake.vfs["Calibri-Regular.ttf"] = await getBase64("fonts/Calibri-Regular.ttf");
window.pdfMake.vfs["Calibri-Bold.ttf"] = await getBase64("fonts/Calibri-Bold.ttf");
window.pdfMake.vfs["Calibri-Italics.ttf"] = await getBase64("fonts/Calibri-Italics.ttf");
window.pdfMake.vfs["Calibri-BoldItalics.ttf"] = await getBase64("fonts/Calibri-BoldItalics.ttf");

pdfMake.fonts = {
    'Roboto': {
        'normal': 'Roboto-Regular.ttf',
        'bold': 'Roboto-Medium.ttf',
        'italics': 'Roboto-Italic.ttf',
        'bolditalics': 'Roboto-Italic.ttf'
    },
    'Calibri': {
        'normal': 'Calibri-Regular.ttf',
        'bold': 'Calibri-Bold.ttf',
        'italics': 'Calibri-Italics.ttf',
        'bolditalics': 'Calibri-BoldItalics.ttf'
    }
};

// create JSON array for pdfMake API
var docDefinition = {
    pageSize: { width: 3.58267 * 72, height: 0.708661 * 72 },
    pageMargins: [3,2,2,1],
    content: data,
    defaultStyle: {
        font: 'Calibri'
    }
};

// Create PDF and open it in Browser Tab
pdfMake.createPdf(docDefinition).open();
}
```

Figure 5.3: pdfMake Library - PDF Generation

Figure 5.3 shows the implementation of the pdfMake Library by creating a JavaScript function named as *'print()'*. pdfMake requires the Document Definition in which the page size, page margins, pdf contents and default styling are provided. This function is *async* in nature and uses *await* which will be discussed in Fonts Chapter.

# Chapter 6

## Running the Server

The advantage of deploying Django on Apache Server is that you get two Servers to work with. Thus if one of the server is not working, then other server can be used as temporary replacement. This solution involves two Servers to work with as mentioned below.

### 6.1 Apache - The Deployment Server

This is the main and primary server on which this solution is deployed. To start the server, open the terminal and run the command as:

```
$ sudo service apache2 start
```

To stop the server, open the terminal and run the command as:

```
$ sudo service apache2 stop
```

### 6.2 Django WSGI - The Development Server

This is a secondary and backup server on which this solution is developed. Note that the term *'backup'* indicates that this server is to be used in case when the Apache Server fails/crashes. It doesn't keep any kind of backup of the server files nor does it maintain the Server OS state. It is a light-weight server and thus cannot handle huge request and traffic. Hence it is to be used as Development Server and in extreme case, as Deployment Server. To start the server, go to the location where the root of the Django App is located. Now open the terminal and run the command as:

```
$ python manage.py runserver <server_ip>:<server_port>
```

Note that *'python'* points to the reference of *Python3*. To stop this server, press *Ctrl+C* in the current shell of the running terminal.

# Chapter 7

## User Access

Use your favourite Browser which supports built-in **Print Spooler** and **PDF Viewer**. Though the recommended browser is Mozilla FireFox. Kindly enable the Cookies and Pop-ups for working of this website in your Browser.

### 7.1 Accessing Home Page

To access the Home Page, just type the IP address of the Apache Server in the Browser tab as shown in Figure 7.1.

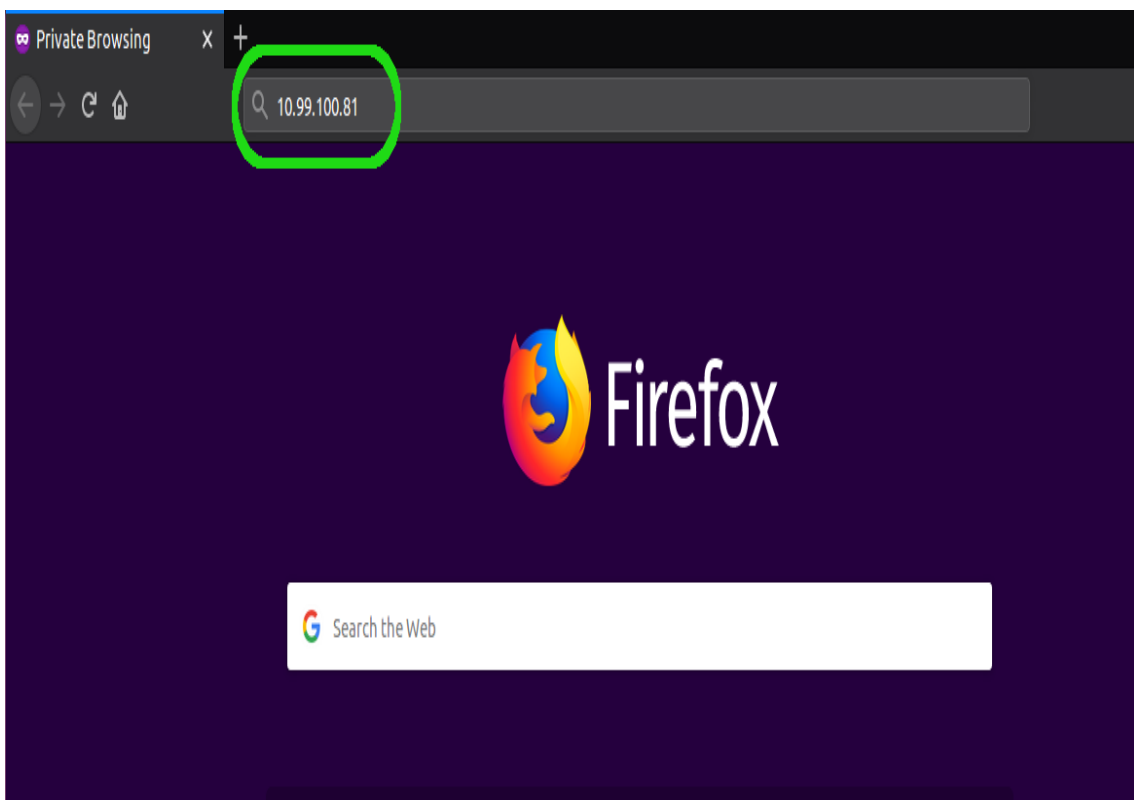


Figure 7.1: Browser with given Server IP

You should be redirected to an interface to Staff Login as shown in Figure 7.2.

## 7.2 Staff Login

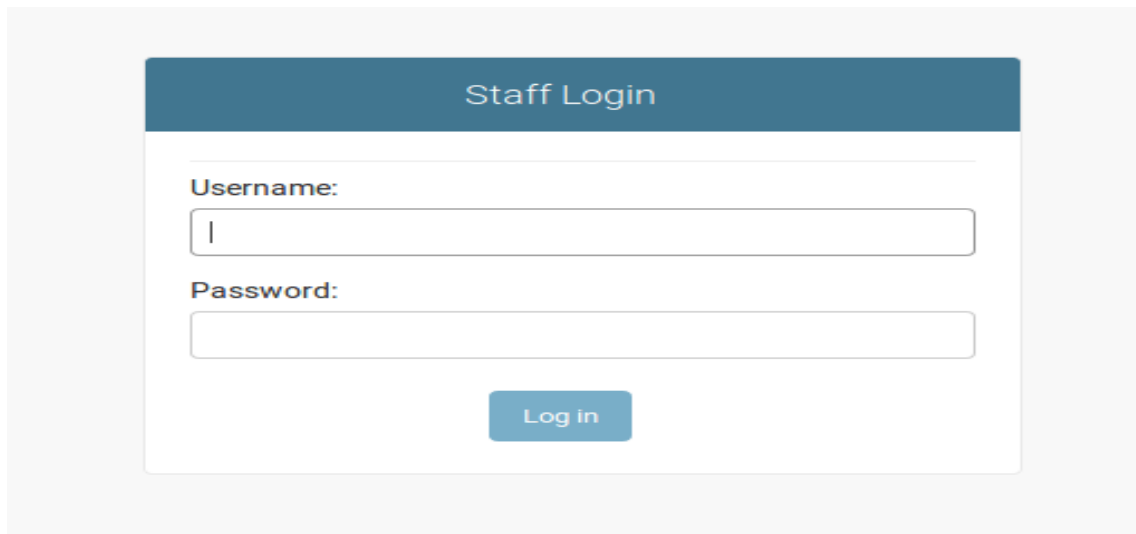


Figure 7.2: Home Page - Staff Login

Login with the credentials provided to you to start working this solution. You should get an interface where you can start searching in the Koha Database.

## 7.3 Search

As shown in Figure 7.3, user is provided with 2 choices:

- A Single Barcode Search
- A Range of Barcode Searches

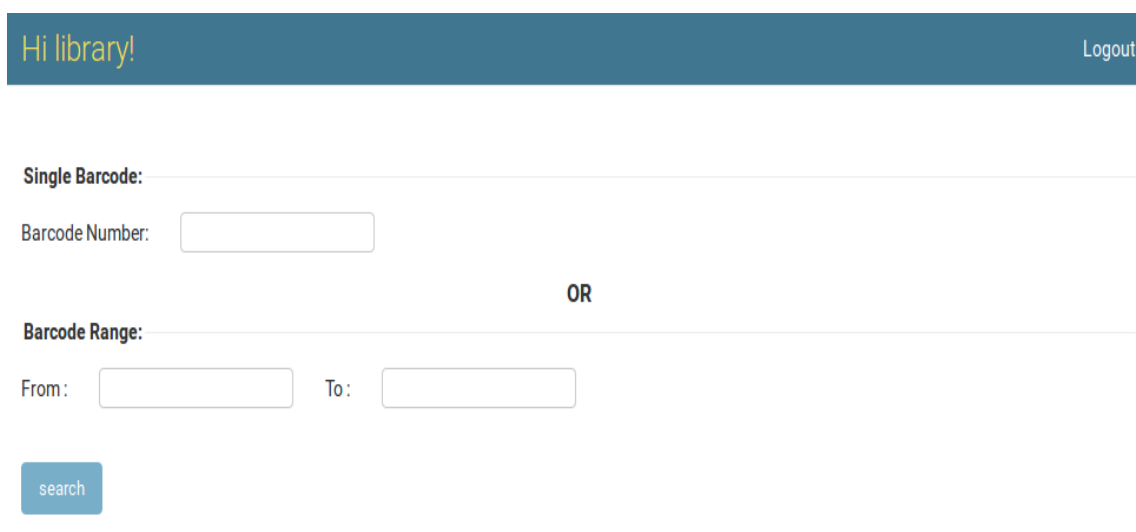


Figure 7.3: Barcode Search (Authenticated Users only)



All the fields of the Barcode inputs are validated with a Regex of the pattern  $[A-Z]^*[0-9]^+$  indicating that the Barcodes may start with Alphabets and must end with digits.

## 7.4 Data Selection

Single Barcode: \_\_\_\_\_

Barcode Number:

OR

Barcode Range: \_\_\_\_\_

From:  To:

---

Search:

| BARCODE ▲ | TITLE   | ▲ AUTHOR            | ▲ CALL NUMBER | ▲ AUTHOR MARK |
|-----------|---|---------------------|---------------|---------------|
| 79300     | Theoretical and experimental investigation of pressure compensated flow control valve (R) | Sinkar, S.R.        | 043:621-82    | Sin           |
| 79301     | Problems in economics   | Forbush, Dascomb R. | 33            | For           |
| 79303     | Urban economic problems   | Muth, Richard F.    | 330.191.3     | Mut           |
| 79306     | Committee on the working of the monetary system : report, Aug. 1959                       | United Kingdom      | 332.402       | Uni           |

Showing 1 to 4 of 4 entries Previous  Next

Figure 7.4: Data Selection and Verification

As shown in Figure 7.4, the result of Barcode search is displayed on a table using DataTable API. Here, the user has many options as listed:

- Pagination of Data - User can see more data on next pages as well as can set the maximum data on each page in gaps of *[10, 25, 50, 100]*.
- Select All
- Deselect All
- Copy Selected Rows (Press *Ctrl+C*)
- Delete Table - It deletes all the displayed contents of the table and **not of the Koha Database**.
- Delete Selected Rows
- Print Barcode Label
- Print Spine Label
- Search option in the table

# Chapter 8

## Admin Access

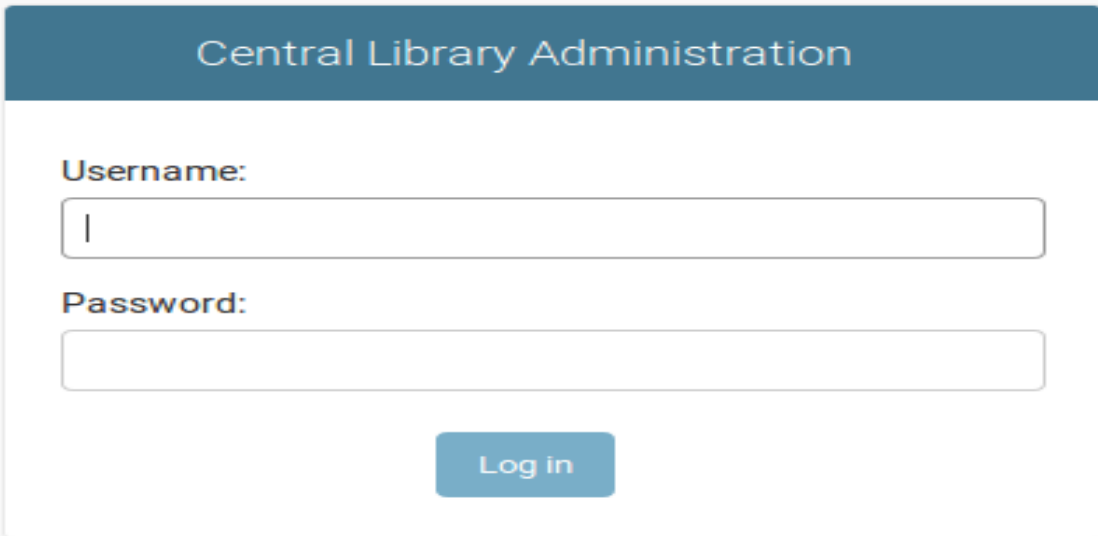
### 8.1 Accessing Admin Page

To access the Admin Page, type the IP address of the Apache Server followed by admin in the Browser tab. For example, if your server IP address is 10.99.100.81, then type in your Browser tab as:

*10.99.100.81/admin/*

You should be directed to an interface to Admin Login Page as shown in Figure 8.1.

### 8.2 Admin Login



Central Library Administration

Username:

Password:

Log in

Figure 8.1: Admin Login

Login with the Admin credentials to manage the website and access the documentation as shown in Figure 8.1. You should get an interface where admin actions can be performed.

## 8.3 Admin Actions

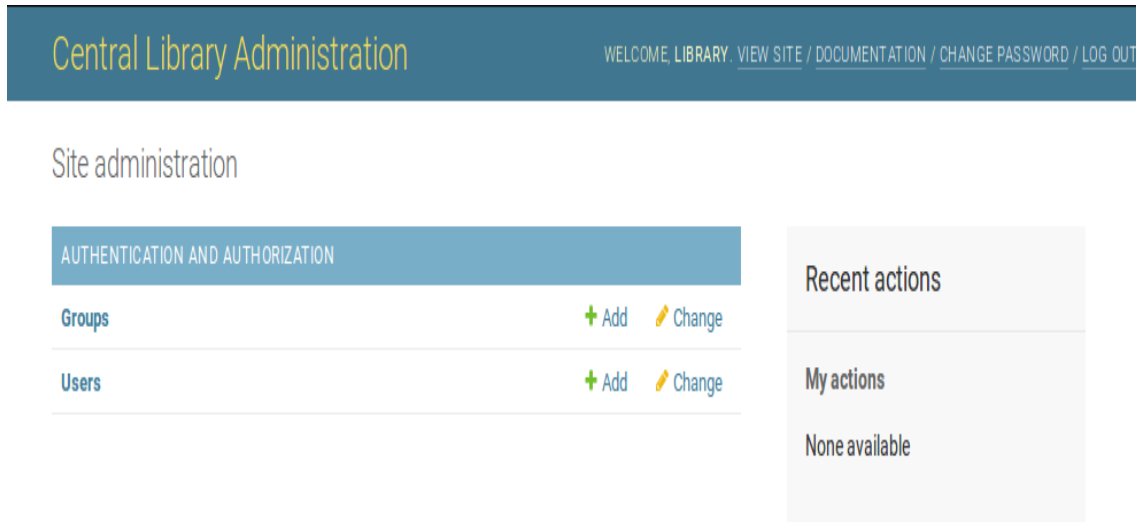


Figure 8.2: Admin Actions

As shown in Figure 8.2, a variety of Admin actions are provided, right from creating user accounts, making user groups, documentation, access to site, making rules for access to Koha Database and so on.

## 8.4 Accessing Documentation



Figure 8.3: Accessing Documentation

Click on the link of Documentation located at upper right corner in the Admin Action page. You should be directed to the detailed and official documentation of this solution as shown in Figure 8.3.

# Chapter 9

## Printing the Labels

To print the data shown in the table, there are two options provided in the interface as shown in Figure 7.4 :

- Print Barcode
- Print Spine

### 9.1 Barcode Label Printing

To print the Barcode labels, click on the '*Print Barcode*' button. A Pop-up will appear, thus opening a PDF in new tab of the Browser window containing the Barcode data adjusted automatically with the set font-styles.



Figure 9.1: Barcode Label PDF

Figure 9.1 shows this interface of generated PDF. Note that only the selected rows of the table will be chosen for printing. If no row is selected, then all the rows in the table will be chosen for printing by-default.

## 9.2 Spine Label Printing

To print the Spine labels, click on the *'Print Spine'* button. A Pop-up will appear, thus opening a PDF in new tab of the Browser window containing the Spine data adjusted automatically with the set font-styles.

|                                   |                                |
|-----------------------------------|--------------------------------|
| <b>043:621-82</b><br>Sin<br>79300 | <b>33</b><br>Eco<br>79302      |
| <b>330.191.3</b><br>Mut<br>79303  | <b>332.402</b><br>Use<br>79304 |
| <b>33:301</b><br>Fin<br>79305     | <b>332.402</b><br>Uni<br>79306 |
| <b>33(038)</b><br>Slo<br>79307    | <b>338.5</b><br>Lya<br>79308   |
| <b>658.15</b><br>Hag<br>79309     | <b>336.12</b><br>Bon<br>79310  |

Figure 9.2: Spine Label PDF

Figure 9.2 shows this interface of generated PDF. Note that only the selected rows of the table will be chosen for printing. If no row is selected, then all the rows in the table will be chosen for printing by-default.

## 9.3 Setting the Print Spooler

To print the above PDFs generated in above sections, click on the Print button in the right-hand corner of the PDF Viewer. A dialog box will appear where certain settings need to be before printing the PDFs. Select the printer to be used for printing the labels.

Figure 9.3 shows the interface when clicked on *'Properties...'* button in the spooler to change the page size which is acceptable by the printer. Change the page size as width : 91 mm and height : 18 mm for the Godex Printer and click on *'OK'* button.

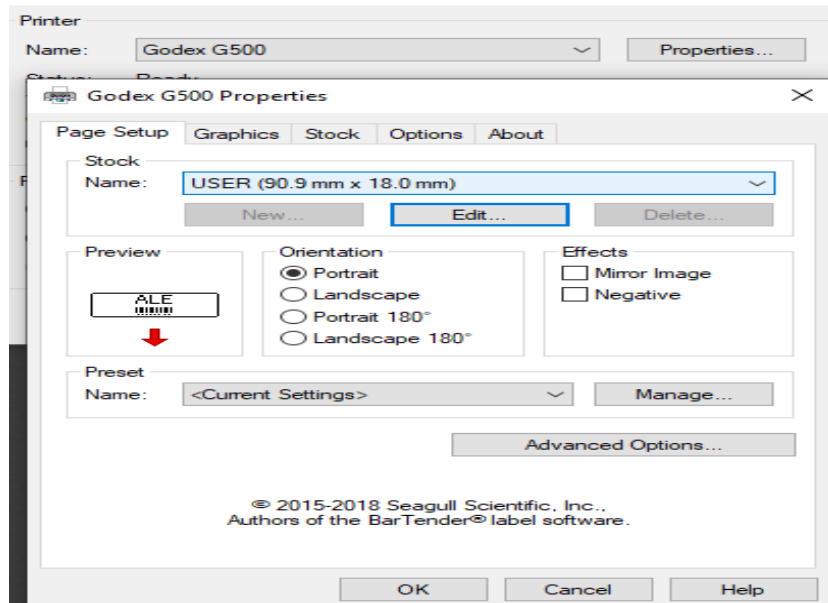


Figure 9.3: Print Spooler - Page Size Setup

Now go to the Graphics tab in Print Spooler as shown in Figure 9.4. Set the **Dithering** option to **None**, click on 'OK' button and then go ahead with the print of the PDF.

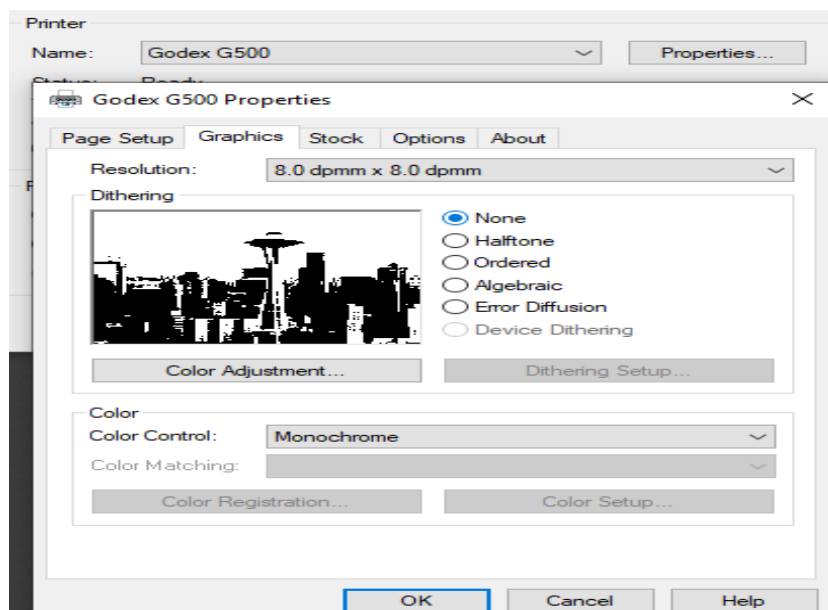


Figure 9.4: Print Spooler - Graphics Setup

Note that this section is not within the scope of this project and totally depends on the spooling interface. However, it has been included to warn the user of the Printer Interface issues that can arise due to which the desired printing action won't be performed.

# Chapter 10

## Custom Fonts

### 10.1 Serving Font-Files

To add a custom font and styling of your own choice, place the **"\*.ttf"** font files under the **"static/fonts"** folder of the Barcode App at the Server so that they can be served by Apache/WSGI Server as shown in Figure 10.1.

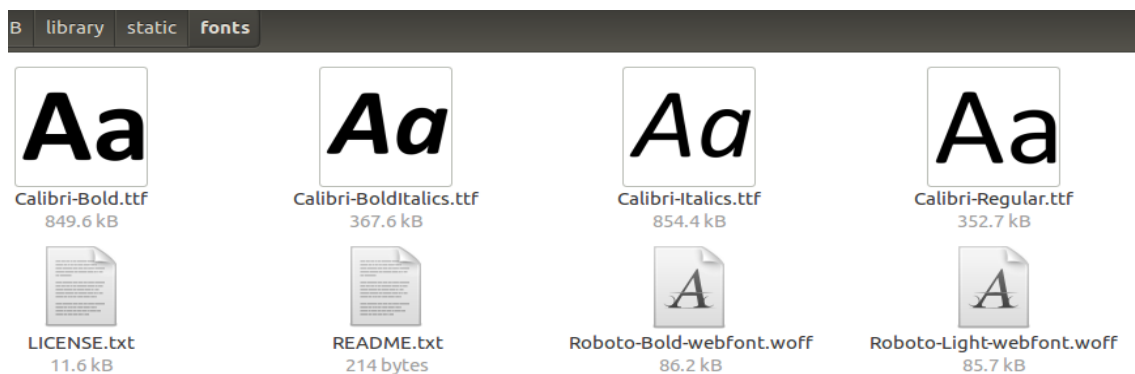


Figure 10.1: Serving of Font Files at Server-Side

### 10.2 Encoding of Font-Files

The pdfMake Library requires **Base64** encoding of the font-files as it does not work directly with the **"\*.ttf"** font-files. As seen in Figure 10.2, an **Ajax** Post call is made to the server with the url field defining the Django view to be used for encoding the font-file specified by the data field.

It can be seen that the success response from the server is being wrapped by the **Promise** object which includes the encoded string. This is because an Ajax call is **asynchronous** in nature and hence the response from the server can be received after entire execution of the logic flow. This may break the regular working flow of this solution. Hence, a Promise object allows to wrap the asynchronous response from the server in a synchronous manner and can be used by other parts of the logic flow through it.



```

/**
 * Encodes the font TTF file to Base64 Encoded String. Note the encoding is Async Task
 *
 * @param {string} title - The path to the TTF File
 * @returns {string} - Base64 Encoded String
 */
function getBase64(fileName) {
  return new Promise(function(resolve, reject) {
    $.ajax({
      type: "POST",
      url: "{% url 'encode' %}",
      data: {csrfmiddlewaretoken: '{{ csrf_token }}',
            text: fileName}, /* Passing the font name */
      success: function(response) {
        resolve(response);
      }
    });
  });
}

```

Figure 10.2: Ajax - Asynchronous Call to Server

### 10.3 Waiting for Server Response

As discussed in previous section, to make the regular working flow synchronous, a Promise object is created. However, this Promise object can be used by those functions which are *async* in nature. Figure 5.3 has already been discussed about the print function. As seen from that figure, it has been declared as *async function* of JavaScript.

```

window.pdfMake.vfs["Calibri-Regular.ttf"] = await getBase64("fonts/Calibri-Regular.ttf");
window.pdfMake.vfs["Calibri-Bold.ttf"] = await getBase64("fonts/Calibri-Bold.ttf");
window.pdfMake.vfs["Calibri-Italics.ttf"] = await getBase64("fonts/Calibri-Italics.ttf");
window.pdfMake.vfs["Calibri-BoldItalics.ttf"] = await getBase64("fonts/Calibri-BoldItalics.ttf"
);

pdfMake.fonts = {
  // Default font should still be available
  'Roboto': {
    'normal': 'Roboto-Regular.ttf',
    'bold': 'Roboto-Medium.ttf',
    'italics': 'Roboto-Italic.ttf',
    'bolditalics': 'Roboto-Italic.ttf'
  },
  // Make sure you define all 4 components - normal, bold, italics, bolditalics - (even if
  they all point to the same font file)
  'Calibri': {
    'normal': 'Calibri-Regular.ttf',
    'bold': 'Calibri-Bold.ttf',
    'italics': 'Calibri-Italics.ttf',
    'bolditalics': 'Calibri-BoldItalics.ttf'
  }
};

// create JSON array for pdfMake API
var docDefinition = {
  pageSize: { width: 3.58267 * 72, height: 0.708661 * 72 },
  pageMargins: [3,2,2,1],
  content: data,
  defaultStyle: {
    font: 'Calibri'
  }
};

```

Figure 10.3: Await - Wait for Promise Object (Server Response)

Figure 10.3 elaborates print function where this function waits for that Promise object by using *await*. and assigns them to the **pdfMake vfs** array. Again, a **pdfMake.fonts** need to be declared for all the fonts to be used in PDF, including the default **Roboto Font**. Finally, set the **defaultStyle** parameter in **docDefinition** with the choice of the font keyword.

# Reference

- **Python Installation**
  - <http://ubuntuhandbook.org/index.php/2017/07/install-python-3-6-1-in-ubuntu-16-04-lts/>
- **Django App on Apache Server**
  - [https://www.digitalocean.com/community/tutorials/how-to-serve-django-applications-with-apache-and-mod\\_wsgi-on-ubuntu-16-04](https://www.digitalocean.com/community/tutorials/how-to-serve-django-applications-with-apache-and-mod_wsgi-on-ubuntu-16-04)
- **Django-Apache Web-Server Setup**
  - <https://www.youtube.com/watch?v=VNBpdT0N8hw>
- **Read More on WSGI Server**
  - <https://wsgi.readthedocs.io/en/latest/>
  - [https://en.wikipedia.org/wiki/Web\\_Server\\_Gateway\\_Interface](https://en.wikipedia.org/wiki/Web_Server_Gateway_Interface)
- **Django Installation**
  - <https://docs.djangoproject.com/en/2.2/topics/install/>
- **Django Custom Decorators**
  - <https://pypi.org/project/django-decorator-include/>
- **Django-Sphinx Documentation**
  - <http://www.sphinx-doc.org/en/master/>
- **Recommonmark**
  - <https://github.com/rtfd/recommonmark>
- **Django-Docs**
  - <https://pypi.org/project/django-docs/>
- **DataTable Plugin**
  - <https://datatables.net/>
- **JsBarcode Library**
  - <https://lindell.me/JsBarcode/>
- **pdfMake Library**
  - <https://pdfmake.github.io/docs/>