Documentation "Payment System"

By Kian Gribi (TBZ, M122, LB02)

Der Code inklusive inline code Dokumentation kann auf GitHub/kiangribi gefunden werden.

Functions

The payment processing software is a simple script that parses a given Input from a server. The parsed XML and TXT files are then uploaded to the Payment server for further processment.

The second part of the script get the processed invoices from the Payment server. It downloads the receipt and creates a ZIP with the invoice. In a second step the ZIP is uploaded to the Customer server and sent by email to the client specified in the initial file.

It is meant to be highly configurable and very lightweight. That means it does not use a lot of non-builtin libraries.

Program must (Required features)

- Be able to download an invoice from the customer server
- Be able to parse an invoice from data to XML and TXT format
- Be able to upload an invoice to the payment server and caching it
- Be able to download a receipt from the payment server
- Be able to ZIP a receipt and an invoice
- Be able to upload the ZIP to the server
- Be able to send an email with the ZIP
- Be able to log everything and exit early properly
- Be able to use a config file for server config

Program can (Optional features)

- Be able to use a config file for formats and patterns

Planning

Milestone	Task	When
	Gather all resources and get specifications	Day 1
	Create a structure plan of the project	Day 1
	Plan the flow of the program	Day 1
*	Chose appropriate language and frameworks	Day 1
	Skeleton for first service	Day 2
	Create common components for both services	Day 2
	Write networking tools	Day 2
*	Skeleton for second service	Day 2
	Design and implement auto-parser	Day 3
	Finish first service	Day 4
*	Test first service and write test cases	Day 4
	Finish second service	Day 5
*	Test second service and write test cases	Day 5
	Write documentation	Day 6
	Execute all test cases and test the application extensively	Day 6
*	Deploy the service on backslash	Day 6

Testing

Test cases for "Service Parse"

ID	Description	Expected Result	Result
P1	Default program flow	Parsed invoice files on server (XML, TXT)	Pass
P2	Handle no invoice file	Service exits early and logs that there are no invoices	Pass
P3	Handle invalid columns in invoice	Service logs the error and skips this invoice	Pass
P4	handle invalid rows in invoice	Service logs the error and skips the invoice	Pass
P5	No Internet connection	Service logs "Server error" and exits early	Pass

Testcases for "Service ZIP"

ID	Description	Expected Result	Result
Z1	Default program flow	Email se## Product Documentationnt, correct ZIP uploaded	Pass
Z2	Handle no receipt file	### UsageService logs "no receipt found" and skips early	Pass
Z3	Handle no cached invoice	Description	CommandPass
Z4	Handle no cached invoice data	Start "Service Parse" as cronjob	In crontab: `15,30,45 * * * * python3 /path/to/service_parse.py &>> ./service_parse.log` Pass
Z6	No Internet connection	Start "Service ZIP" as cronjob	In crontab: `16,31,46 * * * * python3 /path/to/service_zip.py &>> ./service_zip.log` Pass
Z7	SMTP error	### Program flow Server logs email error and exits early	Pass

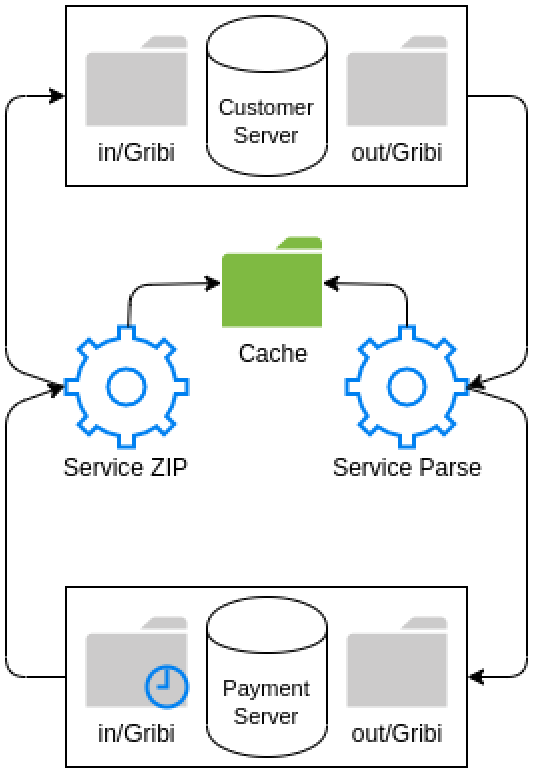
Product Documentation

Usage

Description	Command
Start "Service Parse" as cronjob	In crontab: `15,30,45 * * * * python3 /path/to/service_parse.py &>> ./service_parse.log`
Start "Service ZIP" as cronjob	In crontab: `16,31,46 * * * * python3 /path/to/service_zip.py &>> ./service_zip.log`

Program Sturcture

The Program consists of two small services. The first one to download, parse and upload the files and the second one to download, zip, upload and email the processed files. The following diagram shows the whole process in detail.



There are there parts of the software.

- Service Parser
- Service Zipper
- Common Modules

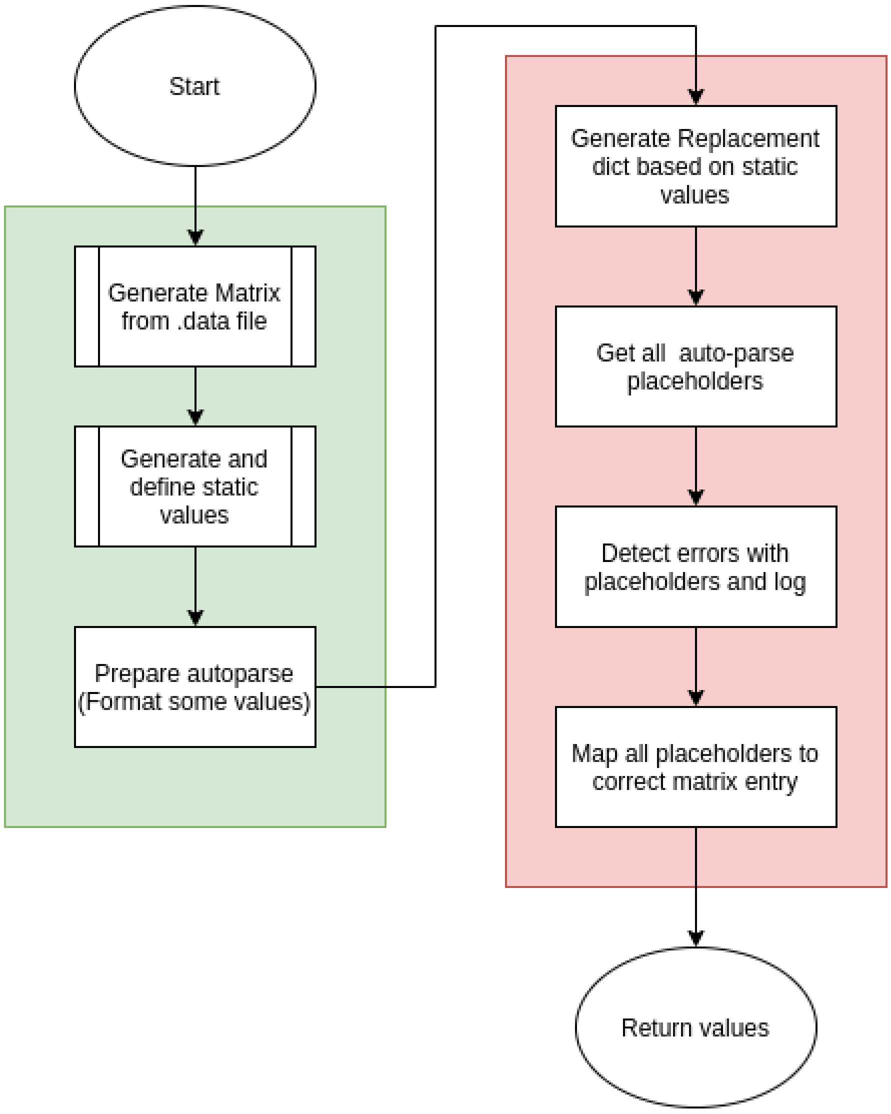
The Parsing and zipper services are visible in the above Illustration. The common modules are meant to provide a neutral interface to both services for common functionalities. The goal was to move most of the code into the common part since this code is generic and can be reused easily.

There are four parts of the common modules part.

- Network Module (Upload and Download Files from a Server)
- Config Module (Read configs form config files)
- Auto Parser Module (Auto Parser to parse files)
- Cache Module (Read and write files to/from cache)

The Auto-Parser

The auto-parser is a small library I wrote, that provides a robust interface to parse a variety of files. It is meant to be used to parse a file, regardless of it's format or type. The auto-parser works with template files, that contain specific placeholders. Here is a brief overview how the auto-parser works.


Start
Generate Matrix from .data file
Generate and define static values
Prepare autoparse (Format some values)
Generate Replacement dict based on static values
Get all auto-parse placeholders
Detect errors with placeholders and log
Map all placeholders to correct matrix entry
Return values

The auto-parser only processes placeholders that are in the format `$autoparse_XY` where X and Y are the coordinates for the data matrix. All other parameters have to specified in the "ignore dictionary" or "static value dictionary". If there are any other placeholders, that are not specified, the program will not process the file and throw an error.

Example

Data File (Data Matrix)

```
value00;value01;value02
value10;value11;value12
```

Before Auto-Parsing

```
<head>
    <title>$autoparse_01</title>
    <script src="$autoparse_12"></script>
</head>
<body>
    $static_content
</body>
```

After Auto-Parsing

```
<head>
    <title>value01</title>
    <script src="value12"></script>
</head>
<body>
    "Lorem Ipsum"
</body>
```

Logging

Each service has an own log file called like the service. Since only the console logs are colored, it is recommended to pipe `STDOUT` to the log file instead of using the built-in functions. This is set by default but can be changed in the main files of each service.

Configuration

The program was built to be highly configurable. I will explain the following configurations and their purpose in the next table.

Config JSON

```
{
    "patterns":{
        "receipt": "quittungsfile\\d{8}_\\d{6}\\.txt",
        "invoice": "rechnung\\d+\\.data"
    },
    "formats": {
        "date_email": "%d.%m.%Y",
        "time_email": "%H:%M:%S",
        "date_file": "%Y%m%d",
        "time_file": "%H%M%S",
        "date_invoice": "%d.%m.%Y"
    },
    "cache_folder": "./data/cache",
    "email_template": "./data/templates/email.txt",
    "email_sender": "payment@mail.ch",
    "email_sender_name": "Payment System",
    "template_invoice_xml": "./data/templates/invoice.xml",
    "template_invoice_txt": "./data/templates/invoice.txt",
    "template_invoice_positions_xml": "./data/templates/invoice_position.xml",
    "template_invoice_positions_txt": "./data/templates/invoice_position.txt"
}
```

Configuration Parameter	Description
`patterns/receipt`	Regex for the receipt file name
`patterns/invoice`	Regex for the invoice file name
`formats/date_email`	The date format that is used user facing (Email and XML)
`formats/time_email`	The time format that is used user facing (Email and XML)
`formats/date_file`	The date format that is used in the receipt file name
`formats/time_file`	The time format that is used in the receipt file name
`formats/date_invoice`	The date format that is required for the invoice
`cache_folder`	The cache folder
`email_template`	Email template location
`email_sender`	Email sender
`email_sender_name`	Email sender name
`template_invoice_xml`	Invoice XML template location
`template_invoice_txt`	Invoice TXT template location
`template_invoice_positions_xml`	Invoice positions XML template location
`template_invoice_positions_txt`	Invoice positions TXT template location

Server Customer, Payment and Email

```
{
    "hostname": "ftp.haraldmueller.ch",
    "username": "schoolerinvoices",
    "password": "Berufsschule8005!",
    "files_out": "out/AP17bGribi",
    "files_in": "in/AP17bGribi"
}
```

```
{
    "hostname": "134.119.225.245",
    "username": "310721-297-zahlsystem",
    "password": "Berufsschule8005!",
    "files_out": "out/AP17bGribi",
    "files_in": "in/AP17bGribi"
}
```

```
{
    "hostname": "smtp.mail.ch",
    "username": "payment@mail.ch",
    "password": "Berufsschule8005!"
}
```

Configuration Parameter	Description
`hostname`	Hostname of the server
`password`	Password to log in with
`username`	Username to log in with
`files_in` (Only Customer and Payment)	File in directory
`files_out`(Only Customer and Payment)	Files out directory

Reflection

This project was a bit special and I did not like to work on it as much as I like to work on other projects. I think the biggest problem were the missing specifications and instructions. The main task was provided in a handwritten file that was difficult to read. Fortunately a fellow student created a digital version but in my opinion this is not the idea behind a documentation.

The second thing were the specifications. The specifications were very loose and I had a lot of questions that did not get answered in the task description.