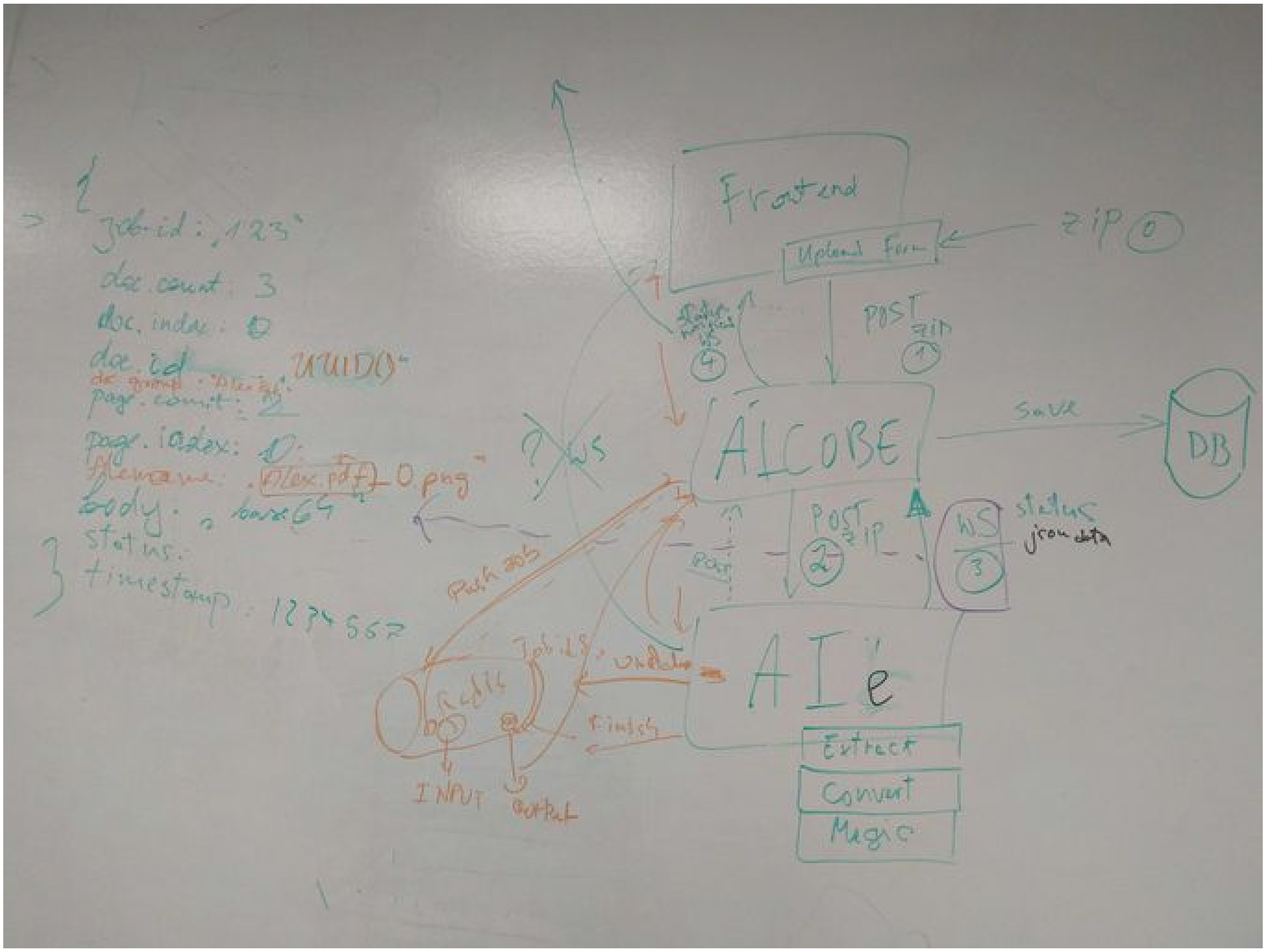
AIE(Upload and Image Extraction) Architecture

Architecture review date	**26 May 2020**
Refactor review date	05 Aug 2020
Project lead	@ Gregor Blichmann
On this page	

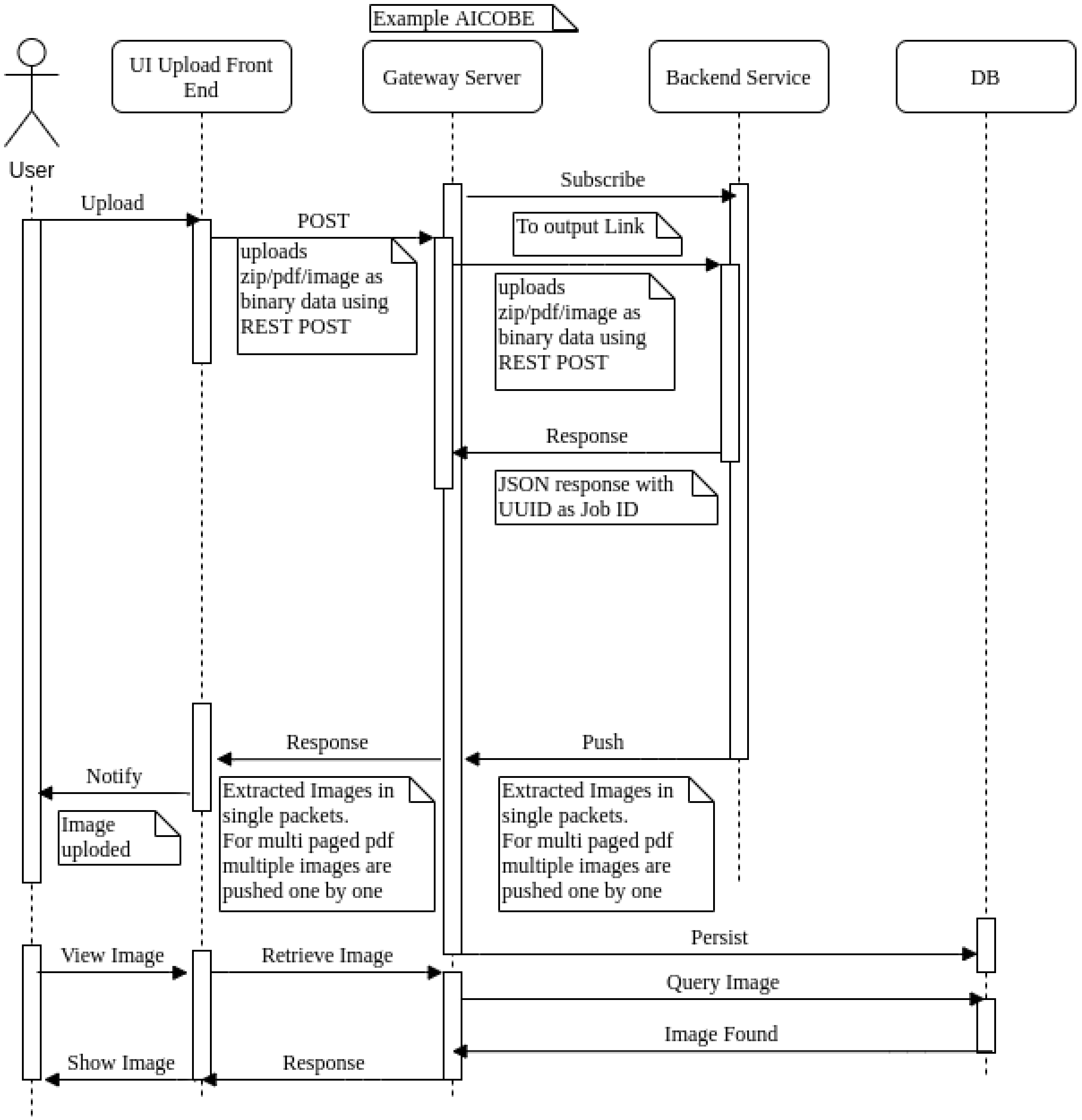
📋 Overview



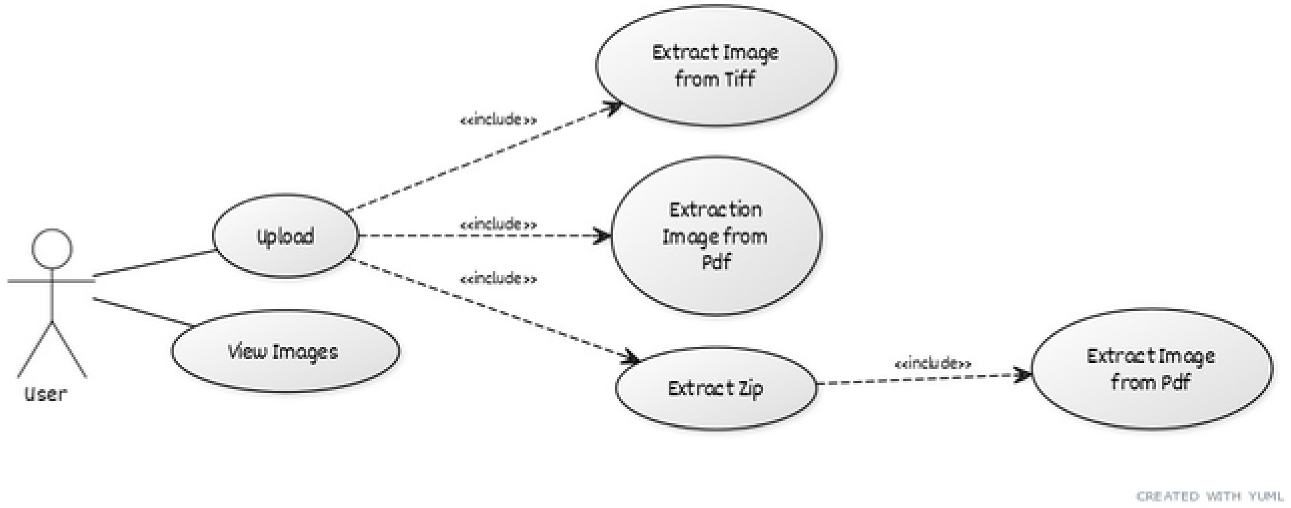
This is our discussion on 25 May 2020 on white board where we put all our different design approaches for the end to end workflow. Below every designs are drawn in detail with explanations.

📈 Workflow Sequence

Overall Idea on the Step by Step Sequence of Image Extraction Workflow

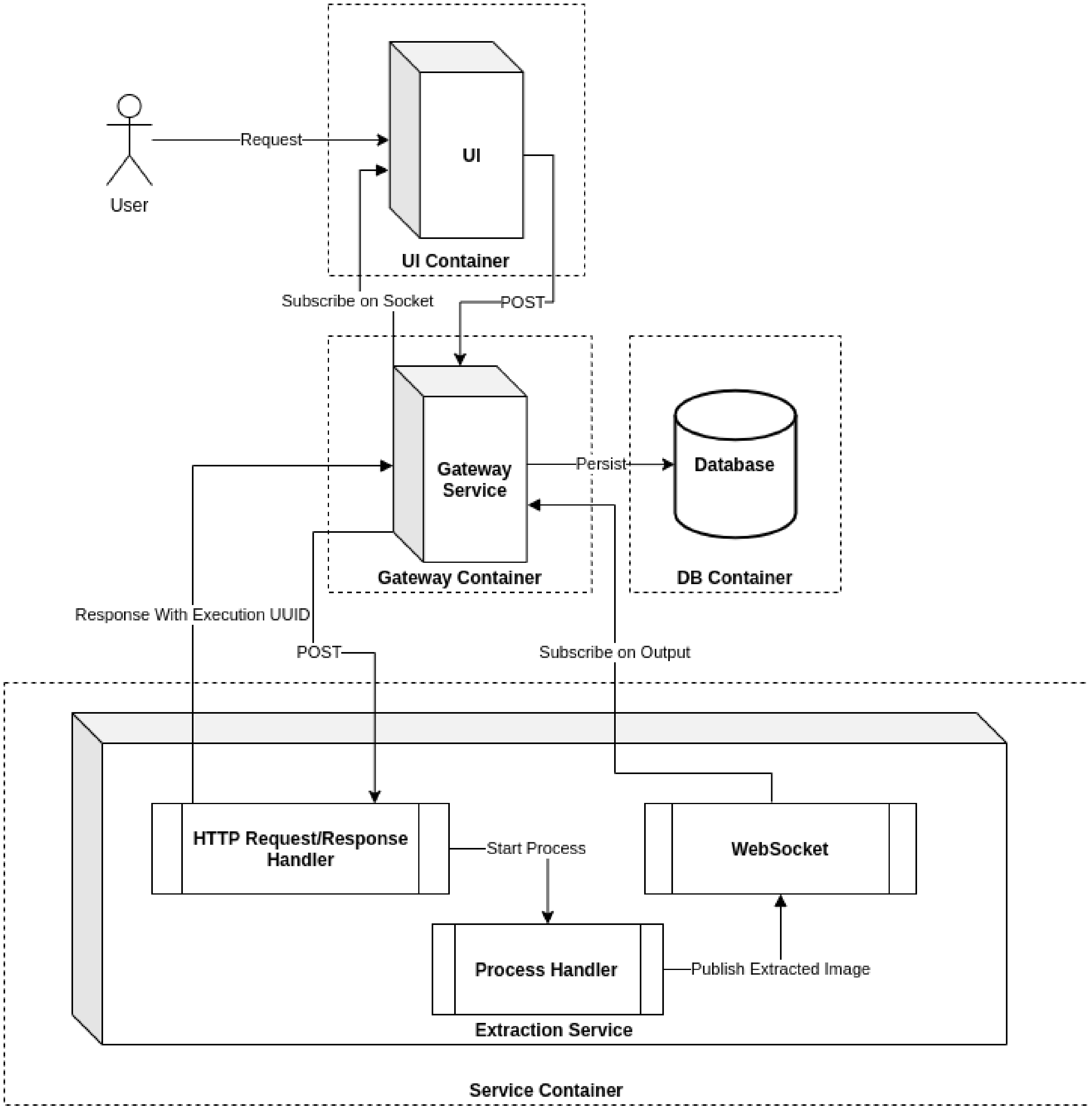

Example AICOBE
User
UI Upload Front End
Gateway Server
Backend Service
DB
Subscribe
Upload
POST
To output Link
uploads zip/pdf/image as binary data using REST POST
uploads zip/pdf/image as binary data using REST POST
Response
JSON response with UUID as Job ID
Response
Push
Notify
Image uploded
Extracted Images in single packets.
For multi paged pdf multiple images are pushed one by one
Extracted Images in single packets.
For multi paged pdf multiple images are pushed one by one
Persist
View Image
Retrieve Image
Query Image
Image Found
Show Image
Response

Use Case

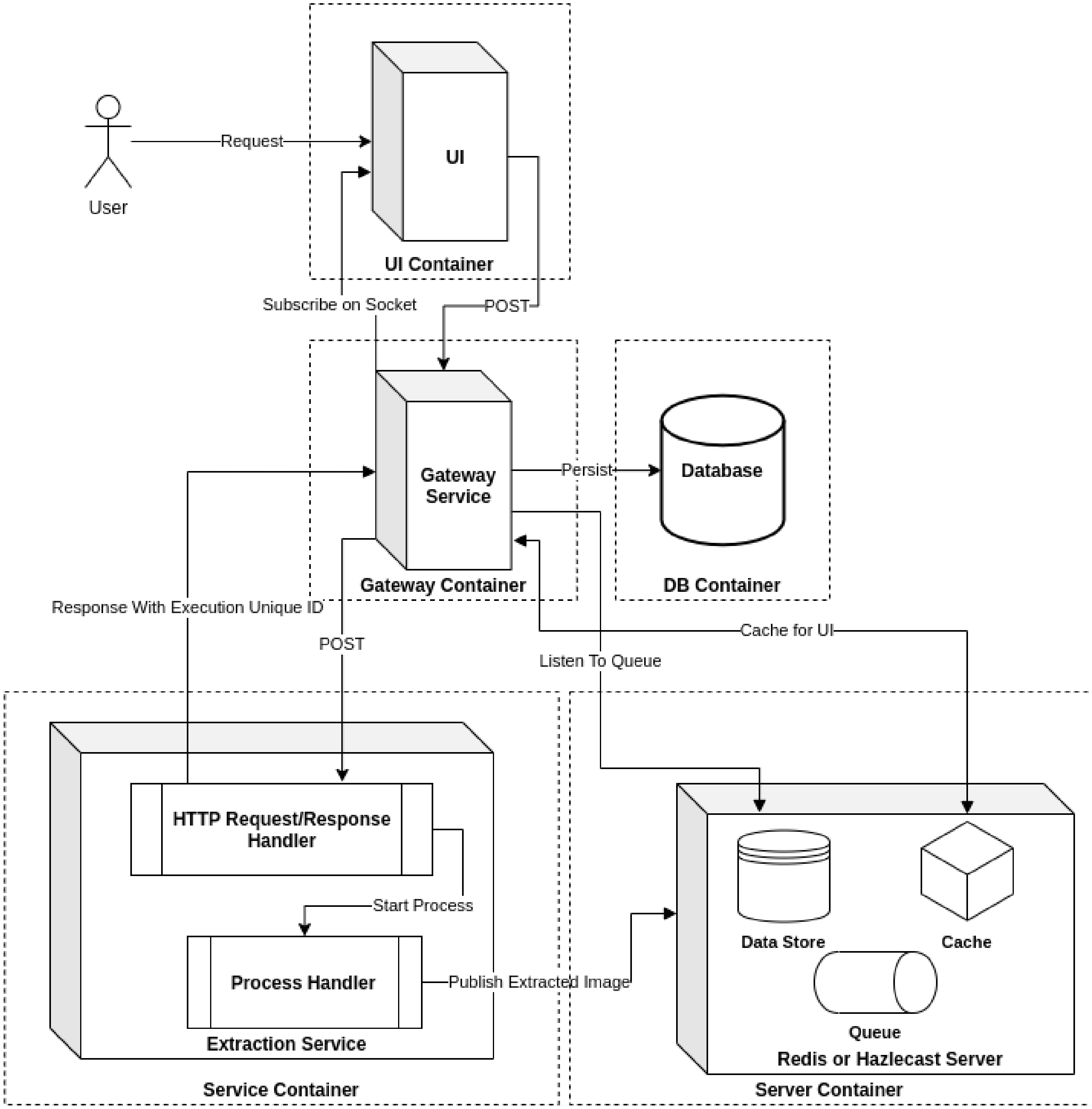


🗺️ Architecture Design Options

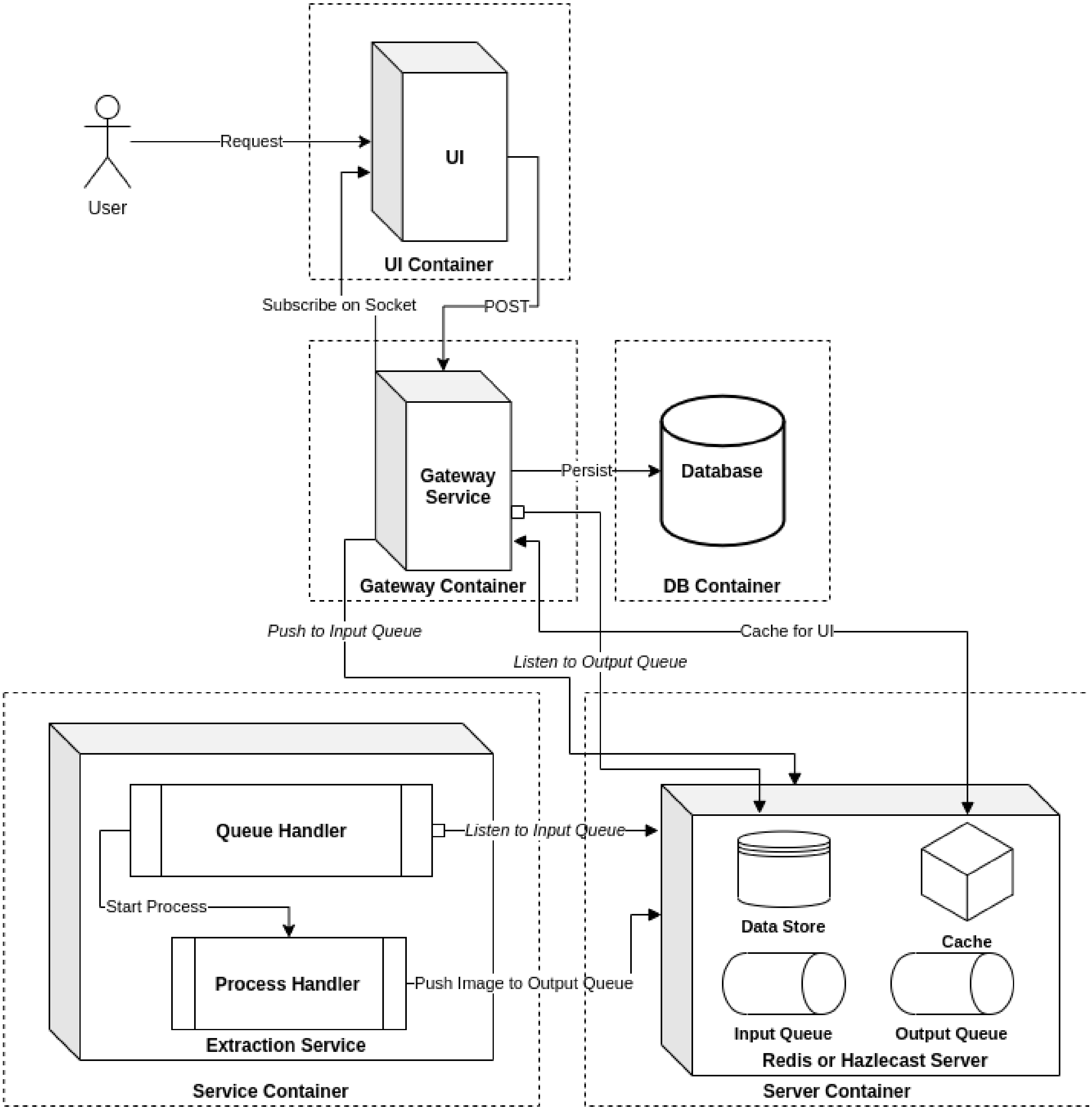
Option 1


User
Request
UI
UI Container
Subscribe on Socket
POST
Gateway Service
Persist
Database
Gateway Container
DB Container
Response With Execution UUID
POST
Subscribe on Output
HTTP Request/Response Handler
Start Process
WebSocket
Process Handler
Publish Extracted Image
Extraction Service
Service Container

Option 2


User
Request
UI
UI Container
Subscribe on Socket
POST
Gateway Service
Gateway Container
Persist
Database
DB Container
Response With Execution Unique ID
POST
Listen To Queue
Cache for UI
HTTP Request/Response Handler
Start Process
Process Handler
Extraction Service
Service Container
Publish Extracted Image
Data Store
Cache
Queue
Redis or Hazlecast Server
Server Container

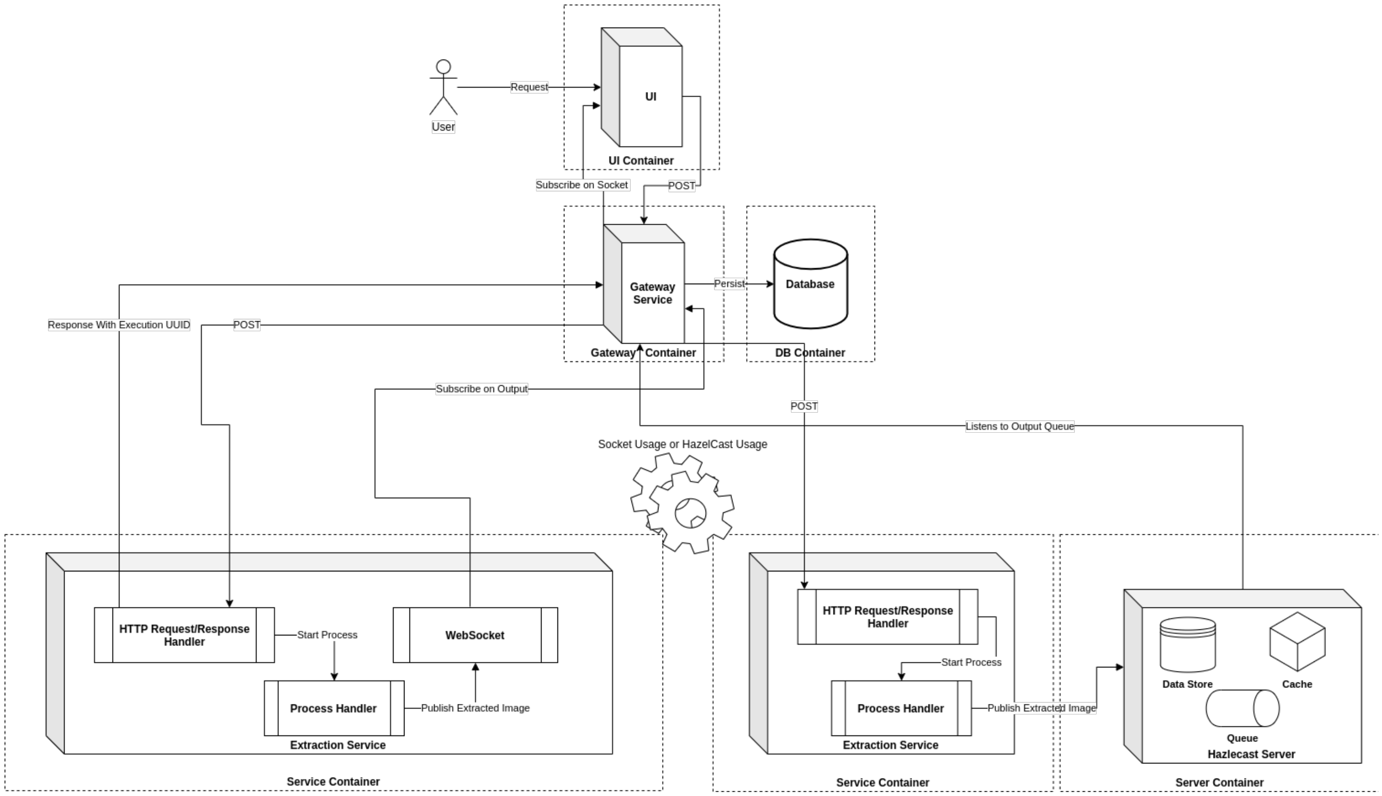
Option 3



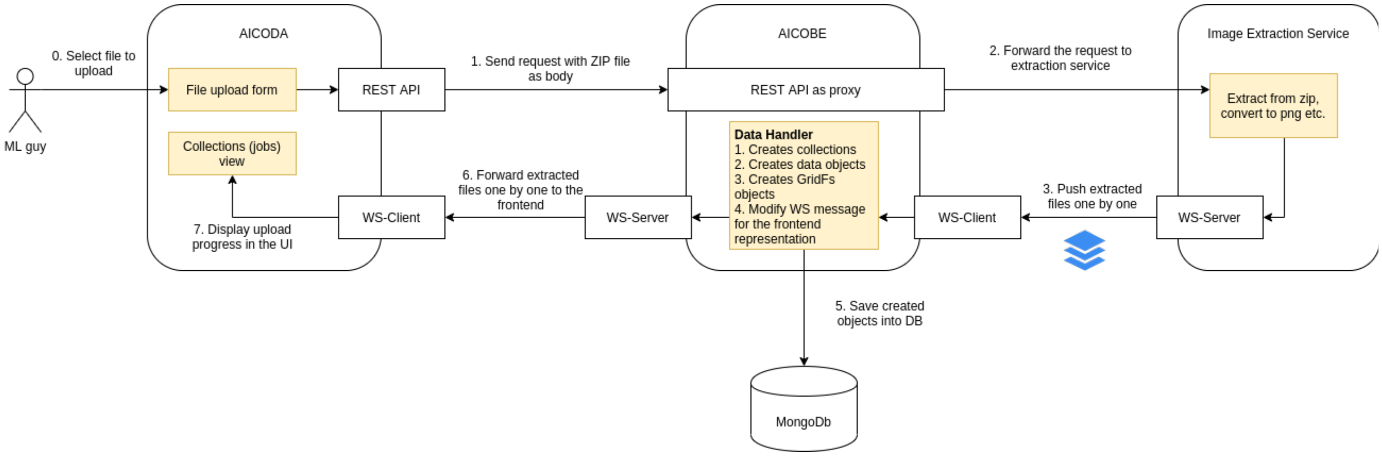
Architecture issues

Architecture Options	Implementation Complexity	Maintenance	External Dependency	Notes
Option 1	HIGH	MEDIUM	LOW	
Option 2	MEDIUM	HIGH	HIGH	
Option 3	MEDIUM	HIGH	HIGH	

Hybrid Architecture: (Option 1 + Option 2)



Data Flow


ML guy
0. Select file to upload
AICODA
File upload form
REST API
Collections (jobs) view
WS-Client
7. Display upload progress in the UI
1. Send request with ZIP file as body
6. Forward extracted files one by one to the frontend
AICOBE
REST API as proxy
WS-Server
Data Handler
1. Creates collections
2. Creates data objects
3. Creates GridFs objects
4. Modify WS message for the frontend representation
WS-Client
2. Forward the request to extraction service
3. Push extracted files one by one
Image Extraction Service
Extract from zip, convert to png etc.
WS-Server
5. Save created objects into DB
MongoDb

Data Model of Extraction Output

Success Message:

```
{
  "jobId": "0ef90871-c72b-411e-9e71-d8fc22b967ce",
  "documentName": "temp_0ef90871-c72b-411e-9e71-d8fc22b967ce.pdf",
  "pageFileName": "temp_0ef90871-c72b-411e-9e71-d8fc22b967ce_0.png",
  "pageCount": 32,
  "pageIndex": 0,
  "pageWidth": 800,
  "pageHeight: 200,
  "pageContent": "iVBORw0KGgoAAAANSUhEUgAABnUAAAkiCAIAAACTwI40...",
//Base64 png image string
  "documentCount":1,
  "documentIndex":0,
  "documentPath":"folderOne/nested",
  "meta": {}
}
```

Failure Message:

```
{
  "jobId": "0ef90871-c72b-411e-9e71-d8fc22b967ce",
  "documentName": "temp_0ef90871-c72b-411e-9e71-d8fc22b967ce.pdf",
  "pageFileName": "",
  "pageCount": 32,
  "pageIndex": 2,
  "pageContent": "", //Base64 png image string - no content since failed
  "documentCount":1,
  "documentIndex":0,
  "documentPath":"folderOne/nested",
  "failureReason": "NullPointerException"
}
```

⚒️ Comparison : Hazelcast vs Redis

Name	**Hazelcast**	**Redis**
Description	A widely adopted in-memory data grid	In-memory data structure store, used as database, cache and message broker
Primary database model	Key-value store	Key-value store
Secondary database models	Document store	Document store Graph DBMS Search engine Time Series DBMS

Website	hazelcast.com	redis.io
Technical documentation	hazelcast.org/-imdg/-docs	redis.io/-documentation
Developer	Hazelcast	Salvatore Sanfilippo
Initial release	2008	2009
Current release	4.0, February 2020	6.0.1, May 2020
License	Open Source	Open Source
Cloud-based only	no	no
DBaaS offerings (sponsored links)		
Implementation language	Java	C
Server operating systems	All OS with a Java VM	BSD Linux OS X Windows
Data scheme	schema-free	schema-free
Typing	yes	partial
XML support	yes	no
Secondary indexes	yes	yes
SQL	SQL-like query language	no
APIs and other access methods	JCache JPA Memcached protocol RESTful HTTP API	proprietary protocol

Supported programming languages	.Net C# C++ Clojure Go Java JavaScript (Node.js) Python Scala	C C# C++ Clojure Crystal D Dart Elixir Erlang Fancy Go Haskell Haxe Java JavaScript (Node.js) Lisp Lua MatLab Objective-C OCaml Pascal Perl PHP Prolog Pure Data Python R Rebol Ruby Rust Scala Scheme Smalltalk Swift Tcl Visual Basic
Server-side scripts	yes	Lua
Triggers	yes	no
Partitioning methods	Sharding	Sharding
Replication methods	yes	Master-slave replication Multi-master replication
MapReduce	yes	no
Consistency concepts	Immediate Consistency or Eventual Consistency selectable by user	Eventual Consistency Strong eventual consistency with CRDTs
Foreign keys	no	no
Transaction concepts	one or two-phase-commit; repeatable reads; read committed	Optimistic locking, atomic execution of commands blocks and scripts
Concurrency	yes	yes
Durability	yes	yes
In-memory capabilities	yes	yes
User concepts	Role-based access control	Simple password-based access control

Stakeholders

Name	**Role**

@ Sankha Sil @ Alexander Naumenko	Software Engineer
@ Sergej Tschigraj	UI Engineer
@ Gregor Blichmann	Product Owner

🧰 Refactoring Points

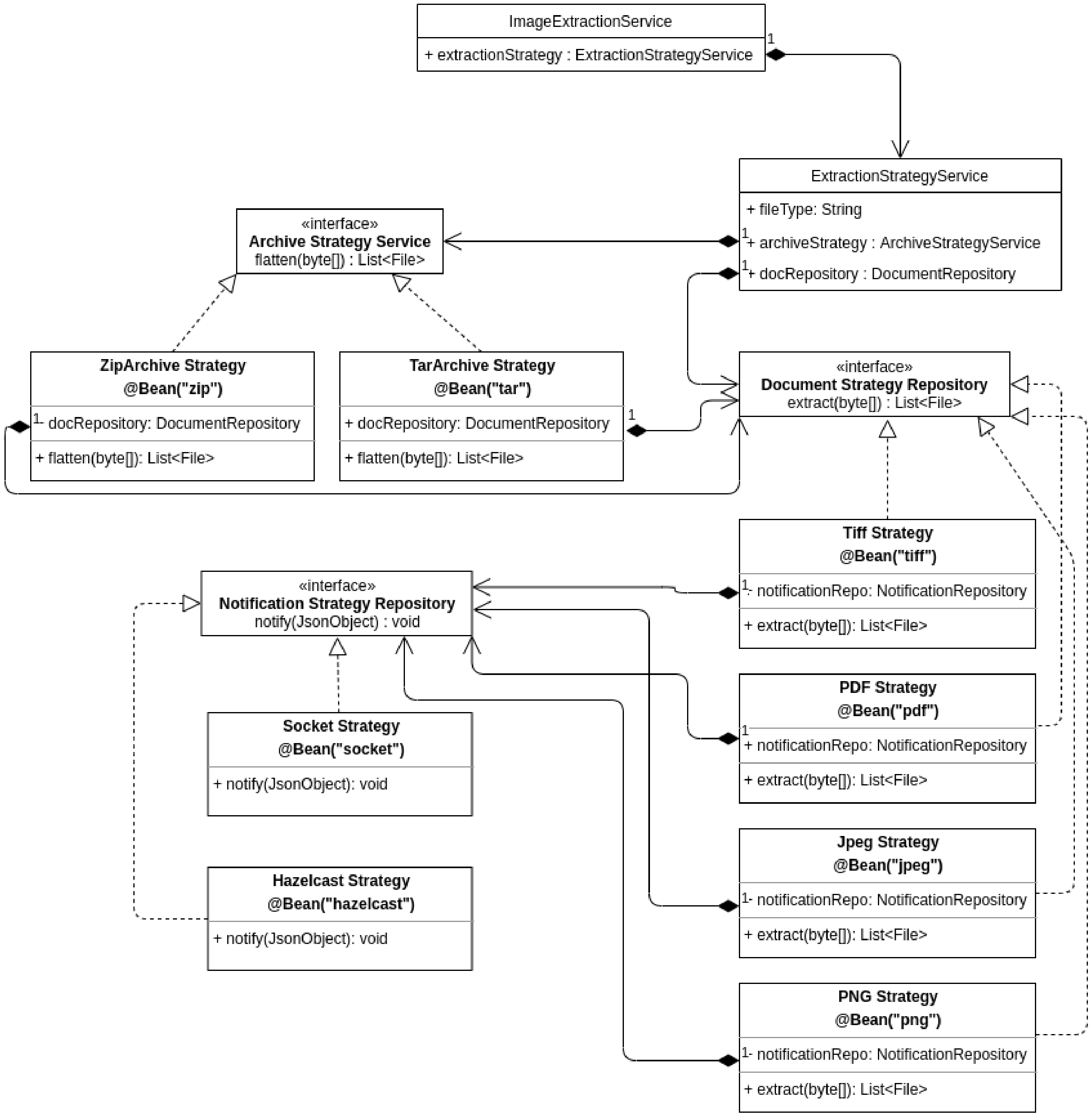
Currently Ixtraction service is running in https://ixtract.ai4bd.org to check one can use https://ixtract.ai4bd.org/image/extraction/ping link.

There lots of areas of improvement regarding code quality, code design and memory leaks. Some are listed below.

1. ~~Controller has overdesign of filetype check~~ 💡 ~~Remove~~
    a. ~~fileType request header. Check content type.~~
2. ~~Controller endpoints are not meaningful.~~ 💡 ~~Remove~~
    a. ~~Get user endpoint~~
    b. ~~detailed request header.~~
    c. ~~as soon as `/info` called remove the entry of hashmap.~~
    d. ~~Change the endpoint urls. Talk to WAPP team and update openAPI.~~
3. ~~Controller has service logic in endpoint of process.~~ 💡 ~~put logic to service~~
4. ~~Service File is huge with all the logic of extraction of image.~~ 💡 ~~Design pattern Strategy pattern,Factory pattern and Helper pattern.~~
    a. ~~First design class diagram to proceed.~~
5. ~~Switch case are used for different type of source type to determine the image extraction logic.~~ 💡 ~~Strategy pattern.~~
    a. ~~Recursive process for archive formats. (upto 3 recursion)~~
6. ~~Too many small methods used.~~ 💡 ~~Put methods into Util class .~~
7. ~~Methods having more than 7 parameters~~ 💡 ~~remove parameter array and put to class.~~
8. ~~Unnecessary Constants used.~~ 💡 ~~put to a common constant or Enum.~~
9. ~~Unit Test coverage is poor.~~ 💡 ~~TDD refactoring , e.g Util test before Util class.~~

1. ~~Methods for Socket/Hazelcast send/push~~ 💡 ~~Observer pattern with using strategy~~
2. Meaningful Logging required.

📕 Strategy Interaction Diagram


ImageExtractionService
+ extractionStrategy : ExtractionStrategyService
1
ExtractionStrategyService
+ fileType: String
1
+ archiveStrategy : ArchiveStrategyService
1
+ docRepository : DocumentRepository
«interface»
Archive Strategy Service
flatten(byte[]) : List<File>
ZipArchive Strategy
@Bean("zip")
1- docRepository: DocumentRepository
+ flatten(byte[]): List<File>
TarArchive Strategy
@Bean("tar")
+ docRepository: DocumentRepository
+ flatten(byte[]): List<File>
1
«interface»
Document Strategy Repository
extract(byte[]) : List<File>
Tiff Strategy
@Bean("tiff")
1- notificationRepo: NotificationRepository
+ extract(byte[]): List<File>
«interface»
Notification Strategy Repository
notify(JsonObject) : void
PDF Strategy
@Bean("pdf")
1
+ notificationRepo: NotificationRepository
+ extract(byte[]): List<File>
Socket Strategy
@Bean("socket")
+ notify(JsonObject): void
Jpeg Strategy
@Bean("jpeg")
1- notificationRepo: NotificationRepository
+ extract(byte[]): List<File>
Hazelcast Strategy
@Bean("hazelcast")
+ notify(JsonObject): void
PNG Strategy
@Bean("png")
1- notificationRepo: NotificationRepository
+ extract(byte[]): List<File>