Internal API resources for financial services

Joan Curto Alonso
Enginyeria del programari
Web development

Nom Consultor/a
Nom Professor/a responsable de l’assignatura: Gregorio Robles Martínez

19/06/2020

FITXA DEL TREBALL FINAL

Títol del treball:	*Internal API resources for financial services*
Nom de l'autor:	*Joan Curto Alonso*
Nom del consultor/a:	*Nom i dos cognoms*
Nom del PRA:	*Gregorio Robles Martínez*
Data de lliurament (mm/aaaa):	*06/2020*
Titulació o programa:	*Enginyeria del programari*
Àrea del Treball Final:	*Web development*
Idioma del treball:	*Anglès*
Paraules clau	*API, resources*

Resum del Treball (màxim 250 paraules): *Amb la finalitat, context d'aplicació, metodologia, resultats i conclusions del treball*

Els consultors deTradeHeader treballen constantment fitxers que pertanyen a diferents estàndards i diferents versions dins de cada estàndard. Aquests estàndards poden tenir customitzacions pròpies de cada client. Aquests recuros necessiten estar centralitzats i accessibles per tots els membres de la organització. Al mateix temps, hem de proveir amb una forma automatitzada per a integrar aquests recursos dins de les nostres solucions de software.

L'objectiu d'aquest sistema es respondre a aquestes necessitats. Per tal de fer-ho he dissenyat i desenvolupat un repositori amb una interfície API que permet als usuaris comunicar-se amb aquest repositori. Aquesta solució API també permet integrar-se amb altres serveis cloud.

Abstract (in English, 250 words or less):

TradeHeader consultant’s constantly work with files that belong to different standards, different versions within every standard and with multiple client focus customizations that need to be centralized and accessible by every member of the organization. At the same time, we need to provide an automated way to integrate these resources into our software solutions.

This system’s target is to solve this issues. To do so I have designed and developed a repository with an API interface that allows the users to comunicate with the repository. This API solution can also be integrated with other cloud services.

Índex

1. Introduction

1.1 Context

The financial data standardization is a complex market with different actors with different roles and necessities:

- Financial entities with data quality problems that need standardization on their internal trade operations and databases while complaining the applicable regulations.
- Financial standard Managers that generate, update and manage financial data standards (they can be financial organizations associations (ISDA, FIX Trading, or ISO/TC 68) [4][5][2][or private organizations (SWIFT)) [3]
- Regulators: Commodity futures trading commission (CFTC) [31] for the USA or European Securities and Markets Authority (ESMA) [32] for Europe.
- Brokers/Execution Platforms/Clearing Houses that intermediate between buyers and sellers of financial instruments. They act as the middleman on behalf of both parties in a financial transaction.

The standards published by the financial standard managers are in constant evolution due to the everchanging nature of the financial environment: new products that the standards must cover, regulatory changes to comply, or natural evolutions and improvements.
There are two main differences between FpML [1] (ISDA's standard), FIX [33](FIX Trading), ISO 20022 (ISO/TC 68's standard), and MT/MX (managed by SWIFT itself):

- The first one is that while, as mentioned before, SWIFT is a private organization, ISDA, FIX Trading, and ISO/TC 68 are non-for-profit organizations.
- The second main difference is that while ISDA, FIX Trading, and ISO only act as specification managers so that FpML and ISO 20022 are specifications that need from third parties to implement them (TradeHeader is one of them), SWIFT acts as a specification manager and as an implementor of MT messages.

Whenever FpML, FIX, and ISO 20022's new updates are published the financial institutions in coordination with their implementors get to decide whether to adopt them, moreover, whenever a financial institution decides to adopt one of this standards get to choose which version to implement and can customize its implementation modifying and extending its initial product coverage.
MT's specification new updates are implemented by the SWIFT organization itself and forced to adapt to by the financial entities that use the standard.
TradeHeader offers consulting services and develops custom software solutions to financial entities about all these three standards.

TradeHeader consultant’s constantly work with different standards, different versions within every standard and with multiple client focus customizations that need to be centralized and accessible by every member of the organization. At the same time, we need to provide an automated way to integrate these resources into our software solutions.

1.2 Objectives

Develop a resource repository accessible by an API [34] to unify TradeHeader’s internal resources.
This way, TradeHeader’s consultants accessibility to the internal resources will improve and also will broaden its resources availability.
As subobjectives we can define the repository and its configuration, the integration of the API with the repository including the upload requests with the repository resources allocation and the download requests to the repository resources.

1.3 Approach and chosen method

Given that TradeHeader does not have any existing product that could be adapted there were only two options left: Using a third-party solution or developing one ourselves.

As of for the third-party solution we are already using a few of them as our existing solution: Part of the resources are being hosted using the google drive tool and another part is hosted in git repositories.
We even have tried to solve the lack of centralization on our resources using a specific git repository to host a group of resources. The solution only solved our problems partially as we were able to access these resources and store them in a centralized way, but we are not able to provide access for our web services to this resources so we needed to maintain multiple copies of them in the back end of each web service.

This experience with existent third-party solutions and the fact that they were unable to solve our problems, we decided to develop our own solution. This way we expect to be able to centralize our resources successfully as well as serving them to our consultants and web services.

1.4 Relationship with other projects

TradeHeader has several web services in production stage that use XML-based [35] files a back-end [36] resource. Once this service is in production stage, this back-end files will be replaced by calls to the resources service.

This project will also impact most of the rest of TradeHeader's projects as they usually involve creating, analyzing, or modifying financial resources. This project will impact how these resources are obtained and stored by our consultants.

1.4 Planification

The required resources for this project are 300 hours of a junior software developer with the support of a senior developer.

This project can be separated into four different processes: Work plan, definition of the project requirements, design of the project and development of the project.

The work plan will take 12 days to develop and will define a set of objectives. These objectives will need to be achieved by the development team in the proposed time frame.

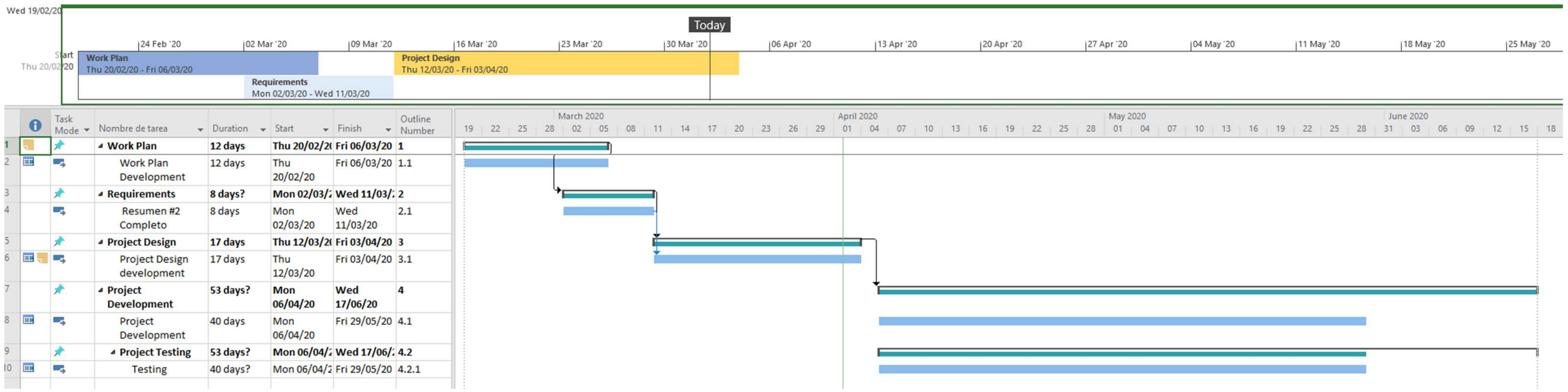
As each of the objectives has its own time limitations it may also help the team to check its progress through the project.

The project requirements part is proposed to last 8 days. During this phase there will be a negotiation between the client and the development team to agree on the requirements that the system needs to meet.

The project design process will take 17 days. During the previous phases we will have had defined what will de system do. This phase focus is to describe how will de system do what it must do. This design includes the technologies used, the project design and the API endpoints that the application will offer.

The last and the most extensive phase of the project is the development. This one has a projected length of 53 days. De development will use all the work done in the previous phases to make a viable and working project. It will include the development of a front end, an API Gateway, lambda functions and the configuration of two repositories,

Gannt diagram



		Task Mode	Nombre de tarea	Duration	Start	Finish	Outline Number
1			**Work Plan**	**12 days**	**Thu 20/02/2**	**Fri 06/03/20**	**1**
2			Work Plan Development	12 days	Thu 20/02/20	Fri 06/03/20	1.1
3			**Requirements**	**8 days?**	**Mon 02/03/2**	**Wed 11/03/2**	**2**
4			Resumen #2 Completo	8 days	Mon 02/03/20	Wed 11/03/20	2.1
5			**Project Design**	**17 days**	**Thu 12/03/2**	**Fri 03/04/20**	**3**
6			Project Design development	17 days	Thu 12/03/20	Fri 03/04/20	3.1
7			**Project Development**	**53 days?**	**Mon 06/04/20**	**Wed 17/06/20**	**4**
8			Project Development	40 days	Mon 06/04/20	Fri 29/05/20	4.1
9			**Project Testing**	**53 days?**	**Mon 06/04/2**	**Wed 17/06/2**	**4.2**
10			Testing	40 days?	Mon 06/04/2	Fri 29/05/20	4.2.1

1.5 Project deliverables

- An API service that complies with the functional requirements
- An API specific documentation that specifies for every available endpoint:
    - Description of the endpoint functionality
    - URL
    - HTTP method information
    - Headers
    - Parameters
    - Sample Request
    - Sample Response
- Project documentation

1.6 Cost Estimation

This cost estimation was calculated using the pricing listed in the AWS web [6] (02/04/2020) and the file size estimation from the Appendix 2 (File size estimation).

For the repository cost estimation, we need to consider: The storage, the data transfer, and the API Gateway [8].

Technology

There are three main storage types: Block storage, object storage and file storage. [37]

- Block storage: This type of storage is very good for virtual machines and databases and other workloads that require low latency and high speeds on input/output operations. They usually have very high cost per gigabyte and limited scalability.
- File storage: This type of storage usually has higher latency than the block storage ones but can reach high throughput. This makes them good for workloads that do not need a very low latency.
  They usually are cheaper than block storage systems.
- Object Storage: This type of storage is designed to offer the best accessibility and reliability. They can be accessed from anyone anywhere using protocols like http. They usually have high parallelization allowing multiple users access the same information at the same time.
  Contrary to what happens in a file system hierarchy, objects are stored in a flat namespace and can be retrieved by searching metadata or knowing the key. This is also why object storage is considered a very good option for storing large sets of unstructured data.
  These systems also hold a higher scalability potential than block and file storage systems.
  They are the cheapest cost per gigabyte of the three of them.

Because of the multiple advantages that object storage solutions have over the other two, object storage has been the chosen technology.
In the AWS ecosystem, there is a service for every technology listed before, as we have chosen the object storage based one:

- Elastic Block Storage (EBS) for block storage
- Elastic File System (EFS) for file storage
- Simple Storage Service (S3) for object storage

S3 storage tiers

There are several different storage tiers in the AWS S3 [38] service: standard, standard infrequent access, one zone infrequent access and glacier.
The standard tier has good performance, durability, and availability. It also has a first byte latency of milliseconds and a minimum of 3 availability zones.

The standard infrequent access has the same features as the standard tier but as the pricing for storing data gets cheaper, the price for retrieving the data gets more expensive.
The one zone infrequent access tier is like the standard infrequent access, but it only has one availability zone.
The glacier is like the standard infrequent access, but the pricing is way cheaper for storing data and way slower and more expensive to retrieve it.
There are two ways to change a bucket tier setting dynamically:

- There is a service called lifecycle management that allows to setup an s3 bucket to change its resources storage tier over time.
- There is a special tier called S3 Intelligent Tiering. This tier, while charging a little bit extra, analyzes the data usage and changes its tiering automatically to the most cost-effective option.

As the specific resource usage will vary depending on the current projects the company is handling at any given time, the S3 Intelligent Tiering presents as a very good option and will be the one used.

Expected storage and data transfers

Expected storage

Our expected storage is 8,36 GB with an expected growth of 1,48 GB per year

Web services

Our web services handle X requests / month. This web services are now using resources integrated in their back-end that will be replaced with calls to this service so I need to consider the requests that this related web services handle. This last month our aggregated requests were X.

Consultants usage

The expected resource upload/download of our 6 consultants, based on their current workflow, is 220 per month.

Aggregate Usage

The aggregate usage is the web services usage plus the consultants usage which is X + 220 = Y
Per month

Cost Estimation

Storage

The expected cost for storing our resources in an s3 storage service is 8,36 GB * 0,023 $ per month = 0,19 $ per month

Data transfers

The expected cost for data transfer is no easy to estimate. We have Y expected requests per month, but the pricing is calculated depending on the amount of GB transferred. The first gigabyte is free. The next tier is from 1 GB to 10 TB per month with an associated cost of 0,09 $ per GB per month. This is the tier we expect this service to be.
The average weight of the current versions of the standards that TradeHeader works with is 110 MB.

This leads us to an expected data transfer of Y * 110 MB = Z MB
Z MB are Z / 1024 = Q GB

The estimated total data transfer cost is Q GB * 0,09 $ = W $ per month

API Gateway

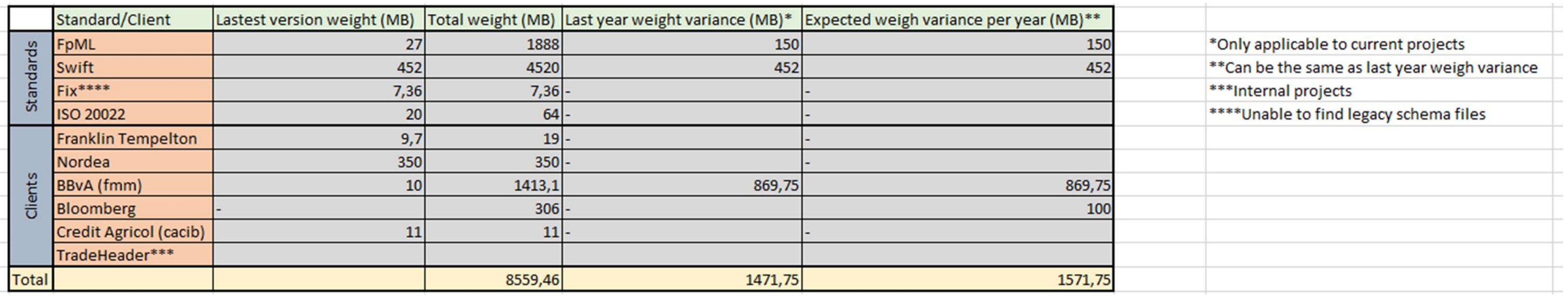
The API gateway cost is 3,50 $ per million request/year for the first 333 million requests.
As we are expecting Y requests per month, this means Y * 12 = E requests per year.
This results in a cost of (3,50/10^6) * E = R $ per year. This results in R/12 = T $ per month.

Total cost estimation

The total cost estimation is 0,19 $/month + W $/month + T $/month = U $/month

File Size Estimation

To design a repository system we need to know the amount of data that we will be able to handle. I have investigated the size of the different resources that the system will need to hold by analyzing each one of the different standards and clients. I have also investigated the size growth of the last year and estimated the size growth of the resources in the future.



	Standard/Client	Lastest version weight (MB)	Total weight (MB)	Last year weight variance (MB)*	Expected weigh variance per year (MB)**
Standards	FpML	27	1888	150	150
	Swift	452	4520	452	452
	Fix****	7,36	7,36	-	-
	ISO 20022	20	64	-	-
Clients	Franklin Tempelton	9,7	19	-	-
	Nordea	350	350	-	-
	BBvA (fmm)	10	1413,1	869,75	869,75
	Bloomberg	-	306	-	100
	Credit Agricol (cacib)	11	11	-	-
	TradeHeader***				
Total			8559,46	1471,75	1571,75

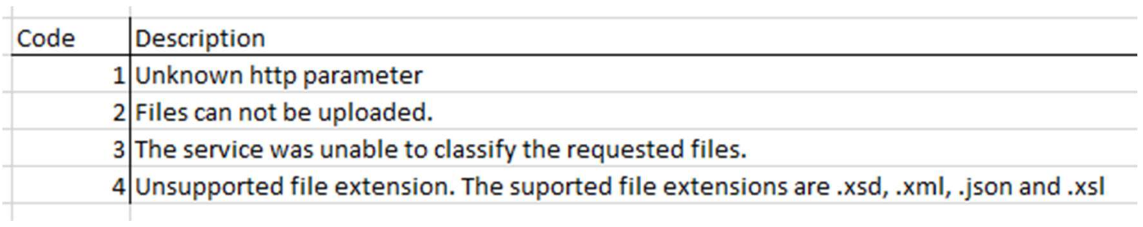
*Only applicable to current projects
**Can be the same as last year weigh variance
***Internal projects
****Unable to find legacy schema files

2. Requisites

2.1 Functional requirements

- The service must be able to process 10.000 file requests per second.
- The service must be able to host 15 GB of files.
- The service capacity must be able to expand.
- The service must provide an endpoint that allows the users to upload based files.
- The service must be able to host the following standards:
    - XSL (Extensible Stylesheet Language) [39]
    - XML (Extensible Markup Language)
    - XSD (Xml Schema Definition) [40]
    - JSON (JavaScript Object Notation) [41]
    - JSON Schema [42]
- The service must not allow the user to upload files that are not included in the supported file extensions.
- The service must provide an endpoint that accepts content requests and serves them to the user.
- The service must be able to organize the uploaded files using flags from the http request and/or data extracted from the uploaded resource.
- The service must retrieve the requested files correctly.
- The service must report to the user using the error coding specification.
- The service will not provide a specific UI to generate http requests or receive and process http responses from the service API.

Error coding specification



Code	Description
1	Unknown http parameter
2	Files can not be uploaded.
3	The service was unable to classify the requested files.
4	Unsupported file extension. The suported file extensions are .xsd, .xml, .json and .xsl

2.2 Potential Risks

Misunderstanding with the users: not delivering the product that the users really need.

Qualitative analysis: Low probability, high impact -> Significant risk

Risk answer planification: Analyze the differences between the delivered product and the expected product. Analyze the required resources to address the product delivery. Adjust the timetable to fit the new time requirements.

Too much or too little details in the requirements.

Qualitative analysis: Medium probability, medium impact -> Significant risk
Risk answer planification: Rewrite the requirements document, keeping the objectives but adjusting the requirements level of detail.

Too many requirement changes.

Qualitative analysis: Low probability, medium impact -> Low risk
Risk answer planification: Analyze the cause of the past requirement changes. Adjust the cause of the changes.

Software bugs.

Qualitative analysis: Medium probability, medium impact -> Significant risk
Risk answer planification: Use functional tests to ensure the API response complies with the requirements.

Developers without the necessary training.

Qualitative analysis: Low probability, medium impact -> Low risk
Risk answer planification: Include training time for the developers in the time planning.

Performance issues.

Qualitative analysis: Low probability, medium impact -> Low risk
Risk answer planification: Analyze the performance of the web services that are available and use the service that fits best with the performance requirements of the project.

Not achieving time goals.

Qualitative analysis: Medium probability, medium impact -> Significant risk
Risk answer planification: A very tight temporal planification may lead to not achieving time goals and ultimately into not achieving the delivery date. Leaving room for time adjustments and modifications while doing the temporal planification is key to solve this issue.

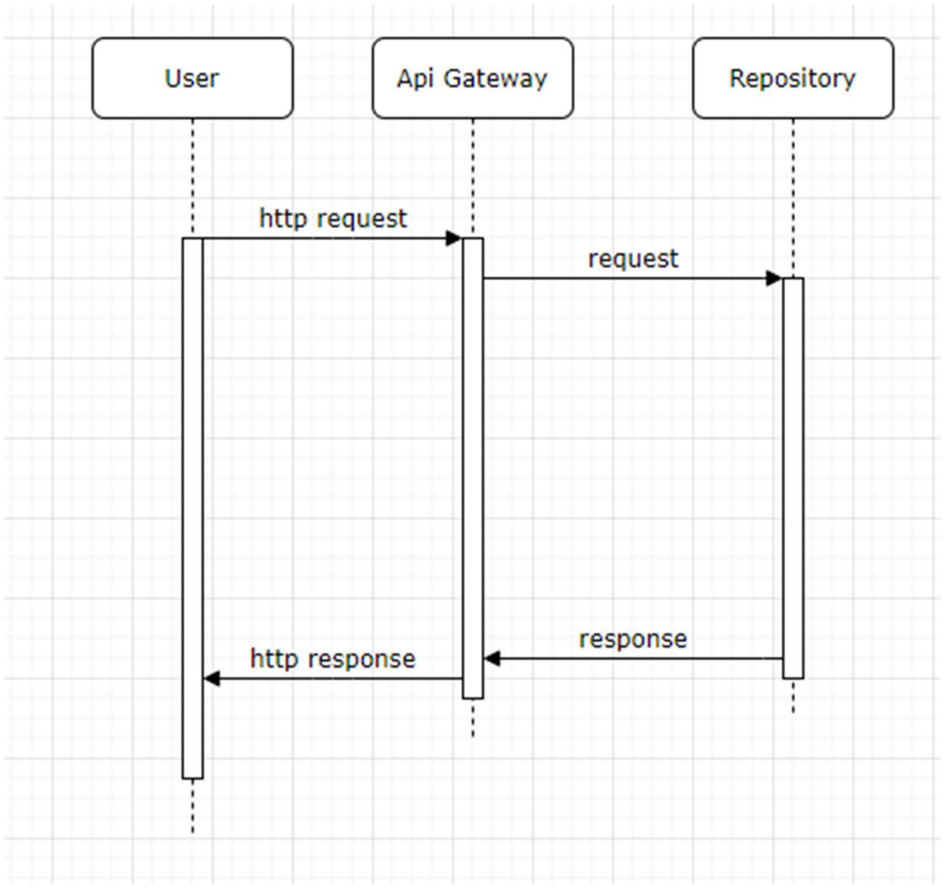
3. Design

3.1 Technologies

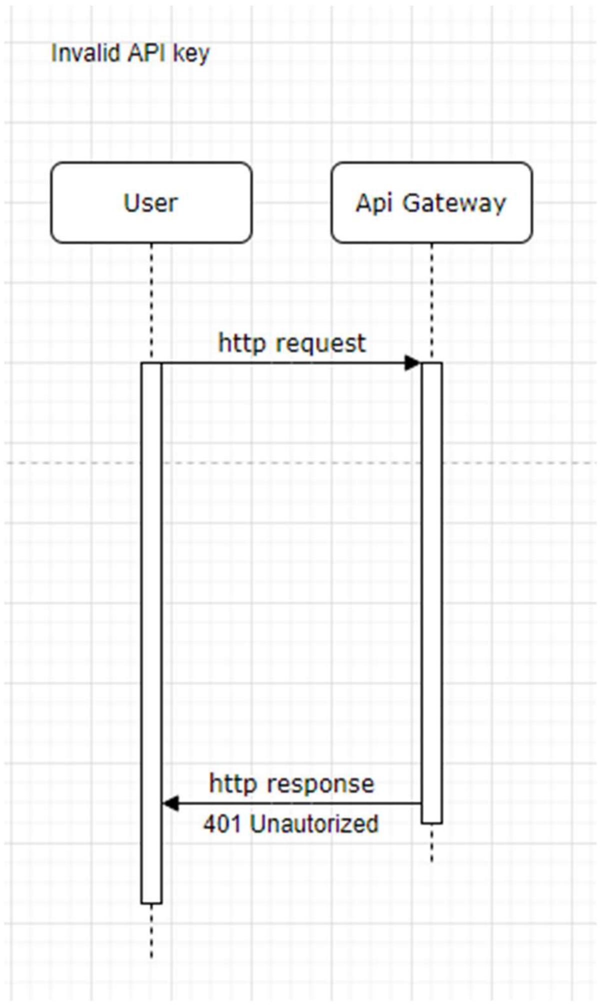
TradeHeader's software solutions live in the cloud: all the code is stored in cloud repositories and we offer a wide array of cloud services to our clients. Integrating this new service into our ecosystem is key to its success. Therefore, this project is going to be cloud-based.
Our cloud system needs a storage and a way for the resources to be uploaded and downloaded.
API REST [43] is a software architecture used to create web services. The REST architecture provides interoperability between computer systems on the Internet that allow systems to create requests and get responses. This requests and responses use the Http protocol to communicate with common Http methods (GET, POST, …)
The API REST architecture is very useful for defining operations over a backend which we can use to access our stored resources.
Every API needs a server to host it. As TradeHeader do not have any web server, a third-party hosting service is required to deploy our API.
Amazon Web Services will be the cloud platform of choice for this project as it offers a wide array of services that fulfill the necessities of the project, a fair and sustainable pricing scheme and the TradeHeader developers already have experience with the platform.
The back-end programing language will be Java as its multiplatform support and great libraries available make it a great fit.
Choosing an already known language for the main developer of the project is also a key factor as this will be the first API that will be designed and build by him.

3.1 Project Design

The system will have two main parts: The API service and the back-end repository. The repository will store the resources in a known way for the API service. This API will receive and process requests from the user, communicate with the resource repository and send a response to the user.



There will be a security layer. This will be handled by the API Gateway [44] validating the API key provided in the http requests it receives.
The AWS (Amazon Web Service) API Gateway provides an integrated API key service that allows you to generate several API key. Each API key can be enabled for each API and stage combination of your choice. This gives a very granular control over the permissions you attach to every API key.

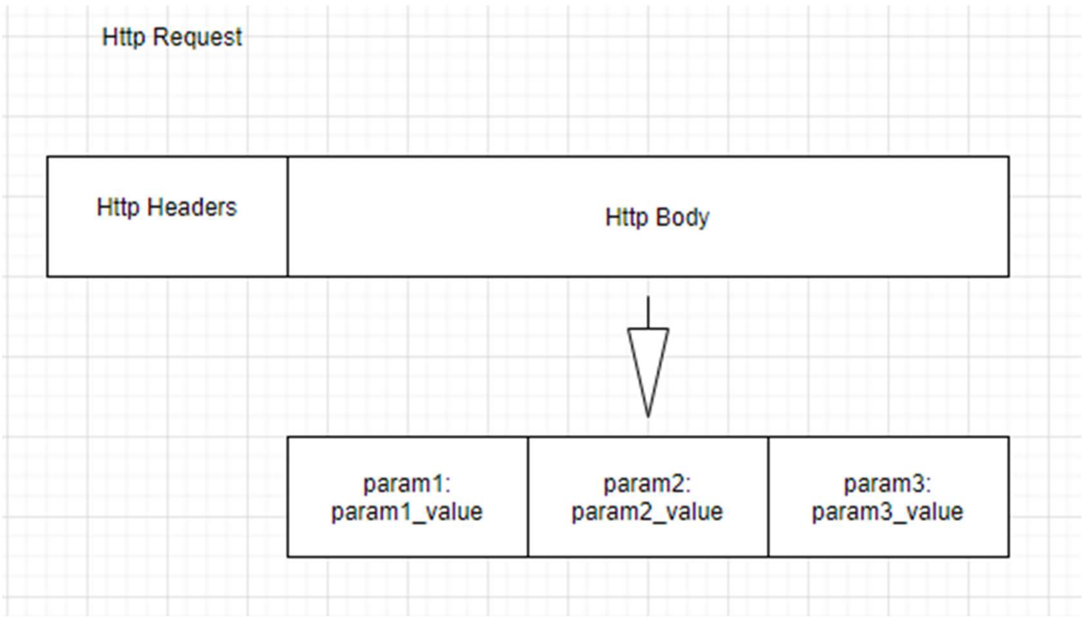


Regarding security as well, the S3 bucked that will hold our resources will only be accessible by our API Gateway. This way we can control how and by who our resources are being accessed.

Several flags will be required for the HTTP requests to be successful. These flags will need to contain information about the resource is being uploaded or downloaded.
This flag system has a handful of purposes:

- For the repository to be able to allocate the uploaded resources so they can be accessed in the future
- For the API Gateway to be able to ask the repository for the resources the user is requesting and retrieve them back to the user.
- For the user to be able to ask the for the actual resources stored in it and how to access them

Whenever a new file is uploaded to the S3 repository no further manual action should be needed as the request handler will use request parameters to recognize the actual s3 resource its being requested to serve to the client.
This also applies if you want to integrate a new standard: The API gateway will be able to find the request calls for the new resources.



API Endpoints

Host Url: ---- (not defined)

Upload file

This endpoint allows the user to upload a file to the System.

URL

/upload

Information

Method: PUT
Request format: application/json
Response format: application/json
Requires authentication? No

Sample request

```
curl -i -w '\n' -H "Content-Type: application/json" -d @file.xsd host_url/upload
```

Download file

This endpoint allows the user to download a file from the system using an specific standard, namespace and version

URL

/download/{standard}/{namespaces}/{version}

Information

Method: Get
Request format: application/json
Response format: application/octet-stream
Requires authentication? Yes

Parameters

Name: standard
Required: Yes
Description: Describes the standard of the required resource

Name: namespaces
Required: Yes
Description: Describes the namespaces of the required resource

Name: version
Required: Yes
Description: Describes the version of the required resource

Sample request

```
curl -i -w '\n' -H “x-api-key: $API_KEY” -H "Content-Type: application/json "  host_
url/download/fpml/http%253A%252F%252Fwww.fpml.org%252FFpML-
5%252Fconfirmation_http%253A%252F%252Fwww.w3.org%252F2000%252F
09%252Fxmldsig%2523 /5-11-8
```

List resources

This endpoint retrieves a list of the resources (grouped by standard, namespace and version) available on the system.

URL

/listResources

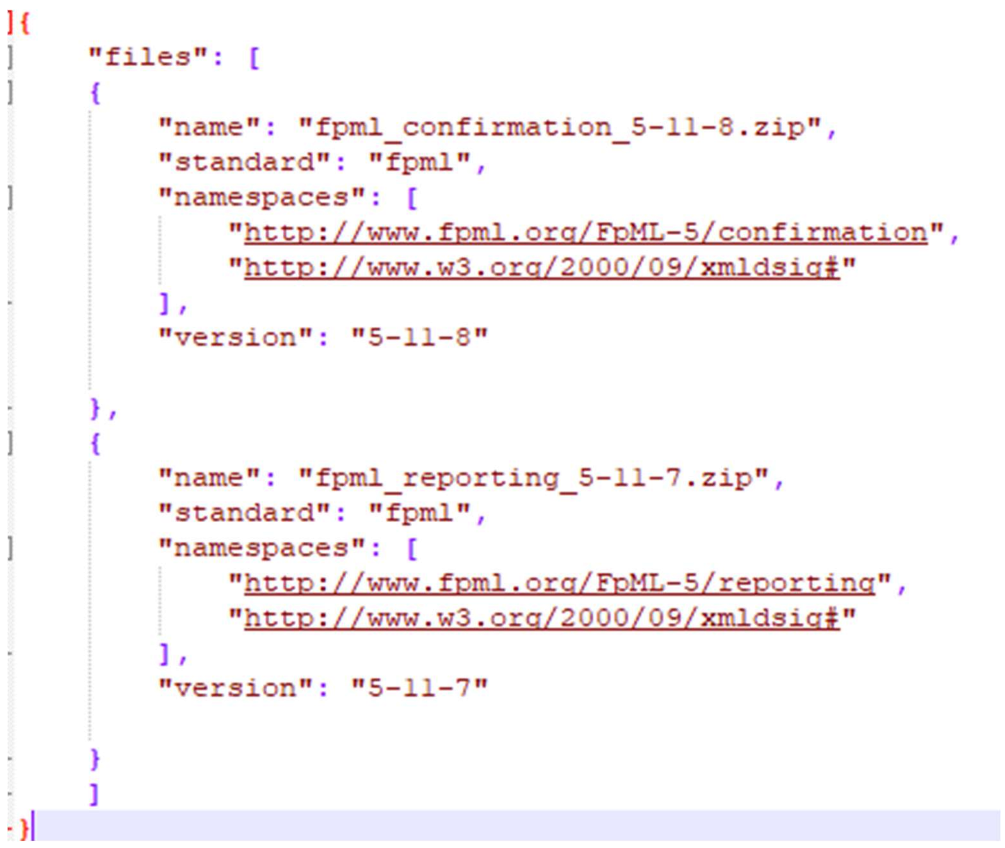
Information

Method: Get
Request format: application/json
Response format: application/json
Requires authentication? Yes

Sample Request

```
curl -i -w '\n' -H “x-api-key: $API_KEY” -H "Content-Type: application/json "  host_
url/download/fpml/ confirmation_xmldsi/5-11-8
```

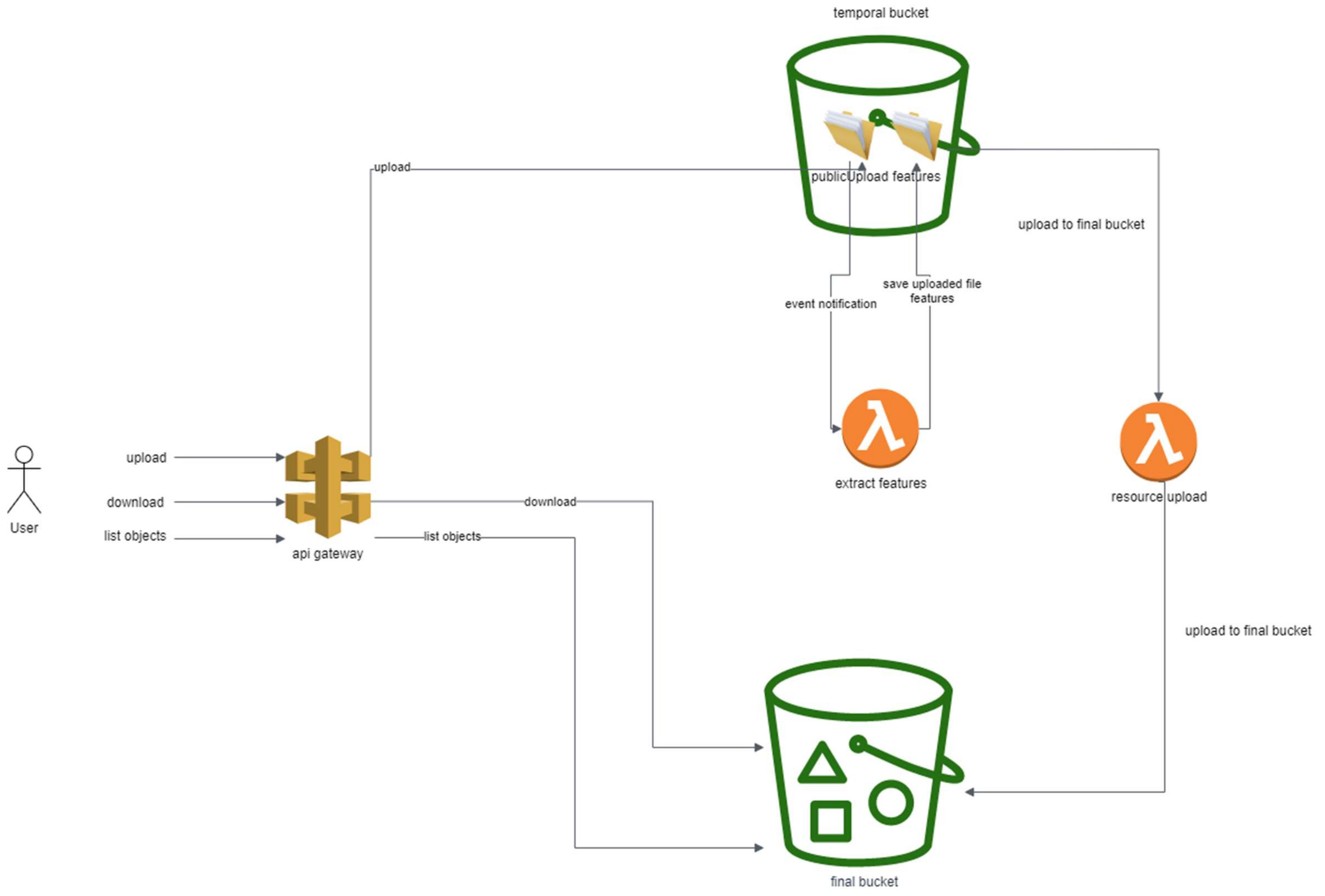
Sample Response



```
{
    "files": [
    {
        "name": "fpml_confirmation_5-11-8.zip",
        "standard": "fpml",
        "namespaces": [
            "http://www.fpml.org/FpML-5/confirmation",
            "http://www.w3.org/2000/09/xmldsig#"
        ],
        "version": "5-11-8"

    },
    {
        "name": "fpml_reporting_5-11-7.zip",
        "standard": "fpml",
        "namespaces": [
            "http://www.fpml.org/FpML-5/reporting",
            "http://www.w3.org/2000/09/xmldsig#"
        ],
        "version": "5-11-7"

    }
    ]
}
```


temporal bucket
publicUpload features
upload
upload to final bucket
event notification
save uploaded file features
upload
download
list objects
User
api gateway
download
list objects
extract features
resource upload
upload to final bucket
final bucket

4. Development

Naming

S3 buckets

External Naming	Project Naming
Temporal bucket	-
Final bucket	-
Front end	Th-resources-front-end

Lambda functions

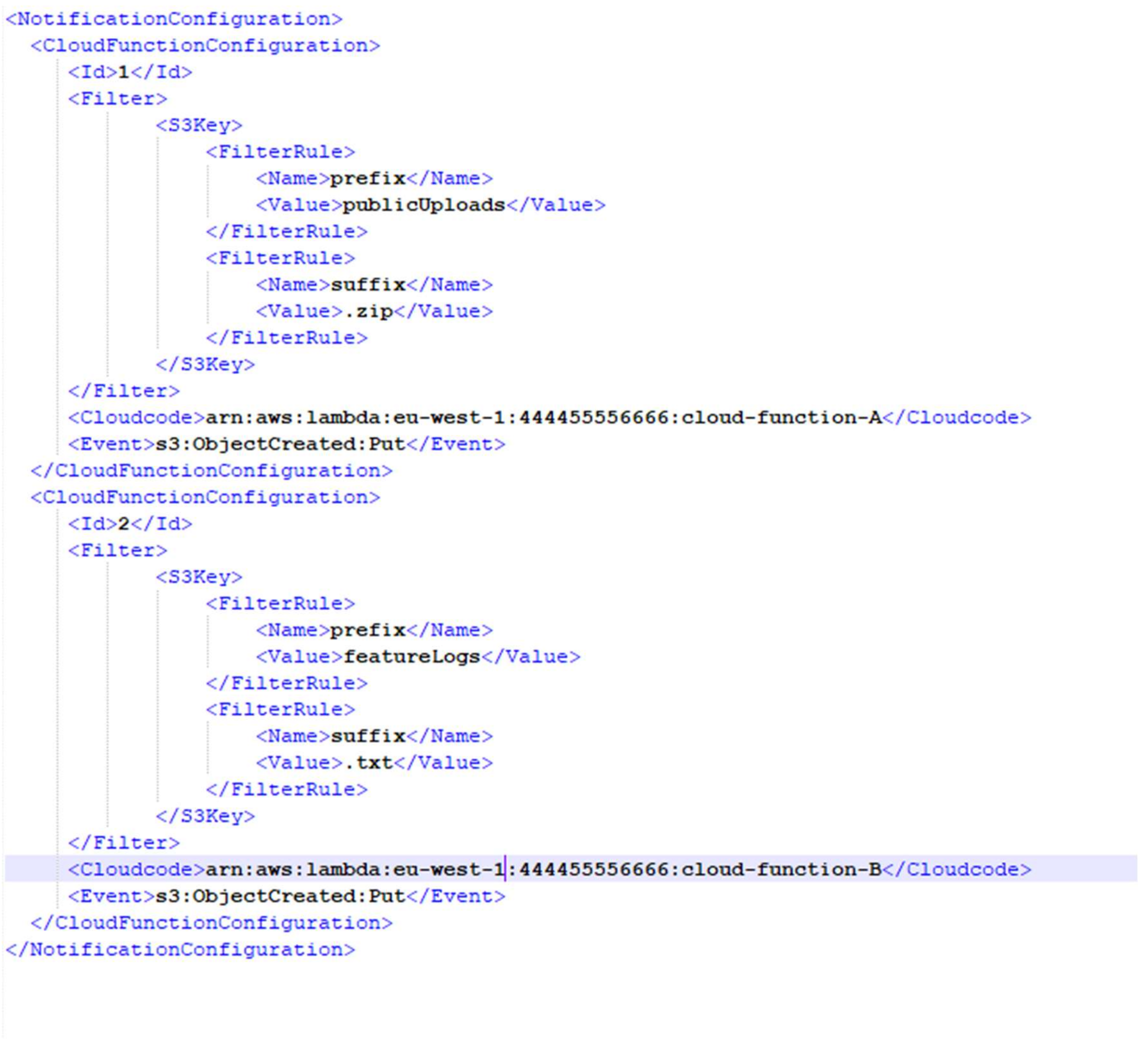
External Naming	Project Naming	Classpath	Role
Features extractor	schema-properties-extractor-lambda	src\main\java\com \tradeheader\aws \conversionHandler \DefaultExtractorHandler.java: handleRequest	S3 read/write to temporal bucket
File upload	resource-upload-lambda	src\main\java\com \tradeheader\aws \uploadFileHandler \DefaultUploadFileHandler.java: handleRequest	S3 read from temporal bucket & S3 write to final bucket
Uploaded resources	list-resources-lambda	src\main\java\com \tradeheader\aws \handler \ListResourcesHandler.java: handleRequest	S3 read from final bucket
File download	resource-retriever-lambda	src\main\java\com \tradeheader\aws \handler \ResourceRetrieverHandler.java: handleRequest.java	S3 read from final bucket

Repository

This part of the system has been implemented using two S3 buckets [45]: temporal and final bucket.
The temporal bucket has been set up with a public access and accepts the file upload requests from the user. The temporal bucket has two separate folders:

- The first one holds the public file uploads.
- The second one holds the features extracted (by the properties extractor lambda) of the uploaded files.

The folder that holds the public file uploads triggers, on new file uploads, an event notification [46] that has been hooked up with a lambda function [47] called properties extractor.
The folder that holds the features extracted triggers, on new features extracted, an event notification linked with the lambda called file upload.
This is the configuration file for the event notifications on the temporal S3 bucket



```
<NotificationConfiguration>
  <CloudFunctionConfiguration>
     <Id>1</Id>
     <Filter>
            <S3Key>
                <FilterRule>
                    <Name>prefix</Name>
                    <Value>publicUploads</Value>
                </FilterRule>
                <FilterRule>
                    <Name>suffix</Name>
                    <Value>.zip</Value>
                </FilterRule>
            </S3Key>
     </Filter>
     <Cloudcode>arn:aws:lambda:eu-west-1:444455556666:cloud-function-A</Cloudcode>
     <Event>s3:ObjectCreated:Put</Event>
  </CloudFunctionConfiguration>
  <CloudFunctionConfiguration>
     <Id>2</Id>
     <Filter>
            <S3Key>
                <FilterRule>
                    <Name>prefix</Name>
                    <Value>featureLogs</Value>
                </FilterRule>
                <FilterRule>
                    <Name>suffix</Name>
                    <Value>.txt</Value>
                </FilterRule>
            </S3Key>
     </Filter>
     <Cloudcode>arn:aws:lambda:eu-west-1:444455556666:cloud-function-B</Cloudcode>
     <Event>s3:ObjectCreated:Put</Event>
  </CloudFunctionConfiguration>
</NotificationConfiguration>
```

S3 bucket folders

Temporal bucket

Action	Key
File upload	uploads/fileName.extesion
Feature upload	features/fileName_features.json

Final bucket

Action	File key
File upload	standard/namespace/version/ fileName.extesion
Feature upload	standard/namespace/version/ fileName_features.json

The file keys are build using three parameters:

- standard
- namespace
- version

While the standard and version can be directly extracted from the feature file, the namespace parameter needs to be processed.
Unlike the standard and version parameters, that are strings, the feature file contains a list of strings that describe the sets of namespaces of the schema. In top of this difference, the XML namespaces usually have, by convention, in URL format. This format contains invalid characters to use as identifiers of a folder in an AWS S3 bucket.
To solve the first difference, the first adaptation performed to the target namespace list is to order it alphabetically. This way I will avoid duplicities that may arise by different orders of the same set of target namespaces. Once the list is ordered, I iterate through it encoding every target namespace and appending each of the encoded target namespace using the underscore sign to link them.

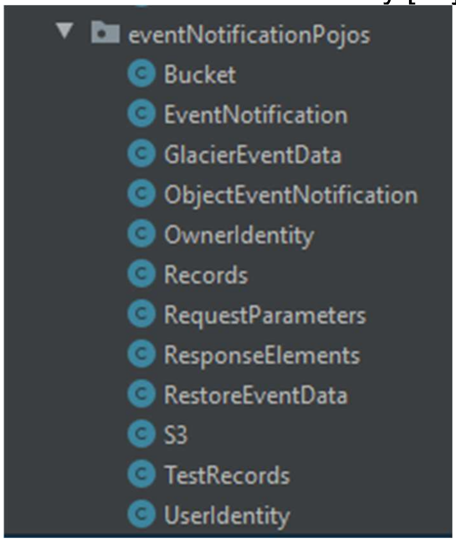
Lambda functions

There are several lambda functions:

- Features extractor
- File Upload
- List resources
- Resource retriever

Event Notification

Both lambda functions receive in their RequestHandler function an InputStream [48] that contains the event notification. Given that there is a definition on the parameters that form an event notification, I have built a set of pojos [50] that I use with the Jackson library [51] to deserialize the request.



Features extractor

The features extractor lambda extracts the key parameter (name of the file that has been uploaded) from the event notification and uses the AWS SDK library [49] (s3Client.getObject) to get the file from the S3 bucket.

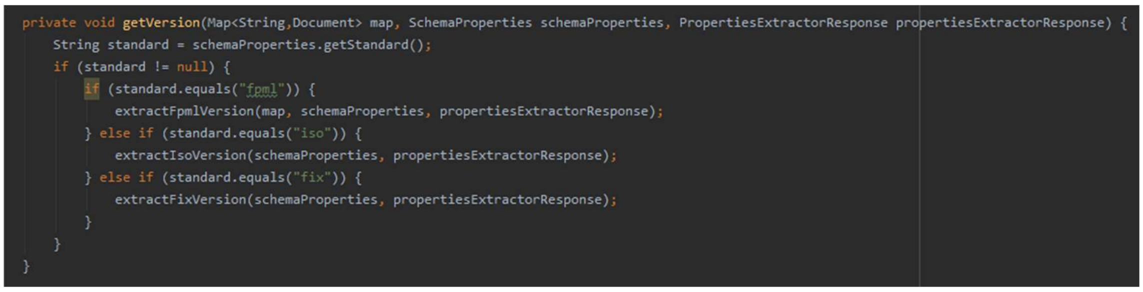
Once the uploaded file has been retrieved from the S3 bucket, the properties extractor, that is expecting a zip file, puts each one of the files that where located inside the zip file and potentially form the schema instance inside a HashMap using the name of each file as key and the content as value.
This function analyzes key features: standard, namespace and version. It generates a response with these features that will be stored as a properties file in a special folder in the temporal S3 bucket.
The features extractor function has support for xml schema files (.xsd extension).
The features that are extracted from the uploaded schema files are:

- Standard
- Namespace
- Version

These features have been selected because they can be used to identify any uploaded xml schema file and can be extracted from any xml schema file.

There are multiple financial standards that use xml schema files. This tool can process and extract features from three different financial standards:

- FpML
- FIX
- ISO 20022



```
private void getVersion(Map<String,Document> map, SchemaProperties schemaProperties, PropertiesExtractorResponse propertiesExtractorResponse) {
    String standard = schemaProperties.getStandard();
    if (standard != null) {
        if (standard.equals("fpml")) {
            extractFpmlVersion(map, schemaProperties, propertiesExtractorResponse);
        } else if (standard.equals("iso")) {
            extractIsoVersion(schemaProperties, propertiesExtractorResponse);
        } else if (standard.equals("fix")) {
            extractFixVersion(schemaProperties, propertiesExtractorResponse);
        }
    }
}
```

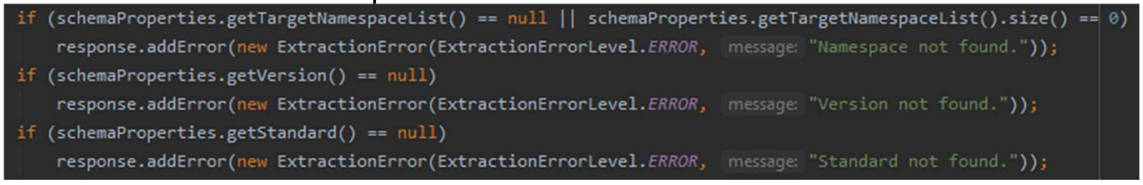
Each standard uses its own set of namespaces. Once the set of namespaces that describes a schema is extracted, this set is compared to a list of known namespace sets so that we can know the standard of the xml schema.

The process of determining the version of the xml schema varies depending on its standard. FIX and ISO 20022 contain this information within the xml schema file but FpML works differently. The FpML version has three parts:

- Major version
- Minor version
- Build number

The major and minor versions can be extracted from the file name of any of the files that form the xml schema. The build number is included in a specific attribute (actualBuild) from a specific xml file (fpml-doc)

This function also controls if the features have not been extracted as expected and adds errors to the response.



```
if (schemaProperties.getTargetNamespaceList() == null || schemaProperties.getTargetNamespaceList().size() == 0)
    response.addError(new ExtractionError(ExtractionErrorLevel.ERROR,  message: "Namespace not found."));
if (schemaProperties.getVersion() == null)
    response.addError(new ExtractionError(ExtractionErrorLevel.ERROR,  message: "Version not found."));
if (schemaProperties.getStandard() == null)
    response.addError(new ExtractionError(ExtractionErrorLevel.ERROR,  message: "Standard not found."));
```

File upload

There will be another lambda function called file upload that, once a properties file is uploaded to the temporal S3 bucket, it receives an event notification.
After deserializing the received event notification, it gets the feature files generated from the Features Extractor lambda that generated the event notification. This feature file contains the original file name. Using this name, it requests the original file from the temporal S3 bucket.
Once both: the file that was uploaded by the user originally and the features file generated by the features extractor have been retrieved by the file upload lambda, it uploads them to the final bucket.

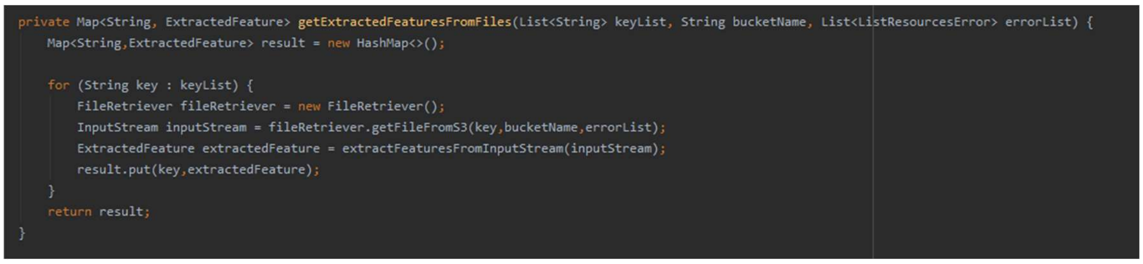
This architecture has been designed to build a system that is able to extract and save the features from the uploaded files while avoiding the potential problems that could happen while uploading big files because of the lambda limitations. The most impacting limitation from the AWS lambda functions is the payload limitation [52]: 6MB. It could cause issues if we tried to use the lambda requests or responses to include the uploaded files.

List resources

This lambda function lists the available files and their features stored in the final repository.

To do so, it uses de AWS SDK method listObjects. From the retrieved list, it filters the files that end with *"_features.json"* and build a list with the feature file keys.

Then, this function gets all the feature files, reads them, and returns a map with the file name as the key, and its features as each entrance value.



```java
private Map<String, ExtractedFeature> getExtractedFeaturesFromFiles(List<String> keyList, String bucketName, List<ListResourcesError> errorList) {
    Map<String,ExtractedFeature> result = new HashMap<>();

    for (String key : keyList) {
        FileRetriever fileRetriever = new FileRetriever();
        InputStream inputStream = fileRetriever.getFileFromS3(key,bucketName,errorList);
        ExtractedFeature extractedFeature = extractFeaturesFromInputStream(inputStream);
        result.put(key,extractedFeature);
    }
    return result;
}
```

Resource retriever

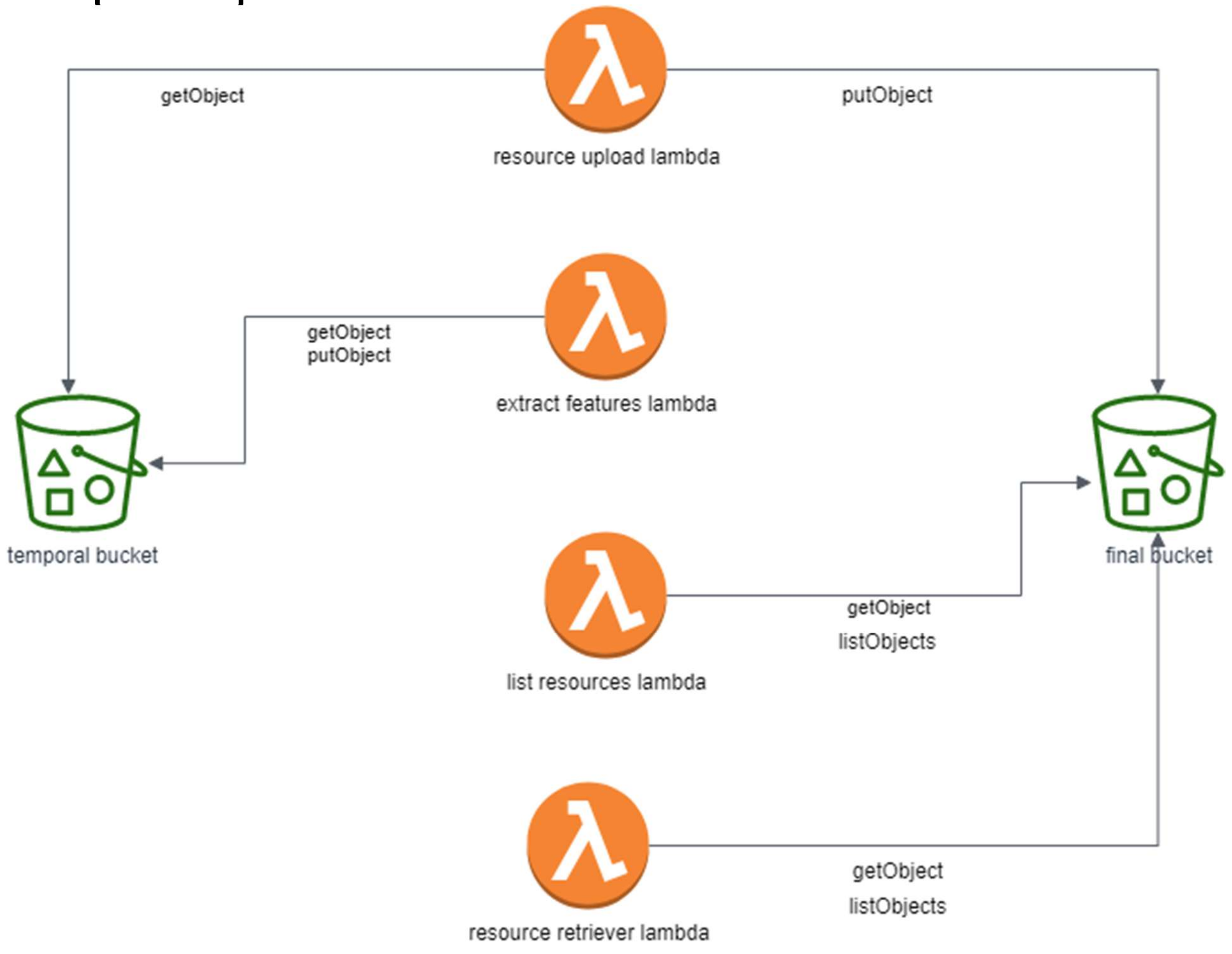
This lambda gets the required file from the S3 final bucket and returns it in a binary format.

It expects a set of features as a request. The handler receives an input stream which it gets deserialized into a POJO.

Then the lambda function lists all the features stored in the final S3 bucket and gets a list of keys of the files that match the requested features.

If the filtered list of file keys is empty or has more than one match the lambda function answers with an error. If there is only one match, requests the matched key to the final S3 bucket and adds the requested file to the response.

Required permissions



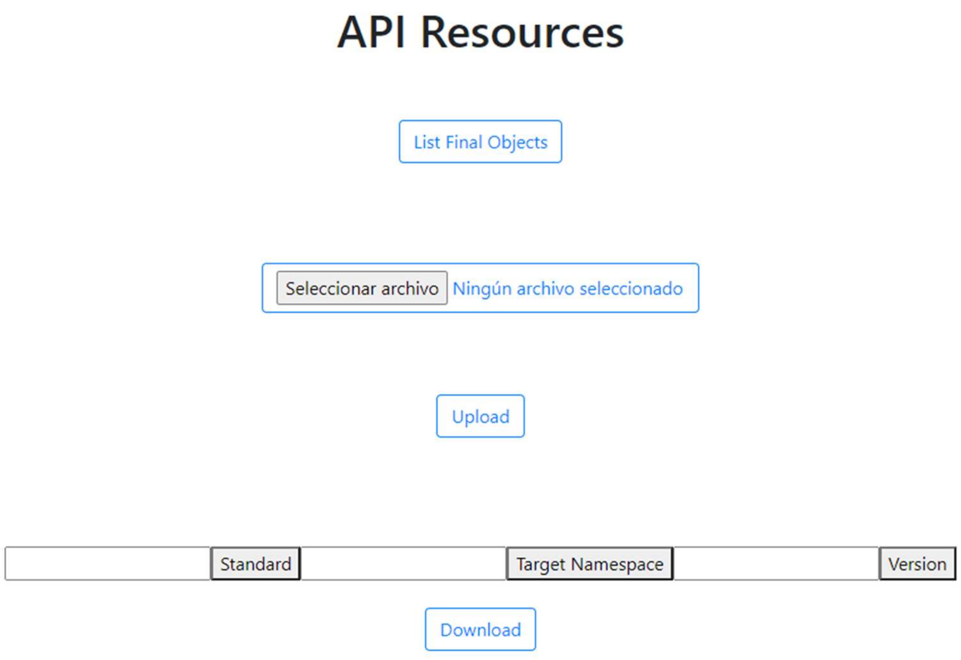
For security sake, both, the list resources lambda, and the resource retriever lambda can only be accessed from the API gateway. The final bucket can only be accessed through the API gateway. By configuring the roles that can get access to the appropriate API gateway endpoint, we can control who gets access to the resources stored in the system. This system complies with the necessity of confidentiality of the resources stored in the system

Front end

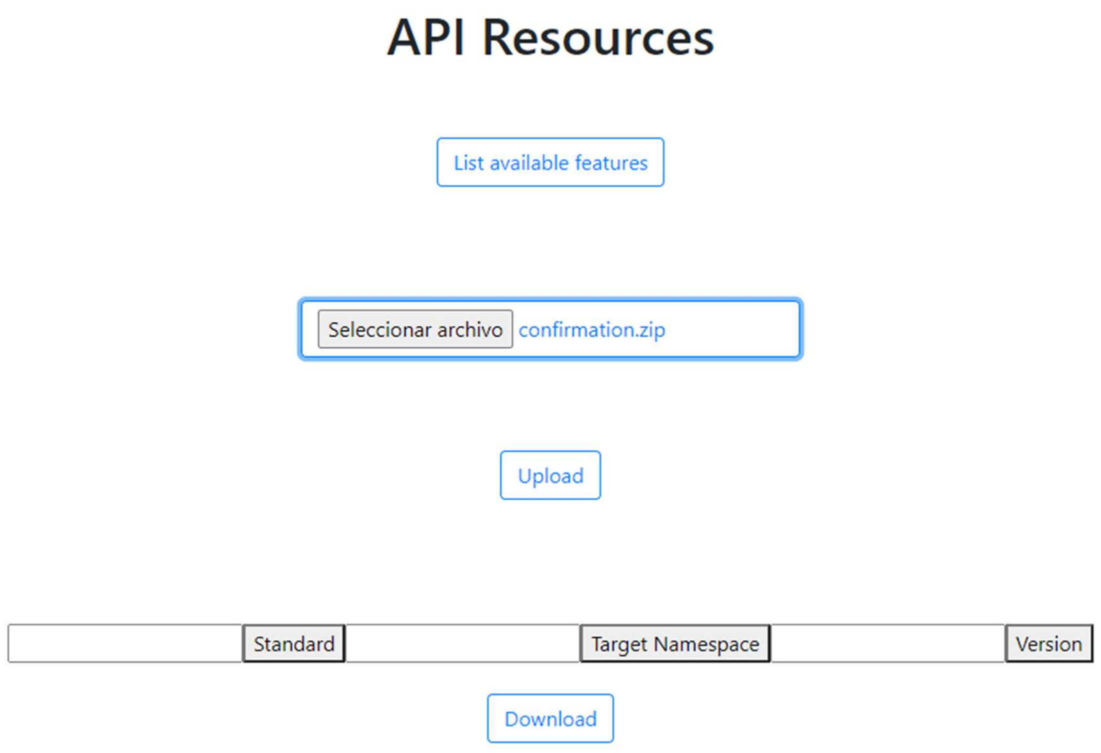
There were no plans to develop a front-end interface in the initial stage of this project. The teacher from the college suggested it in a revision of the project which produced in TradeHeader an internal re-evaluation of the project and agreed on the value that a front end for this system could provide to the company.

The front-end interface is a very simplistic solution but is able to accomplish the necessity to provide a visual solution for the user to interact with the system.

Home



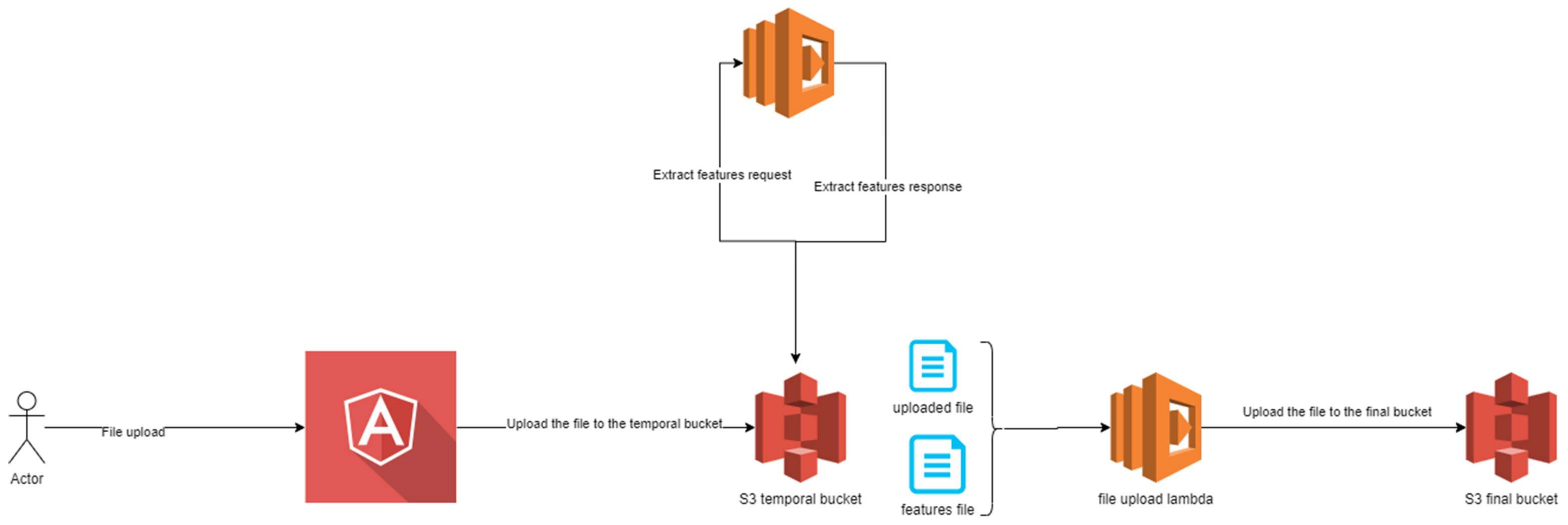
Upload File



The front end is hosted within an S3 bucket. A CORS configuration file has been uploaded.

```
<?xml version="1.0" encoding="UTF-8"?>
<CORSConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
        <CORSRule>
                <AllowedOrigin>*</AllowedOrigin>
                <AllowedMethod>HEAD</AllowedMethod>
                <AllowedMethod>GET</AllowedMethod>
                <AllowedMethod>PUT</AllowedMethod>
                <AllowedMethod>POST</AllowedMethod>
                <AllowedMethod>DELETE</AllowedMethod>
                <ExposeHeader>ETag</ExposeHeader>
                <AllowedHeader>*</AllowedHeader>
        </CORSRule>
</CORSConfiguration>
```

Upload file process



5. Testing

Testing this project has been a challenge since its implementation is very tied with the production environment.

The approach used is the inclusion of unitary tests on the lambda functions.

The tight relationship between the project implementation and the production environment caused that the first tests designed could not be used, as they were dependent on specific files to be present in the production environment.

To avoid this problem, the design of the testing changed to unitary tests that focused on specific processes of the lambda functions.

The resource feature extractor lambda has five different tests:

- Test FpML version
- Test ISO version
- Test FIX version
- Test Standard
- Test target namespace

While the target namespace and the standard processes are common between the different standards, the version extraction process varies depending on the standard. Therefore, each version retrieving processes needs its own testing.

The update file lambda has three unitary tests implemented:

- Two tests checking the extraction of features from the request
- A test that checks if the features file is updated as expected

All the unitary tests have the same procedure:

- Use a controlled input file to generate a request
- Call a specific part of the lambda function
- Compare the generated result with the expected result

6. Conclusions

This is an ambitious project: From writing the whole project documentation in English (which is not my native language) to designing a project from scratch (which I had never done before) in a completely new set of technologies (cloud).

I have invested a lot of hours and energy in investigating, designing, and implementing this project and I have learned a lot while working on it.

Lessons learned

It is hard to number all the things that I learned during this process, but I can confidentially say that I have learned a lot about cloud technologies, how requests and responses work and also a lot of Amazon Web Services specific information about their technologies.

I have invested a lot of time investigating and reading documentation, part of the technologies investigated where not applied in this project (like CloudFormation).

By completing this project I am much more confident while working with cloud technologies and eager to keep learning about them.

Objectives

Functional Objectives

The user has three available actions to interact with the system

- Upload file
- List files in the system
- Download file

The upload file process has been successfully designed and implemented: Once a user uploads a file, the system extracts features from the uploaded file and stores both: the uploaded file and the features file as expected.

In the requirements phase we determined the expected file types supported. Nowadays the system only has support for XSD files. This functionality will be updated to match the original list in the future.

The list files in the system functionality also works as expected: When the client uses the specified endpoint, the system reads the available features and lists them to the user.

The download file is not complete: There is a lot of work done in the direction of having a functional endpoint, but I encountered a technical problem that I could not solve on time. I was not able to resolve this problem because I had too little time left once I found out about the problem. The completed and incomplete characteristics are further explained in the implementation objectives.

Implementation objectives

This system has seven parts:

- Two S3 buckets
  - Temporal bucket
  - Final bucket
- Four lambda functions
  - Features extractor
  - File upload
  - List resources
  - File retriever
- One API gateway

The two S3 buckets and their configuration work as expected: Once a file is uploaded to the temporal bucket folder uploads, an event notification is generated triggering the features extractor lambda. The temporal bucket has another trigger associated that generates an event notification calling the file upload lambda once a file is uploaded to the features folder.

Both these buckets and the triggers associated with the temporal bucket work as planned.

The feature extraction lambda receives the event notification generated by the temporal S3 bucket trigger. This lambda extracts the required features from the uploaded file and saves them as a features file in the features folder of the temporal bucket as expected.

The upload file lambda receives the event notification generated by the upload of the features file. This function uploads both, the original and the feature files to the final repository as expected.

The list files lambda receives an empty request. This function lists all the files in the final bucket, filters the features ones, reads its contents, and returns a map containing all the features as expected.

The file retriever lambda receives a set of features, lists all the available features in the final bucket and matches them with the ones from the request. If the number of matches is different than one it returns an error. If there is only one match it gets the file from the final bucket and returns it as binary. While this function matches the features correctly, I have not been able to retrieve the binary data in the response of this function to the client side. I will fix this issue in the future.

The API Gateway that unifies all the different systems and provides the user with usable endpoints has also been implemented. The download file endpoint, even though the function has a problem, is able to connect with the lambda function that manages the file download process and returns the response to the user.

The testing that I designed initially used the production environment to execute. I detected this was a problem and I changed the testing implementation so that it was not depending on the production environment to execute. While I think the implemented unitary tests implemented are adequate, I would like to complement them with other kinds of tests in the future like load testing.

Given that this is a TradeHeader's internal project, its development will continue by completing the objectives listed in this project and potentially adding features in the future.

Planification

The general idea was right from the beginning, but at the start of the project I was not able to generate a detailed architectural design because of my lack of knowledge in the area. Diagrams like the ones from pages 22 and 29. If I would have had the knowledge to make this diagrams from the beginning, most of the time issues that I have faced in the latest phases of the project would not have occurred.

The main missing idea from the initial planification was the lack of usage of lambda functions. Originally I planned to connect the API Gateway with the S3 bucket but the different actions that I planned the system to support required file processing. This file processing has been implemented by the four AWS lambda functions. This is how while keeping the original idea I have adapted the implementation to the necessities of the project.

The partial planification points where completed as expected, but once on the implementation phase, when I needed to implement a specific functionality which I previously planned during the design phase I needed to investigate the specific details on how to do that implementation which was very time consuming.

Future plans

This project is still an important project for TradeHeader. The project development will not stop once this project gets delivered.

The first step will be to fix the issues related with the downloads so that we have a completely functional system. The next steps will be reinforcing the testing process and widen the number of file types supported by the system to cover the once specified in the requirements section. Once this is working, we will focus on integrating this solution with existing services and start using it in a production environment.

We will need to develop a more sophisticated and feature-rich front-end interface.

Bibliography

[1] FPML: https://www.fpml.org/
Visited the 31/03/2020

[2] ISO 20022: https://www.iso20022.org/iso-20022-message-definitions
Visited the 31/03/2020

[3] Swift: https://www2.swift.com/mystandards/#/c/baseLibraries
Visited the 31/03/2020

[4] ISDA: https://www.isda.org/
Visited the 31/03/2020

[5] FIX https://www.fixtrading.org/
Visited the 31/03/20

[6] AWS: https://aws.amazon.com/
Visited the 02/04/2020

[7] AWS S3 pricing https://aws.amazon.com/s3/pricing/
Visited the 02/04/2020

[8] AWS Api Gateway pricing https://aws.amazon.com/api-gateway/pricing/
Visited in the 02/04/2020

[9] AWS CloudFormation template anatomy:
https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/template-anatomy.html
Visited the 21/04/2020

[10] AWS S3 Transfer Acceleration:
https://docs.aws.amazon.com/AmazonS3/latest/dev/transfer-acceleration.html
Visited the 21/04/2020

[11] AWS ACL canned acl:
https://docs.aws.amazon.com/AmazonS3/latest/dev/acl-overview.html#canned-acl

Visited the 21/04/2020

[12] AWS GetBucketAnalyticsConfiguration:
https://docs.aws.amazon.com/AmazonS3/latest/API/API_GetBucketAnalyticsConfiguration.html
Visited the 21/04/2020

[13] AWS S3 Default Encryption for S3 Buckets:
https://docs.aws.amazon.com/AmazonS3/latest/dev/bucket-encryption.html

Visited the 21/04/2020

[14] AWS Cross-origin resource sharing (CORS):
https://docs.aws.amazon.com/AmazonS3/latest/dev/cors.html
Visited the 21/04/2020

[15] AWS GetBuckedInventoryConfiguration:
https://docs.aws.amazon.com/AmazonS3/latest/API/API_GetBucketInventoryConfiguration.html
Visited the 21/04/2020

[16] AWS S3 Inventory:
https://docs.aws.amazon.com/AmazonS3/latest/dev/storage-inventory.html
Visited the 21/04/2020

[17] AWS Object lifecycle management:
https://docs.aws.amazon.com/AmazonS3/latest/dev/object-lifecycle-mgmt.html
Visited the 21/04/2020

[18] AWS LoggingConfiguration:
https://docs.aws.amazon.com/es_es/AWSCloudFormation/latest/UserGuide/aws-properties-s3-bucket-loggingconfig.html
Visited the 21/04/2020

[19] AWS PutBucketMetricsConfiguration:
https://docs.aws.amazon.com/AmazonS3/latest/API/API_PutBucketMetricsConfiguration.html
Visited the 21/04/2020

[20] AWS api reference: https://docs.aws.amazon.com/apigateway/api-reference/resource/method/
Visited the 22/04/2020

[21] Json plugin: https://github.com/aws-cloudformation/cfn-python-lint
Visited the 24/04/2020

[22] Template schema: https://github.com/aws-cloudformation/aws-cloudformation-template-schema
Visited the 24/04/2020

[23] Supercharging your editor: https://hodgkins.io/up-your-cloudformation-game-with-vscode
Visited the 24/04/2020

[24] Upload file to s3 using java:
https://docs.aws.amazon.com/AmazonS3/latest/dev/UploadObjSingleOpJava.html
Visited the 30/04/2020

[25] AWS java handler: https://docs.aws.amazon.com/lambda/latest/dg/java-handler.html
Visited the 03/05/2020

[26] AWS lambda limitations:
https://docs.aws.amazon.com/lambda/latest/dg/gettingstarted-limits.html
Visited the 04/05/2020

[27] AWS lambda serverless large files:
https://medium.com/circuitpeople/serverless-large-file-downloads-to-s3-a11b4ef4788e
Visited the 05/05/2020

[28] AWS states Language: https://states-language.net/spec.html
Visited the 05/05/2020

[29] AWS upload file http client with credentials:
https://docs.aws.amazon.com/AmazonS3/latest/API/sigv4-post-example.html
Visited the 12/05/2020

[30] API Gateway configuration:
https://docs.aws.amazon.com/apigateway/latest/developerguide/integrating-api-with-aws-services-s3.html

Visited the 02/06/2020

[31] CFTC: https://www.cftc.gov/
Visited the 16/06/2020

[32] ESMA: https://www.esma.europa.eu/
Visited the 16/06/2020

[33] Fix Standard: https://www.fixtrading.org/online-specification/
Visited the 16/06/2020
[34] API: https://en.wikipedia.org/wiki/Application_programming_interface
Visited the 16/06/2020

[35] XML: https://www.w3schools.com/xmL/xml_whatis.asp
Vistied the 16/06/2020

[36] Back end: https://en.wikipedia.org/wiki/Front_end_and_back_end
Visited the 16/06/2020

[37] File storage vs block storage vs object storage
https://www.redhat.com/en/topics/data-storage/file-block-object-storage

Visited the 02/04/2020

[38] AWS S3: https://aws.amazon.com/es/s3/
Visited the 02/04/2020

[39] XSL:
https://www.w3schools.com/xml/xsl_languages.asp#:~:text=XSL%20stands%20for%20EXtensible%20Stylesheet,an%20XML%2Dbased%20Stylesheet%20Language.

Visited the 16/06/2020

[40] XSD: https://www.w3schools.com/xml/schema_intro.asp

Visited the 16/02/2020

[41] Json:_ https://www.json.org/json-en.html

Visited the 16/02/2020

[42] Json Schema: https://json-schema.org/

Visited the 16/02/2020

[43] API REST: https://en.wikipedia.org/wiki/Representational_state_transfer

Visited the 16/06/2020

[44] API Gateway: https://aws.amazon.com/es/api-gateway/
Visited the 16/06/2020

[45] AWS S3: https://aws.amazon.com/es/s3/
Visited the 16/06/2020

[46] AWS event notification:
https://docs.aws.amazon.com/AmazonS3/latest/dev/NotificationHowTo.html
Visited the 16/06/2020

[47] AWS Lambda function: https://aws.amazon.com/es/lambda/
Visited the 16/06/2020

[48] InputStream:
https://docs.oracle.com/javase/8/docs/api/java/io/InputStream.html
Visited the 16/06/2020

[49] AWS SDK: https://docs.aws.amazon.com/AWSJavaSDK/latest/javadoc/
Visited the 16/062020

[50] POJO: https://en.wikipedia.org/wiki/Plain_old_Java_object
Visited the 16/06/2020

[51] Jackson library: https://github.com/FasterXML/jackson
Visited the 16/06/2020

[52] AWS Lambda límits:
https://docs.aws.amazon.com/lambda/latest/dg/gettingstarted-limits.html

Visited the 16/06/2020