INSTITUTO POLITÉCNICO DE LISBOA
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

I-On Integration

Importing academic information from external systems for aggregation and distribution

Progress Report

Miguel Barbosa Teixeira
Samuel Sampaio Costa

BSc in Computer Science and Computer Engineering

Project and Seminary

Supervised by Prof. Pedro Félix and by Prof. João Trindade

May, 2020

INSTITUTO POLITÉCNICO DE LISBOA
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA
BSc in Computer Science and Computer Engineering

I-On Integration

Importing academic information from external systems for aggregation and distribution

43314 Miguel Barbosa Teixeira
43552 Samuel Sampaio Costa

Supervisors:
Prof. Pedro Félix
Prof. João Trindade

Progress Report written for the Curricular Unit of Project and Seminary

May, 2020

1 Acknowledgements

This document is stil in progress. In the next delivery this section will be completed

Contents

List of Figures

2 Introduction

Academic institutions need to publish information related to their academic activities to the public such as programmes curriculums, term calendar, timetables, evaluation schedules and others. Traditionally this information was posted on boards at campus during the beginning of each academic year or at beginning of each semester. Since the Web 2.0 boom [25] the information is now mostly available on the institution's website.

However it is very common that the information is simply uploaded as is, resulting in students and teachers being required to download multiple files spread throughout the website in order to consult it. Even then, in most cases, the files in question contain information pertaining to whole programme when in fact viewers only have interest in a limited subset of the information.

I-On is an academic information aggregation and distribution system that aims to tackle these issues, giving users the ability to configure what information is of interest to them and have it easily accessible in one place. I-On is composed of three sub-projects: Core, Android, Integration and all 3 work in tandem to achieve the goal.

This document will focus mainly on I-On Integration, however, a brief introduction of the I-On project organization can be found on part 3. I-On Integration has the responsibility of collecting the relevant data from external sources, parsing it and finally uploading it to I-On Core. The data will then be made available to the users via I-On Android.

I-On Integration is a *batch processing* [26] application, i.e. it allows to configure and run *batch jobs* [26]. The main advantage of using *batch jobs* is the possibility of scheduling them and ensuring that new data is published on external sources and can be automatically fetched and uploaded to I-On core without human interaction.

Instituto Superior de Engenharia de Lisboa (ISEL) [1] was chosen as use-case. ISEL mainly publishes academic information in *.pdf format*[2].

I-On also sets out to replicate, as much as possible, the industry's best practices. This includes, but not limited to, having quality and documented code, using *pull-requests* [2] as workflow, *continuous integration* and *continuous deployment* [31] with *automated testing*. As I-On continues to grow, these practices should ensure that future contributors have a good base to further improve the project.

This document is organized in 7 parts:

- Part 3 will describe the context of the problem I-On aims to solve in more depth.
- In Part 4 we will focus on the architecture design of the solution, while also highlighting the technologies that support it.
- Part 5 provides a discussion of the implementation details.
- Part 6 focus on the project limitations and how they affect the solution.
- Finally in Part 7 we survey possible future improvements to the project.

[1] Given the fact that the contributors for the I-On project are all enrolled in ISEL this was the best solution. Not only are the necessary documents easily available, we are also experienced with them and the project is supervised by teachers that lecture at ISEL

[2] **P**ortable **D**ocument **F**ormat developed by *Adobe*

2.1 Motivation

We set out to accomplish this project in order to provide a solution aligned with the needs of modern mobile application users. Nowadays it's very common to use a mobile application to access the information we want and most applications let users customize what and how that information is presented.

In the context for academic institutions this means that instead of going through the institution website looking for files to download that are not easy to visualize on a mobile device, we can create an application that display that same information in one place. We allow students to visualize what courses they are enrolled in, check their timetable and even create alerts for specific events such as exam dates. It also allows teachers to more easily share information with students.

It was also very important to create a project that could be further improved in future academic terms allowing students to be part of a growing project. At the same time better preparing us for the job market by making use of current technologies, applying the industry best practices and the knowledge obtained along the course.

3 Problem Formulation

The distribution of information relating to academic activities is critical to the operation of every educational institution, improving the quality of education and promoting cooperation across the different agents in the community it serves.

Each school's governing boards decide in what format it is published. Frequently, different boards inside the same school organize this information differently. Taking as an example the course-offer timetable information, the publication policy varies from one institution to another: some institutions publish a document per program, as is the case in ISEL, which results in an excess of irrelevant information for the people that consult these documents, others like ESELx (Escola Superior de Educação de Lisboa), provide a document per class-section.

Apart from format and granularity it is also worth considering where this information is published. For the schools reviewed, academic activity information is present in their website. However, the visibility of the buttons and the amount of clicks to get to it doesn't reflect the relevance it has for the public.[3]

For its consumers, students and teachers, it is desirable that academic information is easily accessible and tailored to the end-user. For example, in the start of the semester when a student wants to know at what times his classes are, it would be more appropriate if he didn't have to scan through a document containing the timetables for all the class-sections of the program. Furthermore, when a student is enrolled in classes from more than one class-section, he wouldn't have to assemble his own document for the purpose.

I-On attempts to solve the problem of accessibility, as it is an aggregator and distributor of academic information. It aims to support the academic activity. In its first iteration, I-On is composed of three sub-projects: Android, Core and Integration.

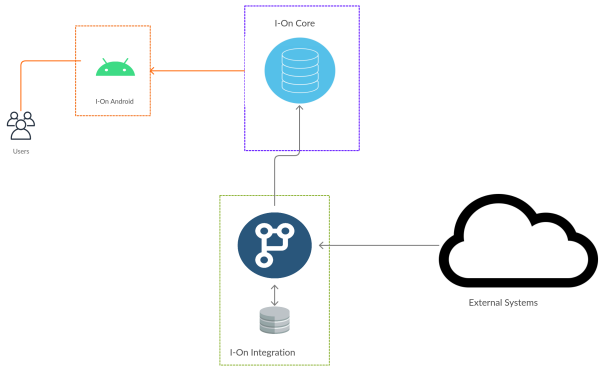

Figure 1: Overview of the I-On project organization

The integration module imports information from external systems, namely ISEL's website, parses and uploads it to I-On Core, which is the central repository. The core module exposes

[3] 4 schools from IPL were reviewed: ISEL, ISCAL, ESELx and ESTC. 2 schools from University of Lisbon were reviewed: FCUL and IST. All the schools reviewed publish timetables, CUF's, faculty, class-sections, exams and school calendar in their websites.

http APIs for communication with the Integration and Android sub-projects. The Android module consumes information from the Core and makes it available to end users, as is shown in figure 1.

A similar system to I-On is Fénix. Fénix was started by Instituto Superior Técnico in 2002, and is currently adopted on most schools of the University of Lisbon. Apart from solving the previously discussed issue, it is more broadly a content management system (CMS), assisting in diverse school activities like the application process, management of curricula, the issuing of diplomas, and other tasks associated with scientific production. Given the amount of code and its complexity, "getting new modules implemented" is "ever-increasingly" difficult [5]. The Fénix team comments in its global description document that time and money constraints advise against technology change within the project, leading to the technology in use being obsolete. Fénix has become Open-Source, whereas I-On is open-source from day one. Fénix has dedicated staff, and some feature extensions have been contracted to private consultants over the years. I-On is to be developed by students, with little costs involved. Having students as main contributors keeps costs down, while allowing knowledge to be shared and encouraging practice and experimentation.

The challenge in defining such a system is that it can include as many and diverse functionalities as the community sees fitting. Then, its development transcends what can be produced by one team during an academic semester. Thus, it is expected that I-On and accordingly I-On integration will span across the following semesters, and that different teams will be contributing for its code base over time. It is also expected that in its future iterations, its objectives will change, as new functionalities are added and existing ones adapt to changing requirements.

Moreover, as the project is open-source, it is available for individual contributors to open pull requests and merge their code to its code-base. In order to account for the ongoing nature of I-On, particularly the integration module, it was established that the configuration of development infrastructure would be regarded as a primary set of objectives, as important as the development of the integration sub-system itself. This simulates what is done in a professional environment, and ensures the quality of the delivered software.

During this semester, objectives for the project are grouped into two categories: Development Infrastructure and Integration Sub-System.

3.1 Development Infrastructure

The objectives for the Development Infrastructure are the following:

- Configure Continuous Integration pipeline providing an automated way of building, packaging and testing, while guaranteeing confidence in implemented code;
- Configure Continuous Deployment pipeline, making the latest version of the project available;
- Configure log analysis platform;
- Adopt a branching model best suited for the project's characteristics;
- Incorporate industry standards in code-review;

- Configure databases to store batch job status information;
- Adopt a hosting solution that is best suited for the project's characteristics, while maintaining costs down.

3.2 Integration Sub-System

The Integration sub-system deals with importing information from external systems, as well as exporting it from such systems to I-On. Some of the problems solved during this first iteration are related to the information I-On provides being public but not owned by the project's teams.

We will start with the timetable and faculty information present in the ISEL's timetable pdf. Given the fact that the source document does not change very often, the application needs to be scheduled to run at a time when there are available resources. Maybe in the start of the semester it will run a couple of times a week and after some weeks it will run at a lower rate.

Also, the consumption of the information does not need to be synchronized with its production, as the document is statically provided and it is expected that at a given time, the current timetable is published online.

For these two reasons, batch processing is a good candidate for the computation model followed by the integration sub-system.

A batch process is defined as one that does not need user interaction to complete. Minella and Syer [29] identify six challenges present in every batch application: usability, maintainability, extensibility, scalability, availability and security. They emphasize that usability in a batch process is not concerned with the user interface, but with error handling and code maintainability. Early on in the development process, it was realized that a standard way of handling errors would have to be defined.

As other teams will work in this project in the future, the components developed for use in batch processes need to be reusable and easily maintainable. Also, adding new features should be easy.

In order to spend as little time as possible debugging or reviewing the logs, one needs to have immediate knowledge of the effect a change made in the code can have in the overall system. That is achieved through having tests covering as many cases as possible.

The batch jobs have to be designed with scale in mind. The results have to be communicated in a timely manner. As of today, each batch job processes one document per course in ISEL. But as the system will become available to other academic institutions, if it processes hundreds of items, reliability i.e. what to do when there is a failure in the processing of an item, making sure the job output is unfailingly communicated and in due time are very tightly coupled and both very important.

The batch jobs that make up the integration sub-system must be configurable. When the system is implanted in a school, jobs must be defined by a technical person or team. These batch jobs need to be configurable via a document, which format will be addressed in the architecture section.

The objectives for the Integration Sub-System in this semester are the following:

- Provide configurable batch jobs to obtain information from external sources and upload it to I-On Core sub-project via HTTP API.
- Enable batch jobs to run periodically or to be triggered by events;
- Create solid and well-documented project following industry best practices, enabling future improvements;

4 Architecture

In the last part we detailed the main problem I-On aims to tackle and specifically the role of I-On Integration sub-project in that task. In this part 4 we'll discuss the architecture design for the module, mainly how it's structured, how it behaves and technologies used.

As discussed before I-On Integration has two main sets of objectives:

- *Development infrastructure and workflow*: This section will focus on the contribution workflow and how it can help maintain code quality. It will also focus on the infrastructure that enables *continuous integration* and *continuous deployment* [31] with *automated testing*.
- *Integration subsystem*: In this section we'll answer how the system is designed according to the identified use-cases.

4.1 Development Infrastructure and Workflow

4.1.1 Version Control

The use of a version control system is a requirement when developing a software project.

It has many benefits, among them [27]:

- Complete log of changes made to every file. This allows to understand the evolution of the file and if needed revert to a previous version.
- The ability to annotate each change with a message that highlights the decisions made enabling a better understanding of the evolution of the project.

For this project, *Git* is used. *Git* [6] is an open source distributed version control system that allows to have multiple local branches that can be entirely independent of each other. That way each team member is allowed to work on its own and when finished merge back to the main branch.

As for code repository *GitHub*[4] was chosen. Not only does it use *Git*, but it also provides many useful features such as code reviews, project management, bug tracking and others [5]. Another solution studied was *Bitbucket*[6] but *GitHub* was ultimately chosen given that its free tier plan offers more options.

4.1.2 Branching Strategy

As mentioned before, one of the main advantages of using *Git* as a version control system is the ability of creating multiple branches. With this ability comes the necessity of defining a strategy of how to best organize the branches.

There are many branching strategies, some more simple, others more complex. For this project we'll use *GitHub flow*[7] as it is quite simple to follow and implement.

[4] `https://github.com/`
[5] `https://github.com/features`
[6] `https://bitbucket.org/product/`
[7] `https://guides.github.com/introduction/flow/`

This simple strategy states that when working on any feature or bug fix, a branch should be created [30], as shown in Figure 2. When it is finished, the work should be merged back to *master branch* via a *pull-request*. This allows all team members to review the code, pointing out errors or improvements and only after the approval of *code owners*[8] can the work be merged.

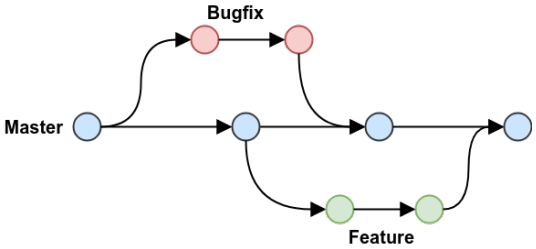

Figure 2: GitHub flow [8]

With this we can guarantee that the work present in *master branch* is always deployable and that it represents the latest iteration of the product. But there is also the need to mark specific iterations, that can be released. For this we'll use *Tags*.

A *Tag* is like a branch that doesn't change [7], basically a snapshot of the code at a given time. This means that it can be used as a release of the product, i.e. *v1.0.0*.

The branching strategy is also extremely important when trying to automate the process of building and deploying the code. This will be further discussed in *Continuous Integration* section.

4.1.3 Linter

Linter is a tool that analyzes source code to flag programming errors, bugs, stylistic errors, and suspicious constructs [28].

The use of such tools is important to reduce errors and improve the quality of the code. It also becomes extremely important when working as a team since it helps to maintain code legible, readable and adhere to coding standards.

Given that *Kotlin* is the main development language used in the project, *ktlint*[9] is used. *Ktlint* is an open-source *kotlin linter*. It provides an *CLI*[10] and also integration with *Maven*[11] and *Gradle*[12].

4.1.4 Unit Tests

Unit tests ensure that the software developed meets the desired behavior. Each unit of code should be tested independently and in isolation.

[8] *GitHub* allows to choose what team members are responsible for code in a repository via the creation of a CODEOWNERS file [1]
[9] https://github.com/pinterest/ktlint
[10] **C**ommand **L**ine **I**nterface
[11] https://maven.apache.org/
[12] https://gradle.org/

To better isolate the tests *mock objects* can be used. With each test focused on a unit of code it's possible to confirm that all individual parts work as expected.

JUnit5[13] is used as the unit testing framework. *JUnit* has been important in the development of test-driven development, and is one of a family of unit testing frameworks collectively known as *xUnit*, that originated with *SUnit* [15].

Kotlin provides a library, *kotlin.test*, that has annotations to mark test functions and a set of utility functions for performing assertions in tests [17]. It also provides as implementation of the *Asserter* class on top of *JUnit5*.

4.1.5 Containers

Traditionally when releasing a new version of a product, the code was built and transferred to the destination machine. This machine would need to have installed all the dependencies necessary to run the project. This process is bug and error-prone.

In order to eliminate this issue, a recent trend, involves packaging up software code and all its dependencies in a container [3]. This allows for the project to run uniformly and consistently on any infrastructure.

For this reason we decide to package the project code in a container, specifically in a *Docker*[14] container.

4.1.6 Continuous Integration

Continuous Integration is a development practice that requires developers to integrate code into a shared repository several times a day. Each check-in is then verified by an automated build, allowing teams to detect problems early [4]. This allows to quickly check errors and maintain the master branch 'clean'. If the build is successful at the end a deployable package is available for testing or releasing.

The *Continuous Integration* pipeline should:

- Build the code;
- Run *Linter*;
- Run *Unit tests*;

This pipeline can be achieve by the use of tools such as *Jenkins*[15] or *Team City*[16] or by the use of services such as *GitHub Actions*[17] or *Travis CI*[18].

The final choice was to use *GitHub Actions* since it's already integrated on *GitHub* which we are using to host the code and it's also contemplated on the free tier plan. *Travis CI* was also a good solution, but it doesn't offer a free tier plan. The use of tools such as *Jenkins* would require a server to host the service and that would amount to additional costs.

[13] https://junit.org/junit5/
[14] https://www.docker.com/
[15] https://www.jenkins.io/
[16] https://www.jetbrains.com/teamcity/
[17] https://github.com/features/actions
[18] https://travis-ci.org/

As we've discussed previously on *Branching Strategy* section, there are 3 major events to take notice: When a *pull-request* to *master* is made, when code is merged to *master* and when a new *tag* is created. These 3 events will trigger the *Continuous Integration* pipeline. We could consider also the event when a new branch is pushed to *GitHub*, but we are limited by free tier plan in order to keep the costs down.

Figure 3 shows the pipeline when *pull-request* to *master* is made. It only has one step which is to build the *Docker* image. This step includes: building the source code, running linter and unit tests.

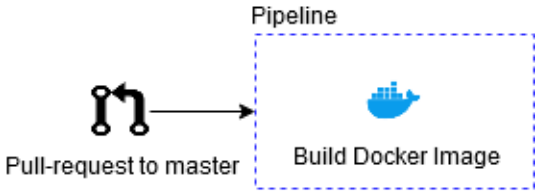

Figure 3: Pipeline when creating a *pull-request* to *master* branch

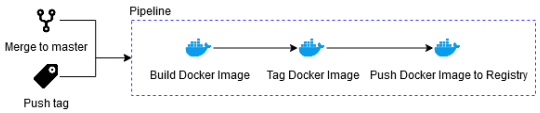

Figure 4: Pipeline when merging to *master* branch or pushing a new *tag*

When code is merged to *master* or when a new *tag* is created, the pipeline is as shown in Figure 4. Additionally to building the *Docker* image, the image is tagged and pushed to *GitHub Packages*[19]. There is a slight difference in the resulting *Docker* image tag between merging to master and pushing a new tag. In the first case, as explained before in *Branching Strategy* section, the merge represents the latest version of the project, in the second the new tag signifies a new release, i.e v1.0.0 and the *Docker* image tag should reflect that to enable the pull of a specific *Docker* image.

4.1.7 Continuous Deployment

I-On Integration is packaged in a Docker container as we saw, and deployed to an instance of Google Compute Engine. Google Compute Engine [13] is the un-managed Virtual Machine option on Google Cloud Services.

We wanted the deployment to be as independent as possible from the cloud provider chosen, however we needed to settle for one. Apart from the free-tier other providers have, Google Cloud Services grants 300 in money credit [10] [20] to be used in the first 12 months. This credit can be used as a safety net if the free tier limits are exceeded.

[19] Container registry where *Docker* images can be pushed and pulled to create containers
[20] Azure provides 170 dollars credit for the free-tier

As our team is small, we didn't specifically want control of administrative tasks on the deployment environment, like updating the operating system, as it would increase admin overhead time. The managed alternatives in the GCS stack were Cloud Run [11], which is a managed serverless alternative, and Google Kubernetes Engine [9], which is a managed Kubernetes cluster option.

Cloud Run could not be used to run scheduled batch jobs in a standard way, as it has a 15-minute inactivity timeout [21]. After the referred period, the instance would be killed.

It would be possible to run scheduled jobs with this limitation however, adding an external scheduler that would trigger the container into activity. Nevertheless, we would not be using what Spring Batch already provides [21]. Furthermore, it would increase the unnecessary administrative burden of having to maintain the scheduler service independently.

We didn't go for GKE as we didn't need the auto-healing, auto-scaling and fault-tolerance capabilities Kubernetes provides. It seemed to us that the project didn't need container orchestration for now. Had we used GKE, we would just need a single running pod. In order to avoid over-provisioning, we didn't choose it.

As Google Compute Engine was chosen, there will be the need to absorb occasional environment administration time. The machine where I-On Integration is deployed to needs to have Docker Engine installed.

In order to make sure the latest version of the software is available for test and use, deployment to the staging and production environments is included in the project's pipelines.

Deployment to the staging environment (figure 5) is triggered by a push event to the repository's master branch. It is done after the necessary steps for continuous integration are completed, such as building the project and its Docker image, tagging and pushing it to the GitHub registry.

Similarly to what is done for the major steps of continuous delivery, this is achieved using a gradle task instead of the Cloud SDK command line [12] directly on the pipeline. This way we do need to rely on the pipeline if we want to test deployment aspects.

[21] "The value you specify must be less than 15 minutes for fully managed Cloud Run", on "https://cloud.google.com/run/docs/configuring/request-timeout", accessed: 2020-05-01

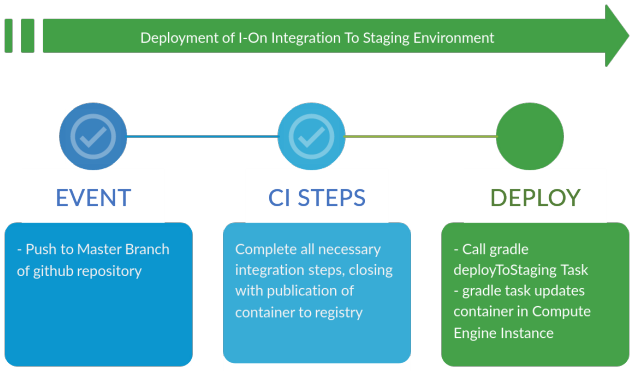

Figure 5: Deployment to staging environment

As for deployment in production environment (figure 6), it's triggered when a tag starting with "v" (for "version") is pushed to the repository. The CI steps and the dedicated gradle task are run. The deployed version will be designated by the version number present in the tag.

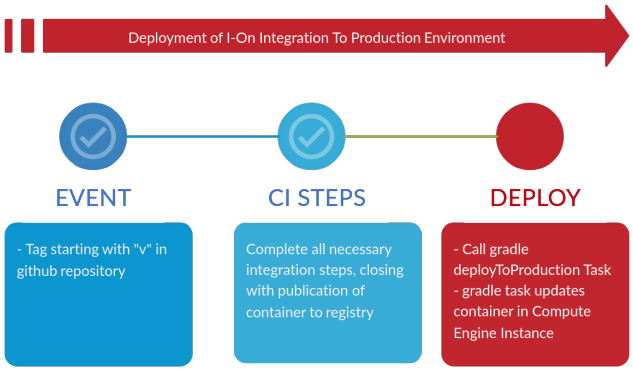

Figure 6: Deployment to production environment

4.2 Integration Subsystem

All I-On sub-projects will be implemented using one or more *Spring Projects* [23]. I-On Integration will use *Spring Batch* [20] in which multiple jobs can be configured and scheduled.

As is the case with the remaining I-On project modules, I-On Integration uses *Kotlin* as the main development language. *Kotlin* is a statically-typed programming language built by *JetBrains* that compiles to bytecodes. It was adopted by *Google* in May 2017 as an official *Android* development language [16], but it has also been used outside the mobile ecosystem. In 2016 [19], Spring added support for the *Kotlin* language in *Spring* Initializr.

For the development of the batch jobs we use *Spring Batch*. It is a mature framework,

having its 1.0.0 version release in 2008. It was developed in connection with the industry and provides batch functionalities out-of-the-box, such as retry and skip policies when an operation is unsuccessful, a wide range of built-in classes for I/O operations and job status persistence, which is important in a robust, enterprise-grade application.

Spring Batch abstracts the common building blocks of a batch application. Its layered architecture favors code reuse. It has three layers: application, core and infrastructure.

Figure 7 shows these three layers and its relation. The application layer consists of custom code and configurations used to build new batch processes. Our intervention will be at the application level. The core layer contains the interfaces that define the batch domain (e.g. *JobLauncher*, *Job*, *Step*). The infrastructure layer handles reading and writing to files and databases, in addition to what to do when a job is retried after failure.

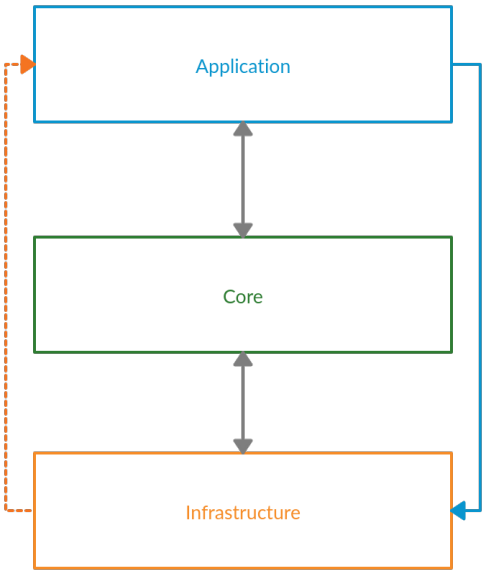

Figure 7: The Layered Spring Batch Architecture

The *Spring Batch* Documentation has a section on domain-specific language [22]. Most important to the present context are the concepts of *Job*, *Step*, *ItemReader*, *ItemProcessor* and *ItemWriter*.

Following is a summary of the concepts that are relevant in order to understand the organization of the batch jobs:

- *Job* - A process that executes from start to finish without interruption or interaction, consisting of one or more steps. It can have associated retry logic;
- *Step* - Independent phase of a batch job;
- *Chunk* - Fixed amount of items;
- *ItemReader* - Abstraction that represents Step input, per item;
- *ItemProcessor* - Abstraction that represents Step processing logic, associated to the domain of the application;

- *ItemWriter* - Abstraction that represents Step output, per item, or per chunk.

A *JobScheduler* runs a job launcher that uses the information retrieved from *JobRepository* to execute a *Job*. Both *Job* and *Step* have an execution context which stores its state, enabling its progression and re-execution after failure.

A *Step* can be defined in terms of a *Tasklet* or *ChunkTasklet*. A *ChunkTasklet* reads and processes a chunk and writes it once. Then, for a *ChunkTasklet*, an *ItemReader* and *ItemWriter* have to be specified. Optionally, an *ItemProcessor* can be defined, to do some intermediary data transformation. A *Tasklet* is a more flexible piece with no necessity of configuring readers and writers.

Steps can be chained to run in a determined order. A *Step* can can be composed of more than one *Tasklet* or *ChunkTasklet* that run in parallel, making visible data independence within the job and making the most out of present-day hardware.

Apart from using *JUnit* for unit testing, I-On Integration also uses the package spring-batch-test to test the job components: steps, tasklets, readers, processors, writers, as it provides domain-specific methods and classes that fit very well the *Spring Batch* domain language. It allows simulation of *Spring* beans that have the *Step* and *Job* scope.

Many of the source documents, such as the timetable and exam schedule are in the pdf format. For parsing them we use two libraries: *Tabula* [24] and *iText* [14]. The first is used to extract information from tables and the second for non-tabular information. These libraries were chosen over alternatives like *Apache PDFBox*, as they are easy to configure and more performant by default. For non-tabular information, comparing *iText* and *PDFBox* in terms of performance favors textitiText. *PDFBox* processes text glyph by glyph, whereas textitiText processes it chunk by chunk, being less I/O-resource intensive [18].

Tabula receives a path to a file, which makes it necessary to have the file stored in the local file-system. *iText* reads pdf per page, whereas *Tabula* reads all pages of a document in bulk.

The goal of each job is to acquire the needed information, transform it and send it to the repository - I-On Core. One of the common phases of each job is uploading to I-On Core via its http API. Then, apart from the retry functionalities embedded in *Spring Batch*, we need to account for communication failures, unavailability of service, etc.

These failures in the uploading phase, most probably will occur at a stage where we don't want to retry writing all the items of the chunk that were already sent. Then, another kind of retry mechanism is in place to ensure that information is written exactly once. That will be addressed for each batch job.

For the non-spring-batch-supported retry capability, the followed policy is to retry the operation a number of times. That amount is configurable via the custom configuration file. After upload retries have reached the limit, the associated job will fail, alerting someone in charge.

4.2.1 ISEL Timetable Extraction Batch Job

The following diagram (figure 8) shows the sequence of actions necessary to complete the timetable extraction process. It is meant as a first approach to what are possible states within the process and their sequence. There is no direct correspondence between each state of the

finite state machine and the steps within the job.

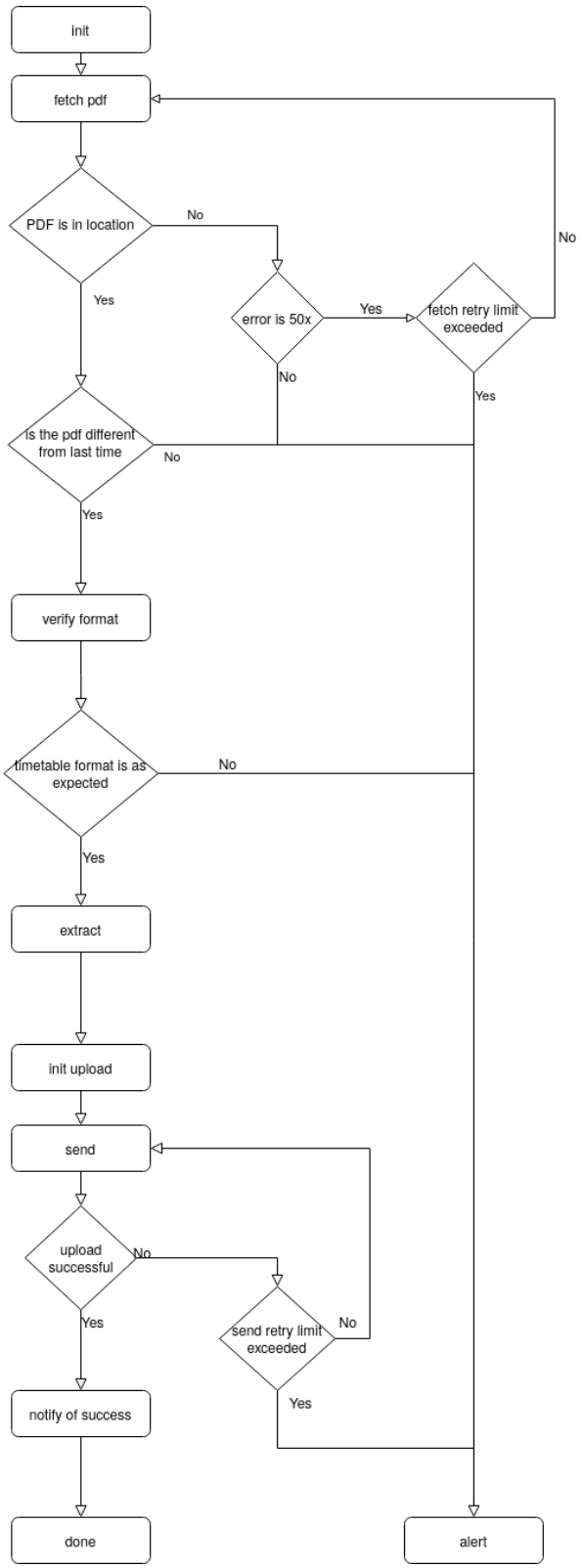

Figure 8: Representation of the ISEL timetable extraction as a Finite State Machine

The following diagram (figure 9) presents the sequence of steps included in the ISEL timetable extraction job:

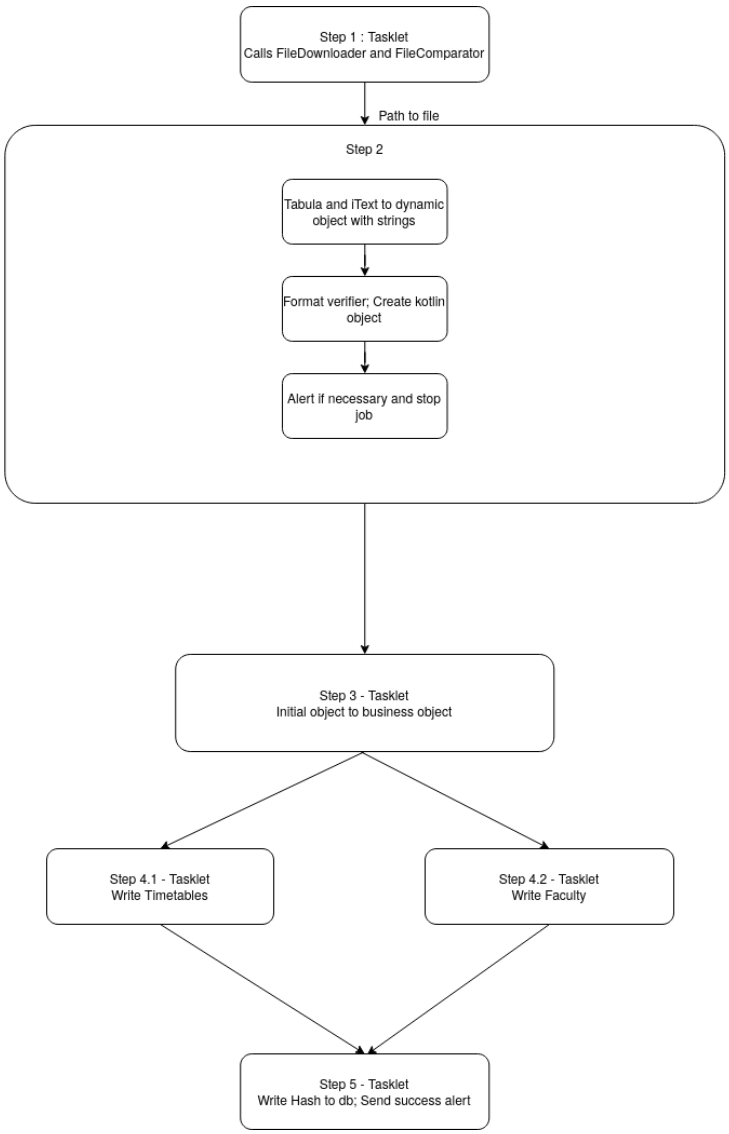

Figure 9: Steps included in the ISEL timetable extraction job

Step 1 makes sure that that the timetable document is reachable in the URL specified in the configuration document (more information on the document is given in the Configuration Document section of this chapter), and that it wasn't parsed in the past.

The extraction process strongly depends on the information being in the expected format. So, the second step compares the actual format to the expected.

Next, step 3 maps the verified object to a business object, more suited to the upload.

The fourth step is comprised of two tasklets that are executed in parallel, which upload timetable and faculty information to I-On Core.

The final step updates the database with the hash of the parsed file and notifies watchers of success.

A more detailed description of what is done in each step is provided in the following paragraphs. Accompanying it is an explanation for the design decisions taken.

4.2.1.1 Step 1 - Downloading and Comparing

Given that the timetable extraction job is dependent on information that is not controlled by the I-On Integration project, namely the ISEL timetable PDF, we need to make sure the document is in the designated location. Also, we don't want to send information that is already present in I-On Core.

This step downloads the pdf document, (most likely from ISEL's website, but the location is configurable through the configuration document) and writes it to the local file-system.

It calculates a hash of the file. Then it reads from a database the value of the hash of the document used in the last time the job ran successfully. Assuming a non-broken cryptographic hash and no collisions, if the two hashes are the same, then we know that the document content is certainly the same. In that case, the job does not proceed, as the extracted information is already present in I-On Core.

On the other hand, if the saved hash is different from the one just calculated, then the file contents have changed since the last run of the job.

Because the integration project doesn't save state, this is a fairly inexpensive way of avoiding processing repeated information. It is suited to this document in particular, since it is not expected that it should change frequently. It is normally posted in the start of the semester. There may be an isolated change in the following weeks, but then the document persists until the end of the semester.

4.2.1.2 Step 2 - Verifying format

The second step is configured with a reader, processor and writer. The chunk size is equal to one, as an input item contains information for all the document. The input item is a dynamic object with two string arrays, one for the output of *Tabula* and one for non-tabular information, obtained with *iText*. The arrays contain as many elements as the number of pages in the pdf.

The reader iterates through the pages of the document, enriching the dynamic object. The processor verifies the format. It does not need to verify more than one page. It also maps the dynamic object to a *Kotlin* object with a format more suited to the extraction itself. Then this object will be published to a state instance that is declared in the configuration file and injected in the processor of step 2 and the *Tasklet* of step 3.

Communicating state across steps in a job can be done via the *JobExecutionContext* , but this has limited capability, deeming it not suitable to this use case. Maintaining state only in memory is sufficient as we don't need the data available outside the scope of a job execution. Also, if we had a database for this we would have to deal with the overhead of reading from and writing to it. Having chunk size equal to 1 enables us to atomically update the inter-step state

object.

The writer of the present step is a no-op if the format of the source document is correct. But if the format is not, then all configured alerts are generated and the job is stopped.

4.2.1.3 Step 3 - Mapping

From the data generated in step 2, step 3 builds business objects that have type compatibility with what will be sent to I-On Core. Then it makes this information visible to a reference that is shared with step 4. It was declared in the configuration class and injected to both this *Tasklet* and the ones in step 4.

4.2.1.4 Step 4 - Uploading to I-On Core

This step is comprised of two tasklets that can run in parallel. One of them uploads timetable information, one class-section set at a time to I-On core, and the other uploads faculty with the same granularity. The conceptual model of the uploaded information is defined in a sub-section of this chapter.

The integration module holds no state across multiple executions of the same job, apart from the file hash mentioned in step 1. If data does not reach the I-On Core system for some reason, it needs to be calculated again. With this in mind, if some HTTP request fails, it is retried a number of times. This value can be configured via the configuration file. This prevents any additional fault in data integrity in I-On Core.

If by chance the I-On Core system is down at the time of upload, the job fails after retrying a specified number of times.

4.2.1.5 Step 5 - PostUpload

As the job approaches termination, after the information was successfully extracted and uploaded, we register in a database the hash of the successfully parsed document. This is done in order to prevent wasting time, memory and processing capabilities by running the job again with the same timetable document as input.

Finally, someone in charge of job supervision will be notified of success. As is the case with failure notification, the configuration file specifies who to notify in this case.

4.2.1.6 Retry and Skipping Capabilities

To be able to use the retry mechanism embedded in Spring Batch, all that needs to be done is configuring a step using *StepBuilderFactory*, call the faultTolerant method of *SimpleStepBuilder* and specify the retry limit and the Exceptions upon which retry is attempted.

Spring Batch also enables programmers to specify what is done if one step cannot be completed after the specified number of retries. As the successful execution of each step is mandatory for good conclusion the timetable extraction job, it does not make use of any skip policy.

4.2.1.7 Configuration Document

This document is stil in progress. In the next delivery this section will be completed

4.2.1.8 Conceptual model of information to be uploaded

From the timetable document there is a wide variety of information that can be extracted about school activities and organization, including course offer and the teaching staff of the course.

Information is divided according to its source, as it is found on the page header or on the tables in the center and footer of the page. This categorization is relevant in the context of the batch job, because the library used to parse the section of the page that contains tabular data (*Tabula*) is not the same that is used to parse the headers (*iText*). These two segments of the job are to be handled by separate utilities.

Following is the information that can be extracted from the timetable document grouped in two clusters of independent data, timetable and faculty, in json format.

Timetable data:

```
{
    "school": "INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA",
    "programme": "Licenciatura Engenharia Informática e Computadores",
    "term": "2019/20-Verão",
    "class": "LI11D",
    "courses": [
        {
            "course": "ALGA[I]",
            "course_type": "(T)",
            "room": "E.1.31",
            "begin_time": "14:00:00",
            "end_time": "15:30:00",
            "duration": "1:30:00",
            "weekday": "Monday"
        }
    ]
}
```

Faculty data:

```
{
    "school": "INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA",
    "programme": "Licenciatura Engenharia Informática e Computadores",
    "term": "2019/20-Verão",
    "class": "LI11D",
    "faculty": [
        {
            "course": "ALGA",
            "course_type": "(T)",
            "teachers": [
                {
```

```
                    "name": "João Trindade"
                }
            ]
        }
    ]
}
```

It is evident why it is needed to send the term and class when sending timetable and faculty. As I-On can be implanted in more than one school, term and class collisions may occur. In that case, if the extraction is occurring at roughly the same time, the core module has no way of pinning a course to a programme. Furthermore, different schools can have a programme with the same name. In that case, the name of the school is necessary to disambiguate.

5 Implementation

This document is stil in progress. In the next delivery this section will be completed

6 Limitations

This document is stil in progress. In the next delivery this section will be completed

7 Further Improvements

This document is stil in progress. In the next delivery this section will be completed

8 Reformulation of Project's Scope

The integration sub-project took the responsibility of spearheading the research on infrastructure for all the I-On modules. This may force a rescheduling of tasks, as the initial plan did not forecast this.

Documenting our decisions in a way that the other teams can benefit from them consumes extra time. But rescheduling isn't the only impact this change has for our project. It also modifies the project's scope.

9 Conclusion

The main factor that is impacting the progress of the project at the moment are delays. There were delays in deciding which cloud platform to use, in defining the ISEL timetable batch job architecture and in defining communication with the Core module.

Studying cloud alternatives and reading documentation took us more time than expected. Initially we wanted to deploy the project to a managed environment. The criteria was to cut time spent managing the deployment machines. We thought Cloud Run was a good fit, as it is fully managed. Nevertheless, in the following week we realized scheduling jobs using Spring Batch wouldn't be possible, as was discussed in the Continuous Deployment Section of the Architecture chapter. Additional time was spent looking for an alternative.

Defining the information model for uploading to I-On core also went off the project timeline. That, in turn, suspended progress in the batch job definition, as we couldn't specify how upload would be done. The batch job definition also required us to investigate Spring Batch in more depth, which was not factored in at the time of planning.

Using pull requests has the advantages pointed out in the architecture chapter. However, it also means that integration of new code takes longer, as it has to be reviewed by the repository code owners.

As a result, our initial plan is slightly delayed, but the more substantial architecture and definition decisions are taken. The ensuing tasks have a more implementation-related component, so it is expected that the progress pace will increase moderately.

References

[1] About code owners. https://help.github.com/en/github/creating-cloning-and-archiving-repositories/about-code-owners. Accessed: 2020-05-01.

[2] About pull requests. https://help.github.com/en/github/collaborating-with-issues-and-pull-requests/about-pull-requests. Accessed: 2020-04-28.

[3] Containerization. https://www.ibm.com/cloud/learn/containerization. Accessed: 2020-04-29.

[4] Continuous integration. https://www.thoughtworks.com/continuous-integration. Accessed: 2020-04-29.

[5] Fénix global project description. https://ciist.ist.utl.pt/projectos/Fenix.pdf pp. 14,15. Accessed: 2020-04-29.

[6] Git. https://git-scm.com/. Accessed: 2020-04-30.

[7] Git tag. https://www.atlassian.com/git/tutorials/inspecting-a-repository/git-tag. Accessed: 2020-05-01.

[8] Github flow. http://files.programster.org/tutorials/git/flows/github-flow.png. Accessed: 2020-04-29.

[9] Google cloud kubernetes engine product page. https://cloud.google.com/kubernetes-engine. Accessed: 2020-04-01.

[10] Google cloud platform free tier conditions. https://cloud.google.com/free. Accessed: 2020-04-01.

[11] Google cloud run product page. https://cloud.google.com/run. Accessed: 2020-04-01.

[12] Google cloud sdk command line for interaction with compute engine. https://cloud.google.com/sdk/gcloud/reference/compute/instances. Accessed: 2020-04-01.

[13] Google compute instances product page. https://cloud.google.com/compute. Accessed: 2020-04-01.

[14] itext official webpage. https://itextpdf.com/en. Accessed: 2020-04-29.

[15] Junit 5. https://junit.org/junit5/. Accessed: 2020-05-01.

[16] Kotlin is announced as official supported language for android. https://blog.jetbrains.com/kotlin/2017/05/kotlin-on-android-now-official. Accessed: 2020-04-29.

[17] kotlin.test. https://kotlinlang.org/api/latest/kotlin.test/. Accessed: 2020-05-01.

[18] Performance itext vs.pdfbox. https://stackoverflow.com/questions/22340674/performance-itext-vs-pdfbox. Accessed: 2020-04-30.

[19] Spring announces kotlin support in spring initializr. https://spring.io/blog/2016/02/15/developing-spring-boot-applications-with-kotlin. Accessed: 2020-04-29.

[20] Spring batch. https://spring.io/projects/spring-batch. Accessed: 2020-04-28.

[21] Spring batch documentation - asynchronous execution of jobs. https://docs.spring.io/spring/docs/current/spring-framework-reference/integration.html#scheduling. Accessed: 2020-04-01.

[22] Spring batch documentation - domain-specific language. https://docs.spring.io/spring-batch/docs/current-SNAPSHOT/reference/html/domain.html. Accessed: 2020-04-30.

[23] Spring projects. https://spring.io/projects. Accessed: 2020-04-28.

[24] Tabula's github repository. https://github.com/tabulapdf/tabula-java. Accessed: 2020-04-29.

[25] Web boom 2.0. http://content.time.com/time/magazine/article/0,9171,1570789,00.html. Accessed: 2020-04-28.

[26] What is batch processing? https://www.ibm.com/support/knowledgecenter/zosbasics/com.ibm.zos.zconcepts/zconc_whatisbatch.htm. Accessed: 2020-04-28.

[27] What is version control. https://www.atlassian.com/git/tutorials/what-is-version-control. Accessed: 2020-04-29.

[28] Richard Bellairs. Why is linting important? and how to use lint tools. https://www.perforce.com/blog/qac/why-linting-important-and-how-use-lint-tools. Accessed: 2020-05-01.

[29] M. Minella and D. Syer. *The Definitive Guide To Spring Batch. 1st ed. Chicago, IL: Apress, p.18.* Apress, 2019.

[30] Lorna Mitchell. Choose the right git branching strategy. https://www.creativebloq.com/web-design/choose-right-git-branching-strategy-121518344. Accessed: 2020-05-01.

[31] Sten Pittet. Continuous integration vs. continuous delivery vs. continuous deployment. http://web.archive.org/web/20080207010024/http://www.808multimedia.com/winnt/kernel.htm. Accessed: 2020-04-28.