

INSTITUTO POLITÉCNICO DE LISBOA
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA



I-On Integration

**Importing academic information from external
systems for aggregation and distribution**

Final Version 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

July, 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

*Final Version Report written for the Curricular
Unit of Project and Seminary*

July, 2020

Acknowledgements

This project signifies that my journey on ISEL is almost ending. It has been an awesome ride where I was able to grow and learn and I'll use this space to thank those that were part of this journey.

First I have to say thanks to Samuel Costa for being my partner on this project. We worked really well together and we were able to achieve a great product. Also want to say thank you to both supervisors, first for inviting me to be part of the project and for all the assisting during the project's development.

I'd also like to thank the colleagues that have been part of this journey from the beginning, André Mendes, Pedro Pereirinha, Ricardo Marramaque and Samuel Costa.

Also want to mention some of the teachers that had great impact on me, helped me in multiple occasions and from whom I've learned valuable skills such as Artur Ferreira, Carlos Martins, Jorge Fonseca, Paula Simões, Pedro Miguens, Pedro Sampaio and Tiago Dias.

Finally I want to thank my girlfriend Sónia for supporting my decision of going to ISEL and encouraging me every day. To my mother and sister thank you for always being there. To IMBAS thanks for being the best friends.

July 2020, Miguel Teixeira

This project would not be possible without the assistance of our supervisors, reviewing our work and guiding us. I would like to thank them for their generous effort in assisting us.

I also want to thank Miguel for the companionship, the professionalism and the confidence with which he inspired me.

Words cannot express the gratitude I feel towards both mine and my partner's family for setting everything in place so I could focus on school during these 4 years. A special word to Sara, my partner, for her support and understanding.

July 2020, Samuel Costa

Contents

Acknowledgements	iii
Table of Contents	v
1 Introduction	1
1.1 Motivation	2
2 Problem Formulation	3
2.1 Academic Information consumption system	4
2.2 Individual batch jobs	5
2.3 Integration and deployment environment	6
3 Architecture	7
3.1 Academic Information consumption system	7
3.1.1 Flexible data models	8
3.1.2 Data consumption	8
3.1.3 Data production	8
3.2 Scheduling and running batch jobs	8
3.3 Individual batch jobs	9
3.3.1 ISEL Timetable Extraction Batch Job	10
3.3.2 Generic Batch job	13
3.3.2.1 Common Building Blocks	13
3.3.2.2 Configuring the generic batch job	13
3.3.2.3 From the blocks to the generic batch job	14
3.3.3 Retry and Skipping Capabilities	15
3.4 Integration and deployment environment	16
3.4.1 Version Control	16
3.4.2 Branching Strategy	16
3.4.3 Linter	17
3.4.4 Unit Tests	17
3.4.5 Containers	18
3.4.6 Continuous Integration	18
3.4.7 Continuous Deployment	19
4 Implementation	20
4.1 Academic Information consumption system	20
4.1.1 Configuring Batch jobs	20
4.1.2 Running batch jobs	21
4.1.3 Scheduling batch jobs	21
4.2 Batch jobs	21
4.2.1 ISEL Timetable Batch Job	21
4.2.1.1 Step 1 - Downloading and Comparing	21
4.2.1.2 Step 2 - Verifying format	22
4.2.1.3 Step 3 - Mapping	22
4.2.1.4 Step 4 - Uploading to I-On Core	25
4.2.1.5 Step 5 - PostUpload	25
4.2.1.6 Configuration Document for the timetable job	25
4.2.1.7 Conceptual model of information to be uploaded	26
4.2.2 Generic Batch job	27

4.2.2.1	Yaml input files	27
4.2.2.2	Transforming the Integration Internal Model to Core Model . . .	29
4.2.2.3	The retry mechanism of GenericCoreWriter	30
4.2.2.4	Adding a new job type to the generic batch job	30
4.2.3	Retry and Skipping Capabilities	30
4.3	Integration and deployment environment	30
4.3.1	Continuous Integration	30
4.3.2	Continuous Deployment	32
5	Limitations and Further Improvements	35
5.1	Academic Information consumption system	35
5.2	Batch jobs	35
5.2.1	ISEL Timetable Batch Job	36
5.2.2	Generic Batch job	36
5.3	Integration and deployment environment	36
6	Conclusion	37
	References	38

List of Figures

1	Overview of the I-On project organization	3
2	The Layered Spring Batch Architecture	8
3	Representation of the ISEL timetable extraction as a Finite State Machine	11
4	Steps included in the ISEL timetable extraction job	12
5	Steps included in the Generic Batch Job	14
6	GitHub flow [1]	17
7	Timetable pdf - detail of column identifier: the left cell limit	24
8	Timetable pdf - detail of cell height	24
9	Timetable pdf - detail of course taught by more than one teacher	25
10	Pipeline when creating a <i>pull-request</i> to <i>master</i> branch	31
11	Pipeline when merging to <i>master</i> branch or pushing a new <i>tag</i>	31
12	Deployment to staging environment	33
13	Deployment to production environment	33

1 Introduction

Academic institutions need to publish information related to their activities to the public such as programmes, curricula, 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. Nowadays the information is 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 the 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 focuses mainly on I-On Integration, however, a brief introduction of the I-On project organization can be found on chapter 2. 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 is then made available to the users via I-On Android.

I-On Integration is a *batch processing* [3] application, i.e. it allows to configure and run *batch jobs* [3]. The main advantage of using *batch jobs* is the possibility of scheduling them and ensuring that whenever new data is published on external sources it is 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 is not limited to, having quality and documented code, using *pull-requests* [4] as workflow, *continuous integration* and *continuous deployment* [5] 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 6 chapters:

- Chapter 2 describes the context of the problem I-On aims to solve in more depth.
- In Chapter 3 we focus on the architecture design of the solution, while also highlighting the technologies that support it.
- Chapter 4 provides a discussion of the implementation details.
- Finally Chapter 5 focuses on the project limitations, how they affect the solution and pointing possible future improvements.

¹Given 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

²Portable Document Format developed by Adobe

We identified three main vocations of the project in the first phase: (1) being an academic information consumption system, (2) an application composed of individual batch jobs that add functionalities and (3) a professional integration and deployment environment. The following chapters reflect these 3 separate concerns by being divided into 3 sub-chapters.

1.1 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 of 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's best practices and the knowledge obtained along the course.

2 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. It must be 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.

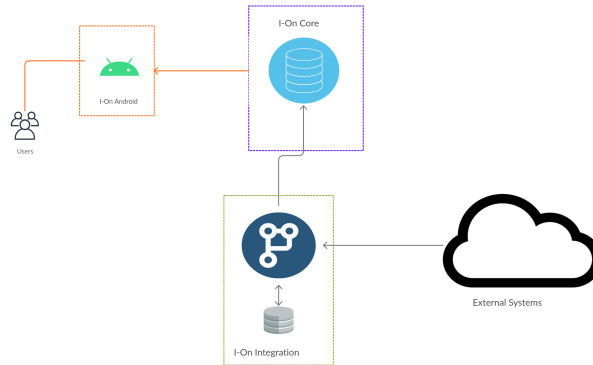


Figure 1: Overview of the I-On project organization

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

³4 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 [6]. 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 simulated what is done in a professional environment, and contributed to the quality of the delivered software.

During this semester, objectives for the project were grouped into three categories: academic information consumption system, individual batch jobs to import information to I-On Core and integration and deployment environment.

2.1 Academic Information consumption 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 were related to the information I-On provides being public but not owned by the project's teams.

In the integration module, we started 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 process of extracting this data needed to be scheduled to run at a time when there are available resources. Maybe in the start of the semester it runs a couple of times a week and after some weeks it runs at a lower rate.

Also, the consumption of the information did not need to be synchronized with its publica-

tion, as the document was statically provided and it is expected that at a given time, the current timetable is published online.

For these two reasons, batch processing was 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 [7] 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.

The objectives for the Integration Sub-System in this semester were 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;

2.2 Individual batch jobs

As other teams will work in this project in the future, the components developed for use in batch processes needed 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, we needed to have immediate knowledge of the effect a change made in the code had in the overall system. That was achieved through having tests covering as many cases as possible.

The batch jobs had to be designed with scale in mind. The results had to be communicated in a timely manner. As of today, each batch job instance processes at most one document per programme in ISEL. But as the system becomes 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 deployed 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 is addressed in the architecture section.

The objectives regarding batch jobs this semester were the following:

- Import timetable, exam schedule and academic calendar information into I-On Core via the creation of separate batch jobs

2.3 Integration and deployment environment

The objectives for the Development Infrastructure were 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 Architecture

In the last chapter 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 chapter 3 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 three main sets of objectives:

- *Academic Information consumption system*: This section answers how the system is designed according to the identified use-cases, how batch jobs are scheduled and executed.
- *Individual Batch jobs*: This section goes over the functionalities added by individual batch jobs that were developed to import information, shedding light on some design choices.
- *Integration and deployment environment*: This section focuses on the contribution workflow and how it helps maintain code quality. It also focuses on the infrastructure that enables *continuous integration* and *continuous deployment* [5] with *automated testing*.

3.1 Academic Information consumption system

All I-On sub-projects were implemented using one or more *Spring Projects* [8]. I-On Integration used *Spring Batch* [9] in which multiple jobs could be configured and scheduled.

As is the case with the remaining I-On project modules, I-On Integration used *Kotlin* as the main development language. *Kotlin* is a statically-typed programming language built by *JetBrains* that compiles to *JVM bytecode*⁴. It was adopted by *Google* in May 2017 as an official *Android* development language [10], but it has also been used outside the mobile ecosystem. In 2016 [11], Spring added support for the *Kotlin* language in *Spring Initializr*.

For the development of the batch jobs we used *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 2 shows these three layers and its relation. The application layer consists of custom code and configurations used to build new batch processes. Our intervention was 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.

⁴Instruction set of the **J**ava **V**irtual **M**achine

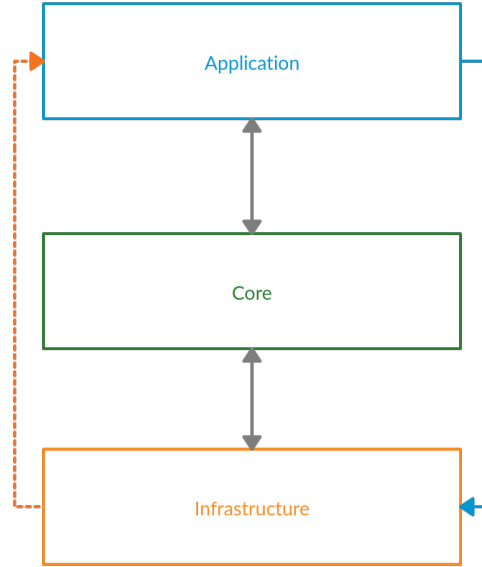


Figure 2: The Layered Spring Batch Architecture

The communication with I-On Core is made via an *Write API*. This *API*⁵ was designed in conjunction with I-On Integration with the following goals:

3.1.1 Flexible data models

All data models should be flexible enough to accommodate data pertaining to any academic institution, e.g. academic calendar from an University or from an High School.

3.1.2 Data consumption

I-On core has to easily consume and process data sent from I-On Integration, regardless of the source, but still be able to identify which academic institution it belongs to.

3.1.3 Data production

At the same time I-On Integration should be able to transform and build the data models with ease, considering that all the necessary data has to be publicly available.

Another key issue pertains to the removal of data. The decision was that I-On Integration would always publish the latest data, being the responsibility of I-On Core to compare and remove outdated information.

3.2 Scheduling and running batch jobs

Sprint Batch offers the possibility of scheduling jobs to run at fixed rate, fixed delay and according to a cron pattern. The integration module uses cron patterns to schedule jobs to run when there is little probability the source files will change. At this time, the batch jobs run on saturday at 1,2 and 3 AM given that it should be a period of reduced traffic on I-On Core.

⁵Application Programming Interface between softwares

3.3 Individual batch jobs

The *Spring Batch* Documentation has a section on domain-specific language [12]. 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 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 were in the pdf format. For parsing them we used two libraries: *Tabula* [13] and *iText* [14]. The first was used to extract information from tables and the second for non-tabular information. These libraries were chosen over alternatives like *Apache PDFBox*, as they were easy to configure and more performant by default. For non-tabular information, comparing *iText* and *PDFBox* in terms of performance favored *iText*. *PDFBox* processes text glyph by glyph, whereas *iText* processes it chunk by chunk, being less I/O-resource intensive [15].

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, occur at a stage when we don't want to retry writing all the chunk's items that were already sent. Then, another kind of retry mechanism would need to be in place to ensure that information is written exactly once. That is 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 `application.properties` file, as it is general configuration, not depending on the batch job. After upload retries have reached the limit, the associated job will fail, alerting someone in charge.

3.3.1 ISEL Timetable Extraction Batch Job

The following diagram (figure 3) 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.

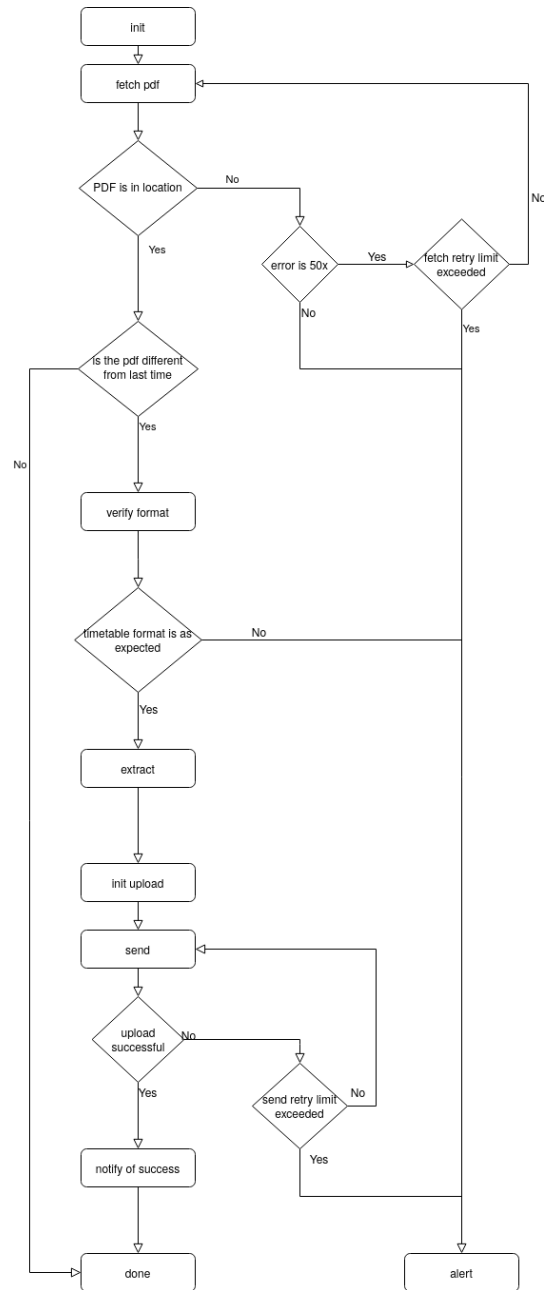


Figure 3: Representation of the ISEL timetable extraction as a Finite State Machine

The following diagram (figure 4) presents the sequence of steps included in the ISEL timetable extraction job:

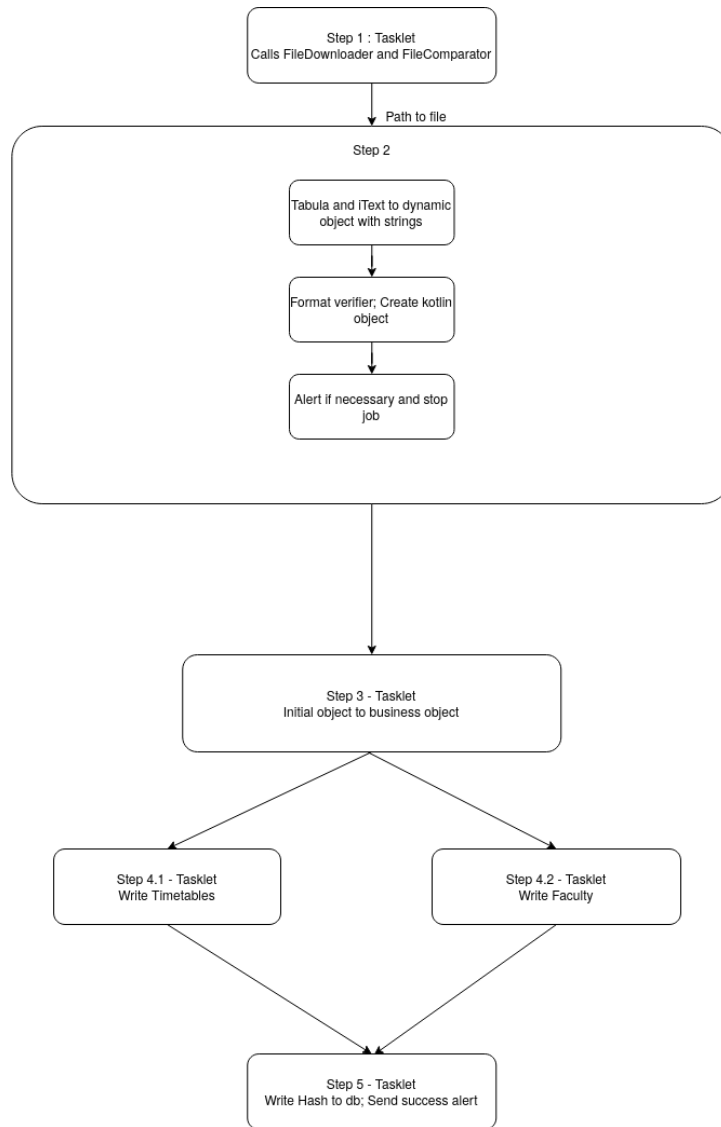


Figure 4: Steps included in the ISEL timetable extraction job

Step 1 makes sure 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 depended 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 next chapter. Accompanying it is an explanation for the design decisions taken.

3.3.2 Generic Batch job

During the development stage of the first phase of I-On Integration, it was acknowledged that importing information from external systems into I-On had some common elements, regardless of what sort of data was at hand.

The team realized that having a custom logic, made from scratch for each batch job would lead to a hard to maintain solution, which would also be overly complex and too cumbersome to extend.

Apart from minimizing maintenance overhead and amount of code written, we also wanted that people not associated with the project could be producers of information. This is primarily a practical requirement, as the project does not have a back-office. We went over the available formats, having picked out yaml, as it is human-readable.

From these realizations came the idea of developing a generic batch job, that would use the same code base to import information regarding academic calendar, exam schedule and timetable. Apart from these categories, the generic job should be easily expanded to support importing new information from a yaml file.

The generic batch job should also be configurable, as the isel timetable batch job, via a configuration document.

3.3.2.1 Common Building Blocks

The common building blocks of most importing operations that we identified include:

- Downloading a file from an external source;
- Mapping the file to a kotlin object in memory
- Transforming this tentative model to a format that I-On Core can receive
- Sending the payload to I-On Core in a format previously agreed upon

3.3.2.2 Configuring the generic batch job

The generic batch job is configurable through a file with the extension `.properties`. The following properties are accepted:

Property	Description
<code>srcRemoteLocation</code>	Url of the yaml file to be used on the job.
<code>alertRecipient</code>	Email of the point-of-contact to notify about job outcome
<code>jobType</code>	Category of information to be exported: EXAM_SCHEDULE, ACADEMIC_CALENDAR or TIMETABLE (Currently not supported)

The application starts an instance of the generic batch job for each file in the subfolders of `src/main/resources/config/generic`. The parameters passed for each job instance are the properties contained in a file of that directory.

3.3.2.3 From the blocks to the generic batch job

As the common stages were already present in the first batch job that was developed, the ISEL timetable batch job, some of the code was reused, which sped the development process.

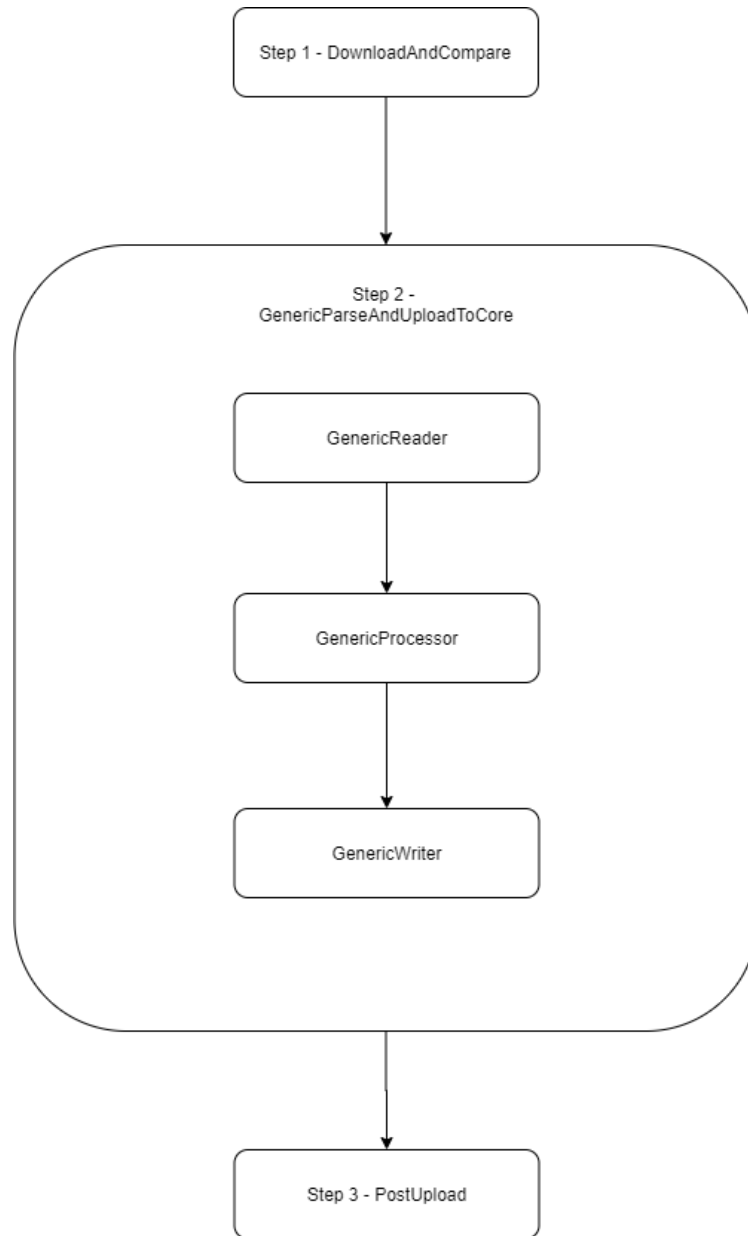


Figure 5: Steps included in the Generic Batch Job

Examples of code reuse are the `DownloadAndCompareTasklet` and the `PostUploadTasklet`. The first downloads a file, verifies that its format complies with the configured rules, and before saving its contents to the local file-system, compares its hash to that of the file the job success-

fully executed last time with. The `PostUploadTasklet` records a hash saved to the execution context and sends a success e-mail to a person responsible for overseeing the importing process.

The abstract building blocks of the generic batch job were materialized in the sequence of steps outlined in the diagram in figure 5.

In step 1, the `DownloadAndCompareTasklet` is used, as was stated. Step 2 has reader, processor and writer which were called `Generic`, because they are the same for each `jobType`, although with different behavior. Supported job types are `ACADEMIC_CALENDAR`, `EXAM_SCHEDULE`. The `TIMETABLE` is defined, but a yaml file was not defined and the transformation logic was not implemented.

According to `jobType`, a `Factory` class is obtained for parsing the yaml file. The Jackson library is used to de-serialize yaml, as it is well established. A yaml file was defined for each job type. Examples of these files which are used in the staging environment can be found in the integration sources repository [16].

The factory class yields an object that implements the interface `InternalModel`. This interface has the virtual method `toCore` that transforms an instance of `InternalModel` to a `CoreModel` instance, that is suited to being sent to I-On Core. This transformation is performed in the process phase of the step. More details on the transformation from `InternalModel` to `CoreModel` are provided in the implementation chapter.

The writer of the second step is responsible of sending the `CoreModel` object to I-On Core. After retrieving the i-on core address from the app properties, it sends the payload to the specified route. A similar retry logic to what was implemented in the ISEL Timetable Batch job is in place.

A Core service abstraction had been developed in the context of the ISEL Timetable Batch job for sending information to Core. It was completely reused. Two new methods were added: `pushCoreTerm` to send academic calendar one term at a time, and `pushExamSchedule` to send all the exams for each programme in one semester. However, these two methods call a lower-level method that was already developed for the first job.

Finally, step 3 of the generic batch job is defined in the `PostUploadTasklet`, which is already covered in this section. Further details on this tasklet can also be found in the ISEL Timetable Batch Job architecture document and in the ISEL Timetable Batch job section of the implementation chapter.

3.3.3 Retry and Skipping Capabilities

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, we did not make use of any skip policy.

3.4 Integration and deployment environment

3.4.1 Version Control

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

It has many benefits, among them [17]:

- 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* was used. *Git* [18] 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*⁶ 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 ⁷. Another solution studied was *Bitbucket*⁸ but *GitHub* was ultimately chosen given that its free tier plan offers more options.

3.4.2 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 used *GitHub flow*⁹ as it is quite simple to follow and implement.

This simple strategy states that when working on any feature or bug fix, a branch should be created [19], as shown in Figure 6. 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*¹⁰ can the work be merged.

⁶<https://github.com/>

⁷<https://github.com/features>

⁸<https://bitbucket.org/product/>

⁹<https://guides.github.com/introduction/flow/>

¹⁰*GitHub* allows to choose what team members are responsible for code in a repository via the creation of a CODEOWNERS file [20]

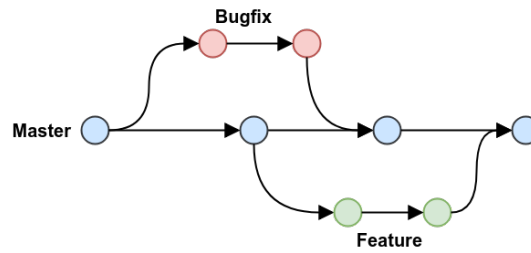


Figure 6: GitHub flow [1]

With this we could guarantee that the work present in *master branch* was always deployable and that it represented the latest iteration of the product. But there was also the need to mark specific iterations, that can be released. For this we used *Tags*.

A *Tag* is like a branch that doesn't change [21], 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 was also extremely important when trying to automate the process of building and deploying the code. This will be further discussed in *Continuous Integration* section.

3.4.3 Linter

Linter is a tool that analyzes source code to flag programming errors, bugs, stylistic errors, and suspicious constructs [22].

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* was the main development language used in the project, *ktlint*¹¹ was used. *Ktlint* is an open-source *kotlin linter*. It provides an *CLI*¹² and also integration with *Maven*¹³ and *Gradle*¹⁴.

3.4.4 Unit Tests

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

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

*JUnit5*¹⁵ was 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

¹¹<https://github.com/pinterest/ktlint>

¹²**C**ommand **L**ine **I**nterface

¹³<https://maven.apache.org/>

¹⁴<https://gradle.org/>

¹⁵<https://junit.org/junit5/>

known as *xUnit*, that originated with *SUnit* [23].

Kotlin provides a library, *kotlin.test*, that has annotations to mark test functions and a set of utility functions for performing assertions in tests [24]. It also provides an implementation of the *Asserter* class on top of *JUnit5*.

3.4.5 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 was error-prone as version compatibility needed to be ensured. Configuring the development and production environments included making sure the target machine and the development machines had the same versions of the dependencies the software needed to execute.

In order to eliminate this issue, a recent trend involves packaging up software code and all its dependencies in a container [25]. This allows for the project to run uniformly and consistently on any infrastructure.

For this reason we decided to package the project code in a container, specifically in a *Docker*¹⁶ container.

3.4.6 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 [26]. This allows to quickly check errors and maintain the master branch 'clean'. If the build is successful a deployable package is available at the end for testing or releasing.

The *Continuous Integration* pipeline:

- Builds the code;
- Runs *Lint*er;
- Runs *Unit tests*;

This pipeline can be achieved by the use of tools such as *Jenkins*¹⁷ or *Team City*¹⁸ or by the use of services such as *GitHub Actions*¹⁹ or *Travis CI*²⁰.

The final choice was to use *GitHub Actions* since it was already integrated on *GitHub* which we are using to host the code. It was also contemplated on the free tier plan. *Travis CI* was also considered, 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.

¹⁶<https://www.docker.com/>

¹⁷<https://www.jenkins.io/>

¹⁸<https://www.jetbrains.com/teamcity/>

¹⁹<https://github.com/features/actions>

²⁰<https://travis-ci.org/>

3.4.7 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 [27] 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 [28] ²¹ to be used in the first 12 months. This credit can be used as a safety net if the free tier limits are exceeded.

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 [29], which is a managed serverless alternative, and Google Kubernetes Engine [30], 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 ²². 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 [31]. 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.

²¹ Azure provides 170\$ credit for the free-tier

²² "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

4 Implementation

The present chapter highlights some aspects of integration sub-system's implementation that we found needed to be recorded for future reference.

4.1 Academic Information consumption system

4.1.1 Configuring Batch jobs

Configuration is of vital importance to batch jobs in Spring Batch, so much so that Spring Batch developers chose to identify a batch job instance as the conjunction of the batch job name and its parameters, making it impossible to run the same job to completion with the same parameters.

Spring Batch provides a way to pass external configuration to a job through the command line. It is similar to command line switches, but with no double hyphen.

For example: `java -jar configurations-0.0.1-SNAPSHOT.jar key=value key2=value2`

Parameters passed this way are translated to a `JobParameters` instance on application start, with inferred parameter type. As they are stored in a column of a relational database, the supported types are Date, Double, Long and String [32].

This approach has some drawbacks. First, as with command line switches, the java command can become very large if the number of parameters grows. Moreover, launching multiple instances of the same job with different parameters would require running more than one spring batch application.

In order to avoid these downsides, the integration subsystem combines the use of `JobParameters` with properties files for external configuration.

On application launch, the jobs that are to be run are specified, as well as the directories where to look for configuration files. For example, if the job-1 runs for school-a and school-b, the method `runJob` will be executed with the job name and the configuration Path.(e.g. `runJob("job-1", "src/main/resources/config/job-1/school-a")`, and `runJob("job-1", "src/main/resources/config/job-1/school-b")`))

This method will list all the files in the directory. Then, for each file, it will inject its properties on a `JobParameters` object. Finally, it will start a job instance with the added parameters, including a timestamp in order to be able to run the same job with the same parameters more than once.

Job configuration files are included in the project's image, in `.../resources/config/{batch-name}/{school-name}/`.

The configuration file name can be chosen freely, but for reasons of organization it should be named `{school-name}-{course-name}.properties`(e.g. `isel-leic.properties`). There is no need to include the job name in the properties file name because the path of the file already identifies the school.

All properties are injected in `JobParameterBuilder` as strings, but before usage, job parameters are coerced to more appropriate types to ensure configuration correctness.

4.1.2 Running batch jobs

The method `setupAndRunJob` on the `JobEngine` class receives the name of the job and the folder where the config files are located. For every configuration file, a batch job instance will be started. Every configuration property is injected as a job parameter.

As Spring Batch does not allow the execution of the same job with the same parameters, a timestamp is provided as job parameter to differentiate between batch job executions.

4.1.3 Scheduling batch jobs

The `JobEngine` class concentrates the logic of scheduling and execution of jobs. It needs to be annotated as a Configuration class, in order to "be processed by the Spring container to generate bean definitions and service requests for those beans at runtime".

It also needs to be annotated with the `@EnableScheduling` annotation in order to be able to use the `@Scheduled` annotation. Spring Batch scheduling works at a method granularity. Adding a new scheduled job to our application should include register a method with a `@Scheduled` annotation, where the `setupAndRunJob` method is called with job name and the configuration folder.

4.2 Batch jobs

4.2.1 ISEL Timetable Batch Job

As stated in the last chapter, the present section provides details as to the steps that comprise the isel timetable job.

4.2.1.1 Step 1 - Downloading and Comparing

Given that the timetable extraction job was dependent on information that is not controlled by the I-On Integration project, namely the ISEL timetable PDF, we needed to make sure the document was in the designated location. Also, we didn't want to send information that was 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 different, then we know that the document content is not the same. In that case, the job proceeds otherwise it stops 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 was a fairly inexpensive way of avoiding processing repeated information. It was suited to this document in particular, since it was 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 remains unaltered until the end of the semester.

4.2.1.2 Step 2 - Verifying format

The second step was 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 a string and a list of strings, the first for the output of *Tabula* and the second 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 is published to a state instance that was declared in the job configuration class 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.

Application of business rules to transform json and text data to domain object

In step 2 of the timetable batch job, the information extracted from the pdf was stored in an instance of a dynamic object (*ISELRawData*) in two formats - a string in the json format, containing all the data from tables found in the document, and a list containing a string per pdf page, from which header and footer information will be retrieved.

In the context of Step 3, this data is mapped from the dynamic object (*ISELRawData*) into a domain object (*TimetableTeachers*).

To perform the mapping we implemented a builder of *TimetableTeacher* instances. The builder pattern was followed as it is fit for scenarios where not all the input information for the operation is available at the same time. This was not the case with *ISEL*, but can be for other institutions. As the mapping information is very much adapted to the *ISEL* case, the builder is called *ISELTimetableTeachersBuilder*.

The ISELTimetableTeachersBuilder implements the TimetableTeachersBuilder interface, which exposes two methods, setTimetable and setTeachers, to convert timetable and teachers raw data to TimetableTeachers, which is ultimately more suited to being uploaded to I-On Core.

As was stated, the output model of this operation is the TimetableTeacher class. This type is composed of two lists - one with the entire course offer (Timetable) and another with all the associations between teacher and course (CourseTeacher).

The algorithm to yield the TimetableTeacher from ISELRawData can be broken down into four steps:

- For each element in the textData list, initialize a Timetable and CourseTeacher instances;
- Set the common data;
- Get Course and Faculty lists;
- Add the instances to an output list;
- After all the elements of the textData were iterated through, initialize a TimetableTeachers instance with the two resulting lists.

The first item was already covered. As for item 2, the Timetable and CourseTeacher are enriched with general data that is the same for each class section, such as school, programme, calendar term and calendar section. This information is obtained searching for matches for regular expressions in one page's string data.

The only source of information for item 3 is the result of de-serializing the json string in ISELRawData - a list of tables.

Each page of the isel timetable pdf has two tables, one with the course events, and another with the faculty association to courses. Then, in an array of tables, which is the result of de-serializing the json string in ISELRawData, element 0 was used to get course information, and element 1 to get faculty information in page 1. The same rule applies to the following pages.

The actual source of information is not the table element itself, but rather a field called data, which is a two-dimensional array of cells.


The list of courses was added upon, with the courses from one class section at a time. In order to obtain the list of courses for a class section we can further break down the algorithm into 6 steps:

- Populate weekdays;
- For each cell, if its text property is null or empty, ignore it;
- Get begin time;
- Store course details in triple (course, course type and location)
- Obtain duration of class
- Add Course to courseList

The weekday in which a course event occurs is determined by the column in which a cell appears in the table. The column can be identified by its left limit attribute, which is an x coordinate on the page. This limit is visually highlighted in figure 7. We collect all mappings from left cell limit to weekdays. As weekdays are in portuguese, and I-On Core requires them to be sent in english abbreviated form, a simple mapping is performed.

Licenciatura em Engenharia

Turma: LI11D



	Segunda	Terça
8.00 - 8.30		
8.30 - 9.00		
9.00 - 9.30		
9.30 - 10.00		

Figure 7: Timetable pdf - detail of column identifier: the left cell limit

Begin time for a class is determined by the line in which the cell occurs. When iterating over the two-dimensional array of cells, it is certain that we iterate over the first cell of the line before any of cells of the next line. So it's safe to assume that we process all the course events for that time before any of the course events which occur after.

During iteration we can maintain a state variable for current begin time, that will be uniform and applicable to all the cells in that line, which will occur before any of the line below. A regex is used to identify a cell which has a time interval in its text property.

Course details are obtained from the cells that do not fall into any other category, i.e. which are neither empty, not hold day of week, nor time slot. Such cells hold three details: course acronym, course type - either T(heory) or P(ractice) - and Store course details in triple (course, course type and location) location. As course type is delimited by parenthesis, obtaining details is fairly straightforward.

Duration of course event is determined by cell height and fall into three categories: higher than 47, which correspond to 3-hour events, between 17 and 47 - hour and a half events, and lower or equal to 17 for 30 minutes events. Figure 8 shows cells in the first and second categories.

	13.30 - 14.00		
	14.00 - 14.30		
	14.30 - 15.00	ALGA[I] (T)	E.1.31 E[I] (C)
	15.00 - 15.30		
	15.30 - 16.00		
	16.00 - 16.30		
	16.30 - 17.00	Pg[I] (T)	G.0.08 LSD[I]
	17.00 - 17.30	Pg[I] (P)	L_S1 LSD[I]
	17.30 - 18.00		
	18.00 - 18.30		
	18.30 - 19.00		

Figure 8: Timetable pdf - detail of cell height

The table that holds faculty data has four columns, the first and third hold the name of a course and the second and fourth hold the name of the lecturer associated with that course.

In a line where the course cell is empty but the lecturer has text, it is assumed that the lecturer teaches the course in the preceding line, as is shown in figure 9

E[1] (P)	Fernando dos Santos Azevedo
	João Manuel Ferreira Martins

Figure 9: Timetable pdf - detail of course taught by more than one teacher

After getting faculty and courses lists for each class section and getting common data, Timetable and CourseTeacher are used to build an instance of TimetableTeachers with the two lists.

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 is configured via the application.properties 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 the 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 Configuration Document for the timetable job

The integration application starts an instance of the ISEL Timetable Batch job for each file present in src/main/resources/config/timetable/isel. The parameters passed for each job instance are the properties contained in a file of that directory.

The following properties are accepted:

Property	Description
srcRemoteLocation	Url of the timetable pdf to be used on the job.
alertRecipient	Email of the point-of-contact to notify about job outcome

Table 1: Properties supported in the ISEL Timetable Job

4.2.1.7 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 was divided according to its source, as it was found on the page header or on the tables in the center and footer of the page. This categorization was relevant in the context of the timetable 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": {
    "name": "INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA",
    "acr": ""
  },
  "programme": {
    "name": "Licenciatura em Engenharia Informática e de Computadores",
    "acr": ""
  },
  "calendarTerm": "2019/20-Verão",
  "calendarSection": "LI11D",
  "language": "pt-PT",
  "courses": [
    {
      "label": {
        "acr": "ALGA"
      },
      "events": [
        {
          "title": "",
          "description": "Aulas Teóricas de ALGA",
          "category": 2,
          "location": [
            "E.1.08"
          ],
          "startDate": "2020-01-20T12:30",
          "endDate": "2020-01-20T15:30",
          "weekday": [
```

```

        "FR"
      ]
    }
  ]
}

Faculty data:

{
  "school": {
    "name": "INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA",
    "acr": ""
  },
  "programme": {
    "name": "Licenciatura em Engenharia Informática e de Computadores",
    "acr": ""
  },
  "calendarTerm": "2019/20-Verão",
  "calendarSection": "LI111D",
  "language": "pt-PT",
  "courses": [
    {
      "label": {
        "acr": "ALGA"
      },
      "teachers": [
        {
          "name": "Teresa Maria de Araújo Melo Quinteiro"
        }
      ]
    }
  ]
}

```

It is evident why it was 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 was occurring at roughly the same time, the core module would have 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 was necessary to disambiguate.

4.2.2 Generic Batch job

4.2.2.1 Yaml input files

We saw earlier that what motivated us to define yaml files as input for the generic batch job were mainly practical reasons. However, we wanted the model to work not just for ISEL, not just for higher education, but for any education institution.

Defining the exam schedule was simpler than the academic calendar. Apart from school and programme information, one exam identifies the curricular unit it refers to, the time it starts and the time it finishes. A sample is included below:

```

school:
  name: Instituto Superior Engenharia Lisboa
  acr: ISEL
programme:
  name: Licenciatura em Engenharia Informática e de Computadores
  acr: LEIC
calendarTerm: 2019/20-Verão
exams:
  - name: AED
    startDate: 2020-07-09T14:00:00.000
    endDate: 2020-07-09T17:00:00.000
    location:
  - name: AED
    startDate: 2020-09-03T19:00:00.000
    endDate: 2020-09-03T22:00:00.000
    location:

```

As for the academic calendar, as it is school-wide, we don't need to include programme information. We have to identify the calendar term, interruptions, evaluations and details. In the details, the beginning and end of classes is specified. As for other events, they are mainly administrative deadlines.

```

school:
  name: Instituto Superior Engenharia Lisboa
  acr: ISEL
terms:
  - calendarTerm: 1920i
interruptions:
  - name: Férias Natal
    startDate: 2019-12-23
    endDate: 2020-01-04
evaluations:
  - name: Exames época normal
    startDate: 2020-01-13
    endDate: 2020-02-01
    duringLectures: false
  - name: Exames época recurso
    startDate: 2020-02-03
    endDate: 2020-02-15
    duringLectures: false
  - name: Exames época especial
    startDate: 2020-02-26
    endDate: 2020-03-07
    duringLectures: true
details:
  - name: Turmas 1º Semestre
    curricularTerm:
      - id: 1
      startDate: 2019-09-16
      endDate: 2020-01-11
  - name: Turmas excepto 1º Semestre

```

```

    curricularTerm:
      - id: 2
      - id: 3
      - id: 4
      - id: 5
      - id: 6
    startDate: 2019-09-09
    endDate: 2019-12-21
  otherEvents:
    - name: Divulgação de horários
      startDate: 2019-07-22
      endDate: 2019-07-22
    - name: Abertura das atividades letivas 2019/2020
      startDate: 2019-09-02
      endDate: 2019-09-02
    - name: Data limite para lançamento de classificações no Portal Académico (frequência)
      startDate: 2020-02-21
      endDate: 2020-02-21
    - name: Data limite para lançamento de classificações no Portal Académico (época especial)
      startDate: 2020-03-14
      endDate: 2020-03-14

```

4.2.2.2 Transforming the Integration Internal Model to Core Model

In this section, we will go into more detail about the conversion process between internal model - the model that a factory returns - and the CoreModel - the model that InternalModel can convert to, and that is best suited for sending to I-On Core. First we will go over the academic calendar and then we'll cover the exam schedule.

Academic Calendar I-On core receives academic calendar information one term at a time. Fortunately, that is also the format in which the information is layed-out in the academic calendar yaml file.

The only difference is in what sort of elements are present in a term object. For the integration internal model, which for the present case is the same that is on the yaml file, there are different sorts of events in the term object.

Each term has details - class begin and end dates for the different curricular semesters -, evaluations - which can include lectures or not, by way of a boolean value -, interruptions and other events - typically administrative deadlines.

A Term object for core however, identifies school, calendarTerm, begin and end date, language and a list of intervals. The begin and end dates for the semester are the minimum and maximum dates of its intervals. An interval, apart from begin and end date, carries a name. Optionally, it also has curricularTerm (e.g. 1-6 in a 3-year undergraduate degree), categories - which is a list that can take up the values in the I-On Core documentation [33] - and excludes, which is also a list of identifiers defined in the same section. Excludes signals to Core that there cannot be events of the specified type in that interval.

The transformation process is relatively straightforward. We need to calculate begin and end date of the term, as was already stated. Start dates cannot be after end dates. If there is a period of evaluations during lectures, they are of type lecture and exam, if not they are only

of type exam. When a period of evaluations is not during lectures, it excludes lectures. Interruptions also exclude lectures. The details events are of type lectures, as they are by definition periods in which classes occur.

Exam Schedule The exam schedule core model was based on the timetable model. In the timetable model, the events sent were recurrent events, whereas exams are non-recurrent events.

Apart from school and programme information, a `CoreExamSchedule` object has calendarTerm, language and a list of exam events. Exam events are grouped by a label that is the acronym of the course the exam refers to. As was the case with the academic calendar events, start date cannot be after end date. The fields start date and end date are date time, as the hour of the exam is relevant.

4.2.2.3 The retry mechanism of GenericCoreWriter

The retry mechanism in place when uploading information to I-On Core has the same logic in the ISEL timetable batch job or the generic batch job. However, the code could not be reused because the context of the first was a tasklet and the second an ItemWriter.

In a tasklet, the execute method returns RepeatStatus. So, returning RepeatStatus.CONTINUABLE triggers calling the execute method once again. In the context of an ItemWriter, which method write returns void, that out-of-the-box repeating capability is not in place. However, placing the retry logic inside a while loop achieves the same behavior.

4.2.2.4 Adding a new job type to the generic batch job

Defining a new flow for the generic batch job requires defining a new JobType enum, defining an internal model factory in GenericFactory.getFactory, defining an internal model implementation, including the toCore method. The Writer also needs intervention, namely adding a case to a when expression in mapJobTypeToUploadType, specifying the method of core service that should be called for upload. Developing a new method in core service for uploading the new Core Model class instance.

4.2.3 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.

4.3 Integration and deployment environment

4.3.1 Continuous Integration

As we've discussed previously on the *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 have considered also the event when a new branch is pushed to *GitHub*, but we were limited by

free tier plan in order to keep the costs down.

Figure 10 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.

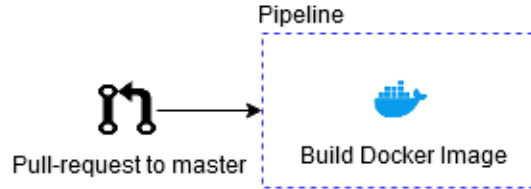


Figure 10: Pipeline when creating a *pull-request* to *master* branch

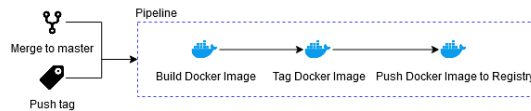


Figure 11: Pipeline when merging to *master* branch or pushing a new *tag*

When code is merged to *master* or when a new *tag* is created, the pipeline is as shown in Figure 11. Additionally to building the *Docker* image, the image is tagged and pushed to *Google Cloud Container Registry*²³. 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.

Given that we used *GitHub Actions*, these pipelines were described via *YAML*²⁴ files. Next we have the *YAML* file that describes the pipeline when *pull-request* to *master* is made:

```

name: I-On Integration Staging

on:
  push:
    branches:
      - master
    paths-ignore:
      - 'docs/**'

env:
  GCPLOUD_PROJECT_ID: ${ secrets.GOOGLE_CLOUD_PROJECT_ID }

jobs:

```

²³Container registry where *Docker* images can be pushed and pulled to create containers

²⁴YAML is a human-readable data-serialization language

```

push_integration:
  runs-on: ubuntu-latest
  if: github.event_name == 'push' && contains(github.ref, 'heads')

  steps:
    - name: Checkout
      uses: actions/checkout@v2

    - name: Setup gcloud CLI
      uses: GoogleCloudPlatform/github-actions/setup-gcloud@master
      with:
        project_id: ${{ secrets.GOOGLE_CLOUD_PROJECT_ID }}
        service_account_key: ${{ secrets.GOOGLE_CLOUD_SERVICE_ACCOUNT_KEY }}

    - name: Gcloud as credential helper
      run: gcloud auth configure-docker

    - name: Build image
      id: integration_build_image
      run: ./gradlew buildDockerImage

    - name: Push image
      id: integration_push_image
      run: ./gradlew tagPushDockerImage

    - name: Deploy
      id: integration_deploy
      run: ./gradlew deploy

```

The 'on:' parameter describes the event upon which the pipeline should be triggered. In this case, it is triggered when a push is made to master, with the exception of files under the docs folder. It is possible to define *environment variables* to be available during the pipeline execution under 'env:'.

The 'jobs:' parameter describes the steps executed each time the pipeline runs, and their order, namely, the project checkout, building and tagging the docker image, and finally the deployment. There are also steps involving the setup and use of *Google Cloud CLI* that are needed for the deploy step.

4.3.2 Continuous Deployment

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

Deployment to the staging environment (figure 12) 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 Google Cloud Container Registry.

Similarly to what was done for the major steps of continuous delivery, this was achieved using a gradle task instead of the Cloud SDK command line [34] directly on the pipeline. This

way we didn't need to rely on the pipeline if we wanted to test deployment aspects.

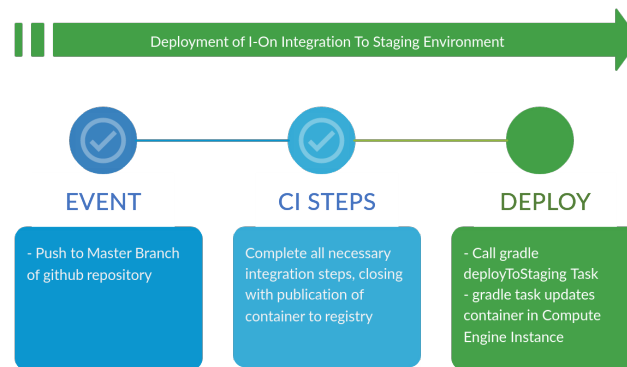


Figure 12: Deployment to staging environment

As for deployment in production environment (figure 13), 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.

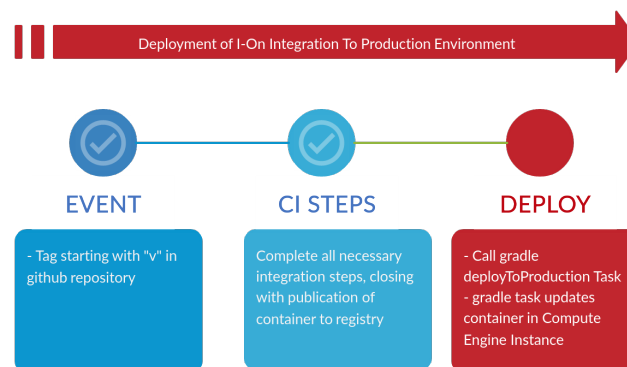


Figure 13: Deployment to production environment

To enable this strategy, two *Google Cloud Compute Instances*²⁵ were created: i-on-integration-staging and i-on-integration-production. Both instances were configured to use a *Container Optimized Operating System*²⁶ recommended by *Google*, and to have static public *IP address*.

Also two *Google Cloud SQL*²⁷ instances were created with the same names, given *Spring Batch* requirements to persists data related to batch jobs executions. Both were configured to only allow connections from the public *IP address* assigned to the respective *Google Cloud*

²⁵<https://cloud.google.com/compute/>

²⁶<https://cloud.google.com/container-optimized-os/>

²⁷<https://cloud.google.com/sql/>

Compute Instances.

One thing to be noted is the impossibility to reserve static public *IP address* for the *Google Cloud SQL* instances. This presented a challenge, since the application needed to know the database url.

The solution found was to configure a startup script to run on both *Google Cloud Compute Instances* that would start an instance of *Google Cloud SQL Proxy*²⁸. The application communicated with the *proxy*²⁹ that used a secure tunnel to communicate with the respective *Google Cloud SQL* instance.

²⁸<https://cloud.google.com/sql/docs/postgres/sql-proxy>

²⁹Application that acts as an intermediary for requests from clients to servers

5 Limitations and Further Improvements

In this chapter we'll discuss the project limitations that were identified and, when possible, ideas on how to improve them. At the end of each topic there will be an *hashtag* followed by a number that represents the respective *GitHub issue*³⁰, allowing future contributors to improve the project.

5.1 Academic Information consumption system

In the final version of the project it is very easy to add additional batch job types and even use some of the developed modules, e.g. *FileDownloader*, *PDFExtractor* and others. But in order for a batch job to be triggered it is necessary to add specific code that launches the new job or schedules the launch.

By having each job implement a specific interface and then using reflection, it would be possible to find and launch all jobs, thus removing that responsibility from the developer. **#123**

Another limitation relates to the scheduling ability. Even though the project enables the scheduling of batch jobs, the implemented solution ties the scheduling to the code that launches each type of batch job, instead of each configuration file. For instance, a desired behavior could be to import a ISEL course timetable data every Wednesday, but another every Thursday. Unfortunately this is not possible given that these are 2 executions of the same batch job type, in this case the ISEL Timetable Batch Job. **#124**

Additionally, while still possible to run the project on *Windows* platform, there is the need to update the configuration files location paths since they follow the *UNIX filesystem layout*³¹. While running with *Docker* bypass this issue, the project should still be able to run on all major platforms, i.e. *Windows*, *Linux* and *macOS*, without the need of changing anything. To solve this issue the paths should be relative instead of absolute. **#89**

Still related, a few of the unit tests may fail when executed repeatedly on *Windows* platform since some files are not deleted. **#125**

Finally, in the future, I-On Integration could also assume the role of producer, by enabling the publishing of information to external systems, e.g. *Moodle*.

5.2 Batch jobs

After reading chapters 3 and 4, you may come to the conclusion that both the ISEL Timetable Batch Job and Generic Batch job share much of the same logic. Unfortunately the final implementation consist of two separate batch job definitions.

Instead there should be only a single batch job definition with moving parts, e.g. using *factory* and/or *builder* patterns³², maximizing the reuse of modules. **#126**

³⁰<https://github.com/i-on-project/integration/issues>

³¹https://en.wikipedia.org/wiki/Unix_filesystem#Conventional_directory_layout

³²<https://refactoring.guru/design-patterns/creational-patterns>

5.2.1 ISEL Timetable Batch Job

The batch job fails for some of ISELs programmes. This is due to the fact that the respective Timetable PDF follows a slightly different layout, making it hard to define rules that apply to all cases.

Still additional effort should be made in order for the batch job to be able to successfully import timetable data from all of ISELs programmes. **#127**

5.2.2 Generic Batch job

One of the major differences between this batch job and the ISEL Timetable Batch Job is that for the latter we don't have the knowledge when there is a change to the data, but for the former we do.

This means that instead of the implemented solution where the job is triggered based on a schedule, resulting in unnecessary consumption of resources, any update to the data should be the stimulus to trigger the job. This could be achieved by the use of GitHub Webhooks³³, i.e. the merge of new data would trigger a call to be made to I-On Integration triggering the batch job execution.

Additionally the exam schedule job type right now fails due to the functionality not being fully developed by I-On Core. Once the functionality is available very little effort remains to finalize the exam schedule job type. **#128**

Finally the timetable job type should be implemented, allowing timetable data from other institutions besides ISEL to be imported to I-On Core. **#129**

5.3 Integration and deployment environment

Currently each *Google Cloud SQL* database is configured to only accept connections from the respective environment counterpart *Google Cloud Compute Instance*, but don't require *SSL/TLS certificates*³⁴ to connect. In order to improve security *SSL/TLS certificates* should be required for all connections. **#130**

Additionally both *Google Cloud Compute Instance* are cost-optimized VMs which could lead to performance issues. Even though *Google Cloud* provides logging tools, the log data could be pushed to another database, i.e. *Elasticsearch*³⁵, and then use data-visualization tools, i.e. *Kibana*³⁶ or *Grafana*³⁷ to help monitoring and management tasks.

Finally on the repository that contains the *yaml* source files that are processed by the Generic Batch job, there is no pipeline to automate the process of validating added or modified files. **#131**

³³<https://developer.github.com/webhooks/>

³⁴<https://www.websecurity.digicert.com/security-topics/what-is-ssl-tls-https>

³⁵<https://www.elastic.co/enterprise-search>

³⁶<https://www.elastic.co/kibana>

³⁷<https://grafana.com/>

6 Conclusion

I-On Integration is currently deployed on Google Cloud and running multiple scheduled batch jobs that import academic information from external systems to I-On Core.

The main factor that impacted the progress of the project during the first two and a half months were 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.

One of the main goals was to have the project deployed and running in a production environment, which was achieved.

However 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.

Another main goal focused on the quality of the code. By requiring that all code that is merged to master has to not only be reviewed, but additionally pass the pipeline give us a lot of confidence. This has a minor drawback in that integration of new code takes longer and the teams velocity slows down.

As a result, our initial plan was slightly delayed. As the more substantial architecture and definition decisions were then taken, and the ensuing tasks had a more implementation-related component, the progress pace increased moderately.

Despite these delays, the project's deadline were met.

During the latter stage of the project, code reuse helped speed up the development process. Some of the code used in the generic batch job was written for the ISEL Timetable batch job.

In retrospective, we feel that developing a custom batch job for a school does not have a good cost-benefit relation. We feel that a solution more aligned with the generic batch job is better as the amount of information extracted by unit of time spent developing is greater.

References

- [1] “Github flow.” <http://files.programster.org/tutorials/git/flows/github-flow.png>. Accessed: 2020-04-29.
- [2] “Web boom 2.0.” <http://content.time.com/time/magazine/article/0,9171,1570789,00.html>. Accessed: 2020-04-28.
- [3] “What is batch processing?.” https://www.ibm.com/support/knowledgecenter/zosbasics/com.ibm.zos.zconcepts/zconc_whatishbatch.htm. Accessed: 2020-04-28.
- [4] “About pull requests.” <https://help.github.com/en/github/collaborating-with-issues-and-pull-requests/about-pull-requests>. Accessed: 2020-04-28.
- [5] S. 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.
- [6] “Fénix global project description.” <https://ciist.ist.utl.pt/projectos/Fenix.pdf> pp. 14,15. Accessed: 2020-04-29.
- [7] M. Minella and D. Syer, *The Definitive Guide To Spring Batch. 1st ed. Chicago, IL: Apress, p.18*. Apress, 2019.
- [8] “Spring projects.” <https://spring.io/projects>. Accessed: 2020-04-28.
- [9] “Spring batch.” <https://spring.io/projects/spring-batch>. Accessed: 2020-04-28.
- [10] “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.
- [11] “Spring announces kotlin support in spring initializr.” <https://spring.io/blog/2016/02/15/developing-spring-boot-applications-with-kotlin>. Accessed: 2020-04-29.
- [12] “Spring batch documentation - domain-specific language.” <https://docs.spring.io/spring-batch/docs/current-SNAPSHOT/reference/html/domain.html>. Accessed: 2020-04-30.
- [13] “Tabula’s github repository.” <https://github.com/tabulapdf/tabula-java>. Accessed: 2020-04-29.
- [14] “itext official webpage.” <https://itextpdf.com/en>. Accessed: 2020-04-29.
- [15] “Performance itext vs.pdfbox.” <https://stackoverflow.com/questions/22340674/performance-itext-vs-pdfbox>. Accessed: 2020-04-30.
- [16] “Integration sources repository.” <https://github.com/i-on-project/integration-sources>.
- [17] “What is version control.” <https://www.atlassian.com/git/tutorials/what-is-version-control>. Accessed: 2020-04-29.
- [18] “Git.” <https://git-scm.com/>. Accessed: 2020-04-30.
- [19] L. Mitchell, “Choose the right git branching strategy.” <https://www.creativebloq.com/web-design/choose-right-git-branching-strategy-121518344>. Accessed: 2020-05-01.

- [20] “About code owners.” <https://help.github.com/en/github/creating-cloning-and-archiving-repositories/about-code-owners>. Accessed: 2020-05-01.
- [21] “Git tag.” <https://www.atlassian.com/git/tutorials/inspecting-a-repository/git-tag>. Accessed: 2020-05-01.
- [22] R. 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.
- [23] “JUnit 5.” <https://junit.org/junit5/>. Accessed: 2020-05-01.
- [24] “kotlin.test.” <https://kotlinlang.org/api/latest/kotlin.test/>. Accessed: 2020-05-01.
- [25] “Containerization.” <https://www.ibm.com/cloud/learn/containerization>. Accessed: 2020-04-29.
- [26] “Continuous integration.” <https://www.thoughtworks.com/continuous-integration>. Accessed: 2020-04-29.
- [27] “Google compute instances product page.” <https://cloud.google.com/compute>. Accessed: 2020-04-01.
- [28] “Google cloud platform free tier conditions.” <https://cloud.google.com/free>. Accessed: 2020-04-01.
- [29] “Google cloud run product page.” <https://cloud.google.com/run>. Accessed: 2020-04-01.
- [30] “Google cloud kubernetes engine product page.” <https://cloud.google.com/kubernetes-engine>. Accessed: 2020-04-01.
- [31] “Spring batch documentation - asynchronous execution of jobs.” <https://docs.spring.io/spring/docs/current/spring-framework-reference/integration.html#scheduling>. Accessed: 2020-04-01.
- [32] “Spring batch jobparameterbuilder.” [tps://docs.spring.io/spring-batch/docs/4.2.x/api/org/springframework/batch/core/JobParametersBuilder.html](https://docs.spring.io/spring-batch/docs/4.2.x/api/org/springframework/batch/core/JobParametersBuilder.html).
- [33] “I-on core write api documentation.” <https://github.com/i-on-project/core/blob/master/docs/api/write/insertClassSectionEvents.md#Constants>.
- [34] “Google cloud sdk command line for interaction with compute engine.” <https://cloud.google.com/sdk/gcloud/reference/compute/instances>. Accessed: 2020-04-01.