AIFA scraper

Federico Magnolfi
federico.magnolfi2@stud.unifi.it

Software Architectures and Methodologies
March 2021

Contents

Chapter 1

Introduction

1.1 AIFA

The Italian Medicines Agency (Agenzia italiana del farmaco, AIFA) is the public institution responsible for the regulatory activity of pharmaceuticals in Italy. It carries out all activities related to the regulatory process relating to the drugs, from registration and commercial authorization to control of production workshops and of manufacturing quality [1], [2].

Databases are the essential tools with which AIFA monitors and analyzes the use of drugs at national level. The medicines database prepared by AIFA is the only official database that allows consultation of the summaries of the product characteristics and the updated leaflets of the medicines authorized in Italy [3].

AIFA provides a web interface to search for drugs by drug name, active ingredient or company name [3]. This website obtains data from an API service which unfortunately is not documented for those wishing to use it in their own applications.

1.2 Objective

The goal is to create a JavaEE [4] back-end with the following responsibilities:

- extract the list of AIFA drugs from the (drug) agency website
- organize them in a reasonable domain model
- make them accessible with some REST endpoints

1.3 Document structure

In chapter 2 we will analyze the services made available by AIFA: how to formulate requests, what is the format and the content of responses.
In chapter 3 we will describe the details of the created software.
In chapter 4 we will describe how to run the application through Docker.
In the appendix, we will describe what tools were used to build this software and other things that may be useful for future work.

Chapter 2

AIFA services

The AIFA website that allows you to search by drug name, company name, or active ingredient name, queries an API service under the hood.

Theoretically, this service could be queried by any app that wants to display the data of the various drugs.

Though, to the best of our knowledge, this API service is not documented. Therefore, it may be difficult to make certain requests directly to this service. This motivates the development of a server that scrapes data from the AIFA API, saves them in its own database, and then serves them through a well-documented RESTful API.

All we know about the AIFA API has been discovered by analyzing the browser's network traffic within the web interface and by seeing the responses to many hand-made requests. In this chapter we'll explain what we've discovered.

2.1 Request format

Endpoint

`GET https://www.agenziafarmaco.gov.it/services/search/select`

Optional request parameters

- `fl` (Fields List): [String] comma-separated list of fields to have in the response, useful to reduce the network traffic. Default: return all fields
- `q` (Query): [String] filter results (more details in the next paragraph). Default: no filter
- `rows`: [Integer] maximum number of returned results. Default: 10
- `start`: [Integer] number of results to skip, useful to paginate queries. Default: 0
- `wt`: [String] {csv|json|xml} format of the response. Default: xml

Queries

There are many ways to express queries

- generic: `q` must be `valueToSearch` (the search is probably done in all the descriptive fields)
- specific: `q` must be `fieldName:valueToSearch`
- combination: it's possible to combine many of the above queries in a single query using AND, OR and round brackets. It's also possible to combine queries with + or spaces (same effect as AND in both cases)

Some remarks on values:

- in generic queries, no spaces are allowed in the value
- in specific queries, spaces and other special characters (such as :) are allowed if value is enclosed in double quotes
- for string-type fields, the character * can be used as wildcard to catch any string (empty string also). Unfortunately, the same thing does not hold for date and datetime fields
- the wildcard * is not allowed when value is enclosed in double quotes
- the case-sensitivity of values seems to depend on the field: descriptive fields seem to be case-insensitive, whereas other fields are undoubtedly case-sensitive, or they exhibit an unpredictable behavior

For temporal fields, we did not find a way to search by year, month or day. We tried possible wildcards such as * or % without any success: the only way to filter by date seems to be inserting the exact datetime as `YYYY-MM-DDTHH:mm:ssZ` between double quotes.

2.2 Response format

The XML and JSON formats are more complete: they have more metadata, such as the total number of matching results or the type of each field. However, the CSV format requires less network traffic, and it is closer to the data structure of a relational database. We have chosen to use the CSV format.

Response Fields Naming Schema

Many but not all fields have a prefix that identifies their type:

- `bs_`: Boolean
- `dm_field_`: Date
- `ds_`: Datetime
- `sm_field_`: String

Important Fields

- `bundle`: type of the row. Possible values: {farmaco, confezione_farmaco}. Using 'confezione_farmaco' we get more details.
- `ds_last_comment_or_change`: timestamp of the last modification, useful to know if row changed from last scrape
- `sm_field_aic`: (AIC: Autorizzazione all'Immissione in Commercio; Marketing Authorisation) code that identifies both the drug and the packaging
- `sm_field_codice_atc`: (ATC: sistema di classificazione Anatomico, Terapeutico e Chimico; Anatomical Therapeutic Chemical classification system) code that identifies the active ingredient
- `sm_field_codice_confezione`: code that identifies the packaging, given the drug
- `sm_field_codice_ditta`: code that identifies the company
- `sm_field_codice_farmaco`: code that identifies the drug
- `sm_field_descrizione_atc`: description of the active ingredient
- `sm_field_descrizione_confezione`: description of the packaging
- `sm_field_descrizione_ditta`: description of the company
- `sm_field_descrizione_farmaco`: description of the drug
- `sm_field_link_fi`: (FI: Foglietto Illustrativo) url of the patient information leaflet of the drug
- `sm_field_link_rcp`: (RCP: Riassunto delle Caratteristiche del Prodotto) url of the summary of product characteristics
- `sm_field_stato_farmaco`: administrative state of the packaging. Possible values: {A, S, R}

Anatomical Therapeutic Chemical (ATC) Classification System

Active ingredients are classified into groups at five different levels:[5]

First level (one letter)
Indicates the anatomical main group. There are 14 main groups:[6]
`A`: Alimentary tract and metabolism
`B`: Blood and blood forming organs
`C`: Cardiovascular system
`D`: Dermatologicals
`G`: Genito-urinary system and sex hormones
`H`: Systemic hormonal preparations, excluding sex hormones and insulins
`J`: Antiinfectives for systemic use
`L`: Antineoplastic and immunomodulating agents
`M`: Musculo-skeletal system
`N`: Nervous system
`P`: Antiparasitic products, insecticides and repellents
`R`: Respiratory system
`S`: Sensory organs
`V`: Various
Example: C Cardiovascular system

Second level (two digits)
Indicates the therapeutic subgroup.[7]
Example: C03 Diuretics

Third level (one letter)
Indicates the therapeutic/pharmacological subgroup.
Example: C03C High-ceiling diuretics

Fourth level (one letter)
Indicates the chemical/therapeutic/pharmacological subgroup.
Example: C03CA Sulfonamides

Fifth level (two digits)
Indicates the chemical substance.
Example: C03CA01 furosemide

Packaging state

The packaging state [8] can be:

- A (Authorized): the medicinal product has obtained the marketing authorization (AIC) and can be marketed throughout the national territory
- S (Suspended): the medicinal is authorized for placing on the market, but this authorization is temporarily suspended and the drug is not available on the market
- R (Revoked): the marketing authorization has been revoked and the medicinal product is no longer authorized for marketing. If the withdrawal is not due to safety reasons, any stocks of the drug may, for a certain period of time, still be on the market

Other Fields

The response from AIFA services contains other fields (Table 2.1), which however are not used in our software.

Other fields	
bs_status	is_uid
bs_promote	label
bs_sticky	path
bs_translate	site
bundle_name	sm_field_chiave_confezione
content	sm_field_tipo_procedura
dm_field_last_update	spell
dm_field_upload_date	ss_language
ds_changed	ss_name
ds_created	ss_name_formatted
entity_id	teaser
entity_type	timestamp
hash	tos_name_formatted
id	url
is_tnid	

Table 2.1: Other fields in the response.

Date handling

We didn't find a way to filter by year, month or day on a certain column. However, it seems to be possible to filter by year using a generic query, with the desired year as value. In this case, it's (probably) returned every row in which the specified year appears in at least one of the date columns.

On February 22, 2021, we queried the services both for all records and both for specific year (from 2013 to 2021), analyzing the "numFound" entry in the JSON response. The queries were:

- endpoint: `https://www.agenziafarmaco.gov.it/services/search/select`
- parameters for all rows: `wt=json&q=bundle:confezione_farmaco`
- parameters for the year e.g. 2021: `wt=json&q=2021+bundle:confezione_farmaco`

With the following results:

- total rows: 203105
- total sum of single-year rows: 211238
  - 2013: 749
  - 2014: 43103
  - 2015: 2056
  - 2016: 24234
  - 2017: 11454
  - 2018: 28704
  - 2019: 21049
  - 2020: 41605
  - 2021: 38284

For years after 2021 or before 2013, there was no matching result.
Given the results, it can be hypothesized that even doing only queries per year all rows can be retrieved and only a small part of them is retrieved more than once. This is important because analyzing only the rows modified in the current year significantly reduces the server load during the scrape.

Chapter 3

Software documentation

The proposed solution consists of a JEE application, running on a WildFly server [9], capable of scraping data from AIFA services, save them in a MariaDB database [10], and make data accessible to clients via REST calls. Gradle [11] and Docker [12] are used to facilitate dependency management, compilation and execution of the backend and database.

3.1 System architecture

Figure 3.1 shows how our solution integrates into the overall system. We developed the backend and the database, integrating them with AIFA services and with an existing frontend application.

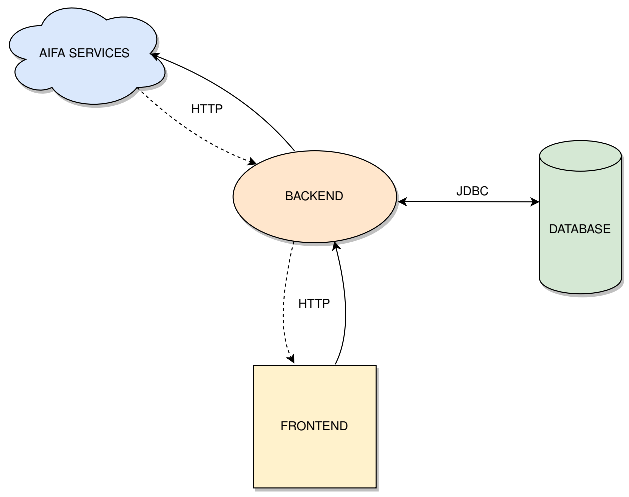

Figure 3.1: System architecture

3.2 Interactions

The relationship among different components of the architecture is better clarified by showing how the components interact on two example of requests (scrape and search).

Scrape

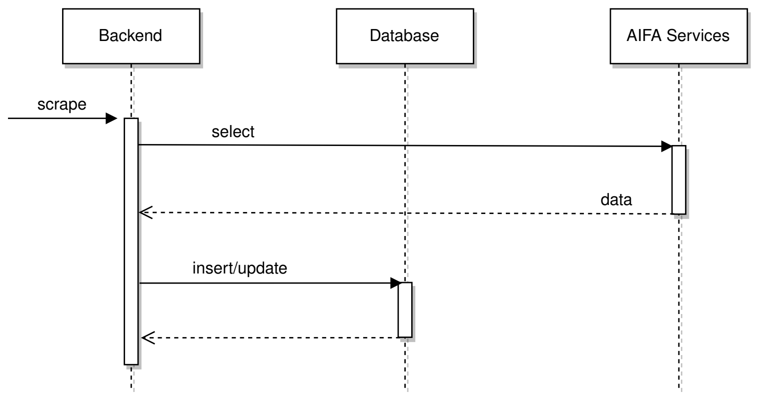

Figure 3.2: Scrape request: interaction among components

Search

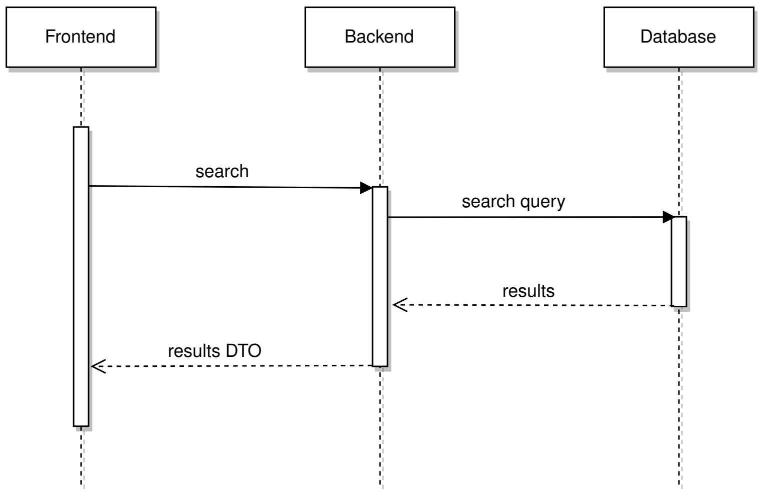

Figure 3.3: Search request: interaction among components

3.3 Database

Data are persisted in a relational database, using MariaDB as DBMS.
Data coming from AIFA services are essentially a CSV file (changing the format of the response from AIFA services is not helpful to better organize the data). These data have been divided into entities as reported in Figure 3.4. In addition to the entities that store drug data, the LastUpdates entity is added to reduce the server's workload when updating data: it allows to discard the records that have not been modified from the last update.
The entire structure of the database is automatically generated by the ORM. When using Docker (see chapter 4), the database itself is auto-generated too.

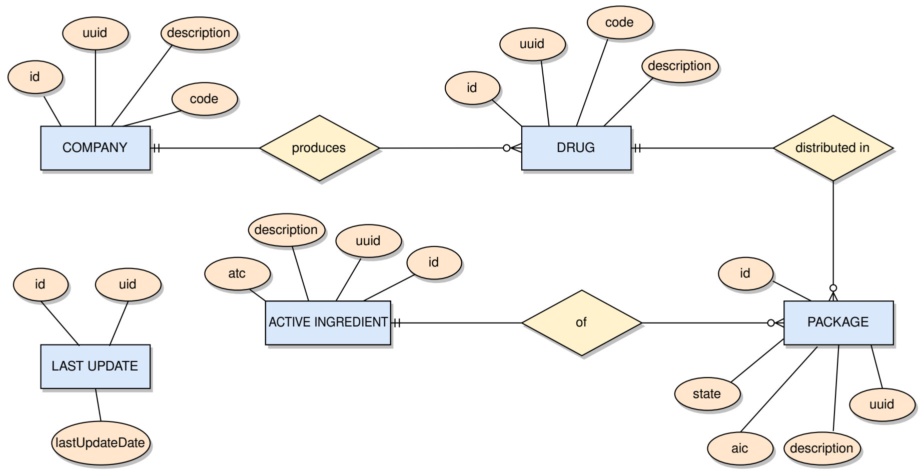

Figure 3.4: Entity-Relationship Diagram of the database.

3.4 Backend

The backend is a JEE application running on a WildFly server.

Packages

The software is subdivided into packages, each with different responsibility:

- `api`: exposes the endpoints
- `config`: enables CORS
- `controller`: business logic
- `dao`: provides data operations without exposing details of the database
- `dto`: defines the format of the objects sent from the server to the client
- `mapper`: converts domain model objects to DTO
- `model`: domain model
- `requester`: handle requests to AIFA services

The relationships between packages are described in Figure 3.5.

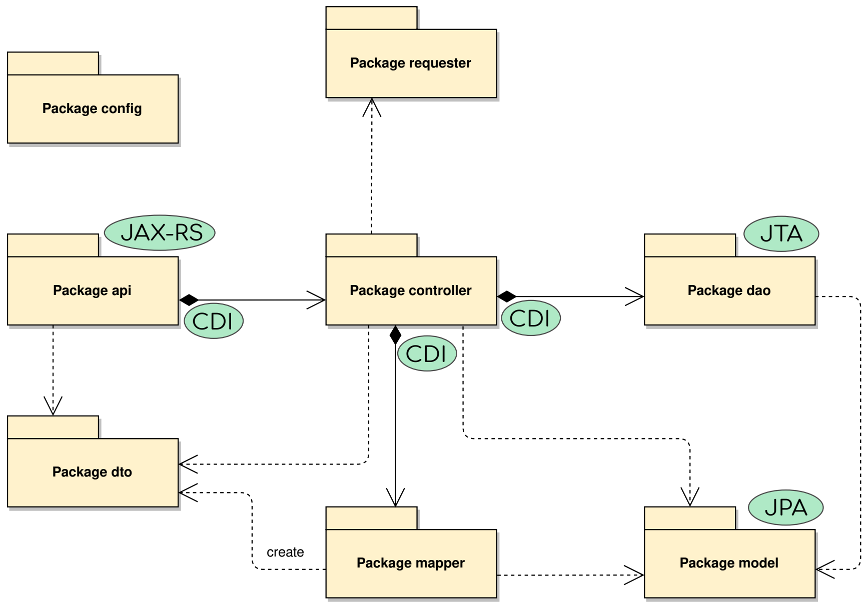

Figure 3.5: Packages Diagram: the dashed arrow represents a dependency: at least one obect of the other package is used in at least one function; the solid line with the rhombus identifies a stronger dependency: at least one reference to an object of the other package is mantained (and it is injected via CDI).

Domain model

The domain model (Figure 3.6) follows directly from the entities in the database (Figure 3.4). Each entity extends the same base abstract class, which contains the common operations: identity handling in constructor and equals() method, automatic conversion into JSON in toString() method. A factory creates the instances of all entities.

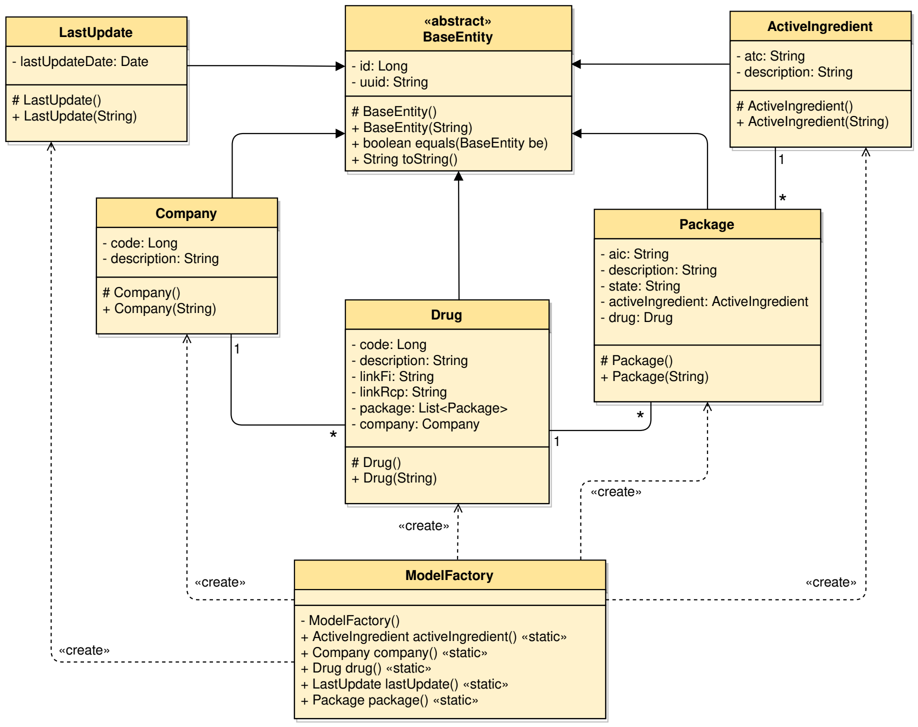

Figure 3.6: Class Diagram of the model package. Each private field has the corresponding getter and setter public methods, omitted here for simplicity.

Data Transfer Objects

Each entity is converted into a DTO (Data Transfer Object), which in turn will be transformed into the JSON to be sent to the client. The choice of the format of the DTOs was not free, but dictated by the need to be compatible with an existing client. The structure of the DTOs is shown in Figure 3.7.

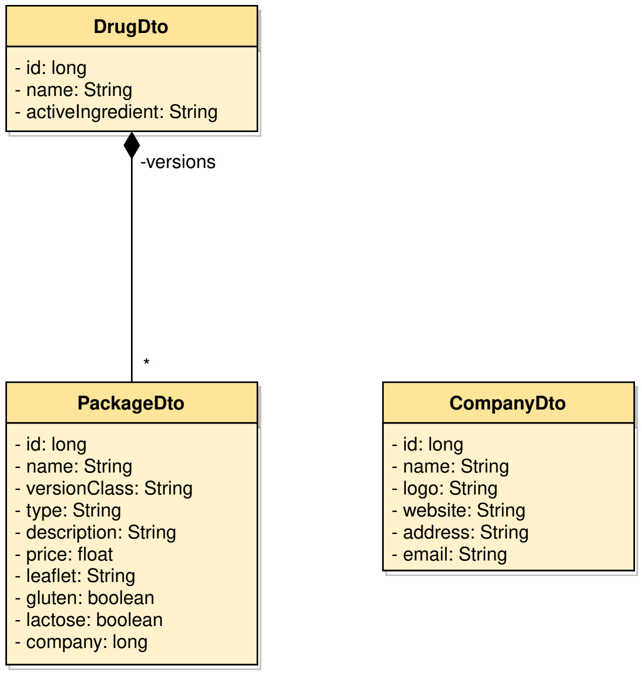

Figure 3.7: Data Transfer Objects

Endpoints

The system has many endpoints: one to start the scraping procedure, one to perform searches, and others to obtain the instances of drug, active ingredient, and company. The complete list is:

- `scrape`: start the scraping procedure
- `search|medications`: search on the database
- `activeIngredients`: get all active ingredients (or a specific one)
- `companies`: get all companies (or a specific one)
- `drugs`: get all drugs (or a specific one)

Scrape

Full endpoint: `PUT {backend_url}/api/scrape`
Parameters are reported in table 3.1.
Returns: a text reporting the elapsed time

To manage the available resources, two different pagination techniques are scheduled: one for the download of data from AIFA services (controlled by parameters *firstResult*, *downloadSize*, *maxResults*), and one for the number of insert in each transaction (controlled by parameter *transactionSize*).

Example (how to manually start the scrape procedure):

```
curl -X PUT http://localhost:8080/aifa-scraper/api/scrape?maxResults=300000
```

Example (schedule the scrape once a week, on Sunday at 04:00 a.m., using cron):

1. open the cron file: `crontab -e`
2. append the line: `00 04 * * SUN write_curl_command_here`

Parameter	Type	Default	Description
url	String	null*	url of remote service
file	String	null	path to csv file
firstResult	Integer	0	index of first result in download
maxResults	Integer	10	max results in all downloads
downloadSize	Integer	*maxResults*	max results per download
transactionSize	Integer	1000	max insert per transaction
checkLastUpdate	boolean	false	if true, discard records not updated since last time
year	Integer	null	request records with *year* in one of the date fields

Table 3.1: Parameters for scrape endpoint. Remark: at most one between *url* and *file* can be set.
*: if both *url* and *file* are *null*, *url* is set to
`https://www.agenziafarmaco.gov.it/services/search/select`

Search

Full endpoint: `GET {backend_url}/api/search`
For compatibility reason, there is an alias: `{backend_url}/api/medications`
Parameters are reported in table 3.2.
Type of returned JSON: `List<DrugDto>`

Parameter	Type	Default	Description
drug	String	null	value to search in drug description
name	String	null	alias for drug (compatibility reasons)
activeIngredient	String	null	value to search in active ingredient description
company	String	null	value to search in company description
text	String	null	value to search in all descriptions
firstResult	Integer	0	index of first result
maxResults	Integer	null	number of maximum results
retired	boolean	false	if true, return retired packagings too

Table 3.2: Parameters for search endpoint. Remark: one and only one parameter between *drug*/*name*, *activeIngredient*, *company* and *text* can be set.

Active ingredients

All active ingredients:

- full endpoint: `GET {backend_url}/api/activeIngredients`
- type of returned JSON: `List<ActiveIngredientDto>`

Specific active ingredient:

- full endpoint: `GET {backend_url}/api/activeIngredients/{id}`
- type of returned JSON: `ActiveIngredientDto`

Companies

All companies:

- full endpoint: `GET {backend_url}/api/companies`
- type of returned JSON: `List<CompanyDto>`

Specific company:

- full endpoint: `GET {backend_url}/api/companies/{id}`
- type of returned JSON: `CompanyDto`

Drugs

All drugs:

- full endpoint: `GET {backend_url}/api/drugs`
- type of returned JSON: `List<DrugDto>`

Specific drug:

- full endpoint: `GET {backend_url}/api/drugs/{id}`
- type of returned JSON: `DrugDto`

External Dependencies

JEE specifications

The following is a list of used JEE specifications:

- `CDI`: to perform dependency injection mechanisms
- `JAX-RS`: to expose the endpoints
- `JPA`: to obtain the persistence of system entities
- `JTA`: to demarcate the transactions boundaries

Of course an implementation for each of these specification is required. The code should be compatible with any valid implementation: there are almost no configurations or imports of specific implementations. The only exception is the persistence.xml file, which contains specific properties of Hibernate for the configuration of the persistence unit.

Libraries

Since returned strings from AIFA services containes HTML escape characters, they must be unescaped. To organize data in different entities, we have to execute grouping operations on the csv returned by AIFA. One other common operation is to transform objects into JSON before sending them to che client. To do these operations, we take advantage of existing libraries, as reported in Table 3.3.

Library	**Role**	**Dependent packages**
Apache Commons Text	Unescape HTML	controller
Tablesaw	Manage dataframes	controller
Jackson	Convert objects in JSON	dto, model

Table 3.3: Used external libraries, their roles, and dependent packages.

Chapter 4

Run the application

The source code of the application can be found at:
https://github.com/fedem96/aifa-scraper

As a build automation tool, we choose to use Gradle: the Gradle wrapper allows to compile the entire application without having Gradle installed on the machine.

4.1 Requirements

The following must be already installed:

- JDK 8
- (optional): Docker Engine 18.06.0+, Docker Compose 1.22.0+

4.2 Compile

With the Gradle wrapper, the compilation requires just one command.
Linux/macOS: `./gradlew war`
Windows: `gradle.bat war` The compilation generates a *aifa-scraper.war* file inside the *build/libs/* subdirectory.

4.3 Run

With Docker: `docker-compose up`
Without Docker: move the *aifa-scraper.war* file into the *standalone/deployments/* subdirectory of WildFly.

Chapter 5

Appendix

5.1 Postman

To try different requests to AIFA services, we used Postman [13].
The following code was written into the "Tests" tab to visualize the CSV response directly in Postman.

```
var template = `
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/
    bootstrap/3.3.7/css/bootstrap.min.css" integrity="sha384-
    BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/
    K68vbdEjh4u" crossorigin="anonymous">
<style>
    .table {
        font-size: 12px;
    }
</style>
<div id="root"></div>
<script src="https://cdnjs.cloudflare.com/ajax/libs/react/15.4.2/
    react.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/react/15.4.2/
    react-dom.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/babel-
    standalone/6.21.1/babel.min.js"></script>
<script src="https://cdn.jsdelivr.net/npm/react-csv-to-table@0.0.2/
    dist/index.min.js"></script>
<script type="text/babel">
let Table = window['react-csv-to-html-table'].CsvToHtmlTable;

class App extends React.Component {
    render() {
        return (
            <Table
                tableClassName='table table-striped table-dark
                    table-hover'
                csvDelimiter={this.props.csvDelimiter}
                {...this.props.data}
            />
        );
    }
}

pm.getData((err, data) => {
    ReactDOM.render(
        <App data={data} />,
        document.getElementById('root')
    );
});

</script>
`;


// Provide the props as per the documentation
// https://github.com/marudhupandiyang/react-csv-to-table
const reg = /,(?=([^"\\]*(\\.|"([^"\\]*\\.)*[^"\\]*"))*[^"]*$)/g;
let tableProps = {
    data: pm.response.text().replace(reg, "$"),
    csvDelimiter: '$',
    hasHeader: false
};

pm.visualizer.set(template, tableProps);
```

Bibliography

[1] Wikipedia contributors. Italian medicines agency — Wikipedia, the free encyclopedia, 2021. [Online; accessed 18-February-2021].

[2] Wikipedia. Agenzia italiana del farmaco — Wikipedia, l'enciclopedia libera, 2021. [Online; in data 18-febbraio-2021].

[3] AIFA. Trova farmaco — Agenzia Italiana del Farmaco, 2021. [Online; accessed 18-February-2021].

[4] Oracle. Java EE 8 — Oracle.

[5] WHO Collaborating Centre for Drug Statistics Methodology. Structure And Principles — whocc, 2018.

[6] WHO Collaborating Centre for Drug Statistics Methodology. ATC/DDD Index — whocc, 2020.

[7] sdrugs.com. The Anatomical Therapeutic Chemical (ATC) classification system — sdrugs.

[8] AIFA. Authorisation of medicinal products — Agenzia Italiana del Farmaco.

[9] JBoss. WildFly — JBoss.

[10] Michael Widenius. MariaDB Server: The open source relational database — MariaDB Foundation.

[11] Hans Dockter, Adam Murdoch, Szczepan Faber, Peter Niederwieser, Luke Daley, Rene Gröschke, Daz DeBoer, and Steve Appling. Gradle Build Tool — Gradle.

[12] Solomon Hykes. Docker: Empowering App Development for Developers — Docker, Inc.

[13] Abhinav Asthana, Ankit Sobti, and Abhijit Kane. Postman: The Collaboration Platform for API Development — Postman, Inc.