Master Thesis

Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Computer Science

Web Media Content Aggregator

Bc. Petr Cipra

Supervisor: Mgr. Miroslav Blaško, Ph.D.
Field of study: Open Informatics
Subfield: Software Engineering
May 2024

Acknowledgements

I would like to thank Mgr. Miroslav Blaško, Ph.D. for his helpful, valuable advice, consultation and above all his helpfulness and willingness while writing this thesis.

Declaration

Abstract

This thesis deals with designing and implementing a Semantic web compliant web application for extracting information of media content on the web, including its quality, available languages and source URLs, and presenting it to users. The application uses Java for the backend technology and ReactJS for the frontend technology. The resulting application provides the extracted data to a user through REST API enhanced to support Linked data principles.

Keywords: Media content, Semantic web, Web scraping, JSON-LD

Supervisor: Mgr. Miroslav Blaško, Ph.D.
Praha, Resslova 307/9, E-305

Abstrakt

Tato práce se zabývá návrhem a implementací webové aplikace kompatibilní se sémantickým webem pro získávání informací o mediálním obsahu na webu, včetně jeho kvality, dostupných jazyků a zdrojových adres URL, a k jejich prezentací uživatelům. Aplikace využívá Java pro backend technologii a ReactJS pro frontend technologii . Výsledná aplikace poskytuje extrahovaná data uživateli prostřednictvím rozhraní REST API rozšířeného o podporu principů Linked data.

Klíčová slova: Mediální obsah, Sémantický web, Extrakce dat z webu, JSON-LD

Překlad názvu: Agregátor webového mediálního obsahu

Contents

Figures

Tables

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Cipra Petr**	Personal ID number:	**483692**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Computer Science**		
Study program:	**Open Informatics**		
Specialisation:	**Software Engineering**		

II. Master’s thesis details

Master’s thesis title in English:

Web media content aggregator

Master’s thesis title in Czech:

Agregátor webového mediálního obsahu

Guidelines:

The aim of the project is to design and implement a web application in Java and React framework for aggregating web media content, i.e., series and movies, from at least 5 selected websites. The application will regularly extract information from the chosen websites and provide users with the ability to search, filter, and display this information in the user interface. The extracted information will be provided by the application for machine processing using Semantic Web technologies.
1) Familiarize with Semantic Web technologies for representation (OWL, RDF, JSON-LD), querying (SPARQL), and knowledge persistence (GraphDB).
2) Analyze relevant models for representation and tools for extracting information from websites.
3) Select at least 5 websites with relevant media content.
4) Analyze requirements for the application and define application scenarios.
5) Design and implement a prototype of the application.
6) Test the prototype on selected scenarios with at least 3 users.

Bibliography / sources:

1) Ledvinka, Martin, and Petr Křemen. "JOPA: accessing ontologies in an object-oriented way." International Conference on Enterprise Information Systems. Vol. 2. SciTePress, 2015.
2) Tomaszuk, Dominik, and David Hyland-Wood. "RDF 1.1: Knowledge representation and data integration language for the Web." Symmetry 12.1 (2020): 84.
3) JSON for Linking Data, online at https://json-ld.org/
4) Schema.org, online at https://schema.org/

Name and workplace of master’s thesis supervisor:

Mgr. Miroslav Blaško, Ph.D. skupina znalostních softwarových systémů

Name and workplace of second master’s thesis supervisor or consultant:

Date of master’s thesis assignment: **15.02.2024** Deadline for master's thesis submission: **24.05.2024**

Assignment valid until: **21.09.2025**

Mgr. Miroslav Blaško, Ph.D.		prof. Mgr. Petr Páta, Ph.D.
Supervisor’s signature	Head of department’s signature	Dean’s signature

III. Assignment receipt

The student acknowledges that the master’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master’s thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student’s signature

Chapter 1

Introduction

Today, there are many websites that provide media content, such as TV shows, TV series, movies or documentaries. Websites such as imdb.org or csfd.cz aggregate media content in a single place, but their purpose is to provide information about the movies or TV shows themselves, not on what website or in what format or language they are actually available to be watched and even whether they require a subscription to a service.

There exist standards for representing such data, most notably schema.org, which is used by most websites today due to it being used in search engines. For obtaining the needed information there exist many services, applications and libraries that are made to be used for this exact purpose.

The goal of this project is to create a web application that provides information about available TV shows, TV series, movies and documentaries from various websites, such as their genre, actors, media qualities, audio languages and where to watch them. The user of the application should be able to search and filter the provided information and possibly do more actions, such as enabling notifications for when an episode of a TV series is released.

Furthermore, the whole process of obtaining the needed information should be automated as much as possible, and the processed information by the application should be machine readable. For the purpose of machine readability Linked data technologies were chosen.

1.1 Document structure

Since there are many media content websites available, five major websites are selected. In the first chapter, these websites are analyzed in terms of the media content they provide and how to obtain the information about it. Also, it should shed some light on what is needed in order to implement it all.

In the second chapter, an analysis of available services, software and libraries takes place to get to know some tools which may be used when implementing the application.

In the third chapter I discuss a possible data model that may be used to represent the extracted data and what data model the selected websites themselves publicly present.

Finally, the design of the application is developed to closely define what should be implemented and how it could be done.

Chapter 2

Background

In this chapter I will provide some background information and introduce terms and technologies that will be used in this project and document.

2.1 Semantic web

Semantic web[1] is an extension of the World Wide Web which provides applications with metadata that can be more easily parsed and interpreted. This allows computers to make meaningful interpretations of the data, similarly to the way humans process information.

In the semantic web ecosystem there are many technologies that may be used for representing or storing data. In the following sections I will mention those that may be used in this project.

2.2 RDF

RDF[2] stands for Resource Description Framework, it is used for describing information about resources. Resources may be anything, such as documents, websites, people, abstract concepts, etc. Particularly, RDF may also be used for publishing and interlinking data on the Web.

The RDF data model is made of statements represented as triples, each consisting of subject, predicate and object. Subject and object are resources and predicate represents the relationship between them. A predicate is also called a property.

RDF may also use a schema that is used to specify the vocabulary for the data. Each schema may provide classes for resources (i.e. subjects and objects) and properties for predicates. For example, one such vocabulary is schema.org.

2.3 JSON-LD

JSON-LD[3] is one of the serialization formats that may be used for RDF. It is based on JSON[4] and is specifically used to serialize Linked Data. Linked Data[5] are used to link some data to some other, related, data.

2.4 SPARQL

SPARQL[6] is the standard query language for databases that store RDF triples. In its syntax it is similar to SQL, a query language that is used for relational databases. SPARQL was designed to be used for Linked Data on the Semantic web.

2.5 GraphDB

GraphDB[7] is a graph database that supports RDF and SPARQL. A graph database[8] consists of graphs which are made of nodes, edges and properties. RDF triples may be represented as a graph, each triple as two nodes connected by an edge.

Chapter 3

Selected websites

This section specifies what websites were selected for this project and what exactly is to be extracted from them.

3.1 TV Nova

TV Nova is a Czech TV channel. It has many sub-channels, each targeting different types of audiences, such as Nova Cinema, Nova Fun, Nova Action, and more.[9] Its biggest sources of media content are Voyo,[10] which is a subscription-based service providing a large selection of media content for a price, and the main website of TV Nova[11] that provides more limited content but for free. There are also non-free videos present on the main website of TV Nova but they are just redirects to the Voyo website. In conclusion, since most of the media content is on these two websites, we select both.

3.1.1 Data to extract

The goal is to extract the following data from both the Nova Voyo website and the main TV Nova website:

- Information about all TV shows, movies, and documentaries.
- Information about all episodes (and seasons) of TV shows, movies and documentaries.
- Information about available media sources for all episodes, movies and documentaries.

3.2 iPrima

iPrima is a Czech TV channel. Same as TV Nova, it has many sub-channels, each targeting different types of audience, such as Prima ZOOM, Prima Cool, Prima Action and more.[12] Its biggest sources of media content are Prima+,[13] which is a subscription-based service providing a large selection of media content for a price, and the website of Prima ZOOM[14] that provides some documentaries for free. However, some documentaries are redirected to the Prima+ website.

3.2.1 Data to extract

The goal is to extract the following data from both the Prima+ website and the Prima ZOOM website:

- Information about all TV shows, movies and documentaries.
- Information about all episodes (and seasons) of TV shows, movies and documentaries.
- Information about available media sources for all episodes, movies and documentaries.

3.3 Česká televize

Česká televize is a public television broadcaster in the Czech Republic. It has many sub-channels that provide different types of content. On the web, there are sub-websites for these sub-channels - ČT24, Sport, iVysílání, Déčko, Art, Edu. However, in the present all of the media content is broadcasted from the iVysílání website.[15] All videos are free to watch, but may not always be available, some cannot be played due to licensing restrictions and some are available only for a limited time.

3.3.1 Data to extract

The goal is to extract the following data from the iVysílání website:

- Information about all TV shows, movies and documentaries.
- Information about all episodes (and seasons) of TV shows, movies and documentaries.
- Information about available media sources for all episodes, movies and documentaries.

3.4 TV Markíza

TV Markíza is a Slovak TV channel. Its main media-providing websites are the main website of TV Markíza[16] and Markiza Voyo.[17]

The websites are very similar to those of TV Nova. Creating and managing extractors for both of these websites would be more time consuming, therefore only the extractor for Markiza SK will be considered.

3.4.1 Data to extract

The goal is to extract the following data from the Markiza SK website:

- Information about all TV shows.
- Information about all episodes (and seasons) of TV shows.

- Information about available media sources for all episodes.

3.5 TV JOJ

TV JOJ is a Slovak TV channel. They have multiple websites that provide media content, mainly JOJ Videoportál[18] and JOJ Play.[19] Since the JOJ Play website provides the most media content it will be selected. The JOJ Videoportál website will not be considered in the scope of this project because it would increase the time required to implement all its sub-websites.

3.5.1 Data to extract

The goal is to extract the following data from the JOJ Play website:

- Information about all TV shows, movies and documentaries.
- Information about all episodes (and seasons) of TV shows, movies and documentaries.
- Information about available media sources for all episodes, movies and documentaries.

Chapter 4

Analysis

This chapter covers all of the analysis in order to decide what would be necessary for the project to work properly, how the selected websites are implemented, how to extract data from them and what to use for the extraction.

4.1 Used terms

In the rest of this document I will refer to TV shows, TV series, movies, and documentaries as programs (programmes)[20] to group them and to keep the text shorter, if not writing specifically about only one of them. All are captured in the meaning of the word program (programme) since they are broadcasted, however not in the television but on the internet.

4.2 Existing websites

For Netflix exclusively there exists a website called uNoGS.com[21] that aggregates information about what TV show or movie is available in what country. Another existing website is called Trakt,[22] it targets more services than just Netflix and even contains some popular Czech TV shows. However, the advanced filter is included only in the VIP features, which are not free.

Both of these websites present additional information, such as the number of series, the number of episodes in them, actors, genres and more. But they are lacking in information like what video qualities are available and, particularly for episodes of Czech TV shows, there is no direct link to where to watch them.

4.3 Selected websites

This section dives into the analysis of the selected websites to determine what tools or libraries would be required to implement the extraction process for them.

The following analysis describes the extraction processes as of 20. 05. 2024. The websites may change at any time and thus some processes may be different after this date.

4.3.1 TV Nova - Nova Voyo

Nova Voyo is a paid service that requires a user account and an active subscription to watch the available content. Therefore, the information about the media itself, such as quality or audio languages, will be unavailable without an active account. However, the list of TV shows and episodes is freely available.

Programs

Nova Voyo does not provide a single page that displays all available programs therefore we must obtain it another way. It splits the programs into TV shows, TV series, Movies, Sports, and Kids.[10] Except for Sports each of them also has a picker for genre. That is a lot of pages to scrape. Luckily, the website itself uses an API when loading more items on a page, for example when viewing some genre of available TV shows.

An API call consists of an HTTP GET request to the endpoint `https://voyo.nova.cz/api/v1`, where in the URL path we specify what type of information we want to obtain and in the query arguments we specify the filtering of that information. The response is an HTML content.

For example the URL `https://voyo.nova.cz/api/v1/shows/genres?category=voyo-3&genre_id=16&sort=title__asc&limit=24&page=2&pageId=16` obtains the second page of Krimi TV shows sorted by title in the ascending order. The meaning of the URL query arguments:

- `category` - Describes what kind of content it is: voyo-3 = TV series, voyo-4 = TV shows, voyo-5 = Movies, voyo-6 = Kids.
- `genre_id` - We can ignore this argument since we want to obtain all items in no genre in particular. Omitting this argument causes no issues.
- `sort` - Can be kept to at least have the items somewhat ordered.
- `limit` - Specifies how many items per page we should obtain, it can be an arbitrary positive integer.
- `page` - Specifies the offset of the data, so for `limit=24` and `page=0` the request contains data from index from `page*limit` (inclusive) to `(page+1)*limit` (exclusive).
- `pageId` - An important argument and without it the request returns an error. From observation alone it is paired to the category argument. Each of the mentioned categories has its pageId: 17 = TV shows, 16 = TV series, 18 = Movies, 20 = Kids.

The returned HTML content consists of regular div elements with an anchor (link) and a title of the program. We can extract this information by parsing the HTML and selecting the elements with CSS selectors. More precisely, each program item can be obtained by the selector `.c-video-box`, inside the item the title by `.title` (and getting the inner text) and the link to the program by `.title > a` (and getting its href attribute). Each program also has a program ID, which we will actually need later, it can be obtained from the div element's `data-resource` attribute value and by stripping the show. prefix.

Episodes

To obtain episodes of a TV series, we can reuse the API. The URL path for this is `show/content` and its arguments are:

- `showId` - The program ID we obtained when getting all programs.
- `type` - For TV shows always `episodes`, since we want to obtain episodes.
- `season` - The ID of a season. This is not a simple number, such as 1 for season one, but a numerical ID which we first need to obtain somehow.
- `orderDirection` - The ordering of items, either `asc` (ascending order) or `desc` (descending order).
- `offset` - The offset of items. The items returned are from index from `offset` (inclusive) to `offset+count` (exclusive).
- `count` - The number of items.
- **url** - The URL path of the program.

The returned content is also HTML. The CSS selectors to use:

- Episode item: `article`.
- Inside the episode item - URL: `.title > a`.
- Inside the episode item - Title: inner text of `.title > a`.

As mentioned above, we also require a season ID, not just a simple number. The ID can be obtained by getting the HTML content of a program's detail. More particularly, by calling the API with `page/detail-url` as the path and the following arguments:

- `layout_parts`

  - The layout of the detail, always `40-10`.
- `url` - The URL path of the program.

In the returned HTML content, we can extract the seasons from the dropdown menu, again by CSS selectors:

- Season item: `#episodesDropdown + .dropdown-menu .dropdown-item`.

- In a season item - Season ID: The value of attribute `data-season-id`.

By getting all the seasons ID and then looping through all offsets of episodes till the returned HTML content is empty (or if the number of items does not equal the specified limit), we are able to obtain all the episodes of a TV series. For movies and documents it is simpler since they do not have episodes. Therefore no episode loop is required and we can return the program as is.

The episodes have their episode number in the title and the season o them is already known when obtaining them. Therefore, there is no need for backpropagation of information from the episode page itself.

Media sources

The video is displayed by an iframe having CSS selector `.js-detail-player .iframe-wrap iframe`. This iframe points to a page with a video player whose settings are directly specified in the HTML in a script element.

The script element contains a JavaScript code but there is actually no need to parse it as JavaScript. First, we find the text `Player.init` and then find the first opening curly brace. From this position we find the closing curly brace related to the opening curly brace we found earlier, i.e. by counting opening and closing braces and ignoring them in strings. This way between the curly braces we actually obtain a JSON string, which we can parse by a JSON parser.

In this JSON there is an object called `tracks` in which there are the media sources specified. It contains the URL of the source, its language and whether it is protected by DRM and its "DRM token" that is later sent to the license server.

However, this whole process actually requires the script to be authenticated first. For this we need a user account with a subscription plan active. Then, before the first request to the episode's or movie's URL, we must authenticate by doing the authentication process.

Authentication process

The following steps are needed to successfully log in to the account:

- POST request to `https://voyo.nova.cz/prihlaseni`
  - Body (application/x-www-form-urlencoded):
    - `email` - The user's email
    - `password` - The user's password
    - `login` - The string Přihlásit
    - `_do` - The string content186-loginForm-form-submit
  - Headers:
    - `Referrer` - The string "`https://voyo.nova.cz/prihlaseni`"
  - Result:
    - Should redirect to the URL `https://voyo.nova.cz/muj-profil` with valid credentials.
    - We must save the received `PHPSESSID` cookie.

For DRM-protected content a device token cookie is required, otherwise the sources won't be in the HTML content. We can find this device token in the cookie `votoken`, that is returned in the response when logging in. We can reuse this device token for later since each authentication would create a new one and the number of active devices is limited to 5.[23]

4.3.2 TV Nova - Main website

Programs

In the case of the main TV Nova website, it is actually simpler. All the programs exist on a single page: `https://tv.nova.cz/porady` Also, they are already in the HTML content, so there is no need to call any API.

To extract them, we can once again use CSS selectors:

- Each program item: `:not(.tab-content) > .c-show-wrapper > .c-show`.

- Inside the program item - URL: The value of attribute `href`.
- Inside the program item - Title: The inner text of element `.title`.

Episodes

The episodes exist either on the Celé díly page (`videa/cele-dily`) or on the Reprízy page (`videa/reprizy`) in the case of some TV series. The items on these pages are either static, i.e. there are less than 6 items, or loaded dynamically by JavaScript, i.e. by doing an HTTP request. In both cases, the HTML content is structurally the same.

For the dynamic case, we have to send an HTTP request to the URL `https://tv.nova.cz/api/v1/mixed/more` This URL must have the `content` URL query argument that can be obtained only from a button for loading more items. This button is present only in the case there are more than 5 items in total. Therefore, we first send an HTTP request to obtain the HTML content of the page with static items, i.e. `videa/cele-dily` or `videa/reprizy`, if it exists. The load more button can be obtained by CSS selector `.js-article-load-more .c-button` and the URL is in the value of its attribute data-href. From this URL we can then extract the value of the `content` argument.

Further, we then use the value of the content argument in the following URL `https://tv.nova.cz/api/v1/mixed/more?page=0&offset=OFFSET&content=CONTENT`, where:

- `page` - The page number, must be present, but is actually useless, therefore can be the constant `0`.
- `offset` - The offset of items, may be negative or positive.
- `content` - The content ID whose items to obtain, i.e. the value of the `content` argument.

Some episodes have the episode number in their titles, some do not, because either the episodes are not being numbered, or they are identified by a date and time. In either case we have to use backpropagation of data from JSON-LD on an episode's page.

Media sources

There is an iframe on the video's page that embeds the video player with the video. It can be selected by the CSS selector `iframe[data-video-id]`. Its `data-src` attribute then points to the URL in whose HTML content there is the information about media sources. In the content we have to find a script element that contains the string `player:`, then get the following opening curly brace, get the respective closing curly brace and extract the content between those curly braces. This content is actually a JSON string and when parsed, on the path `lib.source.sources` there is the information about media sources. Each object contains the URL, the type (HLS/DASH) and DRM protection, if present.

4.3.3 iPrima - Prima+

Programs

The programs are separated into categories of TV shows, Movies and Kids. Category page displays so called strips that contain the program items. Each strip has its unique ID using which we can obtain all its items without actually visiting the page itself. For this we may use the API that the website also uses.

Since I don't know any way of listing all strip IDs, I needed to obtain some manually to actually get the items. From purely observing the website, reading through the source code and analyzing its networking (HTTP requests and responses), I found the following strips to include the most items without actually using tens of them:

- `8ab51da8-1890-4e78-8770-cbee59a3976a` - Seriály
- `1d0e2451-bcfa-4ecc-a9d7-5d062ad9bf1c` - Seriály (Nejnovější)
- `82bee2e2-32ef-4323-ab1e-5f973bf5f0a6` - Pořady z TV
- `8138baa8-c933-4015-b7ea-17ac7a679da4` - Filmy (Doporučené)
- `3a2c25d8-4384-4945-ba37-ead972fb216d` - Filmy (Nejnovější)
- `7d92a9fa-a958-4d62-9ae9-2e2726c5a348` - Filmy (Nejsledovanější)

The Prima API uses JSON-RPC[24] under the hood, its endpoint is `https://gateway-api.prod.iprima.cz/json-rpc/` JSON-RPC request is just an HTTP request with specific headers and JSON content. The headers must include a request ID (`id`), the JSON-RPC version (`jsonrpc`) and the method (`method`). The ID may be constant, i.e. `1`. for our purposes and the version used by Prima is `2.0`. In terms of parameters Prima requires to have at least the `params.profileId` parameter set, for our case we can use `null`. Other parameters rely on what request is being sent. All responses are in the JSON format.

For obtaining the programs, the required parameters are:

- `method` - The string `strip.strip.bulkItems.vdm`
- `params`
  - `deviceType` - The string WEB.
  - `stripIds` - Array of the strip IDs.
  - `limit` - The number of items per page.
  - `filter` - Additional filter for filtering the items.

Important parts of the response has the following format:

- result.data
  - `recommId` - The recommendation ID, for `profileId=null` it should be the same ID for all requests.
  - `isNextItems` - true or false whether there are more items.
  - `items` - The items themselves.
    - `id` - The ID of the program
    - `title` - The title of the program
    - `type` - movie for movies and `series` for TV series.
    - `additionals.webUrl` - The URL of the program

We can then loop each strip till `isNextItems` is `false`. In this way we obtain all the programs.

Episodes

From the section above we know there are only two types of programs - movies and TV series. For movies we do not consider episodes. For TV series we use the API again.

Firstly, we are actually required to be authenticated at this point. Therefore, we must login before making any other requests. See Authentication process for more information.

Secondly, we obtain the HTML content of the program's page. Prima uses the Nuxt.js framework,[25] so on each page that uses it, there is a script element with the ID `__NUXT_DATA__` with JSON-like content, but with a more special format based on object substitution. How the format is parsed may be obtained from the source code Prima uses (`https://dwl2jqo5jww9m.cloudfront.net/_nuxt/entry.8c888bb8.js` Search for `"__NUXT_DATA__"` to see how it works. The code is minimized. Or see it in the source code of the application.), it is actually a little cumbersome process. However, in the end we are able to parse the content and obtain a JSON that contains all the information we need.

The JSON has objects in the root object, only those objects that have the property `title` are useful to us. We obtain the first such object and from `title.id` obtain the program ID.

Using the program ID we can then get all the seasons by sending the following RPC request of method `vdm.frontend.season.list.hbbtv` and the following parameters:

- `_accessToken` - The access token of a user.
- `id` - The program ID.
- `pager`
  - `limit` - The number of items, we can use the constant `999`.
  - `offset` - The offset of items, we can use the constant `0`.

Important parts of the response are in the format:

- `result.data`
  - `id` - The season ID.
  - `seasonNumber` - The season number.

With the seasons obtained, we can continue obtaining episodes for each of them. This can be accomplished by sending the following RPC request with method vdm.frontend.episodes.list.hbbtv and the following parameters:

- `_accessToken` - The access token of a user.
- `id` - The season ID.
- `pager`
  - `limit` - The number of items, we can use the constant `999`.
  - `offset` - The offset of items, we can use the constant `0`.
- `orderding`
  - `field` - The string `episodeNumber`
  - `direction` - The string `desc`

Important parts of the response are in the format:

- `result.data.episodes`
  - `title` - The title of the episode.
  - `additionals`
    - `webUrl` - The URL of the episode.
    - `pisodeNumber` - The episode number.

Media sources

For obtaining the media sources we have to be authenticated. We first obtain the HTML content of the movie or episode page. Again, in the content there is Nuxt data we have to parse. From it we get the first object that has the property `content` and from it we obtain the video play ID by getting the value of `content.additionals.videoPlayId`. Then we send an HTTP GET request to `https://api.play-backend.iprima.cz/api/v1/products/play/ids-PLAY_ID`, where `PLAY_ID` is the play ID. From the JSON response we can extract all sources by looping all parent objects and from each of them getting the array `streamInfos`. Each of the items in this array contains information about a single source, such as its URL, audio language, type (HLS/DASH) and information about the DRM protection, if present.

Authentication process

Prima uses OAuth2[26] to authenticate its users. The login process is as follows:

1. Send HTTP GET request to `https://auth.iprima.cz/oauth2/login`:
   a. Obtain the value of the CSRF token, it is in an input with the name `_csrf_token`.
2. Send HTTP POST request to `https://auth.iprima.cz/oauth2/login`:
   a. With valid credentials this will redirect to the profile selection page. Here we may select any profile, so we choose the first one. There should always be at least one profile available.
3. Send HTTP GET request to `https://auth.iprima.cz/user/profile-select-perform/%%7Bprofile_id%7Ds?continueUrl=/user/login`, where `PROFILE_ID` is the ID of the previously chosen profile.
   a. With a valid profile ID this will redirect to a URL that in its query arguments has the `code` argument. We save its value for the next step.
4. Send HTTP POST request to `https://auth.iprima.cz/oauth2/token`:
   a. Body (application/x-www-form-urlencoded):
      (i) `scope=openid+email+profile+phone+address+offline_access`
      (ii) `client_id=prima_sso`
      (iii) `grant_type=authorization_code`
      (iv) `code=CODE` (the code from the previous step)
      (v) `redirect_uri=https://auth.iprima.cz/sso/auth-check`
   b. The response is JSON with `access_token` and `refresh_token` properties. We need just the access token and the whole response string.

5. Send HTTP GET request to `https://auth.iprima.cz/oauth2/authorize?response_type=token_code&client_id=sso_token&token=TOKEN`, where `TOKEN` is Base64-encoded string of the response from the previous step. The JSON being encoded must contain the access token and the refresh token.
   a. The response will contain JSON with `code` property. We save this value for the next step.
6. Send HTTP GET request to `https://www.iprima.cz/sso/login?auth_token_code=CODE`, where `CODE` is the code from the previous step.
   a. If the response status code is `302`, we have been successfully logged in.

4.3.4 iPrima - Prima ZOOM

Programs

The API for Prima+ cannot be used in this case, since it is solely for Prima+. However, programs can be obtained by sending an HTTP GET request to `https://prima.iprima.cz/iprima-api/ListWithFilter/TYPE/Content?filter=all&channel_restriction=zoom`, where `TYPE` is either `Series` for TV series or `Movies` for movies. The response is a JSON object with property `content` that has HTML content of the program items. The information may be extracted by using these CSS selectors:

- Program item - `.component--scope--cinematography > a`.
- Inside the program item - URL - The value of attribute `href`.
- Inside the program item - Title - The value of attribute `title`.
- Inside the program item - ID - Parse the value of attribute `data-item-json` as JSON and get the value of the integer property `id`.

Episodes

First, we send an HTTP GET request to the program's URL. There we have to find a content ID for the program. This can be done by looking for a script element that contains an instantiation of `InfiniteCarousel` object, where in the constructor arguments there is the ID (this can be done using a regular expression), or by looking for a script element that contains a `dataLayer.push` method call, where in its object argument there is the ID. Each of these methods have also a different so-called `snippetType`, `videos-episode` for the first method and `programme_episodes` for the second one. The ID may actually be multiple IDs separated by comma, but that is fine.

After obtaining the ID, we send an HTTP GET request to `https://zoom.iprima.cz/_snippet/TYPE/COUNT/OFFSET/PROGRAM_IDS`, where:

- `TYPE` - The snippet type.
- `COUNT` - The number of items.
- `OFFSET` - The offset of items.
- `PROGRAM_IDS` - The extracted ID.

By looping and changing the offset till the response is not empty, we can obtain all the episodes.

`Media sources`

For getting the media sources we need a so-called `productId`. There are few methods how to obtain it (at least one should always work):

1. There is an iframe that embeds the video player. It can be obtained by CSS selector `iframe.video-embed`. In its `src` attribute there is an URL argument of name `id`, that is the `productId`.
2. Look up all script elements of type text/javascript, in one of them there should be a variable called `productId`, its value is the `productId`.
3. Look up all script elements that are not of type text/javascript, in one of them there should be a variable called `videos`, its value is the `productId`.

After obtaining the productId, we send an HTTP GET request to `https://api.play-backend.iprima.cz/api/v1/products/play/ids-PRODUCT_ID`,where `PRODUCT_ID` is the `productId`. For this to work for all availablemedia we have to be authenticated. In the JSON content of the response of this request there is an array `streamInfos` where each of its children objects represents a single media source, for which there is information about its URL and audio language.

4.3.5 Česká Televize - iVysílání

API

The iVysílání website uses a GraphQL API. This kind of API works on the principle of sending a POST request to the API endpoint with a body that contains the operation name of what we want to get and some additional arguments, all in the JSON format. The response of such a request is also in the JSON format. The endpoint of this API for iVysílání website is `https://api.ceskatelevize.cz/graphql/` We will use this API to obtain all the wanted information.

Programs

The iVysílání website shows programs based on a category, each of which has an internal ID. There are these categories available (the internal ID in the parenthesis): Seriály (3976), Filmy (3947), Dokumenty (4003), Zpravodajství (4124), Sport (4142), Zábava (4068), Historie (4079), Pro děti (4118), Kultura (4029), Rady a recepty (4055), Společnost (4093), Příroda (4106), Spiritualita (4191).

For each category we then send a GraphQL request with these arguments:

- operationName - The string GetCategoryById
- query - The GraphQL query GetCategoryById (See the section GraphQL queries, subsection GetCategoryById)
- variables
  - `categoryId` - The internal ID of the category

  - `length` - The number of items per page (there is a maximum limit of 40)
  - `offset` - The offset of the items
  - `order` - The string `asc`
  - `orderBy` - The string `alphabet`

The response is in the this format:

- `data.showFindByGenre`
  - `totalCount` - The total number of items
  - `items` - The items themselves
    - `id` - The internal ID of the program
    - `slug` - The URL of the program
    - `title` - The title of the program

We can iteratively increase the offset till it reaches the total number of items. This way we obtain all the available programs from the category. Then just do the whole procedure for each of the categories.

Episodes

To obtain episodes of a TV series, we must first obtain its IDEC, which is effectively just an internal ID. It may be obtained from the source code of the program's page.

The iVysílání website uses Next.js,[27] therefore there is a script tag with `id=__NEXT_DATA__`, whose content may be parsed with a JSON parser. That will be useful in extracting the seasons of a TV series, however the easiest way to obtain just the IDEC is to search for it using the following regular expression: `"idec":"(?<idec>[^"]+)"`, the IDEC is then in the named group `idec`.

To obtain the seasons of a TV series, we must first parse the content of the Next.js data, as stated in the previous paragraph. The format is as follows:

- `props.pageProps.data.show.seasons`
    - child collection
        - `id` - The ID of the season

Finally, if the program is a movie, we define that it has just a single season of ID `null`.

After we obtain the seasons, we may use the API again to obtain the episodes. For each of the seasons, we send the following GraphQL request:

- `operationName` - The string `GetEpisodes`
- `query` - The GraphQL query `GetEpisodes` (See the section GraphQL queries, subsection GetEpisodes)
- `variables`
    - `idec` - The internal ID of the program
    - `limit` - The number of items per page (there is a maximum limit of 40)
    - `offset` - The offset of the items
    - `orderBy` - The string `oldest`
    - `seasonId` - The season ID

The response is in the this format:

- `data.episodesPreviewFind`
    - `totalCount` - The total number of items
    - `items` - The items themselves
        - `id` - The internal ID of the episode
        - `title` - The title of the program
        - `playable` - Whether the episode can be played. This may be false when the episode or movie is not already or yet available or there are some licensing restrictions.

In the response there is no URL of the episode, but it may be obtained by concatenating the program's URL and the ID of the episode: `PROGRAM_URL + ID + "/"`.

Again, as with obtaining the programs, we iteratively increase the offset till we reach the total count.

Media sources

Given a URL of an episode, a movie, a documentary or a video, we first have to extract its IDEC. This may be done the same way as extracting the IDEC of a program when obtaining its episodes. Then we use the VOD API v1.

The VOD API is a simple API where we just send a GET request and receive a response with information about the available sources of the video.

Given a video with its IDEC, we send a GET request to `https://api.ceskatelevize.cz/video/v1/playlist-vod/v1/stream-data/media/external/IDEC`, where `IDEC` is the IDEC of the video. The response is in the following format:

- `streams`
  - `url` - The URL of the video stream, mostly to a MPD file.
  - `subtitles` (optional)
    - `language` - The language of the subtitles
    - child collection
      - `files`
        - child collection
          - `url` - The URL of the subtitles file, often a VTT file.

GraphQL queries

This section explicitly defines all GraphQL queries used when extracting the wanted data.

GetCategoryById.

```
query GetCategoryById(
    $limit: PaginationAmount!,
    $offset: Int!, $categoryId: String!,
    $order: OrderByDirection,
    $orderBy: CategoryOrderByType
) {
    showFindByGenre(
        limit: $limit
        offset: $offset
        categoryId: $categoryId
        order: $order
        orderBy: $orderBy
    ) {
        items {
            ...ShowCardFragment
            __typename
        }
        totalCount
        __typename
    }
}

fragment ShowCardFragment on Show {
    id
    slug
    title
    __typename
}
```

GetEpisodes.

```
query GetEpisodes(
    $idec: String!,
    $seasonId: String,
    $limit: PaginationAmount!,
    $offset: Int!,
    $orderBy: EpisodeOrderByType!,
    $keyword: String
) {
    episodesPreviewFind(
        idec: $idec
        seasonId: $seasonId
        limit: $limit
        offset: $offset
        orderBy: $orderBy
        keyword: $keyword
    ) {
        totalCount
        items {
            ...VideoCardFragment
            __typename
        }
        __typename
    }
}

fragment VideoCardFragment on EpisodePreview {
    id
    playable
    title
    __typename
}
```

4.3.6 TV Markíza - Markíza SK

The markiza.sk website is very similar to the main website of TV Nova, however there are some small differences.

Programs

All programs are available on a single page `https://www.markiza.sk/relacie` and their information exists in the HTML content statically.

To extract the programs, we may use CSS selectors:

- Each program item: `:not(.tab-content) > .c-show-wrapper > .c-show`.
- Inside the program item - URL: The value of attribute `href`.
- Inside the program item - Title: The inner text of element `h3`.

Episodes

The episodes exist on the Celé epizody page (`videa/cele-epizody`). The items on this page are either static, i.e. there are less than 6 items, or loaded dynamically by JavaScript, i.e. by doing an HTTP request, if there are more episodes in total. In both cases, the HTML content is structurally the same.

For the dynamic case, we have to send an HTTP request to the URL `https://www.markiza.sk/api/v1/mixed/more?page=0&offset=OFFSET&content=CONTENT` This URL must have the `content` URL query argument that can be obtained only from a button for loading more items. This button is present only in the case there are more than 5 items in total. Therefore, we first send an HTTP request to obtain the HTML content of the page with static items, i.e. `videa/cele-epizody`. The load more button can be obtained by CSS selector `.js-article-load-more .c-button` and the URL is in the value of its attribute `data-href`. From this URL we can then extract the value of the `content` argument.

We then use the value of the content argument in the following URL `https://www.markiza.sk/api/v1/mixed/more?page=0&offset=OFFSET&content=CONTENT`, where:

- `page` - The page number, must be present, but is actually useless, therefore can be the constant `0`.
- `offset` - The offset of items, may be negative or positive.
- `content` - The content ID whose items to obtain, i.e. the value of the `content` argument.

Some episodes have the episode number in their titles, some do not, because either the episodes are not being numbered, or they are identified by date and time. In either case we have to use backpropagation of data from JSON-LD of media sources back to their respective episodes.

Media sources

There is an iframe on the video's page that embeds the video player with the video. It can be selected by the CSS selector `iframe[data-video-id]`.

Its src attribute then points to the URL in whose HTML content there is the information about media sources. In the content we have to find a script element that contains the string `player:`, then get the following opening curly brace, get the respective closing curly brace and extract the content between those curly braces. This content is actually a JSON string and when parsed, on the path `lib.source.sources` there is the information about media sources. Each object contains the URL, the audio language and DRM protection details, if any protection is present.

4.3.7 TV JOJ - JOJ Play

JOJ Play uses Firebase (Firestore) as its database, thus communication with the Firestore API may be used. This way we can extract all required information using just a single method of communication.

Firebase

To communicate with the firebase database, we first need to open the so-called Firebase channel. It is a simple GET request to the endpoint `https://firestore.googleapis.com/google.firestore.v1.Firestore/Listen/channel` This request is open for 60 seconds and is then closed, if more communication is needed, the GET request may be repeated. Subsequently we send POST requests with specific content with what we want to obtain. In the response of the GET request there is the content of these POST requests. The POST request itself just returns a meta information about where in the GET response's content the data may be found.

A Firebase channel communication is made of regular requests and responses. However, there is also a concept of targets. Since there may be many concurrent requests to the singular endpoint, the responses must be able to convey what context we are currently in. This is done using so-called targets, where the basic operations are add and remove. To add a target we send a addTarget request, to remove a target we send a removeTarget request. The target may change at any time during the reading of responses from the Firebase channel. There are also so-called documents that, similar to targets, may change at any time during the response reading. Documents are not really targets, they are stored data records.

Each Firebase channel request is made of a URL, headers and a query.

Headers are used only when authenticating. The URL is made of the base URL `https://firestore.googleapis.com/google.firestore.v1.Firestore/Listen/channel` and its query arguments (various arguments are sent during various operations):

- `database` - The string `projects/tivio-production/databases/(default)`. Always present.
- `VER` - The version, currently the constant `8`. Always present.
- `gsessionid` - The Google session ID. Obtained during authentication. Present for authenticated requests.
- `SID` - The other session ID. Obtained during authentication. Present for authenticated requests.
- `RID` - Monotonically increasing integer value for the authentication request and regular requests. The string `rpc` for the opening request.
- `AID` - Monotonically increasing integer value. Present for the opening request and regular requests.
- `zx` - Random string of length `12`. Always present.
- `t` - The constant `1`. Always present.
- `CVER` - The client version, currently the constant `22`. Only sent during the authentication request.
- `X-HTTP-Session-Id` - The string `gsessionid`. Only sent during the authentication request.
- `$httpHeaders` - The headers in HTTP-like format. Only sent during the authentication request.
- `CI` - The constant `0`. Present only for the opening request.
- `TYPE` - The string `xmlhttp`. Present only for the opening request.

Each Firebase channel request is in the following format:

- `count` - The number of requests in the body.
- `ofs` - Monotonically increasing integer value.
- `reqN___data__` - The query string, where N is the index of the query.

There are two main types of query strings:

- addTarget
    - `database` - The string `projects/tivio-production/databases/(default)`.
    - `addTarget` - The object with target data
        - `targetId` - The target ID. Must be present.
        - `query.structuredQuery` - The object with the data of a structured query. Present during regular queries.
        - `query.parent` - The parent reference of the query. Present during regular queries.
        - `documents.documents` - The array of document references to be obtained. Only present during the document query.
- removeTarget
    - `database` - The string `projects/tivio-production/databases/(default)`.
    - `removeTarget` - The target ID

The structured query is the specific query of what we would like to obtain and is encoded into a JSON string. It consists of a parent and its data that may include the following:

- `from` - From what source to obtain the data.
- `where` - The filter criteria, either composite or simple. Composite Where clause is just a collection of simple or composite Where clauses and a logical operation between them, i.e. AND or OR. Simple Where clause may be either a field to constant value comparison, search in an array for a constant value and more.
- `orderBy` - The sort criteria, i.e. by what field to sort.
- `limit` - The maximum number of items to return.

After a POST request is sent, the server sends a response to the POST request with meta information in a specific format, but also sends the actual data to the open GET request.

The response to the POST request is in the following format: `LENGTH`
`n[1,EVENT_ID,7]`
`n`, where LENGTH is the length of the JSON array, i.e. from the character `[` to the character
`n`, and the `EVENT_ID` is the event ID which we need to get the actual data.

With the event ID from the POST response we may find the respective data in the response of the GET request. The format of a single response in the GET request's response is as follows: `LENGTH`
`n[EVENT_ID,[CONTENT]`, where `LENGTH` is the length of the JSON array, i.e. from the outer character `[` to the last character `]`, `EVENT_ID` is the event ID from the POST response and `CONTENT` is the actual content of the requested data.

Using the Firebase channel, open request, authentication request, regular requests, target changes, document changes and structured queries we are able to obtain all the required data we want.

Authentication process

To open the Firebase channel for JOJ Play we have to have an authentication token first. It is just a simple POST request to the URL `https://www.googleapis.com/identitytoolkit/v3/relyingparty/verifyPassword?key=AIzaSyB02udgMkNLADkLJ_w5YNBMR2VR1WHfusI` with this JSON content:

- `tenantId` - The string `XEpbY0V54AE34rF07dB2-i9m04`
- `email` - The account email address
- `password` - The account password
- `returnSecureToken` - The boolean value `true`

The response is in the following format:

- `idToken` - The authentication token

After obtaining the authentication token, we open a new session in the Firebase channel. We have to send an authentication request with the following:

- URL:
  - `https://firestore.googleapis.com/google.firestore.v1.Firestore/Listen/channel`
  - query arguments:
    - `database` - The string `projects/tivio-production/databases/(default)`.
    - `VER` - The constant `8`.
    - `CVER` - The constant `22`.
    - `RID` - The constant `0`.
    - `X-HTTP-Session-Id` - The string `gsessionid`.
    - `$httpHeaders` - The string `X-Goog-Api-Client:gl-js/fire/8.10.1Content-Type:text/plainX-Firebase-GMPID:1:1006888934987:web:60408b1ce75bfb5f8cb7ceAuthorization:BearerID_TOKEN`, where `ID_TOKEN` is the authentication token.
    - `zx` - A random string of length `12`.
    - `t` - The constant `1`.
- Structured query
  - `parent` - `projects/tivio-production/databases/(default)/documents`
  - `from` - Collection `videos`
  - `orderBy` - By field `__name__`, direction Ascending
  - `limit` - `1`

The response's body of this request is as follows: `[0,["c","SID","",8,12,30000]]]`, where `SID` is the Session ID we need. And from the response's header `x-http-session-id` we obtain the Google session ID. With the Session ID and the Google Session ID we may now send regular requests.

Programs

Internally all programs are either a TV show or a movie. To obtain programs we have to send a two regular requests:

- For TV shows:
  - We send the following query:

    - parent - `projects/tivio-production/databases/(default)/documents/organizations/dEpbY0V54AE34rF07dB2`
    - from - `tags`
  - And filter the items by the following:
    - An item is considered a TV show only if it has the value tvProfiSerialId in its metadata. I.e.: There exists such a collection in fields.metadata.arrayValue.values that has the string tvProfiSerialId on the path `mapValue.fields.key.stringValue`.
  - And then map them as follows:
    - URL - `https://play.joj.sk/series/SLUG`, where `SLUG` is the string after the last forward slash in the `name` property.
    - Title - The value of the `fields.name` property.
    - Reference - The value of the `name` property.

- For movies:
  - We send the following query:
    - parent - `projects/tivio-production/databases/(default)/documents`
    - from - `videos`
    - where (AND relation)
      - `contentType = FILM`
      - `externals.tvProfiType IN (movie, film, dokument)`
  - And filter the items by the following:
    - An item is considered a movie only if
      - it has the property `fields.urlName`, and
      - it does not have the property `fields.originalVideoRef` or have the property `fields.originalVideoRef.nullValue`, and
      - it does not have property `fields.externals.mapValue.fields.tvProfiSeriesName`.
  - And then map them as follows:
    - URL - `https://play.joj.sk/videos/SLUG`, where `SLUG` is the string after the last forward slash in the `name` property.
    - Title - The value of the `fields.name` property.
    - Reference - The value of the `name` property.

Episodes

Given a program and its reference:

- If the program is a movie:
    - Get its document using the document query.
    - And construct the result:
        - URL - `https://play.joj.sk/player/SLUG`, where `SLUG` is the string after the last forward slash in the `name` property.
        - Title - The program's title.
- If the program is a TV show:
    - Get its document using the document query and extract seasons from it. They are present on the path `fields.metadata.arrayValue.values` in a child collection which has the value of `mapValue.fields.type.stringValue` equal to `AVAILABLE_SEASONS`.
    - Loop through the seasons
        - Get the episodes of the season using a regular request:
            - parent - `projects/tivio-production/databases/(default)/documents`
            - from - `videos`
            - where (AND relation)
                - `tags ARRAY_CONTAINS_ANY REF`, where `REF` is the program's reference
                - `publishedStatus = PUBLISHED`
                - `transcodingStatus = ENCODING_DONE`
                - `seasonNumber = SEASON_NUMBER`, where `SEASON_NUMBER` is the current season's number
        - And for each episode construct the result:
            - URL - `https://play.joj.sk/player/SLUG`, where `SLUG` is the string after the last forward slash in the `name` property.
            - Title - The value of the `fields.name` property.

Media sources

Given a URL of an episode or a movie, we have to obtain its document. First, get its slug as the string after the `player/` string in its URL path. Then obtain its document using a regular query:

- parent - `projects/tivio-production/databases/(default)/documents`
- from - `videos`

- where
  - `urlName.sk ARRAY_CONTAINS SLUG`, where `SLUG` is the slug of the episode or movie.
- orderBy
  - By field `__name__`, direction Ascending
- limit - 2

From the document obtain the video ID as the value after the last forward slash in the name property.

Then send a POST request to the URL `https://europe-west3-tivio-production.cloudfunctions.net/getSourceUrl` with the following JSON content:

- `data`
  - `id` - The video ID
  - `documentType` - The string video
  - capabilities (JSON array)
    - For DASH:
      - `codec` - The string `h264`
      - `protocol` - The string `DASH`
      - `encryption` - The string `none`
    - For HLS:
      - `codec` - The string `h264`
      - `protocol` - The string `HLS`
      - `encryption` - The string `none`

If the video is monetized an Authentication HTTP header must be specified with the following value: `Bearer ID_TOKEN`, where `ID_TOKEN` is the authentication token from the authentication process.

In the response to this request the URL of the media source can be found in the property `result.url`.

Conclusion

From the analyses above, it can be concluded that the application requires at least these functionalities to support the selected websites:

1. Functionality to send HTTPS requests with the support of GET and POST methods and cookie management.
2. Functionality to parse an HTML content, select elements in it by CSS selectors (or at least by traversing the elements in some other way) and the textual content of an element and the values of its attributes.
3. Functionality to parse a JSON content, traverse its objects, arrays, properties and items.
4. Functionality to do basic string operations and use regular expressions.

The following solution may be chosen to fulfill those requirements:

1. Java 11 has added java.net.HttpClient that can send HTTPS requests.
2. Any HTML parsing library may be used, such as Jsoup.[28]
3. Any JSON parsing library may be used, such as JSON-java.[29]
4. Java can do basic string functions and supports regular expressions.

4.4 Analysis of web scraping tools

In the previous chapter websites were selected and analyzed to provide the information about what is required to implement the application. This chapter focuses on finding out whether there already exist solutions for extracting information from the web that may be used for this project. A search was conducted to find them. It targeted third-party services, software or similar tools to be used externally by the application and Java libraries to be used internally in the source code of the application.

4.4.1 Services

A service in this context is any third-party service that provides web-scraping abilities, such as sending HTTP requests, extracting HTML content, etc. Ideally, a single service should be used, therefore it should provide all of the required functionality.

The search for services was conducted on search engines, such as Google, targeting any web-scraping service available, no special keywords were used to filter them by specific functions.

List of relevant services that were found:

- Apify - Limited free tier
- AvesAPI - Limited free tier
- Bright Data - No free tier
- Dexi - Free trial
- Diffbot - No free trial
- Grepsr - No free trial
- Import.io - No free trial
- Mozenda - Free trial
- Nanonets Web Scraping Tool - Free is just a tool for converting website to text
- OctoParse - Free trial
- Oxylabs Scraper API - Free trial
- ParseHub - Limited free credit, user-friendly interface for selecting what to scrape
- Scrape-It.Cloud - Limited free credit, API
- Scrape.do - Free trial only
- Scraper API - Limited free credit, API
- Scrapestack - Limited free credit
- ScrapingBee - Limited free credit

- ScrapingBot - Limited free credit
- Scrapingdog - Limited free credit
- Smartproxy - No free option
- Web Scraper - Free option is for local-only use
- ZenRows - Free trial, API

Web interface extractors			
Name	**Free trialFree credit**	**Paid**	**Supported websites**
Apify	Yes (Free monthly credit)	Yes ($49/m - Starter)	Any website
AvesAPI	Yes (First time credit)	Yes ($50/m - Starter)	Any website
Bright Data	Yes (Free trial)	Yes ($3.40/CPM - cost per mil.)	Any website
Dexi	Yes (Free trial)	Unspecified	Any website
Diffbot	No	Yes ($899/m (Plus)	Any website
Grepsr	No	Yes ($299/m - Special offer, then $599/m)	Any website
Import.io	No	Yes ($399/m)	Any website
Mozenda	Yes (Free trial)	Unspecified	Any website
Nanonets Web Scraping Tool	Yes (Free tool)	Yes ($499/m - Pro)	Any website
OctoParse	Yes (Free plan)	Yes ($75/m - Standard)	Any website
Oxylabs Scraper API	Yes (Free trial)	Yes ($50/m - Micro)	Any website
ParseHub	Yes (Free plan)	Yes ($189/m - Standard)	Any website
Scrape-It.Cloud	No	Yes ($29/m - Individual)	Any website
Scrape.do	No	Yes ($29/m - Hobby)	Any website
Scraper API	No	Yes ($49/m - Hobby)	Any website
Scrapestack	Yes (Free plan)	Yes ($19.99/m - Basic)	Any website
ScrapingBee	No	Yes ($49/m - Freelance)	Any website
ScrapingBot	Yes (Free plan)	Yes (€39/m - Freelance)	Any website
Scrapingdog	No	Yes ($30/m - Lite)	Any website
Smartproxy	Yes (Free plan)	Yes ($2/1k requests - 25k requests plan)	Any website
Web Scraper	Yes (Browser extension)	Yes ($50/m - Project)	Any website
ZenRows	No	Yes ($49/m - Developer)	Any website

Table 4.1: Summary of analysis of services

4.4.2 Software

A software in this context is any application that may be run alongside the application as an external process that provides web-scraping functionality. Ideally, only a single software should be used to not have to include many software alongside the application.

The search for software was conducted on search engines, such as Google, targeting any software that provides web-scraping abilities. No special keywords were used to further filter them by specific functionality.

List of relevant software that were found:

- Apache Nutch - Java-based web crawler
- Heritrix - Java-based web crawler
- Norconex HTTP Collector - Web crawler
- StormCrawler - Java-based web crawler
- WebSPHINX - Customizable web crawler

Software extractors				
Name	**Programming language**	**Supports command line**	**Documentation available**	**Output format**
Apache Nutch	Java	Yes	Yes	Segments and Database*
Heritrix	Java	No (Web interface)	Yes	In web Interface
Norconex HTTP Collector	Java	Yes	Yes	XML files
StormCrawler	Java	Yes	Yes	Database
WebSPHINX	Java	No (GUI)	Yes	HTML files

Table 4.2: Summary of analysis of software

* Apache Nutch creates so-called segments that contain the actual content of the pages. It also uses a database to store the URLs.

4.4.3 Libraries

A library in this context is any Java library that may be used in the source code of the application to provide web-scraping functionality. The functionalities of a single library is often limited to a single function, such as HTML parsing or JSON parsing, therefore multiple libraries may be used.

The search for software was conducted on search engines, such as Google, targeting any Java libraries that are often used by others for web-scraping. Keywords like HTML parsing, JSON parsing, web scraping, etc. were used.

List of relevant libraries that were found:

- Gecco - Java-based web crawler library
- Htmleasy - Java-based HTML parsing library
- HtmlUnit - Java-based framework, GUI-less browser
- Jaunt - Java-based web scraping library
- Jauntium - Java-based library for web scraping
- Jsoup - Java-based HTML parser
- Selenium - Java-based library for extracting data and automation
- Web-Harvest - Java-based web extraction library
- WebMagic - Java-based web crawler framework

Library extractors		
Name	**Static/Dynamic**	**Documentation available**
Gecco	Static (HTML Parser)	No (Only Quick start)
Htmleasy	Static (HTML Parser)	No (Only Usage with examples)
HtmlUnit	Dynamic (Headless browser)	Yes
Jaunt	Dynamic (Headless browser, based on Jauntium)	Yes
Jauntium	Dynamic (Headless browser)	Yes
Jsoup	Static (HTML Parser)	Yes
Selenium	Dynamic (Headless browser)	Yes
Web-Harvest	Static (HTML Parser)	No (Only Usage with examples)
WebMagic	Static (HTML Parser)	Yes

Table 4.3: Summary of analysis of libraries

4.4.4 Conclusion

There exist many extractors or web scrapers for extracting information from the web. Some provide extracting as a service through a web interface or by an API, some provide extraction as an application with the user defining a configuration file that is then used by the application, and the others are targeted for developers in the form of libraries to be used in a custom software.

No service was found that is free and since the services are priced per request, it would cost money to use them. Therefore they are not suitable for this project. Moreover, introducing additional requests might also cause unnecessary delays.

Many extractors exist in the form of an application. They have the advantage over services that they are local, therefore no additional requests to a third party API is required. They may even provide multiple required functionalities at once. However, none of the software found supports functionalities such as communication with a Firebase database that would simplify some things. Therefore more than one would have to be chosen to do all of the required functionality. This would mean including multiple software with the application. Moreover, working with additional processes is more resource intensive and requires creation of configuration files or passage of arguments to the binary file of the software.

Libraries often provide a single functionality, such as just HTML parsing, therefore multiple libraries might have to be used by the application. They are used directly in the source code, removing the need to communicate with an external process or handling external files, etc.

In conclusion, libraries provide a lightweight internal solution to the required functionalities. No need to handle any external APIs using HTTP requests, no need to handle communication between processes or creation and deletion of external files (temporarily) created by an external application. Therefore using libraries is the preferred method and will be chosen for this project.

4.5 Data models for media content

In this chapter we look at what data the selected websites publicly present on different pages, such as the page of a program or an episode. From this, we may be able to decide what data model to use in the application.

4.5.1 Selected websites

On each of the selected websites various media-related pages were examined to look for data they provide about either a program, an episode or an actor (if available).

4.5.2 TV Nova

Both the main website and Voyo website use Linking data in the JSON-LD format with schema from Schema.org.

TV Show

```
{
    "@context": "http://schema.org",
    "@type": "TVSeries",
    "name": "TV_SHOW_NAME",
    "headline": "TV_SHOW_SHORT_DESCRIPTION",
    "url": "TV_SHOW_URL",
    "thumbnailUrl": "THUMBNAIL_IMAGE_URL",
    "image": {
        "@type": "ImageObject",
        "url": "IMAGE_URL",
        "width": IMAGE_WIDTH,
        "height": IMAGE_HEIGHT
    },
    "numberOfEpisodes": NUMBER_OF_EPISODES,
    "description": "TV_SHOW_DESCRIPTION",
    "countryOfOrigin": {
        "@type": "Country",
        "name": "COUNTRY_NAME"
    },
    "actor": [
        {
            "@type": "Person",
            "name": "ACTOR_NAME"
        },
        // ... Other actors
    ]
}
```

Episode

```
{
    "@context": "http://schema.org",
    "@type": "TVEpisode",
    "name": "EPISODE_NAME",
    "description": "EPISODE_DESCRIPTION",
    "url": "EPISODE_URL",
    "thumbnailUrl": "EPISODE_THUMBNAIL_IMAGE_URL",
    "image": {
        "@type": "ImageObject",
        "url": "IMAGE_URL",
        "width": IMAGE_WIDTH,
        "height'': IMAGE_HEIGHT
    },
    "episodeNumber": EPISODE_NUMBER,
    "partOfSeason": {
        "@type": "TVSeason",
        "name": "SEASON_NAME",
        "seasonNumber": SEASON_NUMBER,
        "numberOfEpisodes": SEASON_TOTAL_NUMBER_OF_EPISODES
    },
    "partOfSeries": {
        "@type": "TVSeries",
        "name": "TV_SHOW_NAME",
        "url": "TV_SHOW_URL",
        "numberOfEpisodes": NUMBER_OF_EPISODES,
        "thumbnailUrl": "THUMBNAIL_IMAGE_URL",
        "image": {
            "@type": "ImageObject",
            "url": "IMAGE_URL",
                "width": IMAGE_WIDTH,
                "height": IMAGE_HEIGHT
            },
            "description": "TV_SHOW_DESCRIPTION",
        },
        "video": {
            "@type": "VideoObject",
            "name": "EPISODE_NAME",
            "description": "EPISODE_DESCRIPTION",
            "thumbnailUrl": "EPISODE_THUMBNAIL_IMAGE_URL",
            "uploadDate": "UPLOAD_DATE_ISO_FORMAT",
            "url": "EPISODE_URL",
            "width": EPISODE_THUMBNAIL_IMAGE_WIDTH,
            "height": EPISODE_THUMBNAIL_IMAGE_HEIGHT,
            "duration": "VIDEO_DURATION_ISO_FORMAT",
            "embedUrl": "VIDEO_EMBED_URL"
        }
}
```

Actor

```
{
    "@context": "http://schema.org",
    "@type": "Person",
    "name": "ACTOR_NAME",
    "description": "ACTOR_DESCRIPTION",
    "image": {
        "@type": "ImageObject",
        "url": "ACTOR_IMAGE_URL",
        "width": ACTOR_IMAGE_WIDTH
    }
}
```

4.5.3 iPrima

Both Prima+ website and Prima ZOOM website use Linking data in the JSON-LD format with schema from Schema.org.

TV Show

```
{
    "@context": "https://schema.org",
    "@type": "TVSeries",
    "url": "TV_SHOW_URL",
    "name": "TV_SHOW_NAME",
    "description": "TV_SHOW_DESCRIPTION",
    "dateCreated": "TV_SHOW_YEAR",
    "genre": [
        "GENRE_NAME",
        // ... Other genres
    ],
    "image": "TV_SHOW_IMAGE_URL",
    "countryOfOrigin": [
        "COUNTRY_ABBREVIATION",
        // ... Other countries
    ],
    "containsSeason": [
        {
            "@type": "TVSeason",
            "name": "SEASON_NAME",
            "episode": [
                {
                    "@type": "TVEpisode",
                    "name": "EPISODE_NAME",
                    "datePublished": "EPISODE_PUBLISH_DATE",
                    "episodeNumber": EPISODE_NUMBER
                },
                // ... Other episodes
            ]
        },
        // ... Other seasons
    ]
}
```

Episode

```
{
    "@context": "https://schema.org",
    "@type": "TVEpisode",
    "url": "EPISODE_URL",
    "name": "EPISODE_NAME",
    "image": "EPISODE_IMAGE_URL",
    "description": "EPISODE_DESCRIPTION",
    "episodeNumber": EPISODE_NUMBER,
    "partOfSeason": {
        "@type": "TVSeason",
        "name": "SEASON_NAME"
    },
    "partOfSeries": {
        "@type": "TVSeries",
        "name": "TV_SHOW_NAME"
    },
    "datePublished": "EPISODE_PUBLISH_DATE",
    "genre": [
        "GENRE_NAME",
        // ... Other genres
    ],
    "countryOfOrigin": [
        "COUNTRY_ABBREVIATION",
        // ... Other countries
    ],
    "duration": "EPISODE_DURATION_ISO_FORMAT"
}
```

4.5.4 Other websites

It may be useful to also look to other websites than those that were selected. In this section a foreign websites will be analyzed to see how they publicly present the data about programs and episodes.

Netflix

Netflix also provides Linked Data in the source code of pages of TV shows and movies. The following paragraphs show the structure of the data. It can be seen that they are very similar to those on the Czech websites.

TV Show.

```
{
    "@context": "http://schema.org",
    "@type": "TVSeries",
    "url": "TV_SHOW_URL",
    "contentRating": "TV_SHOW_RATING",
    "name": "TV_SHOW_TITLE",
    "description": "TV_SHOW_DESCRIPTION",
    "genre": "TV_SHOW_GENRE",
    "image": "TV_SHOW_THUMBNAIL_IMAGE",
    "dateCreated": "TV_SHOW_CREATED_DATE",
    "actors": [
        {
            "@type": "Person",
            "name": "ACTOR_NAME"
        },
        // ... Other actors
    ],
    "creator": [
        {
            "@type": "Person",
            "name": "CREATOR_NAME"
        },
        // ... Other creators
    ],
    "numberOfSeasons": TV_SHOW_SEASONS_COUNT,
    "startDate": "TV_SHOW_START_DATE"
}
```

Movie.

```
{
    "@context": "http://schema.org",
    "@type": "Movie",
    "url": "MOVIE_URL",
    "contentRating": "CONTENT_RATING",
    "name": "MOVIE_TITLE",
    "description": "MOVIE_DESCRIPTION",
    "genre": "MOVIE_GENRE",
    "image": "MOVIE_THUMBNAIL_URL",
    "dateCreated": "MOVIE_CREATED_DATE",
    "actors": [
        {
            "@type": "Person",
            "name": "ACTOR_NAME"
        },
        // ... Other actors
    ],
    "director": [
        {
            "@type": "Person",
            "name": "DIRECTOR_NAME"
        },
        // ... Other directors
    ]
}
```

4.5.5 Schema.org

This section deals with schema of entities for media content. The structure and information were obtained from `https://schema.org/`. It does not list all possible properties but only those that were deemed important for this project.

TVSeries

Represents a TV show with seasons and episodes.

Property name	**Type**	**Description**
countryOfOrigin	Country	The country of origin of the series.
numberOfSeasons	Integer	The number of seasons in the series.
startDate	Date or DateTime	The start date and time of the series (in ISO 8601 date format).
endDate	Date or DateTime	The end date and time of the series (in ISO 8601 date format).
actor	Person or List of Person	An actor or actors in the series.
genre	Text or URL	Genre of the series.
image	ImageObject or URL	A thumbnail image of the series.
description	Text or TextObject	A description of the series.
name	Text	The title of the series.
url	URL	The URL of the series.

TVSeason

Represents a season of a TV show.

Property name	Type	Description
numberOfEpisodes	Integer	The number of episodes in the season.
seasonNumber	Integer or Text	Position of the season within an ordered group of seasons.
partOfSeries	CreativeWorkSeries	The series to which the season belongs.
startDate	Date or DateTime	The start date and time of the season (in ISO 8601 date format).
endDate	Date or DateTime	The end date and time of the season (in ISO 8601 date format).
image	ImageObject or URL	A thumbnail image of the season.
description	Text or TextObject	A description of the season.
name	Text	The title of the season.
url	URL	The URL of the season.

Episode

Represents an episode in a season of a TV show.

Property name	Type	Description
episodeNumber	Integer or Text	Position of the episode within an ordered group of episodes.
duration	Duration	The duration of the episode in ISO 8601 date format.
partOfSeason	CreativeWorkSeason	The season to which this episode belongs.
partOfSeries	CreativeWorkSeries	The series to which this episode belongs.
datePublished	Date or DateTime	Date of first broadcast/publication.
image	ImageObject or URL	A thumbnail image of the season.
description	Text or TextObject	A description of the season.
name	Text	The title of the season.
url	URL	The URL of the season.

Movie

Represents a movie. Differs from TV series in that there are no seasons.

Property name	Type	Description
countryOfOrigin	Country	The country of origin of the movie.
duration	Duration	The duration of the movie in ISO 8601 date format.
datePublished	Date or DateTime	Date of first broadcast/publication.
actor	Person or List of Person	An actor or actors in the movie.
genre	Text or URL	Genre of the movie.
image	ImageObject or URL	A thumbnail image of the movie.
description	Text or TextObject	A description of the movie.
name	Text	The title of the movie.
url	URL	The URL of the movie.

Person

In the context of a TV show or movie it represents an actor.

Property name	Type	Description
name	Text	The name of the actor.

4.5.6 Summary

The selected websites and major websites, such as Netflix, use JSON-LD with schema from Schema.org, therefore it should be sufficient to also use the schema from Schema.org in this application.

In the following sections all important properties that may be used by the application are presented.

TVSeries

Property name	Type	Description	Optional
countryOfOrigin	Country	The country of origin of the TV show.	Yes
numberOfSeasons	Integer	The number of seasons in the TV show.	No
startDate	Date or DateTime	The start date and time of the TV show (in ISO 8601 date format).	Yes
endDate	Date or DateTime	The end date and time of the TV show (in ISO 8601 date format).	Yes
actor	List of Actor	An actor or actors in the TV show.	Yes
genre	Text or URL	Genre of the TV show.	Yes
image	URL	A thumbnail image of the TV show.	No
description	Text	A description of the TV show.	Yes
name	Text	The title of the TV show.	No
url	URL	The URL of the TV show.	No
season	List of TVSeason	The list of available seasons of the TV show.	No

TVSeason

Property name	Type	Description	Optional
numberOfEpisodes	Integer	The number of episodes in the season.	No
seasonNumber	Integer or Text	Position of the season within an ordered group of seasons.	No
startDate	Date or Date-Time	The start date and time of the season (in ISO 8601 date format).	Yes
endDate	Date or Date-Time	The end date and time of the season (in ISO 8601 date format).	Yes
image	URL	A thumbnail image of the season.	Yes
description	Text	A description of the season.	Yes
name	Text	The title of the season.	No
url	URL	The URL of the season.	No

Movie

Property name	Type	Description	Optional
countryOfOrigin	Country	The country of origin of the movie.	Yes
duration	Duration	The duration of the movie in ISO 8601 date format.	Yes
datePublished	Date or Date-Time	Date of first broadcast/publication.	Yes
actor	List of Actor	An actor or actors in the movie.	Yes
genre	Text or URL	Genre of the movie.	Yes
image	URL	A thumbnail image of the movie.	Yes
description	Text	A description of the movie.	Yes
name	Text	The title of the movie.	No
url	URL	The URL of the movie.	No

Episode

Property name	Type	Description	Optional
episodeNumber	Integer or Text	Position of the episode within an ordered group of episodes.	No
duration	Duration	The duration of the episode in ISO 8601 date format.	Yes
season	TVSeason	The season to which this episode belongs.	No
series	TVShow	The series to which this episode belongs.	No
datePublished	Date or DateTime	Date of first broadcast/publication.	Yes
image	URL	A thumbnail image of the season.	Yes
description	Text	A description of the season.	Yes
name	Text	The title of the season.	No
url	URL	The URL of the season.	No

Person

Property name	Type	Description	Optional
name	Text	The name of the person.	No

Chapter 5

Application design

5.1 Software requirements

Software requirements are separated into functional and non-functional requirements in the following sections. Requirements, either functional or non-functional, are grouped by a specific category or area in the application. Each group has a unique identification in the form a prefix FR, for functional requirement, or NFR, for non-functional requirement, and a number of the group. Each individual requirement is then prioritized using the MoSCoW method.[30]

5.1.1 MoSCoW method

The MoSCow method is a prioritization technique used in project management or software development to provide a common understanding in terms of priority of each requirement. It consists of the following priorities:

- **M (Must have)**
  - Requirements with this priority are critical to the software. In the end, if the software is delivered without them, it may not be usable at all.

- **S (Should have)**
  - Requirements with this priority are important but not deemed necessary. The requirements may be of the same importance as the Must have ones, but are not time-critical.
- **C (Could have)**
  - Requirements with this priority are desirable but also not necessary. The software is fine to be delivered without them, but user experience may suffer due to the lack of them.
- **W (Won't have)**
  - Requirements with this priority are the least-critical ones. They may never be delivered in the software and are agreed to be not appropriate at the time of writing of the requirements.

5.1.2 Users

For the application there are three types of users - *casual (regular) user*, *API user* and *host user*. The casual user is a normal visitor of the frontend of the application, they browse the pages of the application to read the content. The API user is any entity that uses the API of the application, such that it calls the various endpoints and reads the responses. The host user is the user that provides the application as a self-hosted service for other types of users to use. The host user is also able to extend the application however they want, for example by adding support for another website that other host users do not provide.

5.1.3 Functional requirements

This section mentions all functional requirements that should be taken into account while implementing the application.

FR1: Frontend - Programs listing

- [M] **FR1.1:** The listing displays an item for each TV show and movie.
- [M] **FR1.2:** The listing displays items using pagination.

  - There exist many TV shows and movies. Loading them all on a single page will cause slow page loads. Thus, either using a normal pagination with pages, or infinite-scrolling, is a must to provide better user experience.

- [M] **FR1.3:** Item displays an image or a placeholder for each TV show and movie.
- [M] **FR1.4:** Item displays a title and type, either TV show or movie, for each TV show and movie.
- [M] **FR1.5:** The listing provides a search input.
- [M] **FR1.6:** The listing provides filter controls.
  - Filter controls may for example include genre selector, published year selector, and more.
- [S] **FR1.7:** The listing displays items in a more space-efficient manner.
  - For viewports where horizontal space is not a problem, such as the desktop viewport, the use of a grid or other form of container where there are multiple items on a single row, will result in better user experience.
- [C] **FR1.8:** Item displays additional information about the TV show or movie.
  - The presence of information such as the published year or genre may provide better ability to distinguish between multiple entities with the same name, or just provide better user experience.

FR2: Frontend - TV show detail

- [M] **FR2.1:** System displays the title of the TV show.
- [M] **FR2.2:** System displays the thumbnail or placeholder of the TV show.
- [M] **FR2.3:** System displays the description of the TV show.
- [M] **FR2.4:** System displays for TV shows their seasons and episodes.
- [S] **FR2.5:** System displays additional information about the TV show.
  - Additional information may include the published year, genre, actors and more.
- [C] **FR2.6:** System displays the latest episode as highlighted.

FR3: Frontend - Movie detail

- [M] **FR3.1:** System displays the title of the movie.
- [M] **FR3.2:** System displays the thumbnail or placeholder of the movie.
- [M] **FR3.3:** System displays the description of the movie.
- [M] **FR3.4:** System displays links to the actual movie page.
  - Links that redirect the user to the actual page on the website from which the movie was extracted. This page commonly contains a video player. However, since some movies may not be available for free, it might require authentication in the form of logging in using user credentials.
- [M] **FR3.5:** System displays available media source qualities.
  - For example, it may be displayed as a list of tags with contents like 720p, 1080p, etc.
- [S] **FR3.6:** System displays available subtitles.
  - For example, it may be displayed as a list of tags with contents like CS, SK, EN, etc.
- [S] **FR3.7:** System displays additional information about the movie.
  - Additional information may include the published year, genre, actors and more.

FR4: Frontend - Episode detail

- [M] **FR4.1:** System displays the title of the episode.
- [M] **FR4.2:** System displays the season and episode numberof the episode.
- [M] **FR4.3:** System displays links to the actual episode page.
  - Links that redirect the user to the actual page on the website from which the episode was extracted. This page commonly contains a video player. However, since some episodes may not be available for free, it might require authentication in the form of logging in using user credentials.
- [M] **FR4.4:** System displays available media source qualities.

  - For example, it may be displayed as a list of tags with contents like 720p, 1080p, etc.

- [S] **FR4.5:** System displays available subtitles.
  - For example, it may be displayed as a list of tags with contents like CS, SK, EN, etc.
- [S] **FR4.6:** System displays the description of the episode.
- [S] **FR4.7:** System displays additional information about the movie.
  - Additional information may include the release date, genre, actors and more.

FR5: Frontend - Registration

- [C] **FR5.1:** System displays registration form.
- [C] **FR5.2:** System displays error messages for invalid registration attempts.
- [C] **FR5.3:** System displays a success message for a successful registration attempt.
- [W] **FR5.4:** Users may register using a third-party account.
  - Such as Google, Apple, Facebook, etc.

FR6: Frontend - Login

- [C] **FR6.1:** System displays login form.
- [C] **FR6.2:** System displays error messages for failed login attempts.
- [C] **FR6.3:** For a successful login attempt the user is redirected to the account dashboard.
- [W] **FR6.4:** Users may login using a third-party account.
  - Such as Google, Apple, Facebook, etc.

FR7: Frontend - Account dashboard

- [C] **FR7.1:** Regular user and host user can change its password
- [C] **FR7.2:** System displays account preferences form.
- [C] **FR7.3:** System displays a logout button.
- [C] **FR7.4:** System allows changing of preferred TV shows and movies.
    - Preferred TV shows and movies will be displayed in the listings in the top positions.
- [C] **FR7.5:** System allows changing of email notifications.
    - Email notifications are sent in cases, such as when a new episode of a TV show is available.

FR8: API

- [M] **FR8.1:** The API provides information about extracted TV shows and movies.
- [M] **FR8.2:** The API provides information about extracted episodes of TV shows.
- [M] **FR8.3:** The API provides information about media sources of episodes.
    - Information such as quality, format, whether they are protected using DRM (Digital Rights Management), etc.
- [S] **FR8.4:** The API provides information about subtitles of episodes.
- [M] **FR8.5:** The API uses predefined schema for entities.
    - This schema uses common schema, such as from schema.org, for specific entities, such as TVSeries, Movie, etc. Each entity has its own properties which are then shown in the API.
- [M] **FR8.6:** The API uses REST-like URIs for entities to display information.
- [S] **FR8.7:** The API is functionally separated from the frontend.
    - If the frontend is not functional (offline), the API is still functional (online) and vice-versa.

FR9: Extractor

- [M] **FR9.1:** The extractor works independently of the frontend and the API.
- [M] **FR9.2:** The extractor crawls the selected websites for TV shows and movies.
- [M] **FR9.3:** The extractor crawls the TV shows for their seasons and episodes.
- [M] **FR9.4:** The extractor crawls the movies and episodes for information about available media sources.
- [M] **FR9.5:** The extractor periodically updates the extracted information.
    - The extractor runs periodically and either adds new entities and information, or updates an existing one.
- [C] **FR9.6:** The extractor does not remove previously present but now absent information.
    - Not removing currently absent information is better in terms of already existing URIs in the API not being removed, thus always existing. Marking the entity as removed should be sufficient. The downside may be that due to this approach the amount of data in the database only ever grows. Also, there may be an issue with normalization of data and merging of two or more entities that are semantically equivalent to each other.
- [M] **FR9.7:** The extractor is extensible for new websites.
    - Experienced users may be able to add a new website that will be then extracted using a module or plugin that is loaded upon starting the extractor. This should allow extraction from a simple website with static content to a more complex one that requires API calls for example.
- [C] **FR9.8:** The extractor notifies the other parts of the system of failure.
    - Since websites may change, that website extractor may stop working. In that case at least a notification that the extraction process failed for that website should be conveyed to the other parts of the system.
- [M] **FR9.9:** The extractor does not stop its work when a failure of extraction occurs.
    - When a specific extractor fails to extract information from its website, for example when extracting episodes from a TV show page, it should not stop processing other TV shows, movies or other websites.

FR10: Frontend - Movie detail, Episode detail, TV show detail

- [S] **FR10.1:** System contains JSON-LD snippet.
  - JSON-LD snippet is a code snippet in the script HTML tag that contains structured data for search engines, such as Google (for more information, see: `https://developers.google.com/search/docs/appearance/structured-data/intro-structured-data`)

5.1.4 Non-functional requirements

This section mentions all non-functional requirements that should be taken into account while implementing the application.

Accessibility

- [S] **NFR1:** The website is navigable using a keyboard.

Localization

- [M] **NFR2:** The website is localized in the English language.
- [S] **NFR3:** The website is localized in the Czech language.

Compatibility

- [M] **NFR4:** The website works on the latest versions of major browsers.
  - Latest versions of major browsers (as of Q1/2024) - Google Chrome (Windows), Mozilla Firefox (Windows), Google Chrome (Android), and optionally Safari (Mac OS) and Safari (iOS).
- [M] **NFR5:** The website displays correctly on devices with width 320px or bigger.

    - Most mobiles are of width 320px or bigger.[31]

Maintainability

- [M] **NFR6:** The frontend is tested using user testing.
    - Should be tested on at least 3 users.
- [S] **NFR7:** The application provides an extension mechanism to add extractions of new websites.

5.1.5 Model

The application should use various entities to provide the required functionality:

- For representing data there are TVShow, Movie, TVSeason and Episode. Each consists of important attributes that are specified in the summary of the analysis section. Since there are more than a few attributes, a builder may be used when constructing these entities. This should result in a cleaner approach because some attributes are optional or may not be available at all.
- For the extraction process itself there are Plugin, Extractor and Crawler. Plugin is conceptually any collection of Java classes that may be dynamically loaded at startup of the application and provide instances of Extractors. These Extractors are registered to an ExtractorRegistry and are later used by the Crawler to extract data from websites.
- For additional functionality there is a User entity that apart from storing the email address and password of an application's user, it also stores their preferences and most importantly notifications. These notifications are then used by a Notifier to notify the user to their email address about various events, such as a new episode of their subscribed TV show.

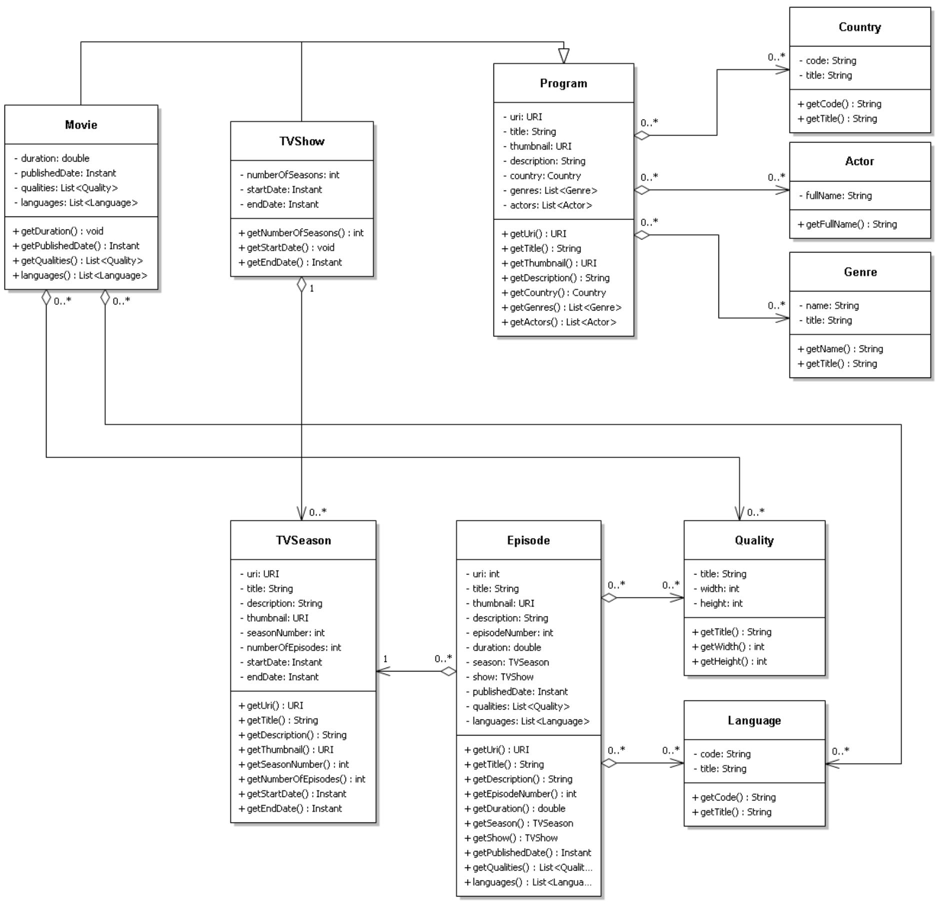


Figure 5.1: Data model diagram: Program, Movie, TVShow, TVSeason, Episode are from schema.org

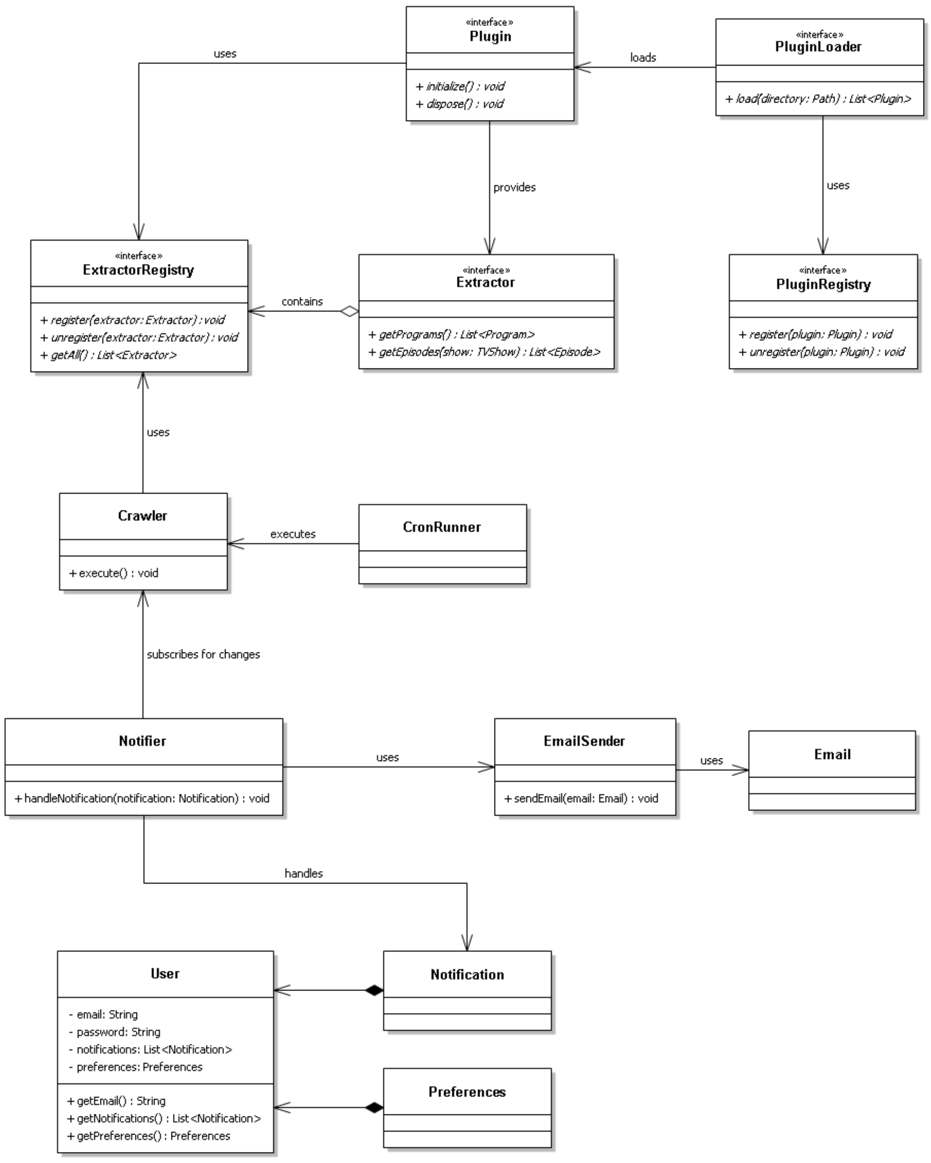

«interface»
Plugin
+ initialize() : void
+ dispose() : void
loads
«interface»
PluginLoader
+ load(directory: Path) : List<Plugin>
uses
provides
uses
«interface»
ExtractorRegistry
+ register(extractor: Extractor) : void
+ unregister(extractor: Extractor) : void
+ getAll() : List<Extractor>
contains
«interface»
Extractor
+ getPrograms() : List<Program>
+ getEpisodes(show: TVShow) : List<Episode>
«interface»
PluginRegistry
+ register(plugin: Plugin) : void
+ unregister(plugin: Plugin) : void
uses
Crawler
+ execute() : void
executes
CronRunner
subscribes for changes
Notifier
+ handleNotification(notification: Notification) : void
uses
EmailSender
+ sendEmail(email: Email) : void
uses
Email
handles
User
- email: String
- password: String
- notifications: List<Notification>
- preferences: Preferences
+ getEmail() : String
+ getNotifications() : List<Notification>
+ getPreferences() : Preferences
Notification
Preferences

Figure 5.2: Class model diagram

5.1.6 Use cases

In this section diagrams of use cases can be found. They represent common actions that may be carried out in the application either as a casual user or a host user.

Register a new account

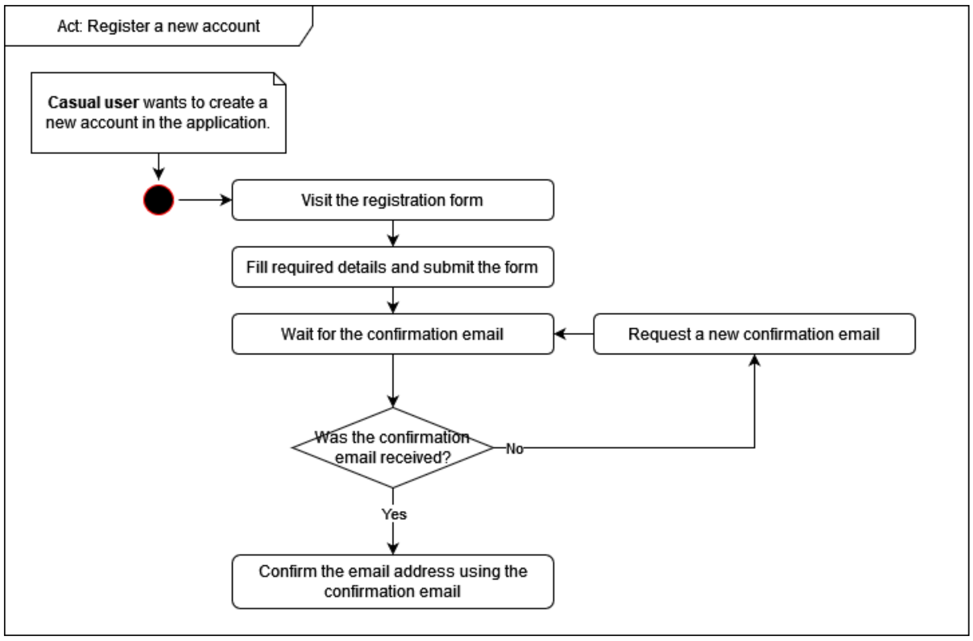

Act: Register a new account
Casual user wants to create a new account in the application.
Visit the registration form
Fill required details and submit the form
Wait for the confirmation email
Request a new confirmation email
Was the confirmation email received?
No
Yes
Confirm the email address using the confirmation email

Figure 5.3: Use case: Register a new account

Find a program by name

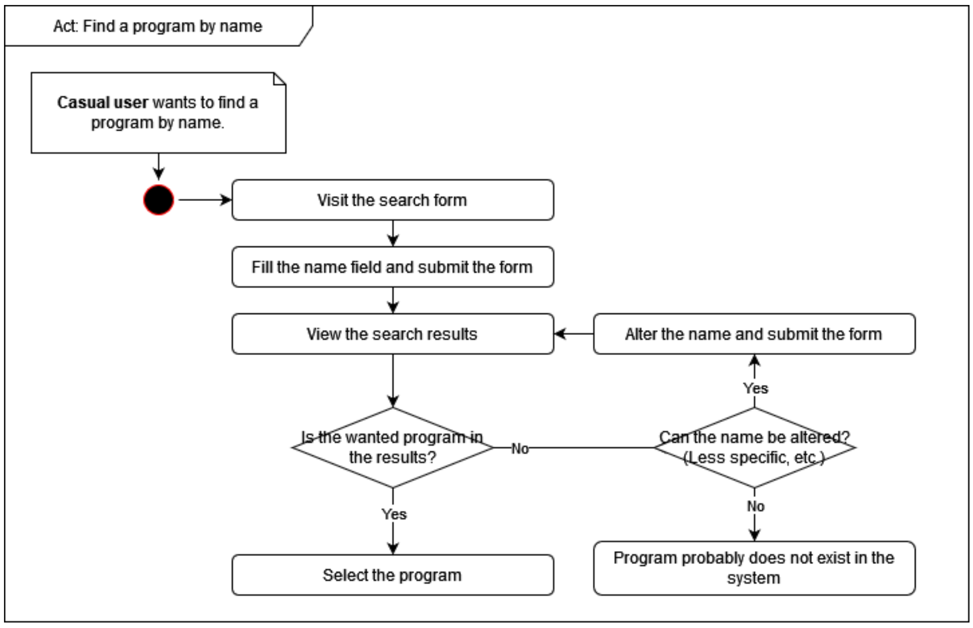

Act: Find a program by name
Casual user wants to find a program by name.
Visit the search form
Fill the name field and submit the form
View the search results
Alter the name and submit the form
Is the wanted program in the results?
No
Yes
Can the name be altered? (Less specific, etc.)
Yes
No
Select the program
Program probably does not exist in the system

Figure 5.4: Use case: Find a program by name

Find a program by properties

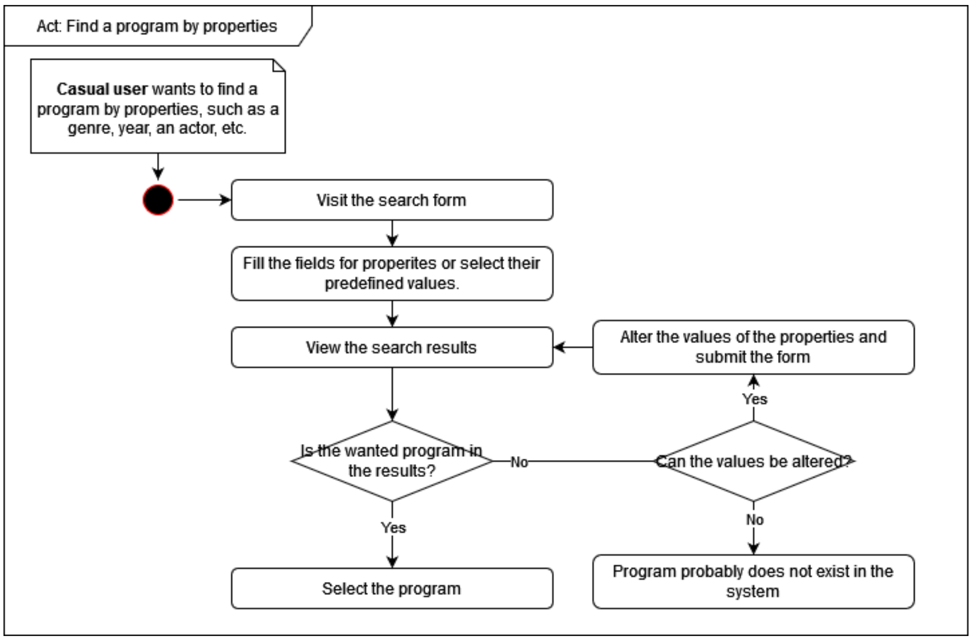

Act: Find a program by properties
Casual user wants to find a program by properties, such as a genre, year, an actor, etc.
Visit the search form
Fill the fields for properites or select their predefined values.
View the search results
Alter the values of the properties and submit the form
Yes
Is the wanted program in the results?
No
Can the values be altered?
Yes
No
Select the program
Program probably does not exist in the system

Figure 5.5: Use case: Find a program by properties

Show episodes of a TV series

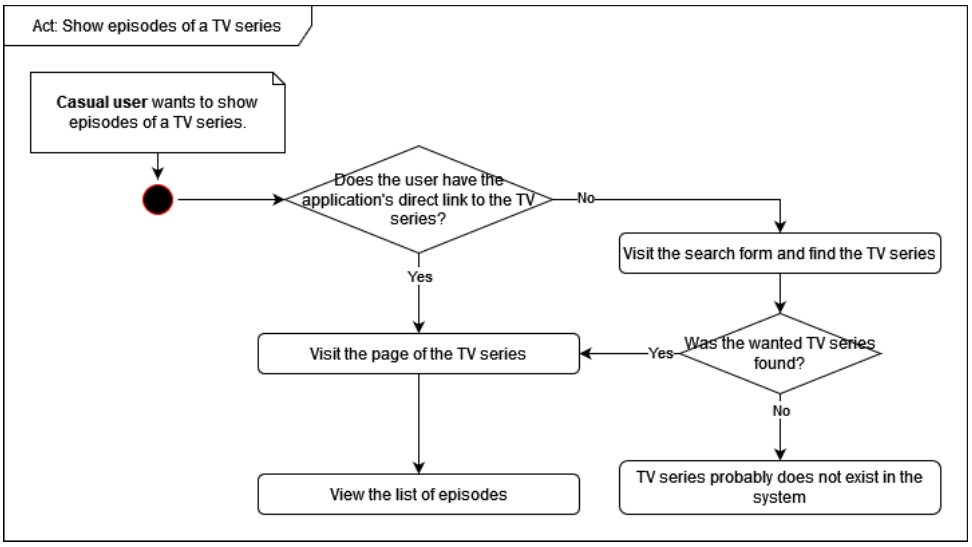

Act: Show episodes of a TV series
Casual user wants to show episodes of a TV series.
Does the user have the application's direct link to the TV series?
No
Visit the search form and find the TV series
Yes
Visit the page of the TV series
Yes
Was the wanted TV series found?
No
View the list of episodes
TV series probably does not exist in the system

Figure 5.6: Use case: Show episodes of a TV series

Show properties of an episode

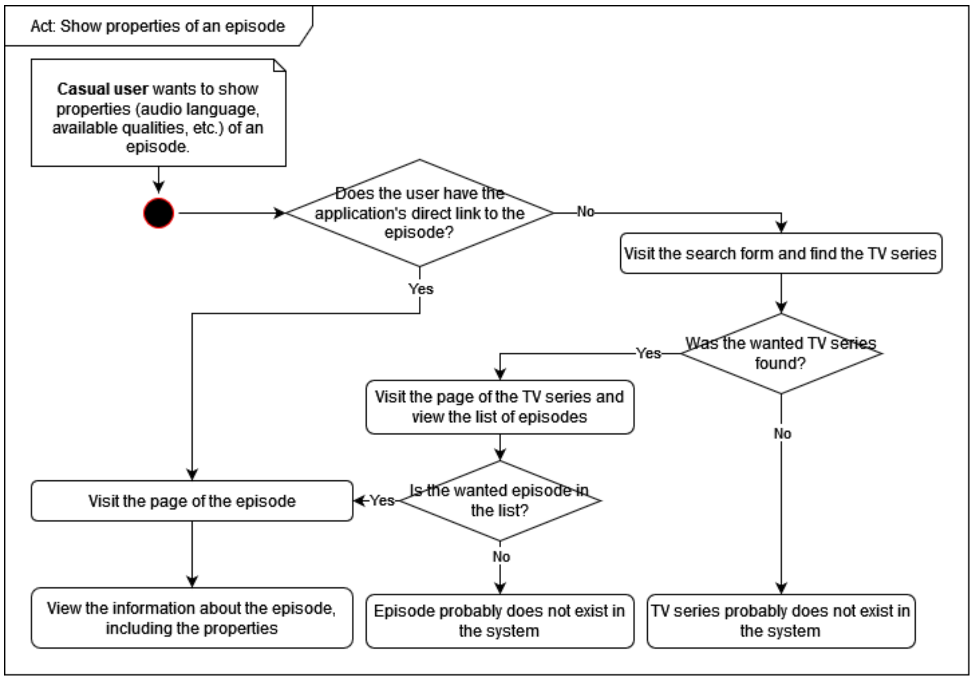

Act: Show properties of an episode
Casual user wants to show properties (audio language, available qualities, etc.) of an episode.
Does the user have the application's direct link to the episode?
No
Visit the search form and find the TV series
Yes
Was the wanted TV series found?
Yes
Visit the page of the TV series and view the list of episodes
No
Visit the page of the episode
Yes
Is the wanted episode in the list?
No
View the information about the episode, including the properties
Episode probably does not exist in the system
TV series probably does not exist in the system

Figure 5.7: Use case: Show properties of an episode

Visit the official webpage of a program

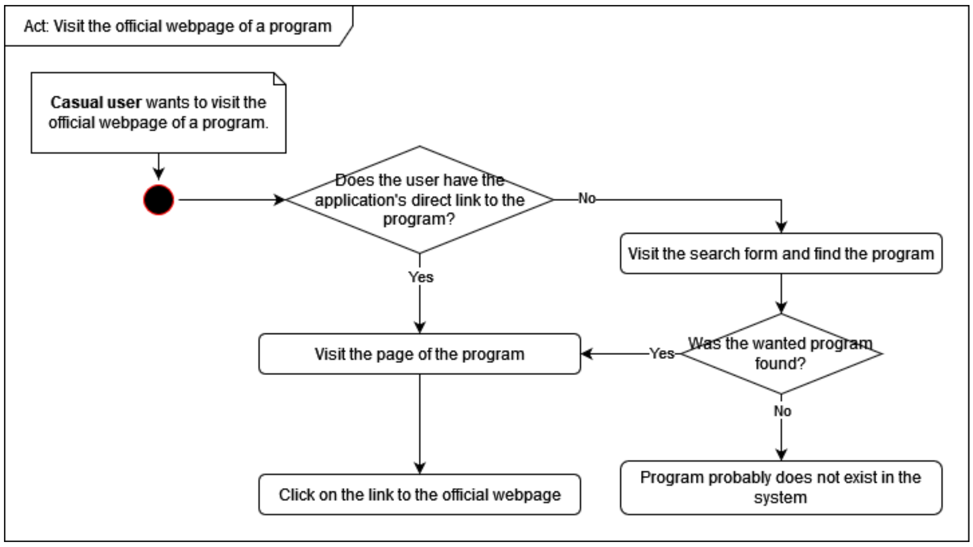

Act: Visit the official webpage of a program
Casual user wants to visit the official webpage of a program.
Does the user have the application's direct link to the program?
No
Visit the search form and find the program
Yes
Visit the page of the program
Yes
Was the wanted program found?
No
Click on the link to the official webpage
Program probably does not exist in the system

Figure 5.8: Use case: Visit the official webpage of a program

Enable email notifications for a TV series

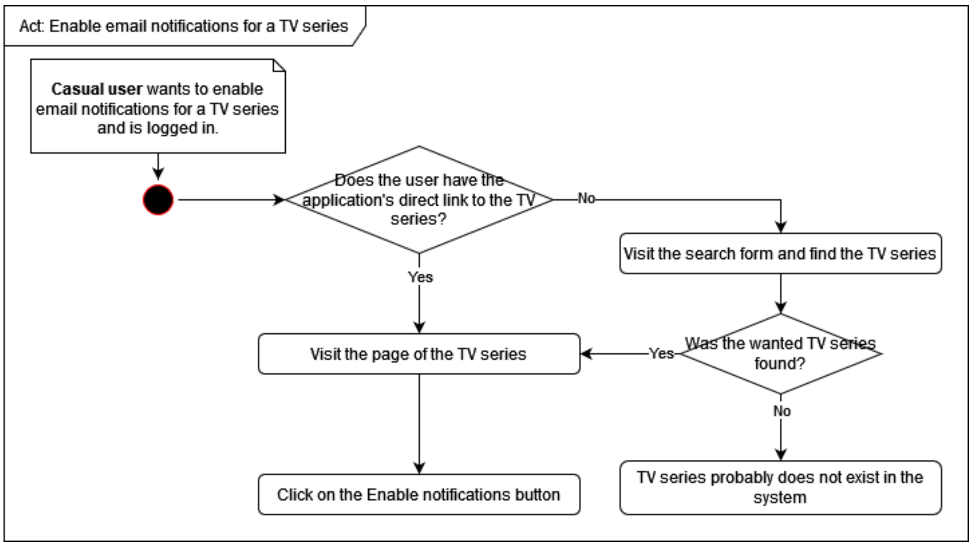

Act: Enable email notifications for a TV series
Casual user wants to enable email notifications for a TV series and is logged in.
Does the user have the application's direct link to the TV series?
No
Yes
Visit the search form and find the TV series
Was the wanted TV series found?
Yes
No
Visit the page of the TV series
Click on the Enable notifications button
TV series probably does not exist in the system

Figure 5.9: Use case: Enable email notifications for a TV series

Disable email notifications for a TV series

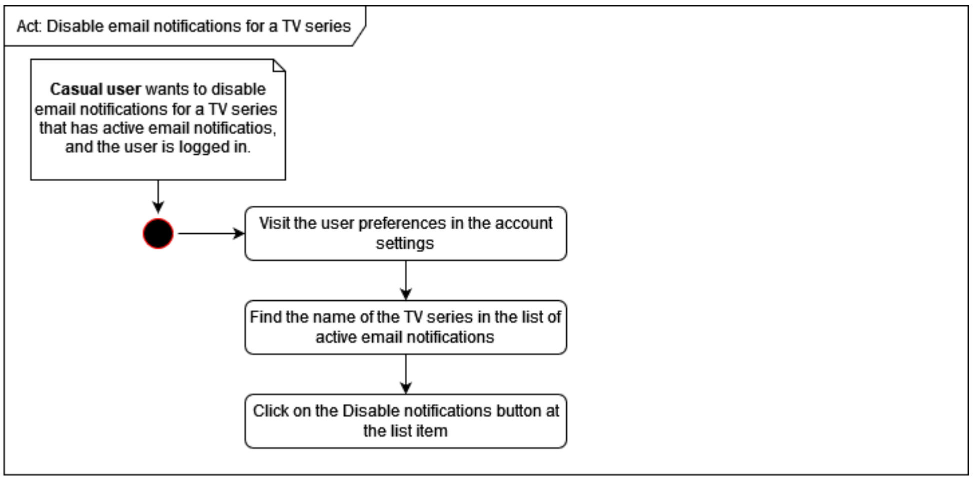
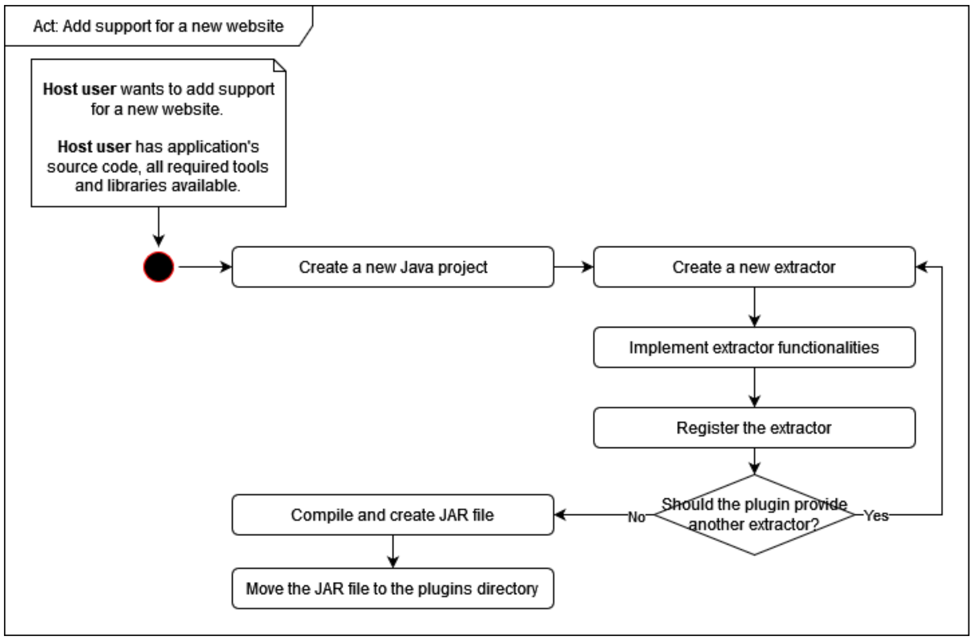

Act: Disable email notifications for a TV series
Casual user wants to disable email notifications for a TV series that has active email notificatios, and the user is logged in.
Visit the user preferences in the account settings
Find the name of the TV series in the list of active email notifications
Click on the Disable notifications button at the list item

Figure 5.10: Use case: Disable email notifications for a TV series

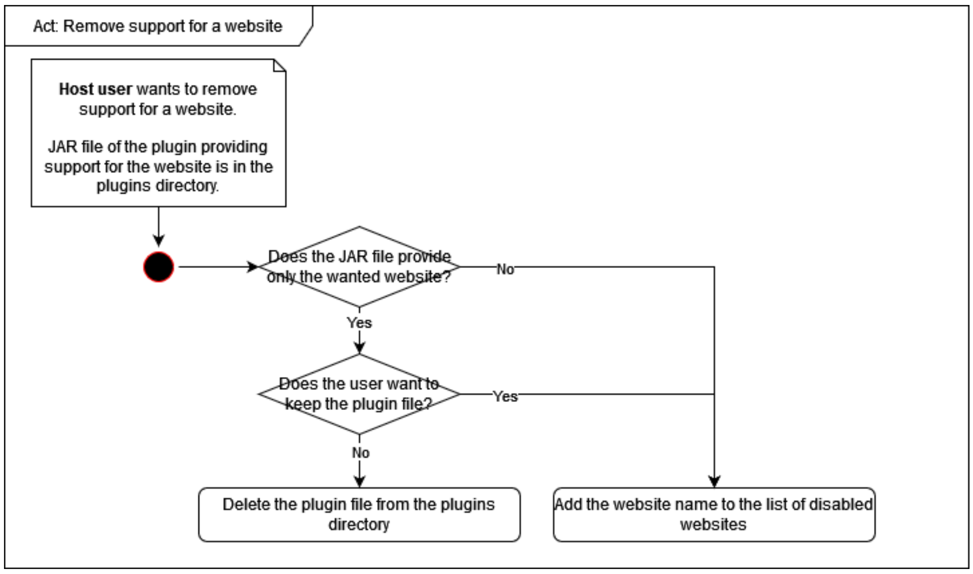
Add support for a new website



Act: Add support for a new website
Host user wants to add support for a new website.
Host user has application's source code, all required tools and libraries available.
Create a new Java project
Create a new extractor
Implement extractor functionalities
Register the extractor
Should the plugin provide another extractor?
No
Yes
Compile and create JAR file
Move the JAR file to the plugins directory

Figure 5.11: Use case: Add support for a new website

Remove support for a website



Act: Remove support for a website
Host user wants to remove support for a website.
JAR file of the plugin providing support for the website is in the plugins directory.
Does the JAR file provide only the wanted website?
No
Yes
Does the user want to keep the plugin file?
Yes
No
Delete the plugin file from the plugins directory
Add the website name to the list of disabled websites

Figure 5.12: Use case: Remove support for a website

5.1.7 Graphical user interface

The application's graphical user interface will be primarily used for browsing programs and information about them. Therefore, it should most importantly consist of the following pages:

- A page that displays a list of all programs in some way with search and filtering options.
- A page that displays information about a TV show, i.e. its details and list of its seasons and their episodes.
- A page that displays information about a movie or an episode, e.g. available qualities, languages, subtitles, etc.

Additionally, since the application may provide a way of notifying users of some events, such a new episode of a program, it may also provide the following pages:

- A registration page with a registration form.
- A login page with a login form.
- An account dashboard page with account settings, such as a form for managing notifications.

For the general layout and the visual design of the pages, we can look at the selected websites themselves. They all share a similar layout and look, so it probably works well.

Programs listing page

As seen on the Nova Voyo and Prima+ websites, we can display the items beside each other with an image of the program and its title. Since it should be a proper listing with pagination or infinite scrolling, we can display the items in a grid, instead of a carousel as seen on the mentioned websites. This grid should probably be displayed below a search and filter form, that itself is placed below the website title.

The search and filter form may have two variants, a simple one and an advanced one. The simple one displays just the search input field, a search button and a button to display the advanced variant. The advanced variant should display advanced filtering options underneath the search input field, such as an input for a year or year range, genre and actor input, etc. The switch from a simple variant to the advanced one should only display the additional fields and hide them in reverse.

The items in the grid may be dynamically updated when a change happens in the form, or may be updated only after clicking on the Search button.

Clicking on an item in the grid should show the details of the program to the user. Additional information may be revealed when hovering over the item with a mouse or focusing it using a keyboard.

TV show detail page

A TV show is a special case of program, since it does not have any media-related information, such quality or subtitles, but consists of TV series that themselves have episodes, which have the media-related information. Therefore, the TV show detail page should display the general program information (image, title, year, genre, etc.) about the program with a listing of its TV shows. Either as a list or a grid. Each season item in a list or a grid should consist of its season number, additionally its title, if it has any, and by selecting the item it should display the episodes themselves. Each of the episodes should then display the information about it.

The layout of the page may be a header with the TV show image and its title, with the other information below the image, followed by the seasons listing.

Movie and episode detail page

A movie has general program information and media-related information, such as quality or subtitles, therefore it should display this information. An episode does not have the general program information so it should display only the media-related ones with the addition of the episode-specific ones (e.g. episode number).

The layout of the page may be similar to the one of the TV show detail page with the addition of the media-related information.

Registration and login page

The registration and login page should consist of a simple registration and login form respectively. The form itself consists of fields and a submit button. Error messages may be displayed either above the form or above the submit button.

Account dashboard

The page should display the username of the currently logged in user and a section dedicated to managing notifications.

The notification section may display all active notifications as a list. Each item in the list represents a single notification and displays the name of the program, the type of notification and a disable button.

Notifications

Notifications are used for notifying the user of some event. The first version of the application may at least support the notification of the event when a specific program has a new episode added. A new episode may be either the latest episode or an older one that was added to the website as a replay. The user should be able to choose whether they want to be notified of the latest episode or a replay one.

When any such event occurs, the system should send an email to the user with information about the event. In the case of the latest episode it may include information such as the title of the program, the season and episode number of the episode and its title, the direct link to the original website and the direct link to the episode detail page in this application. The email itself may either be a plain text, or may use HTML for the direct links or visual display.

A user should be able to activate notifications for a program by visiting its page and by clicking the notification button. When having notifications active for a program, the notification button should then function as a button to disable notifications for that program. Also, a user should be able to view all of their active notifications and disable them from a single place for convenience.

Data normalization

Two websites may contain the same program or may reference the same actor. On each of those two websites the name of the program or actor may be in a different format or may include additional characters or words. However, as such they should still be considered the same and should be merged in the application. Also, these two names should then be considered as aliases to the final name. Therefore there has to be some normalization process to ensure that.

An external normalization is a normalization that uses a third-party service that is not necessarily used just for normalization. The main concept is that for a given name or title we find a normalized name or title, i.e. its canonical form, by searching or looking up the name or title. This way the external normalization is just a black box that for the alternative names returns their normalized form. Websites such IMDB or CSFD may be used, more specifically their search function.

An internal normalization is a normalization that is done by the application itself. Some basic normalization is trivially done in modern programming languages, such as trimming a text of white space characters (in Java using the `String::strip` method) or transforming a text to its normalization forms (either NFD - Canonical decomposition - or NFC - Canonical decomposition followed by canonical composition[32]). However this is just the first step that simplifies further processing. A name may be a single word but more often it is a sequence of words. In the case of an actor's full name, we don't know on the first glance, whether it is in the order of last name first and then the first name or the other way around. There may also be many formats for the names themselves, such as "Last name, First name" (with a comma) or "Last name First name" (without the comma). In essence it is not trivially done without some help, such as a set of possible first names, but even in that case there exist last names that are also first names.

For the internal normalization we can simplify the problem a bit to just specify a format for a field, such as an actor's name, and a website. For example, we can specify that the TV Nova website uses the format "First name Last name". There may also be some hints in the HTML source code or even better, in the Linked data themselves, if they are present.

This, however, still does not solve the problem of normalizing the names of TV shows and movies. To normalize them we can use a database of them, use the aforementioned external normalization, or just simply use the name itself with just the basic normalization applied (trimming, normalization forms).

The issue with using a third-party tool is that the requests may be blocked due to the total amount of them, it causes the third-party server load to increase, uses more bandwidth, and may even be slower due to latency and other factors, such rate-limiting, retries, etc.

Finally, there exists yet another option in the form of user-managed content. That is, that users themselves check and possibly edit the names and merge two programs that should be the same but are not in the application. However, this option is against the goal of this application, i.e. to minimize the need for user intervention when aggregating the media content, and would also require quite a lot of additional work.

Extraction process

From performed tests it took 6-8 hours just to obtain all programs and their episodes from the Česká televize website. At the time of the test there were around 12000 programs and the number will surely grow in the future. This amount of time is not feasible to be spent just to extract from a single website since it would be beneficial to do the extraction process at least once a day. Also, it would put a strain on the server and on the website itself. Therefore a strategy where not all programs and episodes are always extracted should be devised.

To define some terminology that will be used, an extraction iteration is the process of going through all supported websites and extracting all relevant information. The goal of an extraction strategy is to not spend many hours just doing one extraction iteration since it will be run at least once a day. From the goal and the performed tests it may be concluded that in a single extraction iteration it is not feasible to visit everything and therefore always have the most recent information. Therefore there should be some logic on what to extract and when to extract it.

To ease the logic behind a extraction strategy, types of programs may be defined, and that is as follows:

- An active program is a program that has had a recent change made to it. For a TV show it may be either a new season or a new episode in a season. For a movie, it may be a change to its media sources. For any program it may be when any information about it, such as list of actors, description, etc. has been changed.
- An inactive program is simply not an active program, meaning that nochange has been made to it recently.

This terminology should help the overall extraction strategy in the following way:

- An active program should be extracted more often, since it may have another change in the near future. Think of it as an ongoing TV show releasing a new episode every week or even every day.
- An inactive program may be extracted less often to save computation power and especially time. Think of it as a TV show not having any new episodes or seasons in the last year, there may still be some probability that it will have a new season in the near future but it is not changing often.

If a program is inactive it does not mean that the program will never be extracted again. If the program is a TV show it will have its seasons probed to switch to the active state. If the program is a movie it may take a longer time to detect a change to its media sources or a description, but it will still be probed once in a while.

To specify what "while" actually is, there is probably no exact metric to be chosen from. However, a reasonable period should be chosen, i.e. once a week.

To further cut down the time it takes to do a full single extraction iteration, the inactive programs may be split in multiple groups. Every active program is probed once a day and may some time later become an inactive one. This is the work that will be done every day. However, to actually visit the inactive programs as well, we have to add the inactive programs to this constant work at some point. To do it all at once is not feasible since that would mean we would do the whole full single extraction iteration in one go. As was mentioned we can split the inactive programs in multiple groups, for example based on the time they became inactive. This way we may probe the recently inactive programs more often than those that have been inactive for a year.

From all of this we have:

- G is a group of programs. G_A is a group of active programs, G_i, where i is a number, is a group of inactive programs. Two groups do not contain the same program, i.e. each program is only in one group.
- $P(G)$ is a period for group G. We have a relationship that binds them all together: $P(G_A) < P(G_1) < P(G_2) < ... < P(G_n)$.
- We are choosing periods and they are mathematically a multiplication, therefore there will be a day D where G_1 and G_2 should be both visited. Trivially the day is $D = P(G_1) * P(G_2)$. More specifically the day is $D = lcm(P(G_1), P(G_2))$, where lcm is the Least Common Multiple of its arguments.
- When such a day occurs, the group of a lower number should be visited, i.e. if G_2 and G_4 should be visited on a day D, then G_2 should be visited.
- There will be some days D_k, where $D_k \neq m * P(G_i)$ for any i and any m. On these days the group that was visited further in the past should be visited, i.e. if G_2 has not been visited in a week and G_3 in a month, G_3 should be visited.
- If there are multiple groups G_{i_1}, G_{i_2}, ..., G_{i_k} that have the same time of visitation that is the furthest in the past, we simply choose the one with the lowest number. If $i_1 < i_2 < ... < i_k$, then we choose G_{i_1}.

Using these rules each day all active programs plus a group of inactive programs will be visited, groups with smaller periods will be visited more often than groups with longer periods, and all groups will be eventually visited.

To provide an example of how to split and then how to visit inactive programs, we have to first choose the period and the number of groups. The groups may be as follows:

1. Inactive programs having inactive time less than a week.
2. Inactive programs having inactive time less than a month.
3. Inactive programs having inactive time less than half a year.
4. Inactive programs having inactive time less than a year.
5. Other inactive programs.

The periods themselves should have minimal collision. It was stated that a collision occurs on day $D = lcm(P(G_i), P(G_j))$ for some groups G_i, G_j. LCM can be computed as follows: $lcm(A, B) = (A * B)/gcd(A, B)$, where the maximum of the lcm function is when $gcd(A, B) = 1$. This means that A, B must be relatively prime. Therefore $P(G_1)$, $P(G_2)$, $P(G_3)$, $P(G_4)$ and $P(G_5)$ must be all relatively prime. $P(G_A)$ is always equal to 1.

To fulfill this condition we can simply choose the primes: $P(G_1) = 2, P(G_2) = 3, P(G_3) = 5, P(G_4) = 7, P(G_5) = 11$. For this configuration there will be, for example, day $D = 13$ on which none of the groups should be visited, but from the rules above, the group that was not visited in the longest time is selected, specifically group G_3. There is also the day $D = 1$, where no group should be visited, in this case we choose the group G_2.

At the beginning all programs will be in the active group, however, over time they will be separated to the other previously mentioned groups. We can speed up this process with setting the changed date of a program to a changed date of information on its page. However, it may not always be available.

Application extensibility

The application must be extensible in terms of extraction. This can be achieved using plugin or modules that will be either dynamically or statically loaded to the application. Or it can also be achieved by using external files written in some programming, scripting or markup language. These files may just contain definitions that specify how to extract the data.

Plugins. A plugin is a collection of Java classes that are loaded using a ClassLoader to the application. They can be loaded either dynamically at runtime or statically during startup. The dynamic approach allows not to stop the application just for loading or unloading a plugin, defer the loading of plugins, filter the plugins based on a condition, manage them at runtime and more. It is a more flexible approach than the static one and will be preferred to it. Either way this collection of classes is often contained in a JAR file that is then loaded.

To recognize that a JAR file is a plugin for the application, there must exist some indicator. This may be done using an annotated class. First, we define an annotation that may contain additional information about the plugin, such as its author, version, etc, and annotate a single class in the JAR file with it. This class should also implement a specific interface such that when the application finds this class, it can call a specific method to initialize the plugin so that the plugin may register its extractors to the application.

This is an example of how it may look like:

```
@Plugin(
    name="plugin.website_name",
    author="The author",
    version="1.0"
)
public class ThePluginClass extends PluginInterface {

    @Override
    public void initialize() throws Exception {
        // Do initialization, i.e. register an Extractor
    }

    @Override
    public void dispose() throws Exception {
        // Do disposal, i.e. unregister an Extractor
    }
}
```

This class and possibly other classes in the JAR must be compiled with the application's source code to have the required classes available. This approach is the most flexible one, because the author of the plugin has all Java/application classes available and can write their own logic in any way they can.

Using this approach there may be some security concerns since an external class file is loaded to the JVM and its code is run. However, this is an inherent issue of this method and should be taken into account. Only trusted plugins should be loaded to the application.

Scripting files. Another method to make the application extensible is to allow host users to execute an external file that is written in another programming or scripting language, such as Lua, Python, etc. The application reads this file and interprets it. In the contents of the file there are calls to provided functions and provided data types are used. This allows to limit the usage of specific functions that are otherwise available in plugins written in Java. However, since an actual interpreter of the language will probably be used, it can still call potentially dangerous functions that are provided by the language itself.

A custom language can be also used to provide this functionality, however this requires more work to implement a parser and evaluator.

Definition files. More restricted method that may be used for simpler websites is the usage of definition files. A definition file is a file that contains definitions or simple procedures of how to obtain the data that are wanted. This may be done in any language, format or form, even in plain text. This is possibly the most restrictive method since a custom interpreter must be created for it to work. This interpreter has only a limited "instruction set" that may be used in definition files. However, the file may become rather long and verbose due to this.

This method may not be suitable for websites where custom logic or more complex operations are required since it is dependent on the available "instructions".

Authentication

All of the selected websites do not require any kind of authentication to obtain the TV shows, movies and episodes, but some of them (iPrima, Nova Voyo, JOJ Play) require authentication to obtain the available media sources for episodes and movies.

The process of authentication itself is straightforward to implement using

HTTP requests, HTTP headers and cookies, however care should be taken in terms of storing the credentials used for the authentication.

Passwords should not be present in the code of the plugins themselves for security concerns. They may be stored in an environmental variable, database or a special storage. Since the plugins may be loaded dynamically, it must be possible to load the credentials dynamically. Purely in Java the environment variables cannot be changed for an already running process[33] and the database should be used solely for storing the entities. Therefore the only other option is to use a special storage. For this purpose the Java's KeyStore API may be used.

A password may be chosen to secure the key stores where actual credentials to the websites are stored. The password is set as an environment variable and later on, when the key store is being read dynamically, it is used to decrypt it. This way the credentials are never stored as a plain text and are not mixed with the extracted data.

Geolocation blocking

Some content on the selected websites is blocked for other than specific geographical locations. This is called geolocation blocking, geoblocking or geolocking.[34] The server that is running the application is located in some country, this country may not be present in the list of the countries that are whitelisted for a specific movie or episode. Therefore a care should be taken before requesting the media sources.

One way to fake the location to the server is to use a VPN. There are many available VPNs that may be used,[35] but a good-enough free option is preferred. One of the options is to use OpenVPN and select a specific country before carrying out the request to obtain the media sources.

Assume that the country of the server is not the same as the country that is required to access the media sources of a specific movie. Other movies on the website may be obtained successfully because the server's country is fine but changing it to the previously required country may result in not obtaining the media sources for those movies. Therefore we cannot route all network traffic through the VPN for all items and since there may be concurrent requests present in the application, we cannot route it just for some time period. Thus a special network proxy must be created to use the VPN exclusively only for specific requests.

Request-exclusive VPN may be done using a Docker image where the VPN is running. This Docker image has an HTTP proxy opened with which we may route specific requests. This way we can still do requests in the application itself as if no proxy exists but select which requests are actually routed through the VPN and which are not.

There are already existing Docker images for this exact purpose which may work, e.g. `https://github.com/kizzx2/docker-openvpn-client-socks`.

Speed of extraction

If there is a desire to quickly obtain at least basic information about new programs, episodes or media sources, an alternative order of execution during the extraction process may be used. Instead of obtaining additional information, such as description or thumbnail image, during the processing of a program or episode, it is possible to delay it to a next "wave". The extraction iteration is therefore split into two "waves", where in the first one we obtain just the basic information, e.g. URI, and after everything in the "wave" is processed, the next "wave" begins, such that it processes everything again but it only obtains the additional information.

Additionally, threads can be used to further speed up the extraction process. For example, it may be used when obtaining multiple categories of programs so that it is done in parallel for some websites. This cuts down the required time to obtain the list of programs or episodes. However, it causes more load to the server, it may cause the connections to timeout or be rejected by the server, and in the overall time that is required to obtain everything, the time saved would probably not be really significant.

Chapter 6

Implementation

The application consists of backend, that provides the API endpoints and runs the extraction process, frontend, that provides the user interface and uses the API to obtain extracted data, and lastly the database, that stores the extracted data.

The backend is implemented using Java programming language, specifically Java 17. It uses Maven to generate sources, compile the source code and build the final JAR file. It is also split into modules:

- The core module - provides the base classes that may be used in every other module, most notably the various utility classes (Web, Net, HTML, etc.)
- The backend module - provides the API and interacts with the database.
- The plugins - Each of the previously specified websites is its own module that uses the core module.

The frontend uses ReactJS, specifically Vite, to render the user interface. It requests data from the API using the JavaScript fetch function.

6.1 Architecture

The application consists of three main parts - the backend with API, the frontend and the database. Additionally, there are proxies that are used during the extraction process, by default for the Czech Republic and Slovakia. They may be turned off but it must be done before running the backend module of the application.

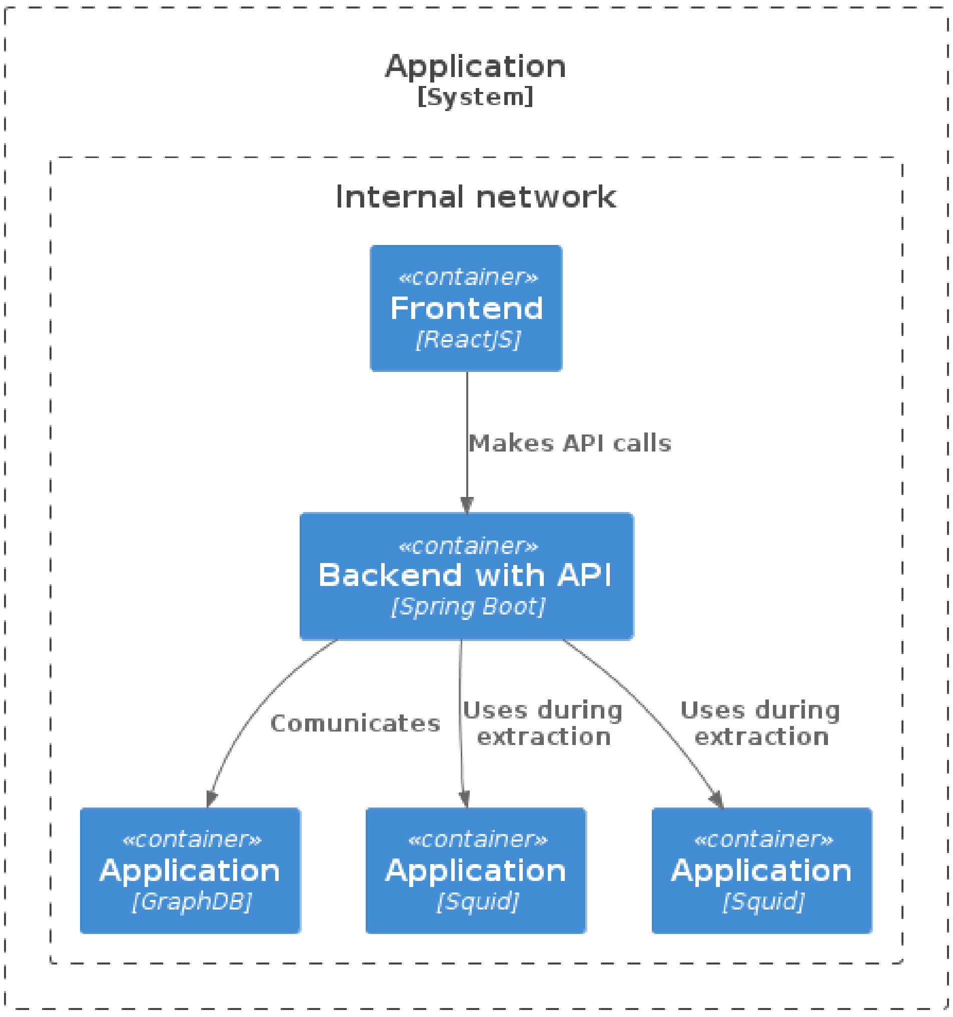


Figure 6.1: Architecture of the application

6.2 Application extensibility

Based on the analysis of selected websites from previous sections there is an actual need for more complex operations and logic than the definition files may provide. Also, using the scripting may result in a longer and more verbose file that may not be maintainable in the future. Therefore the better approach is to use plugins. However, it may be desirable to also implement one of the other methods to ease the process of adding a custom website to the application and should be considered.

6.2.1 Plugins

Plugins are loaded during the application initialization. This is done after the Spring boot itself is initialized. Plugins directory is scanned in the current working directory of the application for JAR files. Each JAR file is first probed for the `Plugin` annotation that is used to represent the main class of a plugin. This process is done using the library Annotation detector by INFOMAS ASL. It reports the class that has this annotation. If such a class is found in the JAR file, the JAR file itself is loaded by its own URLClassLoader. Finally, the class that has the Plugin annotation is instantiated. Because the class has to extend the `PluginBase` class, the initialization method therefore exists and this method is called immediately after instantiation. In this method the plugin may register the website it provides.

6.3 Geolocation blocking

To solve the problem of geolocation blocking a proxy is used. When a website extractor requires a specific country to be in, it sends requests using this proxy. This proxy is an HTTP proxy and there exist multiple proxies, each for a single country. To ensure that the proxy is in a specific country, a VPN provider is used.

The implementation uses multiple Docker containers for each proxy. They consist of a WireGuard container and a Squid proxy container. It is then combined using Docker compose. The WireGuard container uses a predefined WireGuard configuration that is obtained from the VPN provider. This configuration sets the traffic to be exited in the specific country so that it seems as if the application itself is running in that country. The Squid proxy container then runs the proxy itself that acts on the network interface that uses the WireGuard configuration. It exposes a port to the host system that then can be used as an HTTP proxy to any application that wishes to use it. Each country proxy has uses its own port.

In the application it is then possible to simply choose a port on the local system as an HTTP proxy. The traffic is then routed through it, through the Squid proxy and the WireGuard network interface in the container. Using this setup we are able to selectively route any specific request through a proxy that acts as if the application was running in a specific country, but other requests may be routed normally.

To be more specific, using this setup we are able to route only those requests for TV Markiza or TV JOJ websites that actually require it, such as when obtaining media sources for a movie or an episode. All other requests are routed normally, such as obtaining the list of programs.

This should also help with some internal rate limiting at the VPN provider itself or to not be blocked by their filters that may be present for some of them.

This solution stems from the compose-wireguard-squid (`https://github.com/master-hax/compose-wireguard-squid`) project on GitHub. It was revised and optimized to use only 2 containers instead of 3.

The WireGuard Docker container requires the Linux kernel to support CONNMARK, which may not always be available, for example in the case of the stock WSL2 Linux kernel. To solve this issue it is required to compile a custom kernel image and set it as the default one for the WSL2. Instructions on how to do this are in the project's ReadMe file.

Lastly, there may be an issue with using the WireGuard in a Docker container due to the connection shutting down after some time. This may be due to the NAT/Firewall mapping being dropped. This can be solved by setting the PersistentKeepalive peer property in the WireGuard configuration to some non-zero value[36] manually.

6.4 Data normalization

In the current implementation the only normalization that is done to titles and names is a text normalization. It mostly uses Unicode normalization forms, more specifically the NFKD - Normalization Form KD (Compatibility Decomposition).[?] Additionally it also removes non-base glyphs when normalizing the names of countries, people, languages and genres. These normalized names are then used in their identifiers and are also URL encoded. Finally, the normalized text or name is stripped of whitespace characters from the beginning and from the end. Finally, for possible HTML input, such as descriptions, the HTML tags are stripped and only the text itself is kept. This is done using the Jsoup library.

6.5 Credentials

In the current implementation the credentials are handled by a single Credentials file (`.credentials`). This file simply contains multiple key-value properties that are then loaded by the application and later used when a website requests them. The Credentials file is in the same format as Spring's properties file.

All of the implemented plugins use two properties each, one for a username and one for a password. Each such property has a common prefix for the website it is used for and then a suffix that represents the name of the property, such as username or password.

6.6 Model

The final model differs from the model specified in the application's design. The biggest difference is in representing the media sources.

The media sources of a movie or an episode are represented using the schema.org's VideoObject type. The type itself holds information about the video itself but also contains the embedded audio of type AudioObject. This way it is possible to map internal representation of media sources (the `Media` class) to these types, that are then shown in the API's output.

Some other properties were added or removed to better support available data.

Lastly, in the final class model there is only a single extractor and all website's extraction functionality is handled by the `Website` class.

6.7 API

The API is specified using the OpenAPI specification, where the API endpoints and used entities are described. Java sources for the API are generated using Maven by the openapi-generator Maven plugin. The generated sources are then used for implementing the actual procedures for each of the endpoints.

The API specification follows the schema of schema.org so that the results may be read and parsed by another software.

6.8 Design

The final design was mostly inspired by the websites that were selected. However, some components are implemented in a different way.

6.8.1 Homepage, TV Series page and Movies page

All of Homepage, TV Series and Movies display a list of programs in a grid with a list filter. The list filter allows to filter the list. There is also a pagination present to limit the number of results per page, otherwise it would cause performance issues. The filtering and searching, including the pagination is all dynamic and done using the Vite framework.

Items in the grid consist of their image and their title. The title is displayed after focusing or hovering the item.

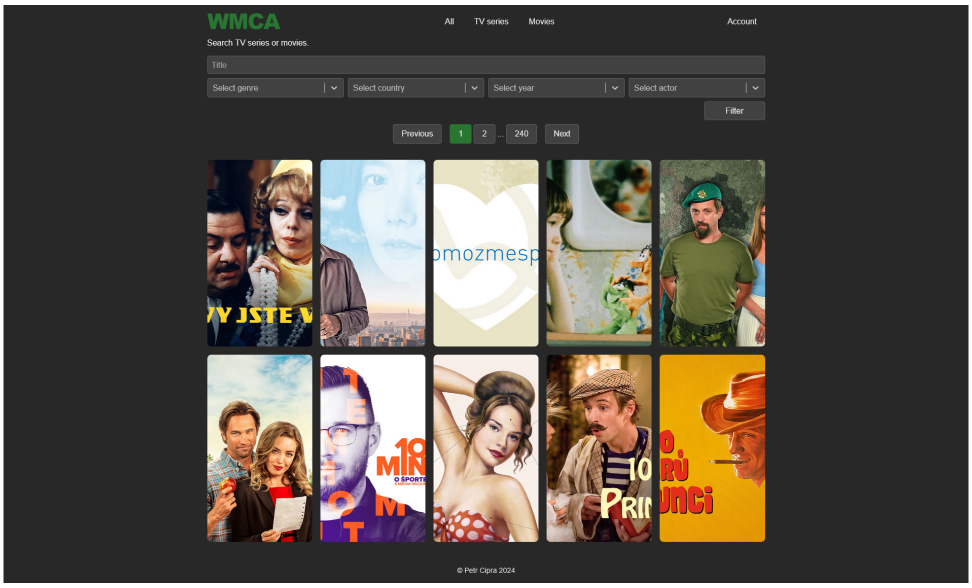
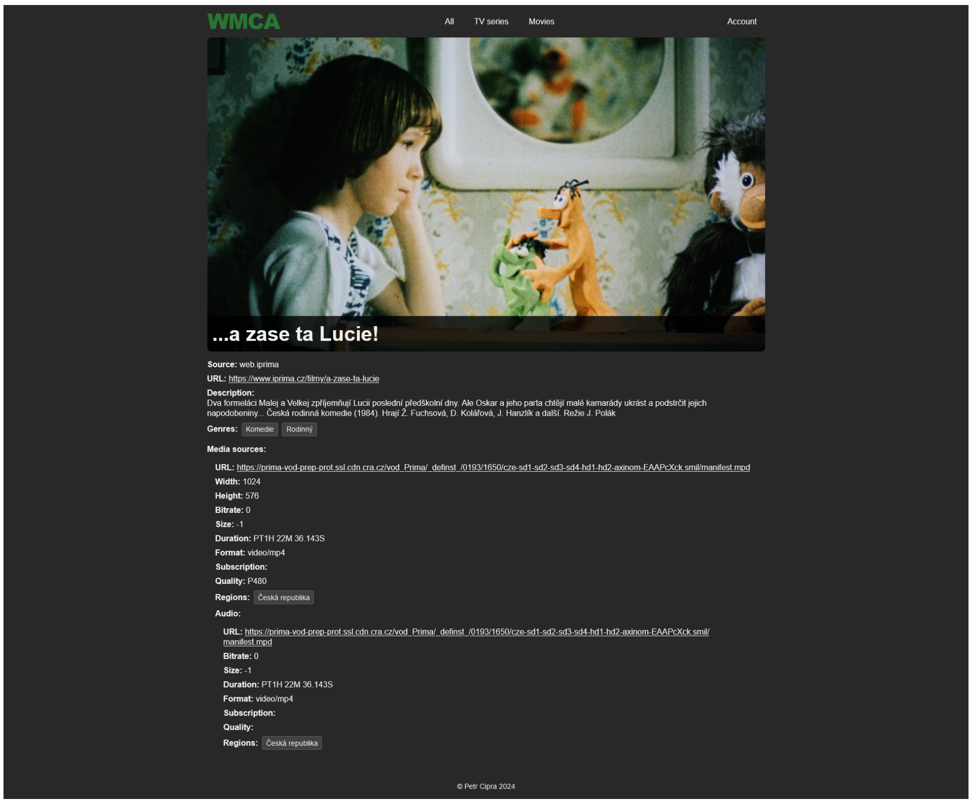


Figure 6.2: Detail page of a movie

6.8.2 Detail pages

Detail pages all display a header with the item's image and title, followed by common information, such as URL, description, etc. After the common information, specific information is display in a textual form. For movie and episode it is its media sources, duration, etc., for TV series it is its seasons and episodes.



...a zase ta Lucie!

Source: web.iprima

URL: https://www.iprima.cz/filmy/a-zase-ta-lucie

Description:
Dva formeláci Malej a Velkej zpříjemňují Lucii poslední předškolní dny. Ale Oskar a jeho parta chtějí malé kamarády ukrást a podstrčit jejich napodobeniny... Česká rodinná komedie (1984). Hrají Ž. Fuchsová, D. Kolářová, J. Hanzlík a další. Režie J. Polák

Genres: Komedie Rodinný

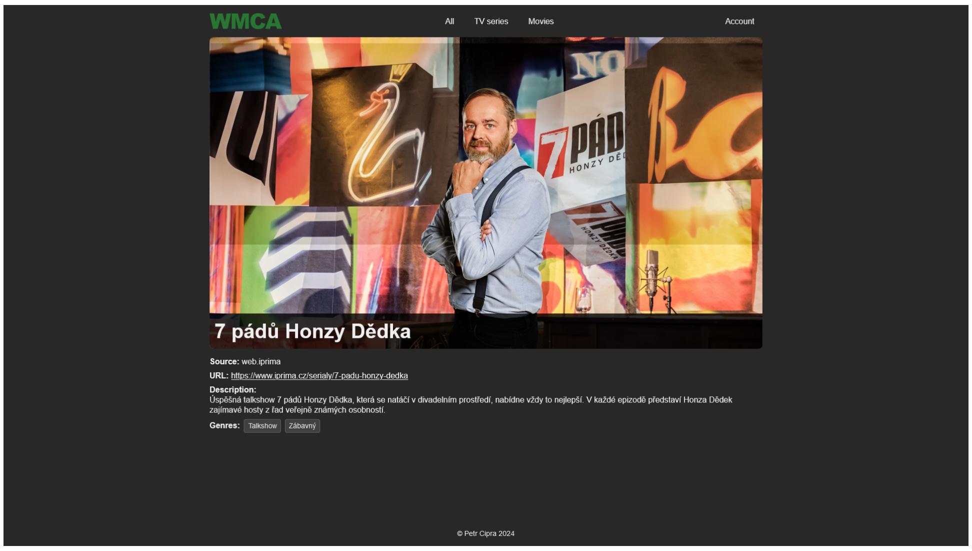
Media sources:

URL: https://prima-vod-prep-prot.ssl.cdn.cra.cz/vod_Prima/_definst_/0193/1650/cze-sd1-sd2-sd3-sd4-hd1-hd2-axinom-EAAPcXck.smil/manifest.mpd
Width: 1024
Height: 576
Bitrate: 0
Size: -1
Duration: PT1H 22M 36.143S
Format: video/mp4
Subscription:
Quality: P480
Regions: Česká republika
Audio:

URL: https://prima-vod-prep-prot.ssl.cdn.cra.cz/vod_Prima/_definst_/0193/1650/cze-sd1-sd2-sd3-sd4-hd1-hd2-axinom-EAAPcXck.smil/manifest.mpd
Bitrate: 0
Size: -1
Duration: PT1H 22M 36.143S
Format: video/mp4
Subscription:
Quality:
Regions: Česká republika

© Petr Cipra 2024

Figure 6.3: Detail page of a movie



7 pádů Honzy Dědka

Source: web.iprima

URL: https://www.iprima.cz/serialy/7-padu-honzy-dedka

Description:
Úspěšná talkshow 7 pádů Honzy Dědka, která se natáčí v divadelním prostředí, nabídne vždy to nejlepší. V každé epizodě představí Honza Dědek zajímavé hosty z řad veřejně známých osobností.

Genres: Talkshow Zábavný

© Petr Cipra 2024

Figure 6.4: Detail page of a TV series

6.9 CRON

The extraction process is executed every day at midnight. If a previous extraction process is still running at the time when the next should already be run, it waits for the previous run to finish and then it is executed.[37] The application uses the Spring's `@Scheduled` annotation to enable this CRON-like functionality.

6.10 Searching and filtering

The search uses a simple regex filter by using the SPARQL's FILTER and REGEX functions, and it is able to search programs by their titles.

The filter simply filters the searched (or all results) by applying the settings chosen by the user. It filters the results in the following way:

- Different types of settings (e.g. genre, year) are in the AND relationship. That is, if the user selects both a genre and a year then the final results have both the selected genre and the selected year.
- Values of a single type of setting (e.g. multiple genres) are in the OR relationship. That is, if the user selects multiple genres then the final results have any of (at least one of) the selected genres.

6.11 Extraction process

The extraction process is done using an Extractor that is based on Nodes. A Node is just an abstraction where all required parameters are passed to the Node during its creation, then when it is time to process the Node, it is simply executed in the context of an Extractor. Nodes are saved in a queue that is sequentially processed, no multi-threading is involved. Although, it would be possible to implement multi-threading on either the Website or Node level.

At the beginning there is just a list of websites from which the Extractor should extract. Then each of them is processed as such that its programs are obtained. Each program creates a Node that is saved to the queue. This program's Node then creates more Nodes for each of its episodes and finally, each episode's Node processes its media sources. If a program's Node is a movie Node, then it directly processes its media sources. The processing of media sources uses an internal media source representation to create a representation that is used in the database and the API. This is done due to the fact that not all information is actually preserved in the database and available in the API and thus in the user interface of the application.

An additional logic is present for the extraction process to be more efficient in the form of speed of the overall process. That means that not everything is processed every time. As specified in the application design section, there exists an extraction strategy with the groups of programs are as follows:

- G_A: Active entities
- G_1: Entities changed in the last week at most
- G_2: Entities changed in the last month at most
- G_3: Entities changed in the last year at most
- G_4: Other entities

6.12 Order of extraction

The order in which the nodes are processed is important due to the fact that each node in the queue consumes some memory. Since a media source node (video or audio) has much more information than any other node and there are many movies and episodes that each has multiple media sources the amount of these nodes may result in consumption of a lot of memory. That can lead to out of memory errors.

To solve this issue we may simply change the location to which we insert new nodes. Therefore instead of treating the queue as a queue we may treat it as a stack. New node is therefore inserted to the head of the queue and processed next. This will lead to a short queue because every time a new node is inserted to it, it is expanded and all its child nodes are then immediately processed.

However, this means that each program is expanded to either its episodes, if it is a TV series, or its media sources, if it is a movie. Then each of these expanded nodes are processed leading to more child nodes. This results in the next program after the currently processed one to be processed much later and therefore not being added to the list of all programs quickly.

In conclusion, it is up to the host user which order is better, but the overall less memory consumption, that may otherwise lead to gigabytes of memory being used, is probably worth it at the expense of delayed addition of programs and episodes.

Chapter 7

Testing

Testing of the program was done throughout the development of the program. During the implementation, the program was continuously tested on selected browsers, namely Firefox 125, Google Chrome 125, Microsoft Edge 125. More specifically, it was checked whether all actions (e.g. button clicks, form submits, etc.) works and whether it displays correctly on multiple viewports: 1920px - desktop, 991px - tablet, 767px - smartphone and 560px - minimal.

Further testing was carried out at the end of the development using user testing.

7.1 User testing

The program was given to three users, each with a different level of experience with such applications. One of them uses Firefox, the other two use Google Chrome.

The users were given the task of going through the application and trying to find some information about a program and an episode. The whole testing process was under supervision and the shortcomings of the application and the comments of the users were written down in individual lists. The received feedback was processed and a list of suggested modifications was created. After implementing some of the modifications, the final feedback was received and the user testing was concluded.

7.2 Feedback

User 1, Google Chrome browser (version 125).

- Missing indicator when searching or filtering.
- The filtering takes too long.
- The main menu should be placed beside the logo to save some vertical space.
- The images of programs should be smaller in the grid.
- Missing country selection.
- Filter selection boxes have too many items.
- Pagination should be displayed above so that there is no requirement to scroll down.

User 2, Firefox browser (version 125).

- Missing indicator when searching or filtering.
- Slow response times.
- Big program images in the grid.
- No country selection present.
- Missing information about the source of the program.

User 3, Google Chrome browser (version 125).

- Long wait time when filtering or searching.
- Genres at a TV series or movie are harder to read.

- Missing information about about the source of the program.
- The media sources are harder to read.
- The duration and date of a program is hard to read.

7.3 Proposed modifications

As it can be seen from the feedback, all of the lists contain some repeating issues with the user interface of the application. From these lists almost all of the feedback notes were selected. The only feedback notes that were not considered after time estimation and time constraints were the issue that the media sources are harder to read and the duration and date of a program being harder to read. The first issue would require some better designing to be done to actually solve the issue, and the second one has not been done due to time constraints.

- **Missing indicator when searching or filtering**
  Solution: When a change happens, such as updating the text in the search field, selecting a new genre, etc., display a loading indicator at the list. The position of the indicator was chosen to be an overlay and centered graphics in the shape of a circular sector in the theme color. The indicator is hidden after the request is finished and the results are displayed in the list.
- **Slow response times when searching or filtering**
  Solution: The slowness was caused by sending requests every time a change is made to the filter, most notably when a search text is updated. To solve this throttling was added such that there is a timeout of 250ms and when it times out, it does the request. However, if another change is done before the 250ms time out, the timeout is cancelled and a new one is created. This means that the request is done only after no change is made for at least 250ms, i.e. when user stops writing to the search filed.
- **No country selection**
  Solution: Since the extractor was already extracting and saving countries, the work done was to just add it in the same way as the selection for genres.
- **Filter selection boxes have many items**
  Solution: The selection box was reworked to allow searching by typing. This is just a mitigation of the issue and there may be a better solution.

- **Big program images in the grid**
  Solution: Changed the number of items per row in the grid to be greater, i.e. 5 instead of 3.

- **Missing information about the program's source**
  Solution: A new property called "source" was added to programs. The contents of this property is currently just a non-translated prefix of "web.*", where * is a website's name (e.g. tvnova, iprima). It would be better to display it as an icon, for example.

- **The main menu should be placed beside the logo to save some vertical space.**
  Solution: The main menu was placed into the center of the header to the right of the logo.

- **Pagination should be displayed above so that there is no requirement to scroll down.**
  Solution: Pagination was moved from below the list to above it.

7.4 Retesting

The retesting was conducted after implementing the proposed modifications and it was done in the same way as the first user testing.

7.5 Feedback

User 1, Google Chrome browser (version 125).

- The detail page of an episode is rather confusing.

User 2, Firefox browser (version 125).

- No major issue was provided.

User 3, Google Chrome browser (version 125).

- The media sources are still hard to read.
- The duration and date of a program are still hard to read.

As can be seen from the list above, the issues that have not been implemented are still present.

7.6 Conclusion

Due to the time it took to collect the feedback and time constraints, no further user testing was concluded. The remaining issues are major issues that should be solved in the future. However, thanks to the feedback, the application was improved considerably.

Chapter 8

Conclusion

In this document I specified the goal of this project as a web application that aggregates information about media content on various websites. Following with the selection of websites from which to extract the information and how to obtain it. I also did analysis of existing tools that may be used for this project, and provided an application design. Finally, the project was implemented and tested using user-testing.

From the analysis of the selected websites and the analysis of available tools for web scraping it was decided to just use Java libraries and not external third-party services or applications, since it would suffice to implement all required functionality.

From the analysis of what the selected websites publicly use to represent various data and from analyzing schema.org, it was determined that schema.org is sufficient to represent the extracted data.

For the application extensibility plugins in the form of JAR files were chosen. This option provides the most functionality when implementing a website extractor since everything is done in code. Also, it only requires implementing a plugin loader that loads these files, no parsing of external files or custom scripting languages are required.

OpenAPI was chosen to specify the application's API, and Maven plugins were chosen to automatically generate files related to the API. This simplifies the whole process of implementing an API.

The extraction process was designed and implemented from scratch with a custom strategy on how to handle many programs over time. This should allow not having to spend time on programs that may not update frequently and thus speeding up the extraction process itself.

The frontend was implemented using Vite and it uses the API the backend provides to access the extracted data.

In the overall implementation most of the functions have been implemented with the exception of user-specific functions due to the limited time and many problems during implementation of more important functions.

Finally, the application was user-tested with three users on different browsers. The result of this testing was a feedback from which a list of proposed modifications was constructed. Some of these modifications were then implemented in the application. This helped the application to be more user friendly.

In conclusion, the application fulfilled its important requirements and provides the core functionality to the users.

8.1 Future development

The application can surely be improved in many ways and the missing functionalities can be implemented. In this chapter I would like to mention some improvements that may be done in the future.

8.2 Serving images

The application in the current version displays images directly from the origin server. That means that the user's browser sends requests to a third-party server that contain the user's IP address, user agent and more. This behavior may not be desirable.

The solution to this is to use a proxy such that the actual request to obtain the image from the origin server will be done by the application itself. Using this solution there is also the advantage that the image may be cached and served directly without contacting the origin server, furthermore the images may also be scaled to better fit the dimensions in which they are displayed in the application's user interface. The issues with this approach may be that we are storing the image, and therefore most probably a copyrighted material, on the server, and that the image itself has some size and thus the available disk space may fill just by caching images. However, it is still a possible solution that prevents the user's information from leaking.

8.3 User customization

More user customization may be added, for example, the option to customize the listings. This may be helpful if the user does not want to show particular programs or episodes in the lists. They can simply hide them and the hidden items will be filtered out from the lists.

The ability to add TV series or movies to favorites may also be implemented. This may be useful when the user wants to access those programs or episodes quickly, or if they just want to save them for later. This may be expanded to allow users to create multiple lists instead of just being able to add items to the favorites list. For example, the user may create lists Favorites and Watch later. These lists may also be used during filtering or searching such that the user may search in a list or may exclude all items from a list from the listing.

There are surely many more customization options and functionalities that may be implemented, are useful to the user and improve the user experience.

8.4 User experience

The website may also be improved in terms of user experience. Currently, it is a barebone application that displays the data textually without much styling. It would also help to reorganize the website to be easier to navigate, and to provide more of the extracted data.

Appendix A

Bibliography

[1] What is semantic web. https://www.ontotext.com/knowledgehub/fundamentals/what-is-the-semantic-web/.

[2] Rdf primer. https://www.w3.org/TR/rdf11-primer/.

[3] Json-ld. https://www.w3.org/TR/json-ld/.

[4] Json main website. https://www.json.org/.

[5] Linked data. https://www.w3.org/DesignIssues/LinkedData.html.

[6] What is sparql. https://www.ontotext.com/knowledgehub/fundamentals/what-is-sparql/.

[7] Graphdb documentation. https://graphdb.ontotext.com/documentation/.

[8] What is graph database. https://www.oracle.com/cz/autonomous-database/what-is-graph-database/.

[9] List of tv channels of tv nova. https://tv.nova.cz/program.

[10] Voyo tv nova. https://voyo.nova.cz/.

[11] The main website of tv nova. https://tv.nova.cz.

[12] iprima programs. https://www.iprima.cz/tv-program.

[13] Prima+. https://www.iprima.cz/.

[14] Prima zoom. https://zoom.iprima.cz/.

[15] Česká televize ivysílání. https://www.ceskatelevize.cz/ivysilani/.

[16] The main website of markiza sk. https://www.markiza.sk/.

[17] Voyo markiza sk. https://voyo.markiza.sk/.

[18] Joj videoportál. https://videoportal.joj.sk/.

[19] Joj play. https://play.joj.sk/.

[20] The meaning of the word program (us spelling of programme). https://dictionary.cambridge.org/dictionary/english/programme.

[21] unogs.com. https://unogs.com/.

[22] Trakt. https://trakt.tv/.

[23] Faq voyo pro správu zařízení. https://voyo.nova.cz/faq/24-na-kolika-zarizenich-mohu-sledovat-voyo.

[24] Json-rpc. https://www.jsonrpc.org/.

[25] Nuxt.js framework. https://nuxt.com/.

[26] Oauth2. https://oauth.net/2/.

[27] Next.js. https://nextjs.org/.

[28] Jsoup java library. https://jsoup.org/.

[29] Json-java. https://github.com/stleary/JSON-java.

[30] Moscow method. https://www.agilebusiness.org/dsdm-project-framework/moscow-prioririsation.html.

[31] Bootstrap 5.0 - breakpoints.
https://getbootstrap.com/docs/5.0/layout/breakpoints/.

[32] Unicode normalization forms.
https://www.unicode.org/reports/tr15/.

[33] Java documentation - process class. https://docs.oracle.com/en/java/javase/17/docs//api/java.base/java/lang/Process.html.

[34] What is geo-blocking.
https://nordvpn.com/blog/what-is-geoblocking/.

[35] List of vpn providers.
https://www.saasworthy.com/list/vpn-software.

[36] Wireguard quick start guide.
https://www.wireguard.com/quickstart/.

[37] Scheduling in spring framework. https://docs.spring.io/spring-framework/reference/integration/scheduling.html.

[38] Petr Cipra. Media downloader application.
`https://github.com/sunecz/Media-Downloader`.

[39] Petr Cipra. Media downloader default plugins.
`https://github.com/sunecz/Media-Downloader-Default-Plugins`.

[40] Ledvinka, Martin, and Petr Křemen. *JOPA: accessing ontologies in an object-oriented way.* SciTePress, 2015.

[41] Tomaszuk, Dominik, and David Hyland-Wood. *RDF 1.1: Knowledge representation and data integration language for the Web.* Symmetry, 2020.