

**FACULTY
OF MATHEMATICS
AND PHYSICS**
Charles University

SOFTWARE PROJECT

LinkedPipes Applications: Detailed Specification

Bc. Altynbek Orumbayev, Bc. Esteban Jenkins,
Bc. Ivan Lattak, Bc. Marzia Cutajar, Bc. Alexandr
Mansurov

Title: LinkedPipes Applications

Students: Bc. Altynbek Orumbayev, Bc. Esteban Jenkins, Bc. Ivan Lattak, Bc. Marzia Cutajar, Bc. Alexandr Mansurov

Faculty: Faculty of Software Engineering

Supervisor: RNDr. Jiří Helmich

Consultants: Doc. Mgr. Martin Nečaský, Ph.D., RNDr. Jakub Klímek, Ph.D.

Annotation: The goal of the project is to create a new component of the LinkedPipes platform - LinkedPipes Applications. The tool should be focused on easily letting non-technical users work with data published according to the Linked Data principles. This new application will provide the user with a simple web interface that will orchestrate other tools from the LinkedPipes platform on the background, especially LinkedPipes ETL and LinkedPipes Discovery.

Contents

1	Introduction	3
1.1	Previous work	3
1.1.1	LDVM	3
1.1.2	LinkedPipes Discovery	4
1.1.3	LinkedPipes ETL	4
1.1.4	LinkedPipes Assistant	4
1.2	Structure	4
2	Requirements	5
2.1	Functional Requirements	5
2.1.1	User authentication	5
2.1.2	Create Application	6
2.1.3	Configure Application	8
2.1.4	Publish and Embed Application	9
2.1.5	View Applications	9
2.1.6	Visualizers	9
2.2	Non-functional Requirements	12
2.2.1	Error and Exception Handling	12
2.2.2	Reliability under large datasets	13
2.2.3	Continuous Integration / Continuous Delivery	13
2.2.4	Testing and Code Coverage	15
2.2.5	Coding Conventions	15
2.2.6	Code Quality	16
2.2.7	Secure access	16
3	Architecture	17
3.1	Overview	17
3.2	Backend	18
3.2.1	Logical structure	18
3.2.2	Code structure	21
3.3	Frontend	21
3.3.1	Code structure	23
3.4	Component Integration and Communication	23
3.4.1	Discovery	23
3.4.2	ETL	24
4	Technologies	25
4.1	Frontend	25
4.1.1	Frontend development stack	25
4.2	Backend	26
4.2.1	Backend development stack	26
4.3	Database and SOLID storage	28
4.4	CI/CD	29
4.4.1	CI/CD development stack	29

5 Project Execution	31
5.1 Difficulty estimation	31
5.1.1 Sprints	31
5.1.2 Project implementation plan	32
List of Figures	33
List of Tables	34
Glossary	35
Acronyms	36

1. Introduction

Linked data is a method for publishing structured data in a way that its semantics are also expressed. This semantic description is implemented by the use of vocabularies, which are usually specified by the W3C as web standards. However, anyone can create and register their vocabulary, for example in an open catalog like LOV.

Linked data is usually dispersed across many sites on the internet. Each site usually contains only a part of the entire data available, thus a machine or a person trying to interpret the data as a whole needs to link this partial information together using unique entity identifiers shared across the data stores, hence the name ‘linked’ data.

As more and more open data is published as Linked Data, there has emerged the need for a tool that allows people without any technical knowledge of Linked Data, RDF and other related technologies, to consume this data. The primary goal of this project is to develop such a tool, in which with a simple web interface any user with no prior experience with concepts of Semantic Web¹ can explore, visualize and interact with Linked Data.

1.1 Previous work

The application will interact with other components of the LinkedPipes ecosystem that were previously developed: LinkedPipes ETL and LinkedPipes Discovery. Moreover, this project is based on methods and ideas used in LPAssistant. In this section, these components will be briefly described, and they will be further explained in Chapter 3.

1.1.1 LDVM

Linked Data Visualization Model (LDVM) is essentially an abstraction of the process of visualizing linked data. In this model, there are four stages:

1. **Transform data sources:** the input data, whose format is not bound to Resource Description Framework (RDF), is transformed into an RDF representation to be used in subsequent stages.
2. **Analytic abstraction:** in this stage, the relevant data is extracted from the RDF representation, and can be further transformed or aggregated.
3. **Visualization abstraction:** a further abstraction is generated based on the model of the previous step, applying more transformations to the data such that it suits a particular type of visualization (or visualizations).
4. **View:** the generated visualization abstraction is simply converted into a graphical representation and displayed to the user.

These sequential four steps make what is called a Pipeline: a process in which the raw input data is transformed until it is displayed on-screen to a user with a particular visualization.

¹https://en.wikipedia.org/wiki/Semantic_Web

1.1.2 LinkedPipes Discovery

LinkedPipes Discovery component is a Scala application that takes a random data sample from a dataset and *discovers* (hence the name) different Pipelines that can be applied to it. In this process of discovery, the semantics of the data are used to identify different transformations that can be made. Essentially it discovers the different pipelines that can be applied to a given dataset.

1.1.3 LinkedPipes ETL

LinkedPipes ETL is an RDF-based, lightweight Extract, Transform and Load (ETL) tool that was developed by a team of Ph.D. students and researchers from computer science universities in Prague. It is not only a stand-alone application, but it also exposes an API through which third-parties can execute the ETL process.

1.1.4 LinkedPipes Assistant

This software that was also developed as a master thesis of the Department of Software Engineering of Charles University is an assistant to let users create interactive views of RDF data sources selected by the user. The Assistant analyzes the data and offers a list of visualizers that can be used to display the data in a useful way.

Our project shares a similar goal with the one of LinkedPipes Assistant. However, the architectures of the projects vary greatly: LinkedPipes Assistant has its own embedded discovery and ETL, whereas our project interacts with LinkedPipes Discovery and LinkedPipes ETL, which are standalone and improved applications.

1.2 Structure

This document is structured into several chapters. Chapter 2 explains in detail the requirements for this project, both functional and non-functional. Then, Chapter 3 outlines the proposed architecture of the system: the internal and external components, connections and interconnections. Chapter 4 gives an overview of all the different technologies that will be used in the project. Finally, Chapter 5 describes the methodology that will be used for the development, the timeline of the project, and the implementation plan.

2. Requirements

In this chapter are described both the functional and non-functional requirements for the system. Functional requirements are those that describe what the system should do. Non-functional requirements describe how the system should work.

The overall goal is to create a web-based tool that would allow generation of interactive visualizations by some domain expert, that can then be embedded in an online article or on another web page or perhaps simply accessed as a standalone page.

To avoid confusion with naming, in this chapter "tool" will refer to the actual application that is to be developed for this project, while "application" or "applications" refer to pre-configured interactive visualizations that a user creates using the tool.

2.1 Functional Requirements

This section describes the essential functionality that the system must provide as user stories.

2.1.1 User authentication

As a user of the tool, I must be able to register an account in the application, log in and log out. Moreover, once I am logged in, I should be able to create, configure and publish applications. To view an application, however, it is not necessary to be logged in or even registered.

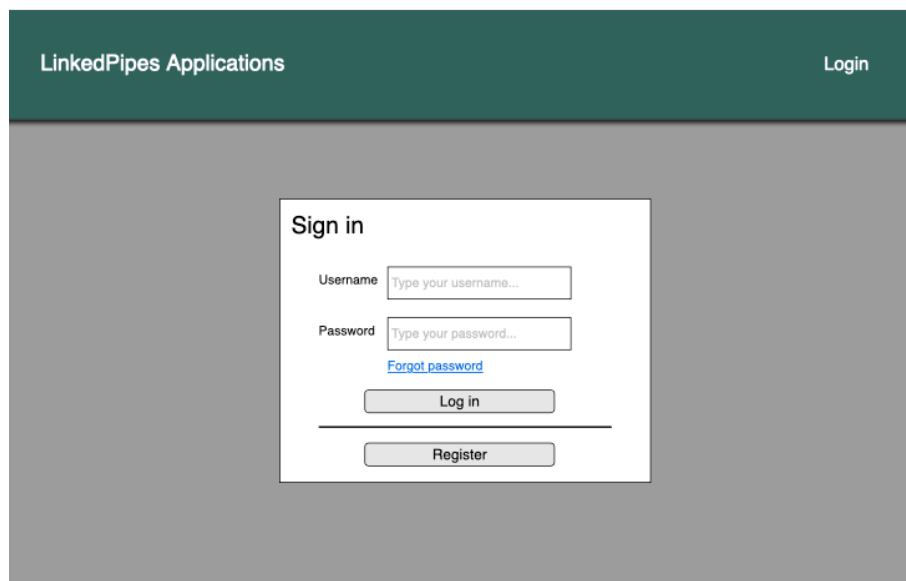
The image shows a web page mock-up for 'LinkedPipes Applications'. At the top, there is a dark green header bar with the text 'LinkedPipes Applications' on the left and a 'Login' link on the right. The main content area has a light gray background. In the center, there is a white rectangular box titled 'Sign in'. Inside this box, there are two input fields: 'Username' with the placeholder text 'Type your username...' and 'Password' with the placeholder text 'Type your password...'. Below the password field is a blue link that says 'Forgot password'. Underneath the link are two buttons: 'Log in' and 'Register'. A horizontal line separates the 'Log in' button from the 'Register' button.

Figure 2.1: Mock-up of the login page, where users get authenticated

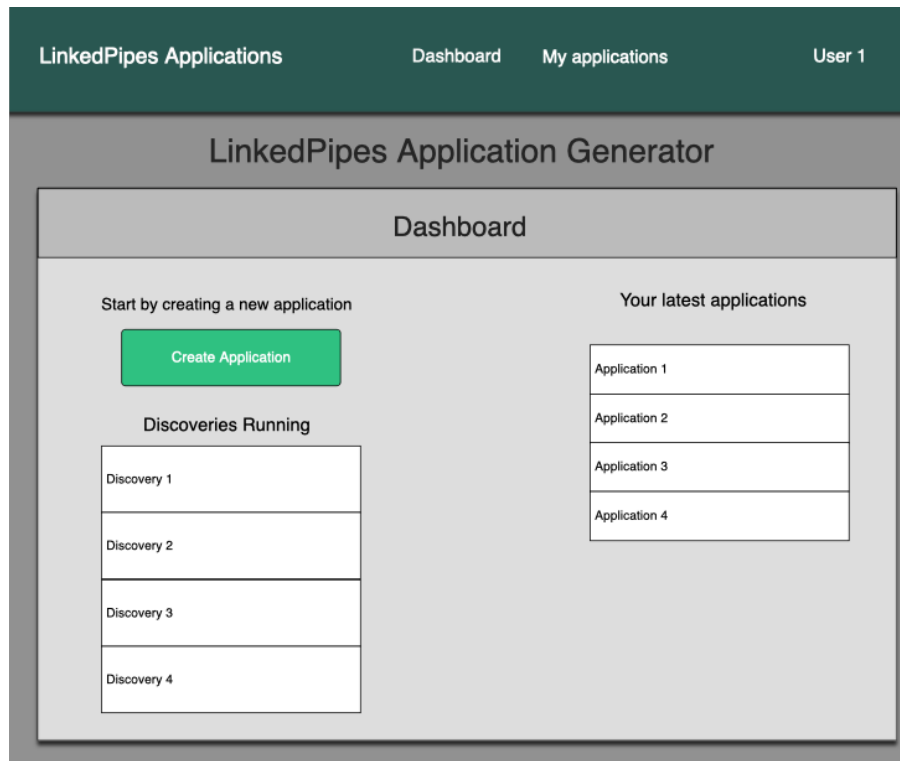


Figure 2.2: Dashboard of an authenticated user

2.1.2 Create Application

This section uses the following terms and acronyms:

- Internationalized Resource Identifier (IRI) – IRIs are a superset of Uniform Resource Identifiers (URI) which allow for the inclusion of Unicode characters such as Chinese or Cyrillic symbols in the identifier string. IRIs are extensively used as entity identifiers in Linked data.
- SPARQL endpoint is an interface through which a user can query and inspect data stored in a particular RDF data storage.
- dataset is a collection of data available for access or download from a single data store such as a catalog or a SPARQL endpoint.
- TTL – short for Terse RDF Triple Language, one of the RDF serialization formats.

As a user, I want the platform to provide the functionality to specify the data sources to be utilized so that an instance of an interactive application would be created based on provided data sources. To do so, I need to have four alternatives:

- Provide a set of dataset IRIs which the tool will de-reference to get the dataset.
- Specify a SPARQL endpoint from which data will be queried and extracted.
- Upload a file in TTL format, containing data source specifications.
- Use some sample dataset provided by the tool

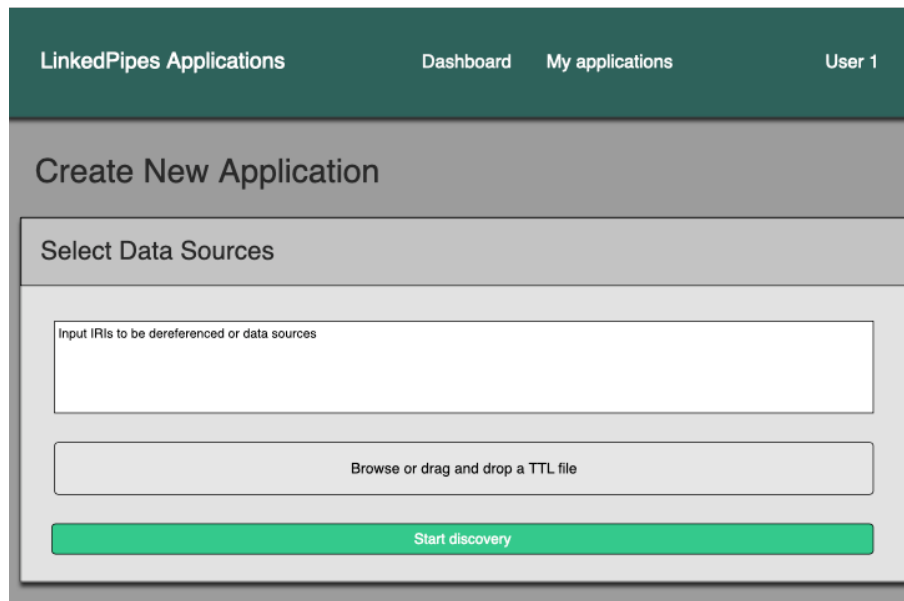


Figure 2.3: Mock-up of the step where the data sources are selected

The tool has to be able to obtain and consume RDF data in any of the above forms and analyze it to provide a list of possible visualization types automatically. Then I will be able to choose one of the given visualizations so that the newly created application will display the data according to the selected visualizer. The generated visualization will not be a static view, but rather an interactive one with auto-generated controls.

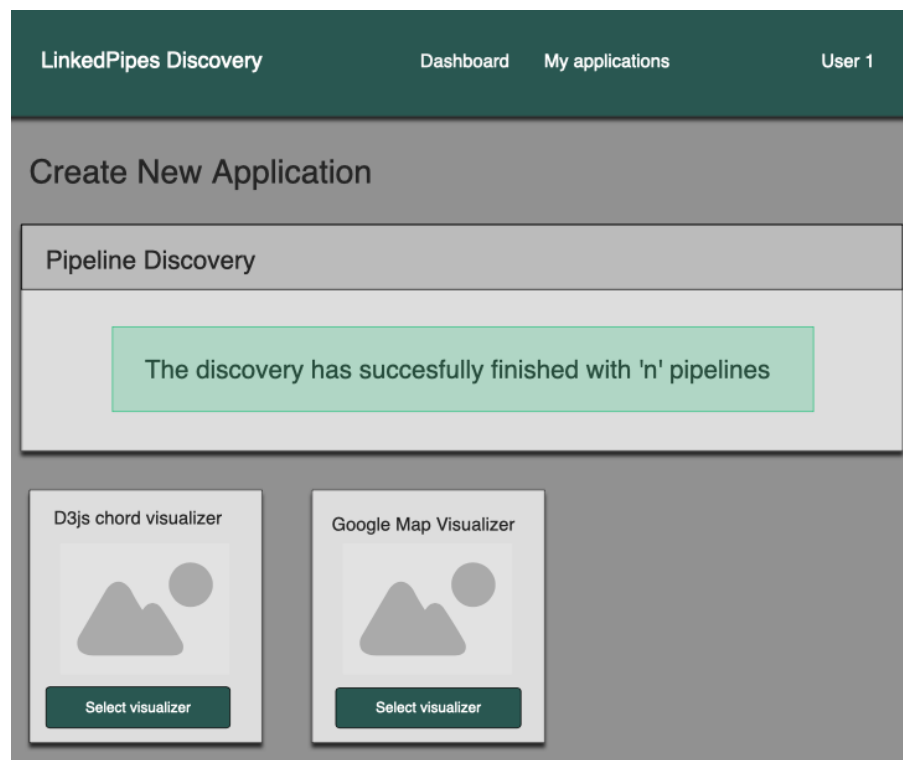


Figure 2.4: Mock-up of the step where the user selects the desired visualization

Before creating the actual application, the user will be provided with an option to

preview the selected visualization. This preview will use a subset of the processed data available for the application creation to increase loading and rendering speed of the preview.

It is important to note that the size of the data sources is not known in advance, which means that the tool has to be able to work smoothly with any size of data and allow its user to browse all of it.

2.1.3 Configure Application

As a user, I should be able to configure a previously created application. Each application, depending on the visualization, will have particular settings that can be set. Furthermore, I must be able to control, with the use of filters, which data is to be used and displayed. The available filters should be automatically derived based on the properties and semantics of the data. This is a list of the things I should be able to do with filters:

- Removing whole data filters to ignore the values of specific data properties
- Removing selected values of some filters
- Setting fixed values for some filters
- Set/change a name for the application

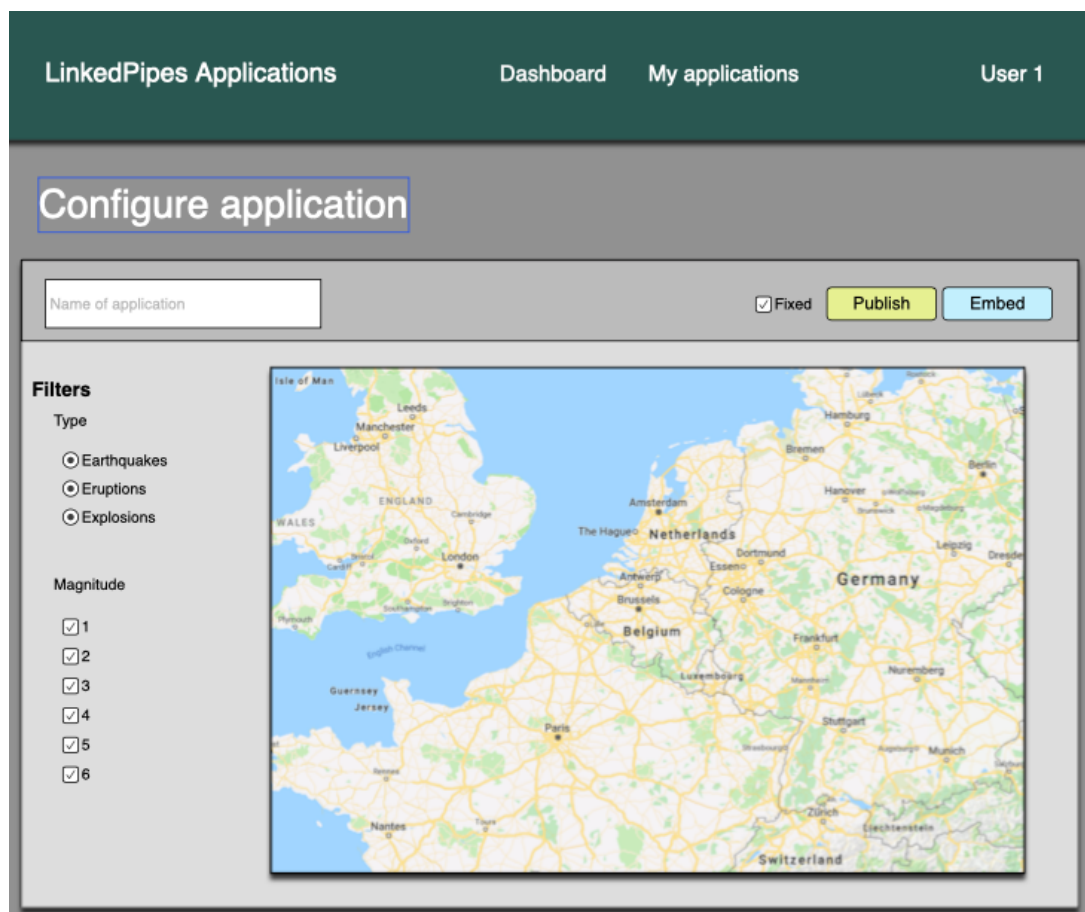


Figure 2.5: Mockup of the configuration of an application

It is also important that this configuration is persisted in a database together with the general application details so that I can publish it exactly as I configured it. This way, the viewers of my application are not presented with a random subset of the data, nor the complete data set, but only with those data chosen by me.

2.1.4 Publish and Embed Application

As a user, I want the tool to provide an interface for publishing the configured interactive application so that it will be possible to host the data view and make it accessible using a permanent link. Furthermore, when a third party clicks this *permalink*, the browser should open the LinkedPipes Applications website with the respective application opened, as configured by the publisher.

Besides, when publishing an application I want the tool to offer the possibility to choose one of the below two settings:

- use and display cached data, making the published application a fixed view
- regularly refresh the data from the previously chosen data sources of the interactive application

The tool will also need to provide the ability to embed the published view into a data journalist's web page, for example, using an `iframe`.

Furthermore, as a user, I want this publish and embed functionality to be provided in the form of buttons as can be seen in figure 2.5.

By publishing or embedding an application, the domain expert would be sharing the pre-configured application with end users who can access it as well as control it through a user interface with controls allowing simplified filtering capabilities. This will allow them to interact with and analyze the data.

2.1.5 View Applications

As a user of the tool, I must be able to manage all my previously created applications so that I want to have to repeat the process in every iteration. Therefore, there should be a page in the tool where these applications are listed, so that the user can perform several actions on any of them, that is viewing, editing, publishing, deleting, etc.

2.1.6 Visualizers

As a user, I must be able to choose the visualizer that best suits the data. For this, the tool must provide me with a list of visualizers that can depict the data extracted from the data sources. I should be able to have at least the following visualizers:

Geo-coordinates Visualizer

Geo-Coordinates Visualizers are a set of visualizers designed for representing data related to geolocation and coordinates on a map. The LinkedPipes Applications platform should provide the ability for a user to create applications based on geo-coordinates visualizers. A sample representation of the visualizer is presented in Figure 2.6.

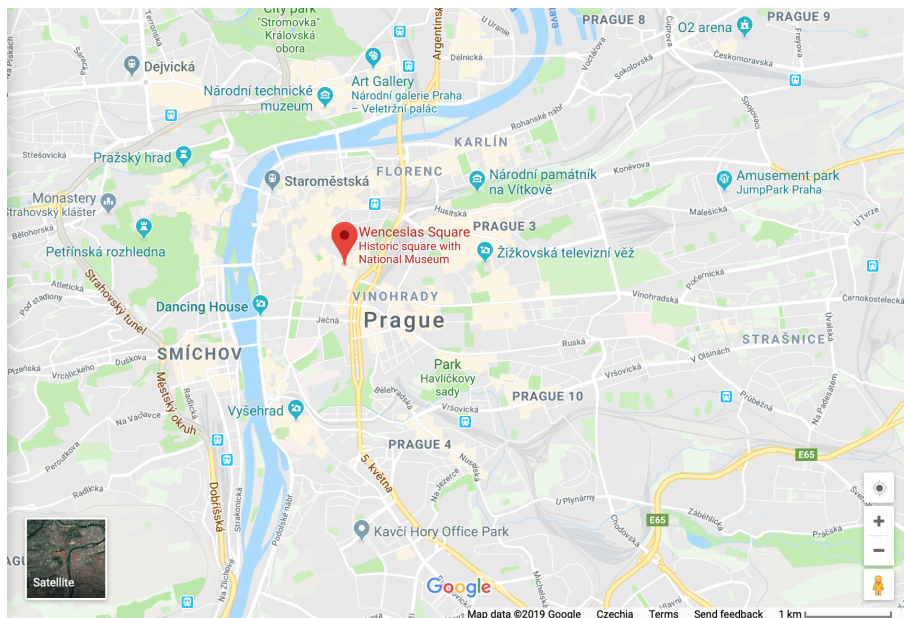


Figure 2.6: The example of geo-coordinates visualizer implemented with GoogleMaps

The specific set of requirements for that category can be described as follows:

1. Must be able to process data represented as a tuple of latitudes and longitudes called 'markers'.
2. Must be able to visualize and display markers on the map.

Chord

Chord visualizer is used for efficient and descriptive visualization of directed or undirected relationships within a group or between multiple groups of entities.

One particular example of such a relationship is a network data flow diagram. Figure 2.7 shows a Chord diagram visualizing the data flow between four network peers over a period of time. A larger outer circle sector signifies higher data bandwidth of that entity. A thicker colored strip between two sectors means higher traffic between those two peers.

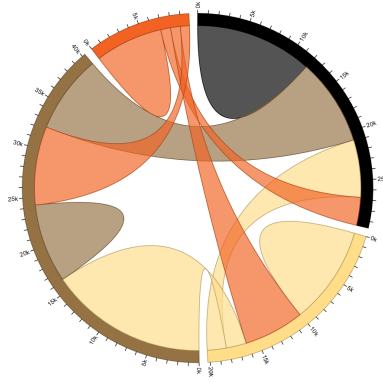


Figure 2.7: D3.js Chord visualizer example

Hierarchical

The nature of some data is hierarchical, and as so, there must be a visualizer able to represent these hierarchical relationships in a visual way. Several visualization methods can be implemented:

1. Treemap
2. Sunburst
3. Tree diagram



Figure 2.8: An example of a treemap, sunburst and tree diagram visualization

At least one of these visualizers must be implemented. As a user, I should be able to drill down in the hierarchy to see more details about the subset of the data that is being visualized.

Timeline

Certain data can have a time value or interval attached to individual data points, and the user might be interested in visualizing the values over time. The simple case to be implemented is when values are numerical.

There are several ways how to show such data (see Figure 2.9 for samples):

1. simple line graph

2. stacked area chart
3. bar chart
4. calendar view/heat map
5. stream graph

At least one of such visualizers should be implemented.

As a user, I might be interested in showing values from a specific period and also in filtering of data points according to additional attributes.



Figure 2.9: An example of a simple line graph, stacked area chart, bar chart, calendar view, and stream graph visualization

2.2 Non-functional Requirements

This section describes the steps that will be taken to ensure that the system operates and behaves correctly, as well as the criteria used to judge its quality.

2.2.1 Error and Exception Handling

It is important to have robust exception-handling code to preserve execution flow in the system, as well as to provide enough details to users and developers for them to understand what went wrong.

It was decided that a custom exception class will be used on the backend side to throw exceptions with appropriate error codes and messages. These exceptions will then be caught by a global exception handler, which if applicable will log the original exception, and return the desired error code and message back to the frontend. In this way, the users will not be shown error messages that they cannot understand, while the system developers and administrators can refer to the history of error logs to identify and resolve potential bugs.

2.2.2 Reliability under large datasets

This non-functional requirement implies that all components of the system must be able to handle a large amount of data to guarantee a good user experience. In general, several optimizations will need to be done at each stage of the process so that only the minimum data is used to minimize processing time and bandwidth usage.

2.2.3 Continuous Integration / Continuous Delivery

CI systems automate the builds of software. Travis CI will be used to check that newly committed code does not break the build and consequently the system. This will ensure that developers are not disrupted and that the system remains stable. It will also run the automated tests available to further check that the system is working correctly, even if the build didn't break.

Another requirement is to have a continuous delivery mechanism in place. This is an extension of the continuous integration to make sure that new changes can be released quickly and in a sustainable way. Thus, the release process will be automated so that the application can be deployed at any point in time by the simple click of a button.

Docker deployment

As a part of continuous delivery, it is assumed that each of the implemented components of LinkedPipes Applications is going to use Docker for containerizing components as micro-services interacting with each other independently. Therefore, the consequent requirements can be defined as follows:

1. Each component of LinkedPipes Applications should have a dedicated *Dockerfile* and have a dedicated repository in DockerHub for hosting the latest images.
2. Every start of deployment of latest changes to production server, should start with an automated process of building the Docker images.
3. The end of the automated deployment should push the latest images to DockerHub and notify the production server to pull the latest images.

Automated development flow

The following is a description of the various configurations that will be used for automated testing and delivery of the latest code of *backend* and *frontend* components.

The *master* and *develop* branches require usage of pull requests: no one within the team has access to push changes directly to those branches. *Develop* will be mainly used as an aggregator of all stable and relatively stable changes before merging into master. Any pull request to those branches trigger Travis CI that executes automated tests in parallel for *backend* and *frontend* components.

The figure below represents a general development flow when interacting with the *master* branch. Whenever tests with the latest codebase changes pass on pull-request, Travis

CI also automatically makes new Docker images of *frontend* and *backend* components and pushes them into DockerHub.

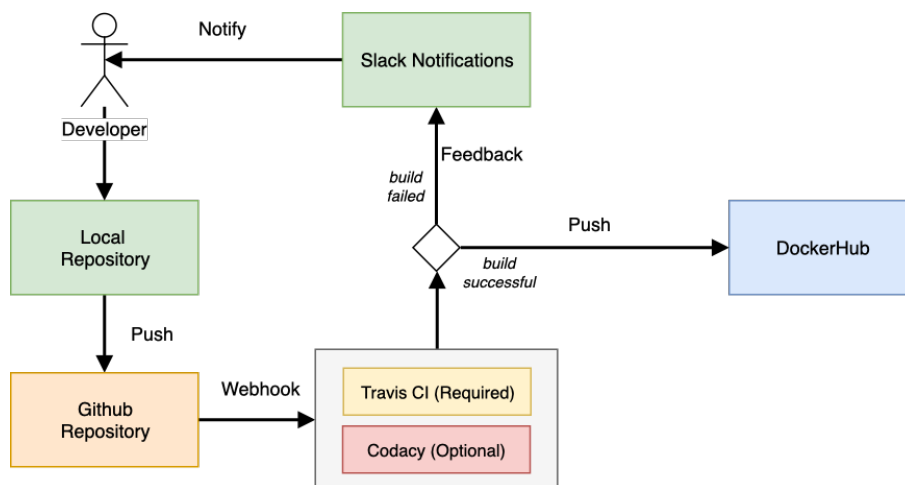


Figure 2.10: Continuous delivery flow on a pull request to the *master* branch

The flow for branches other than *master* slightly differ as can be observed in the figure below. The main difference is that it is only used to report test results for committed changes to the dedicated *Slack* channel. Developers then can manually observe the reports and merge the changes if they satisfy the requirements.

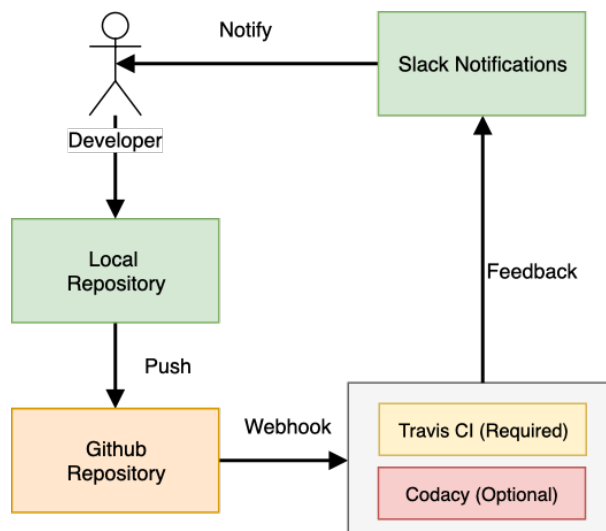


Figure 2.11: Continuous delivery flow on a pull request to any branch except *master*

Aside from the interactions with Travis CI, upon every merge completion into the *master* branch, there is a separate Github Webhook.

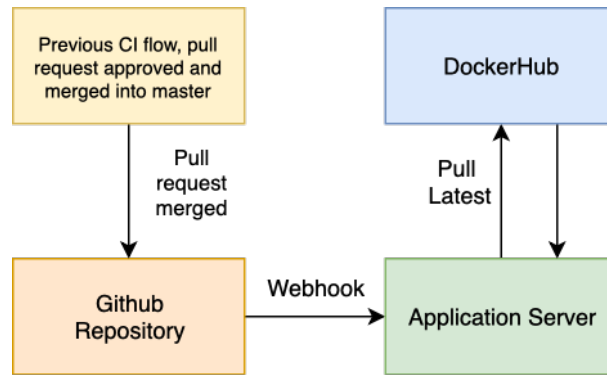


Figure 2.12: GitHub webhook flow used for continuously pulling latest *frontend* and *backend* images from DockerHub

This Webhook triggers a script on Application Server (the server on which production ready platform is hosted for end users) to pull the latest *frontend* and *backend* images from DockerHub. This process can be observed in figure 2.12.

2.2.4 Testing and Code Coverage

Code coverage is merely a measure of the proportion of the code that is executed in a test suite. The project should have a code coverage above 75%, including both unit testing and integration testing.

Having a testing plan that provides a good code coverage is extremely important to detect bugs in early-stage of the development and ensure that the system is as error-free as possible. Because of the architecture of the tool, the testing is done differently for the backend and the frontend.

The backend testing will be done using Spring Boot’s test utilities as well as JUnit for the unit and integration testing.

For the frontend, the testing will be done using React’s TestUtils, which essentially renders a React component by putting its DOM in a variable in which several things can be checked, such as:

- whether or not all children were rendered
- conditional statements are working as expected
- styling is applied as it should
- the content of elements is correct

2.2.5 Coding Conventions

Code conventions are important as they improve software readability and maintainability. The code must be well-commented and easy to understand, especially for someone who is unfamiliar with the project.

Java code conventions will also be used for good code readability on the backend side. For the frontend, the Airbnb React/JSX Style Guide will be followed and ensured using a linter.

2.2.6 Code Quality

Ensuring high code quality makes system maintainability much easier. For this reason, it was decided that the Codacy tool will be used to automate code reviews and monitor code quality in git commits and over time.

Appropriate design patterns will be used to make sure the system developed is made up of cohesive modules with minimal coupling. This will make the overall system easier to understand and maintain.

2.2.7 Secure access

The connection between the user's browser and the server hosting the application must be secured using an SSL/TLS certificate created specifically for the **application.linkedpipes.com** domain. This certificate will be generated using Let's Encrypt since it offers free industry-standard certificates and it is a trusted Certificate Authority (CA).

3. Architecture

In this chapter, the project will be analyzed and described from the architectural point of view. The main modules of the project will be identified, and their functionality will be determined. Each module will be divided into components and described in more detail. For more detailed information about technologies and terms used in this chapter, refer to chapter 4.

3.1 Overview

The LinkedPipes Applications is a platform that consists of multiple components actively interacting with each other. Figure 3.1 represents a high-level overview of all components defined within LinkedPipes Applications. In general, the following content of this section will define each of the components presented in figure 3.1.

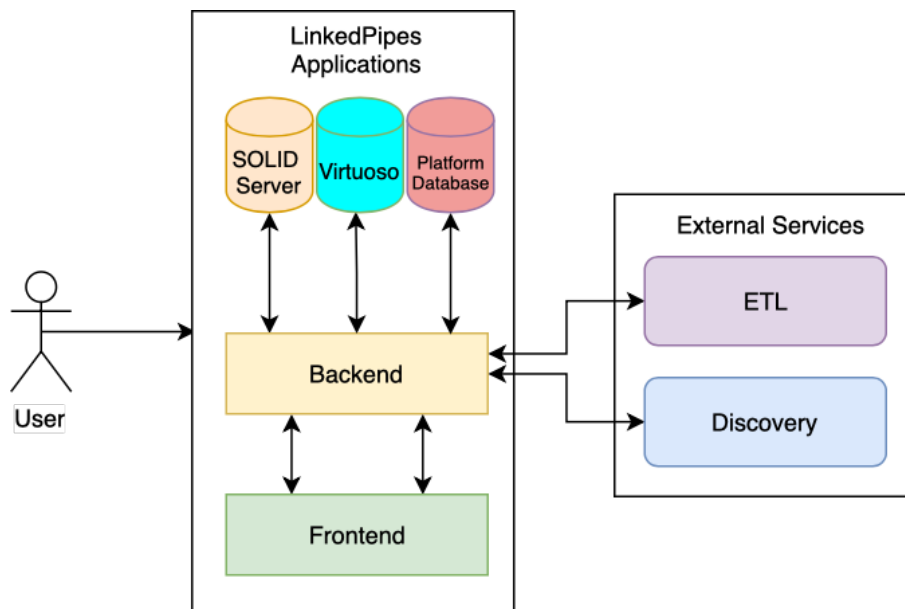


Figure 3.1: High-level overview of LinkedPipes Applications

- **Frontend** – the web application that provides a way for the user to interact with the LPA. Written in React.js. Main functionality includes features such as:
 - Adding, deleting and modifying data sources.
 - Displaying the discovery results for the end user.
 - Discovering pipelines.
 - Executing a pipeline.
 - Publishing an application.
- **Backend** – the backend application written in Java using Spring Boot. Main functionality includes:

- Communication with the Discovery Service and the ETL Service.
- Datasource management
- Discover pipelines for the given set of data sources.
- **Discovery** – a backend application of which the task is to discover pipelines for a set of data sources it receives from the LPA backend.
- **ETL** – an Extract Transform Load service for Linked Data. Generally, each term can be described as follows:
 - ‘Extract’ - Extracts the data from a certain source to a given application or a running process.
 - ‘Transform’ - Transforms the source data to target representation.
 - ‘Load’ - Loads the data from the given application or a process outside - web server.
- **Virtuoso** – the RDF Store is used to store data results from the ETL Service, and data is retrieved from this storage for visualization to the end user in the frontend application.
- **Platform Database** – the PostgreSQL database used for all user account related information, user sessions, templates for assembling the visualizers, etc.
- **SOLID Server** – an instance of a SOLID server holding so-called *pods*, that represent decentralized personal storages for holding all applications created with Linked-Pipes Applications platform.

3.2 Backend

The function of the backend component is to provide a RESTful API which is used by the frontend component to execute user-requested actions and can also be used by other developers to create their user applications. Backend then implements the communication protocols with external services like LP-ETL, LP-Discovery or databases.

3.2.1 Logical structure

As mentioned, the function of the backend component is to provide a RESTful API which is used by the frontend component to execute user-requested actions and can also be used by other developers to create their user applications. To fulfill this goal the backend component is further divided into multiple sub-components of different logical types. These logical types are *controllers*, *services*, *models*, *query providers*, and *result extractors*.

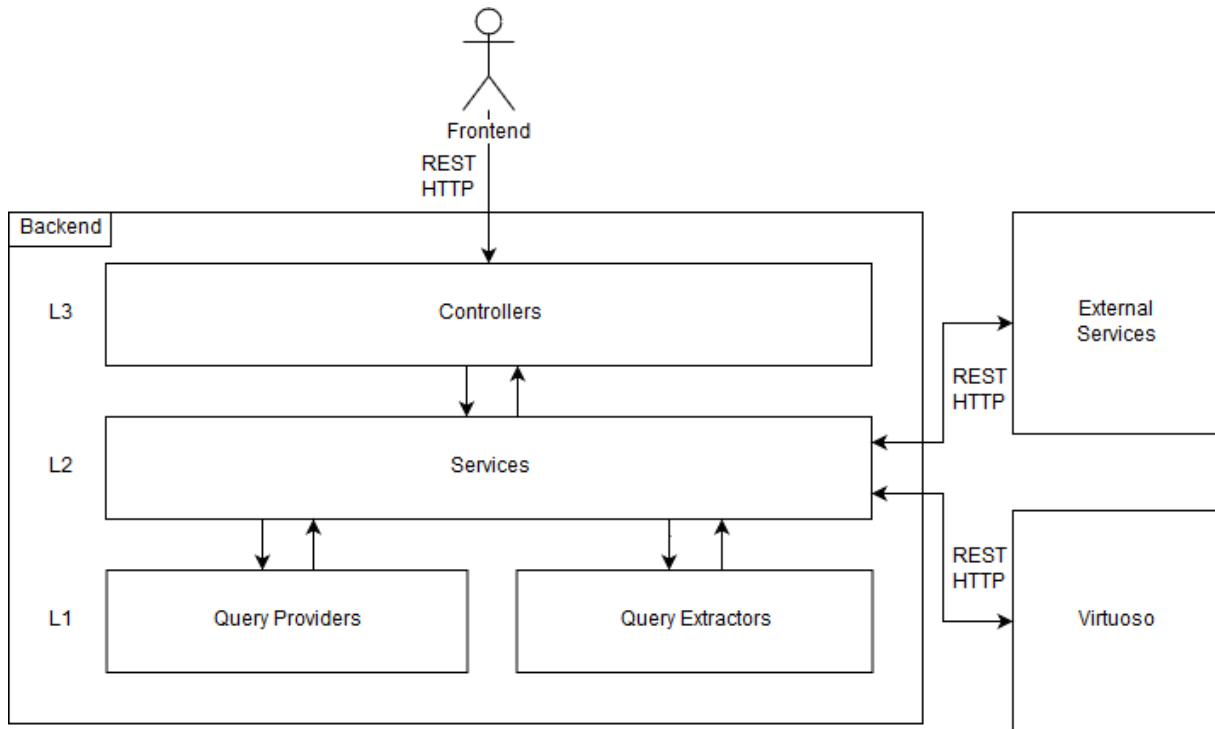


Figure 3.2: Backend component architecture overview

Controllers

Controllers are sub-components responsible for identifying HTTP requests and mapping them to appropriate service sub-components. In a layered view of the backend architecture, a controller is located in the top-most layer L3, as seen in Figure 3.2. In detail, a controller’s responsibilities are to:

- define RESTful API endpoints,
- read HTTP requests to determine the correct function to call,
- extract parameters from the requests,
- invoke appropriate services with the extracted parameters,
- process output from services, and
- send appropriate HTTP responses.

Services

Services are intermediate sub-components which facilitate internal functions of the component to the controller. Because they are used to implement the endpoints of a RESTful API, service sub-components shall be kept stateless at all times to better facilitate the statelessness requirement of RESTful requests. In a layered view of the backend architecture, a service is located in the intermediate layer L2, as seen in Figure 3.2.

Services can be either external or internal. External services are LinkedPipes Discovery, LinkedPipes ETL and SOLID Storage. Each service has an interface and implemen-

tation class. For external service, the implementation class calls the external service by executing the appropriate HTTP request.

Internal services inspect and operate on data stored in the Virtuoso RDF graph store, which is generally a set of data that the user of our tool wants to visualize in an application, and is usually a product of an LP-ETL pipeline execution. The internal services perform inspection using SPARQL queries. The queries themselves are being supplied by the subordinate query provider sub-components. The results of the inspections are then extracted from the RDF responses by sub-components of type result extractor.

Models

Models provide classes for data representation across the backend component. Due to the nature of LinkedPipes Applications, we need several types of models:

- models related to external services for data passed to those services,
- models related to RDF data for internal processing, and
- models related to data presentation for data passed to the frontend component.
- A special type of a model is an exception, used to carry information about an error condition occurring in the component.

Query providers

Query providers are subcomponents responsible for supplying the service sub-components with parameterized SPARQL queries used to inspect and manipulate data stored in the Virtuoso RDF graph store.

Among other uses, these will enable retrieval of specific types of data from an RDF graph (such as geographical coordinates, SKOS concepts, etc.) to aid in visualization and filtering of data in created applications.

These sub-components are located on the bottom layer, L1, in the layered model of the backend component architecture.

Result extractors

Result extractor sub-components are utilized by the service sub-components to transform visualization data from the result set returned by the query execution into a format displayable by the frontend visualizers.

Result extractors are logically placed on the L1 layer of the layered model of the backend component architecture.

3.2.2 Code structure

The implementation of the whole backend component is intended to be located inside the `src/backend` base directory. It is a Gradle-based project, thus it shall follow the Gradle directory structure:

```
src/
|-- main/
|   |-- java/           for the Java source code of the component
|   |-- resources/      for the related classpath resources
|-- test/
|   |-- java/           for the test source code of the component
|   |-- resources/      for the related classpath resources
```

Backend main module (`src/main/`)

The main module of the backend component shall contain the entire functional source code of the backend component. It shall be composed of the following java packages. Various logical types of the sub-components of the backend component are mentioned in this list. For a detailed explanation of the responsibilities of each sub-component type, please consult section 3.2.1.

- `com.linkedpipes.lpa.backend.controllers` shall contain classes whose instances fulfill the responsibilities of the controller type sub-components,
- `com.linkedpipes.lpa.backend.services` shall contain classes whose instances fulfill the responsibilities of the service type sub-components,
- `com.linkedpipes.lpa.backend.sparql.queries` shall contain classes whose instances fulfill the responsibilities of the query provider type sub-components,
- `com.linkedpipes.lpa.backend.sparql.extractors` shall contain classes whose instances fulfill the responsibilities of the result extractor type sub-components,
- `com.linkedpipes.lpa.backend.util` shall contain additional utility classes providing functionality utilized by the other sub-components.

Backend test module (`src/test/`)

The test module of the backend component shall contain the entire source code for unit tests testing the correct functionality of classes contained in the backend main module. It shall be composed of java packages similarly matching those described for the main module so that it is evident which tests test which group of components and classes.

3.3 Frontend

As mentioned at the beginning of the chapter, the frontend provides a way for the user to interact with the LinkedPipes Applications. Redux and React are the leading

technologies to be used during implementation. Therefore, the architecture on figure 3.3 is demonstrated from the viewpoint of interactions between those technologies.

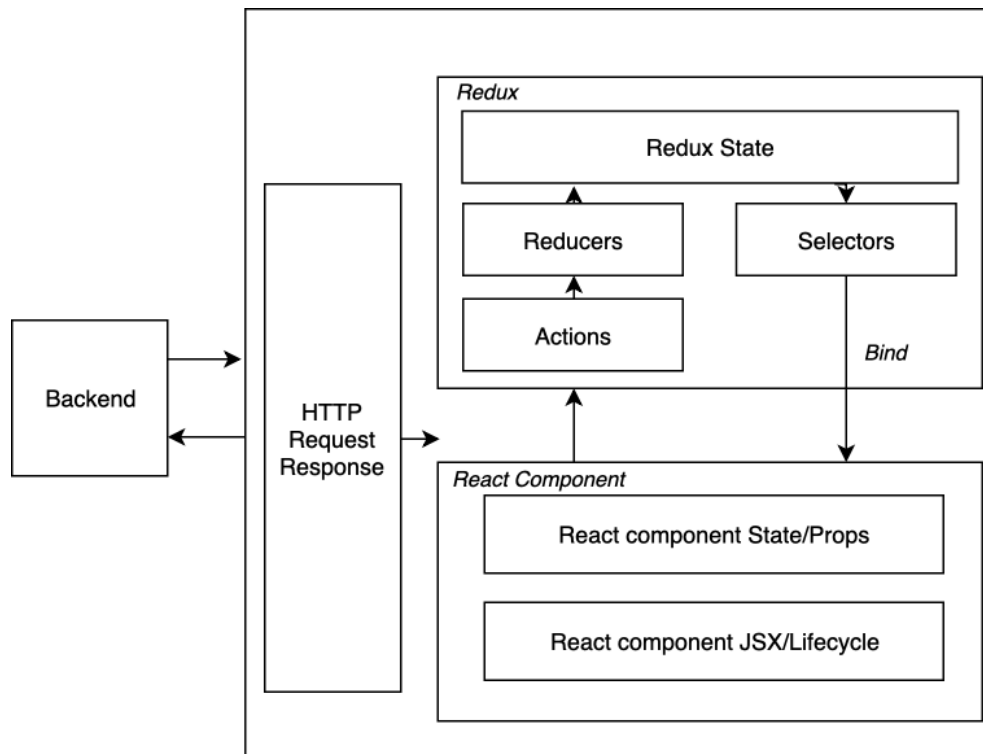


Figure 3.3: General architecture of Redux store and React components within *frontend*

The main elements displayed on 3.3 are described as follows:

- **React Component** – a JavaScript class or function that optionally accepts inputs, i.e., properties(props), and returns a React element that describes how a section of the UI (User Interface) should appear.
- **Redux** – represents a container that stores various states of the web application per individual webpage.
 - **State** – refers to the single state value that is managed by the store and returned by `getState()`. It represents the entire state of a Redux application, which is often a deeply nested object.
 - **Reducer** – specifies how the application’s state changes in response to actions sent to the store.
 - **Actions** – are payloads of information that send data from your application to your store. They are the only source of information for the store. You send them to the store using `store.dispatch()`.
 - **Selector** – is simply any function that accepts the Redux store state (or part of the state) as an argument, and returns data that is based on that state.

3.3.1 Code structure

The implementation of the frontend component is intended to be located inside `src/frontend` and will contain the following structure:

- **__actions** – shall contain all Redux actions.
- **__constants** – shall contain all constants used throughout the frontend component implementation.
- **__helpers** – shall be dedicated to various handy utility classes, variables, and methods.
- **__reducers** – shall contain all Redux reducers.
- **__selectors** – shall contain all Redux selectors.
- **__services** – shall contain a set of wrappers for performing calls to Backend API that communicates with Discovery and ETL services.
- **__styles** – shall contain global setup for various components as well as the specification for the MaterialDesign theme used in the project.
- **components** – essentially will be the core of the *frontend*. Shall contain all `.jsx` components, visualizers and various UI elements implementations.
- **containers** – shall define layouts for specific web-pages of the web app.

3.4 Component Integration and Communication

The LinkedPipes Applications relies on a set of external components called ETL and Discovery. For more details about definitions of those components refer to the glossary. This section is used to describe and demonstrate how LinkedPipes Applications interacts and incorporates those components inside the platform. The figure 3.4 demonstrates one of the primary interaction flows that is happening when *backend* receives the list of resources from a user and starts interacting with *Discovery* and *ETL* to extract and process the information for visualizers.

3.4.1 Discovery

The purpose of *Discovery* within the LinkedPipes Applications is the processing of the provided datasources to identify whether the platform can provide any of the supported visualizers for the extracted data. The general flow can be described as follows:

1. *Discovery* extracts a small chunk of the data from the whole dataset found in Linked Open Data Cloud (LOD Cloud).
2. A set of validation operations performed on the extracted chunk.
3. Response to the *backend* provided as a list of datasources for which supported visualizers could be used.

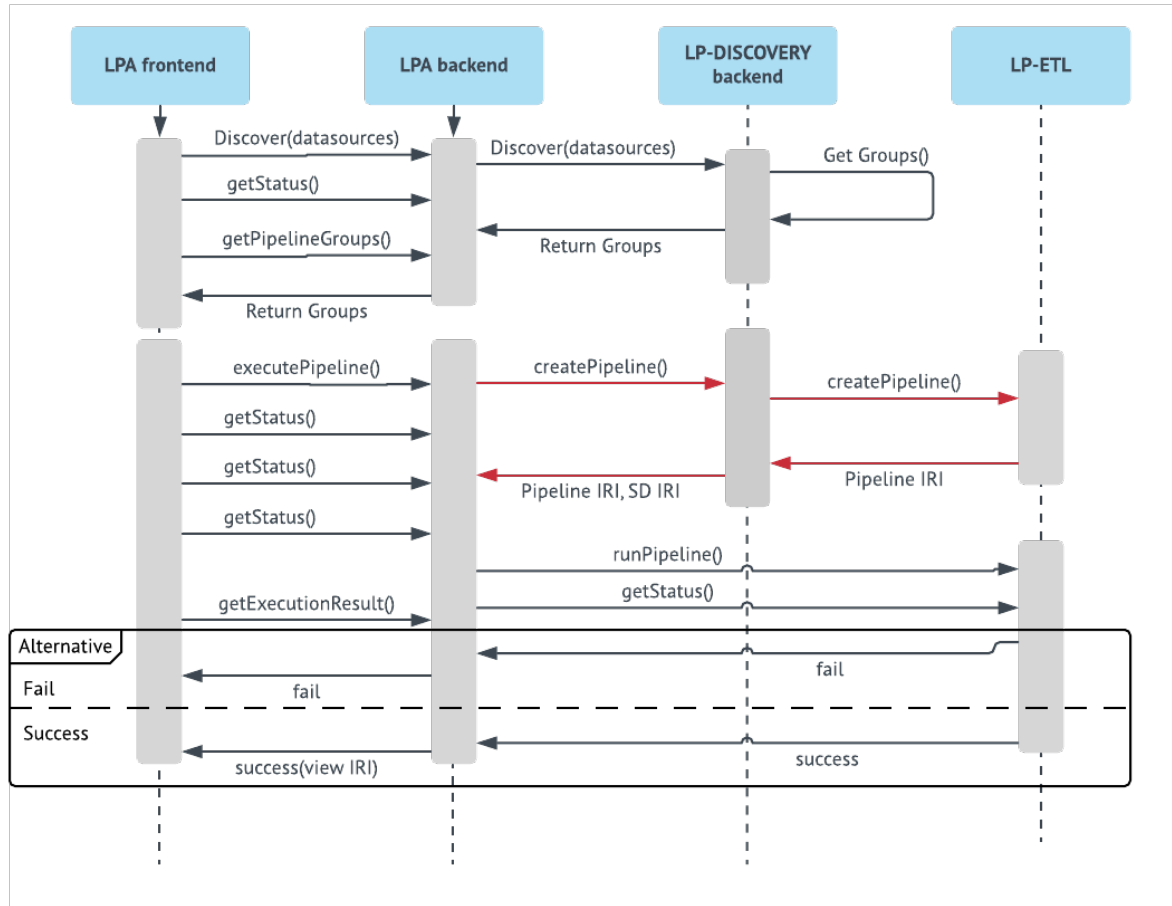


Figure 3.4: Sequence diagram of API interactions between *backend*, *Discovery* and *ETL*

3.4.2 ETL

The purpose of *ETL* within the LinkedPipes Applications is to prepare a dataset for a specific visualizer, that is later used by *frontend* to construct the visualizer and display the data. In general terms the work of *ETL* can be described as follows:

1. After *Discovery* finished processing and user selected supported datasource, *backend* sends the request to *ETL* to assemble a pipeline for that datasource.
2. After *ETL* assembled the pipeline, *backend* sends the request back to execute that pipeline.
3. After *ETL* finishes execution of the pipeline, prepared data for visualizer is exported into *Virtuoso*. *Backend* and *frontend* then use the data to continue the workflow.

4. Technologies

The following chapter is describing the main stack of technologies that will be used for development of LinkedPipes Applications. In general, the first set of descriptions describe technologies used for the frontend, and the consecutive set is dedicated to the backend, continuous deployment, and delivery setup.

4.1 Frontend

The web user interface of LinkedPipes Applications is one of the main components of the project. It provides a toolset for users to interact, preview, generate, store and publish their applications. JavaScript¹ was chosen as a primary development language due to the experience of the team with the language, as well as the availability of a large set of open-source libraries for interacting with LinkedData and SPARQL.

4.1.1 Frontend development stack

The LinkedPipes Applications frontend is entirely separated from the backend component. That means that all interactions with backend are performed over the RESTful API provided by backend implementation. The following section provides a detailed overview of the main development stack that includes such technologies as React², Redux³, and Material-UI⁴.

React

React (also known as React.js) is a JavaScript library for building user interfaces. React can also be used as a base in the development of single-page or mobile applications. Complex React applications usually require the use of additional libraries for state management, routing, and interaction with an API. The following list represents the main set of features provided by the framework.

- Declarative: React makes it painless to create interactive UIs. Design simple views for each state in your application, and React will efficiently update and render just the right components when your data changes. Declarative views make your code more predictable, simpler to understand, and easier to debug.
- Component-Based: Build encapsulated components that manage their own state, then compose them to make complex UIs. Since component logic is written in JavaScript instead of templates, you can easily pass rich data through your app and keep the state out of the DOM.

¹<https://www.javascript.com>

²<https://reactjs.org>

³<https://redux.js.org>

⁴<https://material-ui.com>

- **Learn Once, Write Anywhere:** We don't make assumptions about the rest of your technology stack, so you can develop new features in React without rewriting existing code. React can also render on the server using Node and power mobile apps using React Native.

Redux

Redux is an open-source JavaScript library for managing application state. It is most commonly used with libraries such as React or Angular for building user interfaces. Redux can also be described as a predictable state container for JavaScript apps. It allows to write applications that behave consistently, run in different environments (client, server, and native), and are easy to test.

Material-UI

Material-UI is a collection of React components that implement Google's Material Design.

4.2 Backend

The backend component of LinkedPipes Applications contains the main logic for interacting with such services as LinkedPipes Discovery and LinkedPipes ETL as well as communicating both with the Database and SOLID Storage.

4.2.1 Backend development stack

Java⁵ was chosen as the main development language due to extensive use of Spring Boot⁶ which is a Java-based open source library used for creating Micro Services. Aside from that, Gradle⁷ was chosen as the main build platform due to its simplicity and easy integration with various CI/CD aspects of the project.

Spring Boot

The Spring Framework is an application framework and inversion of control container for the Java platform. The framework's core features can be used by any Java application, but there are extensions for building web applications on top of the Java EE (Enterprise Edition) platform. Although the framework does not impose any specific programming model, it has become popular in the Java community as an addition to, or even replacement for the Enterprise JavaBeans (EJB) model. The Spring Framework is open source.

⁵<https://www.java.com/en/>

⁶<https://spring.io>

⁷<https://gradle.org>

Apache Jena

Apache Jena⁸ is an open source Semantic Web framework for Java. It provides an API to extract data from and write to RDF graphs. The graphs are represented as an abstract "model". A model can be sourced with data from files, databases, URLs or a combination of these. A Model can also be queried through SPARQL 1.1.

Jena is similar to RDF4J (formerly OpenRDF Sesame); though, unlike RDF4J, Jena provides support for OWL (Web Ontology Language). The framework has various internal reasoners, and the Pellet reasoner (an open source Java OWL-DL reasoner) can be set up to work in Jena.

Jena supports serialization of RDF graphs to:

- a relational database
- RDF/XML
- Turtle
- Notation 3

Gradle

Gradle is an open-source build automation system that builds upon the concepts of Apache Ant and Apache Maven and introduces a Groovy-based domain-specific language (DSL) instead of the XML form used by Apache Maven for declaring the project configuration. Gradle uses a directed acyclic graph (DAG) to determine the order in which tasks can be run. Gradle was designed for multi-project builds, which can grow to be quite large. It supports incremental builds by intelligently determining which parts of the build tree are up to date; any task dependent only on those parts does not need to be re-executed.

Lightbend Config

Lightbend Config is a configuration library for JVM languages. Due to extensive usage of various Docker configurations and usage of Docker Compose, the library was chosen as an optimal solution for passing various configurations to components before they are being loaded inside the container. Library provides a convenient set of features described as follows:

- Supports files in three formats: Java properties, JSON, and a human-friendly JSON superset merges multiple files across all formats.
- Can load from files, URLs, or classpath.
- Good support for "nesting" (treat any subtree of the config the same as the whole config).
- Users can override the config with Java system properties

⁸<https://jena.apache.org>

- Supports configuring an app, with its framework and libraries, all from a single file such as `application.conf`.

4.3 Database and SOLID storage

The LinkedPipes Applications platform is using a Database instance for storing all information related to users of the platform such as:

- User accounts.
- Running Discovery instances.
- Running ETL pipelines.
- Custom templates of Discovery datasources generated by users and etc.

PostgreSQL⁹ was chosen as the main technology for the database of LinkedPipes Applications. Open-source community support, the maturity of the solution as well as the previous experience of team members with that platform were the significant factors for choosing this technology.

Aside from PostgreSQL, another important database technology is OpenLink Virtuoso¹⁰ which is used for storing all LinkedData processed by *ETL* pipelines, and that is later used by LinkedPipes Applications visualizers. The important note to mention is that in contrast with PostgreSQL that is only used for all individual platform user-related information that needs to be stored, Virtuoso is only used for storing the LinkedData. Therefore, the usage of both databases is justified.

In addition to that, one of the unique features of LinkedPipes Applications is the usage of SOLID¹¹, which is a new project by Tim Berners Lee¹². Every application created with LinkedPipes Applications platform is encoded into RDF and stored inside a so-called SOLID *pod*, which is decentralized personal storage where the user can specify who can access that data. More details on the definition of SOLID will be provided later in the section below.

PostgreSQL

PostgreSQL, often simply Postgres, is an object-relational database management system (ORDBMS) with an emphasis on extensibility and standards compliance. It can handle workloads ranging from small single-machine applications to large Internet-facing applications (or for data warehousing) with many concurrent users; on macOS Server, PostgreSQL is the default database. It is also available for Microsoft Windows and Linux (supplied in most distributions).

⁹<https://www.postgresql.org>

¹⁰<https://virtuoso.openlinksw.com>

¹¹<https://solid.mit.edu>

¹²https://en.wikipedia.org/wiki/Tim_Berners-Lee

Virtuoso

Virtuoso Universal Server is a middleware and database engine hybrid that combines the functionality of a traditional Relational database management system (RDBMS), Object-relational database (ORDBMS), virtual database, RDF, XML, free-text, web application server and file server functionality in a single system. Rather than having dedicated servers for each of the aforementioned functionality realms, Virtuoso is a "universal server". It also enables a single multithreaded server process that implements multiple protocols.

SOLID

Solid (derived from "social linked data") is a proposed set of conventions and tools for building decentralized social applications based on Linked Data principles. Solid is modular, extensible and it relies as much as possible on existing W3C standards and protocols. More details on how exactly SOLID is being utilized inside LinkedPipes Applications are described in the Architecture section.

4.4 CI/CD

One of the main requirements for implementation of LinkedPipes Applications was to establish a reliable and efficient continuous integration and delivery. Therefore, the team was aiming to implement a simple yet effective flow that is described in detail in subsequent sections. In addition to that, the general overview of technologies used is provided.

4.4.1 CI/CD development stack

Each of the components of LinkedPipes Applications has its own *Dockerfile* that can be observed inside the specific component folder inside `/src/` folder. In addition to that, the whole application is set to be executed using Docker Compose that is responsible for managing Dockerized *frontend*, *backend*, *storages* as well as *Discovery* and *ETL* which are external services used for exploring and extracting Linked data. Aside from that, this section provides an overview of Travis CI and several continuous integration services that are being used within the project.

Docker

Docker is a computer program that performs operating-system-level virtualization, also known as "containerization". It was first released in 2013 and is developed by Docker, Inc. Docker is used to run software packages called "containers". Containers are isolated from each other and bundle their own application, tools, libraries and configuration files; they can communicate with each other through well-defined channels. All containers are run by a single operating system kernel and are thus more lightweight than virtual

machines. Containers are created from "images" that specify their precise contents. Images are often created by combining and modifying standard images downloaded from public repositories.

Docker Compose

Docker Compose is a tool for defining and running multi-container Docker applications. With Compose, the configuration on which containers to run is specified inside YAML file to configure your application's services. Then, with a single command, you create and start all the services from your configuration.

Travis CI

Travis CI is a hosted, distributed continuous integration service used to build and test software projects hosted at GitHub.

Renovate

Renovate is an automated dependency updater. Multi-platform and multi-language. Due to various dependencies used in the *frontend* web component, it is crucial to keep the project up to date with the latest stable software releases. Renovate is setup and being triggered using Github Webhooks each weekend to check for updates in *package.json* and make a pull request to *master* and *develop* branches if any available.

Codacy

Codacy is an automated code review tool that allows developers to improve code quality and monitor technical debt. Codacy automates code reviews and monitors code quality on every commit and pull request. It reports back the impact of every commit or pull request in new issues concerning code style, best practices, security, and many others. It monitors changes in code coverage, code duplication, and code complexity. It allows developers to save time in code reviews and tackle efficiently technical debt.

5. Project Execution

5.1 Difficulty estimation

The project is to be carried out by 5 team members following the agile methodology of development. The deadline is 9 months from the start of the work on the project. The official start date of the project is 15 November 2018. The project is estimated to be ready for defense in July 2019. The rest of the section provides a general overview of work and task distribution among team members as well as the carried out plan of high-level goals to be accomplished during monthly sprints.

5.1.1 Sprints

The work among team members is organized in weekly sprints. At the end of each sprint, a meeting with project supervisors is carried out where the team discusses the results of the previous week, identifies new problems to solve and asks questions to supervisors. After the meeting event with supervisors, another event is carried out where all team members meet, discuss the results of the meeting with supervisors, define the problems as tasks on a collaborative task management tool called Asana¹ and then the next weekly sprint starts. Communication among team members and supervisors is carried out in a team collaboration tool called Slack², that is as mentioned before, also used as a workspace for aggregating notifications from various CI/CD integrations and bots working within LinkedPipes Applications.

¹<http://asana.com/>

²<https://slack.com>

5.1.2 Project implementation plan

The section represents the table with a carried out plan of project implementation.

Month	Implementation plan
1	<ul style="list-style-type: none">• setting up team processes (VCS, e-mail group, issue tracking, etc.)• getting familiar with Linked Data, RDF• getting familiar with LinkedPipes platform• requirement analysis• determination of data types for which interactive application will be implemented for (map, data cube, ...)
2	<ul style="list-style-type: none">• analysis and design of individual applications• determination of common application parts and components that will be implemented into the framework itself• design of the framework API (will be implemented by individual apps)
3	<ul style="list-style-type: none">• deployment setup via Docker and related technologies.
4	<ul style="list-style-type: none">• framework implementation• applications implementation
5	<ul style="list-style-type: none">• testing of the current implementation• requirements refinement
6	<ul style="list-style-type: none">• revision of the framework implementation based on the refined requirements
7	<ul style="list-style-type: none">• testing of the current implementation• documentation
8	<ul style="list-style-type: none">• debugging
9	<ul style="list-style-type: none">• acceptance testing and finalization of documentation

Table 5.1: Implementation plan for a duration of nine months

Each of the weekly sprints is referring to a monthly goal defined in the table above when new tasks are being created and distributed, in order to have a high-level monthly goal defined for each team member and improving productivity.

List of Figures

2.1	Mock-up of the login page, where users get authenticated	5
2.2	Dashboard of an authenticated user	6
2.3	Mock-up of the step where the data sources are selected	7
2.4	Mock-up of the step where the user selects the desired visualization	7
2.5	Mockup of the configuration of an application	8
2.6	The example of geo-coordinates visualizer implemented with GoogleMaps .	10
2.7	D3.js Chord visualizer example	11
2.8	An example of a treemap, sunburst and tree diagram visualization	11
2.9	An example of a simple line graph, stacked area chart, bar chart, calendar view, and stream graph visualization	12
2.10	Continuous delivery flow on a pull request to the <i>master</i> branch	14
2.11	Continuous delivery flow on a pull request to any branch except <i>master</i> . .	14
2.12	GitHub webhook flow used for continuously pulling latest <i>frontend</i> and <i>backend</i> images from DockerHub	15
3.1	High-level overview of LinkedPipes Applications	17
3.2	Backend component architecture overview	19
3.3	General architecture of Redux store and React components within <i>frontend</i>	22
3.4	Sequence diagram of API interactions between <i>backend</i> , <i>Discovery</i> and <i>ETL</i>	24

List of Tables

5.1	Implementation plan for a duration of nine months	32
-----	---	----

Glossary

Assistant Refers to the Linked Pipes Visualization Assistant, which allows users to create, configure and publish visualizations based on input data sets. 34

Data source Refers to any source of data, such as an RDF file, CSV, database, etc. 34

Data descriptor An SPARQL ASK query associated with a visualizer that determines if an input data graph can be visualized in the corresponding visualizer. 34

Data cube multi-dimensional array of values. 34

Endpoint An endpoint is one end of a communication channel. 34

Linked Open Data Cloud The largest cloud of linked data that is freely available to everyone. 34

Linked Data a method of publishing structured data so that it can be interlinked. 34

LinkedPipes ETL The service in charge of the ETL process in the LinkedPipes ecosystem. 4, 26, 34

LinkedPipes Discovery The service of the LinkedPipes ecosystem in charge of the process of discovering pipelines to be executed in a particular dataset.. 3, 26, 34

Pipeline in the current context refers to the process in which the application takes any data source, applies a series of transformations to it and then hands over the output to a visualizer component, which then produces a visual representation of the data. 3, 34

pipeline discovery The process taking input descriptors for all visualizers and attempt to combine the respective transformation registered to achieve a specific data format. 34

Semantic web an extension of the World Wide Web through standards by the World Wide Web Consortium. 34

SOLID to write. 26, 34

SPARQL Protocol and RDF Query Language query language for retrieving and manipulating data stored in RDF format. 34

Acronyms

API Application Programming Interface. 4, 34

DOM Document Object Model. 15, 34

ETL Extract, Transform and Load. 4, 34

IRI Internationalized Resource Identifier. 6, 34

LDVM Linked Data Visualization Model. 3, 34

LDVMi Linked Data Visualization Model implementation. 34

LOD Cloud Linked Open Data Cloud. 23, 34

LOV Linked Open Vocabularies. 3, 34

LPA LinkedPipes Application. 34

RDF Resource Description Framework. 3, 4, 34

SPA Single-page Application. 34

SPARQL SPARQL Protocol and RDF Query Language. 6, 34

SSL Secure Sockets Layer. 16, 34

TLS Transport Layer Security. 16, 34

TTL Turtle (syntax). 6, 34

URI Uniform Resource Identifier. 34

URL Uniform Resource Locator. 34

W3C World Wide Web Consortium. 3, 34