FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of master's thesis

Title:	User-friendly metadata and dataflow extensions in the MANTA project
Student:	Bc. Yauheniy Buldyk
Supervisor:	Ing. Michal Valenta, Ph.D.
Study program:	Informatics
Branch / specialization:	Web and Software Engineering, specialization Software Engineering
Department:	Department of Software Engineering
Validity:	until the end of summer semester 2021/2022

Instructions

The aim of this work is to design and implement a prototype module for the user-defined extensions of metadata and dataflow in the MANTA project. The module will have a graphical user interface and will be part of the MANTA software.
Follow these steps:
• Analyze the requirements and design wireframes of the module.

• Design the software architecture according to needs and other existing software modules.

• Implement the prototype as a web application.

• Test and document the prototype properly.

Electronically approved by Ing. Michal Valenta, Ph.D. on 16 January 2021 in Prague.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Master's thesis

User-friendly metadata and dataflow extensions in the MANTA project

Bc. Yauheniy Buldyk

Department of Software Engineering
Supervisor: Ing. Michal Valenta, Ph.D.

May 6, 2021

Acknowledgements

I would like to thank the supervisor of my work, Ing. Michal Valenta, Ph.D., for the possibility to process the chosen topic. I would also like to thank the members of the MANTA team who helped me design and implement the practical part of this work. Last but not least, I want to thank my girlfriend for her incredible support throughout the whole writing process.

Declaration

I hereby declare that I have authored this thesis independently, and that all sources used are declared in accordance with the "Metodický pokyn o etické přípravě vysokoškolských závěrečných prací".

I acknowledge that my thesis (work) is subject to the rights and obligations arising from Act No. 121/2000 Coll., on Copyright and Rights Related to Copyright and on Amendments to Certain Laws (the Copyright Act), as amended, (hereinafter as the "Copyright Act"), in particular § 35, and § 60 of the Copyright Act governing the school work.

With respect to the computer programs that are part of my thesis (work) and with respect to all documentation related to the computer programs ("software"), in accordance with Article 2373 of the Act No. 89/2012 Coll., the Civil Code, I hereby grant a nonexclusive and irrevocable authorisation (license) to use this software, to any and all persons that wish to use the software. Such persons are entitled to use the software in any way without any limitations (including use for-profit purposes). This license is not limited in terms of time, location and quantity, is granted free of charge, and also covers the right to alter or modify the software, combine it with another work, and/or include the software in a collective work.

In Prague on May 6, 2021 . . . .

Citation of this thesis

Buldyk, Yauheniy. *User-friendly metadata and dataflow extensions in the MANTA project.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2021.

Abstrakt

Tato práce si klade za cíl navrhnout a implementovat funkční prototyp webové aplikace pro software Manta Flow. Aplikace umožní rozšíření existujícího dataflow grafu o nová data, jako jsou assety a datové toky. Aplikace je navržena jako vylepšení stávajícího řešení, které spočívá v manuálním vytváření specifických souborů CSV.

Obsahem práce je sběr požadavků, analýza, návrh a implementace prototypu webové aplikace. Vytvořený prototyp bude zahrnut do existujícího softwaru MANTA Flow jako další modul v aplikaci Admin GUI.

Klíčová slova webová aplikace, uživatelské rozhraní, datové toky, MANTA Flow, Java, JavaScript, React

Abstract

This work aims to design and implement a functional prototype of a web application for Manta Flow software. The application will allow extending the existing data lineage with new data such as assets and flows. The application is intended to upgrade an existing solution, which consists of manually creating specific CSV files.

The basis of the thesis is the collection of requirements, analysis, design, and implementation of a prototype web application. The created prototype will be included in the existing MANTA Flow software as another module in the Admin GUI application.

Keywords web application, user interface, data flows, MANTA Flow, Java, JavaScript, React

Contents

Introduction **1**
Goals of the Thesis . . . 1
Structure of the Thesis . . . 2

1 Basic concepts **3**
1.1 Web application . . . 3
1.2 Manta and Manta Flow . . . 3
1.3 Data Lineage . . . 3
1.4 MANTA Admin GUI . . . 4

2 Technologies used **7**
2.1 Backend specific technologies . . . 7
2.1.1 Java . . . 7
2.1.2 Spring . . . 7
2.1.3 JavaDoc . . . 8
2.1.4 Apache Subversion . . . 8
2.2 Frontend specific technologies . . . 8
2.2.1 JavaScript . . . 8
2.2.2 React . . . 8
2.2.3 Redux . . . 8
2.2.4 Redux-Saga and axios . . . 8
2.2.5 JSDoc . . . 9
2.2.6 Git . . . 9
2.3 Other technologies . . . 9
2.3.1 IntelliJ IDEA . . . 9
2.3.2 Pencil . . . 9
2.3.3 draw.io . . . 9

3 Requirements **11**
3.1 Introduction to the issue . . . 11

3.2 Existing solution - Custom Metadata 11
3.3 Open MANTA Extensions requirements 13
3.3.1 Scope 13
3.3.2 Functional requirements 14
3.3.3 Technical requirements 15

4 Analysis and Design 17
4.1 Use cases 17
4.2 Wireframes 19
4.2.1 Main page 19
4.2.2 Preview page 22
4.3 High-Level architecture 24
4.3.1 Component diagram 24
4.3.2 Sequence diagrams 25

5 Implementation 29
5.1 Phase 1 - Repository tree 29
5.1.1 Backend 29
5.1.2 Frontend 30
5.2 Phase 2 - Create, edit, and remove operations 32
5.2.1 Backend 33
5.2.2 Frontend 33
5.3 Next Phases 36

Conclusion 39

Bibliography 41

A Acronyms 43

B Open MANTA Extensions use case diagrams 45

C Contents of enclosed CD 51

List of Figures

1.1 Example of dataflow visualization for Excel in the MANTA Flow [4] 4
1.2 Log management in the MANTA Admin GUI [5] 5

3.1 Repository Page of the MANTA Flow Viewer 12

4.1 Main Page of the Open MANTA Extensions UI 21
4.2 Preview Page of the Open MANTA Extensions UI 23
4.3 Open MANTA Extensions component diagram 24
4.4 Repository tree processes sequence diagram 25
4.5 Create/Edit/Remove processes sequence diagram 27

5.1 Open MANTA Extensions resources retrieving sequence diagram . 34
5.2 Open MANTA Extensions resources saving sequence diagram . . . 35
5.3 Open MANTA Extensions prototype demonstration 37

B.1 Open MANTA Extensions use case diagram for the repository . . 46
B.2 Open MANTA Extensions use case diagram for layers and resources 47
B.3 Open MANTA Extensions use case diagram for assets 48
B.4 Open MANTA Extensions use case diagram for saving changes . . 49

List of Tables

4.1 Open MANTA Extensions use cases 17

Introduction

The amount of data on our planet is growing exponentially every year, as is their importance. The larger the company, the more data it collects and the more valuable the data are, but analyzing such large volumes of data can be challenging. MANTA Flow software is designed to solve these problems. It can analyze either SQL scripts or a BI tool report and automatically build a dataflow graph based on it. With such a graph, it is simple to trace the origin of data and view its lifecycle stages, saving analysts up to several workdays.

However, there are many different tools, and it is not possible to support them all; some customers even use their private internal solutions. Therefore, MANTA Flow also supports the ability to add data of various formats to the lineage manually. The main disadvantage of the current solution is the absence of a suitable tool; all data must be entered manually into a CSV file in the exact format. The complexity and hostility of such an approach can discourage potential customers, which is the main reason for this work. The result is a functional prototype of a web application that allows creating custom assets and dataflows through the user interface.

Goals of the Thesis

This work aims to design and implement a prototype module for the user-defined extensions of metadata and dataflow in the MANTA project. The module will have a graphical user interface and will be part of the MANTA software.

The theoretical part intends to collect requirements, perform analysis, and design a prototype. The practical part then focuses on implementing a functional prototype as a web application with subsequent testing and documentation of the code.

Structure of the Thesis

This work consist of the following five chapters:

- **Basic concepts** The first chapter explains the main concepts necessary for a general understanding of the work.
- **Used technologies** The second chapter provides an overview of the most critical technologies and tools used during the prototype implementation.
- **Requirements** Chapter 3 is about the prototype requirements, which describes the main functionalities.
- **Analysis and Design** This chapter describes the performed request analysis, including the use cases. We will also take a look at the application design creation, wireframes, and high-level architecture models.
- **Implementation** In the last chapter, we will focus on the implementation itself. We will describe the problems solved in the frontend and backend parts and explain the main processes in more detail with examples.

Chapter 1

Basic concepts

1.1 Web application

A web application (or web app) is application software that runs on a web server, unlike computer-based software programs that are run locally on the operating system (OS) of the device. Web applications are accessed by the user through a web browser with an active network connection. These applications are programmed using a client–server modeled structure—the user (”client”) is provided services through an off-site server that is hosted by a third-party. [1]

1.2 Manta and Manta Flow

MANTA, or officially Manta Tools, s.r.o., is a Czech startup company, which was initially a project of Profinit, s.r.o. and developed in cooperation with FIT CTU. The company's main product is MANTA Flow software, which is used today by large organizations in the Czech Republic and worldwide.

MANTA Flow is a tool enabling automatic analysis of databases and data warehouses SQL scripts (e.g., Oracle, PostgreSQL, Teradata), BI tools (e.g., IBM Cognos, OBIEE, Microsoft SSRS), or programming languages (e.g., Java, C#).

The software can build a clear map of dataflows across the BI environment, the so-called Data Lineage, based on the performed analysis. In practice, this is mainly used to optimize data warehouses, reduce software development costs and perform impact analyzes. [2]

1.3 Data Lineage

Data lineage includes the data origin, what happens to it and where it moves over time. Data lineage gives visibility while greatly simplifying the ability to trace errors back to the root cause in a data analytics process. [3]

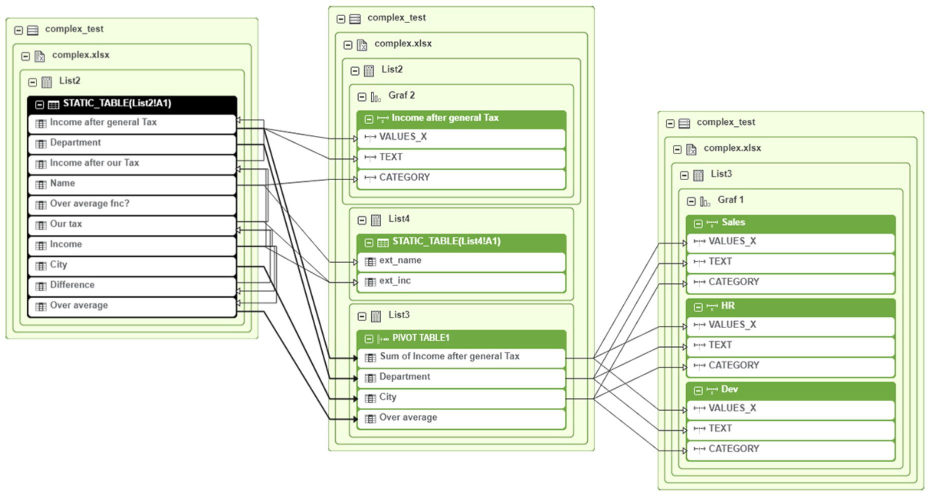


Figure 1.1: Example of dataflow visualization for Excel in the MANTA Flow [4]

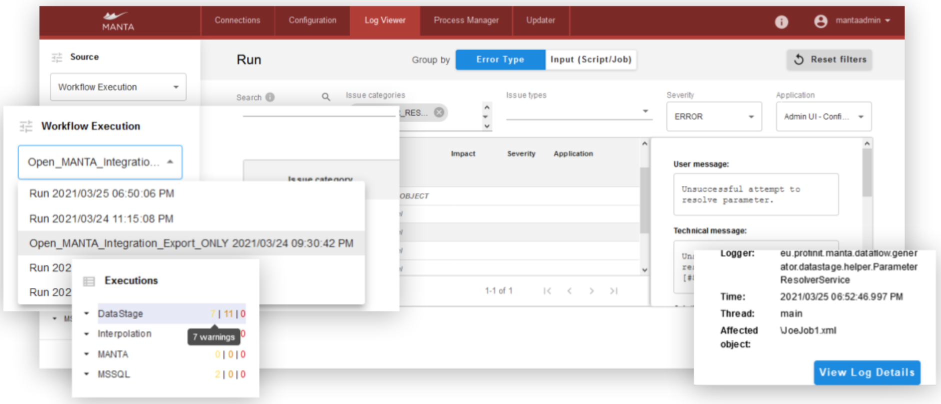
The Manta Flow software visualizes data in the form of an oriented graph, where nodes represent objects or data structures and edges are dataflows between them (the direction is always from the data source to the resulting object). The following figure (1.1) illustrates one of the possible resulting visualizations.

1.4 MANTA Admin GUI

MANTA Admin GUI is MANTA Flow's utility web application that acts as a graphical and programming interface for the installation, configuration, update, and overall maintenance of MANTA Flow.

The application consists of several modules/tools.

- **MANTA Installer** installs MANTA applications (MANTA Flow, MANTA Admin GUI)
- **MANTA Updater** updates MANTA Flow Server and MANTA Flow CLI instances to newer versions
- **MANTA Configurator** configures the general behavior of MANTA services
- **MANTA Process Manager** allows to define and execute MANTA Workflows (sequence of processes such as metadata extraction and analysis)



Run 2021/03/25 06:50:06 PM
Run 2021/03/24 11:15:08 PM
Open_MANTA_Integration_Export_ONLY 2021/03/24 09:30:42 PM

User message:
Unsuccessful attempt to resolve parameter.

Logger: eu.profinit.manta.dataflow.generator.datastage.helper.ParameterResolverService
Time: 2021/03/25 06:52:46.997 PM
Thread: main
Affected object: \JoeJob1.xml

Figure 1.2: Log management in the MANTA Admin GUI [5]

- **MANTA Log Viewer** manages logs from the MANTA Flow Server and a MANTA Flow CLI

A new module for metadata and dataflow extensions will be a part of the Admin GUI.

Chapter 2

Technologies used

The chapter describes the primary technologies and tools used during the thesis processing. Technologies are separated into three categories: backend, frontend, and common.

2.1 Backend specific technologies

This section describes the tools used to create the backend part of the application. The backend processes, stores, and uploads data; connects the frontend to the filesystem and other parts of the MANTA software.

2.1.1 Java

Java is an object-oriented programming language and is currently one of the most widely used in the world. Java is used to develop many applications in various segments (both desktop and web), and MANTA is no exception. The whole backend part is written in this language, and we assume that the reader has a basic knowledge of Java.

2.1.2 Spring

Spring's flexible and comprehensive set of extensions and third-party libraries let developers build almost any application imaginable. At its core, Spring Framework's Inversion of Control (IoC) and Dependency Injection (DI) features provide the foundation for a wide-ranging set of features and functionality. [6]

Spring allows us to simplify many things during implementation. We will use Spring Beans to create singleton objects, which we will use in various places in the code by Dependency Injection. We will also use the Spring MVC a lot for communication between the individual components of the application.

2.1.3 JavaDoc

JavaDoc tool is a document generator tool in Java programming language for generating standard documentation in HTML format. It generates API documentation. It parses the declarations ad documentation in a set of source file describing classes, methods, constructors, and fields. [7]

2.1.4 Apache Subversion

Subversion (SVN) is an open-source version control system that maintains a history of source code changes and simplifies their control. We will use SVN during the implementation of the backend part of the prototype.

2.2 Frontend specific technologies

In this section, we will look at the technologies used in the frontend part, represented as the web application UI.

2.2.1 JavaScript

JavaScript is an interpreted object-oriented language best known as the scripting language for web pages, making them interactive. In our case, we will use JavaScript and its frameworks to create the frontend part of the application.

2.2.2 React

React is a popular JavaScript library for creating user interfaces. It allows developers to build large web applications with an object-oriented approach. React splits the application into so-called components and can dynamically re-render them without reloading the entire page.

2.2.3 Redux

Redux is a predictable state container for JavaScript apps. [8] It maintains an entire application state in a single immutable tree object, which cannot be changed directly. Redux helps the application behave consistently, makes it easier to detect and fix errors, and reduces code size.

2.2.4 Redux-Saga and axios

Redux-Saga is a middleware library that allows the Redux store to communicate with other services asynchronously. We will use Redux-Saga along with the *axios* library, providing a promise-based HTTP client for JavaScript.

2.2.5 JSDoc

JSDoc is an API documentation generator for JavaScript code, similar to the JavaDoc (see 2.1.3).

2.2.6 Git

Git is one of the best and most popular version control systems, similar to Subversion (see 2.1.4). [1]

2.3 Other technologies

2.3.1 IntelliJ IDEA

IntelliJ IDEA from JetBrains is one of the most used IDEs for developers. It supports both Java and JavaScript languages, simplifies code writing and formatting, and allows convenient code debugging and testing. IntelliJ also increases productivity and code quality; it is used for both backend and frontend parts.

2.3.2 Pencil

Pencil is a convenient open-source tool for creating application prototypes, wireframes, or mockups. We will use it when designing the appearance of the module.

2.3.3 draw.io

draw.io is a tool for creating diagrams and graphs. It has a large selection of shapes and elements, which allows us to build all the necessary diagrams and models during the analysis phase.

[1] Usually, only one version control system is used in one project, but MANTA is currently switching from SVN to Git, so we use both.

Chapter 3

Requirements

The first phase of prototyping was the collection of requirements. Based on the analysis of the existing solution and the user's needs, the scope of changes to the user interface was determined.

3.1 Introduction to the issue

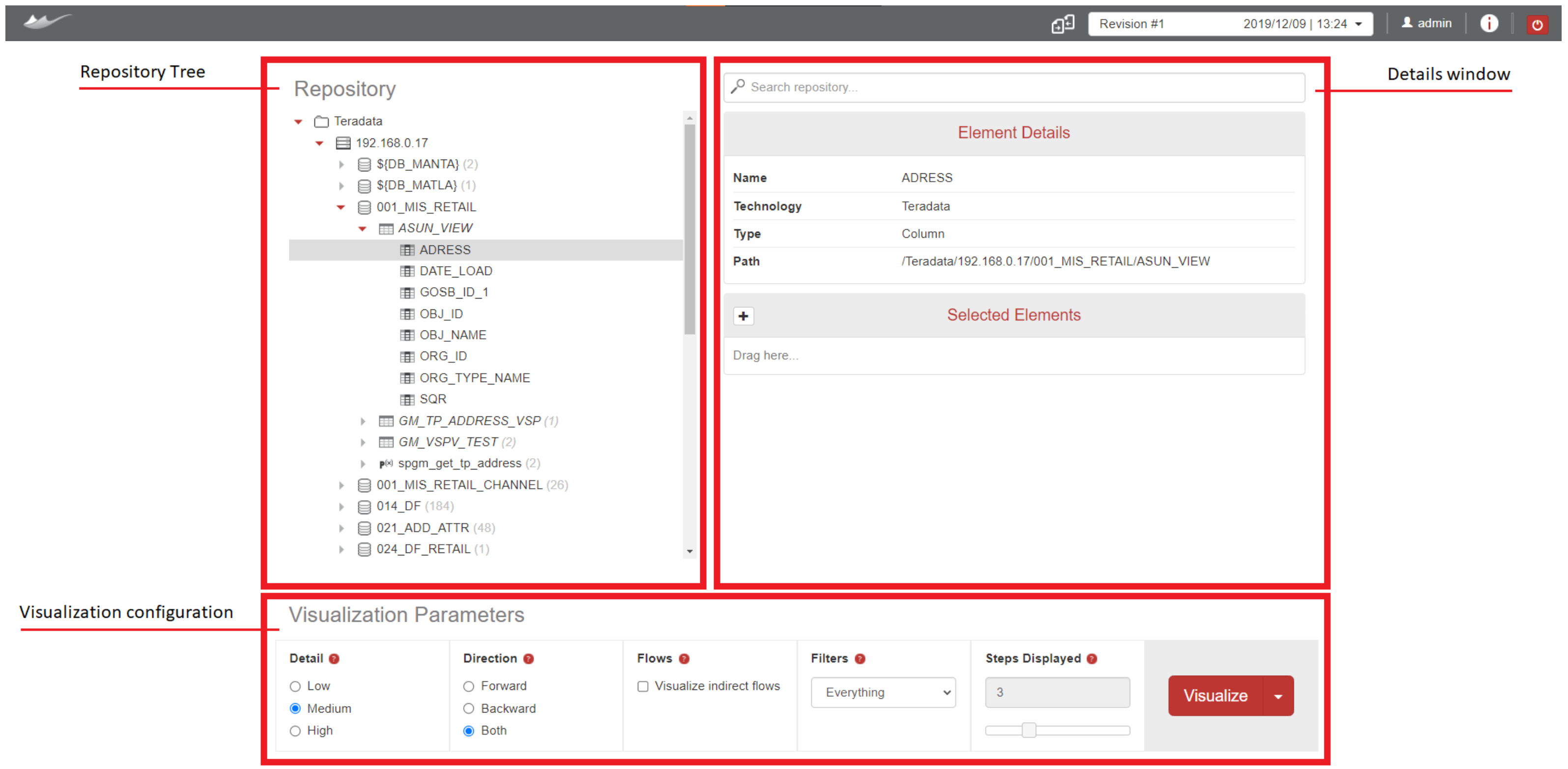
One of the MANTA software modules is the MANTA Flow Viewer, which is used to visualize dataflows and so-called assets (presented as nodes in the graph). In addition to the visualizer itself, the Viewer also has a page for selecting nodes and configuring the visualization - Repository Page (see 3.1).

On the left is the so-called Repository Tree, where the list of all nodes is presented in a hierarchical form (for example, when analyzing the dataflow in a database, such a hierarchy from parent to leaf may look like this: database $\rightarrow$ schema $\rightarrow$ table $\rightarrow$ column). To the right is the Details window, which contains the asset's basic parameters, and a list of its attributes, if any. Below are visualization configuring tools that are not relevant to this work.

Typically, all assets are created automatically without user involvement using special generators that analyze databases or BI tools and build dataflows graph. However, MANTA only supports a part of many existing technologies, so there are situations where the customer needs to add an unsupported or even a custom tool to the graph. In these cases, MANTA allows adding assets manually using so-called Custom Metadata.

3.2 Existing solution - Custom Metadata

The Custom Metadata is based on the use of the CSV file. The user defines assets, their attributes, and other items in the appropriate files, loaded into the server, and imported as new custom nodes and edges into the graph.



Repository

- Teradata
  - 192.168.0.17
    - ${DB_MANTA} (2)
    - ${DB_MATLA} (1)
    - 001_MIS_RETAIL
      - *ASUN_VIEW*
        - ADRESS
        - DATE_LOAD
        - GOSB_ID_1
        - OBJ_ID
        - OBJ_NAME
        - ORG_ID
        - ORG_TYPE_NAME
        - SQR
      - *GM_TP_ADDRESS_VSP* (1)
      - *GM_VSPV_TEST* (2)
      - spgm_get_tp_address (2)
    - 001_MIS_RETAIL_CHANNEL (26)
    - 014_DF (184)
    - 021_ADD_ATTR (48)
    - 024_DF_RETAIL (1)

Search repository...

Element Details

Name	ADRESS
Technology	Teradata
Type	Column
Path	/Teradata/192.168.0.17/001_MIS_RETAIL/ASUN_VIEW

Selected Elements

Drag here...

Visualization Parameters

Detail	**Direction**	**Flows**	**Filters**	**Steps Displayed**
Low	Forward	☐ Visualize indirect flows	Everything	3
[✓ Medium]	Backward			
High	[✓ Both]			

Figure 3.1: Repository Page of the MANTA Flow Viewer

There are a total of six separate files.

- `layer.csv` contains information about the metadata layers. Each element belongs to a specific layer (e.g., tables and files belong to the physical layer, business assets to the business layer). There could be several layers at the same time; they are described by ID, name, and type
- `resource.csv` contains information about resources. Resources are specific nodes representing individual technologies. They are always the roots of subtrees and serve as containers for the assets of the technology. Resource parameters are ID, name, type, description, layer ID
- `node.csv` describes the nodes in the graph, i.e., assets. Node information includes its ID, ID of the parent node, name, type, and resource ID
- `edge.csv` contains edges for the graph. Edges represent dataflows between assets. A typical example is the ETL transformation between database tables (in this case, the data lineage can look like this: source table $\rightarrow$ ETL transformation $\rightarrow$ target table). Edges are described by edge ID, ID of the source node, ID of the target node, type, and ID of the nodes resource
- `node_attribute.csv` contains further attributes of nodes, described by node ID, name, and value
- `edge_attribute.csv` contains further attributes of edges, described by edge ID, name, and value

As we can see, CSV files work similarly to tables in relational databases - we have a separate file for each object, and the relations are set using references via ids. Such an approach is not user-friendly and increases the possibility of making mistakes when manually filling these files. In addition, there are other restrictions in some files that are not clear without reading the documentation. Therefore, it is necessary to extend the existing solution with a convenient tool - Open MANTA Extensions UI.

3.3 Open MANTA Extensions requirements

3.3.1 Scope

The goal is to provide the UI as part of the Admin GUI with the ability to

- create entirely new custom assets, resources, and layers
- create new assets within existing ones (e.g., add a new table to already scanned database)

- update already existing assets by adding attributes
- create new flows between existing assets
- save changes to the CSV files, so they can be used later to import data to the server.

3.3.2 Functional requirements

Based on the existing solution, the following application requirements were determined. [2]

- Ability to create new assets within existing ones. The user will be able to
  - view all existing assets from the server
  - define new Layers through the UI
  - define new Resources through the UI
  - define new assets with attributes
  - view all newly created assets during the edit session
  - remove new (not yet saved) assets during the edit session
- Ability to update already existing assets. The user will be able to
  - view all existing assets from the server
  - change every parameter or attribute on an existing asset
  - add new parameters or attributes to an existing asset
  - view all changed/added parameters during the edit session
  - remove new (not yet saved) parameters during the edit session
- Ability to create new flows between existing assets (including newly created custom ones). The user will be able to
  - define new flows of different types with attributes between assets.
  - create new links using ”drag-and-drop”
  - connect higher levels (e.g., after adding a link between two schemas/tables, all columns will be connected automatically)
  - view all newly added links during the edit session

[2] Note that the requirements provided relate to the final version of the application only. This work aims to create a functional prototype, which will meet only part of the requirements due to the enormous scope and complexity of used tools and the implementation itself.

  - remove new (not yet published) links during the edit session

- Ability to save made changes
  - All changes can be saved under some name (the name will be used for a container folder for CSV files)
  - There can be several saved changes, and the user will be able to switch between them. When multiple changes are selected, all assets will have a property that determines which changes they come from
  - Different saves will be stored in the corresponding folders under the ”import” directory (MANTA Flow uses it by default to store Custom Metadata CSVs)

3.3.3 Technical requirements

- New Open MANTA Extension UI will be a part of the Admin GUI and will use appropriate styles
- The UI will communicate with the server (e.g., for receiving information about assets)
- The UI will use existing Custom Metadata CSV files for saving changes to the filesystem
- Only users with administrator rights will have access to the UI

Chapter 4

Analysis and Design

This chapter describes the phases preceding the implementation. We will create a list of use cases defining the possibilities of future application during analysis. As part of the design, we will build several diagrams and models illustrating the architecture and behavior of the module.

4.1 Use cases

The next phase of the work is the creation of use cases based on written requirements. The results describe the user's needs and how they will be implemented in the application. For greater clarity, we will collect the use cases in a table 4.1. We have also created use case diagrams; they can be found in appendix B.

Table 4.1: Open MANTA Extensions use cases

Begin of Table

Use Case	**Description**	**Scenario**
View existing repository	The user is able to view the whole repository with the last revision	There is the Repository tree on the main page that loads nodes from the server. As the tree expands, the assets gradually load (so-called lazy loading). The application reads unsaved changes from the database and saved changes from CSV files.
Create new layer	The user is able to create a new layer in the Repository tree	There is a button in the layer selection menu to create a new layer
Edit layer	The user is able to edit the name of the created layer	New layers names can be changed ”on-the-fly” in the selection menu

Continuation of Table 4.1

Use Case	Description	Scenario
Remove layer	The user is able to remove the created layer	Next to each added layer in the selection menu is a button to remove this layer.
Create new resource	The user is able to create a new resource in the Repository tree	There is a button in the Repository tree to create a new resource.
Edit resource	The user is able to edit the parameters of the created resource	New resource names can be changed ”on-the-fly” in the tree. Additional parameters can be edited in the Details window.
Remove resource	The user is able to remove the created resource	Next to each added resource in the Repository tree is a button to remove this resource.
Create new asset	The user is able to create a new asset in the repository tree	Next to each node in the Repository tree is a button to create a new child asset.
Edit asset	The user is able to edit the parameters of the created asset	New asset names can be changed ”on-the-fly” in the tree. Additional parameters can be edited in the Details window.
Remove asset	The user is able to remove the created asset	Next to each added asset in the Repository tree is a button to remove this asset.
Add new attribute	The user is able to add a new attribute to an asset	There is a button in the Details window to add a new attribute.
Edit attribute	The user is able to edit the name and the value of the added attribute	New attribute names and values can be changed ”on-the-fly” in the Details window
Remove attribute	The user is able to remove the added attribute	Next to each added attribute in the Details window is a button to remove this attribute.
Add new flow	The user is able to add a new flow between assets	New flows can be added with drag-and-drop in the Repository tree or with a button in the Details window.
Edit flow	The user is able to edit the parameters of the added flow	Source, target, and flow type can be changed in the Details window

Continuation of Table 4.1

Use Case	Description	Scenario
Remove flow	The user is able to remove the added flow	Next to each added flow in the Details window is a button to remove this flow.
Save changes	The user is able to save the made changes	There is a button on the main page to save changes.
Preview changes	The user is able to view all the changes on a single page	There is a preview page where the user can view all made changes, go back and make additional changes or publish them.

As we can see, the use cases are pretty simple and straightforward, which was also our primary intention - to replace the complicated filling of several CSV files with a user-friendly clicking on buttons on a single page. Such a UI can significantly reduce errors and typos in CSV files, speed up creating custom assets and flows, and make the user experience more enjoyable overall.

4.2 Wireframes

After gathering the requirement and determining the use cases, we already have enough data for the first phase of the design - wireframes.

Not dissimilar to an architectural blueprint, a wireframe is a two-dimensional skeletal outline of a webpage or app. Wireframes provide a clear overview of the page structure, layout, information architecture, user flow, functionality, and intended behaviors. As a wireframe usually represents the initial product concept, styling, color, and graphics are kept to a minimum. Wireframes can be drawn by hand or created digitally, depending on how much detail is required. [9]

Thanks to the wireframe model, we can accurately show the appearance of the application and its functionality. Wireframes allow solving some business and UX design problems even before starting the implementation. Furthermore, they are also used as an example of what the final application might look like.

4.2.1 Main page

The main advantage of wireframes is that their creation is relatively fast and cheap; actually, it is enough to use paper with a pencil. However, we will use the Pencil tool, whose specific goal is to create wireframes and mockups. It has a set of pre-prepared forms, supports drag-and-drop operations, and can generate an interactive web page or PDF file from a wireframe.

This section describes the main page of the future application and details its parts and their behavior. [3]

Figure 4.1 shows the wireframe of the main screen of the Open MANTA Extensions UI. Like the Repository Page Viewer (see 3.1), it consists of two parts: the Repository Tree and the Details window.

On the left part, there are Repository trees (we need two of them to enable drag-and-drop edge creation), which display the current list of nodes from the server and the flows between them. Up here, we see the layer selection menu, where the user can choose the layer to work with. All created resources and nodes will be automatically added to the selected layer, so the user does not have to deal with any ID or even know it exists. The user can add a new layer or remove a layer already added directly in this menu.

In the tree itself, we then have a button to create a resource; after clicking, it adds the default resource, and the user will be able to change its parameters in the Details window. Adding new nodes works similarly: after selecting a node (or resource), a button will appear below. The button adds a new node, and then the user can edit it in the Details window. Both resources and nodes can be later removed using the button that appears next to the node when it is selected. Therefore, users get a simple and intuitive way to add a new resource or node. They do not even have to solve the difference between them - the application does everything automatically and saves the changes to the corresponding files.

To the right is the Details window, where the user can view the node parameters, such as name and type, the list of node attributes, and the list of incoming and outgoing flows of the node. The user can add attributes to each node and, if the node is newly created, can also modify its parameters. Below is the Flows section where incoming and outgoing flows are defined. Here the user can add new flows using the button, or it is also possible to create new flows directly between the Repository trees using the ”drag-and-drop” feature.

[3]Note that only a part of the wireframes is described in this chapter, the whole set of wireframes can be found in the enclosed CD.

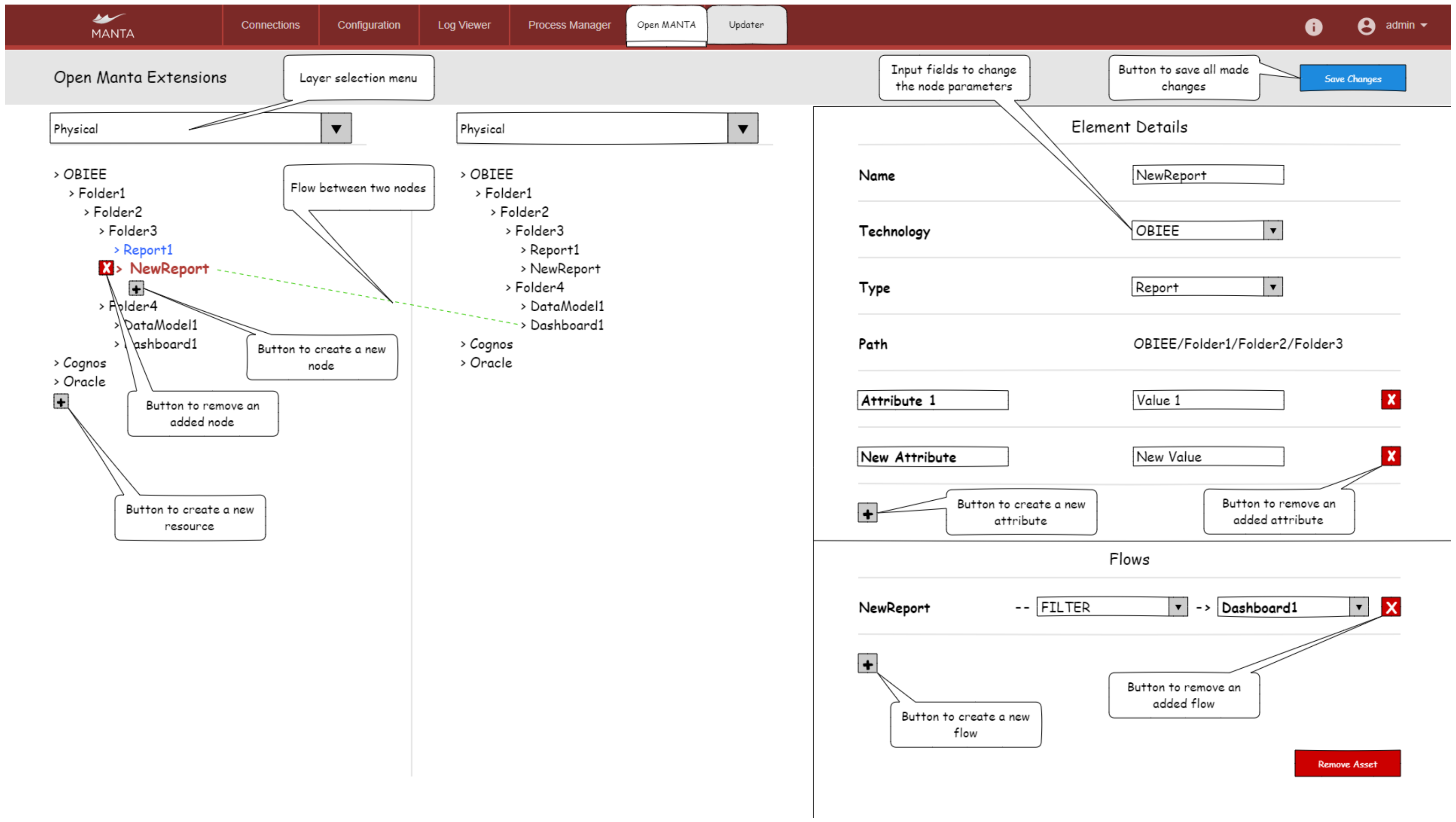


Figure 4.1: Main Page of the Open MANTA Extensions UI

4.2.2 Preview page

Another page we will discuss here is the Preview page. The user is redirected to this page after clicking on the ”Save Changes” button. Here we can view all the made changes split into five groups:

- **New Layers** - all the newly created layers
- **New Resources** - all the newly created resources
- **New Assets** - only the newly created assets
- **Changed assets** - assets with some changes (attributes added)
- **New Flows** - all the newly created flows

Next to each item is an edit button that redirects the user back to the Repository page to make further changes if needed. All described items are shown in Figure 4.2.

Overall, thanks to the Open MANTA Extensions UI, the user will be able to intuitively create new layers, resources, assets and their attributes, and flows. It will be possible to summarize all the changes and save everything with a single button. This way, the user no longer has to deal with unique IDs and relations between CSV files and does not have to study the documentation. Saving to files is done automatically, and additional input validations significantly reduce the chance of making errors.

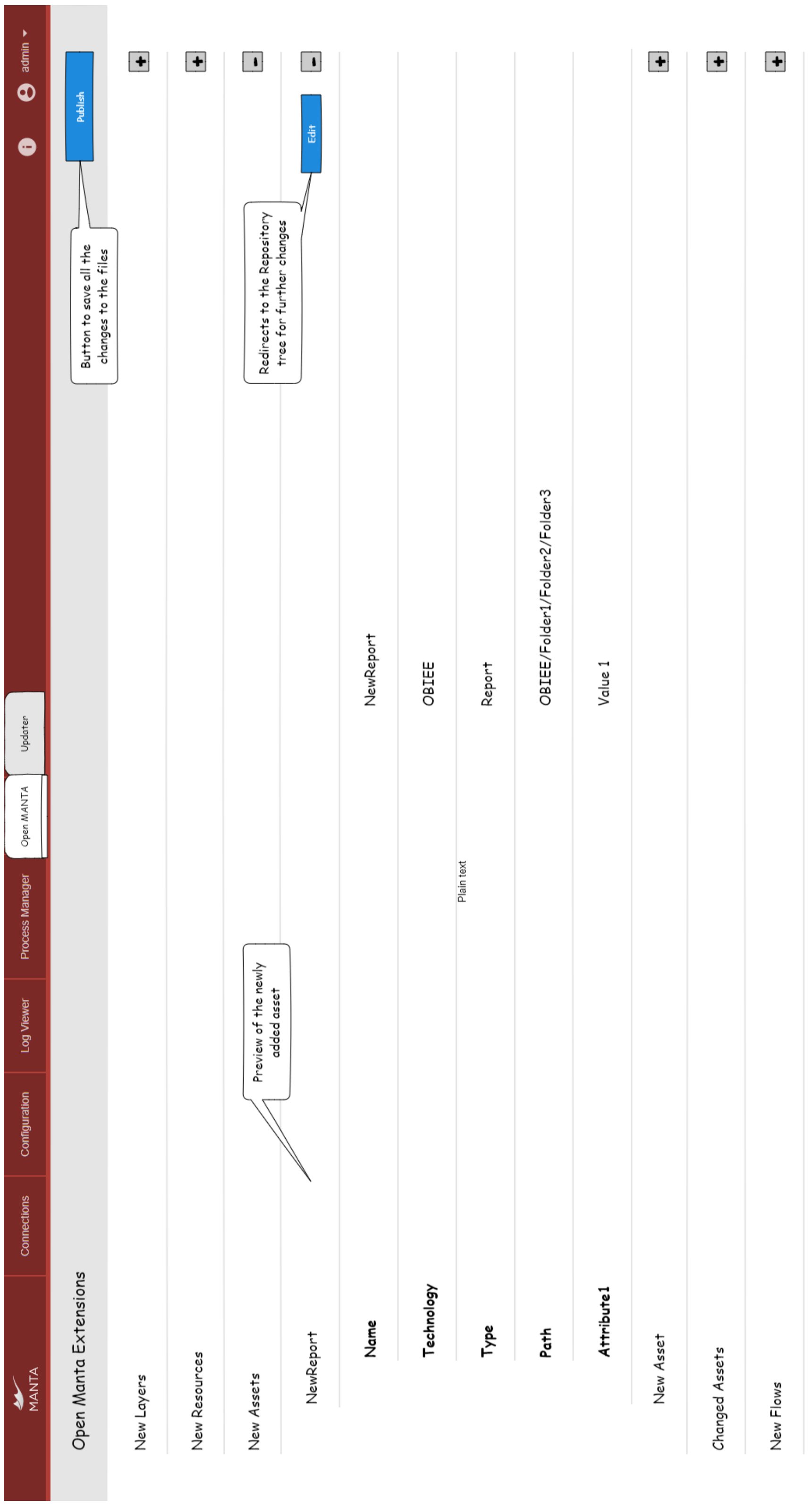


Figure 4.2: Preview Page of the Open MANTA Extensions UI

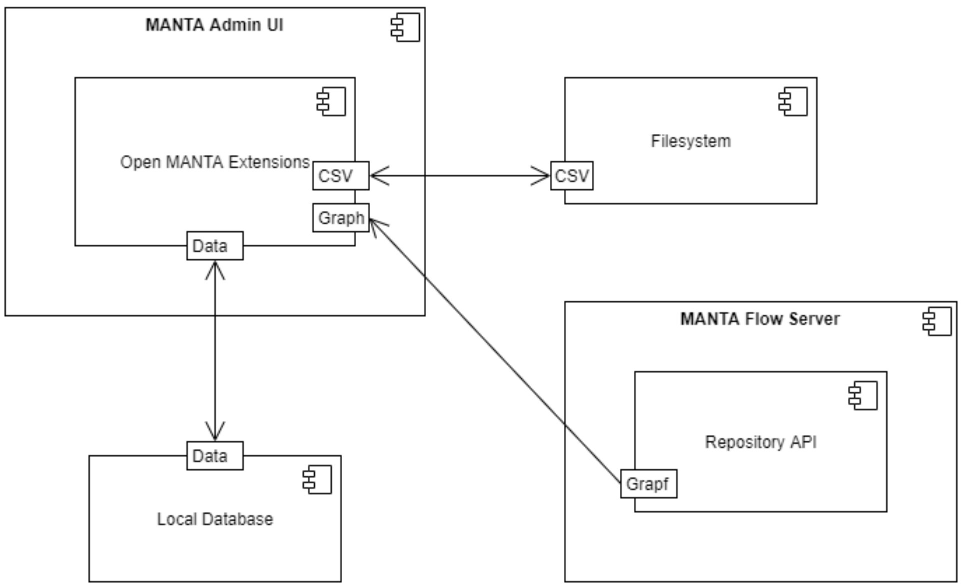


Figure 4.3: Open MANTA Extensions component diagram

4.3 High-Level architecture

The last part of the design consists of creating diagrams describing the model and behavior of the application. Such diagrams help to understand the whole structure of the program and its parts; they also will be used as a hint for the implementation itself.

4.3.1 Component diagram

The first step is to create a component diagram that describes the parts of the application and how they communicate with each other.

As we can see in Figure 4.3, the Open MANTA Extensions will be part of the Admin GUI and communicate with three other components. First - the MANTA Flow server - is needed to retrieve the repository information. For example, we can obtain information about resources and nodes through the Repository API to fill up our Repository trees. The second part is the filesystem, where we will store the changes as CSV files, which will be uploaded to the server later. The last component is the internal database for storing temporary changes and some other information about the application state.

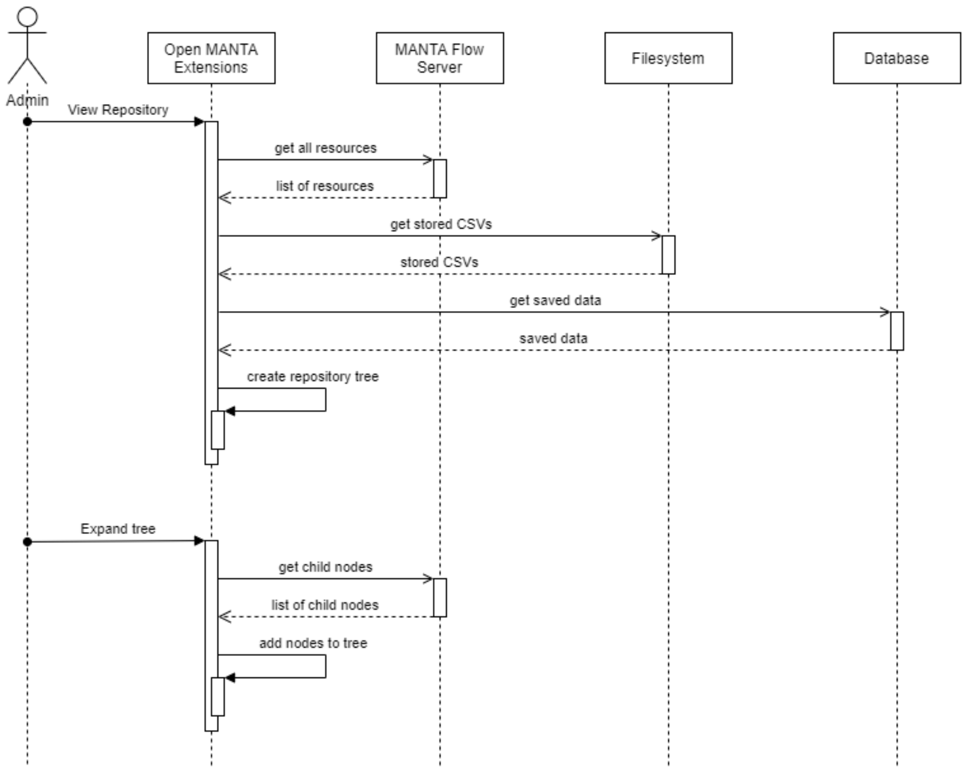


Figure 4.4: Repository tree processes sequence diagram

4.3.2 Sequence diagrams

The second step is to create a sequence diagram describing the application behavior and its communication with other components.

Figure 4.4 shows the processes of the Repository tree and what happens behind the user actions.

When the user opens the main page, the application starts retrieving data from the other components. It gets information about resources from the server through the Repository API; then it receives already saved changes from the CSV files on the filesystem; finally, the app loads temporary changes and other data from the database. Based on this data, it can build the basis of the Repository tree. The rest of the tree is loaded dynamically as the user dives deeper (server storage can be enormous and contain thousands of nodes, so we cannot afford to load the entire tree at once). The application sends a request to the server to obtain child nodes of the selected one and adds the received nodes to the tree.

The second diagram (see figure 4.5) describes creating, editing, and deleting an element. All actions look similar and continuously update the database state. Finally, pressing the ”Save Changes” button opens a preview page and saves all data to the CSV files on the filesystem after approval.

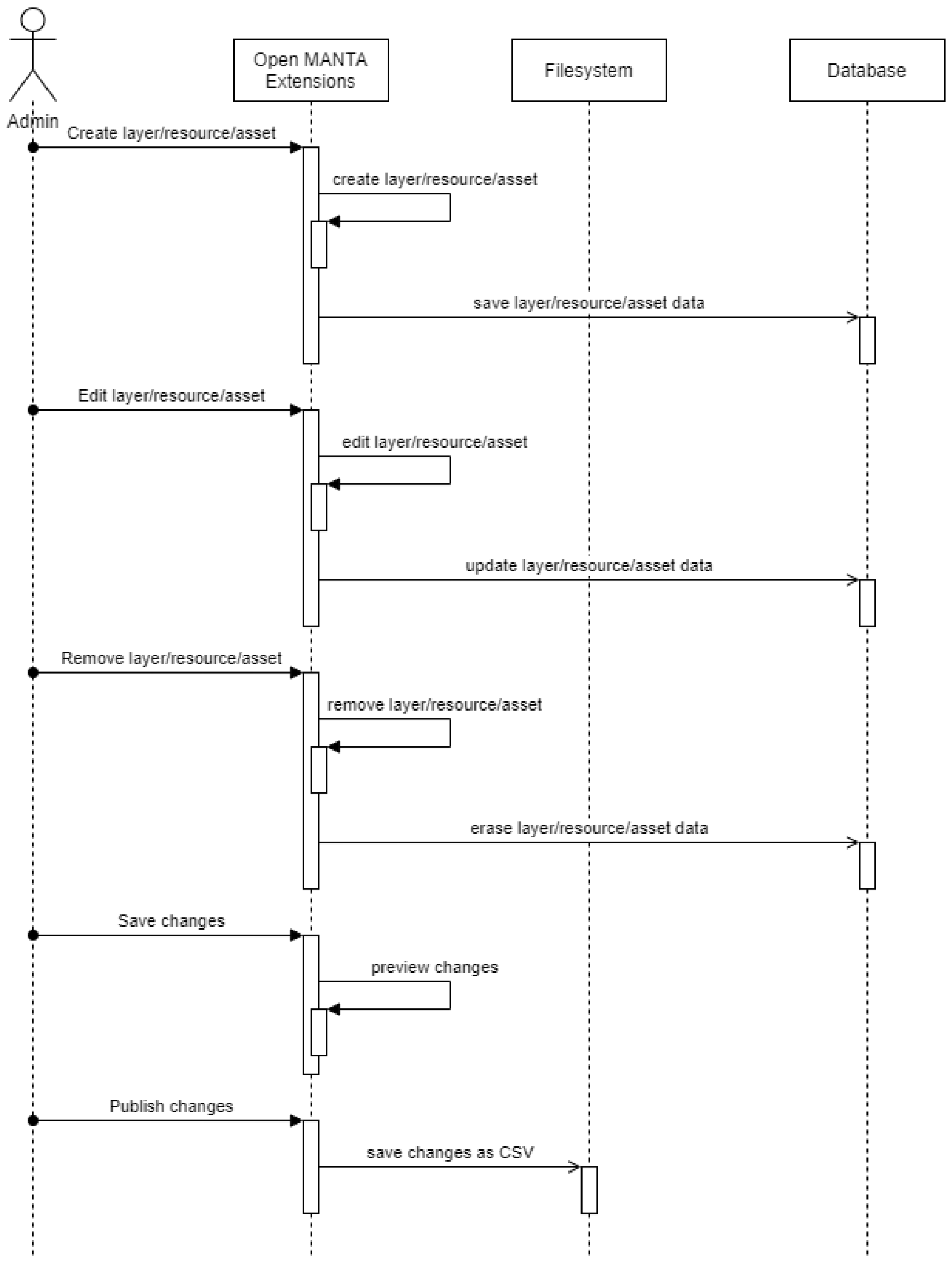

Admin
Open MANTA Extensions
Filesystem
Database
Create layer/resource/asset
create layer/resource/asset
save layer/resource/asset data
Edit layer/resource/asset
edit layer/resource/asset
update layer/resource/asset data
Remove layer/resource/asset
remove layer/resource/asset
erase layer/resource/asset data
Save changes
preview changes
Publish changes
save changes as CSV

Figure 4.5: Create/Edit/Remove processes sequence diagram

Chapter 5

Implementation

After completing all analyzes and design preparations, we can start the implementation itself. In this chapter, we will look at the structure, the main logic, and the individual classes of the module. The whole development process was divided into 5 phases, and we will describe each of them (both backend and frontend).

5.1 Phase 1 - Repository tree

The first phase covers building the application skeleton and creation the Repository tree logic, such as retrieving resources from the server and rendering the tree structure on the page.

5.1.1 Backend

The backend of the application consists mainly of three parts:

- **controller** - includes controllers that create REST API and enable communication between backend and frontend components. Receiving a request and sending a response is guaranteed by Spring. Controllers should not have any logic; their main goal is to collect data from various services and send them to the frontend.

- **service** - part of the backend containing the main logic of the entire application. For example, with the help of the service, we can receive data from the MANTA Flow server using an HTTP client. Another group of services is used to store data in an internal database. The last one deals with saving data to the CSV files and retrieving data from them.

- **model** - the model includes classes for storing the information about the objects we work with. These are usually POJO objects, such as

Node or Resource. Another type is DTO - special immutable classes in which data are stored structured according to the needs of the front end. Controller converses regular classes to DTO and vice versa with the help of special converter classes. The last type is DAO, which allows us to communicate with the internal database.

For the first phase, the backend part is pretty simple. There are two model classes - Resource and Node; further, there are two corresponding DTO files and a converter between them. These classes are used to store information about the Repository tree items received from the server and to transfer them to the frontend. To response to frontend requests, we have implemented `OpenMantaRepositoryTreeController` with two endpoints:

- `getResourceNodes` returns a list of all resources from the server
- `getChildNodes` returns a list of child nodes of the given node

The frontend uses these endpoints to dynamically build the Repository tree (see section 5.1.2 for details).

5.1.2 Frontend

The frontend part already has a more complex structure and uses many different frameworks to facilitate the work. The main parts of the module are

- **interface** - JavaScript is generally a dynamically typed language, and it is unusual to use interfaces or classes here. However, because we use the Flow library, which simulates static typing, we can define classes and interfaces similar to the Java language. This package has the same role as the backend model and should exactly match the backend DTO so that all parameters can be mapped correctly.
- **components** - using React, we can build an HTML DOM from smaller parts called components. We can then use the classic OOP approach with components and treat them as classes. Thanks to React, the code is much more readable and understandable, better structured, and easier to support.
- **actions** - consists of one file representing a set of simple objects (they have only two parameters - type and payload) - the so-called actions. We can think of actions as events, we can invoke them anywhere in the code, and redux or saga will respond to that.
- **redux** - the part of the application that holds it together and unites its particular elements. It contains a state accessible from all places in the code; for example, nodes for the Repository tree are stored here.

- **saga** - middleware guaranteeing asynchronous communication between frontend and backend.

In the first phase, the main goal was to implement the dynamic Repository tree. It would be more straightforward to store data in a tree structure and simply render it one-to-one. Unfortunately, the redux state is immutable, and we must create a new state and replace the old one every time we need to update it. With such an approach, it is complicated to maintain a tree structure, and it will not allow making some modifications based just on the node id. Therefore, we decided to use normalization and save the tree as a node map. So we can find any node from the tree by id, and if we need its children, then each node has a list of its children's ids so we can easily find them in the map. The resulting node map can look like this:

```
{
  "1": {
    "id": 1,
    "name": "node1",
    "type": "Database",
    "resourceName": "Oracle",
    "path": "/Oracle",
    "attributes": [],
    "childNodes": [2],
    "flows": []
  },
  "2": {
    "id": 2,
    "name": "node2",
    "type": "Table",
    "resourceName": "Oracle",
    "parentId": 1,
    "path": "/Oracle/node1",
    "attributes": [],
    "childNodes": [3],
    "flows": []
  },
  "3": {
    "id": 3,
    "name": "node3",
    "type": "Column",
    "resourceName": "Oracle",
    "parentId": 2,
    "path": "/Oracle/node1/node2",
    "attributes": [],
    "childNodes": [],
    "flows": []
  }
}
```

We used the `TreeView` component from the Material-UI framework to create the tree; it solves many things by itself, such as expanding and collapsing nodes. The entire workflow of the Repository tree then looks like this:

1. We call the `getResourceNodesRequest` action during the initial rendering of the tree component; it is caught and managed by the corresponding saga. The saga then sends the requests to the backend, which returns results from the Repository API already prepared for the frontend purposes. Finally, the saga saves the received data to the redux, the tree component re-rendering starts automatically and displays the resource list on the page.

2. When we click on the node, it checks if the children have not already been loaded; if not, the `getChildNodesRequest` action is invoked. The saga catches it and sends a request to the backend, which returns the necessary data in the same way as above. However, we cannot save the received data directly to redux because the state is normalized, so we need to perform a proper merge:

   ```
   {
       ...state,
       nodes: {
           ...state.nodes,
           ...keyBy(action.payload.childNodes, 'id'),
           [action.payload.nodeId]: {
               ...state.nodes[action.payload.nodeId],
               childNodes: action.payload.childNodes
               .map(childNode => childNode.id)
           }
       }
   }
   ```

   First, we need to copy the previous state and replace the `nodes` attribute in it. For `nodes`, we must perform the same operation and add all received nodes to the list. Finally, we need to add the node IDs to the list of children in the parent node.

5.2 Phase 2 - Create, edit, and remove operations

In the second phase, we implemented creating, editing, and deleting the application elements, such as nodes and their attributes, resources, layers, and flows. All these operations have the same principle and look the same, with a few exceptions, so that we will discuss the implementation of these parts using the resource processes as an example.

5.2.1 Backend

We will first discuss the way from the backend to the frontend. When the `OpenMantaRepositoryTreeController` receives a request from the frontend, it passes data to the `RepositoryService`, which resolves the connection to the storage API server. Communication between the backend and the MANTA Flow server is implemented using an HTTP client from the Spring framework. It can connect to the desired endpoint and return data in JSON format, which is then easily mapped to a Java object using the Jackson framework. This way, we get the data into our internal `Resource` representation, but the frontend expects a slightly different structure. Therefore, we use the `ResourceNodeDTOConverter` to convert a `Resource` object to a `ResourceDTO` object. Now the data is fully prepared, and we will send it to the frontend the same way using the Spring. The whole process is described in figure 5.1.

The reversed process starts when we press the ”Save Changes” button; it should save the information about new resources to the appropriate file. Because all data are sent at once, there is a special `OpenMantaChanges` container, where the changes are stored. Once we convert the JSON object from the request to the DTO and then to the backend model object, we can pass it to the `ChangesService`, which contains the logic for storing the data in a file. We have prepared specific classes describing one CSV line for each file (for resources is the `ResourceCsvConfigEntry`). So we can convert objects to CSV rows and vice versa, here we can also provide additional data validation before saving or after loading. Figure 5.2 illustrates the whole process.

5.2.2 Frontend

On the frontend, operations for different elements also look similar; these are creating new elements, editing existing or newly added elements, and deleting new ones. As described earlier, all data from the backend are stored directly in the redux state and then loaded into components as needed. However, we will use a different approach to store new elements. To manage input forms, we use the Formik library, which controls input data from the user and allows us to validate them. It keeps all data in its own state, and we will use this state to store all the new elements. Thanks to this solution, we do not have to deal with updating data in the redux, and all changes are saved automatically. In addition, after pressing the ”Save Changes” button, we get access to the Formik state with all changes.

Let us take a closer look at the whole process on the example of a node attribute. In the initial state, the node has no attributes, so the button to add a new one appears in the list. When pressed, the application generates a

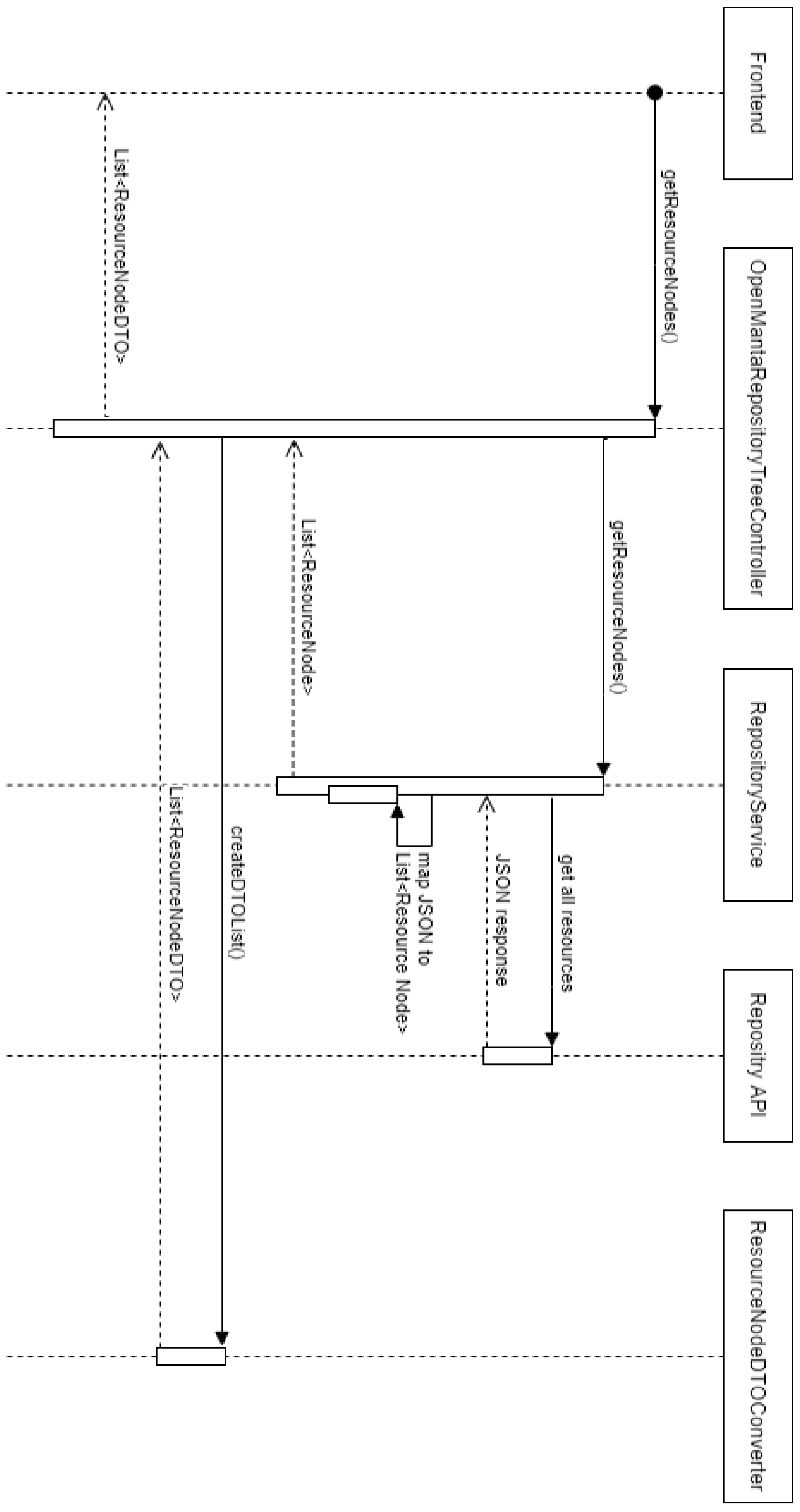


Figure 5.1: Open MANTA Extensions resources retrieving sequence diagram

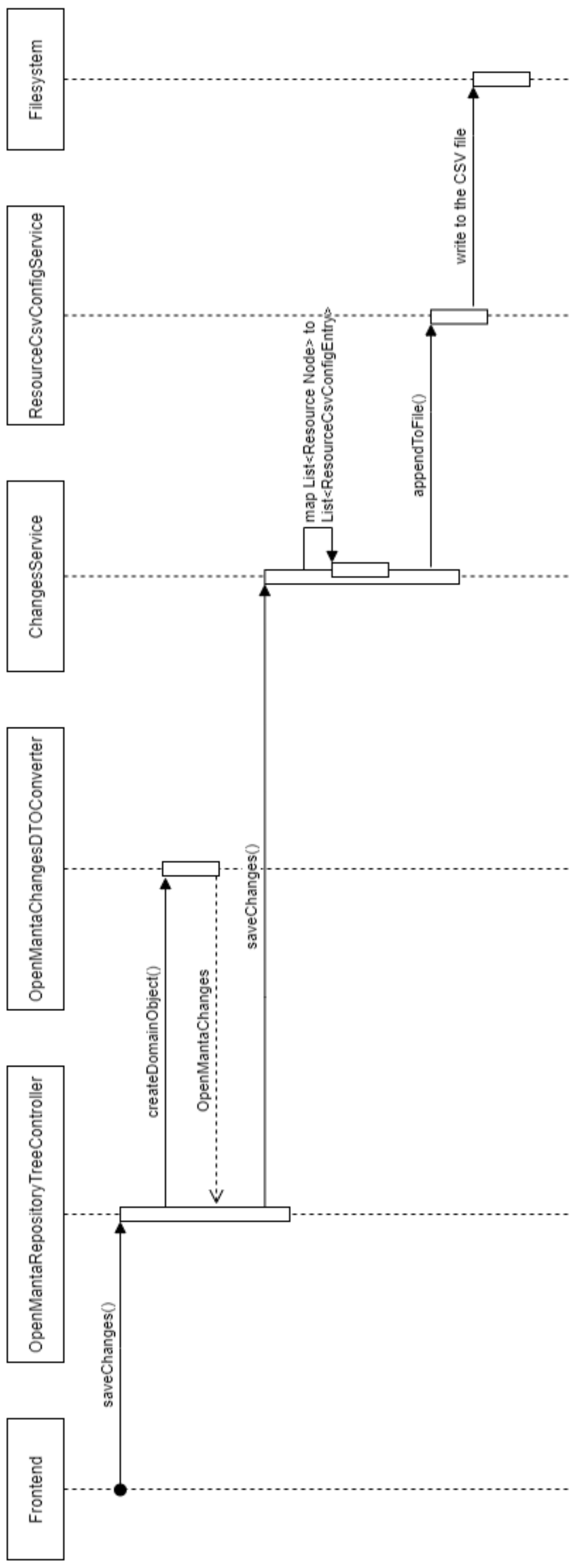


Figure 5.2: Open MANTA Extensions resources saving sequence diagram

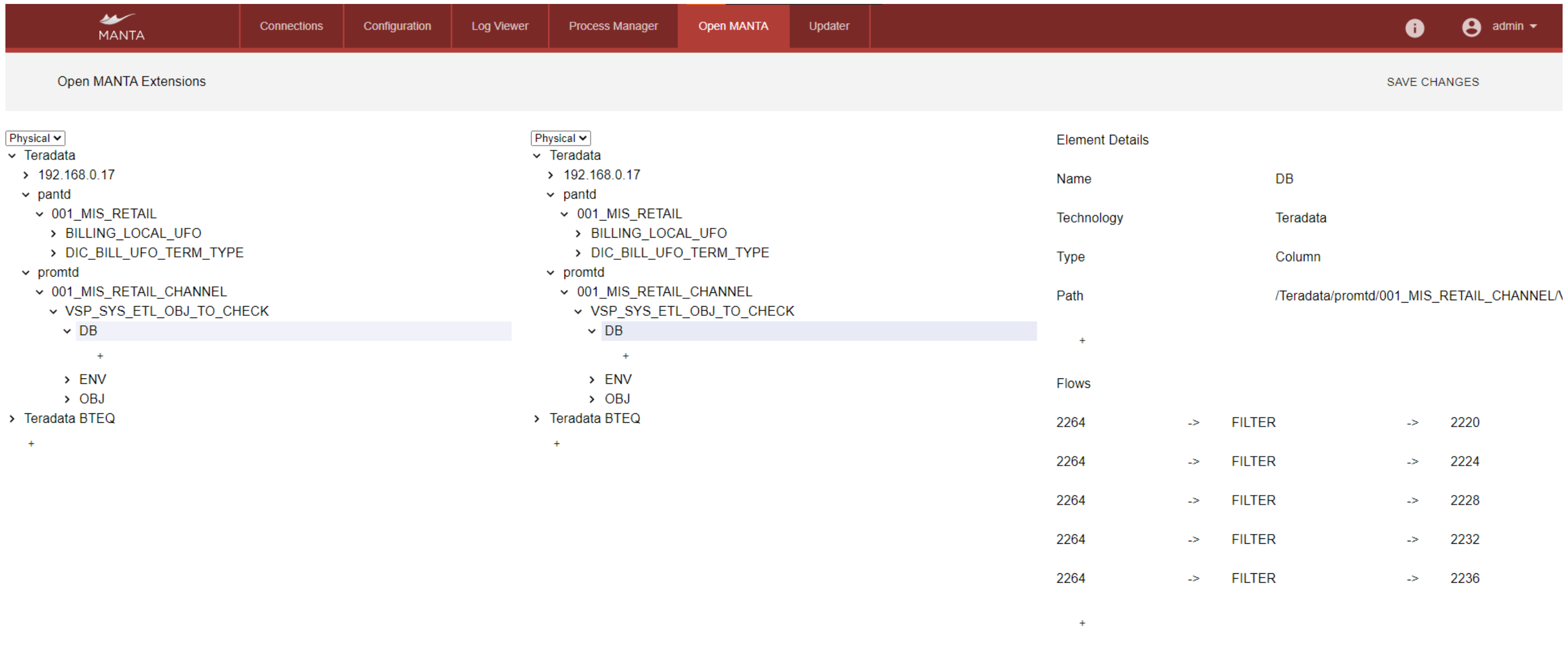
new attribute with a unique id within the node and inserts it into the Formik state. After rendering, we see the input forms for the attribute name and value, which are already synchronized with the state. There is also a button to remove the attribute that erases the record from the Formik state. After pressing the ”Save Changes” button, all attributes are combined into one list and sent to the server for further processing.

5.3 Next Phases

As the entire scope is too large, only the main parts of the module were implemented in this work. The result is illustrated in Figure 5.3.

Although the prototype is already functional and meets most requirements, many functions remain unimplemented. The following improvements are already planned:

- Adding the possibility to create flows directly between the Repository trees by drag-and-drop. This feature makes it easier for the user to use the application and speeds up his work.
- Adding the Preview Changes page that summarizes all the changes made. Thanks to this page, the user will see all the changes in one place and be confident that nothing was forgotten.
- Adding user input validation and other restrictions, such as creating borders for leaf nodes only, unique resource names, and only defined node types.
- Implementing a temporary data store in the internal database. This feature allows to store unsaved changes on the fly and to prevent data loss in the case of a connection failure.
- Implementing loading saved changes from the CSV files. The user will be able to go back to the saved changes even after restarting the application.
- adding the ability to work on multiple changes at once. It allows the user to save changes under unique names and work in multiple contexts simultaneously. It will also be necessary to resolve conflicts between multiple changes before saving.
- The last point is to use the Admin GUI style for the module. An unstyled prototype is enough to demonstrate functionality, but users need to have nice-looking software.



```
Physical
⌄ Teradata
  › 192.168.0.17
  ⌄ pantd
    ⌄ 001_MIS_RETAIL
      › BILLING_LOCAL_UFO
      › DIC_BILL_UFO_TERM_TYPE
  ⌄ promtd
    ⌄ 001_MIS_RETAIL_CHANNEL
      ⌄ VSP_SYS_ETL_OBJ_TO_CHECK
        ⌄ DB
            +
        › ENV
        › OBJ
› Teradata BTEQ
  +
```

```
Physical
⌄ Teradata
  › 192.168.0.17
  ⌄ pantd
    ⌄ 001_MIS_RETAIL
      › BILLING_LOCAL_UFO
      › DIC_BILL_UFO_TERM_TYPE
  ⌄ promtd
    ⌄ 001_MIS_RETAIL_CHANNEL
      ⌄ VSP_SYS_ETL_OBJ_TO_CHECK
        ⌄ DB
            +
        › ENV
        › OBJ
› Teradata BTEQ
  +
```

Element Details

Name	DB
Technology	Teradata
Type	Column
Path	/Teradata/promtd/001_MIS_RETAIL_CHANNEL/\

+

Flows

2264	->	FILTER	->	2220
2264	->	FILTER	->	2224
2264	->	FILTER	->	2228
2264	->	FILTER	->	2232
2264	->	FILTER	->	2236

+

Figure 5.3: Open MANTA Extensions prototype demonstration

Conclusion

The main goal of this work was to design and implement a functional prototype of a web application module for user-defined extensions of metadata and data flow in the MANTA project. As part of the work, we collected the requirements and then performed their analysis. We designed and implemented a module that provides a user-friendly way to expand the data flow based on the performed analysis. The resulting module is fully documented using JavaDoc and JSDoc. In this way, we met all the primary and partial goals of this work.

The created module has limited functionality because it is only a prototype, but it is ready for improvement. We plan to continue expanding it, add the ability to create edges with drag-and-drop, implement an internal database for continuous saving of changes, and much more. The module was created with the assumption of subsequent integration into the MANTA Admin GUI and uses all the necessary system resources from the beginning of the implementation. Therefore, we can soon put the prototype into production as a demo version.

This work has become a great personal benefit because many different and new tools have been used during the implementation, particularly JavaScript and React. During the implementation of the prototype, I expanded my knowledge of this language and its frameworks and gained a lot of experience with creating MVC web applications. I truly liked the whole process of creating the module, and I would definitely continue to improve it and expand it with other functions.

Bibliography

1. Wikipedia: The Free Encyclopedia. *Web application* [online]. 2021-04 [visited on 2021-04-27]. Available from: `https://en.wikipedia.org/wiki/Web_application`.
2. Manta Tools, s.r.o. *Co je to Manta Flow?* [Online]. 2018-01 [visited on 2021-04-25]. [Internal resources of the company].
3. Wikipedia: The Free Encyclopedia. *Data lineage* [online]. 2021-02 [visited on 2021-04-26]. Available from: `https://en.wikipedia.org/wiki/Data_lineage`.
4. Plskova, Katerina. *Data Lineage from… Microsoft Excel?!?* [Online]. 2019-07 [visited on 2021-04-26]. Available from: `https://getmanta.com/blog/data-lineage-from-microsoft-excel`.
5. Szczech, Zosia. *MANTA 3.32: Google BigQuery, Microsoft SQL Server, Docker, Admin UI Enhancements, and More!* [Online]. 2021-04 [visited on 2021-04-27]. Available from: `https://getmanta.com/blog/manta-3-32-google-bigquery-microsoft-sql-server-docker-admin-ui-enhancements-and-more`.
6. *Why Spring?* [Online]. © 2021 [visited on 2021-05-06]. Available from: `https://spring.io/why-spring`.
7. Yash_Maheshwari. *What is JavaDoc tool and how to use it?* [Online]. 2020-11 [visited on 2021-05-01]. Available from: `https://www.geeksforgeeks.org/what-is-javadoc-tool-and-how-to-use-it`.
8. Abramov, Dan. *Getting Started with Redux* [online]. © 2015–2021 [visited on 2021-05-01]. Available from: `https://redux.js.org/introduction/getting-started`.
9. Hannah, Jaye. *What Exactly Is Wireframing? A Comprehensive Guide* [online]. 2021-04 [visited on 2021-05-03]. Available from: `https://careerfoundry.com/en/blog/ux-design/what-is-a-wireframe-guide/#1-what-is-a-wireframe-and-who-uses-them`.

Appendix **A**

Acronyms

API Application Programming Interface.

BI Business Intelligence.

CLI Command Line Interface.

CSV Comma Separated Values.

CTU Czech Technical University in Prague.

DAO Data Access Object.

DI Dependency Injection.

DOM Document Object Model.

DTO Data Transfer Object.

ETL Extract, Transform, Load.

FIT Faculty of Information Technology.

GUI Graphical User Interface.

HTML Hypertext Markup Language.

HTTP Hypertext Transfer Protocol.

IBM International Business Machines.

IDE Integrated Development Environment.

IoC Inversion of Control.

JSON	JavaScript Object Notation.
MVC	Model, View, and Controller.
OBIEE	Oracle Business Intelligence Enterprise Edition.
OOP	Object-Oriented programming.
PDF	Portable Document Format.
POJO	Plain Old Java Object.
REST	Representational State Transfer.
SQL	Structured Query Language.
SSRS	SQL Server Reporting Services.
UI	User Interface.
UX	User Experience.

Appendix B

Open MANTA Extensions use case diagrams

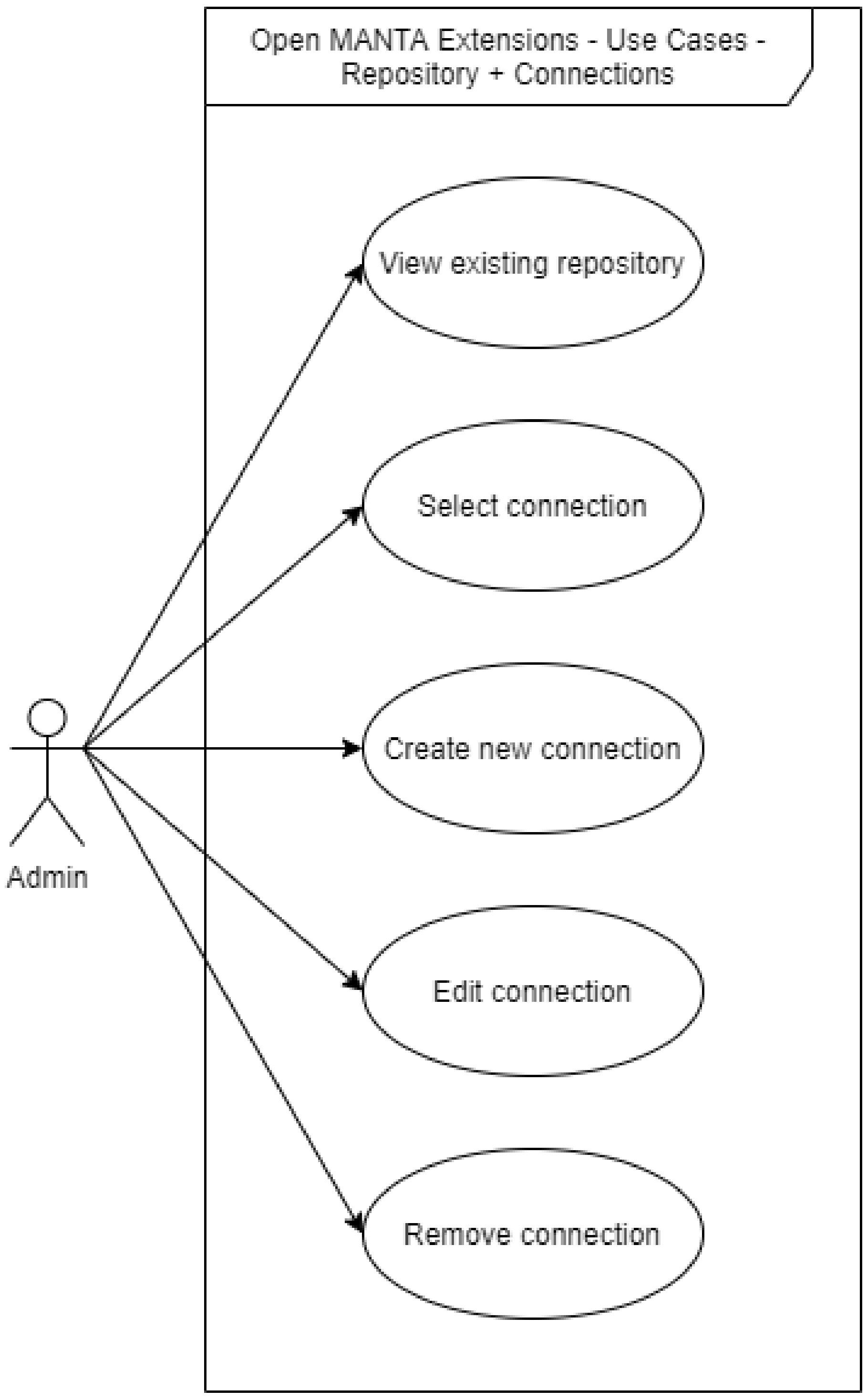


Figure B.1: Open MANTA Extensions use case diagram for the repository

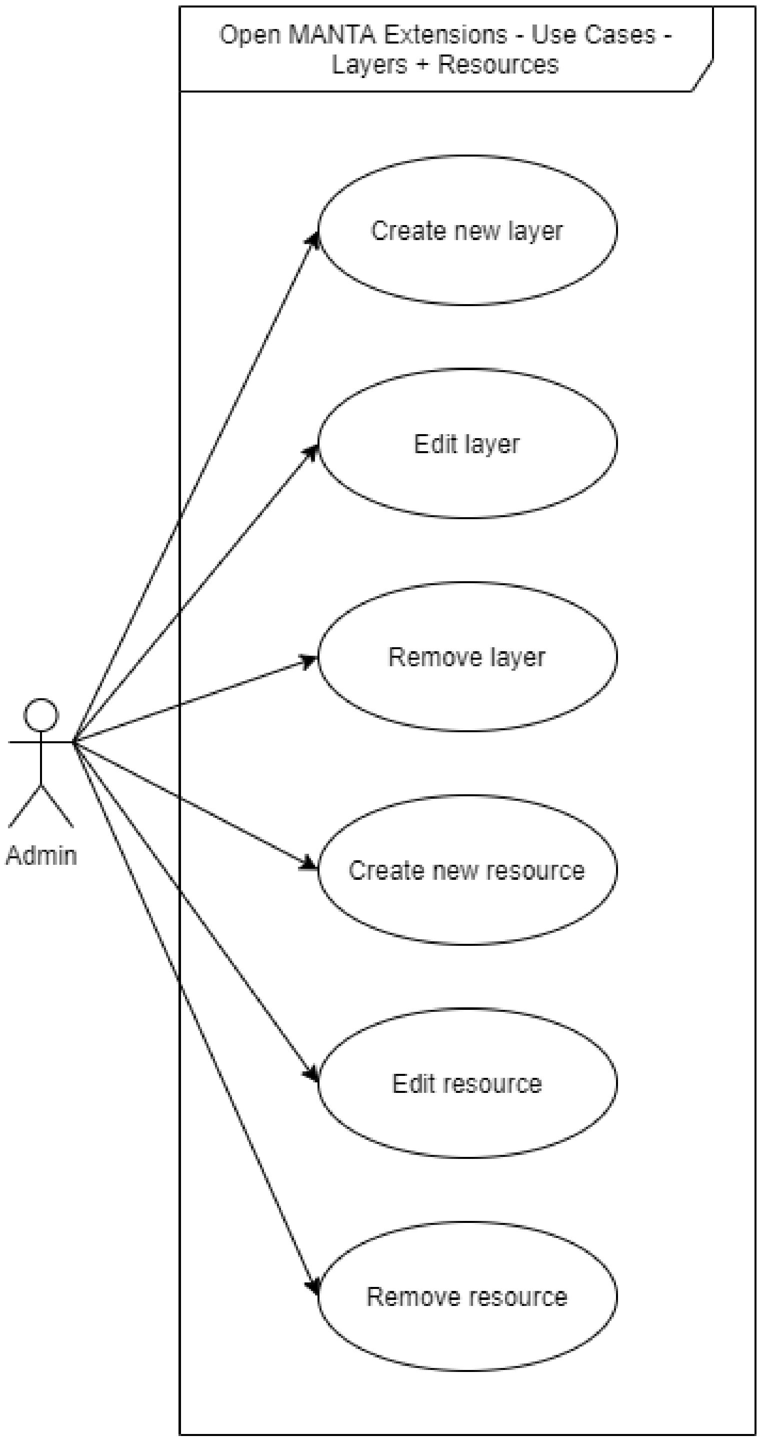


Figure B.2: Open MANTA Extensions use case diagram for layers and resources

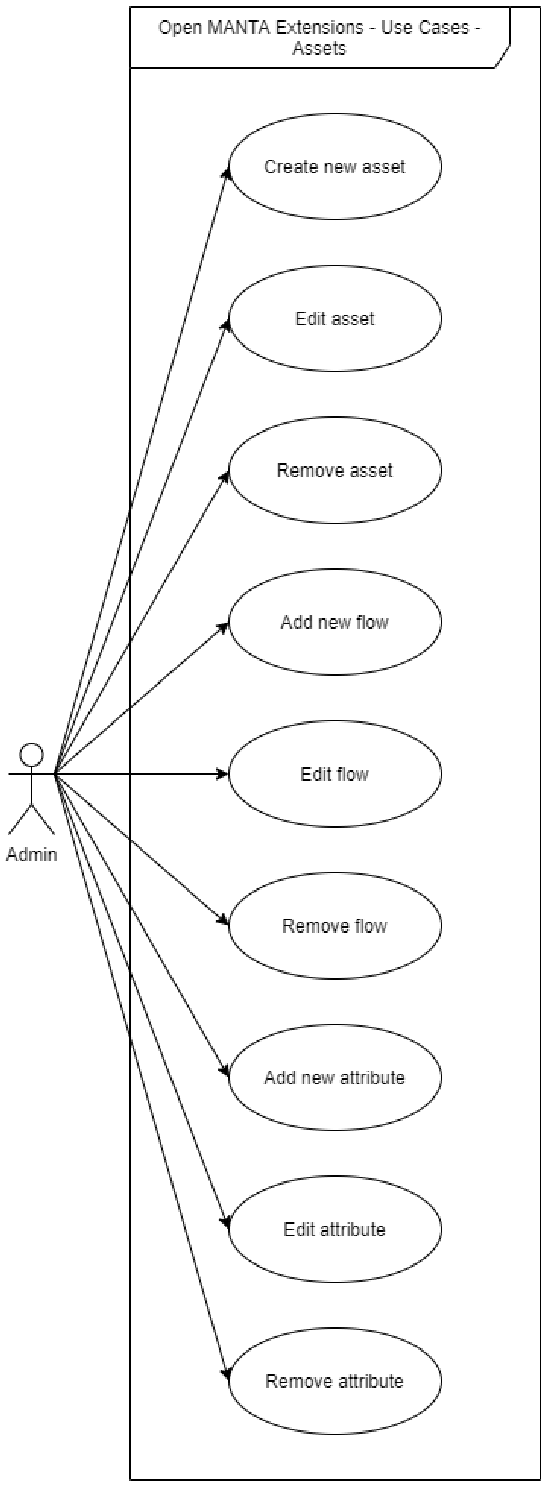


Figure B.3: Open MANTA Extensions use case diagram for assets

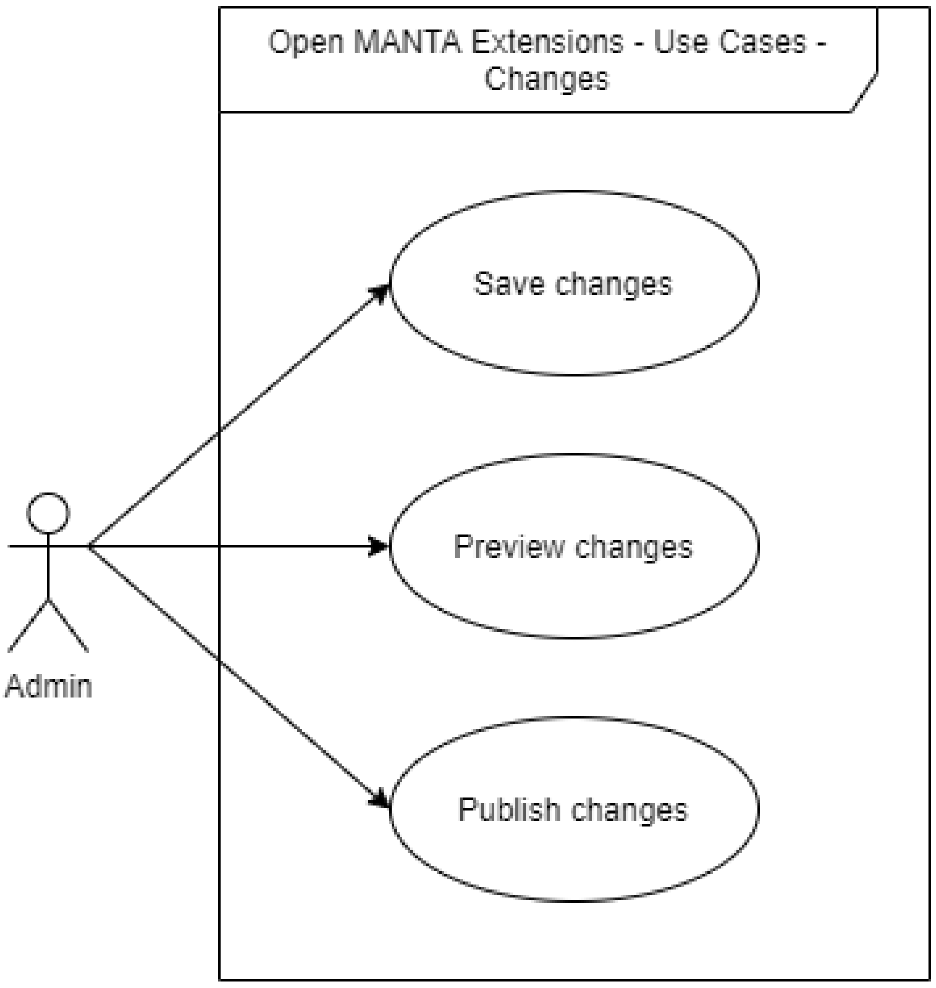


Figure B.4: Open MANTA Extensions use case diagram for saving changes

Appendix C

Contents of enclosed CD

```
readme.txt ...................... the file with CD contents description
src.................................the directory of source codes
  impl..................................implementation sources
  thesis.............the directory of LaTeX source codes of the thesis
  wireframes ........................... the directory of wireframes
text ......................................the thesis text directory
  DP_Buldyk_Yauheniy_2021.pdf ........ the thesis text in PDF format
```