**UNIVERSITY OF PLOVDIV "PAISII HILENDARSKI"**

**FACULTY OF MATHEMATICS AND INFORMATICS**

**DEPARTMENT OF "SOFTWARE TECHNOLOGIES"**

**Veselina Rumenova Naneva**

# Web-Based Data Visualization Tools

# ABSTRACT

of DISSERTATION WORK

for awarding the educational and scientific degree "*Doctor*"

in the Higher education field: *4.* Natural sciences, mathematics and informatics,

Professional field: *4.6. Informatics and computer science*;

Doctoral program *Informatics*

**Scientific supervisors:**

Professor Angel Golev, PhD

Professor Nikolay Pavlov, PhD

Plovdiv, 2023

The dissertation work has been discussed and scheduled for defense at an extended meeting of the "Software Technologies" department at the Faculty of Mathematics and Informatics of Plovdiv University "Paisiy Hilendarski" on 18.10.2023.

The dissertation "Web-Based Data Visualization Tools" contains 121 pages, of which 106 are main text. Additionally, a 19-page Appendix with accompanying code snippets is included. The cited literature includes 86 sources. The list of author publications on the subject contains 4 titles, all of which are in English.

The defense of the dissertation will take place on 22.01.2024 of 12:00 h. in the Meeting Hall of the New Building of Plovdiv University "Paisii Hilendarski", Plovdiv.

The defense materials are available to those interested at the FMI secretariat, New PU Building, cab. 330, every working day from 8:30 AM to 5:00 PM.

**SCIENTIFIC JURY:**

**Chairman:**

Prof. Asen Kanchev Rahnev, PhD (Plovdiv University "Paisii Hilendarski", Plovdiv)

**Members:**

1. Prof. Eng. Todor Atanasov Stoilov, DSc (Institute of Information and Communication Technologies, Bulgarian Academy of Sciences, Sofia);
2. Prof. Georgi Teoharov Tuparov, PhD (New Bulgarian University, Sofia);
3. Prof. Kolyo Zlatanov Onkov, PhD (Agricultural University, Plovdiv);
4. Prof. Asya Georgieva Stoyanova-Doycheva, PhD (Plovdiv University "Paisii Hilendarski", Plovdiv).

Author: Veselina Rumenova Naneva

Title: "Web-Based Data Visualization Tools"

Plovdiv, 2023

# Table of contents

## Abbreviations used

**BI** – Business Intelligence

**ETL** – Extract, Transform, Load

**Power BI** – Microsoft Power BI

**API** – Application programming interface

**Triangle** – “Insurance Claims Triangle”

# Introduction

## Relevance of the topic

Many businesses today have a preference for the software products they use based on factors such as cost, availability, consistency of service collection, and more. For their part, the providers of Business Intelligence solutions compete periodically in updating the provided functionalities and capabilities of their products. This constant competition brings out a fundamental problem, which is the main focus of the present dissertation work. Migrating from one BI environment to another, or developing views that work correctly in Microsoft Power BI and Tableau environments at the same time, from the perspective of the custom extensions implemented to date, would cost both additional costs and time to learn the new technology. The implemented visualizations for enterprise purpose should meet the same customer requirements for database functionality, styling and accessibility as those of the originally selected product.

Despite conducting thorough research in recognized closed-access databases such as Web of Science, Scopus, and others, as well as exploring open-access databases containing periodicals, books, theses, conference papers, and other works, I was unable to find a library that offers the capability to build platform-independent custom visuals.

## Purpose and tasks of the dissertation

The aim of the dissertation is to model an architecture for the unified development of custom data visualization for Microsoft Power BI and Tableau environments with the possibility of integration with their specific APIs.

To attain the established objective, it is necessary to solve the following tasks:

1. To prepare a comparative description of the selected environments for interactive presentation of information, such as Microsoft Power BI and Tableau, focusing not only on their features as tools, but also on the available types of visualizations they provide;
2. To analyze and summarize the features in the application-program interfaces for the development of new visualizations, which each system imposes. Subtasks:

   2.1. Reflect on the specific custom view development technologies that Microsoft Power BI and Tableau support, indicating their advantages and disadvantages;

   2.2. Mention the steps for configuring components and the corresponding dependencies on external libraries and indicate the differences in the configuration files containing the metadata for the visualization type, version and usage license;

   2.3. To derive the main interfaces and constants common to both environments, providing the successful initialization and operation of the components.
3. To model an architecture for the unified creation of additional visualizations to Business Intelligence tools Microsoft Power BI and Tableau.
4. To develop visualization prototypes using the new unified architecture for creating custom data visualization for Microsoft Power BI and Tableau platforms.

## Dissertation Structure and Volume

The dissertation consists of an introduction, four chapters, a conclusion, an author's reference to the contributions, an approbation of the results, prospects for future development, a bibliography and an appendix.

The **first chapter** examines the specifics of data visualization in relation to the field of Business Intelligence. The main focus of the chapter is placed on the characteristic features of the two basic BI environments for the application of the architecture model laid down in the dissertation, namely Microsoft Power BI and Tableau. Their essences are indicated when creating a digital reference.

The **second chapter** defines the peculiarities of working with the application programming interfaces of the Microsoft Power BI and Tableau software products in terms of styling and data access. Sample visualizations are created with each specialized tool.

In the **third chapter**, the developed architecture model for unified creation of custom visualizations called TabWerBI is demonstrated. The main focus of the chapter is placed on the abstractions of the ways of accessing the data and the ways of shaping stylistic changes in the view by the user. Attached schematics and class diagrams illustrate the main interfaces, classes, objects and constants that should be used when implementing new visual elements.

The **fourth chapter** reflects a demonstration of a sample application of the unified visualization development architecture for Microsoft Power BI and Tableau by building an insurance claim chart, single field data label, Lollipop and Tree, and presents their capabilities in the two BI systems.

In the **conclusion**, the implementation of the formulated tasks is covered and the achievement of the set goal is analyzed. The possibilities for enrichment and development of the architecture in the future are presented.

**Appendix 1** includes all the code snippets used to illustrate the exact implementation of the components highlighted in the dissertation work.

# Chapter 1. Web-based data visualization

## 1.1. Business Intelligence

By definition, Business Intelligence is the process by which an enterprise uses algorithms and technology to analyze current and historical data in order to improve strategic decision-making and provide a competitive advantage thanks to it [50]. A variety of technologies are available that cover every single aspect specific to the field. The established process of extracting, processing and loading data called ETL stands out in the composition of BI, which is characterized by the application of heterogeneous technological tools for working and administering unstructured data. In the first paragraph of the dissertation, the tasks solved by the formed algorithm for managing raw data, storage and accessibility of a relational database are described in detail. An important aspect regarding the final BI product is the identification of principles in creating a dynamic report. Two of the most used BI environments for data visualization are Microsoft Power BI and Tableau.

## 1.2. Microsoft Power BI

Microsoft Power BI is an application that provides the ability to analyze and share business knowledge through the development of functional interactive reports. It is a tool implemented by Microsoft to transform and present data through ready-made visualizations that can be accessed

from Power BI Service, which refers to Software as a Service (SaaS) in the Power Apps category. In addition, interactive visual reports can also be created from the workspace using the on-premises version of Power BI called Power BI Desktop. Every dashboard created is a combination of a well-constructed data model and its appropriate presentation. Visualizations included in the corresponding panel require a different field type and position. New columns and calculations can be added using a specialized calculation expression language. A dashboard can combine local and cloud-based information, which provides a consolidated view, no matter where it is located [3, 4].

Despite the variety of ready-made visualizations, the environment provides the opportunity to add additional ones. If a suitable view is not found in the available visualization "store", the environment provides the option for the interactive reference to work with an added custom Power BI visual component to be used by the individual business or the entire Power BI community [60]. Power BI visuals are available through `.pbiviz` file packages, which include code for rendering the data and building visuals that the system requires.

### 1.3. Tableau

The field of Business Intelligence and data analysis use Tableau as a representative platform in the decision-making process, based on a variety of information processing methods. There are many subsystems in the Tableau ecosystem, each with a specialized purpose [31]. It offers various functionalities in terms of data sources, successfully working with Excel files, relational databases or tools from external cloud services. Another important aspect of data visualization is the dynamic report structure. There is a strict distribution of graphics, the combination of visual components and business history. Starting from the perspective of a visualization, it should be stored in a so-called "workbook". Each of these parts of Tableau requires fields to add columns and rows. Depending on the data type, i.e. text, integer, date and others, different extensions are available in the Show me tab.

The Tableau environment provides the ability to load both ready-made views and external ones implemented by authorized developers or views locally added according to the customized requirements of the given company. The new component integrates into the BI environment analogously to a web application and can have a configuration modal window, with the help of which the user can apply certain style settings [38, 39, 42]. For the purpose of proper integration of a new external extension, it should load the manifest file with a `.trex` extension to access the name, version number, license, and component files of the preview.

## Chapter 2. Development of customized solutions for presenting information

The reviewed BI tools Microsoft Power BI and Tableau have stand-alone APIs for building additional enterprise visualizations. These collections of classes, methods, interfaces, and constants assist developers in accessing data integrated into the databases of the respective environments, as well as configuration techniques for individual user setup. This chapter will focus on the possibilities for implementing new visualizations, the dependencies and constraints that each API requires.

### 2.1. Power BI Visual API

The application programming interface of the Power BI software product helps developing custom visualizations that work in concert with the tool's integrated database and communicate with other charts and panels. Power BI visuals are available in `.pbiviz` file packages. The

application of the custom visualization integrated into Power BI influences how the environment handles communication between the user, the visual, and the host. Actions and subsequent updates in Power BI can be initiated manually or automatically [66-68]. Types of updates include interactions of a different nature, detailed in the dissertation. The sequence of actions between the user, the visualization and the application programming interface depends entirely on the specifics of the customized extension itself, the volume of data that will be processed and the possibilities that it will provide to its users [12, 63, 67, 80, 81].

There are specialized methods and classes in the application programming interface that should be used when asserting the type of data access. Specificity stands out in that Power BI requires a specific mapping of the data to the view of the final visualization, not only when accessing the information itself, but also when modeling the configuration files underlying the extension. The visualization implemented through the Power BI API has the ability to access the specifics of the environment regarding user selections from a formatting panel.

### 2.1.1. Implementation technologies, dependencies and limitations

By default, development of custom extensions using API is done via TypeScript, because when installing dependencies, the file structure is created as a `.ts` oriented project. In terms of external services and libraries, the application programming interface maintains good compatibility with all JavaScript-based frameworks, and their features can be freely implemented. A detailed overview of the technologies is included in subsection 2.1.3 of the dissertation.

### 2.1.2. An example visualization implemented using the Power BI Visual API

In order to introduce the specifics of using application programming interface for Power BI, the development and implementation of a custom Lollipop visual will be explored. It usually presents a comparison between categories in a slightly more attractive way. It is important to specify the configuration component, which is the first main file, including a lot of meta information. This is `pbiviz.json`. Apart from that, the `capabilities.json` is also properly configured and should be adjusted according to the relevant requirements of the visual element, including the appropriate data mapping, the style modifications that will be applied and the type of fields that will be visualized in the environment. The graphic element itself is drawn using the D3.JS library for building visual charts and graphs. The main method provided by the relevant API when updating the status is demonstrated by Code 8 in the dissertation.

The end result is a colorful Lollipop chart that can get three fields: category, values, and target information displayed on the right side of the chart. The visualization is presented in Figure 1. The corresponding diagram interacts with the others bilaterally and can be customized from the individually created formatting panel (Figure 2).


*Figure 1. Lollipop visualization in a Power BI environment*

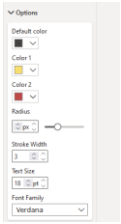


*Figure 2. Formatting panel in Power BI*

## 2.2. Tableau Extensions API

API for developing custom views for Tableau allows developers to build web applications that can interact with the Tableau environment [53, 55-58]. Due to the specificity of the software product, as described in Chapter 1, it is essential to distinguish the level of detail of the given visualization, because if the development is designed to reach the content of the entire dashboard, then the extension will demonstrate contact with all elements in it, including worksheets, markups, filters, parameters, and data sources. In addition, API namespaces are containers that contain classes and methods for communicating with Tableau components. The named top-level container specifies the global definition of the environment in which the extension will be deployed, namely tableau. There are also several named collections with narrow applications that represent a set of responsibilities and lists that manage the functionality of the visual component.

Regarding custom extension development, it should be noted that the API is tailored to work with pure JavaScript or TypeScript as means of developing the visualization functionality. Although API is designed as a JavaScript library, the Tableau environment also provides typed definitions that allow developers to take advantage of annotations and the application of object-oriented concepts.

In case of necessity to create a custom style setting for an additional visual element, Tableau's API provides an external configuration window. It is a kind of additional web application that connects to the methods to refresh the settings of the elements in the main visualization, having identical access to the given ones. To set up such functionality, a callback function should be set when the extension is initialized on the dashboard [36-38, 60].. The function creates a configuration option allowing the opening of a popup window (modal dialog) for the corresponding extension. It is possible to express a specific feature to load instantly when starting the preview or when searching.

### 2.2.1. Implementation technologies, dependencies and limitations

A visual application tailored for deployment in Tableau can work in combination with API technologies to build JavaScript-based functionalities. TypeScript development is also provided, which requires the use of recompiling and packaging the functional code with the **`.ts`** extension to a synthesized **`.js`** view. This in turn leads to the use of several dependencies such as Webpack [26] and JQuery that have relations to the extension’s source code. A detailed description of their features and purpose is available in subsection 2.2.3 of the dissertation.

### 2.2.2. Example visualization implemented using the Tableau Extensions API

In order to present an example visualization developed as a custom Tableau add-on, the steps to design a Tree graphic will be covered. Such a diagram type refers to a hierarchical representation of a data set, where the nodes are corresponding parent-child relationships rather than represented in a table. Visualization database selected in **`.xls`** format and integrated into Tableau Public. An active dashboard widget is required to load the new extension. A link to add a visual with access to a manifest **`.trex`** file from a local computer is reflected in the Dashboard section of the Tableau Public.

When initializing the view, a modal window with settings is provided. To implement the configuration, additional HTML is handled that will load a dialog screen to access certain data from the workbook. Also, a method is specified in the API to close the modal window and return the updated state of the extension. Figure 3 demonstrates how the extension is loaded into the

dashboard with a title coming from the corresponding HTML file. Despite a successful initialization, the preview does not present the aggregated data because there are configuration steps to consider first. The user should select each of the hierarchical levels from the corresponding worksheet [56]. This is an individual approach for the given newly added extension and is not a mandatory practice for all deployed graphics.

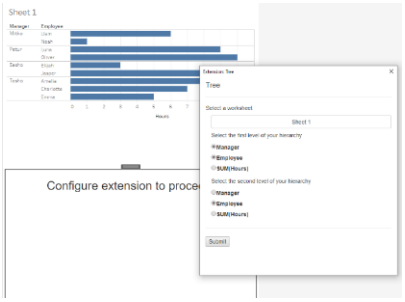


*Figure 3. Implemented Tableau Dialog Extension*

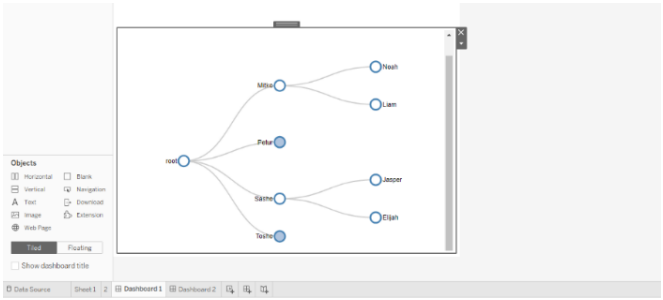


*Figure 4. Tree visualization in Tableau*

The process of developing a custom chart covers almost the same steps as creating a web component. The difference can be found in the functions that are associated with Tableau API to build visualizations and ensure the correct initialization in the respective software environment.

# Chapter 3. Architecture model for unified creation of additional visualizations to BI tools

In this chapter, we will consider the stages of shaping an architecture model for the unified creation of additional visualizations to the BI tools Microsoft Power BI and Tableau. It will trace the implementation of unifying external service dependencies through inheritance patterns, as well as specifying abstractions in the database data access model, the overall change in visualization and ways of styling it, summarized in the TabWerBI architecture.

## 3.1. Development technologies, dependencies and workflow

With a view to forming maximally uniform technological features, the following lines will present the selected development methods for the TabWerBI architecture, as well as the dependencies that should be foreseen in its use. When starting a new project, the developer should prepare, in addition to the initialization of the architecture for unified creation of visualizations, the set of external services, which will be specified in the subsection “Preliminary preparation”.

### 3.1.1. Preliminary preparation

#### Node.js и TypeScript

Given the fact that both BI systems require their extensions to be loaded as a standard web-based application, the need to install Node.js should be defined. The need for the service is described in detail for each of the approaches to developing personal views against the relevant APIs. Since TabWerBI brings them together, Node.js should be pre-installed in order to successfully initialize the new visualization.

Given that the Power BI Visual API uses TypeScript by default, and Tableau enables a typed development approach, TypeScript has also been chosen as the base tool for TabWerBI.

Choosing this programming language, attention should be paid to the need for **`tsconfig.json`** modification to be uniform for the purpose of unifying creation of new views. It is for this reason that a template has been developed whose inheritance in the base directory of a new visualization makes this possible. Usage is reflected as inheriting the specific template in tsconfig.json in the root directory of the given project via **`"extends":"./node_modules/ tabwerbi/ tsconfigtableau.json"`** or **`"extends":"./node_modules/tabwerbi/tsconfigpbi.json"`** depending on the deployment environment.

In addition, in subsection 3.1.1 of the dissertation, the need for the use of the **`@types/jquery`** package, the availability of **`webpack-cli & webpack`**, **`ts-loader`** and **`Semistandard`**, which have a close application in shaping the functional logic for the new visualization.

### 3.1.2. Power BI workflow

Regarding the workflow for creating additional extensions, the algorithm for developing a visualization through the Power BI API should be considered. As shown in Figure 5, it consists of six main steps. The corresponding command function, described in more detail in Chapter 2, initializes the compressed Power BI visual project. This forms a complete project ready for deployment in the respective environment. After a successful implementation, the manifest file coming from the API initialization is modified.

### 3.1.3. Tableau workflow

Project development through Tableau's API starts with shaping the visual concept of the graphic element. It goes through several basic steps for the environment, detailed in subsection 3.1.3 of the dissertation work. Having more complex functionality in additional configuration windows requires that they also be specified in webpack.config.js as modules for recompiling and packaging from TypeScript into readable JavaScript code. Figure 6 illustrates the steps required to create and load a new visual application into the given BI environment.

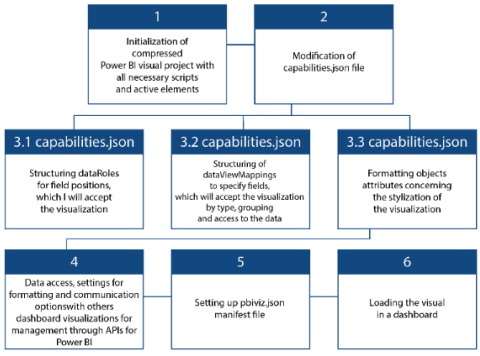


*Figure 5. Algorithm for developing and loading a visualization through the Power BI API*

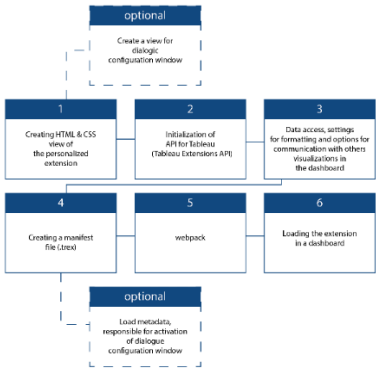


*Figure 6. Algorithm for developing and loading a visualization using the Tableau API*

### 3.1.4. TabWerBI workflow

Regarding the TabWerBI workflow, a combinatorial algorithm is derived, which is fundamental to the functioning of the model of the developed architecture. The schematic in Figure 7 shows the four-base environment-specific configurations. These include initializing the extension, accessing the database, and generating a specialized manifest file based on system-

specific requirements. Three steps are formed for loading and using TabWerBI, which reflect the installation of the architecture itself, the inclusion of references to the required data access space, and the basic styling. Additionally, the two custom commands to launch the visualization in the development process are also brought out, which fully work with the architecture.

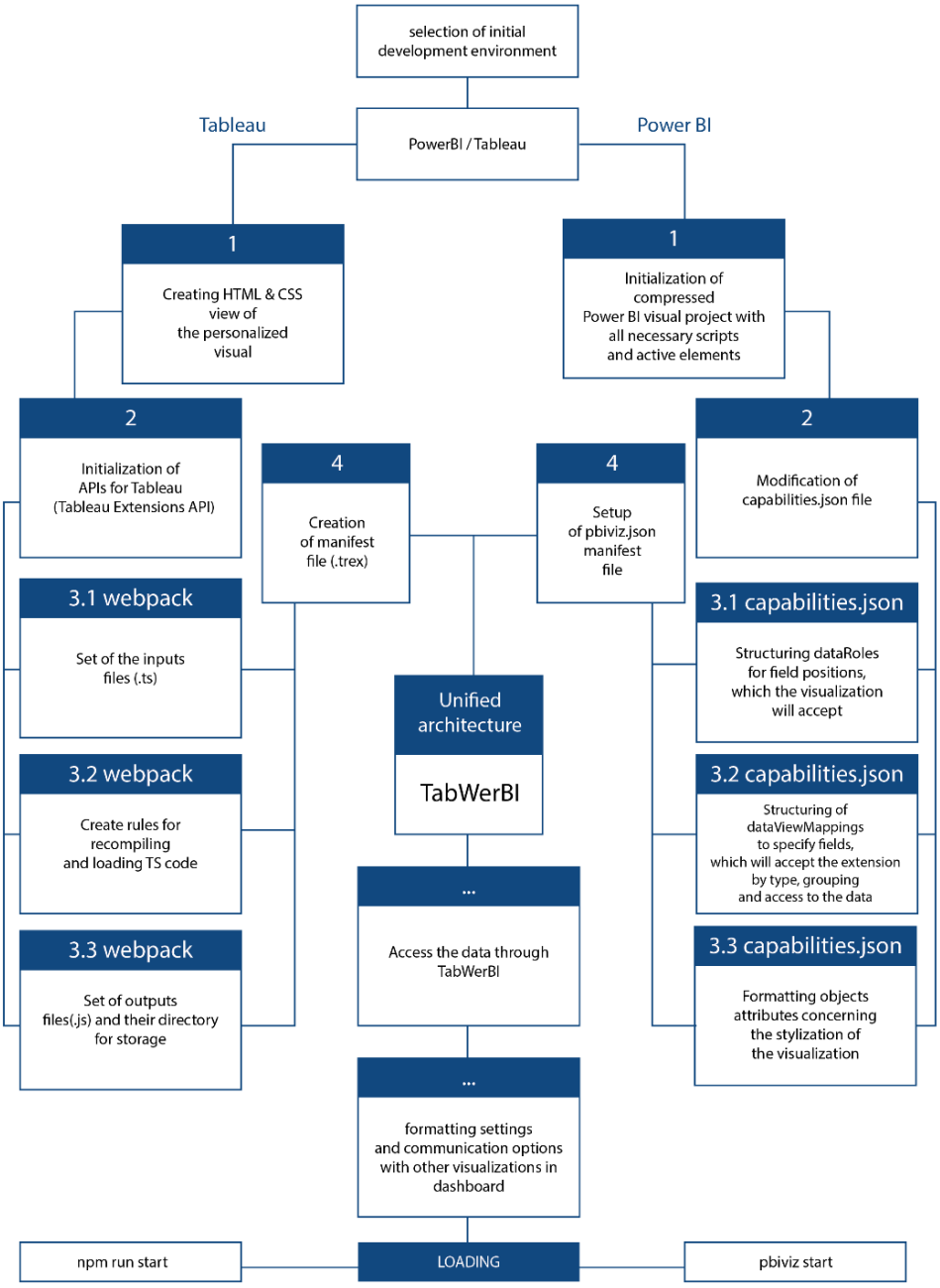


***Figure 7. Algorithm diagram for developing and loading a custom visualization using TabWerBI***

## 3.2. Abstraction of the database access model

Regardless of the choice of the initial tool, by going through the steps schematically indicated in the previous subsection, it is possible to work with the TabWerBI architecture model. It is installed as a standard external service and included in the base file containing the main logic

of the extension. Six methods have been developed to access information needed for a visualization that works with a single field, a graphical element presenting categorical data, and a tabular extension. They are summarized in two namespaces `TabWerBITableau` and `TabWerBIPowerBI`. Data access for the three visualization types is made uniform by instantiating the `getSingleData()`, `getCategoricalData()`, and `getTableData()` methods. When changing the environment in which the corresponding custom product is deployed, the namespace name and required arguments in the instantiated object should be changed.

### 3.2.1. Access the data through the Power BI API

To create a generalized access to the data in the form of a category view, a single visualization and one suitable for the construction of a table, three specific functions should be implemented to distribute the data in the desired way. For this purpose, methods `getSingleData()`, `getCategoricalData()`, `getTableData()` have been selected, the algorithm of which will be thoroughly reviewed in the following lines.

#### CustomSingleView

In terms of creating a visualization that works with a single field, the next step is to map one to a measure that returns a particular selected aggregation. In the specifics of the Power BI environment, the role of data in `capabilities.json` should be reflected. A `DataView` object is used when creating a method to work with single data. This makes it possible to handle the `single.value` property of the `dataView` object. The dissertation work details the step-by-step validations that are required to reach the actual implementation of the `getSingleData()` method to access a single field from the database. The method is demonstrated by Code 19.

#### CustomCategoricalView

Creating visualizations that resemble an axis chart, distributing one field along the `X`-axis and a value along the `Y`-axis, should be approached initially with a change to `capabilities.json`. Categorical data mapping is used to obtain independent groups or data distributed by category. After the configuration file has been successfully modified, the data fields should be processed using an appropriate method. For this purpose, `getCategoricalData()` has been created, which can be viewed via Code Snippet 20.

#### CustomTableMatrix

The matrix view in relation to the custom views enables the creation of multi-purpose visualizations. This type of data access forms columns, rows and values in a form suitable for the end user. Additionally, matrix mapping represents the rows hierarchically. The modification of the fields should again be reflected in `capabilities.json`. In order to shape the data access structure, a `getTableData()` method should be developed. Its logic is detailed in subsection 3.2.1 of the dissertation and is specified by Code 22.

### 3.2.2. Access the data through the Tableau Extensions API

Given the fact that there are no precisely formed classes responsible for shaping the three specific types of visualizations, similar to the previously described application programming interface, it is necessary to specify a separate way to access the data. Any custom extension developed through the Tableau Extensions API inherits the core Extensions interface. The function fundamental to each visualization is `initializeAsync()`. When mapping the data for Tableau environment, in this subsection are divided into three categories, respectively `CustomSingleView`

for single visualizations, CustomCategoricalView for categorical charts and graphs and `CustomTableMatrix` for table/matrix layout.

**CustomSingleView**

Due to the requirement that the `CustomSingleView` proccess a single calculated field, a way to retrieve the data from the current worksheet in the Tableau dashboard should be reflected in the single value visualization one. Code snippet 23 demonstrates the asynchronous function `getSingleData()`, which accesses summary data from a worksheet and calculates the number of unique values in the first column of that data.

**CustomCategoricalView**

Based on the requirement that the line chart type visualization has categorical access to the information; it is necessary to shape the data against the `X` and `Y` axis. Code 24 demonstrates the implementation of the `getCategoricalData()` method. Similar to the previous approach, asynchronous data extraction should be created using the `getSummaryDataAsync` method, and the obtained data is stored in `sumdata`. Additionally, `sumdata.data.map(row => ({ ... }))` maps each row of data from the aggregates into an object with two properties: `x` and `y`.

**CustomTableMatrix**

Regarding `CustomTableMatrix`, a way to retrieve the data from the current worksheet in the Tableau dashboard should be specified for the matrix view layout. The `getSummaryDataAsync()` method returns the summary data. After successfully extracting them, they should be processed. This is done via `.then((sumdata) => { ... })`. When working with the data, it is necessary to distribute it by rows and columns, and for this purpose a new `dataTable` object is created. It contains `columns: sumdata.columns.map(...)` and `data: sumdata.data.map(...):`, detailed in the dissertation. The implementation can be viewed through Code 25.

### 3.2.3. Model abstraction in TabWerBI

The unified architecture for creating custom extensions has access to the two main APIs for developing extensions for Tableau and Power BI. To implement the abstraction of the data access model, a TypeScript file is created in the root directory that includes the dependencies of the corresponding working libraries. Two namespaces have been developed that will summarize the methods and classes for the two systems under consideration. The namespaces are namespace `TabWerBITableau {…}` and namespace `TabWerBIPowerBI {…}` respectively. In order to access the specialized methods and classes when processing the data in the visualizations, relational connectivity with the namespace should be added. This is made possible by one of the following lines depending on the integration environment: `import {TabWerBITableau} from "tabwerbi/tabwerbi-pack/DataAccess/TableauTableDataAccess"` or `import {TabWerBIPowerBI} from "tabwerbi/tabwerbi-pack/DataAccess/PowerBITableDataAccess".` In the dissertation work, there is a formed place of the new architecture, divided by “A”, “Б” and “B”, which will be respectively named here by “A”, “B” and “C”, as follows:

**A → Access table view**

**TabWerBITableau**

To retrieve the data in Tableau, the `GetTableData` interface is used, which declares a worksheet property. The interface also reflects a method named `getTableData()`. The implementation is described in detail in the dissertation in subsection 3.2.3. In order to use the

custom architecture, in terms of matrix or table formation, the `GetTableData` interface should be implemented in the base class. Upon successful installation and inclusion of `TabWerBI` in a new project, it is necessary to create a new object of this `GetTableData` class in order to form the logic for retrieving the data and placing it in the respective desired positions.

**TabWerBIPowerBI**

When designing the table data access logic for a Power BI visualization environment, an interface that has a `getTableData()` method should be defined. The declared method has a `columnHeaders: string[]` and `rows: any[][]` structure that should be returned as a result. An overview of the properties it possesses is made in the relevant subsection of the dissertation. To use `GetTableData` in the main Power BI environment project, the class must implement the `GetTableData` interface.

**B → Single field access**

**TabWerBITableau**

Similar to the previously described model, for the correct implementation of the class that will process the single field for the Tableau environment, an interface with a worksheet property and the corresponding method should be viewed. In the current case, it's `getSingleData()`, which should return a Promise on data type any. The `GetSingleData` class implements the `GetSingleData` interface, as can be seen in Code 31. It is sufficient to create a variable that calls a new object from the `TabWerBITableau` namespace using the `GetSingleData` class.

**TabWerBIPowerBI**

For the operation of `getSingleData()` method in the Power BI environment, two interfaces `GetSingleDataResult` ((1), Code 32) and GetSingleData ((2), Code 32) should be implemented. `GetSingleDataResult` specifies a single property that is named `dataValue` and should be of type string. For the base `GetSingleData` interface, a `getSingleData` method is derived with a `dataView` parameter, which is required to return an object conforming to the `GetSingleDataResult` interface. For the correct use of this method, a variable should be used to initialize a new object of the `GetSingleData` class and call the field that the user placed in the corresponding UI component of the Power BI environment.

**C → Category access**

Categorical information is extracted to create visualizations that typically have `X` and `Y` axes based on different data types for both axes.

**TabWerBITableau**

To reflect access to the data through the Tableau Extensions API, a `GetCategoricalData` interface should be created, which has a worksheet property, similar to all the others discussed so far, and a `getCategoricalData()` method responsible for retrieving the categorical data. What is specific about this asynchronous method is that it expects to return an array of objects of the corresponding types described in Code 34. The way in which the `getCategoricalData()` method works is described in detail in subsection 3.2.2, and therefore only the implementation is demonstrated in Code 35 of the base class. To take advantage of the abstraction of the categorical data access model in the Tableau environment, the developer should create a new object of the `GetCategoricalData` class and, similarly to the other two methods, pass the required argument.

#### TabWerBIPowerBI

Regarding the correct implementation of TabWerBI to work with Power BI APIs, a `GetCategoricalData` interface should be created. It has a single method `getCategoricalData()` which takes as an attribute a `categoricalDataView` expecting an object of type `DataViewCategorical`. Code 36 demonstrates the `GetCategoricalData` interface. The data extraction algorithm for `getCategoricalData` completely matches that discussed in subsection 3.2.1. The use of the abstraction is analogous to the previously described.

Despite the existence of a large set of data mappings for both Power BI and Tableau environments, the unified architecture model for creating custom extensions covers the three main types of information access.

## 3.3. An abstraction of the custom visual extension output change formatting model

Another important aspect to consider covers the modification of a style layout of a custom extension. The two data visualization environments have differences in the configuration of the view formatting, which impose peculiarities in the implementation of the abstraction.

### 3.3.1. Custom Extension Styling via Power BI Visual API

The Power BI API requires a new object of the `VisualSettings` class to be created to access the formatting of the visualization. It inherits `DataObjectsParser`. For its correct operation, the interfaces working with the "Format your visual" section of the environment should be loaded. This core interface supports several functions and static objects detailed in subsection 3.3.1 of the dissertation work. Regarding the application of a style in the custom extension developed with the Power BI API, it is chosen to demonstrate how the background color, font, and text size change based on a user selection. The objects that can be specified in the style construction such as `backgroundColor`, `headerColor`, `fontSize` and `fontFamily` are pre-rendered. The way they are visualized in the patch panel is initially by specifying objects in `capabilities.json`. An example distribution of all implemented stylizations in the architecture for the unified creation of custom extensions is specified in Code 38.

The principle of operation of the personal preview at its launch, discussed in detail in Chapter 2, paragraph 2.1, requires that all changes be defined in the `update()` method. It is for this reason that for the initialization of formatting capabilities it should be specified `this.settings` in the specified method. The snippet from Code 41 and Code 42 can illustrate how `VisualUpdateOptions`, `parseSettings` (1) and `enumerateObjectInstances` (2) are accessed, responsible for overall changing the size of the visualization, the data it works with, its type and more.

### 3.3.2. Custom Extension Styling via Tableau Extensions API

The specifics of styling access to the custom graphics view cover the requirement to create a new modal window and include functional settings to change the formatting. This in turn means that the base class of the given extension should have a `configure()` method that accesses the following dependencies `dialogOptions`, `ui.displayDialogAsync()`, `settings.set()` and `settings.saveAsync()`. It is necessary to reflect the size of the modal window via `dialogOptions`. The main method that is specific to loading a dialog is `displayDialogAsync()`. The `initializeDialogAsync()` function is the entry point for a visualization that runs on a given

configuration window created by the UI namespace. In subsection 3.3.2, the method of forming a modal window is discussed in detail. The implementation of the architecture model provides specific id attributes that the developer needs to include in their web view to enable the dialog data to be sent to the base extension.

When developing a specific example dialog with a certain look, you should go through several basic steps, which are demonstrated by a snippet of Code 43, which also includes an implementation of the `closeDialog()` method to close the modal screen after a selection is made. In the base class of the additional extension, a `configure()` method is accessed, visible from Code 45. It contains access to the .html of the dialog box and the logic that should communicate with it. Given the fact that the developer can change the name of the .html file, as well as the dimensions of the popup modal, it would be desirable in the architecture not to unify these components.

### 3.3.3. Model abstraction in TabWerBI

The formation of an abstract model for inferring a change in the formatting of a personal visual extension aims to unify the way a developer will access an API algorithm with this focus. Due to the differences in the two environments and how they work, it is not possible to skip the configuration specifics.

#### TabWerBITableau

The abstraction of styling in a Tableau environment requires the creation of an interface that specifies the type of fields that are expected to reflect the defined formatting styles. In this particular case, they are strings, and for that reason each takes the type string. Through the `VisualSettingDialogTableau` class, the formation of a dialog box is brought out to the Tableau Extensions API. Because `displayDialogAsync()` takes as arguments access to the .html view of the formatting panel and its corresponding size on initialization, it forms an asynchronous method that requires `popupUrl` and `DialogSize` arguments when called.

Subsection 3.3.3 details the types of style settings that the method will need to handle and record in the appropriate variable. The main abstraction of the styling panel is handled via `initializeDialogAsync()`. The functional logic for it is in the additional configuration file to the base extension. In view of the fact that the environment distinguishes the main visualization from the one created for formatting settings, the method responsible for initializing a modal window in TabWerBI should be named appropriately. If custom settings are needed, the developer should use `DialogLogic()` from the namespace to access `initializeDialogAsync()` from the Tableau API.

In the TabWerBI architecture model for Tableau environment, `color1`, `color2`, `fontFamily` and `textSize` are provided to handle, which should satisfy the necessary basic style modifications that may be required when forming an individual corporate identity for the custom extension.

#### TabWerBIPowerBI

In terms of working with APIs for Power BI, the main dependencies `import { dataViewObjectsParser } from "powerbi-visuals-utils-dataviewutils"` have been added to the `TabWerBIPowerBI` namespace; and `import DataViewObjectsParser = dataView ObjectsParser. DataViewObjectsParser;`. This makes it possible to access `VisualSettings` inheriting `DataViewObjectsParser`. The main four formatting settings are formatted, with appropriate names given to reflect `color1`, `color2`, `textSize` and `fontFamily`. To define the abstraction for the Power BI widget deployment environment, four methods have been created that serve to access the relevant style settings. Each method has a single parameter that calls a `VisualSettings` type. Thus,

the `VisualSettings` class from `TabWerBIPowerBI` can be included in the core functionality of the visualization base file. The developed methods are discussed in detail in subsection 3.3.3.

# Chapter 4. Custom visualizations based on the TabWerBI architecture

This chapter will cover the development stage of a custom extension using TabWerBI and TypeScript, choosing a visual component aimed at presenting insurance claims and a single field to demonstrate a data label as an example application. In addition, two more specialized visualizations implemented in Power BI and Tableau are discussed.

## 4.1. Nature and application of the "Triangle of insurance claims"

Visualizations representing insurance claims are among the most important and specific forms of reporting in the insurance industry. The relevant experts aim to use a kind of claim triangles to predict the company's periodic losses to the most accurate figure possible, serving to maintain cost reserves. A thorough overview of the specifics of visualization and its application is given at the beginning of Chapter 4.

### 4.1.1. Output of necessary dependencies and configuration files

**Initially selected environment Microsoft Power BI (Power BI First)**

When starting development from a Power BI environment, the steps demonstrated for the basic TabWerBI workflow in Chapter 3 should be followed through a specific setting on both the `pbiviz.json` manifest file and the `capabilities.json` to reflect field access capabilities and style settings.

**Initially selected environment Tableau (Tableau First)**

When starting development from a Tableau environment, or after migrating from Power BI to your current environment, you should go through the elements covered as system-specific in Chapter 3. In a standard html file, you need to describe the structure of the extension, including basic identifier tags and classes that are subsequently used to access and process the data. Detailed configuration of the webpack package is demonstrated in subsection 4.1.1 of the dissertation work. Additionally, the `.trex` configuration file has been modified to reflect the ability of the visualization to be styled via the relevant metadata.

### 4.1.2. Developing the basic vision of the graphic element

The Insurance Claims Triangle preview is a table. In this particular case, the D3.JS library was chosen to create the peculiar table structure, as it provides a set of functions for binding data to the DOM and applying managed transformations to the document [71-73].

### 4.1.3. Access to the data required for visualization

When developing visualizations for the respective environments, the initialization is not uniform in the architecture and it is necessary to use the specific operations from the adjacent APIs in their base methods. Code 56 in the dissertation reflects the two main methods for loading a new view and changing its state, with (1) and (2) marking the two separate implementations.

Depending on the interactive reference software product in which the new visualization will be implemented, an appropriate namespace for accessing the developed methods is reflected. It would be active if the architecture was successfully installed and its dependency included via `import {TabWerBITableau} from " tabwerbi/tabwerbi-pack/DataAccess/TableauTableData`

`Access"` for Tableau deployment environment or `import {TabWerBIPowerBI} from " tabwerbi/tabwerbi-pack /DataAccess/ PowerBITableDataAccess"` for Power BI.

As illustrated in Code Snippet 57, a new instance of the `GetTableData` class is created, passing two necessary arguments `this.dataViewMatrix` and `this.target`. The layout logic in the Tableau deployment environment is identical. Thanks to the fact that a single framework is used to build the visual identity D3.JS, the data is accessed through the lines indicated in Code 58. It is enough to pass `this.worksheet` as an argument to the instance of the `GetTableData` class and observe the asynchronous mechanism, required by the respective environment.

### 4.1.4. Visual element formatting settings

As already detailed in Chapter 3, the APIs of the two environments differ except in the formation of style methods and classes and in that in Power BI direct access to the configuration panel is available, while in Tableau one should be built as a dialog window.

In the Power BI First approach, the import `VisualSettings = TabWerBIPowerBI. VisualSettings;` dependency should be added to load the main methods used to change the style settings. It is necessary to add several dependencies at the beginning of the visualization base file, as their structure and specifics are set out in subsection 3.3.1. For proper functionality, the `backgroundColor`, `headerColor`, `fontSize` and `fontFamily` objects demonstrated in Code 59 are reflected in `capabilities.json`. The method summarizing the construction of the table is `displayTable()`.

According to the Tableau First approach, a config.html file should be created that contains the necessary structure to simulate a configuration window for choosing colors, font, and text size. This is achieved by reflecting manipulation logic in `config.ts` against the required html attribute IDs. Code 65 demonstrates an implementation of a settings refresh method whereby a particular id with the contents of the custom extension is deleted and subsequently redrawn via `displayTable()`.

Regardless of the approach chosen for the implementation of the new visualization, additional steps should be taken in terms of stylization. As already mentioned, when developing the triangle, the initial environment was chosen to be Power BI, which necessitated the additional creation of config.html and `config.ts` for its implementation in Tableau. Additionally, settings specific to webpack.config.js and the manifest `trex` file should be noted, as it serves as a means of accessing the extension's business logic in a Tableau Public environment.

### 4.1.5. Implement visualization in Power BI and Tableau

After the basic visual build has been developed using D3.JS and TypeScript, all dependencies from other external services have been installed, and the TabWerBI architecture has been added via `npm install tabwerbi`, some basic configuration should be specialized in the individual BI environment. These are required for the custom extension to initialize correctly.

#### Loading the extension in environment Microsoft Power BI

Starting the developed visualization in the specified environment is done using `pbiviz start`. The dissertation work demonstrates the default view of the environment that presents a triangle with a sample database. Information is distributed by rows, columns and cells with no style settings applied. The fields are dragged into the corresponding positions, which are named and displayed thanks to the setting in `capabilities.json`. Also shown is the styling of the custom

view in Figure 9. The Format visual screen is enabled. It provides the ability to format visualizations.

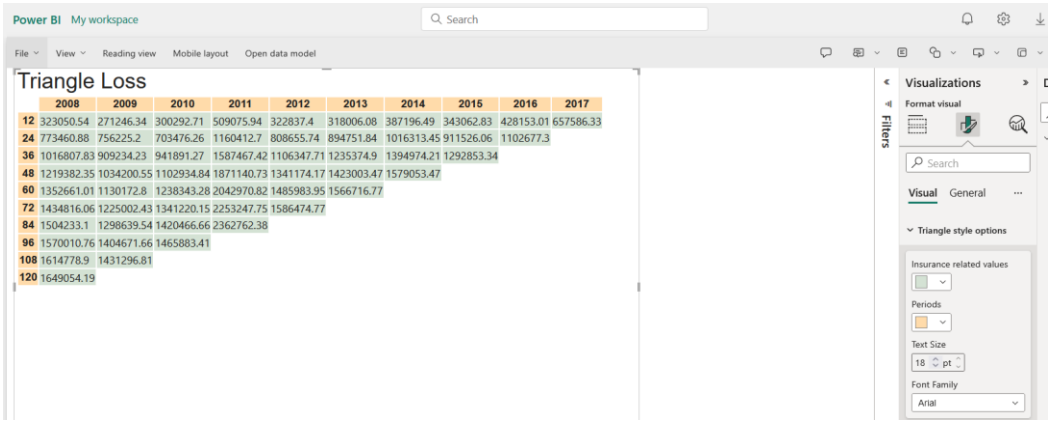


***Figure 8. View of a stylized preview through formatting panel settings***

### Loading the extension in environment Tableau

When deploying the visualization in the following BI environment, all installation dependencies that are properly described in the relevant subsection should be passed. Since the custom extension will have a styling addon, it should use the formulated render mechanism with an additional modal screen. To access the Tableau environment, the **`npm run start`** command line should be used.

The triangle view is rendered by default with no style settings. When selecting the configure option for the given extension, it can be seen what the configuration dialog looks like, which at the time of initial loading does not support user selection. When selecting given colors, font, and text size, the user sees their modal screen with preferences as written below each setting, and after choosing an Apply button, the dialog closes, applying the style modifications.

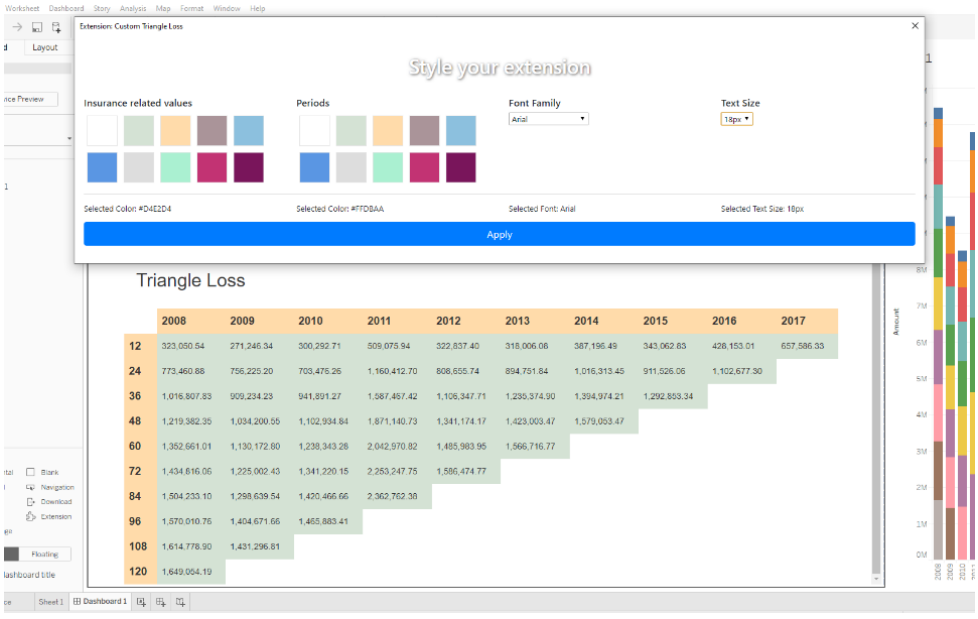


***Figure 9. View of a stylized preview by selecting settings from a dialog box***

## 4.2. Nature and application of “Data Label”

Developing visualizations that represent data labels form dynamic views that work with single fields. Their purpose is characterized by the fact that they can present information that can serve as titles, directional labels, and more.

### 4.2.1. Output of necessary dependencies and configuration files

Regarding the “Data Label” format, the steps, dependencies and specifics of the configuration files are identical to those discussed in subsection 4.1.1. The visual element requires the formation of a styling settings panel, which means that the Power BI First and Tableau First sequences of actions must be followed regarding the reflection of the specific attributes in `capabilities.json` and metadata in `DataLabel.trex` (manifest file for the new visualization.

### 4.2.2. Developing the basic vision of the graphic element

To form the view of the custom component, it is chosen to use the D3.JS library again. By looking at the basic structure of a `svg` element all of its style components can be modified. A dynamic rectangle is built that resizes depending on the selected custom text size and font setting that should be accessed relative to the deployment environment.

### 4.2.3. Access to the data required for visualization

To form for unified access to the single fields, a `GetSingleData` class implementing the `GetSingleData` interface, which is valid for both namespaces, should be used. Regardless of how they are formed in TabWerBI, loading the value required to represent a self-calculated data label defaults to accessing the number of unique records in the called field. This is done via the `getSingleData()` method, discussed in more detail in the dissertation work.

### 4.2.4. Visual element formatting settings

In terms of creating a style layout for the custom data visualization, identical steps should be followed to those reflected in the Triangle styling. A difference stands out in the specific elements that each formatting panel will have. When changing the view, it should be taken into account that the background of the Data Label element will change, as well as the size and font of the two text elements separately. This in turn means that the required style elements can be defined as `backgroundColor`, `headerFontSize`, `headerFontFamily`, `AmountFontSize` and `AmountFontFamily`.

### 4.2.5. Implement visualization in Power BI and Tableau

After finalizing the construction of the visual layout with D3.js and TypeScript, identical steps to those reflected in the Triangle should be followed. The TabWerBI architecture is similarly added via `npm install tabwerbi`.

#### Loading the extension in environment Microsoft Power BI

The information is targeted for use as a single value of a given field. It can be of different types. For the data label preview, the field is dragged into the appropriate position, which is named and rendered thanks to the setting in `capabilities.json`. The basic initialization method by which the Power BI environment communicates with the API provided by it is `update()`. The `displaySingle()` method is used in its body. The start of the visualization is similarly done via `pbiviz start` and is implemented again in the cloud environment of the corresponding BI system. A view of the element implemented in Power BI is demonstrated through Figure 10.

### Loading the extension in environment Tableau

Since the custom view should be modifiable to individual business preferences, the base method that should be present first in asynchronous **`initialize()`** is **`initializeAsync({ 'configure': async () => this.configure() }.`** In the initialization body, a **`displaySingle()`** method is called, which, similar to the implementation in a Power BI environment, serves to process the data, draw the graph, and use the style settings in the context of D3.JS.To access the Tableau environment, you should command line **`npm run start`** is used which activates analogously to the Triangle activates the necessary scripts The final already styled element in Tableau environment is illustrated by Figure 11.

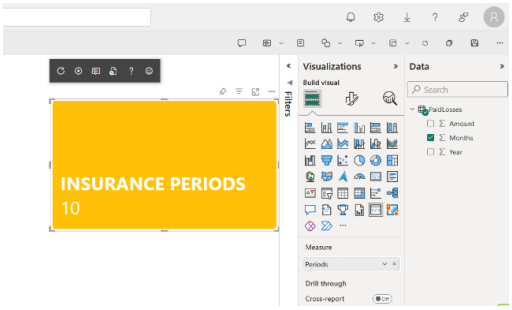


*Figure 10. "Data Label" visualization in Power BI environment*

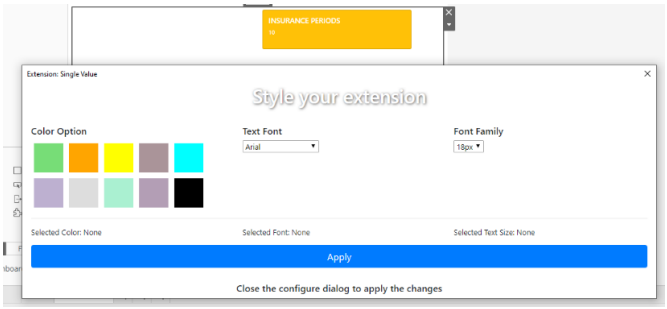


*Figure 11. "Data Label" visualization in Tableau environment*

## 4.3. Visualizations "Lollipop" and "Tree"

Establishing a development process for new visualizations involving the direct use of TabWerBI would greatly facilitate a possible subsequent migration to a different environment than the original one. The cases where the prototype architecture is implemented in an already built view of a visual component through "Lollipop" and "Tree" visualizations are considered.

Figure 12 demonstrates a "Lollipop" graph implemented in a Tableau environment that presents data on available radio frequency tower subscribers in the Washington area. The reference is pre-generated and the custom view [86] is implemented in it. The data included in the visual is presented in thousands relative to 2021 and reflects the number of users accessing the radio tower service, broken down by month. In order to obtain a clear indication of whether the optimal level of consumption has been reached, a target line is included in the visualization, relative to which the point data are positioned. The data was accessed through TabWerBI, with the data mapping type corresponding to a categorical model visualization. A **`getCategoricalData()`** method is used for both deployment environments. Due to the nature of data mapping in Power BI, the specific data distribution should be reflected in the **`capabilities.json`** configuration file.

Looking at the styling of the visual element, similarly to other implementations, **`const MainBackgroundColor = TabWerBIPowerBI`** should be accessed. **`getMainBackgroundColor (this.settings)`** when needed to change the color for a deployment environment running the **`TabWerBIPowerBI`** namespace. Modeling style settings for the Tableau platform require reflection of a new modal window with options corresponding to **`TabWerBITableau`**.

Figure 13 illustrates the application of a tree structure in a Power BI environment. A "Tree" visualization is designed to hierarchically represent given information by distributing the values in a stepwise manner relative to the highest "branch" of the tree. The visual element is implemented

so that all levels of the hierarchy are modeled when the data is loaded. In the demonstrated example, one can see such in a small company, as direct reporting employees, are tied graphically to their manager. For a cleaner look, the remaining managers are filtered by the filter panel in the Power BI environment. Data access corresponds to the category structure as a feature would be required in the definition of **`dataRoles`** and **`dataViewMappings`** objects specific to the configuration **`capabilities.json`** file against Power BI. Accessing the Tableau Category View data is made possible by using the **`this.worksheet`** argument in the reflected method.

In terms of visual styling, a personal configuration window with settings, including the ability to change the color of the point data, size and font of the text labels, should be created. Based on the **`VisualSettingDialogTableau`** class, an abstraction is derived for the formation of a dialog window to Tableau that would reflect the settings selected by the user. In a Power BI deployment environment, the formatter generated by **`capabilities.json`** and the **`VisualSettings`** class from **`TabWerBIPowerBI`** should be accessed**.**

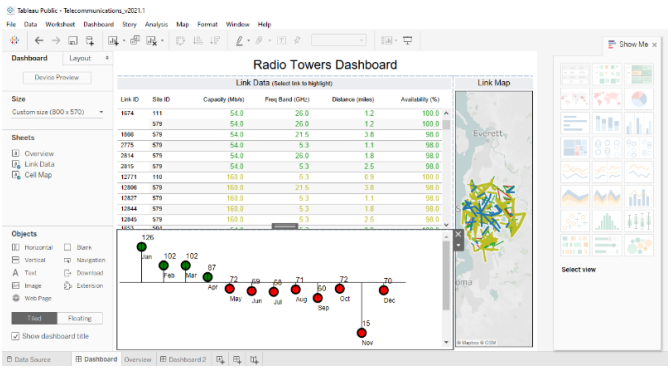


***Figure 12. "Lollipop" visualization***

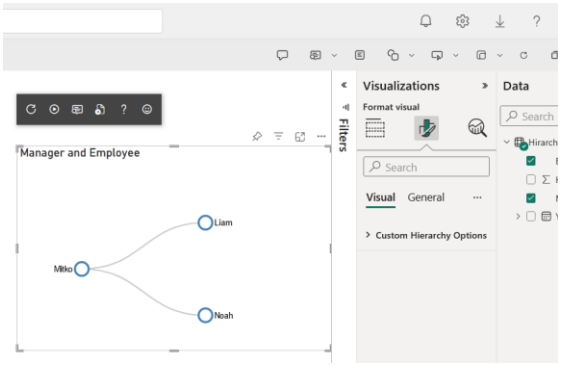


***Figure 13. "Tree" visualization***

## Conclusion

The primary objective of the dissertation work has been achieved: the development of an architecture model for the unified construction of customized extensions for the two considered BI systems. This model encompasses two abstraction patterns – data access and styling. Using this architecture, visualizations "Triangle of Insurance Claims", "Data Label", "Lollipop" and "Tree" have been developed, which are implemented in the two mentioned environments.

## Author reference of contributions

**The results of the dissertation work can be summarized in the following scientific-applied and applied contributions:**

1. Both the necessary dependencies and specifics, as well as the intersection points of the APIs for Power BI and Tableau are derived, with a view to forming a uniform structure;
2. An architecture for unified development of TabWerBI custom visualizations is created, which covers two aspects of abstraction: at the level of data access and basic styling of the new extension;

3. The specific requirements of a specific subject area, i.e. insurance, have been studied, and an industry-recognizable visualization of the development of the quality of an insurance portfolio has been developed;
4. Developed several prototype visualizations with a specific focus using TabWerBI unified custom visualization architecture.

The relationship between each contribution, the task set in the dissertation work, the position of the description, as well as the corresponding included publication, is shown in Table 1.

***Table 1. Tabular distribution of contributions based on assignments, dissertation paragraphs, and scholarly publications***

| Contribution | Contribution Type | Tasks | Paragraph | Publications |
|---|---|---|---|---|
| 1 | scientific and applied | 1, 2 | 1.1., 1.2, 1.3, 2.1., 2.2. | 1, 2, 3, 4 |
| 2 | scientific and applied | 3 | 3.1, 3.2, 3.3 | - |
| 3 | applied | 4 | 4.1 | - |
| 4 | applied | 4 | 4.1., 4.2., 4.3 | - |

## Future work

Future development of the TabWerBI architecture model working with the current APIs of the mentioned tools is possible to cover several main aspects regarding:

- access to the database by offering enrichment of the created collection of data mapping.
- the provided options for formatting the elements of the new additional graphic product.
- the styling of custom infographics, which can be extended with more options.
- providing a way to generate the necessary manifest files containing the meta information specific to each environment.

Although some of the most common BI systems are Microsoft Power BI and Tableau, the specifics of creating custom extensions for other current interactive reporting systems such as QlikSense, Dundas BI, and others could be an object of further research.

## Approbation

**Some of the results obtained in the dissertation work are:**

- **implemented as prototype visualizations in an information system of an insurance company in the Netherlands as follows:**

1. “Insurance Claims Triangle”;
2. “Data Label”.

- **reported at the following international conferences:**

1. **Naneva, V**., K. Stefanova, One Approach for Developing a Custom Extension For Tableau, *International Conference on Technology (IConTech'2022),* 16-19 November, 2022, Antalya, Turkey;
2. **Naneva, V**., K. Stefanova, Implementation of a Custom Lollipop Visual in Power BI, *International Conference on Technology (IConTech'2022),* 16-19 November, 2022, Antalya, Turkey.

- **included in the laboratory exercises of the following disciplines tutored by the author:**

1. "Internet technologies" for the "Software Engineering" specialty, III year, full-time and part-time study;
2. "Information technologies on the Internet" for the specialty "Software technologies and design", IV year, full-time and part-time study.

**Author's participation in scientific projects:**

1. Scientific project MU19-FMI009 "Modeling through mathematics and informatics and their symbiosis with ICT" at the NPD of PU "Paisiy Hilendarski", 2019-2020, participant;
2. National scientific program D01-205 "Information and communication technologies for a single digital market in science, education and security" (ICT in science, education and security – ICTvNOS), participant;
3. Scientific project MU21-FMI-007 "Symbiosis between mathematics and informatics" (SMI in FMI) at the NPD of PU "Paisiy Hilendarski", 2021-2022, participant;
4. Scientific project MU21-FMI-009, "Systematic innovation research in mathematics and informatics" at the NPD of PU "Paisiy Hilendarski", 2021-2022, participant;
5. National project KP-06-N 62/1 at FI at FNI, headed by Prof. S. Hristeva-Kraeva, DSc, title "Mathematical and informational modeling of dynamic processes – new theoretical results, research methods and applications", 2022-2025, participant;
6. Scientific project MUPD23-FMI-009 "Development of ICT through new research and technological solutions" to the NPD of PU "Paisiy Hilendarski", 2023-2024, participant.

## Publications

**Publications in journals and conference proceedings indexed in Scopus, Web of Science and other databases:**

1. **Naneva, V.**, K. Stefanova, Implementation of a custom visual in BI tools, *International Journal of Differential Equations and Applications*, 2021, Vol. 20, No. 2, pp. 225–234, ISSN (Print): 1311-2872; ISSN (Online): 1314-6084; doi: 10.12732/ijdea.v20i2.9, (SJR 2021: 0.214, Q4, Indexed in Scopus)
2. **Naneva V.**, K. Stefanova, A Comparative Analysis of Popular Technologies for Developing Custom Visuals, 46th International Conference Applications of Mathematics in Engineering and Economics, Sozopol, Bulgaria, 7-13 June 2020, *AIP Conference Proceedings*, 2021, 2333, 070009, ISBN: 978-0-7354-4077-7, https://doi.org/10.1063/5.0042201, (SJR 2021: 0.189, Indexed in Scopus & WOS)

**Publications in proceedings of international conferences:**

1. **Naneva, V.**, K. Stefanova, Implementation of a Custom Lollipop Visual in Power BI, *Proceedings of International Conference on Technology (IConTech)*, 2022, pp. 164–171, ISBN: 978-605-72832-5-2, https://www.isres.org/conferences2022_Antalya/ICONTECH2022_Proceedings.pdf
2. **Naneva, V.**, K. Stefanova, One Approach for Developing a Custom Extension for Tableau, *Proceedings of International Conference on Technology (IConTech)*, 2022, pp. 172–179, ISBN: 978-605-72832-5-2, https://www.isres.org/conferences/2022_Antalya/ICONTECH2022_Proceedings.pdf

## Acknowledgments

The results obtained in the dissertation work were partially financed under Project No. KP-06 N 62/1, "Mathematical and informational modeling of dynamic processes - new theoretical results, research methods and applications", supervised by Prof. S. Hristeva-Kraeva, DSc, 2022 - 2025.

I would like to express my sincere gratitude to Prof. Angel Golev, PhD and Prof. Nikolay Pavlov, PhD who, as my academic advisors, guided and encouraged me throughout my work on the dissertation. I am grateful for the opportunity to be their doctoral student. Additionally, I extend my heartfelt thanks to Assoc. Prof. Kremena Stefanova, PhD for our collaborative science work and her continuous support both within and beyond the academic places. Furthermore, I appreciate the valuable guidance and feedback provided by the members of the extended department council of the Software Technologies Department during the preliminary dissertation discussion

## References

Scientific publications in English and Bulgarian:

[1] R. Galici, L. Ordile, M. Marchesi, A. Pinna, R. Tonelli, Applying the ETL Process to Blockchain Data. Prospect and Findings, *Information*, 2020, 11 (4), 204, https://doi.org/10.3390/info11040204

[2] M. Patel, D. Patel, Progressive Growth of ETL Tools: A Literature Review of Past to Equip Future, *Rising Threats in Expert Applications and Solutions. Advances in Intelligent Systems and Computing*, 2021, Vol. 1187, Springer, Singapore, https://doi.org/10.1007/978-981-15-6014-9_45

[3] M. Dobreva, N. Pavlov, A. Rahnev, Integrate Power Bi With Wpf Desktop Applications, Proc. of the Scientific Conference "Innovative ICT in Research and Education: Mathematics, Informatics and Information Technologies", 2019, pp. 65–72, ISBN: 978-619-202-439-0

[4] N. Pavlov, M. Dobreva, Pivot Reporting Tool, Сборник с доклади на научна конференция „Иновационни ИКТ: Изследвания, разработка и приложения в бизнеса и обучението", гр. Хисар, 11–12 ноември 2015 г., стр. 21–30, ISBN: 978-954-8852-56-7

[5] В. Хаджиев, А. Рашидов, Обзор и анализ на методи и модели за структуриране, съхраняване и обработка на данни в интернет, *Автоматика и Информатика 2*, 2019, стр. 27–33, ISSN: 0861-7562

[6] И. Велкова, Обработка и анализ на неструктурирани данни с помощта на изкуствен интелект, *Автоматика и Информатика*, 2022, стр. 27–30, ISSN: 0861-7562

[7] V. Lucie, Factors, Affecting the Adoption of Electronic Data Interchange, *Acta Universitatis Agriculturae et Silviculturae Mendelianae Brunensis*, 2017, 65(6), pp. 2123–2130, ISSN: 12118516

[8] S. Narayanan, A. Marucheck, R. Handfield, Electronic Data Interchange: Research Review and Future Directions, *Decision Science Journal compilation*, Decision Sciences Institute, 2009, 40 (1), pp. 121–163, doi: 10.1111/j.1540-5915.2008.00218.x

[9] N. Elgendy, A. Elragal, Big Data Analytics in Support of the Decision-Making Process, *Procedia Computer Science*, 2016, Vol. 100, pp. 1071–1084

[10] Ajah, H. Nweke, Big Data and Business Analytics: Trends, Platforms, Success Factors and Applications, *Big Data Cogn. Comput*., 2019, 3, 32, https://doi.org/10.3390/bdcc3020032

[11] J. Bharadiya, A Comparative Study of Business Intelligence and Artificial Intelligence with Big Data Analytics, *American Journal of Artificial Intelligence*, 2023, Vol. 7, No. 1, pp. 24–30. doi: 10.11648/j.ajai.20230701.14

[12] J. Bharadiya, Machine Learning and AI in Business Intelligence: Trendsand Opportunities, *International Journal of Computer (IJC)*, 2023, pp. 123–134, ISSN: 2307-4523 (Print & Online)

[13] S. Fahad, A. Yahya, Big Data Visualization: Allotting by R and Python with GUI Tools, 2018 *International Conference on Smart Computing and Electronic Enterprise (ICSCEE)*, Shah Alam, 2018, pp. 1–8, https://doi.org/10.1109/ICSCEE.2018.8538413

[14] M. Muntean, D. Dănăiaţă, L. Hurbean, C. Jude, A Business Intelligence & Analytics Framework for Clean and Affordable Energy Data Analysis, *Sustainability*, 2021, 13, 638, https://doi.org/10.3390/su13020638

[15] C. Smatt, R. Pratt, T. Abegaz, A. Dobbs, Towards effective customer data visualization: data-driven documents (D3.js) vs. Google charts, *A Journal Of International Academy Of Business Disciplines*, 2020, pp. 207–222, ISSN: 2334-0169

[16] S. Prakash, M. Suhal, M. Mithesh, R. Santhiya, The Power of D3.js – Data Visualization, *International Research Journal of Engineering and Technology (IRJET)*, 2020, Vol. 07, Issue 07, pp. 5750–5755, e-ISSN: 2395-0056

[17] Bansal, A. Upadhyay, Microsoft Power BI, *International Journal of Soft Computing and Engineering (IJSCE)*, July 2017, Vol. 7, Issue 3, pp. 14–20, ISSN: 2231-2307

[18] H. Mei, H. Guan, Ch. Xin, X. Wen, W. Chen, DataV: Data Visualization on large high-resolution displays, *Visual Informatics*, 2020, Vol. 4, Issue 3, pp. 12–23, ISSN: 2468-502X, https://doi.org/10.1016/j.visinf.2020.07.001

[19] S. Karabtsev, R. Kotov, I. Davzit, E. Gurov, Building data marts to analyze university faculty activities using Power BI, *E3S Web of Conferences 419*, 2023, 02014, WFCES 2023, https://doi.org/10.1051/e3sconf/202341902014

[20] Z. Sun, L. Sun, K. Strang, Big Data Analytics Services for Enhancing Business Intelligence, *Journal of Computer Information Systems*, 2016, pp. 162–169, doi: 10.1080/08874417.2016.1220239

[21] V. Naneva, K. Stefanova, Implementation of a custom visual in BI tools, *International Journal of Differential Equations and Applications*, 2021, Vol. 20, No. 2, pp. 225–234, ISSN (Print): 1311-2872, ISSN (Online): 1314-6084, doi: 10.12732/ijdea.v20i2.9, (SJR 2021: 0.214, Q4, Indexed in Scopus)

[22] V. Naneva, K. Stefanova, A Comparative Analysis of Popular Technologies for Developing Custom Visuals, 46th International Conference Applications of Mathematics in Engineering and Economics, Sozopol, Bulgaria, 7-13 June 2020, *AIP Conference Proceedings*, 2021, 2333, 070009, ISBN: 978-0-7354-4077-7, https://doi.org/10.1063/5.0042201, (SJR 2021: 0.189, Indexed in Scopus & WOS)

[23] V. Naneva, K. Stefanova, Integration of a Custom Visual in a Web Application, 47th International Conference Applications of Mathematics in Engineering and Economics, Sozopol, Bulgaria, 7-13 June 2021, *AIP Conference Proceedings*, 2022, 2505, 060010, ISBN: 978-0-7354-4396-9, https://doi.org/10.1063/5.0102163, (SJR 2022: 0.164, Indexed in Scopus & WOS)

[24] V. Naneva, K. Stefanova, Implementation of a Custom Lollipop Visual in Power BI, *Proc. of International Conference on Technology (IConTech)*, 2022, pp. 164–171, ISBN: 978-605-72832-5-2, https://www.isres.org/conferences2022_Antalya/ICONTECH2022_Proceedings.pdf

[25] V. Naneva, K. Stefanova, One Approach for Developing a Custom Extension for Tableau, *Proc. of International Conference on Technology (IConTech)*, 2022, pp. 172–179, ISBN: 978-605-72832-5-2, https://www.isres.org/conferences/2022_Antalya/ICONTECH2022_Proceedings.pdf

[26] S. Batt, T. Grealis, O. Harmon, P. Tomolonis, Learning Tableau: A data visualization tool, *The Journal of Economic Education*, 2020, 51: 3-4, pp. 317–328, ISSN: 317-328, doi: 10.1080/00220485.2020.1804503

Books:

[27] F. Zammetti, *Modern Full-Stack Development*, USA: Press Media, LLC is a California, 2020, ISBN: 9781484257371

[28] D. Vanderkam, *Effective TypeScript: 62 Specific Ways to Improve Your TypeScript*, 1st Edition, O'Reilly Media Inc., 2019, ISBN: 9781492053712

[29] D. Choi, *Full-Stack React, TypeScript, and Node*, Packt Publishing, 2020, ISBN: 9781839219931

[30] T. Runkler, *Data Analytics*, Springer Fachmedien Wiesbaden, 2012, ISBN: 978-3-8348-2588-9

[31] P. Zikopoulos, C. Eaton, D. de Roos, T. Deutch, G. Lapis, *Understanding Big Data, Analytics for Enterprise Class Hadoop and Streaming Data*, New York: McGraw-Hill, 2012, ISBN: 978-0-07-179053-6

[32] B. Devlin, Business Intelligence: Insight and Innovation beyond Analytics and Big Data, Technics Publications, 2013, ISBN: 978-163-462-032-1

[33] F. Provost, T. Fawcett, *Data Science for Business*, O'Reilly Media Inc., 2013, ISBN: 978-144-936-132-7

[34] L. Favero, P. Belfiore, *Data Science for Business and Decision Making*, Academic Press, 2019, ISBN: 978-012-811-217-5

[35] C. Sinha, Mastering Power BI Build Business Intelligence Applications Powered with DAX Calculations, Insightful Visualizations, Advanced BI Techniques, and Loads of Data Sources, BPB Publications, 2021, ISBN 978-939-103-072-8

[36] Haldorai, A. Ramu, S. Khan, *Business Intelligence for Enterprise Internet of Things*, Springer International Publishing, 2020, ISBN: 978-303-044-407-5

[37] G. Shmueli, N. Patel, P. Bruce, Data Mining for Busiess Intelligence: Concepts, Techniques, and Applications in Microsoft Office Excel with XLMiner, John Wiley and Sons, 2011, ISBN: 111-812-604-1

[38] M. Meier, D. Baldwin, K. Strachnyi, Mastering Tableau 2021: Implement advanced business intelligence techniques and analytics with Tableau, Packt Publishing Ltd., 2021, ISBN: 978-180-056-074-1

[39] R. Skyrius, *Business Intelligence: A Comprehensive Approach to Information Needs, Technologies and Culture Progress in IS, Springer International Publishing*, 2022, ISBN: 978-303-067-034-4

[40] R. Sherman, Business Intelligence Guidebook: From Data Integration to Analytics, Newnes, 2014, ISBN: 978-012-411-528-6

[41] Business Science Reference, Business Intelligence: Concepts, Methodologies, Tools, and Applications: Concepts, Methodologies, Tools, and Applications, IGI Global, 2015, ISBN: 978-146-669-563-4

[42] R. Sleeper, *Practical Tableau*, O'Reilly Media Inc., 2018, ISBN: 978-149-197-728-6

[43] R. Cherny, Programming TypeScript: Making Your JavaScript Applications Scale, O'Reilly Media Inc., 2019, ISBN: 978-149-203-762-0

[44] D. Vanderkam, *Effective TypeScript: 62 Specific Ways to Improve Your TypeScript*, O'Reilly Media Inc., 2019, ISBN: 978-149-205-369-9

[45] H. Rocha, Learn D3.js: Create interactive data-driven visualizations for the web with the D3.js library, Packt Publishing Ltd, 2019, ISBN: 978-183-864-960-9

[46] D. Knight, M. Pearson, B. Schacht, E. Ostrowsky, Microsoft Power BI Quick Start Guide: Bring your data to life through data modeling, visualization, digital storytelling, and more, Packt Publishing Ltd., 2020, ISBN: 978-180-056-994-2

[47] E. Brown, Learning JavaScript: JavaScript Essentials for Modern Application Development, O'Reilly Media Inc., 2016, ISBN: 978-149-191-493-9
[48] D. Sarka, J. Rihar, K. Vončina, Advanced Analytics with Power BI and Excel: Learn powerful visualization and data analysis techniques using Microsoft BI tools along with Python and R (English Edition), Orange Education Pvt Ltd., 2023, ISBN: 978-9391246709
[49] В. Спасова, *Интелигентни системи за вземане на решения в условията на индустрия 4.0*, Варненски свободен университет „Черноризец Храбър" – Университетско издателство, 2019, ISBN: 978-954-715-675-3

Internet sources:

[50] Business Intelligence Definition, https://www.heavy.ai/technical-glossary/business-intelligence (last accessed on 30.08.2023)
[51] Ключови показатели за ефективност, https://www.novavizia.com/klyuchovi-pokazateli-za-efektivnost/ (last accessed on 18.08.2023)
[52] Mastering the Art of Creating Effective and Intuitive Data Views: Unveiling the Power of Extra Mile Efforts, https://www.linkedin.com/pulse/mastering-art-creating-effective-intuitive-data-views-nikher-verma/ (last accessed on 18.08.2023)
[53] Tableau Extensions API Basics, https://tableau.github.io/extensions-api/docs/trex_api_about.html (last accessed on 18.08.2023 г.)
[54] Get Started with Dashboard Extensions, https://tableau.github.io/extensions-api/docs/trex_ getstarted.html(last accessed on 18.08.2023.)
[55] What Happens When you Reload an Extension, https://tableau.github.io/extensions-api/docs/trex_reload.html (last accessed on 18.08.2023)
[56] Get Data from the Dashboard, https://tableau.github.io/extensions-api/docs/trex_getdata.html (last accessed on 18.08.2023)
[57] GetSummaryDataAsync() method, https://tableau.github.io/extensions-api/docs/interfaces/ work sheet. html# getsummarydataasync (last accessed on 18.08.2023)
[58] HTTPS and Tableau Extensions, https://tableau.github.io/extensions-api/docs/trex_security.html (last accessed on 18.08.2023)
[59] Get Data from the Dashboard, https://tableau.github.io/extensions-api/docs/trex_getdata.html#get-the-worksheet-object (last accessed on 18.08.2023)
[60] Events and Event Handling, https://tableau.github.io/extensions-api/docs/trex_events.html (last accessed on 18.08.2023)
[61] Power BI visuals system integration, https://learn.microsoft.com/en-us/power-bi/developer/visuals/power-bi-visuals-concept (last accessed on 18.08.2023)
[62] Power BI visual project structure, https://learn.microsoft.com/en-us/power-bi/developer/visuals/ visual-project-structure (last accessed on 18.08.2023)

[63] Objects and properties of Power BI visuals, https://learn.microsoft.com/en-us/power-bi/developer/visuals/objects-properties?tabs=getFormattingModel (last accessed on 18.08.2023)
[64] Import a Power BI visual from AppSource into your workspace, https://learn.microsoft.com/en-us/power-bi/developer/visuals/import-visual (last accessed on 18.08.2023)
[65] Guidelines for publishing Power BI custom visuals, https://learn.microsoft.com/en-us/power-bi/developer/visuals/guidelines-powerbi-visuals (last accessed on 18.08.2023)
[66] Test a Power BI custom visual before submitting it for publication, https://learn.microsoft.com/en-us/power-bi/developer/visuals/submission-testing (last accessed on 18.08.2023)
[67] Create Power BI visuals with Python, https://learn.microsoft.com/en-us/power-bi/connect-data/desktop-python-visuals (last accessed on 18.08.2023)
[68] Understand data view mapping in Power BI visuals, https://learn.microsoft.com/en-us/power-bi/developer/visuals/dataview-mappings (last accessed on 18.08.2023)
[69] Library QuickStart, https://basarat.gitbook.io/typescript/library, (last accessed on 18.08.2023)
[70] TypeScript's Type System, https://basarat.gitbook.io/typescript/type-system (last accessed on 18.08.2023)
[71] How to develop a Typescript Library, https://medium.com/@cameronadams1225/how-to-develop-a-typescript-library-ade8d329636 (last accessed on 18.08.2023)
[72] Creating Awesome TypeScript NPM Packages, https://levelup.gitconnected.com/creating-awesome-typescript-npm-packages-5d97ac342361 (last accessed on 03.09.2023)
[73] Step by step: Building and publishing an NPM Typescript package, https://itnext.io/step-by-step-building-and-publishing-an-npm-typescript-package-44fe7164964c, (last accessed on 03.09.2023)
[74] Why JavaScript everywhere? And what does it mean for software development?, https://www.linkedin.com/pulse/why-javascript-everywhere-what-does-mean-software-gulshan-sharma/ (last accessed on 03.09.2023)
[75] Typescript Webpack, https://webpack.js.org/guides/typescript/ (last accessed on 03.09.2023)
[76] Using webpack with TypeScript, https://blog.logrocket.com/using-webpack-typescript/ (last accessed on 03.09.2023)
[77] JavaScript Standard Style, https://standardjs.com/ (last accessed on 03.09.2023)
[78] Interface Worksheet, https://tableau.github.io/extensions-api/docs/interfaces/worksheet.html (last accessed on 03.09.2023)
[79] getSelectedMarksAsync, https://tableau.github.io/extensions-api/docs/interfaces/worksheet.html #getselectedmarksasync (last accessed on 03.09.2023)
[80] addEventListener, https://tableau.github.io/extensions-api/docs/interfaces/worksheet.html# addeventlistener (last accessed on 03.09.2023)

[81] How To Create A TypeScript Library, https://marketsplash.com/tutorials/typescript/how-to-create-a-typescript-library/ (last accessed on 03.09.2023)
[82] WHAT ARE THE TYPES OF GUIDS?, https://www.webopedia.com/definitions/guid/ (last accessed on 03.09.2023)
[83] Color picker tool, https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_colors/Color_picker_tool (last accessed on 03.09.2023)
[84] Loss Triangles: How to Find Ultimate Loss for Financial Statements, https://www.waterstreet company .com/ loss-triangle -analytics/, https://www.waterstreetcompany.com/loss-triangle-analytics/ (last accessed on 03.09.2023)
[85] Insurance Claims triangle – A jab at SQLScripting, https://blogs.sap.com/2015/06/04/insurance-claims-triangle-a-jab-at-sqlscripting/ (last accessed on 03.09.2023)
[86] Telecommunications Analytics, https://www.tableau.com/data-insights/dashboard-showcase/telecommunications-analytics (last accessed on 29.09.2023 )