University of West Bohemia
Faculty of Applied Sciences
Department of Computer Science and Engineering

Bachelor's thesis

Visualization of computation power for retail business processes

Pilsen, 2021 Jan Lešek

Místo této strany bude zadání práce.

Declaration

I hereby declare that this bachelor's thesis is completely my own work and that I used only the cited sources.

Pilsen, 3rd May 2021

Jan Lešek

Abstract

The bachelor thesis is focused on designing and developing a new tool named Cloud sizing tool for company Eurosoftware s.r.o.. The tool is going to substitute current time consuming variant creation, thus making the whole process more efficient. The tool is intended for company's consultants to choose a specific variant of company's product that suits the customer's needs. The variant is based on necessary speed of data management, type and size of chosen database, price of the whole product and other requirements. The consultants will enter the data that were previously discussed with a customer. During the data input, the tool will calculate values, that will guide the consultant to choose an appropriate solution for the customer.

Abstrakt

Tato bakalářská práce je zaměřena na design a vývoj nového nástroje Cloud sizing tool pro firmu Eurosoftware s.r.o.. Nástroj nahradí aktuální časově náročný proces výběru varianty produktu. Udělá tak celý proces efektivnější. Nástroj je určen pro konzultanty, aby vybrali konkrétní variantu produktu, která bude vyhovovat požadavkům zákazníka. Varianta bude založena na požadované rychlosti zpracování dat, typu a velikosti dané databáze, ceny celého produktu a dalších požadavků. Konzultanti uvedou data, která dříve prodiskutovali se zákazníky. V průběhu zadávání těchto dat nástroj bude provádět výpočet hodnot, které povedou konzultanta k výběru vhodného řešení pro zákazníka.

List of Acronyms

API Application Programming Interface. 13, 22, 48, 49, 57

CLI Command Line Interface. 34

CSS Cascading Style Sheets. 8, 22–27, 57

DOM Document Object Model. 23

ECON Enterprise Controller. 7, 13, 19, 20, 31

EFT Electronic Funds Transfer. 13

FXML FX Markup Language. 7, 22, 23

GUI Graphic User Interface. 7, 8, 10, 22, 23, 25, 26, 34, 56, 57

HA High Availability. 14, 20

HTML Hypertext Markup Language. 7, 22, 24–28, 35, 36, 38–40, 45, 49, 56, 57

IPC Inter-Process Communication. 13, 15, 24, 25, 35

JS JavaScript. 7, 22–27, 37, 38, 45, 48, 55–57

JSON JavaScript Object Notation. 28, 38, 39, 44–46, 57

K8s Kubernetes. 7, 13, 15, 16, 20

MD Master Data. 13

MVC Model View Controller. 23, 36, 56, 57

PHP Hypertext Preprocessor. 25

POS Point of Sale. 7, 11–13, 17–21, 28, 29, 31, 39, 40, 50

RTF rich text format. 22

SAP Systems-Applications-Products in data processing. 18, 31

SAP PI SAP NetWeaver Process Integration. 13

SDC Store Device Controller. 7, 13, 19, 20, 31

TO Transport Object. 8, 17–19, 40, 50–52

TS TypeScript. 7, 24, 26, 27, 56, 57

TX Transaction. 8, 13, 19, 50–52

UI User Interface. 8, 28, 35, 37, 38, 44, 47, 55, 56

VAT Value Added Tax. 19

XML Extensible Markup Language. 13, 19

Contents

1 Introduction

Company Eurosoftware s.r.o. (subsidiary of GK Software SE) provides to its customers their software bundle that customers want to use in theirs retail business. Mentioned software is a property of GK Software SE. However, every customer has different requirements on the software based mainly on the size and methods of managements of their business. Company needs to properly size their product for each customer individually. Product sizing means choosing correct quantity and type of particular system's components. The system is sized in a way to satisfy customer's demands.

The process of choosing the actual components of the product variant based on said requirements is realised by a group of company's consultants. The frequent customer-employee communication is crucial since there is no other guide for customer to help him choose the solution that suits his needs. Also, direct choosing done by the customer himself would be impractical due to the large assortment of available components. In addition, company can assume customer does not necessarily know about the used technology.

The goal is to provide to the consultants a convenient tool which will help and guide them through the whole variant creation process. This will substitute the currently used excel sheet used by consultants in the process of sizing the product.

The tool must be cross-platform to be able to support multiple operating systems for computers (mainly Linux and Windows). The Graphic User Interface should be simple, easy to use and have a plenty of available clickable hints if needed to provide the user with user friendly experience. In order for the user to be able to express all the crucial requirements, the tool has to contain a sufficient amount of appropriately selected elements (such as input, combobox or slider) for each of the previously said requirements.

The thesis will describe a detailed designing and developing process of the tool as well as analysing appropriate options of available technologies and reasons for the final choices. The thesis requires a modular basis of the aforementioned tool, rather than a full release. The company will take over the development of the tool after submission of the thesis.

2 Analysis of variant creation process

The thesis is a request of Eurosoftware s.r.o. The company is a subsidiary of GK Software SE. The tool is used by consultants to properly size the product for each individual customer. Acquired information about the customer's requirements are manually processed through excel sheet to convert the requirements to correct product variant. The current process is not convenient and efficient enough for today's standards. The excel maintenance is time consuming, calculations extensibility and debugging is difficult to execute and proper testing is not possible. Cloud sizing tool is meant to largely supplement this process.

2.1 About company

GK Software SE (GK stands for last names of founders Rainer Gläß and Stephan Kronmüller) develops products and tools used worldwide. It covers the whole process of shopping, including managing transactions in database and the interface used in Point of Sale.

Part of their solution is developing software products using cloud technologies and artificial intelligence. The company's goal is to be the leading provider of the retail solution.

GK Software SE is active in research of new technologies. They intend to recognize the future of software developement. Nowadays they are engaged in application development with the use of virtual reality.

The company's used leading technologies include Java, SAP HANA, Oracle, MS SQL, Microsoft Azure, Kubernetes, SQL Server, Tomcat, Machine learning, Cloud native techniques, Speech recognition.[1]

GK/Retail Omni-Channel is the software product that GK Software SE sells. It is a complex retail solution that provides company's customers with Point of Sale software used in customer's retail businesses. Some of its components are described down bellow in this section. Knowledge of their purposes and functionality is essential for development of the thesis tool. It is divided into two major variants:

On premise represent the older variant. Customer provides himself the hardware for the POS units and servers on which the software will be installed and run.

cloud4retail is the solution that Cloud sizing tool is intended for. The main difference opposed to On premise variant is that the customer does not need to worry about the enterprise part (servers) at all since it is managed in company's cloud and provided by them remotely. The customer then only needs to provide Point of Sale units (cash registers) and internet connection. One prerequisite for cloud4retail solution is that the Point of Sale software has to be containerized to be properly scalable.

2.2 GK/Retail applications and components introduction

Both on premise and cloud4retail solution variants share the same components structure since the main difference is in the execution of the solution. Big picture of the components can be seen at 2.1.

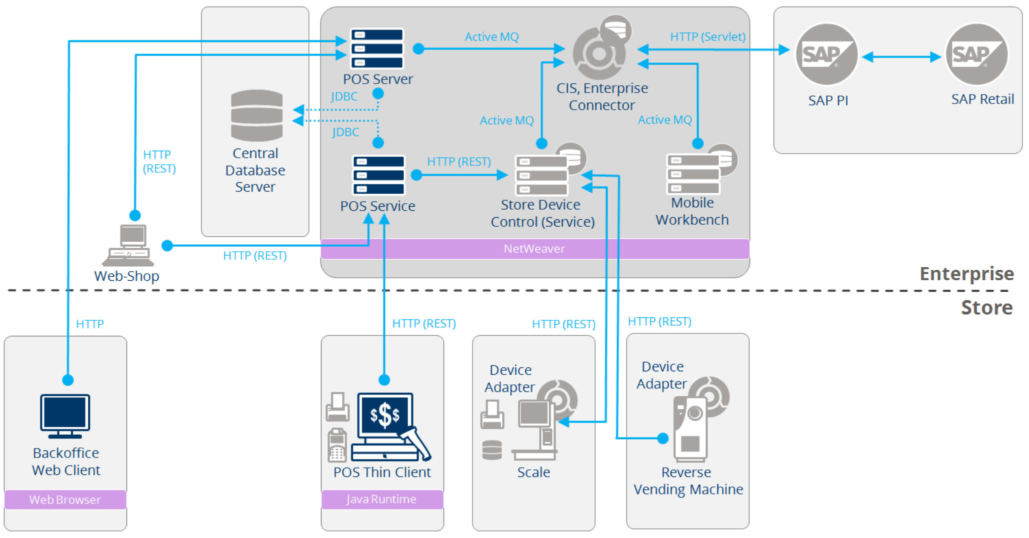


Figure 2.1: Diagram representing Big picture of the GK Business Retail suite in ThinPOS variant.[2]

2.2.1 Point of Sale

Mostly referring to GK/Retail OmniPOS which is basically the end part of GK/Retail Omni-Channel used by cashiers to serve customers. It is used as a solution for selling and returning items, allowing various types of payments, redeeming gift cards, handling discounts etc. Depending on the store policies, it integrates different types of hardware like bar code scanner, terminals (Electronic Funds Transfer) and cash drawers. POS can be distributed mainly in three forms: ThinPOS (databases are remotely accessed via services in central), FatPOS (databases are embedded within) or SmartPOS (combination). Database is divided in two separate units for Master Data (MD) and Transaction (TX).[3]

2.2.2 Store Device Controller

Purpose of Store Device Controller (SDC) is processing of the master data and distributing it to other applications in the GK/Retail OmniPOS. The distribution can be done with data replication or by services like SDC API.[4]

2.2.3 Enterprise Controller

Enterprise Controller (ECON) is the entrance for the master data to the GK/Retail OmniPOS. It manages transportation of data and events through import and export with SAP NetWeaver Process Integration (SAP PI) (other systems possible also). The data are converted in ECON from external formats to internal Extensible Markup Language (XML) and vice versa.[5]

2.2.4 POS Server

It is responsible for storing and processing transactions. Managing transactions includes loading, saving, searching or replication of each transaction. Through POS Server various reporting and accounting functions can be accessed.[6]

2.3 cloud4retail solution

GK Software SE uses containerization of GK/Retail components, usually through Docker. The main benefits, that come from this, are the applications abilities to use the resources of hardware that they run on while simultaneously being isolated from other processes on that same machine. However, GK/Retail components need IPC among them. This is provided by orchestration through Kubernetes (K8s) technology. K8s operates mainly with clusters, nodes and pods. This section is dedicated for proper description of GK Software's used cloud architectures (diagram can be seen at 2.2).

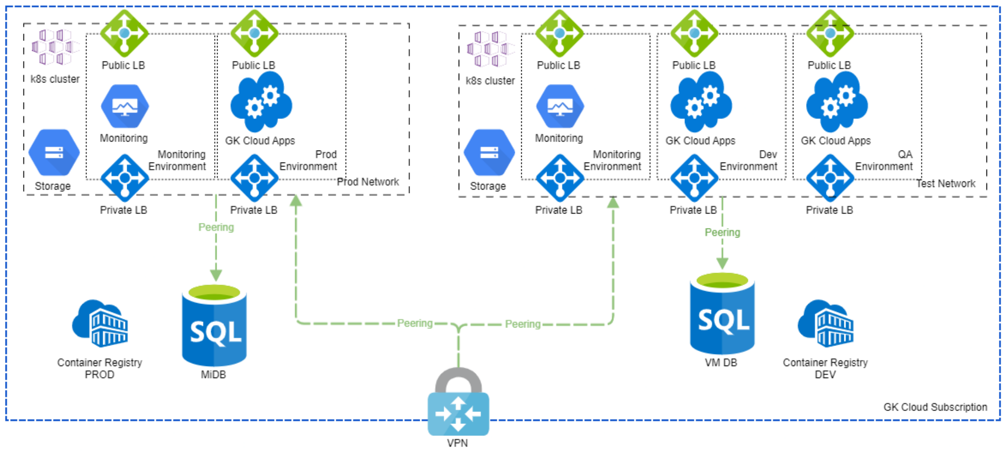


Figure 2.2: Diagram representing GK Software SE cloud4retail solution for an idea of the whole system [7]

2.3.1 Performance scaling principles

Scaling

GK Software's cloud solution implements scalability of components to ensure process efficiency. Scalability is an ability of a system to adapt to changing performance needs. The system adds supportive resources to itself while in runtime if needed.[8] In other words, no system restart is needed and the system has a variety of resources to use. Scaling can be organised into these two types:

Vertical scaling means powering up the current components (e.g. RAM, CPU). It is also referred to as *scaling up*.

Horizontal scaling represent process of supporting the current machines by adding more of them together. It is also known as *scaling out*[9].

High Availability

High Availability (HA) is describing a system behaviour that ensures its components are able to be continuously operational for a long time span. It can be measured relatively either to *"100% operational"* or *"never failing"*.[10] GK/Retail components are designed with HA mode.

Non-functional testing

Non-functional testing is characterised as procedures that test application or system for its non-functional requirements (the way a system operates, at which speed and performance, rather than specific behaviours and data validations).[11]

Company periodically runs non-functional tests to see the performance of each components. This testing is crucial since GK/Retail component's reliability and scalability are desired to be working correctly and efficiently as much as possible. It is a process used to ensure sufficient performance and to obtain actual sizing parameters.

2.3.2 Containerization (Docker)

In order to ensure deployment consistency, correct network exposure and needed safety, each GK/Retail component runs in a container. The dominant technology used for containerization in GK Software SE is Docker.

Docker is an open platform used in development and deployment of applications. The main idea is to separate application from the infrastructure where it runs on. It offers methodologies for shipping, deployment and testing of the code.[12]

2.3.3 Orchestration (Kubernetes)

To provide Inter-Process Communication among used components, Kubernetes (K8s) is used for process orchestration (2.3). It takes care of automating deployment, scaling, and overall management of GK/Retail applications.

Kubernetes is an open-source tool used for management of containerized applications and systems.[13]

Cluster Cluster is a deployment unit of K8s.

Nodes Every cluster has one Node at minimum. It represents a set of worker units that run containerized systems.[14]

Pods Pods represent the components of the application workload. To ensure decent high-availability and fault tolerance, multiple pods of an application are running at all times. Number of pods is derived from processing speed requirement of the customer, therefore it is considered as one of the most important aspects of the sizing.

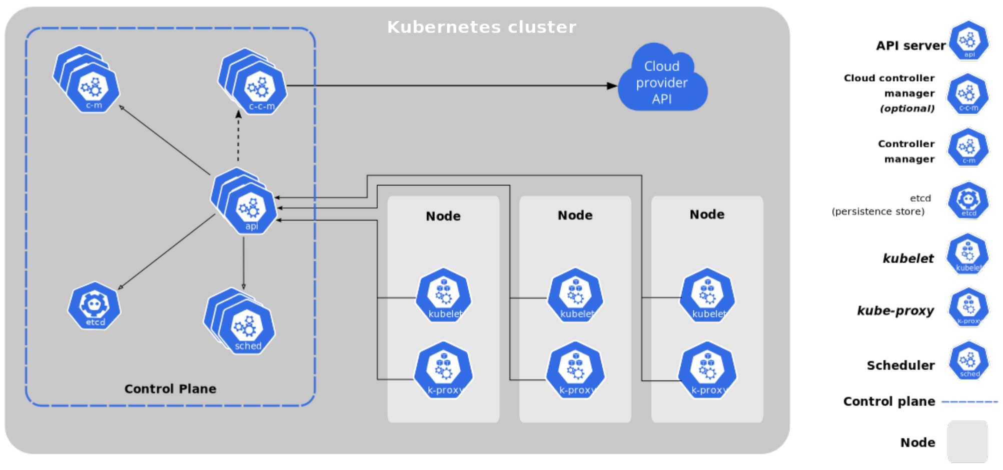


Figure 2.3: Diagram representing Kubernetes components.[14]

2.4 Retail process calculation

GK Software SE follows a strict process for creating optimal solution for a customer. The process derives the sufficient number and type of each GK/Retail component. For example, it gives a desired number of high-availability pods for specific GK/Retail OmniPOS component. Partial automatization of said process is a part of the idea for this thesis.

The core of the whole calculation process are the requirements expressed by a customer. The calculations, of course, run in a different manner for a small town retail store and for a large corporate franchise.

2.4.1 Master data

Master data is a data container that holds steady set of unique identifiers and properties about the core entities of the enterprise, such as customers, clients, suppliers or operator credentials.[15]

Master data in GK/Retail OmniPOS contain information such as:

Item and its GTIN code, identifier, variant (colour or size), category, price, discount etc.

Registered customers including their full name, date of birth, phone number, sex, age or product interests.

Active gift cards with its date of use, id number and the name of the customer who bought it.

POS includes more types of store clients (or edge components) such as Mobile POS, Self Scanning and OMS Client.

Stores represent buildings or offices where POS devices are active and used in open hours. Depending on diversity of store types of the business itself, ratio POS per store can be considered as a meaningful attribute in calculations.

Items are stored in master data in database of each POS device. Usually taken as products at the store.

Daily changes occur to every item at the store periodically. Can happen multiple times per day. For example: in the morning before store opening, full data pump (described in 2.4.3) is processed to every POS unit at the store. Throughout the day occurs happy hour, meaning prices of every item from a certain category get lowered to 50% of its previous price. That means before the happy hour starts, one TO (described in 2.4.3) has to be distributed updating the product prices to 50% and at the end of the event another TO has to be sent to revert the prices back to normal.

Opening days (per week or per year) value works simply as a multiplier for nearly all of the aforementioned values. The main use of this information is to derive it to the process per hour (or minute) ratio, which can presume to a certain extent the performance peak. Some businesses close on weekends or on any public holiday. This can dramatically change the final product.

2.4.2 Offline availability

The whole GK/Retail OmniPOS is designed to be capable of operating in offline environment. That means POS does not need to look up every item scanned in online database since it has previously replicated the master data in its own local database. GK/Retail OmniPOS is designed that way so it can also work properly in areas with unstable internet connection. However, this also means that master data need to be updated regularly, even during open hours to provide up-to-date information about item stocks, discounts and others.

POS count:	10 pieces per store
Store count:	12
POS count total:	10 * 12 = 120
Average database item size:	10 kB
Number of items in the database:	7'500 pieces
Item's database size:	10 kB * 7'500 = 75 MB
Registered customer's average size:	3 kB
Registered customers count:	640 people
Customer's database size:	3 kB * 640 = 1.920 MB
Database size:	75 MB + 1.920 MB = 76.920 MB
Data-pump replication size:	76.920 MB * 120 = 9.230 GB

Figure 2.4: Table showing attributes of a retail business.

2.4.3 Master data processing

Master data processing is fundamentally a process that distributes master data from external source (mostly SAP) to any POS across the whole retail business. Distributing the master data is called replication. Replication can be done in two use-cases:

Full data pump means sending the whole master data to the target POS. It is used when integrating a new POS machine, resetting the data or at opening a brand new store.

Transport Object (TO) represents a fraction of the whole master data in a manner of size. TO is preferred over full data pump in situations like the following example. During open hours one item price changes, therefore every other POS machine needs to be informed about the new status of the item's new price. In that way, sending a whole master data only for this one information would be extremely inefficient.

Size of all the replications in one day can vary between megabytes and gigabytes per POS and most of the customers have multiple POS units at each store.

Size of all the replications for one retail business can acquire various sizes (in bigger businesses up to tens of gigabytes) in one day, depending on the size of the business. For illustration, a retail business has attributes shown in table 2.4 on page 18.

To fully replicate all POS devices in the retail, 9.230 GB is needed to do so in the mentioned example. Calculation process has to consider customer's requirements like how much time it should take to do a full data-pump. Every single component of GK/Retail OmniPOS has to be configured based on these requirements while keeping the lowest possible price.

The previous example is over-simplified for illustrative purposes. The master data takes in consideration only items and registered customers. In real-life scenario, master data contains 10 to 100 different attributes in couple of variants. Also, the example takes in consideration only one data-pump (that usually takes place in the morning) and ignores the TO transfers through the day. The variety of the scenarios depends only on the different customer's data volumes and daily changes.

The process starts by transferring the data from an external source in XML format to ECON. ECON validates the data structure and formatting. If the data are valid, ECON passes the data to SDC. ECON and SDC should be at the same performance level. If they are not, the faster component must always wait for the slower one. SDC validates the data in semantics level. If it is valid, SDC distributes the data to any of its POS devices on request.

2.4.4 Transaction processing

Transaction (TX) carry information about monetary interaction with a customer. In cases of large businesses or retails where plenty of transactions are to be expected daily, more useful value is ratio transactions per day. Transactions share similar behaviour to database transactions in a manner that they lock the POS local replicated master data with the start of the transaction (e.g.: scanning the first item) and release the lock after payment is finished or the transaction gets cancelled. In any time between these two events, prices in the whole local database cannot change. This behaviour is demanded by the law for customer protection.

The process shares the same principle as the master data processing but in reverse and with some alterations. Transactions are sent to the SDC, then to ECON, where are finally transferred to the external source. Transactions need to contain different data than in master data. Most notably:

Value Added Tax (VAT) is demanded in retail business by law.

Price paid for the whole transaction.

Payment type varies in different currencies supported by the store. Transaction can be also paid by credit card, debit card, coupon, gift card, invoice and other methods. Every one of them has to be noted inside the transaction.

Items purchased and all of their relevant attributes.

Customer info. If registered, then the account's identifier and eventually applied discounts or customer points.

Transactions need to have some retention. Retention is defined as amount of time each transaction is stored in the database since its completion before it is deleted. Usually it is from 30 to 90 days, but it can also cover a warranty couple of years long. Retention is used as a way to return products to the store too. Each transaction needs to be unique with its new previously unused identifier. In a situation when a customer buys a product and after a few minutes returns it, two separate transactions with unique identifiers are created and stored for the whole retention period. Every one of these aspects need to be considered when choosing the right components for the GK/Retail OmniPOS system, for them to be able to handle this amount of data continuously.

2.5 Company's Kubernetes accessories

2.5.1 Pods

In the company over 30 applications are used during the product's runtime (including POS server, ECON etc.). Each one of them is represented in cloud4retail solution as as an unique K8s pod.

The thesis will focus only on POS server, SDC and ECON pods. It would be too much time consuming and excessively difficult to cover the whole enterprise in a span of one bachelor thesis due to the system's complexity. The other pods will be reffered to as *others* and their data for calculations will be fetched from the tool's static mock data, which will be created from the Excel draft.

2.5.2 Nodes

Depending on the pods count, number and type of nodes will be chosen. Two types of nodes are available.

Idle node is utilised for standard course without any major activity peaks.

High Availability node is intended for sales peaks and sudden high performance loads.

2.5.3 Clusters

Cluster is a group of nodes provided to the customer. Their sum is divided into two types.

MIN is based solely on the numbers of POS units, transactions per day and other attributes of this kind. It is the minimum number of nodes (in a certain ratio of types) that is required to serve the system in its standard course.

MAX is counted similarly to MIN but with the inclusion of desired import and export times in the calculations. This contains the highest number of nodes that will be available at any time for performance peaks.

3 Analysis of available technologies

The key feature is to make the tool cross-platform. According to this, languages, frameworks and other technologies that support this feature are preferred.

3.1 GUI Elements and design

3.1.1 Hypertext Markup Language

Hypertext Markup Language (HTML) is a computer language primarily used for designing web pages. It is the core language of structure of web content. Closely related languages take care of graphics and appearance (CSS) and functionality (JS). Hypertext represents links located on the pages that refer to other pages and lead its users there. One of the most prominent characteristics of HTML is its usage of markups to annotate its elements (text, videos, images, sections, canvases etc.) which are distinctively represented by tags surrounded by $<$ and $>$ symbols.[16]

3.1.2 Java Swing

Java Swing is a GUI framework included in Java language. It provides rich set of widgets like sliders, colour choosers, tables, text areas, buttons, trees, check boxes etc. Every Swing component is graphically and functionally customizable in a simple process. It also allows to display rich text format (RTF).[17]

3.1.3 JavaFX and FX Markup Language

JavaFX is an open source framework extension for Java language. It is intended for creating GUI client applications. Its library is available as a public API. Since Java 9, it is not a built-in feature of the language anymore, so it has to be included separately (e.g. through maven or ant build automation tools). JavaFX provides user with a scene graph consisted of various hierarchically arranged visual elements (nodes), such as input, check or combo boxes, frames, sliders, buttons, text areas and panes.[18]

FXML is an XML-based language used with JavaFX to create GUI elements. Its main benefit is that it separates visual part from the application logic, therefore it is used to preserve Model View Controller (MVC) architecture. The FXML files serve as the view, creating the visual part, whereas associated Java files represent the controller part, taking care of the application logic and calculations.

3.2 Functionality

3.2.1 Java

Java is an objective programming language invented by Sun Microsystems in 1995. It provides C-like user friendly syntax, strong type checking and a rich set of supported platforms. One of the most advantages that Java provides is platform independence on the source and binary levels. Binary files with extension *class* can be transported among various platforms and be run without the need of having the source and compiling the code again.[19]

3.2.2 JavaScript

JavaScript (JS) is object-oriented, cross-platform, script language with C-like syntax. Though its syntax is similar to C, C++ and Java, its semantics are fairly different. It was invented by Brendan Eich in 1995 and was added to the Netscape Navigator. The name JavaScript was chosen just as a marketing ploy for the language to be recognised more thanks to the popularity of Java programming language. With the addition of Document Object Model (DOM), JavaScript became more useful and popular, since the elements on the page were easier to add, remove or edit.

The original intent of JavaScript was to be able to get and modify any of present html elements on the loaded page. In other words, to be able to provide dynamic user interaction like validation of input boxes (name, password, email, phone number...), showing confirmation alerts and using its responses etc. JavaScript goes hand-in-hand with Cascading Style Sheets (CSS) making it a powerful tool in dynamic pages and applications.[20]

Asynchronous JavaScript and XML

Asynchronous JavaScript and XML (known as Ajax) is a process used primarily on web pages and its intent is to access server side in runtime without the need to refresh the page. It is a main process responsible for a term known as Web 2.0, which represents web pages that are considered more like standalone programs.[20]

ECMAScript 6

ECMAScript 6 was the second major revision to JavaScript, ECMAScript 6 is also known as ES6 and ECMAScript 2015.[21] ECMA abbreviation stands for European Computer Manufacturer's Association.[22] ECMAScript is a programming language based on JavaScript and JScript. Originally designed as a Web scripting language with the main intent to make the user experience more visually entertaining and to perform server side computations. Nowadays ECMAScript is used for a whole spectrum of complex programming tasks in numerous different environments. It is currently a fully featured general-purpose programming language.[23]

3.2.3 TypeScript

TypeScript (TS) is an open-source language built on JS. The most prominent contribution is the addition of the static type definitions. With static typing also comes code validation and a more detailed way to define an object or a reference. Any valid JS code is TS compatible (with possible type-checking errors). TypeScript is transpiled into JavaScript with TypeScript compiler or Babel.[24]

3.2.4 Electron

Electron provides a desktop runtime to run HTML, CSS, JS or TS respectively. In other words, it enables development and usage of web pages on desktop with all its technologies. It is a cross-platform framework, that can run on Windows, Linux and macOS. The runtime is divided into two processes:

Main process creates a BrowserWindow instance with its own Renderer process.

Renderer runs its dedicated webpage. Multiple Renderer processes do not interfere with each other. Destruction of BrowserWindow terminates its own Renderer automatically.

Main process is always singular and best practice is to forbid remote access from Renderer. To access the Main functions and features, IPC module is used to send request and then await for the response. The biggest advantage of this setup is that the program can control and filter out all the undefined or prohibited attempts to access the Main process. This prohibits remote code execution attack done from the client side. In this manner, Renderer is permitted to access any of the predefined Main processes by sending an IPC request.[25]

3.2.5 Node.js

Framework that runs JavaScript outside the web browser on the back-end of web application. By following "JavaScript everywhere" paradigm it unifies development of web applications with a single programming language, opposed to earlier style (HTML, CSS, JS for client side and mostly PHP for server side). Node.js design is based on event-driven architecture with ability to handle events asynchronously.[26]

3.3 Design

3.3.1 Cascading Style Sheets

Cascading Style Sheets (CSS) utilisation is to format the layout of assigned GUI elements. It can define text styles, colours, alignments, backgrounds, alpha channel and other attributes. It is mostly associated with HTML. The main advantage of using CSS is to globally define or change any attribute (e.g. colour or size) of an element from a specific group can be done by simply changing one attribute in an included stylesheet file.[27]

Customising style of elements include adding border around boxes, rounding corners of a button, adding background colour, changing font and size of a text etc. These customisation rules can apply to single elements or whole groups.[28]

Cascading Style Sheets is compatible with both HTML and FXML and can be fully utilised. CSS customisations are often pre-made and packed in a framework that is widely used such as:

Bootstrap is widely known CSS framework, mainly used for web development. It is open-source and it is aimed at responsive front-end web development. It offers a variety of stylised HTML elements, such as buttons, forms, input boxes, carousels, sliders and other interface components.[29]

OnsenUI is a open-source framework providing numerous GUI components. It is primarily aimed at application development for iOS or Android mobile operating systems, but it can be used also for developing hybrid applications.[30]

3.3.2 Design development frameworks

HTML pages can be also made with design development frameworks. These frameworks create pages programmatically, therefore eliminating the necessity of writing the pages manually. It can be useful especially with designs with lot of elements repeating itselves. Two widely known frameworks are:

Angular Angular is an framework intended for design development of single-page applications. It uses HTML with CSS and the code is written in TypeScript.[31]

React Open-source JavaScript library for development of front-end interfaces.[32] Opposed to Angular, React is more appropriate for complex applications.

3.4 Chosen technologies

The tool has to be a portable multi-platform desktop application with a simple and easy-to-use GUI. Technologies that are built based on these demands are desired.

3.4.1 GUI Elements and design

According to user's requirements, I decided that the best option would be a simple application with web-like interface. Simplicity is one of the most prominent demands, therefore basic HTML interface, which is well known world wide, is applicable. HTML was chosen as the most preferable tool to create GUI.

3.4.2 Functionality

Since HTML was established as the preferred way to implement GUI, following tools have to be chosen according to their compatibility with HTML. JavaScript is a suitable candidate due to its native compatibility with HTML. JavaScript is a high-level programming language, so it does not provide the same processing speeds as C/C++. However, due to the nature of the tool (e.g. no need for advanced 3d rendering or memory limitations for embedding) JS is sufficient.

ES6 was chosen as the JS standard revision for the project for its arrow functions, better modules handling and unified clean and more readable syntax.

JavaScript runs only on servers on its own and the tool has to be a desktop application, therefore Electron represents an applicable framework to provide Node.js desktop runtime.

TypeScript will be used in certain parts of the code, like Main process of Electron, for its static typing, thus for better variables control and security.

JavaFX and Swing were left behind Electron with JS, because Electron application load times are much faster and overall the experience tends to be smoother and lighter.

3.4.3 Design

Design is an important aspect of the tool. The look should be appealing to the eye while comprehensible at the same time. HTML is tightly bond to CSS. Creating project's own stylesheets would be time consuming and ineffective. Better solution is to include an existing open-source CSS framework. Bootstrap will be used for common HTML elements and OnsenUI for web browser-like tabs. These tabs will be located on the top of the screen and utilised for switching between masks.

Neither Angular or React is used due to insufficient amount of time to work on this thesis. Learning any of these two frameworks from the start would be much more time consuming. Because none of the design development frameworks is involved, TypeScript is not a necessity, thus vanilla JS will be used for view implementation.

Preference of JS over TS has been decided throughout the development, when TypeScript's static data types and strong typing turned out inefficient and rather obstructive in view development. TypeScript is used only with Node.js (Main process in Electron application).

4 Implementation

The calculation process can be divided into several sections, such as importing items or setting up total amounts of stores and POS units. These sections will be represented by masks. Mask is a representation of a single HTML page. Each mask serves a different purpose so the application is simple-looking and well-arranged.

4.1 Design of the User Interface

The User Interface (UI) consists of an overview and masks contained within an iframe element (element showing another HTML page embedding itself in its parent HTML document). Both overview and mask are views and renderer processes. Overview is later reffered to as Index and mask as Iframe. This section describes the view part of the two.

4.1.1 Overview

Overview is composed from three parts:

Header contains tabs representing each of the masks. Clicking on the tab will unload the current mask and load the mask to which is the tab assigned to.

Body is just an iframe element reserved for the mask.

Footer consists of three subsections:

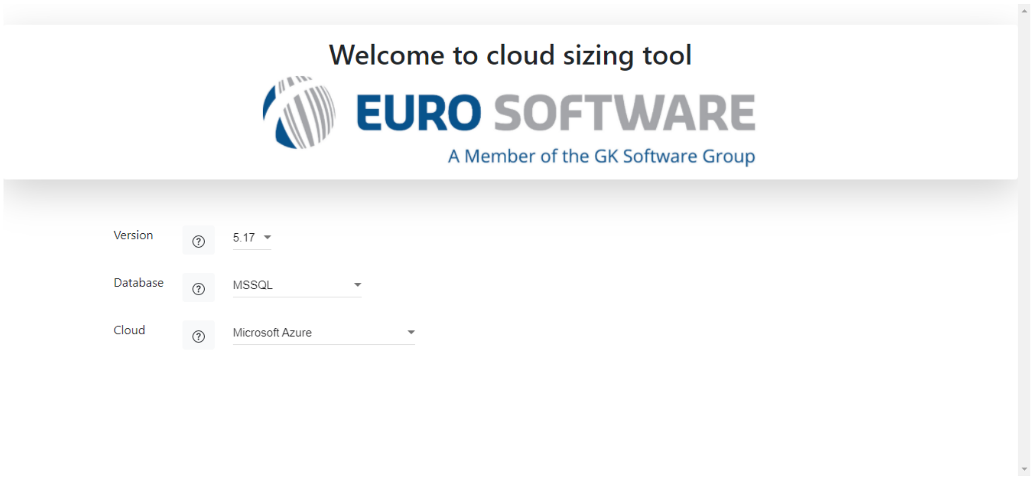
> **Previous and Next buttons** serving the same purpose as clicking on the tabs.
>
> **Footer text** displaying company's name and year of the latest build.
>
> **Save and Load buttons** for keeping user's progress outside of the application storing it in JSON files.

4.1.2 Masks

The whole calculation process can be divided into some main steps. Each of these steps are to be represented by masks. The masks are ordered in the same logic as how the user would proceed in the excel draft. That means if user updates second mask, it will affect the third, fourth and all of the remaining masks.

Customer might also in some cases need to split up the entered attributes for each of the business's store formats. Two or more store formats are needed when customer's business runs in different types and sizes (e.g. hypermarket format is going to have less stores and more POS units opposed to supermarket which will be most likely the complete opposite). Masks that are able to make use of multiple store formats are reffered to as multicards in the code.

Cloud selection (4.1) is the opening mask. The user can choose his/her preferred product version, database engine and cloud vendor. The choice here will mainly influence the firmware used for the following calculations (e.g. different versions of cloud might affect the import speed).

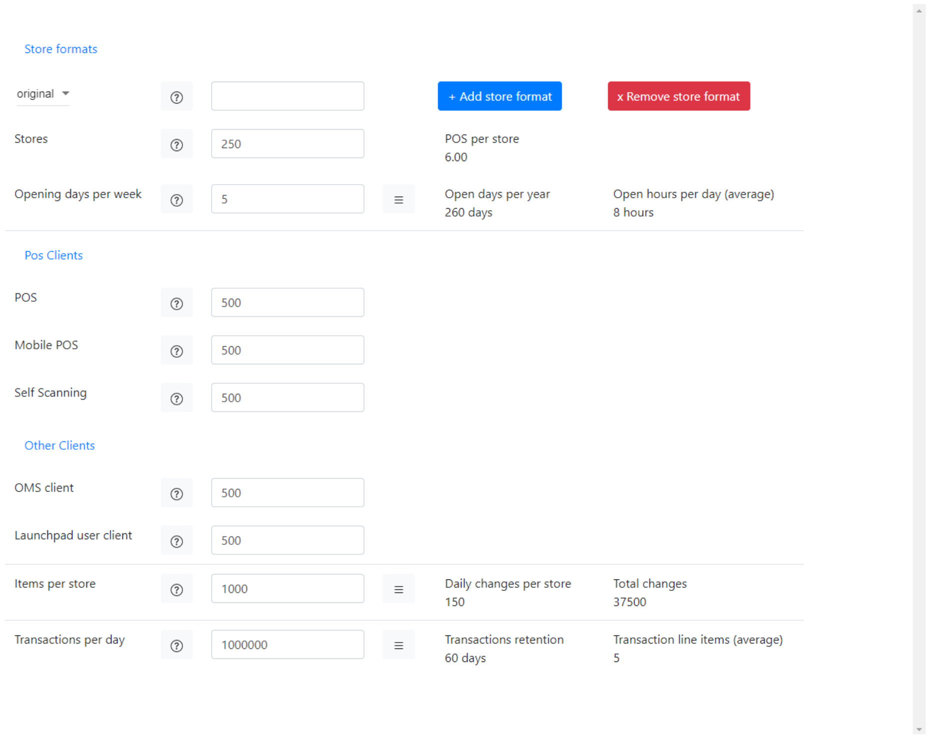

Welcome to cloud sizing tool

EURO SOFTWARE
A Member of the GK Software Group

Version 5.17
Database MSSQL
Cloud Microsoft Azure

Figure 4.1: Screenshot of the Cloud selection mask.

Retailer data volumes (4.2) contains data about types and numbers of POS units. Number of transactions, database items or ratios, such as transactions per day, open hours per week are also stored here. Every one of these attributes can be stored in here multiple times for different store formats.



Store formats

original ▾ ⓘ [ ] + Add store format | x Remove store format

Stores	250	POS per store 6.00	
Opening days per week	5	Open days per year 260 days	Open hours per day (average) 8 hours

Pos Clients

POS	500
Mobile POS	500
Self Scanning	500

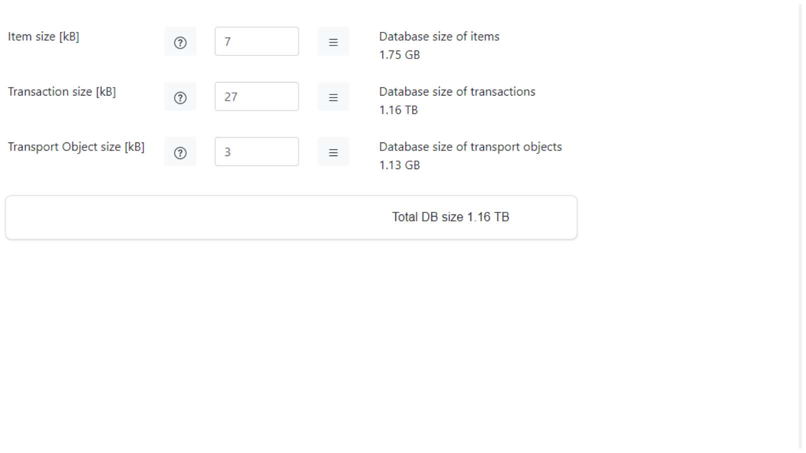
Other Clients

OMS client	500
Launchpad user client	500

Items per store	1000	Daily changes per store 150	Total changes 37500
Transactions per day	1000000	Transactions retention 60 days	Transaction line items (average) 5

Figure 4.2: Screenshot of the Retailer data volumes mask.

Database size (4.3) serves for choosing an average real size of a transaction, transport object and database item in kilobytes. By taking data from the previous mask and combining them with its own input values, it calculates estimated real size of the whole database needed for the business.

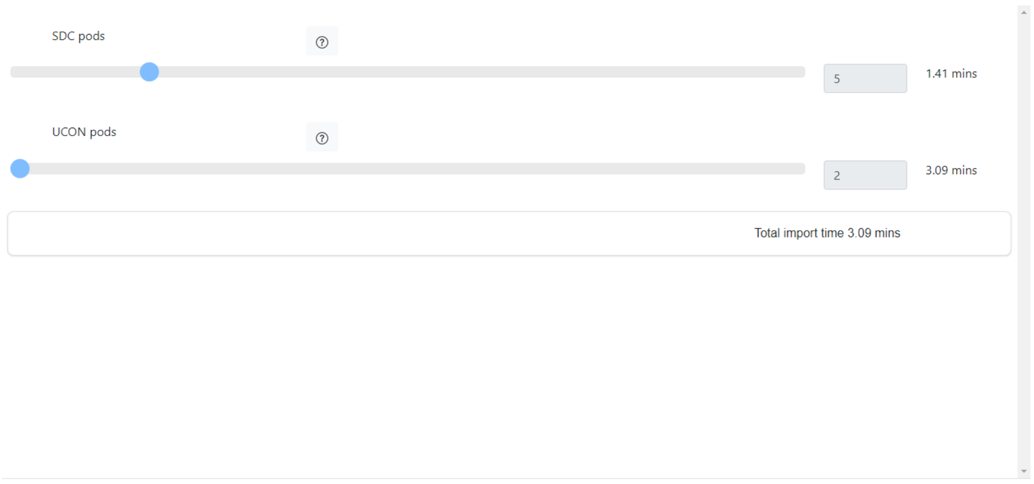


Item size [kB]	7	Database size of items 1.75 GB
Transaction size [kB]	27	Database size of transactions 1.16 TB
Transport Object size [kB]	3	Database size of transport objects 1.13 GB

Total DB size 1.16 TB

Figure 4.3: Screenshot of the Database size mask.

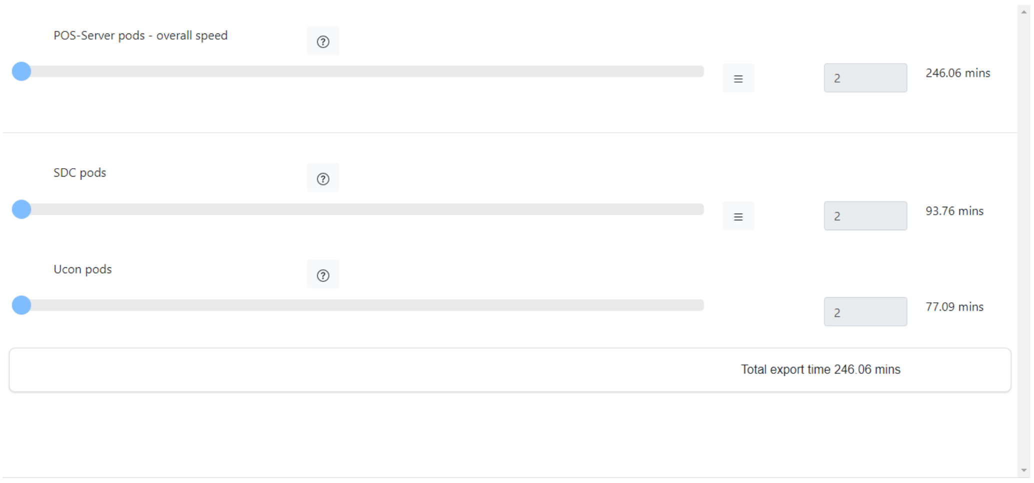
Import items (4.4) takes care of number of the pods for ECON and SDC. Main focus in this mask is on the time that it takes to import Master data from SAP to SDC.



Total import time 3.09 mins

Figure 4.4: Screenshot of the Import items mask.

Transaction processing (4.5) manages number of pods intended for POS server, POS and ECON regarding calculation for accounting and export of transactions from POS to SAP. Layout is very similar to the previous mask.



Total export time 246.06 mins

Figure 4.5: Screenshot of the Transaction processing mask.

Cluster size (4.6) shows the final number of clusters. Cluster's quantity is calculated throughout the whole application. Cluster MIN stands for number of clusters that are essential for basic usage. Cluster MAX is the number of nodes needed for performance peaks (happy hours, Black Fridays, grand opening etc.).

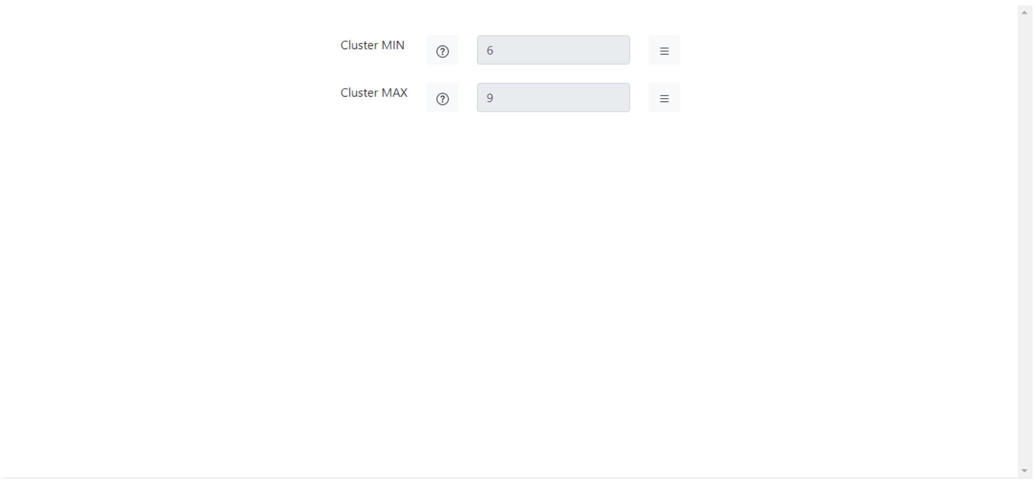
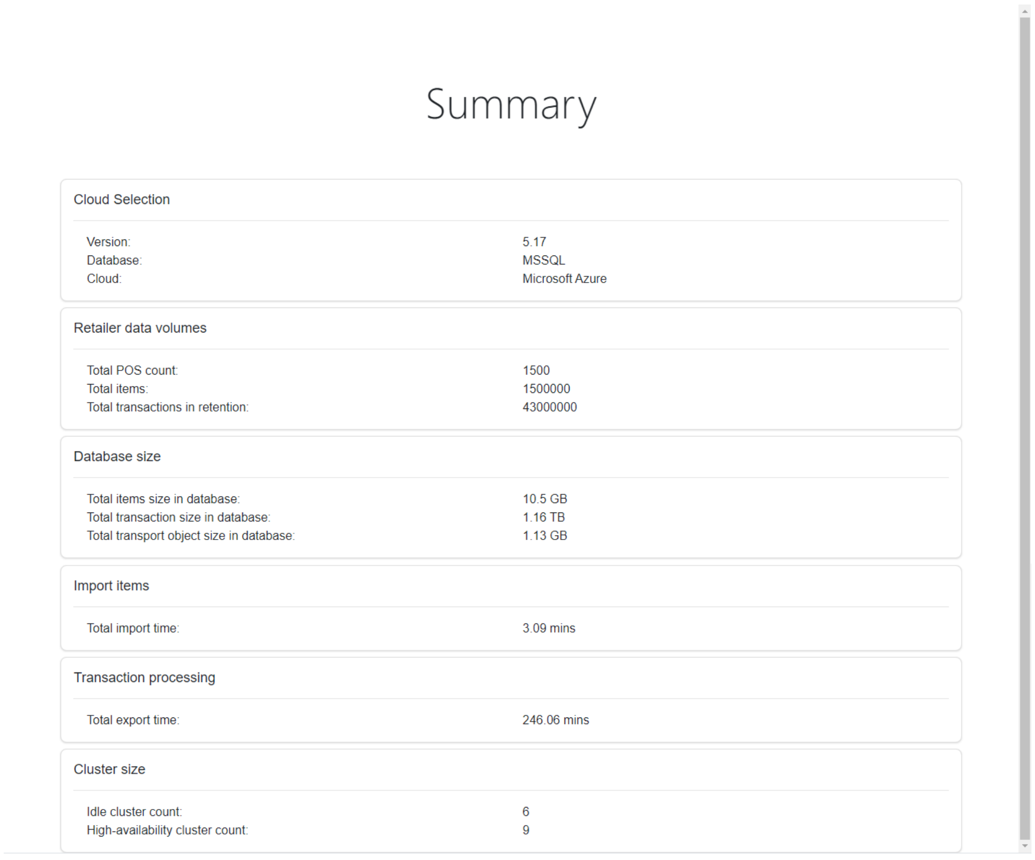

Cluster MIN 6
Cluster MAX 9

Figure 4.6: Screenshot of the Cluster size mask.

Summary (4.7) sums up the whole progress of the user. It also checks if each of the masks has been properly checked and if any of the masks need recalculation by re-visiting it (caused by non-linear editing of masks).



Summary

Cloud Selection

Version:	5.17
Database:	MSSQL
Cloud:	Microsoft Azure

Retailer data volumes

Total POS count:	1500
Total items:	1500000
Total transactions in retention:	43000000

Database size

Total items size in database:	10.5 GB
Total transaction size in database:	1.16 TB
Total transport object size in database:	1.13 GB

Import items

Total import time:	3.09 mins

Transaction processing

Total export time:	246.06 mins

Cluster size

Idle cluster count:	6
High-availability cluster count:	9

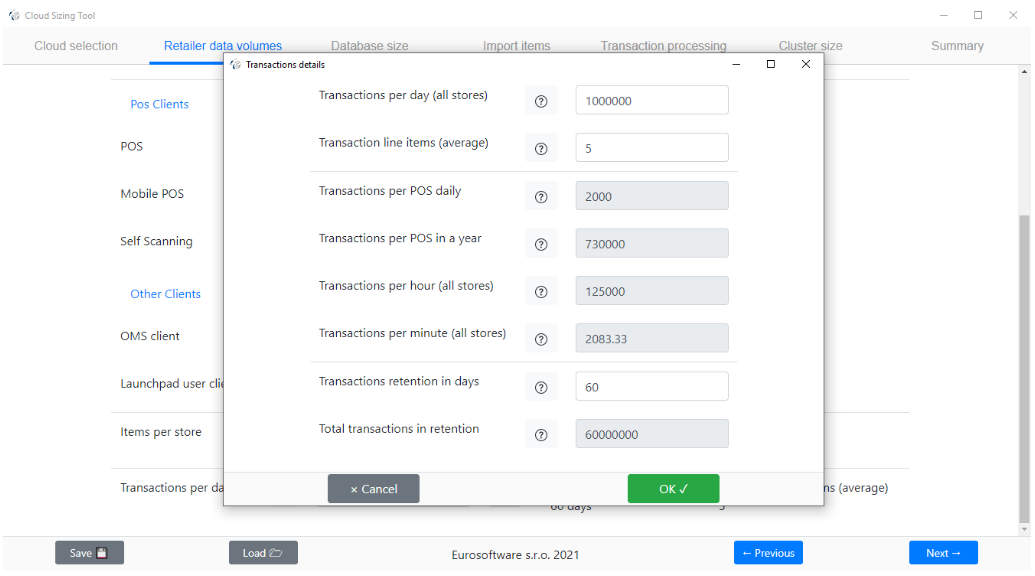
Figure 4.7: Screenshot of the Summary mask.

Submasks

4.8 In most of the masks hamburger menu buttons can be seen. These open new modal windows one at a time for a more detailed view of its subject. Each submask is a standalone renderer process also reffered to as Modal in the code.

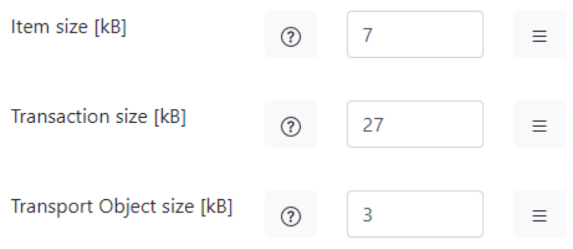
Hints

4.9 Almost every mask contains question mark buttons which upon clicking show a small alert. The alert describes the input value, that is put near the clicked question mark button, in more detail.



Transactions per day (all stores)	1000000
Transaction line items (average)	5
Transactions per POS daily	2000
Transactions per POS in a year	730000
Transactions per hour (all stores)	125000
Transactions per minute (all stores)	2083.33
Transactions retention in days	60
Total transactions in retention	60000000

Figure 4.8: Screenshot of submask Transaction details from Retailer data volumes mask in a separate modal window



Item size [kB]	7
Transaction size [kB]	27
Transport Object size [kB]	3

Figure 4.9: Screenshot from Database size mask depicting Hints (in the middle) and Submask buttons (on the right)

4.2 Flow

Electron uses two types of processes:

Main can be perceived as the server side of the application, due to its privileges and usage of Command Line Interface (CLI). The application starts with launching Main process first, which is then able to launch any other Renderer processes. It is capable of creating and shutting down modal windows, manipulating the system's file system (with saving and loading external files) or shutting down the whole application including itself.

Renderer represents the GUI part of the application. On its creation it is given an HTML file as its index page. Renderer opens up in a Chromium window showing its index page.

4.2.1 Renderer processes

Every one of the Renderer processes has its own data set. That makes them behave like separate applications, therefore getting any static class data that was set during runtime is no use. This is a considerable difference from static class behaviour in Java or C/C++. To acquire data from a different process, Inter-Process Communication (IPC) has to be used (described more in Features section bellow).

The application's Renderers can be categorised into three main processes:

Index is the first and also the main Renderer process that throughout the whole tool does not get reloaded once (meaning the index.html is loaded all the time since launch of the application). It is the most used gateway to IPC with Main. It handles most of the calculations, data management and UI updating.

Iframe is the embedded window within Index displaying masks. It takes care of functionality unique for each mask, thus behaviour that can not be generalised in Index.

Modal can be portrayed as Iframe (similar functionality) detached in a new modal window. Upon closing with an intent to save the changed data (clicking on the OK button), Iframe needs to be updated.

Index is always shown in the application's runtime and it is only one. Iframe and Modal's index files are dynamically assigned depending on which mask or submask the user enters. However, every one of the Renderer processes can be shown one at a time, meaning the application is preventing the user to open up two or more masks or submasks at once under one running instance of the tool.

4.2.2 Features

Inter-Process Communication

It is possible for Renderer to acquire most of the privileges Main process has, however it is strongly recommended not to, due to security issues. The preferred way is to create an preload.js file, that exposes functions *send* and *receive* through custom valid channels. This preload is then used as an attribute in Renderer creation.

This way for example Renderer can send through valid channel a request for reading data of an external file through preload. If the channel is correct and data is in a valid predefined format (can be the name of the file in a string), request gets to the Main. Main has a listener on this request, which gets triggered, Main reads the file and sends the content back to Renderer through preload, but via different channel.

The channels are distinctively named with suffixes *_REQ* for request and *_SUB* for submit where request is the initiating signal and submit is the reaction on it (usually comes with some data).

Switching masks

Switching masks involves four steps:

1. Saving data from the current mask to mark down changes.
2. Changing Iframe src attribute to the file adress of the new mask.
3. Loading data from the default data set or previously saved, if the mask was visited.
4. Setting up HTML elements in the mask's body. This involves everything from loading stylesheets, assigning correct minimum/maximum attributes of input elements to setting up event listeners.

The whole process is covered in a loading screen to prevent user from seeing HTML elements not fully loaded with correct data and stylesheets or using event listeners that yet has not been completely set and possibly generating some errors.

Save and Load

From Index is sent the whole data set containing the chosen values by the user to Main for it to save. On load it behaves the same, except the data set travels the opposite way.

4.3 Code

Source code follows Model View Controller (MVC) architecture. Model is used for mock-data and contains a prototype of Database adapter that is yet to be implemented in future development. Controller and view share the same structure of sub-folders which is as follows:

```
|  - masks
|  |__- submasks
|  |   |__- mask1 submasks
|  |   |   |__+ mask1 submask A
|  |   |   |__+ mask1 submask B
|  |   |__- mask2 submasks
|  |   |   |__+ mask2 submask A
|  |   |__...
|  |__+ mask1
|  |__+ mask2
|  |__...
|__+ index
```

The main difference between Controller and View in the aforementioned structure is the extension type of all the files (.html in View and .js in Controller). Other changes include:

Assets folder in View containing tools stylesheets and tools for them.

data_handlers folder in Controller with classes handling the data flow (described in more detail further down).

helpers folder in Controller with helper modules used for data or UI handling.

main.ts and preload.ts files in Controller.

4.3.1 Conventions

private and public

Since JavaScript does not specify visibility, the tool's source code uses Python-like convention where underscore as the first character in a name of a function/variable implies that it is meant to be private. Everything else is taken as public.

uppercase and underscore

To make clear distinction about variables and functions where they come from, two writing styles are used for their declarations.

Usage of uppercase for first letter of each word in a name is used only in class-related elements (e.g. _posItemsTransationsMask(), getIDPoses(), totalMap, const dMap).

Using lowercase only and underscore as a word separator in names is used in modules (e.g. set_received_map(datamap), update_pps_ratio(), const pos_per_store).

static constants in classes

JavaScript does not use static constants in classes, so convention of naming these full uppercase with underscore as a word separator is used (e.g. static T_POS, static SUBM_FOOTER_ID, static IFRAME_LOAD_SUB).

4.3.2 Data handlers

From the perspective of abstract data flow (and not the UI functionality itself) the most important classes are the ones included in the *data_handlers* directory in the Controller's source code.

Constants

A class consisted solely of static attributes. It is used throughout the whole code for referencing global keywords such as HTML id, class or other attribute names, masks identifiers or preload's channel names.

DataMap

The core of the tool's data storage. Every HTML element in a view, that needs its value to be stored, contains special *datamap* attribute. This instructs Controller to read element's value from DataMap on loading the view. Saving the value is triggered on view's unloading.

Class itself is a singleton with five key Object type attributes:

_DATA JSON with data intended for every key HTML element in each view. Example can be seen at 4.10. Values are loaded from this structure by default. In future development, values will change due to changes in Model's adapter triggered with changes in Cloud selection mask. Values can also be arrays if the element is a select. Structure of the object is as follows with each level nested in its parent:

```
_DATA
└─mask id
   └─store format id (default 'original')
      └─actual data
```

_choice has the exact structure as _DATA. Its purpose is to store user's chosen values. Its initial state contains all the masks on the first level,

```
_DATA = {
    "mask−1" : {
        "original" : {
            "element−1" : 5,
            "element−2" : 7,
            "select−element" : [
                "option1",
                "option2",
                "option3"
            ]
        }
    },
    "mask−2" : {
        "original" : {
            "element−A" : "value−A"
        },
        "another−store−format" : {
            "element−A" : "value−B"
        }
    }
}
```

Figure 4.10: Example of a small _DATA with two masks. The later is a multicard.

each of them having one default store format, which contains empty JSON.

If any mask has this state in _choice, that indicates the mask has not yet been visited because no data is set here. Upon leaving any mask, every *datamap* HTML element gets saved into _choice.

In a nutshell, _choice[mask id][store format id] can be either empty (not visited), same as its _DATA counterpart (visited but nothing changed) or with the same keys as in _DATA but with different values. In case of select HTML elements, array from _DATA gets converted into one selected value.

_totalValues is a JSON structure that contains any data crucial for calculations that can not or are not appropriate to be stored as a *datamap* value for the sake of simplicity (e.g. total amount of POS units or complete size of all transactions stored in database). Values stored in _totalValues should be related in some way to total counts.

_view holds selected HTML elements and its custom attributes to be set during view loading. Example can be seen at 4.11.

```
_view = {
    "mask-1" : {
        "element-1" : {
            "max" : 5
        }
    }
}
```

Figure 4.11: Example of a _view that sets attribute *max* with value 5 to *element-1* which is located in *mask-1*.

_measurments is a structure containing preloaded firmware mostly for mocking purposes.

CalcFunctions

Every key calculation that is not exclusively connected to a single view takes place in CalcFunctions class. That means what calculations can be seen in a view (e.g. Retailer data volumes mask's POS per store value) do not belong here. CalcFunctions takes care of calculations such as total size of Transport Objects in database where it is necessary to get values from multiple masks. Also calculations of any support values that are needed in the former calculations are implemented in this class (e.g. total amount of items is later used in counting total size of items in database). All of the calculated values are either returned or saved into DataMap's _totalValues structure for later use.

Masks have their own private parameter-less methods reserved (conventionally named _maskNameMask()). Since the masks are ordered in a one-way (changes in a mask influences all masks on the right, excluding Cloud selection and Summary), CalcFunctions mask methods get called in the same manner. Some of the masks like Database size have its own public methods with specific return types. They are used for dynamic calculations while in the mask (hence the method's naming convention maskName_Dynamic).

CalcFunctions uses both labels functions and methods. Functions are the special mask functions (one for each mask, name starts with _ prefix and ends with *Mask* suffix). Any other method is reffered to as a method.

5 Testing

Testing is covered by automated Mocha.js tests. Mocha.js is a simple and customisable library intended for unit and integration testing in Node.js environment. [33]

Tests in the thesis cover the class CalcFunctions since it takes care of sophisticated calculations. The whole class is covered by tests in a file *CalcFunctions.test.js* located in directory *src/test*. The test class covers 77 use cases.

Tests are divided into several sections:

creating instances tests are focused on constructor and Singleton pattern.

getters cover methods with a return type (name starts with *get* or *_get*) that are not mask functions.

mask functions take care of unique methods dedicated to each mask. It is those methods that CalcFunctions class stores in its static array *_functionsArray*.

_Dynamic are dedicated to methods that share suffix *_Dynamic*. Helper methods for some of the masks.

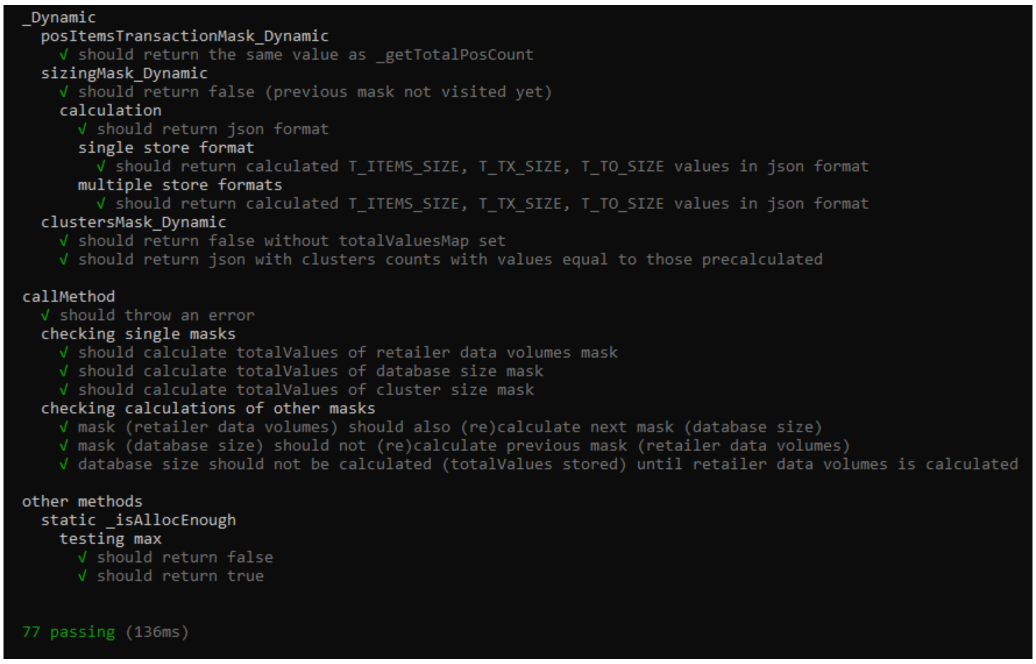
callMethod tests cover only the callMethod.

other methods include any other methods that do not fit in any of the previous categories.

5.1 Launching

At the root folder run following command in command line. Tests should be executed one after another with understandable descriptions. Project has to be installed and npm version 7.5.4 is required to launch tests. Successful launch in a command line can be seen at 5.1.

```
npm test
```



```
_Dynamic
  posItemsTransactionMask_Dynamic
    √ should return the same value as _getTotalPosCount
  sizingMask_Dynamic
    √ should return false (previous mask not visited yet)
    calculation
      √ should return json format
      single store format
        √ should return calculated T_ITEMS_SIZE, T_TX_SIZE, T_TO_SIZE values in json format
      multiple store formats
        √ should return calculated T_ITEMS_SIZE, T_TX_SIZE, T_TO_SIZE values in json format
  clustersMask_Dynamic
    √ should return false without totalValuesMap set
    √ should return json with clusters counts with values equal to those precalculated

callMethod
  √ should throw an error
  checking single masks
    √ should calculate totalValues of retailer data volumes mask
    √ should calculate totalValues of database size mask
    √ should calculate totalValues of cluster size mask
  checking calculations of other masks
    √ mask (retailer data volumes) should also (re)calculate next mask (database size)
    √ mask (database size) should not (re)calculate previous mask (retailer data volumes)
    √ database size should not be calculated (totalValues stored) until retailer data volumes is calculated

other methods
  static _isAllocEnough
    testing max
      √ should return false
      √ should return true


77 passing (136ms)
```

Figure 5.1: Finished tests for _Dynamic, callMethod and other methods test groups in a Windows command line.

6 Verification

6.1 User Interface extensibility

The tool should be extensible so in future development other developers can easily contribute to the project.

6.1.1 Extending an existing mask

Developer wants to add a new input datamap element into the first mask. The element should have id *new-el.* Adding the new element should be done by following these steps:

- Insert the new element somewhere into the body of the first mask.

  ```
  <input id="new-el" datamap>
  ```

- In model structure inside mock data add a new key with id of the new element inside the JSON substructure of the first mask. The new entry should have some default value to the developer's liking.

  ```
  "cloud-selection": {
     ...
     "new-el" : 5
   }
  ```

- *Optional:* In view structure inside mock data, developer can add new entry under the first mask, defining attributes to the element such as type, minimum or maximum.

  ```
  "cloud-selection": {
      "new-el" : {
          "min" : 1,
          "max" : 10,
          "type" : "number"
      }
  },
  ...
  ```

After following these steps the new element is added to the first mask (6.1). It shares the same functionality as any other datamap element, meaning it

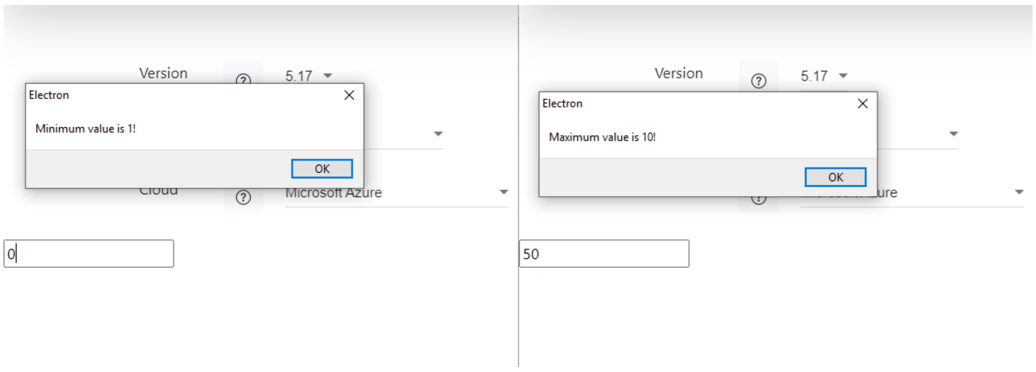


Figure 6.1: Newly added input element with dynamically set minimum ’1’ and maximum ’10’ attributes at the bottom of Cloud selection mask.

is effectively saved into the DataMap object and can be exported to a JSON file.

6.1.2 Adding a new mask

Developer want to add a new mask in between the Cloud selection and Retailer Data Volumes masks (the first one and the second one). The mask is going to have one input, same as the one in the previous case. For the input element same methods can be used to customise it.

- Add a new HTML file in directory *src/view/masks* named *new-mask.html*. The file must contain the necessary data including the new input element.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>New mask custom name</title>
    <title-id id="new-mask"></title-id>
</head>

<body id="iframe-body">
    <input id="new-el" datamap>
</body>

</html>
```

- Add a new JS file in directory *src/controller/masks* named the same as the view file in the previous step, but with *.js* extension instead. The file will contain a script that adds a new listener to the input. The input element on change will trigger an alert telling the user *Input changed!*.

```
document.getElementById('new-el')
.addEventListener('change', () => {
    alert('Input changed!')
})
```

- At index.html file add a new label between the two previously mentioned masks.

```
...
<div header>
    <label id='01-intro.html'></label>
    <label id='new-mask.html'></label>
    <label id='02-pos_items_transaction.html'
        multi-subcard></label>
...
```

- In model structure inside mock data add a new entry on the first level (preferably between the first two masks to be easily readable). The entry has to have the id of the mask for a key and JSON containing every datamap object in the mask, providing it with a value.

```
...
},
"new-mask" : {
    "new-el" : 5
},
"retailer-data-volumes": {
...
```

- In Constants class add the mask's id to the MASK_IDS structure and a custom name (that will be displayed on its tab) to the MASK_NAMES structure, both at the correct position.

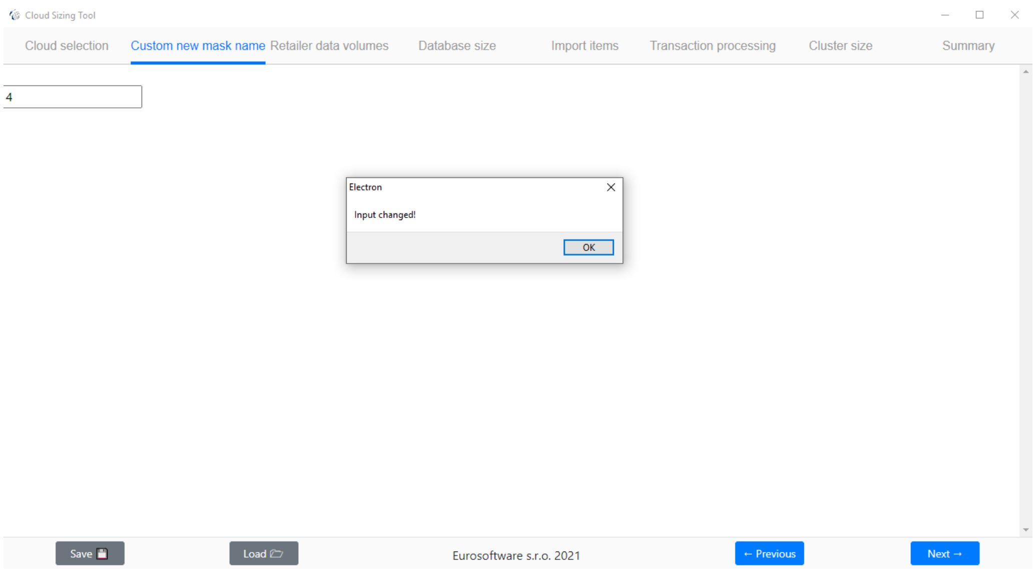
```
static MASK_IDS = [
    "cloud-selection",
    "new-mask",
    "retailer-data-volumes",
...
static MASK_NAMES = [
    "Cloud selection",
    "Custom new mask name",
    "Retailer data volumes",
    ...
```

- *Optional:* In CalcFunctions class declare a new function (can be empty if nothing is needed to calculate at the moment) named *_newMaskMask()* and add a reference to it into the *_functionsArray* structure at the right position (between the first two functions).

```
...
 _functionsArray = [
    this._introMask,
    this._newMaskMask,
    this._posItemsTransactionMask,
...
_newMaskMask() { /*empty*/}
...
```

New mask appears after launch between the first two masks in the header. Upon clicking, a simple mask with one input of value 5 is loaded. When changing the value a new alert pops up showing the *Input changed!* message (6.2).

The tool's UI is fully and easily extensible fulfilling one of the most important requirements on this thesis.



Electron

Input changed!

Figure 6.2: Screenshot depicting the newly added mask. The alert is a result of the input's on change function being triggered.

6.2 Data manipulation

Cloud sizing tool should be able to correctly handle data from API. At this moment, API is represented by mock data in *src/model* directory, later in future development to be implemented by adapter module. The only significant difference is the mock data being static, therefore the current version's data handling can be verified.

6.2.1 Altering default data

In DataMap class its structure *_DATA* is used as a default data set to load, in case user has not yet visited the loaded mask. Data to create the structure are fetched from the API, therefore it can be altered here. Any datamap element's default value can be changed inside of the mock data's model structure. The developer should keep in mind the limits and attributes of the edited element.

6.2.2 Altering incorrectly

As mentioned above altering the data can be done quite easily. Nevertheless type of the current element needs to be respected (input of type number expects numbers, select expects an array of values etc.). Due to JavaScript's nature, passing number as a string will not result in any errors, but it is not recommended to do so due to probable future incidents mistaking the '+' symbol for addition and not concatenation ("10" + "20" = "1020"). On the other hand, passing incompatible values or objects (e.g. string containing non-numeric characters to a number-type input) element will trigger a warning in console and left the element empty. Passing a single value into an select (which is expecting an array of values) will result in error stopping the course of the script.

The data fetched from API can be altered and aside from the data types of the HTML elements, they are not statically dependent on the code.

6.3 Calculations

The tool cannot be directly compared to the current Excel draft calculations because it does not cover the whole process. The primary objective was to create a frontend application that is going to replace the Excel draft later in future development. The whole process includes more than a hundred of calculations that are spread over around 40 lists of the Excel draft. As mentioned before, to include the entire process, it would take a time span of another bachelor thesis.

Based on the data set in 6.3 table on page 50, correct total database size can be manually calculated. This number can be also seen on the Database size mask as well as the auxiliary calculations from which the final number is composed of.

6.3.1 Items size

Complete sum of items in each store needs to be added up to a one number. This is counted by multiplicating average items per store value with total amount of stores.

```
Sum of items = average(items~per~store) * sum(stores)
Sum of items = 1’000 * 250
Sum of items = 250’000
```

This number has to multiplied with real average size of one item in database.

```
Items size = sum of items * one items size
Items size = 250’000 * 7 kB
Items size = 1.75 GB
```

Number of stores	250
Open hours per day	8 hours
Open days per week	5
POS units	500
Mobile-POS units	500
Self-scanning units	500
All POS clients per store	6
OMS-client	500
Launchpad users	500
Items per store	1,000
Daily changes	150
Total changes	37,500
Transaction per day	1,000,000
Transaction retention	60 days
Transaction line items	5
Item size	7 kB
Transaction size	27 kB
Transport Object size	3 kB
Transport Object days	10 days

Figure 6.3: Table showing input values for following calculations.

6.3.2 Transactions size

Similarly to the previous calculation, transaction size and total amount of transaction need to be multiplicated by each other to get the real size of amount of transactions stored in database. One crucial attribute is TX retention (Transaction retention) which stands for how many days each transaction is meant to be stored in database. For one transaction per day, TX size 1 kB and TX retention 30 days total size of TX database would need 30 kB of space.

TX retention is measured in absolute days (seven per week), but TX per day applies only on how many days per week is the store opened. These values need to be synchronised to absolute scale to get correct amount of average transactions each day stored in database including the ones in retention. In case of multiple store formats, Total TX retention calculation has to be repeated with each store format and added up to a total sum.

```
Total TX retention =
            ceil( days for retention * open days per week / 7 )
            * transactions per day
Total TX retention = ceil( 60 * 5 / 7 ) * 1’000’000
Total TX retention = ceil( 42.8571~) * 1’000’000
Total TX retention = 43 * 1’000’000
Total TX retention = 43’000’000
```

To get the final sum of TX size in database, total TX retention needs to be multiplied with average TX size.

```
Transactions size = Total TX retention * TX size
Transactions size = 43’000’000 * 27 kB
Transactions size = 1’161 GB
```

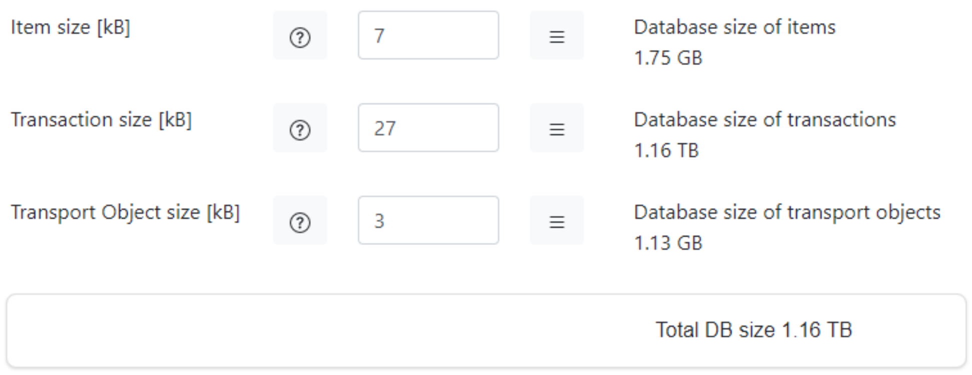
6.3.3 Transport Objects size

To get the real database size of Transport Objects, total changes with TO days and TO size need to be multiplied with each other. Total changes, as well as in the case with total TX retention, needs to be calculated separately for each store format (if more than one is present) and summed up before the final calculation.

```
Transport objects size = TO size * TO days * total changes
Transport objects size = 3 kB * 10 * 37’500
Transport objects size = 1.125 GB
```

Size of all the items	1.750 GB
Size of all the Transactions	1'161.000 GB
Size of all the Transport Objects	1.125 GB
Total size of the database	1'163.880 GB

Figure 6.4: Table showing manually calculated sizes of database's parts.



Item size [kB]	7	Database size of items 1.75 GB
Transaction size [kB]	27	Database size of transactions 1.16 TB
Transport Object size [kB]	3	Database size of transport objects 1.13 GB

Total DB size 1.16 TB

Figure 6.5: Screenshot of the tool's Database size mask depicting calculated values.

Total size of the database

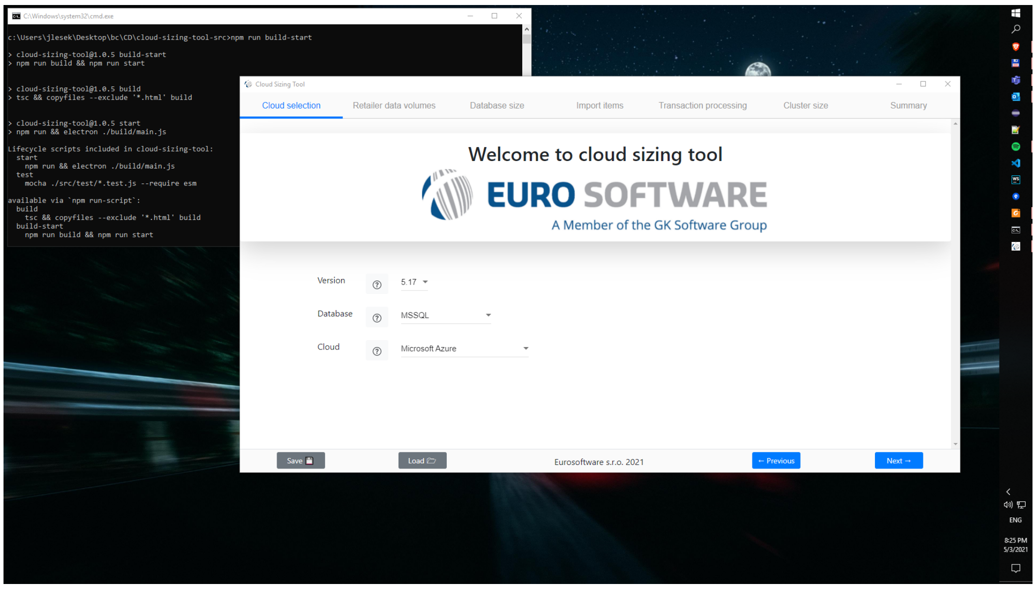
Complete size is counted simply by adding up each of the three previously calculated values.

```
Total size of the database = Items size
           + Transactions size + Transport objects size
Total size of the database = 1.75 GB + 1'161 GB + 1.13 GB
Total size of the database = 1'163.88 GB
```

All these previously mentioned numbers manually calculated are written in table 6.4 on page 52 to compare them to screenshot of the same values, only calculated by the tool (6.5). There is a slight difference in the rounding, but that is not significant, because the final value in future versions is going to be rounded up to fit in whole number of pod units.

6.4 Cross-platformity

One of the crucial requirements of the application is to be multi-platform. Multi-platformity was tested on Windows and Linux.



```
c:\Users\jlesek\Desktop\bc\CD\cloud-sizing-tool-src>npm run build-start

> cloud-sizing-tool@1.0.5 build-start
> npm run build && npm run start


> cloud-sizing-tool@1.0.5 build
> tsc && copyfiles --exclude '*.html' build


> cloud-sizing-tool@1.0.5 start
> npm run && electron ./build/main.js

Lifecycle scripts included in cloud-sizing-tool:
  start
    npm run && electron ./build/main.js
  test
    mocha ./src/test/*.test.js --require esm

available via `npm run-script`:
  build
    tsc && copyfiles --exclude '*.html' build
  build-start
    npm run build && npm run start
```

Welcome to cloud sizing tool

Version: 5.17

Database: MSSQL

Cloud: Microsoft Azure

Eurosoftware s.r.o. 2021

Figure 6.6: Windows 10 running live version of the tool.

6.4.1 Windows

Both versions run without problems. Live can be seen at (6.6) and package at (6.7). Both versions were tested on Windows 10 with following specifications:

Edition: Windows 10 Enterprise

Version: 1809

6.4.2 Linux

Live version was tested on Kubuntu version 20.04 with following specifications:

KDE Plasma Version: 5.18.5

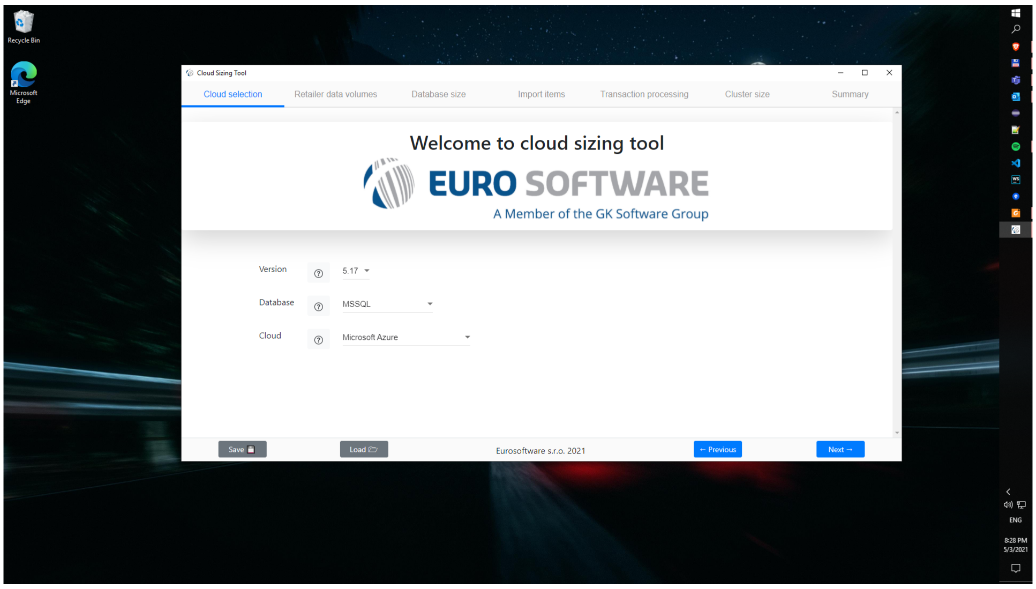
KDE Frameworks Version: 5.68.0

Qt Version: 5.12.8

Kernel Version: 5.8.0-50-generic

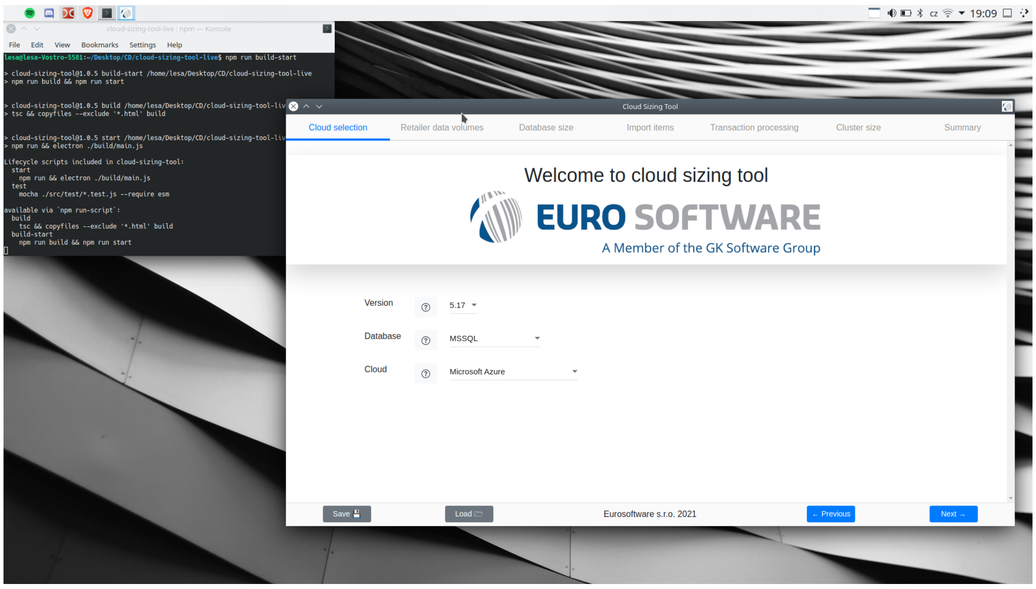
OS Type: 64-bit

Application runs without problems (can be seen at 6.8). Packaged version could not be tested since it is Windows exclusive. Packaged versions for other platforms may be included in future development.



Version 5.17
Database MSSQL
Cloud Microsoft Azure

Figure 6.7: Windows 10 running packaged version of the tool.



```
lesa@lesa-Vostro-5581:~/Desktop/CD/cloud-sizing-tool-live$ npm run build-start

> cloud-sizing-tool@1.0.5 build-start /home/lesa/Desktop/CD/cloud-sizing-tool-live
> npm run build && npm run start


> cloud-sizing-tool@1.0.5 build /home/lesa/Desktop/CD/cloud-sizing-tool-liv
> tsc && copyfiles --exclude '*.html' build


> cloud-sizing-tool@1.0.5 start /home/lesa/Desktop/CD/cloud-sizing-tool-liv
> npm run && electron ./build/main.js

Lifecycle scripts included in cloud-sizing-tool:
  start
    npm run && electron ./build/main.js
  test
    mocha ./src/test/*.test.js --require esm

available via `npm run-script`:
  build
    tsc && copyfiles --exclude '*.html' build
  build-start
    npm run build && npm run start
```

Version 5.17
Database MSSQL
Cloud Microsoft Azure

Figure 6.8: Kubuntu 20.04 running live version of the tool.

7 Discussion

7.1 Design of the UI and masks

Designing took place in the beginning of the tool's development. It was conducted according to the Excel draft structure to follow its logical flow. Base of the calculation process was formed this way and number, order and content of the masks came out of it. Each mask represents a certain section in the Excel draft. Header of the application can be perceived as a indirect substitute to the draft's lists. This decision was the most logical and convenient for later draft users to adapt to it.

The Excel draft can be perceived as overwhelming and difficult to understand at first glance. Submasks were introduced to hide complex details into separate windows in order to aerate and lighten the first moment experience. To acquire steeper learning curve, hints (question mark buttons) were added to the masks, showing detailed info upon clicking about each input element to possibly eliminate user's uncertainty.

For more modern look, the application window could have been frame-less with drag-and-drop mechanics to move it around. That was the original plan, however it turned out to be way more difficult to develop and for time reasons, the task was skipped.

7.2 Technology choice

The selection of Electron framework with combination of JavaScript turned out as an appropriate technology choice. The application was run both on Windows and Linux systems throughout the development and its runtime was smooth and reactive most of the time. Combination with Bootstrap and OnsenUI frameworks worked flawlessly.

In retrospect Angular framework would be suitable for this kind of application. The view section in code would be assumably more compact, cleaner and more flexible. However vanilla JavaScript in ECMAScript 6 was chosen instead due to my lack of knowledge of Angular in the early development.

7.3 Code architecture

The code is structured into a clean Model View Controller architecture and thoroughly uses module and class handling capabilities of ECMAScript 6, effectively separating each language used. Most of the JavaScript classes and some complex modules are well documented for easier further development. As explained in 6.1, the code is flexible and easily extensible mainly in terms of masks management.

As mentioned in 3.4.3, TypeScript was marked as inferior over JavaScript for scope of this application, due to its over-protective syntax checking causing unnecessary trouble when handling HTML documents. However classes CalcFunctions, DataMap and Constants (known as data handlers) should have used more TypeScript for better type control. Any of these classes do not handle any pure HTML code, thus the overly protective syntax is not a problem here.

7.4 Comparison with company's Intranet

The company utilises Intranet web application mainly for the needs of the personnel department. Intranet is a web application utilising Node.js as its backend and Angular as its frontend technology.

When comparing Intranet and Cloud sizing tool, there are mainly two biggest differences in terms of used technologies. Cloud sizing tool does not use Angular and Intranet does not use Electron, since it is a web application. UI structures of both applications are similar, both of them are divided into several pages or masks. Intranet is more advanced in GUI since it uses Angular-compatible components like floating panels. Its HTML code tends to be shorter because it is dynamically generated with Angular based on its keywords contained in the view, therefore the view is more readable that way. Cloud sizing tool is going to grow in size with further development and the code structure may get more complicated, thus Angular might be implemented in the future to lighten the code and make it more manageable.

Cloud sizing tool and Intranet are meant to be used for entirely different purposes. However both of the applications were created to substitute old technology with newer and better one. Also both of them are large-scale projects with used technologies like JavaScript, TypeScript and Node.js, therefore company should have sufficient resources to fund and manage applications with these technologies in the future.

8 Conclusion

In a span of this thesis, I was acquainted with the whole calculation process of computation power. That required also getting well-informed about all of the technologies that cloud4retail solution uses, as well as becoming familiar with the most important GK/Retail applications and components that are used in the calculation process.

In order to pick the most applicable technologies for the tool, I examined and studied some of the most prominent tools and frameworks for modern frontend application development. Electron in combination with HTML, JavaScript (ECMAScript 6 variant) and TypeScript for languages and Cascading Style Sheets with Bootstrap and OnsenUI framework for the design.

Since the tool was being built from the ground up, the whole Graphic User Interface was to be designed. The design's flow was inspired by Excel draft's structure. The GUI has an emphasis on being light-weight and user-friendly. API was built in a way to properly suit the handling of JSON data structures.

Code structure is done in a Model View Controller architecture and using ECMAScript 6 module and class handling, making the code cleaner and more readable for future development. Calculations were implemented up to database size counting, rest of them were done by pseudo-calculations using static firmware. Automatic Mocha.js testing covers the whole CalcFunctions class which takes care of more complex calculations. Detailed documentation is present in the source code on the attached CD.

Cloud sizing tool's development succeeded in a time span of this bachelor thesis regarding creation of a modular basis for future development. It is a cross-platform frontend application with a simple and easy to use GUI.

Final version of the tool in the thesis span has been handed over to the company. The company plans to develop the tool further to be able to use it in practice. The following milestone is to develop a version that company's employees can use for auxiliary calculations to serve real customers more efficiently. The principles for elaboration of this bachelor's thesis are met.

Bibliography

[1] EuroSoftware s.r.o. *Úvod - Co děláme*. 2021. URL: `https://www.eurosoftware.cz/#co-delame` (visited on 31/01/2021).

[2] Miroslav Štrobl. *Big Picture - GK Development Academy - Dashboard*. Source placed on CD. 20th Apr. 2016. URL: `confluence/Big_Picture-v11.png` (visited on 12/02/2021).

[3] Miroslav Štrobl. *OmniPOS Introduction - GK Development Academy - Dashboard*. Source placed on CD. 16th Apr. 2021. URL: `confluence/OmniPOS_Introduction-v30.png` (visited on 31/01/2021).

[4] Miroslav Štrobl. *SDC Introduction - GK Development Academy - Dashboard*. Source placed on CD. 16th May 2019. URL: `confluence/SDC_Introduction-v15.png` (visited on 31/01/2021).

[5] Miroslav Štrobl. *Enterprise Connector (ECON) - GK Development Academy - Dashboard*. Source placed on CD. 28th May 2018. URL: `confluence/Enterprise_Connector_(ECON)-v6.png` (visited on 31/01/2021).

[6] Miroslav Štrobl. *POS Server Introduction - GK Development Academy - Dashboard*. Source placed on CD. 14th Aug. 2020. URL: `confluence/POS_Server_Introduction-v32.png` (visited on 31/01/2021).

[7] Miloš Kožina. *Cloud Big Picture and Building Blocks - End-User Documentation: cloud4retail - Dashboard*. Source placed on CD. 21st Aug. 2020. URL: `confluence/Cloud_Big_Picture_and_Building_Blocks-v6.png` (visited on 31/01/2021).

[8] André B. Bondi. "Characteristics of scalability and their impact on performance". In: *Proceedings of the 2nd international workshop on Software and performance*. WOSP '00. New York, NY, USA: Association for

Computing Machinery, 1st Sept. 2000, pp. 195–203.
ISBN: 978-1-58113-195-6. DOI: 10.1145/350391.350432. URL:
https://doi.org/10.1145/350391.350432 (visited on 12/02/2021).

[9] Molly Wojcik. *Scaling Horizontally vs. Scaling Vertically | Section.* 24th July 2020. URL: https://www.section.io/blog/scaling-horizontally-vs-vertically/ (visited on 31/01/2021).

[10] Ben Lutkevich and Alexander S. Gillis. *What is High Availability? - Definition from WhatIs.com.* SearchDataCenter. Mar. 2019. URL: https://searchdatacenter.techtarget.com/definition/high-availability (visited on 31/01/2021).

[11] Krishna Rungta. *What is Non Functional Testing? Types with Example.* 1st Jan. 2020. URL: https://www.guru99.com/non-functional-testing.html (visited on 12/02/2021).

[12] *Docker overview.* Docker Documentation. 28th Jan. 2021. URL: https://docs.docker.com/get-started/overview/ (visited on 31/01/2021).

[13] Cloud Native Computing Foundation. *What is Kubernetes?* Kubernetes. 1st Feb. 2021. URL: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/ (visited on 12/02/2021).

[14] Cloud Native Computing Foundation. *Kubernetes Components.* Kubernetes. 2021. URL: https://kubernetes.io/docs/concepts/overview/components/ (visited on 31/01/2021).

[15] *Definition of Master Data Management (MDM) - Gartner Information Technology Glossary.* Gartner. URL: https://www.gartner.com/en/information-technology/glossary/master-data-management-mdm (visited on 03/02/2021).

[16] Mozilla and individual contributors. *DevDocs - HTML.* 2020. URL: https://devdocs.io (visited on 31/01/2021).

[17] Janalta Interactive - Techopedia. *What is Java Swing? - Definition from Techopedia.* Techopedia.com. 2021. URL: http://www.techopedia.com/definition/26102/java-swing (visited on 12/02/2021).

[18] Andreas Pomarolli. *JavaFX Programming Cookbook*. 2016. URL: `https://jp.zlibcdn2.com/book/3428015/2bc89b` (visited on 28/04/2021).

[19] Laura Lemay and Charles Perkins. *Teach Yourself Java in 21 Days.* 1995, p. 516. ISBN: 9781575210971.

[20] Robin Nixon. *Learning PHP, MySQL & JavaScript: with jQuery, CSS & HTML5.* 4. ed. OCLC: 931718164. Beijing: O'Reilly, 2015. 780 pp. ISBN: 978-1-4919-1866-1.

[21] Refsnes Data. *JavaScript ES6.* URL: `https://www.w3schools.com/js/js_es6.asp` (visited on 31/01/2021).

[22] Sreemaha. *What is the difference between JavaScript and ECMAScript?* 25th Jan. 2018. URL: `https://www.tutorialspoint.com/What-is-the-difference-between-JavaScript-and-ECMAScript` (visited on 31/01/2021).

[23] Ecma International. "ECMAScript® 2019 Language Specification". In: 11 (July 2019), p. 764. URL: `https://262.ecma-international.org/11.0/` (visited on 29/04/2021).

[24] Microsoft Corp. *Typed JavaScript at Any Scale.* URL: `https://www.typescriptlang.org/` (visited on 31/01/2021).

[25] Microsoft Corp. *Quick Start Guide | Electron.* 18th Mar. 2021. URL: `https://www.electronjs.org/docs/tutorial/quick-start` (visited on 31/01/2021).

[26] Jolie O'Dell. *Why Everyone Is Talking About Node*. Mashable. 2021. URL: `https://mashable.com/2011/03/09/node-js/` (visited on 12/02/2021).

[27] P. Christensson. *CSS (Cascading Style Sheet) Definition.* 2006. URL: `https://techterms.com/definition/css` (visited on 12/02/2021).

[28] Jon Duckett. *HTML & CSS: design and build websites.* OCLC: ocn871305670. Indianapolis, Indiana: John Wiley & Sons Inc, 2014. 490 pp. ISBN: 9781118871645.

[29] Mark Otto. *Bootstrap v4.6.0*. Bootstrap Blog. 19th Jan. 2021. URL: `https://blog.getbootstrap.com/2021/01/19/bootstrap-4.6.0/` (visited on 12/02/2021).

[30] Monaca Inc. / Asial Corporation. *Getting Started*. Onsen UI. 2021. URL: `https://onsen.io/v2/guide/` (visited on 12/02/2021).

[31] Google LLC - Angular Team. *Angular - Introduction to the Angular Docs*. 2021. URL: `https://angular.io/docs` (visited on 31/01/2021).

[32] Inc. Facebook. *Getting Started – React*. 2021. URL: `https://reactjs.org/docs/getting-started.html` (visited on 31/01/2021).

[33] Igor Sarcevic. *Getting Started with Node.js and Mocha - Semaphore Tutorial*. Semaphore. 20th Feb. 2020. URL: `https://semaphoreci.com/community/tutorials/getting-started-with-node-js-and-mocha` (visited on 24/04/2021).