NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE

CLÁUDIA CARABINEIRO DA SILVA ROSALINO

Bachelor in Computer Science

COMPLEX VISUAL QUERYING WITHOUT SQL: MASHUP IN-MEMORY DATA AND PERSISTENT DATA

MASTER IN COMPUTER SCIENCE

NOVA University Lisbon
November, 2021

NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE

COMPLEX VISUAL QUERYING WITHOUT SQL: MASHUP IN-MEMORY DATA AND PERSISTENT DATA

CLÁUDIA CARABINEIRO DA SILVA ROSALINO

Bachelor in Computer Science

Advisers: Rui Nóbrega
Assistant Professor, NOVA School of Science and Technology
Teresa Romão
Associate Professor, NOVA School of Science and Technology

Co-adviser: Tiago Simões
Principal Product Designer, OutSystems

Examination Committee:

Chair: Ricardo João Rodrigues Gonçalves
Assistant Professor, NOVA School of Science and Technology

Rapporteur: Carlos Alberto Pacheco dos Anjos Duarte
Assistant Professor, FCUL

Adviser: Rui Pedro da Silva Nóbrega
Assistant Professor, NOVA School of Science and Technology

MASTER IN COMPUTER SCIENCE

NOVA University Lisbon
November, 2021

Complex Visual Querying without SQL: Mashup in-memory data and persistent data

ACKNOWLEDGEMENTS

I would like to thank my thesis advisers, Rui Nóbrega (Assistant Professor) and Teresa Romão (Associate Professor), and also my co-adviser, Tiago Simões (Principal Product Designer), for guiding me through this final step of my master's degree with such exceptional expertise.

I would also like to thank NOVA School of Science and Technology and OutSystems for this collaboration and opportunity. To my friends and colleagues, I would like to express my gratitude for sharing with me great achievements and also the greatest difficulties.

Finally, a special thanks to my grandparents, mother and brother, who unconditionally supported me through my academic path.

ABSTRACT

This dissertation has the intent of increasing the level of expressiveness of a visual interface that allows users to query data without resorting to textual query languages. Although it was in the 1970s that Relational Database Management Systems appeared, the standard way to interact with them remains to be through SQL, a textual query language.

One of the main problems with these kinds of languages is that they require technical skills and knowledge of query language, syntax and domain schema. Consequently, database management was considered only accessible to experienced users for a long time. In recent years, with the explosion of the Web, the volume of data available to everyone grew exponentially. This way, it became necessary to make data retrieval accessible not only to expert users but also to users without database knowledge. Visual Query Systems emerged in the late 1970s to hide the complexity of query languages behind a visual interface and improving the effectiveness of human-computer communication. Since then, many different approaches have been proposed and studied. The OutSystems platform provides a graphical query interface called Aggregates that allows its users to formulate queries through the manipulation of visual components. However, this tool does not yet support the same level of expressiveness as SQL.

This dissertation aims at increasing the level of expressiveness of the Aggregates by proposing different solutions for the implementation of the IN and NOT IN clauses without compromising the global experience for any kind of user. In order to achieve this, an iterative development process was used, including the design, implementation and evaluation of prototypes. In this dissertation, we present a functional solution, integrated into the OutSystems platform. The results show that we were able to turn the filtering of persistent data by in-memory data very accessible to OutSystems Developers and also to regular Developers with no or very little experience using the OutSystems platform.

Keywords: Visual Querying, Visual Query Systems, Low-Code Development, Data Visualization, Human-Computer Interaction

RESUMO

Esta dissertação tem como objetivo aumentar o nível de expressividade de uma interface visual que permite aos seus utilizadores consultar dados sem recorrer a linguagens de consulta textuais. Apesar de ter sido na década de 1970 que surgiram os Sistemas de Gestão de Bases de Dados Relacionais, a forma mais usual de interagir com esses sistemas continua a ser através de *SQL*, uma linguagem textual de consulta de dados.

Um dos principais problemas da utilização deste tipo de linguagens é que requerem habilitações técnicas e conhecimento sobre linguagens de consulta, sintaxes específicas e esquemas de domínio. Consequentemente, a gestão de bases de dados foi, durante muito tempo, considerada apenas acessível a utilizadores experientes. Recentemente, com a explosão da *Web*, a quantidade de dados disponíveis para toda a gente cresceu exponencialmente. Deste modo, tornou-se necessário tornar o acesso a esses dados possível tanto para utilizadores experientes como para utilizadores sem conhecimentos de bases de dados. Os Sistemas Gráficos de Consulta de Dados emergiram no final da década de 1970 com o objetivo de ocultar a complexidade das linguagens de consulta por detrás de uma interface visual e melhorar a eficácia da interação pessoa-máquina. Desde então, muitas abordagens diferentes foram propostas e estudadas. A plataforma da *OutSystems* fornece uma interface gráfica de consulta chamada *Aggregates* que permite aos seus utilizadores formular consultas através da manipulação de componentes visuais. No entanto, esta ferramenta ainda não suporta o nível de expressividade do SQL.

Esta dissertação visa, assim, aumentar o nível de expressividade dos *Aggregates*, propondo diferentes soluções para a implementação das cláusulas *IN* e *NOT IN* sem comprometer a experiência global para qualquer tipo de utilizador. Para tal, foi utilizado um processo de desenvolvimento iterativo, incluindo a concepção, implementação e avaliação de protótipos. Nesta dissertação apresentamos uma solução funcional, integrada na plataforma OutSystems. Os resultados mostram que fomos capazes de tornar a filtragem de dados persistentes por dados em memória muito acessível tanto para programadores OutSystems como para programadores regulares com nenhuma ou muito pouca experiência de uso com a plataforma OutSystems.

Palavras-chave: Consulta Visual de Dados, Sistemas Gráficos de Consulta de Dados, Desenvolvimento *low-code*, Visualização de Dados, Interação Pessoa-Máquina

CONTENTS

List of Figures

List of Tables

ACRONYMS

ANSI American National Standards Institute 1

API Application Programming Interface xi, 51, 52

DBMS Database Management System 1

HCI Human-Computer Interaction 6, 8

IOS International Organization for Standardization 1

QBD* Query-By-Diagram 23, 24

QBE Query-By-Example 18, 21

RDBMS Relational Database Management System 1

REST Representational State Transfer xi, 51, 52

SQL Structured Query Language x, xi, 1, 2, 3, 14, 19, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 34, 40, 41, 42, 46, 48, 53, 54, 55, 56, 57, 63, 66, 68, 72, 73

VQS Visual Query System 2, 3, 18, 19, 21, 23, 24, 25, 27

1

INTRODUCTION

In an era of application development where databases are accessed through specific and standardized languages, far from natural language, any low-code platform faces the challenge of making access to databases accessible to non-experienced users.

It is increasingly important to develop applications that can be easily integrated into various platforms, extensible, customizable to the needs of different customers and, above all, faster to be developed. The Information Technology (IT) industry presents a demand in human resources that is superior to the offer that IT teams can support. This has led to continuous growth of the low-code market, with more and more companies recognizing that it can be a great help to accelerate the development process and can bring several benefits [38]. As it is generally known to any programmer or anyone related to the Information Technology sector, practically in any application development, regardless of the sector with which it is related, it is necessary to access databases to store and/or obtain information. This is where tools that aim at making that access more visual became very interesting in the context of the low-code market.

1.1 Context

It was in the 1960s that the first Database Management Systems (DBMSs), such as IMS and IDS [16], appeared. However, they used heterogeneous data structures, with each application storing its data in a single structure. Therefore, when another application needs to use the same data, it was necessary to study and know this structure well.

To solve this problem, Relational Database Management Systems (RDBMSs) emerged, providing a uniform way of representing and accessing data that could thus be used by any application. The process of accessing these databases can be done through queries formulated in Structured Query Language (SQL), which was developed by IBM in the early 1970s and which is currently recognized as an official standard by the American National Standards Institute (ANSI) and the International Organization for Standardization (IOS). Although structured query languages such as SQL are quite expressive, they require an array of technical skills and knowledge on query language, syntax, and domain schema, which continues to restrict its use to advanced

users [48].

With the exponential growth of data available on the web for everyone, it has become crucial to have ways that allow ordinary users to be able to search and access that data. The concept of Visual Query System (VQS) has been growing from this need and this type of systems were initially defined as "systems for querying databases that use a visual representation to depict the domain of interest and express related requests" by Catarci *et al.* [6]. The main purpose of this visual approach is to make data querying accessible to non-experienced users that don't have a base knowledge of data querying languages, but expert users can also potentially benefit from visual query formulation as it leads to ease in learning, high-efficiency rate, reduced error rate, and more satisfactory experiences [48]. This way, when developing a visual querying interface it is important to have in account all types of users.

As mentioned by Sourav S. Bhowmick [4], since the appearance of Database Management Systems the academic and commercial efforts have been mostly focused on textual approaches for querying data.

The main problem underlying this paradigm is that it limits its usage to expert users with high knowledge of specific languages. Querying databases has become a daily task for many end-users that don't have specific skills for it, and the development of visual interfaces to data can broadly expand the audience for databases. However, most of the existing visual querying interfaces lack both visual expressiveness and/or ease of use, which is an essential trade-off [5].

1.2 Motivation

Low-code is a software development approach that enables the delivery of applications faster and with minimal hand-coding. This dissertation was elaborated together with Outsystems, which is currently one of the main low-code development companies in the world, having been founded in Lisbon in 2001. OutSystems platform allows users to create complete applications visually using a drag-and-drop interface composed by:

- A **visual IDE** that allows users to define UIs, workflows and data models to create an application with the option of adding hand-written code;
- **Connectors to various back-ends or services** that automatically handle data structures, storage, and retrieval;
- **Application lifecycle manager** with automated tools to build, debug, deploy, and maintain the application in test, staging, and production.

In order to allow users to retrieve data from databases, OutSystems provides components called Aggregates that allow this process to be done through visual interactions. Using Aggregates any type of user is able to apply filters, join entities, sort or make aggregation operations over the data in a faster and much more intuitive way than using a textual query language like, for example, SQL. The aggregates also provide a section with a table-based interface that allows

the user to see the query output produced while manipulating the data. The level of expressiveness of the Aggregates already covers most of the basic use cases, but it is still very compromised when compared to the level of expressiveness of structured querying languages. One of the limitations is that it is not possible to intuitively filter persistent data from the database by an in-memory list. In this thesis, we aimed at extending the visual query platform to enable this. However, when trying to increase the expressiveness of a VQS it is very important to have in mind the trade-off between expressiveness and usability. The goal was to increase the expressiveness as much as possible, without compromising its usability. Complex semantics expressed with visual constructs and operations may result in inefficiency for expert users and difficulty in learning and use for users with low database querying experience [48].

Although the OutSystems Aggregates already allow users to perform many basic query operations as mentioned above, there are still some more advanced SQL functionalities that can't be performed this way. OutSystems platform offers an alternative for users to perform SQL queries to compensate for the lack of expressiveness that still exists in the Aggregates, but this results in the inexistence of a complete and uniform visual querying platform. At the time that this dissertation started, some examples of functionalities that were still not supported by the Aggregates were: IN, NOT IN, EXISTS, NOT EXISTS, DISTINCT, UNION and the possibility to perform subqueries.

As there were still many functionalities that needed to be implemented, after looking at already existing statistics regarding the percentages of use of each of those operations in SQL queries (see Annex I for more details), seeing the Outsystems Community forum most voted ideas and some discussion with the OutSystems stakeholders, it was decided that this work would focus on the IN and NOT IN clauses. These clauses are particularly important because they demonstrate the inexistence in the Aggregates of a proper way of filtering a database with an in-memory list of values. There are many possible designs for adding this functionality and that's why before implementing the final one we needed to explore and evaluate them.

This way, the main motivation for this thesis was to find the best design for the integration of the IN and NOT IN clauses in the Outsystems aggregates and implement it in order to increase its expressiveness and provide an improved visual querying platform that results in a better user experience.

1.3 Research Questions

This dissertation focused on trying to answer the following main broader question:

Can we make OutSystems Aggregates more expressive with the IN and NOT IN clauses without compromising the global experience for any kind of user?

In order to answer this question, it was necessary to divide the main question into the following more specific questions:

Research Question 1: Is intersecting an in-memory list with the results of a database query a common problem?

In order to start defining a solution, it was very important to research how the users currently implement the IN and NOT IN clauses and how often it happens to guarantee that we were answering the right needs of the Outsystems platform users. There is no point in developing any solution that doesn't add any real value.

Research Question 2: Are there other solutions to this problem that can solve more use cases?

Although there was a main problem that we intended to solve by adding the IN clause in the OutSystems Agreggates, if there could be found a solution that could solve more use cases than the ones already identified then it needs to be considered.

Research Question 3: Is it possible to augment the expressiveness of OutSystems Aggregates for this use case in a way that is natural for experienced developers?

When adding new features to an already existing and working platform, we wanted to make sure that we didn't ruin the experience for the already experienced users of it.

Research Question 4: Is it possible to augment the expressiveness of OutSystems Aggregates for this use case while making it accessible to non experienced users?

When adding new features to an already existing and working platform, we wanted to make sure that we are not increasing the learning curve substantially and that new users could intuitively find and learn how to use the newly added features.

1.4 Contributions

The work presented in this thesis includes a study of the current state of the Aggregates, considering the nonexistence of an effective way of intersecting in-memory lists with persistent data accessible by Aggregates, and a study of how OutSystems users currently overcome this limitation, along with a research of their complaints and requests related to it.

The main contribution was the design and evaluation of solutions that allow augmenting the expressiveness of the OutSystems Aggregates, without compromising the experience for

experienced and non-experienced users. This was firstly done using low fidelity prototypes and performing usability tests. In the end, we implemented a final prototype, integrated into the Service Studio, in order to evaluate with high fidelity the progress achieved.

This way, the final contribution was an evaluated and integrated prototype that allowed us to conclude if the level of expressiveness of the Aggregates was improved when compared to the previous state and if it was improved without compromising the global user experience.

1.5 Document Structure

The remainder of this document is organized as follows:

- **Chapter 2** - Background: Introduces base concepts that were used in the elaboration of this work and provides insight about the OutSystems Platform and technology;
- **Chapter 3** - Related Work: Presents an overview of Visual Query Systems evolution;
- **Chapter 4** - Requirements and Methodology: Summarizes the process used to understand the problem and defines how to solve it and validate solutions;
- **Chapter 5** - Initial Platform Evaluation: Analysis the current state and the proposed alternative approaches;
- **Chapter 6** - Iterative Design, Implementation and Evaluation: Describes all the phases of the solution conception;
- **Chapter 7** - Conclusion and Future Work: Includes the final remarks and defines what should be addressed in future work.

2

BACKGROUND

This chapter introduces base concepts that were used in the elaboration of this work and gives insights about the OutSystems platform. In particular, this chapter gives an overview of the OutSystems Aggregates, which are the main components addressed in this thesis.

2.1 Human-Computer Interaction

The base concepts of Human-Computer Interaction (HCI) were well established in the 1980s but it has really expanded since then. As technology rapidly evolved, the interactions established between people and computers increased. Consequently, the way those interactions are made and who performs them has changed. We have come to a point where we want to design "experiences for all manner of individuals (and not just users), in all manner of settings, doing all manner of things" [42].

HCI aims to design interfaces that fit between the user, the machine and the required services, in order to achieve a certain performance both in quality and optimally of the services. There are two main terms that should be considered when talking about HCI: functionality and usability. Functionality refers to the actions or services that a system allows its users to perform. Usability refers to the "extent to which a system, product or service can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use" [40]. The effectiveness of a system is, therefore, measured by the balance between these two concepts [19].

2.1.1 Iterative Development

Before the arising of agile software development practices, the models that existed for the development of software were sequential and didn't include user feedback throughout the process. Those models are now called traditional models, being the most famous one the Waterfall model, and their implementation begins with the elicitation and documentation of a "complete" set of requirements, followed by architectural and high-level design, development, and inspection [10].

At the beginning of the 1990s, developers began to find these initial steps frustrating and the fact that technology began to evolve very fast made requirements change very frequently. These, together with the fact that users started to find it increasingly hard to state their needs upfront and that users' expectations of software continue to rise, gave origin to the Agile Methods. The core idea behind these methods is iteration based on feedback. Their focus is on extensive communication, rapid iteration, continuous collecting of information and continuous adjustments, instead of trying to plan the entire process right on the start [45].

Iterative development is an agile methodology and was defined by Larman as "An approach to building software (or anything) in which the overall lifecycle is composed of several iterations in sequence. Each iteration is a self-contained mini-project composed of activities such as requirements analysis, design, programming, and test. The goal for the end of an iteration is an iteration release, a stable, integrated and tested partially complete system." [21].

Iterative development deals well with change, as the only complete requirements necessary are for the current iteration (the requirements for the next iteration are provisional). Consequently, this approach allows achieving flexibility in technology and requirements change.

2.1.2 Requirements Analysis

In order to know what is wanted from a system, one has to start by understanding who will use it (user profile) and which functionalities will they want to perform on it (task analysis):

User profile - There isn't a user interface style or approach that best suits all types of users. Specific interface design alternatives that optimize the performance of some types of users may actually degrade the performance of other types of users. For example, for an infrequent/casual user, it is more important that the interface is easy-to-learn and easy to remember, but for a high frequency/expert user it is more important that the interface is efficient, powerful, and flexible [10].

This way, it is necessary to know the classes of people who will use the system. By knowing the users' work experience, educational level, age, previous computer experience, and so on, it is possible to anticipate their learning difficulties to some extent and to better set appropriate limits for the complexity of the user interface. The users' work environment and social context also need to be known. Some possible methods for collecting this information are market analysis, observational studies and questionnaires/interviews [27].

Task analysis - Allows involving the entire team in understanding the users of the system. It provides ways to organize the usually unstructured data that is retrieved from field studies or site visits. It is an essential part of the process of creating any product, as they are tools for users to accomplish goals by performing tasks [10].

It consists in elaborating a list of all the functionalities that the users will want to achieve with the system (the goals), the preconditions, the steps that need to be taken and interdependencies between them, all the outputs and reports that need to be produced, the criteria used to determine the quality and acceptability of these results, and the communication needs of the users [27].

2.1.3 Prototyping

As we discussed in previous sections, HCI is a user-centered and iterative field. The user is placed at the center of the user-centered design process from the first phases of analysis of user requirements until the last phases of testing and evaluation. Prototypes support this goal by allowing users to experience and evaluate a system before it is built. This way, designers are able to identify earlier the functional requirements, usability issues and performance problems. Once the identification is made, the designer can change the design accordingly in order to obtain an improved design. Prototypes also allow the evaluation of concrete representations of design ideas, revealing the strengths and weaknesses of each one. In iterative design, implementation and testing phases are made several times, allowing for continuous improvement [45]. Prototyping can be defined in four steps [14]:

1. **Functional Selection -** The first step is to choose which functions the prototype should exhibit. These should be the ones that are based on the most relevant work tasks and that can serve as model cases for demonstration. The range of features offered by a prototype can define it as "vertical prototype" or "horizontal prototype". In vertical prototyping, the range of features is lower but are implemented in more detail. In horizontal prototyping, the range of features is higher but are not implemented in great detail. These two approaches can be combined in the same prototype as needed;

2. **Construction -** When developing a prototype, the effort needed should be much smaller than the effort required to develop the final product. This can be achieved by making a proper functional selection and through the use of suitable tools and techniques. When constructing a prototype, the focus should be on the intended evaluation;

3. **Evaluation -** It's the most important step since it provides the feedback needed for the further development process. Evaluation should include users from all the relevant groups and follow explicit documented criteria for the evaluation. It should also specify the steps to be performed with the system, taking into account the context in which those steps will be made;

4. **Further use -** Depending on the type of prototype developed and on the available production environment, the prototype may merely serve as a learning vehicle and be thrown away after it serves that purpose or it can be used fully or partially in the development of the final system.

In the context of usability testing, prototypes can be low fidelity, high fidelity or somewhere in between. Fidelity refers to the number of features implemented, the detail in the implementation of those features, the look/aesthetics of the prototype, how users will work with it and how close it is to the final product [45].

Low-fidelity prototypes are usually easy and fast to develop and contain a high level of interactivity. An example of this kind of prototype is a paper prototype. Doing usability tests

with this kind of prototype requires a team member that knows the products very well, to act as a facilitator according to the participant's input. As they are quick and easy to build, they allow starting usability testing early in development. This way, the team is able to fix design problems while it is still cost-effective. Another advantage of using low-fidelity prototypes is that they facilitate the think-aloud process. However, they may discourage exploration of the product, be difficult to use by people with disabilities, difficult to collect quantitative data, not easy to integrate with remote testing tools or even lead to a wrong idea of how obvious the changes will be when made by a computer.

High-fidelity prototypes contain more functionalities and are more expensive to develop than low-fidelity ones. In these prototypes, the interaction between the user and the prototype is equal to the one that would be with the final product. These provide better documentation of the product but can deceive the managers and users regarding the level of completion of the project.

It is worth mentioning that a prototype may be low fidelity concerning a certain aspect and medium or high fidelity concerning another aspect. For example, paper prototypes are low fidelity in look but they may be high fidelity in-breath, number of features implemented. When it comes to deciding which type of prototype to use, low-fidelity prototypes are considered most useful at the beginning of development to address high-level conceptual issues. On the other hand, high-fidelity prototypes are considered most useful later in development, after more design decisions have been worked out. It is common to use both types of prototypes throughout the process.

2.1.4 Usability Testing

Usability testing was introduced in the late 1980s and is currently a widely used technique to evaluate user performance and acceptance of products and systems [50]. There are six fundamental characteristics that make a usability test valid [45]:

- **The focus is on usability:** The purpose of the test is to allow its participants to talk about their reactions to the test session;

- **The participants are end-users or potential end-users:** The participants must be part of the targeted market of the product. To find the right users one can make a user profile. This should take into account two types of characteristics: characteristics that the users share and characteristics that make a difference among the users. There are usually many relevant groups for testing and that's why the test team should decide which are the most relevant in order to allocate the resources the best way;

- **There is a product or a system to evaluate:** Usability tests can be performed with almost any type of product or technology;

- **The participants perform tasks, usually while thinking aloud:** When executing a test, the test administrator should incentivize the participant to say out loud what he is thinking while performing the test and interact with him. The participant should be told that the test is about the usability of the system and not about him. Selecting tasks is one of the main requirements of every usability test, as when performing a test the time is limited and the selected tasks will define the scope of the test. The tasks selected should have one of the following characteristics: frequently performed on the system; important for accomplishing basic goals on the system; critical for other tasks; include tasks that probe areas where usability problems are likely; include tasks that probe the components of a design; be central to the business goals for the product; have been redesigned in response to the results of a previous test; be new to the product. Estimating the time for each task is also important in order to determine how many tasks to include in the test. The tasks are usually presented to the participant as task scenarios. Test scenarios try to simulate an eventual normal use of the system. The order in which these are made may be important, as they might have precedence between them. Assisting the participant with the tasks is also a common practice;

- **The data is recorded and analyzed:** A usability problem usually affects more than just one measure. The tester should observe the problems during the sessions and record them, in order to be able to analyze them later in order to identify the basic causes. The problems found should be classified according to their severity in order to know which should be solved and in which order. Positive issues can also be identified and noted;

- **The results of the test are communicated to appropriate audiences:** Presently this is done more informally because organizations started to accept this kind of test with users as a valid, useful tool and an important part of their regular development process. The results can be communicated for example at a meeting, and there is no need to tell all the details of the tests and procedure, as the organizations just want to know what problems they encountered and what needs to be done about them.

2.1.4.1 Estimating the Number of Subjects

Thinking aloud methods with potential users has been employed by usability engineering experts as one of the main methods for improving interfaces through user testing.

Nielsen performed studies in order to find the number of test subjects that results in the best benefit/cost ratio for finding usability problems in an interface, and he claims that that number is 5 users [26]. He also says that as the goal of this kind of testing is to identify problems that will be corrected in future interactions, it is not worth to use more than 5 subjects since the additional value falls exponentially from there, as can be seen in Figure 2.1.

This way, Nielsen concludes that it is better to use fewer subjects and run more phases of testing [28], because it is very expensive to find all the problems and the next interactions will

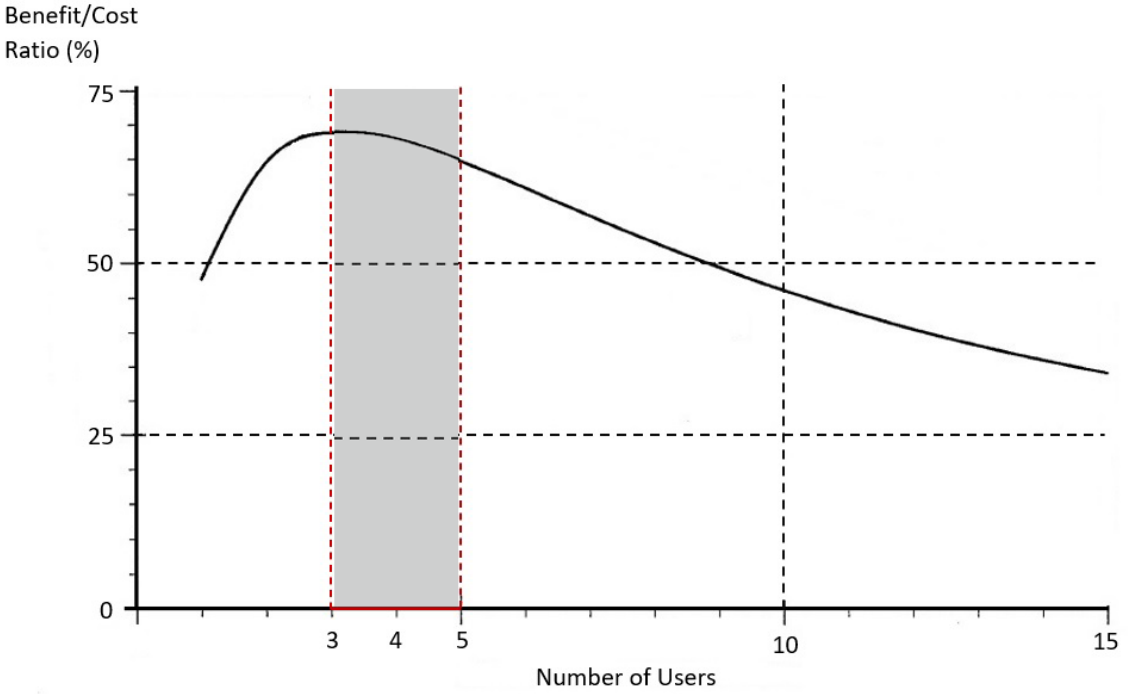


Figure 2.1: Ratio between benefits and costs for using various number of test users to find usability problems in a medium-large software project [29]

probably introduce some new problems. Besides that, the most important usability problems are likely to be the ones that are firstly found [25].

2.2 OutSystems

OutSystems is a low-code application delivery platform that aims to accelerate the delivery of mobile and web applications. With the OutSystems platform, even users with little programming knowledge can use a single, integrated development environment that covers the entire development lifecycle to create an application. This is made possible because the OutSystems platform offers a visual programming language that promotes high levels of abstraction [18, 36].

2.2.1 Architecture

An overview of the OutSystems architecture including its components and tools [35] can be seen in Figure 2.2. The main components that can be seen are:

- **Platform Server:** Consists in a set of servers that compile, deploy, manage, run, and monitor the applications inside your infrastructure. You can connect to the platform server from Service Studio, which is the OutSystems development environment for Web and Mobile Apps and will be explained in more detail in Section 2.2.2. Once it is connected, developers can create and publish applications to the platform server. Each version of those applications will be stored in the Platform Data database. The platform server

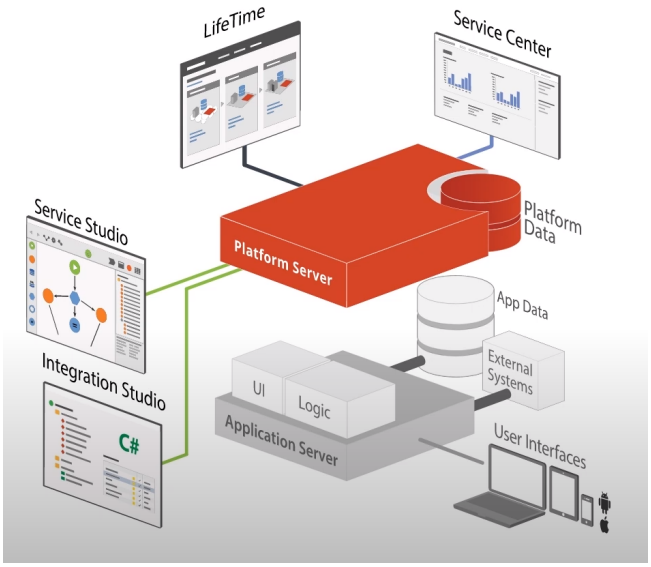


Figure 2.2: OutSystems Platform Architecture [35]

will then compile and generate optimized code for those applications and deploy them to a standard application server. The application server uses traditional databases and external systems to run the applications you've created.

- **Integration Studio:** Development environment that allows the users to create extensions to the platform itself. Integration Studio provides several accelerators to integrate with external resources such as C# code and databases. Integration Studio takes those resources and creates representations of them inside the OutSystems world. Once those representations exist inside an extension, they can be published to the server and used inside Service Studio as normal OutSystems resources.

- **Service Studio:** Service Studio is the visual development environment for developers to create great Mobile and Web Applications. In the following section it will be described in more detail.

2.2.2 Service Studio

The low-code and visual development environment of the OutSystems platform is called Service Studio (see Figure 2.3). It allows developers to create applications and Modules on the server.

On the Application Layer Tabs (on the top-right side of the screen as can be seen in Figure 2.3) users can select one of the following four application layers [33] and manipulate it:

- **Processes layer:** This layer gives us a lot of information about logic and tasks that can occur at the highest level. Processes have two major groups of elements: Processes and Timers. Processes include business processes and human and automated tasks which can also have decisions, events and waits. Timers are scheduled actions that can occur at

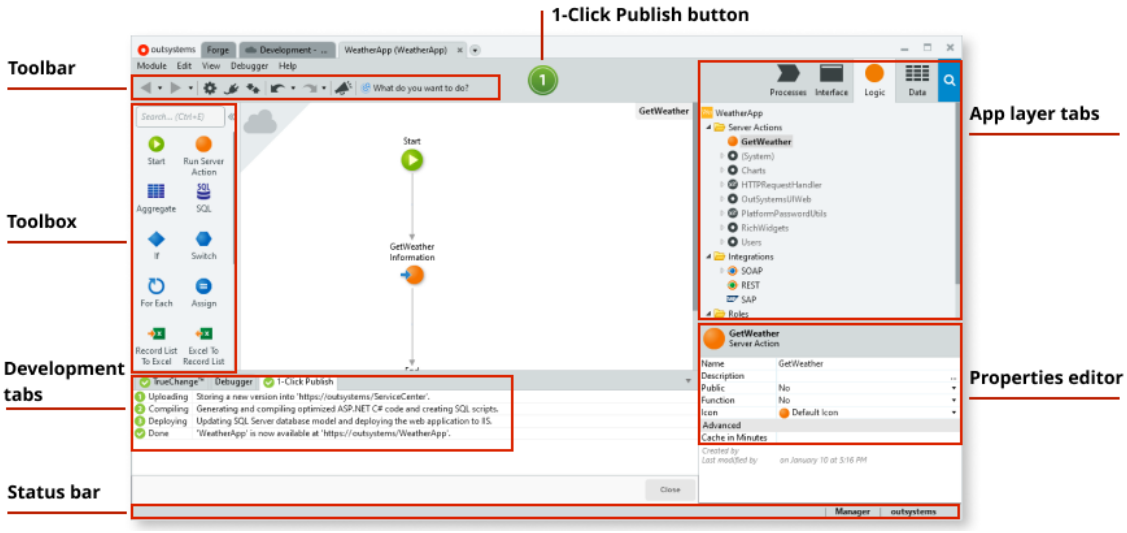


Figure 2.3: Service Studio Interface [37]

a specific time and can be rescheduled to occur periodically. These timers can be given different priorities and can also have timeouts.

- **User Interface layer:** Focuses on the different components that will make up the user interface. In this layer we can manage groups of screens and blocks (which are called User Interface Flows). For each screen we can edit, remove or add variables, visual elements and actions. In this layer we can also assign the elements to action flows that will define the client-side logic of the screen and define the flow between screens.

- **Logic layer:** It is where we define the application logic that will run on the client or on the server. Client Actions run on the client-side, be that a mobile device or a web browser, while Server Actions will run on the server. This layer includes logic elements that allow the users to integrate with external systems, roles that can be assigned to users in order to know exactly which users have access to which resources and exceptions that can be thrown or handled. It's also in this layer that the user can add the actions related to data retrieval.

- **Data layer:** Inside the data layer, we can define the different Entities that would be available in the database or locally on the device storage, which we call local storage. If you wish to visually represent these and be able to see them, you can create Entity diagrams. In short, here the user can define the data model of the application.

2.2.3 Aggregates

As is common knowledge, most applications need to fetch data from a database and the applications made with the OutSystems platform are no exception. Given that the whole purpose of OutSystems is to make application development accessible to everyone, finding a way that

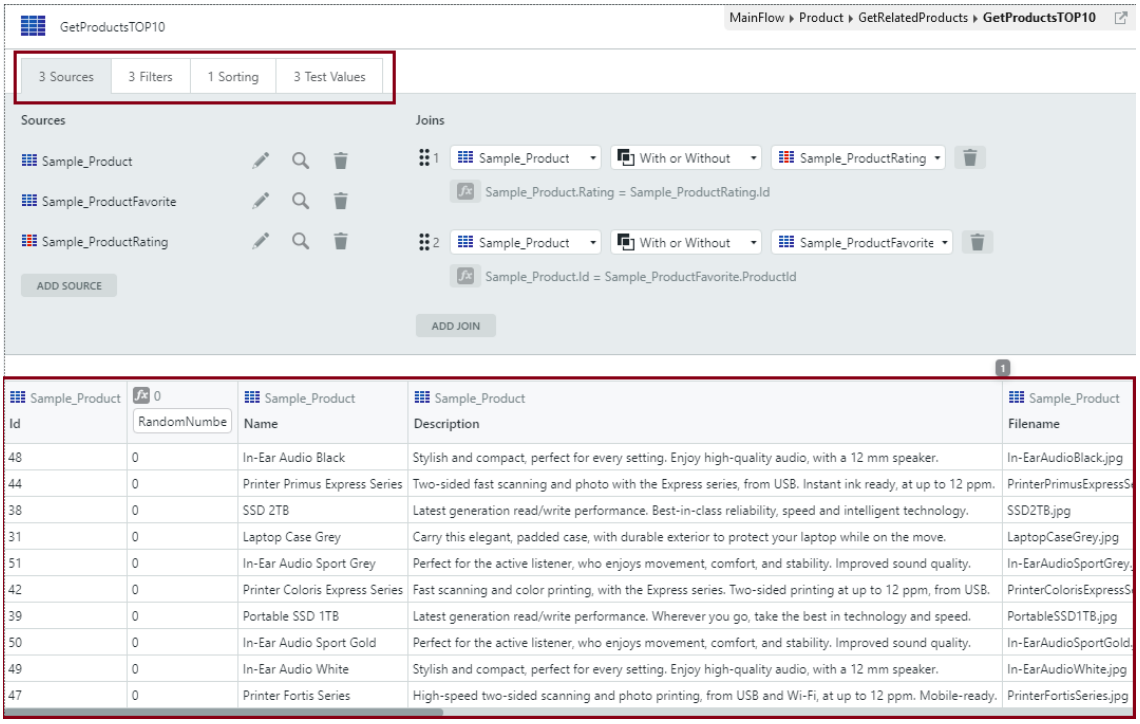


Figure 2.4: Aggregates Editor

would allow non experienced users to query the data was needed. That's why OutSystems created a visual element called Aggregates that can be run in the server-side logic or in the client-side logic [31]. The Aggregates make it possible for users to create queries in a visual way, without the need to have databases and structured query languages knowledge.

In Figure 2.4 we can see the Aggregates editor default opening presentation. The editor has two main sections: a top menu where it is possible to define the source entities, filter data according to some criteria, sort the data as needed or even test the query output; a preview pane of the query output data presented with a style similar to the Excel spreadsheets with the purpose of making it more familiar. Considering the top menu section it offers the following functionalities [30, 32]:

- **Sources:** Expanding this tab the user can reveal the source entities (Figure 2.5). The sources within the Aggregate determine from where the data is retrieved. Aggregates support more than one entity as source. In this tab the user can add, edit or remove entities from the query and even establish the join relationship between them. At the current state there are three types of join available: "only with", "with or without", and "with". The respective joins in SQL are: "inner join", "left join", and "full outer join". Each of these types is represented with a icon based on Venn Diagrams to make them more intuitive.

- **Filters:** Once we have the sources of the Aggregate defined we might want to define the Aggregate filters (Figure 2.6), which are similar to the SQL WHERE clause. As you add

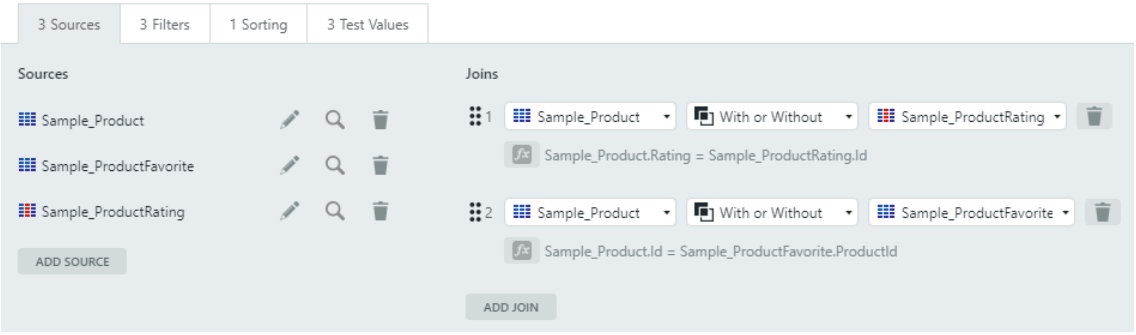


Figure 2.5: Aggregates Sources

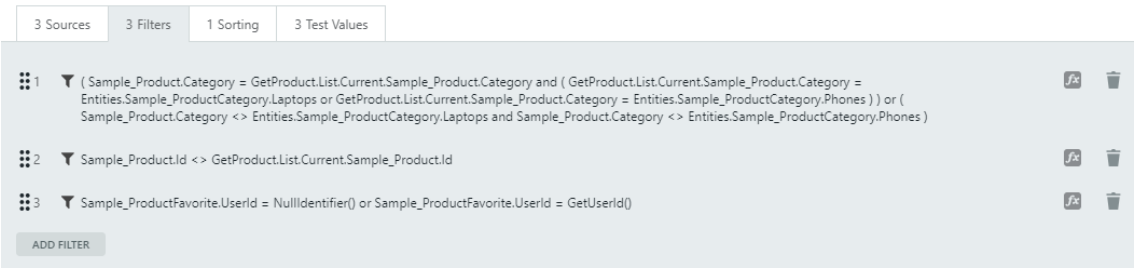


Figure 2.6: Aggregates Filters

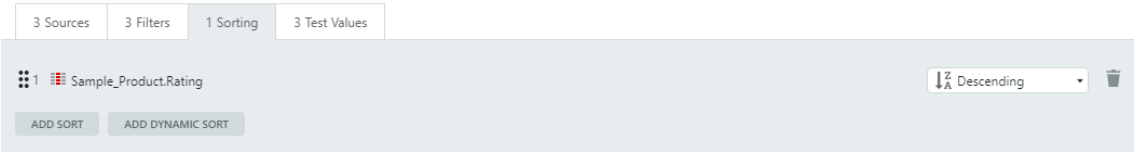


Figure 2.7: Aggregates Sorting

filters, the preview data automatically refreshes and displays only the records that match all filters. Multiple filters may be added, and the records returned are only the ones that evaluate to "true for all filters". Each filter usually has some sort of logical operator, such as equal to, not equal to, and so on. It is also possible to use some of the built-in functions to manipulate dates or other values. This may be useful to get records whose date is in the future, for instance.

- **Sorting:** In the sorting tab the user can choose the entity that he wants to sort and add one or more sorting criteria (Figure 2.7). For example the sorting can be done Ascending or Descending. Multiple sorts can be added. The order in which the sorts are defined is important, as it will influence the Aggregate result, with the first one having more precedence, followed by the second one, and so on.

- **Test Values:** The user can also set test values that don't affect what will be actually running inside the application but will allow him to see what the output might be in a particular case. So if the user gives the input field a value, it will evaluate the query and refresh the data preview.

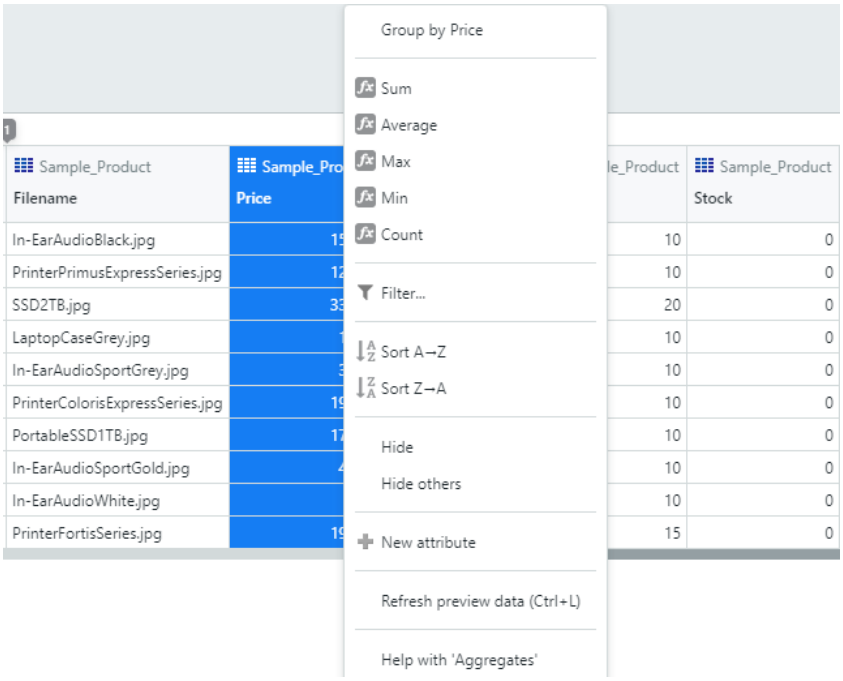


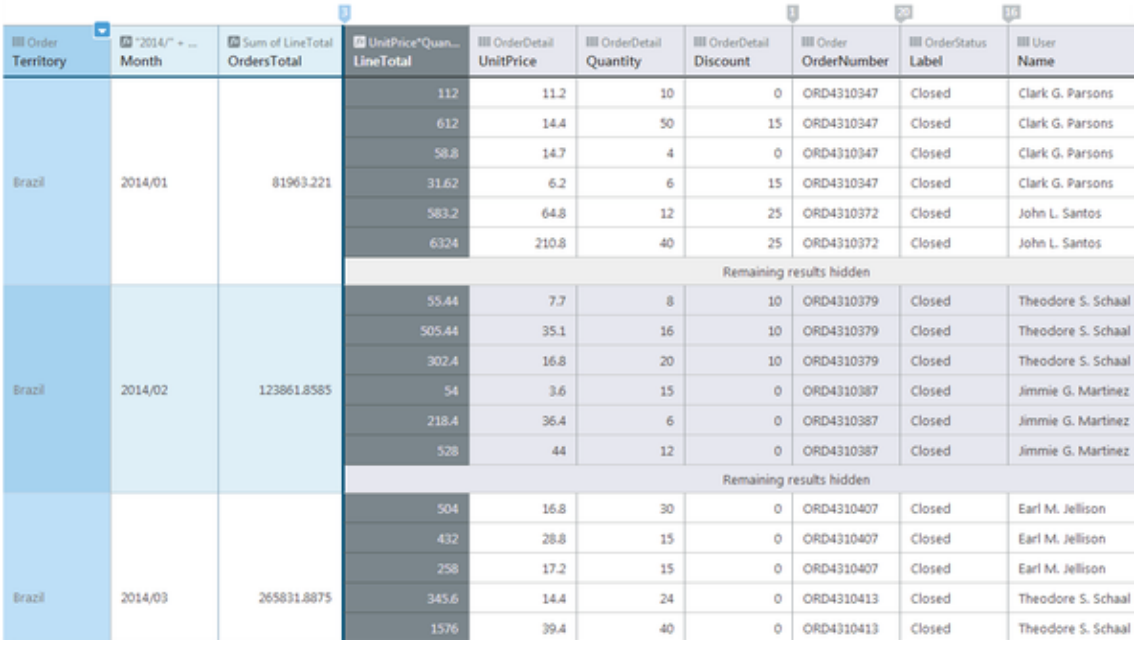
Figure 2.8: Aggregates Preview Pane Functionalities

Considering the query output previewing pane, there are also some functionalities that allow the user to add to its query visual formulation by selecting columns with a right-click or by clicking on “New Attribute”.

As can be seen in Figure 2.8, if a user performs a right-click on a column, a list of options is presented. In order to make the output preview more understandable, the options “Hide” and “Hide others” allow hiding columns for previewing purposes. This is particularly important when there are a lot of attributes in the previewer, to help the user visualize the most relevant ones. Hiding columns do not affect the output of the Aggregate at runtime, unlike the rest of the options presented. In the list, there are also options that perform aggregation on the rows fetched by the Aggregate. The available aggregation functions are Group, Sum, Average, Max, Min and Count.

Aggregates also support calculated attributes by selecting “New Attribute”. Sometimes there’s information that’s not directly stored in the database, but with some combination of attributes or calculations one can easily get that information. So, calculated attributes are custom values that can be computed from other attributes in the Aggregate, creating a new column in the output of the Aggregate. These calculated attributes can be created via Expressions, which have access to all the attributes in the Aggregate, as well as OutSystems built-in functions and variables accessible by the Aggregate.

In Figure 2.9 we can see that when grouping rows together or using the aggregation functions, only the Aggregated columns of the Aggregate will be part of the output. They can be easily identified by the columns in blue.



Order Territory	"2014/" + ... Month	Sum of LineTotal OrdersTotal	UnitPrice*Quan... LineTotal	OrderDetail UnitPrice	OrderDetail Quantity	OrderDetail Discount	Order OrderNumber	OrderStatus Label	User Name
			112	11.2	10	0	ORD4310347	Closed	Clark G. Parsons
			612	14.4	50	15	ORD4310347	Closed	Clark G. Parsons
			58.8	14.7	4	0	ORD4310347	Closed	Clark G. Parsons
Brazil	2014/01	81963.221	31.62	6.2	6	15	ORD4310347	Closed	Clark G. Parsons
			583.2	64.8	12	25	ORD4310372	Closed	John L. Santos
			6324	210.8	40	25	ORD4310372	Closed	John L. Santos
			Remaining results hidden						
			55.44	7.7	8	10	ORD4310379	Closed	Theodore S. Schaal
			505.44	35.1	16	10	ORD4310379	Closed	Theodore S. Schaal
			302.4	16.8	20	10	ORD4310379	Closed	Theodore S. Schaal
Brazil	2014/02	123861.8585	54	3.6	15	0	ORD4310387	Closed	Jimmie G. Martinez
			218.4	36.4	6	0	ORD4310387	Closed	Jimmie G. Martinez
			528	44	12	0	ORD4310387	Closed	Jimmie G. Martinez
			Remaining results hidden						
			504	16.8	30	0	ORD4310407	Closed	Earl M. Jellison
			432	28.8	15	0	ORD4310407	Closed	Earl M. Jellison
			258	17.2	15	0	ORD4310407	Closed	Earl M. Jellison
Brazil	2014/03	265831.8875	345.6	14.4	24	0	ORD4310413	Closed	Theodore S. Schaal
			1576	39.4	40	0	ORD4310413	Closed	Theodore S. Schaal

Figure 2.9: Aggregates Output Preview with Aggregation Functions [39]

3

Related Work

In this chapter, an overview of Visual Query Systems types and evolution is presented. In the end, an analysis is made of the current use of Visual Query Interfaces in commercial tools.

3.1 Visual Query Systems

Visual Query Systems (VQSs) were defined by Catarci *et al.* in 1997 as "systems for querying databases that use a visual representation to depict the domain of interest and express related requests" [6]. Different types of visual interfaces have been proposed and evaluated since then, but the problem of making database querying accessible to all types of users still remains open.

3.1.1 Effects of Abstraction in Database Interfaces

The interface of a database is responsible for how users perceive the database system, as it allows the communication of the semantics between the system and the user. These interfaces can be classified according to their level of abstraction [46]:

- **Physical** - Interfaces with a very low level of abstraction and that require the user to know details about physical storage and access structures used in that system;
- **Logical** - In this interfaces, the queries are specified in terms of abstract structures for data and operations. The most common examples of this level of abstraction are the relational databases interfaces. This level of abstraction doesn't require users to know the physical details of how data is stored or accessed but requires knowledge of joining operations and their purpose to specify relationships. Without this knowledge, users are very prone to errors when using these interfaces.
- **Conceptual** - This is the level with highest abstraction. Interactions are expressed in terms of real-world concepts such as entities, objects and relationships. A data model suitable for this level of interaction is the entity-relationship (ER) model.

Siau et al. [46] compared a logical interface, Query-By-Example (QBE) [51], and a conceptual interface, Visual Knowledge Query Language (VKQL) [47], with the purpose of studying how

the level of abstraction of interfaces and query complexity affect the accuracy, time taken to formulate the queries and confidence in query performance by novice users.

The experiment involved three tests: an initial test, a retention test and a relearning test. The subjects were randomly selected from a population of 480 computer science students with some computing but no database experience and randomly assigned to the conceptual and logical groups. By choosing novice subjects it was possible to study the learning effect of both interfaces.

The results confirmed that users in the conceptual group achieved higher accuracy and higher confidence in all three tests, and even took less time in query formulation. This way, the study concluded that the use of conceptual interfaces in organizational databases can achieve better efficiency and effectiveness.

3.1.2 Visual Interfaces

Visual Query Systems use interfaces based on abstractions and can be classified according to the way they represent the data model (the representation of the reality of interest) and how they express the queries (query representation) to the users [22], as described in Table 3.1.

Jorge Lloret-Gazo [22] extended the work of Catarci *et al.* [6]. In that older article, where the VQSs reviewed were from 1975 to 1996, it was concluded that VQSs still had various problems, as they were not user friendly or very powerful, and the available data types were also limited. In the extended and more recent version, the author reviewed VQSs from 1997 to 2017, focusing on visual queries to structured information. The main type of interfaces found for database representation was diagrammatic, and regarding query representation the distribution was more balanced, except for the faceted type that was only found once.

However, very few papers were still found that offered a prototype that had been tested in a real environment (Polaris [49], now known as Tableau, was the only system found that was web-oriented and that was tested in a real environment), leading to the conclusion that VQSs had not been a widely accepted solution for novel users yet.

Despite that, Jorge Lloret-Gazo concludes by saying that he strongly believes that research in this direction should continue and that a solution for naive users is necessary.

There have also been efforts in a slightly different direction: instead of trying to replace SQL try to make it faster to use and easier to learn and understand. Some examples of systems developed with this purpose are QueryViz [11], which aims at allowing users to read and understand existing queries faster, and QueRIE [1] that aims at assisting non-expert users of scientific databases by tracking their querying behavior and generating personalized query recommendations. However, this thesis is not focused on understanding SQL but rather on developing an alternative approach that enables SQL to be visually formulated, so we won't focus on this kind of system.

Table 3.1: Classification of visual query interfaces

Type	Representation of the reality of interest	Query representation
Diagram-based	This representation offers the user a diagrammatic representation of the data model, made with typical graphical representation for its elements. An example of this type of representation is the entity/relationship metamodel, in which usually entities are presented with rectangles, relationships with diamonds and attributes with ovals.	In diagram-based representations the query is expressed through a diagrammatic representation of the data model.
Icon-based	In this type of representation the user doesn't have the complete data model available and only has iconic representations of some elements. This type of representation is mainly directed to users who don't have knowledge of data models and it's concepts. The icons aim to be metaphorical representations of concepts in a way that they are more intuitive to the users. The main challenge in this approach is to find a way to represent the concepts without being ambiguous to different users with different contexts.	There are two different approaches to how icon based-based representation addresses the expression of a query: 1 - the system offers icons to represent the elements that are part of the query, and for building it the user can drag and drop icons to the canvas; 2 - the icons represent both the entities and the available functions of the system.
Form-based	Typical web pages forms work as a representation of the database. Those forms consist of grids with rectangular components. Forms usually allow the nesting of components.	Form-based facilitates the process of expressing a query because it allows users to select from available options in different components in the form. The disadvantage is that not always the available logic in the form meets the user's logic.
Faceted	The data is modeled as classifications that organize a set of items into multiple and independent categories. Each classification is called a facet and a collection of these is known as faceted metadata. Category labels within a facet are called facet values.	Queries are specified by the user by clicking on the appropriate links. In these systems, data and metadata coexist on the same page.

3.1.2.1 Form-based

In the late 1970s, the first generation of VQSs was marked by the introduction of Query-By-Example (QBE), the first form-based system. Its simple tabular query mechanism proved that database querying was no longer restricted to expert users. The main contribution of this system was that it allowed removing from the user the tasks of inputting or remembering attributes names. Selecting which attributes to include in the result could be done by simple check-marking through a form [23]. Many variations have originated from QBE, but it is core ideas have persisted.

As said by Alan Dix [13], querying a database is often a cyclic process. OVI-2 [23] is a form-based query interface that is based on this affirmation and that started being officially used at Aalto University's Otaniemi campus in 2008. It aimed to adequately express complex queries while being user-friendly, by following a two-phased approach for querying: a first phase focusing on retrieving all tuples and attributes that may be of interest; a second phase allowing the user to narrow down the set of tuples and select only the attributes that are actually needed. This two-phased mechanism for querying visualization is the main characteristic that distinguishes OVI-2 from many other VQSs.

Technically speaking, OVI-2 makes a distinction between a primary form, which visualizes a primary concept, and a target form, which is an auxiliary form to visualize the target concept that is related to the active primary concept. The primary concept is the main subject or principal actor of the query, since it determines the primary key of the tuples that are going to be

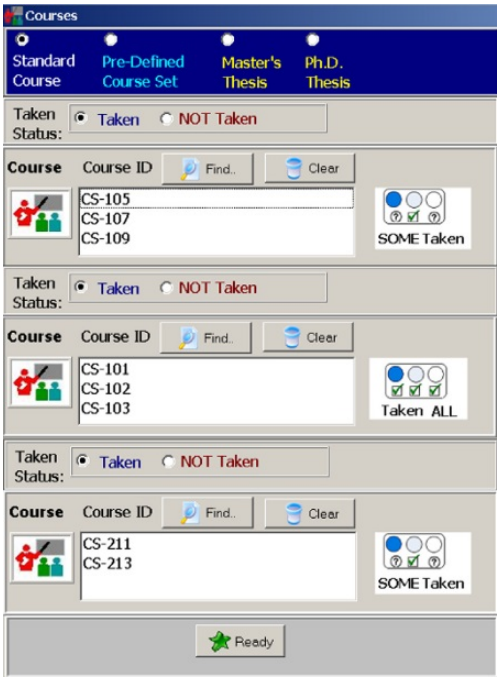


Figure 3.1: OVI-2 visual interface for Query 1 built with the set memberships displayed in the COURSES-form [23]

returned through the query. The target concept acts as the object or target of the query, as it determines the foreign key through which additional tuples from other entities of interest can be associated with the primary concept's primary key. With those two types of forms, users are able to express complex queries using set membership operators. These membership operators are all associated with the same single relationship that prevails between the primary and target concepts. This relationship is a many-to-many relationship so that complex queries are possible.

Set membership testing refers to testing whether a given attribute's value occurs in a given set of values. This mechanism of OVI-2 is particularly interesting for this thesis, given that its behavior is related to the SQL IN operator. The general structure for defining a query with set memberships is the following:

Member of PrimaryConcept FOR WHOM {*MembershipOp* OF B}
IS IN A_x FOR TargetConcept
[WITH *TargetRestriction*]

Where:

MembershipOp is by default the Some operator but may be any of the following operators: *Some, None, All, NotAll or Only*;

B is the user-defined target set of one or more items from a target concept;

x is the subject that is being queried, and is an instance of the primary concept;

Ax is the set with the items that are key values for the subject *x* that is being queried;

TargetRestriction is a set of optional restrictions that can be applied further and that narrow down the target concept (usually are boolean conditions applied over the attributes of the target concept).

To better understand how set membership works, let's look at the following query example presented in the article [23]:

Query 1 The Introductory Course Requirement for Computer Science (CS) is fulfilled by taking any single course of the three courses in the set C1 = {CS-101; CS-102; CS-103} plus all of the three courses in the set C2 = {CS-105; CS-107; CS-109} and any single course from the two courses in the third set C3 = {CS-211; CS-213}. Find those CS students who have fulfilled their Introductory Course Requirement.

This query can be represented with the general notation used previously as:

$Student_x$ FOR WHOM *Some* OF {CS-101; CS-102; CS-103}
AND *All* OF {CS-105; CS-107; CS-109}
AND *Some* OF {CS-211; CS-213}
IS IN A_x FOR Courses

Using the OVI-2 visual interface presented in Figure 3.1 we can see the window for Query 1 built with the three set memberships operations presented on the rectangular boxes on the right, each associated to the respective course ids.

This paper shows that form-based VQSs can adequately express complex queries, and in particular shows that complex queries involving universal quantification or negation can be substantially improved in terms of user-friendliness through the use of simple set membership operators like *Some, None, All and Only*.

3.1.2.2 Diagram-based Vs SQL

Around 1990, T. Catarci and G. Santucci [7] described an experiment aiming at estimating the advantages/disadvantages of a visual approach to database querying against a traditional one. In order to do this, they compared Query-By-Diagram (QBD*) [2], a visual query system based on a diagrammatic representation of the Entity-Relationship schema describing the underlying (relational) database balancing a high expressive power with a noticeable facility of use, against the well-known SQL language.

To compare both systems, the task that users performed was query writing and a teaching period for both technologies proceeded the evaluation. The purpose of the evaluation was to understand which one is the easiest to use by different user classes (in other words, compare the usability of the two systems). The method used was observational evaluation, involving real users that were observed when interacting with the systems.102 were involved in the experiment and were divided into three groups: 56 naive users (persons having little or no knowledge about computer science), 30 intermediate users (persons having general knowledge about computer science but naive about databases) and 16 expert users (persons having a precise knowledge about databases). Each system was tested with the same number of persons from which group. The users answered queries of increasing complexity and for each group they measured the number of errors plus the time spent in formulating the query.

Regarding the Overall average of correctly completed queries, there was a significant difference in accuracy between the results of SQL and QBD* for naive and intermediate users in favor of the later system, but not for expert users. Concerning the Overall average completion time, there wasn't a significant difference between the results of SQL and QBD* for naive users. There was however a significant difference between the results for intermediate and expert users, in favor of QBD*.

In summary, they concluded that that QBD* had the following advantages over SQL: it offers a complete view of the database at a higher abstraction level; the syntax is much more

simple (the user doesn't need to remember tables or attributes names, just needs to choose them navigating on the schema); the time needed to directly manipulate objects on the screen is less than the time needed for writing SQL statements; the error rate is reduced (in particular for simple queries); the users find QBD* more attractive, so they are not bored by working with it.

The results of the experiment confirmed the belief presented by Catarci *et al.* [6] that a visual approach is particularly suited for medium and naive users, and for certain types of queries.

3.1.3 Spreadsheet Concepts in Visual Querying

In 2016, Bakke *et al.* considered that, in order to be a successful alternative to the traditional approach of SQL, any VQS has to have the following three characteristics:

1. Query specification through direct manipulation of results;
2. The ability to view and modify any part of the current query without departing from the direct manipulation interface;
3. SQL-like expressiveness.

They decided to prove the hypotheses that database querying is hard, but can be made significantly easier using a direct manipulation interface. So they developed a prototype called SIEUFERD [3], a VQS aimed at meeting all these three requirements in a single design. In order

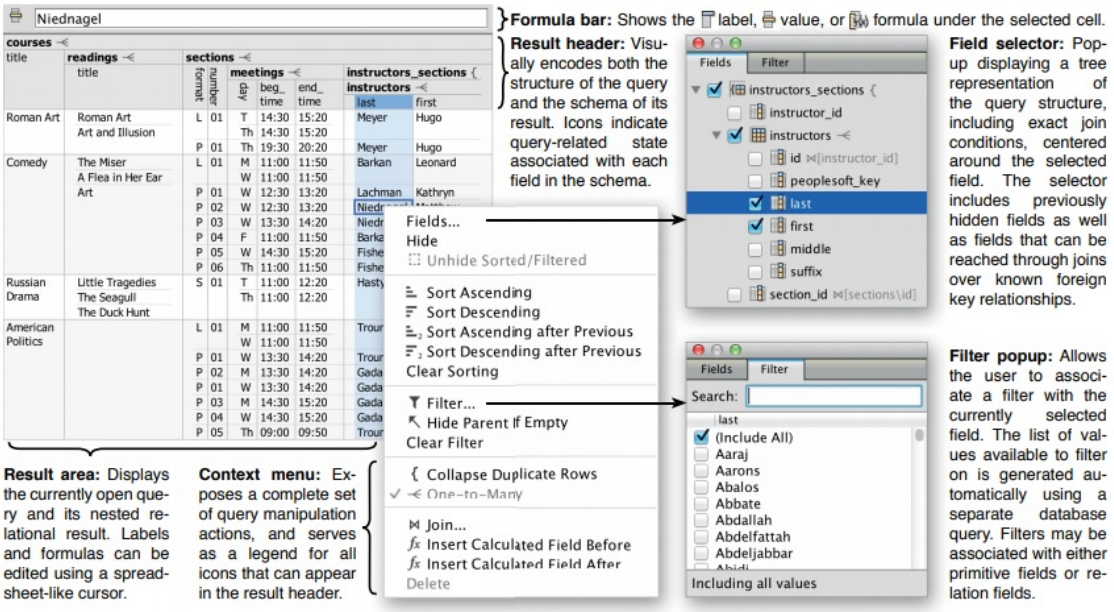


Figure 3.2: SIEUFERD visual query interface [3]. In this example is presented a query that instantiates six database tables, contains five joins and is evaluated using five generated SQL queries.

to reach it, they used spreadsheet idioms (like formulas and filters) and allowed direct manipulation of nested relational results. This way, the users could express a relationally complete set of query operators, including calculation, aggregation, outer joins, sorting, and nesting operations, while always being able to see the current status of the query and modify it with appropriate actions.

In Figure 3.2 is presented the core query building interface of SIEUFERD. The user interaction starts in the **Result Area**, where the currently open query and its nested relational result is presented in the format of a nested table layout. By selecting any field (column), a **Context menu** is open with many query-related actions that can change the query state. Whenever the query state is modified, the displayed flat results, fields and icons are updated. In the **Result header**, the user has access to a set of icons that can be used to augment the information that can be derived from the names and positions of fields. The **Filter icon** indicates the presence of a filter on that field, which can be manipulated by opening the filter popup from the context menu. The **Formula icon** indicates that it is a calculated field with an associated spreadsheet-style formula. The actual formula can be edited using the Formula bar or directly in any non-header cell belonging to the field's column. The **Field selector**, which can be accessed through the Context menu, also serves to suggest new joins over known foreign key relationships, modeled as pre-existing hidden fields, and to display exact join conditions. Finally, another interesting feature of this visual interface is the possibility to hide non-relevant fields (columns), making it easier for the users to get a sense of the data.

In short, we can identify the following key features in the system: visual stability, decoupled query and result updates, interruptible queries, automatic query limiting, high-level error handling and undo/redo operations.

Although the authors conclude the article saying that in the current query interface some queries are expressible yet awkward to construct, they also affirm that it achieves SQL-like expressiveness, without sacrificing expressiveness or hiding the actual query from the user like many other VQSs. Spreadsheet concepts are very interesting and promising, as they make it possible to integrate the query and its result into a single interactive visualization.

3.2 Visual Query Builders Today

In the Gartner Magic Quadrant published in August of 2021 (see Annex II for more details), it is possible to see that SalesForce, OutSystems and Mendix could be considered leaders in Enterprise Low-Code Application Platforms.

As presented in this thesis, since 2013 that OutSystems offers a visual alternative for querying called Aggregates, covered in Section 2.2.3. Its design is based on spreadsheet concepts that are also used in popular software like Microsoft Excel, and also with similarities to the system presented in Section 3.1.3.

Regarding SalesForce [1], they launched a Beta version of their first visual query builder in

[1] SalesForce https://www.salesforce.com/eu/?ir=1, 2021

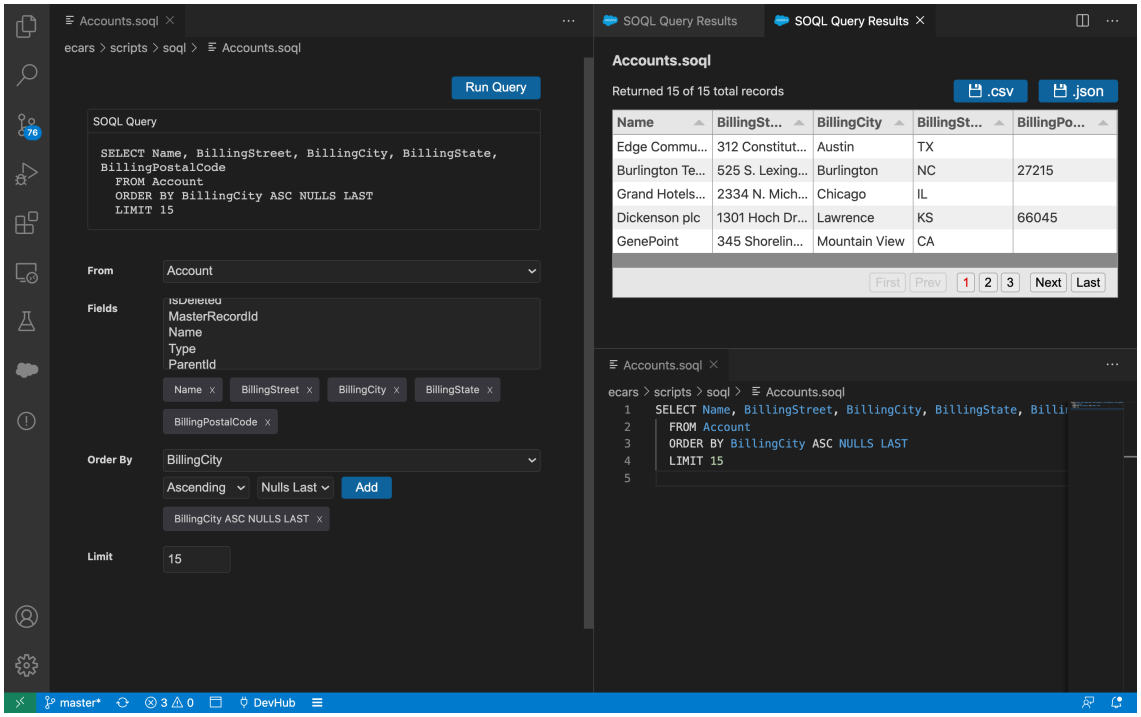


Figure 3.3: SOQL Query Builder interface [43]

November of 2020 called SOQL Query Builder [43]. It is a Visual Studio Code extension that enables users to interactively build a SOQL query via a form-based visual editor (Figure 3.3), view the query while it is built and save the output to a .csv or .json file. At the moment it allows to build simple queries that include: FROM clause for only one Object type; SELECT clause to pick fields from the selected Object; WHERE clause to filter your data; ORDER BY clause with support for ASC, DESC, NULLS FIRST, and NULLS LAST; LIMIT clause. For the context of this thesis is relevant to point out that regarding the WHERE clause this builder only supports simple expressions, combining conditions using AND or OR, but not both [44].

Concerning Mendix [2], it supports several query languages being the main one XPath [24], which is an easy-to-use query language that enables users to select data from Mendix objects and their attributes or associations. It can be used to get objects that the user wants to display or edit in pages, as well as to modify these objects through microflows. XPath allows to add query aggregate functions and to filter results by adding constraint functions to any Xpath query.

All these three largely used low-code platforms try to offer to their users alternatives to the traditional textual query languages like SQL, proving that there has been a progressive commercial effort towards making querying databases accessible to all types of users.

Regarding tools that aim at helping formulating SQL, Chartio [3] and Devart dbForge [4] are some of the most popular platforms in that area. Focusing on query filtering, these kinds of systems make the process easier by offering graphical interfaces. Regarding Chartio, they released

[2] Mendix https://www.mendix.com/, 2021
[3] Chartio https://chartio.com/, 2021
[4] Devart dbForge https://www.devart.com/dbforge/, 2021

a new interface last year called Visual SQL [9] based on browsing through tables, intelligent and adjustable grouping and aggregation, direct writes to SQL that can be edited and results table in spreadsheet style. In Chartio Visual SQL, the user selects a column to apply the filter and then a dropdown list with possible filter operators for that column data type appears. It is also possible to create a chain of AND-OR filter conditions. Chartio Visual SQL supports the "is one of"and "is not one of"filter operators that simulate the SQL IN and NOT IN operators respectively [8]. Devart dbForge offers a tool called Query Builder [12] that helps users to design queries visually. It includes a pane that displays the query in a diagram format and a pane that displays the SQL text of the query. Starting formulating a query can be done by drag-and-drop tables from the Database Explorer to the diagram pane and joins are created automatically. In the tabbed editor, the user can select one of the following tabs to edit: Selection, Joins, Where, Group By, Having and Order By. Selecting the Where tab, the user can choose to add a new condition and an empty triple appears with the default format "<enter a value> = <enter a value>". Clicking on the first "<enter a value>", a form with two lists appears and the user can select the attribute that he wants to filter. Then selecting the "=", a list with the possible operators appears. The IN and NOT IN operators are available. Finally, the user can click on the second "<enter a value>"and write a value for the match.

However, none of these solutions is what we are trying to do in this thesis, as their context is restricted to analyzing data or helping to formulate SQL. This also applies to several other interfaces in the data analytic domain, like Tableau [5], Microsoft Power BI [6], etc. The purpose of this thesis is to implement a way of passing a list of values that comes from a program to a VQS to filter by its values, which is a more complex context.

3.3 Summary

In this chapter different types of VQSs were presented and analyzed in order to understand how they have evolved through time. It is clear that visual querying is getting more and more popular, as the amount of data made available with the Web keeps growing. Consequentially, there is an increasing interest in inexperienced users being able to use systems that require data consultation, from academic administration environments (Section 3.1.2.1) to more complex environments of application development (Section 3.2).

Although there are already several tools that use visual approaches to visualize and filter data, visual query interfaces with the same level of expressiveness as SQL in the context of low-code application development do not yet exist. This made the research process to elaborate the Related Work difficult, as there are no direct competitors to compare with what is proposed in this thesis. However, the approaches used by those systems for filtering are relevant in the study of design options for the new interface to be developed.

[5] Tableau https://www.tableau.com/, July, 2021
[6] Microsoft Power BI https://powerbi.microsoft.com/en-us/, July, 2021

4

Requirements and Methodology

Although OutSystems has been making efforts to improve the usability of the Aggregates, the expressiveness limitations that it presents still remain one of the main reasons why many users still prefer to use their SQL tool. This chapter focuses on exploring and analyzing the expressiveness limitations existent at the current state of the Aggregates, in order to understand the current problems, propose solutions and define the work process.

4.1 Problem Exploration

In order to understand and start analyzing the problem intended to be solved, it was necessary to start by following the OutSystems "Becoming a Reactive Web Developer" tutorials and do some self-exploration of the interface in order to get familiar with it.

The next step was to try, as a beginner user, to create an app that would allow to see a list of products, sort them, see their details and, the most important part, filter them. Each product should be composed by a name, category and price. No problems were found using an Aggregate to retrieve and filter the products by one category. However, when trying to filter by two or more categories, difficulties started to appear, as the Aggregates don't provide the IN and NOT IN functionalities that would make this implementation simple.

4.1.1 Current Alternative Solutions Analysis

After initial research regarding how OutSystems users currently overcome the absence of the IN and NOT IN operators in the Aggregates, we found out that in the OutSystems Community the "IN clause in Aggregates"is the most trending and liked idea in the Aggregates & Queries category, with 333 likes, 5936 views and 63 followers. Looking at the date of creation of this discussion, we could conclude that the addition of this operator in the Aggregates has been asked by OutSystems users since, at least, 2015.

Following a deeper investigation, three alternative solutions were discovered and analyzed in order to achieve the filtering of a list of entities by two or more in-memory values in the initial state of the platform. The first two solutions only use Aggregates and the third one uses the OutSystems SQL tool:

Use the Index function: This first solution was the one for which we found the most mentions (besides using SQL) and consists in using a Built-in Function called Index(), which is the equivalent to the C# String.Contains function. The Index() function can be used inside the filters of an Aggregate and, together with a Text variable that the user can build as a concatenation of Text values that he wants to use to filter (for example: a coma separated Text variable of IDs), it can achieve a similar behavior to the one from the IN SQL operator. The main difference is that this function only works with values of type Text. So what this function basically does is look for the first Text parameter in the second Text parameter. If it's found, the function returns the position where it was found. Otherwise, it just returns -1, meaning that it was not found. So, for example, if we add a filter condition with something like "Index(Report.Id, SelectedIds) >= 0"it would return true if Id is contained inside SelectedIds and, in that way, that Report would be returned by the Aggregate.

However, besides not being a clean solution, it also becomes very inefficient if there are many values in the second parameter. Another potential problem can occur if there are values in the second parameter that share some part with other values. To prevent that it is very important that the values are delimited by some special char (e.g., #).

Filter after fetching data: Another possible solution found that doesn't require SQL knowledge was to filter the records after fetching all of them (see Figure 4.1). In other words, it means applying the filtering process after using the Aggregate for the retrieval of all the products. The problem with this solution is that, despite requiring even more logic components than the first solution, it also requires the retrieval of all the data and only then the filtering is applied, which results in an efficiency problem too.

Use the SQL tool: This solution has the advantage of being more intuitive for those users that already know SQL and, in particular, know the IN operator. However, it requires SQL syntax knowledge (See Figure 4.2). In this solution, the user also has to be aware of security problems like SQL injection. To help prevent SQL injection vulnerabilities, the user should use the server action BuildSafe_InClauseTextList when creating the concatenated variable with the values for filtering [34].

4.1.2 IN Operator Usage in the OutSystems Platform

According to statistics made in 2020 by P.S.Rodrigues [41], 15.08% of the SQL queries used in OutSystems applications that involved operations not supported by Aggregates used the IN operator (Annex I). However, in those statistics the percentage shown for the IN operator included its usage together with subqueries, which is not relevant for the context of this thesis and, consequentially, should not be counted. Additionally, that percentage didn't include the other alternatives besides using SQL (presented in Section 4.1.1) to simulate the same behavior as the IN operator. For those reasons, it was necessary to research further. In order to assess if the IN functionality is often required in the development of OutSystems applications, we divided our research into three parts: "IN operator in SQL Queries", "IN operator behavior simulated using the Index() function"and "In operator behavior simulated through post-processing logic".

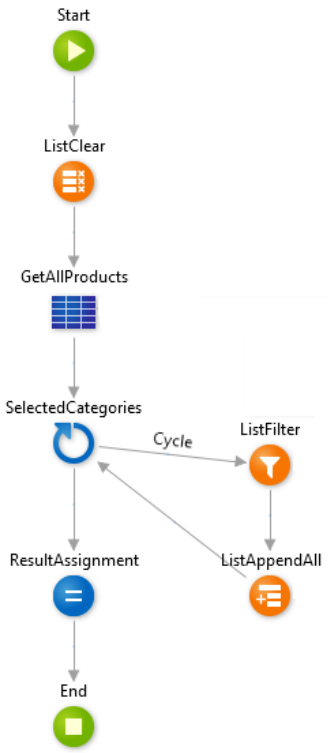


Figure 4.1: Example of a post-process of records

- **IN operator in SQL queries**: Regarding the usage of the IN operator in SQL queries without being associated with subqueries, the OutSystems AI team determined that 8.42% of all the Select statements that cannot be represented by Aggregates use the IN operator without subqueries.

- **IN operator behavior simulated using the Index() function**: In order to elaborate new statistics, we used a program called QueryGrabber. This program allows OutSystems internal developers to navigate through all the application flows and analyze their components. After some research and conducting interviews with OutSystems Community users, it was concluded that the main alternative for performing filtering of persistent data with a local list of values without using SQL was using the built-in function called Index(). This way, a new extension was added to this program with the intention of quantifying how many Aggregates contain the Index() function in their filters. We used the Microsoft PowerBI tool to answer the following question: What is the percentage of (distinct) customers that have used one or more Aggregates that have the Index function in its Filters?

  As can be observed in Figure 4.3, **47.7%** of the customers analyzed have used one or more Aggregate that uses the Index() function. In total, **8948 modules** and **1106 customers** were analyzed.

- **In operator behavior simulated through post-processing logic**: It is also important to

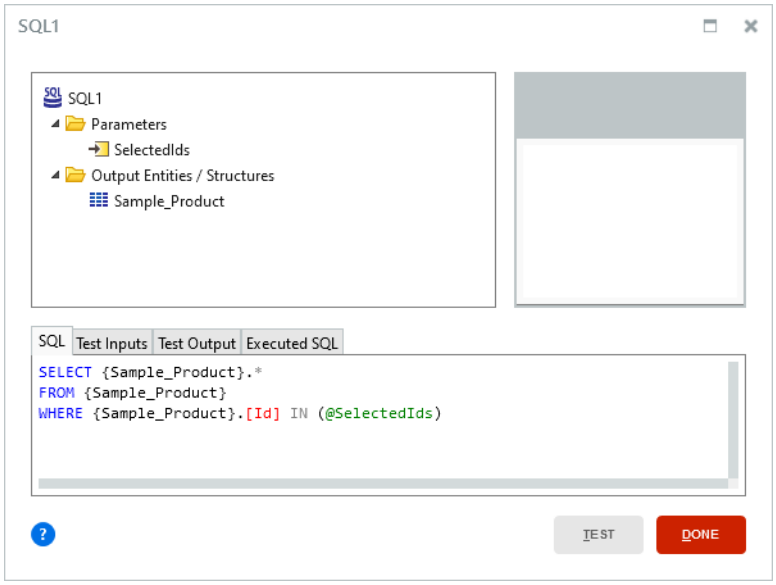


Figure 4.2: SQL editor with a query using the IN operator

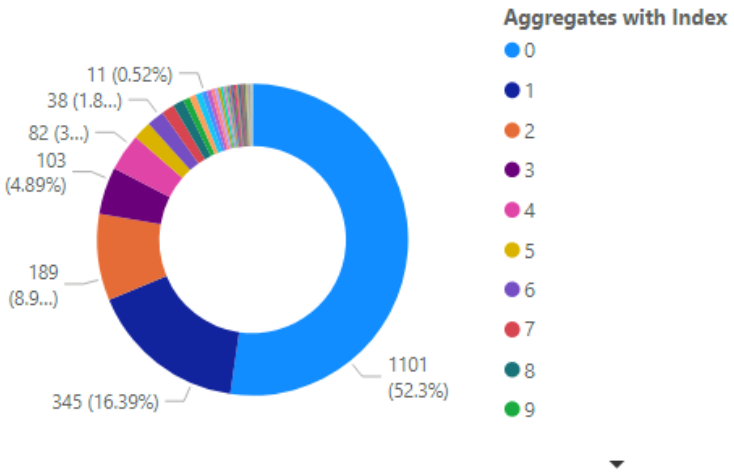


Figure 4.3: Count of Costumers by number of Aggregates with Index

point out that there are other alternatives that should also be accounted for. For example: The users can use the Aggregate to get all the products and then add the intended ones to a new list; The users can make a call to the database for each category intended and add all the results to a new list, etc. Given that this logic is developed outside the Aggregate and, because of that, can be performed in different places and with more or less complexity, it was not possible to quantify the usage of these types of solutions using the QueryGrabber.

Given the verified frequency of this use case and taking into account how difficult to implement the workarounds that are currently being used by several users are, we concluded that this is a very relevant language design challenge that must be addressed.

4.2 Alternative Approaches

After analyzing how the OutSystems users currently overcome the problem previously presented, we devised the following four possible solutions:

1. Add the IN operator in the Aggregate filters expression editor;
2. Allow adding Lists as Sources of an Aggregate and allow using joins to match them with database entities (tables);
3. Create a new list system action that would make post-processing simpler;
4. Add the IN operator in the expression editor of the condition property of the already existent ListFilter list system action.

The first solution was proposed because it seemed to be the more intuitive one for users that have SQL knowledge. It also seemed to be less disruptive for those who don't need this functionality. The second solution was considered knowing that it could be a less intuitive solution, as those who don't know what a Join is probably wouldn't understand how to get there. However, it had the potentiality of also solving other use cases of mashup in-memory data with persistent data. The third and fourth solutions were considered less efficient than the previous two, but could cover use cases involving filtering between two in-memory lists.

4.3 Methodology

The methodology that we present in this section was applied 3 times, in the following 3 different sets of user studies:

1. Platform Current State (Section 5.1);
2. Low-fidelity Prototype (Section 6.1);
3. Service Studio Prototype (Section 6.3).

The work described in this document followed the methodology presented in Figure 4.4, which is an iterative design strategy (see Section 2.1.1 for more details), always maintaining a user-centered design approach. The first phase covered the **Relevance Cycle**, which bridges the contextual environment of the research project with the design science activities, with the main purpose of understanding the problem and defining how to solve it. The second phase of this thesis focused on the **Design Cycle**, which consists in iterating between the core activities of building and evaluating the design artifacts and processes of the research, and the **Rigor Cycle** that connects the design science activities with the knowledge base of scientific foundations, experience, and expertise that informs the research project [17].

We planned the work process to be made in 4 phases, as illustrated in Figure 4.5. The first phase was regarding the **Platform Current State**. This phase was very important because it

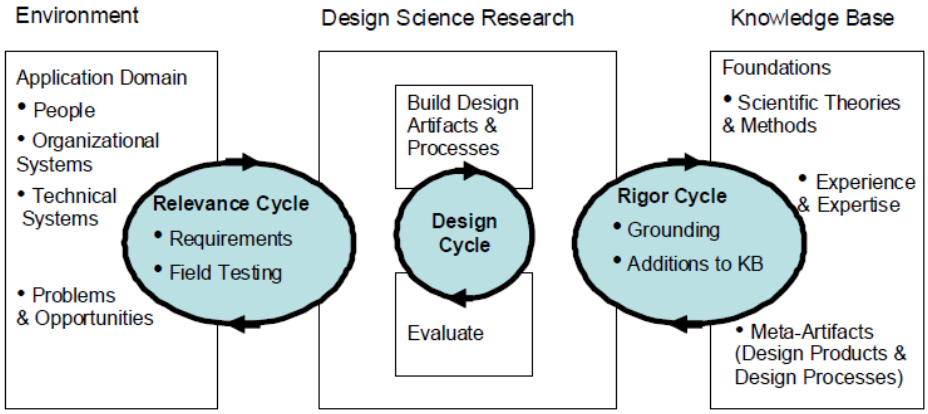


Figure 4.4: Design Science Research Overview [17]

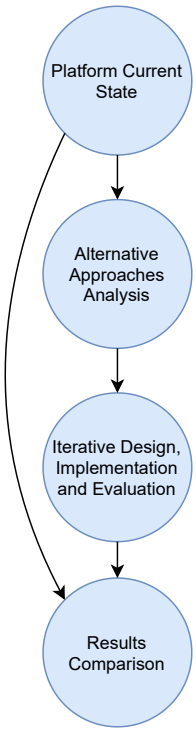


Figure 4.5: Summarized Workflow Diagram

allowed us to have real contact with the problem and also to evaluate the improvements of the new solution compared to its previous state at the end. In the following phase, **Proposed Solutions Analysis**, we proceeded to design and explore possible solutions. The third phase, **Design and Implementation**, started with the implementation of a paper prototype, interactively improved, in order to evaluate the chosen solution from the ones identified in the previous phase. This phase ended with the implementation of a high-fidelity prototype in the Service Studio, using C#, Typescript and React. The work process ended with the **Results Comparison** phase, where we analyzed the results and improvements achieved.

Table 4.1: User Groups

Non-Developers	Users without programming and databases knowledge. They might have experience with other applications such as Microsoft Excel, Google Sheets or other similar software.
Software Developers	Users with traditional programming languages and databases knowledge. These users have a software engineer background and are familiarized with languages such as Java, C# and SQL. Although they are experts regarding software development, they have little or no experience using low-code platforms.
OutSystems Developers	Users that, independently of their background, have medium or high experience level using the OutSystems Platform. They may have other traditional programming languages knowledge and know SQL.

4.3.1 User Groups

This work followed a user-centered design approach, which means that the users played a very important role in the whole design and implementation process. Once we are in the context of software that not only aims at helping experienced users with technical backgrounds but also users with little or no technical knowledge, it was crucial to include in our evaluations a wide diversity of users regarding their technical skills levels. We defined 3 groups, described in Table 4.1, which were tested equally throughout the entire evaluation process as characterization of the target users of the system evaluated.

In order to assess each user's skills with more detail and determine in which group the user should be included, a Google Form was created and sent to each user to fill at the end of each test. For each user, in each prototype, we collected the following main information:

- Frequency of use of the OutSystems platform;
- Number of OutSystems certifications;
- Level of experience with OutSystems;
- Level of programming languages knowledge;
- Level of SQL knowledge;
- Experience with Microsoft Excel, Google Sheets or other similar software.

How many users were tested from each user group for each of the evaluated main solutions is presented in Table 4.2. To perform the usability tests, we followed the methodology proposed by S.Krug [20]. We never repeated users to avoid bias of the results. For each of those solutions, we used 5 users representative of each group to test, using Nielsen's heuristics as justified in Section 2.1.4.1. The tests took into account how much time the user required to execute the proposed task and their effectiveness in the goal achievement (Achieved or Not Achieved). Regarding a more qualitative evaluation of each solution, the user opinion and suggestions in

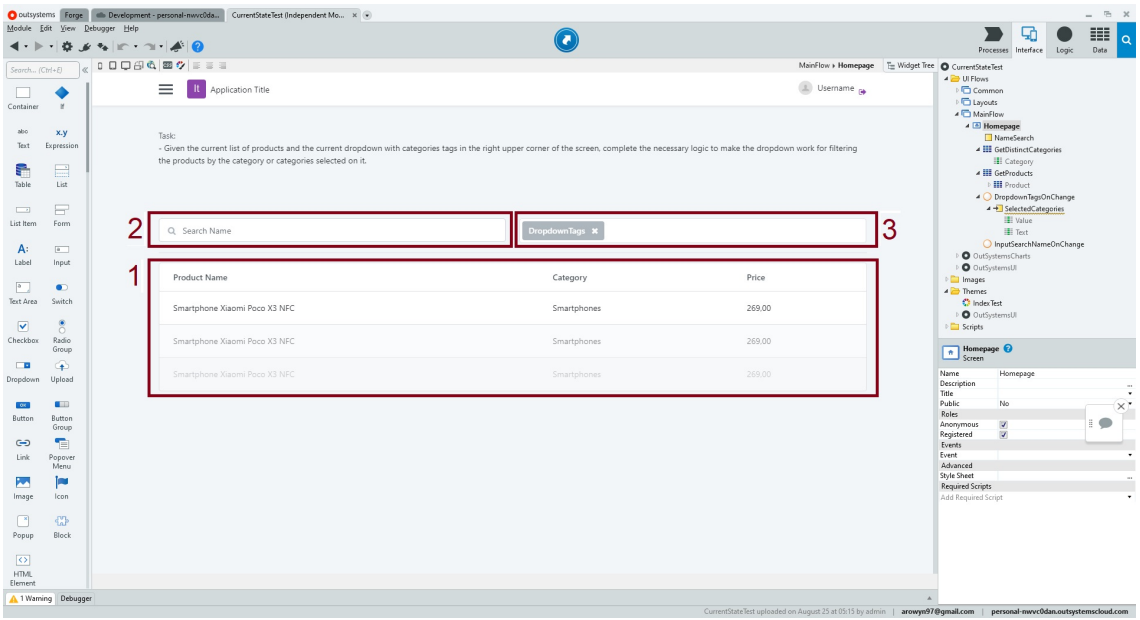


Figure 4.6: Test Scenario: 1 - Table Component; 2 - Input Component; 3 - DropdownTags Component.

each phase were also collected in order to help evaluate the user satisfaction and what should be changed in the next iterations.

4.3.2 Testing Scenario

In Figure 4.6 we can observe the starting screen of the scenario proposed to perform the user tests in this thesis, which was adapted in 4 different evaluation phases. In the center of this screen, the users would start by seeing the following 3 components:

1. (Implemented) - A Table showing a list of products. For each product its Name, Category and Price are presented;

2. (Implemented) - A Input working for filtering the products in the Table by Name;

Table 4.2: Number of tests by user group for each main evaluation phase

Users Group	Current Implementation	Low-fidelity Prototype	Service Studio Prototype	Total
Non-Developers	5	5	5	15
Software Developers	5	5	5	15
OutSystems Developers	5	5	5	15
Total	15	15	15	45

3. (Incomplete) - A DropdownTags already with the options of the Categories available for the user to select. However, at the initial state if the user selects a category nothing happens besides adding the tag in the component itself.

On the right side of the scenario, we can see the Interface tree menu, presented in more detail in Figure 4.7. In this menu, the user could see that the test scenario was composed of only one page, the "Homepage". That page included:

1. A variable, "NameSearch", that is used together with the input component to filter the products by the name present in it;

2. An Aggregate, "GetDistinctCategories", that is used to fill the options of the DropdownTags component with the distinct existing categories;

3. Another Aggregate, "GetProducts", that retrieves the products that are presented in the table component;

4. A Client Action, "DropdownTagsOnChange", which is associated to the DropdownTags component and it's called every time there is a change on it. For example, if the user selects a category then there is a change in the component and the Client Action is activated. Initially this Client Action is presented to the user with an empty logic flow. By default, when adding a new DropdownTags component to a screen, it automatically creates a handler for it with an input parameter. That input parameter is created with the data type "DropdownItem List", which structure is composed by two attributes, "Value"and "Text".

5. Another Client Action, "InputSearchNameOnChange", which is called when there is a change in the Input component and already included the logic necessary to filter the Table products according to it.

The task that was asked for the users to perform in every test was:

Given the current list of products and the current DropdownTags with all the existing categories in the right upper corner of the screen, complete the necessary logic to make the DropdownTags work for filtering the products by the category or categories selected on it.

With this task, we had the main goal of evaluating if different types of users could achieve the filtering of the Products, which are persistent data accessed through the Aggregates, by two or more categories dynamically selected by the user through the DropdownTags component, which are in-memory data. We decided to include the Input component in the Test Scenario as a working example of a similar, but simpler, filtering of persistent data by in-memory data.

While achieving filtering of the products by only one category could be easily done in this testing scenario, for it could be made by adding a new filter condition and using the existing "="operator in the GetProducts Aggregate, when we had more than one category selected in the DropdownTags to filter that operator does not work anymore. The equivalence operator

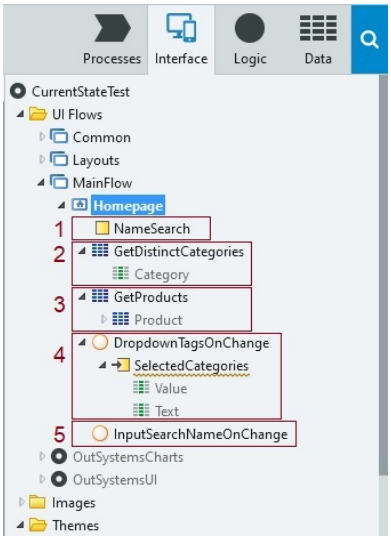
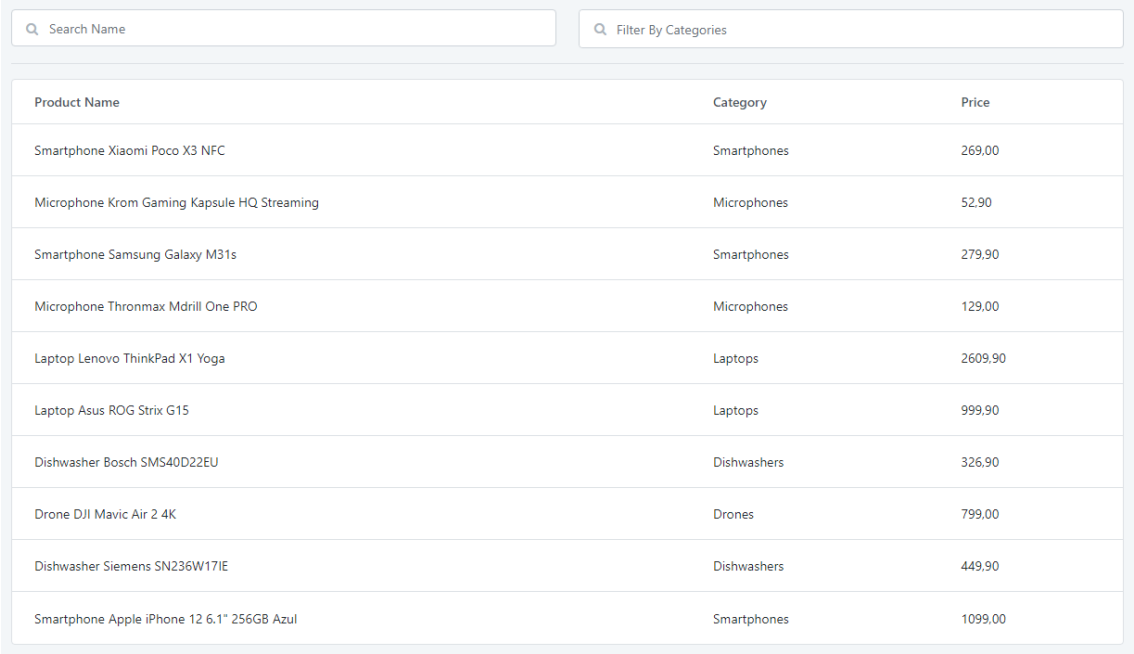


Figure 4.7: Test Scenario Interface Tree: 1 - Variable “NameSearch”; 2 - Aggregate “GetDistinctCategories”; 3 - Aggregate “GetProducts”; 4 - Client Action “DropdownTagsOnChange”; 5 - Client Action “InputSearchNameOnChange”.

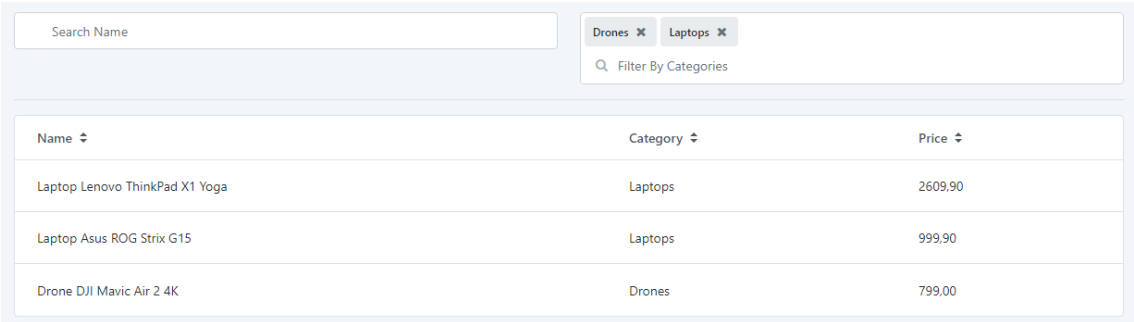
only works for comparing a single value on the left with another single value on the right. That’s where the IN operator comes very handy, as its purpose is to see if a single value is equal to any value in a set of values. In Figure 4.8 it’s illustrated an example of the expected final behavior to successfully complete the task, where the user initially has no filters and then selects the "Drones"and "Laptops"categories in the DropdownTags component and the table is refreshed, showing only the products which are from one of those categories.

4.3.3 Workflow

In Figure 4.9 the whole process described in this section is presented in more detail. The reason why there are two colors, blue and purple, is because the first one represents the main path followed in this dissertation, while the purple path was a parallel exploration that emerged from the main path.



(a) List of products before filtering by any category



(b) List of products after filtering by the "Drones"and "Laptops"categories

Figure 4.8: Testing Scenario expected behavior

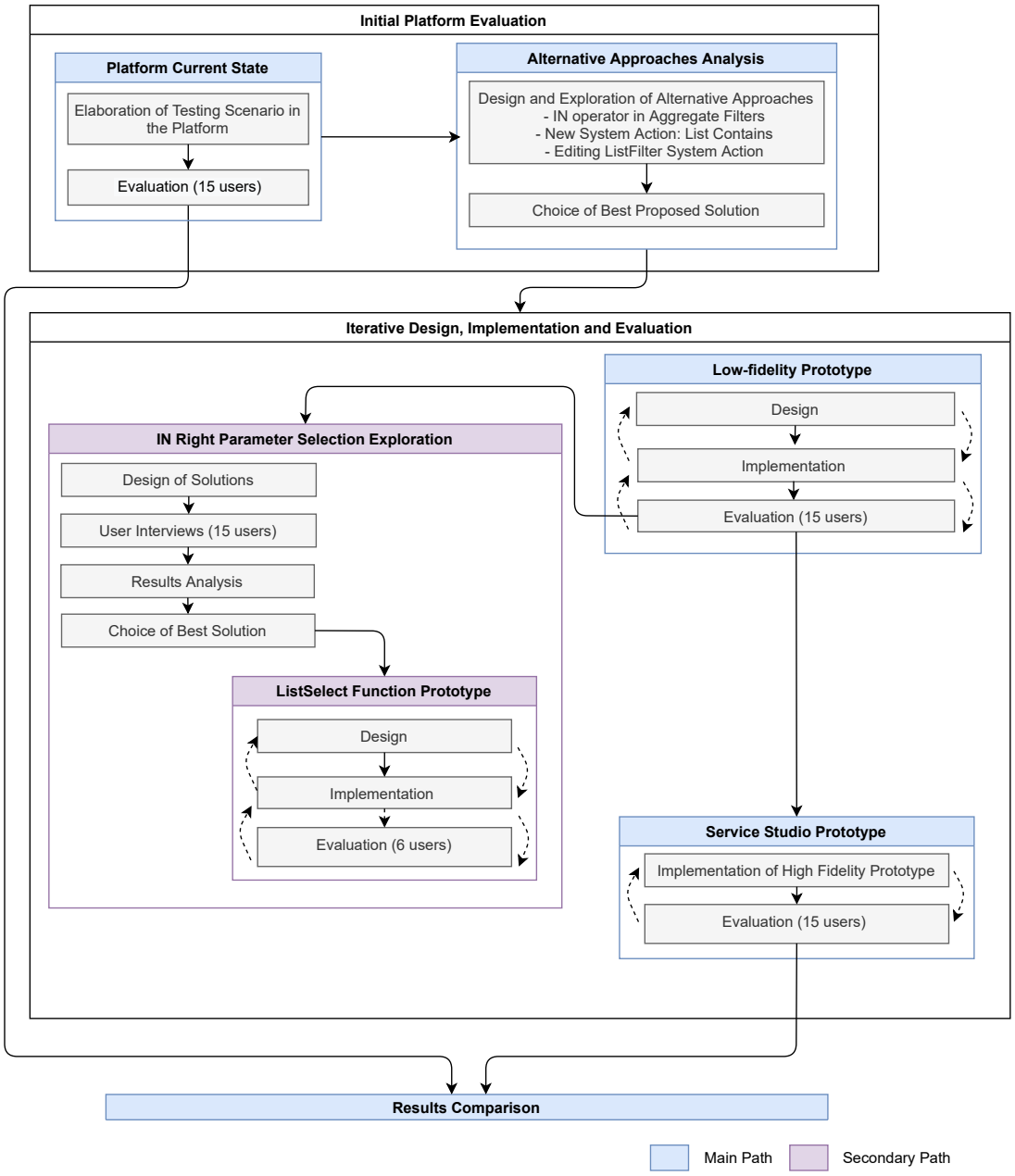


Figure 4.9: Detailed Workflow Diagram

5

Initial Platform Evaluation

Since we intend to improve the expressiveness of the OutSystems platform, we need to start by evaluating the current state of the platform. That evaluation is covered in this chapter, allowing us to understand what needs to be improved and also to be able to measure the progress achieved at the end.

5.1 Platform Current State

In Section 4.1.2, we were able to analyze the frequency of use of the IN functionality in the OutSystems Platform and conclude that it is, indeed, a relevant problem. However, it is not only the frequency of use of a functionality that matters in order to ensure that it has good usability. It is equally important to access how hard it is to use.

In order to evaluate how difficult it is, at the current state of the OutSystems platform, to filter persistent data by in-memory data, 15 usability tests were performed, following the methodology presented in Section 4.3.

5.1.1 Solution Examples

In this subsection, we present, in detail, the main possible solutions for the testing scenario in the current state of the platform. Although there are many possible minor variances in either of them, the examples presented cover the most common and suggested paths in the OutSystems Community to solve this kind of problem.

5.1.1.1 Using SQL

In this solution, the user needs to add a new Text variable (CategoryListCSV) and a new Data Action (DataAction1) in the Homepage screen. In the DropdownTagsOnChange Client Action (see Figure 5.1), the user needs to concatenate, according to SQL syntax, the selected categories from the DropdownTags CurrentList, assign them to the new text variable and, finally, refresh the newly created Server Action (DataAction1), which will retrieve the filtered products to be displayed in the Table. For that, the user needs to add a SQL node in it and open its expression editor to write the query using the IN operator (see Figure 5.2). Having the query correctly

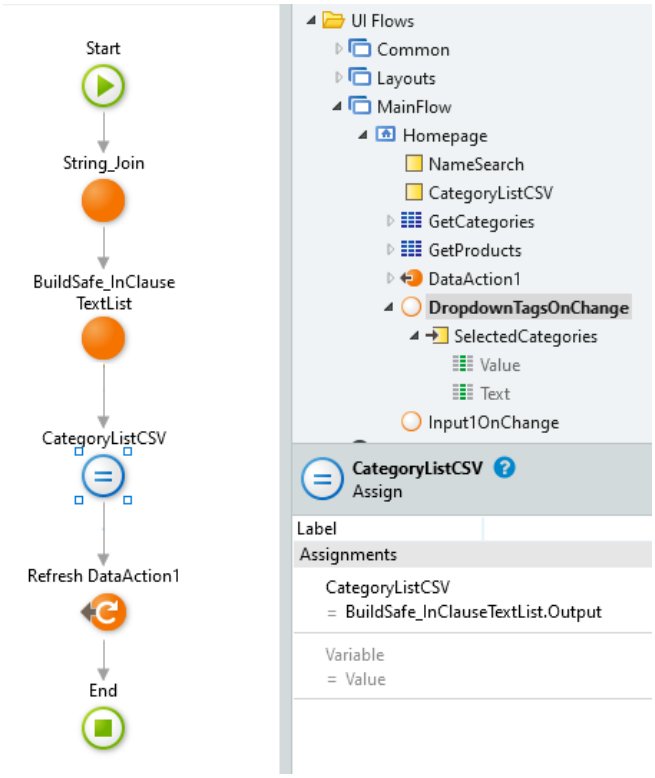


Figure 5.1: DropdownTagsOnChange logic flow: assigning the selected categories to CategoriesListCSV

written, the user will only need to assign to the original list of Products, which is associated with the Table, the new list that contains only the filtered products returned by the SQL query.

5.1.1.2 Making post treatment of the list with all products

In this solution, the user needs to create a new Text List Variable (CategoriesToFilter) and move the GetProducts Aggregate into a new Data Action (GetProducts)in the Homepage. Starting by the DropdownTagsOnChange Client Action (see Figure 5.3), the user needs to add the selected categories from the Dropdowntags CurrentList to the new Text List Variable. For that, he can use the ListAppendAll list system action and then refresh the new Data Action. When the Data Action is refreshed, the flow starts by calling the Aggregate that retrieves all the products (see Figure 5.4). Next, using 2 ForEach components, one for iterating the list with all the products and another to iterate the list variable with all the selected categories, the user is able to check if the category of the current product being iterated (from the list with all products) is equal to the currently selected category being iterated (from the CategoriesToFilter variable). If they are equal, the product is added to a new list. This comparison is repeated for every product. In the end, the user can assign the new list, that only contains the filtered products, to the list of products associated as the Source of the table.

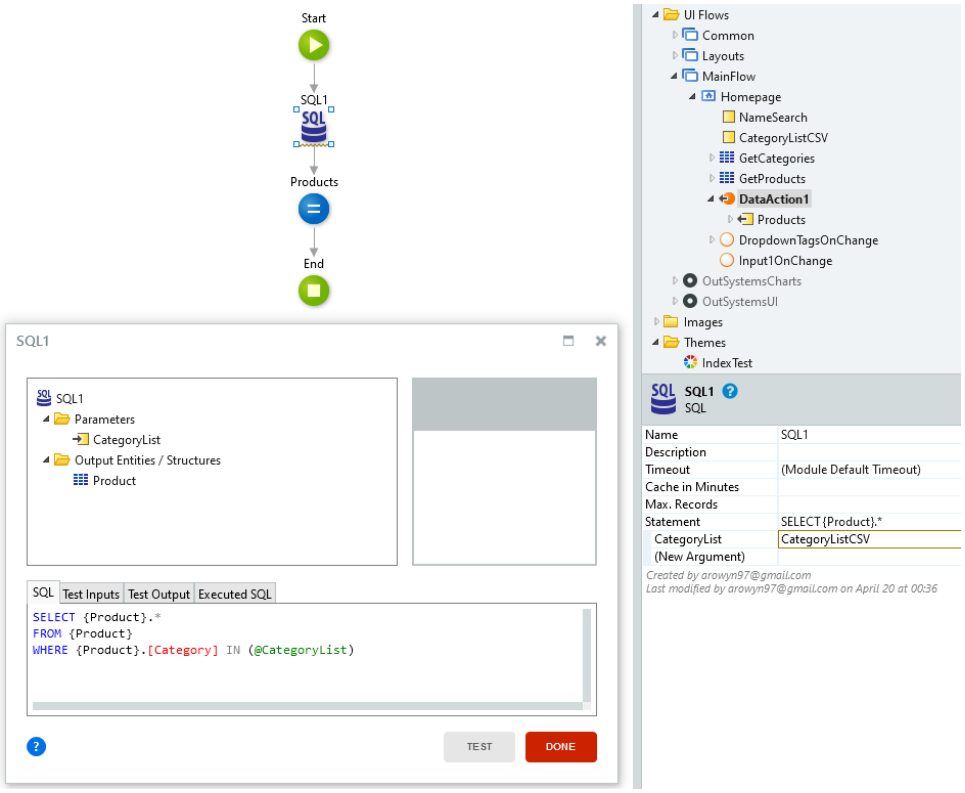


Figure 5.2: DataAction1 logic flow: using the SQL IN operator

5.1.1.3 Using the Index() function

In this solution, the user needs to create a new Text Variable (CategoriesFilter), which will contain the selected categories from the DropdownTags CurrentList, separated by a coma. This can be done in the DropdownTagsOnChange Client Action (See Figure 5.5), by adding the following expression in the Assign component associated with the ForEach component that iterates the CurrentList with the selected categories:

If(CategoriesFilter = "", "'"+ CurrentList.Current.Value + "'", CategoriesFilter + ",'"+ CurrentList.Current.Value + "'")

Next, the user should add a new filtering condition in the already existing GetProducts Aggregate. In that condition, the user can make use of the Index() function, which allows checking if a Text value is contained in another Text value. This way, it is possible to see if the category of each product, which is of type Text, is contained in the concatenated Text Variable of selected categories, CategoriesFilter. If so, the function returns the index where the category was found (which means that the value is greater than -1) and returns that product. Finally, the user only needs to add a refresh of the GetProducts Aggregate in the DropdownTagsOnChange.

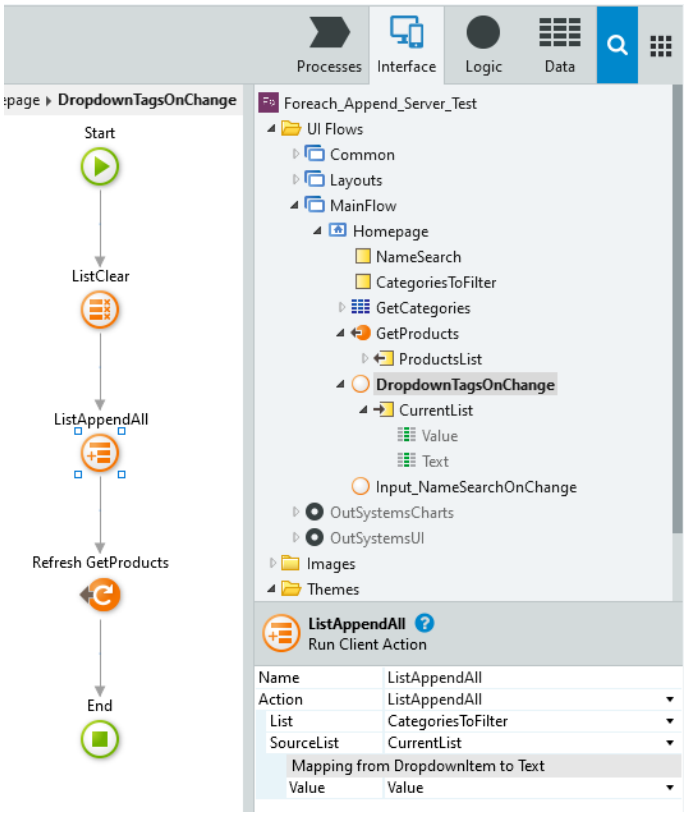


Figure 5.3: DropdownTagsOnChange logic flow: Appending the selected categories to CategoriesToFilter

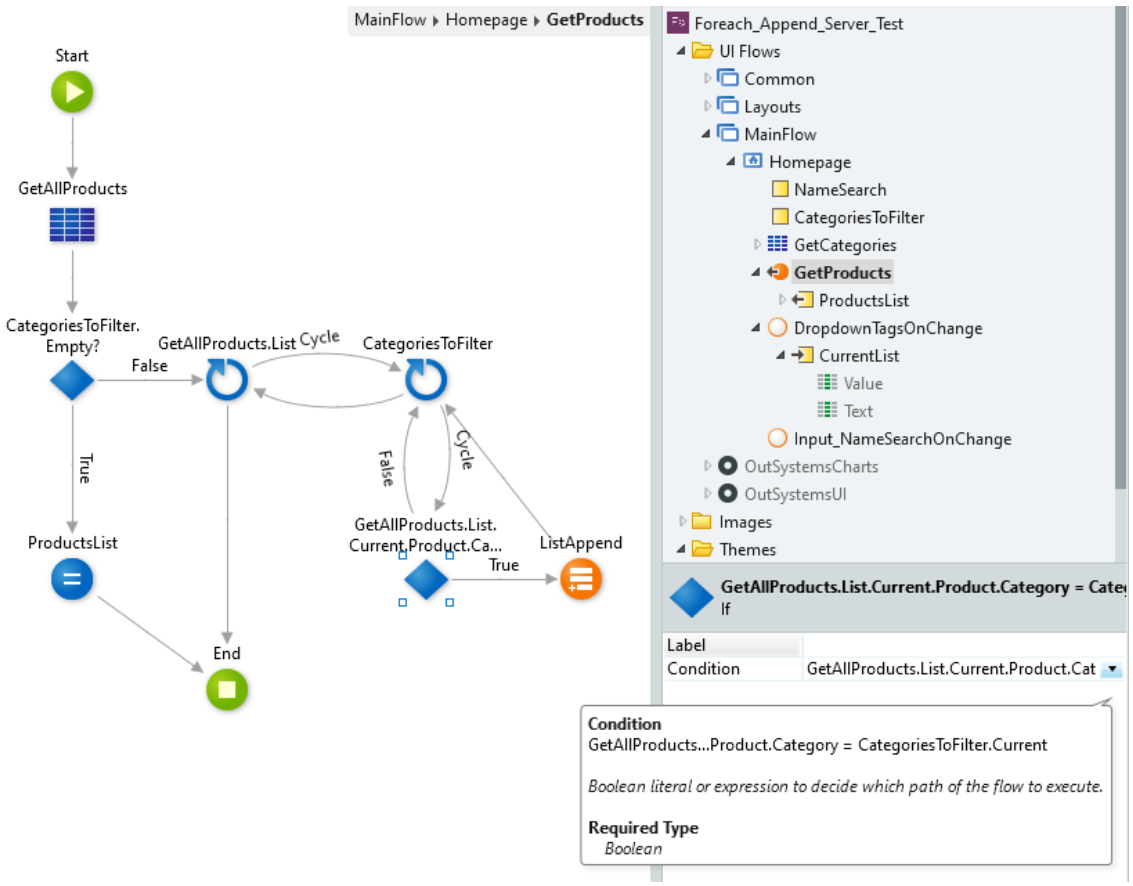


Figure 5.4: GetProducts Data Action logic flow: filtering the products by the selected categories in CategoriesToFilter

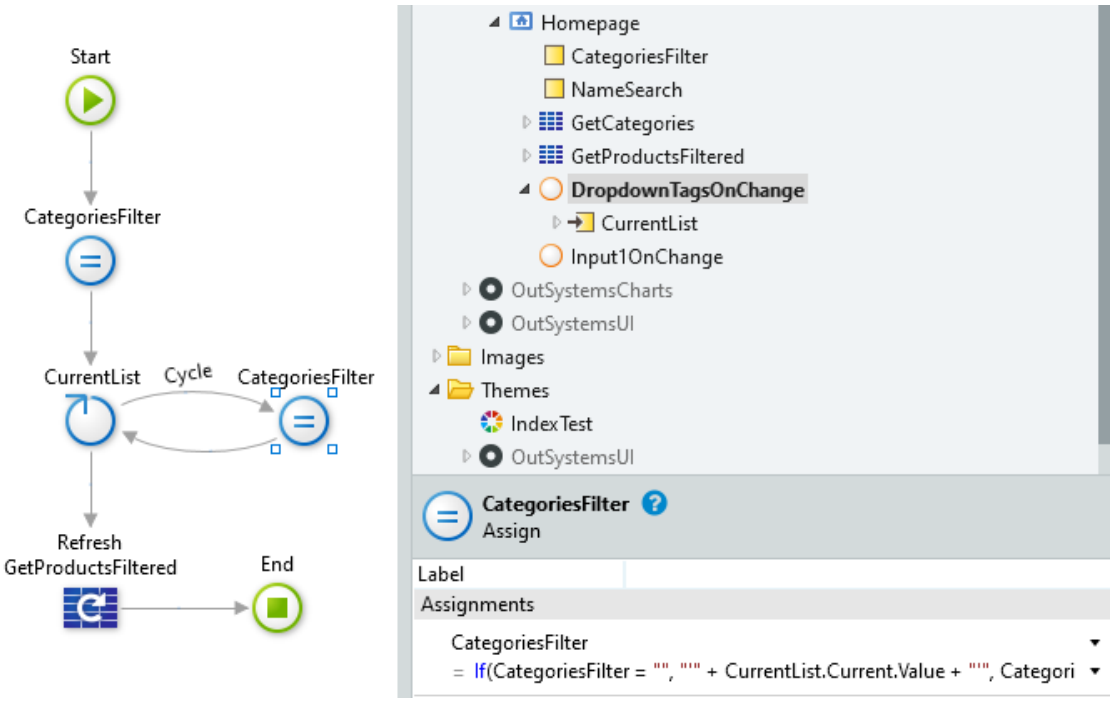


Figure 5.5: DropdownTagsOnChange logic flow: creating the coma separated text variable with the selected categories and refreshing the Aggregate with the new filter condition

5.1.2 Evaluation

For this evaluation, we tested 5 users from each group presented in Section 4.3.1: Non-Developers, Software Developers and OutSystems Developers.

Regarding the 5 OutSystems Developers, 3 of them had OutSystems certifications. All of them had experience with other programming languages (such as Java, C#) and affirmed to be Medium to High experienced users in OutSystems. Regarding their SQL knowledge, 3 of them considered having a high experience while the other 2 said that they had little experience or haven't worked with it in a long time.

Concerning the 5 Software Developers, all of them had previous experience with the OutSystems platform but none of them had OutSystems certifications. All of them also had a lot of knowledge about other programming languages (such as Java, Kotlin and C#). As for their SQL knowledge, only 1 of them declared to have a lot of experience while the other 4 answered that they had little experience or haven't worked with it in a long time.

The 5 Non-Developers had no experience at all with OutSystems and only 1 of them mentioned having some very basic experience with programming languages (such as Pascal and Visual Basic). Despite of that, all of them had experience with software tools like Microsoft Excel and Google Sheets (see full answers in Appendix A.1).

We implemented the testing scenario presented in Section 4.3.2 using version 11 of the Service Studio. The task that was asked for the users to complete was the same that is presented in that Section:

Given the current list of products and the current DropdownTags with all the existing categories in the right upper corner of the screen, complete the necessary logic to make the DropdownTags work for filtering the products by the category or categories selected on it.

For the Non-Developers and Software Developers, a small introduction to the platform preceded every test, in order to give them an idea of the IDE organization and basic concepts. All tests were done remotely and by giving the user remote control, in order to simulate, as close as possible, the same circumstances for every user test. For each user test, we collected the following information, which can be seen in detail in Appendix B:

1. Previous experience with Aggregates;
2. Familiarity with this type of problem;
3. Starting looking place to solve the task;
4. Resource to Google;
5. Difficulty using SQL;
6. Mention of the SQL IN operator;
7. Knowledge of the Index() built-in function;

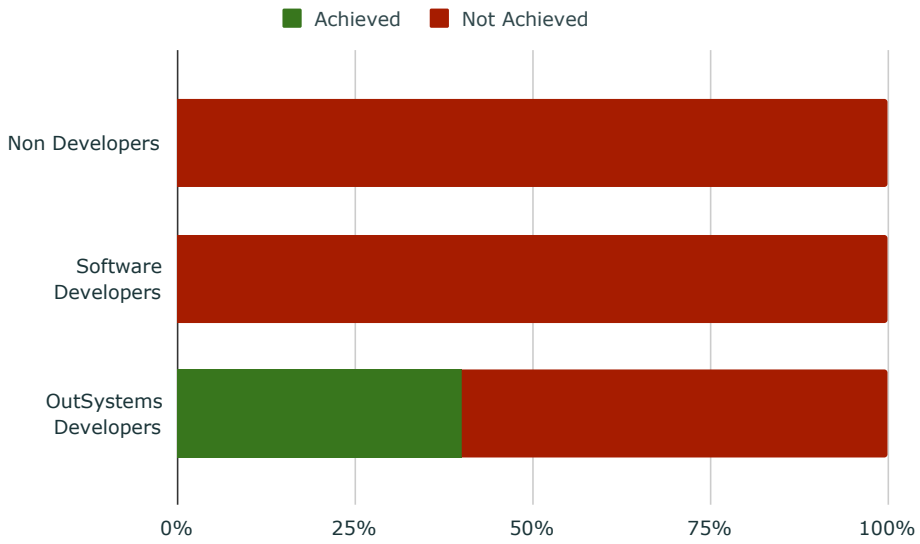


Figure 5.6: Success rate of the Current State phase tests by each group of users

8. Difficulty understanding the DropdowTags component;
9. Difficulty using the ForEach component;
10. Knowledge of system actions;
11. Difficulty choosing appropriate system actions;
12. Intended final solution;
13. Result regarding task completion;
14. Test duration.

The results regarding the success rate of this phase tests are presented in Figure 5.6. As was initially expected, none of the Non-Developers was able to complete the task. The users from this group started to get frustrated very early and, on average, only took 26 minutes and 52 seconds to give up. However, these tests were interesting in the sense that they gave us an idea of where someone that doesn't have any programming knowledge finds it more intuitive to start looking to solve this problem:

- 3/5 Non Developers started looking in the DropdownTagsOnChange Client Action;
- 2/5 Non Developers started looking in the GetCategories Aggregate.

Regarding the Software Developers group, none was also able to successfully complete the task. The most interesting observation taken from these tests was that all the Software Developers started by trying to solve the problem in the Filters Tab of one of the Aggregates.

When trying to add a new filter:

- 1 user tried to loop inside the expression editor;
- 2 users wanted to use a function like "contains";
- 1 user looked for the IN operator;
- 1 user wanted to use a function like "any";

When told that those functions and operators didn't exist there, the Software Developers tried to solve in the DropdownTagsOnChange Client Action, recurring to post-processing techniques. 3 of them used System Actions in their attempts. On average, Software Developers tried for 38 minutes and 33 seconds, and developed a lot more than the Non-Developers, but none of them was also capable of completing with success.

The third and final group of users, the OutSystems Developers group, was the only one where there were successful results, with 2 users being able to complete the task. In fact, 3 out of the 5 OutSystems Developers had already faced this problem, including the 2 users that were able to finish with success. 4 out of the 5 OutSystems Developers started by using the SQL tool and the other 1 started by trying to solve in the Aggregate Filters tab. From those 4 users that used SQL, 3 of them had difficulties using it. On average, OutSystems Developers tried for 39 minutes and 41 seconds. It is relevant to notice that while one of the users that successfully finished the task only took 13 minutes and 48 seconds, because he had recently faced this problem in his work and more than one time, the other one faced many difficulties and took 40 minutes and 15 seconds to complete.

It was also possible to observe that, by changing the Dropdowntags component to have an output parameter of Text List data type instead of the default DropdownItem List data type, the understanding of where the selected categories were being saved was much easier for the users.

5.1.3 Discussion

Overall, the most important final conclusions of the Platform Current State phase were:

- 7/12 users that didn't know that the IN operator doesn't exist in the Aggregates filters started looking there, while the other 3 started looking in the client action associated with the DropdownTags component;
- Only 4/15 users tried to use SQL, and only 1 of them didn't have difficulties using it;
- 5/15 users used list system actions in their solutions;
- Only 2/15 users tested were able to complete the task.

With this low success rate and observations, we could conclude that, at the current state of the OutSystems platform, solving use cases that involve filtering of persistent data by in-memory data it's too difficult and needs to be improved. Even using the SQL tool the users faced many difficulties, which proves that it isn't a viable alternative.

5.2 Alternative Approaches Analysis

Initially, 4 different solutions were proposed in Section 4.2 for the implementation of the IN functionality in the OutSystems platform, in order to improve the mashup of in-memory data and persistent data.

However, after discussing with the OutSystems stakeholders and interviewing OutSystems developers, we have come to the conclusion that the Sources tab of the Aggregates was already very complex for beginner users to understand, as this is where the Join actions between Sources are performed. Including the IN functionality in this section would only increase the already existing high complexity present on this section of the Aggregates. Besides that, no user, regardless of its group, looked into the Sources tab to try to solve the usability test of the current state. This way, the second proposed solution in that section was discarded.

Given the feedback received by the stakeholders and the fact that in the previous phase tests the Aggregates Filters was the most visited starting place, we decided to focus on prototyping the first solution, which evolves the addition of the IN operator in the Aggregate filters expression editor. We started by sketching solutions and Figma [1] was the software tool that we decided to use in order to elaborate the low fidelity prototype. For that, we followed an iterative sketching process, that will be described in the following sections.

5.2.1 New system action vs Editing ListFilter

In the Platform Current State phase, we observed that 5 out of the 15 users tested used at least one list system action in their solutions. This motivated us to also suggest the implementation of the IN operator as a list system action. This suggestion does not replace the implementation of the IN operator in the Aggregate filters expression editor. Instead, it works as a complement and, in order to achieve this, two alternatives were proposed:

1. Implement a new system action;
2. Add the IN operator in the already existing ListFilter system action .

5.2.1.1 New system action: ListContains

In Figure 5.7 we can see the main List System Actions that currently exist in the OutSystems platform. While the name IN for this functionality is adequate in the context of Aggregates, which is a context of database querying, here we found the name Contains, which was also very mentioned in the usability tests of the following phases, more adequate given that the usage context of the list system actions is more general.

The main advantage of having a new System Action is that it excludes the need of opening the expression editor to define the IN condition. As can be seen in Figure 5.8, in the input "Attribute to be compared" the user would only need to select which attribute from the SourceList he wants to use for the comparison and in the background the system would check, for each

[1] Figma https://www.figma.com/, July, 2021

Figure 5.7: Main system actions in the OutSystems platform

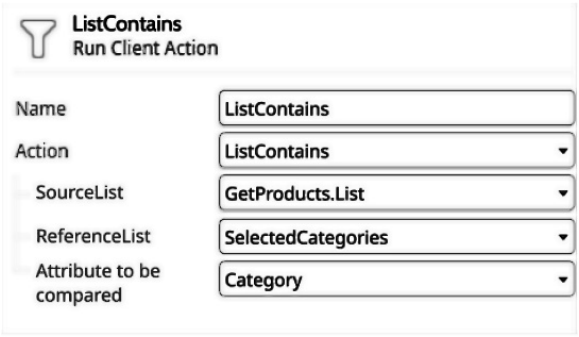


Figure 5.8: ListContains proposed design and attributes

item of the SourceList, if the specified attribute is contained in the items of the ReferenceList. The ReferenceList needs to be a list with only 1 attribute.

5.2.1.2 Editing the existing ListFilter action

During the Platform Current State tests, some of the users that tried to solve the task by using list system actions tried to solve it with the existing ListFilter. This allowed us to conclude that by creating a new list system action we would have to deal with the ambiguity between the new action and the already existing ListFilter action, given that using the IN operator can be seen as a way of filtering. To prevent that, in this solution we propose the addition of the IN operator in the Condition expression editor of the existent ListFilter system action. As represented in Figure 5.9, having the IN operator there the user can specify the condition in the same way he would specify in the Aggregate Filters expression editor.

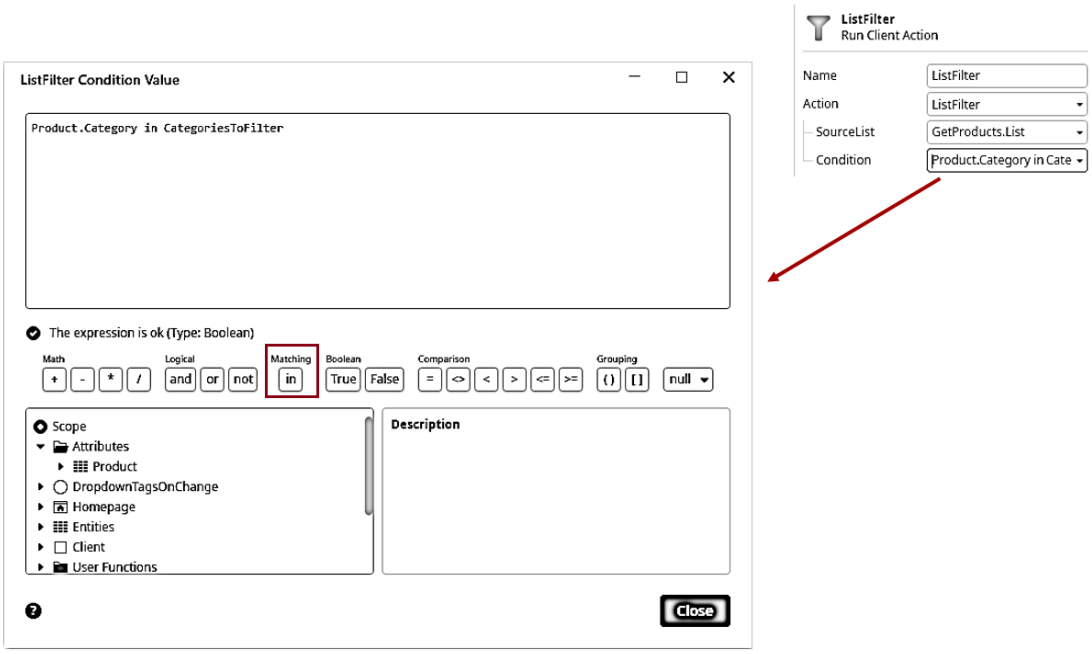


Figure 5.9: ListFilter action attributes and proposed addition in the condition expression editor

5.2.2 Use cases

What makes these proposals really interesting for the context of this thesis is that either one of them allows 4 extra use cases to be solved:

1. Filtering an in-memory list by the values in another memory list (needs to have only 1 attribute), specifying the attribute of the first list for comparison;

2. Filtering the output list of an external integration method by the values in the output list of another external integration method (needs to have only 1 attribute), specifying the attribute of the first list for comparison;

3. Filtering an in-memory list by the values in the output list of another external integration method (needs to have only 1 attribute), specifying the attribute of the first list for comparison;

4. Filtering the output list of an external integration method by the values in another memory list (needs to have only 1 attribute), specifying the attribute of the first list for comparison;

Looking in more detail to the following example, which corresponds to the use case number 3. In Figure 5.10, we have a GetPosts REST API method that returns a list of Posts, where each post has the attributes UserId, Id, Title, etc. Imagine that we also have a local list containing Users, where each user has the attributes Id, Name, Username, etc. We want to list only the posts that are from any of the users present in the local list of users. Assume that the Assign component was used to copy only the values of the Id attribute from the UsersList to a new

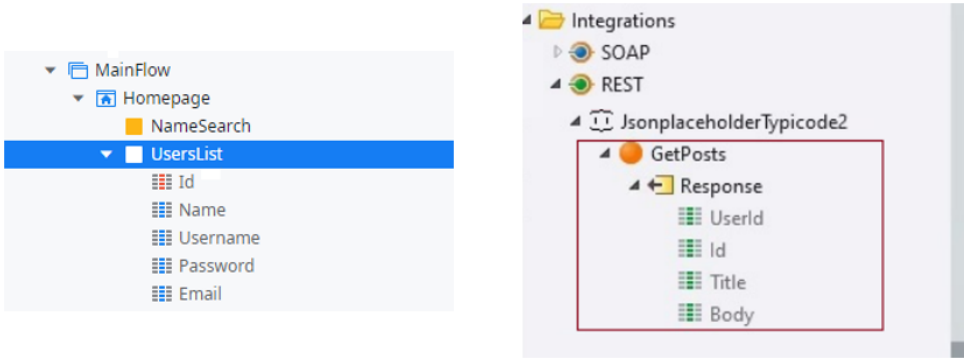


Figure 5.10: UsersList (Local Variable) and GetPosts (REST API method) structures

simple list, usersIdsList. This particular use case can be solved using the 2 proposed solutions as described below:

1. With the new ListContains, making the following assignments:
   SourceList = GetPosts.Response
   ReferenceList = usersIdsList
   Attribute to be compared = UserId

2. Having the IN operator in the ListFilter, making the following assignments:
   SourceList = GetPosts.Response
   Condition = Post.UserId in usersIdsList

Although no tests were performed in order to evaluate which alternative would be the best, at the end of every test of the Design and Implementation phase we asked the users what they thought of this alternative and all of them agreed that it would be beneficial to also have one of these solutions complementary to the main one.

6

Iterative Design, Implementation and Evaluation

Throughout this chapter, the design, implementation and evaluation process of each solution are presented in detail, from the first low-fidelity prototype until the final Service Studio implementation. This chapter includes 3 cycles of design, implementation and evaluation, one for each of the following sections:

1. Low-fidelity Prototype (6.1)
2. IN Right Parameter Selection Exploration (6.2)
3. Service Studio Prototype (6.3)

Each cycle follows the methodology presented in Section 4.3, with some small adaptations.

6.1 Low-fidelity Prototype

In the Platform Current State phase, it was concluded that most of the users that were not aware of the absence of the IN operation in the OutSystems Aggregates started looking for solving this problem in an Aggregate filters tab. With that in mind, we decided to elaborate a low-fidelity prototype (See Figures 6.1 and 6.2) to evaluate the implementation of the IN operator in that place. For that evaluation, we performed 15 usability tests, following the methodology presented in Section 4.3.

6.1.1 Design and Implementation

In our design and implementation of the low-fidelity prototype, we decided to focus on the following 3 main questions:

- **Q1 - Where should the IN operator be added in the expression editor?**

  In the first iteration of the prototype, we decided to add the new IN operator next to the LIKE operator (see Figure 6.3), which is also an SQL operator, and near the NOT operator,

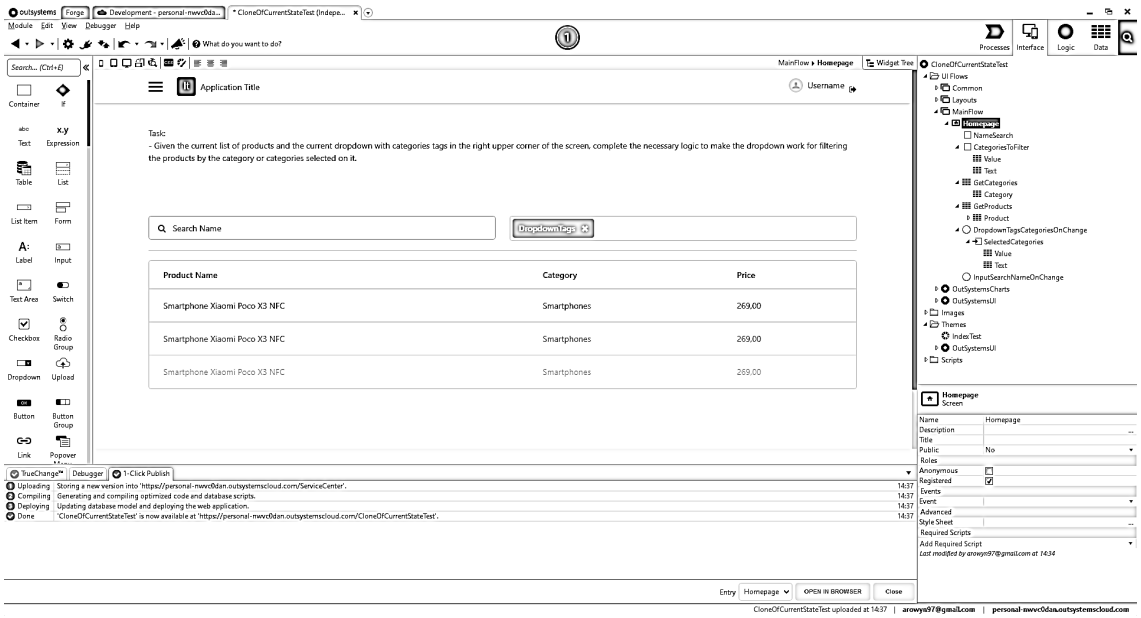


Figure 6.1: Proposed Solution Prototype: Homepage

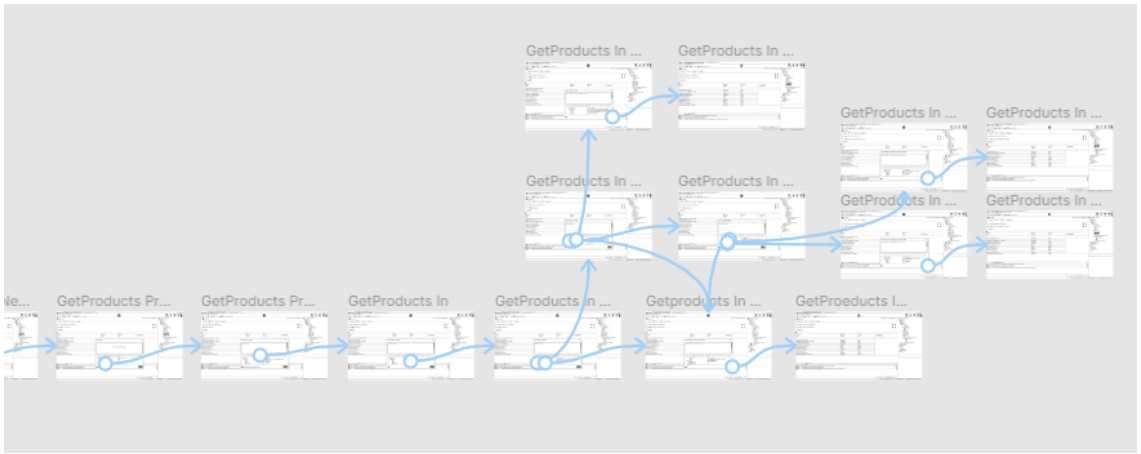


Figure 6.2: Example of the flow configuration (in Figma) to turn the low fidelity prototype testable

in order to be more intuitive that it can be used together with it to form the NOT IN functionality. The IN was placed visibly because, similarly to the LIKE operator, its use is most common in the Aggregates, unlike existing built-in functions that are widely used inside and outside the Aggregates. In the second iteration of tests, a new version, which is presented in Figure 6.4, for the placement of the new IN operator in the operators' bar was suggested and tested. This suggestion included the addition of labels for each group of operators, with the intuit of helping the users better understand where to look for the intended operation. A new group, called "Matching", was also added in order to accommodate the LIKE and IN operators, which are both very related to SQL and have higher complexity than the other operators.

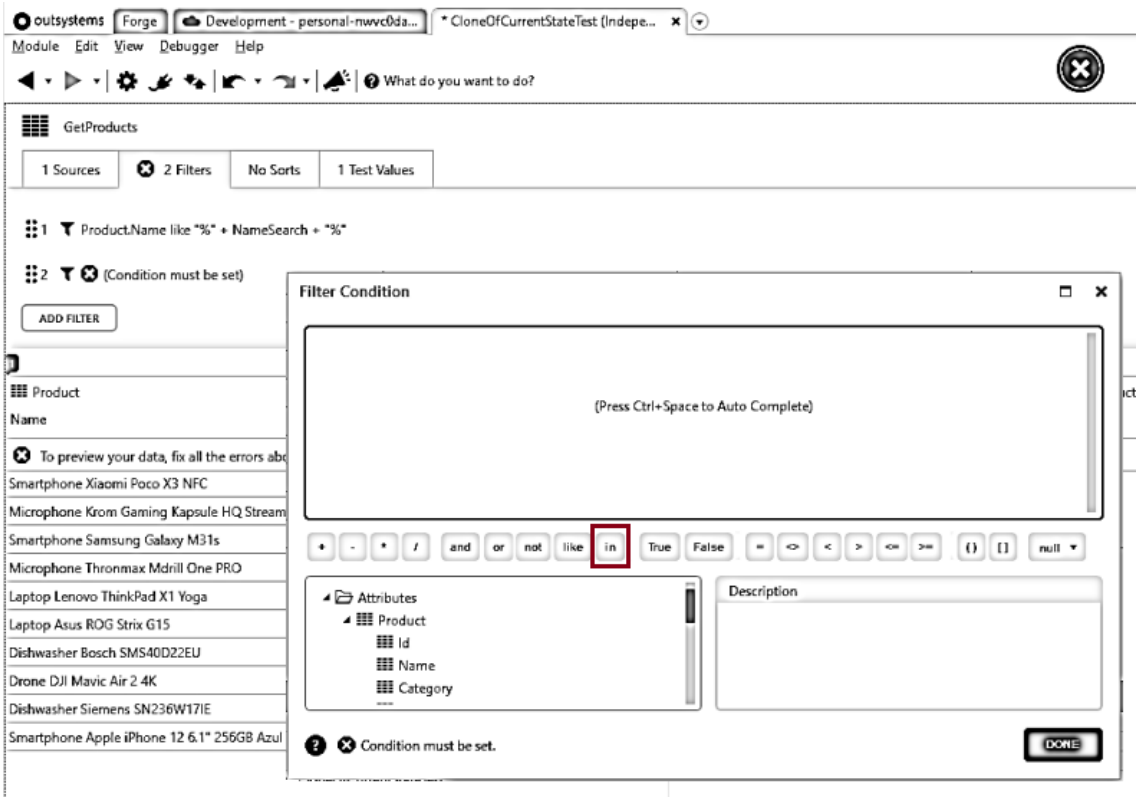


Figure 6.3: Proposed Solution Prototype: Addition of the IN operator in the "Add Filter"Expression Editor

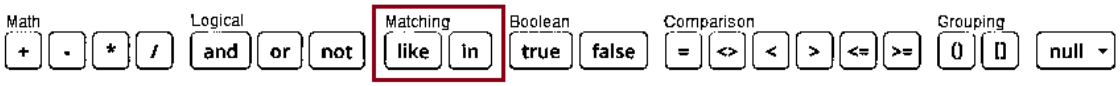


Figure 6.4: Operators Bar new proposed design

- **Q2 - Is the name “in” intuitive for all types of users?**

  With this prototype, we also intended to evaluate which is the best syntax for the desired new operator. In the previous phase, we had some users mentioning function names like "in", "contains"and "any". We decided to test the use of the SQL syntax "in", because the already existing operator LIKE also uses the same name as in SQL and the majority of the OutSystems Developers has SQL knowledge. With that in mind, we also wanted to check if the IN name could also be intuitive to users that don’t know SQL.

- **Q3 - What do users expect to put next to the IN?**

  If the list that the user wants to put next to the IN is a list with only one attribute, then the only concern should be that the type of that attribute is the same or can be implicitly converted to the same type as the value on the left side of the operator. But **what if the list has more than one attribute for each element?**

  In OutSystems language, there are several components that use, by default, data types that

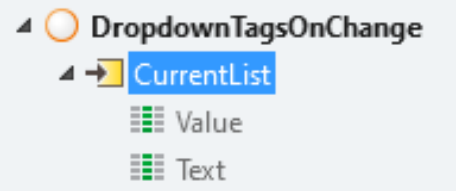


Figure 6.5: CurrentList structure

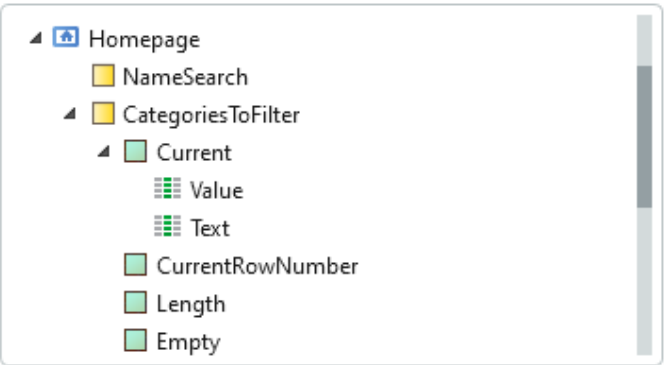


Figure 6.6: DropdownItem List "CategoriesToFilter"expanded options in the scope of the Agrgegate "Add Filter"expression editor

are composed of 2 or more attributes. The developed test scenario contains a DropdownTags component which, by default, creates a list of type DropdownItem record, named CurrentList, composed by 2 attributes (each CurrentList item has two attributes: Value and Text), as can be seen in Figure 6.5.

If we have a complex list like that, it is necessary to specify the attribute that the user wants to use for the comparison. This way, this prototype also aimed at evaluating what the users expect to put next to the IN operator when confronted with this problem. The main problem is that, in the current state of the platform, the only way to access attributes of a list is by expanding the property “Current” (see Figure 6.6). This name is not intuitive in this context, as we are not iterating the list.

We needed to find a solution that could allow a level of simplicity as close as possible to just selecting the whole list for the users with fewer skills, while also allowing the selection of attributes. The basic proposed solution was to use the IN operator like in SQL, requiring that the list to put on the right is already well defined with only one attribute:

Product.Category in CategoriesToFilter

Pros: This solution is very clear and simple if the list is of a simple type that only has 1 attribute (for example: list of text), which is most of the cases.

Cons: What if the list has 2 or more attributes? The user will need to leave the editor to convert the complex list to a simple list, only with the wanted attribute values for the comparison.

6.1.2 Evaluation

We tested our prototype with 5 users from each group presented in Section 4.3.1: Non-Developers, Software Developers and OutSystems Developers.

Regarding the 5 OutSystems Developers, all of them had OutSystems certifications and had experience with other programming languages (such as Java and C#) and affirmed to be Medium to High experienced users in OutSystems.

Concerning the 5 Software Developers, only 3 of them had ever worked with Outsystems and none of them had OutSystems certifications. All of them stated to be beginner users regarding OutSystems development but had a lot of knowledge of other programming languages (such as Java, C# and PHP).

All the OutSystems Developers and all the Software Developers had high knowledge of SQL (see full answers in Appendix A.2).

The 5 Non-Developers had no experience at all with OutSystems and only 2 of them mentioned ever learning some basic programming languages (such as HTML and CSS). 1 of them mentioned that he had made some basic work with SQL a long time ago. Despite that, all of them had experience with software tools like Microsoft Excel and Google Sheets (see full answers in Appendix A.2).

Our prototype consisted of an incomplete implementation of the testing scenario presented in Section 4.3.2, with the addition of the IN operator in the expression editor of Aggregate filters. It was incomplete because, as is common practice in usability testing, we focused on the happy path. Because of that, whenever a user tried to click in a functionality that was not implemented we, as moderators, would describe what would happen. Like all the other evaluations made in the context of this thesis, the task was the one also presented in that Section.

For the Non-Developers and Software Developers, a small introduction to the platform preceded every test, in order to give them an idea of the IDE organization and basic concepts. All tests were done remotely and by giving the user remote control, in order to simulate, as close as possible, the same circumstances for every user test. As this evaluation was conducted with an incomplete prototype, the users were less prone to exploration and many functionalities were not available and had to be only verbally described.

Because it was an exploratory prototype and we were still not sure what we wanted to consider as the final right way of solving the task, we didn't have in count the success rate in this evaluation. The following information was registered for each user and can be seen in more detail in Appendix C:

1. SQL knowledge;

2. Previous knowledge about the lack of the IN operator in Aggregates;

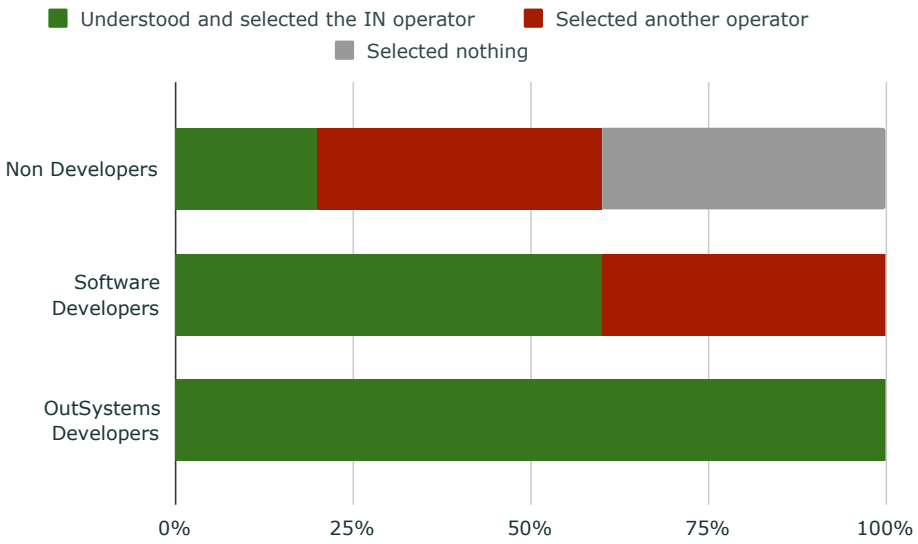


Figure 6.7: Selection of the operator by user group

3. Understanding of the SelectedCategories input parameter;
4. Operator selection;
5. Selection behaviour regarding the right parameter of the IN operator;
6. Opinion on the "Current"list property;
7. Opinion regarding the usage of the syntax "in"for the operator;
8. Opinion concerning the location of the operator in the prototype.

6.1.2.1 Q1 and Q2 - IN location and syntax results

The results concerning the selection of the IN operator are shown in Figure 6.7. As we can see, all OutSystems Developers and the majority of the Software Developers rapidly and correctly selected the IN operator. This allowed us to conclude that it was an adequate name and location for both those types of users. Regarding the Non-Developers, only one of them selected the IN operator. The majority of them started by choosing the "="operator (because they didn't understand that it doesn't work with more than one value) and then selected the "like"operator (probably because it was the operator used in the already existent filter example).

Although all the users found the IN name adequate for the operator when asked about it at the end of each test, CONTAINS was another name mentioned by many of them as a viable alternative. For the users that started constructing the condition with the selected categories saved in the CategoriesToFilter local list in mind, it was more intuitive to think with a CONTAINS operator, as the logic order is the inverse of the IN logic. However, both ways of thinking are correct, as shown in the following 2 examples:

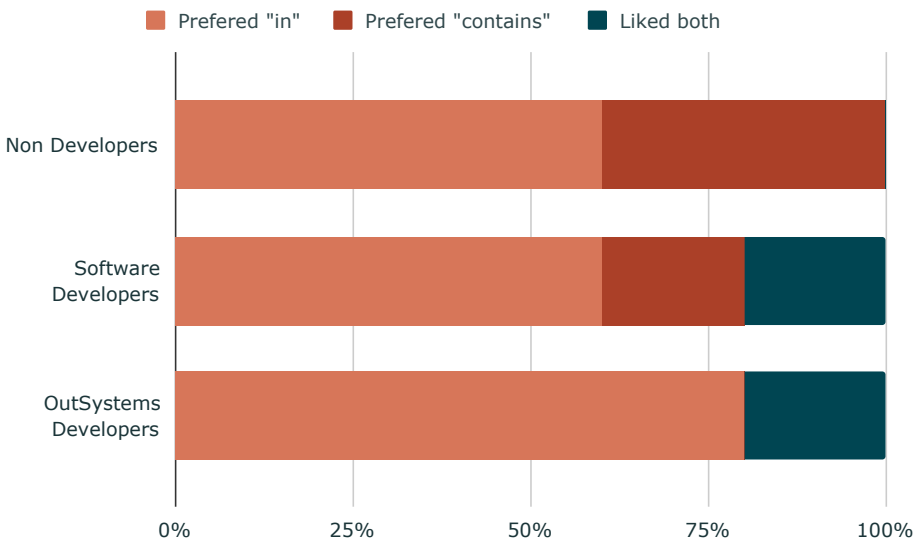


Figure 6.8: Operator name preference by user group

CategoriesToFilter **contains** Product.Category

or

Product.Category **in** CategoriesToFilter

When asked if they preferred the name IN or CONTAINS, the answers were distributed as presented in Figure 6.8. Although the name IN achieved good results regarding its selection and preference against CONTAINS, the results highlighted the importance of also using the name Contains in the description of the IN operator.

The feedback received concerning the new operators' bar design was positive, being considered a good new asset, and every user agreed with the location of the new IN operator in the prototype.

6.1.2.2 Q3 - IN right parameter selection results

From the results presented in Figures 6.9, 6.10 and 6.11, we could conclude that users with the lowest technical skills just expected to select the whole list. On the other hand, users with higher technical skills showed more awareness that they needed to specify an attribute, and were faced with the absence of a way of doing it.

It was also proven that the name "Current"dissuaded users to search there for the attributes of the list, as 62,5% of the users that started by expanding the list ended up either going back and selecting the whole list or selecting nothing.

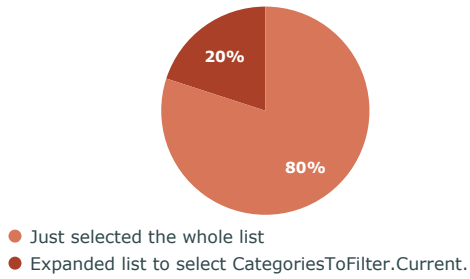


Figure 6.9: Selection by Non Developers

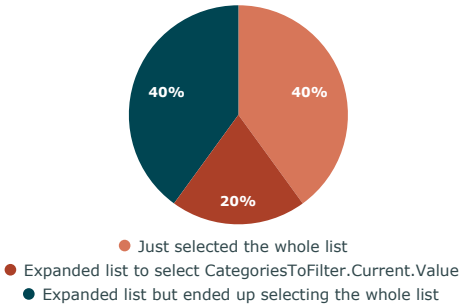


Figure 6.10: Selection by Software Developers

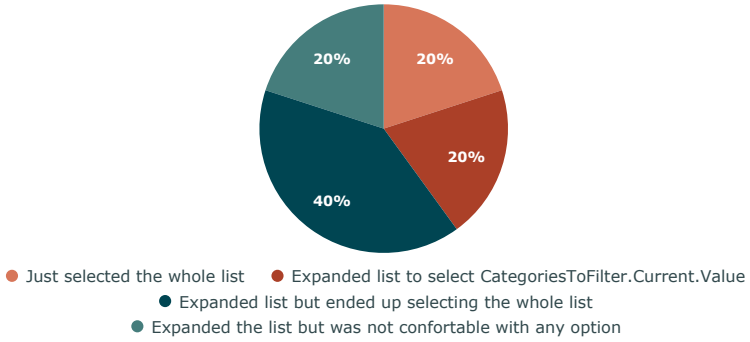


Figure 6.11: Selection by OutSystems Developers

6.2 IN Right Parameter Selection Exploration

If we look again at Figure 4.9, we can see that this section corresponds to the secondary path (represented by the purple color) of our workflow. In order to try to mitigate the Cons of the basic solution presented at the end of Section 6.1.1, we resorted to brainstorming techniques and ended up proposing 4 alternative solutions:

1. IN operator be used like a function, with parentheses and with 2 arguments, being the first argument the list and the second argument the list attribute to be compared:

   Product.Category in (CategoriesToFilter, Text)

   Pros: Making the IN a function makes it consistent with a syntax that is already widely used in OutSystems (example: If(), Index())

   Cons: Currently the attributes of the list are not available at the scope of the aggregate filters expression editor.

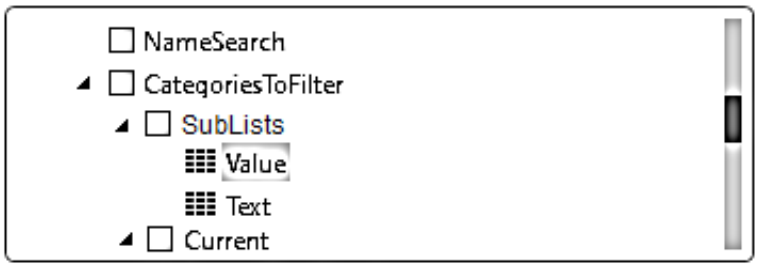


Figure 6.12: Proposed new list property SubLists

2. IN operator be used with two keywords, the word “by” would be added automatically if detected it was a list with more than one attribute.

   Product.Category in CategoriesToFilter by Text

   Pros: This one is similar to solution 2, but uses keywords that make it closer to natural language.

   Cons: Has the same problem as solution 2, currently the attributes of the list are not available at the scope of the aggregate filters expression editor. The expression “by” may not be intuitive enough.

3. Have a new list property besides “Current” that would have separate sub lists for each attribute (a list of Value and a list of Text), with an intuitive name (ex: SubLists)

   Product.Category in CategoriesToFilter.SubLists.Text

   Pros: It would make the whole syntax of the IN easily made only with “clicks”, which increases the discoverability and follows a path where writing lines of code is increasingly unnecessary;

   Cons: This new property would appear for every list through the OutSystems Platform editors and could encourage new user errors by introducing some ambiguity between the new property and the already existing “Current” property.

4. Make the IN operator only work with simple lists, similarly to the basic solution, and have a new function that converts “complex lists” to “simple lists”.

   Product.Category in ListSelect(CategoriesToFilter, Item => Item.Text)

   Pros: The user does not need to leave the editor to convert the complex list to a simple list and can use it together with the IN operator.

   Cons: The function will be hidden in the expression editor and beginner users won’t know of its existence. Probably only with good error messages and auto-corrections users will find it.

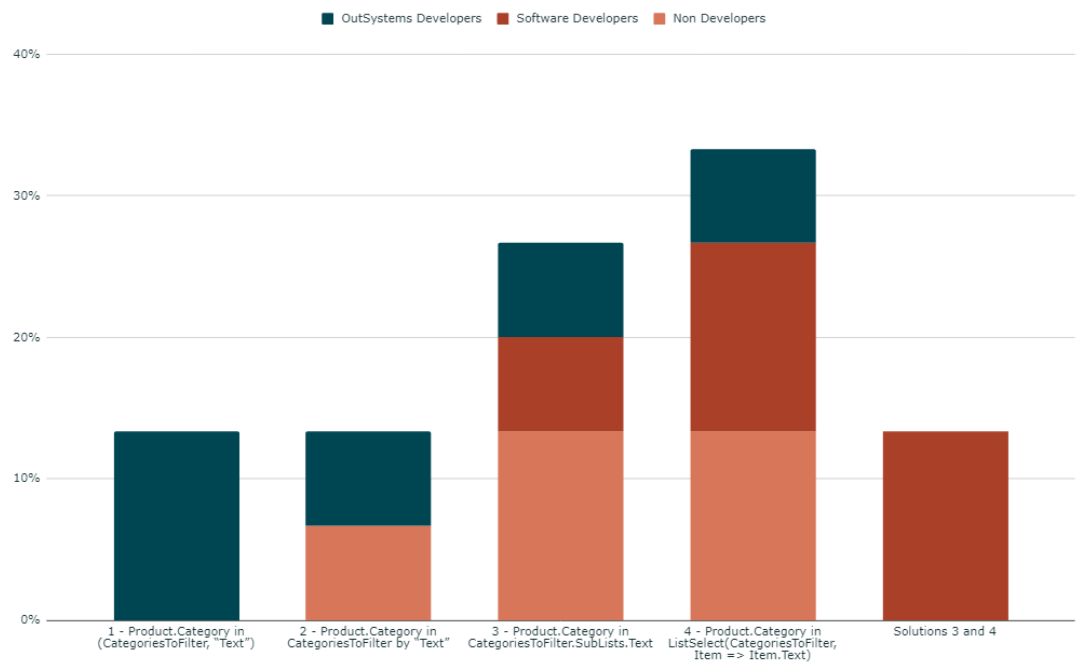


Figure 6.13: Solutions preference by user group

In order to determine the best alternative solution, we resorted to User Interviews. The users interviewed were the same from the Low-fidelity prototype phase and the detailed answers can be seen in Appendix D. The results are presented in Figure 6.13. As we can see, solutions 3 and 4 were the most voted ones and were also the only ones with votes from each group of users, which indicates that it could make the use case simpler to users with lower technical skills without compromising the experience for the most expert ones. In the following section, we will describe the design, implementation and evaluation of the most voted solution, the ListSelect() Function solution (Solution 4).

6.2.1 ListSelect Function Prototype

Having determined the best alternative solution in the previous section, we resorted to Figma to develop a low-fidelity prototype that would allow us to evaluate the usability of the proposed new function ListSelect().

Differently from what was done in the other evaluations, we decided to not test this solution with Non-Developers, as this solution involves the addition of a new function that it's of high complexity. This way, we only tested our ListSelect() function prototype with 6 users (2 OutSystems Developers and 4 Software Developers).

Regarding the 2 OutSystems Developers, both of them had OutSystems certifications and had experience with other programming languages (such as .NET) and affirmed to be expert OutSystems users. Regarding their SQL knowledge, they answered that they had already worked with it several times.

Concerning the 3 Software Developers, none of them had ever worked with Outsystems. All of them stated to be beginner users regarding OutSystems development but had a lot of knowledge of other programming languages (such as Java and C#). 2 of them mentioned having high experience using SQL while the other 2 said that they had only worked with it a few times or very long ago (see full answers in Appendix A.3).

In the following section, we propose and describe mechanisms that can help and guide the user through the process of using the new function to solve the testing scenario presented in Section 4.3.

6.2.1.1 Rapid Development

Through our evaluations, we took notes regarding the following aspects which can be seen in full in Appendix D:

1. User's SQL knowledge;
2. User's IN operator knowledge;
3. Ease of understanding and selection of the IN operator in the prototype;
4. Expectations regarding what to put next to the IN operator;
5. Difficulty founding the hint system;
6. Error message clarity;
7. User's first perception of the ListSelect() function;
8. Expressiveness of the symbol "=>";
9. User opinion regarding the name Record.

In our implementation, if the user selects an object with a data type that has more than one attribute next to the IN operator, a helping system is introduced, represented by a light bulb icon that appears on the left side of the editor window, in-line with the place where the error occurred (the error is indicated by a red underline), as shown in Figure 6.14.

The reason why it is placed there is to avoid overlapping and hiding of the remaining expressions written. The light bulb icon was initially presented without fill, but it was later painted entirely in orange to make it more clearly visible given that, in the first tests, the users weren't noticing the new icon appearing in the editor. A new phrase "Check the light bulb for suggestions." was also added at the end of the error message in order to encourage people to look up for the suggestions of correction. Initially, we called it "hint bulb" instead of "light bulb" but that name was proven ambiguous by some users that after reading thought it was referring to the icon on the left bottom corner of the editor composed by a question mark icon. We also concluded that splitting the error message in two lines helped to make users read the error

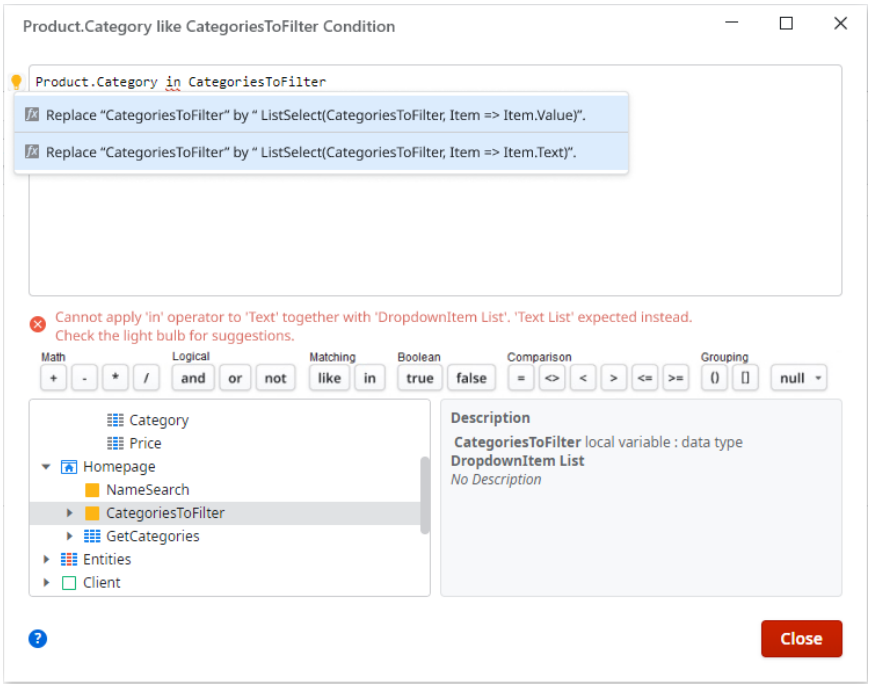


Figure 6.14: ListSelect Prototype: Aggregate filter condition expression editor

message until the end, as in the initial tests some of the users would only read the first part of the message where the error is explained.

By clicking on the light bulb, suggestions are presented to the user to select. Clicking in a suggestion will auto-replace the expression with the selected suggestion expression. It is important to note that in this prototype the new ListSelect function was also added in the User Functions but, as initially expected, no user reached it through the scope tree.

Regarding the new function syntax, we started by using the name Record to represent each list element, but the first 3 users tested had difficulty understanding it. We decided to change it to Item and we got better results with the following 3 users. All 4 developers and the 2 OutSystems Developers easily understood the "=>" icon (similar to the "arrow function" in other languages).

6.2.2 Outcome

All the proposed solutions concerning the selection of the right parameter of the IN operator are very interesting regarding the selection of one property in the context of Relational Databases, where each entity usually is defined by several attributes. OutSystems works with those kinds of databases and, consequently, there are several use cases where OutSystems users have lists of entities, which usually are lists with more than one attribute for each item, and they only pretend to use one of those attributes in their logic. It is worthy of note that the ListSelect function could also be used outside of Aggregates (e.g: used inside the list system actions ListFilter, ListSort, etc.), allowing the users to express more declarative code, in an excel-like manner. It would

even allow scenarios such as making the cast of a list or adding the IVA value to each element of a list.

However, after completing the 6 tests for this prototype we found an easier way to solve this specific problem in the DropdownTags component itself: the user could create a new version of the component where he changes the default CurrentList that comes with the component to the type intended and then make the mapping for only one value. This way, when the user gets to the Aggregate filters expression editor he doesn't need to deal with more than one attribute in the list with the selected categories. For this reason, the basic solution initially presented at the end of Section 6.1.1 was the one that ended up being implemented in the next phase, as a Service Studio prototype.

6.3 Service Studio Prototype

As we just justified at the end of the previous section, we decided to implement the IN operator in the Service Studio as working with simple lists (lists that only have one attribute). This decision was due to the finding of a simpler solution that would remove the complexity of the complex list in the component itself.

6.3.1 Implementation

In the Service Studio implementation, we decided to add a new group of operators in the filters Expression Editor, the "Matching" group, and new validations to ensure that an IN condition is correctly built. In order to be a valid IN condition, the following validations must be satisfied:

- The data type of the left parameter must be either a basic type or an entity identifier;
- The right parameter of the expression needs to be of type List;
- The list needs to have only one attribute;
- The elements of that list need to be of the same type or implicitly convertible to the left parameter type in the expression;

The new operators' group "Matching", which contains the operators LIKE and IN, was implemented in order to only appear in the context of an Aggregate filter. As the operator LIKE was previously already made available in the intended place, we only needed to make the IN operator also available there. In the rest of the IDE, this group is not visible when the user opens the Expression Editor and neither can these two operators be recognized as valid, even if the user tries to write them.

For users that never used the IN operator in another language, it was needed to find a way to make them understand that what needs to be in the right side of this operator is a list of values. This way, it was decided that we needed to add the following new error message that appears in the expression editor in case the user tries to put something that is not of type list after the

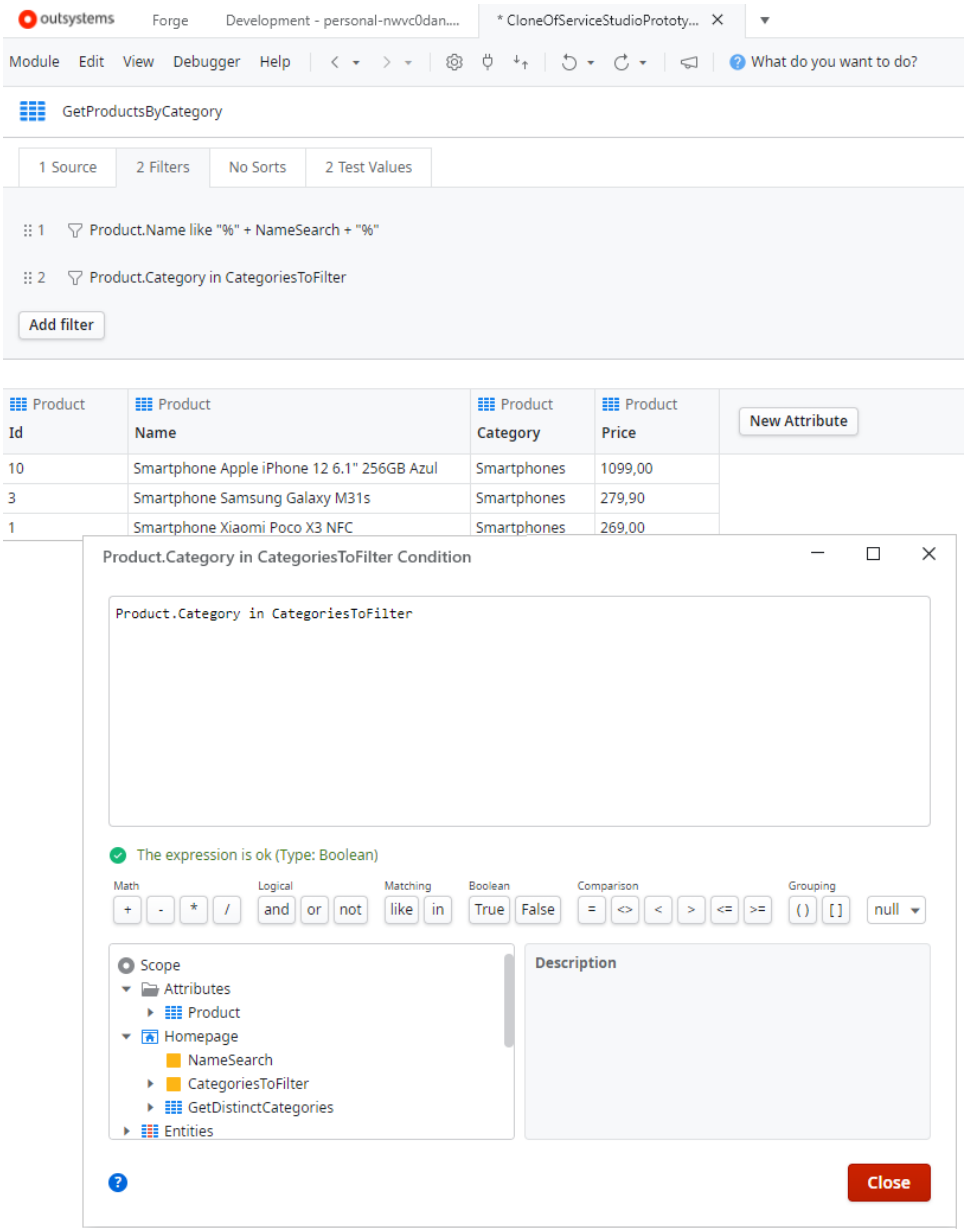


Figure 6.15: Aggregate filter condition with IN operator in the Service Studio implementation

IN operator: **"Cannot apply 'in' operator to 'Text' together with 'Text'. The right side of the 'in' needs to be a list".** If the user tries to put a list that has more than one attribute next to the IN operator, the following message is shown: **"Cannot apply 'in' operator to 'Text' together with 'DropdownItem List'. The right side of the 'in' needs to be a list of a Basic Type or Entity Identifier".** We also added the following description that appears when the user hovers the mouse over the IN button: **"Matches any value in a list of values"**.

Given that in the usability tests of the first prototype we observed that some users tried to use "()" before selecting the list to put next to the IN operator, similarly to what happens in SQL, we decided to allow both syntaxes in the Service Studio implementation: **"Entity.Attribute in**

List" and **"Entity.Attribute in (List)"**

With our implementation of the new IN operator in the Aggregate filters expression editor, the following use cases become possible to solve with Aggregates:

- Check if an attribute of the Aggregate source entity matches any of the values in a local list variable (Note: only works if the local list only has one attribute);
- Check if an attribute of the Aggregate source entity doesn't match any of the values in a local list variable (Note: only works if the local list only has one attribute);

At the current version of the OutSystems platform, the user is not allowed to access in the expression editor of an Aggregate filter to another Aggregate output list or another Data Action output list. For efficiency reasons, if the user tries to access these structures, the following error message is displayed: **"Only the Current, CurrentRowNumber, Length and Empty properties of Lists from 'Another Aggregate Name' Aggregate / 'Data Action Name' Data action can be used in 'Aggregate Name' Aggregate"**. With the addition of the IN operator in the Aggregate filters expression editor, we suggest opening an exception in order to also allow the following use cases:

- Check if an attribute of the Aggregate source entity matches any of the values in the output list of another aggregate (Note: only works if the output list of the other aggregate only has one attribute);
- Check if an attribute of the Aggregate source entity matches any of the values in the output list of a Data Action (Note: only works if the output list of that Data Action only has one attribute);

6.3.2 Evaluation

For this final evaluation, we performed 15 usability tests following, once again, the methodology presented in Section 4.3.

Regarding the 5 OutSystems Developers, 3 of them had OutSystems certifications. All of them had experience with other programming languages (such as C++, Java and Javascript) and affirmed to be Medium to High experienced users in OutSystems. Regarding their SQL knowledge, 4 of them considered having high experience while the other 1 said that he had little experience or haven't worked with it in a long time.

Concerning the 5 Software Developers, only 1 of them had previous experience with the OutSystems platform but didn't have any OutSystems certifications. All of them had a lot of knowledge about other programming languages (such as Java, C# and Python). As for their SQL knowledge, 4 of them declared to have a lot of experience while the other 1 answered that he had little experience or haven't worked with it in a long time.

The 5 Non-Developers had no experience at all with OutSystems or other programming languages. Despite that, 1 of them had tried to use SQL and all of them had experience with software tools like Microsoft Excel and Google Sheets (see full answers in Appendix A.4).

We implemented our solution in the Service Studio itself, in order to make it comparable to the results from the Platform Current State phase concerning the level of functionality. We made a small change in the testing scenario regarding the initial state of the program, justified by our finding in Section 6.2.2: we changed the DropdownTags component so that its output list had only one attribute, making it a simple list.

For the Non-Developers and Software Developers, a small introduction to the platform preceded every test, in order to give them an idea of the IDE organization and basic concepts. All tests were done remotely and by giving the user remote control, in order to simulate, as close as possible, the same circumstances for every user test. The information taken from each test was the following (see Appendix E for full results):

1. Previous experience with Aggregates;
2. Familiarity with this type of problem;
3. Starting looking place to solve the task;
4. Resource to Google;
5. Difficulty using SQL;
6. Previous use of the IN operator;
7. Mention of a "Contains"operator;
8. Difficulty understanding the DropdownTags component;
9. Solution thoughts;
10. Result regarding task completion;
11. Test duration;

As we can see in Figure 6.16, with our final solution all the OutSystems Developers and almost all the Software Developers were able to finish the task with success. The biggest improvement achieved was, undoubtedly, with Software Developers. These users, even without having experience using the OutSystems platform, became able to perform the task with our new solution. This can be explained by the right choice of placement for the new feature and by how simple it is to use for those users, given their background and experience writing conditions. As we saw in Section 5.1.2, in the evaluation of the Current State no Software Developer was able to complete the task with success. This was mainly because the currently existing solutions require a much higher knowledge of the OutSystems language and the available features of the Service Studio.

Unfortunately, it remained not possible for any Non-Developer to finish the task successfully. It is important to note that this group of users had no programming experience at all, and were getting their first contact with the OutSystems Service Studio. Given that the Service Studio is

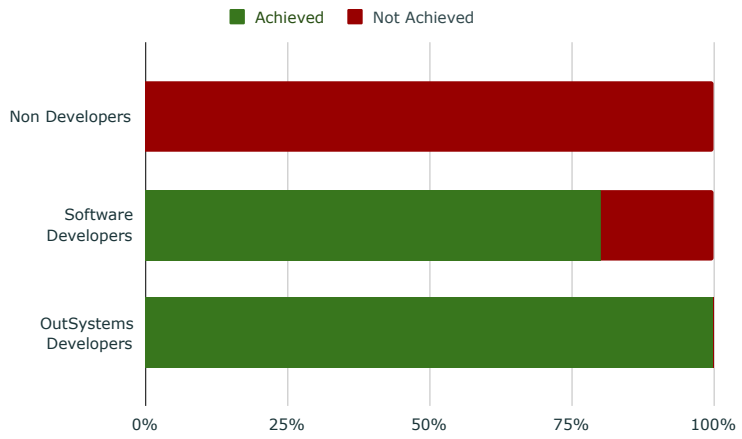


Figure 6.16: Success rate of the Service Studio phase tests by each group of users

a very complex tool, we can argue that a great part of the problem for both Non Developers and Software Developers that had never used OutSystems was to know where to look. For Non-Developers, that difficulty was even further aggravated by the lack of understanding of conditions formulation and database basic concepts.

6.4 Results Comparison

In this section, we compare the most relevant results from our first evaluation phase, the Current State phase, with the ones from our final evaluation, the Service Studio Prototype phase.

Success rate

As can be seen in Figure 6.17, we achieved a success rate progress of 80% regarding the Software Developers group and a progress of 60% for the OutSystems Developers group. In total, the registered progress concerning all the 3 groups of users was 46.67%. Only with the Non-Developers group we weren't able to achieve better results.

Logic flow

In the current state, it would take an Assign (or ListAppendAll system action) component in the event of the DropdownTags to copy the selected values to a screen global variable, together with around 7 components in a server or data action, to complete the necessary filtering logic without using SQL or the Index() function (which only works with the Text data type). All that process can now be done inside the event of the DropdownTags component, using only an Assign component and 1 Aggregate that has the capability to filter by himself. This way, besides our successful increase in the task completion rate, we were also able to reduce the complexity of the development flow, as shown in Figure 6.18.

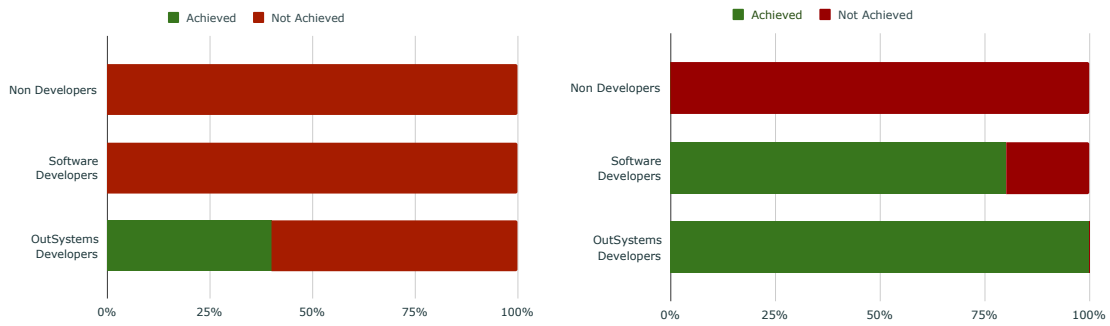


Figure 6.17: Success rate by each user group: before (left) and after (right) new solution implementation

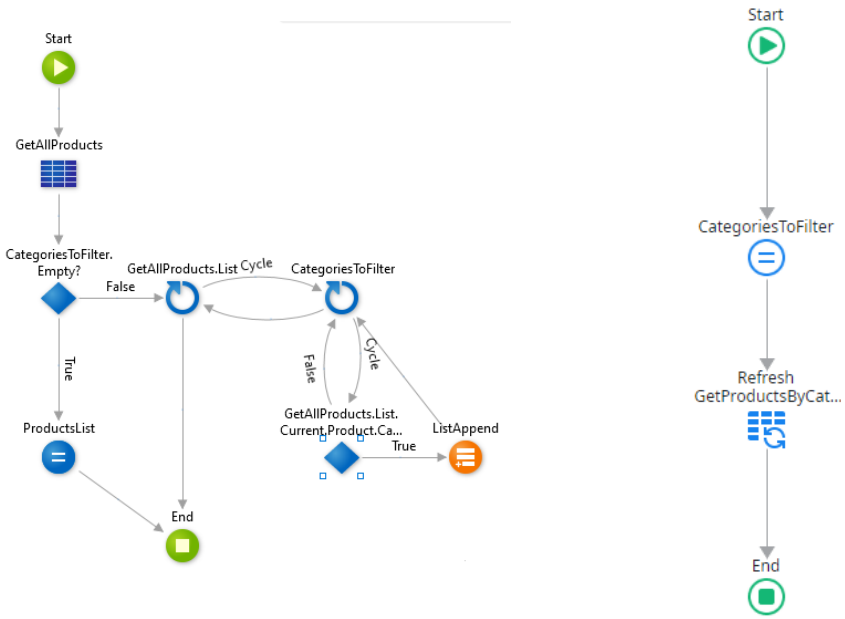


Figure 6.18: Logic flow to filter the products: before (above) and after (below) new solution

Table 6.1: Average test duration by User Group: before (left) and after (right) new solution implementation

	Current State		Service Studio Prototype	
	Average Test Duration (h:m:s)	Standard Deviation (m:s)	Average Test Duration (h:m:s)	Standard Deviation (m:s)
Non-Developers	00:26:52	4 min 37 sec	00:26:17.2	8 min 37 sec
Software Developers	00:38:33	4 min 57 sec	00:25:15.2	9 min 12 sec
OutSystems Developers	00:39:40.6	17 min 3 sec	00:06:42	3 min 55 sec

Tests duration

Concerning the duration of the tests for both the Current State and the Service Studio prototype evaluation phases, presented in Table 6.1, we can see that, in accordance with what we saw in the evolution of the success rate, the average test time of the Non-Developers group remained very similar. This means that these users still got frustrated and gave up as fast as in the Current State.

For the Software Developers group, there was a decrease in the average test duration which, together with the improvement in the success rate previously presented for that group, can be interpreted as a good indicator that it was easier for them to understand what they had to do and where to develop.

Regarding the third and final group, the OutSystems Developers, it was the group with the most impressive decrease in the average test duration. For these users, the solution for this use case became very clear and straightforward.

The lowest the value of the standard deviation value is, the more confident we are that our conclusions apply to the real total population of each of these groups.

7

Conclusion and Future Work

In this chapter, we present the final conclusions of this dissertation, while answering the research questions indicated at the beginning of this document and presenting what can still be improved in future work.

7.1 Conclusions

In Section 1.3 we presented the following research questions which we will now answer. It is worth noticing that, given the samples size, these answers are merely indicative and are not definitive answers.

Research Question 1: Is intersecting an in-memory list with the results of a database query a common problem?

Answer: **Yes**. In our research (see Section 4.1 for more details) we found out that the IN operator has been highly requested in the OutSystems Community and we were even able to determine that 8.4% of all the SQL Select statements that cannot be represented by Aggregates use the IN operator for intersecting an in-memory list with database results. To add to these findings, we were also able to determine that approximately 47.7% of the customers have used one or more Aggregate that uses the Index() function, which is one of the main alternatives that the users currently use.

Research Question 2: Are there other solutions to this problem that can solve more use cases?

Answer: **Yes**. In Section 5.2 we analyzed and designed 2 solutions that can solve use cases regarding the intersection between in-memory data (either as internal data or as data coming from an external integration). Although we determined that those weren't the best solutions for solving our main use case, we ended up concluding that also having them could add substantial value to the platform.

Research Question 3: Is it possible to augment the expressiveness of OutSystems Aggregates for this use case in a way that is natural for experienced developers?

Answer: **Yes**. By using the same syntax for the operator as in SQL, experienced developers found it very familiar and easy to understand. Even for developers that never used the IN operator in SQL, there are other programming languages that use this syntax and that they might have had contact with.

Research Question 4: Is it possible to augment the expressiveness of OutSystems Aggregates for this use case while making it accessible to non-experienced users?

Answer: **No**. Even with the addition of the IN operator in the Aggregates and independently of the syntax used, non-developers still weren't able to complete the task. This is largely due to the complexity of the Service Studio tool, which makes it very hard for people that never used the platform to know where to look. This is further aggravated in the case of users without any programming knowledge or databases knowledge, which was the case of the non-developers group. This way, we concluded that, given the complexity of the Service Studio, in future studies we should take into account that it is difficult to test new features with users that don't have previous experience using the platform.

7.2 Future Work

In Section 5.2, we presented 2 designs that address use cases of mashup between in-memory data, without evolving Aggregates. Although we started the process of sketching and design, we had to focus on one main solution for this thesis and, because of that, we didn't get to the evaluation phase for these 2 solutions which would allow us to choose the best design. This way, we propose as future work further research concerning a solution that would allow augmenting the OutSystems language to also be able of solving use cases of mashup between in-memory data.

In Section 6.2, we started addressing the issue of the complex lists as the right parameter of an IN condition. We performed a brainstorm of solutions, followed by user interviews to decide the best one. After that, we even performed 6 usability tests with the winning solution, the ListSelect() function. Those tests achieved promising results and the expressiveness power that such a function with a lambda expression, which does not need to be restricted to Aggregates, could bring to the OutSystems language is definitely worth researching.

BIBLIOGRAPHY

[1] J. Akbarnejad, G. Chatzopoulou, M. Eirinaki, S. Koshy, S. Mittal, D. On, N. Polyzotis, and J. S. V. Varman. "SQL QueRIE Recommendations". In: *Proc. VLDB Endow.* 3.1–2 (Sept. 2010), pp. 1597–1600. ISSN: 2150-8097. DOI: 10.14778/1920841.1921048 (cit. on p. 19).

[2] M. Angelaccio, T. Catarci, and G. Santucci. "Query by diagram: A fully visual query system". In: *Journal of Visual Languages & Computing* 1.3 (1990), pp. 255–273 (cit. on p. 23).

[3] E. Bakke and D. R. Karger. "Expressive query construction through direct manipulation of nested relational results". In: *Proceedings of the 2016 International Conference on Management of Data*. 2016, pp. 1377–1392 (cit. on p. 24).

[4] S. S. Bhowmick. "We Don't Need No Education: From Building for Coders to Building for Users". In: *Proceedings of the 2nd Joint International Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA), Amsterdam, The Netherlands, 30 June 2019*. Ed. by A. Arora, A. Bhattacharya, and G. H. L. Fletcher. ACM, 2019, 2:1. DOI: 10.1145/3327964.3328491 (cit. on p. 2).

[5] C. R. Borges and J. A. Macıas. "Feasible database querying using a visual end-user approach". In: *Proceedings of the 2nd ACM SIGCHI Symposium on Engineering Interactive Computing System, EICS 2010, Berlin, Germany, June 19-23, 2010*. Ed. by N. Sukaviriya, J. Vanderdonckt, and M. Harrison. ACM, 2010, pp. 187–192. DOI: 10.1145/1822018.1822047 (cit. on p. 2).

[6] T. Catarci, M. F. Costabile, S. Levialdi, and C. Batini. "Visual query systems for databases: A survey". In: *Journal of Visual Languages & Computing* 8.2 (1997), pp. 215–260 (cit. on pp. 2, 18, 19, 24).

[7] T. Catarci and G. Santucci. "Diagrammatic vs textual query languages: a comparative experiment". In: *Working Conference on Visual Database Systems*. Springer. 1995, pp. 69–83 (cit. on p. 23).

[8] Chartio. *Filter, Group, and Aggregate Operators in Visual SQL*. URL: %5Curl%7Bhttps://chartio.com/docs/visual-sql/operators/#filter-operators%7D,%20accessed%202021-02-16 (cit. on p. 27).

[9] Chartio. *We Made SQL Visual - Why and How*. https://chartio.com/blog/why-we-made-sql-visual-and-how-we-finally-did-it/, accessed 2021-02-16 (cit. on p. 27).

[10] D. Cohen, M. Lindvall, and P. Costa. "Agile software development". In: *DACS SOAR Report* 11 (2003), p. 2003 (cit. on pp. 6, 7).

[11] J. Danaparamita and W. Gatterbauer. "QueryViz: Helping Users Understand SQL Queries and Their Patterns". In: *Proceedings of the 14th International Conference on Extending Database Technology*. EDBT/ICDT '11. Uppsala, Sweden: Association for Computing Machinery, 2011, pp. 558–561. ISBN: 9781450305280. DOI: 10.1145/1951365.1951440 (cit. on p. 19).

[12] Devart. *Devart dbForge Query Builder*. https://www.devart.com/dbforge/sql/querybuilder/resources.html, accessed 2021-02-16 (cit. on p. 27).

[13] A. Dix. "Interactive Querying-locating and discovering information". In: *Second Workshop on Information Retrieval and Human Computer Interaction*. Citeseer. 1998 (cit. on p. 21).

[14] C. Floyd. "A systematic look at prototyping". In: *Approaches to prototyping*. Springer, 1984, pp. 1–18 (cit. on p. 8).

[15] Gartner. *Magic Quadrant for Enterprise Low-Code Application Platforms*. https://www.gartner.com/doc/reprints?id=1-24TEFBCB&ct=201214&st=sb, accessed 2021-02-15 (cit. on p. 109).

[16] T. Haigh. "How Data Got its Base: Information Storage Software in the 1950s and 1960s". In: *IEEE Annals of the History of Computing* 31.4 (2009), pp. 6–25. DOI: 10.1109/MAHC.2009.123 (cit. on p. 1).

[17] A. R. Hevner. "A three cycle view of design science research". In: *Scandinavian journal of information systems* 19.2 (2007), p. 4 (cit. on pp. 32, 33).

[18] A. ICT. *What is OutSystems?* https://www.ada-ict.nl/en/what-is-outsystems/, accessed 2021-01-23 (cit. on p. 11).

[19] F. Karray, M. Alemzadeh, J. Abou Saleh, and M. N. Arab. "Human-computer interaction: Overview on state of the art". In: (2008) (cit. on p. 6).

[20] S. Krug. *Don't make me think!: a common sense approach to Web usability*. Pearson Education India, 2000 (cit. on p. 34).

[21] C. Larman. "Agile and Iterative Development: A Manager's Guide". In: 2003 (cit. on p. 7).

[22] J. Lloret-Gazo. "A Survey on Visual Query Systems in the Web Era (extended version)". In: *CoRR* abs/1708.00192 (2017). arXiv: 1708.00192. URL: http://arxiv.org/abs/1708.00192 (cit. on p. 19).

[23] S. El-Mahgary and E. Soisalon-Soininen. "A form-based query interface for complex queries". In: *Journal of Visual Languages Computing* 29 (2015), pp. 15–53. ISSN: 1045-926X. DOI: https://doi.org/10.1016/j.jvlc.2015.03.001 (cit. on pp. 21, 22).

[24] Mendix. *XPath*. https://docs.mendix.com/refguide/xpath, accessed 2021-02-15 (cit. on p. 26).

[25] J. Nielsen. "Estimating the number of subjects needed for a thinking aloud test". In: *International Journal of Human-Computer Studies* 41.3 (1994), pp. 385–397. ISSN: 1071-5819. DOI: https://doi.org/10.1006/ijhc.1994.1065 (cit. on p. 11).

[26] J. Nielsen. *How Many Test Users in a Usability Study?* https://www.nngroup.com/articles/how-many-test-users/, accessed 2021-02-07 (cit. on p. 10).

[27] J. Nielsen. *Usability engineering*. Morgan Kaufmann, 1994 (cit. on p. 7).

[28] J. Nielsen. *Why You Only Need to Test with 5 Users*. https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/, accessed 2021-02-07 (cit. on p. 10).

[29] J. Nielsen and T. K. Landauer. "A Mathematical Model of the Finding of Usability Problems". In: *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems*. CHI '93. Amsterdam, The Netherlands: Association for Computing Machinery, 1993, pp. 206–213. ISBN: 0897915755. DOI: 10.1145/169059.169166 (cit. on p. 11).

[30] OutSystems. *Advanced Aggregates*. https://www.outsystems.com/training/lesson/1972/advanced-aggregates?LearningPathId=18, accessed 2021-01-24 (cit. on p. 14).

[31] OutSystems. *Aggregate*. https://success.outsystems.com/Documentation/11/Reference/OutSystems_Language/Data/Handling_Data/Queries/Aggregate, accessed 2021-01-24 (cit. on p. 14).

[32] OutSystems. *Aggregates 101*. https://www.outsystems.com/training/courses/126/aggregates-101/?LearningPathId=18, accessed 2021-01-24 (cit. on p. 14).

[33] OutSystems. *Application Layers*. https://www.outsystems.com/training/lesson/2186/application-layers?LearningPathId=18, accessed 2021-01-23 (cit. on p. 12).

[34] OutSystems. *Building Dynamic SQL Statements the Right Way*. https://success.outsystems.com/Documentation/Best_Practices/Development/Building_Dynamic_SQL_Statements_the_Right_Way, accessed 2021-01-21 (cit. on p. 29).

[35] OutSystems. *Components and Tools*. https://www.outsystems.com/training/lesson/2183/components-and-tools?LearningPathId=18, accessed 2021-01-23 (cit. on pp. 11, 12).

[36] OutSystems. *Developing with OutSystems*. https://www.outsystems.com/evaluation-guide/developing-with-outsystems/, accessed 2021-01-23 (cit. on p. 11).

[37] OutSystems. *Service Studio Overview*. https://success.outsystems.com/Documentation/11/Getting_started/Service_Studio_Overview, accessed 2021-01-23 (cit. on p. 13).

[38] OutSystems. *The Low-Code Market in 2021*. https://www.outsystems.com/blog/posts/low-code-market/, accessed 2021-02-17 (cit. on p. 1).

[39] OutSystems. *What's (Not) New in OutSystems: A Product Timeline*. https://www.outsystems.com/blog/posts/not-new-product-timeline/, accessed 2021-01-24 (cit. on p. 17).

[40] O. B. Platform. *ISO 9241-11:2018(en) Ergonomics of human-system interaction — Part 11: Usability: Definitions and concepts*. URL: %5Curl%7Bhttps://www.iso.org/obp/ui/#iso:std:iso:9241:-11:ed-2:v1:en%7D,%20accessed%202021-01-30 (cit. on p. 6).

[41] P. S. Rodrigues. "Accelerating SQL with Complex Visual Querying". unpublished thesis. MA thesis. FCT-UNL, 2020 (cit. on pp. 29, 108).

[42] Y. Rogers. "New theoretical approaches for human-computer interaction". In: *Annual Review of Information Science and Technology* 38.1 (2004), pp. 87–143. DOI: https://doi.org/10.1002/aris.1440380103. eprint: https://asistdl.onlinelibrary.wiley.com/doi/pdf/10.1002/aris.1440380103 (cit. on p. 6).

[43] SalesForce. *Introducing the SOQL Query Builder, Now in Beta!* https://developer.salesforce.com/blogs/2020/11/introducing-the-soql-query-builder-now-in-beta.html, accessed 2021-02-15 (cit. on p. 26).

[44] SalesForce. *SOQL Builder (Beta)*. https://developer.salesforce.com/tools/vscode/en/soql/soql-builder/, accessed 2021-02-15 (cit. on p. 26).

[45] A. Sears and J. A. Jacko. *Human-computer interaction: Development process*. CRC Press, 2009 (cit. on pp. 7–9).

[46] K. L. Siau, H. C. Chan, and K. K. Wei. "Effects of query complexity and learning on novice user query performance with conceptual and logical database interfaces". In: *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans* 34.2 (2004), pp. 276–281 (cit. on p. 18).

[47] K. L. Siau, H. C. CHAN, and K. P. TAN. "Visual knowledge query language". In: *IEICE TRANSACTIONS on Information and Systems* 75.5 (1992), pp. 697–703 (cit. on p. 18).

[48] A. Soylu, M. Giese, E. Jiménez-Ruiz, E. Kharlamov, D. Zheleznyakov, and I. Horrocks. "Ontology-based end-user visual query formulation: Why, what, who, how, and which?" In: *Univers. Access Inf. Soc.* 16.2 (2017), pp. 435–467. DOI: 10.1007/s10209-016-0465-0 (cit. on pp. 2, 3).

[49] C. Stolte, D. Tang, and P. Hanrahan. "Polaris: A system for query, analysis, and visualization of multidimensional relational databases". In: *IEEE Transactions on Visualization and Computer Graphics* 8.1 (2002), pp. 52–65 (cit. on p. 19).

[50] A. M. Wichansky. “Usability testing in 2000 and beyond”. In: *Ergonomics* 43.7 (2000). PMID: 10929833, pp. 998–1006. DOI: 10.1080/001401300409170 (cit. on p. 9).

[51] M. M. Zloof. “Query-by-example: A data base language”. In: *IBM systems Journal* 16.4 (1977), pp. 324–343 (cit. on p. 18).

A

USERS ANALYSIS

In this appendix are included the profiles of all the users tested in this thesis and the form that was used to collect the users information.

A.1 Platform current state users

Platform Initial State - Users details				
Timestamp	Identifier	Have you ever worked with the OutSystems Platform?	If you answered "Yes" in the previous question, when was the last time you worked with it?	Do you have any OutSystems certification?
3/30/2021 11:52:47	OutSystems Dev 1	Yes	<3 Years	No
4/6/2021 23:59:39	OutSystems Dev 2	Yes	<3 Years	Yes
4/14/2021 14:20:45	OutSystems Dev 3	Yes	3 or more years	Yes
4/14/2021 16:57:20	OutSystems Dev 4	Yes	<6 Months	Yes
4/19/2021 11:36:17	OutSystems Dev 5	Yes	<6 Months	No
4/9/2021 14:01:57	Software Dev 1	Yes	<6 Months	No
4/12/2021 18:35:16	Software Dev 2	Yes	<1 Year	No
4/13/2021 11:38:38	Software Dev 3	Yes	<6 Months	No
4/15/2021 18:44:56	Software Dev 4	Yes	<6 Months	No
2021/04/23 11:53:54	Software Dev 5	Yes	<6 Months	No
2021/04/28 20:20:23	Non Dev 1	No		No
2021/05/05 17:09:24	Non Dev 2	No		No
2021/07/22 13:48:03	Non Dev 3	No		No
2021/07/23 09:20:40	Non Dev 4	No		No
2021/09/01 14:29:54	Non Dev 5	No		No

How do you classify your level of experience with OutSystems?	Do you have any knowledge of programming languages?	If you answered "Yes" in the previous question, can you tell some of them?
Medium (Frequent User)	Yes	Java, kotlin
Medium (Frequent User)	Yes	Java, c#, python
High (Expert User)	Yes	Java, C#, Javascript
Medium (Frequent User)	Yes	Javascript, Java
High (Expert User)	Yes	C++, C#, Javascript
Low (Beginner User)	Yes	Java, Python, C, HTML/CSS, JavaScript, Rust, Go e Erlang.
Low (Beginner User)	Yes	Java, C#, C++, JS, React
Low (Beginner User)	Yes	C#, TypeScript, Kotlin, Swift
Low (Beginner User)	Yes	JS, C, Java, Kotlin
Low (Beginner User)	Yes	Java (including Android specific), JavaScript, Python, Objective-C (a little)
None	Yes	Pascal and Visual Basic in high school
None	No	
None	No	
None	No	
None	No	

Do you have experience with SQL?	Do you have experience with Microsoft Excel, Google Sheets or other similar software?
Yes, but only a few times or very long ago	Yes
Yes, I have already worked with it several times	Yes
Yes, I have already worked with it several times	Yes
Yes, but only a few times or very long ago	Yes
Yes, I have already worked with it several times	Yes
Yes, I have already worked with it several times	Yes
Yes, but only a few times or very long ago	Yes
Yes, but only a few times or very long ago	No
Yes, but only a few times or very long ago	Yes
Yes, but only a few times or very long ago	Yes
No	Yes
No	Yes
No	Yes
No	Yes
No	Yes

A.2 Low-fidelity prototype users

Low-fidelity prototype - Users details				
Timestamp	Identifier	Have you ever worked with the OutSystems Platform?	If you answered "Yes" in the previous question, when was the last time you worked with it?	Do you have any OutSystems certification?
2021/04/27 14:03:26	OutSystems Dev 1	Yes	<6 Months	Yes
2021/04/26 15:08:36	OutSystems Dev 2	Yes	<6 Months	Yes
2021/04/29 15:57:28	OutSystems Dev 3	Yes	<3 Years	Yes
2021/04/29 21:00:41	OutSystems Dev 4	Yes	3 or more years	Yes
2021/05/03 15:57:00	OutSystems Dev 5	Yes	<6 Months	Yes
2021/04/28 15:08:21	Software Dev 1	Yes	<6 Months	No
2021/05/04 19:17:08	Software Dev 2	No		No
2021/05/28 16:07:47	Software Dev 3	No		No
2021/05/11 00:00:19	Software Dev 4	Yes	3 or more years	No
2021/05/13 17:27:39	Software Dev 5	Yes	<6 Months	No
2021/05/13 20:59:05	Non Dev 1	No		No
2021/05/16 17:27:39	Non Dev 2	No		No
2021/05/10 14:57:58	Non Dev 3	No		No
2021/06/01 18:28:21	Non Dev 4	No		No
2021/06/01 19:24:18	Non Dev 5	No		No

How do you classify your level of experience with OutSystems?	Do you have any knowledge of programming languages?	If you answered "Yes" in the previous question, can you tell some of them?
High (Expert User)	Yes	ruby on rails, c#, c++, java
Medium (Frequent User)	Yes	C#, JAVA, REACT, ANGULAR, PYTHON
High (Expert User)	Yes	Outsystems, Java, CSS
High (Expert User)	Yes	Java, JS, C, SQL
High (Expert User)	Yes	C++, Java, Python
Low (Beginner User)	Yes	Java, C#, JavaScript, Flutter, SQL, PHP
Low (Beginner User)	Yes	Java, Javascript, Kotlin
None	Yes	Java, JavaScript, C#, PHP, Python, Lua
Low (Beginner User)	Yes	C#, Dart, Java, JavaScript, PHP
Low (Beginner User)	Yes	Java, C#, SQL, HTML, JavaScript
None	No	
None	No	
None	Yes	Html, css, java script, swift
None	Yes	Basics of HTML, CSS...
None	No	

Do you have experience with SQL?	Do you have experience with Microsoft Excel, Google Sheets or other similar software?
Yes, I have already worked with it several times	Yes
Yes, I have already worked with it several times	Yes
Yes, I have already worked with it several times	Yes
Yes, I have already worked with it several times	Yes
Yes, I have already worked with it several times	Yes
Yes, I have already worked with it several times	Yes
Yes, I have already worked with it several times	Yes
Yes, I have already worked with it several times	Yes
Yes, I have already worked with it several times	Yes
Yes, I have already worked with it several times	Yes
No	Yes
No	Yes
Yes, but only a few times or very long ago	Yes
No	Yes
No	Yes

A.3 ListSelect function prototype users

ListSelect prototype - Users details				
Timestamp	Identifier	Have you ever worked with the OutSystems Platform?	If you answered "Yes" in the previous question, when was the last time you worked with it?	Do you have any OutSystems certification?
2021/05/17 18:47:54	OutSystems Dev 1	Yes	3 or more years	Yes
2021/05/18 13:40:41	OutSystems Dev 2	Yes	<6 Months	Yes
2021/05/18 16:28:08	Software Developer 1	No		No
2021/05/20 17:47:05	Software Developer 2	No		No
2021/05/25 17:27:12	Software Developer 3	No		No
2021/05/28 17:44:58	Software Developer 4	No		No

How do you classify your level of experience with OutSystems?	Do you have any knowledge of programming languages?	If you answered "Yes" in the previous question, can you tell some of them?
High (Expert User)	Yes	.Net, PowerBuilder, Delphi7
High (Expert User)	Yes	.Net, Java
None	Yes	C#, C/C++, Java, Angular, React, Flutter...
None	Yes	Java, Kotlin, C#
None	Yes	C/C++, C#, Java, Python, SQL
None	Yes	C#, Java, Kotlin

Do you have experience with SQL?	Do you have experience with Microsoft Excel, Google Sheets or other similar software?
Yes, I have already worked with it several times	Yes
Yes, I have already worked with it several times	Yes
Yes, I have already worked with it several times	Yes
Yes, but only a few times or very long ago	No
Yes, I have already worked with it several times	Yes
Yes, but only a few times or very long ago	Yes

A.4 Service Studio prototype users

Service Studio Prototype - Users details				
Timestamp	Identifier	Have you ever worked with the OutSystems Platform?	If you answered "Yes" in the previous question, when was the last time you worked with it?	Do you have any OutSystems certification?
2021/06/22 17:35:11	OutSystems Dev 1	Yes	<6 Months	Yes
2021/06/24 13:03:55	OutSystems Dev 2	Yes	<6 Months	No
2021/06/28 15:31:57	OutSystems Dev 3	Yes	<6 Months	No
2021/06/28 17:30:00	OutSystems Dev 4	Yes	3 or more years	Yes
2021/07/01 15:18:19	OutSystems Dev 5	Yes	<6 Months	Yes
2021/06/15 10:56:19	Software Dev 1	Yes	<3 Years	No
2021/06/15 13:22:48	Software Dev 2	No		No
2021/06/15 16:05:54	Software Dev 3	No		No
2021/06/17 14:16:48	Software Dev 4	No		No
2021/06/21 16:25:46	Software Dev 5	No		No
2021/07/05 18:57:06	Non Dev 1	No		No
2021/07/06 20:34:36	Non Dev 2	No		No
2021/07/08 14:18:24	Non Dev 3	No		No
2021/07/15 18:20:39	Non Dev 4	No		No
2021/07/22 16:53:03	Non Dev 5	No		No

How do you classify your level of experience with OutSystems?	Do you have any knowledge of programming languages?	If you answered "Yes" in the previous question, can you tell some of them?
High (Expert User)	Yes	I learned the basics of programming in the university, with C++.
Medium (Frequent User)	Yes	C, C++, Java, JS, Prolog, R...
Medium (Frequent User)	Yes	Java, C, SQL, Javascript, php
High (Expert User)	Yes	C, Java, Javascript
Medium (Frequent User)	Yes	SQL (low to intermediate) / .NET (very little knowledge) / LANSA (intermediate) / JS (very little knowledge)
Low (Beginner User)	Yes	Java, C, Go, Erlang, Rust, Javascript, Python
None	Yes	Java, JavaScript, Dart, Go, C#, C, Kotlin, Python
None	Yes	Java, C#, Python, Typescript, Javascript
Low (Beginner User)	Yes	Javascript, SQL, Python, C++
None	Yes	Java, Python, JavaScript, Golang, C#
None	No	
None	No	
None	No	
None	No	
None	No	

Do you have experience with SQL?	Do you have experience with Microsoft Excel, Google Sheets or other similar software?
Yes, I have already worked with it several times	Yes
Yes, but only a few times or very long ago	Yes
Yes, I have already worked with it several times	Yes
Yes, I have already worked with it several times	Yes
Yes, I have already worked with it several times	Yes
Yes, I have already worked with it several times	Yes
Yes, I have already worked with it several times	Yes
Yes, I have already worked with it several times	Yes
Yes, I have already worked with it several times	Yes
Yes, but only a few times or very long ago	Yes
Yes, but only a few times or very long ago	Yes
No	Yes
No	Yes
No	Yes
No	Yes

B

Platform Current State Evaluation: Full results

In this appendix are included the detailed results from every test from the Current State phase.

	1. Has ever worked with Aggregates?	2. Has already faced this problem?	3. Starting looking place	4. Used google?	5. If used SQL, had difficult w/ it?	6. Mentioned the SQL IN clause?	7. Knew the Index built-in function?	8. Had difficulty understanding the dropdown tags?	9. Had difficulty using the ForEach component?
								Non-Developers	
Non-Developer 1	No	No	DropdownTags handler	No	-	No	No	Yes	Did not use
Non-Developer 2	No	No	DropdownTags handler	No	-	No	No	Yes	Did not use
Non-Developer 3	No	No	GetCategories Aggregate	No	-	No	No	Yes	Did not use
Non-Developer 4	No	No	DropdownTags handler - System Actions	No	-	No	No	Yes	Did not use
Non-Developer 5	No	No	GetCategories Aggregate	No	-	No	No	Yes	Did not use
								Developers	
Soft. Dev 1	Yes	No	Aggregate filters (wanted to make a cycle junction of “or” with “=”)	No	-	No	No	Yes	No (only used 1)
Soft. Dev 2	Yes	No	Aggregate filters (wanted to use a function like “contains”)	No	-	No	No	No	No (only used 1)
Soft. Dev 3	Yes	No	Aggregate filters (wanted to use IN)	No	-	Yes	No	No	No (only used 1)
Soft. Dev 4	Yes	No	Aggregate filters (wanted to use a function like “contains”)	Yes (found the Index function and a solution with List Remove)	-	No	No	No	No (only used 1)
Soft. Dev 5	Yes	No	Aggregate filters (wanted to use a function like “any”)	Yes (tried to find examples like this problem but did not find it)	-	No	No	Yes	No (did not use)
								OutSystems Developers	
OutSystems Dev 1	Yes	No	Aggregate filters	No	-	No	No	No	Yes (used 2 in a row)
OutSystems Dev 2	Yes	No	SQL	Yes (to search how to do loops w/ SQL)	Yes	No	No	Yes	Yes (used 2 in a row)
OutSystems Dev 3	Yes	Yes	SQL	No	No	Yes	Yes	No	Did not use
OutSystems Dev 4	Yes	Yes	SQL	Yes (to see SQL syntax and IN clause)	Yes	Yes	Yes	No	No (only used 1)
OutSystems Dev 5	Yes	Yes	SQL	Yes (to see String Join action)	Yes	Yes	Yes	No	No (only used 1)

10. Knew system actions?	11. Had difficulty choosing appropriate system actions?	12. Intended final solution	13. The user was able to complete the task?	14. Duration (h:m:s)
No	Yes (used ListFilter)	Use the ListFilter action to filter the list with all the products	No	00:26:40
No	Did not use	Copy the filter of the Input name Search	No	0:26:01
No	Did not use	Tried to add a filter in GetProducts but did not know how to write a condition	No	0:31:53
A little	Yes (didn't select any)	Use a if component with many "0r"s and "="	No	0:18:50
No	Did not use	Do a similar filter condition to the one for the name	No	00:30:56
No (started by using assign)	No	Add the results of an aggregate for each category selected in a new list	No	00:39:50
No	Yes (started by using ListInsert)	Add the results of an aggregate for each category selected in a new list	No	0:32:00
No (used assign)	Did not use them	Add the categories in a String and then use a contains in the aggregate filter	No (only for one selected category)	0:36:30
Yes	Yes (tried using ListRemove)	Remove from the products list the products that do not have any of the selected categories	No (problem removing item from the list that is being iterated)	0:37:20
No	Did not use them	Wanted to iterate the categories and refresh the aggregate one by one	No (only for one selected category)	00:47:05
Yes	No	Remove from the list of products the non-intended ones	No	0:29:13
Yes, but did not remember to use them	Yes (used assign instead of list append)	Add to a new list the intended products from the list of products	No	0:53:20
Yes	No	Use SQL with the IN clause	Yes (but without products on start and with a query error message)	0:13:48
Yes	Did not use	Use SQL with the IN clause	No	1:01:47
Yes	Did not use	Use SQL with the IN clause	Yes	0:40:15

C

Low-Fidelity Prototype Evaluation: Full results

In this appendix are included the detailed results from every test from the Low-Fidelity Prototype phase.

	Knew SQL?	Knew that the IN didn't previously exist in Aggregates?	Correctly understood the SelectedCategories?	Easily understood and selected the new IN operator?	Started by selecting the whole list or expanding?	Ended selecting the whole list or SelectedCategories. Current.Value/Text?	Found the name "Current" inadequate for the use case?
Non-Developers							
Non-Developer 1	No	No	No	No (started by selecting equals and had many difficulties in understanding database concepts)	Whole list	Whole list	No
Non-Developer 2	No	No	No (thought value was the price and text was the category)	Sort of (selected "like")	Whole list	Whole list	No
Non-Developer 3	Not much (few times and a long time ago)	No	No (thought Value was the price, Text the name)	No (started by selecting "=", then "like" and finally "in")	Expanded	CategoriesToFilter. Current.Value	No
Non-Developer 4	No	No	No	Sort of (selected "like")	Whole list	Whole list	No
Non-Developer 5	No	No	No	Sort of (started by selecting "=" but switched when understood that it was a list)	Whole list	Whole list	No
Developers							
Soft. Dev 1	Yes	No	Yes	Yes (but started by searching for "contained")	Expanded	Whole list	Yes
Soft. Dev 2	Yes	No	No (thought it had had the categories with a boolean field indicating the selected ones)	Sort of (started by selecting "=")	Expanded	CategoriesToFilter. Current.Value	No
Soft. Dev 3	Yes	No	No (didn't notice the attributes)	Yes	Whole list	Whole list	No
Soft. Dev 4	Yes	No	No (though that didn't need to select the attribute to compare)	Yes	Expanded	Whole list	Yes
Soft. Dev 5	Yes	No	No, though it was a list of String	Sort of (started by selecting "=")	Whole list	Whole list	No
OutSystems Developers							
OutSystems Dev 1	Yes	Yes	Yes	Yes	Whole list	Whole list	-
OutSystems Dev 2	Yes	Yes	Yes	Yes	Expanded	CategoriesToFilter. Current.Value	No
OutSystems Dev 3	Yes	Yes	Yes	Yes	Expanded	Whole list	Yes
OutSystems Dev 4	Yes	Yes	Yes	Yes	Expanded	-	Yes
OutSystems Dev 5	Yes	Yes	Yes	Yes	Expanded	Whole list	Yes

Agreed that IN is an intuitive name for the operation?	Agreed with the location of the new IN operator?
Yes (but prefered Contains and started with CategoriesToFilter)	Yes
Yes	yes
Yes (after thinking about it, but found "Contains" much more intuitive)	Yes
Yes	Yes
Yes	Yes
Yes (also liked "contains")	Yes
Yes	Yes
Yes, but found Contains more intuitive	Yes
Yes	Yes
Yes (also mentioned Contains)	Yes
Yes (didn't like "contains")	Yes
Yes (also liked "contains")	Yes
Yes	Yes
Yes	Yes
Yes	Yes

D

IN RIGHT PARAMENTER EXPLORATION: FULL RESULTS

In this appendix are included the detailed answers of every user regarding their solution preference and the results from the usability tests with the ListSelect() function prototype.

Table 1 - User Interviews Answers
Table 2 - ListSelect() Function Prototype evaluation results

	Basic solution	Proposed solution 1	Proposed solution 2	Proposed solution 3	Proposed solution 4
Non-Developers					
Non-Developer 1	No	No	Better than solution 2	Liked it (favorite one)	Liked the hint
Non-Developer 2	No	No	Better than solution 2	-	Liked (favorite one)
Non-Developer 3	Didn't like this one, sounded too complex	-	Said that "by" is not intuitive	"Current" and "Attributes" might be confusing	Liked this one (favorite one)
Non-Developer 4	No	Better than solution 3	No	Favorite one (text and value as new properties)	Liked
Non-Developer 5	-	No	Yes, more intuitive for begginer users (Favorite one)	2nd	Too complex
Software Developers					
Soft. Dev 1	-	Liked this one	-	Favorite one (new list property "Content")	-
Soft. Dev 2	-	-	-	-	Favorite one
Soft. Dev 3	No	No	No	Liked it	Liked it
Soft. Dev 4	-	-	-	-	Liked this one (favorite one)
Soft. Dev 5	Did not like it	-	Found better than with parenthesis	Mentioned a new property "Element", but recognized ambiguity	Found it too complex and long
Outsystems Developers					
OutSystems Dev 1	-	Favorite one (IN used always with the parentheses)	-	-	-
OutSystems Dev 2	Didn't like this one	Favorite one (with auto-match by name instead of type)	-	-	-
OutSystems Dev 3	-	-	-	Favorite one (new list property "List")	-
OutSystems Dev 4	-	-	-	-	Favorite one
OutSystems Dev 5	-	-	Favorite one (with "by" being added automatically)	-	-

	Knew SQL?	Knew the IN operator?	Easily understood and selected the new IN operator?	What did the user first select next to the IN?	Easily found and used the hint?	Found the error message usefull?	Correctly understood the new ListSelect function?	Did the user found the symbol "=>" intuitive?	Did the user found the name "Record" intuitive?
Software Developers									
Soft. Dev 1	Yes	Yes	Yes	Entire List	No	No (though it was the other blue button)	Yes	Yes	Yes
Soft. Dev 2	Yes	No	No, started by selecting "="	CategoriesToFilter. Current.Text	No, found it but didn't click in the suggestions	Yes	Yes	Yes	No
Soft. Dev 3	Yes	Yes	No, started by selecting "like" and then used "ORs"	CategoriesToFilter. Current.Text	No	Yes	Yes	Yes	Yes
Soft. Dev 4	Yes	Yes	Yes	CategoriesToFilter. Current.Text	Yes	Yes	Yes	Yes	No
OutSystems Developers									
OutSystems Dev 1	Yes	Yes	No, but only because he knew that it didn't previously exist	CategoriesToFilter. Current.Value	Yes	Yes	Yes	Yes	Yes
OutSystems Dev 2	Yes	Yes	Yes	Entire List	No	Yes	Yes	Yes	No

The user completed the task succesfully?	Extra observations	Changes for next iteration
No	Wanted to use "()" with the IN operator	Change name to "light bulb" instead of "hint bulb"
Yes		
Yes		Separate the error message in 2 lines
Yes	Mentioned similarity with Linq	
Yes	Wanted to use "()" with the IN operator	
Yes	Wanted to use "()" with the IN operator	Fill the entire icon in orange

E

SERVICE STUDIO PROTOTYPE EVALUATION: FULL RESULTS

In this appendix are included the detailed results from every test from the Service Studio Prototype phase.

	1. Has ever worked with Aggregates?	2. Has already faced this problem?	3. Starting looking place	4. Used google?	5. If used SQL, had difficult w/ it?	6. Already knew IN operator?	7. Mentioned "Contains"?	8. Had difficulty understanding the DropdownTags?
Non-Developers								
Non-Developer 1	No	No	DropdownTags component	No	-	No	No	Yes
Non-Developer 2	No	No	DropdownTags component	No	-	No	No	Yes
Non-Developer 3	No	No	DropdownTags OnChange	No	-	No	No	No
Non-Developer 4	No	No	DropdownTags component	No	-	No	No	Yes
Non-Developer 5	No	No	DropdownTags component	No	-	No	Yes	Yes
Software Developers								
Soft. Dev 1	No	No	GetDistinctCategories Aggregate	No	-	No	No	No
Soft. Dev 2	No	No	GetProducts Aggregate	No	-	Yes	No	No
Soft. Dev 3	No	No	ListFilter system action	No	-	Yes	Mentioned Includes	No
Soft. Dev 4	No	No	GetProducts Aggregate	No	-	Yes	No	No
Soft. Dev 5	No	No	Created a new dropdowntags handler	No	-	Yes (from python)	No	Yes (because created a new one and so it had a list with two attributes)
OutSystems Developers								
OutSystems Dev 1	Yes	Yes	GetProducts Aggregate	No	-	Yes	-	No
OutSystems Dev 2	Yes	Yes	GetProducts Aggregate	No	-	Yes	-	No
OutSystems Dev 3	Yes	No	GetProducts Aggregate	Yes (to search how to make a filter in the aggregate filters and found the post about the inexistence of the IN)	-	Yes	Yes	Yes
OutSystems Dev 4	Yes	Yes	GetProducts Aggregate	No	-	Yes	-	No
OutSystems Dev 5	Yes	No	GetProducts Aggregate	No	-	Yes	-	Yes (didn't understand that the SelectedCategories input parameter already had the selected categories)

12. Solution thoughs	13. The user was able to complete the task?	14. Duration (h:m:s)
Tried to explore the dropdowntags but didn't understand how to view and edit the associated event	No	00:19:20
saw that GetProducts had a filter for the other component and wanted to make another aggregate similarly but he didn't know how to do it	No	0:43:02
used the like operator; thought it would be indifferent to use like or in from the help messages	No	0:22:46
tried to put an if in the Inputsearchnameonchange action	No	0:25:27
got stuck in the filter condition part	No	00:20:51
wanted to put a component in front of the aggregate refresh to filter (basically wanted a list action)	Yes	00:32:57
started by wanting to make the filter as a junction of Or clauses	Yes	0:17:01
Wanted a filter component (tried searching the search bar for "filter")	Yes	0:36:19
First chose the like but then realized he wanted more than one tag in the comparison	Yes	0:12:16
selected "like" because he saw the logic of the already made example.	No	00:27:43
Use the IN operator in the Aggregate filters	Yes	0:03:15
First he wanted to solve by concatenating the selected categories and using the "like" operator	Yes	0:04:04
Use the IN operator in the Aggregate filters	Yes	0:13:18
Use the IN operator in the Aggregate filters	Yes	0:03:44
Use the IN operator in the Aggregate filters	Yes	0:09:08

I

DATA ANALYSIS

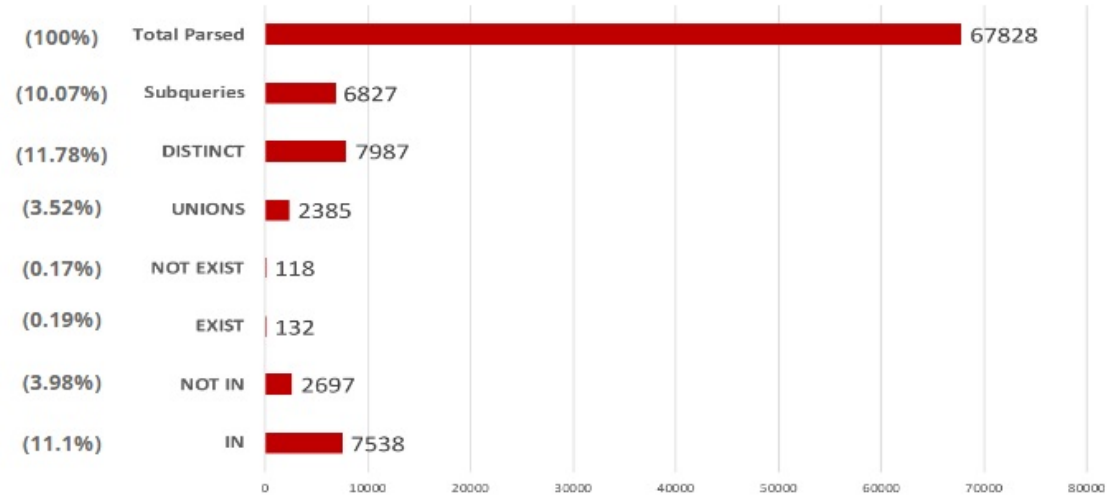


Figure I.1: Distribution of operations not supported by Aggregates in 67.828 total SQL queries parsed [41]

II

GARTNER MAGIC QUADRANT

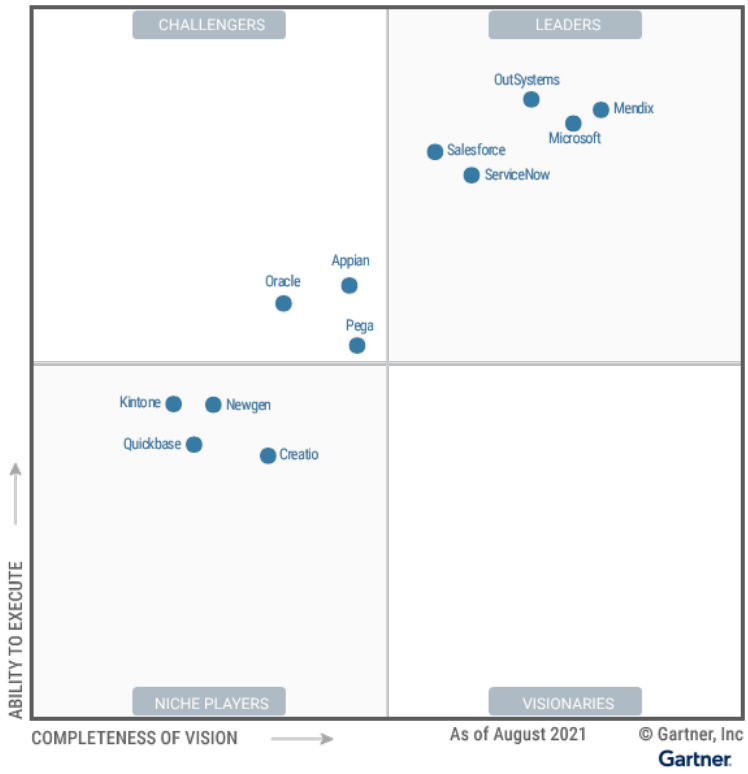


Figure II.1: Magic Quadrant for Enterprise Low-Code Application Platforms - August 2021 [15]