JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 27 (2023) 154–161
Received: 25 March 2023
Accepted: 17 April 2023

Comparative analysis of frameworks and automation tools in terms of functionality and performance on the Salesforce CRM Platform

Analiza porównawcza szkieletów programistycznych i narzędzi automatyzujących pod względem funkcjonalności i wydajności na platformie Salesforce

Damian Sebastian Ciechan*

Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract
Article describes comparative analysis of both code and low-code automation tools together with frameworks used for developing graphical user interfaces that are available on the Salesforce Platform. The research is being carried out due to lack of such comparison in the available literature and due to popularity of the Salesforce CRM. Four automation tools were put together: code-based *Apex Triggers* and three *point-and-click* tools: *Workflow Rules, Process Builder, Flow Builder*. In each of the frameworks (*Visualforce, Aura Components, Lightning Web Components*) an application module was developed and example logic was implemented in each of the automation tools. DML operations *insert, update, delete* were compared in terms of performance and each technology was analyzed in terms of provided functionalities and limitations. It was concluded that the most efficient automation tool is *Flow Builder* and the *Lightning Web Components* framework is the best choice for developing graphical user interfaces.

Keywords: Salesforce; performance; low-code tools; frameworks

Streszczenie
Artykuł opisuje analizę porównawczą narzędzi automatyzujących (niskokodowych i programistycznych) oraz szkieletów do budowania interfejsu graficznego użytkownika dostarczanych wraz ze środowiskiem Salesforce. Badania zostały przeprowadzone ze względu na brak takowych w dostępnej literaturze i ze względu na popularność systemu Salesforce. W zestawieniu porównano cztery narzędzia automatyzujące: oparte na bazie kodu *Apex Triggers* i trzy narzędzia pozwalające na budowanie logiki metodą wskaż i kliknij: *Workflow Rules, Process Builder, Flow Builder*. W każdym ze szkieletów (*Visualforce, Aura Components, Lightning Web Components*) wytworzone zostały trzy analogiczne moduły aplikacji I zaimplementowano logikę w każdym z narzędzi automatyzujących. Operacje DML tworzenia, aktualizowania i usuwania rekordów porównano pod względem wydajnościowym, a każdą technologię przeanalizowano pod względem udostępnianych funkcjonalności i ograniczeń na platformie. Z przeprowadzonych badań wywnioskowano, że najwydajniejszym narzędziem jest *Flow Builder*, a szkielet *Lightning Web Components* jest lepszym wyborem do tworzenia interfejsu graficznego niż jego konkurenci.

Słowa kluczowe: Salesforce; wydajność; narzędzia niskokodowe; szkielety programistyczne

*Corresponding author
Email address: damian.ciechan@pollub.edu.pl (D. S. Ciechan)

1. Introduction

Low-Code Software Development is a new, emerging application development technique that combines minimal amount of source code with graphical user interfaces to reduce development time [1]. In recent years, increasing number of organisations have chosen to use Low-Code Development Platforms (*LCDP*). In many cases, low-code application are developed by so-called *Citizen Developers* [2], i.e. company employees who do not have deep technical or programming knowledge. According to the Gartner report, by 2024, around 65% of large enterprises will be using Low-Code Development Platforms to some degree, and the market is expected to be worth more than $31 billion [3-4].

The popularity of *LCDP* noticeably correlates with the popularity of Customer Relationship Management (CRM) Systems. Providers such as Microsoft or Salesforce provide solutions for both low-code application development and the CRM software themselves. Integrated tools allow employees to customize and extend functionalities in implemented CRM system to support new business requirements. Despite its many advantages, the development of such system can bring new challenges as the business grows. The main one is the size of the data to be processed – out-of-the-box modules and tools have pre-defined limits on how many records they can process simultaneously. Flexibility also has its limits – despite providing ready to use connectors to integrate with external systems, in many cases integration may require deeper technical knowledge and programming skills to ensure everything works flawlessly.

Considering above observations, the comparative analysis was conducted to determine the most performant automation tool and the best framework for front-end development in terms of functionalities. Evaluation criteria consist of used CPU time on DML operations, whole transaction time, heap memory and network usage. Due to its high market share and popularity, the research was focused on Salesforce products.

2. Salesforce Platform

According to International Data Corporation raport named *Worldwide Semiannual Software Tracker*, Salesforce ranked first as worldwide CRM System provider. This is the ninth consecutive year on the podium with a 23.8% market share [5]. Salesforce provides its products in the *Software as a Service (SaaS)* delivery model, meaning all of the system functionalities are accessible by users from the web browser. This way, *SaaS* minimizes the need to maintain advanced IT infrastructure and all information and data within the CRM System is stored on the provider's servers and disk space. This reduces costs in terms of maintenance and ensures system availability level at >99%, as all updates are carried out remotely, usually during the lowest load hours.

Out-of-the-box, Salesforce provides ready to use environment with functionalities such as:

- cloud applications (Sales Cloud, Marketing Cloud, Service Cloud) all within one environment,
- predefined objects (tables in database nomenclature),
- low-code automation tools (*Workflow Rules, Process Builder, Flow Builder*),
- priopriетary, object oriented programming language *Apex* with database language *SOQL (Salesforce Object Query Language),*
- dedicated front-end frameworks (*Visualforce, Aura Components, Lightning Web Components*),
- Integrated Development Environment – *Visual Studio Code* extension with *sfdx* command line interface,
- REST and SOAP API access to environment.

The platform's architecture is based on *multitenancy* and *metadata*. *Metadata-driven* design means that when creating new field or object, Salesforce internally registers those changes as data (records) in its database table. No data definition operation is executed (i.e. ALTER TABLE) which could block reading and writing data for the duration of processing a potentially lengthy operation. Thanks to *metadata-driven* design, multiple independent environments (*tenants*) can make changes to their instances simultaneously. Although the metadata is physically stored in the same, shared database with identical structure, each tenant has isolated access only to its metadata. Due to its cloud-based architecture, Salesforce enforces limits on each tenant which cannot be exceeded and are taken into consideration in the research:

- CPU time usage per transaction (10 seconds in synchronous context),
- total heap size (6 MB),
- total number of records processed per transaction (10000),
- data storage (10 GB for most licenses, 5 MB in *Developer Edition* license used in research).

2.1. Low-Code automation tools

Workflow Rules is the oldest and most limited tool. In response to insert or update events, it can only perform an update of a field in a given record, send an email to the users associated with the record, create a Task record or send a record SOAP message. Multiple actions can be performed in a single *Workflow*, but the order in which they are performed cannot be modified. Figure 1 shows an example view of defined *Workflow Rule* which sends an email alert.

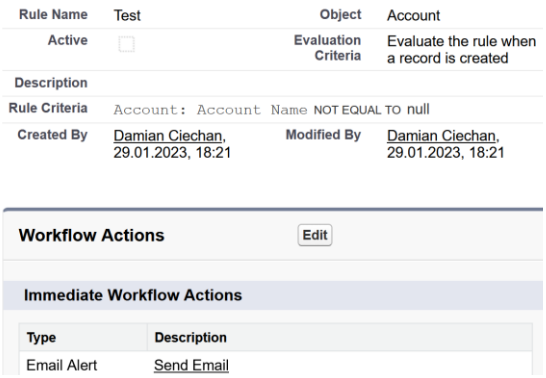

Figure 1: Example of *Workflow Rule.*

Process Builder and *Flow Builder* internally have the same architecture, but the former is better suited for simple tasks. *Process Builder* allows the construction of conditional sets of actions performed one after another (*if – else if - ... - else*), while *Flow Builder* allows the branching of the performed operations and their arrangement on the GUI in any manner. Both tools also offer much greater capabilities in terms of available actions to perform compared to *Workflow Rules* – they can update fields on related records, create record of any object, send notifications, execute code from *Apex* classes. Additionally, using *Screen* elements in Flow *Builder* a component can be created that can be embedded into an application view and allow for user interaction. Figure 2 shows automation which was created in *Process Builder* tool, and figure 3 shows the same automation previewed as a *Flow*.

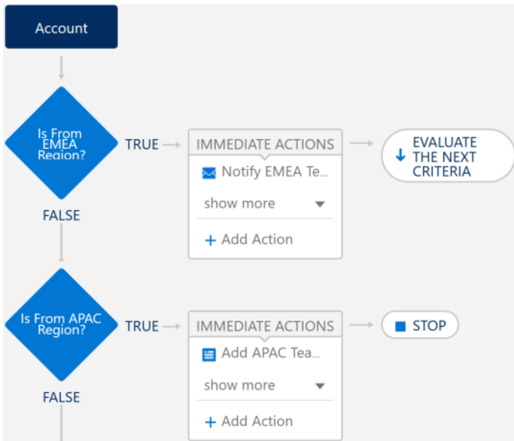

Figure 2: Example *Process Builder* view.

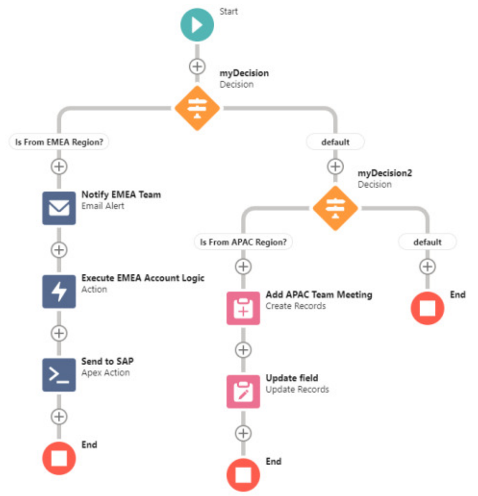

Figure 3: *Process Builder* previewed as a *Flow.*

Table 1 summarizes and compares the capabilities of each tool.

Table 1: Comparison of low-code tools capabilities

Operation	Workflow Rules	Process Builder	Flow Builder
Record creation	Only Task	✓	✓
Record's fields update	Only context record	Context, child and parent record	Any record in the system
Sending email message	Only to users related to the record	Only to users related to the record	Any user in the system
Sending SOAP message	✓		
Sending to approval process		✓	✓
Sending system notification		✓	✓
Can be reused?		✓	✓
Chatter post creation		✓	✓
Apex class invocation		✓	✓
DML listening	Insert, update	Insert, update	Insert, update, delete
Database queries			✓
Deleting records			✓
Logic branching			✓
Versioning		✓	✓
Executing on a regular time interval			✓
Can be used as front-end component?			✓
Debug mode			✓

2.2. Front-end frameworks

There are three available frameworks for developing graphical user interfaces on the Salesforce Platform: *Lightning Web Components, Aura Components, Visualforce*.

The most modern one, *Lightning Web Components*, is built upon standardized W3C Web Components with additional elements required to integrate with Salesforce. As *LWC (Lightning Web Components)* code is run natively by browser's engine and is open source [6], this framework can be used to build any web application (unrelated to Salesforce). The newest version of HTML and Javascript is used to build components, so the learning curve for developers with experience in other front-end frameworks is not very high. Salesforce also provides *sfdx-lwc-jest* CLI add-on to create and run unit tests for the components.

Unlike *LWC*, *Aura Components* is a Salesforce-specific framework; it is not possible to use it to develop applications outside CRM. *Aura* does not support the latest *ECMAScript (European Computer Manufacturers Association Script)* specification – Javascript code must comply with ES5 standards, although some ES6 features are available (e.g. promises). Thus, the entry-level is higher, as the code syntax used in some cases is platform-specific. *Aura Components* is tightly coupled to the Salesforce Platform – it uses its own component model and engine to render the views, resulting (in theory) in lower performance than *LWC*. *Aura Components* also allows unit tests to be written for components, but this requires more configuration – there is a need to manually install *Lightning Testing Service* package on the environment and configure it properly.

The oldest of Salesforce frameworks, *Visualforce*, can be compared to the *JavaServer Pages* technology. It allows pages to be developer using server-side Apex code and platform-specific HTML-like markup language. All business logic is placed in an Apex class associated with the page, called controller. When modifications are made to the page, the platform server compiles the markup language into a set of instructions that can be interpreted by *Visualforce* engine, which returns ready to use HTML document. Unlike previous frameworks, generating the view is done on the server side and consumes resources (time and memory) of the environment instance's processor, consequently offering lower performance. The use of Javascript (ES5 version) is very limited and amounts to placing logic between *<script>* tags – there is no separate file where actions can be delegated. *Visualforce* pages only work within Salesforce, as they are heavily dependent on the platform's server-side language. However, this framework offers a functionality that is absent in other frameworks – native PDF document generation. However, due to limitations in the ability to include CSS styles in such documents (supported only CSS 2.1 version), the preferred approach is to use external Javascript libraries or plugins installed directly on the environment.

3. Literature review

The available literature related to Salesforce is dominated by presentations of various custom applications developed using *Visualforce* and *Aura Components* framework. At the time of the research, no article containing information about *Lightning Web Components* or *Flow Builder* automation tool was available.

The most recent publications [7-8] presents applications for monitoring statistics about Covid-19 disease. In [7] Thanduparakkal et al. using *Aura* framework have developed dashboard named *COVID-19 Tracker* visualizing new cases and number of deaths due to coronavirus. The source of their data was open source REST API *covid19api*. In the article [8] Sharma et al. used fully no-code *Salesforce Einstein Analytics* tool in order to create reports, dashboard and data mining related to the pandemic. As *Einstein Analytics* is powered by artificial intelligence, the publication also shows how the said tool can predict data based on found patterns.

Poniszewska-Maranda et al. [9] presented the *Top 16* Manager application implemented for the Polish Snooker and Billards Association for the management of tournaments in pool games. *Visualforce* framework was used to help the main referee in smoothly managing the tournament by:

- entering match results into the system,
- automatic calculation of players' score,
- automatic generation of competition ladder,
- automatic assignments of referees and players to individual tables,
- email notifications of upcoming matches to players and all tournament participants.

Free *Developer Edition* license was used, which allows up to two users to use the system. From the point of view of Top 16 tournament, that is more than enough as only one person needs access to the system at a given time.

One may question the usefulness of the solution presented by Gupta et al. [10]. Authors have developed a *Visualforce* application for booking metro tickets in the city of Nagpur, India. They mentioned that registration is needed to use the application, but they did not include the information on whether the *Visualforce* site is made public for guests using *Public Site* or *Experience Cloud*. If authorization to internal Salesforce would be required, such solution would be too expensive to implement on wider scale.

In the available literature it is also possible to find articles related directly to the performance of the Salesforce Platform. Miącz [11] in his work analyzed the loading performance of pages created with *Visualforce* framework and the out-of-the-box list views. The main comparison criteria were average number of network requests, file download size, page response and loading times measured with *Chrome Developer Tools*. In the study, the best performant tool was concluded to be standard list view, although the presented results did not differ significantly from each other. The study was also conducted only on 4 and 100 records – in order to obtain a more accurate comparison, the number of records can be increased to 2500 (the limit of data storage in free Salesforce environment edition) or repeat the measured activities multiple time in the system.

The authors of the [12] article focused on the analysis of asynchronous data processing using *Batch, Future, Queueable, Schedulable* and synchronous *Apex Trigger* methods. Total transaction time and the number of records processed per second were chosen as the main comparison criteria. DML insert, update, delete operations were performed on 10000, 50000 and 100000 records. The results obtained by the authors clearly identified *Queuable* as the fastest method for processing data regardless of the number of records (with 885 records created per second in a batch of 100000 records), but they concluded, that *Batch* remains the preferred method if there is a requirement to more closely monitor and manage the amount and order of input data. *Queueable* does not have the transaction splitting mechanism that *Batch* has. Table 2 summarizes the results obtained in article [12] for insert operation – *Apex Trigger* does not include results for more than 10000 records, since synchronous limit is equal to 10000.

Table 2: Number of processed records during insert operation [12]

Method	Number of records		
	10000	50000	100000
	Performance (records/s)		
Batch	653	612	635
Future	399	359	292
Queueable	884	934	885
Schedulable	440	369	396
Trigger	420	-	-

Dan Appleman and Robert Watson at the *Dreamforce 2016* conference [13] performed a detailed analysis of the platform's CPU time usage during the execution of various operations. The authors focused on examining how the certain operations in the Apex language and how the various low-code tools affect the platform's CPU time consumption during transactions. Among other things, they concluded that a static assignment is 30 times faster (~0.58 microseconds vs. ~18 microseconds) than a dynamic assignment and one run of the most efficient *for* loop construction is more than five times faster than the slowest construction (Fig. 4).

```
for (Integer i : array) {
    calculateExecutionTime(); //4 microseconds
}
for (Integer i = 0; i < array.size(); i++) {
    calculateExecutionTime(); //2.5 microseconds
}
Integer s = array.size();
for (Integer i = 0; i < s; i++) {
    calculateExecutionTime(); //0.75 microseconds
}
```

Figure 4: Comparison of *for* loop constructions.

In the context of automation tools, *Apex Trigger, Process Builder* and *Workflow Rules* were compared against each other during insert operation on 200 records. *Process Builder* performance proved to be the worst (almost 3 times slower than *Workflow Rules*). Well-designed code proved to be the most performant choice (table 3 presents results obtained by authors of [13]).

Table 3: CPU time consumption for 200 records insert [13]

Automation Tool	CPU time consuption per record (ms)
No automation	1.1
Apex Trigger	2.2
Workflow Rule	2.8
Process Builder	8.2

Above study [13] is particularly relevant for the research carried out in this work. Based on this, one can presume about the low performance of the *Process Builder* tool and it describes good practices that will be used during the development of the individual application modules. The current state of the literature does not include analysis of the *Flow Builder* and whether the choice of framework affects the execution time of server operations. This research will be extended to include the latest tools and the comparison criteria will be expanded.

4. Research method

For the benchmarking, three application modules were created using *Visualforce, Aura Components* and *Lightning Web Components* frameworks. In each of the frameworks, a page was developed that met the same functional requirements – display list of records, buttons to perform insert, update, delete and select list controlling the number of records in the operation (200, 2500). 2500 is the limit of data storage on the *Developer Edition* license. Then, automation logic in each tool (*Apex Trigger, Workflow Rules, Process Builder, Flow Builder*) was implemented which performs the same actions (update Boolean, date, datetime, number and text field values). The Salesforce instance parameters are shown in Table 4. Frankfurt and Paris instance location means that each transaction is replicated in both locations to minimize errors, increase service availability and avoid single points of failure in the Salesforce infrastructure.

Table 4: Salesforce instance parameters

Parameter	Value
License	Developer Edition
Instance	EU46
Location	Frankfurt, Paris
System version	Spring '23 Patch 9.2

Hardware parameters used to conduct the research are shown in Table 5.

Table 5: Hardware parameters

Parameter	Value
Processor	Intel Core i7-8650U
RAM	24 GB
Storage	512 GB SSD
Graphics	Intel UHD Graphics 620
Operating System	Windows 11 Pro, 22H2
Web Browser	Google Chrome 110

Execution time of each individual task was used as the main comparison criterion. The execution time of DML operation, the time of the entire transaction and the amount of heap memory used were also measured. *Chrome Developer Tools* was used to compare the frameworks – count, time and size of network requests were registered. A newly created object containing only standard set of Salesforce fields and 5 custom fields (*CheckboxField*, *DateField*, *DatetimeField*, *NumberField*, *TextField*) was used. No *Validation Rules, Sharing Rules, Scoping Rules, Restriction Rules* and *Record Types* were defined on this object. The tests were performed by going through the steps from the following scenario:

1. Insert *n* records to the database.
2. Display table of records.
3. Update *n* records in the database.
4. Display updated table of records.
5. Delete *n* records.
6. Display updated of records (empty table).
7. Download measured parameters.
8. Rollback to initial database state.
9. Repeat 1-8 steps for *n:* 200, 2500.

Scenario was executed for each framework and enabled automation combination and repeated 30 times.

5. Results

5.1. Operations processing time

Figures 5-7 show average CPU DML processing time for 200 records grouped by automation tool. Those results are independent from the source of the operation – in this case front-end framework.

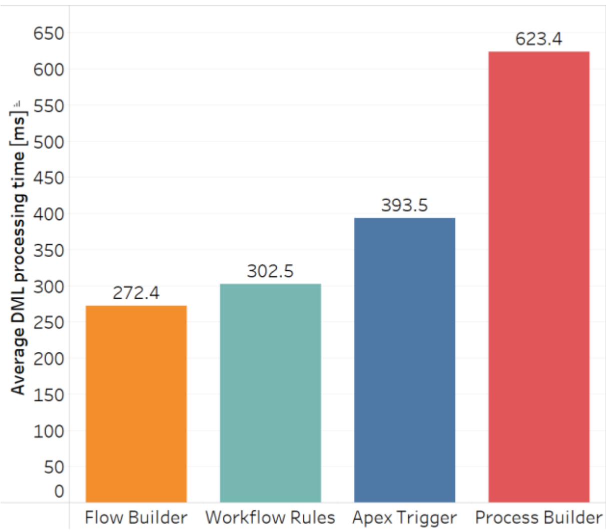

Figure 5: Average DML insert processing time for 200 records.

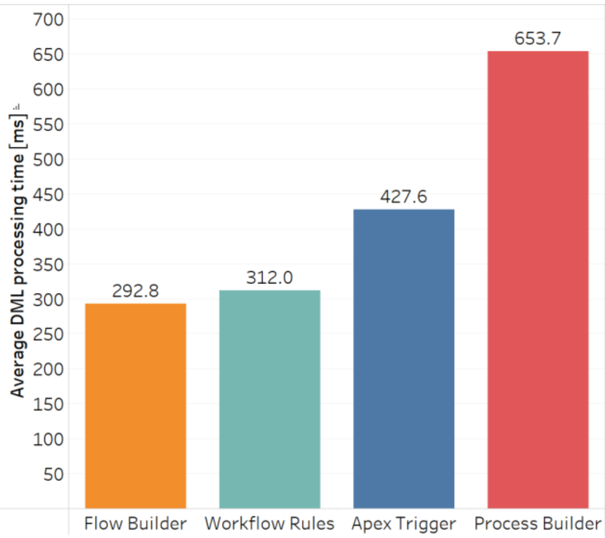

Figure 6: Average DML update processing time for 200 records.

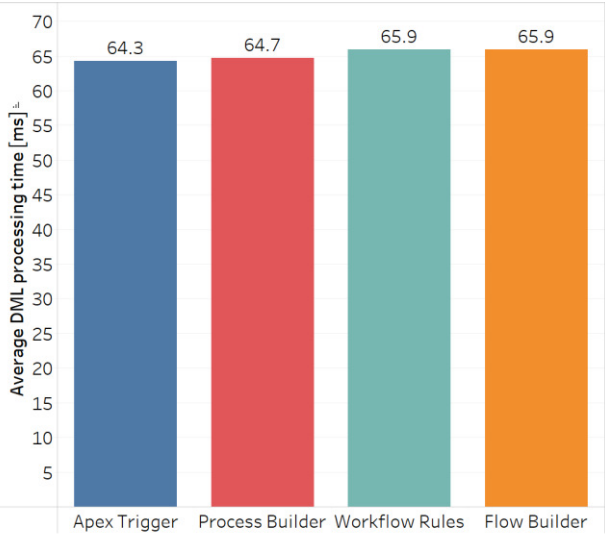

Figure 7: Average DML delete processing time for 200 records.

Similar results were obtained for 2500 records processing with the *Process Builder* having the most influence and greatly extending processing time. Figures 8 to 11 shows average processing time per record for each automation tool.

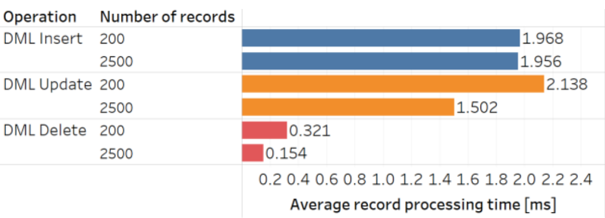

Figure 8: Average record processing time for *Apex Trigger*.

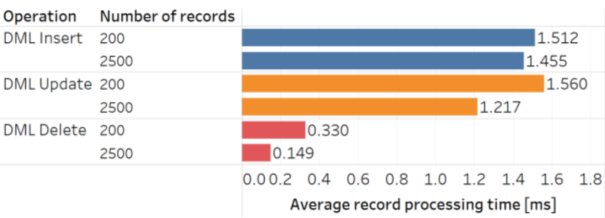

Figure 9: Average record processing time for *Workflow Rules*.

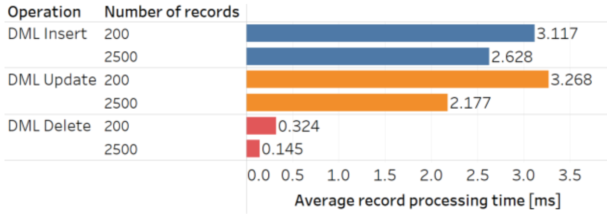

Figure 10: Average record processing time for *Process Builder*.

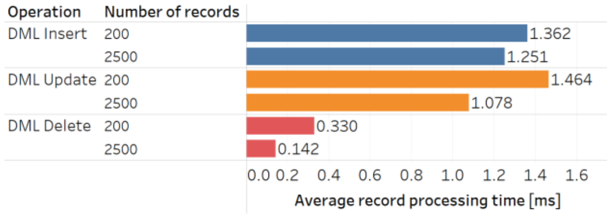

Figure 11: Average record processing time for *Flow Builder*.

Table 6 summarizes average time for whole transaction grouped by DML operation type, number of records and framework.

Table 6: Average transaction time by operation and framework

		Framework		
Operation	Number of records	LWC (ms)	Aura (ms)	Visualforce (ms)
insert	200	1130	1137	2045
	2500	13158	12957	16431
update	200	1148	1200	2258
	2500	13325	13851	17037
delete	200	860	784	962
	2500	5621	5765	5873

5.2. Heap memory consumption

Heap memory consumption depended not on automation but the source of the operation (framework), number of records and type of the operation. Figures 12 to 14 show heap memory consumption for each operation grouped by framework and number of records.

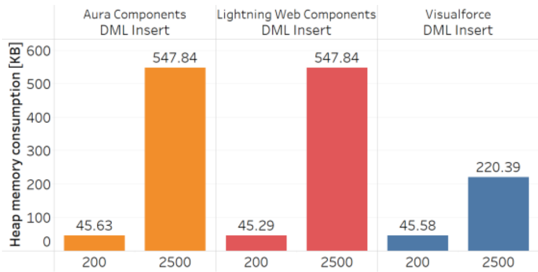

Figure 12: Heap memory consumption for insert operation.

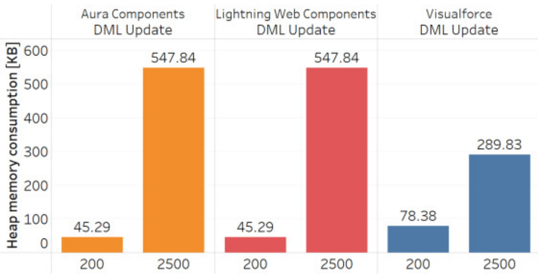

Figure 13: Heap memory consumption for update operation.

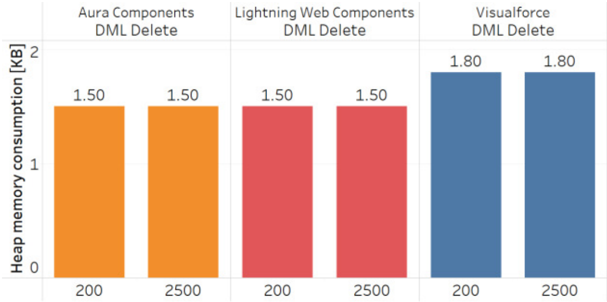

Figure 14: Heap memory consumption for delete operation.

5.3. Network requests

Using *Chrome Developer Tools*, time, count and size of the network requests were measured for each operation. Figure 15 shows average network request size grouped by framework, operation type and number of records.

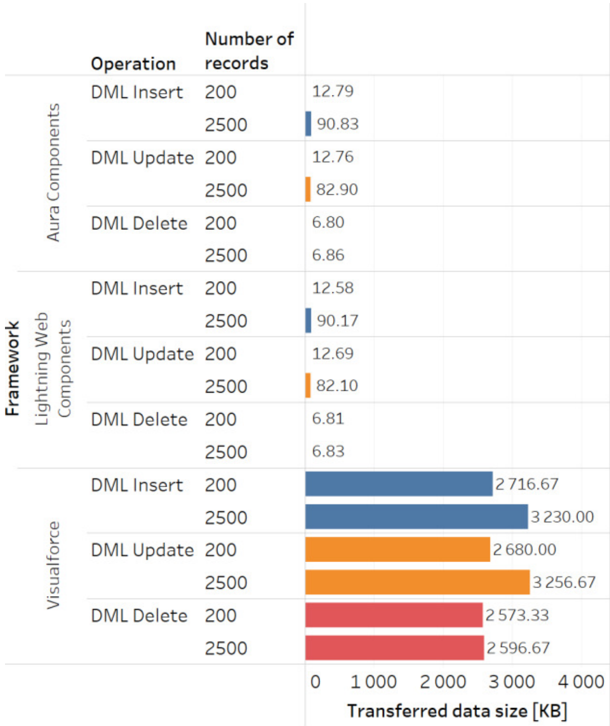

Figure 15: Average network request size by framework.

Table 7 summarizes the number of requests sent in total when executing the scenario 30 times. This value did not vary by chosen automation tool.

Table 7: Overall number of requests sent

		Framework		
Operation	Number of records	LWC	Aura	Visualforce
insert	200	33	33	810
	2500	33	33	810
update	200	33	33	810
	2500	33	33	810
delete	200	33	33	810
	2500	33	33	810

6. Results analysis

The type of enabled automation did not affect the time of the delete operation. The deletion time for 200 records oscillated at ~62 ms, while for 2500 records it was around 350 ms (figures 16 and 17). The difference between the minimum and maximum value for 2500 records is less than 250 ms, i.e. only 2% of the CPU time limit.

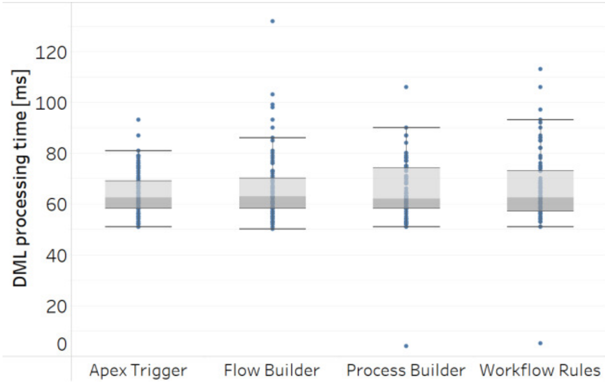

Figure 16: Deletion time for 200 records

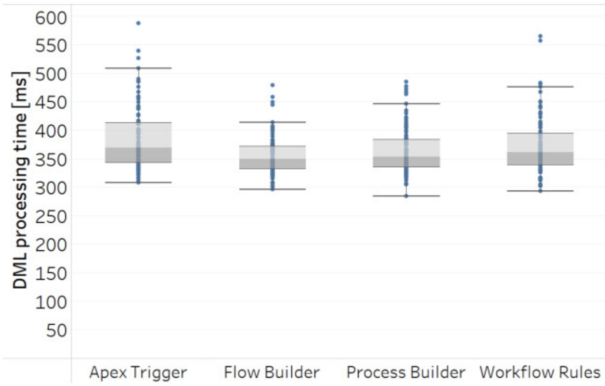

Figure 17: Deletion time for 2500 records.

Process Builder has the most negative impact on record creation and update. Despite its simplified graphical user interface, both insert and update operation were 2 times slower than the fastest *Flow Builder* tool. Having platform's limits in consideration, *Process Builder* uses up to 65% of available CPU time during bulk (2500) operations, leaving limited amount of time for the other operations.

Apex Trigger code ranked penultimate of the four analyzed automation tools. *Workflow Rules* proved to process records ~30% times faster compared to the code. The most performant solution was found to be *Flow Builder* – regardless of the number of records being processed and regardless of the operation type, it performed those operations the fastest.

Correlation can be observed between the front-end framework and the use of server and network resources. *Visualforce* used on average almost twice as much server-side time when creating and updating 200 records compared to the Javascript-based frameworks. When creating or updating 2500 records, the time increased by ~3 seconds. For number of network requests, *Aura* and *Lightning Web Components* achieved similar results, with a total of 33 requests per DML operation sending minimal amounts of information (a maximum of 90.83 kB, and a minimum of 6.8 kB). *Visualforce*, due to its server-side rendering technology, uses much larger amount of network and server resources – for 200 records, it sent around 2.5 – 2.7 MB of data and for 2500 records – 3.2 MB. Total number of requests added up to 810 during execution of the whole scenario.

Every framework during operation on 200 records achieved similar results in context of heap memory usage. Only for insert and update operations *Visualforce* achieved lower memory usage of 47% (update) and 60% (insert) than the *Aura* and *Lightning Web Components*.

7. Conclusions

In case of automation tools, both in terms of functionalities and the performance of the operations performed, *Flow Builder* turned out to be unquestionable choice. *Workflow Rules* has many limitations and *Process Builder* is highly unoptimized for record processing. *Flow Builder* is a tool tailored for development by citizen developers and can be supported with actions provided by *Apex* developers. Obtained results partly overlap with those presented by Appleman [13], where *Process Builder* was also found to be the slowest tool, although the author obtained worse results for single record processing.

For creating graphical user interfaces, *Lightning Web Component* is the preferred framework.

Visualforce is an outdated tool, offering the lowest performance. Despite using less heap memory, it consumes incomparably more client network resources, which is a higher priority criterion in this analysis.

There are no noticeable differences in the performance results obtained for *Aura* and *Lightning Web Components*, but *LWC* is better suited to modern application development standards than *Aura Components*. Within the Salesforce Platform, both frameworks offer the same capabilities, but *LWC's* open source nature, support for the latest *ECMAScript* specification and architecture based on native *Web Components* make it more suitable choice in the long run – the entry threshold should not be high for developers with experience in another front-end technology.

The results presented in this paper suggest a path for related research in the future. For a more thorough analysis, it would be useful to narrow scope of the benchmarking tests (e.g. comparing only *Flow Builder* with *Apex* code) but implement more complex actions. The *Lightning Web Component* framework allows application to be developed outside of the Salesforce platform, enabling comparative analysis against another popular framework: Angular or React.

References

[1] R. Waszkowski, Low-code platform for automating business processes in manufacturing, IFAC-PapersOnLine 52(10) (2019) 376–381, https://doi.org/10.1016/j.ifacol.2019.10.060.

[2] N. Carroll, L. Móráin, D. Garrett, A. Jamnadass, The importance of citizen development for digital transformation, Cutter IT Journal 34(3) (2021) 5–9.

[3] J. Wong, M. Driver, P. Vincent, Low-code development technologies evaluation guide, Gartner, 2019.

[4] Gartner Forecasts Worldwide Low-Code Development Technologies Market to Grow 20% in 2023, https://www.gartner.com/en/newsroom/press-releases/2022-12-13-gartner-forecasts-worldwide-low-code-development-technologies-market-to-grow-20-percent-in-2023, [06.03.2023].

[5] Salesforce ranked #1 CRM Provider for Ninth Consecutive Year, https://www.salesforce.com/news/stories/salesforce-ranked-1-crm-provider-for-ninth-consecutive-year/, [21.01.2023].

[6] Lightning Web Components, https://lwc.dev/, [07.02.2023].

[7] H. Thanduparakkal, P. Shahad, C. G. Raji, Using Salesforce to Build Real Time Covid 19 Tracker with Cloud Computing Technology, Proceedings of the International Conference on Applied Artificial Intelligence and Computing, ICAAIC, Salem, India (2022) 942–948, https://doi.org/10.1109/ICAAIC53929.2022.9792802.

[8] V. Sharma, S. Saraswat, S. Verma, P. Banga, D. Gupta, Cost-Effective Data Mining Application Covid-19 Analyzer, Proceedings of the 5th International Conference on Information Systems and Computer Networks, ISCON, Mathura, India (2021) 1–5, https://doi.org/10.1109/ISCON52037.2021.9702328.

[9] A. Poniszewska-Maranda, R. Matusiak, N. Kryvinska, Use of Salesforce platform for building real-time service systems in cloud, Proceedings of the IEEE International Conference on Services Computing, SCC, Honolulu, HI, USA (2017) 491–494, https://doi.org/10.1109/SCC.2017.72.

[10] R. Gupta, S. Verma, K. Janjua, Custom application development in cloud environment: Using salesforce, Proceedings of the 4th International Conference on Computing Sciences, ICCS, Jalandhar, India (2018) 23–27, https://doi.org/10.1109/ICCS.2018.00010.

[11] D. R. Miącz, Analiza wydajności metod tworzenia aplikacji w technologii Salesforce, Journal of Computer Sciences Institute 10 (2019) 24–27, https://doi.org/10.35784/jcsi.189.

[12] W. Marańda, A. Poniszewska-Marańda, M. Szymczyńska, Data Processing in Cloud Computing Model on the Example of Salesforce Cloud, Information 13(2) (2022) 85, https://doi.org/10.3390/info13020085.

[13] D. Appleman, R. Watson, The Dark Art Of CPU Benchmarking, https://www.salesforce.com/video/296515/, [16.02.2023].