sciendo

Carpathian Journal of Electronic and Computer Engineering 14/1 (2021) 11-19

DOI: 10.2478/cjece-2021-0003

Designing a Low-Code CRUD framework

1st Eng. Bogdan Văduva
Automation and Computer Science Faculty
Technical University of Cluj-Napoca
Baia Mare, Romania
bogdan_vaduva@yahoo.com

2nd Phd. eng. Honoriu Vălean
Automation and Computer Science Faculty
Technical University of Cluj-Napoca
Cluj-Napoca, Romania
honoriu.valean@aut.utcluj.ro

***Abstract*— Nowadays programmers write source code for inserting, editing and deleting records of a relational table. The majority of commercial relational databases include a specific management tool that offers such possibilities and most database programmers take this ability as granted. When it comes to real life applications, programmers use Object Oriented (OO) paradigm to build user friendly windows/screens/forms for database operations. The current work shows a different approach using a Low-code CRUD (Create, Read, Update, Delete) framework. Views and guidelines of how to design a Low-code CRUD framework will be detailed. "Low-code" motivation is due to the fact that the new framework will provide the ability to use less code in order to build fast and efficient complex applications. It will be up to the reader to envision a specific framework.**

Keywords—Low-code, database design

I. Introduction

Low-code frameworks emerged in the world of programming, because, the application designers realized, that some coding practices included repetitive steps and those steps could be automated. By doing so, they've enhanced the way coding was done. Programmers became more efficient and were able to write more application related code. Now days the Low-code frameworks offers visual tools with drag-and-drop features. In the near future we think that AI (Artificial Intelligence) will even enhance the way these Low-code, No-code and Zero-code frameworks works.

The term "Low-code" was introduced in 2014 by Forrester to refer, platforms focused on development simplicity and ease of use [1] [2] [5]. The main facility of a framework is the fact that it is used by many programmers and the end result (code) become unified, clean and easier to understand. The programmers frequently and efficiently use frameworks, therefore, today a new term emerged: "No-code"; within this paradigm the users, based on their permissions and roles can build or enhance parts of applications [1][2].

In this paper we will focus only on the term Low-code.

Forrester definition of low code is:

"Low-code platforms enable rapid delivery of business applications with a minimum of hand-coding and minimal upfront investment in setup, training, and deployment". [2] [5]

A similar Forrester's definition of "Low-code" was introduced by Gartner [5]:

"Low-code development both describe platforms that abstract away from code and offer an integrated set of tools to accelerate application delivery" [5].

Low-code term do not refer to user ability to build / enhance its own application, but to the programmer's ability to use a tool that allows them to build faster and better [1-10].

Low-code platforms were categorized as: workflow-based and code-generation [8]. Our paper can be placed in the first category as workflow-based, because it focuses on enhancing database operations and as result the workflow.

Motivation of the current work

During a 2005 extended project the initial analysis revealed that a huge amount of time will be spent on building CRUD related windows and screens. But can those operations be automated? Here we found ourselves in life changing dilemma. Should we go with a OO approach where we have to build classes for every table and then do the CRUD operations using those classes or build some kind of framework to automate the whole CRUD window/screen building process. We went with the second approach. At that time the term Low-code was not very well known but for the project was an efficient direction to follow. The "Low-code" approach helped the company to finish the project within budget and time.

Later the following questions arise:

Such framework can be built for any programming language or application type, desktop or web? Yes, it can.

Some of the main results of using a Low-code framework [5] are:

- The investment in training programs for employees will be reduced.

- The new employers will faster build new applications, with less training and therefore the business will have more revenue.

The paper continues in section II with the design, improvements and description of the framework, followed in section III by some practical results and performance test; the paper concludes with several observations over the proposed framework.

II. Design The Framework

A. Initial assement of the job

Our way of designing a Low-code framework was a little different (we think) from others approach. We start building the framework from a user kind of view and not from a programmer point of view. Here were our issues that we had to solve:

- Different users have different views of the application based on their application role/permissions
- Different views have different screens/user interfaces
  - Different screens/user interfaces have different fields to be shown (Figure 1)

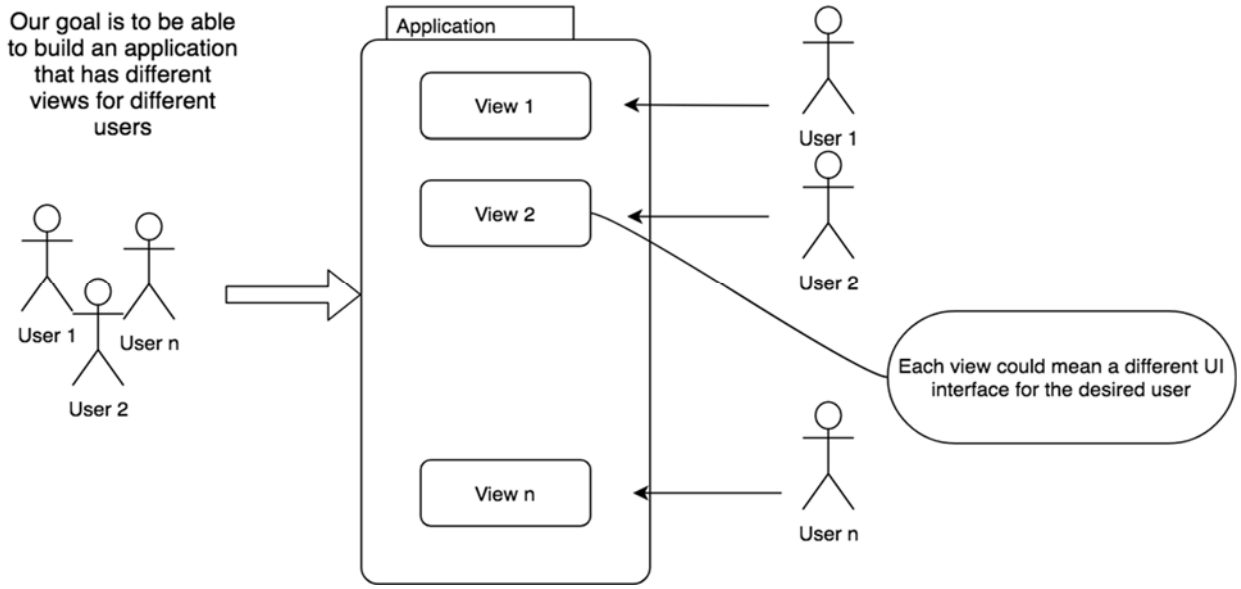

Fig. 1. Initial assement for building the framework

Based on these considerations two kind of information: user and role, are important for the beginning. The next step at this point was to figure out a place where to keep all the information required by the future framework.

Two types of approaches are required by the framework: integrate all the information required into an application database or include all that information into one or multiple configuration file(s). Either approach have advantages and disadvantages. The paper further uses the second approach, which was to include all the information into configuration files. Data was stored into configuration files as JSON (JavaScript Object Notation).

B. Design details

After the initial framework assessment, we moved forward towards the configuration files. So, the first configuration file was at first as in Code 1 (Table1):

```
{
        "user": {
                "name": ….
                "role": ….
                "description": ….
                }
        "table name": {
                "field1": "type"
                 ...
                }
}
```

Table 1 - Code 1 – First attempt of a configuration file

At this point our framework did not have any information about tables location (where the data is stored), the components required to display table fields and which table fields are the ones that the current user can see or modify. But there is more. An application's screen/user interface can display information from multiple tables. How are we gone handle that? So, the following modifications were made as in Code 2 (Table 2):

```
{
        "user": {
                "name": ….,
                "role": ….,
                "description": ….
                }
        "tables": {
                "table name1": {
                        to do
                         ...
                        },
                "table name2": {
                        to do
                         ...
                        },
                …
                }
}
```

Table 2 - Code 2 – The second attempt of a configuration file.

Furthermore, we tried to figure out how the future framework might properly work. We realized that the above structure will cause to make a lot of configuration files for every case, which was not acceptable. The need of multiple configuration files surfaced. So, we considered the following structure Figure 2, Figure 3, Figure 4 and Figure 5.

1. a table configuration file

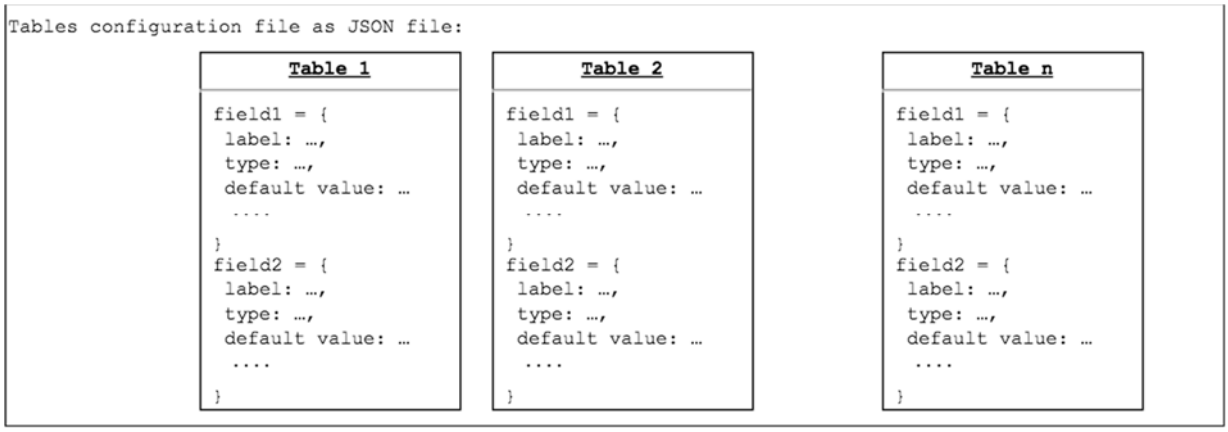

Fig. 2. Table configuration file

2. a role configuration file

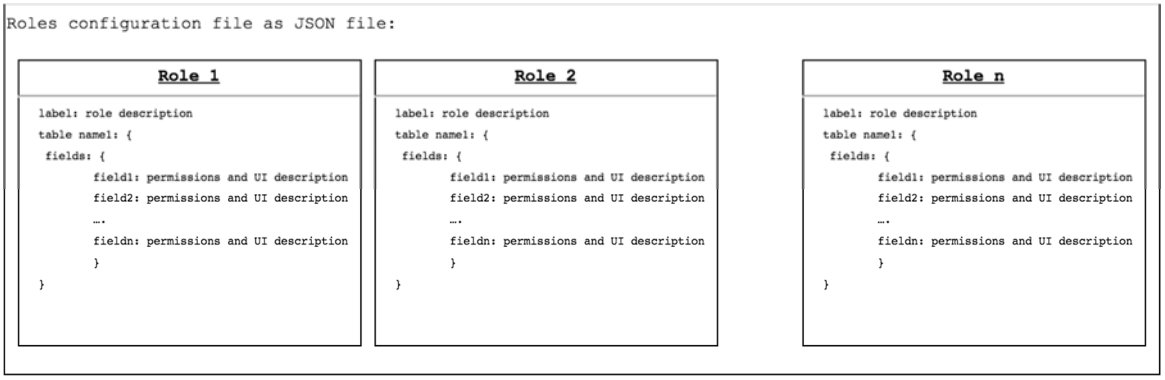

Fig. 3. Role configuration file

3. a screen/user interface configuration file which will take in consideration the current user's role

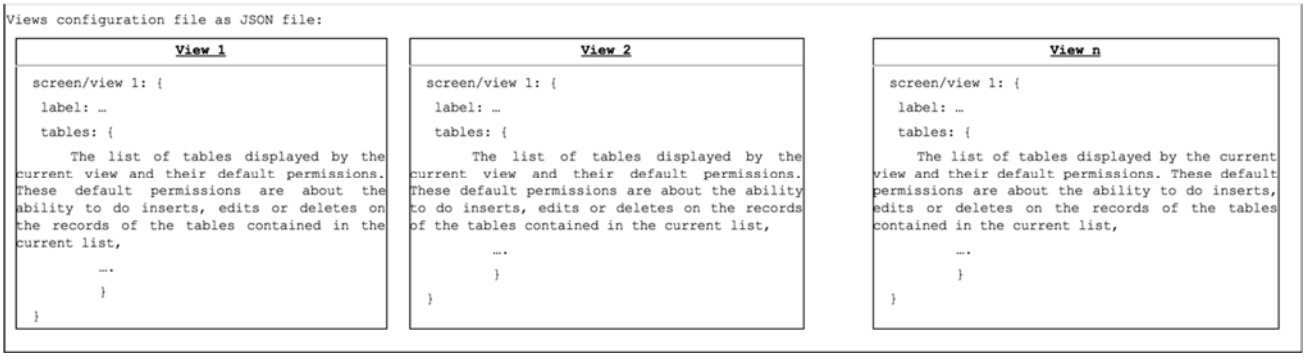

Fig. 4. Screen configuration file

4. a user configuration file:

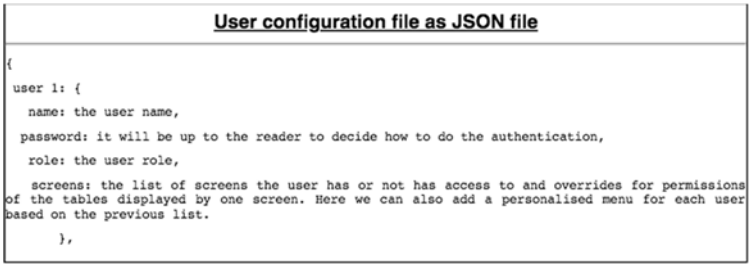

Fig. 5. User configuration file

Using configuration files as in Figures 2, 3, 4 and 5 we were able to build a CRUD framework. A detailed explanation of the current configuration benefits, will allow the reader to build a low-code framework. At first, all databases table will have a description that will be kept in the Figure 3 format.

Looking at the initial assesment of what the framework should do, we realized that any application that will use the framework should be built as following. The business analyst will evaluate the business model and try to put that model into a database and than describe the tables from the database into some configuration files. We realize that configuration files will overlap with database tables in terms of what are the table name, table fields and type, but the configuration files will have more (see Table 3). As can be seen in Table 3 we added into the configuration files labels, display type, default values, display format and other. Some extra information are kept for foreign keys. Usually foreign keys are displayed as list boxes, but a list box needs information regarding which fields from the foreign table are used in the list box as labels and which field is the actual foreign key. That information is kept in our table configuration file in three fields: "foreign_table", "foreign_table_label" and "foreign_table_value".

To actually visualize how a configuration file looks like we will give an example that can be seen in Code 3 (Table 3):

```
…
"tbluser": {
    "id":
    {
        "label": "ID", // here I used localized files values
        "type": "numeric, /* available types will be: char, numeric, date, boolean and other three framework types: hyperlink, label, calculated */
        "display_type": text, / * here I used a set of my own defined values: text – input, select – combo box / listbox, autocomplete – input with autocomplete, date – input with date selection, booleean – checkbox,  multicheckbox – a set of checkboxes with labels, feature – selection of a geographic feature, label – will be used with label type for adding dividers, hyperlink – will only be used in datagrid/datatable for having master / slave type of forms, calculated – will only be used in datagrid/datatables for displaying a calculated value, password – password input, textarea – textarea input, spinner – spinner input */
        "default value": "null,
        "foreign_table": null, // a foreign table containing the values for the current field
        "foreign_table_label": null, // a foreign table to be used for getting field options
        "foreign_table_value": null, // foreign table field value
        "foreign_table_label": null, // foreign table fields that will be used as labels
        "foreign_table_filter": null, // table filter
        "foreign_table_saved_value": null, // will be useful for have linked combo boxes
        "required": 0, // 0 or 1
        "required_message": null //default message will be shown if required will be 1
        "display_format": null //for formatting the displayed data
    },
    …
}
…
```

Table 3 - Code 3 – A real table configuration file

Beside the elements from Figures 3, 4, 5 and 6 the readers could add their own items. Next on our agenda was the ability to show different views/screen/forms for every user. We also said at the beginning that we want an application that has different views for different users. What we envision here was that users can login into the same application but get different user interfaces, depending on their role and permissions.

To do so, our proposal is detailed in Code 4 (Table 4):

```
{
    "Administrator": {
    "label": "This role was built for keeping the Administrator rights"
    "tbluser": {
        "label": "User table" /* this description could differ depending on the role */
        "fields": {
          "id": {
                "insert width": 100,
                "insert height": 25,
                "insert_row": 1,
                "insert_page": 0, /* 0 means that insert form will not have tab pages */
                "insert_permission": 1, // 0 –not shown at insert or 1 shown at insert
                "edit_width": 100,
                "edit_height": 25,
                "edit_row": 1,
                "edit_page": 0,
                "edit_permission": 1,
                …
                },
                …
          }
        }
```

Table 4 - Code 4 – Role specific settings

If we look at the above code we can see that we added information about how table fields should be displayed for a role/user.

Furthermore, we add descriptions for each view/screen in Code 5 (Table 5).

```
{
    "scren_tbluser": {
      "label": "A screen for managing application users",
      "tables": {
        "tbluser": {/* here we added default permission values for the table */
```

```
            "insert": 1,
            "edit": 1,
            "delete": 1,
            "view": 1
            },
            …
    }
}
```

Table 5 - Code 5 – User's screen specific permissions

The above code added some tweaks when it comes to display the screens. By using this configuration one user can have the ability to insert and edit the records of a table and other user will only have the ability to view the same information from the table. By enabling or disabling "insert", "edit", "delete" or "view" of the tables displayed into a view, users can have different user interfaces.

Discussions

The proposed low-code framework parsed the configuration files in order to build up a CRUD displaying element. Regarding this step we could arise a lot of questions like:

- What kind of patterns were used?

- How the code was organized?

- How efficient the code is (performance)?

In our opinion, these topics are strict related to each implementation and there are no success/error recipes.

At this point we believe that we should describe what we did in order to build our CRUD element. A CRUD element has a minimum of three views: insert view, edit view and table view. Each one is related to a specific task, like inserting records, editing records or viewing table records. At first, we focus on the table view which usually is the default view of a table. Here we used the table configuration file to build the SQL needed for extracting the information from the database table. We also used the role configuration file to restrict what information is displayed for a specific role. But this was just for viewing table information (for a role). Next, we focus on insert and edit view. We also used the same configuration files (table and role) to build the actual user interface. At this step we needed information like where to put an element into the user interface. This was accomplished by using configuration fields like: "label", "insert_row", "insert_page", "insert_height", "insert_width", "edit_row", "edit_page" and more (Table Code 3 and 4).

We said we want different users to have different user interfaces, but what happened if the users have the same role but still different user interfaces. Here it comes into place the next two configuration files: screen and user configuration files (Figures 4 and 5). By using those configuration files, we can further restrict or expand a user permission and thus change the user interface.

This paper focuses on how the configuration files were designed and what are the benefits of the current format. This focus on the configuration files is very important because gives the reader the ability to do his own design of the configuration files or database. At the beginning of our paper we said that we choose a solution where the Low-Code CRUD framework keeps it description in some JSON files, but the same description can be kept in a database. Keeping descriptions in a relational database or in some NoSQL databases gives the Low-Code CRUD framework architect more tools to work with. Here we are thinking about the ability to use existing designs as templates for future designs.

Several guidelines for the reader follows:

1. For every table passed to the CRUD framework, we parsed the configuration files to build the appropriate SQL needed for displaying the table data, to update the table data, to insert data into the table or delete data from the table
2. The SQL were built using the permissions from the configuration files for every user
3. Insert/edit screens were also built parsing the configuration files
4. Some particular enhancements should be allowed. For example, there are cases after an insert or update, when we want something to happen.
5. There are cases when we want to override the default permissions

The above guidelines are given only to raise the awareness to the readers.

As we already mentioned the framework was built using ASP.NET-C#, JAVA-JSP, JAVA-JSF, JavaScript/Angular 2+ and PHP, each implementation was different, depending on the particularities of each programming language/framework. The common element of all the implementations was the way we organized the configuration files, with small differences due to the programming languages.

As we said above we implemented the framework in many programming languages, but the most tested one are the one in JAVA-JSF and the one in JavaScript/Angular 2+. The first one created, was the one in JAVA and the second one, was the one in JavaScript/Angular 2+. The second one (JavaScript/Angular 2+) needs backend services for building table view, insert view and edit view. For those services we just used the old JAVA-JSF implementation with a few tweaks. It means we reengineered the original classes that built the table view, insert view and edit view to accommodate the new paradigm.

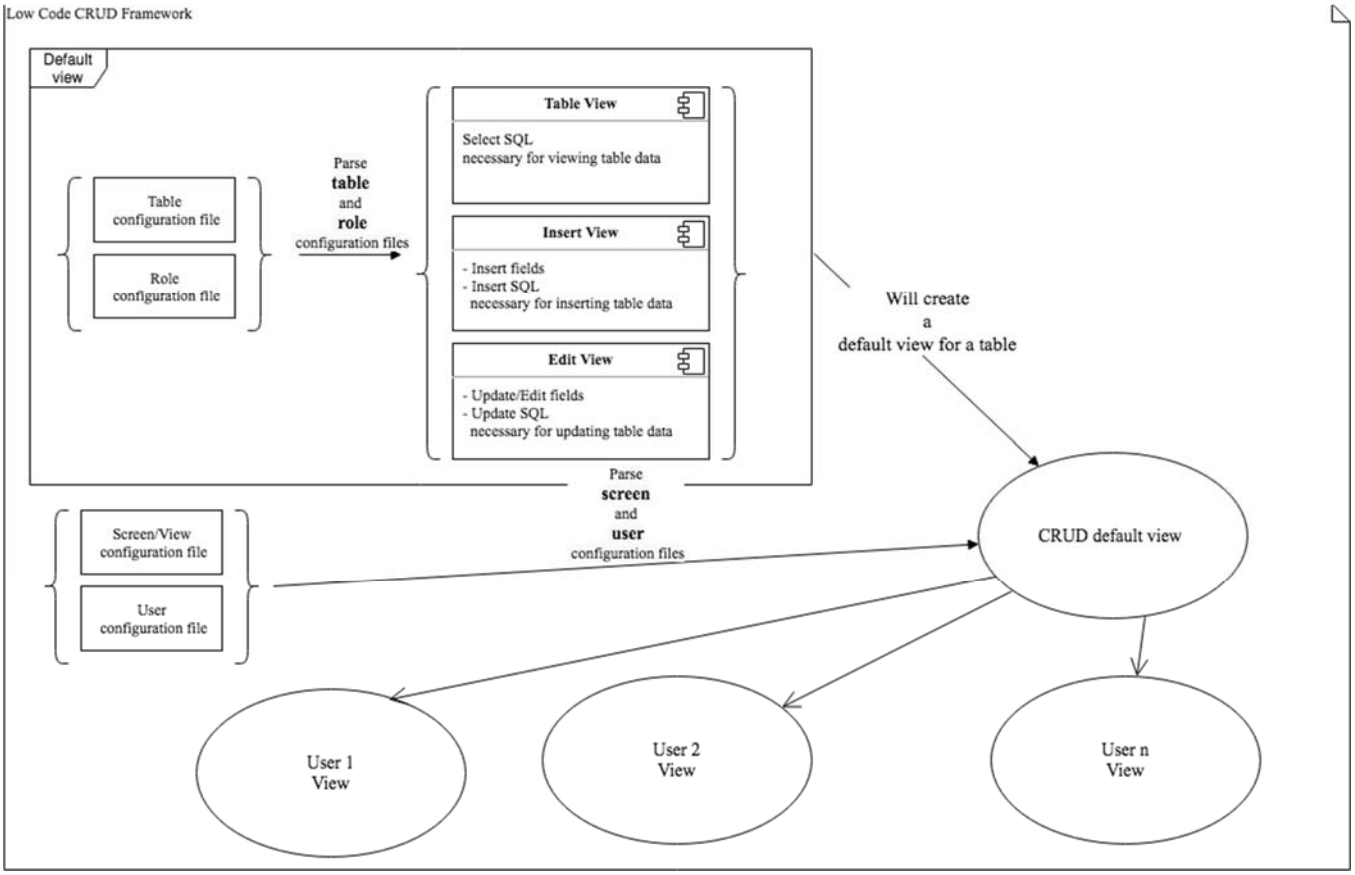

Fig. 6. CRUD framework workflow

At this point we had to add a few details about how the framework was used in a real-life application. Our use cases involved an application with 3+ relational databases (MariaDB, PostgreSQL, Oracle) where the CRUD user interfaces were described in the above configuration files. We said "an application" which is correct, because our application unified those relational databases.

We described all the application tables (table views, insert views, edit views, permissions) only once when we created a table or when we wanted to use an existing table from a relational database. After doing the descriptive phase a table was ready to be used in the whole application by simply calling the CRUD framework.

Overall, the newly created framework could be called "Low-code" because allowed programmers to build fast and efficient applications with a lot less code.

Improvements

At this moment one improvements seen by us is related to the way table and role configuration files are been being built. Here we believe that the database table creation step and CRUD framework table descriptive step could be somehow merged together. By doing so, the user/programmer time spent here will reduce even more.

Another improvement is related to the integrated development environment (IDE) were the CRUD framework should have a tool panel. Here we can add a table/role creation/description panel that will simplify the programming.

III. RESULTS

The obtained results are further shown with two use cases: one in Java/JSF and the other one in JavaScript/Angular 2+. We also implemented the new framework in ASP.NET and PHP having the same results: fast and efficient code.

At this moment we took some time to test and compare the performance of the two implementations. For this, we used an 8 GB of RAM, SSD computer with i7, 4th generation processor.

The first implementation, the one in JAVA/JSF had a time of 1282 ms for loading the framework (Figure 7).

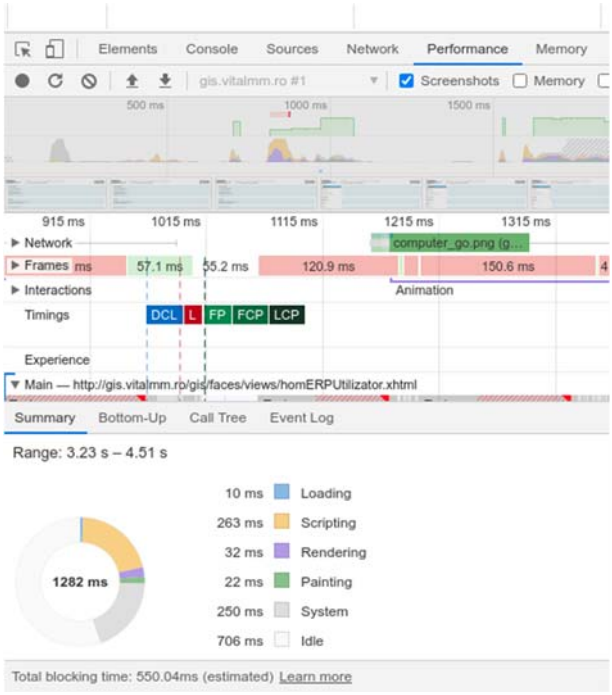

Fig. 7. Java/JSF CRUD framework first performance test

The second implementation, the one in JavaScript/Angular had a time of 1078 ms for loading the framework.

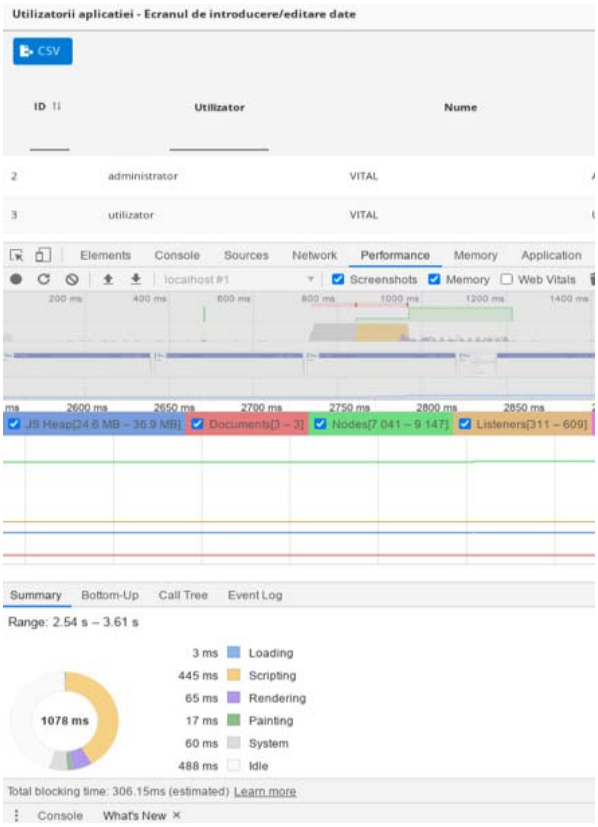

Fig. 8. JavaScript/Angular CRUD framework performance

Comparing the two implementations we can see some differences in loading time. Further we thought that performance can be influenced by computer performance, so we have redo the performance test for JAVA-JSF implementation (the most used one).

We used for the second test a 4 GB of RAM, SSD computer, i5 2th generation processor, obtaining a loading time of 1698 ms (Figure 9).

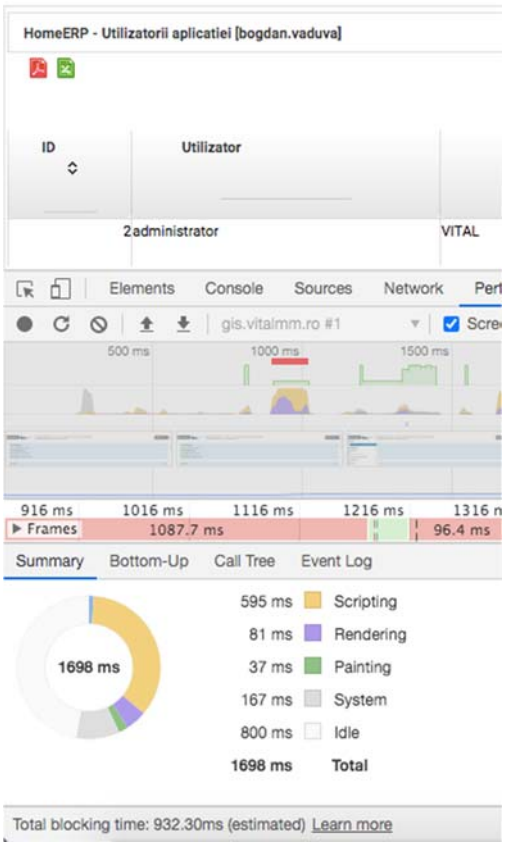

Fig. 9. JAVA/JSF CRUD framework second performance test

At this point we can say that our CRUD framework performs very well in different environments, giving us what we wanted: an easy to use and fast framework.

Java/JSF implementation called the framework as in Table 6 and the user interface is shown in Figures 9 and 10.

```
<herp:TabelaDataGrid id="uti" tabela="tblutilizator"
cheieprimara="id" inserareInchis="true" clientID="uti"
pagesize="15" casetype="5"
actionDupaEditare="utilizatorBean.dupaEditare"/>
```

Table 6 – Actual use case of Java/JSF framework

```
<homerp-dynamic-form [tableName]="'tblutilizator'"
[insertCollapsed]="true" (insert)="insert($event);"
(edit)="edit($event);"></homerp-dynamic-form>
```

Table 7 – Actual use case of JavaScript/Angular framework

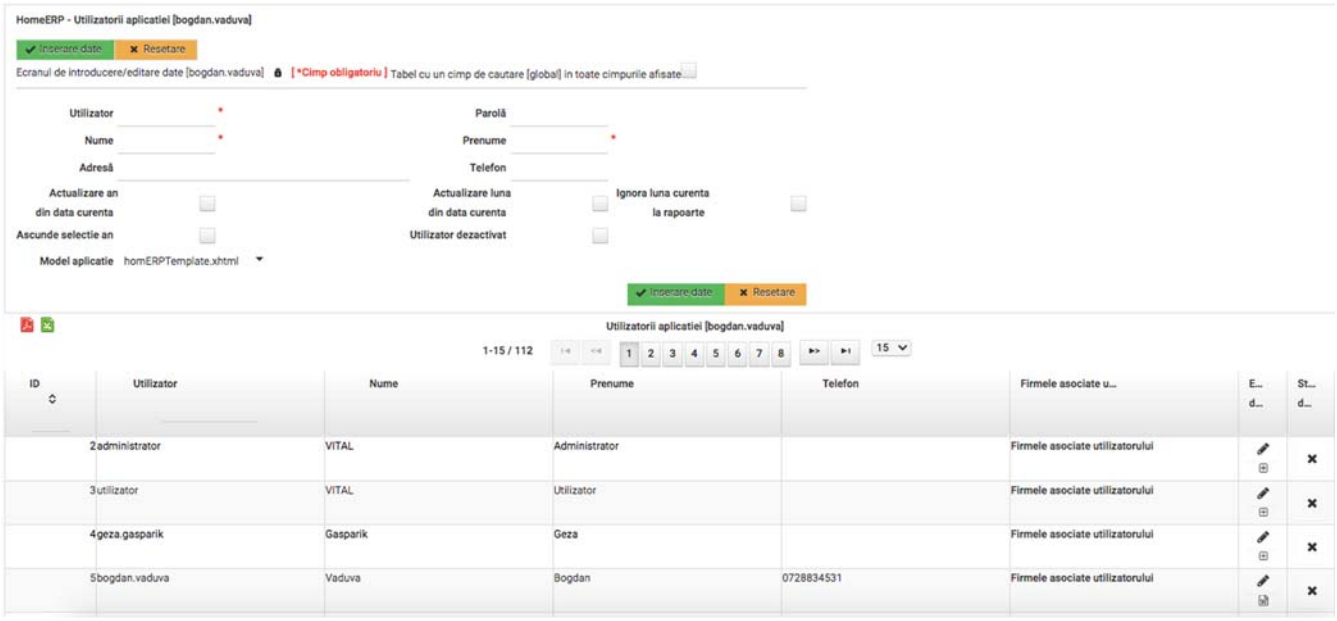

Fig. 10. Java/JSF CRUD framework example – default view

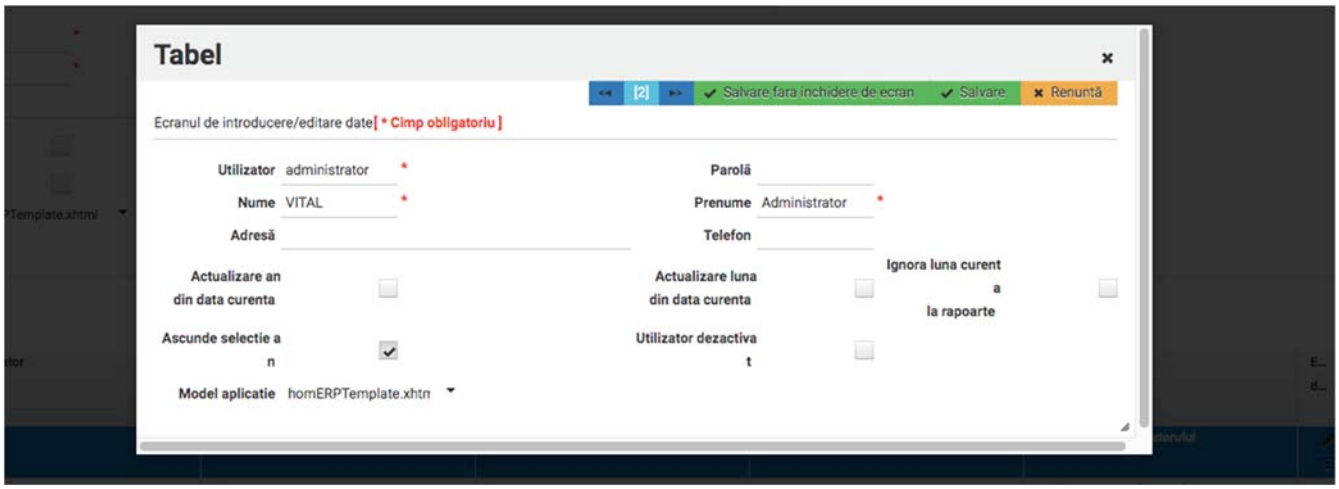

Fig. 11. Java/JSF CRUD framework example – edit view

JavaScript/Angular 2+ implementation called the framework in similar manner as in Table 7 and the actual user interface is shown in Figures 11 and 12.

JavaScript implementation was done using Angular 2+ (what is shown above is using Angular 9), but it also could be done using other JavaScript frameworks like React or Vue.

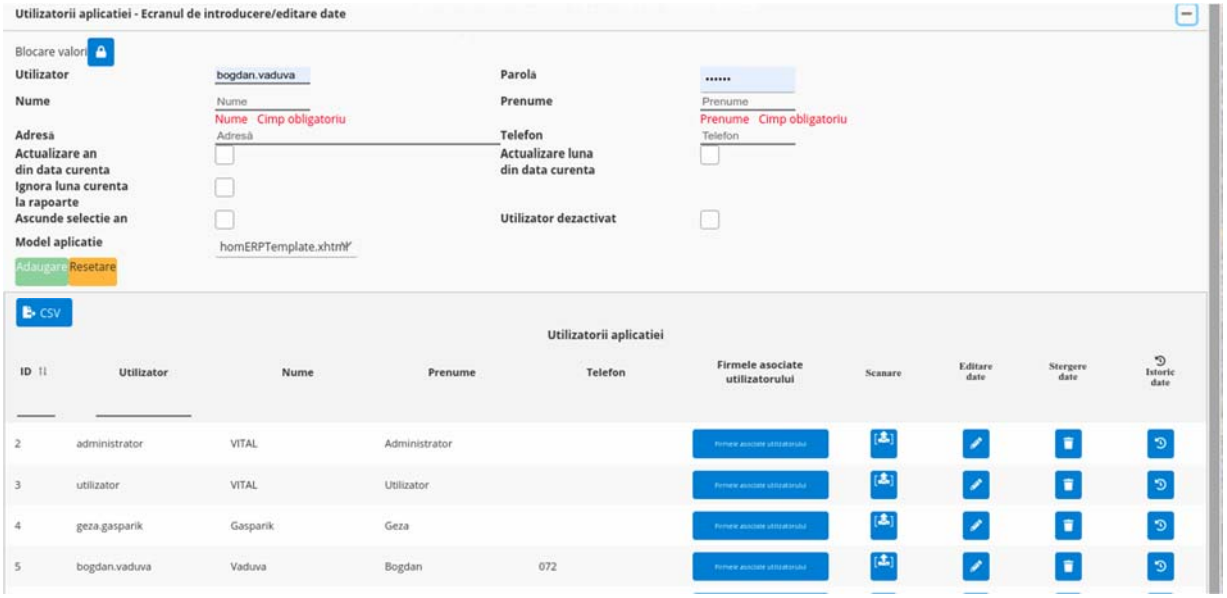

Fig. 12. JavaScript/Angular 2+ CRUD framework example – default view

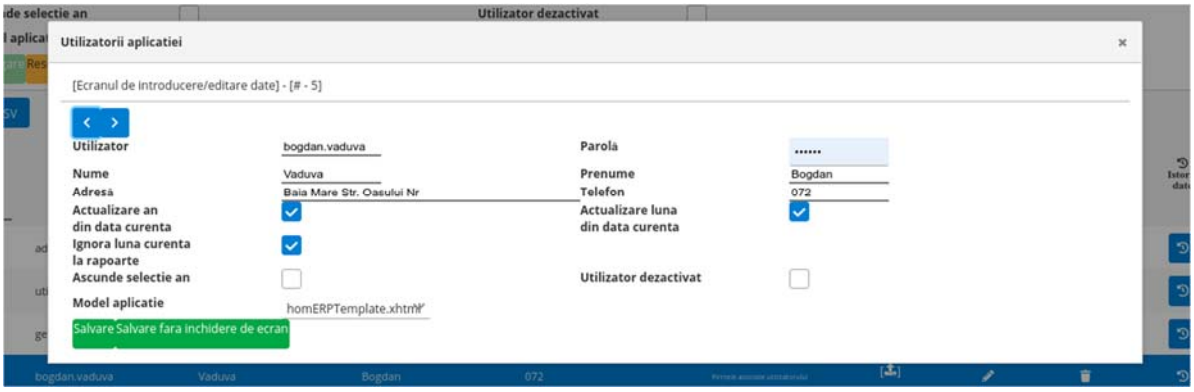

Fig. 13. JavaScript/Angular 2+ CRUD framework example – edit view

IV. Conclusions

In the current paper we showed a new perspective of building a Low-code CRUD framework. The new successful framework allowed us to build applications in a fast and efficient way. The platform preserves the quality of software. The framework could be further improved.

An overall conclusion is that the term “Low-code” is for nowadays, but, in the future the terms like "No-code" and "Zero-code" will be used for building applications. We consider that these terms combined with machine learning will deliver applications with less programming.

References

[1] Richardson, Clay (June 9, 2014). "New Development Platforms Emerge For Customer-Facing Applications".

[2] Richardson, Clay. "The Forrester Wave™: Low-code Development Platforms, Q2 2016". www.forrester.com. Forrester Research.

[3] Mendix. 2020. Mendix: IoT Application Development with a Low-Code Platform.https://www.mendix.com/building-iot-applications/. Retrieved May 2020 from www.mendix.com.

[4] Paul Vincent, Kimihiko Iijima, Mark Driver, Jason Wong, and Yefim Natis.2019. Magic Quadrant for Enterprise Low-Code Application Platforms.https://www.gartner.com/. Last accessed June 2020

[5] Kissflow. Your Starter Guide to Low-Code Development. Retrieved December 2020 from www.kissflow.com.

[6] Kony Inc. 2020. Leading Multi Experience Development Platform. Retrieved July, 2020 from https://www.kony.com

[7] Doug Hudgeon. Low-Code Platforms and the Rise of the Community Developer: Lots of Solutions, or Lots of Problems?, QCon Plus: Uncover Emerging Trends and Practices. Retrieved February 2021 from https://www.infoq.com/articles/low-code-community-developer/?itm_campaign=rightbar_v2&itm_source=infoq&itm_medium=articles_link&itm_content=link_text

[8] Miguel Valdes Faura. The Many Flavors of “Low-Code”, QCon Plus: Uncover Emerging Trends and Practices. Retrieved February 2021 from https://www.infoq.com/articles/many-flavors-low-code/

[9] Jason Bloomberg. Low-Code/No-Code? HPaPaaS? Hereś What Every-body Is Missing. Retrieved July , 2020 from https://www.forbes.com/sites/jasonbloomberg/2018/07/30/low-codeno-code-hpapaas-heres-what-everybody-is-missing (HPaPaaS - High Productivity Application Platform-as-a-Service)

[10] Faezeh Khorram, Jean-Marie Mottu, Gerson Sunyé. Challenges & Opportunities in Low-Code Testing. ACM/IEEE 23rd International Conference on Model Driven Engineering Languages and Systems (MODELS ’20 Companion), Oct 2020, Virtual, Canada.