iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 11 **Issue:** VII **Month of publication:** July 2023

DOI: https://doi.org/10.22214/ijraset.2023.54928

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 11 Issue VII Jul 2023- Available at www.ijraset.com

Kore: A Friend in Web Development

Chandrakant Koneti[1], Likhith Reddy Rechintala[2], Jaya Sriharshita Koneti[3]
[1]*Department of Information Technology, Indian Institute of Engineering Science and Technology, Shibpur, India*
[2, 3]*Department of Computer Science and Engineering, Chaitanya Bharathi Institute of Technology, Hyderabad, India*

Abstract: ***The technological breakthroughs and innovations that are taking on across the world today are endless. Things are getting indisputably simple in all imaginable ways. Technologies simplify people's lives, as they perfectly fill the void and propel people towards a solution. Nowadays, people tend to endorse the technical advancements that ease their life. Anything that bridges the gap between people's issues and solutions gets highly upvoted. But these technological advancements bring the necessity of extreme technical abilities that might be a hindrance for a few people to get a hand on them. In today's modern era, web applications have an extraordinary impact on people's lives. They play a significant role in individuals' lives, as well as in enterprises. However, developing one is tiresome. A few open-source softwares exist that expedite the web application generation process. "JHipster" is one such software that assists people in generating a web application. However, working with JHipster is a bit challenging as people need insights into the technical aspects. To address this issue, we propose a solution, "Kore". Kore is a creative approach to minimising a person's effort to generate a web application. This paper discusses the implementation of Kore and its role in solving the above-discussed issues. Kore can demolish the impediments and assists people wanting to generate a web application effortlessly and lets them experience the impact of these technological advancements.***

Keywords: Kore, JHipster, database creation, backend, software, generation tool, monolithic and microservice architecture, low code platform, text-to-speech, speech-to-text, Android.

I. INTRODUCTION

Technology is driving the world smarter day by day. Today, we can witness many technical advancements in various aspects of technology. The Internet is one such example that brought the entire world together. Today, we can talk free of cost with a person who lives in distinct nations around the globe, and the praise for this goes to internet calling. Also, we have many tools that support the differently abled in doing their daily chores, like the text-to-speech tool, image reading tools, AI tools that recognize hand gestures, and many more. Text-to-speech and speech-to-text are among the advancements that help differently-abled people. It is a perfect technological development that aids a wide-ranging audience around the globe. Technology fits accurately in the void space that people feel today. Our daily work has been much more comfortable with all the technical advancements until now.

Web and mobile applications are crucial, as they have a considerable influence on individuals across the globe. There were days when people were unfamiliar with what web applications were. Now, we live in a world where people own web applications, and significant advances in web applications paved the way for code-generation tools. Web applications have become an exigency to peoples' tasks in the modern world. Creating a web application from scratch is a tardy process, and code-generator tools came in as a saviour in encountering this issue. Code generator tools write the code for a web application by collecting a little information on the technologies that web applications must possess. People create most web applications that differ only in the types of technologies used, but they remain similar in architecture. Hence, a significant extent of this web application creation process could be automated. The only things that vary between web applications would be the different technologies used. Code-generation tools assist in generating web application code by automating the process as discussed above. Code-generator tools can automatically build a web application using the information from the users about the various technologies required.

One such code-generation tool is JHipster, which generates a web application based on users' preferences. The open-source software termed JHipster is used as a development platform to create an application's backend code [1]. It is an advanced tool among a few currently available solutions that allow a user to build a web application by accepting a few inputs [2]. People can build applications with either monolithic architecture or microservice architecture using JHipster. In the simplest terms, the microservice architectural style is a means to create a single application that comprises several little services, each of which runs independently and communicates with other services using a limited set of tools, often via an HTTP (Hyper Text Transfer Protocol) resource API (Application Programming Interface). These services are made to be self-contained and deployable using completely automated deployment equipment [3]. However, working with JHipster needs some mandatory dependencies like JHipster installation on their PC (Personal Computer), prerequisites about the tool, and many more.

JHipster is a bit complex to work on, and we had an idea of improvising the process to make it easier for a naive developer. Novice developers might not be capable of comprehending the prerequisites for generating a web application using JHipster. Also, they might find the flow of this high-level software a bit more sophisticated and get lost amid the code generation process. There was a possibility that could fill in the gap between highly sophisticated software and a fledgling developer leading us to the rise of *"Kore."*

Our application, *"Kore,"* is a one-of-a-kind tool that generates the web application based on the user's choice of input using the text-to-speech approach. Kore is a mobile application built on the JHipster open-source tool. We have a wide range of support for users as we use an input type as text-to-speech. The agenda of Kore is to help people generate a web application more simply and effortlessly. In this paper, we put forth the vision of our project titled *"Kore."* Kore is a very user-friendly interface used to generate web applications. Kore's UI (User Interface) is very comprehensible and encourages the users to provide their required web application details vocally. Users can acquire the web application with their choice of technologies instilled in it in the form of a ZIP file just with their voice commands. Though Kore internally works on the base of JHipster for generating the web application, the user isn't mandated to interact with the intricate JHipster tool directly. As an outcome, the procedure is now more streamlined and straightforward.

II. METHODOLOGY

Developing a web application is a tedious task, as discussed above. Every web application has the same fundamental aspects common to all of them.

A developer should build all these for every web application they want to develop. Hence, this process becomes redundant and can be cut short with the help of a code-generator tool like Kore. We are automating the code generation process using the users' choices taken in the form of verbal input. We provide a text-to-speech feature wherein the application interactively speaks to the user and helps them to answer the required questions in the application's code generation process. We interact with the user in every stage of the application generation process and proceed to the next steps accordingly. Users must submit vocal input for some technologies mandated for functional application code generation.

We have developed an Android mobile application named *"Kore"* to perform all these actions. We have used Android, Java, XML (Extensible Markup Language), a build tool known as Gradle, and an SQLite database. Kore UI communicates with the backend via REST (Representational State Transfer) API. Any data that UI passes to our backend gets transmitted via REST API. Our UI task is to gather the data from the user, use built-in Android speech-to-text libraries to convert users' verbal input to a transcript, send it to our backend application, and reiterate the questions in case users provide an invalid response. Validating the users' responses is handled by our backend application.

We gather data from the users in two distinct ways. The first approach resembles an automatic process where the UI narrates questions to the users, and once UI gets done with this, it starts listening to the response. Users will have fifteen seconds to reply with their choice right after Kore UI narrates the question. Our UI would listen throughout this time to capture users' input and thereby passes the vocal response to the speech-to-text library. If users fail to provide information within the stipulated time, our UI notifies the users and reiterates the question. The second approach is a more straightforward method where users can click on the voice input button to interrupt and provide voice input immediately.

The provided verbal response is converted into text and shown to the users on the UI. The converted text will then be sent to our backend via API calls to validate them. If it is an invalid response to the question asked, our backend application sends a message to the UI that the provided input is invalid. If it is valid, the application stores it and sends information to the UI that provided information is correct. We use a rule-based approach to save the data if user-provided data is valid. To recognise named items using language-specific properties, rule-based techniques are intuitive [4]. We define triggers and actions to collect accurate data. We specify actions attached to the predefined rules we set so that corresponding actions execute when relevant triggers invoke. If any keyword we are searching, particular to the question, is identified, the predetermined rules we define take effect. These actions store the validated data that goes into the code generation process.

Our backend application sends this data to JHipster by communicating through REST API calls and executing several commands in JHipster CLI (Command Line Interface). Once the web application gets generated, users can download the code. Kore reduces a developer's effort by communicating with JHipster and performing the necessary actions on their behalf. Figure 1 depicts the entire flow of our application.

Figure 1. Kore flow chart

The various swim lanes displayed in Figure 1 demonstrate the distinguished responsibilities of the entities present. Figure 1 exhibits the segregated activities corresponding to the users, Kore UI, and Kore backend.

III. IMPLEMENTATION

Users are essential in initiating the process of developing web applications with Kore. Kore is an Android application designed to assist users in developing a web application in fewer steps. Kore ensures that its users have minimum interaction with the interface in a creative way. Kore offers an intelligent platform powered by sophisticated libraries, machine learning algorithms, and AI (Artificial Intelligence). Users are just required to speak to our application, and we generate a web application for them. We categorized this whole process into three phases.

A. Gathering information from the user

This phase is all about gathering information from the user, which is essential for generating code for a web application. Initially, people can sign up to become Kore users if they are not a member of the Kore community. Subsequently, they can log in to the website to employ the features and functionalities of Kore. Once they have signed in successfully, they can begin building a web application on our platform. Our application starts asking questions about the required web application soon after the user chooses to generate one. These questions include the technologies for developing the web application, such as the database, authentication types, client frameworks, and many more. This phase involves two-way interaction between users and Kore UI. It mandates the users to provide permissions for taking verbal input, as the consequent process relies on this. Our application will prompt users to choose among the options we provide for every question. Immediately after the UI narrates the question, it listens to the users' input. If users respond by speaking anything during this period, it will get captured by the UI. The UI would wait until 15 seconds to capture the data user provides. And if no response gets collected within the stipulated time, it will reiterate the question to the users. For example, to know the database to embed in the desired web application, our UI seeks feedback from the user in the following way.

*"**Kore UI**: Would you like to have a database for your web application?"*

Users can prefer to say *"Yes"* or *"No"* in response to this question. The following would be a possible response from the users.

*"**Users**: Yes"*

For the above question, if the user says "Yes," we will further inquire about the database type they like to choose. And if the user decides not to have a database by saying *"No,"* we will proceed without any database. In this case, the web application generated will not have any database connected to it. The following will be a possible conversation exchange between users and Kore UI if they say *"Yes,"* as in the above scenario.

*"**Kore UI**: Please let us know the type of database your web application must have.*
*Say **One** or **SQL**.*
*Say **Two** or **MongoDB**.*
*Say **Three** or **Cassandra**.*
*Say **Four** or **Couchbase**.*
*Say **Five** or **Neo4j**.*
*Say **Six** if anything works."*

For the above question, users can provide their input by communicating in return to the interface in the following way.

*"**Users**: One"*

In the scenario discussed above, Kore notes that the user wants to proceed with *"SQL,"* as he confirms by saying, *"One,"* which refers to the same. Alternatively, users can also vocally say *"SQL."* It is considered a valid response as the user implies having an *"SQL"* database in their web application and has confirmed it by verbally providing the keywords in the option. If the user has opted for an SQL database, we advance by asking about the production and development database type. Since the users have mentioned having an SQL database in this case, which is a relational database, they can select among H2, PostgreSQL, MySQL, MariaDB, Oracle, or MSSQL. All the databases listed above come under the category of relational databases that JHipster supports. A similar flow goes with non-relational databases too. The questions might vary depending on the type of database chosen in the first place. Users are asked appropriate questions according to the option type they opt for. Users can select their desired option by expressing the respective numbers or the keywords in the choices. Alternatively, if the user says *"Six,"* we will pick one among the supported database types on behalf of the users and proceed further if they do not want to pick one specifically. This feature of letting Kore choose the best option for users stands out from the native JHipster application, and it helps users if they are sceptical of the choice to select. Information about some technologies for web applications is mandatory, like the application name, the application type (*monolithic*, *microservice* or *gateway application*), the type of authentication, the tool for building the backend, and testing frameworks. Therefore, we mandate users to express their choice in such questions and will not proceed until they provide a valid answer. Since an application can work with or without databases or client frameworks, users can continue without opting for them.

Users can also enter the database details at the end if they choose to have one for their web application. They can choose between relational databases and non-relational databases. Users must provide every detail about the database, including table names, attributes, relations among tables, and pagination if they decide to have a relational database. Users can be able to perform this action in two ways. The first step involves creating another file. They can create a JHipster-supported file known as a JDL (JHipster Domain Language) file. This file includes the database details mentioned above. Users can create this file with the assistance of JDL Studio, provided by the JHipster community or manually with the file extension as ".jdl". Users must upload the generated JDL file if users choose this way to create database models. Users can see a separate screen where they can upload the generated JDL file. Our backend application will use this JDL file to create the custom database models. For this, users must be familiar with this file creation process which might be a little complex for a few developers. We offer the users another way that simplifies and enhances this process further. Using our approach, users may submit database specifics vocally, similar to how technology details get captured. Users can continue their journey of web application generation smoothly since they do not need to step out of the flow to create another file and use it to generate custom database models. Instead, they can proceed similarly by entering the details verbally. Users will see a prompt to select the way they want to input the database details if they choose to have a relational database. This prompt appears once all the answers about the intended web application technologies are collected. The following depicts the situation discussed above.

*"**Kore UI**: Please select a way to enter the database details.*
*Say **One**, or **Upload a JDL file**.*
*Say **Two**, or **Proceed with voice input**."*

For the question above, users can say *"Two"* or provide the keywords in the option vocally. Say, users express their choice in the following way.

*"**Users**: Proceed with voice input."*

As the user opted for entering the details of custom models vocally instead of the JDL file, we proceed to the next screen wherein the user will be asked questions about the entity details like entity name, attributes list, and relations with other entities (if opted for a relational database). Once the user finishes giving the input verbally, we validate it and proceed after the user is satisfied with the captured details. We also provide the facility to input details of multiple entities as well. To add more entities, the user has to choose the *"Add Entity"* option shown on the Kore UI or can say *"Add Entity"* at the end of capturing older entity details. Once users finish entering the details of one entity, our application asks them if they wish to enter the details of another entity, and this process goes on until the users decide to stop. The procedure of database details capturing, if users desire to have one embedded in their web application, ends here.

Also, the UI validates the user-provided choices and reiterates the question to the users if they are invalid. This validation happens at our backend application. Our UI and the backend application communicate through REST API calls. Every user-provided answer is sent to our backend, where the verbal input gets converted into a transcript initially, gets ratified later, and is stored in our database if it is a valid response. If the reply is invalid, our backend notifies the UI, which further informs the users that the answer they provided is incorrect. In this case, the UI repeats the question to the users to retrieve a valid response. In addition, several questions are posed to the user to learn more about the technologies to incorporate into the user-requested web application. Our application follows the rule-based system. It seeks the necessary information by examining keywords in the transcript generated from the user's verbal input. The rule-based system makes our application less complex and much easier to understand. Using the rule-based approach, we define "triggers" and "actions" that should take place by looking into the users' responses. These actions involve selecting the related option from the provided list for every question. The flow happens as mentioned below.

After the user-provided verbal response gets converted into a transcript, we examine for the keywords required to decide the option with which the users want to proceed. The corresponding actions would take place based on the triggers invoked. For example, let us assume we define a "trigger," which examines if the transcript has the word "Session." Also, we define an action where "Session based authentication" would be chosen as the type of authentication. Say, if the user decides to provide voice input as "Session," our defined trigger gets invoked, and the respective action, choosing "Session based authentication," would take place. As a result, the user succeeded in selecting an authentication type for their web application. A similar process goes on for all the questions regarding the web application. And the validation of users' responses also happens similarly, as discussed above. Our application will not advance unless users give complete information about the desired web application. Kore ensures that there is no possibility of data loss by saving the user-provided details every minute. Every user-supplied detail is stored and kept in the event of an interruption. Furthermore, this avoids the need to re-enter the details every time there is an interruption. Once Kore UI has gathered all the data successfully, we move on to the next phase, Code Generation.

B. Web Application Code Generation

Here, all the chosen options would come into play and help generate a web application. By the start of this phase, we have all the technologies that the users insist their web application must possess, along with the database details, and we will proceed accordingly to generate a web application. This phase requires no user-related actions and is automated. Users only need to wait until this automated process terminates so that they can access their desired web application. The user-selected technologies and the custom database model details get transmitted to our backend application through REST API calls. The communication between Kore UI and the backend application happens with the help of API calls.

Our backend application plays a crucial role in this phase, as we can access the complete information regarding the web application from here. With the gathered data given by the user, our backend application will interact with JHipster CLI to start the web application code generation. This phase is an entirely predefined process where only the user-provided data varies. The actions to perform and the commands to execute are predefined, and once our backend application gets the data, it automatically starts communicating with JHipster CLI. Our backend application will look after the steps needed to undergo and executes the commands accordingly. Once the web application code gets generated, we move on to the next step, downloading the code.

C. Providing ZIP file back to the user

In this phase, which is the final phase, users can download the generated code as a ZIP file onto their local machine. This ZIP file consists of all the files and folders related to the web application. Here, users can look at all the options they have chosen. Users can notice two different options here. If they are satisfied with all the preferred options, they can advance to download the code. If the users are not pleased and want to change anything in their inputs, they can return to the home screen. Users are now enabled to re-enter all the details about the web application.

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 11 Issue VII Jul 2023- Available at www.ijraset.com

Our UI asks the users whether they want to download the generated web application vocally once the code generation gets accomplished. Users can reply with a "Yes," or "Download" to download the code. Additionally, users can say "Return to Home" to restart the whole process if they are dissatisfied with the generated web application. The ZIP file, which is in a compressed state, should be uncompressed before starting to work on the web application. Post uncompressing the ZIP file, users can place its contents in their choice of directory. The generated web application, consisting of Java code to a large extent, can be deployed as a Spring boot project. These folders and files get created precisely as an actual spring boot project.

However, users must update a few things to run their web application immediately after downloading the code. Failing to do so will throw an error while building the project. Since there are two types of databases, production and development, users can notice two different *"application.yml"* files. Users must initialize one database by setting an active profile in the corresponding *"application.yml"* file. Also, users must have their chosen database on their local machine. If it is not present, they face an exception related to the database. In addition, users must enter their database credentials in the respective *"application.yml"* file to successfully communicate with the database. After these modifications, users can successfully run their generated web application. Finally, This concludes the procedure to create a web application using Kore.

IV. RESULTS

Kore enables its users to generate a web application effortlessly. Using Kore, users can simplify the process of code generation. The features of Kore ensure that users need not invest additional efforts to generate code for a web application. Unlike JHipster, where users need to interact with an advanced and complex CLI and run several commands to get the generated code, they can straightaway talk to our application to get the web application code generated in a single flow. We provide a single point of contact, *"Kore,"* where users can input all their requirements and wait to download the generated code rather than having to perform complex actions. Kore streamlines the code generation process with a unique approach, which only needs users to vocally provide the web application details, such as its architectural type, authentication type, database type and several others. Figure 2 illustrates the screen where users can enter the database details.

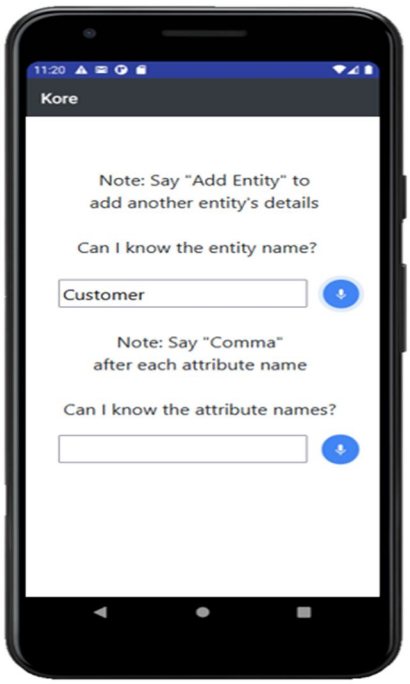


Figure 2. Database details capturing screen

Users can tap on the voice button displayed to input the options verbally or provide the data conventionally when the UI listens to the users' responses. Users need to enter the attributes separated by *"comma (,)"*. Once the users provide all the details, confirm them, and request to generate the web application code, they can observe a screen, as shown in Figure 3.

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 11 Issue VII Jul 2023- Available at www.ijraset.com

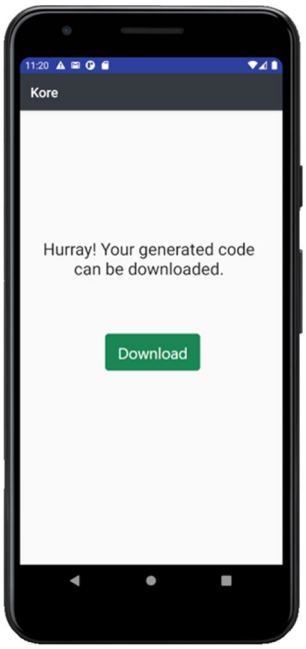


Figure 3. Successful code generation screen

Users can avoid the complications of JHipster CLI and proceed smoothly only by talking to the application. Our application, Kore, collects user-provided choices and handles the complexities of JHipster CLI for the users by automating the process. JHipster mandates users to run numerous commands, switch through several directories, create a JDL file separately, and run a few additional commands to generate custom database models. They must perform every action listed above by themselves, and various novice developers may find it complex to do. A scope of error can appear while following numerous steps to generate a web application. Addressing this issue, Kore proposes an effortless approach using which users can create a web application by skipping all the actions listed above. *"Kore,"* with an automated flow, performs all the activities for the users by collecting the required data from them. Using Kore, users need not perform multiple complicated actions to get a web application. In our application, the only two action items of users are to talk to our application to enter details about the web application and download their web application after Kore generates it. This approach diminishes the scope of error and even simplifies generating code for a web application. A widespread audience can use our application effortlessly to create a web application within minutes. Our application also enables support to visually impaired people and supports them in their web development journey.

V. CONCLUSION

Advances in technology nowadays have surged unimaginably. People look for options that ease their work and help them accomplish with minimal effort. Anything that assists people in successfully finishing their work is in high demand today. As discussed, generating a functional web application requires strenuous efforts but can be built by some predefined processes. Low-code generation tools come into play here, which assist people in creating fully functional web application code with minimal user interference. These tools take up most of the laborious tasks and will execute them for users. Most of the activities that are to be done by the user to create a web application can be automated using such low-code generation tools. JHipster, one of the code-generator tools, helps to generate backend code by letting the user interact with its CLI. However, some naive developers may find it difficult to interact with JHipster and may not cope with the complications that it brings to the table. Also, it takes a lot of time and effort to comprehend and get used to JHipster to put it to use. Users might also need to get familiarized with the JHipster CLI commands to execute them and generating a web application through JHipster would not be possible without getting acquainted with the commands. Diverse situations necessitate distinct CLI commands, and JHipster often requires user interaction to create a web application.

To address these issues and attempt to solve them, we came up with our Android mobile application, *"Kore"*. Kore eases the code generation process by automating it and lessening the necessity of user interaction throughout the process. Users need not directly interact with complex and sophisticated high-level software, JHipster, to get their web application.

Instead, they can use our application Kore, which simplifies the whole process by serving as an intermediary platform to interact with JHipster in a much simpler way. It handles all the complicated steps involved while interacting with JHipster. Users will notice the ease of creating a web application using Kore. Kore acts as a single point of contact to generate a web application for the users since they need not look out for additional resources to assist them as it comprises all the necessary features a user may utilize to feel a sense of ease.

VI. FUTURE WORK

Kore can develop a basic web application by gathering complete data about the technologies that the users choose to employ. It assists users in generating web applications without letting them communicate with complex high-level software JHipster. However, softwares in today's modern world are always open to any possible refinements. We thought of a few enhancements that devise the working of Kore and are in our pipeline. We can consider implementing the below features to enhance the functionalities of Kore.

1) Kore, at present, generates code for a web application by gathering users' preferences about technologies to use. We acquire this information by asking users a series of questions, one after the other. We want to leverage a single vocal input to create a web application. The single voice input must contain details about all the technical information we want to capture.

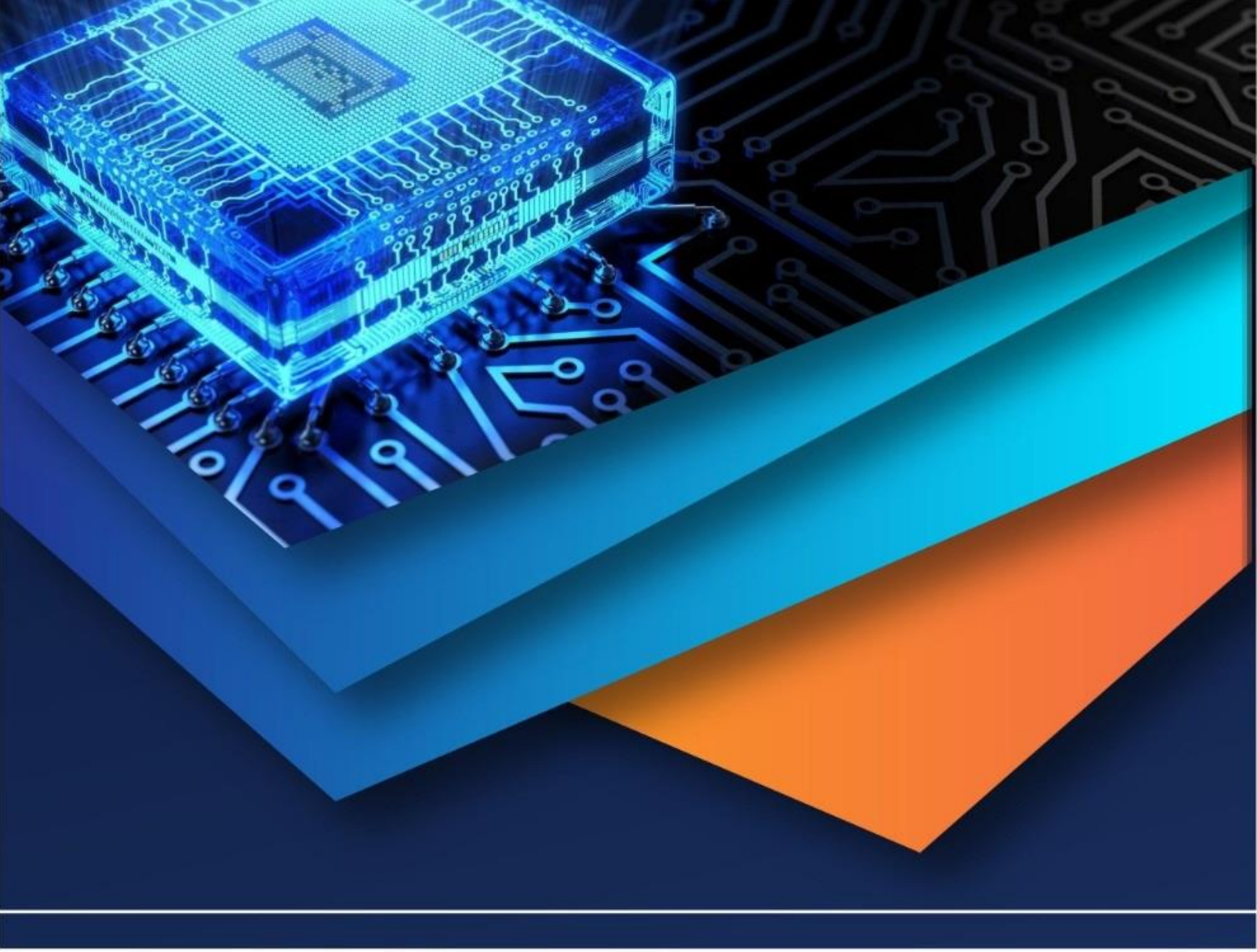
2) We let users input a JDL file or talk to our UI about each entity and its attributes to create custom database models. In future, we are planning to generate custom user-requested models from an ER diagram.

3) Users are now allowed to create only individual projects. We plan to bring a new functionality where multiple users can come together and collaborate on a single web application.

4) We intend to introduce an all-new section called *"Collaborations,"* where users may find projects that they collaborate on with others.

REFERENCES

[1] O. Al-Debagy and P. Martinek, ‘‘A comparative review of microservices and monolithic architectures,’’ in Proc. IEEE 18th Int. Symp. Comput. Intell. Informat. (CINTI), Nov. 2018, pp. 149–154.
[2] Baeldung, “JHipster with a Microservice Architecture,” Baeldung, 08-Jun-2022. [Online]. Available: http://www.baeldung.com/jhipster-microservices. [Accessed: 21-Feb-2023]
[3] J. Lewis and M. Fowler, “Microservices,” martinfowler.com, Mar. 25, 2014. [Online] https://martinfowler.com/articles/microservices.html. [Accessed: 17-Feb-2023]
[4] P. Sun, X. Yang, X. Zhao, and Z. Wang, “An Overview of Named Entity Recognition,” 2018 International Conference on Asian Language Processing (IALP), Nov. 2018, doi: 10.1109/ialp.2018.8629225.
[5] M. Villamizar et al., “Cost comparison of running web applications in the cloud using monolithic, microservice, and AWS Lambda architectures,” Service-oriented Computing and Applications, vol. 11, no. 2, pp. 233–247, Jun. 2017, doi: 10.1007/s11761-017-0208-y.
[6] N. Dragoni et al., “Microservices: Yesterday, Today, and Tomorrow,” in Springer eBooks, 2017, pp. 195–216. doi: 10.1007/978-3-319-67425-4_12.
[7] N. Sethi, A. Kumar, and R. Swami, Automated web development. 2019. doi: 10.1145/3339311.3339356.
[8] R. Chen, S. Li, and Z. Li, From Monolith to Microservices: A Dataflow-Driven Approach. 2017. doi: 10.1109/apsec.2017.53.
[9] Q. Guo, S. Wang, and F. Wan, “Research on Named Entity Recognition for Information Extraction,” 2020 2nd International Conference on Artificial Intelligence and Advanced Manufacture (AIAM), Oct. 2020, doi: 10.1109/aiam50918.2020.00030.
[10] A. Goyal, V. Gupta, and M. Kumar, “Recent Named Entity Recognition and Classification techniques: A systematic review,” Computer Science Review, vol. 29, pp. 21–43, Aug. 2018, doi: 10.1016/j.cosrev.2018.06.001.
[11] A. Halin, A. Nuttinck, M. Acher, X. Devroey, G. Perrouin, and P. Heymans, Yo variability! JHipster. 2017. doi: 10.1145/3023956.3023963.
[12] Z. Zhang, T. C. Chen, K. Vigneswaren, F. Hussain, S. Sharieh, and A. Ferworn, Automated Generation and Dynamic Rendering of Web-based Data Collection Systems. 2019. doi: 10.1109/iemcon.2019.8936143.
[13] D. A. Turner, M. Park, J.-H. Kim, and J. Chae, An AutomPated Test Code Generation Method for Web Applications using Activity Oriented Approach. 2008. doi: 10.1109/ase.2008.61.
[14] K. Shinde and Y. Sun, Template-Based Code Generation Framework for Data-Driven Software Development. 2016. doi: 10.1109/acit-csii-bcd.2016.023.
[15] L. Li, T. F. Bissyandé, J. Klein, and Y. L. Traon, An Investigation into the Use of Common Libraries in Android Apps. 2016. doi: 10.1109/saner.2016.52.
[16] Jhipster Team, “JHipster - Full Stack Platform for the Modern Developer!,” JHipster. https://jhipster.github.io/ [Accessed: Dec-22]



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY