IJEEC

International Journal of Electrical Engineering and Computing
Vol. 9, No. 1 (2025)

Original research paper
UDC 004.822:004.42`234-055.25
DOI 10.7251/IJEEC2501001V

CodeCrafter – Efficient Code Generator for Modern Single-page Web Applications

Danijela Vukosav[1], Danijela Banjac[1], Miloš Ljubojević[1], Mihajlo Savić[1]

[1] University of Banja Luka, Faculty of Electrical Engineering, Banja Luka, Republic of Srpska, Bosnia and Herzegovina

E-mail address: danijela99vukosav@gmail.com, danijela.banjac@etf.unibl.org, milos.ljubojevic@etf.unibl.org, mihajlo.savic@etf.unibl.org

Abstract—**Web applications and REST API based applications can be seen as the most common type of application today and their number is constantly increasing. Due to the pressure to produce more code and be faster at producing it, developers have been using various programming languages, libraries and frameworks suitable for such applications. Two of the most popular technologies are Spring on the back-end, and React on the front-end. While each technology provides satisfactory functionality, the amount of code that needs to be produced is non-trivial, especially in cases where the application has to provide even basic security and auditing functionalities. One solution for this issue is the use of code generator tools that produce both back- and front-end code from a model. While a wide variety of generators exist, none of the analyzed ones fulfill all the requirements of a modern web application. In this paper, we present CodeCrafter, a web-based code generation application that produces Spring and React code based on relational data models in DDL or JSON format, and provides developers with a simple and efficient tool to generate fully functional foundational code. We compare it to existing code generation tools and measure its performance as a function of the number of tables in the database. We show that CodeCrafter produces code in a very short amount of time even for very complex databases, while providing features and functionalities not present in other analyzed tools. We also give a short overview of possible use in conjunction with LLM based coding tools.**

Keywords-component; code generation; single-page web application; REST API; relational databases

I. INTRODUCTION

Efficient planning, design, and implementation are crucial in software system development, yet these processes can often be time-consuming and error-prone. A significant portion of development involves standardizing folder and file structures, planning database schemas, and implementing authentication, authorization, and CRUD (Create, Retrieve, Update, Delete) operations. These tasks frequently follow identical patterns across various projects, regardless of their specific domains [1]. Thus, automating template-based aspects of development is a logical strategy to reduce time consumption and enhance productivity.

Model-Driven Development (MDD) is a concept closely associated with software development automation [2]. MDD emphasizes the creation of abstract models that depict the system's structure and behavior, rather than direct coding. These models often utilize standardized languages, such as Unified Modeling Language (UML) or Domain-Specific Languages (DSL). By transitioning from these models to code through automated generation, development can be significantly accelerated while minimizing errors.

While adopting MDD yields numerous benefits, including reduced manual effort and an expedited development timeline, it can also significantly alter the development process [3]. The use of models fosters standardization and component reusability while decreasing the likelihood of implementation errors. Changes can be directly made to models, minimizing the need for manual code alterations. Nevertheless, the MDD approach presents challenges, including a steep learning curve associated with mastering modeling techniques and tools. The quality of the generated code can vary, potentially affecting efficiency and readability. Furthermore, technological advancements may necessitate updates to dependent models and tooling.

Today, we are witnessing the ever-increasing use of Large Language Models (LLMs) [4] in software development, especially as a form of code completion or code generation tools [5]. As with any technology, there are tradeoffs, both positive and negative, and the modern developer is in an unenviable situation to choose a position on a spectrum between conservative, robust, but complex and time consuming approach, and carefree, but hallucination prone heavy use of LLMs. Regardless of one's opinion on such LLM aided tools, they are very present in the field and are heavily promoted and integrated into various popular development environments [6].

For this study, CodeCrafter, a data-model-driven code generator, has been developed. It converts models — specifically Data Definition Language (DDL) scripts or JavaScript Object Notation (JSON) specifications — into functional source code suitable for Spring [7] and React [8] applications. This generator produces a foundational system structure with default configurations and functionalities,

reducing development time while enabling the creation of stable, scalable systems with minimal error risk. The generated system serves not only as a standalone product but also as a robust foundation for more complex projects.

II. Related Work

A. Overview of Existing Solutions

The increasing complexity of modern software systems has spurred a growing need for automated code generation. This development aims to minimize manual work, eliminate repetitive tasks, and enhance efficiency, enabling development teams to focus on more complex problems. Several tools have emerged in this space, each offering unique functionalities:

- **Spring Roo** - This tool accelerates application development within the Spring ecosystem, enabling rapid generation of CRUD functionalities. It generates code based on command-line inputs, maintaining a separation between automatically generated and manually written code through the use of aspect files and annotations. The output includes Java files for entities, repositories, services, and controllers, along with configuration files and, optionally, a simple client application. [9]
- **OpenAPI Generator** - An open-source tool that utilizes OpenAPI specifications to generate both server-side and client-side code, along with database schemas and documentation. The generator supports multiple programming languages and frameworks, allowing users to define API operations and data schemas via JSON or YAML documents. Custom templates facilitate code generation based on user selection through CLI or graphical interfaces. [10]
- **Yeoman Generators** - Yeoman automates project generation through predefined templates known as generators. Users interact with the CLI to specify project details, such as name and preferred technologies. Generators leverage libraries like Inquirer.js to prompt for user input, tailoring the output files based on the provided details, resulting in well-structured project setups. [11]
- **CodeSmith Generator** - Utilizing predefined templates and user inputs, CodeSmith automates the generation of various code types. Templates can be defined in languages like T4 or Razor, and user input replaces placeholders within these templates. The output includes SQL scripts, backend and frontend code, as well as configuration and documentation files. [12]
- **JHipster** - An open-source generator for modern web applications and microservices, JHipster supports both server-side (Spring Boot) and client-side (Angular, React, and Vue) development. Users are prompted for application configuration, including authentication type and database selection. It also allows for code generation based on JHipster Domain Language (JDL), facilitating entity modeling through predefined models. [13]

These tools collectively contribute to the automation of code generation processes, enhancing development efficiency and promoting higher-quality software output.

B. Comparison with CodeCrafter

The primary objective of various code generators, including CodeCrafter, is to automate the code generation process to reduce development time and enhance efficiency. Many functionalities exhibit similarities across these solutions, particularly in generating server-side and client-side implementations that facilitate complete CRUD operations, along with essential features such as authentication and authorization.

However, distinct differences exist among these systems. Some solutions rely on user input or predefined models for code generation, whereas CodeCrafter utilizes DDL scripts and JSON files — formats widely recognized in software development. This approach simplifies the generation process by eliminating the need for extensive data entry or the development of specific models. Moreover, CodeCrafter is designed with a more intuitive interface compared to the command-line and graphical interfaces of other systems, thereby enhancing user accessibility.

A notable advantage of CodeCrafter is its rapid code generation capabilities, allowing for the creation of new systems in mere milliseconds. Additionally, it offers unique functionalities not commonly supported by competing solutions. These include the ability to audit selected tables through the automatic addition of timestamps and user identifiers (createdAt, updatedAt, createdBy, and updatedBy), configurable authorization settings for table columns, visibility options for table views and single-object views, and the precise definition of columns that facilitate searching and sorting.

TABLE I. Comparison of Available Solutions

Feature	Spring Roo	Code Smith	OpenAPI Generator	JHipster	Code Crafter
Generate backend	Y	Y	Y	Y	**Y**
Generate frontend	N	N	N	Y	**Y**
Security and authen-tication	Y	N	N	Y	**Y**
User authori-zation	N	N	N	Y/N	**Y**
Auto table auditing	N	N	N	N	**Y**
Security-critical columns	N	N	N	N	**Y**
Modular data filtering	N	N	N	Y/N	**Y**
Modular data sorting	N	N	N	N	**Y**
Use in business systems	Y/N	Y	Y	Y	**Y**
Extensi-bility	Y/N	Y	Y/N	Y	**Y**

a. Y/N denotes a feature under development or not fully supported

Conversely, it is important to acknowledge that the aforementioned solutions have been developed over many years by substantial teams of developers. As a result, these systems tend to be more complex and offer a broader range of capabilities. This complexity includes support for a wider array of technologies compared to CodeCrafter, which currently focuses on Spring for backend development and React for frontend development.

As can be seen from Table I, there is ample space in the market for a tool that fully provides features missing from other available systems, such as user authorization, automatic table auditing, proper handling of security sensitive columns, as well as the support for modular data filtering and sorting. It is also worth noting that one must take care of performance aspects of code generation as well as the possibility to adapt and extend the solution in use.

III. CODECRAFTER

A. Introduction to CodeCrafter

The CodeCrafter client web application allows users to upload files that are used as input for code generation. Currently, the system supports two types of input files: DDL (Data Definition Language) script and JSON file. Although the system is limited to DDL scripts for MySQL DBMS, using JSON files offers a technology-agnostic representation of the application that will be generated. Uploaded JSON file must conform to the predefined JSON schema that is used to describe and validate JSON data.

The uploaded file is sent to the CodeCrafter backend application through the RESTful API [14]. The backend application processes input files based on their type. In the case of the DDL script, the application parses the script to extract relevant data, such as entity names, attributes, and relationships and transforms this information into a corresponding JSON object. In case the input is a JSON file, it is expanded to include inferred relationships based on column configurations. After processing the input file the application generates a JSON object and returns it to the client web application.

The CodeCrafter web application processes the JSON response and allows the user to configure parameters for the application that will be generated. First, the user configures whether to generate a Spring Boot application, a React application, or both. Afterwards, the user configures parameters for each entity which includes auditing settings, column visibility, filtering, and sorting settings. Enabling auditing allows transparent tracking of who created or changed each entity and when the change happened.

The user can select which columns will be displayed in tabular views or detailed views within the React application. This feature is particularly beneficial for entities with numerous attributes, helping to simplify the interface and protect sensitive information, such as passwords. The user can specify filtering and sorting settings for each entity. Filtering configuration includes a selection of columns that will support filtering adjusted to the corresponding data type. For example, for numeric columns, filtering enables range selections, while for text columns, it enables partial or full-text searches. The user can choose one or multiple columns to enable sorting in ascending or descending order. In the case of sorting multiple columns, the priority can be defined.

The usage scenario of the system is shown in Fig. 1.

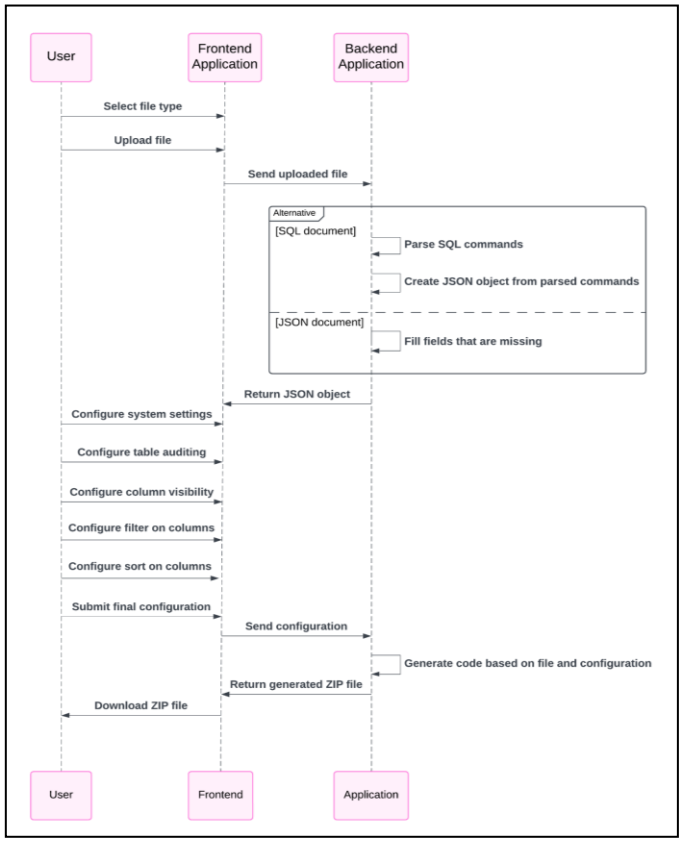

Figure 1. Usage of CodeCrafter System

B. Process of Code Generation

Upon selecting the model and its specifications, users proceed to the code generation phase. CodeCrafter utilizes pre-prepared templates, which are source code files derived from a complete application, tailored to a comprehensive database schema that encompasses various data types, table relationships, and structural aspects. Within these templates, code segments dependent on the specific schema are represented as placeholders. During generation, CodeCrafter dynamically replaces these placeholders with values aligned with the user's requirements.

The code generation process involves several key steps:

1. Creating the Root Folder – a main folder is created to represent the future system's structure top organizational node.
2. Generating Subsystems – based on user selections:
   - A Spring folder is created for back-end software component generation.
   - A react-app folder is created for front-end single-page web application.
3. Copying Common Files – universal files, independent of the specific model and configuration, are copied into the designated folders.
4. Generating Files from Templates – CodeCrafter iterates a list of entities for which the code needs

to be generated and replaces placeholders with actual values, ensuring files are tailored to the specified system.

5. Downloading the Generated System – upon completion, users can download the application as a ZIP file, containing all necessary files for further development.

This automated approach significantly enhances software development efficiency, minimizing the need for manual coding and streamlining the application creation process.

C. Overview of Generated System

The generated application is a web-based platform enabling users to perform CRUD (Create, Read, Update, Delete) operations on defined entities. The frontend, developed using React, offers an intuitive interface for user authentication and data management, allowing users to enter new data or retrieve existing records. User requests are transmitted as REST API calls to the backend, which is built with the Spring framework and serves as the core component for processing data, executing business logic, and managing database interactions. The classic client-server architecture used by the generated application is illustrated in Fig. 2.

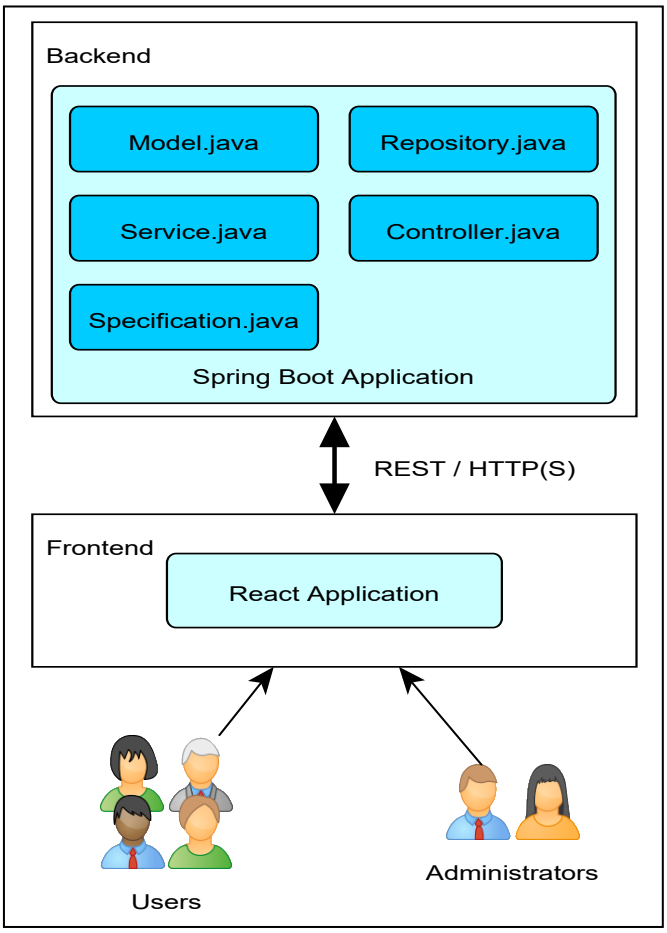

Figure 2. The architecture of application generated by CodeCrafter

The file structure of the generated back-end application in Spring Boot consists of the following:

- audit – the implementation of AuditorAware and the required base configuration.
- auth – with controllers for user management as well as basic authentication/authorization. This directory also contains the models required for the proper functioning of user accounts, tokens, and user roles and permissions. The code is organized in specification, repository, and service pattern in order to enable easier understanding and adjusting of the code by developers.
- security – containing basic functionality required for working with JWT [15] and related security functionalities.
- utils – containing various utility functionalities required for several application components on the back-end, such as working with filtering and string manipulation.
- a set of directories for tables present in the database, one directory per table. Similarly to the auth section, all functionality is organized in controller, specification, repository, and service organization pattern which takes into consideration foreign key relations between tables in the database and enables more efficient maintenance and further development of the generated code.

The main file structure of the front-end application consist of the following directories:

- api – directories for general services and hooks.
- authService – that contains the provider and service used for basic user authentication/authorization.
- generalComponents – that contains a collection of components used throughout the application, divided into the following sets: common, filtering, form, index, sidebar, and singlePage.
- hooks – that contains hooks for sorting, filtering, and pagination.
- loginComponents – used for login.
- pages – with one directory per processed table. Each directory, in turn, contains components, service and singlePage directories with files generated for each table. The main components cover table and table row, header, form and modal per entity.
- router – which sets up all the basic routes for generated entities, as well as generic routes for login and logout.
- styles – with all the CSS files organized for efficient use and potential adjustments.
- systemUsers – for system-wide user and role management.

As the generated front-end system uses TypeScript, all the required types are properly generated and can be used for code

completion and syntax checking by development environments.

The system distinguishes between two types of users, administrators and regular users, with each user type possessing a certain set of allowed functionalities:

- **Administrators** possess elevated privileges, including the ability to add new users (both regular and administrative) and assign specific access rights (view, add, update, delete) for individual tables.
- **Regular users** are granted a restricted set of CRUD operations as defined by an administrator.

Upon the initial startup of the application, a default administrator account is created in the database with predefined credentials (username and password). This account can subsequently be used for logging into the system, after which the administrator can add further regular users or other administrators.

The platform also facilitates efficient data management, featuring functionalities such as filtering and sorting for improved data visibility and usability. As all the generated code follows the best practices, it is possible to adapt all of the functionalities in an efficient and streamlined manner, with the additional possibility of adapting the source templates in order to provide the same baseline generated code for multiple projects, if such a need exists.

D. Generated back-end Application

The generated Spring Boot application provides comprehensive CRUD (Create, Read, Update, Delete) functionality for all defined entities, along with robust user authentication and authorization features. It integrates search and sorting capabilities with support for pagination.

Each entity is organized within a dedicated folder that includes all necessary components, such as entity classes, repositories, service classes, and controllers. User authentication is managed via the Spring Security framework, utilizing JWT tokens to secure access.

Configuration details are handled in the automatically generated application.properties file, where users specify essential database connection parameters. The application can be built, tested, and executed using Gradle, which creates all necessary files, including the build.gradle that outlines required dependencies.

This application is compatible with Java 17 and the Spring Boot 3.2.3 framework, ensuring adherence to modern development standards.

Packages generated for the application are also shown in Fig. 2 and include: Model.java, Repository.java, Service.java, Controller.java and Specification.java.

E. Generated Frontend Application

The generated frontend application is a React-based platform that offers users an intuitive interface for system interaction.

The generation process automatically creates all necessary files and configurations for the React application, including project structure, components, routes, styles, and configuration files. Users can customize styles and components to fit their preferences.

The frontend application, as of the writing of this paper, is built using React version 18.2.0 and Node.js version 20.13.1.

An example of web application look is shown in Fig. 3.

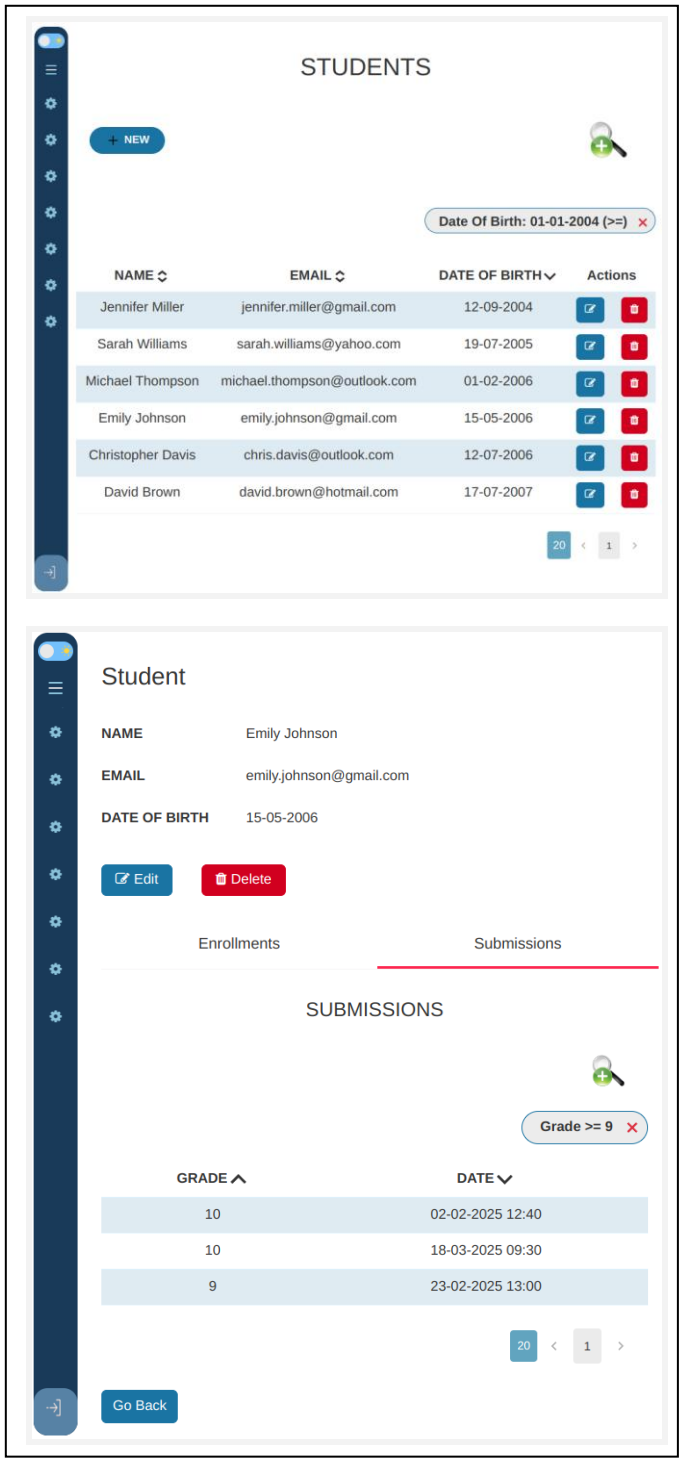

Figure 3. Examples of UI of a generated frontend application

Access is restricted to authenticated users, who must enter valid credentials on the login page. Upon the first launch, users are prompted to change the initial password assigned by the

administrator. Session management is handled using JWT tokens.

Once logged in, users are granted access only to the entities they are authorized to interact with, with any unauthorized actions—such as create, edit, or delete—hidden from view.

User authentication is required for application access and the access control in the system is based around privileges, with built in support for fine grained access rules for each generated entity type, as is shown in Fig. 4. Basic permissions that are supported by the generated application are: CREATE, READ, UPDATE, and DELETE (as in CRUD) and the names of privileges are created by concatenation of entity name, underscore and privilege name in uppercase.

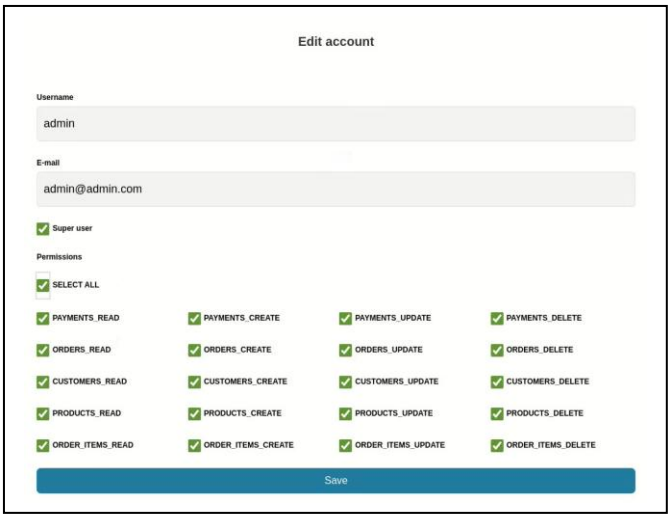

Figure 4. Managing user privileges UI example

IV. PERFORMANCE MEASUREMENTS

While the time required for code generation is rarely seen as one of the critical performance metrics, in order to fully integrate code generation tools in modern toolchains, especially ones using coding assistants, the ability to produce high quality code in short period of time is of significant value. Scenarios in which this metric is critical will be more thoroughly described in discussion section of the paper.

In order to measure the performance of CodeCrafter, a set of assignments was created, ranging from trivial database schema with a single table, over a series of progressively complex schemas with 5, 10, 15, 40, 50, 60, 96, and 121 schemas, and, finally, with a database schema consisting of 192 tables. This wide spectrum of complexities and numbers of tables covers a vast majority of system sizes. In order to extrapolate the performance in databases with an even larger number of tables, a measure of per-table code generation time should also be observed. In the experiment, the generation time of Spring and React applications was measured across varying numbers of entities, employing both serial and parallel generation approaches.

Serial Generation involved the sequential creation of all necessary files for each entity, where one application was fully generated before proceeding to the next.

Given that the same files were generated for each entity, parallelization of the parts of this process was feasible. Additionally, due to a well-defined API, both applications could be generated simultaneously.

Parallel Generation encompassed the simultaneous creation of both applications, along with the parallel generation of files for entities within each application.

Performance measurements for the generation of code for back-end, front-end, and combined front- and back-end applications, in both serial and parallel generation modes, are presented in Table II.

TABLE II. PERFORMANCE MEASUREMENTS OF CODECRAFTER

Number of tables	Spring		React		Total	
	Serial	Parallel	Serial	Parallel	Serial	Parallel
1	13.17	11.90	15.39	17.53	28.56	21.71
5	13.78	12.35	21.14	23.29	34.92	23.94
10	17.43	15.86	33.65	24.04	51.07	24.60
15	19.96	14.49	39.41	25.53	59.37	26.25
40	44.46	18.80	89.41	34.78	133.87	35.60
50	47.66	22.96	101.49	39.07	149.15	40.01
60	74.91	29.09	144.99	44.84	219.90	45.49
96	97.30	34.51	200.44	57.83	297.75	58.77
121	106.51	46.27	226.75	74.95	333.26	74.52
192	155.57	56.60	342.40	91.64	497.97	92.20

a. All times are in milliseconds

The measurement results, recorded in milliseconds, demonstrated the efficiency and high performance of the CodeCrafter generator. The accompanying diagram illustrates that complex structures, such as those with up to 200 entities, can be generated in less than 100 milliseconds.

From the results, it is obvious that there is an initial start-up time needed to initialize the system and generate the files for the first table, while additional tables are processed in, on average, 0.24 ms (with minimum being 0.11 ms, and maximum being 0.44 ms) for Spring back-end in parallel mode, and for React front-end, on average, 0.92 ms (with minimum being 0.44 ms, and maximum being 1.33 ms). This strongly suggests that the scaling is linear and that the system should work with satisfactory performance in even larger databases with hundreds or thousands of tables.

This feature of the system enables its use in very dynamic development environments as the added time delay by CodeCrafter is negligible in a very large spectrum of database sizes and should not interfere with the developer experience.

The results are also graphically presented in Fig. 5.

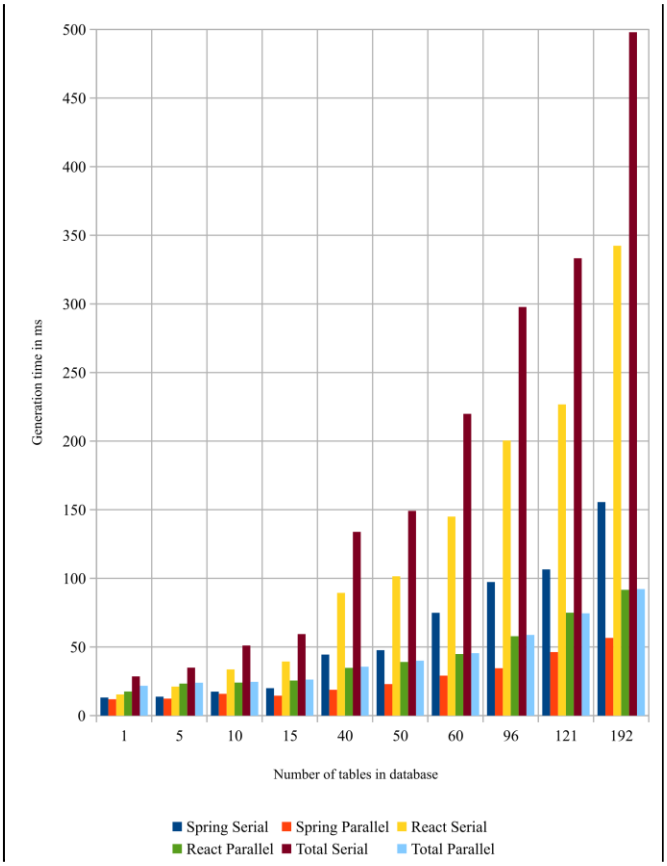

Figure 5. Generation times as function of number of tables

V. DISCUSSION

When it comes to feature parity, it is visible from Table I that none of the existing solutions on the market fulfills all the analyzed features. Critical features that are not supported, or are not fully supported include:

- User authorization – this feature is often "outsourced" to external tools or platforms and is ignored during the code generation. However, adding such critical security features after-the-fact is rarely the best option. Generated code also seldom supports fine grained user permission control and the ability to alter it at runtime.
- Automatic table auditing – the ability to provide data on who and when performed modifications on a record level can be very useful, or even required in certain applications. The ability to provide such functionality with negligible developer effort is an example of improving developer experience by code generation tools.
- Security-critical columns – most code generation tools make no distinction between various columns and will divulge potentially sensitive or secret data by default. It is up to the developer to find and alter the code responsible for hiding the values of security-critical columns. CodeCrafter, on the other hand, provides a clear interface that enables developers to avoid potential security pitfalls by hiding select columns.
- Modular data filtering and sorting – by offering developers the ability to define which columns should be filterable and which should be sortable by web application end-users, the development process of potentially complex user interfaces and interactions can be significantly simplified.

Key observations derived from the analysis of performance measurement results include:

- No significant performance difference was observed between serial and parallel generation for a small number of entities.
- As the number of entities increased, the advantages of parallel code generation became more pronounced.
- The total sequential generation time was found to equal the sum of individual application generation times.
- Conversely, the parallel generation time was determined by the longer of the two processes, which was the React application due to the greater number of files required compared to the Spring application.

These findings underscore the scalability and efficiency of the CodeCrafter generator, particularly when dealing with larger applications that deal with a significant number of tables in the database.

Additionally, a static code analysis of the generated code was performed using the Qodana tool. During this analysis, the code was examined to identify potential errors, security vulnerabilities, and other issues. The results confirmed that no errors or warnings were present, indicating that the generated code meets high standards of quality and security. This is a significant finding that demonstrates that the generated code can represent a solid and secure basis for further development even in very demanding and sensitive application areas.

A. Modern LLM-aided development environments

Since the appearance of widespread use of LLMs [4] in the past few years, there exists a clear trend to use LLMs, especially ones fine-tuned on code datasets, as a means to generate program code and integrate them closely with popular development environments, either as a form of a more powerful code completion tools or as a replacement for code generators in general [6]. While there are promising results, there are also some mixed ones as well, especially regarding security aspects of analyzed or generated code [16].

An additional problem is presented by the fact that the whole LLM and LLM code completion field is rapidly developing and any studies deemed current and representative tend to be outdated in very short time, especially as any tests proposed or failings identified by authors are quickly ingested in the next round of model training.

Another approach that has shown promise and is widely used in various applications of LLMs is the use of Retrieval Augmented Generation (RAG) which is shown to increase accuracy and reduce hallucinations in output [17]. RAG functions by providing the LLM with additional useful data in the query, but is heavily dependent on the provided data being

valid [18]. As CodeCrafter can generate valid code based on simple JSON model that can be generated by LLM, it opens an avenue for extending the functionality of LLM by it using CodeCrafter as an external tool. More recently, there are have been industry-wide efforts to standardize similar approaches by utilizing Model Context Protocol (MCP) [19]. In this scenario, CodeCrafter would be used as a tool, but could also use sampling to query the LLM if the need exists. The high performance of CodeCrafter is very important in this scenario as any prolonged delays during coding and code completion severely decrease the developer experience.

VI. CONCLUSIONS

The presented solution, CodeCrafter, represents an advancement in the field of automated code generation, streamlining the development process while enhancing efficiency. By utilizing widely accepted input formats such as DDL scripts and JSON files, CodeCrafter simplifies the generation process, allowing users to bypass lengthy code entry and basic model development.

The intuitive interface further distinguishes CodeCrafter from traditional command-line and GUI tools, making it accessible to a broader range of developers. The rapid code generation capabilities enable the creation of comprehensive applications—including those with complex structures—within hundred milliseconds, thus significantly reducing development time. This feature also makes future use in LLM aided code completion systems possible without degrading the developer experience.

Unique functionalities, such as table auditing and configurable features for column visibility and authorization, position CodeCrafter as a versatile tool that meets contemporary software development needs. Although it currently focuses on Spring for backend and React for frontend development, it effectively addresses core requirements, including CRUD operations, authentication, and authorization.

While established competitors offer broader capabilities and support for a wider array of technologies, CodeCrafter's focused approach and distinctive features provide a compelling solution for developers seeking efficiency and simplicity in code generation. As software development continues to evolve, tools like CodeCrafter are essential for optimizing workflow and allowing teams to concentrate on solving complex challenges, thereby contributing to higher-quality software outputs.

REFERENCES

[1] M. Fowler, Patterns of enterprise application architecture. Addison-Wesley, 2012.

[2] T. Stahl, M. Völter, and K. Czarnecki, Model-driven software development: technology, engineering, management. John Wiley & Sons, Inc., 2006.

[3] B. Hailpern and P. Tarr, 'Model-driven development: The good, the bad, and the ugly', IBM systems journal, vol. 45, no. 3, pp. 451–461, 2006.

[4] Y. Chang et al., 'A survey on evaluation of large language models', ACM transactions on intelligent systems and technology, vol. 15, no. 3, pp. 1–45, 2024.

[5] J. T. Liang, C. Yang, and B. A. Myers, 'A large-scale survey on the usability of ai programming assistants: Successes and challenges', in Proceedings of the 46th IEEE/ACM international conference on software engineering, 2024, pp. 1–13.

[6] D. Cambaz and X. Zhang, 'Use of AI-driven code generation models in teaching and learning programming: a systematic literature review', in Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1, 2024, pp. 172–178.

[7] "Spring Boot," Spring Boot. Accessed: May 1, 2025. [Online]. Available: https://spring.io/projects/spring-boot

[8] "React." Accessed: May 1, 2025. [Online]. Available: https://react.dev/

[9] "Spring Roo - Reference Documentation." Accessed: May 1, 2025. [Online]. Available: https://docs.spring.io/spring-roo/docs/current/reference/html/

[10] OpenAPITools/openapi-generator. (May 1, 2025). Java. OpenAPI Tools. Accessed: May 1, 2025. [Online]. Available: https://github.com/OpenAPITools/openapi-generator

[11] "The web's scaffolding tool for modern webapps | Yeoman." Accessed: May 1, 2025. [Online]. Available: https://yeoman.io/

[12] "CodeSmith Tools." Accessed: May 1, 2025. [Online]. Available: https://www.codesmithtools.com/product/generator

[13] Jh. Team, "JHipster - Full Stack Platform for the Modern Developer! | JHipster." Accessed: May 1, 2025. [Online]. Available: https://www.jhipster.tech/

[14] L. Richardson and S. Ruby, RESTful web services. O'Reilly Media, Inc., 2008.

[15] M. B. Jones, J. Bradley, and N. Sakimura, "JSON Web Token (JWT)," Internet Engineering Task Force, Request for Comments RFC 7519, May 2015. Accessed: May 1, 2025. [Online]. Available: https://datatracker.ietf.org/doc/rfc7519/

[16] C. Negri-Ribalta, R. Geraud-Stewart, A. Sergeeva, and G. Lenzini, 'A systematic literature review on the impact of AI models on the security of code generation', Frontiers in Big Data, vol. 7, p. 1386720, 2024.

[17] P. Lewis et al., 'Retrieval-augmented generation for knowledge-intensive nlp tasks', Advances in neural information processing systems, vol. 33, pp. 9459–9474, 2020.

[18] H. Tan et al., 'Prompt-based code completion via multi-retrieval augmented generation', ACM Transactions on Software Engineering and Methodology, 2024.

[19] "Specification," Model Context Protocol. Accessed: May 1, 2025. [Online]. Available: https://modelcontextprotocol.io/specification/2025-03-26

Danijela Vukosav received B.Sc. degree in electrical engineering from the Faculty of Electrical Engineering, University of Banja Luka, Bosnia and Herzegovina. She is currently studying for M.Sc. degree at the same institution. She has over four years of experience in full-stack development for web and mobile applications using various back- and front-end technologies. She has worked on a wide range of projects, including scalable AI-powered platforms, custom CMS solutions, and real-time IoT applications. I'm passionate about building high-performance, user-focused systems, with a strong interest in AI integration and optimizing complex systems for better user experience.

Danijela Banjac and M.Sc. degrees in electrical engineering from the Faculty of Electrical Engineering, University of Banja Luka, Bosnia and Herzegovina. She is a senior teaching assistant and PhD student at the Faculty of Electrical Engineering, University of Banja Luka (Bosnia and Herzegovina). She is a member of M-lab Research Group. Her research interests include model-driven software development, business process modelling, object-oriented information systems, and UML. She has published several research papers and articles.

Miloš Ljubojević received the B.Sc. and M.Sc. degrees in electrical engineering from the Faculty of Electrical Engineering, University of Banja Luka, Bosnia and Herzegovina, and the Ph.D. degree in information technology from the Faculty of Organizational Sciences, University of Belgrade, Serbia. He is an associate professor at the Faculty of Electrical Engineering, University of Banja Luka. His teaching topics are in the fields of computer networks and computer science and informatics.

Mihajlo Savić received the Diploma Engineer, M.Sc. and Ph.D. degrees from the Faculty of Electrical Engineering, University of Banja Luka, Banja Luka. He is currently employed as a Assistant professor for the Faculty and is currently involved in the following areas at the Department of Computer Science and Information Technology: operating systems, information systems, computer networks, and parallel and distributed computing. He has been involved in SEE-GRID, SEE-GRID-2, SEE-GRID-SCI, SEEFIRE, SEEREN2, HP-SEE, EGI-InSPIRE, VI-SEEM, NI4OS-Europe, and SARNET projects.