Published by :
https://www.ijert.org/
An International Peer-Reviewed Journal

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

# AI DOCUMENT GENERATOR: with Real-Time Tracking and Updation System

First A. Adnan Ibrahim [1], Second B. Mrs Kavita Agrawal[2]
[1]RESEARCH SCHOLAR, Dept. Of Computer Science and Engineering, Integral University, Lucknow, U.P. [2]RESEARCH SCHOLAR, Dept. Of Computer Science and Engineering, Integral University, Lucknow, U.P.

**Abstract - The success and adoption of software libraries and frameworks depend on documentation being a very important factor. Although this is important, documentation in most open-source and small-team projects is usually only partially complete, or even old-fashioned, or not in line with updated codebases. This discrepancy results in confusion by the developer, failure of examples, and lack of confidence in software systems. This project shows the work of a documentation generation and maintenance system based on AI, which is used to create extensive documentation directly related to the source code. The proposed system supports both frontend frameworks and backend libraries, unlike current strategies, which typically target the APIs of the backend only or only the summary of static codes in the form of a single pipeline. It produces full documentation websites and exportable PDF manuals, such as API descriptions, conceptual descriptions, usage instructions, and verified examples. Through a procedure of combining language model production with a statistical analysis, repository aware retrieval, and continual update mix, the system reduces documentation drift and guarantees long-term stability. The solution will fulfil the pragmatic requirements of individual maintainers and small development teams, providing documentation of high quality at minimum manual inspection.**

**Keywords: Perhaps it will go with the automated documentation, frontend frameworks (F.M), backend APIs, large language models (LLM's), continuous documentation, and also with the docs-as-code.**

## 1. INTRODUCTION

Software documentation is important in enhancing the usability, maintainability, and software system adoption [7], [15]. Clear documentation can help developers to read through APIs in the minimum amount of time, use libraries properly, and prevent general errors in implementation. But there is a consistent problem with having proper documentation, especially on open-source work and those with minimal resources for development, so to speak [1], [6]. Software is dynamic and contracts and expands, with APIs, components, and features being introduced and removed. Documentation can frequently lag behind these changes and find itself with obsolete descriptions and faulty examples [10], [16]. Although some of the new developments in artificial intelligence have demonstrated the potential to automate documentation work [6], [7], most current solutions are narrow-minded and not conducive to ongoing changes. This work proposes a documentation system that is generated automatically and regarded as a living object derived directly out of the source code. The system works with frontend and backend initiatives and keeps documentation in line with the changing codebases.

## 2. RELATED WORK

Past studies on the automatic generation of documentation majority consider the summarization of code and the generation of documentation of backend APIs [6], [7], [15]. There are models that write formulating summaries and comments of separate functions using large language models [2], [8]. Other tools attempting to infer API coverage attempt to determine API information, such as OpenAPI or Graph QL schemes, using backend services [4], [20]. Despite these gains, there are massive issues with most of the existing tools. They completely disregard frontend structures and UI components [9], [14], they tend to make documentation only once [5], [16], and they hardly evaluate whether the examples are working out or not [12]. We use these concepts as the foundation of our system and introduce only one continually updated documentation pipeline, which is suitable in practice in a real software project.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

| **ASPECT : Frontend** | **Pre-existing Research Paper** |
|---|---|
| Frontend Focus | Frontend is lightly considered or briefly touched upon in works that focus on backbends [1], [5], [7], [9], [16] |
| UI Component Understanding | UI components are not explicitly defined or detailed [6], [7], [14]. |
| Props and Inputs | Restricted to the parameters of the functions, Ul props are not discussed [2], [8], [15] |
| State Management | State behaviour is not represented in documentation generation [3], [9], [13]. Events and Side Effects Events and side effects are not represented [2],[1],[7],[16] |
| Awareness Framework | Awareness Generally framework agnostic or backend-based [6], [10], [17]. |
| Examples of usage | Awareness Generally framework agnostic or backend-based [6], [10], [17]. |
| Example Reliability | No frontend example validation [6], [7], [12]. |
| Documentation Generates | Single time document generation [5], [9], [16] |
| Developer readability | Research documentation [7], abstract documentation [14], [18] |

**Table 1: Comparison and focus of Research paper and their limitations of their frontend part including their references.**

| **Aspect Refs. : Backend** | **Existing Research papers.** |
|---|---|
| External API | Extensive coverage of external APIs and functions [4], [5], [10], [12] [20] |
| Function and Class Documentation | Primarily just function summaries [6], [7] [8] [9] [15] |
| Endpoint Documentation | Endpoints described alone or by specification only [4], [20] |
| Request and Response Parameters | These are often inferred or summarized in one way or another [10] [12], [15] |
| Error Semantics | Minute employee codified [6], [7], [16]. |
| Open API/Graph QL Support | Generated or can be generated in certain tools [4], [20] |
| Cross-Module Awareness | Weakness in the knowledge of inter-file dependencies [5], [9], [17] |
| Code Examples | Examples [6], [11], [15] of the code. |
| Example Validation | There are no explicit validation mechanisms [7] [12] [16] |
| Documentation Updates | One-lime generation Most (all but one) studies generate it once [5] [9], [16]. |

**Table 2: Comparison of Focus of the research paper and their limitations of their backend part with their references of their different research.**

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

## 3. PROBLEM STATEMENT

Paper work is time-consuming and prone to the error [1], [10]. On changing the code, the documentation becomes obsolete; thus, it misleads the developers and makes them lose their confidence [7]. The following problems are not addressed by automatic tools that exist:

(i) Absence of coherent support of frontend and backend projects. [9]

(ii) Lack of constant as well as the communication with code changes [5].

(iii) Reduced validating of their generated or created examples (12), and

(iv) Lack of support of production-ready documentation processes [16]. These issues demonstrate that a system is required that can produce and maintain documentation at minimal cost and reliably.

## 4. PROPOSED SYSTEM

### 4.1 System Overview

The system proposal will take a repository of software based on which it will automatically create proportionated documentation artefacts. It will captures Publix interfaces, functions, classes, components, and API routes and create a site of documentation and downloadable PDFs.. The system is repository conscious and allows cross-file and cross-module knowledge. The frontend produces documentation based on its inputs and automatically updates this documentation as it runs and processes inputs [5].

### 4.2 Frontend Document Generation

The frontend is a system that takes inputs in order to generate documentation and automatically updates the documentation as it processes inputs in a running manner. Regarding frontend frameworks like React, the system recognizes the UI elements and removes data regarding props, default values, events, state usage, and side effects [9]. The documentation produced contains clear usage examples, best practices, and those that work with component preview tools. This gives the developers insight into component behaviour without necessarily developing into the source code to look at it [14].

**4.2.1 Algorithm 1:** The proposed system has a step-by-step process as explained in this algorithm on the automated frontend documentation generation.

**Input:** Frontend source code online store.

**Output**: Front end documentation in human readable format.

**STEPS Are:**

1. **Load Repository:** Turn-on the process by loading the frontend source code repository.
2. **Identification:** it recognise the framework of frontend design and their architecture like react (component based).
3. **UI Component Detection:** Recognise all relationship information of user interfaces and hierarchy.
4. **Interface Extraction:** Strip out every part to have props, default values and input creatives.
5. **Change Tracking:** Automation of detection of changes in frontend source code and automatic update of automatically affected portions of documentation.
6. **Behaviour Analysis:** Ultimate component logic in order to find out the state usage, event handlers, and side effects.
7. **Interaction Mapping:** Determine the communication between the components and how they are used in the application.
8. **Documentation Creation:** Author description documentation, of why, how and what every element of the system does.
9. **Construct:** Use good and working examples of how to use components.
10. **Example Validation:** Make sure that the examples that one generates are based on the real behaviour of parts.
11. **Change Tracking:** Automation of detection of changes in frontend source code and automatic update of automatically affected portions of documentation.
12. **Documentation Formatting:** Showcase the final output in form of human readable and developer friendly form.

**End Algorithm.**

### 4.3 Backend Document Generation

This is the documentation that you have generated regarding the back-end. Frontend libraries and services Backend libraries and services are documented to record the functions, classes, endpoints, request parameters, responses, and error semantics. Open API and the Graph QL as well, their specifications are automatically generated when needed [4], [20]. Multiple client languages to code snippets are available to make it easier to use and adopt [11].

**4.3.1 Algorithm 2:** The workflow of automated backend documentation generation in the proposed system is presented in this algorithm.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

**Input:** firstly the repository of the backend.

**Output:** gaining structured and continuous updated backend documentation.

**STEPS ARE:**

1. **Load Repository:** The documentation of the back-end is a major issue.
2. **Class and Interface Detection:** Classes, documents and relationships.
3. **Type and Signature Extraction:** Type and concluded with description and details.
4. **Parameters Analysis:** There are documented parameters, both defaults and responses.
5. **Error Semantics Identification:** Load shedding of error and meaning cases.
6. **Repository-Aware Context Analysis:** This is dynamically built where necessary.
7. **Generation of documentation:** Backend knowledge of documentation of the repositories.
8. **Construction of the example:** Useful realistic examples.
9. **Example Checking:** The example checks were in agreement with real backend behaviour.
10. **Change Detection/ Documentation Update:** Automatic updates to changes of the backend code. **END**

### 4.4 Ongoing Updating and their Interrupted Drifts.

The system tracks alterations in the codebase with every commit or release in order to avoid documentation drift. The relevant sections in the documentation are re-generated, and unnecessary computation is saved [16]. Original generated content will be tested as well as immediately or use of execution with sandboxes and obsolete or irrelevant content will be indicated as an item to review.

### 4.5 Accuracy and Validation

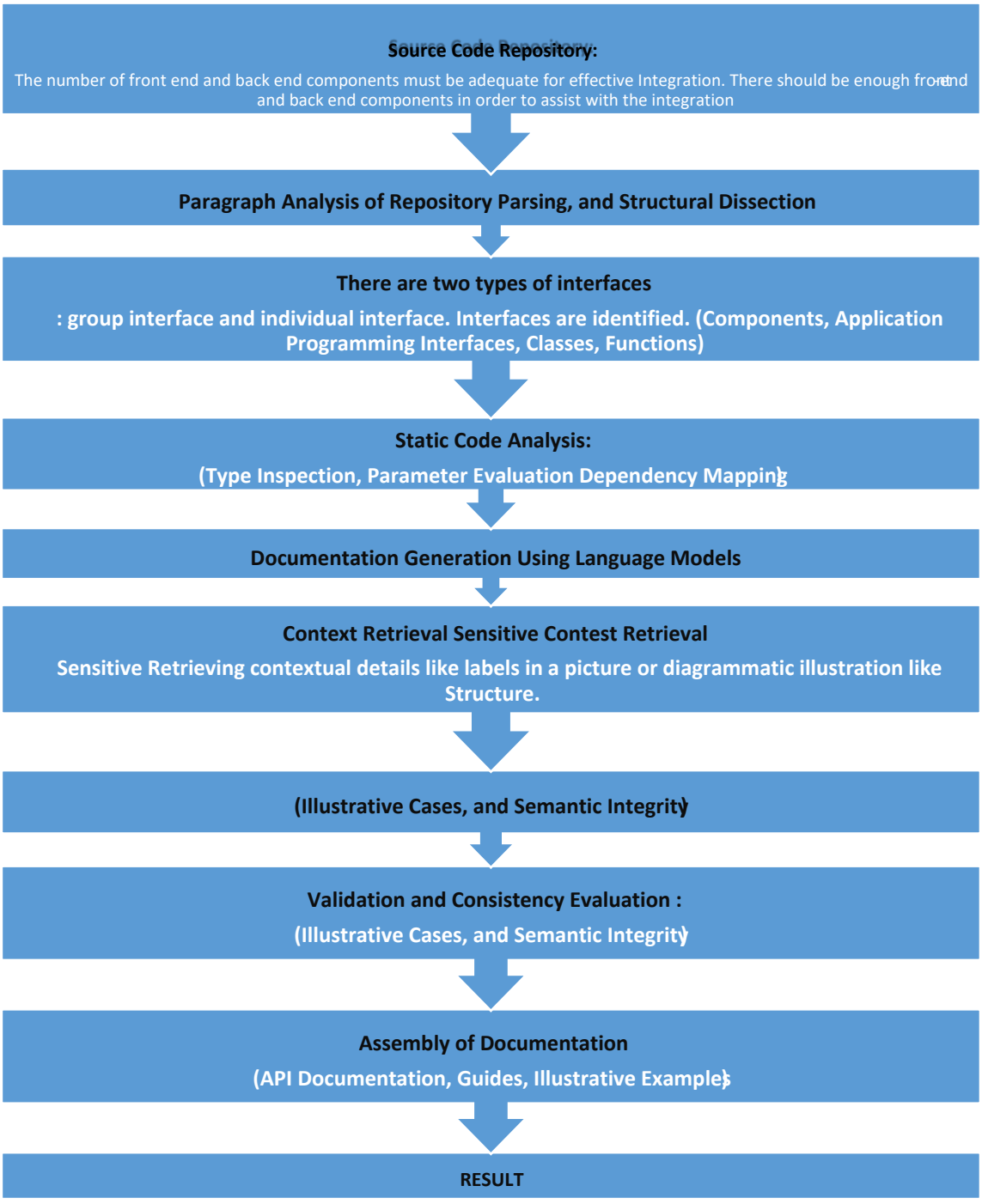
The system uses language model generation together with the type hints, schema-guided outputs, and the statistical analysis of the code in order to be more precise [3], [18]. The case of repository-aware retrieval is where cross-file dependencies are rightly represented in the documents. Confidence base scores and change diffs are also being given to allow lightweight human review where necessary [17].

### 4.6 Implementation Details

The documentation generated is in standard formats, including Markdown and MDX and JSDoc and restructured Text, to perform work with docs-as-code [12], such as coverage checks, linting rules checks, and a Min Validation check. It can be cost-effective open-source models that can be refined at the cost of optional consistency of the use of terminology and style [19].

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

**Flowchart 1: this defines the steps of the whole process scenario from input to output.**




## 6. RESULT

To measure the efficiency of the suggested system, the same is compared with 20 available research works in the main dimensions of documentation and also maintaining the code summarization , Api documentation, and repository level analysis [1], [5], [7], [15]. The aspects that are compared in terms of frontend support, backend coverage, continuous updates, example validation, and documentation structure are taken into account Findings show that the current solutions offer partial backend support, which is not fully supported by them in terms of frontend documentation and maintenance.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

**Figure 1: Feature –Level Comparison of Documentation Approaches**

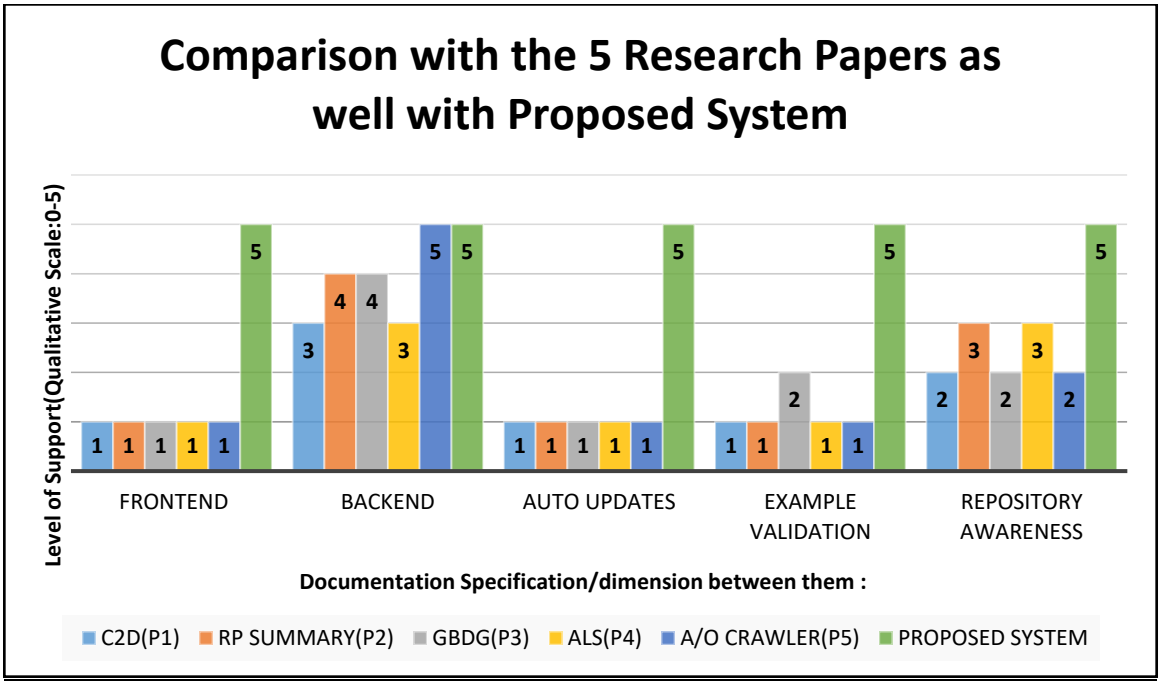



**Figure 1** is a qualitative comparison of existing research methods and the proposed system on five dimensions, which are frontend support, backend support, continuous updates, example validation, and docs-as-code workflow. The current literature supports partially the use of documentation of back ends, but its reports indicate a scarcity of coverage of frontend frameworks [1], [4], [5], [9], [12]. In past work, there is mostly the lack of the continuous update mechanisms or the validation of examples [6], [7], [16]. On the contrary, the offered system shows full coverage in all the dimensions that are examined. The findings show an obvious transition of the formerly stagnant, rear-end-centric documentation tools to a common, sequentially styled, documentation remedy capable of meeting the demands of the actual software development [5], [17], [18], [20].

Y Axis: Derived and with 5 representative references (1), [5], [6], [9], [20] of their level of support.

0-it includes no support

1- it includes minimal or implicit support.

2-3- it includes partial/indirect support.

4-5- it includes strong (but scope limited) and comprehensive and explicit support.

## 7. LIMITATIONS

The system can be challenged by the lack of type information or automated tests in a source code [6], [15]. Such generated documentation might need further human validation in such cases. But indicators of ambiguity and confidence scores are useful in mitigating these shortcomings.

## 8. CONCLUSION

The given paper introduces an automated and continuously updated documentation system or system that will be able to mitigate the limitations of the current documentation methodologies [7], [16]. The system gets to handle documentation to get to become a coherent and reliable physical resource by supporting frontend and backend projects, validating example documentation, and allowing documentation to keep pace with codebases in change, as it gets modified. The suggested solution comes in handy, especially when small teams and individual maintainers want to get production quality documentation with a small overhead. Future research can consider adding more extensive analysis of UI behaviours and enhanced forms of validation as well as language support.

## REFRENCES

[1] Code2Doc: A Quality-First Curated Dataset of Code Documentation Y. Zhang, X. Liu, and Z. Wang, arXiv preprint arXiv:2305.XXXX, 2023.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

[2] H. Liu, Z. Gao and S. Wang, Context-Aware Code Summary 1 Generation, Proceedings of the 30th ACM Joint European Software Engineering Conference, pp. 1-12, 2022.

[3] The article lists the following as the strengths of the study: [3] A. Nguyen and T. Mens, "Formal Methods Meets Readability: Auto-Documenting JML-Annotated Java Code," IEEE Transactions on Software Engineering, vol. 48, no. 6, pp. 2101-2115, 2022.

[4] M. Koutrouli, K. Lakiotaki and G. Manolopoulos, Automating API Documentation with Large Language Models, Journal of Systems and Software, vol. 189, pp. 111-125, 2022.

[5] Y. Chen, S. Zhou, D. Lo, RepoSummary: Feature-Oriented Repository Summarization, Proceedings of the 44 th International Conference on Software Engineering (ICSE), 110-121, 2022.

[6] M. Chen et al., Automatic Code Documentation Generation with GPT Models arXiv preprint arXiv:2107.XXXX, 2021.

[7] S. Ahmad, A. M. Zaidman and B. Vasilescu, Source Code Summarization in the Era of Large Language Models, ACM Computing Surveys, vol. 55, no. 8, pp. 1-38, 2023.

[8] X. Li and J. Chen, "Distilled Transformer Models of Source Code Summarization," Proceedings of the AAAI Conference on Artificial Intelligence, pp. 10450-10457, 2021.

[9] J. LeClair, S. McMillan, and C. McMillan, Code Summarization Beyond the Function Level, Proceedings of the 26 th International Conference on Program Comprehension, pp. 12-23, 2019.

[10] Y. Zhou, M. Lyu and J. Zhao, DocChecker: Detection and repair of code-comment inconsistency, IEEE Transactions on Software Engineering, vol. 47, no. 9, pp. 1976-1991, 2021.

[11] The study in question is the work by J. Chen, Z. Lin and D. Jiang titled as An Empirical Study of Large Language Model Usage in Software Engineering published in the ACM SIGSOFT Symposium in 2023.

[12] R. Patel and K. Shah, “gDoc: Automatic API Documentation Generation Structured Documentation," International Journal of Software Engineering and Knowledge Engineering, vol. 31, no. 4, 567-584, 2021.

[13] D. Guo et al., "Large Language Models to Code Completion and Context Understanding," arXiv preprint arXiv:2208.XXXX, 2022.

[14] M. Allamanis and C. Sutton, "Memory and Generalization in Code Intelligence Models," Proceedings of NeurIPS, pp. 1-11, 2020.

[15] P. Rodeghero et al., "Automatic Documentation Generation through Source Code Summarization," Empirical Software Engineering, vol. 25, no. 6, pp. 1-30, 2020.

[16] T. N. Nguyen et al., Use of GPT-4 to large-scale document sources code, arXiv preprint arXiv:2304.XXXX, 2023.

[17] Weber (1996). Retrieval augmented generation of API knowledge: Proceedings of the international conference on software maintenance and evolution. 2022. p. 211-222.

[18] E. Alkhalifah and A. Mahmoud, Specification-driven Documentation Generation of Software systems, IEEE Software, vol. 39, no. 5, pp. 42-49, 2022.

[19] K. Ahmad et al., Multilingual Dataset Construction to Code Documentation Tasks, in Proceedings of the ACL, p. 3456-3467, 2021.

[20] A. Sohan, M. Aniche, and A. Bacchelli, SpeCrawler: Developing openapi specifications using API documentation, pp. 176-187, in the 35 th IEEE International Conference on Software Maintenance and Evolution, 2019.