IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940



International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 11, November 2026

DOI: 10.17148/IJARCCE.2026.1511195

AI-DRIVEN GUIDING SYSTEM

Parvati Kushthagi[1], Rakshita Patil[2], Uday Mirji[3], Kiran Pujar[4], V. B. Hunagund[5]

Department of Computer Science and Engineering, Basaveshwara Engineering College,

(affiliated to Visvesvaraya Technological University), Bagalkot, Karnataka, India[1,2,3,4]

Assistant Professor, Department of Computer Science and Engineering, Basaveshwara Engineering College,

(affiliated to Visvesvaraya Technological University), Bagalkot, Karnataka, India[5]

*Corresponding Author

Abstract: A software idea has to pass through several stages before it becomes a working application. The team must clarify what the application should do, plan its structure, design its data and interfaces, write the code, test the result and prepare supporting documents. When these tasks are carried out in unrelated tools, information can be repeated or become inconsistent. This paper presents the AI-Driven Guiding System, a web platform intended to organize these activities in one project workspace. A user begins with a description of the proposed application. The system then supports domain analysis, preparation of a Software Requirements Specification (SRS), architecture visualization, database and REST API planning, code generation, review, testing and export. The project documentation identifies React, TypeScript, FastAPI, PostgreSQL, SQLAlchemy, JWT, React Flow, Three.js, Monaco Editor and Docker as parts of the implementation. AI-assisted functions are used to help process requirements and code-related tasks; users can inspect and edit the resulting artifacts. The available project material does not provide independently verified performance measurements. Accordingly, this paper describes the system and its intended evaluation without claiming a measured improvement in productivity, accuracy or code quality.

Keywords: artificial intelligence, requirements engineering, software development, code generation, large language models, software testing, code review.

I. INTRODUCTION

A software application is rarely produced by coding alone. Before implementation begins, developers need to understand the problem, identify users and their needs, decide how the system will be organized, and determine what information it must store. The resulting design then guides implementation and testing. If one of these decisions changes, related documents and code may also need to change.

In student projects and small development teams, these activities are often spread across word-processing files, diagrams, database tools, code editors and testing utilities. Moving information between these tools can take time. It can also make it difficult to check whether a function in the finished application was included in the original requirements.

Research has explored the use of natural-language processing in requirements engineering [1], while code-focused language models have been studied for program understanding and generation [3], [4]. These methods can assist developers, but generated output still needs review. Prior work on code generation and software-task evaluation also shows the importance of checking whether generated solutions work, rather than relying only on the text they produce [5], [8].

The AI-Driven Guiding System is designed around this practical need. It provides a sequence of project activities in one workspace and carries information from one stage to the next. The platform is intended as an assistant for students and developers; responsibility for confirming requirements, reviewing code and approving a release remains with the user.

II. PROBLEM DEFINITION AND OBJECTIVES

A. Problem Definition

A project idea may be clear to its author but still lack the detail needed for implementation. Requirements, architecture diagrams, database structures, API definitions, code and tests may be created separately. As a result, the relationship between these artifacts can become unclear, especially when requirements change. The project addresses the need for a guided environment that helps users prepare and review these related outputs within a single workflow.

 This work is licensed under a Creative Commons Attribution 4.0 International License



International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 11, November 2026
DOI: 10.17148/IJARCCE.2026.1511195

B. Project Objectives

- Accept a project description written in everyday language and organize it as a software project.
- Identify likely domain concepts, user roles, functional requirements and constraints for user review.
- Prepare an editable SRS and support architecture, database and REST API planning.
- Generate related backend files and allow users to inspect and change the generated code.
- Provide assistance with code review, issue identification, test preparation and documentation.
- Support project export and deployment preparation using Docker-related files.

C. Scope

The work focuses on connecting common software-development activities rather than proposing a new language model. Its main contribution is the staged workspace and the use of intermediate artifacts—such as the SRS, architecture view, database design and API description—to guide later work. The system does not remove the need for developer decisions or independent testing.

III. RELATED WORK

Zhao et al. surveyed natural-language processing techniques used in requirements engineering. Their study describes applications such as extracting information from requirement text and supporting requirement quality and traceability [1]. This area is relevant to the domain-analysis stage of the proposed platform, where a free-form project description is converted into information that can be reviewed.

Requirements engineering for AI-based systems also presents challenges around specification quality, explainability and communication between stakeholders. Habiba et al. discuss these issues and emphasize the importance of clear requirements and collaboration [2]. These concerns support keeping generated requirements visible and editable instead of treating them as final decisions.

For programming tasks, CodeT5 is an encoder-decoder model designed for code understanding and generation [3]. Chen et al. studied large language models trained on code and discussed the opportunities and risks associated with their use [4]. AlphaCode explored generating and selecting candidate program solutions [5]. Together, these studies provide context for AI-assisted code creation, while also reinforcing the need to verify generated programs.

Multi-agent approaches have also been investigated for coordinating software-development work. ChatDev describes communication among agents performing software tasks [6], and MetaGPT organizes agent activity around defined roles and procedures [7]. The present project uses a sequence of stages and reviewable outputs; it should not be understood as a reproduction of either framework.

SWE-bench evaluates systems on real software issues drawn from GitHub repositories [8]. Its test-oriented approach is useful when planning an evaluation for a platform that generates or modifies code, because an output should be assessed by whether it meets a task's requirements.

A. Identified Need

The reviewed work covers requirements analysis, code generation, collaboration and evaluation. The project focuses on the practical connection between these areas: keeping the original project description, requirements document, architecture, database design, API plan and generated code aligned as development proceeds.

IV. SYSTEM DESIGN AND METHODOLOGY

The platform is organized as a sequence of seven stages. Each stage uses information already entered into the project workspace and produces an artifact for the next stage. Users should review the output before relying on it, particularly when the input description is incomplete.

A. Workflow Stages

1. Authentication and project workspace: A user signs in through JWT-based authentication and createsa workspace. Project details and technology preferences are entered here, and the workspace provides access to the artifacts produced later.
2. Domain analysis: The system processes the project description to identify domain entities, user roles,functional needs and constraints. These findings form an initial analysis, not a substitute for confirmation by the project team.

 This work is licensed under a Creative Commons Attribution 4.0 International License

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 11, November 2026

DOI: 10.17148/IJARCCE.2026.1511195

3. SRS preparation: The analysis is organized into an eleven-section SRS draft following the IEEE 830 style described in the project material. The draft is editable in Markdown and is intended to guide design and implementation.
4. Architecture visualization: React Flow is used to display components and their relationships in atwo-dimensional view. Three.js and React Three Fiber support a three-dimensional view of the proposed structure.
5. Database and API planning: The system prepares entity information, ER-diagram data, SQLdata-definition statements and REST endpoint descriptions following OpenAPI 3.0 conventions. These outputs need to be checked against the SRS.
6. Code generation: The documented implementation generates FastAPI backend artifacts, includingmodels, schemas, routers and CRUD operations. Monaco Editor provides a place to inspect and edit the generated files.
7. Review, testing and export: The workflow includes code-quality review, security checks, Pytest testsuites and ZIP export. Docker and Docker Compose files support packaging and deployment preparation.

B. Application Architecturezx

The application follows a full-stack design. The React and TypeScript frontend presents the sign-in screens, project workspace, documents, diagrams and code editor. FastAPI provides backend endpoints and coordinates analysis and generation requests. PostgreSQL stores project information and generated artifacts, while SQLAlchemy maps application objects to database records. The project documentation identifies OpenAI GPT-4o-mini and a custom domain-analyzer fallback for AI-assisted processing.

TABLE I MAIN APPLICATION COMPONENTS

Part	Role in the workflow
Frontend	Project workspace, SRS display, diagrams and code editing
Backend and AI service	API handling, domain analysis and generation support
Database	Storage of project details and generated artifacts
Development outputs	SRS, architecture, ER data, SQL, APIs, source files and tests

C. Technologies

Area	Technology
Frontend	React 19, TypeScript, Vite, Tailwind CSS v4
Visualization	React Flow, Three.js, React Three Fiber
Code editor	Monaco Editor
Backend	Python 3.13, FastAPI, Uvicorn
Database	PostgreSQL 16, SQLAlchemy 2.0
Authentication	JWT, OAuth2 Password Bearer, Bcrypt
AI support	GPT-4o-mini and custom domain-analyzer fallback
Packaging	Docker, Docker Compose, Nginx Alpine

V. IMPLEMENTATION STATUS AND EVALUATION

The project material describes the workflow and identifies the principal technologies used in the application. It covers project workspaces, domain analysis, SRS preparation, architecture views, database and API planning, code generation, review, testing and export. It also identifies Pytest as the testing framework and Docker-related files as part of packaging and deployment preparation.



International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 11, November 2026
DOI: 10.17148/IJARCCE.2026.1511195

The material available for this paper does not include a controlled experiment, a verified benchmark dataset, measured response times, test-coverage results or a comparison with manual development. For that reason, the paper does not report a numerical accuracy score or claim that the platform reduces development time or defects. The functions described here should be understood as documented project capabilities, not as independently measured outcomes.

A. Proposed Evaluation

A useful evaluation would begin with several project descriptions of different sizes and domains. For each input, the team should preserve the generated SRS, diagrams, database design, API definitions, source files, user edits, build logs and test results. Reviewers could then compare the outputs with an agreed reference specification.

- Check whether the domain analysis identifies the expected entities, actors and constraints.
- Review the SRS for missing, unclear or conflicting requirements.
- Compare architecture, database and API artifacts with the approved requirements.
- Build and run generated modules in a controlled environment and record failures.
- Use known code issues to assess correct findings, missed issues and false alarms.
- Record test results and collect structured feedback from users completing defined tasks.

If the platform is compared with a manual workflow, both groups should receive the same requirements and acceptance criteria. The model version, prompts, software environment, participant numbers and evaluation procedure should be recorded so that the results can be interpreted and repeated.

VI. DISCUSSION, LIMITATIONS AND FUTURE WORK

The platform's main design choice is to keep project activities together. A project description is used to prepare requirements; the requirements then inform architecture, data and API planning; and those artifacts provide context for code generation. This sequence gives users several opportunities to identify missing decisions before they reach implementation.

The editable SRS and code workspace are important because AI-generated material can be incomplete or incorrect. A generated model may not reflect every business rule, a proposed API may omit an operation, and code may contain logical errors or unsuitable dependencies. Tests can reveal some problems, but passing tests do not prove that every requirement has been met. Users must review the artifacts and test the application in its intended environment.

There are also operational considerations. A project description may contain confidential information, so users should understand how an external AI service handles submitted data. Generated code should be run only in an appropriately controlled environment, and dependencies should be checked before deployment. Changes to requirements can make earlier diagrams, SQL statements or API descriptions outdated; the platform should make such dependencies visible.

A. Future Work

- Evaluate the connection between requirements and each generated artifact using real projectexamples.
- Measure build success, test outcomes, review quality and document completeness.
- Add automated checks for mismatches between the SRS, architecture, database and API definitions.
- Compare the AI service and domain-analyzer fallback using the same inputs and evaluation criteria.
- Extend generation support to additional frontend and backend frameworks.
- Strengthen execution isolation, dependency checks and permission controls.
- Introduce version history and change-impact information for updated requirements.

VII. CONCLUSION

The AI-Driven Guiding System is a web-based project intended to help users move from an initial software idea to a set of connected development artifacts. Its seven-stage workflow covers workspace creation, domain analysis, SRS preparation, architecture visualization, database and API planning, code generation, and review, testing and export.

The documented implementation combines React and TypeScript, FastAPI, PostgreSQL,

SQLAlchemy, JWT authentication, AI-assisted processing, React Flow, Three.js, Monaco Editor and Docker. Its central contribution is the organization of these activities in one workspace, with intermediate outputs that users can inspect and revise.

 This work is licensed under a Creative Commons Attribution 4.0 International License



International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 11, November 2026
DOI: 10.17148/IJARCCE.2026.1511195

The current project material does not provide measured performance results. A controlled evaluation is therefore needed before conclusions can be drawn about accuracy, productivity or software quality. Such an evaluation should include independent reviews, reproducible test cases, build results and user feedback.

ACKNOWLEDGMENT

The authors thank Basaveshwar Engineering College, the Department of Computer Science and Engineering, the project guide and faculty members for their guidance and support.

REFERENCES

[1]. L. Zhao, W. Alhoshan, A. Ferrari, K. J. Letsholo, M. A. Ajagbe, E.-V. Chioasca, and R. T. Batista-Navarro,"Natural Language Processing for Requirements Engineering: A Systematic Mapping Study," ACM Computing Surveys, vol. 54, no. 3, Art. no. 55, 2021, doi: 10.1145/3444689.

[2]. U.-e. Habiba, M. Haug, J. Bogner, and S. Wagner, "How mature is requirements engineering for AI-basedsystems? A systematic mapping study on practices, challenges, and future research directions," Requirements Engineering, vol. 29, pp. 567–600, 2024, doi: 10.1007/s00766-024-00432-3.

[3]. Y. Wang, W. Wang, S. Joty, and S. C. H. Hoi, "CodeT5: Identifier-aware Unified Pre-trained Encoder-DecoderModels for Code Understanding and Generation," in Proc. EMNLP, pp. 8696–8708, 2021.

[4]. M. Chen et al., "Evaluating Large Language Models Trained on Code," arXiv:2107.03374, 2021, doi:48550/arXiv.2107.03374.

[5]. Y. Li et al., "Competition-level code generation with AlphaCode," Science, vol. 378, no. 6624, pp. 1092–1097,2022, doi: 10.1126/science.abq1158.

[6]. C. Qian et al., "ChatDev: Communicative Agents for Software Development," in Proc. 62nd Annual Meetingof the Association for Computational Linguistics, vol. 1, pp. 15174–15186, 2024, doi:18653/v1/2024.acl-long.810.

[7]. S. Hong et al., "MetaGPT: Meta Programming for a Multi-Agent Collaborative Framework," in Proc. ICLR,2024, arXiv:2308.00352.C. E. Jimenez et al., "SWE-bench: Can Lan

 This work is licensed under a Creative Commons Attribution 4.0 International License