WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

PROJECT IDEA RECOMMENDER

Nidish Kumar S[1], Kaviyarasan S[2], Ragul S[3], Ms. Vijayalakshmi S[4]

Assistant Professor, Department of Information Technology,
Sri Shakthi Institute of Engineering and Technology, Coimbatore-641062, Tamil Nadu, India. Email ID: nidishkumar2007@gmail.com

ABSTRACT :

Choosing a suitable software project is a difficult early decision for students of computing programmes, yet most students must make it without a reliable way to judge whether an idea fits their programming skills, team size and available time. Project ideas are scattered across many sources, are described in different ways, and are rarely tailored to the person reading them, so unsuitable topics are often discovered only after weeks of effort. This paper presents the Project Idea Recommender, a web-based application that accepts a student's preferences, compares them with a structured catalogue of project records, and presents a ranked list of suitable projects through a single interface. The system combines user authentication, preference capture, weighted multi-factor matching, threshold-based filtering, factor-by-factor score explanation, side-by-side comparison, skill gap analysis, automatic weekly roadmap generation and recommendation history storage. The application is implemented with a React and TypeScript frontend built with Vite, Firebase Authentication, Cloud Firestore and Firebase Hosting. Each project receives a score out of one hundred from seven weighted factors, and only projects scoring above a threshold are shown. The catalogue in the current implementation holds 305 projects in 20 domains. The scoring is rule-based and transparent, and it does not use machine learning. Automated checks of the code and the data have been run, while the functional and usability evaluation with students remains to be completed.

Keywords: Project Recommendation, Recommender Systems, Weighted Matching, Skill Gap Analysis, Explainable Scoring, Personalised Learning, React, Firebase.

INTRODUCTION

Completing a software project is a central requirement of engineering and computing programmes. Mini projects, academic projects and final-year projects are used to apply theory, build practical skills and prepare students for employment. The first and often most difficult step is selecting a topic. A good topic has to match the languages the team already knows, the number of members, the weeks available before submission and the level of difficulty the team can handle, and students seldom have a reliable way to check these conditions.

Web applications, cloud services and recommender-system techniques have created new possibilities for personalised guidance. A structured catalogue can describe projects in a uniform way, a scoring procedure can compare each description with the circumstances of a student, and an interface can show the result together with the reasons behind it. Combining these elements may help create a single tool that supports students while they decide on, and prepare for, a project.

Existing challenges in project selection

Existing sources include lists of project ideas, code repositories, online forums, earlier batches and suggestions from faculty. Although these provide useful inspiration, the lists are usually static, are written in different formats, and present the same ideas to every reader. A student must read many descriptions and then judge alone whether each idea is feasible. General recommendation services for courses, videos and products rank items by popularity or browsing behaviour, and therefore they do not state whether an item can be completed within a particular number of weeks by a particular team.

Another challenge is the dependence on experience. Students with limited experience may underestimate the effort of a topic, overlook technologies that must be learned first, or choose a team size that does not match the work. These problems are normally discovered late, after time has been spent, and can lead to incomplete work or a change of topic close to the deadline. Any automated tool can only estimate suitability from the information that students and catalogues provide, so its output should be treated as guidance and not as a guarantee.

Role of recommender systems and explainable scoring

Recommender systems filter a large set of items according to the needs of a user. Content-based approaches compare the attributes of items with a profile of the user, collaborative approaches use the behaviour of similar users, and hybrid approaches combine the two [1]–[4]. For project selection, the attributes needed for a content-based comparison, such as domain, languages, duration, team size and difficulty, are known in advance, and no history of earlier users is required, which suits a new application with few users.

Explanation is an important part of recommendation, because users are more likely to understand and trust a result when they can see why it was produced [5]. A weighted score built from a few stated factors can be shown factor by factor, so that the student sees which factor raised or lowered a percentage. Such a score is easy to inspect and adjust, although it does not learn from data, and its quality depends on the chosen factors, weights and catalogue.

Problem statement and objectives

Many existing approaches leave students to search several sources, compare descriptions manually and estimate feasibility on their own. This makes it difficult to obtain a quick and understandable shortlist of projects that fit a particular team. There is therefore a need for an integrated application that provides convenient access to personalised project suggestions while considering transparency, practicality and ease of use.
The main objective of this work is to develop a web-based Project Idea Recommender that combines a structured catalogue, weighted matching, skill gap analysis and roadmap generation in one platform. The system aims to simplify project selection, explain the reasons behind each recommendation, and help students prepare for the project they choose.

The specific objectives are:

- **i.** To provide a user authentication mechanism for controlled access to the application.
- **ii.** To capture the domain, known languages, skill level, team size, duration, project type and interests of a student through a simple form.
- **iii.** To store projects as structured records in a cloud database with a uniform set of fields.
- **iv.** To compute a weighted percentage score for every project and show only those above a threshold in descending order.
- **v.** To explain each score factor by factor.
- **vi.** To identify the skills a student still needs for a selected project and to measure readiness.
- **vii.** To generate a week-by-week roadmap with trackable tasks, and to support favourites,comparison and history.
- **viii.** To evaluate the application through functional, integration and usability testing.

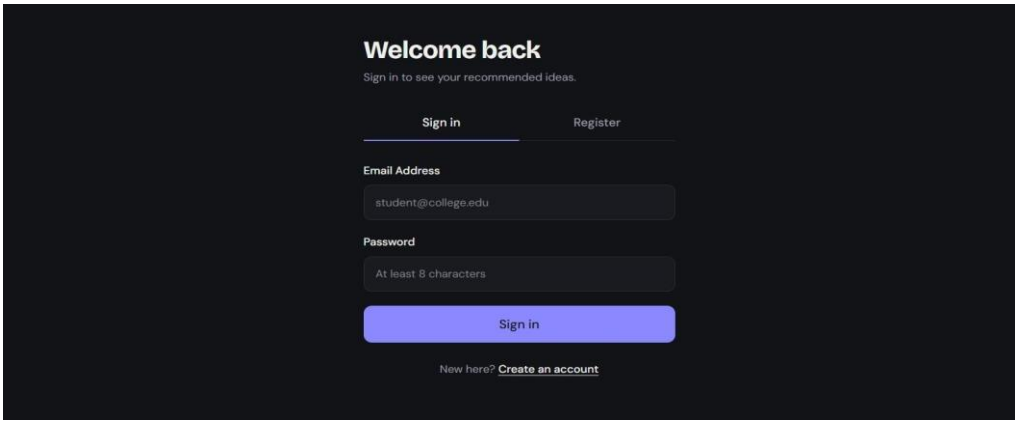
MATERIALS AND METHODS

Development approach

The project follows a structured software development approach consisting of problem identification, requirement analysis, system design, implementation, integration, testing and evaluation. During problem identification, the main difficulties faced by students while choosing a project are considered.
Requirement analysis identifies the major functions required by the application, including authentication, preference capture, scoring and ranking, explanation, comparison, skill gap analysis, roadmap generation and history storage. System design defines the relationship between the user interface, the application logic, the matching engine and the data storage components.
The implementation stage develops the modules using React, TypeScript and Firebase services. Integration connects the modules so that they exchange information correctly. The interface was refined over several design iterations, including a redesign of the sign-in page and the dashboard and the addition of a light and a dark theme. Testing is then used to identify errors, examine the behaviour of the application and determine areas requiring improvement.

System architecture

The system is organised into multiple layers, including the user interface layer, application logic layer, authentication layer, recommendation layer and data storage layer.
The user interface layer provides the screens through which students register, sign in, enter preferences, view ranked projects and open project details. The application logic layer reads and writes data through service modules and coordinates the other layers. The authentication layer, provided by Firebase Authentication, verifies the identity of the student, and protected routes allow only signed-in students to open the dashboard and the detail pages. The recommendation layer performs the matching, skill gap analysis and roadmap generation in the web browser.
The data storage layer, provided by Cloud Firestore, holds the project catalogue, the student profiles and the recommendation history, and security rules restrict access so that each student can reach only the student's own records. Favourites, comparison selections, skills marked as learned and roadmap progress are held in the local storage of the browser for each student. The architecture keeps the modules logically separate, so that the matching engine or the data source can be changed without redesigning the whole application.



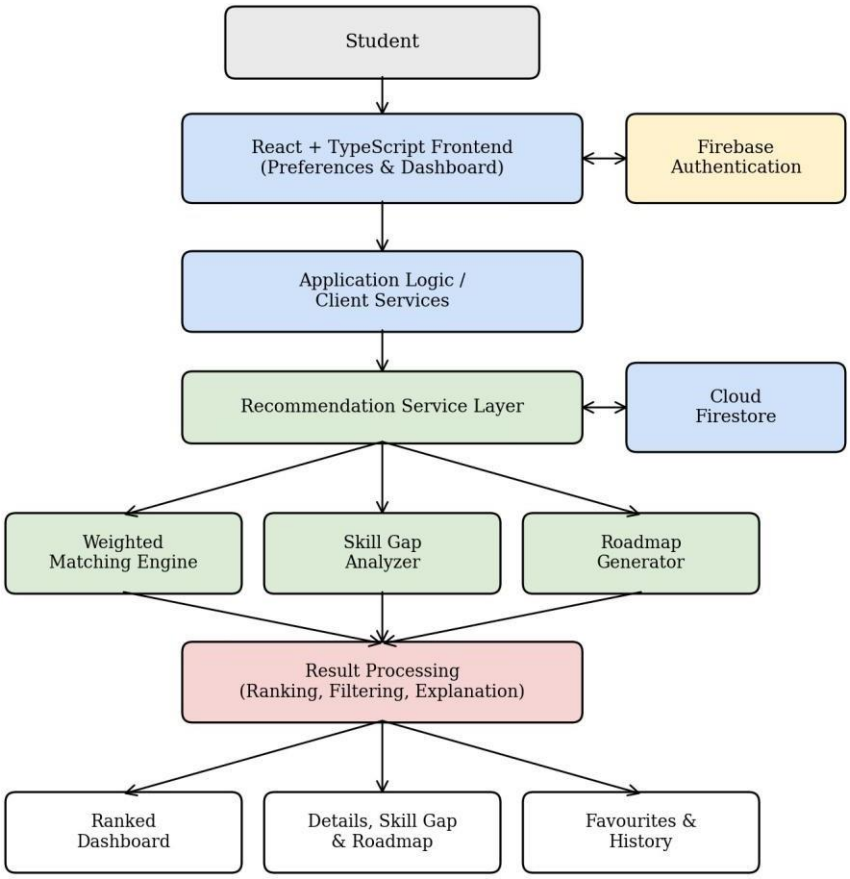


Fig. 1. System Architecture of the Project Idea Recommender.

Technology stack and development environment

The application is a single-page web application written in TypeScript with React, built with the Vite tool and navigated with React Router. Icons are provided by the Lucide library, and the interface uses a shared set of CSS variables for the light and dark themes. Firebase Authentication manages accounts, Cloud Firestore stores data, and Firebase Hosting delivers the built application [8]–[10].

Maintenance scripts written for Node.js use the Firebase Admin SDK to load the project catalogue into Firestore, to verify its contents and to check it for duplicates. These scripts run on the computer

of the developer with a private service account key that is never included in the application or in the source repository. Source code is managed with Git and GitHub, and the application is deployed with a build command followed by a Firebase Hosting deployment.

Development and testing were performed using a code editor and a supported web browser. The code was checked with a linter and the TypeScript compiler, and the production build was generated with Vite.

Data collection and processing

The system processes the information required to provide its recommendation features, such as account details, the preferences entered by a student, project records, recommendation results and the progress marked on a roadmap. The project catalogue was prepared for this work, and each record follows the same structure. No data from other users is needed to produce the recommendations.

The catalogue in the current implementation contains 305 project records in 20 domains, and each domain has at least ten records. Of these records, 76 are of beginner level, 142 intermediate and 87 advanced. The recommended team size is two members for 114 records and three members for 133 records, with smaller numbers for one, four and five members. Each record has twenty fields, which include the title, domain, description, languages, technologies, difficulty, team size, duration, project type, interest keywords, features, modules and implementation steps.

Personal information is collected only when it is necessary for the functions of the application. The application validates the preferences before processing them, for example by requiring at least one language and by limiting the duration to the supported range. A maintenance tool loads records with fixed identifiers so that repeating the load does not create duplicates, and a checking tool detects repeated identifiers, repeated titles and titles with a very high word overlap before records are added.

Database design

The database is organised as document collections in Cloud Firestore. The projects collection holds one document for each project, identified by a stable name formed from its title. The users collection holds one profile document for each student, created at registration, and the recommendation history collection holds one document for each search.

A project document stores its title, domain, descriptions, languages, technologies, difficulty, recommended team size, duration, project type, interest keywords, features, modules and implementation steps. A profile document stores the identifier, name, e-mail address and creation time of the student. A history document stores the identifier of the student, the preferences submitted, the identifiers of the top recommendations and the scores of the highest-scoring records.

Security rules allow signed-in students to read the projects, allow changes to the projects only through an administrator credential, and allow each student to reach only the student's own profile and history. Only the information required for the application is retained, and the retention of stored records should be reviewed as the number of users grows.

Matching and scoring

For each project, the matching engine computes a partial score for seven factors and adds them to obtain a score out of one hundred. The factors are the domain, the languages and technologies, the duration, the team size, the skill level, the interests and the project type, and their maximum contributions are shown in Table 1. The weights give the greatest influence to the domain and to the languages, because these decide most strongly whether a project can be started.

Each partial score lies between zero and the maximum for its factor. Examples are the overlap between the languages known to the student and those needed by the project, the closeness of the available weeks to the estimated duration of the project, and the number of interest keywords found in the text of the project. When a student leaves a factor unspecified, a neutral partial score is given, so that the omission neither rewards nor penalises any project. The total is rounded to a whole percentage.

Only projects whose percentage is strictly greater than a threshold of seventy-five are retained. The retained projects are ordered from the highest to the lowest percentage, and projects with equal percentages are ordered alphabetically by title. If no project qualifies, the application suggests changing the preferences. In an early test session with a catalogue of 105 projects, 71 projects exceeded the threshold for the preferences entered.

The scoring should be evaluated with preferences of known suitability, for example a student who knows only one language and has four weeks. Its behaviour should be assessed for consistency and for the possibility of unsuitable suggestions. Because the score depends on how complete the catalogue records are, the application presents the result as guidance, together with the factor-by-factor explanation, and not as a certain judgement. The score does not use machine learning and does not learn from the choices of students.

Factor	Maximum score	Compared inputs
Domain	25	Selected domain and project domain
Languages and technologies	25	Languages known and those required by the project
Duration	15	Available weeks and estimated duration
Team size	10	Team size and recommended team size
Skill level	10	Skill level and difficulty of the project
Interests	10	Interest keywords and project text
Project type	5	Selected type and project type
Total	100	Sum of the seven partial scores

Table 1. Factors of the matching score and their maximum contributions.

Recommendation workflow

The workflow begins when the student signs in and submits preferences through the form. The application validates the input, retrieves the project records and computes the score of every project. If the catalogue cannot be read from the database, the application falls back to a built-in copy of the catalogue so that the student can still continue.

The retained projects are displayed on the dashboard as cards showing the domain, title, short description, percentage and key attributes. The student can search by keyword, filter by domain and difficulty, save favourites and select up to three projects for comparison. Opening a project shows its details, the factor-by-factor explanation, the skill gap analysis and the roadmap. Each search is stored in the history of the student.

After processing, the application should give clear feedback about the state of the result. A list of recommendations should be distinguished from an empty result, and an error in reading the data should be distinguished from a successful search. These states are handled separately in the interface wherever the implementation supports them.

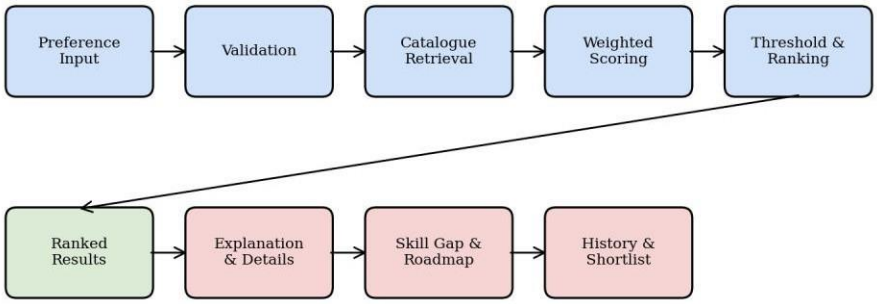


Fig. 2. Workflow of the Project Idea Recommender.

Testing and evaluation

Testing is required to evaluate the correctness and reliability of the application. Unit testing can be used to examine individual functions such as the scoring function, the skill gap analyzer and the roadmap generator, while integration testing checks communication between the interface, the authentication service and the database. At the time of writing, the project does not include automated unit tests, and these are planned as future work.
System testing evaluates the overall workflow, including registration, sign-in, entry of preferences, ranking, comparison, skill gap analysis, roadmap tracking and history. Usability testing examines readability, navigation, responsiveness and the light and dark themes on screens of different sizes [7].
The evaluation should also include error conditions such as invalid credentials, empty or incomplete preferences, a search that returns no project, and an unavailable network connection. Evaluation of recommender systems commonly combines measures of accuracy with studies of user satisfaction [6], and any quantitative claims about recommendation quality, response time or satisfaction should be reported only when supported by actual measurements.

RESULTS AND DISCUSSION

Implemented system functionality

The application brings the main steps of project selection into one interface. These include user authentication, preference capture, weighted scoring with a threshold, explanation of the score, favourites, comparison of up to three projects, skill gap analysis, a weekly roadmap with progress tracking, and a history of searches. A light and a dark theme are available, and the layout adapts to screens of different sizes.
Automated checks were run on the implemented application. The code analysis tool reported no errors and two warnings, the TypeScript compilation and the production build completed successfully, a duplicate checker found 305 unique projects with no repeated identifiers or titles, and
a dry run of the loading script passed the structure and count checks. These checks show that the code and the data are consistent, but they do not measure the quality of the recommendations.
The production build is about 1.76 MB before compression, which is large because a built-in copy of the catalogue is bundled with the application.

Functional evaluation

Functional evaluation examines whether the implemented modules behave according to their requirements. Representative test scenarios for each module are summarised in Table 2. The results should be documented according to the observed behaviour of the application, and features that have not been tested should be clearly identified as pending evaluation.

Test ID	Module	Test scenario	Expected behaviour
T1	Authentication	Sign in with valid and invalid credentials	Valid student accepted; invalid student rejected with a message
T2	Preferences	Submit the form without selecting a language	Validation message shown; no search made
T3	Matching	Preferences that fit a project in every factor	Project appears in the list with a high percentage
T4	Threshold	Project scoring exactly 75 and project scoring 76	75 excluded; 76 shown
T5	Ordering	Two projects with equal percentage	Ordered alphabetically by title
T6	Skill gap	Project requiring a language the student does not know	Language listed under skills to learn; readiness below 100%

T7	Roadmap	Tick and untick tasks, then reload the page	Progress updates and is kept for that student
T8	Data access	Request the profile or history of another student	Request denied by the security rules

Table 2. Representative functional test scenarios for the application.

Any failed tests should be investigated and repeated after corrections are made. The scoring function in particular should be tested with preferences whose expected ranking is known in advance, and the results should be reported for each factor and not only as one overall figure.

Comparison with conventional approaches

Conventional approaches require students to read many project lists, to judge feasibility by themselves and to plan the work separately. The proposed system aims to bring ranking, explanation, skill analysis and planning together in one interface, potentially improving accessibility and learning value.
The use of a transparent weighted score makes the reasoning visible, but it also means that the suggestions are only as good as the chosen weights and the catalogue. The proposed system should not be considered more accurate than other approaches or tools without comparative testing and feedback from students.

Discussion

The architecture provides a framework for integrating project cataloguing, matching, skill analysis and planning. A single interface can make project selection easier, and modular development can simplify the addition of new domains, languages and options, since these values are defined in one shared configuration.
The system also faces practical challenges. The score depends on the completeness of the records and on the weights, which were chosen by the developers and have not yet been tuned against feedback from students. The threshold can leave students with very unusual preferences with few results. Favourites, comparison selections and progress are kept in the browser, so they are not shared between devices. The built-in copy of the catalogue increases the download size and can hide database errors from the student, and the project does not yet include automated tests.
Privacy and security are also important because the application stores student accounts and searches. Authentication, security rules and the separation of private keys from the application are therefore necessary [11]. Further evaluation should measure the satisfaction of students, the quality of the suggestions compared with their own choices, and the loading time with larger catalogues. These measurements would provide stronger evidence of the effectiveness of the system in practical situations.

CONCLUSION

The Project Idea Recommender presents a practical approach to helping students choose and prepare for a software project through a single web application. The system combines a structured catalogue, weighted multi-factor scoring with a threshold, factor-by-factor explanation, comparison, skill gap analysis and an automatic weekly roadmap.
The modular architecture provides a foundation for adding more projects, domains and features while considering usability, transparency and data security. However, the practical effectiveness of the application depends on the quality of the catalogue, the suitability of the weights and further testing with real students.
Future development can focus on saving favourites and progress in the database so that they follow the student across devices, loading projects in pages, adding automated tests for the scoring function, tuning the weights with feedback, providing tools for administrators to manage projects, and linking each skill to learning resources. Further testing and user evaluation are required to determine the usefulness of the system in real classes.
In conclusion, the proposed application offers a foundation for a project recommendation and planning system. With appropriate validation, user feedback and security safeguards, it can contribute to making project selection more organised, personal and understandable for students.

ACKNOWLEDGMENTS

The authors thank Sri Shakthi Institute of Engineering and Technology for the academic environment and facilities that made this work possible. They are also grateful to the Head of the Department of Information Technology, to the project guide and to the faculty members for their advice and encouragement during the project.

REFERENCES

1. F. Ricci, L. Rokach, and B. Shapira, Eds., Recommender Systems Handbook, 3rd ed. New York,NY, USA: Springer, 2022.
2. G. Adomavicius and A. Tuzhilin, “Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions,” IEEE Transactions on Knowledge and Data Engineering, vol. 17, no. 6, pp. 734–749, 2005.
3. R. Burke, “Hybrid recommender systems: Survey and experiments,” User-Modeling and User-Adapted Interaction, vol. 12, no. 4, pp. 331–370, 2002.
4. P. Lops, M. de Gemmis, and G. Semeraro, “Content-based recommender systems: State of the artand trends,” in Recommender Systems Handbook, F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, Eds. Boston, MA, USA: Springer, 2011, pp. 73–105.
5. N. Tintarev and J. Masthoff, “A survey of explanations in recommender systems,” in Proc. IEEE23rd Int. Conf. Data Engineering Workshops, 2007, pp. 801–810.

6. J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl, "Evaluating collaborative filtering recommender systems," ACM Transactions on Information Systems, vol. 22, no. 1, pp. 5–53, 2004.

7. J. Nielsen, Usability Engineering. San Diego, CA, USA: Academic Press, 1993.

8. React. React Documentation. Available at: https://react.dev/

9. Google Firebase. Firebase Documentation. Available at: https://firebase.google.com/docs

10. Vite. Vite Documentation. Available at: https://vite.dev/guide/

11. OWASP Foundation. OWASP Application Security Guidance. Available at: https://owasp.org/