WWW.IJRPR.COM

International Journal of Research Publication and Reviews

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

Student AI Assistant

Mithun M[1], Mohamed Yaseen N[2], Prakash Rajan P[3], Ragul S[4], Priyadharshini R[5]

[1](714024243120), [2](714024243124), [3](714024243148), [4] (714024243154)

Department of Artificial Intelligence and Data Science, Sri Shakthi Institute of Engineering and Technology, Coimbatore, Tamil Nadu, India

[5]Project Guide, Department of Artificial Intelligence and Data Science, Sri Shakthi Institute of Engineering and Technology, Coimbatore, Tamil Nadu, India

Email: mithunmithun8461@gmail.com

ABSTRACT

Most students who prepare for exams or placements end up using four or five different tools at once. They solve problems on one website, note down study hours in a timer app, keep reminders in the phone clock and write notes somewhere else. None of these tools knows about the others, so the student has no single place to see what has been done and what is still left. In this paper we describe the Student AI Assistant, a web application that brings these things together. It has a coding practice section similar to LeetCode, a study timer, an alarm and reminder section, and a notes section organised by topic. The topics a student must finish are saved in a database, and every problem attempt, study session, alarm and note is linked to one of those topics. On top of this data we added an AI assistant that gives hints for problems, explains concepts, summarises notes and suggests what to study next. The system is built in three layers: the user interface, a set of REST services, and a relational database together with a sandbox that runs submitted code. We tested the main functions of every module and discuss how the combined design helps in daily study. We also point out the weak parts of the present version and the work we plan to do next.

Keywords: student assistant, coding practice, study timer, alarm, topic notes, database, AI assistant, spaced revision.

1. INTRODUCTION

For a computer science student, getting a job or clearing a competitive exam depends a lot on problem-solving ability. Websites such as LeetCode, HackerRank and CodeChef let anyone practise coding problems with instant feedback. Separate apps help with focus timers, alarms and note taking. Each of them does its own job well, but they were not designed to work together, and a student has to carry the whole plan in their head.

Consider a normal study day. A student solves two problems on a coding website, starts a timer on the phone, sets an alarm for the evening revision and types a few notes into a notebook app. At the end of the week there are simple questions that are hard to answer. How many hours went into graphs and how many into databases? Which problems failed more than twice? Which topics from the syllabus have not been touched at all? The answers are scattered over several apps, and some of them were never recorded. Without a record, no tool can suggest what the student should do next.

Language models have become good enough to explain concepts, give hints and summarise text [7][8]. A model like this is far more useful when it knows something about the person it is helping. If it can see which topics a student has pending, which problems they failed and what they wrote in their notes, its answers can be specific instead of generic. This idea is the starting point of our work.

1.1 Problem Statement

Students do not have one place where they can practise coding problems, measure study time, get reminders, keep notes for each topic and track the topics they are required to complete. Because of this, data is scattered, study time is poorly tracked, revision is forgotten and planning is done by guesswork.

1.2 Objectives

We set the following objectives for the project:

- to build a coding practice module in which problems are grouped by topic and difficulty, code is run against test cases and every submission is saved;
- to build a study timer that records how long the student works on each topic;
- to build alarms and reminders for study slots, revision and deadlines;

- to let the student keep notes for each topic and search them later;
- to keep the required topics and all the above activity in one database;
- to add an AI assistant that uses this stored data to help the student.

1.3 Contribution

The paper makes three contributions. The first is a design in which practice, timing, reminders and notes share one topic model and one database. The second is a simple way of combining a rule-based study planner with an AI assistant, so that suggestions come from the student's own records. The third is a description of how we tested the prototype, together with an honest list of its limits. The system is meant for individual self-study and can also be used next to regular classes.

1.4 Organisation of the Paper

Section 2 reviews earlier work. Section 3 explains the design and architecture. Section 4 describes how the system was built and the methods used inside each module. Section 5 gives the results and a discussion. Section 6 concludes the paper and Section 7 lists future work.

2. LITERATURE SURVEY

Our system touches several areas: online judges, research on how people remember, time-management methods, learner modelling, and AI help for programming. We go through each of them briefly and then explain what is missing.

2.1 Online Judges and Coding Practice

Online judges give learners a large set of problems and check their code automatically against hidden test cases. Topic tags and difficulty levels make it easy to practise in a planned way. Roediger and Karpicke [4] showed that trying to recall or apply knowledge helps memory more than reading the same material again, and solving a problem is a form of such active recall. A broad review of study techniques by Dunlosky et al. [12] also rated practice testing and distributed practice among the most useful methods for students. The weakness of existing judges is that they look only at problem solving. They know nothing about the student's timetable, reminders or notes.

2.2 Spacing and Revision

Ebbinghaus [1] observed that memory fades with time and that repeated review slows the fading. A later review of many experiments by Cepeda et al. [3] found that practice spread over days gives better retention than the same amount of practice done at once. Spaced repetition software uses this idea by scheduling the next review according to how well an item was remembered. We use a much simpler form of the idea: topics that were not revised for a long time, or that gave poor results, are pushed up in the study plan.

2.3 Time Management

Many students divide their work into short intervals with breaks in between, a method popularised as the Pomodoro Technique [2]. Sweller's work on cognitive load [11] gives a reason why this can help, since working memory is limited when a person solves a hard problem. Focus timer apps are common, but they usually save only the total time. They do not record which topic the time was spent on, which is exactly what a student needs to know.

2.4 Knowledge Tracing

Knowledge tracing tries to estimate what a learner knows from their history of answers. Corbett and Anderson [5] proposed a Bayesian model for this, and Piech et al. [6] later applied deep neural networks. Such models work well when large amounts of interaction data are available and are normally built into tutoring systems for one subject. For a single student with a few weeks of data they are too heavy, so we use simple per-topic statistics instead.

2.5 AI Help for Programming and Learning

Transformer models [10] and large language models [7] can write and explain text, and models trained on code can explain and fix programs [8]. This is useful for learners, but there is a risk. If a model hands over the full solution at once, the student learns very little. For this reason we limit the assistant to hints and explanations, and we feed it the student's own context.

2.6 Summary and Gap

Type of tool	What it does well	What is missing
Online judges	Many problems, automatic judging, tags and difficulty levels	No timer, alarms or notes; not linked to the student's own topic list
Focus timer apps	Easy time-boxing and break reminders	Time is not tied to a topic or a problem
Alarm and calendar apps	Reliable notification at the set time	Do not know the study goals or progress
Note-taking apps	Flexible writing, folders and search	Not connected to practice results or time spent
Tutoring systems / knowledge tracing	Estimate what the learner knows	Need large data; usually for one subject; heavy for self-study
General AI chat assistants	Answer questions on almost any topic	No stored personal data; may give full solutions; not tied to practice

Table 1. Comparison of existing tool categories

Looking at this survey, each kind of tool covers one part of preparation, while the research on learning suggests that active practice, spaced revision and focused time all matter together. We could not find a light-weight tool that stores the student's required topics in a database and connects practice, timing, reminders and notes to them, with an AI assistant working on the same data. The Student AI Assistant is our attempt to fill that gap.

3. SYSTEM DESIGN AND ARCHITECTURE

3.1 Design Goals

We kept five goals in mind while designing the system. All modules should share the same topics and the same database. Everything the student does should be stored, so that progress can be checked later. Common actions such as starting the timer or opening a note should feel quick. The modules should be independent enough to be developed and replaced one at a time. Finally, code written by users must run in an isolated place, and each student's data must be protected.

Another point we paid attention to is that the tool should not demand extra effort. A student who already has a busy day will not enter data by hand into five different screens. For this reason the timer and the alarms ask for as little as possible, usually only a topic, and the coding module records every attempt without any action from the student. The more of the record that fills itself, the more reliable the planner and the assistant become.

3.2 Overall Architecture

The system has three layers, shown in Fig. 1. The top layer is the user interface, which contains the dashboard, the code editor with problem statements, the study timer, the alarm manager and the notes editor. The middle layer is a set of REST [9] services for login, problems and judging, timers, alarms, notes and topics, and the AI assistant. The bottom layer holds the relational database and the outside services that the middle layer calls, namely the code execution sandbox and the AI language model.

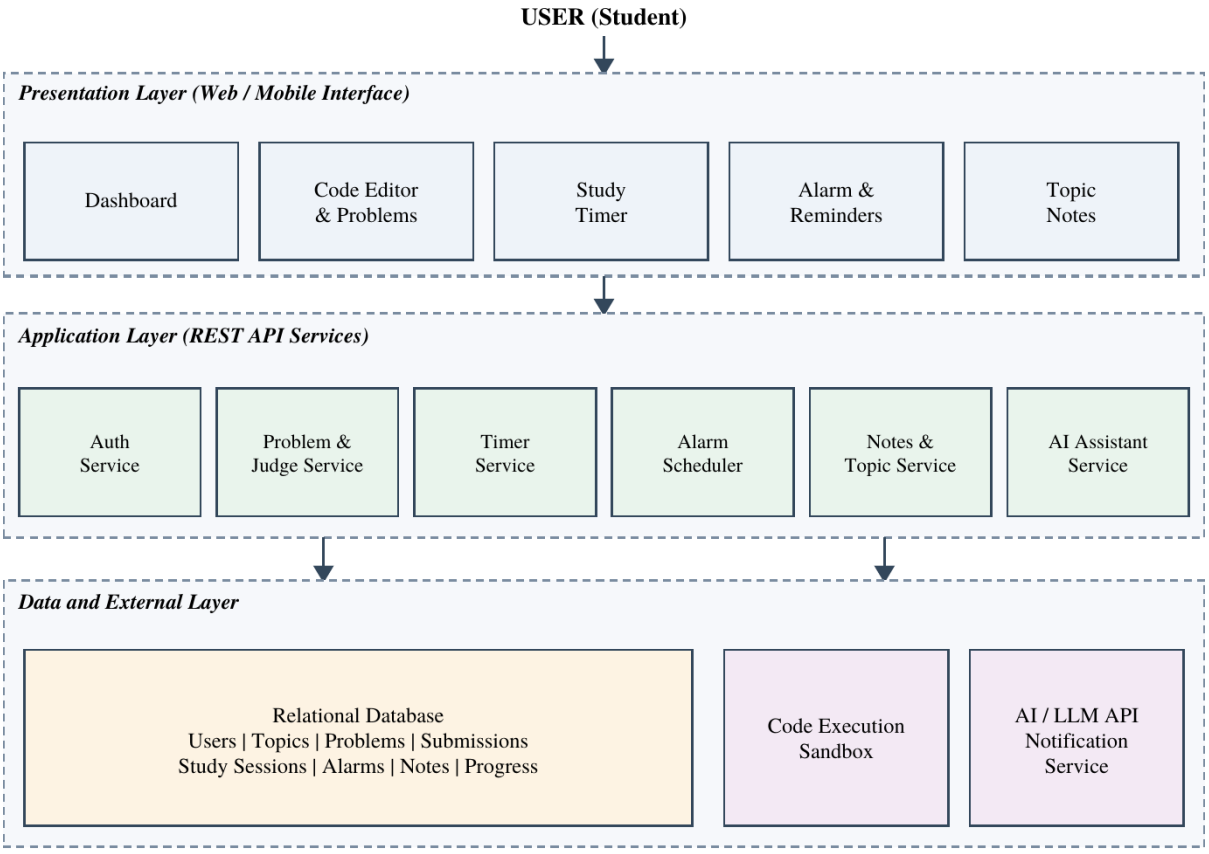


Fig. 1. Layered architecture of the Student AI Assistant

3.3 Modules

Module	What it does	What is stored
Coding Practice	Shows problems by topic and difficulty; code editor; runs code on sample and hidden test cases; shows verdict and run time	Problems, test cases, submissions, verdicts, number of attempts
Study Timer	Start, pause and stop a study session for a topic; short breaks; daily and weekly totals	Session topic, start time, duration, type (study or break)
Alarms and Reminders	One-time and repeating alarms for study slots, revision and deadlines; snooze and dismiss	Title, time, repeat rule, linked topic, status
Topic Notes	Create, edit, tag, search and delete notes for each topic; summary on request	Note text, topic, tags, time of last edit
Required Topics	List of syllabus or interview topics with status and priority	Topic, status (pending, in progress, done), priority, last revised date
AI Assistant	Hints, explanations, next-topic suggestion, note summary	Optional log of questions and suggestions

Table 2. Modules of the system and the data each one uses

3.4 Database Design

We chose a relational database [13] because the data is well structured and the dashboard needs to join tables, for example to find the time spent and the problems solved for one topic. The main tables are listed in Table 3. The *topics* table is the centre of the design. Problems, notes, study sessions and alarms each carry a topic key, and this is what makes a combined progress view possible.

Table	Main fields	Relationship
users	user_id (PK), name, email, password_hash, created_at	One user owns many topics, sessions, notes and alarms
topics	topic_id (PK), user_id (FK), title, category, priority, status, last_revised	Parent of problems, notes, sessions and alarms
problems	problem_id (PK), topic_id (FK), title, statement, difficulty, tags	Many problems per topic
test_cases	case_id (PK), problem_id (FK), input, expected_output, is_hidden	Many test cases per problem
submissions	submission_id (PK), user_id (FK), problem_id (FK), language, code, verdict, runtime_ms, submitted_at	Many submissions per user and problem
study_sessions	session_id (PK), user_id (FK), topic_id (FK), start_time, duration_sec, type	Many sessions per topic
alarms	alarm_id (PK), user_id (FK), topic_id (FK), title, trigger_time, repeat_rule, status	Many alarms per user
notes	note_id (PK), user_id (FK), topic_id (FK), title, content, tags, updated_at	Many notes per topic

Table 3. Main database tables

3.5 Data Flow

When the student opens the dashboard, the interface asks the server for a summary. The server reads the database and works out how many topics are finished, how much time went into each topic this week, how the recent submissions turned out and which alarms are due in the next day. All this comes back in one response. When the student submits code, the problem service sends the code and test cases to the sandbox, gets back a verdict and run time, saves the submission and updates the statistics of that topic. When the timer is started, a session record is opened, and it is closed when the timer stops. The alarm scheduler runs in the background, looks for alarms that are due and sends the notification.

3.6 AI Assistant Design

Every request to the assistant is built from three parts: a fixed instruction describing its role, some context taken from the database, and the student's question. The instruction tells the assistant to give hints and explanations and not to write a full working solution unless the student asks for it after several failed attempts. The context includes the current problem, the verdict of the last submission, the title and recent notes of the linked topic and the topic status. For suggestions, the assistant is given the ranked topic list produced by the planner in Section 4 and only has to put it into friendly words. Because the context comes from the student's own records, the answers stay on the point.

3.7 Security and Other Requirements

Passwords are stored as salted hashes, every request needs a session token, and every database query is filtered by the logged-in user, so one student cannot read another's data. Submitted code runs in a sandbox with limits on time, memory and network, and it cannot reach the application database. Inputs are checked on the server and queries are parameterised to prevent injection. Only the context needed for a request is sent to the external AI service. Session and alarm records are written to the database at once, so refreshing the browser or losing the connection for a moment does not lose data. The services keep no state of their own, which means several copies can run side by side if the number of users grows.

3.8 A Typical Day with the System

To make the design concrete, here is how a student might use the platform in one day. In the morning the dashboard shows three suggested topics, for example Graphs, Dynamic Programming and SQL Joins, because Graphs has a low acceptance ratio and has not been touched for nine days. The student picks Graphs, starts the timer and opens two problems. The first one is solved on the second attempt. On the second problem the student gets stuck and asks for a hint, and the assistant first points out that the problem is about finding connected components before suggesting a traversal.

After the session the student writes a short note about breadth-first search and asks the assistant to summarise it, then saves the summary under the same topic. An alarm is set for the next evening with the title *Revise Graphs*. At night the dashboard shows the total time spent per topic for the day, the two submissions and the note that was added. Nothing in this routine required leaving the application, and every step has left a record in the database for the next day's suggestions.

4. METHODOLOGY

4.1 Development Approach

We built the system iteratively, in line with common software engineering practice [14], as shown in Fig. 2. We started by listing the problems students face: scattered tools, no idea of time spent and missed revision. From this we drew up the architecture and the database. The modules were then written one by one, connected through the shared topic model and tested. Whatever we found during testing and from student feedback went back into the requirements for the next round.

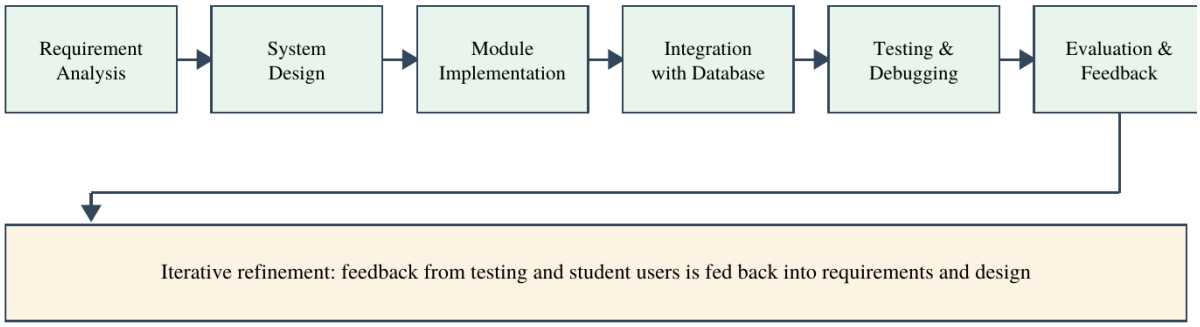


Fig. 2. Iterative development process

4.2 Coding Practice Module

A problem is stored with its statement, difficulty, topic and test cases, some of which are hidden. The *Run* button executes the code on the sample cases and shows the output. The *Submit* button runs all cases and produces one of these verdicts: Accepted, Wrong Answer, Time Limit Exceeded, Runtime Error or Compilation Error. Each submission is saved with its verdict and run time, and the attempts, accepted problems and acceptance ratio of the topic are updated. The planner uses these numbers later.

4.3 Study Timing Module

The timer works in short focus intervals with breaks, in the style of the Pomodoro Technique [2], and the lengths can be changed. The student picks a topic before starting, so the time is counted for that topic. The start time is saved when the session begins, and the duration is calculated from timestamps and not from a counter in the browser. This way the record stays correct even if the page is refreshed. Daily and weekly summaries come from grouping the sessions by topic and date.

4.4 Alarm and Reminder Module

Each alarm has a trigger time and, optionally, a repeat rule such as daily, weekly or on chosen days. A background scheduler checks every short interval for active alarms whose time has passed. It sends the notification and then either marks the alarm as done or calculates the next trigger time if the alarm repeats. Snooze simply moves the trigger time ahead by the chosen number of minutes. An alarm can be linked to a topic, so a reminder such as *Revise Graphs* can open the notes and problems of that topic.

4.5 Topic Notes Module

Notes are written inside a topic and have a title, formatted text and tags. A full-text search over title, content and tags helps the student find old material quickly. On request, the content of a note is sent to the AI assistant, which returns a short summary and the key points. The student can keep this summary as a new note. Since notes belong to topics, the revision screen of a topic can show the student's own notes beside the practice problems.

4.6 Required Topics and Study Planner

The required topics, such as a syllabus or a list of interview topics, are entered once and stored with a status and a priority. To suggest what to study next, the planner gives every unfinished topic a score:

Score(t) = w1 x Weakness(t) + w2 x Staleness(t) + w3 x Importance(t)

Weakness is one minus the acceptance ratio of recent submissions on the topic. Staleness is the number of days since the topic was last studied, scaled between 0 and 1. Importance is the priority set by the student. The weights w1, w2 and w3 can be changed in the settings. Topics are sorted by score and

the first few are shown as the plan for the day. The idea comes from the spacing effect [1][3] and from the practice of watching a learner's weak areas [5], but it is simple enough to work for one student with no training data.

4.7 AI Assistant

The assistant is used in four situations: a hint for a problem, an explanation of a concept, a suggestion for the next topic, and a summary of a note. Each has its own prompt template which is filled with data from the database. For hints, the template asks for the help to come in steps, first a small nudge about the concept, then a suggested approach, and pseudo-code only at the end, so the student still has to do most of the thinking, which fits the idea that some difficulty during learning is useful [15]. The reply is shown on screen and can be saved to the notes of the topic.

4.8 Testing

We tested at three levels. Unit tests checked small functions such as the duration calculation, the next trigger time of repeating alarms and the scoring formula. Integration tests checked that the modules and the database work together, for example that a submission changes the statistics of its topic. System tests ran whole scenarios such as planning a day, solving a problem, running a timed session and writing a note, and we also asked student users for their opinion on ease of use and usefulness. Response times of the main operations were measured under normal load.

4.9 Implementation Notes

A few practical choices kept the system simple. The interface and the services talk only through JSON over HTTP, so a mobile client can be added later without touching the services. Times are stored in one standard time zone and converted to local time only when displayed, which avoids wrong alarms when the device time zone changes. Slow operations like running code or waiting for the AI are done asynchronously and the interface shows progress. Indexes on the foreign keys and the date columns keep the dashboard query fast as the tables grow.

Settings such as timer lengths, scoring weights, snooze time and sandbox limits are kept in a settings table and not in the source code, so they can be tuned for different groups of students without redeploying. The judge, the scheduler and the AI service write logs, which helps in finding the cause when something fails.

4.10 Summary of the Main Procedures

For clarity, the two background procedures of the system can be summarised as follows. The planner runs when the dashboard is opened. It reads all topics of the student whose status is not done, collects the recent submissions and the last-studied date of each topic, calculates the weakness, staleness and importance values, applies the weights, sorts the topics by score and returns the first three. If a topic has no submissions yet, its weakness is taken as a neutral value so that new topics are not ignored.

The alarm scheduler runs at fixed short intervals. It selects the alarms whose status is active and whose trigger time is not later than the present time, sends a notification for each of them, and updates the record. A one-time alarm is marked done, a repeating alarm gets its next trigger time from the repeat rule, and a snoozed alarm gets the new time chosen by the student. Because the status is changed in the same step as the notification, an alarm is not sent twice even if the scheduler is restarted.

5. RESULTS AND DISCUSSION

5.1 Experimental Setup

The prototype was run on an ordinary development machine and used through a web browser. The database was loaded with sample problems on common topics (arrays, strings, linked lists, stacks and queues, trees, graphs, dynamic programming and databases), a list of required topics and a few alarms. Test users then went through the scenarios described in Section 4.8.

5.2 Functional Results

Table 4 lists the main test cases with the expected and the observed behaviour. Besides the normal flow of each module, we included boundary cases such as an empty note, an alarm set in the past and a submission with a compilation error.

No.	Test case	Expected result	Observed
1	Submit a correct solution	Verdict Accepted; submission saved; topic statistics updated	Pass
2	Submit a solution with wrong output	Verdict Wrong Answer; failing case shown	Pass

No.	Test case	Expected result	Observed
3	Submit code with an infinite loop	Verdict Time Limit Exceeded; sandbox stops the process	Pass
4	Start and stop a study session	Session saved with correct duration and topic	Pass
5	Refresh the browser during a session	Session continues; duration taken from stored start time	Pass
6	Create a one-time alarm	Notification at trigger time; status set to done	Pass
7	Create a repeating alarm and snooze it	Next trigger calculated; snooze delays correctly	Pass
8	Create, edit, search and delete a note	Note saved and found by keyword	Pass
9	Ask for the next-topic suggestion	Topics ranked by score; top topics shown	Pass
10	Ask for a hint on a problem	Hint returned without the full solution	Pass

Table 4. Functional test cases

5.3 Performance

We did not carry out a formal load test on the prototype. Instead, we fixed response-time targets for the main operations during design, so that the interface would always feel quick to a student using it. Table 5 lists these targets. Measuring the actual response times on the final deployment, under different numbers of users, is planned as part of the longer trial described in Section 7.

Operation	Description	Design target
Dashboard load	Summary of progress	Under 1 s
Run code	Sandbox run on sample cases	Under 3 s
Submit code	Run on all cases and update database	Under 5 s
Start / stop timer	Write session record	Under 0.5 s
Save note	Write note to database	Under 0.5 s
AI hint	Build prompt and get model reply	Under 10 s

Table 5. Response-time targets for the main operations

5.4 User Feedback

Student users were asked about ease of use, the usefulness of the combined dashboard, the next-topic suggestion and the quality of the hints. This feedback was informal and is reported only in a qualitative way; a structured survey with numerical ratings is planned for the longer trial. Users mainly liked having the timer, alarms, notes and practice in one place, because they no longer had to switch between apps and could see how much time each topic had taken. The next-topic suggestion was seen as helpful when the list of pending topics was long.

5.5 Discussion

The results suggest that the combined design does what it was meant to do. Since every module refers to the same topic, the system can answer questions that separate tools cannot, such as how the time spent on a topic relates to the acceptance ratio of its problems. Storing sessions, alarms and notes in the database also keeps progress safe between visits. Calculating session length from timestamps and running alarms on the server made both features survive a page refresh or a short loss of connection.

The AI assistant helps in three ways. It gives hints in steps, which keeps the student working on the problem. It turns the planner's ranking into a readable plan for the day. And it shortens long notes into summaries for quick revision. Because it is given the student's own context, its answers were more to the point than those of a general chat assistant that knows nothing about the topic or the last submission. Its quality still depends on the language model behind it, and a hint can sometimes be incomplete or wrong, so students should treat hints as guidance and check the reasoning themselves.

5.6 Comparison with Using Separate Tools

We also compared the workflow of the platform with the usual way of working, in which a coding website, a timer app, a phone alarm and a notes app are used separately. The comparison below is based on the steps a student needs for common tasks and not on measured times, and it should be read as a description of effort, not as a performance claim.

Task	Using separate tools	Using the platform
See time spent on one topic	Add up timer entries by hand or not possible	Shown directly on the dashboard
Decide what to study next	Student decides from memory	Ranked suggestion from stored data
Revise a topic before an exam	Open notes app, find the website, check old attempts	One topic page with notes, problems and history
Set a revision reminder	Open the phone clock; no link to the topic	Alarm linked to the topic
Check weak topics	Look through submission history on the website	Weakness score for every topic

Table 6. Effort for common tasks with and without the platform

5.7 Limitations

The present version has some clear limits. The problem set is small compared with big judges, and the hidden test cases may miss some edge cases. The planner uses fixed weights and does not learn from the student's behaviour. The AI part needs an internet connection and adds some delay. The evaluation involved a small group of users, so we cannot yet make claims about the effect on marks or interview results; a larger and more varied group is needed for that.

5.8 Threats to Validity

Some factors may affect how far the above results can be trusted. The sample problems and topics were chosen by the developers, so they may be easier or better organised than the material a real student would load. The test users were few and were already comfortable with web applications, which may have made the interface look simpler than it is for a beginner. The usefulness of the hints was judged by the users and not checked against a fixed rubric, so different judges might score them differently. Finally, the study was short. A tool of this kind should be judged over a whole semester, because the real question is whether students keep using it and whether their preparation improves.

5.9 Responsible Use

A system that stores study habits and notes holds personal information, and a system that gives AI answers can be misused. We therefore keep the data of each student private by default and send only the minimum context to the AI service. The assistant is instructed to guide and not to hand over finished solutions, because the aim of the platform is learning and not only a green tick on a submission. If the platform is used in a college, teachers should make clear to students what is stored and who can see it.

5.10 Practical Implications

For students, the main benefit is that one tool now holds the whole preparation record. A student preparing for placements can see at any time which topics are done, which are weak and how many hours each has taken, and can plan the coming week with that picture in front of them. For colleges, the same design could help departments run structured practice programmes, since the topic list and the problem bank can be shared among a batch while each student keeps an individual record. Faculty could use the aggregated data, with the permission of the students, to find topics where many students struggle and to spend class time on them.

6. CONCLUSION

In this paper we presented the Student AI Assistant, a web platform that puts coding practice, a study timer, alarms and topic notes in one place and keeps the student's required topics and all activity in a single database. The shared topic model lets the modules work together, and an AI assistant that reads the stored data gives hints, explanations, summaries and topic suggestions. Functional testing showed that the main flows of all modules work as intended, and our discussion explained why having everything together is more convenient than juggling separate apps.

The work also shows that personal guidance does not need a very complex system. A relational database and a simple scoring rule are enough to give a student useful suggestions without a large training dataset. After more testing with a bigger group of students, and with the extensions listed below, we believe the platform can be a practical companion for both semester study and placement preparation.

7. FUTURE SCOPE

There are many ways in which the system can grow. The planner can learn its weights from each student's history, and proper knowledge tracing models [5][6] can be used to estimate how well a topic is understood. Revision alarms can be created automatically using expanding intervals, following the spacing effect [1][3]. The coding module can get more languages, a larger problem bank, contests and an analysis of the time and space complexity of submitted code.

A mobile application with local alarms and offline notes, synchronised with the database when the network returns, would make the tool easier to use every day. Voice commands for the timer, alarms and hints are another useful addition. Richer charts, such as study heat maps, accuracy trends for each topic and weekly goals, would help students see their progress at a glance.

The platform could also support teachers and mentors, who could assign topics and, with the student's permission, follow progress, and it could allow study groups. Flashcards and short quizzes generated from the student's own notes would support active recall [4]. A simple wellbeing feature could notice very long study stretches and suggest a break. Finally, we want to try smaller language models that run on the student's own device, so that notes do not have to leave it.

Another direction is to study the effect of the platform on real results. A longer trial with students of different years, in which one group uses the platform and another group continues with separate tools, would show whether the combined view really improves study habits, practice hours and performance in tests and interviews.

For real deployment in a college, an administrator panel would be needed to add problems, manage users and watch the load on the sandbox and the AI service. Automated database backups and a simple monitoring page should also be added.

ACKNOWLEDGEMENT

The authors thank their project guide, Priyadharshini R, and the Department of Artificial Intelligence and Data Science, Sri Shakthi Institute of Engineering and Technology, Coimbatore, for the guidance and facilities provided for this work, and the students who took part in testing the prototype and shared their feedback.

CONFLICT OF INTEREST

The authors declare that they have no conflict of interest regarding the publication of this paper. No external funding was received for this work.

REFERENCES

[1] H. Ebbinghaus, *Memory: A Contribution to Experimental Psychology*, 1885 (trans. H. A. Ruger and C. E. Bussenius, Teachers College, Columbia University, 1913).

[2] F. Cirillo, *The Pomodoro Technique*, FC Garage, 2006.

[3] N. J. Cepeda, H. Pashler, E. Vul, J. T. Wixted and D. Rohrer, "Distributed practice in verbal recall tasks: A review and quantitative synthesis," *Psychological Bulletin*, vol. 132, no. 3, pp. 354-380, 2006.

[4] H. L. Roediger and J. D. Karpicke, "Test-enhanced learning: Taking memory tests improves long-term retention," *Psychological Science*, vol. 17, no. 3, pp. 249-255, 2006.

[5] A. T. Corbett and J. R. Anderson, "Knowledge tracing: Modeling the acquisition of procedural knowledge," *User Modeling and User-Adapted Interaction*, vol. 4, no. 4, pp. 253-278, 1994.

[6] C. Piech, J. Bassen, J. Huang, S. Ganguli, M. Sahami, L. J. Guibas and J. Sohl-Dickstein, "Deep knowledge tracing," *Advances in Neural Information Processing Systems*, vol. 28, 2015.

[7] T. B. Brown et al., "Language models are few-shot learners," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[8] M. Chen et al., "Evaluating large language models trained on code," arXiv:2107.03374, 2021.

[9] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, University of California, Irvine, 2000.

[10] A. Vaswani et al., "Attention is all you need," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[11] J. Sweller, "Cognitive load during problem solving: Effects on learning," *Cognitive Science*, vol. 12, no. 2, pp. 257-285, 1988.

[12] J. Dunlosky, K. A. Rawson, E. J. Marsh, M. J. Nathan and D. T. Willingham, "Improving students' learning with effective learning techniques: Promising directions from cognitive and educational psychology," *Psychological Science in the Public Interest*, vol. 14, no. 1, pp. 4-58, 2013.

[13] R. Elmasri and S. B. Navathe, *Fundamentals of Database Systems*, 7th ed., Pearson, 2016.

[14] I. Sommerville, *Software Engineering*, 10th ed., Pearson, 2015.

[15] E. L. Bjork and R. A. Bjork, "Making things hard on yourself, but in a good way: Creating desirable difficulties to enhance learning," in *Psychology and the Real World*, Worth Publishers, 2011, pp. 56-64.