GENERATIVE AI

SmartStudy AI: An AI-Assisted Web Platform for Academic Task Planning and Study Guidance Using Flask and Google Gemini

B. Swanika, Bhavadheesh Kapu, Aditya Chidambaram, Ch. Gayathri, Sure Mamatha

Department of Computer Science and Engineering, Hyderabad Institute of Technology and Management, Hyderabad, Telangana, India

Published November 2026

Abstract

Students manage lectures, assignments, examinations and competitive-exam preparation through separate notes, calendars and task lists, which makes it difficult to turn competing deadlines and limited study time into an actionable plan. SmartStudy AI is a web-based prototype that combines academic task management with conversational study guidance in a single application. The system is built with Python and Flask, uses Flask-SQLAlchemy with a MySQL database to persist users, tasks and chat records, and calls Google's Gemini model through the google-genai SDK to produce study schedules from natural-language requests. A blueprint-based architecture separates the authentication, planner, dashboard and chatbot modules, and a dedicated prompt module guides the model toward time-slotted plans that include breaks and an explicit list of tasks that do not fit. Each task stores a title, subject, priority, due date and status, and the dashboard lists the tasks of the signed-in student. During development, a generated study plan was reported to add tasks to the planner, while model-access (404) and service-unavailability (503) errors from the external API were also encountered. The application is intended to support student planning rather than replace academic judgement, and it has not yet been evaluated for learning outcomes. Calendar synchronization, automatic rescheduling and analytics are identified as future work. The prototype shows how a lightweight web stack and a hosted language model can be combined into an accessible study-planning assistant.

Keywords: study planning; task management; educational technology; large language model; Google Gemini; Flask; MySQL; student productivity

1 Introduction

Students rarely work on a single obligation at a time. Lectures, laboratory records, assignments, internal examinations and preparation for competitive examinations all compete for a limited number of study hours. These commitments are usually tracked in separate places, such as note-books, phone calendars and chat messages, so students must decide for themselves what to study first and how to divide their time.

A conventional task list records what must be done but does not help with deciding how to do it. Conversely, a standalone AI chat assistant can suggest a study schedule, but the advice is not connected to the student's saved tasks and disappears once the conversation ends. A platform that keeps task records and AI-generated guidance together can reduce this gap.

SmartStudy AI is proposed to address this need. It is a Flask-based web application in which a student can register, record academic tasks, view them on a dashboard, and ask a Gemini-powered

assistant for a study plan in natural language. The work reported here describes a developing prototype; it does not claim a validated adaptive scheduler.

The main contributions of the proposed system are:

1. A modular Flask web application with separate authentication, planner, dashboard and chatbot components.
2. A relational data model, implemented through Flask-SQLAlchemy on MySQL, for users, tasks, chat history and saved study plans.
3. A Gemini-based study-guidance helper with a centralized prompt module and fallback handling for temporary service unavailability.
4. A task-creation and dashboard workflow in which each task records a subject, a priority, a due date and a status.
5. A documented test plan and an analysis of the limitations and risks of the current build.

2 Related work

2.1 Self-regulated learning and tutoring systems

Zimmerman describes self-regulated learning as a cycle in which learners set goals, manage time, apply learning strategies, evaluate their progress and seek help when needed [1]. This view motivates tools that turn academic goals into scheduled actions. VanLehn's review compares human tutoring, intelligent tutoring systems and other computer-based tutoring approaches and supports the relevance of computer-based learning assistance [2]. SmartStudy AI is not a step-based intelligent tutoring system, and it has not been evaluated for learning gains.

2.2 Academic planning and recommendation

Shao et al. formulate multi-semester degree planning as a recommendation problem using historical course records and future courses of interest [3]. Atalla et al. present an academic-advising recommender that combines curriculum analysis, graph methods and performance modeling [4]. Both works operate on long planning horizons with substantial academic records. SmartStudy AI instead targets short-term, task-level study planning and currently has no predictive model or training dataset.

2.3 Language-model assistance and web frameworks

Hosted language models can generate text from natural-language instructions through an API [5, 9]. The prototype relies on this capability instead of training a custom model. Flask [6], Flask-SQLAlchemy [7], Flask-Login [8], SQLAlchemy [10] and MySQL [11] provide the web, session, object-relational mapping and storage layers. These sources document the technologies used; they do not by themselves demonstrate that the integration works.

2.4 Research gap

Task lists and conversational assistants usually exist as separate tools. A generated answer does not automatically become a validated task, and adaptive scheduling requires reliable completion history, explicit constraints and evaluation. SmartStudy AI addresses the usability gap by placing task persistence and AI guidance in one application. Its contribution is an applied integration prototype and not a new algorithm.

3 Methodology

3.1 System design

The system follows a client-server architecture in which the Flask server renders HTML templates, processes form submissions and communicates with the database and the external language-model service. The architecture, shown in Figure 1, consists of the following components:

1. HTML/Jinja templates and CSS for the user interface.
2. Flask blueprints for authentication, planner, dashboard and chatbot routes.
3. Flask-SQLAlchemy models for users, tasks, chat history and study plans.
4. A MySQL database for persistent storage.
5. A Gemini helper module with a separate prompt module.
6. The Google Gemini API as an external generative service.

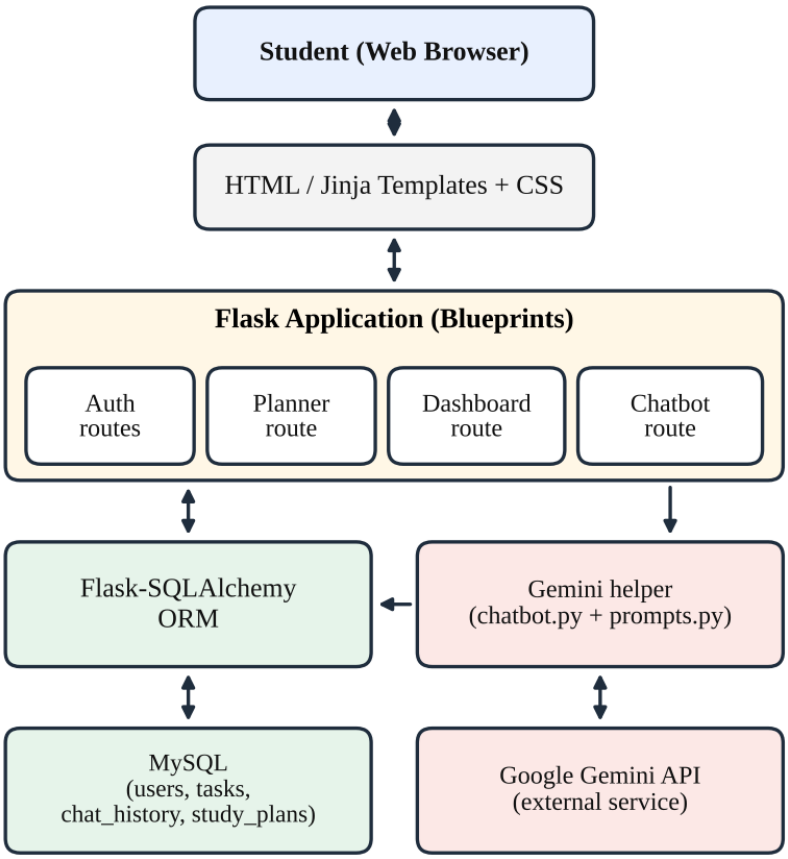


Figure 1. Architecture of SmartStudy AI.

3.2 Data model

The data model is derived from the application models and the development history. Table 1 lists the entities and their known fields. Exact column types, constraints and indexes are defined in the model code and should be read from the database schema.

Table 1. Database entities.

Entity	Known fields	Notes
User	User account fields	Owner of tasks and chat records
Task	id, user_id, title, subject, priority, due_date, status	Belongs to one user
ChatHistory	id, user_id, user_message, ai_response, created_at	Stores each prompt and answer
StudyPlan	Saved-plan fields	Saved-plan workflow in progress

3.3 AI study-guidance module

The chatbot helper creates a google-genai client using an API key that is read from the environment configuration. It combines a system prompt with the student's message and calls the model's content-generation method, returning the stripped response text. The system prompt, stored in a separate module as a Python string, instructs the model to produce a realistic plan with time slots and breaks and to list tasks that do not fit in the available time.

Because the external service can be temporarily unavailable, the helper tries alternative model names when a 503 (UNAVAILABLE) or high-demand error occurs, and re-raises other exceptions so that they are not hidden. Prompt instructions guide the output but cannot guarantee that a plan is correct or feasible.

3.4 Workflow

The workflow, from the student's point of view, is as follows:

1. The student registers and logs in.
2. The student submits a task with a title, subject, priority and due date; the planner route stores it and redirects to the dashboard.
3. The dashboard retrieves the tasks that belong to the current user.
4. The student sends a natural-language request in the chat page, for example a plan for an evening with four available hours for an Operating Systems exam, DAA practice, Python and GATE revision.
5. The helper calls Gemini, the response is displayed, and the exchange can be stored as a chat record.

4 Implementation

4.1 Frontend

The frontend is server-rendered. Jinja templates for the dashboard, planner and chatbot pages are styled with separate CSS files. The dashboard presents the signed-in student's tasks, the planner page provides the task form, and the chatbot page accepts a request and displays the generated response.

4.2 Backend

The backend is implemented in Python with Flask. The application factory loads configuration, initializes the database and the login manager, and registers the route blueprints. Table 2 lists the main endpoints.

Table 2. Main endpoints.

Path	Purpose
/	Home page linking to register, login and dashboard
/register	New-user registration
/login	User login
/dashboard	Lists tasks filtered by the current user
/planner	GET shows the form; POST creates a task and redirects
/chatbot	Sends a prompt to Gemini and shows the reply

4.3 Persistence

Flask-SQLAlchemy maps the models to MySQL tables. The dashboard query filters tasks by the current user's identifier. During development the existing tasks table lacked the subject column that the model expected, which showed that db.create_all() creates missing tables but does not alter existing ones; a backed-up migration is therefore required for schema changes.

4.4 Configuration and security

The Gemini API key and database credentials are kept in environment variables loaded with python-dotenv, and the environment file must be excluded from version control. Flask-Login is initialized for session handling. An inconsistency was found between Flask-Login-protected routes and older raw-session logic with direct password comparison; unifying them, hashing passwords and checking record ownership on every operation are required before any real deployment.

5 Results

5.1 Implementation status

Table 3 summarizes the status of each module according to the evidence available from the development history. No performance, accuracy or learning-effectiveness figures are reported, because none were measured.

Table 3. Module status.

Module	Status
Authentication	Partial; session logic and Flask-Login need unification
Task planner	Task creation logic developed; final retest pending
Dashboard	Per-user task query developed; retest pending
AI chatbot	Gemini helper working in at least one reported run; 404 and 503 errors also seen
Chat history	Model fields defined; saved rows to be verified
Saved plans	Model referenced; workflow to be verified
Calendar, rescheduling, analytics	Planned; not implemented

5.2 Observed behavior

A study-plan prompt submitted through the website was reported to produce a response that added tasks to the planner. This was a single observation, and repeatability and correctness of the added tasks remain to be tested. Errors from the external service were also observed: a 404 error indicating that a configured model was not accessible with the key, and a 503 error indicating high demand.

Other problems were found and addressed during development, including import-path errors when running outside the project root, a function-name mismatch in the chatbot import, a blueprint registration error, an uninitialized login manager, MySQL access errors, the missing subject column, and prompt text that had been pasted into the prompt module as raw code instead of a string.

5.3 Testing status

An 18-case test plan was defined covering startup, registration, login, authorization, task creation, validation, schema, ownership, Gemini success and failure, the prompt module, chat history, task insertion, calendar and rescheduling, secrets, performance and persistence. Table 4 gives the current status.

Table 4. Test plan status.

Status	Test cases
Partially demonstrated (AI response, AI task insertion)	2
Issue observed, retest required (login, schema, Gemini failure, prompt module)	4
Action required (secret exposure check)	1
Not yet executed	11
Total	**18**

Expected behavior in the plan is not a pass result; only executed tests with recorded outputs should be counted as passed. A defined evaluation measuring request success rate, latency, task-field correctness, duplicate-task rate and plan quality against a rubric is left for future work.

6 Discussion

SmartStudy AI shows that task persistence and conversational guidance can be combined with a small, widely used Python stack and a hosted language model, without training a custom model. The modular blueprint structure keeps the planner, dashboard and chatbot concerns separate, which simplifies later additions such as calendar integration.

The system has important limitations. The application depends on an external service whose availability and model access can change. Generated text may be inaccurate, infeasible or inconsistently structured, so any conversion of a plan into tasks must validate fields, dates, time budgets and duplicates. Prompts are sent to an external provider, so students should not enter sensitive personal information. Authentication, schema migration and secret handling need to be completed, and no formal evaluation of usability or learning benefit has been carried out. The expected benefits of easier task capture and centralized visibility are therefore not yet measured.

7 Conclusion

SmartStudy AI is a web-based prototype that brings academic task management and Gemini-based study guidance into one Flask application backed by MySQL. The development history shows a modular design, task-creation and dashboard logic, a Gemini helper with fallback handling, and a reported successful AI-generated plan that added tasks. The evidence supports a prototype-level outcome and not a validated or production-ready adaptive planner.

Future work includes unifying authentication with password hashing and ownership checks, protecting and rotating secrets, stabilizing the Gemini integration with bounded retries and timeouts, adding database migrations, converting generated plans into tasks through structured output with validation and user approval, and completing task edit, delete and completion operations. Further extensions include Google Calendar integration [12] with conflict detection, rescheduling of missed tasks, progress analytics, secure deployment, and a formal usability and effectiveness evaluation.

Acknowledgment

The authors express their sincere gratitude to their project guide for valuable guidance and support throughout the development of SmartStudy AI. They are grateful to the Head of the Department, the Principal and the faculty members of the Department of Computer Science and Engineering at Hyderabad Institute of Technology and Management for providing the academic environment and resources, and thank their friends and family for their encouragement during the design, implementation and testing of the project.

References

[1] Zimmerman BJ. Becoming a self-regulated learner: An overview. *Theory Into Practice*. 2002;41(2):64–70. https://doi.org/10.1207/s15430421tip4102_2

[2] VanLehn K. The relative effectiveness of human tutoring, intelligent tutoring systems, and other tutoring systems. *Educational Psychologist*. 2011;46(4):197–221. https://doi.org/10.1080/00461520.2011.611369

[3] Shao E, Guo S, Pardos ZA. Degree planning with PLAN-BERT: multi-semester recommendation using future courses of interest. *Proceedings of the AAAI Conference on Artificial Intelligence*. 2021;35(17):14920–14929. https://doi.org/10.1609/aaai.v35i17.17751

[4] Atalla S, Daradkeh M, Gawanmeh A, Khalil H, Mansoor W, Miniaoui S, et al. *An intelligent recommendation system for automating academic advising based on curriculum analysis and performance modeling*. Mathematics. 2023;11(5). https://www.mdpi.com/2227-7390/11/5/1098

[5] Google. *Gemini API: generating content*. Google AI for Developers [Internet]. [cited 2026 Oct 25]. Available from: https://ai.google.dev/api/generate-content

[6] Pallets Projects. *Flask documentation* [Internet]. [cited 2026 Oct 25]. Available from: https://flask.palletsprojects.com/

[7] Pallets Projects. *Flask-SQLAlchemy documentation* [Internet]. [cited 2026 Oct 25]. Available from: https://flask-sqlalchemy.palletsprojects.com/

[8] Flask-Login contributors. *Flask-Login documentation* [Internet]. [cited 2026 Oct 25]. Available from: https://flask-login.readthedocs.io/

[9] Google. *Google Gen AI Python SDK. GitHub* [Internet]. [cited 2026 Oct 25]. Available from: https://github.com/googleapis/python-genai

[10] SQLAlchemy authors. *SQLAlchemy documentation* [Internet]. [cited 2026 Oct 25]. Available from: https://docs.sqlalchemy.org/

[11] Oracle. *MySQL reference manual* [Internet]. [cited 2026 Oct 25]. Available from: https://dev.mysql.com/doc/

[12] Google. *Google Calendar API: events resource and methods* [Internet]. [cited 2026 Oct 25]. Available from: https://developers.google.com/workspace/calendar/api/v3/reference/events