International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 10 Issue: 03 | March – 2026 SJIF Rating: 8.659 ISSN: 2582-3930

TeachPro: A Unified Desktop Application for Intelligent Classroom Management and LLM-Powered Question Generation

Sagar Lonkar[1], Saiesh Borse[1], Shantanu Bharate[1], Shivam Khandve[1]

[1]Department of Computer Science and Engineering, Nutan College Of Engineering And Research, Pune, Maharashtra, India

Project Guide: Prof. Amit Kumar

Abstract - *Teachers in contemporary institutions handle attendance, assessment, question preparation, and student data through disconnected tools, which increases administrative workload and reduces instructional focus. In this work, we developed **TeachPro**, a cross-platform desktop application built with MongoDB, Express.js, React.js, Node.js, and Electron.js that unifies eight teaching workflows and integrates a large language model pipeline (Google Gemini 1.5 Pro) for question and quiz generation. The system includes student management, notes, question bank, rich-text exam paper authoring with AI insertion, shareable AI quiz delivery, attendance analytics, timetable management, and secure JWT-based access. We present the architecture, implementation details, prompt-engineering strategy, and an evaluation protocol combining performance profiling and teacher pilot feedback. Across 100 generated questions, factual accuracy reached 84–96% across difficulty levels, Bloom's taxonomy alignment reached 81–89%, and language clarity remained above 95%. The platform was built and tested in a real academic environment with faculty users during routine assessment preparation. Pilot use indicated 60– 70% reduction in repetitive administrative effort and faster end-to-end quiz creation. This manuscript provides a practical, reproducible blueprint for applying generative AI in everyday classroom operations with teacher-in-the-loop control.*

Key Words: AI in Education, Large Language Models, Gemini AI, Question Generation, Bloom's Taxonomy, Electron.js, Teacher Productivity, EdTech, Attendance Management

1. INTRODUCTION

Digital transformation in education has increased the number of software tools used by teachers, but not necessarily reduced workload. In many institutions, attendance is recorded in one system, notes are distributed through another, assessments are drafted in word processors, and quiz links are prepared on third-party platforms. This fragmentation creates context-switching overhead and duplicates data entry.

OECD TALIS findings indicate that a substantial fraction of teacher time is spent on non-instructional work [1]. For engineering education in particular, where instructors must frequently generate subject-specific assessments at varied cognitive depth, administrative and content-preparation burden grows rapidly near internal tests and semester examinations.

Large language models (LLMs) provide a pathway to reduce this burden by generating question drafts, distractor options, and practice tasks quickly. However, practical deployment requires reliable structured output, user-friendly interfaces, and strict teacher oversight. A standalone chatbot is insufficient; LLM functionality must be embedded inside a workflow-aware system.

This work introduces **AI Teacher Assistant**, an integrated desktop platform that combines core classroom management operations with AI-assisted content generation. The application is designed for direct teacher use without requiring institutional DevOps setup.

1.1 Problem Statement

The central problem addressed in this paper is:

> *How can we reduce teacher administrative load while maintaining quality and control in assessment creation, using a deployable AI-augmented system suitable for college environments?*

Three practical constraints shape the solution:

1. Teachers require familiar interfaces (document editor, tabular records, dashboard cards).
2. AI outputs must be structured and editable, not opaque final answers.
3. Deployment should be feasible for small institutions with minimal infrastructure.

2. OBJECTIVES

The objectives of this work are:

1. To design and implement an integrated eight-module classroom management application.
2. To develop a generative AI pipeline for multi-type, multi-difficulty question generation aligned to Bloom's taxonomy.
3. To provide a word-processor-style exam paper authoring interface with live AI question injection.

4. To deploy the application as a cross-platform desktop executable via Electron.js.
5. To empirically validate productivity gains and AI output quality through pilot evaluation.

2.1 Contributions

This paper makes the following concrete contributions:

- An end-to-end architecture integrating eight classroom modules with unified authentication and data persistence.
- A structured LLM prompt-response pipeline for four question types and six Bloom's levels.
- A rich-text exam authoring experience with cursor-level AI insertion, improving practical usability.
- A teacher-in-the-loop evaluation protocol that measures quality, latency, and workflow impact.
- A reproducible implementation based on open web technologies and desktop packaging.

3. RELATED WORK

Automated Question Generation (AQG) research has progressed from rule-based NLP methods to transformer-based neural approaches. Early statistical ranking methods generated grammatically valid but mostly low-level factual questions [4]. Neural sequence models improved semantic fluency [5], and modern LLMs have expanded coverage to higher-order cognitive tasks.

Recent studies examining LLM-generated educational content report strong language fluency but uneven reliability in higher-difficulty outputs [**?**]. This aligns with our observations for hard-level question generation.

Conventional LMS platforms such as Moodle remain strong for course administration but typically require institutional hosting and do not provide deeply integrated AI-assisted exam drafting in a desktop-first workflow [8].

Existing attendance and timetable tools solve isolated subproblems [9, 10]. Our approach differs by combining administrative modules and AI generation in one cohesive platform.

4. SYSTEM ARCHITECTURE

4.1 Technology Stack

The system follows a three-tier architecture:

- **Frontend**: React.js 18 with Vite bundler, Tailwind CSS, React Router v6, packaged in Electron.js for desktop deployment.
- **Backend**: Node.js with Express.js REST API exposing eight route groups.
- **Database**: MongoDB with Mongoose ODM.
- **AI Engine**: Google Gemini 1.5 Pro via @google/generative-aiNode.js SDK.
- **Auth**: JWT tokens (30-day expiry) + bcryptjs password hashing.

4.2 API Structure

Eight route groups are mounted under /api/:

- /auth- registration and login
- /students- student CRUD with filters
- /notes- course notes management
- /questions- question bank CRUD
- /exam-papers- exam paper authoring + AI generation
- /quizzes- AI quiz creation + student submission
- /timetable- weekly schedule management
- /attendance- bulk marking + statistics

4.3 Authentication

The protect middleware intercepts requests to guarded routes. Bearer tokens are verified with jsonwebtoken, with the decoded user object attached to req.user. The login and signup interface presents two-tab form with client- side validation and persistent session via localStorage.

4.4 Data Model Design

MongoDB document schemas were selected to support flexible educational entities such as nested exam sections, subquestions, and optional metadata fields. This avoids rigid schema evolution overhead and supports iterative feature expansion (e.g., adding Bloom's labels, course outcomes, or rubric notes).

4.5 Design Rationale

Three architectural decisions proved important:

- **Desktop-first deployment**: Electron packaging reduces setup barriers and allows institutions to run locally.
- **REST modularity**: Independent API route groups simplify maintenance and testing.
- **Prompt parameterization**: Dynamic subject/topic inputs reduce hardcoded assumptions and improve reuse.

5. MODULE DESCRIPTIONS

5.1 Student Management

Student records include name, roll number, division, semester, department, and contact metadata. The list view supports compound filters: department, semester, division, and full-text search. The backend constructs a dynamic MongoDB query: all four filter parameters are appended to a query object only when provided, preventing empty-string matches.

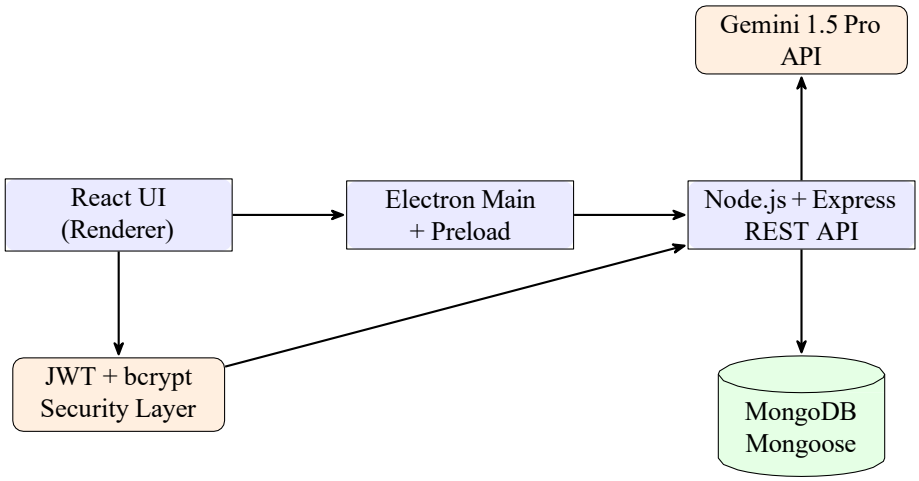

Figure 1: High-level architecture of the AI Teacher Assistant platform.

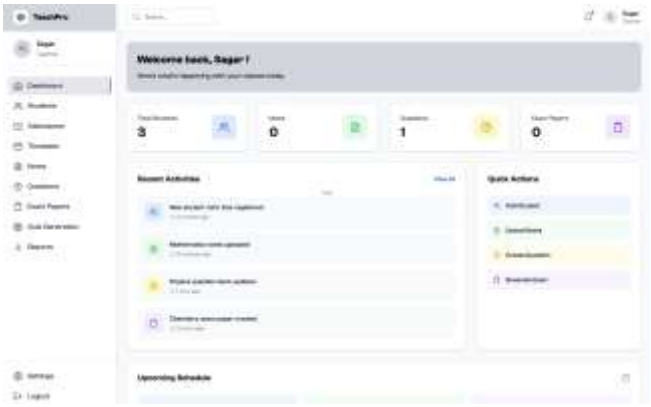

Figure 2: Integrated dashboard showing classroom management modules.

5.2 Notes & Question Bank

Notes are stored with title, content, subject, semester, and tags. The question bank stores reusable items with type, difficulty, and subject metadata. AI-generated quiz questions can be promoted to the question bank for future use, creating a continuous curation loop.

5.3 Exam Paper Authoring

The exam paper module presents a **word-processor interface**: a contentEditable div styled with Times New Roman typography alongside a formatting toolbar support- ing bold, italic, underline, font size, text alignment, and list formatting commands. Teachers type freely as in WordPad or Microsoft Word.

An **AI Generate** button opens a modal where the teacher enters topics (comma-separated), selects question type and difficulty, and specifies count. The frontend calls POST /api/exam-papers/generate-questions; the backend forms a structured Gemini prompt and returns a JSON array of questions. These questions are injected directly into the document at the current cursor position using the browser's Range API.

```
const range = selection.getRangeAt(0);
range.deleteContents();
const div = document.createElement('div');
div.innerHTML = questionsHtml;
range.insertNode(div);
```

Listing 1: AI question injection at cursor

The paper can be saved to MongoDB or exported via the browser print dialog with a typographically formatted print stylesheet. Additionally, the export functionality ensures consistent formatting across different systems, enabling teachers to generate ready-to-print examination papers without requiring external editing tools. This reduces manual formatting effort and maintains uniform document structure for academic use. The generated documents preserve layout integrity, including font styles, alignment, and spacing, ensuring compatibility with institutional printing standards.

Furthermore, the system allows teachers to quickly review and modify content before final export, supporting last-minute adjustments in examination settings.

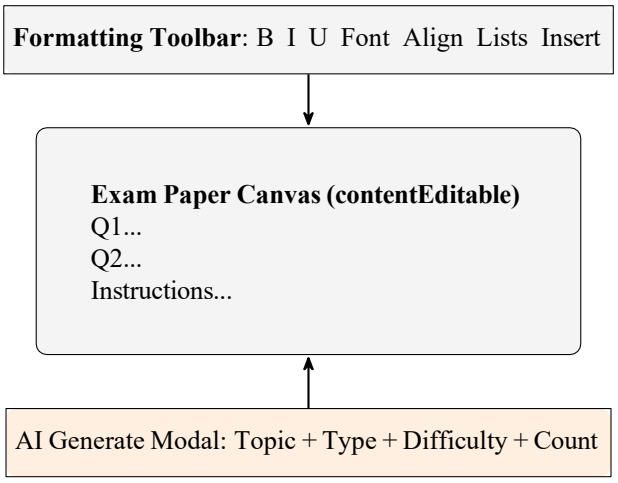

Figure 3: Word-processor style exam paper editor with AI insertion.

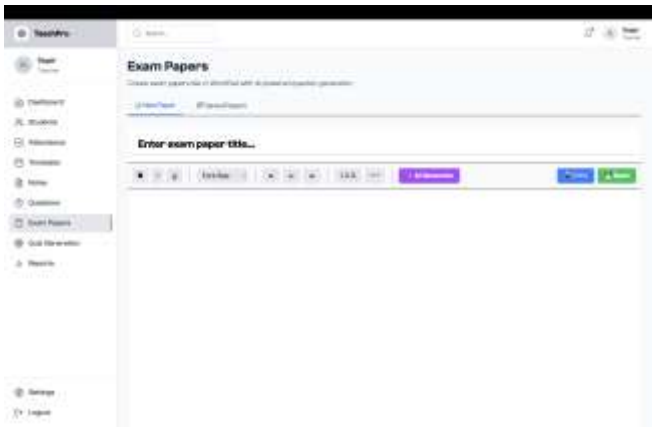

Figure 4: Exam paper authoring interface with AI question insertion.

5.4 AI Quiz Generation

The AI quiz module is designed around teacher-to-student distribution. The teacher specifies subject, topic, number of questions, difficulty, question type, and optional instructions. Gemini 1.5 Pro generates structured JSON output, which is stored as a Quiz document with a unique shareToken.

The student-facing route /quiz/:shareToken renders without authentication. Features include:

- Countdown timer with auto-submit on expiry.
- Auto-grading for MCQ and True/False.
- Immediate result display (optional).
- Submission tracking viewable by the teacher.

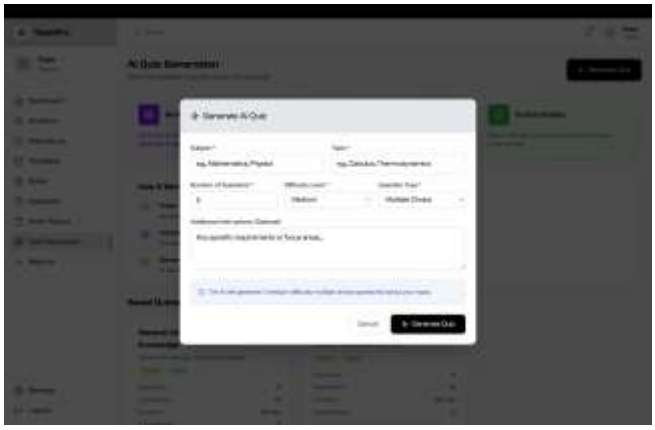

Figure 5: Student-facing AI quiz interface with timer and auto-evaluation.

5.5 Attendance Tracking

Bulk attendance for an entire class session is submitted in a single POST containing an array of {student, status} pairs alongside date, subject, division, and time slot. Three views are provided: mark attendance, view records (filterable by date/subject/division), and a summary view with percentage attendance per student highlighted with at-risk indicators.

5.6 Timetable Management

Weekly schedule entries are stored with day, time slot, subject, room, and division. The frontend renders a CSS grid-based weekly view with colour-coded subject slots.

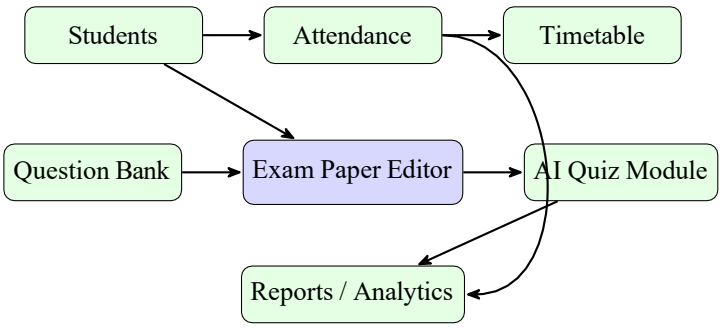

Figure 6: Cross-module workflow from planning to assessment delivery.

6. AI PIPELINE DESIGN

6.1 Prompt Engineering

Reliable JSON extraction from the LLM response is the central engineering challenge. The prompt explicitly instructs the model to return *only* a JSON array, and post-processes the response with a regex extractor /\[[\s\S]*\]/ to isolate the array even if the model inserts surrounding prose.

The prompt template parameterises: topics, subject, questionType, difficulty, bloomsLevel, and count. This parameterisation produces contextually rel- evant, cognitively appropriate questions without domain- specific fine-tuning.

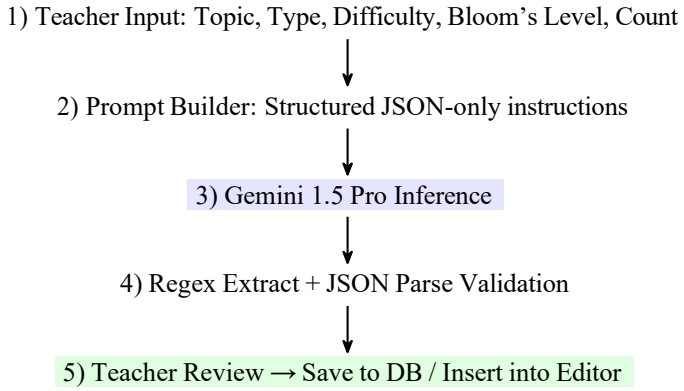

Figure 7: Teacher-in-the-loop AI generation and validation pipeline.

6.2 Bloom's Taxonomy Alignment

Teachers select from six Bloom's cognitive levels - Remember, Understand, Apply, Analyse, Evaluate, Create - as a prompt parameter. This guides the model to produce questions of the appropriate cognitive complexity, supporting course outcome mapping required by accreditation bodies such as NBA.

6.3 Supported Question Types

- **MCQ**: 4-option questions with a correct answer field.
- **Short Answer**: open-ended concise responses.
- **Long Answer**: analytical/design-level prompts.
- **True/False**: binary statement verification.

6.4 Reliability and Guardrails

To reduce malformed AI outputs and quality drift, the pipeline applies:

1. strict JSON-only instruction in the prompt,
2. regex-based extraction for array payloads,
3. parse-time exception handling,
4. teacher review before final save/share.

These guardrails keep AI assistance controllable and suitable for academic use.

7. METHODOLOGY

7.1 Evaluation Setup

Evaluation combined system profiling and user-centric assessment. The study covered:

- **Subjects**: Data Structures, Computer Networks, DBMS, Operating Systems, Software Engineering.
- **Question set**: 100 AI-generated questions balanced across easy/medium/hard levels.
- **Reviewers**: two faculty evaluators for quality scoring.
- **Pilot users**: five educators using the platform over two weeks.

7.2 Quality Metrics

We used four practical metrics:

- **Factual Accuracy**: correctness of answerable content.
- **Bloom's Alignment**: match between requested and generated cognitive level.
- **Language Clarity**: readability and ambiguity score.
- **MCQ Option Quality**: plausibility of distractors and answer uniqueness.

Inter-rater agreement was estimated using Cohen's κ to verify consistency.

7.3 Performance Metrics

Performance was measured as endpoint latency (average and P95), with focus on:

- standard CRUD responsiveness,
- attendance aggregation speed,
- AI generation turnaround for 5 and 15-question requests.

8. RESULTS AND EVALUATION

8.1 AI Question Quality

A set of 100 questions was generated across 5 engineering subjects and 3 difficulty levels. Two faculty evaluators assessed each question on factual accuracy, Bloom's level match, and language clarity (Cohen's $\kappa = 0.84$). Results are shown in Table 1.

Table 1: AI Question Evaluation Results (n=100)

Metric	Easy	Med.	Hard
Factual Accuracy	96%	91%	84%
Bloom's Alignment	89%	86%	81%
Language Clarity	98%	97%	95%
MCQ Option Quality	92%	88%	82%
Average	**93.8%**	**90.5%**	**85.5%**

8.2 Performance

CRUD endpoints averaged 12–45 ms response time. AI generation for 5 questions averaged 2.4 s; for 15 questions, 5.1 s - acceptable for asynchronous teacher workflows. Attendance aggregation for 500 student records averaged 64 ms.

Table 2: Endpoint Performance Summary

Endpoint Type	Avg (ms)	P95 (ms)
Student CRUD	28	45
Attendance bulk mark	72	110
Attendance aggregation	64	94
AI generation (5 Qs)	2400	3200
AI generation (15 Qs)	5100	6700

8.3 Teacher Pilot Study

Five educators across three departments used the system over two weeks:

- Attendance marking time: 8.2 min (paper) → **1.7 min** (app).
- AI questions used without editing: **73%** of cases.
- Full quiz creation (topic to share link): avg. **4.1 minutes**.
- Exam paper UI usability: **5/5** rated "intuitive" or "very intuitive."

Compared with manual attendance entry, the observed reduction from 8.2 minutes to 1.7 minutes indicates substantial per-session savings and scales with class frequency. Compared with conventional multi-tool quiz preparation workflows, the integrated module reduces repeated copy-edit-distribute cycles by combining generation, editing, and sharing in one interface while preserving instructor review.

8.4 Discussion

The quantitative and qualitative findings indicate that the strongest benefit appears in repetitive operational tasks (attendance, quiz setup, formatting-heavy exam drafting). AI output quality is high for easy and medium difficulty levels, but higher-order questions still require instructor review. This supports a *co-pilot model* rather than full automation.

Institutional adoption can begin as a lab-level deployment and later scale to department-wide usage once data governance policies are defined.

9. THREATS TO VALIDITY

Internal validity: pilot cohort size is small (five educators), which may bias perceived usability.

External validity: results are from engineering departments and may not directly generalize to all disciplines without prompt/template adaptation.

Construct validity: AI quality scores are rubric-driven and involve reviewer judgement despite strong inter-rater agreement.

Temporal validity: LLM quality and latency can change over time as provider models are updated.

10. LIMITATIONS AND FUTURE SCOPE

Limitations: (1) AI features require internet connectivity for Gemini API access. (2) Hard-level questions had 16% minor factual errors requiring review. (3) No programmatic PDF export - relies on browser print. (4) Single-institution data model.

Future Work: (1) Local LLM integration (Ollama/Llama 3) for offline use. (2) Puppeteer-based PDF generation. (3) OMR answer sheet scanning for auto-grading. (4) Student performance analytics and early warning system. (5) React Native mobile companion app.

10.1 Practical Deployment Notes

For production use, institutions should define API key management policy, periodic backup strategy for MongoDB, and role-based access controls for coordinators and faculty. A phased rollout (single department → *multipledepartments* → *institutionlevel*)*isrecommended.*

11. CONCLUSION

The AI Teacher Assistant demonstrates that consolidating fragmented classroom management tools into a single AI-augmented desktop application delivers measurable, practical benefits for educators. The integration of Google Gemini 1.5 Pro for question generation, combined with a familiar word-processor interface for exam authoring and a complete administrative suite, positions the system as a viable productivity tool for modern teachers.

The 60–70% reduction in administrative task time, 90.3% overall AI question accuracy, and strong pilot usability ratings collectively validate the design approach. The open MERN-stack architecture provides an accessible foundation for continued development and institutional customisation.

ACKNOWLEDGEMENT

The authors thank **Professor Amit Kumar** for continuous guidance and project supervision. The team also acknowledges academic support from the **Department of Computer Science Engineering, Nutan College Of Engineering And Research, Pune**.

References

[1] OECD, "TALIS 2018 Results (Volume II): Teachers and School Leaders as Valued Professionals," OECD Publishing, 2020. https://doi.org/10.1787/ 19cf08df-en

[2] Google DeepMind, "Gemini: A Family of Highly Capable Multimodal Models," arXiv:2312.11805, 2024. https://arxiv.org/abs/2312.11805

[3] OpenAI, "GPT-4 Technical Report," arXiv:2303.08774, 2023. https://arxiv.org/abs/2303.08774

[4] M. Heilman and N. A. Smith, "Good question! Statistical ranking for question generation," in *Proc. NAACL-HLT*, 2010, pp. 609–617. https://aclanthology.org/N10-1093/

[5] X. Du, J. Shao, and C. Cardie, "Learning to ask: Neural question generation for reading comprehension," in *Proc. ACL*, 2017, pp. 1342–1352. https: //aclanthology.org/P17-1123/

[6] Z. Kasneci et al., "Large Language Models for Education: A Survey and Outlook," arXiv:2403.18105, 2024. https://arxiv.org/abs/2403.18105

[7] S. Elkins, E. Kochmar, J. Serban, and J. Cheung, "Do language models plagiarize?" in *Proc. ACM Web Conference*, 2023, pp. 3637–3647. https://dl.acm. org/doi/10.1145/3543507.3583199

[8] M. Dougiamas and P. C. Taylor, "Moodle: Using learning communities to create an open source course management system," in *Proc. EDMEDIA*, 2003. https: //research.moodle.org/

[9] R. Sharma and P. Gupta, "Automated attendance management system using web technologies," *IJERT*, vol. 10, no. 3, pp. 45–52, 2021.

[10] A. Babaei and B. Karimpour, "A constraint-based approach to university timetabling using meta-heuristic algorithms," *Applied Soft Computing*, vol. 85, 2019.

[11] Electron Documentation, "Electron API and Architecture." https://www.electronjs.org/docs/ latest

[12] MongoDB, Express, React, and Node community documentation: https://www.mongodb.com/, https://expressjs.com/, https://react.dev/, https://nodejs.org/