Edu-Platform: Design and Development of a MERN-Based Personalized Learning Dashboard with Role-Based Access Control and Real-Time Progress Tracking

Prasad Maruti Mahind[1], Varsha Appaso olekar[2], Pravin Khanderao Rathod[3], Aditya Laxman Patil[4], Umesh A. Patil[5]

[1,2,3,4] *Department of Computer Science and Engineering, D.Y. Patil Technical Campus Faculty College of Engineering, Talsande, India*

[5] *Assistant. Professor. Department of Computer Science and Engineering, D.Y. Patil Technical Campus Faculty College of Engineering, Talsande, India*

***Abstract*—When we first started this project, we spent a few weeks using existing platforms as regular students. Most of us gave up halfway through—not because the content was bad, but because there was no real sense of direction. None of the platforms we tested tell a student clearly: this is where you are, this is what comes next, this is how far you have left to go. So we built Edu-Platform—a MERN stack application (MongoDB, Express.js, React.js, Node.js) that gives teachers a proper path builder, gives students a progress dashboard that updates in real time, and gives institutions an admin approval step so content does not go live until reviewed. We also added Gemini API integration for AI-assisted module descriptions after reading VanLehn's work on automated feedback systems producing tutoring-level gains. A four-week pilot at D.Y. Patil Technical Campus with 48 participants yielded a SUS score of 84.5 (Grade A). Course completion in our group was 87.3% versus 50.5% in the control group—a 72.9% improvement that, frankly, surprised us.**

***Index Terms*—MERN stack, learning management system, personalized learning paths, role-based access control, real-time progress tracking, online education platform, e-learning, React.js, MongoDB, blended learning**

I. INTRODUCTION

Something changed in how people learn over the last ten years—and it is not just that more content moved online. The bigger shift is that students started expecting their learning environment to work the way their other apps do: responsively, with real feedback, and with a clear sense of where they are and what comes next. Most learning management systems have not kept up with that expectation. They were built in an era when putting a PDF online counted as digital education.

When we started looking seriously at what was already out there, a few things stood out. Coursera and Udemy are enormous platforms but are fundamentally open marketplaces—an institution using them has almost no control over how a student moves through material, and there is no meaningful approval process for who gets to teach what [2]. Google Classroom sits at the other end: it is institution-linked but essentially a fancier version of emailing homework—there is no concept of a learning path [3]. Moodle technically has all of this, but configuring a Moodle instance from scratch is not a pleasant experience, especially for smaller institutions without a dedicated IT team [5].

One study really shaped how we thought about this project. Chirikov et al. [1] ran a proper randomised controlled trial across three universities, putting over 300 students into in-person, blended, or fully online groups, and found that learning outcomes were statistically the same across all three. Online was just as good as face-to-face, and it cost up to 80.9% less per student. But—and this is the part that mattered most for us—they were upfront that their test platform had no automated feedback, no structured paths, and no progress visibility. That is where Edu-Platform started.

We built this as our final-year capstone at D.Y. Patil Technical Campus, Talsande. The core of the system is three separate role-based dashboards—one for admins, one for teachers, one for learners—each built specifically for what that user type needs to do, not just as cosmetic variations of the same interface. The key contributions of this work are:

- Role-specific dashboards (Admin, Teacher, Learner) with completely separate interfaces and security boundaries, rather than show/hide logic over a single UI.
- A sequential path builder where teachers drag modules into order and the system enforces that sequence—a student on module 2 cannot open module 4.
- Real-time progress tracking: when a student marks a module done, the progress ring and the teacher's per-student row update simultaneously without page refresh.
- An admin approval workflow before any teacher can publish content, preventing quality degradation of the course catalogue.
- Gemini API integration for AI-assisted module descriptions, reducing teacher content-setup time by 64%.
- A four-week deployment with real undergraduate courses and 48 participants, producing a SUS score of 84.5 and a 72.9% improvement in course completion rate.

Section 2 reviews related work. Section 3 describes system architecture and design. Section 4 covers implementation. Section 5 reports pilot results. Section 6 discusses findings, and Section 7 concludes with future directions.

II. LITERATURE REVIEW

We approached the literature review with a specific question: what do we actually know about what makes online learning platforms work, and where have existing systems failed to act on that knowledge?

2.1. Empirical Evidence on Online Education Efficacy

The Chirikov et al. [1] study was the most directly relevant finding. Across three universities in a randomised trial, over 300 engineering students were split across in-person, blended, and fully online delivery. Final exam scores showed no statistically significant difference between groups. Online worked as well as sitting in a classroom at a fraction of the cost—up to 80.9% lower per-student instructor cost for the online group. Critically, the researchers explicitly excluded automated formative feedback and structured paths, flagging this as a limitation and design brief for future work.

Kizilcec and Halawa [8] provided the other side of the picture: even when online learning works in theory, students drop out at alarming rates in unstructured environments. Their analysis showed the biggest predictor of dropout was not content difficulty—it was not knowing where you were in the course or what to do next. Littlejohn et al. [9] and Ryan and Deci

[10] backed this up from a motivational angle: progress visibility and a sense of competence are what keep people going.

2.2. Review of Existing Platforms

Coursera is polished and has genuinely good content from reputable universities, but the moment you try to use it as an institutional tool—to say 'our second-year students will follow this exact path through this material'—you hit a wall. Enrollment is open, sequencing is optional, and educator analytics are thin [2]. Udemy is worse in that respect: it is a course store with no learner-management layer whatsoever.

Google Classroom does the communication-and-assignment loop quite well, but calling it a learning management system feels like a stretch—there is no path structure, no completion tracking beyond grade submission, and no way to visualise whether students are engaging with material between assignments [3]. Moodle is the most deployed open-source LMS on the planet and comes closest to what we wanted. The problem is the interface: setting up a well-structured course in Moodle takes time and technical knowledge that a lot of teaching staff simply do not have [5].

2.3. Adaptive and Intelligent Learning Systems

VanLehn's 2011 review [4] is the paper everyone in this space cites, and it earns the citation. He pulled together decades of tutoring research and found that automated tutoring systems—the kind that detect what a student is struggling with and respond in real time—produce learning gains approaching one-on-one human tutoring. The caveat is cost and complexity. Full intelligent tutoring systems are serious engineering projects. VanLehn's work does suggest

that even partial automation of the feedback loop has measurable value, which pushed us toward using the Gemini API for content assistance. Jaggars and Xu [13] added that structured content sequencing and timely feedback were the two strongest predictors of student performance—more predictive than instructor identity or subject matter.

2.4. MERN Stack in Educational Applications

We picked MERN for reasons that were partly technical and partly practical. React.js was the obvious choice for the frontend because of how it handles state—when a student finishes a module and the progress percentage updates, React re-renders just the component that changed, not the whole page. Node.js with Express gave us a non-blocking backend that could handle several concurrent requests without slowing down—during our pilot, 48 people were sometimes using the platform simultaneously, and we never saw the queuing issues that arise with synchronous server frameworks. MongoDB made sense for the data model because our content structures are not uniform: forcing varied module types into relational tables would have been painful. We could not find existing literature that combined institutional governance, modular path management, real-time progress tracking, and AI-assisted content in one MERN application.

2.5. Summary of Research Gaps

After this review, we identified the following gaps not addressed by existing platforms:

- No platform provides proper learning-path sequencing for institutional use that is also easy to set up and maintain.
- Progress data exists on most platforms but is buried or useless. A teacher should see in one glance which students are behind and on which module.
- Teacher-student interaction is treated as a communication feature, not a learning feature. It should be tied to content—which module the student is stuck on, not just 'any questions?'
- Admin approval workflows are essentially absent from open platforms, leading to low-quality content polluting course catalogues.
- Content is spread across Drive, WhatsApp, email, and YouTube links. There is no single place for a teacher to organise everything accessibly.

Table 1: Comparative analysis of related work and identified research gaps.

Platform / Study	Key Contribution	Limitation	Addressed by Edu- Platform
Chirikov et al. [1]	RCT: online = in-person outcomes; 80.9% cost saving	No adaptive features or structured paths tested	Structured paths, progress tracking, AI assistance
Coursera / Udemy [2]	Large-scale MOOC delivery	No institution-controlled paths or admin governance	Role governance, institution-level control
Google Classroom [3]	Assignment management	No learning paths, no progress analytics	Path-based learning, visual analytics dashboard
VanLehn [4]	ITS achieves tutoring- level gains	High build cost; not web-integrated	Lightweight AI assistance via Gemini API
Moodle / Canvas [5,12]	Full LMS feature set with roles	Legacy architecture, high config overhead	Reactive MERN UI, lower setup barrier
Kizilcec & Halawa [8]	Attrition in unstructured MOOCs	No guided path or progress visibility	Sequential module delivery, visual progress ring

III. SYSTEM ARCHITECTURE AND DESIGN

About midway through development, our MongoDB Atlas free tier hit its storage limit two weeks before the pilot. We migrated to a DigitalOcean droplet in half a day—updating the connection string and one config file was all it took. This was only possible because none of our database queries sat directly inside route handlers. All database logic lived in a service layer. Swap what sits behind that layer and the routes do not notice. The three-layer split (React frontend, Express backend, MongoDB) paid for itself

in that one afternoon.

3.1. Overall Architecture

One decision we debated: build one frontend with show/hide role logic, or three genuinely separate interfaces? The single- frontend approach is faster initially. But we had seen it go wrong—you end up with conditional rendering scattered everywhere, and the security model becomes uncomfortable: if the admin panel is a hidden div, someone with browser dev tools can unhide it. We built three separate dashboard apps sharing one backend. More work upfront, but we never had a security boundary question we could not answer clearly.

Anyone can browse the catalogue without logging in—a deliberate choice so students can see available courses before registering. Login and registration both go through the same JWT flow. Once authenticated, the token comes back with the user's role baked into the payload, and the frontend reads that to decide which dashboard to load. File uploads never land on our server—Multer grabs the file from the request and pipes it straight to Cloudinary's upload stream. The Gemini API key lives only on the backend, preventing accidental exposure.

3.2. Role-Based Access Control (RBAC)

Every protected API endpoint checks the JWT and the role claim independently of whatever the frontend did or did not do. The UI is not a security boundary—it is a convenience layer. We kept the admin role narrow on purpose: they approve or reject teacher applications, can remove content, and see platform-wide statistics. They cannot create or edit learning paths. Mixing content creation with moderation creates ambiguity about responsibility.

Teachers get full control over their own paths once approved: building modules, uploading files, reordering content, and seeing how each of their students is progressing. Crucially, they only see their own data—a teacher cannot browse another teacher's paths or see students not enrolled in their courses. Learners browse, enrol, go through modules in the set order, and check their own dashboard to see how far along they are.

3.3. Data Model

Everything is stored in MongoDB using Mongoose schemas. The User collection handles all three roles through a single document structure—the role field drives which additional fields matter. A learner document has an enrolled Paths array; each element tracks path Id, completed Modules, total Modules, progress Percent, and last Accessed. A teacher document carries status (pending, approved, or rejected), certificates, qualifications, institute, expertise, created Paths, enrolled Students, and revenue. Table 2 covers the key collections.

Table 2: MongoDB collection schema overview.

Collection	Key Fields	Description
Users	_id, name, email, role, status, createdAt	One collection for all roles. The status field drives the teacher approval workflow—pending by default, flipped by admin action.
LearningPaths	_id, title, teacherId, content[], category, level, duration, learners	Content is an ordered array of module objects. The learners array grows as students enrol.
Modules	_id, pathId, title, type, resourceUrl, videoUrl, duration, order	Type is video, pdf, or notes. ResourceUrl is the Cloudinary URL. The order field enforces the sequence.
Enrollments	_id, learnerId, pathId, progress[], completedAt	Progress array grows as modules are completed. Each entry is timestamped for time-on-task calculation.
Analytics	_id, userId, pathId, timeSpent, lastAccessed	Feeds the time-spent chart on the learner dashboard. Written to on every module access.

3.4. System Workflow

The flow through the system in normal operation:

- A teacher submits a registration. Their account is created with status: pending. Nothing else happens until the admin approves or rejects the application.

- The teacher opens the path creation wizard, adds modules (each with type, title, AI-generated description, and file upload), orders them via drag-and-drop, and publishes.
- A student finds the path in the public catalogue, clicks enrol, and their enrollment document is created. They begin on module one.
- There is no way to jump to module three without completing one and two—the frontend enforces this and the backend validates it.
- Each time a student marks a module complete, progressPercent recalculates and both the student's progress ring and the teacher's per-student row update in real time via React state—no refresh needed.
- The teacher can open any of their paths and see a table of enrolled students, how far each has gotten, and when they last accessed the path.

IV. IMPLEMENTATION AND TECHNOLOGY STACK

4.1. Frontend: React.js with Tailwind CSS

React.js v18, functional components throughout. The useState, useEffect, and useContext hooks handled nearly all state management. We did not bring in Redux—the application is not large enough to need a global state store, and adding Redux would have introduced more boilerplate than it solved problems. Tailwind CSS was a good call: we were three developers working on the same frontend, and utility-first CSS meant we were not constantly stepping on each other's stylesheets or arguing about class naming conventions. Responsive breakpoints are built into Tailwind's utilities, so making the dashboards work on mobile did not require a separate pass.

The Learner Dashboard's SVG progress ring is the feature we are most pleased with, design-wise. It is a custom SVG circle with a stroke-dashoffset that updates as progressPercent changes—simple to implement, but visually clear in a way a progress bar is not. The Recharts time-spent chart next to it gives students a weekly view of their engagement. The Teacher Dashboard's path creation wizard was the most complex frontend piece—multi-step form, drag-and-drop module reordering, file upload with a real-time progress bar via SSE, and the Gemini API trigger on the description field.

4.2. Backend: Node.js and Express.js

Node.js v18 LTS, Express.js v4. We organised routes by feature—/api/auth, /api/paths, /api/modules, /api/enrollments, /api/admin, /api/analytics—rather than by HTTP verb, because feature-based organisation makes it much easier to add new middleware to a specific resource without touching unrelated routes. Each route module has its own middleware stack: JWT verification, role check, input validation via express-validator, then the handler. Middleware are written as composable functions so they can be mixed and matched: a teacher-only route gets authMiddleware and requireTeacher; an admin route gets authMiddleware and requireAdmin.

The Multer-to-Cloudinary pipeline was trickier than expected. The naive approach—save to disk with Multer, upload to Cloudinary, delete the local file—works but is slow and leaves orphaned files if the Cloudinary upload fails. We switched to streaming: Multer's memoryStorage keeps the file in a Buffer, which we pipe directly into Cloudinary's upload_stream. No disk writes, no cleanup step, and the upload fails atomically if anything goes wrong. Upload progress is pushed back to the React frontend via a server-sent event stream.

4.3. Database: MongoDB with Mongoose

MongoDB Atlas, AWS ap-south-1. We picked that region because our pilot users were all in India and latency to the nearest Atlas cluster made a measurable difference in dashboard load times during early testing. Mongoose provides schema validation at the application layer, catching accidental writes that are otherwise very hard to debug. We defined required fields and type constraints on every collection.

Two indexes matter most for performance. Atlas Search on LearningPaths.title and LearningPaths.tags powers the course search—without it, a text search is a full collection scan. The compound index on (learnerId, pathId) in Enrollments is the one hit on every module completion event. During JMeter load testing at 100 concurrent users, queries against that index averaged 38 ms—acceptable for the real-time update pattern we needed.

4.4. Authentication and Security

Dual-token JWT: 15-minute access token kept in

memory (not localStorage—localStorage is readable by any script on the page), 7-day refresh token in an HTTP-only cookie. The short access token window means a stolen token is valid for at most a quarter of an hour. The HTTP-only cookie for the refresh token means an XSS vulnerability cannot read it at all.

Beyond tokens: Helmet.js sets Content-Security-Policy, X-Frame-Options, and X-Content-Type-Options headers with sensible defaults. express-rate-limit on auth endpoints caps login attempts at 10 per 15 minutes per IP. express-mongo- sanitize strips MongoDB operator characters from all request bodies before they touch the database—the primary defence against NoSQL injection. CORS is restricted to the exact Vercel deployment URL. We tested all of this manually using Burp Suite during the final week before the pilot.

4.5. Module-Wise Feature Breakdown

During development we divided responsibility across six modules:

- Registration, login, JWT issuance and refresh, OTP-based password reset. Tested aggressively before moving on because everything else depends on it.Authentication Module:
- Teacher approval queue, user listing with deactivation, content moderation screen, and platform-wide statistics dashboard.Admin Module:
- Path creation wizard, module editor with drag-and-drop reordering, Cloudinary upload with live progress, Gemini API trigger, and per-path analytics with individual student progress rows.Teacher Module:
- Public path catalogue, enrolment flow, sequential module viewer that blocks access to later modules until earlier ones are complete, and the personal progress dashboard with SVG ring, module checklist, skill badges, and Recharts chart.Learner Module:
- Backend CRUD for paths and modules, Cloudinary pipeline, ordering logic, and the Gemini API call handler.Learning Path Management Module:
- Writes to Enrollment and Analytics collections on module completion, recalculates progressPercent, and returns the updated value to the frontend without a page reload.Progress Tracking Module:

4.6. Third-Party Integrations

Table 3: Third-party services and integrations.

Service	Technology	Purpose
Media Storage	Cloudinary SDK	All video, PDF, and image uploads. CDN delivery keeps load times fast regardless of student location. Automatic thumbnail generation.
AI Assistance	Gemini API (Google)	Called from the backend when a teacher clicks AI assist. Returns a draft module description and suggested learning objectives. Teachers edit the draft.
Version Control	GitHub / Git	Branch-per-feature throughout. Every feature went through a pull request before merging to main.
Deployment	Vercel (frontend) / Render (backend)	Both connected to the GitHub main branch. A push triggers an automatic deploy. Vercel handles the React build; Render runs the Node.js server as a persistent web service.
Database Hosting	MongoDB Atlas	Managed cluster on AWS ap-south-1. Automated backups ran daily.

V. EXPERIMENTAL RESULTS AND EVALUATION

The pilot ran for four weeks at D.Y. Patil Technical Campus, Talsande, using real undergraduate courses, real students, and real grades. Not a demo, not a usability lab—an actual deployment.

5.1. Experimental Setup

We had 48 participants: 38 undergraduate students across two CSE courses, and 10 faculty members teaching those courses. The 10 teachers built 6 learning paths containing 47 modules between them. For the control group, we took 40 different students from the same courses who used the existing setup:

shared Google Drive folders for notes and recordings, WhatsApp groups for communication. Both groups were assessed on the same course material at the end of four weeks. We collected completion data from the Edu-Platform database, a manual log kept by the control group's teachers, SUS questionnaire responses, satisfaction ratings, and notes from post-pilot teacher interviews.

5.2. Usability Evaluation (SUS)

All 48 participants filled in a SUS questionnaire at the end of week four. Mean score: 84.5, standard deviation 6.2. To put that in context: Bangor et al.'s [15] rating scale marks anything above 80.3 as Grade A (Excellent). Moodle typically scores between 68 and 76 in studies with comparable student populations [5]. We came in about 8 points higher than Moodle's upper range. The two items with the highest individual scores were the role-specific dashboard design and the SVG progress ring, both averaging 4.4 out of 5.

5.3. Learning Outcomes and Engagement

*Table 4: Performance metrics — Edu-Platform vs. conventional methods. (*Estimated time for manual file organisation in control group.)*

Metric	Edu-Platform	Control Group	Change
Course Completion Rate	87.3%	50.5%	+72.9%
Avg. Time-to-Complete (days)	18.4	26.1	-29.5%
Learner Satisfaction (1–5)	4.3	3.1	+38.7%
Instructor Content Setup Time (min)	12.6	~35.0*	-64.0%
SUS Score	84.5 (Grade A)	N/A	Excellent
API Avg. Response Time (ms)	142 ms	N/A	< 200 ms threshold
Concurrent Load (100 users)	99.2% success rate	N/A	Sub-200 ms maintained

The 72.9% completion gap is the headline number. When we broke down the control group's data week by week, the drop-off happened almost entirely in week two—the midpoint of the course. That is a well-known pattern in unstructured online learning: students engage initially, lose track of where they are, and quietly stop. The Edu-Platform group did not show that dip because the next step was always explicit. Students did not have to figure out what to do next—they just looked at their dashboard. The instructor content setup time dropping from approximately 35 minutes to 12.6 minutes was also larger than we predicted, and came up repeatedly in teacher interviews.

5.4. Performance Testing

JMeter load tests at 10, 50, 100, and 200 concurrent users. At 100—more than double our actual pilot size—mean API response time was 142 ms with 99.2% of requests completing successfully. Server memory peaked below 512 MB. The enrollment progress endpoint averaged 38 ms against the compound index. At 200 users, response time climbed to approximately 310 ms and success rate dipped to 97.1%, which is where horizontal scaling or caching would be needed for significant growth beyond the pilot size.

5.5. Qualitative Feedback

We interviewed 8 of the 10 teachers after the pilot ended. Three themes came up across nearly every interview. First, time: several teachers mentioned spending 30 to 45 minutes per course just organising materials before the pilot. With the creation wizard, they said the same task took 10 to 15 minutes because everything lived in one place and the ordering was visual.

Second, specificity: before the platform, office hours were open-ended questions about 'this week's content.' After, teachers could open their dashboard before the session and see exactly which students had not completed module 3 and how many had started module 4. One teacher described it as going from guessing to knowing.

Third, the approval workflow: three teachers independently mentioned that submitting qualifications for admin review made the platform feel different from just posting to a shared folder. One put it directly: it felt like their course material was being taken seriously.

Student feedback focused almost entirely on the interface. More than half the students mentioned the progress ring without being prompted—in several cases in the same breath as saying they had finished a course they would normally have abandoned. A few noted that knowing their teacher could see their progress made them less likely to skip modules, an accountability effect we did not explicitly design for.

VI. DISCUSSION

The 72.9% completion rate improvement is a striking number, but we want to be careful about what it actually means. It does not mean Edu-Platform is 72.9% better than not having a platform—it means that, under the conditions of our pilot, structured sequential delivery with visible progress produced a much higher completion rate than unstructured Drive-and- WhatsApp delivery. The comparison is real but not perfectly controlled.

What we think is actually driving the number is the combination of two things: clarity about what to do next, and visibility into how far you have come. Kizilcec and Halawa [8] showed that disengagement in unstructured MOOCs peaks when students lose track of their position. Ryan and Deci [10] showed that a sense of progress and competence are among the strongest intrinsic motivators. We built for both—locked sequential modules for clarity, progress ring and badges for visible advancement—and the data suggests they worked.

The Gemini API is worth discussing separately. It worked well for general descriptive writing—module introductions for topics like web development, databases, and software engineering came back usable most of the time. But two teachers building paths on more specialised topics—embedded systems and structural engineering—found the output too generic to use without substantial rewriting. This is not surprising; general-purpose language models are not deep in narrow domains. The fix is probably domain-specific prompting with examples, which we did not have time to build properly.

The limitations are real. Forty-eight participants at one institution is a small sample. The control condition—Drive folders and WhatsApp—is weaker than a properly configured Moodle deployment, so the gap might narrow in a fairer comparison. We have four weeks of data, not a semester. And we cannot separate the effect of the platform's design from the novelty effect of using something new—some of the engagement might have been curiosity rather than genuine structural benefit. Replicating this at multiple institutions over a longer period is the only way to answer those questions.

VII. CONCLUSION AND FUTURE WORK

We built Edu-Platform because the platforms that already exist were not doing the things we thought mattered most: giving students a clear path through material, letting teachers see who was stuck and where, and giving institutions control over who published course content. Whether we succeeded is reasonably well supported by the pilot data—84.5 on SUS, 87.3% completion rate, a 64% reduction in instructor content setup time.

But a four-week pilot with 48 people at one institution is a starting point, not a conclusion. The system works. Whether it scales, whether the effects persist over longer courses, and whether the completion improvements hold up against a proper LMS baseline rather than Drive folders—those are open questions that need larger studies to answer.

Planned future work:

- Use a lightweight ML model to suggest the next module based on where a student is spending time and where they are falling behind, so the path adapts rather than just enforcing order [4].Adaptive module recommendations:
- Multi-tenancy to let different institutions share infrastructure while keeping their data and user bases completely isolated.Multi-institution architecture:
- Export standardised learning records to external analytics tools or research databases, enabling longitudinal outcome studies.xAPI integration:
- Module-level comment threads and peer review assignments built on WebSockets. Jaggars and Xu [13] found structured peer interaction to be one of the strongest predictors of online course outcomes.Discussion and peer review:

- A React Native client for iOS and Android, with progress sync across devices.Mobile app:
- Streaks, leaderboards by learning path, and more granular badge categories, building on the motivational gains already observed from the simple progress ring [10].Richer gamification:

ACKNOWLEDGEMENTS

The authors thank the students and faculty of the Department of Computer Science and Engineering at D.Y. Patil Technical Campus, Talsande, for giving their time, using the platform for real coursework, and providing honest feedback that made the final system considerably better than the first version they saw. We also thank our project supervisor for pushing us to be more rigorous about evaluation than we would have been on our own.

REFERENCES

[1] Chirikov, I., Semenova, T., Maloshonok, N., Bettinger, E., Kizilcec, R.F.: Online education platforms scale college STEM instruction with equivalent learning outcomes at lower cost. Science Advances 6, eaay5324 (2020). https://doi.org/10.1126/sciadv.aay5324

[2] Shah, D.: By the numbers: MOOCs in 2020. Class Central (2020). https://www.classcentral.com/report/mooc-stats-2020/

[3] Iftakhar, S.: Google classroom: what works and how? Journal of Education and Social Sciences 3(1), 12–18 (2016).

[4] VanLehn, K.: The relative effectiveness of human tutoring, intelligent tutoring systems, and other tutoring systems. Educational Psychologist 46, 197–221 (2011).

[5] Cavus, N.: Distance learning and learning management systems. Procedia – Social and Behavioral Sciences 191, 872–877 (2015).

[6] Bowen, W.G., Chingos, M.M., Lack, K.A., Nygren, T.I.: Interactive learning online at public universities: Evidence from a six-campus randomized trial. Journal of Policy Analysis and Management 33, 94–111 (2014).

[7] Figlio, D., Rush, M., Yin, L.: Is it live or is it Internet? Experimental estimates of the effects of online instruction on student learning. Journal of Labor Economics 31, 763–784 (2013).

[8] Kizilcec, R.F., Halawa, S.: Attrition and achievement gaps in online learning. In: Proceedings of the ACM Conference on Learning @ Scale, pp. 57–66. ACM (2015).

[9] Littlejohn, A., Hood, N., Milligan, C., Mustain, P.: Learning in MOOCs: Motivations and self-regulated learning. Internet and Higher Education 29, 40–48 (2016).

[10] Ryan, R.M., Deci, E.L.: Intrinsic and extrinsic motivations: Classic definitions and new directions. Contemporary Educational Psychology 25, 54–67 (2000).

[11] Deming, D.J., Goldin, C., Katz, L.F., Yuchtman, N.: Can online learning bend the higher education cost curve? American Economic Review 105, 496–501 (2015).

[12] Instructure Inc.: Canvas LMS Technical Documentation. https://canvas.instructure.com/doc/api/ (2023).

[13] Jaggars, S.S., Xu, D.: How do online course design features influence student performance? Computers & Education 95, 270– 284 (2016).

[14] Brooke, J.: SUS: A quick and dirty usability scale. In: Jordan, P.W. et al. (eds.) Usability Evaluation in Industry, pp. 189–194. Taylor and Francis, London (1996).

[15] Bangor, A., Kortum, P., Miller, J.: Determining what individual SUS scores mean: Adding an adjective rating scale. Journal of Usability Studies 4(3), 114–123 (2009).

[16] Zimmerman, B.J.: Self-efficacy: An essential motive to learn. Contemporary Educational Psychology 25, 82–91 (2000).