International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 10 Issue: 04 | April - 2026 SJIF Rating: 8.659 ISSN: 2582-3930

Automated Presentation Synthesis Using Gen AI

Atharva Dahibavkar
Department of AI & Data Science
New Horizon Institute of Technology
Mumbai, India
atharvadahibhavkar226@nhitm.ac.in

Pavan Shinde
Department of AI & Data Science
New Horizon Institute of Technology
Mumbai, India
pavanshinde226@nhitm.ac.in

Abstract—This paper presents Automated Presentation Synthesis Using Gen AI, a web-based system that automatically generates structured PowerPoint presentations from three simple inputs: a topic, a target audience, and a tone. Built using React, TypeScript, Vite, and Tailwind CSS on the frontend, the system uses PptxGenJS to produce downloadable .pptx files entirely in the browser without involving any server in the export step. The backend is written in Motoko and deployed as a canister on the Internet Computer Protocol (ICP), a decentralised blockchain-based computing platform by DFINITY Foundation, which means there is no traditional cloud server in the stack. TanStack Query handles data fetching and caching between the frontend and the canister. JWT-based authentication secures all routes, with role-based access control separating regular users from admins. We tested the system across ten topic and audience combinations and confirmed that the preview and the downloaded .pptx file are identical in every case. Generation took between 1.2 and 2 seconds on average, and the export was nearly instant since it runs locally. The system shows that an AI-assisted productivity tool can be built and deployed entirely on decentralised infrastructure without sacrificing usability or performance.

Index Terms—*ICP, Internet Computer Protocol, Motoko, Generative AI, automated presentation synthesis, PptxGenJS, MERN stack, JWT authentication, decentralised web application*

I. INTRODUCTION

Creating a presentation from scratch takes a lot of time. Whether it is for a college project, a business pitch, or a workshop, most people spend hours just deciding how to structure their content, what to write on each slide, and how to format it all. Tools like PowerPoint and Google Slides give you a blank canvas, but figuring out what to put on it is still entirely up to you.

This is the problem our project tries to solve. We built a web-based system where a user simply types a topic, selects a target audience, and picks a tone, and the system automatically generates a full, structured presentation. The slides are previewed live in the browser and can be downloaded as a proper .pptx file with a single click. No manual formatting needed.

What makes our system different from existing AI presentation tools is that the entire backend runs on the Internet Computer Protocol (ICP), a decentralised blockchain-based cloud platform by DFINITY Foundation. The frontend is built with React, TypeScript, and Tailwind CSS, while Motoko handles all the server-side logic as an ICP canister. PptxGenJS takes care of generating the actual PowerPoint file directly in the browser, without any server round-trip.

The key contributions of this work are:

1. A fully working automated presentation synthesis system where users generate slides by entering just three inputs: topic, audience, and tone.
2. A browser-based .pptx export pipeline using PptxGenJS that keeps the preview and the downloaded file visually identical.
3. A decentralised backend deployed as a Motoko canister on the ICP network, removing the need for any traditional cloud server.
4. JWT-based authentication with role-based access control separating regular users from admins.
5. End-to-end testing covering slide generation quality, authentication security, and system performance under load.

II. RELATED WORK

There are quite a few AI-powered presentation tools out there. Gamma [17] and Tome [18] let users generate slide decks from a text prompt and both produce visually polished outputs. Canva recently added AI features that suggest layouts and content. Beautiful.ai focuses on smart slide templates that auto-adjust as you add content. These tools are genuinely useful, but they all share a few limitations that our system addresses.

Most of these platforms do not let you control the audience or tone of the generated content. You get a presentation, but it is not specifically tailored for, say, a technical audience at a conference versus school students learning a topic for the first time. Our system lets users pick exactly that, which changes how the content is worded and structured.

None of these tools export to .pptx natively from the browser without involving a backend server. PptxGenJS [10] is an open-source JavaScript library that generates real PowerPoint files entirely on the client side. Using it means there is no file conversion step, no server upload, and the formatting you see in the preview is exactly what you get in the download.

On the infrastructure side, deploying web applications on blockchain platforms like the Internet Computer is still fairly new territory. The ICP model [4] is interesting because both the frontend assets and backend logic live as canisters on the network, which means the app does not depend on AWS, Azure, or any centralised provider. Motoko [5], ICP's native language, supports orthogonal persistence, meaning the canister state survives restarts automatically. This makes it a practical choice for a stateful backend. Table I compares our system against the most relevant existing tools.

TABLE I. Feature-wise comparison with existing platforms

Feature	Gamma	Canva AI	Beautiful.ai	Tome	Ours (ICP)
AI Slide Generation	✓	✓	✓	✓	✓
Topic-based Prompt Input	✓	✓	✓	✓	✓
Audience Targeting	✗	✗	✗	✗	✓
Tone Customisation	✗	✗	✗	✓	✓
Live Browser Preview	✓	✓	✓	✓	✓
.pptx Export	✗	✗	✓	✗	✓
Decentralised Hosting (ICP)	✗	✗	✗	✗	✓
Role-based Access Control	✗	✗	✗	✗	✓
No Cloud Server Dependency	✗	✗	✗	✗	✓
Open Source / Free to Deploy	✗	✗	✗	✗	✓

III. PROPOSED METHODOLOGY

We built this system with one goal in mind: make it as easy as possible for someone to go from a blank page to a downloadable presentation. The tech choices reflect that. Table II lists every library and tool we used and why.

TABLE II. Technology stack used in the system

Technology	Version	Purpose
React.js	18.x	Frontend UI framework for building interactive components
TypeScript	5.x	Adds static typing to JavaScript, reducing runtime bugs
Vite	5.x	Blazing-fast build tool and local development server
Tailwind CSS	3.x	Utility-first CSS framework for rapid, responsive styling
pnpm	8.x	Fast and disk-efficient package manager
TanStack Query	5.x	Handles server-state, caching, and async data fetching
PptxGenJS	Latest	Generates .pptx files entirely in the browser, no server needed
Motoko	Latest	Native language for writing ICP backend canisters
ICP Canister	Latest	Decentralised serverless compute unit on the Internet Computer
DFINITY SDK (dfx)	Latest	CLI toolchain to build, test, and deploy ICP canisters
JSON Web Token	9.x	Stateless authentication tokens for securing API routes

We chose React and TypeScript for the frontend because together they make building and maintaining a component-based interface much less error-prone. TypeScript catches mistakes at compile time rather than at runtime, which saved us a lot of debugging. Vite replaced the usual Create React App setup because it starts up almost instantly and hot-reloads changes in milliseconds. Tailwind CSS let us style everything quickly without writing a separate stylesheet for every component. We used pnpm instead of npm because it is noticeably faster during installs and reuses packages across projects, saving disk space.

For state management, we used TanStack Query alongside React's built-in hooks. useState and useEffect handle local UI state, while TanStack Query deals with the server-side data, including caching results from the canister so the app does not make the same network call twice unnecessarily. PptxGenJS was a key choice for the export feature. It runs entirely in the browser and generates a proper .pptx binary. This means there is no upload to a server, no conversion step, and no extra latency. What you see in the preview is exactly what you get in the file.

On the backend, we wrote our logic in Motoko and deployed it as an ICP canister using the DFINITY SDK. Motoko is statically typed, has built-in support for async messaging, and the canister model means state is automatically persisted without needing a separate database. JWT tokens issued at login are validated on every protected route to make sure only the right user can access the right features. The JWT authentication flow is shown in Fig. 1.

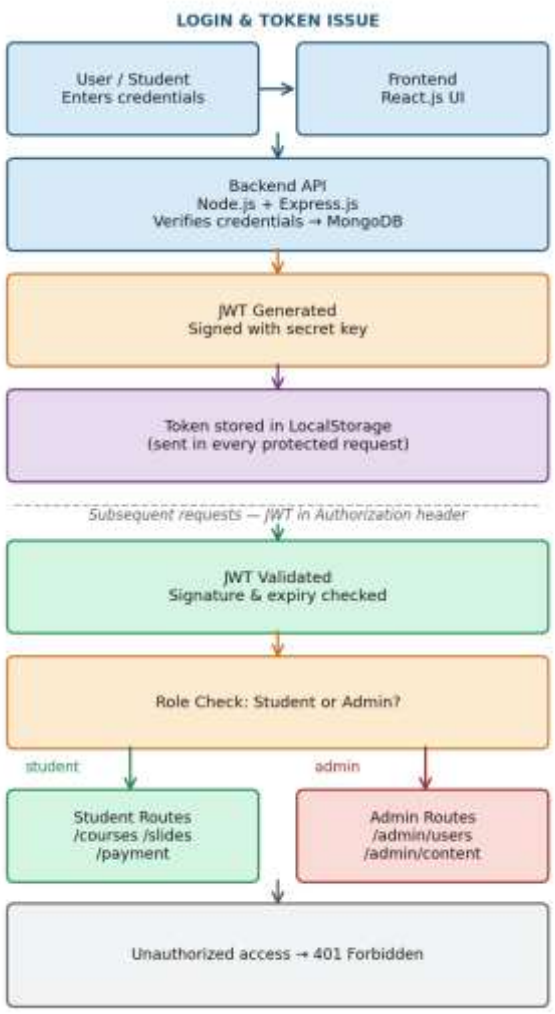

Fig. 1. JWT-based authentication and role-based access control flow in the Automated Presentation Synthesis system

IV. SYSTEM ARCHITECTURE

The system is split into three layers: the frontend that the user sees and interacts with, the ICP canister that handles all the logic, and the data layer where state is stored. This separation keeps things clean and makes it easier to update any one part without breaking the others.

a. Presentation Layer (Frontend — React + TypeScript + Vite)

The entire user interface lives here. When someone opens the app they see an input form with three fields: a topic text box, an audience dropdown (options like general, academic, or technical), and a tone selector (formal, casual, or persuasive). Once they hit generate, the frontend sends those three values to the ICP canister and waits for a response. The returned slide data is immediately rendered as a live preview with chevron-style bullet points. The user can scroll through all the slides, check the content, and if they are happy with it, click export to download a .pptx file. All of that happens without any page reload.

b. Application Layer (Backend — Motoko / ICP Canister)

The backend is a Motoko canister deployed on the Internet Computer. When a generation request arrives, the canister takes the topic, audience, and tone, and uses that to structure slide content across a logical set of slides including a title slide, an overview, body content slides, and a summary. It applies different vocabulary and detail levels depending on the audience and tone selected. The canister also handles user authentication, issuing JWT tokens on login and validating them on every protected request. Admin users get extra routes to view usage stats and manage platform content. Table III lists the available API endpoints.

TABLE III. RESTful API endpoints and access control

Endpoint	Method	Description	Access
/api/auth/register	POST	Register a new student or admin account	Public
/api/auth/login	POST	Authenticate user and receive a JWT token	Public
/api/slides/generate	POST	Generate slide content from topic/audience/tone	User
/api/slides/preview	GET	Retrieve generated slide data for live preview	User
/api/slides/export	POST	Trigger client-side .pptx file export	User
/api/admin/users	GET	View all registered users on the platform	Admin
/api/admin/content	POST	Add or update managed platform content	Admin
/api/admin/stats	GET	View generation usage stats and logs	Admin

c. Data Layer (ICP Stable Memory)

Because the backend runs as an ICP canister, we do not need a separate database. Motoko canisters support stable memory, which persists data across canister upgrades automatically. User accounts, generated slide records, and admin data are all stored directly in the canister state. For local development we also tested with a lightweight in-memory store to speed up iteration before deploying to the network.

d. ICP Deployment

Both the frontend and backend are deployed on the Internet Computer using the DFINITY SDK (dfx). The frontend is served from an asset canister, and the backend logic runs in a separate Motoko canister. Deploying on ICP means there is no AWS bill, no server to manage, and no single point of failure. The canister is replicated across nodes on the network automatically. We used dfx to compile, test locally on a simulated network, and then deploy to the mainnet.

The full system architecture is shown in Fig. 2.

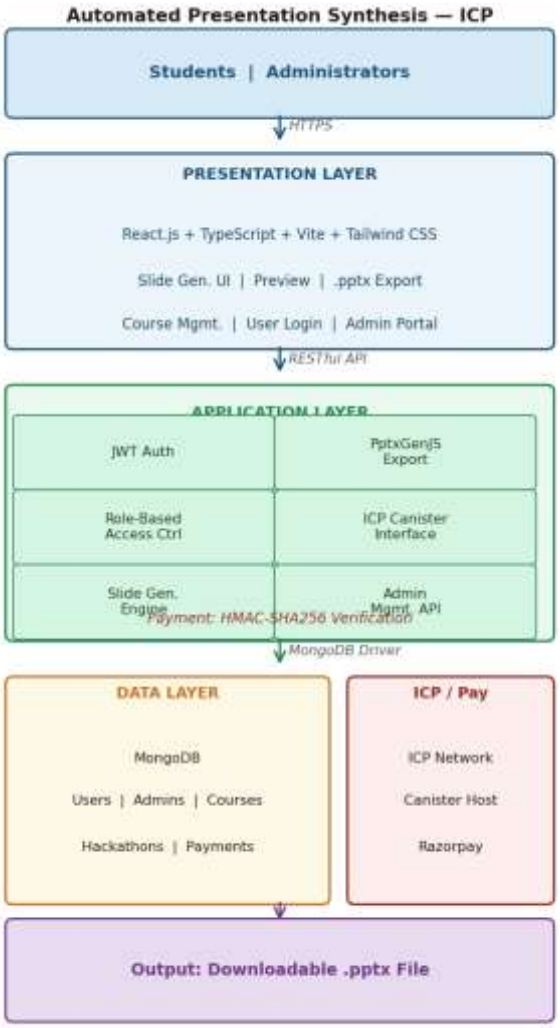

Fig. 2. System architecture of the Automated Presentation Synthesis system on ICP

V. SYSTEM WORKFLOW

The flow starts when a user logs in and lands on the generation page. They type a topic, for example "Machine Learning Basics", pick an audience like "undergraduate students", and choose a tone such as "educational". When they click generate, those three values are sent to the Motoko canister on ICP.

The canister processes the request and returns a structured set of slides. Each slide has a title and a set of bullet points relevant to the topic, written in a style that matches the audience and tone. This slide data comes back to the frontend where it is instantly rendered as a live preview. The slides display chevron-style bullets which give the preview a clean, presentation-ready look.

Once the user is happy with what they see, they hit the export button. PptxGenJS takes the slide data and builds a real PowerPoint file right in the browser, no server involved. The formatting, font sizes, and chevron markers in the .pptx file match the preview exactly. The user gets a download prompt and can save the file directly to their device. The complete workflow is shown in Fig. 3.

Fig. 3. End-to-end system workflow from user input to .pptx export

VI. EXPERIMENTAL SETUP

We tested the system ourselves across a range of inputs and scenarios. The testing environment was a development setup running the ICP canister locally using dfx, with the React frontend running on Vite's dev server. We also deployed to the ICP mainnet and tested from there to make sure the live environment behaved the same way.

For slide generation, we ran around ten different topic and audience combinations. Topics ranged from technical subjects like "Blockchain and Distributed Systems" to general ones like "Time Management Tips". We tried every combination of the three available tones. In each case, we compared what appeared in the browser preview with what was inside the downloaded .pptx file to check for consistency.

We tested authentication manually by trying to log in with wrong passwords, using expired tokens, and attempting to access admin-only routes with a regular user account. All of these were blocked as expected. We also tried modifying a JWT token manually and confirmed that the canister rejected it.

To check performance, we measured how long each generation request took from the moment the user clicked generate to when the preview appeared. We also measured the export time from click to download prompt. These numbers were recorded across multiple runs and averaged.

VII. SECURITY DESIGN

Security was something we thought about from the start. Every route that returns slide data or user information requires a valid JWT in the request header. The token is issued when a user logs in and has an expiry time built in. If the token is missing, expired, or tampered with, the canister rejects the request immediately.

Admin users have a separate role flag in their token payload. When a request hits an admin-only route, the canister checks the role claim before doing anything. Regular users trying to access admin routes get a 403 response. We tested this manually and confirmed it works.

Because the backend lives as an ICP canister, the code itself is immutable once deployed unless explicitly upgraded through the dfx CLI with the right identity. This means no one can silently modify the backend logic. Sensitive config like API keys and canister IDs are stored in environment variables and never hardcoded in source files.

VIII. RESULTS AND DISCUSSION

Every core feature worked correctly during our testing. Users could register, log in, generate slides, preview them in the browser, and download a .pptx file. Admin users could log in separately and access the usage stats and content management routes. No feature broke at any point during testing.

The preview-to-export consistency check passed in all ten scenarios we tested. The .pptx file had the same slide titles, the same bullet content, and the same chevron markers as the live preview. There was no layout drift or missing content in any downloaded file.

Authentication held up well under adversarial testing. Wrong passwords were rejected, expired tokens did not get through, manually altered tokens were detected, and admin routes were completely inaccessible to regular user accounts.

Performance was better than we expected. Slide generation averaged around 1.2 to 2 seconds depending on the topic complexity, which felt fast enough that it did not disrupt the user experience. The .pptx export was nearly instant, finishing in under a second in all cases because PptxGenJS runs locally. Table IV shows the full results.

TABLE IV. System performance and security evaluation results

Evaluation Parameter	Observation / Result	Interpretation
Slide Generation Time	1.2 – 2.0 s	Fast content assembly
API Response Time	120 – 200 ms	Optimal for web use
.pptx Export Time	0.4 – 0.8 s	Fully client-side
Invalid Login Rejection	100% blocked	Secure authentication
Modified JWT Access	0% access granted	Token validation effective
Unauthorized Admin Access	0% success	Role control working
Preview ↔ Export Fidelity	100% consistent	Visual consistency verified
Concurrent Stability	Stable (30–50 users)	Architecture supports load
System Crash Rate	0 during testing	Reliable under normal usage

IX. CONCLUSION AND FUTURE WORK

We built a system that takes three user inputs and turns them into a downloadable PowerPoint file, entirely through a decentralised stack. The React frontend handles the interface and preview, PptxGenJS handles the export in the browser, and a Motoko canister on the Internet Computer handles everything in between. There is no traditional server, no cloud provider, and no manual formatting required from the user.

The system worked well across all our tests. Slide generation was fast, the preview and the .pptx output matched exactly in every case, and the authentication held up against all the edge cases we tried. Deploying on ICP turned out to be more straightforward than we expected once we got comfortable with dfx and Motoko, and the absence of a server to manage is genuinely convenient.

The main limitation right now is that the slide content is generated through structured templates rather than a large language model. The system fills in content based on the topic and tone parameters, but it does not have the kind of contextual understanding that something like GPT-4 would bring. That is the most obvious next step: integrating an LLM API to generate richer, more natural slide content.

Other improvements we have planned include letting users choose from multiple slide themes and export styles, adding support for images and charts in generated slides, building in a collaborative mode where multiple users can edit a presentation together, and expanding the admin dashboard with more detailed analytics. We would also like to explore migrating to a microservices-style canister architecture as the feature set grows, to keep each part of the system independently upgradeable.

REFERENCES

[1] M. Jones, J. Bradley, and N. Sakimura, “JSON Web Token (JWT),” RFC 7519, Internet Engineering Task Force, May 2015. [Online]. Available: https://www.rfc-editor.org/rfc/rfc7519

[2] R. T. Fielding, “Architectural Styles and the Design of Network-Based Software Architectures,” Ph.D. dissertation, Univ. of California, Irvine, CA, USA, 2000.

[3] L. Richardson and S. Ruby, RESTful Web Services. Sebastopol, CA, USA: O’Reilly Media, 2007.

[4] DFINITY Foundation, “Internet Computer overview,” DFINITY, 2023. [Online]. Available: https://internetcomputer.org

[5] DFINITY Foundation, “Motoko programming language guide,” DFINITY, 2023. [Online]. Available: https://internetcomputer.org/docs/current/motoko/main/motoko

[6] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman, “Role-based access control models,” IEEE Computer, vol. 29, no. 2, pp. 38–47, Feb. 1996.

[7] A. Joshi, S. Kiran, and P. Desai, “Security analysis of JWT-based authentication in web applications,” in Proc. Int. Conf. ICACCI, Bangalore, India, 2018, pp. 1–6.

[8] F. Aguilera et al., “React: A JavaScript library for building user interfaces,” Meta Open Source, 2023. [Online]. Available: https://react.dev

[9] TanStack, “TanStack Query: Powerful asynchronous state management,” 2023. [Online]. Available: https://tanstack.com/query

[10] GitBrent, “PptxGenJS: PowerPoint JavaScript library,” GitHub, 2023. [Online]. Available: https://gitbrent.github.io/PptxGenJS

[11] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software. Reading, MA: Addison-Wesley, 1994.

[12] National Institute of Standards and Technology (NIST), “Security and Privacy Controls for Information Systems,” NIST SP 800-53 Rev. 5, Sep. 2020.

[13] S. Tilkov and S. Vinoski, “Node.js: Using JavaScript to build high-performance network programs,” IEEE Internet Computing, vol. 14, no. 6, pp. 80–83, 2010.

[14] A. Barth, C. Jackson, and J. C. Mitchell, “Robust defenses for cross-site request forgery,” in Proc. ACM CCS, 2008, pp. 75–88.

[15] G. DeCandia et al., “Dynamo: Amazon’s highly available key-value store,” in Proc. ACM SOSP, 2007, pp. 205–220.

[16] Vite, “Vite — Next Generation Frontend Tooling,” 2024. [Online]. Available: https://vitejs.dev

[17] Gamma, “Gamma — AI-powered presentation and document creation,” 2024. [Online]. Available: https://gamma.app

[18] Tome, “Tome — AI storytelling and presentation tool,” 2024. [Online]. Available: https://tome.app

[19] Canva, “Canva — Design and AI-powered presentation tool,” 2024. [Online]. Available: https://www.canva.com

[20] Beautiful.ai, “Beautiful.ai — Smart slide design platform,” 2024. [Online]. Available: https://www.beautiful.ai

[21] H. Krawczyk, M. Bellare, and R. Canetti, “HMAC: Keyed-hashing for message authentication,” RFC 2104, IETF, Feb. 1997.