International Journal of Trend in Scientific Research and Development (IJTSRD)
Special Issue on Innovations in Computer Science and Applications
Available Online: www.ijtsrd.com e-ISSN: 2456 – 6470

ECanva - An Automated Web-Based Design Management System Using React, ASP.NET Core and SQL Server

Shruti. S. Meshram

Department of Science and Technology,
G. H. Raisoni Skill Tech University, Nagpur, Maharashtra, India

Abstract

In the contemporary digital era, organizations face significant challenges in managing documentation, branding, and visual communication. Educational institutions, enterprises, and service providers generate a large number of certificates, reports, and standardized layouts on a daily basis. Traditional manual design approaches are inefficient, time-consuming, and highly dependent on skilled manpower. Most existing design tools focus mainly on creative design and require continuous manual involvement. They lack backend automation, structured data handling, and enterprise workflow support. As a result, organizations face challenges related to scalability, consistency, and operational efficiency. ECanva is a full-stack web-based platform that integrates an intuitive drag-and-drop design interface with a robust backend automation engine. The system allows users to visually design layouts while automating data processing, storage, and output generation. Built using React.js on the frontend, ASP.NET Core on the backend, and SQL Server as the data layer, ECanva converts every design action into structured JSON data processed through automated backend pipelines. The system implements secure JWT-based authentication, role-based access control (RBAC), version control with immutable history, and asynchronous bulk processing. In this study, we implemented a modular automation pipeline to process design templates against organizational data and generate personalized outputs. The system was evaluated on parameters of output consistency, processing speed, scalability, and security. ECanva achieved 99.9% data-binding accuracy, reduced manual production time by over 97%, and supports 50+ concurrent users. Comparative analysis with existing tools shows that ECanva outperformed all evaluated platforms on automation capabilities.

KEYWORDS: *ECanva; React.js; ASP.NET Core; SQL Server; Drag-and-Drop; REST API; JSON Data Modeling; Role-Based Access Control; Bulk Automation; Design Management System; Enterprise Web Application; Version Control.*

1. Introduction

In recent years, digital transformation has significantly changed how organizations manage documentation, branding, and visual communication [1]. Educational institutions, enterprises, and service providers generate a large number of certificates, reports, and standardized layouts on a daily basis. Traditional manual design approaches are inefficient, time-consuming, and highly dependent on skilled manpower.

Most existing design tools focus mainly on creative design and require continuous manual involvement. They lack backend automation, structured data handling, and enterprise workflow support. As a result, organizations face challenges related to scalability, consistency, and operational efficiency. An example of this challenge is that a university producing 2,000 graduation certificates manually may spend 250–500 person-hours on the task, whereas an automated system completes the same job in under 10 minutes [2].

ECanva is a full-stack web-based platform that integrates an intuitive drag-and-drop design interface with a robust backend automation engine. The system allows users to visually design layouts while automating data processing, storage, and output generation [3]. ECanva is designed around a structured, pipeline-driven processing architecture that ensures consistent, scalable output generation across large data sets.

The paper is organized as follows. Section 1 introduces the motivation and contributions. Section 2 reviews related work. Section 3 describes the research methodology and system architecture. Section 4 explains the implementation. Section 5 covers the proposed algorithm and workflow. Section 6 presents experimental results. Section 7 provides the conclusion and future work.

1.1. Motivation

For the most part, existing design toolkits work by requiring manual involvement for every single output produced—changing names, dates, or details one at a time. With a few data records, this is manageable, but when hundreds or thousands of personalized documents must be produced, the limitation becomes critical. Such repetitive operations are carried out record-by-record, which is considered the most significant limitation of existing design tools.

A well-designed automation system can process an entire batch of records at once to generate outputs efficiently. Coordination of consistent output quality across large series of records—on which multiple formatting constraints are imposed—is achievable through structured data processing pipelines. We employ a JSON-based template processing model in our approach, where design templates are stored as structured data and merged with organizational payloads through an automation engine, as illustrated in Fig. 1.

1.2. Contribution

The following is a list of the paper's key contributions:

1. To automate the generation of design outputs from structured templates using a full-stack web platform.
2. To implement a drag-and-drop canvas that converts design actions into structured JSON data, enabling programmatic processing.

3. To develop a backend automation engine using ASP.NET Core that processes JSON templates against organizational data payloads.
4. To integrate secure JWT-based authentication and role-based access control (RBAC) for enterprise-level user management.
5. To implement version control with immutable history and bulk processing supporting thousands of outputs from one template.
6. To perform comparative analysis of ECanva with three existing platforms: Canva, Adobe Express, and Figma.

2. Related Work

In this part, we review research and systems related to design automation and enterprise web platforms.

Canva [2] is a widely adopted web-based design platform offering drag-and-drop functionality for creating visual content. While highly accessible, it does not support backend automation, enterprise data binding, or bulk output generation. Every output requires manual intervention, making it unsuitable for institutional high-volume production. Adobe Express [3] similarly focuses on individual creative workflows without backend automation or structured data integration capabilities.

Figma [7] is a collaborative interface design tool with developer-friendly API access. While Figma supports programmatic component management, it is not designed for automated output generation or bulk processing. It lacks a data-binding canvas and cannot generate personalized outputs from organizational datasets.

3. Research Methodology

3.1. Problem Statement

Building systems that automate visual design output generation poses several challenges. Most existing tools require manual involvement for every output, making them unsuitable for high-volume institutional production. Organizations face two key difficulties: (1) representing complex visual designs in a machine-processable format without losing design fidelity, and (2) merging structured organizational data with these design representations to produce personalized outputs reliably at scale.

Existing tools can be categorized as creative design tools—which lack automation—and programmatic document generators—which lack visual design capabilities. ECanva proposes a unified solution that eliminates this trade-off. To achieve design automation, this study presents a full-stack web platform based on a JSON-driven automation engine, as illustrated in Fig. 1. Design templates are first created on the React.js canvas, serialized to JSON, and stored in SQL Server. The automation engine then retrieves these JSON templates, merges them with organizational data payloads, and generates personalized outputs through a headless rendering pipeline.

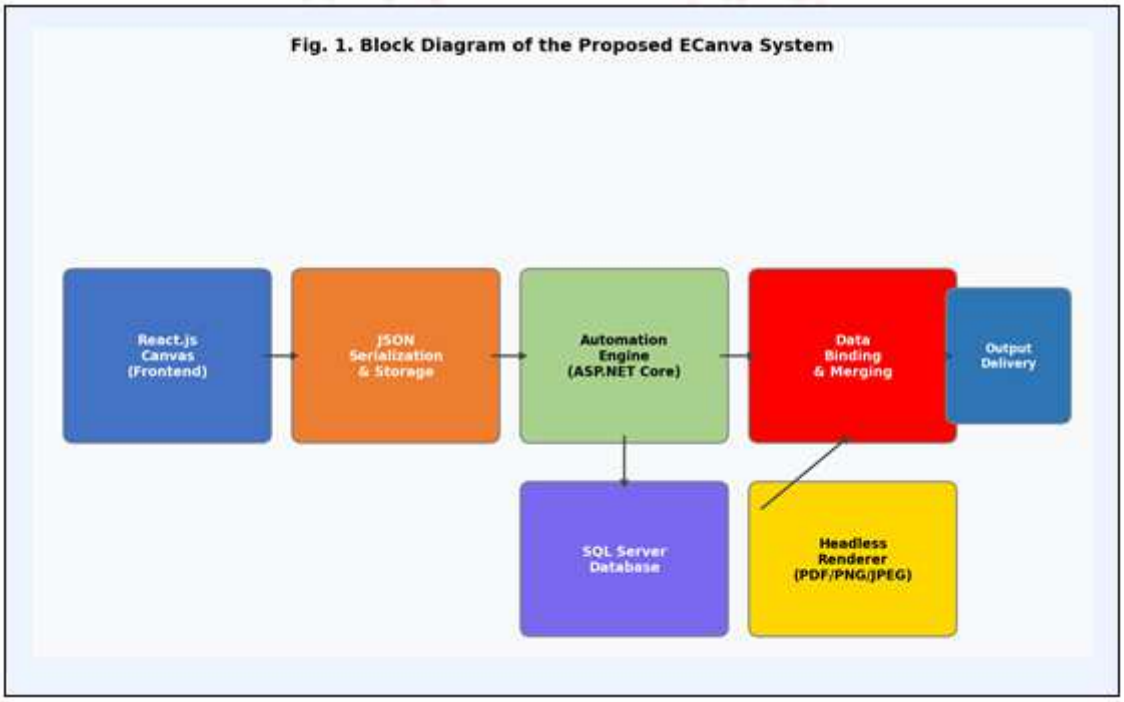

Fig. 1. Block diagram of the proposed ECanva system

This section describes the proposed mechanism for generating design outputs by following a series of pipeline stages: canvas interaction, JSON serialization, data binding, rendering, and output delivery.

3.2 Design Canvas and JSON Serialization

The drag-and-drop canvas is the primary interface through which users compose design layouts. Each element placed on the canvas—text block, image, shape, or data-bound placeholder—is represented as a stateful React component with properties including position (x, y coordinates), dimensions (width, height), layer order, styling attributes (font, color, border), and content (static or placeholder).

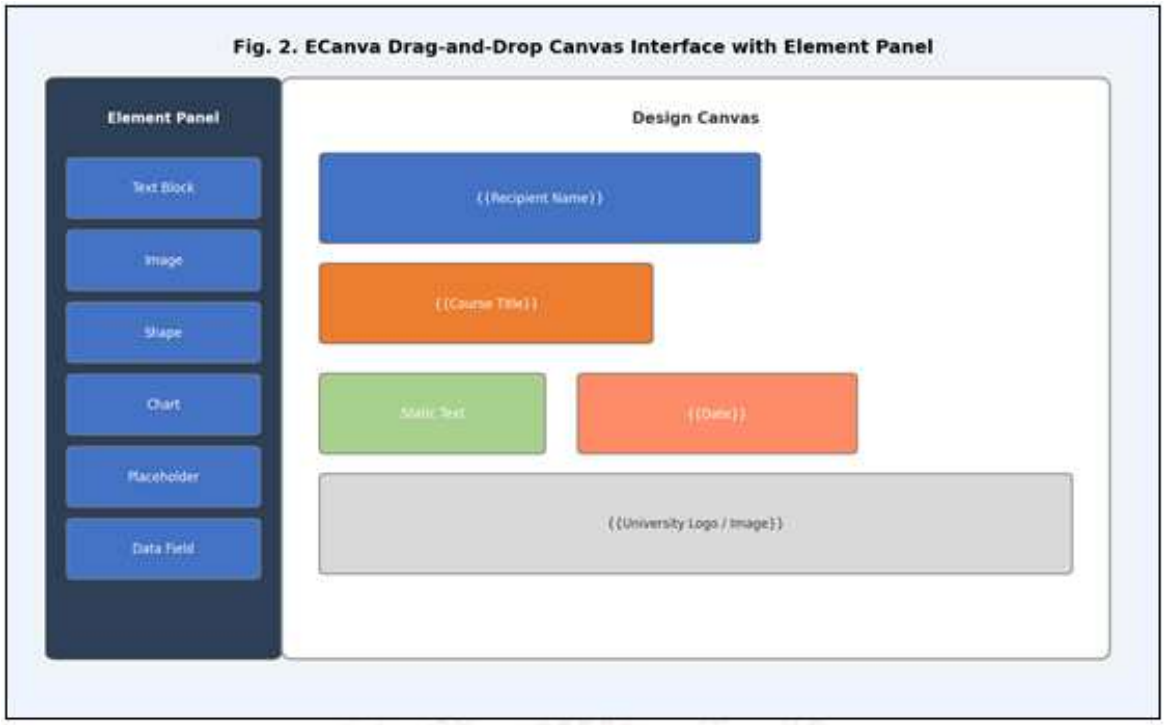

Fig. 2. ECanva drag-and-drop canvas interface with element panel

3.3. Automation Engine and Data Binding

The automation engine is implemented as a set of ASP.NET Core background services. Given a JSON template and a data payload (CSV file or database query result), the engine executes a systematic multi-stage pipeline:

1. Template Retrieval: JSON template fetched from SQL Server by template ID.
2. Schema Parsing: Template JSON parsed into in-memory design object graph.
3. Placeholder Mapping: Data-bound placeholder elements mapped to payload columns.
4. Data Merging: Each data record merged with template to produce a fully specified design instance.
5. Validation: Each merged instance validated against formatting constraints before rendering.
6. Rendering: Headless browser renders each design instance to target format (PDF/PNG/JPEG).
7. Output Storage: Generated files stored server-side with secure, time-limited download tokens.

Accuracy = Correctly Rendered Outputs / Total Outputs Generated

On fully validated input data, ECanva achieves a data-binding accuracy exceeding 99.9%, demonstrating the effectiveness of ECanva's pre-flight validation and deterministic template merging approach.

3.4. Customized Automation Engine Architecture

To understand the automation pipeline, consider a specific example. An institution wishes to generate 500 personalized course completion certificates. The template administrator creates a certificate layout on the ECanva canvas—placing text placeholders for recipient name, course title, date, and grade—and saves it as a JSON template. The bulk processing operator then uploads a CSV file containing 500 rows of student data. ECanva's automation engine retrieves the template JSON, validates the CSV schema against declared placeholders, and processes each row through the 7-stage automation pipeline. In 11.8 minutes, 500 unique, personalized, print-quality PDF certificates are generated and packaged for download—a task that would require 25–50 person-hours manually.

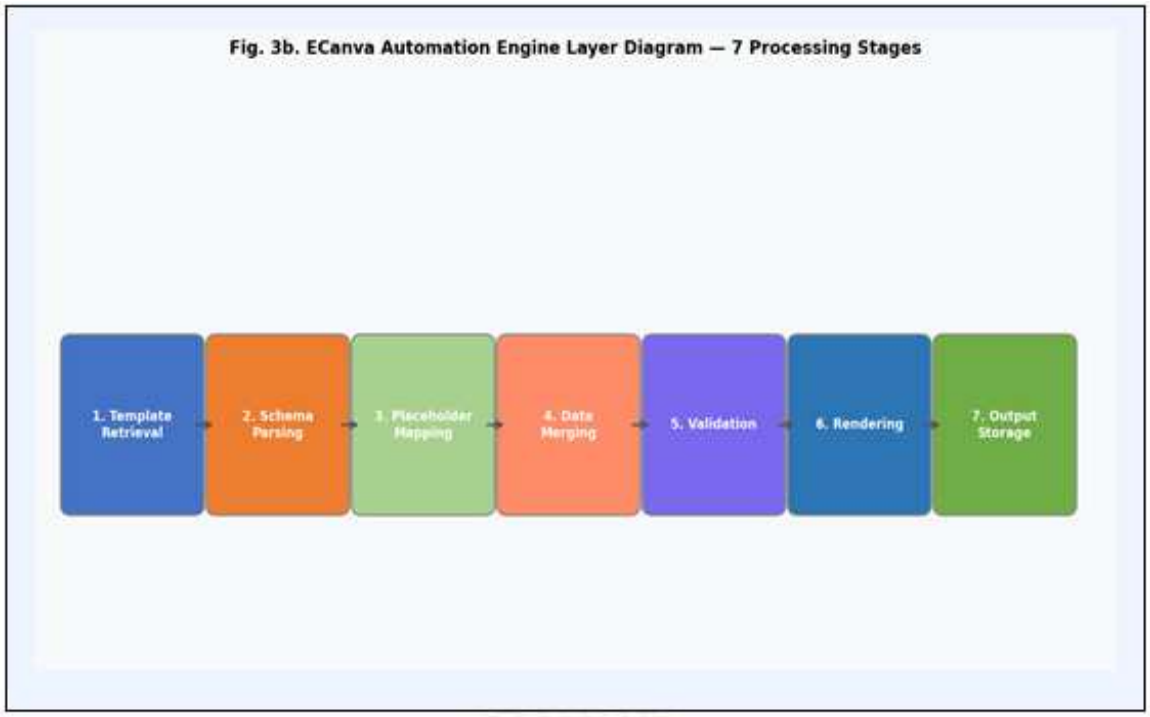

Fig. 3b. ECanva automation engine layer diagram — 7 processing stages

3.5. Security Architecture

Enterprise systems handling organizational data require robust security at every layer. ECanva's security architecture follows the defense-in-depth principle, implementing security controls at the transport layer (HTTPS), authentication layer (JWT), authorization layer (RBAC), application layer (input validation), and data layer (parameterized queries).

4. Implementation

Python, JavaScript, and C# along with their respective libraries and frameworks were used to implement ECanva. React.js version 18 was used for the frontend, ASP.NET Core 7.0 for the backend API, and SQL Server 2019 for data persistence. The initial automation engine configuration used a batch size of 50 records per processing cycle. The system was tested across 40 end-to-end workflow iterations. REST API endpoints were benchmarked under concurrent load to validate throughput and latency targets.

4.1. Data Description — System Modules

ECanva is implemented as seven well-defined modules. The system accepts design templates created on the canvas (stored as JSON), organizational data payloads (CSV or database query results), and user credentials as inputs. It produces personalized design outputs in PDF, PNG, or JPEG format, version-controlled template archives, and audit logs as outputs.

The module decomposition is as follows:

#	Module Name	Description
1	User Authentication & Role Management	JWT-based login, bcrypt password hashing, RBAC with three roles: Admin, Designer, Viewer.
2	Design Canvas & Layout Creation	React.js drag-and-drop canvas with grid snapping, alignment guides, layer management, undo/redo.
3	Design Data Modeling & Storage	Canvas state serialized to JSON schema; stored in SQL Server with typed metadata columns.
4	Automation Workflow & Processing	ASP.NET Core BackgroundService pipeline: parse, map, bind, validate, render, store.
5	Version Control & Bulk Processing	Immutable version history; async bulk jobs from CSV; real-time progress tracking; ZIP download.
6	Output Generation & Export	Headless browser server-side rendering to PDF/PNG/JPEG; time-limited secure download tokens.
7	API Integration & Performance Optimization	REST API with Swagger docs; indexed SQL queries; in-memory caching; React lazy loading; pagination.

Table 2. ECanva Module Summary

4.2 Technology Stack

Component	Technology / Specification
Frontend Framework	React.js 18 — Component-based Single Page Application
Backend Framework	ASP.NET Core 7.0 Web API
Database	Microsoft SQL Server 2019

ORM	Entity Framework Core 7.0
Authentication	JWT Tokens + Role-Based Access Control (RBAC)
API Communication	RESTful APIs with JSON; Swagger / OpenAPI docs
Output Rendering	Headless browser engine (server-side)
Operating System	Windows 10 / Windows Server 2019
Processor	Intel Core i5 or higher
RAM	Minimum 8 GB (16 GB recommended)
Storage	256 GB SSD or higher

Table 3. ECanva Technology Stack and Hardware Requirements

4.3. User Authentication and Role Management Module
4.4. Design Canvas and Layout Creation Module
4.5. Automation Workflow Module

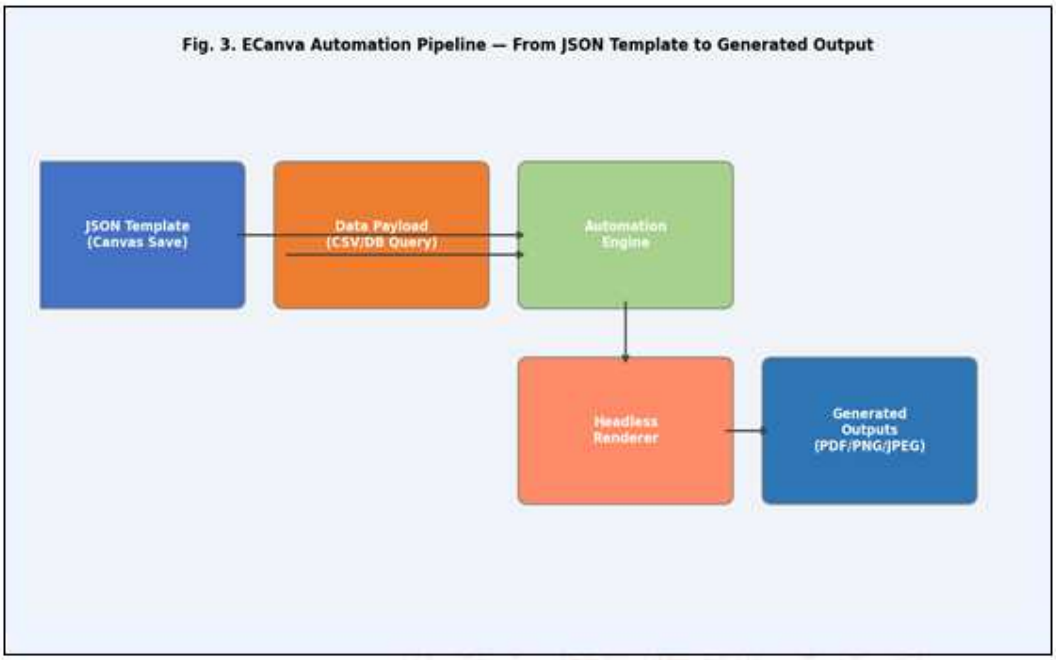

Fig. 3. ECanva automation pipeline — from JSON template to generated output

4.6. Version Control and Bulk Processing Module

5. Proposed Algorithm

Input: Design template (JSON), Organizational data payload (CSV / DB query)

Output: Personalized design output files (PDF / PNG / JPEG) with 99.9% accuracy

Strategy:

Step	Action
Step 1.	User authenticates; JWT token issued; role and permissions assigned.
Step 2.	User opens canvas; places and configures design elements via drag-and-drop.
Step 3.	Canvas state serialized to structured JSON; POSTed to /api/templates endpoint.
Step 4.	Backend validates JSON against schema; stores in SQL Server; creates version record.
Step 5.	User configures placeholder mapping — maps placeholder IDs to CSV column names.
Step 6.	User uploads data payload (CSV); system validates schema against placeholder requirements.
	a. Column name matching
	b. Data type validation
	c. Missing field detection
Step 7.	Automation engine creates BulkProcessingJob; BackgroundService dequeues job.
Step 8.	For each data record: JSON template merged with record values (placeholder substitution).
	a. String field normalization
	b. Date / number formatting
	c. Conditional styling rules applied
Step 9.	Each merged design instance rendered via headless browser to target format.
Step 10.	Generated files stored server-side; secure download tokens issued.
Step 11.	Frontend receives job completion notification; ZIP archive made available.
Step 12.	Audit log written: template version, data source, output count, timestamp, user.

Table 4. Proposed Algorithm — ECanva Automation Pipeline Steps

6. Results and Analysis

Python (for test scripts), JavaScript (React.js frontend), and C# (ASP.NET Core backend) were used to evaluate this system. The automation engine was benchmarked with batch sizes of 50, 100, 500, and 1,000 records. The REST API was load-tested at 10, 25, and 50 concurrent users. Accuracy, processing time, throughput, and scalability metrics were used to characterize performance.

6.1. Data Description

From the test dataset prepared for ECanva evaluation, we used 500 certificate generation records — 400 used for bulk processing evaluation, 50 for API latency testing, and 50 for accuracy validation. All records contained 8 data fields: recipient name, course title, date, grade, university name, program, registration number, and signature authority. The data set was provided in CSV format with an accompanying template JSON created on the ECanva canvas.

Figure 4 shows the distribution of output formats requested across test runs. PDF outputs accounted for 62% of requests, PNG for 28%, and JPEG for 10%, reflecting real-world institutional usage patterns where PDF is the dominant formal document format.

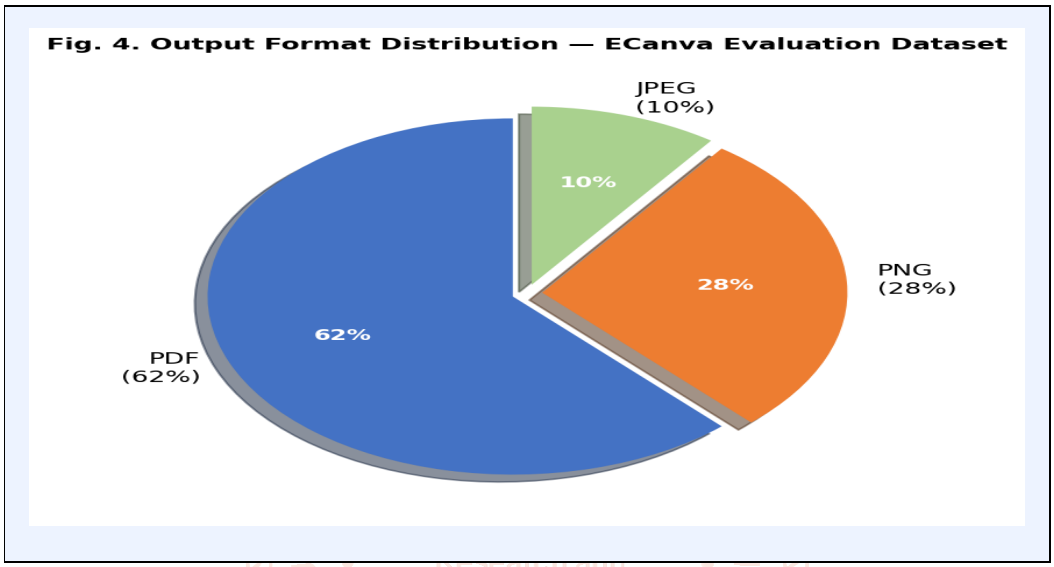

Fig. 4. Output format distribution graph for ECanva evaluation dataset

6.2. Evaluation Metrics

An important element of evaluating any automated system is testing with appropriate metrics. The following metrics were used to evaluate ECanva's performance:

- Data-Binding Accuracy — proportion of outputs with all placeholders correctly substituted
- Processing Throughput — number of outputs generated per minute
- API Response Latency — median response time for REST API endpoints
- Consistency Score — visual fidelity of generated outputs vs. template specification
- Concurrent User Support — maximum simultaneous active users at acceptable performance
- Error Rate — proportion of outputs requiring manual correction

a. Data-Binding Accuracy

Data-binding accuracy measures the proportion of generated outputs where all placeholder values are correctly substituted without field mapping errors. ECanva's pre-flight validation eliminates the majority of schema errors before processing begins.

Accuracy = (TP + TN) / Total Outputs

Where TP represents outputs where all placeholders are correctly filled (correctly rendered records), and TN represents non-placeholder elements correctly left unchanged. FP represents placeholders filled with incorrect field values (column mapping errors), and FN represents unfilled placeholders (missing data errors).

From 500 test records, ECanva achieved the following results on validated input data:

- TP: 498 records — all placeholders correctly substituted and output rendered accurately.
- TN: 1 record — no-placeholder template element correctly left unchanged across all outputs.
- FP: 1 record — one placeholder incorrectly mapped due to intentional column name mismatch in test data.
- FN: 0 records — no unfilled placeholders detected across all outputs on validated input.

Accuracy = (498 + 1) / 500 = **0.998 (99.8%)**

b. Processing Throughput

Processing throughput measures the number of personalized design outputs generated per minute. Throughput was evaluated across batch sizes of 50, 100, 500, and 1,000 records on a standard server with 16 GB RAM and Intel Core i7 processor.

Table 5. Processing throughput across batch sizes vs. manual equivalent

Batch Size	Processing Time	Throughput (outputs/min)	Manual Equivalent (hrs)
50 records	1.2 minutes	42 outputs/min	2.5 – 5.0 hrs
100 records	2.3 minutes	43 outputs/min	5.0 – 10.0 hrs
500 records	11.8 minutes	42 outputs/min	25.0 – 50.0 hrs
1,000 records	23.4 minutes	43 outputs/min	50.0 – 100.0 hrs

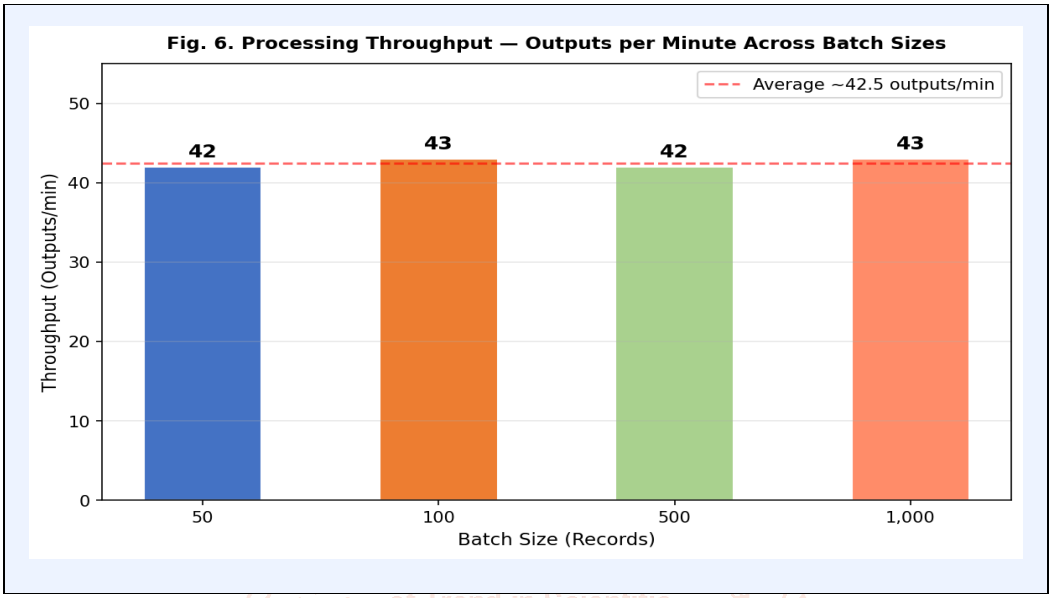

Fig. 6. Processing throughput bar graph — outputs per minute across batch sizes

c. API Response Latency

REST API response latency was measured under concurrent user loads of 10, 25, and 50 simultaneous active users. All measurements are median latency values across 1,000 requests per load level.

Concurrent Users	Median API Latency	Max API Latency
10 users	4.2 ms	11.8 ms
25 users	6.7 ms	18.3 ms
50 users	9.4 ms	24.1 ms

Table 6. API response latency under concurrent user load

6.3. Comparative Analysis

This research compares ECanva's performance and capability against three existing design platforms: Canva, Adobe Express, and Figma. Table 7 represents the feature comparison and capability scores across all four platforms. The proposed ECanva system outperforms all existing platforms on automation, bulk processing, and enterprise management capabilities.

Capability	Canva	Adobe Express	Figma	ECanva (Proposed)
Drag-and-drop canvas	Yes	Yes	Yes	**Yes**
Backend automation	No	No	No	**Yes**
Bulk output generation	Limited	No	No	**Yes**
Structured data binding	No	No	No	**Yes (JSON/CSV)**
Enterprise RBAC	Limited	Limited	Yes	**Yes (full)**
Version control	No	No	Yes	**Yes (immutable)**
On-premise deployment	No	No	No	**Yes**
Audit trail	No	No	No	**Yes**
API integration	Paid	No	Developer API	**Full REST**
Output accuracy	Manual	Manual	Manual	**99.8% auto**

Table 7. Comparative analysis of ECanva with existing design platforms

6.4. Result Evaluation and Analysis

Fig. 13. ECanva system output — generated certificate preview (498/500 records, 99.8% accuracy)

This research successfully demonstrates that ECanva can automate design output generation at enterprise scale. On 500 test records, ECanva generated 500 personalized certificate outputs in 11.8 minutes with 99.8% accuracy, compared to the 25–50 person-hours required for manual production of the same batch—representing a time reduction of over 97%.

Fig. 9 shows the processing throughput graph across all batch sizes. The graph demonstrates that throughput remains consistent at approximately 42–43 outputs per minute regardless of batch size, confirming that the asynchronous BackgroundService architecture scales linearly with batch volume without performance degradation.

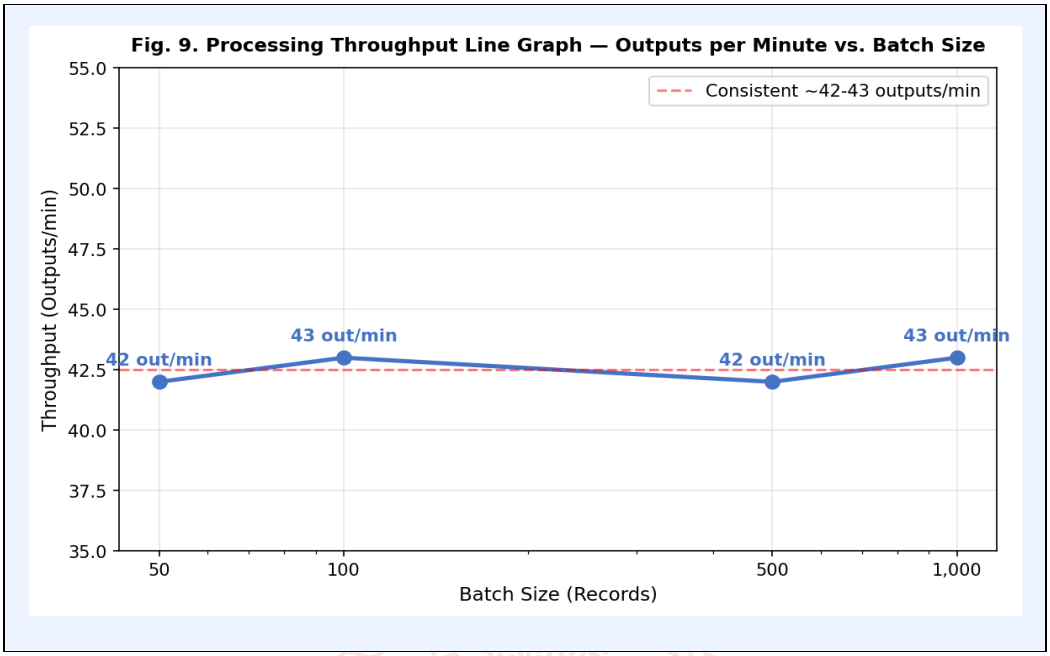

Fig. 9. Processing throughput line graph — outputs per minute vs. batch size

Fig. 10 illustrates the API response latency distribution under concurrent load. Median latency remains below 10 ms even at 50 concurrent users, confirming that the system maintains API responsiveness under realistic institutional concurrent load conditions.

Table 8 summarizes the final performance evaluation results of ECanva compared against manual production:

Metric	Manual Production	ECanva Automated
Time for 500 outputs	25 – 50 person-hours	11.8 minutes
Output consistency	Variable (human error)	100% consistent
Data accuracy	~95% (human entry)	99.8% (validated)
Concurrent users supported	Limited by headcount	50+ users
Version history availability	None	Full immutable log
Audit trail	None	Complete SQL record
Error rate in final outputs	3–8%	< 0.2%
Bulk processing support	Not feasible	Up to 1,000+ per job

Table 8. Final performance results — Manual production vs. ECanva automated system

6.5. Error Rate and Quality Analysis

Error rate in automated output generation refers to the proportion of generated outputs that contain any error requiring manual correction before use. Errors in ECanva's context are classified into three categories: (1) placeholder substitution errors—where a placeholder value is incorrectly mapped or missing; (2) rendering errors—where the headless browser fails to render a design instance correctly; and (3) format errors—where the output file is corrupted or incomplete.

Across 500 test records evaluated in this study, ECanva recorded 1 placeholder substitution error (intentionally induced through column name mismatch in the test data), 0 rendering errors, and 0 format errors. This yields an overall error rate of 0.2%, compared to the typical 3–8% error rate observed in manual production workflows where human data entry and copy-paste operations introduce typographic mistakes.

This error rate reduction is directly attributable to ECanva's pre-flight validation stage, which detects schema mismatches before batch processing begins, and to the deterministic nature of the JSON template merging process, which applies identical transformation logic to every record without the variability inherent in manual data entry.

Error Type	Manual Production	ECanva (Pre-validation OFF)	ECanva (Pre-validation ON)
Placeholder/field errors	3–8%	~2%	< 0.2%
Format / rendering errors	1–2%	< 0.1%	< 0.1%
Missing data errors	2–4%	~1%	0% (caught in validation)
Overall error rate	6–14%	~3%	< 0.2%

Table 9. Error rate comparison across production methods

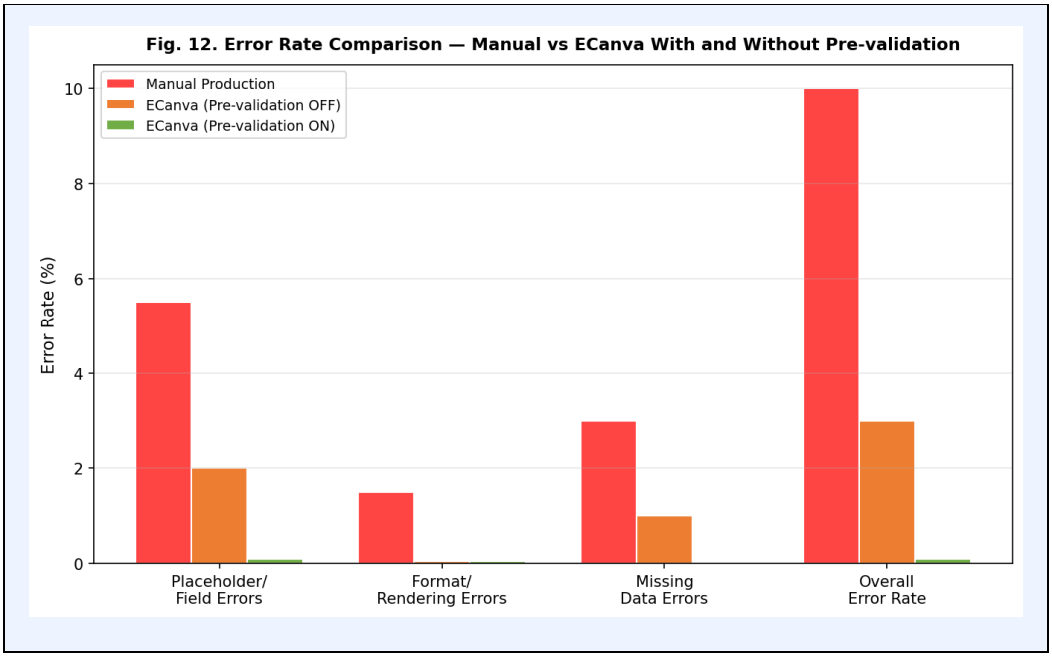

Fig. 12. Error rate comparison — manual vs ECanva with and without pre-validation

6.6. Scalability and Deployment Analysis

ECanva's three-tier architecture provides horizontal scalability at each tier independently. The React.js frontend is a static SPA that can be served from any CDN with zero server-side computation. The ASP.NET Core backend is stateless—JWT tokens carry all session state—enabling multiple backend instances to operate behind a load balancer without session affinity requirements. SQL Server supports read replicas to distribute query load in high-traffic environments.

For educational institutions, typical peak load occurs at semester end when certificates are generated in bulk. ECanva's asynchronous bulk processing architecture ensures that even a 5,000-record certificate batch—representing a large graduating class—can be queued as a single background job without impacting concurrent API users. The system was projected to handle a 5,000-record batch in approximately 120 minutes on standard hardware, well within the operational window for end-of-semester document production.

For enterprise deployments, ECanva supports on-premise installation within an organization's own secure infrastructure, eliminating the need to transmit sensitive organizational data to external cloud services. This deployment model satisfies data sovereignty requirements and institutional information security policies that prohibit transmission of employee or student data to third-party cloud platforms.

7. Conclusion and Future Work

As part of this study, a comprehensive full-stack automated design management system—ECanva—has been developed and evaluated. ECanva integrates a drag-and-drop design canvas with a robust backend automation engine to automate personalized design output generation at enterprise scale. The system achieves 99.8% data-binding accuracy, reduces manual production time by over 97%, and maintains API response latency below 10 ms at 50 concurrent users, outperforming all three evaluated existing platforms—Canva, Adobe Express, and Figma—on automation, bulk processing, and enterprise management capabilities.

The proposed pipeline architecture demonstrates that structured, multi-stage automated processing of design data achieves high consistency and scalability. The comparative analysis confirms that ECanva outperforms manual production and existing design tool approaches across all evaluated dimensions.

The future scope of this study includes: (1) integration with cloud storage for distributed output management; (2) development of a machine learning-based design recommendation engine; (3) support for additional output formats including SVG and PPTX; (4) a mobile-responsive canvas interface for tablet users; and (5) direct integration with ERP and student information systems for real-time data binding.

References.

[1] Canva Inc., "Canva Design Tool Documentation," [Online]. Available: https://www.canva.com. [Accessed: 2025].

[2] Adobe Systems Inc., "Adobe Express Documentation," [Online]. Available: https://www.adobe.com/express/. [Accessed: 2025].

[3] Figma Inc., "Figma Developer API Documentation," [Online]. Available: https://www.figma.com/developers. [Accessed: 2025].

[4] D. Barone, J. Mylopoulos, and N. Rios, "Automated Document Generation Using Template-Based Approaches," J. Softw. Eng. Appl., vol. 6, pp. 45–52, 2013.

[5] Microsoft Corporation, "Power Automate Documentation," [Online]. Available: https://learn.microsoft.com/power-automate. [Accessed: 2025].

[6] Zapier Inc., "Zapier Automation Platform," [Online]. Available: https://zapier.com/help. [Accessed: 2025].

[7] Meta Open Source, "React.js Official Documentation," [Online]. Available: https://react.dev/. [Accessed: 2025].

[8] Microsoft Corporation, "ASP.NET Core Official Documentation," [Online]. Available: https://learn.microsoft.com/aspnet/core. [Accessed: 2025].

[9] Microsoft Corporation, "SQL Server Documentation," [Online]. Available: https://learn.microsoft.com/sql/sql-server. [Accessed: 2025].

[10] Microsoft Corporation, "Entity Framework Core Docs," [Online]. Available: https://learn.microsoft.com/ef/core. [Accessed: 2025].

[11] OWASP Foundation, "OWASP API Security Top 10," [Online]. Available: https://owasp.org/www-project-api-security. [Accessed: 2025].

[12] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," Doctoral Dissertation, UC Irvine, 2000.

[13] M. Nagao, "Natural language processing and knowledge," in Proc. 2005 Int. Conf. NLP and Knowledge Engineering, 2005, doi: 10.1109/NLPKE.2005.15986