http://dx.doi.org/10.18623/rvd.v23.5444

DEVELOPMENT OF A BATCH PROCESSING APPLICATION WITH ZERO-DOWNTIME DEPLOYMENT

DESENVOLVIMENTO DE UM APLICATIVO DE PROCESSAMENTO EM LOTE COM IMPLEMENTAÇÃO SEM TEMPO DE INATIVIDADE

Article received on: 11/18/2025
Article accepted on: 2/13/2026

Süleyman Sarp Koç*
*Innovance Information Technologies, Dept of Software Development, Istanbul, Turkey
Orcid: https://orcid.org/0009-0007-0055-7457
sarp.koc@innovance.com.tr

Mustafa Yanar*
*Innovance Information Technologies, Dept of Software Development, Istanbul, Turkey
Orcid: https://orcid.org/0009-0002-3108-4676
mustafa.yanar@innovance.com.tr

Ceren Ulus**
**EFA Innovation Consultancy, Dept of R&D, Adana, Turkey
Orcid: https://orcid.org/0000-0003-2086-6381
f.cerenulus@gmail.com

Mehmet Fatih Akay***
***Çukurova University, Department of Computer Engineering, Adana, Turkey
Orcid: https://orcid.org/0000-0003-0780-0679
mfakay@cu.edu.tr

The authors declare that there is no conflict of interest

Abstract
In today's era of rapidly advancing software and information technologies, many operational processes involve repetitive tasks, requiring employees to routinely spend time completing them. Such situations significantly reduce work efficiency and lead to wasted time. This study aims to increase operational efficiency and enable employees to focus on more strategic and creative tasks. To this end, a batch processing system has been developed that is easy to use, automates routine operational and software processes, and provides uninterrupted service. This system is built on an architecture of admin and agent modules to coordinate and manage workloads. Angular and TypeScript have been used for front-end, while Python, FastAPI, SQLAlchemy, and PostgreSQL technologies have been used for back-end. Development and packaging tools such as Jenkins and Docker have been utilized. Sonarqube integration has been preferred for static code analysis. A hybrid transition architecture combining "Graceful Shutdown" and "Missed Job Recovery" approaches has been designed for uninterrupted service delivery and data security. With the developed system, processes have been made

Resumo
Na era atual, marcada pelo rápido avanço das tecnologias de software e da informação, muitos processos operacionais envolvem tarefas repetitivas, exigindo que os funcionários dediquem tempo rotineiramente à sua execução. Tais situações reduzem significativamente a eficiência do trabalho e levam à perda de tempo. Este estudo tem como objetivo aumentar a eficiência operacional e permitir que os funcionários se concentrem em tarefas mais estratégicas e criativas. Para isso, foi desenvolvido um sistema de processamento em lote que é fácil de usar, automatiza processos operacionais e de software rotineiros e oferece um serviço ininterrupto. Este sistema é construído sobre uma arquitetura de módulos de administração e de agente para coordenar e gerenciar cargas de trabalho. Angular e TypeScript foram utilizados para o front-end, enquanto as tecnologias Python, FastAPI, SQLAlchemy e PostgreSQL foram utilizadas para o back-end. Ferramentas de desenvolvimento e empacotamento, como Jenkins e Docker, foram utilizadas. A integração com o Sonarqube foi preferida para a análise estática de código. Uma arquitetura de

more effective and efficient through the automation of planned workflows. In this way, employees have been able to focus their time and energy on high value-added tasks, thus significantly increasing workforce productivity.

Keywords: Repetitive Task Automation. Zero Downtime. Batch Processing Application.

transição híbrida, combinando as abordagens de "Desligamento Gradual" e "Recuperação de Tarefas Perdidas", foi projetada para garantir a prestação ininterrupta de serviços e a segurança dos dados. Com o sistema desenvolvido, os processos tornaram-se mais eficazes e eficientes por meio da automação de fluxos de trabalho planejados. Dessa forma, os funcionários puderam concentrar seu tempo e energia em tarefas de alto valor agregado, aumentando significativamente a produtividade da força de trabalho.

Palavras-chave: *Automação de Tarefas Repetitivas. Tempo de Inatividade Zero. Aplicação de Processamento em Lote.*

1 INTRODUCTION

In today's dynamic business world, new workloads are constantly emerging. These new workloads may include different operational tasks, and these processes often involve many repetitions. Especially in large projects, tracking and coordinating tasks is a frequently encountered challenge. Manually tracking which tasks are at which stage, how they are progressing, and who is responsible for them can lead to problems stemming from communication deficiencies. This situation leads to disruptions in business processes and creates a breeding ground for loss of coordination and misunderstandings within the team.

On the other hand, problems arising from manual tracking can also cause service interruptions during maintenance or updates. Such situations can disrupt customer transactions, lead to user dissatisfaction, and damage brand image. Minimizing downtime contributes to the uninterrupted continuation of business processes. Uninterrupted access to systems allows employees to work more efficiently, thus reducing time loss. Additionally, reduced downtime lowers data loss and financial damage, resulting in significant cost optimization and helping to keep maintenance costs under control. This increases customers' ability to access services at all times, maximizing customer satisfaction and trust [1].

Batch processing allows tasks to be accumulated at specific intervals rather than in real time, making it possible to process them all together. This approach minimizes

user interaction and enables the processing of large volumes of data with high efficiency [2]. This study aims to develop a solution that increases operational efficiency and allows employees to focus on more strategic and creative tasks. To this end, a batch processing system has been developed that automates routine operational and software processes, is easy to use, and provides uninterrupted service. This system has been built on an architecture of administrator (admin) and staff (agent) modules to coordinate and manage workloads.

This study addresses three fundamental engineering problems in distributed batch processing systems that have long been known but remain complex to solve in practice. These are defined as achieving zero-downtime update transitions on a distributed architecture, providing automatic task handover and fault tolerance within dynamic and dependent workloads, and modeling the time, resource, and state-based synchronized management of tasks in a scalable manner.

The most complex aspect of the zero-downtime update problem is that it requires updating not only application services but also stateful processes and data-dependent tasks on the live system. In this study, a custom control layer has been developed that monitors changes in process state, maintains data integrity between old and new versions, and ensures the completion of the transferred task.

Dynamic task handover and fault tolerance of agent modules in distributed systems is another technical challenge widely discussed in the literature but not standardized in practice. Within the developed system, each worker can report their health status to their manager at specific intervals. The problem of scheduling tasks and managing them according to dependencies cannot be solved flexibly enough in classic cron-based systems. In particular, there are limited examples in the literature of combinations such as resource sharing among dependent tasks, anomaly detection, and event-driven operation. In this study, a unique orchestration engine has been developed for dynamic priority switching between tasks, an event-trigger and schedule hybrid operation structure, and task lifecycle control. This engine, as an alternative to the Directed Acyclic Graph (DAG) structure, can generate and run multi-dimensional job graphs where processes are managed together based on time/event and state. With the developed system, this study stands out as a work that addresses many gaps in the

literature, with few examples in terms of architectural flexibility, dynamic process management, fault tolerance, and real-time observability.

This study is organized as follows: The relevant literature is presented in Section 2. Following this, details of the system is persented in Section 3. The results of the study has been given in Section 4. Finally, the conclusion of the study is included with Section 5.

2 LITERATURE REVIEW

[3] examined the evolution of Extraction-Transformation-Loading (ETL) from traditional to automated ETL. The benefits of automation, such as scalability, cost efficiency, and consistency, have been analyzed. Challenges such as data security and tool customization have also been highlighted.

[4] aimed to provide an abstract introduction to the necessary components for a successful Business Process Automation (BPA) implementation and to demonstrate the scope and diversity of BPA. To this end, they presented a Systematic Literature Review examining the current state of research on frameworks, key factors, and technologies in the context of BPA. 40 studies collected and reviewed between 2019-2024 identified 24 approaches, 32 key success factors, and the most commonly used technologies in a BPA environment.

[5] focused on applications in modern cloud-based environments, examining various approaches such as incremental traffic shifting, blue-green deployments, and phased releases. The study addressed critical aspects such as data mechanisms, service network architectures, automatic consistency rollback systems, and regulated deployment pipelines. Real-world applications have been examined. The study demonstrated how businesses can achieve seamless transitions while maintaining service availability in distributed environments.

[6] addressed the challenges of detecting coordinated malicious activity across different online platforms. They presented a distributed batch processing architecture for large-scale cross-platform abuse detection. The proposed architecture enabled the integration of platform-specific preprocessing and cross-platform feature normalization through a modular design that separates data collection, preprocessing, distributed

processing, and result merging components. A dynamic batch processing strategy has been implemented that optimized computational resource utilization while keeping detection latency within acceptable limits. A multi-task learning approach with specialized deep learning models for different abuse types and platform-aware adversarial coding have been used to learn platform-independent representations. A dataset has been created by scooping 3.2 million content items from five major platforms. The results show that the proposed approach provides a 12.7% improvement in the cross-platform F1 score compared to platform-specific models and delivers 2.8 times higher throughput compared to simple cross-platform approaches.

[7] focused on the problem of efficiently handling dynamically changing task sets over time in time-triggered systems. The Dynamic Task Set Scheduler (DTSS) algorithm, which accepts, removes, and assigns tasks, and provides local time-triggered applications, has been proposed. The proposed algorithm is based on a rigorous feasibility analysis specifically tailored for dynamically changing task sets. A minimum and sufficient number of constraints have been used for the feasibility analysis. This kept the computational load of DTSS low. In experimental results, significant improvements have been achieved compared to the most advanced methods for the same execution scenarios

[8] analyzed methodologies and best practices for implementing zero-downtime deployments in Kubernetes clusters. Mechanisms provided by Kubernetes, such as incremental updates, blue-green deployments, canary releases, and automatic rollback strategies, have been examined. An approach that minimizes service disruption during software updates has been presented. Kubernetes-specific structures such as Deployment Objects, ReplicaSets, readiness and liveness checks, and service abstractions have been used to provide seamless traffic routing and fault-tolerant updates. Amazon EKS clusters running both stateless and stateful applications have been utilized. Experiments simulating high-load traffic, network latency, and node failures have been conducted. The results showed that the proposed approach successfully maintained service continuity, reduced failed request rates, and enabled rapid rollback in case of deployment anomalies.

[9] analyzed Blue-Green Deployment, Canary Releases, and Feature Flag Management deployment techniques that enable secure and phased deployments in cloud-based environments. This has been combined with experimental evaluations performed on a Kubernetes-based Continuous Integration / Continuous Delivery (CI/CD) pipeline

using tools like Spinnaker, Argo Rollouts, and LaunchDarkly, by examining industry applications. Performance metrics such as latency, failure rates, rollback time, and user impact have been analyzed under simulated failure conditions to assess the robustness of the strategies. The results showed that Blue-Green deployments require more resources while providing fast rollback and high environment isolation. Canary deployments offered higher fault detection capabilities with lower user impact, requiring precise traffic control and observability.

[10] examined how businesses, technology, and competitive dynamics have changed over time, and how Enterprise Resource Planning (ERP) I, ERP II, and today's ERP III approaches have been developed. In addition, blockchain-based, cloud-based, and AI-based ERP systems and their impact on business processes and workflow automation for future businesses have been analyzed.

[11] aimed to provide adaptations for time-triggered systems while ensuring that collision and priority constraints are met. To this end, an Artificial Neural Network (ANN) has been applied to learn timelines. In the evaluations performed, the AI-based timer has been compared with traditional scheduling algorithms such as list scheduling and genetic algorithms in terms of completion time and computation time. When the results have been evaluated, it has been seen that the AI-based timer has potential as the complexity of the scheduling problem increases.

[12] examined the role of Data ETL in scalable business intelligence systems. The Preferred Reporting Elements for Systematic Reviews and Meta-Analyses (PRISMA) framework has been followed. 63 literature studies have been identified and reviewed. The results showed that ETL pipelines, once predominantly focused on batch processing, have expanded towards ELT and streaming paradigms with cloud-based data warehouses and distributed architectures. The literature reviewed revealed central requirements for data quality, metadata management, and pedigree, as well as BI reliability. The results highlighted the distinct strengths of commercial platforms such as Informatica, IBM DataStage, and Microsoft SSIS, as well as the flexibility and cost-effectiveness of open-source frameworks such as Talend, Pentaho, and Apache NiFi. Cloud-based services such as AWS Glue, Azure Data Factory, and Google Dataflow have been shown to integrate scalability and governance into serverless pipelines, while innovations like Apache Spark and Delta Lake provide Atomicity, Consistency, Isolation, Durability (ACID)-compliant

lakehouse capabilities for enterprise analytics. The study observed that ETL pipelines, along with their technical workflows, are socio-technical infrastructures that support BI scalability, enterprise trust, and regulatory legitimacy in global data environments.

[13] examined deployment strategies in a containerized full-stack system incorporating modern Blue-Green deployment and Canary deployment. Furthermore, an architectural model utilizing service network technologies, health checks, and traffic routing has been proposed to mitigate deployment-related risks. An experimental case study demonstrates how incremental validation and phased deployment mechanisms improve system scalability, fault tolerance, and resilience. The study utilized automated infrastructure-code implementations and CI/CD pipelines. Experimental results showed that the proposed solution was able to reduce latency and error rates during transitions, achieving a significant impact in multi-service environments.

[14] compared big data architectures and provided solutions for log data acquisition, enrichment, analysis, and visualization. A complete solution has been compared with a custom-built stack. Datadog, Grafana Loki, and Elastic 8 Stack have been used. These three systems have been implemented and compared according to the requirements. It has been concluded that all three systems are suitable for big data logging and meet most of the requirements, but they exhibit different capabilities when implemented and used.

[15] proposed an ETL framework that integrates data quality through "process history" to improve information processes in the data warehouse development process. The framework has been compared with other frameworks, and its advantages and disadvantages have been evaluated.

[16] presented a simulation-based evaluation of zero-disruption deployment frameworks specifically designed for microservice architectures in financial environments. Using controlled simulation environments, they assessed the feasibility of integrating advanced traffic management, stateful rollback mechanisms, and regulatory compliance validation to achieve true zero-disruption deployments without compromising data integrity or audit requirements. The approach utilized synthetic financial transaction datasets, standardized microservice deployment scenarios, and automated compliance validation protocols to evaluate deployment strategies for financial applications. ACID features have been preserved across distributed operations.

To fully verify regulatory compliance, a controlled test environment simulating complex multi-service updates has been developed. In the simulation study, deployment performance has been evaluated in multiple financial application scenarios, including payment processing, trading systems, and regulatory reporting applications. Experimental results showed a simulated uptime of 99.999% during deployments, a 67% reduction in average deployment time, and rollback within 30 seconds.

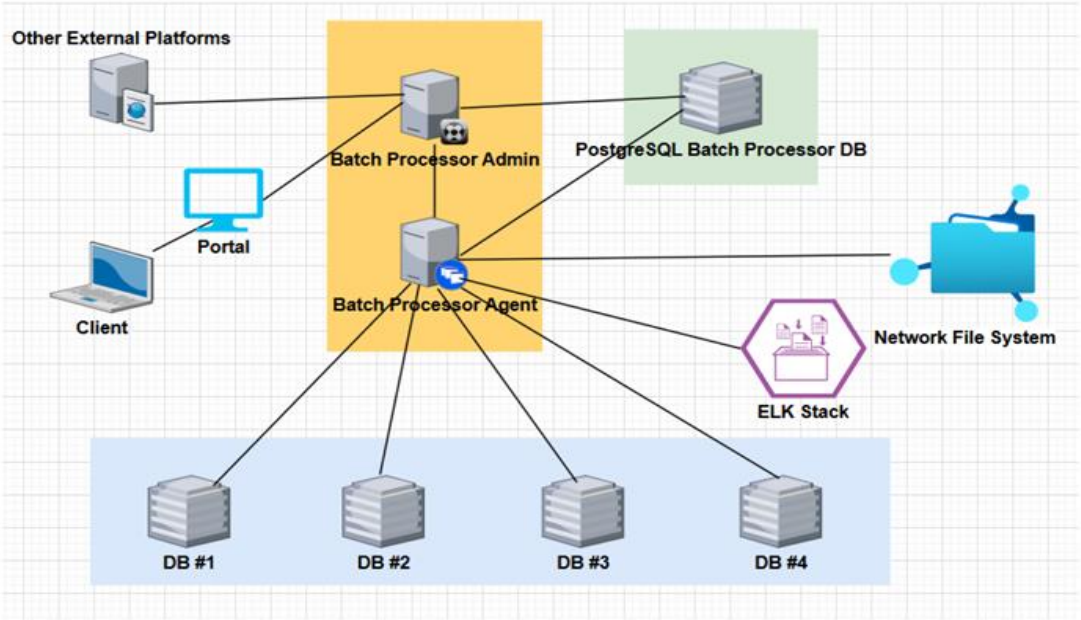
[17] presented a theory-based simulation model that investigates how Information Systems (IS) architecture evolves and what results it produces in various organizational types. In the simulation model, enterprise theory has been used to capture organizational types shaped by different combinations of coercive, normative, and imitative pressures. Complex adaptive systems theory has been used to model the evolutionary and emerging nature of IS architecture. Insights obtained from the simulation experiments have also been summarized. Based on the simulation model and theoretical insights, implications for research and application have been evaluated.

3 DETAILS OF THE SYSTEM

Angular and TypeScript, Python, FastAPI, SQLAlchemy, and PostgreSQL have been used in the system development. In addition, development and packaging tools such as Jenkins and Docker have been used. Sonarqube integration has been preferred for static code analysis. Furthermore, a combination of Elasticsearch, Logstash, and Kibana has been used for logging and error detection. The system architecture is shown in Figure 1.

Figure 1

The system architecture



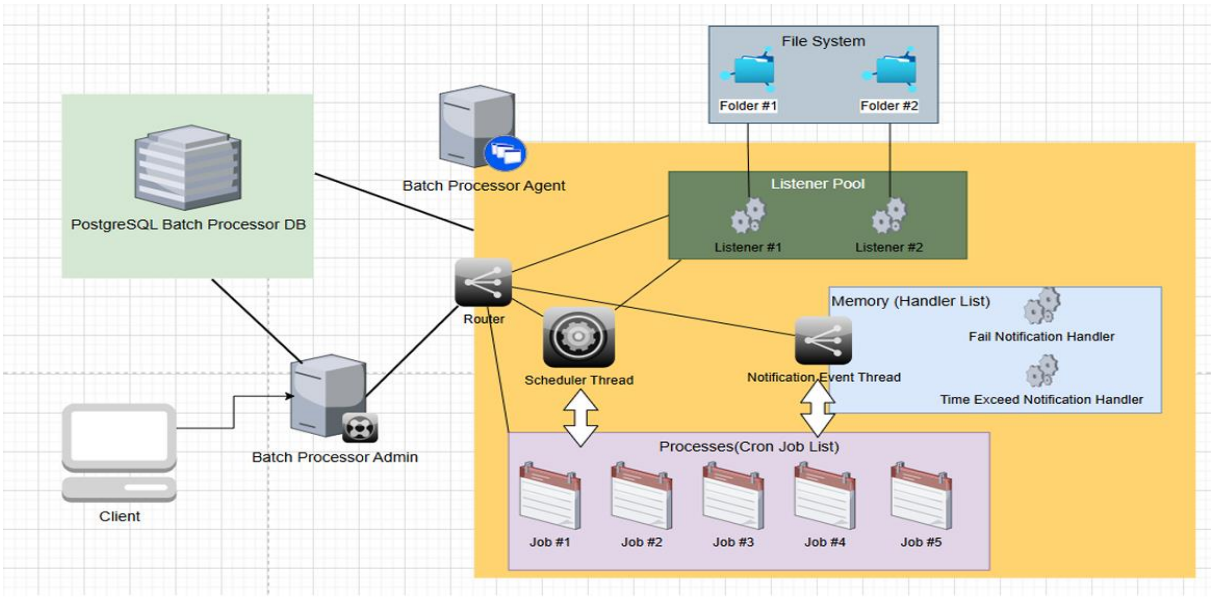
The Batch Processor Admin is the core unit that acts as the brain, the primary point of contact for users. It is responsible for processing records into the database and notifying the Agent module. Communication between the Admin and Agent modules is established via the HTTP protocol. The Batch Processor Agent is the unit that executes the process. Based on requests received from the Admin, it creates the workflow, controls the order, timing, and execution of tasks, and initiates them. This unit manages multiple threads. It also connects to the file system. The PostgreSQL Batch Processor Database is the database where all the essential data used jointly by the Admin and Agent modules is stored. It holds all job definitions, scheduler definitions, listener definitions, and job history records.

3.1 Batch processor agent

The Agent Module is shown in Figure 2.

Figure 2

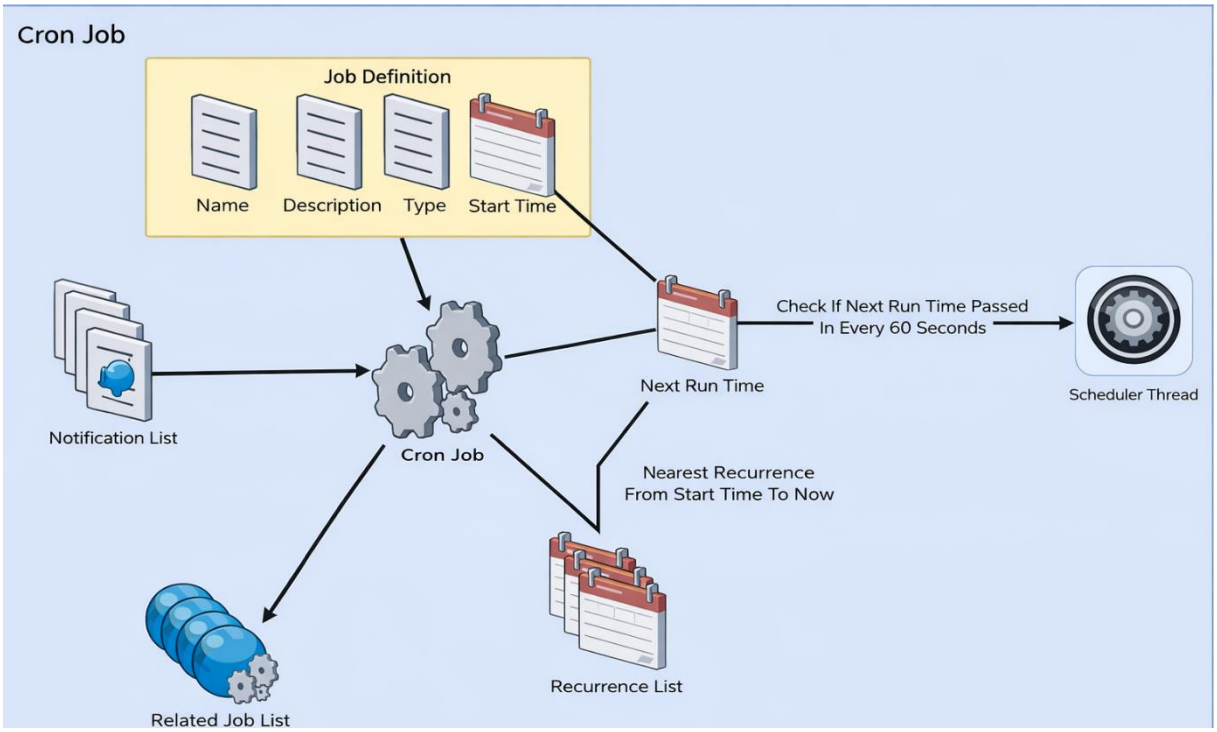
The Agent Module



The scheduler thread is the main event thread that operates on a tick logic, checking every minute for jobs that need to be started. It controls folder listening objects and job schedulers. It starts jobs by creating a new process, thus abstracting jobs from each other and preventing them from interfering. The notification even thread is a helper thread that operates on a messaging queue logic. All job processes send one-way messages to the commonly defined notification queue object, ensuring that user-defined notification types are triggered and notified to the user when needed. It sends an email containing the event details via Simple Mail Transfer Protocol (SMTP). The router manages the Application Programming Interface (API) endpoints and implements the relevant requests received from the Admin module. The listener pool is a collection where listener objects that monitor folder and database changes are stored collectively. Schedulers are stored in the cron job list. Each cron job object is responsible for running the jobs they are referenced to when the time comes. The detailed representation of the cron job is given in Figure 3.

Figure 3

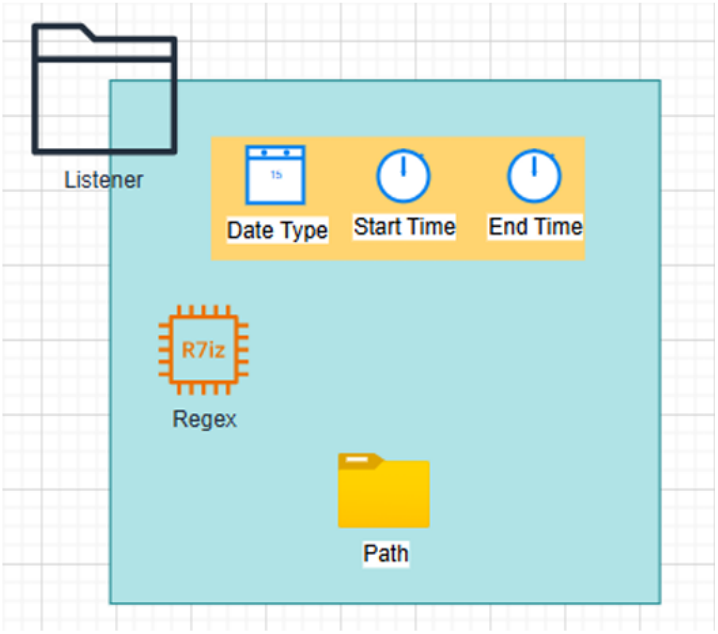
The detailed representation of the cron job



Each job has a customizable system. It can be connected with different notification types if desired, or it can be configured to run before or after different jobs. Additionally, each job thread updates its next run time as needed, allowing for quick checking of whether the run time has arrived. The details of the listener structure are given in Figure 4.

Figure 4

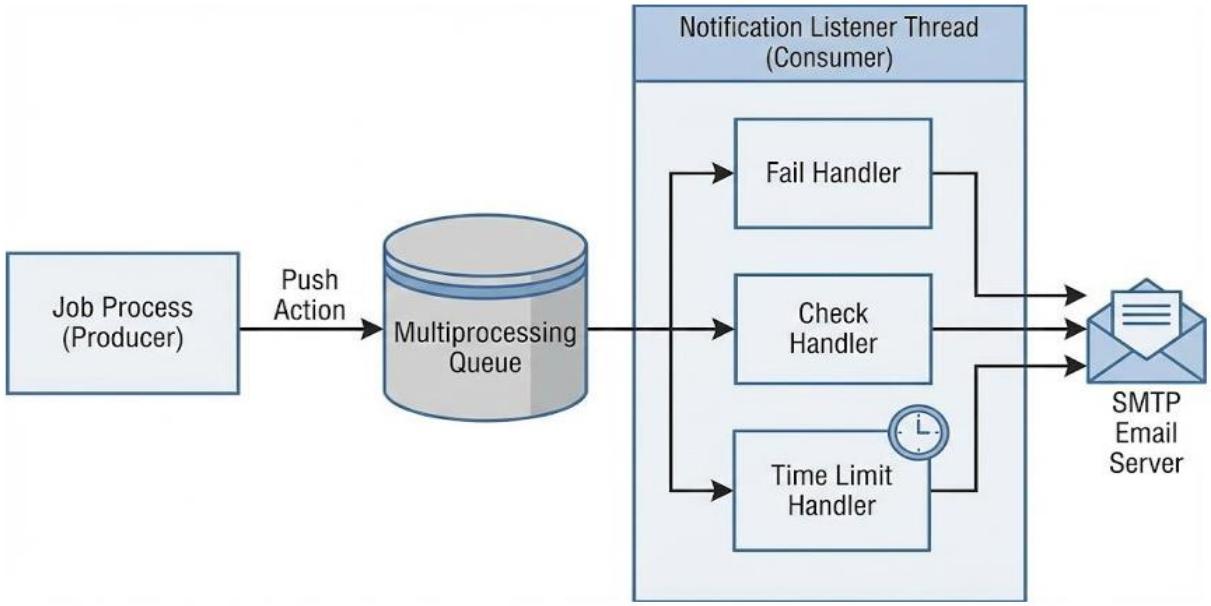
The details of the listener structure



If within the specified hours, a file name scan is performed in the specified folder path using the specified regex. The job is initiated if more files than the specified number and size are found. To prevent unwanted situations such as the initiation of multiple identical jobs, the names of the processed files can be marked as "already processed" in the database if desired. Furthermore, a notification is sent to the relevant person using an alarm definition specified during the configuration process. The notification system is shown in Figure 5.

Figure 5

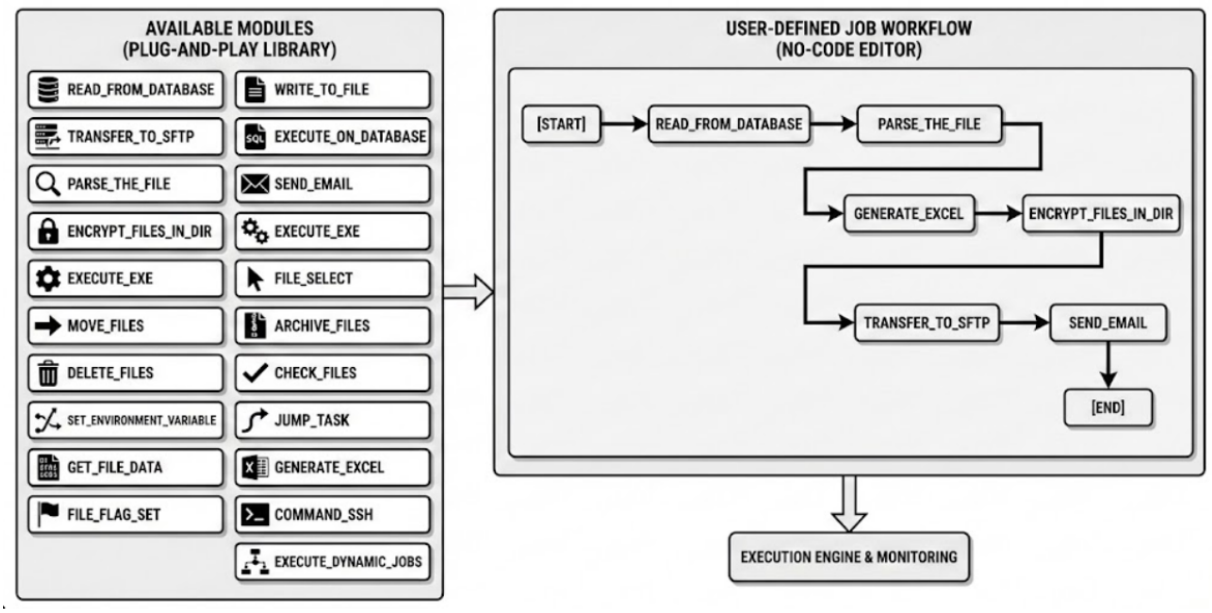
The notification system



The notification system operates using a queue logic. There is a common listener, a notification helper thread, for all jobs. Jobs send a message about the relevant status to the queue during certain pre-defined events. The incoming message is processed in the main notification event loop, and if there is a notification type associated with the job, the relevant notification implementation is carried out. Status information is sent to the email addresses entered during notification definition. Available job types and an example workflow definition are given in Figure 6.

Figure 6

Available job types and an example workflow definition



The system's core working unit consists of plug and play job definitions. Job definitions are created by optionally using and sequencing 21 pre-implemented task types in the project. Each task has its own specific definitions. The user enters these definitions when creating a job definition, and when the job is started, it runs according to these definitions and sequence. An in-memory data structure is used to facilitate data exchange between task snippets. For example, the MOVE_FILES task, used after the FILE_SELECT task, reads the file selected in the previous task from a shared memory object and moves it to the specified location. Modules also establish mandatory contexts between themselves; some tasks require the use of a specific other task beforehand. For example, to use the MOVE_FILES task snippet, the FILE_SELECT task snippet must be used beforehand. Task snippets like READ_FROM_DATABASE determine which database to retrieve information from using the fields they receive during definition, and data retrieval is performed in chunked form depending on the size of the data. Additionally, the workflow can be managed in three different ways in case of an optional exception specified in the job descriptions: continuing the workflow, resulting in an error, and waiting for manual user interaction. In cases managed by waiting for manual user

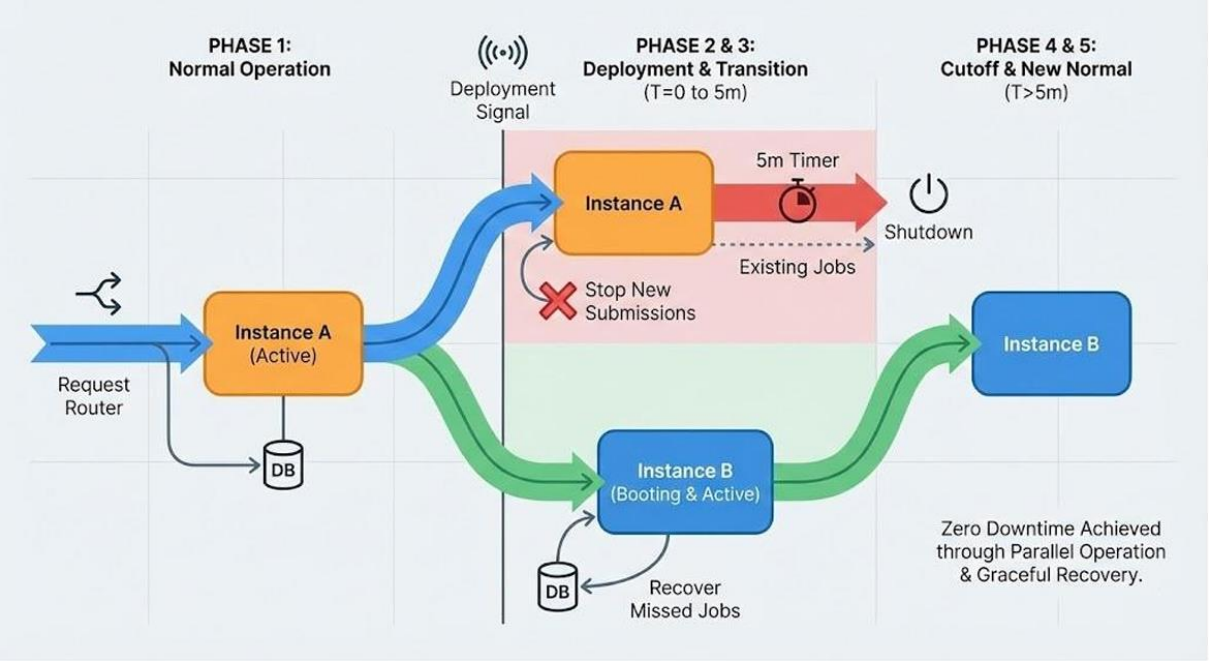
interaction, the events in the workflow are recorded, and the workflow can be resumed from where it left off, depending on the user's decision.

3.2 Zero-interruption batch processing system

A representation of a zero-interruption distribution system is given in Figure 7.

Figure 7

A representation of a zero-interruption distribution system



A hybrid transition architecture combining "Graceful Shutdown" and "Missed Job Recovery" approaches has been designed to ensure uninterrupted service delivery and data security. This process operates in four key steps: draining mode, timed shutdown, instantaneous load transfer, and job recovery.

3.2.1 Draining mode

The current server (Instance A), upon receiving the shutdown signal, stops accepting new jobs. However, processes already in progress are allowed to complete in their natural flow without interruption.

3.2.2 Timed shutdown

The system is given a 5-minute tolerance period to complete the current tasks. If the tasks are completed early, the server shuts down immediately; if the time expires, the remaining tasks are gracefully terminated and the server is disabled.

3.2.3 Instantaneous load transfer

While the old server is in the process of shutting down, the new server (Instance B) simultaneously goes live. All incoming requests are immediately routed to this server, preventing service interruptions.

3.2.4 Job recovery

The new server scans the database as soon as it goes live. During the 5-minute transition window, it identifies scheduled tasks that couldn't be started because the old server has been in "evacuation" mode and automatically runs them. With this architecture, version updates are performed without any data loss or service interruption during batch processing operations.

Agent endpoints are not used by users; they only allow the Admin module to communicate with the Agent module and ensure that database changes are reported, updating and recalculating the relevant locations in the application's memory.

3.3 API endpoints

3.3.1 Admin job

The create job option can be used to create a job description. There are two main job types: VAU (custom job) and GENERIC_JOB (fully parameterized). The request body varies slightly depending on the type. Get all jobs retrieves all jobs with optional pagination and tag filtering. Jobs can be filtered using OR logic with one or more tag IDs. This will return jobs associated with any of the given tags. “Get Jobs by Client” shows

all jobs associated with the specified organization. It also provides information about when the jobs have been last run and their current status. “Get Job Detail” contains detailed information about the job with the specified ID. It shows when the job has been last run, whether it has been successful in its last run, information about the tasks it performed, and the repetitions it is associated with. “Get Job Tracker by Client” shows when jobs have been last run and provides detailed status information about the tasks associated with the jobs according to the given organization.

“Add Tags to Job” operation allows you to assign one or more labels to a specific job. This endpoint is designed to be idempotent; meaning that adding the same label again will not generate an error and will ignore existing associations. The system only adds labels that are not yet present in the relevant job. “Remove Tag from Job” operation removes the link between a job and a label. This endpoint is also idempotent; if the specified label is not already associated with the relevant job, no error will occur and the operation will be successfully completed. This operation only removes the job-label relationship; the label record is not deleted from the system.

3.3.2 Job monitor

Get Job Monitor Tracker shows all available actions for the specified job.Get Job Monitor Tracker by Update Date shows all available actions for the given job sorted by update date.Get Job Monitor Client Status shows the status of jobs associated with an organization. If the "Daily" option is selected, it shows jobs for the current day. The "Weekly" option shows the status weekly. The response includes job statuses such as completed, running, and failed.Get Job Monitor History provides statistical data for all execution statuses of the job based on the given job name.

3.3.3 Job relation

Create Job Relation establishes relationships between jobs. If one job needs to run before another, this is specified with this endpoint.

3.3.4 Scheduler

Enable Scheduler enables the agent scheduler. Restart Scheduler properly restarts the agent scheduler.

3.3.5 Notification

The notification section specifies the person to whom emails regarding the progress of the work should be sent.

3.3.6 Recurrence

Create Recurrence creates a recurrence for a job. In repetition, it is possible to determine when a task will start and how often it will run. Custom repetition types can also be used.

3.3.7 Listener

Listeners provide a powerful mechanism for automatically triggering jobs based on external conditions such as file existence or database changes. They continuously monitor the specified resources and automatically start associated jobs when the conditions are met.

Create listener creates a new listener that monitors a specified resource (file system or database) and automatically triggers the associated task when conditions are met. Listeners can be configured with various scheduling options, including time intervals, intervals, and custom repetition patterns. Create Multiple Listeners creates multiple listeners in a single request. Useful for setting up multiple monitoring configurations for a job or across multiple jobs. Get Listeners by Job ID retrieves all listeners associated with a specific job. Useful for viewing all monitoring configurations for a specific job. Get Listener by ID retrieves all listeners associated with a specific job. Get All Listener retrieves all listeners in the system. Update Listener updates the configuration of an existing listener. All parameters in the creation endpoint are

modifiable. Delete Listener deletes a listener. This stops monitoring and permanently removes the listener configuration. Enable Listener enables a listener that has been previously disabled. The listener will restart monitoring according to its configuration. Disable Listener is used to disable a listener in this endpoint.

3.3.7.1 File flag integration

When `flag_activated duplicate processing: true` is set, the listener integrates with the automatic processing system and works as follows: Automatic Processing Control: Before a job is triggered, the listener checks whether matching files are already marked as processed in the database. Alert System: If processed files are found in the monitoring directory, an alert email is sent instead of triggering the job. Alert Content: Alerts contain details showing which files have been processed and which have not, helping administrators identify potential problems.

3.3.7.2 Dynamic alert email configuration

The `alert_mail` field supports multiple configuration options; in simple email usage, a single recipient address can be defined, addresses for multiple recipients can be specified separated by commas, and in more advanced scenarios, a SQL query can be used that generates dynamic recipient lists depending on the database configuration and returns email addresses.

3.3.7.3 Regex list support

The Regex_list field supports multiple patterns; while using a single pattern, the expression [".*.txt$"] matches all .txt files, multiple patterns can be defined to match different filenames such as [".cecpm..txt$", ".cecpm2..csv$"], and patterns can also include environment variable substitutions for more flexible and dynamic matching.

Additionally, several other endpoints are available. Environment Variable allows the creation of environment variables that can be used in jobs for dynamic value calculation at runtime. Encryption Key imports a Pretty Good Privacy (PGP) key for a

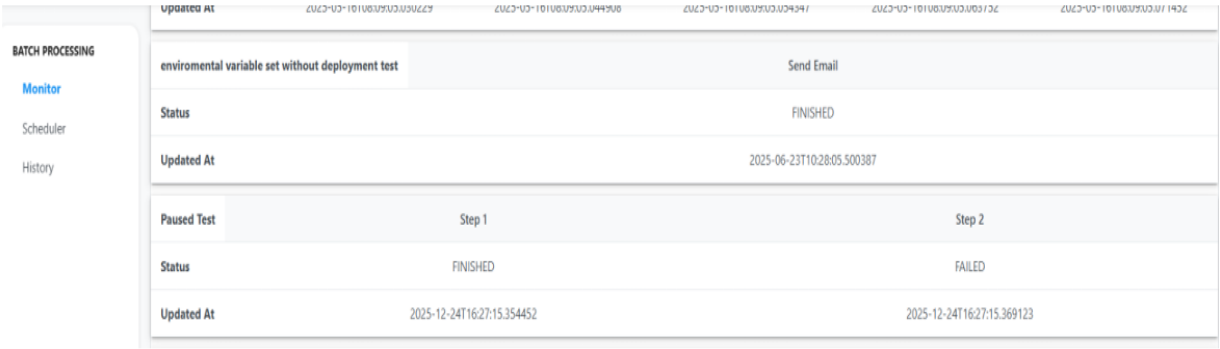
specific organization and enables file encryption features. File Flag retrieves all file flags that track the processed file status. Process Email retrieves all process email configurations. Tag retrieves all tags with optional search filtering. Tag function is useful for autocompletion and listing all available tags. In the query parameters, the “skip” parameter specifies the number of records to skip for pagination (default: 0). “Limit” is the maximum number of records to return (default: 100). “Search” is an optional search term to filter tags by name (case-insensitive partial match). Holiday retrieves all holiday definitions used in business planning.

3.4 Portal task tracking and viewing

The screen view where completed tasks can be tracked is shown in Figure 8.

Figure 8

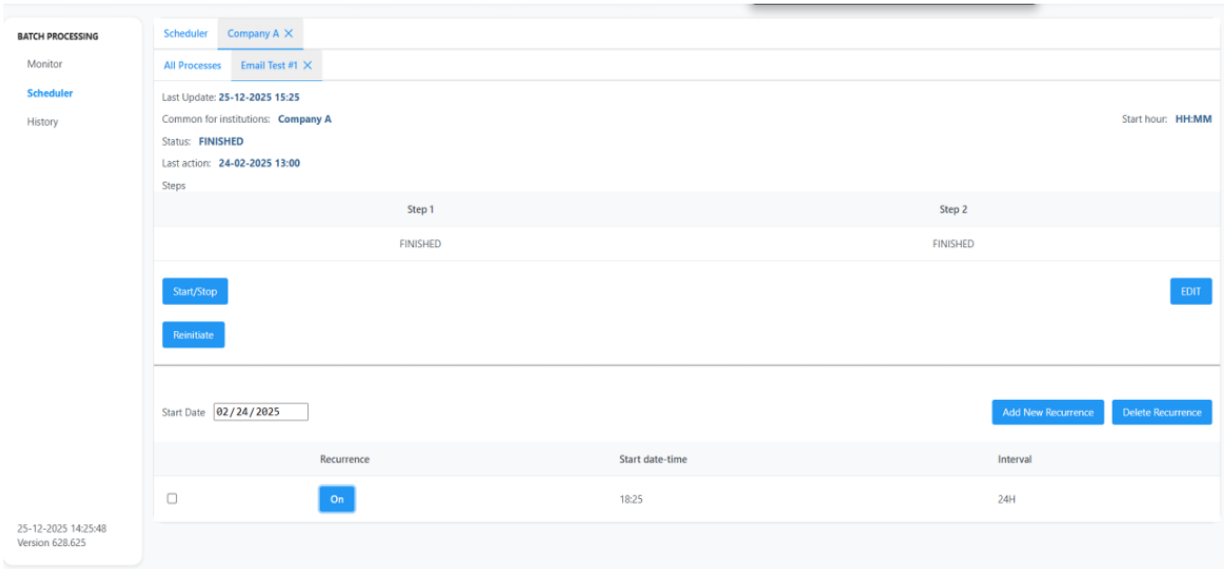
The screen view where completed tasks can be tracked



The screen where the timer and other functions of job descriptions can be customized is shown in Figure 9.

Figure 9

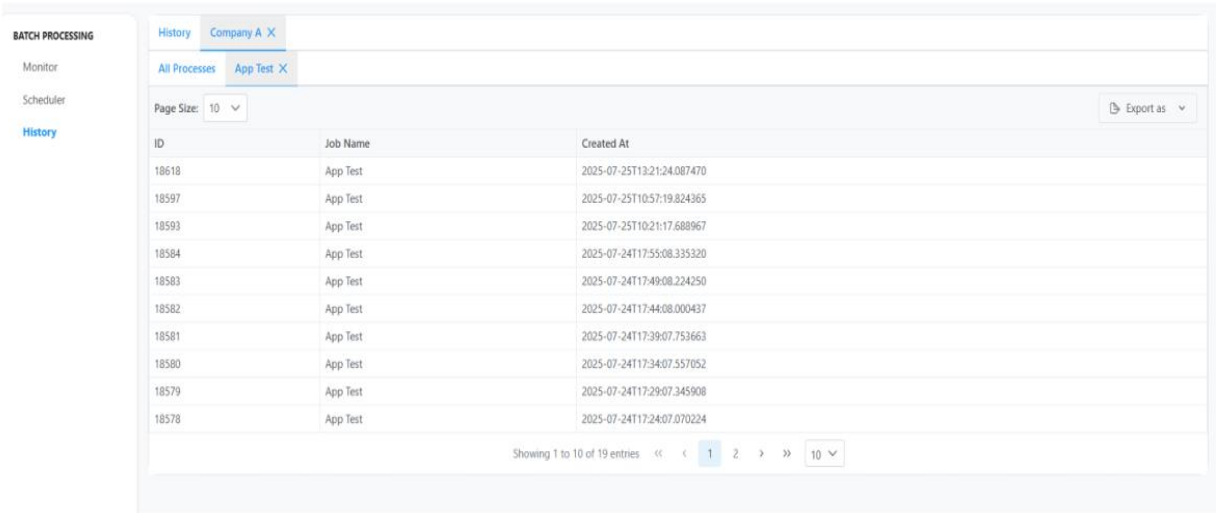
The screen where the timer and other functions of job descriptions can be customized



The screen view showing the execution history of the jobs is given in Figure 10.

Figure 10

The screen view showing the execution history of the jobs



As shown in Figures 8, 9, and 10, tasks are monitored and controlled via a portal. The portal application allows viewing and editing tasks and schedulers defined in the batch processor. It also provides access to the execution history of all tasks. It shows how

each task completed in the last execution of a job. It is also possible to trigger tasks manually. There are three menus. The Monitoring lists the last execution status of all tasks. All tasks can be configured and customized from the Scheduler screen. The History lists the execution history records and statistics of all tasks.

4 RESULTS OF THE STUDY

With the developed system,

- A low-code solution is provided that can be used without knowledge of an additional programming language.
- Each thread updates its next work schedule as needed, allowing for quick checking of whether the work schedule has arrived.
- Version updates are performed in batch processing operations without any data loss or service interruption.
- Job definitions, schedulers, and other processes can be customized within the system.
- Processes have been made more effective and efficient through the automation of planned workflows.
- Employees have been able to focus their time and energy on high value-added tasks,
- The labor productivity has been significantly increased.
- The ability to track processes through a single platform has strengthened communication and coordination between units.

5 CONCLUSION

In today's business world, transaction volumes have increased, leading to the emergence of repetitive tasks. Automating these repetitive tasks is a significant challenge. Furthermore, problems arising from manual tracking cause service interruptions during maintenance or updates, negatively impacting customer satisfaction. This study develops a batch processing system built on a manager and agent module architecture to coordinate and manage workloads. This system automates routine operational and software

processes, providing ease of use and uninterrupted service. The developed system enables the automation of planned workflows, making processes more effective and efficient.

REFERENCES

[1] Kasim, T., Haracic, M., & Haracic, M. (2018). The improvement of business efficiency through business process management. *Economic Review: Journal of Economics and Business*, *16*(1), 31-43.

[2] Manchana, R. (2020). Operationalizing Batch Workloads in the Cloud with Case Studies. International Journal of Science and Research (IJSR), 9(7), 2031-2041.

[3] Ravi, C. (2025). ETL (Extract, Transform & Load) Automation. *International Journal of Emerging Trends in Computer Science and Information Technology*, *6*(1), 52-55.

[4] Rüeck, L., Auer, T., Rösl, S., & Schieder, C. (2025, July). A Systematic Literature Review on Business Process Automation Frameworks and Technologies. In *International Conference on Subject-Oriented Business Process Management* (pp. 198-213). Cham: Springer Nature Switzerland.

[5] THARUN, D. (2025). Zero-Downtıme Mıgratıon Strategıes For Large-Scale Dıstrıbuted Servıces. *Internatıonal Journal*, *11*(2), 179-187.

[6] Wang, H., Qian, K., Ni, C., & Wu, J. (2025). Distributed batch processing architecture for cross-platform abuse detection at scale. *Pinnacle Academic Press Proceedings Series*, *2*, 12-27.

[7] Alkoudsi, M. I., & Fohler, G. (2024, November). Scheduling Dynamic Task-Sets in Time-Triggered Real-Time Systems. In *Proceedings of the 32nd International Conference on Real-Time Networks and Systems* (pp. 48-58).

[8] Petrova, E. V., & Bennett, T. R. (2024). Implementing Zero-Downtime Deployments on Kubernetes.

[9] Ramaswamy, Y. (2024). Zero Downtime Deployments in DevOps: Blue-Green, Canary, and Feature Flag Techniques. *International Journal of Communication Networks and Information Security*, *16*(5), 969-977.

[10] Watson III, E. F., & Schwarz, A. H. (2023). Enterprise and business process automation. In *Springer Handbook of Automation* (pp. 1385-1400). Cham: Springer International Publishing.

[11] Lua, C., Onwuchekwa, D., & Obermaisser, R. (2022, June). Ai-based scheduling for adaptive time-triggered networks. In *2022 11th Mediterranean Conference on Embedded Computing (MECO)* (pp. 1-7). IEEE.

[12] Mahmud, D., & Ikbal, M. Z. (2022). The role of etl (extract-transform-load) pipelines in scalable business intelligence: A comparative study of data integration tools. *ASRC Procedia: Global Perspectives in Science and Scholarship*, *2*(1), 89-121.

[13] Pappula, K. K. (2022). Containerized Zero-Downtime Deployments in Full-Stack Systems. *International Journal of AI, BigData, Computational and Management Studies*, *3*(4), 60-69.

[14] Tiede, D. (2022). Big-Data Solutions for Manufacturing Health Monitoring and Log Analytics.

[15] Munawar. (2021, October). Extract Transform Loading (ETL) Based Data Quality for Data Warehouse Development. In *2021 1st International Conference on Computer Science and Artificial Intelligence (ICCSAI)* (Vol. 1, pp. 373-378). IEEE.
[16] Paladugu, N. (2021). Zero-Downtime Microservices Deployment Strategies for Mission-Critical Financial Applications. *International Journal of Emerging Research in Engineering and Technology*, *2*(3), 79-88.
[17] Haki, K., Beese, J., Aier, S., & Winter, R. (2020). The evolution of information systems architecture: An agent-based simulation model. *MIS Quarterly*, *44*(1), 155-184.

Authors' Contribution

All authors contributed equally to the development of this article.

Data availability

All datasets relevant to this study's findings are fully available within the article.

How to cite this article (APA)

Koç, S. S., Yanar, M., Ulus, C., & Akay, M. F. (2026). DEVELOPMENT OF A BATCH PROCESSING APPLICATION WITH ZERO-DOWNTIME DEPLOYMENT. *Veredas Do Direito*, 23(5), e235444. https://doi.org/10.18623/rvd.v23.5444