# Building a Real-time Exam Integrity Platform: Leveraging REST API for Deploying Cheating Detection Model Using Extreme Programming

Mochammad Dinta Alif Syaifuddin[1, a)], Fitri Bimantoro[1, b)], Ramaditia Dwiyansaputra[1, c)], Mizanul Ridho Aohana[1, d)] and Noor Alamsyah[2, e)]

[1]*Informatics Engineering Department, Faculty of Engineering University of Mataram, Majapahit Street No. 62, Mataram, 83115 Indonesia.*
[2]*King Mongkut's Institute of Technology Ladkrabang, Bangkok 10520, Thailand.*

[a)] Corresponding author: d.nta.workspace@gmail.com
[b)] bimo@unram.ac.id
[c)] rama@unram.ac.id
[d)] mizanul.aohana@gmail.com
[e)] 67076067@kmitl.ac.th

**Abstract**. Education plays an important role in shaping character of the nation. Along with technological advances, the application of technology in education sector has also been digitized. However, challenges such as cheating during exams are still a major concern in maintaining academic integrity. To anticipate, this research proposes a video surveillance (CCTV) based cheating detection system using the YOLOv5 object detection model integrated in REST API architecture. Focus of this research will be on integration to detect cheating in real-time thus enabling flexible integration into various educational platforms. System development is carried out using the Extreme Programming method which is responsive to changing user needs. Based on the test results of several exam simulation videos, the system achieved a detection accuracy of 43% with an average response time of one second, and was able to better identify cheating cases. Hopefully, this research can be a practical solution in improving academic integrity and a reference for the development of similar systems in the future.

## INTRODUCTION

Education plays an important role in driving the development and progress of a nation. It can improve skills, abilities, and make a person's tendency to be more productive, so that more educated individuals are able to show that education not only drives country's economic growth rate but also for the welfare of its people [1]. Talking about education, it is inseparable from learning experiences as an evaluation of abilities that are often measured through examinations [2]. Through evaluation, educators can find out the extent to which students have received and understood education that has been provided, also what needs to be improved or improved in its implementation.

In current digitalization, many efforts have been made to improve system evaluation in education. One of them is automation testing system by using technology. In its development, technology had a significant impact on education systems around the world on implementing and integrating technology [3]. By utilizing technology, this automation can improve efficiency, objectivity, and integrity in the evaluation process at various levels of education [4]. Therefore, this automated testing system can be utilized by educators in providing maximum and consistent assessment of students and exams.

Integrity is one of the crucial challenges in the assessment of testing systems in education. Through integrity, educational learning outcomes can be accurately measured. This is important in setting clear learning objectives, raising awareness of academic misconduct, building relationships with learners, and designing honest assessments to maintain academic integrity [5].

According to Mushthofa research [6], there were at least 260 examinees in total who had cheated including 14 examinees who had never been caught cheating on an exam. There were 158 examinees who claimed to have cheated 1-5 times, 14 participants did it 6 to 10 times, and the rest had done it more than 10 times. This may be because teachers or invigilators are negligent in their supervision, and this opportunity is used by examinees to commit academic fraud.

Based on these problems, teachers or exam supervisors at least need to have tools to minimize the possibility of cheating during the exam. Through Bimantoro's research recently [7], A model was developed to detect cheating by making the most of Closed-Circuit Television (CCTV) installations in classrooms and machine learning (ML) models that use the YOLOv5 algorithm. This model is considered quite accurate because the YOLOv5 architecture performs well and is relatively small, enabling it to better identify cases when predicting images.

However, broader system integration is possible to facilitate information management. This can be achieved through an approach that allows fraud detection services to be accessed flexibly via a REST API architecture. The YOLOv5 model can then be packaged in an API that communicates with web-, desktop-, and mobile-based monitoring applications. This enables teachers or exam supervisors to access prediction results from the ML model in real time and take immediate action if any indication of cheating is detected.

For implementation, to ensure optimal API performance, a Structured Query Language (SQL) was chosen based database to increase efficiency and security in data management. In addition, the use of Docker also allows the system to run in an isolated environment so that the deployment, scaling, and maintenance processes become more flexible. This approach eliminates system dependency on traditional server-specific configurations and allows the system to run consistently in different environments for both testing and production.

Therefore, based on the above references, this research will focus on developing a REST API to maximize the integration of the ML model with the YOLOv5 algorithm [7] with an API-based data management system. This research is expected to help educators and students in increasing the value of academic integrity in education. The integration will be used as a bridge between real-time application services using the REST API to create a distributed and structured system. To ensure that the information system is successfully run properly, the use of one of the Black Box Testing methods will be used in this research. The API service built will consist of two main functions, namely: (1) service for user information and recording of image detection results; (2) service for displaying image detection results on live recordings.

## LITERATURE REVIEW

The development of this REST API is based on several recent research references. In this research, the integration model for the architecture created uses the latest research reference with the research title “Detection of Exam Cheating Through CCTV Using the YOLOv5 Algorithm” by Bimantoro [7], introduced a YOLOv5-based ML model and YOLOv5s algorithm with an accuracy of 43% to analyze input data from Closed-Circuit Television (CCTV). Model classifies the behaviors into five prediction labels: Cheat Sheet (CS), Exchange Paper (EP), Giving Code (GC), Looking Friend (LF), and Normal (N).

An existing challenge in fraud detection systems is their reliance on manual monitoring or rule-based detection methods, which can be inconsistent and prone to human error. Based on Bimantoro [7], YOLOv5 provides a balance between speed and accuracy, making it very suitable for this application. In addition, its lightweight architecture enables fast inference on live CCTV footage, while its advanced feature extraction increases the precision in detecting suspicious behaviors related to fraud. By utilizing YOLOv5, this research overcomes the limitations of previous models, ensuring more reliable and scalable fraud detection.

In order to effectively structure the development process, this research uses the Extreme Programming (XP) methodology. The main reference for the choice of this methodology is the study titled “Development of Backend Server Based on REST API Architecture in E-Wallet Transfer System” [8], which details how XP can be used to create a REST API in a financial transaction system. This approach emphasizes adaptability and rapid development cycles, making it particularly suitable for this research, which involves continuous refinement and integration testing of ML models.

A review of existing studies shows a gap in research related to the integration of ML models with REST APIs for fraud detection. The closest research to this work is “Design and Implementation of REST API for Predicting the Recitation of the Qur'an using Machine Learning” [9], which focuses on ML-based prediction in REST architecture but does not address the domain of cheating detection. This highlights the novelty and importance of this research, as it bridges this gap by implementing the integration of YOLOv5-based ML models with REST APIs. In addition, the research titled “Development of a REST API for the Rinjani Visitor Application using Extreme Programming” [10] and “Development Of Information Systems For Service And Customer Complaints Using The Extreme Programming Method” [11] are also used as a basis for the implementation of the REST API architecture in this research.

Ensuring the reliability of the developed system requires rigorous testing and security. This research uses Black Box testing, a widely used approach to evaluate system functionality. The research “Blackbox Testing of PT Inka (Persero) Employee Performance Assessment Information System Based on Equivalence Partitions” [12] is the main reference that illustrates the effectiveness of this method in validating the functionality of the company's system. For

security issues, JSON Web Token (JWT) is used as authentication that distributes sensitive data to every services. A research "Security measures implemented in RESTful API Development" [13] was used as a basis in implementing security for this research.

To improve communication efficiency in the system, streaming technology is implemented using a combination of HTTP (HyperText Transfer Protocol) and Redis. Research studies such as "Improving web-oriented information systems efficiency using Redis caching mechanisms" [14], "Open Chemistry: RESTful Web APIs, JSON, NWChem and the Modern Web Application" and "A Service Platform for Real-Time Video Streaming Based on RESTful API in IoT Environments" [15] provide valuable insights into the effective integration of Redis based semi micro service, and JSON to develop a video streaming platform in a REST API architecture. In addition, the use of containerization using Docker is also implemented for better distribution of system communication. Referring to the research "A Dockerized Approach to Dynamic Endpoint Management for RESTful Application Programming Interfaces in Internet of Things Ecosystems" [16] Docker can provide modularity and portability for each service to run independently but still communicate internally.

Based on the description above, this research will focus on the use of Extreme Programming (XP) methodology in developing REST API to integrate ML model with YOLOv5 algorithm. The reason for using this methodology is because of its ease of implementation for a small team of 2 to 4 people. By using the REST API, hopefully it can be developed by further application developers.

## METHODOLOGY

This research uses the Extreme Programming (XP) software development method. A development method that focuses on code development. With this method, the possibility of flexibility that exists can provide users with the suitability of using the method according to their wishes and needs, especially when there are sudden changes [10, 11]. This method has several stages, namely exploration, planning, iteration, and release as shown in "Fig. 1" below.

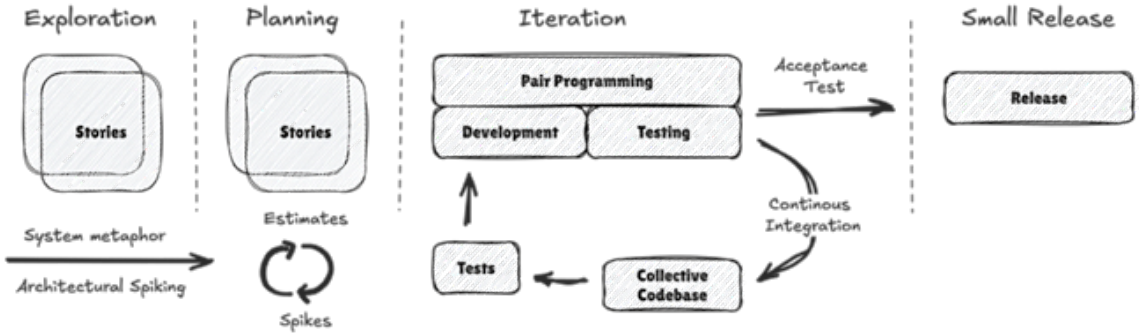



**FIGURE 1.** Extreme Programming Life Cycle

In the exploration stage, the process of data collection and analysis of existing fraud detection methods, identifying their limitations such as reliance on manual surveillance or inefficient rule-based detection and determining how YOLOv5 can address these gaps. Then the Planning stage focused on defining user stories, prioritizing features such as real-time inference and API response efficiency, and estimating development efforts.

During the iterations, the REST API was developed alongside YOLOv5, addressing challenges such as latency, inference speed, and stable streaming. This was accomplished by using asynchronous processing, and pipeline streaming. In the release phase, black-box testing, performance tuning, and preparation for ongoing maintenance. In the release phase, extensive testing including black-box testing was conducted to validate the functionality of the API and ensure the accuracy of YOLOv5 predictions. Performance tuning was also performed to reduce response time and improve system scalability.

In order to facilitate real-time streaming and fast data transmission, this research utilizes Redis Pubsub in combination with REST API for internal communication. Redis Pubsub was chosen because it can transmit fast communication messages in real-time, and its lower latency makes it very suitable for handling continuous video feed analysis.

# RESULT AND DISCUSSION

## Exploration Phase

The exploration phase begins with identifying system requirements based on the identified problem. Through previous research [7], there is already an ML model for fraud detection. Therefore, at this stage, the main focus will include how the integration is done and what technology will be used in the development of the model. In addition, the stages will be divided into sections as follows.

### *User's Usecase*

At this stage, requirements are identified according to user needs. The next step is to create a user use case diagram to describe in detail the relationship between the user and the system. There are several actions in the diagram that include or exclude certain actions. For example, User must sign in to access features like creating and reading live streams, playing streams, managing their profile, and handling reports. Some actions, like playing or creating live streams, extend from reading live data. Profile updates involve updating user info. Meanwhile, Admin can view lists of streams, live sessions, reports, and users. The diagram emphasizes that sign-in is required before accessing other features and separates user and admin responsibilities clearly. Further details, can be seen in "Fig. 2" below.

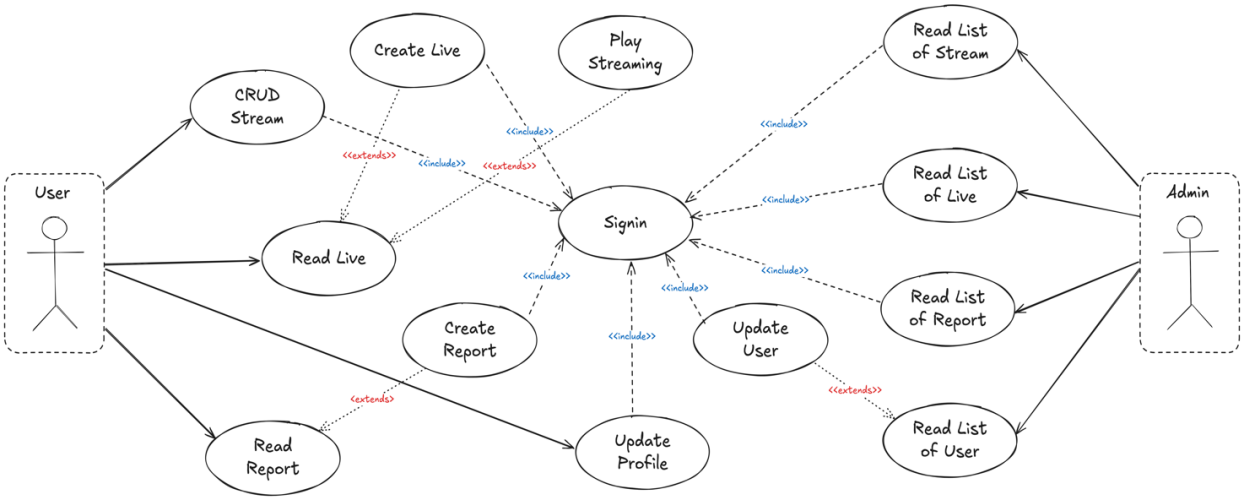



**FIGURE 2.** Use Case Diagram

### *Topology Architecture*

Based on the results obtained previously, the next step is to design the technology to be used in the REST API architecture. The focus of this design is to facilitate the API interface in stateless communication using the HTTP protocol and using JSON (JavaScript Object Notation) as the data format.

Fastapi will be used as a python web framework (Watch Service) that forms the REST API for managing CCTV streaming data. Meanwhile for the General Service, Express.js is employed as a lightweight and efficient Node.js framework to handle more general-purpose functionalities, including user authentication, management of CCTV metadata, report generation, and related administrative tasks. PostgreSQL serves as the main relational database, responsible for storing structured data such as user profiles, CCTV metadata, report logs, and session information. Also, Docker is used to containerize these services, providing isolated environments for consistent deployment across development and production systems. This container-based approach streamlines the development lifecycle, boosts scalability, and improves the system's overall maintainability.

Other technologies are also used, such as Redis (in-memory database to store streaming data), PostgreSQL (the main database to store the main data), and MinIO (a web service to store image and video data). The detailed breakdown can be seen through "Fig. 3" below.

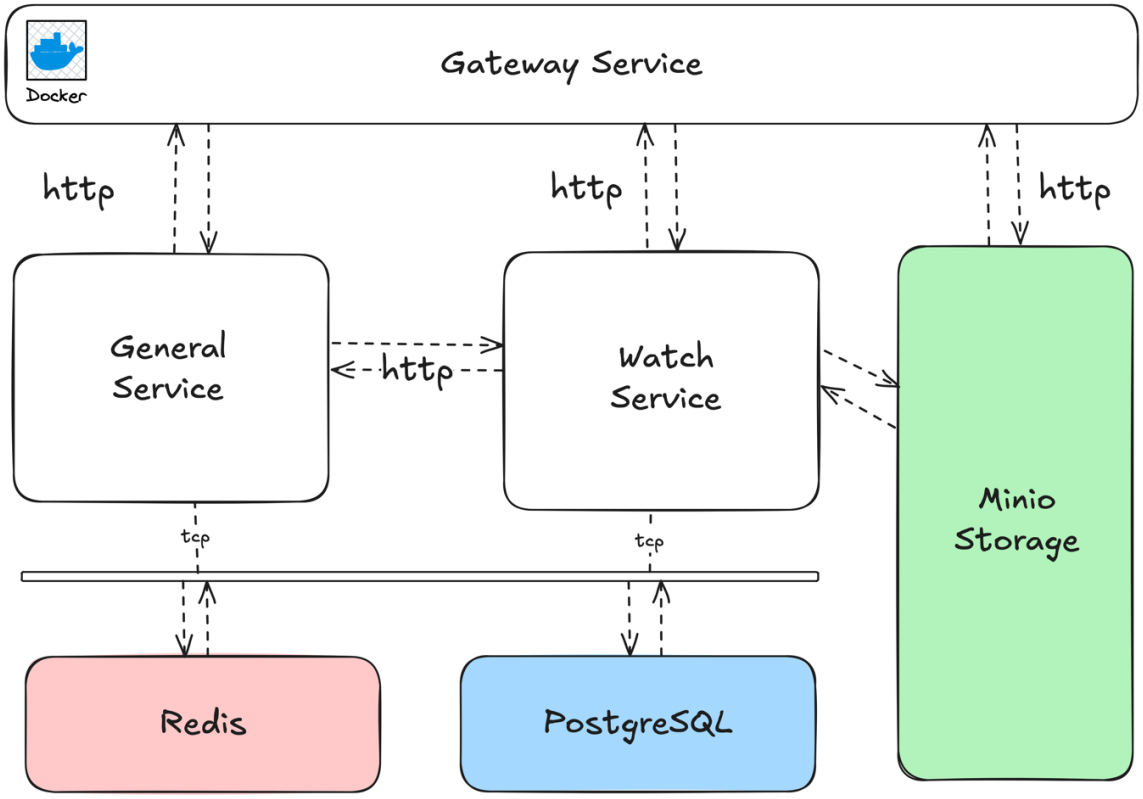



**FIGURE 3.** Topology Architecture Design

*Database Design*

Based on the user stories and use cases that have been created previously, the database is then organized into an Entity Diagram consisting of several tables to be implemented into the database. In this research, the database used is PostgreSQL to manage data on the system. Furthermore, there is a diagram in “Fig. 4” makes it easier to understand the required data structure.

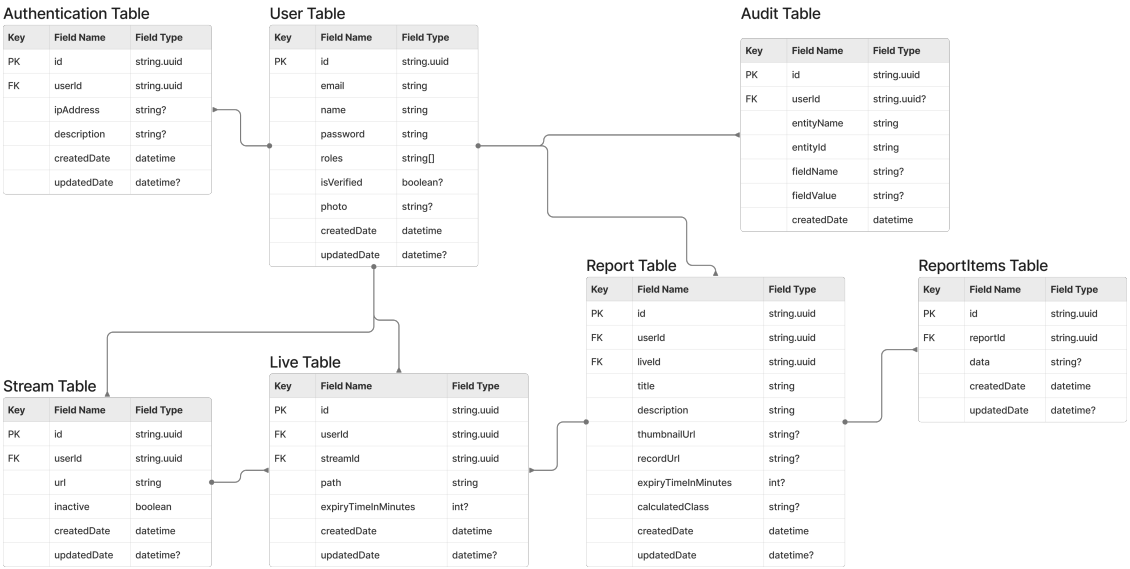



**FIGURE 4.** Entity Relationship Diagram

## Planning Phase

This stage begins with identifying needs related to user needs. Based on these needs, there are two types of users who will use this system. The first type of user needs access to be able to create and view recorded images from IP CCTV with RTSP protocol format. Then, another user needs access to be able to manage user data as an administrator. Therefore, user stories were created which you can see in full through **TABLE 1**.

**TABLE 1**. User Stories

| Code | Domain | Actor | Description |
|---|---|---|---|
| US-01 | Registration | Admin | Features to register an account |
| US-02 | Authentication | User, Admin | Managing account credentials to access more features |
| US-03 | Stream | User | Can view, create, delete stream data such as name and rtsp url for live streaming of their own stream |
| US-04 | Live | User | Can create live streaming whether with prediction or not. |
| US-05 | Report | User | Users can make a report and live recordings so it can be viewed again. |
| US-06 | User | User | Can view and update their own user data |
| | | Admin | Has ability to create and update every single data of user |

Based on the user stories above, the needs that have been designed as user stories before will be determined in the order of their work based on the features that have the highest level of urgency. Full details can be seen in the following description.

**TABLE 2**. Priority Planning

| Code | Feature | Iteration | Priority |
|---|---|---|---|
| US-01 | Registration | 2 | Low |
| US-02 | Authentication | 2 | Low |
| US-03 | Stream | 1 | Medium |
| US-04 | Live | 1 | High |
| US-05 | Report | 2 | Medium |
| US-06 | User | 1 | Low |

## Iteration Phase

This stage is carried out based on the user stories and priority planning that have been designed in the previous stage. Each user story will be divided into processes based on the iterations in **TABLE 2** and each feature has its own priority scale. Each iteration will include programming and Black Box testing using Postman and a web browser to ensure that the results provided match the initial expectations.

*First Iteration*

Through the first iteration, three features from the domains listed in the user stories will be created first. These domains are: Stream (US-03), Report (US-05), and Live (US-04). In the process, dummy accounts will be used and multiple endpoints will be created to implement all the domains as listed in the following table.

**TABLE 3**. Endpoints Result of First Iteration

| Code | Description | Method | Path | Request | Response |
|---|---|---|---|---|---|
| US-06 | Get user list | GET | /user/list | authorization bearer token required as admin. | JSON response contains status and results |
| US-06 | Get logged data | GET | /user | authorization bearer token required | JSON response contains status and results |
| US-06 | Partial user update | PATCH | /user | Updated userId, body and authorization bearer token required as admin | JSON response contains status and results |

| | | | | | |
|---|---|---|---|---|---|
| US-06 | Partial user update | PATCH | /user | Updated user body and authorization bearer token required | JSON response contains status and results |
| US-04 | Add Stream | POST | /stream | url and authorization bearer token required | JSON response contains status and results |
| US-04 | Get Stream | GET | /stream | Optional query params, authorization bearer token required | JSON response contains status and results |
| US-04 | Delete Stream | DELETE | /stream/ { userId } | userId and authorization bearer token required | JSON response contains success status, message |
| US-03 | Create Live | POST | /live | streamId, expiryTimeInMinutes needed as body and authorization bearer token required | JSON response contains success status, message |
| US-03 | Live streaming | GET | /watch/live /{liveId} | liveId is required, prediction as query params is optional and authorization bearer token is optional | Formatted images jpeg returned as keep-alive connection with 1 minute timeout |
| US-03 | Extending live streaming timeout | PUT | /watch/live /{liveId} /extend-more-minutes | liveId is required and authorization bearer token is optional. Streaming will extend to 1 minute ahead | JSON response returned with status code 201 or 404 |

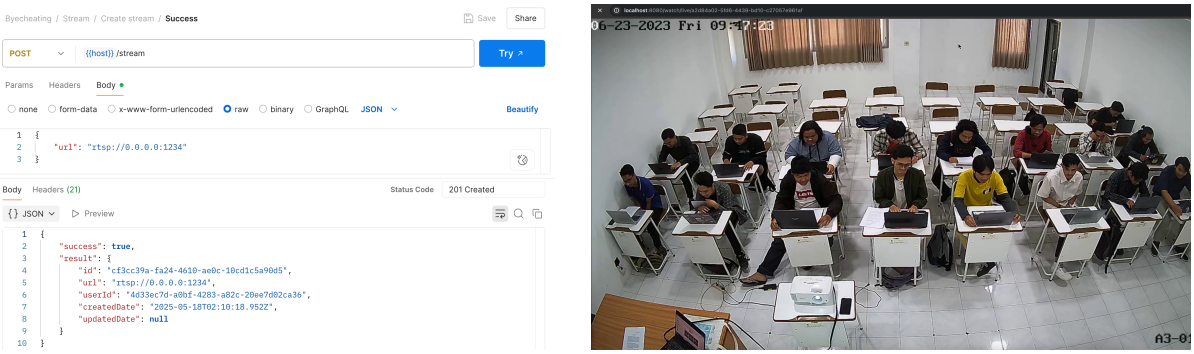
In the first iteration, three main domains will be created. This includes the creation of API service functions such as routes to direct endpoints, controllers as a place to collect the main logic of each domain, middleware as additional functions, validation, and database schema to load the model that has been created from the previous entity relationship diagram.

Based on the list of endpoints in **TABLE 3**, all endpoints with the code US-06 ensure users to have an account so that they can authenticate and access features. Important data such as passwords already use encrypted hashing methods. Furthermore, the US-04 code will give users the ability to manage the list of CCTVs IP that they have. Finally, there is an endpoint coded US-03 that ensures that users can instantly view live CCTV broadcasts, whether ML integrated or not.

**TABLE 4**. Endpoints Testing Example on First Iteration

| Path | Method | Request | Response | Result |
|---|---|---|---|---|
| /user | GET | authorization bearer token required | Status returned with code 200 along with user data if error does not occur, otherwise 500 status code. | Pass |
| /stream | POST | url as body and authorization bearer token required | Status code 201 returned with stream data if successfully created, otherwise returned an error message and status code 400 bad request | Pass |
| /live | POST | streamId needed as body and authorization bearer token required | Status code 201 returned with stream data if successfully created, otherwise returned an error message and status code 400 bad request | Pass |
| /watch/live /{liveId} | GET | liveId is required, prediction as query params is optional and authorization bearer token is optional | Responded as formatted images jpeg with keep-alive connection within 1 minute timeout | Pass |

Then at the same iteration, the testing process is performed by ensuring that the endpoints that have been created in **TABLE 3** can be executed. Therefore, response value will be confirmed during testing as a sign of whether the action is performed correctly or not. For example, only a few key endpoints have been tested as described in **TABLE 4** above.




(a) (b)

**FIGURE 5**. First Iteration of Testing Example in Postman and Web Browser

Based on "Fig. 5" above, it can be seen that there are endpoints that provide responses in the form of JSON as well as images. Tests using Postman consistently returned the status code 200/201 in "Fig. 5(a)", indicating proper functionality. In addition, testing through a web browser in "Fig. 5(b)" ensures that the image stream is delivered efficiently using Keep-Alive connections to optimize performance.

The system updates the live stream every second, delivering images to ensure clear and detailed visuals. However, performance is currently hampered by high CPU utilization, almost reaching 100. This issue arises during testing on a laptop with a 4-core CPU and 4GB RAM that was struggling quite a bit to handle the computational load. The result, high resource consumption led to increased latency and potential slowdowns under high-traffic conditions.

*Iteration Two*

Subsequently, in the second iteration for remaining features of user story will be further developed with medium and low priority. These domains are: Registration (US-01), Authentication (US-02), and Report (US-05). The following endpoints were created to implement the domains as shown in the following table.

**TABLE 5**. Endpoints Result of Second Iteration

| Code | Description | Method | Path | Request | Response |
|---|---|---|---|---|---|
| US-01 | Sign up | POST | /user/signup | Email, name, and password as body, none of authorization token is required | JSON response contains success status, user data, access and refresh token |
| US-02 | Sign in | POST | /user/signin | email and password as body | JSON response contains success status, user data, access and refresh token |
| US-05 | Create report | POST | /report | streamId, expiryTimeInMinutes, and authorization bearer token is required | JSON response contains success status, report, stream and live data |
| US-05 | Get report result | GET | /report | withItems query params is optional and authorization bearer token is required | JSON response contains success status and report data |

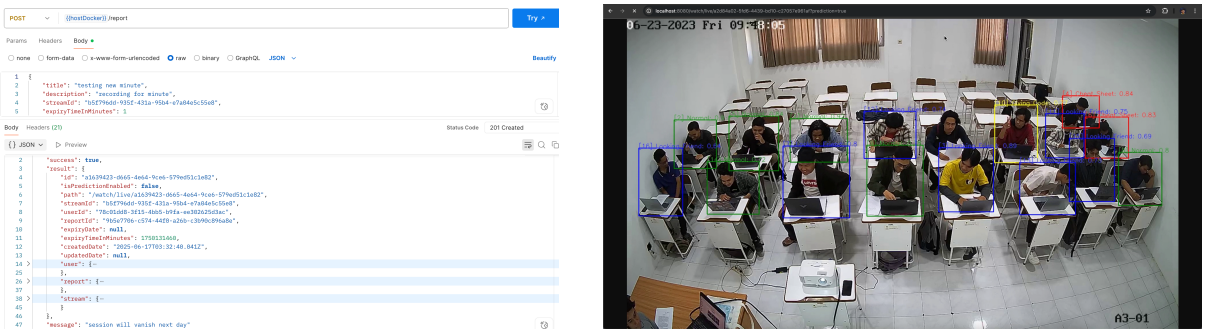
Then, in the same iteration, the testing process will be carried out again by ensuring that the endpoints created in**TABLE 5** can be executed. Here are some examples of tests carried out in the table below.

**TABLE 6**. Endpoints Testing Example on First Iteration

| Path | Method | Request | Response | Result |
|---|---|---|---|---|
| /user/signup | POST | Email, name and password as body | Status returned with code 201 along with user data if error does not occur otherwise 400 status code. | Pass |

| | | | | |
|---|---|---|---|---|
| /user/signin | POST | Email and password as body | Status returned with code 201 along with user data if error does not occur otherwise 400 status code. | Pass |
| /report | POST | streamId, expiryTimeInMinutes, and authorization bearer token is required | Status returned with code 201 along with report, live, stream data if error does not occur otherwise 400 status code | Pass |
| /report | GET | Id is optional query params, and authorization bearer token is required | Status returned with code 200 along with report, live, stream data | Pass |

Based on the list of endpoints in **TABLE 6**, all of endpoints with story code US-01 ensure that users can register an account. Then, the endpoint with US-02 code will give the user a session that can be used as authorization to use the feature. Finally, there is an endpoint with the US-05 code that ensures that the user can perform CCTV recording and the prediction results are saved as a report. In addition, users can also directly view live CCTV recordings, whether integrated with ML or not, through report data.




(a) (b)

**FIGURE 6**. Iteration Two's Testing Example in Postman and Web Browser

Also based on "Fig. 6" above, it can be seen that the endpoint is running properly. The experiment using postman in "Fig. 6(a)" has been successfully performed with a status code range of 201. Then the experiment again using a web browser has also been successfully carried out as in "Fig. 6(b)" with a streaming response in the form of an image. Then the data in the report also appears in the form of recordUrl and calculatedClass, which recordUrl contains an open link to view the recording results and calculatedClass to see a summary of the prediction classification results.

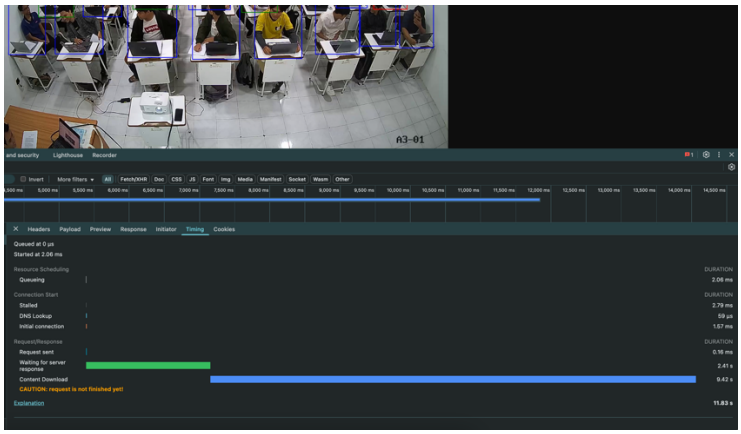
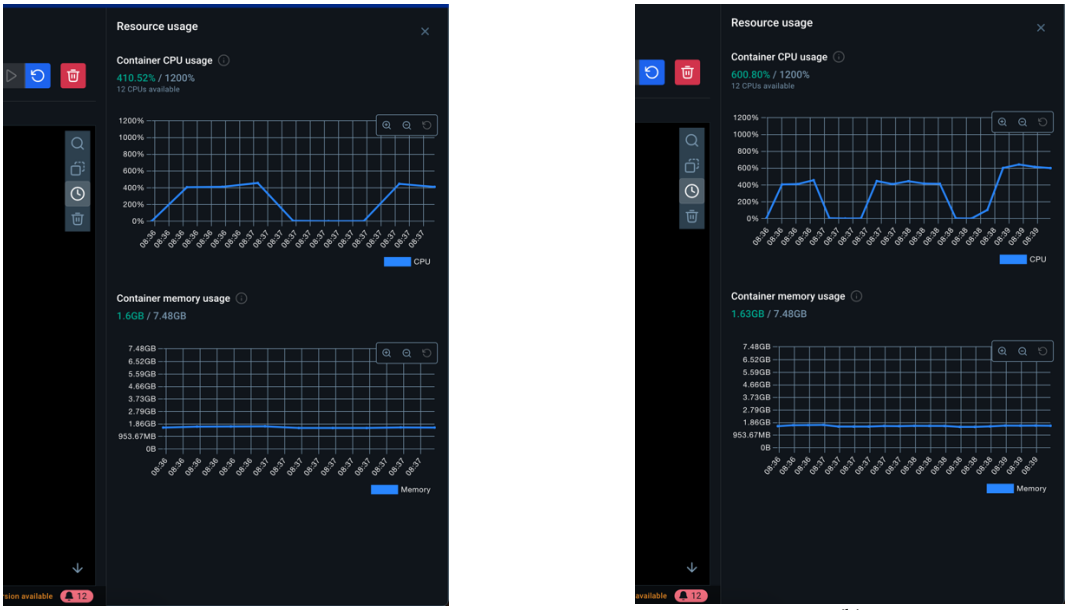



**FIGURE 7**. Iteration Two's System Response Testing

Considering the timing results of the MJPEG streaming request, the total request duration can reach 11.83 seconds with an initial response waiting time of 2.41 seconds. This is relatively slow and has the potential to create heavy backend processes, such as cheating detection or real-time video processing. Through "Fig. 7" also shows that the streaming can run stably for 9.42 seconds without any major pauses, indicating that the connection remains open and data continues to be sent at an average of 1 second/frame.

However, since MJPEG uses a keep-alive connection per user, this can pose scalability issues if the number of users increases. Fortunately, the server already uses a non-blocking architecture and has been optimized for concurrent streaming so the connection remains stable for multiple simultaneous requests.




(a) (b)

**FIGURE 8**. Iteration Two's System Performance Testing

Finally, performance testing was also conducted to see the capabilities of the server with the new computing specifications of 12-core CPU and 8GB RAM. In "Fig. 8(a)", the processing time per frame is consistent with an average of 547 ms resulting in a speed of about 1.83 FPS. CPU utilization is also consistent at 410% of 1200% (about 3 cores), with memory consumption remaining stable at 1.66 GB. Then in "Fig. 8(b)", the average total processing time per frame rises around 850 ms, resulting in a speed of about 1.17 FPS. CPU utilization is also recorded to rise to 600% with memory consumption remaining fairly stable at 1.63 GB.

## Release Phase

Before doing this stage, author ensures that all stages have been completed by retesting according to the endpoints that have been made in iteration phase. Then the architecture will be containerized using docker as software to wrap all existing services according to "Fig. 3" in exploration phase. To perform containerization, a configuration is needed to run one of the Docker tools, namely docker-compose. Inside of Docker containerization can be seen through the following image.

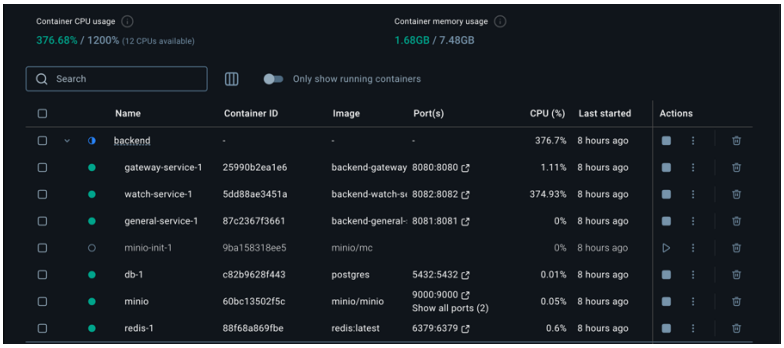



**FIGURE 9**. Docker Compose Script and Dashboard

Based on "Fig. 9", several containers will be created for the REST API, database, file storage, and cache storage. Each container has a port that can be accessed from outside the server architecture. Additionally, an API gateway feature combines all IP addresses in the general and streaming containers in "Fig. 3" into one domain/IP alias. This will allow website developers to easily integrate with the REST API created in this research in the future.

## CONCLUSION

The development results of this Exam Integrity Platform, which integrates information systems with machine learning (ML) models for fraud detection has been successfully completed. This integration requires a comprehensive approach to ensure accurate and reliable detection of people behavior. The use of REST API architecture, combined with the Extreme Programming (XP) methodology, has proven effective in facilitating seamless and efficient system integration. Additionally, studies on system robustness under varying network conditions, and ethical considerations in automated fraud detection can further enrich this area of research.

In summary, the API service has two key roles that support the overall system architecture. First, it manages user information and records image detection results. Second, it displays the outcomes of image detection in real time through live recordings, enabling timely monitoring and responsive actions. Together, these functions form a comprehensive solution that integrates data management with real-time streaming capabilities. This system offers practical benefits in handling multimedia and surveillance data and academic contributions by serving as a reference for microservice-based REST APIs, real-time detection pipelines, and scalable streaming architectures using modern web technologies.

Despite the compromised development stage, some features remain unoptimized but still mostly stable. This can be used for further research.

## ACKNOWLEDGMENTS

Thanks to my family and friends for their unwavering support. They provided encouragement, especially during times when the challenges felt overwhelming.

I would also like to express my deepest gratitude to my lecturers, whose guidance, feedback, and continuous reminders played a significant role in shaping and improving the quality of this research. Their dedication and patience have inspired me to keep pushing forward, thank you.

## REFERENCES

1. Pembangunan Pariwisata Berkelanjutan Untuk Pemulihan Ekonomi Nasional Magelang, S., Hanifah, U., Septiani, Y., Lukis Panjawa, J., Pembangunan, E., Ekonomi, F.: Pengaruh Pendidikan Terhadap Pertumbuhan Ekonomi di Indonesia Tahun 2015-2021. (2023)
2. Sletten, S.R.: Rethinking Assessment: Replacing Traditional Exams with Paper Reviews. J Microbiol Biol Educ. 22, (2021). https://doi.org/10.1128/jmbe.00109-21
3. Timotheou, S., Miliou, O., Dimitriadis, Y., Sobrino, S.V., Giannoutsou, N., Cachia, R., Monés, A.M., Ioannou, A.: Impacts of digital technologies on education and factors influencing schools' digital capacity and transformation: A literature review. Educ Inf Technol (Dordr). 28, 6695–6726 (2023). https://doi.org/10.1007/s10639-022-11431-8
4. Stp, M.F.R.: Pemanfaatan Teknologi Dan Informasi Oleh Manusia: Tren Otomatisasi Di Sektor Pendidikan. JKOMDIS : Jurnal Ilmu Komunikasi Dan Media Sosial. 3, 853–858 (2023). https://doi.org/10.47233/jkomdis.v3i3.1375
5. Sabrina, F., Azad, S., Sohail, S., Thakur, S.: Ensuring academic integrity in online assessments: A literature review and recommendations. International Journal of Information and Education Technology. 12, 60–70 (2022). https://doi.org/10.18178/ijiet.2022.12.1.1587
6. Mushthofa, Z., Rusilowati, A., Sulhadi, S., Marwoto, P., Mindiyarto, B.N.: Analisis Perilaku Kecurangan Akademik Siswa dalam Pelaksanaan Ujian di Sekolah. Jurnal Kependidikan: Jurnal Hasil Penelitian dan Kajian Kepustakaan di Bidang Pendidikan, Pengajaran dan Pembelajaran. 7, 446 (2021). https://doi.org/10.33394/jk.v7i2.3302
7. Bimantoro, F., Gede, I., Suta Wijaya, P., Aohana, M.R.: Pendeteksian Kecurangan Ujian Melalui CCTV Menggunakan Algoritma YOLOv5.

8. Muhammad, I.R.D., Paputungan, I.V.: Development of Backend Server Based on REST API Architecture in E-Wallet Transfer System. Jurnal Sains, Nalar, dan Aplikasi Teknologi Informasi. 3, 79–87 (2024). https://doi.org/10.20885/snati.v3.i2.35
9. Santosa, K., Rofifah, R.H., Riani, A.A., Syawanodya, I., Tawakal, I.: Design and Implementation of REST API for Predicting the Recitation of the Qur'an using Machine Learning. Journal of Software Engineering, Information and Communication Technology (SEICT). 5, 43–52 (2024). https://doi.org/10.17509/seict.v5i1.70600
10. Firdaus, M., Alamsyah, N., Jatmika, A.H.: Development of a REST API for the Rinjani Visitor Application using Extreme Programming.
11. Rumandan, R.J.: DEVELOPMENT OF INFORMATION SYSTEMS FOR SERVICE AND CUSTOMER COMPLAINTS USING THE EXTREME PROGRAMMING METHOD. (2023)
12. Dwi Wijaya, Y., Wardah Astuti, M.: PENGUJIAN BLACKBOX SISTEM INFORMASI PENILAIAN KINERJA KARYAWAN PT INKA (PERSERO) BERBASIS EQUIVALENCE PARTITIONS BLACKBOX TESTING OF PT INKA (PERSERO) EMPLOYEE PERFORMANCE ASSESSMENT INFORMATION SYSTEM BASED ON EQUIVALENCE PARTITIONS. Jurnal Digital Teknologi Informasi. 4, 2021
13. ARAVINDA A KUMAR, Divya TL: Security measures implemented in RESTful API Development. Open Access Research Journal of Engineering and Technology. 7, 105–112 (2024). https://doi.org/10.53022/oarjet.2024.7.1.0042
14. Privalov, M.V., Stupina, M.V.: Improving web-oriented information systems efficiency using Redis caching mechanisms. Indonesian Journal of Electrical Engineering and Computer Science. 33, 1667–1675 (2024). https://doi.org/10.11591/ijeecs.v33.i3.pp1667-1675
15. Xuan, S., Park, D.-H., Kim, D.: A Service Platform for Real-Time Video Streaming Based on RESTful API in IoT Environments. International Journal of Control and Automation. 10, 181–192 (2017). https://doi.org/10.14257/ijca.2017.10.3.15
16. Mabotha, E., Mabunda, N.E., Ali, A.: A Dockerized Approach to Dynamic Endpoint Management for RESTful Application Programming Interfaces in Internet of Things Ecosystems. Sensors. 25, (2025). https://doi.org/10.3390/s25102993