p–ISSN: 2723 - 6609 e-ISSN: 2745-5254
Vol. 5, No. 8 August 2024 http://jist.publikasiindonesia.id/

Backend Infrastructure and Specifications Design Using OpenAPI and API-First on CV Elang Java Mandiri

Yohanes Bagas Ari Widatama[1*], Nizirwan Anwar[2], Agung Mulyo Widodo[3], Arief Ichwani[4]
Universitas Esa Unggul, Indonesia
Email: bagasadvan123@student.esaunggul.ac.id[1*], Nizirwan.anwar@esaunggul.ac.id[2], agung.mulyo@esaunggul.ac.id[3], arief.ichwani@esaunggul.ac.id[4]

*Correspondence

ABSTRACT

Keywords: backend, restful API, open API, OWASP.

Digital transformation brings changes to the business world by increasing efficiency, convenience, security, certainty and operational speed. CV Elang Java Mandiri experienced the positive impact of this by creating software to increase operational efficiency. However, the use of direct communication between the desktop application and the database causes vulnerabilities. The lack of a bridge between the desktop application and the database also indicates a lack of flexibility when adding other applications. Ransomware attacks on desktop applications cause losses and limitations in development. This research focuses on backend updates that use API as an integrator with the API-first method. OpenAPI standards and OWASP security principles are used to increase resilience against security threats. These steps were tested with OWASP ZAP and http test. The goal is to provide solutions to company problems, meet the need for more secure applications, and make application development easier.

Introduction

In the modern era, digital transformation in business is inevitable. Digital transformation is intended to make business processes more efficient. Digital transformation is a topical issue around the world, which is critical for all companies across all sectors, as customer relationships, internal processes, and value creation change. The main concern of stakeholders in this transformation is to define a vision and roadmap that determines the way forward (Zaoui & Souissi, 2020).

CV Elang Java Mandiri, a paper sales company that has tried to digitize by creating a stock management and sales application has faced serious obstacles when the servers that store the source code and data of their desktop applications were attacked by ransomware. This incident led to the loss of data and source code, halting application development and maintenance. Without source code backups, the operations of companies that rely on desktop applications become vulnerable. Although the app is currently still running, the company's owners want steps to rebuild the app with a focus on security and extensibility, including secure remote capabilities and mobile services. In this study, the design and development of the backend uses the API-first method with OpenAPI standards and OWASP security principles. Testing includes black box testing and security testing with OWASP ZAP. This research is expected to solve the company's problems and meet the owner's expectations for a safer and more scalable application.

Several studies related to security and information system development provide important insights. Dwi Cahyani et al. (Cahyani, Dewi, Suryadi, & Listartha, 2021) found that the implementation of Rate Limiting on the school websites studied needed to be improved to improve security, with ZAP's OWASP analysis showing low to medium levels of vulnerability. Gong et al. (Gong, Gu, Chen, & Wang, 2020) highlighted the benefits of separating the front-end and back-end in campus information systems using a micro-services architecture with Spring Cloud and React, which improves system performance and user experience. Research by (Ashari & Denira, 2023) the campus e-learning website that was studied showed low, medium, and high-risk vulnerability analysis with a focus on quickly handling high-level vulnerabilities. (Adam, Besari, & Bachtiar, 2019) designed a REST API-based cashless payment backend system, with recommendations for updating server specifications and database methods. (Dudjak & Martinović, 2020) suggest an API-First methodology for designing a Backend as a Service (BaaS) platform. (Baniaș, Florea, Gyalai, & Curiac, 2021) explored automated testing of REST APIs using OpenAPI 3.0 Standard. (Priyawati, Rokhmah, & Utomo, 2022) conducted a website vulnerability test with the OWASP ZAP tool, identifying 12 vulnerabilities that need to be fixed. Cleveland et al. (Cleveland et al., 2020) discuss the development of the Tapis API with Python using the Tapis Framework and OpenAPI 3, enabling rapid web API development for the integration of microservices in scientific research.

Research Methods

The research method involves interviews with company owners, store operational members, and cashiers, complemented by observation of existing desktop applications. The combination of these methods is expected to provide a comprehensive picture of the redevelopment of the application. Data analysis uses SWOT to assess legacy application performance, resources, and development needs (Dobrović & Furjan, 2020). The software development method combines Waterfall and API First for decision stability, risk management, and robust API interfaces. Software testing is carried out by black box testing and OWASP ZAP.

UML modelling is used to design various aspects of the backend in a systematic and standardized manner. It is hoped that this approach addresses the company's challenges while improving the security and quality of the application.

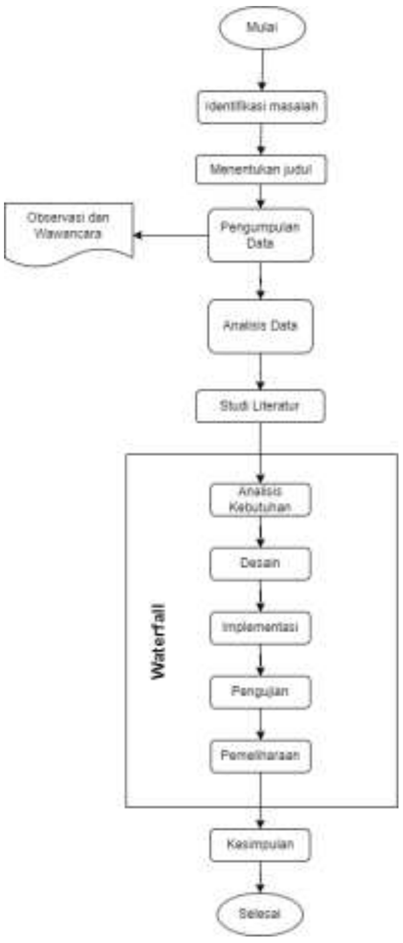


Figure 1 Research Methodology

Backend

The evolution of information technology has progressed rapidly. Web technology development is focused on three main aspects, one of which is the backend. Backend refers to programs and scripts that operate on the server side. It serves as a container for the core logic and operation of an application or information system. The backend system ensures that the data or services requested and sent by the front-end system or application or client-side application are delivered through programming methods. The backend part consists of core application logic, databases, data integration and applications, APIs, and other backend processes.

Restful API

The research conducted by (Adam et al., 2019) explains several other literature that discusses the use of RESTful APIs or REST APIs. REST is not a standard but rather a design pattern of software architecture. REST is a practical approach in web application development where the system needs to be improved or needs a simple way to interact with independent systems. REST is stateless and data-oriented, where everything in a REST architecture is data. Each request is independent, the server does not store any request state. Application Programming Interfaces (APIs) that follow the REST Style are referred to as RESTful APIs. This design pattern uses a Uniform Resource Identifier (URI) to represent the data. For operations on data, the GET method is used to retrieve data, the POST method is used to create new data, the PUT method is used to update data with resource IDs, and the DELETE method is used to delete data or data sets.

Restful API vs Direct Database Access

In today's era of application development, developers are faced with an important consideration between using direct access to databases or utilizing REST APIs as a method of connectivity. Bennet [12] discusses this in his writing on the DreamFactory website. Direct access to the database provides the advantage of simplicity and immediate control, perfect for the prototype stage or controlled environments. However, this approach also raises security-related concerns, especially as the app evolves and must be tested by users outside of the development team.

OpenAPI Specification

The OpenAPI Specification (OAS) plays a central role in the development of API services and microservices with accurate definitions using YAML or JSON format. It allows consumers, such as client applications, to interact with services remotely with minimal implementation logic. OAS not only facilitates service understanding for consumers but also supports efficient and collaborative development, allowing development teams to understand and optimize service functionality. As a best practice in defining API service specifications, OAS is becoming a very useful tool in the exploration of the latest technologies in the field of APIs and microservices, providing a standard and language-agnostic interface for RESTful APIs in the context of scientific research.

Unified Modelling Language (UML)

Unified Modeling Language (UML) is a standard modelling language adopted in 1997 for Object-Oriented Software Engineering. Used in developing expressive diagrams, UML facilitates the design and implementation of a wide range of applications, including embedded systems, web applications, and commercial applications. UML tools generate bug-free code and are ready to deploy. As a key graphics language, UML aids in the visualization, definition, and documentation of software systems with elements such as nodes, paths, and use cases. UML diagrams include conceptual aspects such as business processes, system functions, programming languages, database schemas, and reusable software components.

Model View Controller (MVC) Design Pattern

MVC, which stands for Model View Controller, is an architectural pattern that breaks down the application code into three main parts, as explained by (Hsieh, Li, Wang, & Ke, 2020). First, Layer models serve to encapsulate the data processing methods and business logic of the application, allowing direct data access. This layer is independent of the View and Controller, providing data for a variety of views, as well as avoiding over-coding. Second, the View layer acts as a user interface that presents data from the Model. It can be interpreted as a visual representation of data for various clients, such as website visitors or client applications. Third, the Controller layer is located between the View and the Model, processing the data from the View and returning the analysis results to the Model. Generally, a single controller improves the flexibility and overall configuration of the application. Thus, the use of MVC architectural patterns facilitates the separation of responsibilities in application development.

Containerization

Containerization is the process of packaging software code along with the operating system (OS) libraries and dependencies needed to execute that code, creating a single, lightweight execution package—referred to as a container—that can run consistently on a variety of infrastructures. Containerization allows applications to be "written once and run anywhere.".

Open Web Application Security Project (OWASP)

OWASP (Open Web Application Security Project) is an international non-profit organization that has a high dedication to improving web application security [16]. The organization provides a variety of materials for free, including documentation, tools, videos, and forums, with one of its flagship projects being the OWASP Top 10.

Black box testing

Black box testing, usually emphasizes the examination of the functional requirements of the software. Black box testing allows software engineers to define a set of input conditions that comprehensively evaluate all the functional requirements of a program (Irawan, Muzid, Susanti, & Setiawan, 2018).

Waterfall Methodology

The waterfall method is a type of application development model included in the classic SDLC, which emphasizes sequential and systematic phases. In this development model, the analogy is similar to that of a waterfall, where each stage is executed sequentially from top to bottom. Therefore, each stage should not be done simultaneously. Thus, the difference between the waterfall method and the agile method lies in the SDLC stage. This model also includes software development that is more or less iterative and flexible. Because the process leads in one direction like a waterfall (Firzatullah, 2021).

API-First

API-First Design is a method for designing and developing APIs starting from the design phase. This approach requires that API functionality is planned and described in a standard format and that the code to implement it is built according to that plan. The API lifecycle starts from the business needs that can be answered by the API. The design-first approach is becoming more and more common. API-First Design is also known as API-guided development, API-first design, or schema-first development. This method can be interpreted as an approach to developing APIs or as an approach to developing applications. API development first is desirable, especially if the new application is built on top of the API. While API-First Design can be used even if there is an existing application or functionality, defining an API before implementation remains a useful way (Hämäläinen, 2019).

Results and Discussion

After conducting interviews, observations, and SWOT analysis on CV Elang Java Mandiri, several internal and external aspects were found that needed attention.

Internally, the company's strengths include applications that already meet operational needs and provide significant support. However, there are drawbacks, such as direct communication between desktop applications and databases, the absence of backups for source code, limited application access in LAN networks, and lack of technical documentation. Externally, there is an opportunity for application redevelopment with security and technology improvements, as well as potential improvements on the server and network sides. On the other hand, there are threats related to vulnerabilities in the communication scheme between clients and databases, potential vulnerabilities from internet access, and the possibility of new problems emerging during the redevelopment.

Service Architecture

In the context of service architecture, it is considered between a monolithic and microservice approach. The absence of source code is the main obstacle, so a monolithic approach is chosen to enable the development of the entire application without managing complex infrastructure. This decision also takes into account the ease of management and maintenance.

Current database designs, even though they use Microsoft SQL Server, do not make optimal use of relational concepts. Therefore, it is planned to redesign the database by taking into account the old scheme, stakeholder needs, and the user interface of the desktop application.

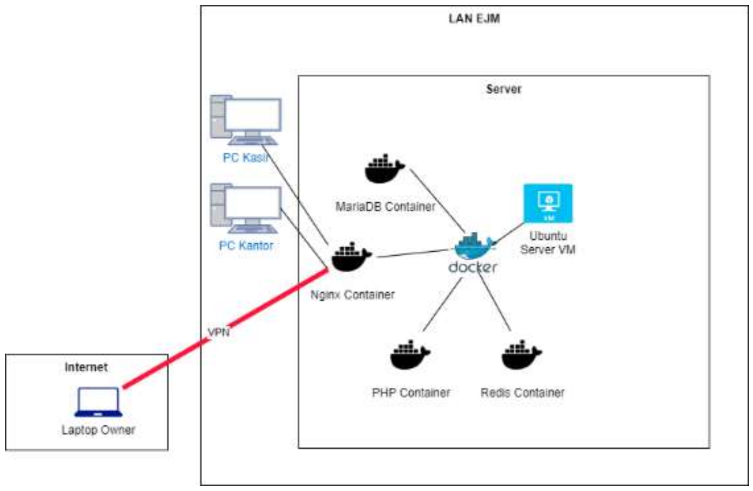


Figure 2 Provided Communication Scheme Solution

The communication strategy between services is planned by utilizing Hyper-V to create Docker VMs and containers on Windows Server. This aims to provide isolation so that attacks do not spread immediately, considering that the company does not have permanent IT personnel.

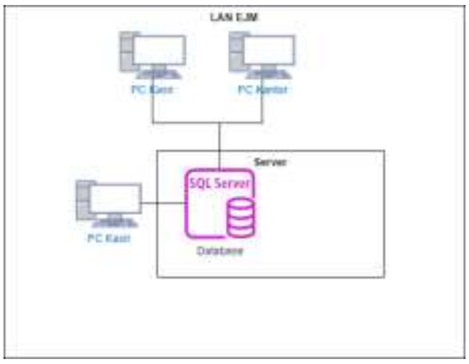


Figure 3. Old Communication Scheme

Then the use of containers and virtualization is also one of the steps to improve technology for security purposes and ease of service management.

The selection of RESTful APIs as the control and data access layer in the redevelopment of application systems is based on considerations of efficiency, security, and flexibility. RESTful APIs provide a high level of abstraction, allowing client applications to communicate with the backend without being bound to the data structure in the database. The backend design is done with the API First method, starting from the definition of the API specification and the redesign of the database. Mapping data to objects on a model in an MVC architecture becomes the focus, following a bit of a Bottom-Up approach.

Technology Selection

To improve the security of the RESTful API backend, several security guidelines are used by OWASP. The Laravel 10 framework was chosen as the basis for development due to its maturity and abundant community support, especially in Indonesia. This framework, based on MVC, not only facilitates development but has also proven to be reliable in addressing various security threats. These security guidelines are implemented through the use of several official packages, such as Laravel Sanctum and Spatie Permission, to handle authentication and authorization effectively. Additional security is provided through the use of the Laravel Eloquent ORM, which ensures database security by implementing techniques such as parameter binding to counter SQL injection attacks. In database management, there is a migration from Microsoft SQL Server to MariaDB. This choice is based on high compatibility with MySQL, which allows for a smooth transition. MariaDB is also known for its optimal performance and high efficiency, providing additional benefits in handling database operations. The sustainability of MariaDB as an open-source project provides certainty of support and updates from the community. The use of Docker as an open-source container platform provides the advantages of portability, fault isolation, simplified management, and simplified security. Redis was chosen as the database for session and cache management because of its high performance and flexibility. Redis stores session data inside RAM improves application responsiveness and serves as a caching mechanism to reduce server load. By implementing these technologies, the RESTful API backend not only prioritizes security but also looks at the overall aspects of performance, efficiency, and cost. These steps are geared towards proactive backend redevelopment to improve the security, responsiveness, and sustainability of the project.

Request and Response Flow

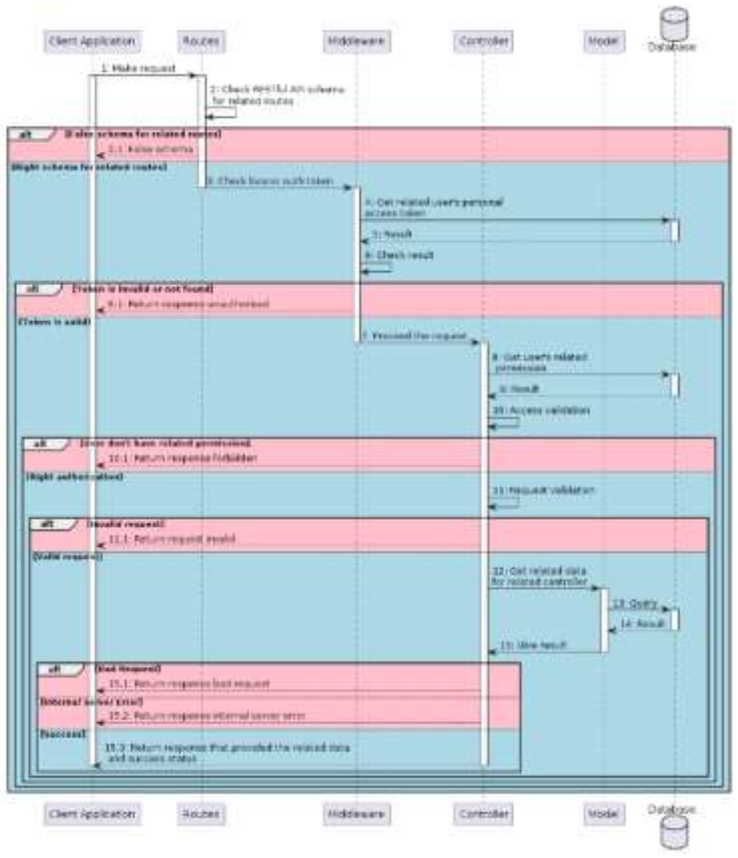

Client Application
Routes
Middleware
Controller
Model
Database
1: Make request
2: Check RESTful API schema for related routes
alt
[False schema for related routes]
2.1: False schema
[Right schema for related routes]
3: Check bearer auth token
4: Get related user's personal access token
5: Result
6: Check result
alt
[Token is invalid or not found]
6.1: Return response unauthorized
[Token is valid]
7: Proceed the request
8: Get user's related permission
9: Result
10: Access validation
alt
[User don't have related permission]
10.1: Return response forbidden
[Right authorization]
11: Request validation
alt
[Invalid request]
11.1: Return request invalid
[Valid request]
12: Get related data for related controller
13: Query
14: Result
15: Give result
alt
[Bad Request]
15.1: Return response bad request
[Internal server error]
15.2: Return response internal server error
[Success]
15.3: Return response that provided the related data and success status

Figure 4. Request and Response Flow on the Designed Backend

Any endpoints that are protected using authentication and authorization (must have an account with a specific authority) will be filtered through the middleware first. The design uses Laravel Sanctum to manage authentication with a token-based concept, where the token will be used by the client application in the Bearer authentication header. In addition, authorization is handled using Spatie Permission where the concept is role-based and permission-based. In addition, Laravel also has rate limiters used in this design for get-token and register endpoints. In addition, each controller is also designed to verify the user's permissions, and whether the user can access the relevant actions or not.

The sequence diagram presented in Figure 4 offers a detailed overview of the interactions in the system that manages the routes protected in the RESTful API. It illustrates the flow of actions initiated by the client application, directed through various components including routes, middleware, controllers, models, and databases. Each component fulfils a different role in the operation of the system, contributing to the overall functionality.

The sequence begins with a client request to the route component, which first verifies the conformance of the requested route with a predefined RESTful API schema. At critical decision points, such as token validation and access permissions, the system branches into alternative paths based on specific conditions. For example, an invalid or non-existent authentication token triggers an unauthorized response, denying access to the requested resource. Instead, a valid token triggers authentication checks and authorizations, facilitating the progress of requests through the system.

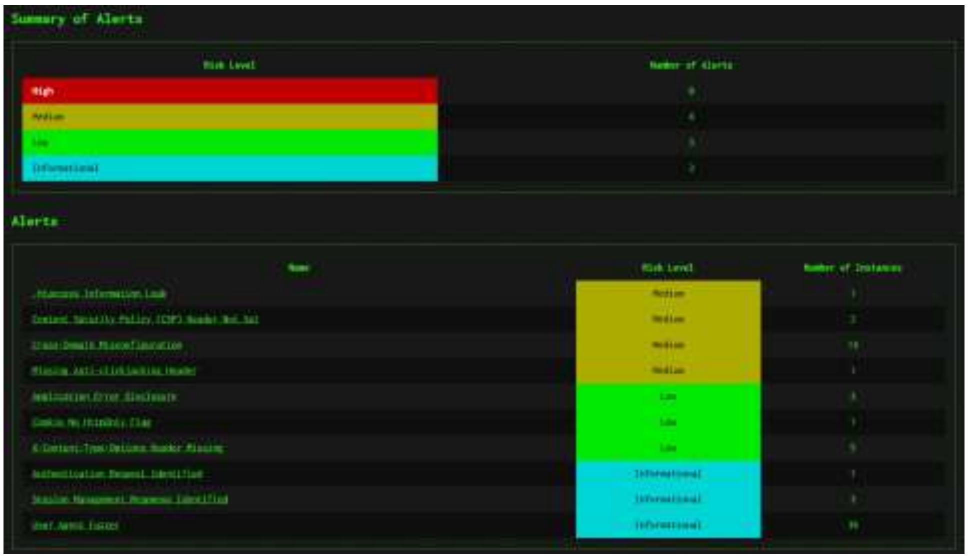
A robust error-handling mechanism is important for system resilience, addressing various error scenarios such as unauthorized access, invalid requests, and database errors. This proactive approach minimizes disruption to the user experience and ensures the system's ability to handle unexpected contingents well.

The design uses Laravel Sanctum to handle authentication with the concept of token-based, where the token will be used by the client application in the Bearer auths header. In addition, authorization is handled using Spatie Permission where the concept is role and permission-based. In addition, Laravel already has a rate limiter used in this design for get-token and register endpoints. In addition, each controller is also designed to verify the permissions that the user has, whether the user can access or not for related actions.

OpenAPI and OWASP

Comprehensive documentation, such as the one embodied in the OpenAPI specification, is critical in dealing with the risk of Lack of Security Logging and Monitoring. The designed backend ensures that every request and response is documented, allowing for effective monitoring of API activity and quick detection of suspicious behaviour. Then through the application of OWASP security principles, the backend design not only prioritizes the security of its applications but also adopts a proactive approach to risk mitigation. With a deep understanding of the security risks identified by the OWASP Top Ten, the backend can provide solid protection against evolving security threats. OpenAPI also makes it easier to test Http and the specification can be used in DAST tools, namely OWASP ZAP Proxy.

Testing



Summary of Alerts

Risk Level	Number of Alerts
High	0
Medium	4
Low	3
Informational	3

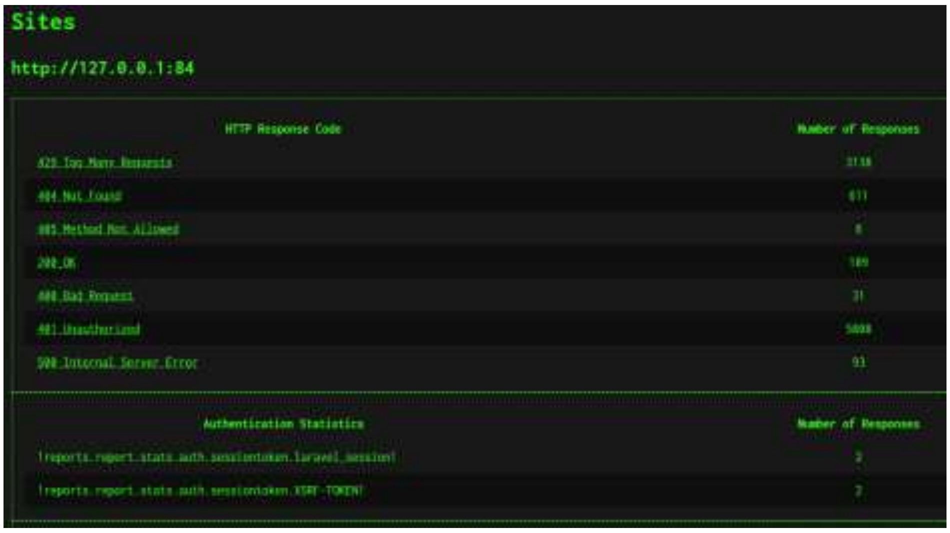
Alerts

Name	Risk Level	Number of Instances
.htaccess Information Leak	Medium	1
Content Security Policy (CSP) Header Not Set	Medium	2
Cross-Domain Misconfiguration	Medium	16
Missing Anti-clickjacking Header	Medium	1
Application Error Disclosure	Low	8
Cookie No HttpOnly Flag	Low	1
X-Content-Type-Options Header Missing	Low	5
Authentication Request Identified	Informational	7
Session Management Response Identified	Informational	8
User Agent Fuzzer	Informational	99

Gambar 5. Summary Alert OWASP ZAP

Testing was carried out on several endpoints in a local environment with results that met the specifications. Furthermore, tests were carried out using OWASP ZAP Proxy and the results were obtained in Figure 5 which identified several warnings with medium and low risk levels. Preventive measures, such as the implementation of throttle or rate limiters, are taken to reduce the risk of attacks and abuses as can be seen from the results of active scans of ZAP in figure 5. The security rules that were successfully bypassed included various potential threats such as SQL Injection attacks, cross-site scripting (XSS), and several other security vulnerabilities. It is recommended to immediately take corrective action, especially on information leaked via .htaccess files, assign proper CSP headers, and fix vulnerable cross-domain configurations. However, these findings can also be used as an opportunity to strengthen the security layer and ensure that the system remains responsive to potential threats.

It is important to keep the system secure by updating and managing the security rules that are successfully bypassed. While some of the warnings are informative, preventive measures need to be taken to prevent potential security threats in the future. Thus, the designed backend can continue to operate with an optimal level of security. In addition, the results of the active analysis of OWASP ZAP scans on this backend include additional information regarding HTTP responses and authentication statistics. In Figure 6, there is a variation in the HTTP response, with most responses resulting in 429 Too Many Requests and 401 Unauthorized status codes. A status code of 429 indicates that several requests exceed the allowed limit in a given period, while a status code of 401 indicates that there is an issue with user authentication.



Sites

http://127.0.0.1:84

HTTP Response Code	Number of Responses
429 Too Many Requests	3138
404 Not Found	611
405 Method Not Allowed	8
200 OK	189
400 Bad Request	31
401 Unauthorized	5008
500 Internal Server Error	93

Authentication Statistics	Number of Responses
!reports.report.stats.auth.sessiontoken.laravel_session!	2
!reports.report.stats.auth.sessiontoken.XSRF-TOKEN!	2

Gambar 6. Summary Active Scan

In addition to the results of the active analysis of OWASP ZAP scans that logged several warnings with medium and low-risk levels, as well as providing an overview of HTTP responses and authentication statistics on the designed RESTful API backend, it should be noted that additional preventive measures have been taken. The implementation of a throttle or rate limiter on the get token and register routes is a critical step to reduce the risk of brute force attacks or potential abuse of these endpoints.

Throttle or rate limiter plays a role in controlling the number of requests received from a single entity in a given period. By implementing this mechanism, the system can recognize and handle requests that exceed the specified limits. As a result, the risk of threats that may arise from unnatural or excessive activity can be minimized.

The positive side of throttle or rate limiter implementations is their ability to protect critical endpoints such as token get routes and registers from potentially harmful attacks. By setting a limit on the number of requests, the system can ensure that only legitimate users with reasonable behaviour can access and use the service. This is a concrete example of the application of security principles at the level of access and use of resources that can provide significant advantages in maintaining the integrity and availability of the system.

Preventive measures like these are in line with the security principles advocated by OWASP, especially in the context of risk mitigation related to authentication and access control. Thus, this backend has not only been able to identify potential threats through OWASP ZAP analysis but has also implemented effective security measures to mitigate the associated security risks. Overall, this approach provides a significant additional layer of defence in the face of complex and evolving potential security threats. This approach provides an additional layer of defence in the face of complex and evolving potential security threats. The entire system is planned to remain responsive to changes and security risks that may arise in the future.

Conclusion

Backend development on the CV Elang Java standalone desktop application project is a strategic step to improve the security, sustainability, and flexibility of the application. The focus on business logic and database interaction allows for the implementation of modern architectures with API principles. The API-first approach ensures the application development interface is well-defined, making it easy to integrate, test, and document. With the implementation of OWASP security principles, such as protection against SQL injection attacks and cross-site scripting, security standards can be improved. The move also opens up opportunities for the development of new features, integration of third-party services, and scalability as per future business needs, creating a strong foundation for more advanced and easy-to-develop applications.

Bibliography

Adam, Bob Maulana, Besari, Adnan Rachmat Anom, & Bachtiar, Mochamad Mobed. (2019). Backend server system design based on REST API for cashless payment system on retail community. *2019 International Electronics Symposium (IES)*, 208–213. https://doi.org/10.1109/ELECSYM.2019.8901668

Ashari, Ilham Firman, & Denira, Siraz Tri. (2023). Analisis Celah Keamanan dan Mitigasi Website E-learning Itera Menggunakan Owasp Zed Attack Proxy. *Dinamika Rekayasa*, *19*(1), 29–36.

Baniaş, Ovidiu, Florea, Diana, Gyalai, Robert, & Curiac, Daniel Ioan. (2021). Automated specification-based testing of REST APIs. *Sensors*, *21*(16), 5375.

Cahyani, Desi Dwi, Dewi, Luh Putu Windy Puspita, Suryadi, Kadek Dika Rama, & Listartha, I. Made Edy. (2021). Analisis Kerentanan Website SMP Negeri 3 Semarapura Menggunakan Metode Pengujian Rate Limiting dan OWASP. *INSERT Information System and Emerging Technology Journal*, *2*(2), 106–112.

Cleveland, Sean B., Jamthe, Anagha, Padhy, Smruti, Stubbs, Joe, Packard, Michale, Looney, Julia, Terry, Steve, Cardone, Richard, Dahan, Maytal, & Jacobs, Gwen A. (2020). Tapis API development with Python: best practices in scientific rest api implementation: experience implementing a distributed stream API. In *Practice and experience in advanced research computing* (pp. 181–187).

Dobrović, Željko, & Furjan, Martina Tomičić. (2020). SWOT analysis in the strategic planning process-Meta-modelling approach. *2020 IEEE 10th International Conference on Intelligent Systems (IS)*, 574–579. IEEE.

Dudjak, Mario, & Martinović, Goran. (2020). An API-first methodology for designing a microservice-based Backend as a Service platform. *Information Technology and Control*, *49*(2), 206–223.

Firzatullah, Raden Muhamad. (2021). Development of XYZ University's Student Admission Site Using Waterfall Method. *Jurnal Mantik*, *5*(1), 201–206.

Gong, Yifei, Gu, Feng, Chen, Kengbin, & Wang, Fei. (2020). The Architecture of Micro-services and the Separation of Frond-end and Back-end Applied in a Campus Information System. *2020 IEEE International Conference on Advances in Electrical Engineering and Computer Applications (AEECA)*, 321–324. IEEE.

Hämäläinen, Oona. (2019). *API-First Design with Modern Tools*.

Hsieh, Chao Hsien, Li, Changfeng, Wang, Ziyi, & Ke, Chih Horng. (2020). Development of Laravel Digital Platform Based on MVC Design Pattern for Compii created Data Structure-Take the Bible for Example. *2020 IEEE 3rd International Conference on Information Communication and Signal Processing (ICICSP)*, 475–480. IEEE.

Irawan, Yudie, Muzid, Syafiul, Susanti, Nanik, & Setiawan, Rhoedy. (2018). System Testing using Black Box Testing Equivalence Partitioning (Case Study at Garbage Bank Management Information System on Karya Sentosa). *The 1st International Conference on Computer Science and Engineering Technology Universitas Muria Kudus*.

Priyawati, Diah, Rokhmah, Siti, & Utomo, Ihsan Cahyo. (2022). Website vulnerability testing and analysis of website applications using OWASP. *International Journal of Computer and Information System (IJCIS), 3*(3), 142–147.

Zaoui, Fadwa, & Souissi, Nissrine. (2020). Roadmap for digital transformation: A literature review. *Procedia Computer Science, 175,* 621–628.