ISSN 2085-4552

Cloud-Based ERP System Backend Design, study case: PT Cranium Royal Aditama

Arnoldus Yitzhak Petra Manoppo[1], Wirawan Istiono[2]
[1,2] Informatic Dept, Universitas Multimedia Nusantara, Tangerang, Indonesia
[1]arnoldus.manoppo@student.umn.ac.id, [2]wirawan.istiono@umn.ac.id

Accepted 26 January 2024
Approved 31 January 2024

***Abstract*— An ERP system is a comprehensive software solution that facilitates the integration of all resources inside a firm. PT Cranium Royal Aditama is a company specializing in the development of Enterprise Resource Planning (ERP) solutions for other businesses. The ERP system of Cranium was constructed on the .Net architecture. Nevertheless, the situation remains unchanged. The Net framework is constrained by its compatibility exclusively with the Windows operating system. Cranium developed a cloud-based ERP system using Java programming language to ensure flexibility and compatibility with all operating systems. The backend of the ERP system is constructed utilizing a monolithic modular architecture, employing Java Springboot as a framework and PostgreSQL as the database. The purchasing, inventory control, and production planning modules are responsible for the design and development process. The design and development of the ERP system backend is now underway, however it is still in the development stage and is currently confined to a basic CRUD technique.**

***Index Terms*— ERP; Java Springboot; PostgreSQL; Modular Monoliths**

I. Introduction

An enterprise resource planning (ERP) system is a comprehensive software solution designed to combine all of a company's resources. Enterprise Resource Planning (ERP) facilitates seamless interconnection across all departments inside the firm [1], [2]. Consequently, numerous firms are inclined to use it due to its facilitation of departmental management and planning. Nevertheless, the process of creating an ERP system is complex and costly[3], [4]. The creation of an ERP system is fraught with dangers and might span across several years. Furthermore, the majority of ERP system developments result in failure [5], [6].

The development of enterprise resource planning (ERP) systems, both cloud-based and on-premise, is the area of expertise of PT Cranium Royal Aditama, a company. For businesses who are interested in constructing ERP systems that are both complicated and high-risk, Cranium provides solutions. Cranium's enterprise resource planning (ERP) system makes use of the.NET Framework. [7], [8]. The fact that the.NET framework is only compatible with the Microsoft Windows operating system, on the other hand, places limitations on the ways in which it can be utilized. This results in an ERP system that is less adaptive and has a lower market potential due to its exclusive compatibility with the Windows operating system. Furthermore, the Linux operating system, which currently occupies a prominent position, is not supported by the ERP system. The utilization of Windows servers in cloud services often results in greater expenses when compared to Linux servers. This is in addition to the difficulties regarding flexibility as previously mentioned [9], [10].

When all of these considerations were taken into account, Cranium came to the conclusion that it would be beneficial to revamp and reconstruct their system by utilizing the Java programming language. Java is well-known for its adaptability, since it can work on multiple platforms, including Windows, Linux, and Mac. The scope of design and development encompasses not only the modification of code but also the introduction of extra capabilities and modules, depending on the requirements. It is believed that Cranium's enterprise resource planning (ERP) system will be able to be adopted by a larger variety of businesses if this strategy is used.

II. Study Literature

A. Modular Monoliths

Modular monoliths serve as an architectural solution that combines the characteristics of classic monolithic programs and the widely embraced micro services. As software systems expand and develop [11], [12], there is a fundamental requirement for equilibrium a balance between the straight forwardness of a monolith and the adaptability of micro services. Introducing the modular monolith [13], an architectural pattern that effectively combines the advantages of two different approaches. When should one contemplate the adoption of a modular monolith [14], [15].

1. Initial Project Phase: In the early stages, opting for a modular monolith helps mitigate the burdens associated with establishing microservices, while still considering future scalability.
2. Elaborate Business Logic: If the system you are constructing contains extensive business logic that

are more manageable in a shared-memory system, a modular architecture can be advantageous.

3. Teams Unfamiliar with Microservices: If your team lacks familiarity with microservices, opting for a modular monolith might facilitate the development of modular thinking without the need for extensive learning.

B. Enterprise Resource Planning

Enterprise resource planning (ERP) is a software system that organisations employ to oversee and control many business functions, including accounting, procurement, project management, risk management and compliance, and supply chain operations. An all-encompassing ERP package comprises enterprise performance management software, which aids in the planning, budgeting, forecasting, and reporting of an organization's financial outcomes [1], [2], [16].

Enterprise Resource Planning (ERP) solutions integrate several corporate processes and facilitate the exchange of data between them. ERP systems gather an organization's shared transactional data from several sources, eliminating data duplication and ensuring data integrity by establishing a single authoritative source [3], [17]. Currently, Enterprise Resource Planning (ERP) systems play a crucial role in the management of numerous enterprises across all sectors and scales. For these companies, ERP is as essential as electricity, which is necessary for maintaining illumination [2], [5].

III. Methodology

Develop and construct the backend infrastructure of the ERP system. The design process commences by constructing an entity relationship diagram (ERD) and a database schema, often known as an ER description, for the ERP module. Next, proceed to clone the module as a starting template for constructing the backend. After the process of cloning, proceed with the development of create, read, update, delete, or CRUD methods and unit tests for each functionality utilised in the ERP system. In addition to that, this system was developed employing a software development life cycle that is of the Agile type. The subsequent information outlines the specific activities conducted throughout the execution of the research project.

1. Collaborated with the team to develop an Entity-Relationship Diagram (ERD) and provide a detailed description of the entities and relationships for the purchasing, inventory control, and production planning modules.
2. Duplicating the purchasing, inventory control, and production planning modules. The cloning method involves the creation of a new Java module within the project for every ERP module. Cloning is performed due to the monolithic modular architecture of the ERP project, which consists of multiple independent modules inside a single project.
3. Implementing CRUD operations for the following features: purchasing order, purchasing bill, inventory item withdraw reservation, inventory item transfer, production order, master plant, master area, master driver, master unit of measurement, and master activity standard.

The research commenced with the implementation of the tech stack and architecture employed in the design and development of ERP systems. The creation of ERP systems incorporates the utilisation of many tech stacks and architectures:

- The Springboot framework is utilised for the development of the backend system of an ERP. The rationale for utilising this framework lies in the fact that Springboot is a Java-centric framework, rendering it highly adaptable and compatible with all major operating systems. Furthermore, Springboot is renowned for being a well recognised framework with comprehensive documentation.
- PostgreSQL Database: PostgreSQL is a relational database that is open-source. Relational databases are utilised due to the multitude of interconnections between tables within the ERP system.
- Monolithic modular architecture refers to a type of architecture that is both monolithic and modular in character. This architectural design partitions the codebase into modules, with each module exhibiting loose coupling and the ability to function independently. Modifications made to one module do not have any impact on the other modules [8]. The decision to adopt a monolithic modular design was based on its superior scalability and maintainability compared to a traditional monolithic architecture. Furthermore, if the microservices architecture is to be used in the ERP system in the future, the task of modifying the architecture will be facilitated due to the codebase being partitioned into multiple autonomous modules.

During the execution of this research, three modules are focused on: purchasing, inventory control, and production planning. Below is the Entity-Relationship Diagram (ERD) representing the three modules.

A. Purchasing Modul

Figure 1 displays the Entity-Relationship Diagram (ERD) representing the purchasing module. There is a grand total of 12 tables, which can be further categorized into 6 main/header tables and 6 detail tables. The six primary tables consist of the request, order, order item receipt, bill, order return, and return bill. The six tables are interconnected in the following manner: the request table has a one-to-one relationship with the order table, the order table has a one-to-one relationship with the order item receipt table, the order item receipt table has a one-to-one relationship with the bill table, the order return table has a one-to-one relationship with the order table, and the return bill table has a one-to-one relationship with the order return table.

Every primary table is associated with a single secondary table through a one-to-many relationship.

B. Inventory Control

Figure 2 displays the Entity-Relationship Diagram (ERD) of the inventory control module. There is a grand total of 12 tables, which can be further categorised into 6 main/header tables and 6 detail tables. The six primary tables consist of item withdrawal reservation, item withdrawal, item transfer reservation, item transfer, item reception, and stock adjustment. Out of the six tables, four are primary tables that have connections with other primary tables. Specifically, the item withdraw reservation table has a one-to-one relationship with the item withdraw table, and the item transfer reservation table has a one-to-one relationship with the item transfer table. Every primary table is associated with one detail table through a one-to-many relationship.

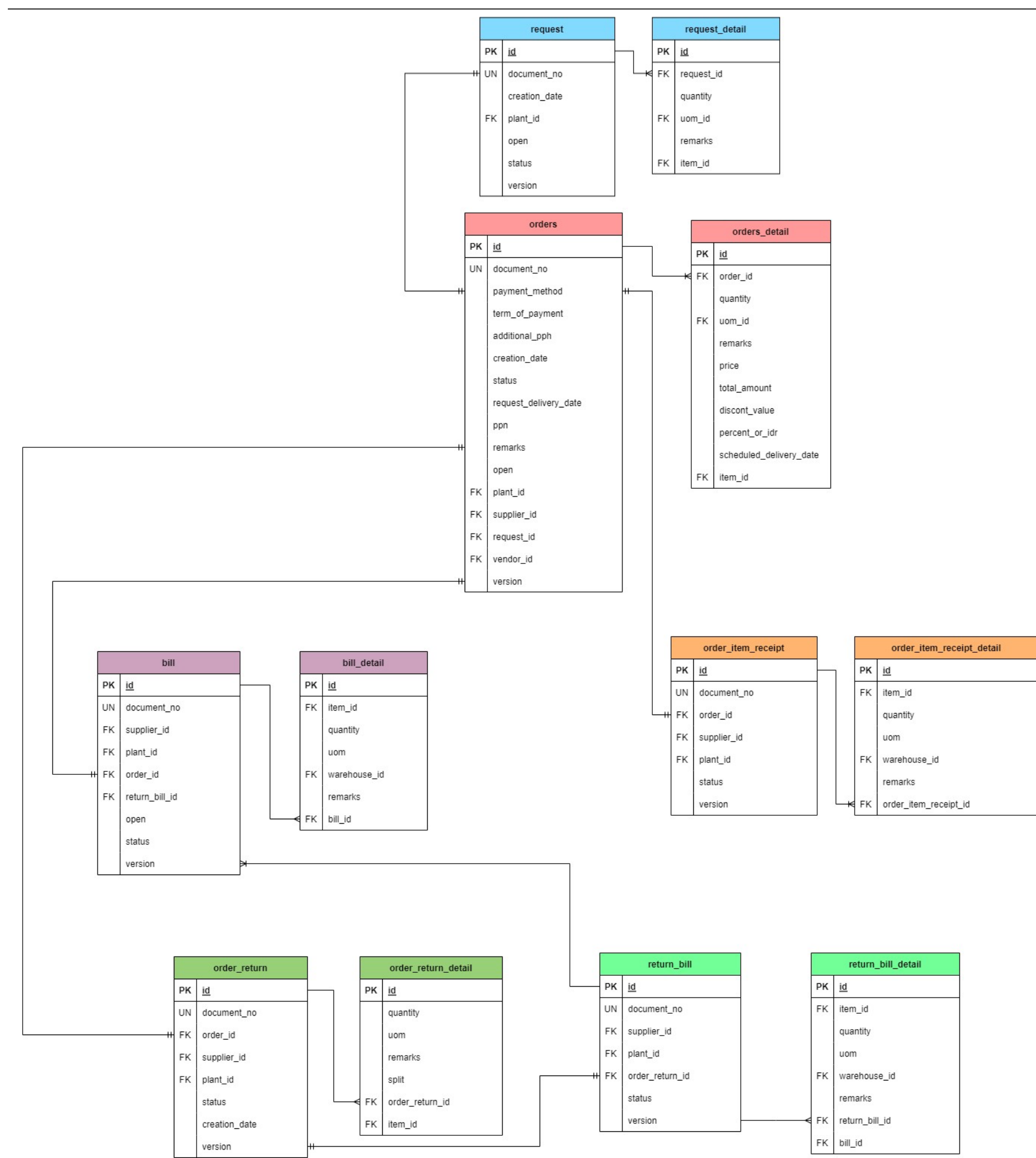

Fig. 1. ERD Purchasing Modul

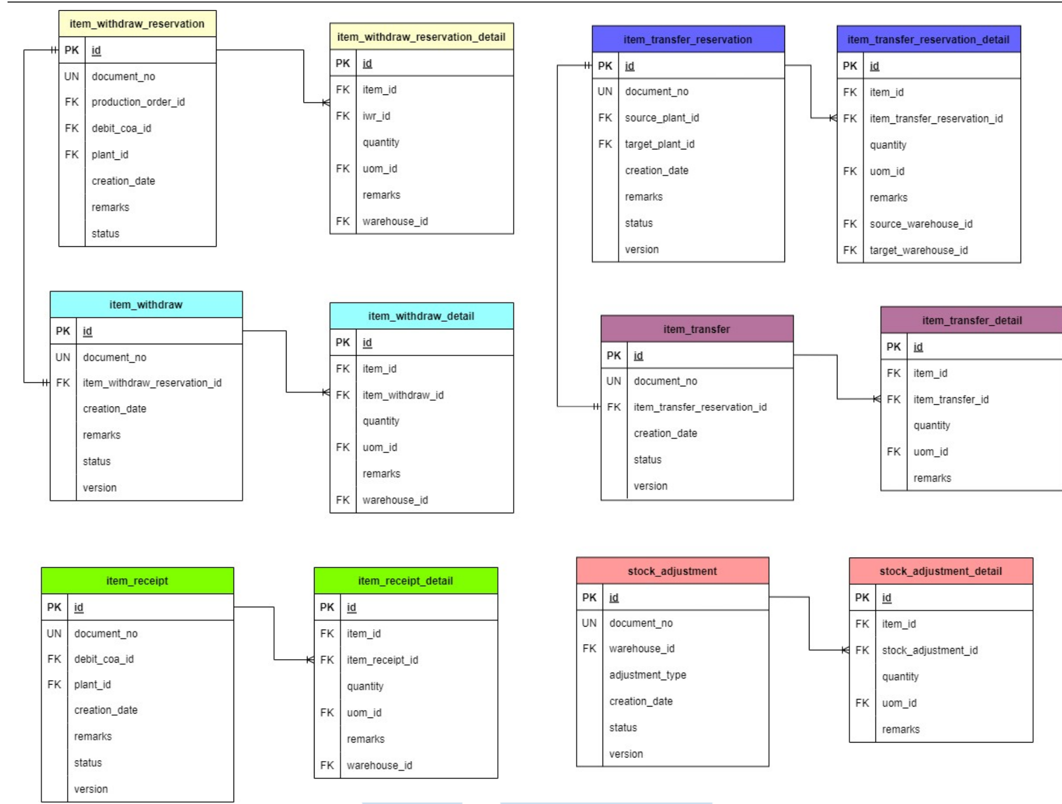

Fig. 2. ERD Inventory Control Modul

C. *Production Planning*

Figure 3 displays the Entity-Relationship Diagram (ERD) of the production planning module. There are a total of seven tables, which include three main/header tables and four detail tables. The three primary tables are order, order item receipt, and planned order. The order table has a one-to-one link with the order item receipt table. Every primary table is associated with one detail table, except for the order table which has two detail tables. Each header table has a one-to-many link with the detail table.

The Springboot architecture comprises four hierarchical levels that engage in communication with one another [9]. The four levels consist of the database layer, persistence layer, business layer, and API layer.

Figure 4 shows the architectural scheme used by Springboot. The following are the functions of the four layers:

1. Database layer: the layer that performs CRUD operations.
2. Persistence layer: the layer that contains storage logic and converts objects into database rows or vice versa.
3. Business layer: the layer that handles business logic, validation, and authorization.
4. API layer: the layer that handles HTTP requests and converts JSON into objects or vice versa.

Figure 4 not only illustrates the architectural framework of Springboot but also elucidates the process of how Springboot manages requests and responses. The client will utilise REST API methods (get, post, patch, delete) to submit queries. When making a request using the post or patch method, JSON data will be transformed into a data transfer object (DTO). A Data Transfer Object (DTO) is an encapsulated object that contains both the request and response data [10]. Subsequently, the API layer or controller will receive the object and subsequently transmit it to the business layer or service. Subsequently, the service invokes the persistence layer or repository. The repository and the database have a close correlation in executing CRUD operations. Upon executing a CRUD action, the repository will provide the service with an entity or object that corresponds to a single database record. The service will transform the entity into a DTO and will furthermore manage the operational rules prior to delivering the DTO to the controller. The controller will provide the Data Transfer Object (DTO) and transform it into JSON format, which will serve as the response received by the client.

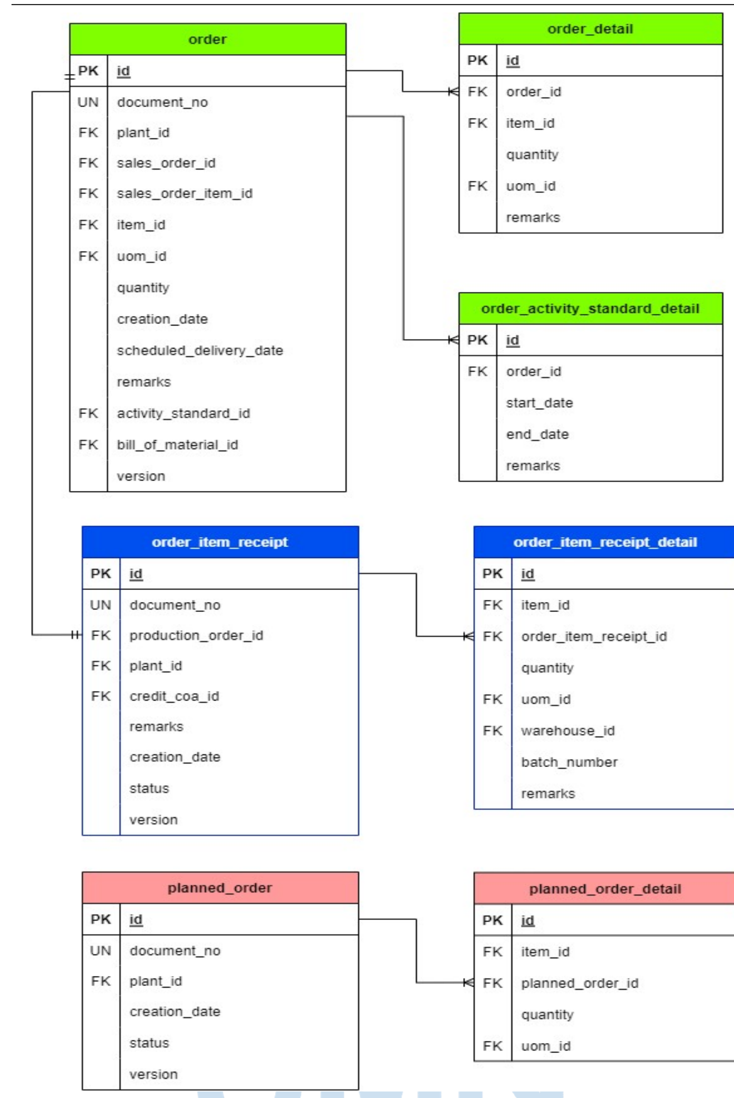

Fig. 3. ERD Modul Production Planning

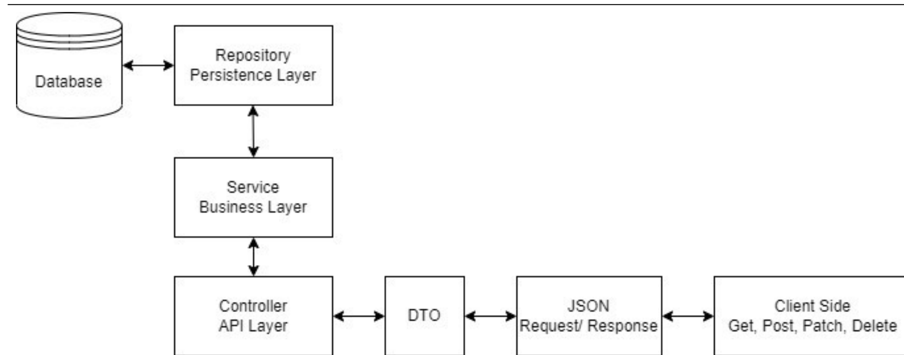

Fig. 4. Spingboot Application Architecture

IV. Result

A. Unit Testing

Unit testing is a form of software testing that examines individual components of a system [11]. Unit testing involves the independent testing of each procedure on the service layer. Unit testing involves evaluating the procedure being tested in two specific scenarios: success and failure circumstances. Unit testing in Springboot involves the use of class assertions or classes specifically designed to test programmes based on predefined assumptions [12]. If the assertion is successful, the project installation can proceed without any issues. However, if the assertion fails, the project installation will not be successful.

```
@Test
void testFindOrderById() throws DataNot
FoundException {
Long orderId = 1L;
String documentNoExpected=" POD0000001 " ;
Purchasing OrderDto
purchasingOrderDto=purchasingOrderService.findOrder
ById(orderId) ;

assertEquals(purchasingOrderDto.gtoueto().documen
tNoExpected) ;
}
```

Code 1. Example of a Successful Unit Test

Code snippet 1 demonstrates a unit testing example where the read one data method is tested for a successful condition. The unit test utilises the assertEquals method to compare the documentNo object (actual value) with the documentNoExpected object (anticipated value). Unit testing is conducted not only to assess the success condition but also to evaluate the failure condition.

```
@Test
void testFindOrderByIdThrowException()throws
DataNotFoundException{
LongorderId=10L;
Exceptionexception=assertThrows(DataNotFoundEx
ception.class
,()->{
    PurchasingOrderDtopurchasingOrderDto=
    purchasingOrderService.findOrderById(orderId);
});
StringexpectedMessage="Order dengan id 10 tida
kada";
StringactualMessage=exception.getMessage();
assertTrue(actualMessage.contains(expectedMessag
e));
```

Code 2. Contoh Unit Test Gagal

Code snippet 2 provides an illustration of unit testing on a failing condition for the read one data method. The unit test executes two assertions, specifically assertThrows and assertTrue. The assertThrows method is used to make a method call with specific circumstances that may result in a data not found exception. The assertTrue method takes the retrieved exception message (actualMessage) and checks if it contains the expected message (expectedMessage).

B. Rest API Testing

REST API testing is conducted to verify the functionality of the 5 REST API endpoints that have been developed. Here is an example of testing the create method on the inventory item withdraw reservation functionality of the REST API.

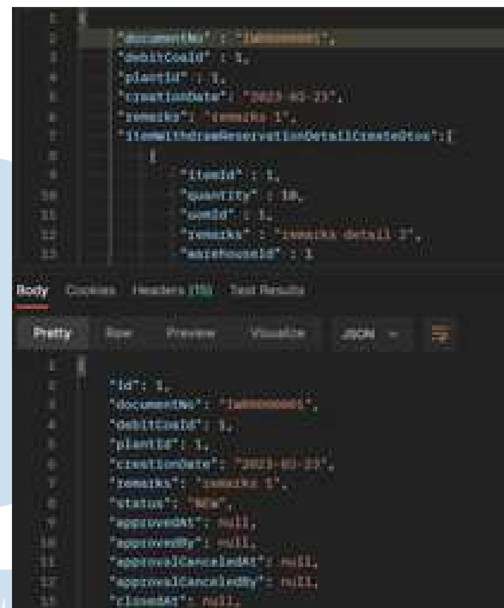

Fig. 5. REST API Method Create Endpoint Test Results

Figure 5 displays the outcome of doing tests on the create method of the REST API. The create method receives a JSON data as a request body, which includes headers and metadata. Information can be transmitted via multiple arrays. Upon a successful request, the status response will indicate a 201 created status code and include a JSON data as the response body.

C. Read Satu Data

Figure 6 demonstrates the testing of the REST API method for retrieving a single data entry in the inventory item withdraw reservation function.

Figure 6 displays the outcome of doing a test on the REST API method for retrieving a single data entry. The "read one data" method requires an id parameter in order to retrieve the desired data. Upon a successful request, the response status will indicate "200 OK" and the response body will be returned as JSON data.

Fig. 6. REST API Endpoint Test Results Method Read One Data

D. Read Paging

Figure 7 demonstrates the testing of the REST API method for reading paging on the inventory item withdraw reservation feature.

Fig. 7. REST API Method Read Paging Endpoint Test Results

Figure 7 depicts the outcome obtained from doing tests on the REST API read paging method. The read paging method requires two parameters: page and size. Upon successful completion of the request, the response status will indicate "200 OK" and the response body will consist of JSON data.

E. Update

Figure 8 exemplifies the testing of the REST API method update on the inventory item withdraw reservation functionality.

Fig. 8. REST API Method Update Endpoint Test Results

Figure 8 depicts the outcome of conducting tests on the update or patch method of the REST API. The update method receives an id argument and JSON data in the request body, which includes headers and information. Information can be transmitted via multiple arrays. Upon a successful request, the response status will indicate "200 OK" and the response body will be returned as JSON data.

F. Delete

Figure 9 demonstrates the testing of the REST API method delete on the inventory item withdraw reservation functionality.

Fig. 9. REST API Method Delete Endpoint Test Results

Figure 9 displays the outcome of doing tests on the delete method of the REST API. The delete method

requires the submission of a single id parameter in order to delete the corresponding data. Upon a successful request, the response status will indicate "200 OK" and the response body will be returned as JSON data.

V. CONCLUSION

The backend for the cloud-based ERP system at PT Cranium Royal Aditama has been designed and developed, however it is still in the development stage. The design and development process has currently only progressed to the point of implementing a basic CRUD mechanism for each feature in the ERP system. The prolonged duration required for designing and constructing the ERP system is attributed to its intricate intricacy.

The design and development process consists of three distinct modules: purchasing, inventory management, and production planning. These modules encompass several essential aspects such as purchasing orders, purchasing bills, inventory item withdraw reservation, inventory item transfer, and production orders.

ACKNOWLEDGMENT

Thank you to Universitas Multimedia Nusantara in Indonesia for providing a space for academics to conduct this journal research. Hopefully, this research will contribute significantly to the growth of technology in Indonesia.

REFERENCES

[1] M. Hadidi, M. Al-Rashdan, S. Hadidi, and Y. Soubhi, “Comparison between cloud ERP and traditional ERP,” *J. Crit. Rev.*, vol. 7, no. 3, pp. 140–142, 2020, doi: 10.31838/jcr.07.03.26.

[2] A. Razzaq, Siti Azirah Asmai, M. Saad Talib, N. Ibrahim, and A. A. Mohammed, “Cloud ERP in Malaysia: Benefits, Challenges, and Opportunities,” *Int. J. Adv. Trends Comput. Sci. Eng.*, vol. 9, no. 5, pp. 7510–7516, Oct. 2020, doi: 10.30534/ijatcse/2020/85952020.

[3] N. M. Alsharari, M. Al-Shboul, and S. Alteneiji, “Implementation of cloud ERP in the SME: evidence from UAE,” *J. Small Bus. Enterp. Dev.*, vol. 27, no. 2, pp. 299–327, Apr. 2020, doi: 10.1108/JSBED-01-2019-0007.

[4] T. Febrianto, D. Soediantono, S. Staf, K. Tni, and A. Laut, “Enterprise Resource Planning (ERP) and Implementation Suggestion to the Defense Industry: A Literature Review,” 2022. [Online]. Available: http://www.jiemar.org

[5] R. Kenge and Z. Khan, “A Research Study on the ERP System Implementation and Current Trends in ERP,” *Shanlax Int. J. Manag.*, vol. 8, no. 2, pp. 34–39, Oct. 2020, doi: 10.34293/management.v8i2.3395.

[6] P. Ruivo, B. Johansson, S. Sarker, and T. Oliveira, “The relationship between ERP capabilities, use, and value,” *Comput. Ind.*, vol. 117, May 2020, doi: 10.1016/j.compind.2020.103209.

[7] F. Matrone *et al.*, “A BENCHMARK for LARGE-SCALE HERITAGE POINT CLOUD SEMANTIC SEGMENTATION,” *Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci. - ISPRS Arch.*, vol. 43, no. B2, pp. 1419–1426, 2020, doi: 10.5194/isprs-archives-XLIII-B2-2020-1419-2020.

[8] W. I. Kevin Hendy, “Efficiency Analysis of Binary Search and Quadratic Search in Big and Small Data,” *Comput. Sci. Tech.*, vol. 7, no. 1, pp. 605–615, 2020, doi: 10.15181/csat.v7i1.2091.

[9] O. Alzakholi, L. Haji, H. Shukur, R. Zebari, S. Abas, and M. Sadeeq, “Comparison Among Cloud Technologies and Cloud Performance,” *J. Appl. Sci. Technol. Trends*, vol. 1, no. 2, pp. 40–47, 2020, doi: 10.38094/jastt1219.

[10] W. Philips and W. Istiono, “Analysis of MinFinder Algorithm on Large Data Amounts,” *Int. J. Emerg. Trends Eng. Res.*, vol. 9, no. 6, pp. 627–632, 2021, doi: 10.30534/ijeter/2021/04962021.

[11] A. J. Clair, J. A. Gabor, K. S. Patel, S. Friedlander, A. J. Deshmukh, and R. Schwarzkopf, “Subsidence Following Revision Total Hip Arthroplasty Using Modular and Monolithic Components,” *J. Arthroplasty*, vol. 35, no. 6, pp. S299–S303, 2020, doi: 10.1016/j.arth.2020.03.008.

[12] R. M. da Ponte *et al.*, “Monolithic integration of a smart temperature sensor on a modular silicon-based organ-on-a-chip device,” *Sensors Actuators, A Phys.*, vol. 317, p. 112439, 2021, doi: 10.1016/j.sna.2020.112439.

[13] S. Djeljadini *et al.*, “Porous PVDF Monoliths with Templated Geometry,” *Adv. Mater. Technol.*, vol. 6, no. 11, 2021, doi: 10.1002/admt.202100325.

[14] A. Agarwala *et al.*, “One Network Fits All? Modular Versus Monolithic Task Formulations in Neural Networks,” *ICLR 2021 - 9th Int. Conf. Learn. Represent.*, pp. 1–30, 2021.

[15] S. Mittal, Y. Bengio, and G. Lajoie, “Is a Modular Architecture Enough?,” *Adv. Neural Inf. Process. Syst.*, vol. 35, no. NeurIPS, pp. 1–14, 2022.

[16] E. S. Kappenman, J. L. Farrens, W. Zhang, A. X. Stewart, and S. J. Luck, “ERP CORE: An open resource for human event-related potential research,” *Neuroimage*, vol. 225, no. October 2020, p. 117465, 2021, doi: 10.1016/j.neuroimage.2020.117465.

[17] V. Christiansen, M. Haddara, and M. Langseth, “Factors Affecting Cloud ERP Adoption Decisions in Organizations,” *Procedia Comput. Sci.*, vol. 196, no. 2021, pp. 255–262, 2021, doi: 10.1016/j.procs.2021.12.012.