Architecting the e-RES (Electric Bike Rescuer) Application Based on Microservices Architecture

Jefri Abdurrozak Ismail [a,1,*]

[a] Magister Teknologi Informasi UPN "Veteran" Jawa Timur, Jl.Raya Rungkut Madya, Surabaya 60294, Indonesia
[1] jefriabdurrozakismail@gmail.com*
* corresponding author

ARTICLE INFO

Keywords
e-RES
Electric bike
Emergency assistance
Microservices
Mobile application

ABSTRACT

The e-RES (Electric Bike Rescuer) application is designed to provide real-time assistance to electric bike riders facing technical issues on the road. The app utilizes a microservices-based architecture to efficiently handle tasks like user authentication, location tracking, assistance requests, notifications, and payments. This journal discusses the system architecture of e-RES. Through diagrams like Use Case, Class, Sequence, Activity, and Entity Relationship Diagrams (ERD), this paper illustrates the interactions between users, technicians, and the system. The application aims to bridge the gap between electric bike riders and technicians, providing a seamless experience from the moment a request for assistance is made, to the resolution of the issue, and final payment processing.

1. Introduction

The growing concern about environmental issues and the increasing need for sustainable transportation solutions have significantly boosted the demand for environmentally friendly vehicles, including electric bikes (e-bikes). These bikes offer a cleaner and quieter alternative to traditional gas-powered vehicles[1]. They contribute to reducing carbon emissions and alleviating traffic congestion, making them an attractive option for eco-conscious commuters. However, despite their popularity, electric bikes come with a unique set of challenges. One of the most pressing issues for riders is the lack of technical support and repair services, particularly when they are on the road. If an e-bike breaks down, riders may find it difficult to quickly access a nearby technician who can resolve the issue, especially in areas that are not well-served by traditional bike repair shops. This is where the e-RES application comes into play.

The e-RES (Electric Bike Rescuer) application is designed to bridge the gap between e-bike riders and technicians by providing a platform that connects them instantly when assistance is needed. The app enables riders to request help in real-time, ensuring that they can get assistance quickly. To achieve this, the e-RES app uses microservices-based architecture. In a microservices architecture, the system is divided into smaller, independent services, each focusing on a specific function[2] (e.g., user authentication, location tracking, notifications, etc.). This approach offers several key benefits, including scalability, flexibility, and maintainability. Individual services can be scaled independently based on demand, ensuring the system can grow and handle more users without compromising performance. The modular design also allows for easier updates and maintenance, as services can be improved or replaced without affecting other parts of the system. Moreover, with each service functioning independently, the system is more resilient; if one service faces issues, others can continue operating without disruption.

The use of mobile technology ensures that the platform is accessible and user-friendly for riders on the go. Riders can use the app on their smartphones to request help, track technicians, and make payments seamlessly, providing a smooth and reliable experience. Ultimately, the e-RES application aims to address a critical need for electric bike riders by making technical assistance more readily available, providing peace of mind, and improving the overall reliability of e-bike usage, especially when riders face unforeseen technical issues on the road.

2. Method

Microservices architecture is a software design pattern where an application is divided into a set of small, independent services, each responsible for a specific function[3]. For the e-RES application, these functions include tasks like user authentication, location tracking, and payment processing. Unlike traditional monolithic architectures, where all components are tightly integrated, microservices operate independently, allowing them to be developed, deployed, and scaled separately[4]. This modular approach enhances the system's maintainability, scalability, and flexibility. It enables easier updates and ensures that changes to one service do not affect the others, improving overall development speed and reducing complexity[5].

Scalability is another significant advantage of microservices. As each service can be scaled independently, the system can handle increasing workloads better[6]. For instance, if the location tracking service experiences high demand, it can be scaled up without impacting other services. This capability ensures that the system remains responsive and capable of handling growth, particularly as the number of users or service requests increases[7].

The Agile development framework complements microservices by promoting flexibility and iterative progress. In Agile, work is divided into smaller segments known as sprints, which typically last from one to four weeks[8]. During each sprint, the development team focuses on completing a specific feature or task. After each sprint, a feedback cycle occurs, where stakeholders review the work and provide input. This feedback allows for adjustments to be made, ensuring that the system continuously improves and aligns with user needs[9]. The Agile approach fosters rapid feature deployment and efficient bug resolution, enhancing the overall performance and reliability[10] of the e-RES application.

To facilitate communication between the various microservices, the e-RES application uses RESTful APIs and Pub/Sub messaging. RESTful APIs are web services that follow the principles of REST (Representational State Transfer)[11], allowing different parts of the system to communicate with each other using standard HTTP methods like GET, POST, PUT, and DELETE[12]. This ensures smooth data exchange between services, such as when a rider makes a request or a technician is notified. Pub/Sub messaging is a messaging pattern where services act as publishers or subscribers[13]. Publishers send messages (e.g., notifications or location updates), while subscribers receive those messages. This allows asynchronous communication, meaning services can send requests or updates without waiting for a response[14]. As a result, data exchange becomes more efficient, and the system operates with low latency, particularly in real-time tasks[15] like location tracking or notifications.

The combination of microservices architecture and the Agile development framework allows the e-RES application to be flexible, scalable, and maintainable. By employing RESTful APIs and Pub/Sub messaging, the system ensures low-latency responses and efficient inter-service communication, which is crucial for delivering a smooth and reliable user experience, especially in a fast-paced, real-time environment like emergency bike assistance.

3. Results and Discussion

The architecture of the e-RES application is designed using a microservices model, where the system is divided into independent core services. This modular approach ensures that the system is scalable, flexible, and easier to manage. One of the key components is the Authentication Service, which handles user authentication through OAuth 2.0, ensuring secure access and validating user identities. Another critical service is the Location Service, which uses GPS technology to track the real-time locations of both riders and technicians, facilitating precise coordination. The Notification Service

adopt the Pub/Sub messaging paradigm to send asynchronous push notifications, ensuring timely communication regarding technician availability, service status, and other important updates. The Payment Service ensures that transactions between riders and technicians are seamless, providing a secure and efficient payment processing system for services rendered. Finally, the History Service stores records of past assistance requests and services provided, allowing users to access historical data for future reference or follow-up needs. These services collectively enable the e-RES app to provide a reliable, scalable, and user-friendly platform for electric bike riders and technicians.

3.1. Use Case Diagram

The Use Case Diagram (Fig. 1) illustrates the primary interaction flow between users, including riders and technicians, and the e-RES system. It captures key scenarios such as assistance request, where riders seek technical help when needed. The diagram includes the tracking feature, enabling riders to monitor the real-time location of technicians. Additionally payment functionality allows riders to make payments directly through the app after receiving assistance. Overall, the diagram emphasizes the system's user-centric design, ensuring a seamless interaction between the application and its users.

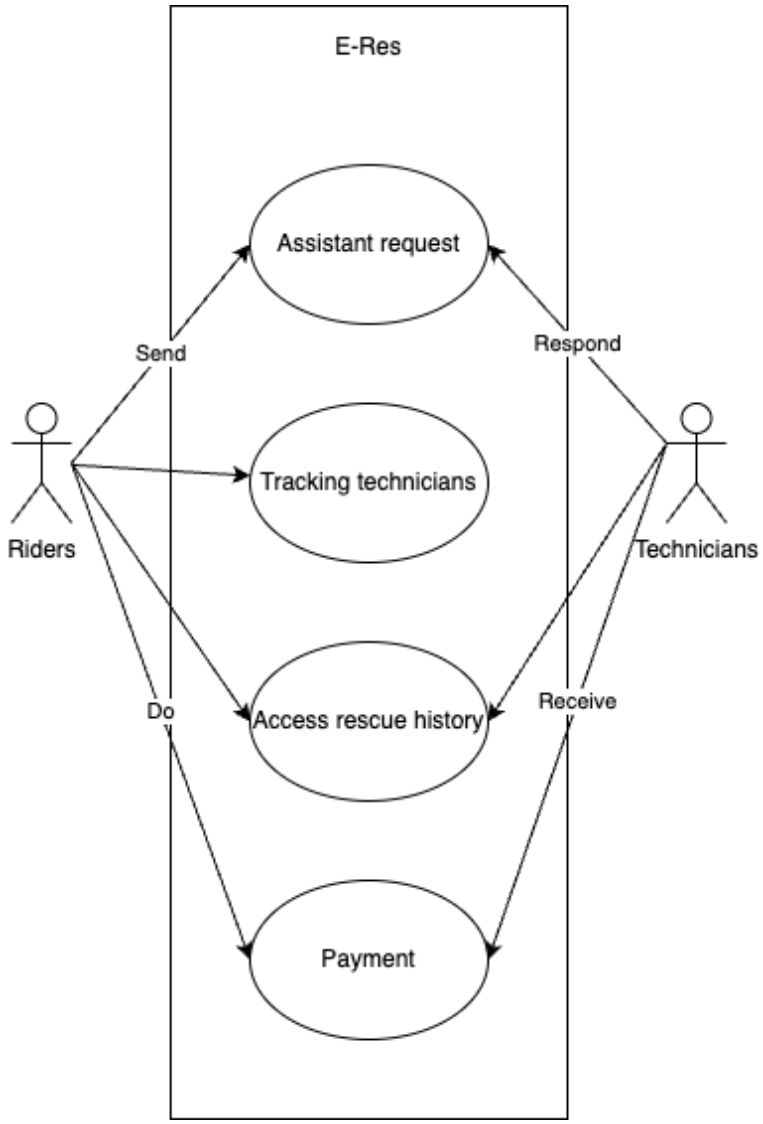

Fig. 1. Use Case Diagram

3.2. Class Diagram

The Class Diagram (Fig. 2) visually represents the data structure of the e-RES application, defining the relationships between key entities. The User class represents any registered individual within the

system, encompassing both riders and technicians. The Rider class, a subclass of User, includes attributes such as request history, location, and assistance status. Similarly, the Technician class, also a subclass of User, is responsible for receiving and responding to assistance requests. The Assistance Request class serves as the core for storing information about a rider's request for help. The Notification class ensures users are kept informed of updates, such as technician arrival times or changes in service status. Lastly, the Payment class manages the payment process between riders and technicians. This well-structured framework allows for efficient data management and relationship handling, ensuring the e-RES app remains scalable.

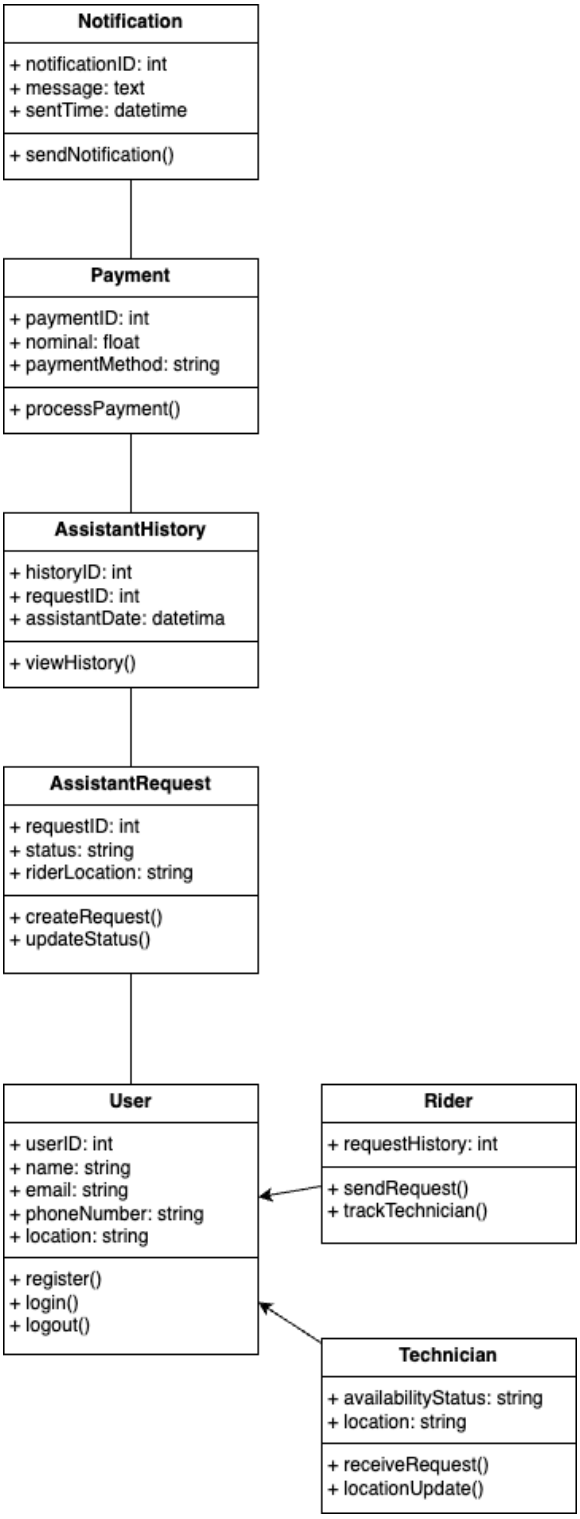

Fig. 2. Class Diagram

3.3. Sequence Diagram

Referring to Fig. 3, the sequence begins when the Rider initiates a request for assistance by sending it to the e-Res Server. This request typically occurs when the rider encounters a technical issue with their electric bike while on the road. Upon receiving the request, the e-Res Server fetches the Rider's location in order to pinpoint their exact coordinates. This location data is crucial for identifying which technician is closest to the rider, thereby ensuring a swift response.

Once the e-Res Server retrieves the rider's location, it proceeds to search for nearby technicians who are available to provide assistance. This process involves scanning the technician database or real-time availability to determine which technicians are closest to the rider's location. The server then sends a notification request to the nearest technician, alerting them about the rider's request and providing the exact location where their assistance is required. The notification includes key details, such as the notes and details of the assistance and the rider's location, ensuring that the technician has all the necessary information before proceeding.

After receiving the notification, the Technician confirms their availability by responding to the e-Res Server, indicating that they are ready and able to assist the rider. This confirmation step is essential to ensure that the rider is not left waiting for a technician who is unavailable. Once the technician confirms, the e-Res Server sends an estimated time of arrival (ETA) to the Rider, informing them of when to expect the technician’s arrival. This gives the rider a clue of how long they will need to wait and reassures them that help is on the way.

After the technician arrives and completes the assistance job, whether it’s a repair, troubleshooting, or any other required task, the e-Res Server processes the payment for the service rendered. The payment system is integrated into the app, allowing riders to pay for the technician's services seamlessly. Once the payment has been processed, the e-Res Server generates and provides the Rider with a receipt. This receipt serves as proof of the completed transaction and the services provided, ensuring that both the rider and the technician have a record of the service performed and the payment made.

Throughout this entire sequence, the e-Res Server plays a crucial role in managing the flow of requests, coordinating the interactions between the rider and technician, ensuring timely communication, and processing payments, all while maintaining a seamless experience for the rider. The integration of real-time location tracking, technician availability, and secure payment processing guarantees that the rider receives reliable assistance when they need it most.

3.4. Activity Diagram

Referring to Fig. 4, the Activity Diagram outlines the step-by-step workflow of the e-RES system when a rider requests assistance. The process begins when the Rider encounters an issue with their electric bike and initiates a request for help. This action starts the sequence and triggers the e-Res Server to begin processing the request and find an available technician.

Once the rider has sent the assistance request to the e-Res Server, the system checks the technician availability. The server searches for nearby technicians who are available to help the rider. If an available technician is found, the e-Res Server assigns the request to the technician and notifies them of the task.

After receiving the assignment, the Technician confirms their availability to assist the rider. This confirmation is crucial to ensure that the rider is not left waiting for an unavailable technician. Once confirmed, the Technician begins assisting the rider to resolve the bike's issue.

Throughout this process, the e-Res Server continuously monitors the assistance status to track the technician's progress. If the assistance is completed, the system prompts the Rider for payment. If the assistance is still ongoing, the system continues to monitor the technician’s status until the task is finished.

Once the technician completes the assistance and payment is requested, the Rider proceeds to pay for the services rendered. The system processes the payment securely, and after the payment is

confirmed, the e-Res Server provides the Rider with a receipt, which serves as proof of the completed service and payment.

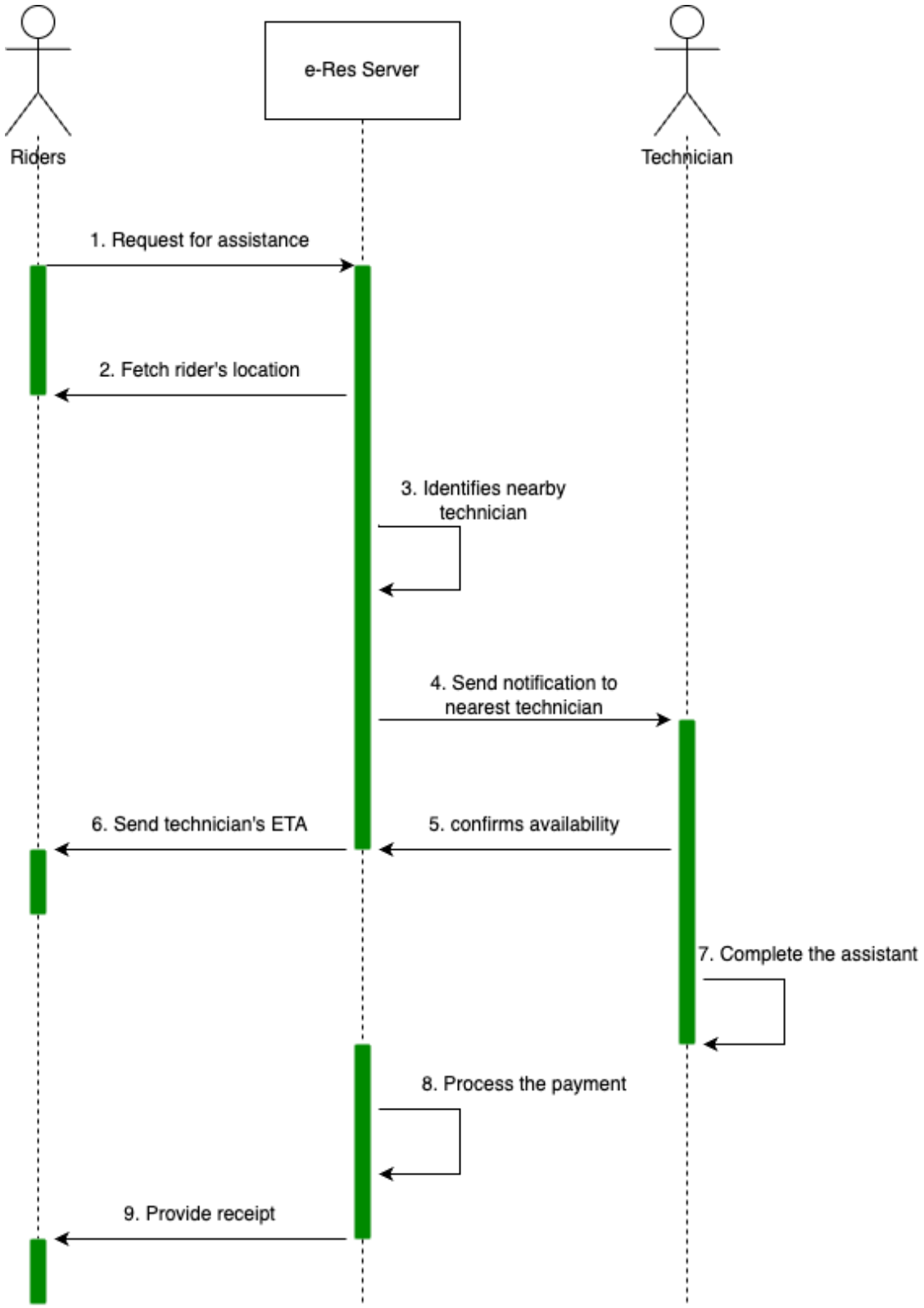

Fig. 3. Sequence Diagram

This sequence, as illustrated in Fig. 4, ensures a smooth and seamless interaction between the rider, technician, and the e-RES system, from the initial request for assistance through to the completion of the payment and receipt process.

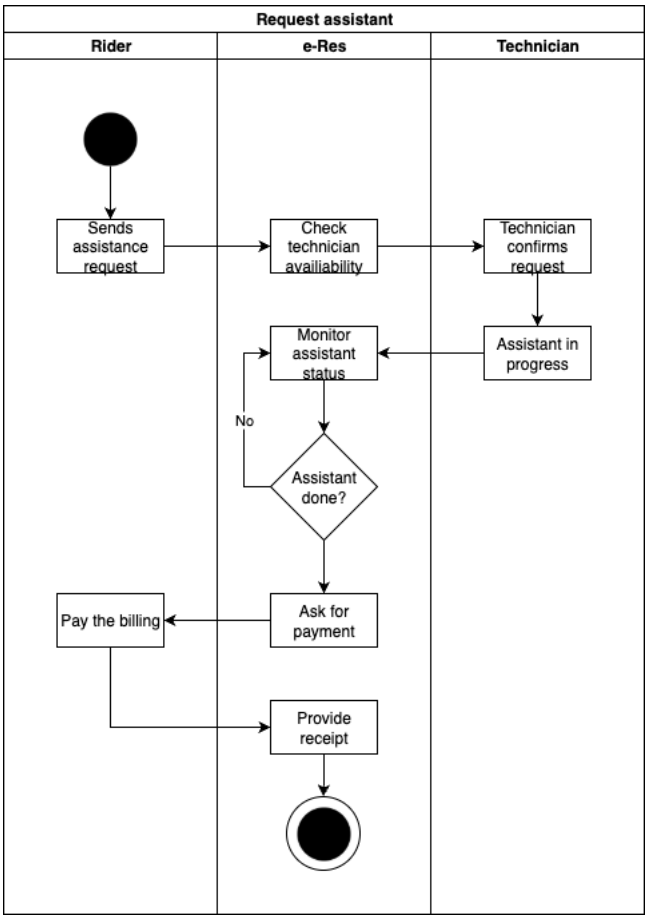

Fig. 4. Activity Diagram

3.5. Entity Relationship Diagram (ERD)

Referring to fig.5: Activity Diagram from your research, A user in the system, which can be a rider or technician, has the ability to generate and receive notifications. Each user can have many notifications, which may inform them about key events, such as updates to their requests, technician availability, or service completion. In addition, a user can submit multiple assistance requests, especially in scenarios where ongoing issues or multiple support needs arise. This reflects the system's flexibility in managing a variety of requests, ensuring that riders can receive assistance as often as needed.

Technicians are also pivotal entities in the system. A technician can handle multiple assistance requests, thus managing several users' needs simultaneously. Additionally, a technician may receive multiple notifications, which keeps them informed about requests for help, the status of their current tasks, or any urgent updates from riders.

The notifications in the system are closely tied to assistance requests. Some notifications will be related to a specific request, providing real-time updates to both users and technicians. However, notifications can also exist independently, offering generic updates or reminders that are not necessarily linked to an immediate request.

The assistance request has a direct link to payment. Each assistance request requires a payment before the process can be considered complete. This ensures that technicians are compensated for their work, and riders can settle their dues for the services provided. The payment relationship solidifies the transactional nature of the system, ensuring that financial exchanges are properly managed after each assistance request is resolved.

This structure promotes an organized flow where users and technicians are notified, tracked, and compensated, maintaining a smooth service experience within the e-RES system.

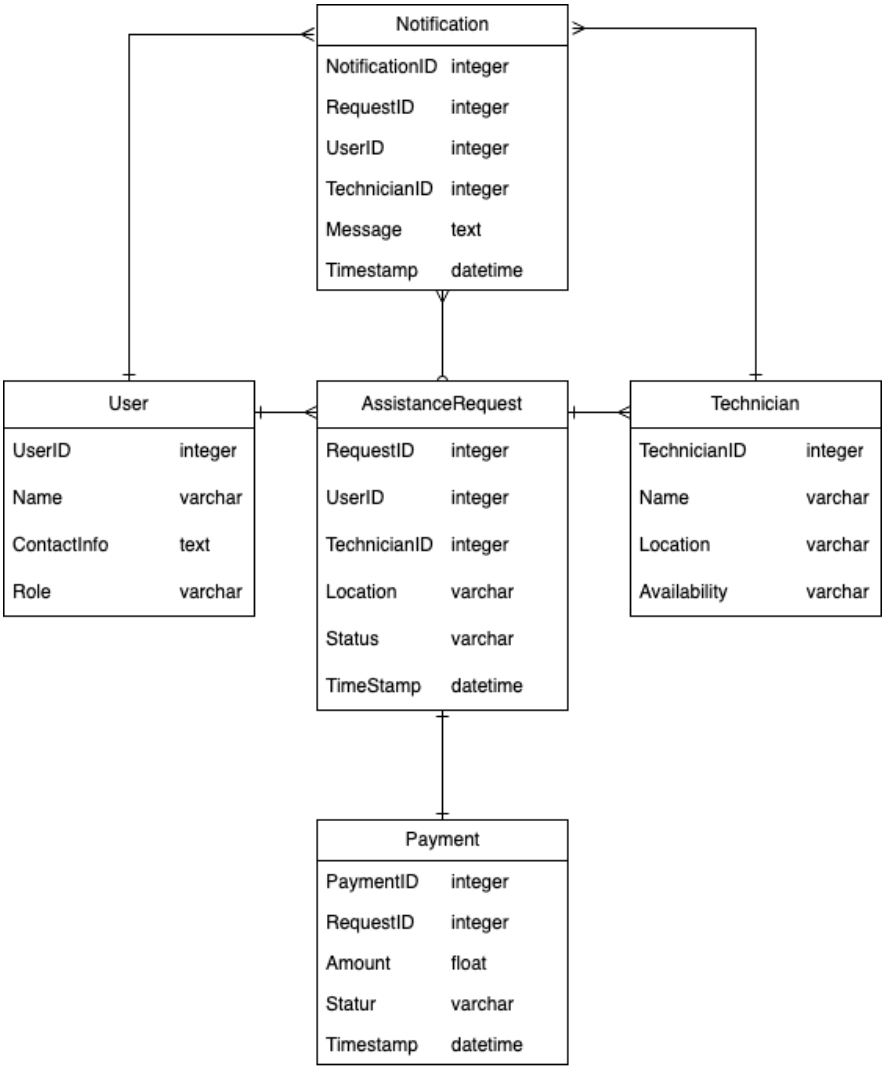

Fig. 5. Entity Relationship Diagram

4. Conclusion

The e-RES application addresses a critical need for electric bike riders by offering real-time technical assistance. By utilizing a microservices-based architecture, the application ensures that each service operates independently, allowing for greater scalability and flexibility. This modular design not only enhances the system's performance but also ensures that individual services can be updated or replaced without disrupting the overall operation of the application. As the number of users grows, this scalability becomes essential in maintaining performance and reliability, especially when managing multiple assistance requests simultaneously.

Moreover, the system's integration of various services—such as authentication, location tracking, notifications, and payment processing—enables seamless coordination between users and technicians. The Use Case Diagram and Class Diagram effectively illustrate the flow of interactions between these entities, showing how the app connects riders with nearby technicians based on real-time data. This allows for swift response times and ensures that riders can receive assistance at the right moment, significantly reducing downtime when a technical issue arises on the road.

The Sequence Diagram and Activity Diagram further demonstrate how the application manages the entire assistance process, from the initial request to payment processing. These diagrams highlight the key stages in the workflow, ensuring that users are informed about their request status, technician availability, and estimated time of arrival (ETA). This transparency in communication improves the overall user experience, giving riders peace of mind knowing that help is on the way and that payments can be securely processed within the app.

The Entity Relationship Diagram (ERD) reveals the relationship between key components, such as users, technicians, notifications, assistance requests, and payments. By managing these relationships, e-RES ensures that the entire process remains streamlined, minimizing delays and maximizing user experiences for both riders and technicians.

In conclusion, the e-RES application stands as a solution for electric bike riders in need of on-the-road assistance. Its use of modern technologies such as microservices, real-time location tracking, and secure payment processing ensures that riders have access to quick, reliable support whenever needed. With its robust system architecture and seamless user experience, the e-RES application is poised to significantly improve the reliability and convenience of electric bike usage, making it an indispensable tool for both riders and technicians.

References

[1] I. Oyeyemi Olayode, E. Jamei, and F. Justice Alex, "Integration of e-bikes in public transportation based on their impact, importance, and challenges: A systematic review," Multimodal Transportation, vol. 4, no. 1, p. 100182, Mar. 2025, doi: 10.1016/j.multra.2024.100182.

[2] A. Ganje, "Microservices in Organizations," Journal of Software Engineering and Applications, vol. 18, no. 02, pp. 76–86, 2025, doi: 10.4236/jsea.2025.182005.

[3] X. Liu et al., "Research on Microservice Architecture: A Tertiary Study," SSRN Electronic Journal, 2022, doi: 10.2139/ssrn.4204345.

[4] L. M. Elnaghi and Moawad. Ramadan, "Microservices Ar vices Architectur chitecture: Evolution, Realizing Benefits, and olution, Realizing Benefits, and Addressing Challenges in the Modern Software Era -A systematic literature review," Future Computing and Informatics Journal, vol. 8, no. 2, pp. 12–18, 2023.

[5] J. A. Suthendra and M. A. I. Pakereng, "Implementation of Microservices Architecture on E-Commerce Web Service," ComTech: Computer, Mathematics and Engineering Applications, vol. 11, no. 2, pp. 89–95, Dec. 2020, doi: 10.21512/comtech.v11i2.6453.

[6] S. Saurav, "The Impact of Microservices Architecture on System Scalability," American Scientific Research Journal for Engineering, Technology, and Sciences, vol. 102, no. 1, pp. 140–148, Jun. 2025.

[7] S. Sharma, "The Impact of Microservices Architecture on System Scalability," American Scientific Research Journal for Engineering, Technology, and Sciences, vol. 102, no. 1, pp. 140–148, Jun. 2025.

[8] C. C. Ekechi, C. D. Okeke, and H. E. Adama, "Enhancing Agile Product Development with Scrum Methodologies: A Detailed Exploration of Implementation Practices and Benefits," Engineering Science & Technology Journal, vol. 5, no. 5, pp. 1542–1570, May 2024.

[9] C. J. Stettina and J. Garbajosa, Agile Processes in Software Engineering and Extreme Programming, vol. 475. Amsterdam: Springer Nature Switzerland, 2023. doi: 10.1007/978-3-031-33976-9.

[10] Adriana N Dugbartey and Olalekan Kehinde, "Optimizing project delivery through agile methodologies: Balancing speed, collaboration and stakeholder engagement," World Journal of Advanced Research and Reviews, vol. 25, no. 1, pp. 1237–1257, Jan. 2025, doi: 10.30574/wjarr.2025.25.1.0193.

[11] S. Di Meglio, V. Pontillo, and L. L. L. Starace, "REST in Pieces: RESTful Design Rule Violations in Student-Built Web Apps," Jul. 2025.

[12] A. Ehsan, M. A. M. E. Abuhaliqa, C. Catal, and D. Mishra, "RESTful API Testing Methodologies: Rationale, Challenges, and Solution Directions," Applied Sciences, vol. 12, no. 9, p. 4369, Apr. 2022, doi: 10.3390/app12094369.

[13] A. Saleh, R. Morabito, S. Tarkoma, S. Pirttikangas, and L. Lovén, "Towards Message Brokers for Generative AI: Survey, Challenges, and Opportunities," Feb. 2024.

[14] V. Rampérez, J. Soriano, D. Lizcano, and C. Miguel, "Automatic Evaluation and Comparison of Pub/Sub Systems Performance Improvements," Journal of Web Engineering, Apr. 2022, doi: 10.13052/jwe1540-9589.2144.

[15] S. R. Gothi, "Event-Driven Microservices Architecture for Data Center Orchestration," International Journal on Science and Technology, vol. 16, no. 2, Apr. 2025, doi: 10.71097/IJSAT.v16.i2.3113.