SOFTWARE REQUIREMENT SPECIFICATION DOCUMENT

Optimizing Deployment: Kubernetes for Seamless Scaling in a MERN Project with Microservices

Submitted To: MOHIT SIR

DR. SHAILENDRA PAL

DR. AMARJEET PRAJAPATI

TANYA VASHISHTHA (21803006)
VIVEK SHAURYA(21803013)
SNEHA (21803028)

Table of Contents

Table of Contents 2

1. Introduction 3

1.1 Documents Conventions 3

1.2 References. 3

2. Purpose 4

2.1 Project Objective 4

2.2 Project Scope 4

3. Overall Description 6

3.1 Product Features 6

3.2 User Classes and Characteristics... 8

3.3 Design and Implementation Constraints 11

3.4 Assumptions and Dependencies 13

4. External Interface Requirements 15

4.1 User Interfaces 15

4.2 Hardware Interfaces... 16

4.3 Software Interfaces 16

4.4 Communications Interfaces 16

5. Other Non Functional Requirements 17

5.1 Database 17

5.2 Tools/Technologies 17

5.3 Languages/FrameWorks 17

1. Introduction

1.1 Documents Conventions

The documentation for our web application, deployed on Kubernetes, adheres to clear and consistent conventions for enhanced clarity and usability. The document title succinctly encapsulates the purpose, incorporating both the application name and Kubernetes. Each version is meticulously tracked with version numbers and dates, distinguishing between draft and final releases. A comprehensive table of contents, featuring hyperlinks for seamless navigation, guides users through the document. The introduction provides an overview of the web app's architecture, emphasizing key components such as the frontend, backend, and databases. The documentation delves into Kubernetes cluster requirements, setup procedures, and installation steps. Configuration details for each component are highlighted, utilizing ConfigMaps and Secrets for managing settings. YAML manifests for Kubernetes resources, including Deployments, Services, ConfigMaps, and Secrets, are presented alongside explanations of their roles. Communication between different components, ingress configuration, load balancing strategies, and scaling considerations are thoroughly documented. Security practices, logging, monitoring, database considerations, and CI/CD pipelines are also covered. The document concludes with a summary, additional resources, and a change log, while contact information is provided for support inquiries. These conventions aim to foster a comprehensive understanding of the web application and its Kubernetes deployment, facilitating collaboration among stakeholders.

1.2 References

1. https://www.youtube.com/watch?v=7XDeI5fyj3w [Accessed: Jan-2024].
2. https://www.youtube.com/watch?v=BUAFNfdarBQ&t=1380s[Accessed: Feb-2024].
3. https://medium.com/aspnetrun/deploying-microservices-on-kubernetes-35296d369fdb [Accessed: Feb-2024].
4. https://loft.sh/blog/a-guide-to-using-kubernetes-for-microservices/ [Accessed: Feb-2024].

2. <u>Purpose</u>

2.1 Project Objective

The project's primary objective is to develop and deploy a resilient and scalable multi-tier web application using Kubernetes as the container orchestration platform. This involves containerizing each component of the web application with Docker and leveraging Kubernetes for efficient orchestration and deployment. The architecture follows a microservices approach, enhancing flexibility and maintainability. Emphasis is placed on implementing robust inter-service communication, ensuring secure and optimized data exchange.This involves configuring Horizontal Pod Autoscaling (HPA) to automatically adjust the number of running instances of each component, ensuring optimal performance during varying levels of user activity. Data persistence and storage are managed using Kubernetes Persistent Volumes and Claims. The project incorporates comprehensive monitoring and logging tools like Prometheus and Grafana, coupled with strong security measures at various levels, including container security and access controls through Kubernetes RBAC. A CI/CD pipeline is established for automated testing and deployment, streamlining the development process. Detailed documentation, adhering to established conventions, serves as a comprehensive guide for stakeholders. The project also considers cost optimization strategies, ensuring resource efficiency without compromising performance. Ultimately, the goal is to deliver a well-architected, scalable, and maintainable web application that capitalizes on Kubernetes' container orchestration capabilities, providing a solid foundation for future enhancements and updates.

2.2 Project Scope

The project scope encompasses the development and deployment of a robust multi-tier web application using Kubernetes as the primary container orchestration platform. This initiative includes the containerization of individual components with Docker, adopting a microservices architecture to enhance flexibility. The project will focus on seamless inter-service communication, data persistence using Kubernetes Persistent Volumes and Claims, and the implementation of robust monitoring and logging tools, such as Prometheus and Grafana, to ensure comprehensive observability. Security measures, including RBAC, will be enforced at multiple levels to safeguard the application. An integral part of the project scope involves the implementation of auto-scaling mechanisms, leveraging Horizontal Pod Autoscaling (HPA) to dynamically adjust the number of running instances based on demand. This ensures optimal resource utilization, cost-effectiveness, and responsiveness during varying levels of user activity. The project's comprehensive documentation will detail these auto-scaling strategies, providing stakeholders with valuable insights into how the system adapts to fluctuating workloads while maintaining scalability as a fundamental aspect of the application's architecture. The overall goal is to deliver a well-architected, scalable, and maintainable web application that maximizes the advantages of Kubernetes, with provisions for future enhancements and updates.

3. <u>Overall description</u>

3.1 Product Features

1. Containerization with Docker:

-Utilize Docker to containerize application components for consistent deployment across environments.

2. Microservices Architecture:

- Brief: Adopt a microservices approach to enhance flexibility and maintainability.

3. Kubernetes Orchestration:

- Brief: Leverage Kubernetes for efficient container orchestration, scaling, and management.

4. React for Frontend Development:

- Brief: Develop a responsive and dynamic user interface using React for an engaging frontend experience.

5. Node.js for Backend Services:

- Brief: Implement backend services using Node.js for high-performance, scalable, and event-driven server-side functionality.

6. MySQL and MongoDB for Database Management:

- Brief: Utilize MySQL and MongoDB as the relational/Non-relational database management system for efficient and structured data storage.

7. Auto-Scaling with HPA:

- Brief: Implement Horizontal Pod Autoscaling (HPA) to dynamically adjust resource allocation based on demand for optimal performance.

8. Inter-Service Communication:

- Brief: Establish efficient communication channels between frontend, backend, and database services within the Kubernetes cluster.

9. Data Persistence Strategies:

- Brief: Implement robust data persistence using Kubernetes Persistent Volumes and Claims for reliable and scalable storage.

10. Monitoring and Logging with Prometheus and Grafana:

- Brief: Integrate Prometheus and Grafana for comprehensive monitoring and logging to ensure system observability.

11. Security Measures with RBAC:

- Brief: Enforce security measures, including Role-Based Access Control (RBAC), to protect the application at various levels.

12. CI/CD Pipeline:

- Brief: Establish a Continuous Integration/Continuous Deployment (CI/CD) pipeline for automated testing, building, and deployment.

13. Comprehensive Documentation:

- Brief: Create detailed documentation adhering to conventions, serving as a comprehensive guide for developers, administrators, and stakeholders.

14. Cost Optimization Strategies:

- Brief: Consider and implement cost-effective strategies, particularly in cloud environments, to optimize resource utilization.

15. Future-Ready Architecture:

- Brief: Design the application's architecture with future enhancements and updates in mind for long-term maintainability and scalability.

3.2 User Classes and Characteristics for TheSocialEdge:

1. User Classes:

1. End Users:
   - Characteristics: Non-technical users interacting with the web application.
   - Needs: User-friendly interfaces, responsive design, and intuitive navigation.

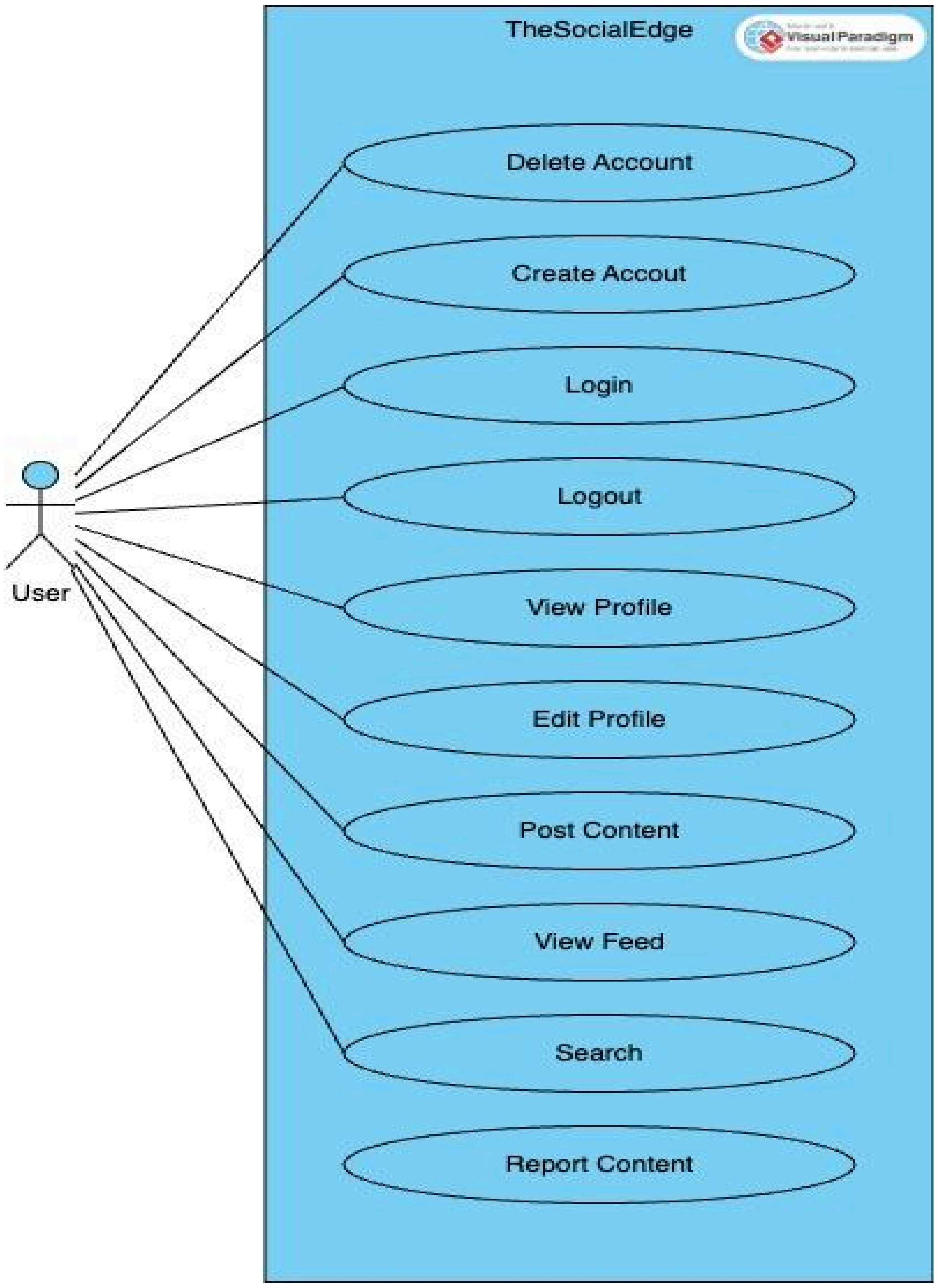
2. Developers:
   - Characteristics: Technical professionals contributing to application development.
   - Needs: Access to comprehensive documentation, clear API specifications, and integration guidelines.

3. System Administrators:
   - Characteristics: IT professionals responsible for system deployment and maintenance.
   - Needs: Efficient deployment scripts, system monitoring tools, and troubleshooting guides.

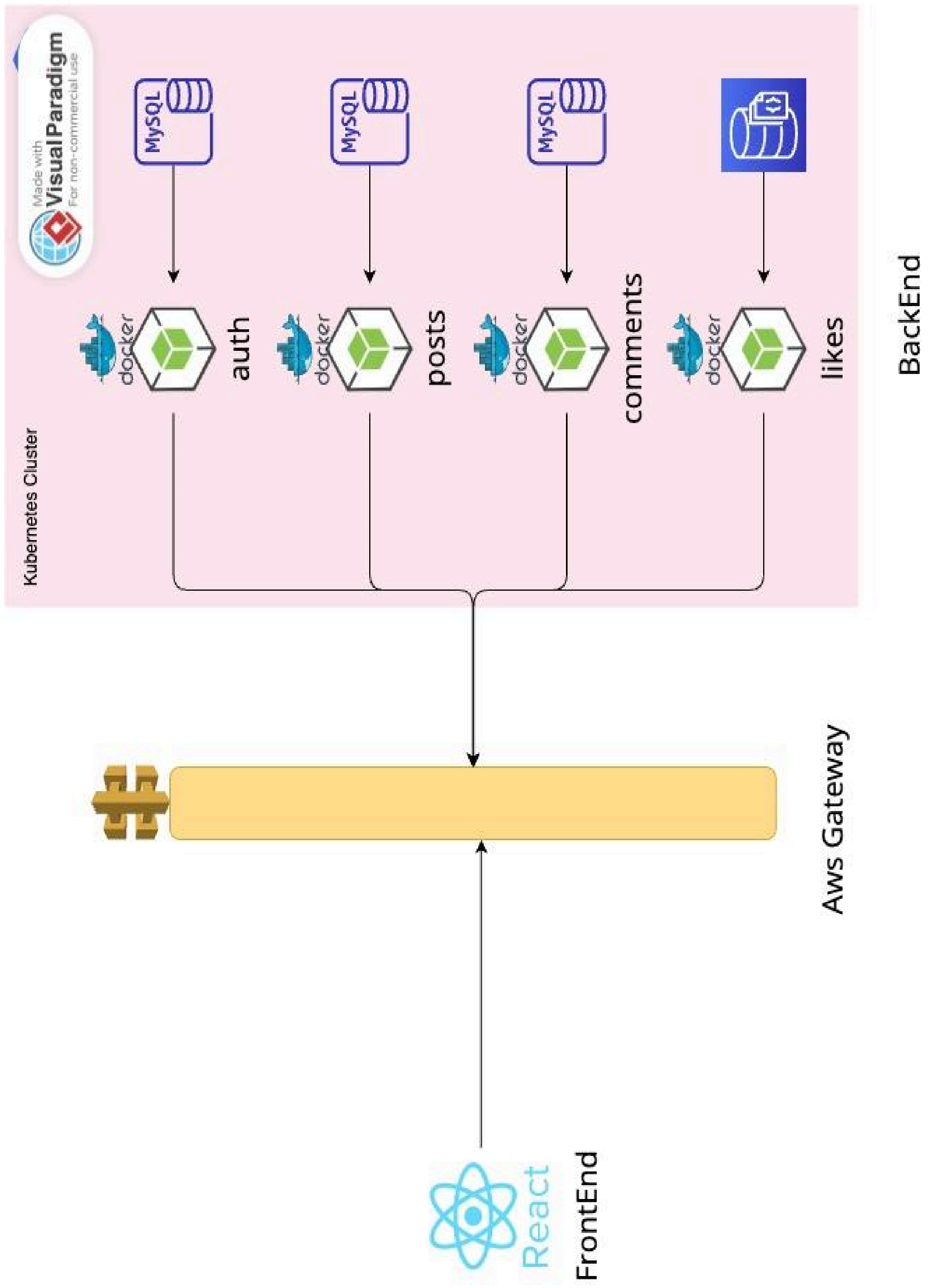
4. Database Administrators:
   - Characteristics: Professionals overseeing database management and optimization.
   - Needs: Database schema documentation, performance monitoring tools, and backup/recovery procedures.

These user classes and their characteristics help tailor the web application's design, features, and documentation to meet the diverse needs of various stakeholders involved in its lifecycle.

USE CASE DIAGRAM:



Project Design:



3.3 Design and Implementation Constraints for TheSocialEdge:

1. Technology Stack:

- Constraint: The project must adhere to the chosen technology stack, including React for the frontend, Node.js for the backend, and MySQL and MongoDB for the database, limiting flexibility in choosing alternative technologies.

2. Kubernetes Dependency:

- Constraint: The application's deployment and scaling are dependent on Kubernetes, which may limit compatibility with alternative container orchestration platforms.

3. Legacy System Integration:

- Constraint: Integration with existing legacy systems may pose challenges due to differences in technology, data structures, or communication protocols.

4. Regulatory Compliance:

- Constraint: Adherence to specific industry regulations or compliance standards may impose limitations on certain design and implementation choices.

5. Security Policies:

- Constraint: Stringent security policies, including encryption standards and access controls, must be followed, potentially restricting certain implementation approaches.

6. Budgetary Constraints:

- Constraint: Project resources, including hardware, software licenses, and cloud services, are subject to budget limitations, influencing the scalability and feature set.

7. Scalability Requirements:

- Constraint: Certain architectural decisions may impose limitations on the system's scalability, requiring careful consideration to meet anticipated growth.

8. Third-Party Service Dependencies:

- Constraint: Dependencies on external services or APIs introduce potential vulnerabilities, downtimes, or changes in service behavior beyond the project's control.

9. Limited Development Timeframe:

- Constraint: A fixed development timeframe may limit the depth of features, testing, and optimization achievable within the given schedule.

10. Geographical Distribution:

- Constraint: The need for data synchronization across geographically distributed regions may introduce latency and connectivity challenges that impact the application's performance.

11. Skillset of Development Team:

- Constraint: The skillset and expertise of the development team may influence the choice of technologies and the complexity of certain features.

12. Client-Specific Requirements:

- Constraint: Client-specific requirements or preferences may limit the freedom to choose certain technologies, designs, or user experience patterns.

13. Compliance with Industry Standards:

- Constraint: The need to comply with industry-specific standards, such as healthcare or finance regulations, may impose restrictions on certain design and implementation choices.

14. User Accessibility:

- Constraint: Accessibility standards must be met, imposing limitations on certain UI/UX design elements and interactions.

Understanding and managing these constraints is crucial for making informed decisions during the design and implementation phases, ensuring that the final web application aligns with project goals, stakeholder expectations, and regulatory requirements.

3.4 Assumptions and Dependencies for TheSocialEdge:

1. Kubernetes Cluster:

- Dependency: Availability and proper functioning of a Kubernetes cluster for deploying and managing containerized applications.

2. Development Environments:

- Dependency: Availability of stable development and testing environments for thorough application testing.

3. Internet Connectivity:

- Dependency: Continuous and reliable internet connectivity for accessing external dependencies and updating libraries.

4. Stakeholder Collaboration:

- Dependency: Active involvement and collaboration from stakeholders for feedback and clarification on project requirements.

5. Team Expertise:

- Dependency: Development team possessing expertise in React, Node.js, Kubernetes, Docker, and MySQL.

6. Server Resources:

- Dependency: Adequate server resources, including compute, memory, and storage, to support the web application.

7. Backup and Recovery:

- Dependency: Implementation and adherence to regular backup and recovery processes for data protection.

8. Security Compliance:

- Dependency: Adherence to organizational security policies to ensure information confidentiality and integrity.

9. External Dependencies Resolution:
   - Dependency: Timely resolution of issues related to third-party services, libraries, or dependencies to prevent delays.

10. Documentation Standards:
    - Dependency: Adherence to clear documentation standards for knowledge transfer and future maintenance.

11. Client Availability for UAT:
    - Dependency: Client availability for User Acceptance Testing (UAT) to validate application expectations.

12. Budget Approval:
    - Dependency: Approval and allocation of the project budget for necessary tools, services, and resources.

13. Browser Compatibility:
    - Dependency: Ensuring web application compatibility with major browsers and specific browser features.

14. Legal and Regulatory Compliance:
    - Dependency: Compliance with legal and regulatory requirements, including data protection laws.

15. Third-Party Services Support:
    - Dependency: Continued availability and support from third-party services or APIs integrated into the application.

Identifying and managing these dependencies is crucial for planning and executing the project effectively, mitigating potential risks, and ensuring successful project outcomes. Regular communication and collaboration with stakeholders are essential to address any changes or challenges related to these dependencies throughout the project lifecycle.

4. External Interface Requirements for TheSocialEdge:

4.1 User Interfaces:

1. Login/Sign Up Page:
   - Secure authentication for users to access TheSocialEdge.

2. User Registration:
   - Simple account creation with necessary details.
   - Quick registration option using existing social media accounts.

3. Visual Content Upload:
   - Users can upload photos and videos to share with their network, fostering a visually engaging experience.

4. Notification System:
   - Real-time notifications for activities such as likes, comments, and new follower requests, enhancing user engagement.

5. Direct Messaging:
   - Private messaging feature for one-on-one communication between users.

6. Explore Page:
   - A curated feed or explore page showcasing popular and trending content for users to discover new connections and interests.

7. Hashtags and Trending Topics:

- Support for hashtags and trending topics to facilitate content discovery and community engagement.

8. Profile Customization:

- Users can personalize their profiles with profile pictures, bios, and other customizable elements.

4.2 Hardware Interfaces:

- Internet Connection:

- A stable internet connection for seamless access to TheSocialEdge.

- Camera:

- Necessary for users to capture and upload photos and videos.

4.3 Software Interfaces:

- Development Environment:

- Utilization of appropriate web development tools and languages for a responsive and visually appealing social media site.

- Database System:

- Robust backend database system to manage user profiles, posts, comments, and other pertinent data.

4.4 Communication Interfaces:

- Communication Standards:

- Adherence to industry-standard protocols (such as HTTPS) for secure data transmission.

- Security Measures:

- ts align with typical features found in social media p Implementation of strong security measures to safeguard user data, including encryption and authentication protocols.

5. Other Non-Functional Requirements for TheSocialEdge:

5.1 Database:

- MongoDB,MySql

5.2 Tools/Technologies:

- VS Code, Github, Material UI,Docker,kubernetes

5.3 Languages/FrameWorks:

- HTML, CSS, JavaScript, React JS, Node JS

5.4 Usability:

- Easy to use

5.5 Maintenance:

- Maintenance can be used easily