TEAM NAME

Code Riders (Team 6)

High Level Document

Team Members

1. Niket Suresh Padole
2. Ajinkya Murlidhar Ingle
3. Ayush Priyadarshi
4. Subhrajit Dash
5. Chamarthi Rohith Kumar Raju

Project Overview

ShareMyTrip is a comprehensive ride-sharing application designed to facilitate seamless and efficient transportation services for users. The application employs microservices architecture to ensure scalability, reliability, and maintainability across various functionalities including user management, ride matching, fare calculation, trip management, and payment processing.

Project Components:

1. **User Management Service:**
    - Framework: Spring Boot
    - Technologies: Spring Security, JWT
    - Database: MySQL
    - Containerization: Docker
    - Orchestration: Kubernetes
2. **Ride Matching Service:**
    - Framework: Spring Boot, Spring Cloud
    - Data Store: MySQL
    - Containerization: Docker
    - Orchestration: Kubernetes
3. **Fare Calculation Service:**
    - Framework: Spring Boot, Spring Cloud
    - Cache: Redis
    - Database: MySQL
    - Containerization: Docker
    - Orchestration: Kubernetes
4. **Trip Management Service:**
    - Framework: Spring Boot, Spring Data JPA
    - Database: MySQL
    - Containerization: Docker
    - Orchestration: Kubernetes
5. **Payment Service:**
    - Framework: Spring Boot, Spring Cloud
    - Integration: REST API integration with payment gateways
    - Containerization: Docker
    - Orchestration: Kubernetes
6. **Frontend:**
    - HTML , Tailwind CSS
    - Libraries: React

7. **Deployment:**
    - Cloud Provider: AWS
    - Services: EKS (Elastic Kubernetes Service), RDS (Relational Database Service), S3 (Simple Storage Service)

High-Level View: ShareMyTrip - Ride-Sharing Application

Overview: ShareMyTrip is an innovative ride-sharing application designed to connect riders with drivers seamlessly. It employs microservices architecture, ensuring flexibility, scalability, and resilience in handling various functionalities. The application encompasses user management, ride matching, fare calculation, trip management, and payment processing to deliver a comprehensive transportation solution.

User Experience:

- **Registration and Authentication:** Users can sign up and log in securely, with authentication handled using Spring Security and JWT tokens.
- **Ride Booking:** Riders can easily request rides through the intuitive frontend interface, which utilizes Node.js and either React or Angular for dynamic and responsive user interactions.
- **Driver Matching:** The application intelligently matches riders with available drivers based on location, preferences, and other relevant factors, facilitated by the Ride Matching Service.
- **Trip Management:** Riders can manage their trips seamlessly, including tracking the ride's progress, viewing trip history, and providing feedback.
- **Payment Processing:** Secure payment processing is integrated directly into the application, allowing riders to pay for their trips effortlessly.

Backend Architecture:

- **Microservices Approach:** The application is divided into multiple independent services, each responsible for a specific domain, such as user management, ride matching, fare calculation, trip management, and payment processing.
- **Spring Boot and Spring Cloud:** These frameworks provide robust support for building and deploying microservices, ensuring reliability, scalability, and easy integration between services.
- **Data Storage:** Different services utilize various databases such as MySQL and MongoDB, chosen based on the specific requirements of each service.
- **Containerization and Orchestration:** Docker containers are used to package each microservice, providing consistency across different environments, while Kubernetes orchestrates the deployment and scaling of containers, ensuring high availability and resource efficiency.

Deployment:

- **Cloud Infrastructure:** The application is deployed on Amazon Web Services (AWS), leveraging services such as Elastic Kubernetes Service (EKS) for container orchestration, Relational Database Service (RDS) for database management, and Simple Storage Service (S3) for storing static assets.
- **Continuous Integration/Continuous Deployment (CI/CD):** DevOps practices are implemented to automate the build, test, and deployment processes, ensuring rapid and reliable delivery of updates and new features.

Security:

- **Authentication and Authorization:** Spring Security and JWT tokens are used to authenticate users securely and manage access control.
- **Data Protection:** Measures are implemented to safeguard sensitive user data, including encryption and compliance with relevant data protection regulations.

Scalability and Performance:

- **Horizontal Scalability:** The microservices architecture allows individual services to scale independently based on demand, ensuring optimal performance even during peak usage periods.
- **Caching:** Redis is utilized for caching frequently accessed data, reducing latency and improving overall system performance.

System Requirements for ShareMyTrip - Ride-Sharing Application

1. Hardware Requirements:

- **Server Infrastructure:**
  - Multiple servers for hosting microservices (recommended minimum: 3 servers for redundancy and load balancing).
  - Adequate CPU and memory resources to handle concurrent requests and data processing.
- **Storage:**
  - Sufficient storage space for database instances and file storage.
  - SSD storage recommended for improved performance.

2. Software Requirements:

- **Operating System:**
  - Linux-based operating systems (e.g., Ubuntu Server, CentOS) preferred for server environments.
- **Containerization Platform:**
  - Docker Engine: Required for containerizing microservices and ensuring consistency across different environments.
- **Container Orchestration:**
  - Kubernetes: Essential for automating deployment, scaling, and management of Docker containers across clusters of hosts.
- **Database Management System (DBMS):**
  - MySQL: Used for storing user data, ride information, fare calculations, etc.
  - MongoDB: Utilized for trip management service to store flexible and schema-less data.
- **Backend Frameworks:**
  - Spring Boot: Used for developing microservices due to its ease of use, robustness, and integration capabilities.
  - Spring Cloud: Provides tools and libraries for building distributed systems and microservices architectures.
- **Frontend Framework:**
  - React Native: Required for running the JavaScript runtime environment.
  - React : Frontend frameworks for building dynamic and interactive user interfaces.
- **Payment Gateway Integration:**
  - REST API integration with payment gateways such as Stripe, PayPal, or others as per business requirements.

- **Development Tools:**
  - Integrated Development Environment (IDE) such as IntelliJ IDEA or STS for backend development.
  - Text editors (e.g., Visual Studio Code) for frontend development.
  - Version Control System (e.g., Git) for managing source code.
  - CI/CD tools (e.g., Jenkins, GitLab CI/CD) for automating build, test, and deployment processes.

3. Networking Requirements:

- **Internet Connectivity:**
  - High-speed internet connection for accessing cloud services and external APIs.
- **Security:**
  - Secure Socket Layer (SSL) certificates for encrypting data transmission over the network.
  - Firewall and network security measures to protect against unauthorized access and attacks.

4. Cloud Services:

- **Amazon Web Services (AWS):**
  - Elastic Kubernetes Service (EKS): For Kubernetes cluster management.
  - Relational Database Service (RDS): Managed database service for MySQL.
  - Simple Storage Service (S3): For storing static assets such as images, CSS files, and JavaScript libraries.
  - Virtual Private Cloud (VPC): Provides isolated network environments for enhanced security.
  - CloudWatch: Monitoring and logging service for tracking application performance and health.

5. Miscellaneous:

- **Documentation and Training:**
  - Comprehensive documentation for system architecture, installation, configuration, and usage guidelines.
  - Training materials and user guides for administrators and end-users.

6. Compliance and Regulations:

- Ensure compliance with relevant data protection regulations such as GDPR, CCPA, etc., depending on the target user jurisdictions.

7. Scalability Considerations:

- Design the system to scale horizontally by adding more instances of microservices and databases as the user base grows.
- Utilize Kubernetes auto-scaling capabilities to automatically adjust the number of containers based on resource usage.

8. Disaster Recovery and Backup:

- Implement regular backups of data and configurations to prevent data loss in case of system failures or disasters.
- Set up disaster recovery mechanisms to ensure business continuity and minimize downtime.

SECURITY

The Application-Level Security Architecture for ShareMyTrip focuses on ensuring the confidentiality, integrity, and availability of data, as well as protecting against unauthorized access and attacks. Here's an outline of the security measures at various layers of the application:

1. Authentication and Authorization:

- **User Authentication:** Utilize Spring Security and JWT tokens for secure authentication of users during login and registration processes.
- **Authorization:** Implement role-based access control (RBAC) to restrict access to certain functionalities based on user roles (e.g., rider, driver, admin).
- **Secure Password Storage:** Store passwords securely using cryptographic hashing algorithms (e.g., bcrypt) to prevent unauthorized access even if the database is compromised.

2. Data Protection:

- **Encryption:** Implement end-to-end encryption for sensitive data transmission over the network using SSL/TLS protocols.
- **Data Masking:** Apply data masking techniques to conceal sensitive information (e.g., credit card numbers, personal identification) in logs and user interfaces to prevent unauthorized access.
- **Data Encryption:** Encrypt sensitive data at rest using encryption mechanisms provided by the chosen database management systems (e.g., MySQL encryption features).

3. Secure Communication:

- **HTTPS:** Ensure all communication between clients and servers is encrypted using HTTPS to prevent eavesdropping and tampering of data.
- **API Security:** Implement secure API endpoints with proper authentication and authorization mechanisms to prevent unauthorized access and misuse.

4. Input Validation and Sanitization:

- **Input Validation:** Validate and sanitize user inputs to prevent injection attacks such as SQL injection, XSS (Cross-Site Scripting), and CSRF (Cross-Site Request Forgery).

- **Parameterized Queries:** Utilize parameterized queries or ORM (Object-Relational Mapping) frameworks to prevent SQL injection attacks when interacting with databases.

5. Secure Session Management:

- **Session Tokens:** Use secure session tokens with short expiration times to manage user sessions and prevent session hijacking.
- **Session Persistence:** Store session data securely and ensure proper session expiration and invalidation mechanisms are in place.

6. Secure Coding Practices:

- **Code Reviews:** Conduct regular code reviews to identify and mitigate security vulnerabilities in the application codebase.
- **Secure Coding Guidelines:** Follow secure coding practices and guidelines to prevent common security issues such as buffer overflows, insecure deserialization, and insecure file operations.

7. Security Logging and Monitoring:

- **Logging:** Implement comprehensive logging mechanisms to record security-related events and actions performed within the application for auditing and investigation purposes.
- **Monitoring:** Utilize security monitoring tools and services to monitor system activity, detect anomalies, and respond to security incidents in real-time.

8. Third-Party Integrations:

- **Security Assessments:** Conduct security assessments of third-party libraries, frameworks, and APIs before integrating them into the application to ensure they meet security standards and do not introduce vulnerabilities.

9. Compliance and Regulations:

- **Data Protection Regulations:** Ensure compliance with relevant data protection regulations such as GDPR, CCPA, HIPAA, etc., depending on the geographical location of users and data processing activities.

10. Regular Security Audits and Penetration Testing:

- **Audits:** Conduct regular security audits and assessments to identify potential vulnerabilities and weaknesses in the application infrastructure and codebase.
- **Penetration Testing:** Perform periodic penetration testing to simulate real-world attacks and evaluate the effectiveness of security controls and defenses.

Deployment

The deployment process involves deploying the loan application to production environments using a Continuous Integration/Continuous Deployment (CI/CD) pipeline. Key steps will include:

Continuous Integration: Automated build and testing of code changes will be performed whenever new code is pushed to the version control repository.

Continuous Deployment: Automated deployment of approved code changes to production environments will be triggered upon successful completion of the CI pipeline.

Deployment Environments: Multiple deployment environments, including development, staging, and production, will be used to test changes in controlled environments before releasing them to production.

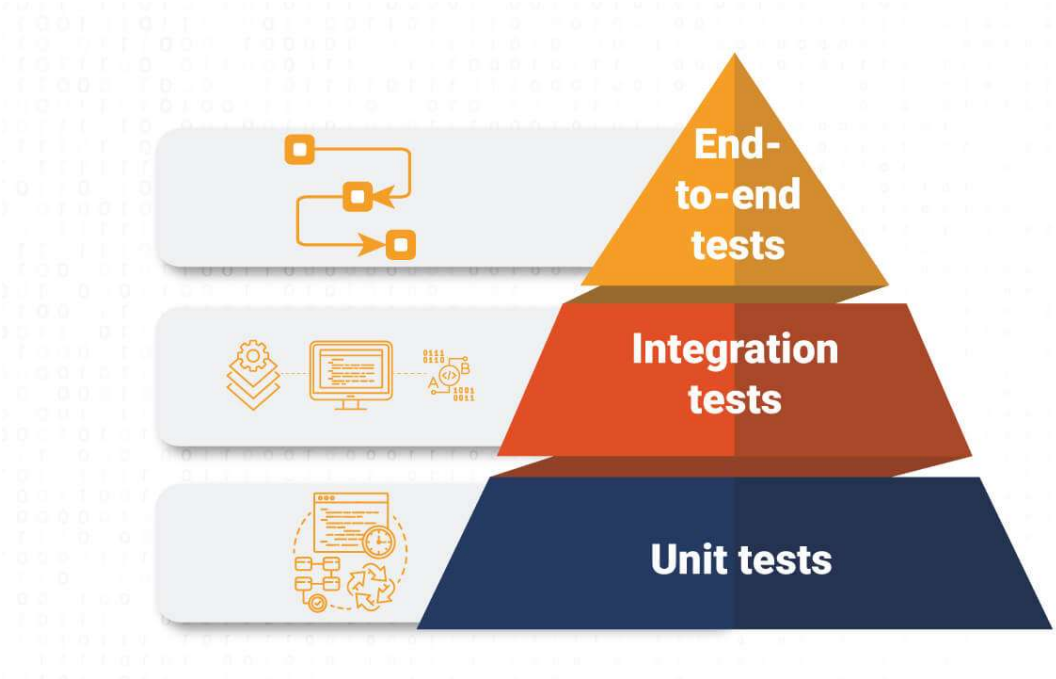
Rollback Mechanism: In case of deployment failures or issues, rollback procedures will in place to revert to the previous stable version of the application.

Deployment may occur on cloud infrastructure (e.g., AWS, Azure) or on-premises servers.

Version control

Git: Git is a free, open-source version control system that allows software teams to track changes to computer files and collaborate on projects of any size.

Testing



Testing is an essential aspect of ensuring the reliability, performance, and security of the loan application. Various testing methodologies will be employed throughout the development lifecycle, including:

Unit Testing: Individual components and functions of the application will be tested in isolation to verify their correctness.

Integration Testing: Interaction between different components/modules of the application will be tested to ensure they work together as expected.

End-to-End Testing: Full end-to-end scenarios will be tested to validate the functionality of the entire application from user interaction to backend processing.

Performance Testing: Load testing and stress testing will be performed to evaluate the application's performance under different levels of user load and stress conditions.

Security Testing: Vulnerability assessments, penetration testing, and code reviews will be conducted to identify and address security vulnerabilities in the application.

Automated testing frameworks and tools such as Selenium, Jest, Postman, and OWASP ZAP could be used to streamline the testing process and ensure comprehensive test coverage. Test cases and test plans will be documented to guide testing efforts and track test results. Regular regression testing will be performed to validate changes and ensure the stability of the application across different environments.