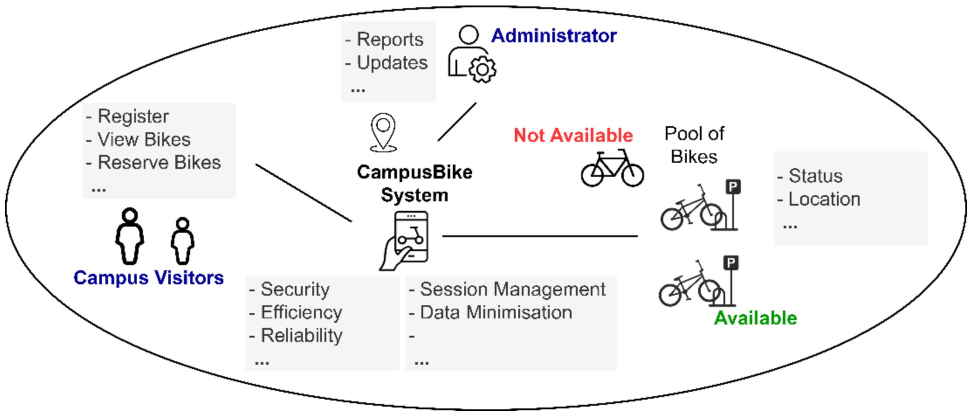
Case Study on Architecting the CampusBike Application

(Nov 2022)

Contents

1. Business Case and Specifications for the CampusBike

With rising traffic congestion and attempt towards minimizing the carbon footprint of vehicles, trams, and public transport, the city administration have introduced CityBike service where riders can opt to use bikes available via pay-per-use renting for enhanced mobility and eco-friendly transportation in and around the city. City administration needs an software application CityBike, available on web and mobile devices that can offer the potential bikers to ustilise the bikes. As the first step towards implementation, CityBike app must be designed adhering to the following specifications (S1 to S13).



Specifications for the CampusBike
S1: Biker must be able to register with the app by providing their name, email address, phone number, and selecting a password, and provide Zip code of residential address (an optional information). The app must ensure that user information is secured with an end-to-end encryption of the information between user device and app database. S2: The app must sent a verification code via an SMS or email that is valid for 30 minutes. If the code is not entered within 30 seconds it is expired **S3**: The biker must be able to login to the app via their email and password or by using their biometric thumb-impression. The login module must flush out user thumb impression as soon as the user has been able to login. **S4**: Biker must be logged out of the app by clicking on the logout button or after an inactivity of 15 minutes to avoid unnecessary (idle) users on the app in a session. **S5**: Biker can view all the available bikes in close proximity defined by the him/her such as specifying that they only want to see the bikes that are available in a proximity of X meters (lets say x = 100). **S6**: App must help the biker to navigate to the available bike that they have selected. Upon reaching the bike, the biker should scan the code on bike scanner or by inputting the four digit code to register the bike for M minutes. **S7**: As soon as the bike is registered it must be removed from the list of available bikes. **S8**: The biker must be able to pay via their bank account or by their debit/credit card. Currently the cash payments are not supported. **S9**: The app must notify the biker X minutes prior to the expiry of their rental time. Any overtime rides are subject to additional charges. **S10**: The biker must be able to rent the bike for specific minutes, a day or for a week at maximum. **S11**: The app allows the biker to view and download their bike usage history upto last 3 months in the form. **S12**: The bike administrator is able to view all the bikes in the city as available or under utilization. **S13**: The administrator must be able to do a hourly and daily usage of the bikes.

Functional Aspects and Quality Attributes	
Functional Aspects	**Quality Attributes**
User Registration	Usability
Bike Reservation	Performance
Bike Unlocking	Security
Payment Gateway	Compatibility
Map Integration	Availability
Bike Status	Scalability
Feedback System	Accessibility
Admin Panel	Reliability
User Profile	Localization
Notification System	Integration
Customer Support	

Constraints

EU Standards:

1. The software should comply with the General Data Protection Regulation (GDPR) and the Privacy Directive.
2. The software should comply with the relevant security and data protection standards, such as ISO 27001.

Constraints for the Campus-Bike Software:

1. The software should be designed to minimize the carbon footprint and promote sustainable mobility.
2. The software should be user-friendly and easy to use for all potential riders.
3. The software should be designed to ensure the security and privacy of user data, including payment information.
4. The software should be compatible with a variety of third-party payment platforms to allow users to pay for the bike rental fee seamlessly.
5. The software should be capable of tracking and displaying the location of available bikes within a 1000-meter radius and unlocking them using a QR code.
6. The software should be accessible through both web and mobile devices to maximize the availability of the service.
7. The software should be designed to comply with local regulations and standards, including EU or China standards, depending on the location of the university.

Tool and Technologies (Infrastructure, design, implementation, testing)

Infrastructure:

1. Container orchestration platform: Kubernetes or Docker Swarm
2. Service registry and discovery: Consul or Eureka
3. API Gateway: Kong or Istio
4. Message broker: Apache Kafka or RabbitMQ
5. Distributed tracing: Jaeger or Zipkin

Design:

1. Domain-driven design (DDD) approach for defining the boundaries of each microservice
2. RESTful API design principles
3. Event-driven architecture for asynchronous communication between microservices
4. CQRS (Command Query Responsibility Segregation) pattern for separating read and write operations
5. Use of API contracts to define interactions between microservices
6. Polyglot persistence to use the best database technology for each microservice's needs

Implementation:

1. Programming languages: Java, Node.js, Python, or Golang
2. Frameworks: Spring Boot, Express.js, Flask, or Gin
3. Containerization: Docker
4. Build and deployment tools: Jenkins, Travis CI, or CircleCI
5. Configuration management: Ansible or Chef
6. Continuous Integration/Continuous Deployment (CI/CD) pipeline

Testing:

1. Unit testing frameworks: JUnit, Mocha, or pytest
2. Integration testing frameworks: REST Assured, SuperTest, or Karate
3. Contract testing tools: Pact or Spring Cloud Contract
4. Load testing tools: Apache JMeter, Gatling, or Locust
5. Chaos engineering tools: Chaos Monkey or Gremlin
6. Code coverage analysis: SonarQube or Codecov