# Architecture Document: Tool Rental Application

Contents

# 1. Introduction:

The Tool Rental Application is designed to facilitate the rental of tools to customers in a store environment. This document elaborates on the architectural design of the application, covering its structure, components, interactions, and key considerations.

# 2. Architectural Styles and Patterns:

## 2.1 Layered Architecture:

- The application follows a layered architecture, comprising presentation, business logic, and data access layers.
- Presentation Layer: Handles user interactions and displays rental information.
- Business Logic Layer: Contains the core logic for rental calculations, tool management, and rental agreement generation.
- Data Access Layer: Manages data persistence and retrieval, interacting with the underlying database or data repository.

## 2.2 Service-Oriented Architecture (SOA):

- The application adopts a service-oriented architecture, where functionalities are encapsulated as services that communicate through well-defined interfaces.
- RentalService: Provides methods for renting tools, calculating rental charges, and generating rental agreements.
- ToolService: Manages tools data, including CRUD operations for tools and retrieval of tool information.

# 3. Key Components:

## 3.1 Rental Calculator:

- Responsible for calculating rental charges based on input parameters such as tool type, rental duration, and discounts.
- Implements business rules for special cases like weekend rentals, holiday rentals, and discounts.
- Exposes methods for rental calculation and agreement generation.

## 3.2 Rental Agreement Generator:

- Constructs rental agreements with detailed information including tool code, type, brand, rental days, charges, discounts, and final charges.
- Formats rental agreement text for display to customers.

### 3.3 Tool Repository:

- Manages the storage and retrieval of tools data, including tool code, type, and brand.
- Supports operations for adding, updating, and deleting tools.

## 4. Design:

### 4.1 Class Diagram:

- **RentalAgreement**: Represents a rental agreement with details such as tool code, tool type, brand, rental days, charges, discounts, and final charges.
- **RentalCalculator**: Calculates rental charges based on input data such as tool code, rental day count, discount, and checkout date.
- **ToolRepository**: Manages the tools data including tool code, type, and brand.
- **HolidayService**: Handles special holiday-related logic such as determining if a given date is a holiday and adjusting rental charges accordingly.

### 4.2 Sequence Diagram:

**Rent Tool Scenario:**

- Customer selects a tool to rent.
- Customer provides rental duration, discount (if any), and checkout date.
- RentalCalculator calculates rental charges.
- RentalAgreement is generated with detailed information.
- RentalAgreement is displayed to the customer.

## 5. Design Considerations:

### 5.1 Scalability:

- The architecture is designed to be scalable, accommodating increasing numbers of customers and tools.
- Horizontal scalability can be achieved by deploying multiple instances of the application behind a load balancer.

### 5.2 Extensibility:

- The design allows for easy extension with new tool types, rental policies, and business rules.
- The rental calculation logic is designed to be configurable, enabling customization based on specific business requirements.

### 5.3 Performance:

- Performance considerations are taken into account during the design to ensure efficient rental calculation and data retrieval.
- Caching mechanisms can be implemented to improve performance by reducing database queries for frequently accessed data.

## 6. Implementation:

### 6.1 Tools Data:

- Tools data including tool code, type, and brand is stored in a database or a data repository.

### 6.2 Rental Calculator:

- Calculates rental charges based on tool type, rental duration, and discounts.
- Handles special cases such as weekend rentals, holiday rentals, and discounts.

### 6.3 Rental Agreement:

- Contains detailed information about the rental agreement including tool code, type, brand, rental days, charges, discounts, and final charges.

### 6.4 Exception Handling:

- Proper error handling and exception reporting are implemented to handle invalid inputs and unexpected errors gracefully.

## 7. Testing:

### 7.1 Unit Testing:

- JUnit tests are implemented to verify the correctness of the rental calculation logic, handling various input scenarios and edge cases.

### 7.2 Integration Testing:

- Integration tests ensure that the components of the application work together correctly, including database interaction and external services (if any).

## 8. Deployment Considerations:

### 8.1 Deployment Environment:

- The application can be deployed on-premises or in the cloud, depending on organizational requirements and infrastructure preferences.
- Docker containers or virtual machines can be used for containerization and deployment.

### 8.2 High Availability:

- High availability considerations include deploying the application across multiple availability zones or regions to ensure resilience against infrastructure failures.

## 9. Testing Strategy:

### 9.1 Unit Testing:

- Unit tests are implemented for individual components to validate their functionality in isolation.
- Mocking frameworks can be used to simulate dependencies and isolate components for testing.

### 9.2 Integration Testing:

- Integration tests verify the interaction between components and ensure that they work together as expected.
- Tests cover scenarios such as database interactions, service communication, and end-to-end functionality.

## 10. Conclusion:

The architectural design of the Tool Rental Application emphasizes modularity, scalability, and extensibility to meet the needs of a dynamic tool rental environment. By adhering to architectural best practices and design principles, the application achieves robustness, performance, and maintainability, providing a seamless experience for customers and administrators alike.