Parking Management System

Project Documentation

Team Members

- Yash Chavan
- Gaurav Salunke
- Aaditi Bais

Table of Contents

Project Overview

Objective

The Parking Management System is designed to automate and streamline parking operations for commercial and residential complexes. The system provides efficient vehicle entry/exit management, parking slot allocation, billing, and administrative controls.

Problem Statement

Traditional parking systems face challenges such as:

- Manual tracking of vehicles and parking slots
- Time-consuming entry/exit processes

- Difficulty in managing parking fees and billing
- Lack of real-time parking availability information
- Security concerns with unauthorized access

Solution

Our Java-based Parking Management System addresses these issues by providing:

- Automated vehicle registration and tracking
- Real-time parking slot availability
- Integrated billing and payment processing
- User-friendly interface for both customers and administrators
- Comprehensive reporting and analytics

System Requirements

Hardware Requirements

- **Processor**: Intel Core i3 or higher
- **RAM**: Minimum 4GB, Recommended 8GB
- **Storage**: 500MB free disk space
- **Network**: Internet connection for database connectivity

Software Requirements

- **Operating System**: Windows 10/11, macOS, or Linux
- **Java**: JDK 8 or higher
- **Database**: MySQL 8.0 or higher
- **IDE**: Eclipse, IntelliJ IDEA, or NetBeans
- **Web Browser**: Chrome, Firefox, or Safari (for web interface)

Technology Stack

Programming Language

- **Java**: Core application development using OOP principles

Database Technology

- **JDBC**: Java Database Connectivity for database operations
- **MySQL**: Relational database management system

Development Tools

- **IDE**: Integrated Development Environment for coding
- **MySQL Workbench**: Database design and management
- **Git**: Version control system

Frameworks & Libraries

- **Swing/AWT**: GUI development
- **MySQL Connector/J**: JDBC driver for MySQL

System Architecture

Architecture Pattern

The system follows a **3-Tier Architecture**:

1. **Presentation Layer (GUI)**
   - User interfaces for customers and administrators
   - Input validation and user interaction handling
2. **Business Logic Layer (Application Layer)**
   - Core business rules and processing
   - Data validation and transformation
   - Service classes for different functionalities
3. **Data Access Layer (Database)**
   - JDBC connections and database operations
   - SQL queries and stored procedures
   - Data persistence and retrieval

Design Patterns Used

- **Singleton Pattern**: Database connection management
- **DAO Pattern**: Data Access Object for database operations
- **MVC Pattern**: Model-View-Controller for UI separation
- **Factory Pattern**: Object creation and management

Database Design

Entity Relationship Diagram

The database consists of the following main entities:

Tables Structure

1. Users Table

```sql
CREATE TABLE users (
    user_id INT PRIMARY KEY AUTO_INCREMENT,
    username VARCHAR(50) UNIQUE NOT NULL,
    password VARCHAR(255) NOT NULL,
    full_name VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    phone VARCHAR(15),
    user_type ENUM('ADMIN', 'CUSTOMER') NOT NULL,
    created_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    is_active BOOLEAN DEFAULT TRUE
);
```

2. Vehicles Table

```sql
CREATE TABLE vehicles (
    vehicle_id INT PRIMARY KEY AUTO_INCREMENT,
    license_plate VARCHAR(20) UNIQUE NOT NULL,
    owner_id INT,
    vehicle_type ENUM('CAR', 'MOTORCYCLE', 'TRUCK') NOT NULL,
    brand VARCHAR(50),
    model VARCHAR(50),
    color VARCHAR(30),
    registered_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (owner_id) REFERENCES users(user_id)
);
```

3. Parking_Slots Table

```sql
```

```sql
CREATE TABLE parking_slots (
    slot_id INT PRIMARY KEY AUTO_INCREMENT,
    slot_number VARCHAR(10) UNIQUE NOT NULL,
    floor_number INT NOT NULL,
    slot_type ENUM('REGULAR', 'DISABLED', 'VIP') DEFAULT 'REGULAR',
    is_occupied BOOLEAN DEFAULT FALSE,
    hourly_rate DECIMAL(10,2) NOT NULL,
    status ENUM('AVAILABLE', 'OCCUPIED', 'MAINTENANCE') DEFAULT 'AVAILABLE'
);
```

4. Parking_Sessions Table

```sql
CREATE TABLE parking_sessions (
    session_id INT PRIMARY KEY AUTO_INCREMENT,
    vehicle_id INT NOT NULL,
    slot_id INT NOT NULL,
    entry_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    exit_time TIMESTAMP NULL,
    duration_hours DECIMAL(5,2),
    total_amount DECIMAL(10,2),
    payment_status ENUM('PENDING', 'PAID', 'CANCELLED') DEFAULT 'PENDING',
    session_status ENUM('ACTIVE', 'COMPLETED') DEFAULT 'ACTIVE',
    FOREIGN KEY (vehicle_id) REFERENCES vehicles(vehicle_id),
    FOREIGN KEY (slot_id) REFERENCES parking_slots(slot_id)
);
```

5. Payments Table

```sql
CREATE TABLE payments (
    payment_id INT PRIMARY KEY AUTO_INCREMENT,
    session_id INT NOT NULL,
    amount DECIMAL(10,2) NOT NULL,
    payment_method ENUM('CASH', 'CARD', 'DIGITAL') NOT NULL,
    payment_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    transaction_id VARCHAR(100),
    payment_status ENUM('SUCCESS', 'FAILED', 'PENDING') DEFAULT 'PENDING',
    FOREIGN KEY (session_id) REFERENCES parking_sessions(session_id)
);
```

Features

Customer Features

1. **Vehicle Registration**: Register and manage vehicle information
2. **Parking Slot Booking**: View available slots and book parking
3. **Entry/Exit Management**: Automated check-in and check-out
4. **Bill Generation**: Automatic calculation of parking fees
5. **Payment Processing**: Multiple payment options
6. **Parking History**: View past parking sessions

Administrator Features

1. **User Management**: Manage customer accounts and permissions
2. **Slot Management**: Add, modify, and maintain parking slots
3. **Rate Management**: Set and update parking rates
4. **Reports Generation**: Generate various reports and analytics
5. **System Configuration**: Manage system settings and parameters
6. **Revenue Tracking**: Monitor income and financial reports

System Features

1. **Real-time Updates**: Live status of parking availability
2. **Security**: User authentication and authorization
3. **Data Backup**: Automated database backup and recovery
4. **Audit Trail**: Log all system activities for security
5. **Multi-user Support**: Concurrent user access handling

Installation Guide

Step 1: Prerequisites Installation

1. **Install Java JDK 8 or higher**

```bash
# Verify Java installation
java -version
javac -version
```

2. **Install MySQL Server**
   - Download and install MySQL Community Server
   - Set up root password during installation

- Start MySQL service

Step 2: Database Setup

1. **Create Database**

```sql
CREATE DATABASE parking_management_db;
USE parking_management_db;
```

2. **Execute Database Scripts**
   - Run the table creation scripts provided in the database design section
   - Insert sample data if needed

Step 3: Project Setup

1. **Download Project Files**

```bash
git clone [repository-url]
cd parking-management-system
```

2. **Configure Database Connection**

```java
// Update database credentials in DBConnection.java
private static final String URL = "jdbc:mysql://localhost:3306/parking_management_db";
private static final String USERNAME = "root";
private static final String PASSWORD = "your_password";
```

3. **Add MySQL Connector JAR**
   - Download MySQL Connector/J
   - Add to project classpath

Step 4: Compilation and Execution

1. **Compile the Project**

```bash
javac -cp ".:mysql-connector-java.jar" *.java
```

2. **Run the Application**

```bash
java -cp ".:mysql-connector-java.jar" MainApplication
```

User Manual

For Customers

1. Registration and Login

1. Launch the application
2. Click "Register" for new users
3. Fill in personal details and create account
4. Login with username and password

2. Vehicle Management

1. Go to "My Vehicles" section
2. Click "Add Vehicle"
3. Enter vehicle details (license plate, type, etc.)
4. Save vehicle information

3. Parking a Vehicle

1. Select "Book Parking" from main menu
2. Choose vehicle from registered vehicles
3. Select available parking slot
4. Confirm booking and receive entry token

4. Exiting and Payment

1. Go to "Exit Parking" section
2. Enter vehicle details or scan exit token
3. System calculates parking duration and fee
4. Choose payment method and complete payment
5. Receive exit confirmation

For Administrators

1. Admin Login

1. Use admin credentials to login
2. Access admin dashboard

2. Managing Parking Slots

1. Navigate to "Slot Management"
2. Add new slots with details (number, floor, type, rate)
3. Modify existing slots as needed
4. Set maintenance status for repairs

3. User Management

1. Go to "User Management" section
2. View all registered users
3. Activate/deactivate user accounts
4. Reset user passwords if needed

4. Reports and Analytics

1. Access "Reports" section
2. Generate daily, weekly, monthly reports
3. View revenue analytics
4. Export reports to PDF or Excel

Code Structure

Package Organization

```
src/
├── com/
│   └── parkingmanagement/
│       ├── main/
│       │   └── MainApplication.java
│       ├── model/
│       │   ├── User.java
│       │   ├── Vehicle.java
│       │   ├── ParkingSlot.java
│       │   ├── ParkingSession.java
│       │   └── Payment.java
│       ├── dao/
│       │   ├── UserDAO.java
│       │   ├── VehicleDAO.java
│       │   ├── ParkingSlotDAO.java
│       │   ├── ParkingSessionDAO.java
│       │   └── PaymentDAO.java
│       ├── service/
│       │   ├── UserService.java
│       │   ├── ParkingService.java
│       │   ├── PaymentService.java
│       │   └── ReportService.java
│       ├── ui/
│       │   ├── LoginFrame.java
│       │   ├── CustomerDashboard.java
│       │   ├── AdminDashboard.java
│       │   └── DialogBoxes.java
│       └── util/
│           ├── DBConnection.java
│           ├── ValidationUtil.java
│           └── DateTimeUtil.java
```

Key Classes Description

1. Model Classes

- **User.java**: Represents user entities with properties and methods
- **Vehicle.java**: Vehicle information and related operations
- **ParkingSlot.java**: Parking slot details and availability status
- **ParkingSession.java**: Active and completed parking sessions
- **Payment.java**: Payment information and transaction details

2. DAO Classes

- **UserDAO.java**: Database operations for user management

- **VehicleDAO.java**: CRUD operations for vehicle data
- **ParkingSlotDAO.java**: Slot management database operations
- **ParkingSessionDAO.java**: Session tracking and management
- **PaymentDAO.java**: Payment processing and history

3. Service Classes

- **UserService.java**: Business logic for user operations
- **ParkingService.java**: Core parking management logic
- **PaymentService.java**: Payment processing and validation
- **ReportService.java**: Report generation and analytics

Testing

Unit Testing

Individual components tested for:

- Database connection and queries
- Business logic validation
- User input validation
- Payment calculations

Integration Testing

System integration tested for:

- Database and application layer communication
- User interface and business logic integration
- End-to-end workflow testing

Test Cases

1. User Registration Test

- **Input**: Valid user details
- **Expected**: User successfully registered
- **Status**: ✅ Passed

2. Vehicle Parking Test

- **Input**: Valid vehicle and available slot
- **Expected**: Parking session created

- **Status**: ✅ Passed

3. Payment Processing Test

- **Input**: Valid payment details
- **Expected**: Payment processed successfully
- **Status**: ✅ Passed

4. Admin Report Generation Test

- **Input**: Date range for report
- **Expected**: Accurate report generated
- **Status**: ✅ Passed

Future Enhancements

Short-term Improvements

1. **Mobile Application**: Develop Android/iOS apps
2. **SMS Notifications**: Send parking reminders and receipts
3. **Online Payment Integration**: Add PayPal, Stripe payment gateways
4. **Barcode/QR Code**: Implement barcode scanning for faster processing

Long-term Enhancements

1. **IoT Integration**: Connect with smart parking sensors
2. **Machine Learning**: Predictive analytics for parking demand
3. **Multi-location Support**: Manage multiple parking facilities
4. **Advanced Reporting**: Business intelligence dashboard
5. **API Development**: REST APIs for third-party integrations

Technical Improvements

1. **Web Interface**: Convert to web-based application
2. **Cloud Deployment**: Deploy on AWS/Azure cloud platforms
3. **Microservices Architecture**: Break down into microservices
4. **Security Enhancements**: Implement OAuth2 and encryption

---

Conclusion

The Parking Management System successfully addresses the challenges of traditional parking management through automation and digitalization. Built with Java, JDBC, and SQL technologies, the system provides a robust, scalable, and user-friendly solution for both customers and administrators.

Key Achievements

- **Automated Operations**: Reduced manual intervention in parking management
- **Real-time Tracking**: Provided instant parking availability information
- **Integrated Billing**: Streamlined payment processing and fee calculation
- **Comprehensive Reporting**: Enabled data-driven decision making
- **User-friendly Interface**: Created intuitive interfaces for all user types

Learning Outcomes

The development team gained valuable experience in:

- **Java Programming**: Advanced OOP concepts and application development
- **Database Design**: Relational database modeling and optimization
- **JDBC Integration**: Database connectivity and SQL operations
- **Project Management**: Team collaboration and version control
- **System Analysis**: Requirements gathering and system design

Project Impact

The system demonstrates practical application of software engineering principles and provides a foundation for real-world parking management solutions. The modular design and comprehensive documentation ensure maintainability and future extensibility.

Appendix

A. Database Scripts

All SQL scripts for table creation, sample data insertion, and stored procedures are included in the `/database` directory.

B. User Interface Screenshots

Screenshots of all major system interfaces are available in the `/screenshots` directory.

C. API Documentation

Detailed API documentation for all service methods is provided in the `/docs/api` directory.

D. Installation Video

Step-by-step installation guide video is available at: [Installation Guide Link]

Project Completed By:

- Yash Chavan
- Gaurav Salunke
- Aaditi Bais

Date: July 2025 **Version**: 1.0 **Institution**: Dr. D.Y. Patil Technical Campus **Course**: Computer Science & Engineering